Modeling language with structured penalties

Anil Kumar Nelakanti

UPMC
SORBONNE UNIVERSITÉS

THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École Doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Anil Kumar Nelakanti

Pour obtenir le grade de
DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Modélisation du Langage à l'aide de Pénalités Structurées

Modeling Language with Structured Penalties

soutenue le 11 Février 2014

devant le jury composé de :

M. Cédric ARCHAMBEAU	Directeur de thèse
M. Francis BACH	Directeur de thèse
M. Massih-Reza AMINI	Rapporteur
M. Thierry ARTIERES	Examinateur
M. Guillaume BOUCHARD	Examinateur

Abstract

Modeling natural language is among fundamental challenges of artificial intelligence and the design of interactive machines. It has applications spanning across various domains, such as dialogue systems, text generation and machine translation among others and has been studied extensively in the past decades. Among problems that are common to most applications of language is to model the distribution over a vocabulary of words that could follow a given sequence of words or the context.

The simplest of models in the literature are based on frequency counts. Since the size of the vocabulary is large, many possible sentences are never observed leading to sparse sampling or data sparsity. This causes overestimation of observed sequences and underestimation of the unobserved ones. Thus it is necessary to use *prior* knowledge to appropriately modify the distribution. Frequency counts are commonly modified using *smoothing techniques*. Learning-based methods are an alternative to frequency-based methods that fit an appropriate model to the data. Numerous learning techniques have been applied to build language models; generative models (Dirichlet, Pitman-Yor and hierarchical Pitman-Yor processes), discriminative models (maximum entropy, conditional random fields) and distributional representations (neural networks, Boltzman machines, log-bilinear models).

We propose a discriminatively trained log-linear model to learn the distribution of words following a given context. Due to data sparsity, it is necessary to appropriately regularize the model using a penalty term. However, simple choices for regularization (like ℓ_2^2 and ℓ_1 penalties) fail to capture long range dependencies. They overfit severely with increasing context length because they are agnostic to the sequential structure of the problem. We show that it is necessary to design a penalty term that properly encodes the structure of the feature space to avoid overfitting and improve generalization while appropriately capturing long range dependencies. Some nice properties of specific structured penalties can be used to reduce the number of parameters required to encode the model. The outcome is an efficient model that suitably captures long dependencies in language without a significant increase in time or space requirements.

In a log-linear model, both training and testing become increasingly expensive with growing number of classes. This is because the normalization factor involved in computing the probabilities grows linearly with the number of classes or categories in the problem. The number of classes in a language model is the size of the vocabulary which is typically very large. A common trick is to cluster classes and apply the model in two-steps; the first step picks the most probable cluster and the second picks the most probable word from the chosen cluster. This idea can be generalized to a hierarchy of larger depth with multiple levels of clustering. However, the performance of the resulting hierarchical classifier depends on the suitability of the clustering to the problem and building a 'good' hierarchy is non-trivial. We study different strategies to build the hierarchy of categories from their

observations. We observe that the choice of the algorithm to build a hierarchy should depend on the data and that in certain situations a hierarchical classifier could improve performance over a simple multi-class classifier.

Résumé

La modélisation de la langue naturelle est l'un des défis fondamentaux de l'intelligence artificielle et de la conception de systèmes interactifs. Ses applications s'étendent à travers divers domaines, tels que les systèmes de dialogue, la génération de texte et la traduction automatique, et a été largement étudié au cours des derni ères décennies. Le problème commun à la plupart des applications du langage est la modélisation de la distribution d'un mot étant donné son contexte, généralement constitué d'une séquence de mots voisins dans le texte.

Les modèles de langue les plus simples sont basés sur la fréquence d'apparition de mots ou de séquences de mots. Comme la taille du vocabulaire est en générale importante, de nombres occurences de phrase probables ne sont jamais observées dans un corpus de textes, donnant lieu a un un problème d'estimation ou le nombre de paramètres est largement supérieur à la taille de l'échantillon d'apprentissage. Cela entraine une surestimation de la probabilité des séquences observées et une sous-estimation des séquences non observées. Ainsi, il est nécessaire d'utiliser des connaissances *a priori* avant de modifier de maniére appropriée la distribution. Les fréquences empiriques sont souvent corrigées grâce à des techniques de lissage. Les méthodes d'apprentissage statistiques basées sur la minimisation d'un coût empirique correspondent à un modèle approprié aux données et sont une alternative interessante aux méthodes basés sur les fréquences. De nombreuses techniques d'apprentissage ont été appliquées à la construction de modèles de langue; modèles génératifs (Dirichlet, Pitman-Yor et les processus Pitman-Yor hiérarchiques), les modèles discriminants (entropie maximale, conditional random fields) et les représentations de distribution (réseaux de neurones, machines de Boltzman, modéles log-bilinéaire).

Nous proposons un modèle log-linéaire discriminatif donnant la distribution des mots qui suivent un contexte donné. En raison de la parcimonie des données, il est nécessaire de régulariser de manière appropriè le modèle en utilisant un terme de pénalité. Cependant, des choix simples de régularisation (comme ℓ_2^2 et ℓ_1 pénalités) ne parviennent pas à capturer les dépendances longues: ils *sur-apprennent* sur les données lorsqu'on augmente de la longueur de contexte, car ils sont agnostiques à la structure séquentielle du problème. Nous montrons qu'il est nécessaire de concevoir un terme de pénalité qui code correctement la structure de l'espace fonctionnel pour éviter le sur-apprentissage et d'améliorer la généralisation, tout en capturant de manière appropriée les dépendances à long terme. Quelques propriétés intéressantes de pénalités structurés spécifiques peuvent être utilisés pour réduire le nombre de paramètres requis pour coder le modèle. Le résultat est un modèle efficace qui capte suffisamment les dépendances longues sans occasionner une forte augmentation des resources en espace ou en temps.

Dans un modèle log-linéaire, les phases d'apprentissage et de tests deviennent de plus en plus chères avec un nombre croissant de classes. Cela est dû au fait que le facteur de

normalisation impliqué dans le calcul des probabilités croît linéairement avec le nombre de classes ou de catégories dans le problème. Le nombre de classes dans un modèle de langue est la taille du vocabulaire, qui est généralement très importante. Une astuce courante consiste à appliquer le modèle en deux étapes: la première étape identifie le cluster le plus probable et la seconde prend le mot le plus probable du cluster choisi. Cette idée peut être généralisée à une hiérarchie de plus grande profondeur avec plusieurs niveaux de regroupement. Cependant, la performance du système de classification hiérarchique qui en résulte dépend du domaine d'application et de la construction d'une bonne hiérarchie n'est pas triviale. Nous étudions différentes stratégies pour construire la hiérarchie des catégories de leurs observations. Nous observons que le choix de l'algorithme pour construire une hiérarchie devrait dépendre des données et que, dans certaines situations, un classificateur hiérarchique peut améliorer les performances par rapport à un classificateur multi-classe simple.

Détail des contributions

Les modèles de langue probabilistes sont utilisés fréquemment pour une grande variété de tâches de traitement du langage et constituent une composante cruciale dans de nombreuses applications, comme le traitement de la parole, les systèmes de dialogues et la traduction automatique (cf. Jurafsky and Martin, 2008; Manning and Schutze, 1999). Ces modèles ont été développés depuis une vingtaine d'années et emploient des techniques comme les fréquences d'apparition de mots, des méthodes basées sur le maximum de vraisemblance, des méthodes Bayésiennes, des réseaux de neurones ou des modèles basés sur des représentations distribuées. Les méthodes les plus simples utilisent des estimations des fréquences d'apparition à partir d'un corpus de texte (Chen and Goodman, 1998); les estimateurs obtenus par optimisation ont été développés à partir des méthodes basées sur les fréquences d'apparition par apprentissage de paramètres d'un modèle statistique aussi à partir de corpus (Chen and Rosenfeld, 2000). Malgré la litérature considérable sur ces nombreux modèles, leur intégration dans un cadre d'apprentissage permettant d'obtenir une information pertinente à partir de corpus de textes reste un défi.

Les méthodes basées sur l'optimisation sont plus génériques et permettent plus naturellement que les méthodes basées sur les fréquences de regrouper l'information venant de sources multiples (Lau, 1994; Della Pietra et al., 1997). Cependant, une utilisation naïve de ces modèles ne permet de généraliser au-delà des exemples d'apprentissage. Ceci est dû au problème classique de sur-apprentissage des modèles statistiques de grande dimension appris à partir d'un échantillon trop faible (Hastie et al., 2009). Une solution classique est alors de restreindre l'espace des modèles pour contrôler l'erreur de généralisation, une technique courante étant la régularisation, ou pénalisation (Vapnik, 1998). Cependant, les choix standards de pénalisation ne sont pas satisfaisants pour les probèmes dont les paramètres ont des contraintes topologiques naturelles (Bach et al., 2012). Les pénalités doivent alors êtres construites pour prendre en compte cette structure. Dans ce manuscrit, nous étudions de telles pénalités pour la modélisation du langage, mais nos techniques s'appliquent plus généralement à la modélisation de séquences.

L'optimisation d'une fonction objectif dédiée à l'apprentissage d'un modèle permet d'apprendre les paramètres à partir des données d'entrainement. Le processus de test permet alors une évaluation sur d'autres données du modèle appris. A la fois entrainer et tester deviennent très coûteux en présence d'un nombre important de catégories. Le nombre de classes pour un modèle de langue est la taille du vocabulaire, qui peut être très grande, typiquement de l'ordre de 10^4 mots ou plus. Ceci rend l'utilisation de modèles statistiques particulièrement astreignante en pratique. Une technique simple pour surpasser le problème est de

regrouper certaines classes (ici des mots) en "clusters" et d'appliquer un modèle à deux niveaux. En généralisant cette idée, il est possible de construire une hiérarchie de clusters et d'appliquer le modèle séquentiellement le long de cette hiérarchie. Un avantage immédiat d'une telle organisation est un gain significatif en temps de calcul pour la phase de test. Cette technique de construction de taxonomie à base d'arbres est générique et a des applications au-delà des modèles de langue.

La première contribution de ce manuscrit est le développement d'un modèle de langue avec une pénalité structurée qui permet une meillleure performance de généralisation que les méthodes précédentes. Un cadre d'optimisation efficace est aussi introduit et donne lieu à des résultats plus compacts qu'avec les méthodes traditionnelles. La deuxième contribution est un cadre algorithmique pour organiser automatiquement un problème multi-classe en une structure d'arbre. Plusieurs méthodes de construction d'arbres sont introduites, à partir de principes "top-down" ou "bottom-up". Ces méthodes sont alors testées sur des jeux de données de langage et de vision artificielle.

En Section 0.1, le problème de modélisation du langage est décrit; les méthodes traditionnelles (méthodes de lissages et méthodes par maximum d'entropie) sont présentées en Section 0.2. Enfin, les motivations sous-jacentes au travail effectué sont présentées en Section 0.3. Finalement, les contributions de cette thèse sont décrites en détail en Section 0.4.

0.1 Modélisation de la langue naturelle

La modélisation du langage correspond à estimer des lois de probabilité sur des séquences de mots tirés d'un vocabulaire fini V. Soit $s = (s^1, s^2, \cdots, s^{|s|})$ une séquence de mots $s^i \in V$ avec $|s|$ la longueur de la séquence. Nous utilisons la notation $s = (xy)$ pour décrire une séquence composée d'un mot $y \in V$ suivant le contexte $x \in V^{|s|-1}$. Le suffixe d'une séquence x de longueur m est noté x_m et est la sous-séquence composée des m derniers mots de x. Notre but est de développer un algorithme $\mathcal{A}$ permettant d'apprendre une loi $\mathbf{P}(s)$ sur les séquences à partir d'un corpus (i.e., une collection de séquences de mots de V).

Etant donné le grand nombre de séquences d'une taille donnée, un corpus $\mathcal{C}$ de taille raisonnable ne peut pas contenir toutes les séquences grammaticalement correctes (qui sont déjà beaucoup moins nombreuses que l'ensemble des séquences). Ceci rend l'estimation d'une loi $\mathbf{P}(s)$ très difficile. Dans cette thèse nous allons étudier le problème plus simple d'estimation de la loi conditionnelle $\mathbf{P}(y \mid x)$ d'un mot y suivant le contexte x. La distribution jointe d'une séquence sera obtenue en prenant des produits de ces lois conditionnelles. En pratique la taille des contextes considérés est bornée par le modélisateur: Les modèles "m-gram" correspondent à des séquences de taille au plus m. Dans les modèles de langue que nous considérons ci-après, la longueur m des contextes est utile pour tabuler les occurences de séquences.

Considérons le modèle d'ordre m pour des séquences d'ordre $|s| = m + 1$. Le modèle simple considère les fréquences d'apparition empiriques

$$\mathbf{P}_{ML}(y|x_m) = \frac{c(x_m y)}{c(x_m)}, \tag{1}$$

où $c(s)$ est le nombre de fois où s est observé dans $\mathcal{C}$, que nous dénommerons "compte". En prenant le produit des probabilités conditionnelles, on obtient une loi

$$\mathbf{P}_{ML}(s) = \prod_{(xy)\in\{(s^1,\cdots,s^i)\}_{i=1}^{|s|}} \mathbf{P}(y|x_m).$$

Ceci permet de donner une probabilité à tout le corpus $\mathcal{C}$:

$$\mathbf{P}_{ML}(\mathcal{C}) = \prod_{s\in\mathcal{C}} \mathbf{P}(s).$$

Il s'avère que le modèle défini en (1) maximise la probabilité $\mathbf{P}_{ML}(\mathcal{C})$ ci-dessus par rapport aux modèles de langage d'ordre m (c'est l'estimateur de maximum de vraisemblance).

Ce modèle simple ne donne cependant pas satisfaction: soit m est trop petit et le modèle est trop pauvre pour prendre en compte les dépendances longues; soit m est trop grand, et il n'y pas assez de données pour apprendre tous ces paramètres avec de plus un comportement particulièrement problématique: une séquence jamais observée dans le corpus aura toujours une probabilité égale à zéro pour le modèle. Plusieurs techniques ont été développées pour résoudre ces problèmes.

0.2 Méthodes existantes

Dans cette section, nous décrivons les méthodes classiques permettant d'apprendre efficacement des modèles avec des longueurs de contexte m plus ou moins élevées.

0.2.1 Méthodes par lissage

Les méthodes de lissage sont basées sur l'estimateur en (1), i.e., sur des estimations empiriques des fréquences d'apparition, mais changent les lois de prédiction, soit en redistribuant la masse de probabilité, soit en mélengeant les distributions venant de plusieurs ordres (voir Chen and Goodman, 1996, 1998, pour une description complète de ces méthodes).

Redistribution de la masse de probabilité.

Le lissage additif consiste à rajouter de faux exemples pour tous les contextes (comme si ils avaient été observés dans $\mathcal{C}$). Ceci correspond au lissage classique de *Laplace* pour les données discrètes (Lidstone, 1920; Jefreys, 1948):

$$\mathbf{P}_{LP}(y|x_m) = \frac{c(x_m y) + d}{c(x_m) + |V|d}, \text{ avec } 0 < d \leq 1. \tag{2}$$

pour un certain d. Toutes les séquence non observées ont alors la même masse. Alternativement, l'estimateur de *Good-Turning* redistribue la masse plus harmonieusement (Good, 1953).

Lissage par marche arrière.

Afin d'estimer $\mathbf{P}(y|x_m)$, une méthode simple pour éviter les zéros est de rechercher des contextes plus petits avec au moins une observation dans $\mathcal{C}$. C'est le lissage par marche arrière proposé par Katz (1987). Dans le cas extrême, le plus petit contexte possible m est égale à zéro et le modèle utilise alors la probabilité "uni-gram", où aucun contexte n'est observé. Afin d'obtenir une loi de probabilité sommant à un, un poids bien choisi $\eta(x_m)$ doit être utilisé et on a la définition récursive suivante:

$$\mathbf{P}_{KZ}(y|x_m) = \begin{cases} \mathbf{P}(y|x_m) & \text{utiliser le modèle d'ordre supérieur si } c(x_m y) > \tau \\ \eta(x_m)\mathbf{P}_{KZ}(y|x_{m-1}) & \text{sinon,} \end{cases} \tag{3}$$

où τ est un certain seuil. Les poids $\eta(x_m)$ peuvent alors être calculés récursivement.

Lissage par interpolation.

Une extension du lissage par marche arrière consiste a combiner linéairement les distributions d'ordres faibles et élevés. Ceci a été proposé par Jelinek and Mercer (1980) avec une loi définie récursivement comme suit

$$\mathbf{P}_{JM}(y|x_m) = \gamma(x_m)\mathbf{P}(y|x_m) + (1 - \gamma(x_m))\mathbf{P}_{JM}(y|x_{m-1}),$$

pour des poids bien choisis $\gamma(x_m)$. Ceux-ci peuvent être estimés à partir de données en reduisant le nombre de paramêtres à apprendre par regroupements (Bahl et al., 1983).

Lissage de Kneser-Ney.

Une alternative à l'interpolation linéaire est (a) de soustraire explicitement les comptes des ordres élevés des comptes d'ordres inférieurs et (b) de repondérer les distributions obtenues. La méthode introduite par Ney et al. (1994) a la forme suivante:

$$\mathbf{P}_{AD}(y|x_m) = \frac{\max\{c(x_m) - \delta,\ 0\}}{c(x_{m-1})} + (1 - \gamma(x_m))\mathbf{P}_{AD}(y|x_{m-1}).$$

Kneser and Ney (1995) ont étendu cette technique en utilisant des comptes modifiés pour les petits ordres de telle sorte que le lissage est optimisé pour les situations où les comptes d'ordres élevés sont petits.

Discussion sur les méthodes de lissage.

Ces méthodes donnent de bonnes performances et sont encore préférées à d'autres modèles moins attractifs en termes de temps de calcul, avec une utilisation classique du lissage de Kneser-Ney (et ses variantes). Cependant, l'absence de paramètres statistiques explicites peut s'avérer limitante car elle ne permet pas d'étendre ces méthodes naturellement à des situations plus complexes, en particulier lorsque certaines tâches precises de traitement du langage naturel sont considérées. Les méthode par maximum d'entropie constituent alors une classe d'alternatives intéressantes.

0.2.2 Entropie maximale et maximum de vraisemblance

Considérons un modèle de langue d'ordre $(m+1)$ qui nécessitent d'estimer la loi conditionnelle $\mathbf{P}_{ME}(y|x_m)$. Chaque séquence s de longueur inférieure à $(m+1)$ dans le corpus $\mathcal{C}$ définit un "feature" (descripteur) :

$$\mathcal{S} = \{s \mid s \in \mathcal{C} \wedge |s| \leq m+1\}.$$

Un paramètre ω_s est associée à chacun des features s dans $\mathcal{S}$ donnant lieu à un modèle avec potentiellement un grand nombre $d = |\mathcal{S}|$ de paramètres.

Les méthodes de maximum d'entropie correspondent à modéliser $\mathbf{P}_{ME}(y|x_m)$ comme le logarithme d'une fonction linéaire et de maximiser la vraisemblance (voir Rosenfeld, 1998; Lau, 1994; Lau et al., 1993; Rosenfeld, 1996; Ratnaparkhi, 1998; Berger et al., 1996). Avec la définition suivante pour $s \in \mathcal{S}$

$$\mathbb{I}(x_m y, s) = \begin{cases} 1 & \text{if } x_k y = s \text{ pour un } 0 \leq k \leq m, \\ 0 & \text{sinon,} \end{cases}$$

la probabilité conditionnelle s'écrit

$$\mathbf{P}_{ME}(y|x_m;\omega) = \frac{1}{Z(x_m;\omega)} e^{\sum_{s\in\mathcal{S}} \omega_s \mathbb{I}(x_m y, s)},$$

avec la constante de normalisation (fonction de partition):

$$Z(x_m;\omega) = \sum_{v\in V} e^{\sum_{s\in\mathcal{S}} \omega_s \mathbb{I}(x_m v, s)}. \tag{4}$$

On obtient alors la probabilité $L(\mathcal{C};\omega)$ du corpus :

$$L(\mathcal{C};\omega) = \mathbf{P}_{ME}(\mathcal{C};\omega) = \prod_{\{x_m y\}\in\mathcal{C}} \mathbf{P}_{ME}(y|x_m;\omega) = \sum_{\{x_m y\}\in\mathcal{C}} \frac{e^{\sum_{s\in\mathcal{S}} \omega_i \mathbb{I}(x_m y, s)}}{Z(x_m;\omega)}. \tag{5}$$

Le problème du maximum de vraisemblance devient donc:

$$\begin{aligned} \mathcal{L}(\mathcal{C};\omega) &= -\sum_{\{x_m y\}\in\mathcal{C}} \ln \mathbf{P}_{ME}(y|x_m), \text{ et} \\ \omega^* &= \operatorname*{argmin}_{\omega\in\mathbb{R}^d} \mathcal{L}(\mathcal{C};\omega). \end{aligned} \tag{6}$$

L'optimisation pour le problème convexe en (6) peut être résolu à l'aide de méthodes du premier ordre convergeant vers le minimum global.

Relations avec les méthodes de lissage.

Contrairement aux méthodes de lissage, les méthodes par maximum d'entropie font une modélisation explicite de la loi avec optimisation de paramètres. Ceci permet une certaine flexibilité et ces méthodes ont été appliquées à de nombreuses tâches en traitement du langage naturel. Cependant, elles sont deux problèmes classiques: le sur-apprentissage et l'inefficacité computationnelle.

Sur-apprentissage et régularisation.

Etant donné le grand nombre de paramêtres, ces méthodes peuvent sur-apprendre. Il est alors classique d'utiliser des pénalités supplémentaires pour réduire la capacité du modèle (Chen and Rosenfeld, 2000).

Les pénalités classiques sont la norme ℓ_2 (Della Pietra and Della Pietra, 1993; Chen and Rosenfeld, 1999), la norme ℓ_1 (Goodman, 2004) qui n'offre que peu d'avantage (la parcimonie n'est pas un bon a priori), alors qu'une combinaison donne lieu à des améliorations (Kazama and Tsujii, 2003; Chen, 2009). Notre but est d'aller au-delà.

Calcul du facteur de normalisation.

En présence de grands vocabulaires, le calcul de Z en (4) est nécessaire à chaque étape des algorithmes d'optimisation, ce qui s'avère très coûteux. Des approches basées sur des caches (Wu and Khudanpur, 2000) ou des regroupements de termes (Goodman, 2001) sont possibles pour réduire le temps de calcul. Dans cette thèse nous proposons une alternative en contruisant une hiérarchie de classes automatiquement à partir des données.

0.2.3 Méthodes récentes.

Depuis quelques années, des modèles de langage ont été développés en utilisant des techniques d'inférence Bayésienne (modèles génératifs basés sur de processus de Dirichlet ou de Pitman-Yor et les processus Pitman-Yor hiérarchiques) ou des reśeaux de neurones (machines de Boltzman, modéles log-bilinéaire). Voir la description dans le corps du manuscrit.

0.3 Motivations

En résumé, de nombreuses méthodes ont été développées pour modéliser le langage. Elles ont toutes des avantages et des inconvénients et certaines sont plus adaptées que d'autres pour certaines tâches.

Alors que la plupart des méthodes se focalisent sur la modélisation de la probabilité conditionnelle d'un mot étant donné les mots le précédant, certaines méthodes rendent la tâche de modélisation plus aisée. Une application typique des modèles de langue nécessite plusieurs couches de traitement de l'information. Dans de telles situations, de nombreux facteurs rentrent en jeu pour déterminer l'applicabilité d'un modèle particulier. Des considérations comme la facilité d'intégration, des aspects de temps de calculs ou de coût en mémoire (à accès rapide ou lent), ou la possibilité d'intégrer des informations supplémentaires, sont importantes en pratique.

Notre principale motivation pour poursuivre le travail sur ces modèles et en proposer des améliorations est de pouvoir appliquer ces modèles à des problèmes *discriminatifs*, qui constituent une classe particulière des modèles présentés en Section 0.2.2, pour lesquels la probabilité d'un mot dépend non seulement des mots précédents mais aussi potentiellement de variables exogènes. Dans ce qui suit, nous utiliserons la génération de texte dans des applications multi-media comme exemple. Nous décrirons ensuite les aspects des modèles discriminatifs que nous étudierions et enfin résumerons les contributions de la thèse.

Modélisation discriminative de données multi-modales.

Dans le contexte global de la compatibilité des modèles pour les applications, nous voyons les modèles discriminatifs comme les modèles permettant la flexibilité nécessaire à des scénarios applicatifs classiques où il est important de faire de la prédiction à partir de plusieurs sources d'informations, typiquement en combinant linéairement les descripteurs.

Par exemple, dans les applications multi-modales, le langage naturel peut être une des modalités (typiquement la modalité cible). Les autres modalités peuvent être un signal audio, une image ou une séquence vidéo (typiquement des modalités sources). La génération automatique de résumés décrivant une image est une application simple à fort impact.

En effet, le problème de la génération automatique de légendes textuelles destinées aux humains est un problème en pleine croissance (voir, par exemple, Gupta et al., 2009; Farhadi et al., 2010; Kulkarni et al., 2011; Yang et al., 2011; Mitchell et al., 2012). Les premiers modèles proposés utilisaient des constructions basées sur des modèles-types de phrases dont le choix était appris à partir de certaines informations dans l'image (obtenues par des techniques de vision artificielle) (Kulkarni et al., 2011). Ces méthodes basées sur des modèles-types sont connues pour être sous-optimales quand des variations textuelles sont nécessaires (Reiter and Dale, 1997). Ces techniques ont depuis été améliorées avec des méthodes qui génèrent directement des arbres syntaxiques permetant ensuite de produire un texte grammaticalement correct (Mitchell et al., 2012; Yang et al., 2011).

Cependant, ces méthodes traitent les deux modalités séparément avec un échange d'information uni-directionnel qui (a) résume une image en quelques mots, souvent des annotations de l'image avec quelques étiquettes et une certaine organisation spatiale (Mitchell et al., 2012; Yang et al., 2011; Farhadi et al., 2010; Ordonez et al., 2011), puis (b) envoie ces informations au modèle de langage pour créer une phrase. Le rôle du modèle de langage est alors de regrouper ces informations trop simples et de former une phrase cohérente en rajoutant typiquement beaucoup de mots autour des mots obtenus de l'image. Les phrases obtenues peuvent alors être cohérentes et grammaticalement correctes, mais ne correspondent plus à ce qu'il se passe réellement dans l'image ou la video (Mitchell et al., 2012).

Produire des phrases grammaticalement correctes *et* qui expriment correctement les signaux sources nécessite un modèle permettant d'utiliser à bon escient les grands corpus de textes (Li et al., 2011). Par ailleurs, ces modèles doivent pouvoir traiter les signaux sources en prédisant *directement* les distributions de mots sans passer par des étapes sur-simplificatrices causant des pertes d'information. Dans de telles tâches, nous considérons les modèles discriminatifs comme naturellement adaptés: en effet, on peut utiliser un modèle de langue obtenus de sources variées que nous pouvons utiliser pour apprendre un modèle conditionnel pour nos données multi-modales. De plus, ces modèles discriminatifs peuvent gérer naturellement des aspects importants du langage, comme les variations de style ou de ton, en changeant certains termes dans les fonctions objectifs à optimiser.

Codage de contextes longs pour la modélisation discriminative.

La génération de phrases complètes dans un cadre discriminatif requiert le traitement adéquat des contextes longs, avec dans certains cas très longs si plusieurs phrases sont impliquées. Ceci pourrait être nécessaire pour plusieurs raisons. Par exemple, il pourrait s'avérer intéressant de supprimer certaines informations des phrases précédentes afin

d'éviter des répétitions, ou de réduire la ré-utilisation de mêmes mots dans les mêmes phrases afin de maintenir une bonne lisibilité, ou de contrôler l'ordre dans lequel le contenu est délivré, etc. Dans de telles situations, un modèle discriminatif doit pouvoir coder de grandes phrases efficacement. C'est un des aspects que nous considérons dans notre travail.

Modèles log-linéaires avec grand nombre de classes.

Ce problème est d'importance plus générale que les modèles de langue, mais il est particulièrement important pour le langage naturel car le nombre de classes est égal au nombre de mots dans le vocabulaire (typiquement supérieur à 10^4). Le verrou computationnel est alors le calcul de la constante de normalisation pour lequel le nombre de produits scalaires à calculer croît linéairement avec le nombre de classes. Ceci ralentit l'apprentissage car à chaque étape d'une méthode de descente de gradient, il faut calculer cette constante.

Afin d'obtenir une complexité logarithmique dans le nombre de classes, il est classique d'organiser les classes dans un arbre pour accélérer à la fois l'entrainement et le test. La difficulté est de construire une telle hiérarchie sans détériorer la performance de prédiction. Dans notre travail, nous proposons différentes méthodes pour construire de telles hiérarchies et testons leur performance pratique.

0.4 Contributions et plan de la thèse

Cette thèse revisite les approches traditionnelles décrites ci-dessus à la lumière d'avancées récentes sur les pénalités structurées et l'optimisation convexe à grande échelle. En particulier, nous nous focalisons sur deux problèmes liées aux modèles de langage discriminatifs: (a) le sur-apprentissage et l'explosion du temps de calcul face à des contextes longs et (b) la nécessité de modèles de sortie hiérarchiques (il est intéressant de noter que les deux contributions utilisent des arbres mais que ces arbres n'y ont pas la même signification). Nos contributions peuvent être résumées comme suit.

0.4.1 Pénalités structurées pour la modélisation du langage

Les modèles de langage discriminatifs sont connus pour sur-apprendre en présence de contextes longs. En effet, le nombre de paramètres grandit rapidement avec la longueur du contexte. Les techniques classiques de régularisation (par pénalité ℓ_2, ℓ_1 ou leur combinaison) ne sont alors pas effectives, comme le montrent les performances prédictives obtenues lors de nos simulations sur un sous-ensemble du corpus "AP-news" (voir Chapitre 3).

Nous étudions ce problème de régularisation inadéquate en reformulant le modèle comme une distribution catégorielle avec une vraisemblance tres proche du modèle de maximum d'entropie en (5) mais avec une contrainte en plus. Un point-clé est que la structure séquentielle des phrases peut alors être interprétée comme une structure de données en *arbre de suffixes*. Nous pouvons alors naturellement utiliser une pénalité convexe adaptée à cette structure d'arbre. Cette pénalité permet d'introduire un a priori nouveau: un contexte long ne peut être choisi que si tous les contextes plus petits sont impliqués dans le prédicteur. Cette pénalité est non seulement adaptée statistiquement (meilleure performance de prédiction pour les longs contextes) mais aussi peut être utilisé à un moindre coût supplémentaire en utilisant finement la structure d'arbre de suffixes.

Les différents modèles à pénalité structurée que nous considérons sont implémentés et comparés aux modèles non-structurés. Cette évaluation montre des améliorations significatives en performance de prédiction quand le contexte augmente. Nous proposons aussi une analyse fine des résultats obtenus en groupant les mots suivant différents critères permettant de mieux comprendre les gains de performance. Cette analyse suggère des améloriations pour des travaux futurs.

Du côté des performances numériques, augmenter les longueurs de contexte augmente de manière très significative le nombre de descripteurs (car la taille du vocabulaire est grande). Sans utiliser la structure d'arbre de suffixes, la complexité serait trop importante. Dans le Chapitre 3, nous étudions la complexité numérique des différentes pénalités considérées et montrons comment certaines d'entre elles peuvent être utiliseées avec des contextes de très grande taille (en utilisant les arbres de suffixes, comme fait pour le lissage de Kneser-Ney). En particulier, nous proposons un nouvel algorithme qui améliore le temps de calcul. Ce travail a été publié au congrès *Conference on Empirical Methods in Natural Language Processing (EMNLP), 2013*.

0.4.2 Apprentissage de la taxonomie

L'idée de grouper des mots en "clusters" afin de rendre l'entrainement et le test plus efficaces en termes de temps de calcul a déjà été explorée dans différents modèles (voir (Goodman, 2001) pour les modèles à maximum d'entropie, (Morin and Bengio, 2005) pour les réseaux de neurones "feed-forward", (Mikolov et al., 2011) pour des réseaux récurrents et (Mnih and Hinton, 2008) pour les modèles log-bilinéaires).

Cependant, la méthode de construction des clusters de mots dans les modèles ci-dessus est soit aléatoire soit ad-hoc, à l'exception de la technique de partitionement de Brown. Nous proposons plusieurs méthodes permettant de grouper les mots en utilisant leurs contextes. Ces méthodes sont génériques avec des applications à tous les problèmes avec beaucoup de classes, que les données soient séquentielles ou non (comme des problèmes à descripteurs continus, communément utilisés pour la reconnaisance d'objets dans les images).

Nous proposons deux approches. Dans l'approche "bottom-up", les classes sont regroupées petit à petit en utilisant un algorithme Hongrois avec un critère de corrélation entre classes. Dans l'approche "top-down", les classes sont divisées récursivement. Afin de diviser un groupe de classes en deux morceaux ou plus, nous utilisons des résultats récents de "clustering" discriminatif, où le but est de séparer les classes en deux morceaux, de telle sorte que si un séparateur linéaire est appris sur ces classes, alors la séparation est maximale. Cette approche donne lieu à une formulation en programmation semi-définie qui peut se résoudre efficacement.

Nous étudions le comportement de ces méthodes sur des jeux de données de textes et d'image et détaillons comment ces méthodes de constructions d'arbres doivent être choisies avec soin afin de ne pas perdre de performance prédictive par rapport à la structure plate. Il est intéressant de noter que dans certains cas, au-delà d'obtenir une meilleure complexité numérique, la performance de prédiction est aussi améliorée.

0.4.3 Plan de la thèse.

Le manuscrit est organisé comme suit. Le Chapitre 2 décrit le cadre de minimisation du risque empirique et la nécessité de régulariser avec un accent particulier sur les pénalités permettant d'incorporer la structure. Aussi, les différents algorithmes d'optimisation convexe adaptés à ces pénalités sont présentés. Le Chapitre 3 détaille les contributions nouvelles sur les pénalités structurées et leur utilisation efficace pour les modèles de langue. Le Chapitre 4 décrit les différentes nouvelles méthodes de regroupement des classes pour obtenir un arbre de classes. Nous résumons les contributions de la thèse et concluons en Chapitre 5, où sont aussi présentées des pistes pour des travaux futurs.

Acknowledgments

I am very thankful to my advisers Cedric, Francis and Guillaume for their guidance and support. They were very patient with me to say the least. I would also like to thank Julien, Rodholphe and Armand for their help with my work. I also wish to express my gratitude to my masters adviser Edmond who offered me an opportunity to move to Grenoble. I am grateful to Xerox Research and the French ANRT for the CIFRE grant that funded me for the whole duration. Group members both in the MLS team at Xerox and in the SIERRA team at Inria have been very helpful to me. I have greatly benefited from the support of my friends and well-wishers who made things a lot easier over these three years. There is a long list of people I must thank by name but I wish not to regret missing someone, something bound to happen with long lists. So, I skip listing them but I am truly thankful to all of them for all their help and support. I use the privileged last line to thank my parents for all their sacrifices.

Contents

List of figures

List of tables

Chapter 1

Introduction

Probabilistic language models are used extensively in a variety of language processing tasks and remain at the core of various applications like speech processing, dialogue systems, and machine translation (see Jurafsky and Martin, 2008; Manning and Schutze, 1999). These models emerged over the last couple of decades and employ techniques like modified frequency counts, likelihood methods, Bayesian methods, neural networks and models involving distributed representations. The simplest method uses frequency estimates from a text corpus (Chen and Goodman, 1998). Optimization-based estimators were developed following frequency-based methods that train models by learning their parameters on a corpus (Chen and Rosenfeld, 2000). Despite the considerable literature available with numerous models, the ease to integrate these models into a learning framework to derive high-level information from text corpora remains challenging.

Optimization-based methods are more generic and amenable to absorb information from multiple sources (Lau, 1994; Della Pietra et al., 1997) than frequency-based methods. However, a naively designed model fit to a dataset does not generalize to unseen examples. This is due to a problem common to statistical models involving high dimensional spaces with sparse sampling (Hastie et al., 2009). The solution is to restrict the set of models so that the generalization error is controlled. This is called regularization or penalization (Vapnik, 1998). However, standard choices of penalization are not sufficient for problems with parameters involving specific patterns (Bach et al., 2012). Penalization needs to be designed in a such a manner that it is aware of the structure within the problem. We study such structure-aware penalization for modeling language or discrete structures in general to improve generalization performance.

The optimization of an objective designed to learn the parameters of a model from some observed data is called training. The process of testing involves evaluation of the trained model on data (not overlapping with the training set) to measure generalization performance. Both training and testing become increasingly expensive with a growing number of categories or classes involved in the problem. The number of classes for language models is the size of the vocabulary which is very large, typically of the order 10^4 or above. This makes the use of statistical models computationally demanding for practical use. A simple technique to overcome this problem is to group together classes (or words) into clusters and then apply the same model at two levels. Generalizing this idea, one can

build a hierarchy of such clusters and apply the model successively along the hierarchy. An immediate advantage of such hierarchical organization of classes is the significant gain in speed when the resulting model is applied to test data. This technique to build taxonomic trees from samples is generic and has applications beyond language modeling.

The first contribution of this work is a language model with a suitable structured penalty that offers better generalization performance than earlier methods. We also present a framework that is quicker to optimize and results in a more compact model than traditional approaches. Our second contribution is a framework to organize classes in a large-scale multi-class problem into a tree structure. We study different tree-building methods in this framework including both top-down and bottom-up procedures to grow such trees. We test these methods on datasets of images and language. In Section 1.1, we introduce the problem of language modeling. We present traditional approaches to modeling language, namely smoothing techniques and maximum entropy models in Section 1.2. In Section 1.3 we present more recent approaches to this problem based on Bayesian framework and neural networks. We motivate the purpose of our work and describe our contributions in Section 1.5.

1.1 Problem of modeling language

Language modeling involves estimating probability distributions of sequences of words. Consider a vocabulary V from which words are drawn to make discrete sequences. Let $s = (s^1, s^2, \cdots, s^{|s|})$ be a sequence with words $s^i \in V$ and $|s|$ being the length or the number of words in the sequence. We use $s = (xy)$ to denote a split of the sequence where $y \in V$ is the word following the context $x \in V^{|s|-1}$. The suffix of a sequence x of length m is denoted by x_m which is the subsequence with the last m words of x. We wish to design an algorithm $\mathcal{A}$ that will learn the probability distribution $\mathbf{P}(s)$ over the sequences from a corpus. A corpus is a collection of sequences of words from V. The input to $\mathcal{A}$ is a text corpus $\mathcal{C}$ that is used to learn the parameters of the model. Apart from the training set $\mathcal{C}$, we will additionally need a validation set, also called held-out data, distinct from $\mathcal{C}$ to tune any necessary parameters. We distinguish the training set $\mathcal{C}$ from the test set $\mathcal{C}_t$ that is used to compare performance of different models.

The corpus $\mathcal{C}$ is a sampling from the space of sequences, however, space of $\mathcal{C}$ is significantly smaller than the complete space of sequences. This is particularly true for language where numerous possibilities are either meaningless or grammatically incorrect and have zero probability. Further, the space of meaningful and grammatically correct sequences is so varied that this space cannot be sampled sufficiently. This makes the estimation of joint distribution of words in a sequence $\mathbf{P}(s)$ very hard. Hence, the focus is usually the much simpler problem of estimating the following conditional distribution $\mathbf{P}(y \mid x)$ of a word y following the context x. The joint distribution of a sequence is inferred by taking the product of the conditionals which is equivalent to predicting one word of the sentence at a time in that order. This makes sense for language modeling since the Markovian assumption applies naturally to data of discrete sequences. In practice, one prescribes the maximal length of dependence in the sequences that restricts the distance along the sequence that a word can influence. This is also called the order of the language model. Using sequences of length at most m leads to an m-gram language model which is also called the $(m-1)$

order language model. A context of length $(m-1)$ is used to predict the following word in such models. In the language models that we will consider in the next section, the order is useful to tabulate occurrences of sequences.

Consider the $(m+1)$-gram model with sequences $|s| = m+1$. A simple model that counts the number of occurrences to assign a probability from frequency estimates is,

$$\mathbf{P}_{ML}(y|x_m) = \frac{c(x_m y)}{c(x_m)}, \tag{1.1}$$

where $c(s)$ is the number of times s was observed in $\mathcal{C}$. Taking the product of the probabilities for predicting each of the words in a sequence s, we get the joint probability,

$$\mathbf{P}_{ML}(s) = \prod_{(xy)\in\{(s^1,\cdots,s^i)\}_{i=1}^{|s|}} \mathbf{P}(y|x_m).$$

The product of the probabilities of predicting all the sequences in a corpus $\mathcal{C}$ gives the probability of generating $\mathcal{C}$ or the likelihood of $\mathcal{C}$,

$$\mathbf{P}_{ML}(\mathcal{C}) = \prod_{s\in\mathcal{C}} \mathbf{P}(s).$$

The probability estimate from the distribution defined in (1.1) maximizes the probability $\mathbf{P}_{ML}(\mathcal{C})$ of generating the corpus $\mathcal{C}$ for a given language order m and hence has the name *maximum likelihood* estimate.

However, the goal is not to maximize the likelihood of generating a given training corpus $\mathcal{C}$ but to minimize the errors in predictions on test set $\mathcal{C}_t$. They are not equivalent since the test set typically has data not seen during training and the maximum likelihood estimator in (1.1) typically assigns them small or zero probabilities because the estimator fits the training set too well. This situation is called overfitting and the resulting error also referred to as *generalization error*. Overfitting is common in problems where the enough data cannot be collected to learn parameters by simply maximizing the likelihood like in equation (1.1). This is referred to as *data sparsity* and there is significant amount of work in statistics and machine learning communities to handle this issue for various models.

The distribution from the estimator in (1.1) varies with the value of m. Models with smaller m have fewer parameters to estimate from more data in comparison to those with larger m that will have more parameters. A more confident estimate can be made when a parameter sees more data, a property characterized by smaller *variance* (often referred to as estimation error), which is the case when m is small. However, a small m assumes short dependence along the sequence discarding information available from the data leading to high *bias* (also called approximation error) in the estimate.

The assumption of short dependence does not hold since randomly drawing words independent of their context will return meaningfuless sequences. On the contrary, a large m uses more information available from the training data resulting in smaller bias but since there are fewer observations with longer context lengths the variance of the estimate shoots up. Bias and variance are two competitive properties of an estimator both of which are

directly proportional to the generalization error. Estimates from larger m will have higher variance and lower bias and vice versa. It is important to balance bias and variance so that the resulting generalization error is small.

Generalization error can be improved for a fixed amount of data without costing us variance. This is achieved by an appropriately chosen modification of the estimator referred to as regularization or smoothing. The trick is to design models that trade some bias to significantly reduce the variance and eventually improve generalization. The trade-off between bias and variance is central to statistical models and will be dealt with in detail in Chapter 2 as approximation and estimation errors. In the following section, we will study such modifications to the estimator in (1.1) called smoothing models that improve generalization. We will also study parametric models called maximum entropy models that optimize the likelihood to learn the distributions $\mathbf{P}(y|x_m)$ from the data. These are approaches that have been traditionally investigated in the language processing literature. In Section 1.3 we will study some recently introduced models for the same problem.

1.2 Traditional approaches

We will see in this section that the statistical estimation process in language modeling suffers from the high dimensionality of the problem, often called the *curse of dimensionality*. It leads to the problem of data sparsity where the datasets are a sparse sampling of the space of sequences. This motivated the design of models that improve performance over the simple estimator in (1.1) for the same amount of data. Traditionally, this has been achieved through smoothing techniques that modify the distribution estimated by (1.1) and regularization in parametric maximum entropy models. We discuss data sparsity and these models in this section.

Data sparsity. In the ideal case of infinitely large number of samples or sufficiently large dataset, the estimator in (1.1) is sufficient to make an estimate with low variance and very good generalization. However, it not possible to acquire a sufficiently large dataset for high dimensional problems. This is because the space of all possible sequences is too large to be sampled sufficiently. Specifically in the case of language, the space of all possible sequences of length m with a vocaulary of size $|V|$ is $|V|^m$. Consider a modestly sized vocabulary of the order 10^4 with a relatively short dependence at $m = 3$ and we already have an unmanagably large space of 10^{12} possible sequences. This is the problem of *data sparsity* where the feature space is too large to be finitely summarized in a statistically representative dataset. A theoretically sound way to handle data sparsity is to exploit the bias-variance trade-off discussed in the previous section. The idea is to design biased estimators that significantly reduce variance and hence improve generalization. We briefly discuss smoothing methods from the literature that achieve this for m-gram models by applying different variations of the estimator in (1.1) above.

1.2.1 Smoothing models

Consider estimating the probability of a sequence $x_m y$ that was never seen in training corpus $\mathcal{C}$. We assume there are no words out of the vocabulary V. The estimator in (1.1) simply sets its probability to zero which is inappropriate since the training data has not

seen all possible sequences. It often overestimates the probability for sequences observed in $\mathcal{C}$ and underestimates it for unseen sequences. This overfitting to training data degrades predictive performance of the model. Smoothing models adjust for this overfitting by modifying the predictive distributions. They achieve this in two different ways; redistribution of probability mass and mixing distributions of multiple orders. Methods that redistribute mass explicitly modify the counts by some chosen heuristic. Models that mix multiple orders rely on lower orders when higher orders are unobserved. We briefly mention some of these methods here that are detailed in the Appendix A.1 (see Chen and Goodman, 1996, 1998, for a complete survey comparing various smoothing models).

Redistribute probability mass. Additive smoothing is the technique of adding some fake samples to original frequencies as if they were observed in $\mathcal{C}$. This is the classical *Laplace smoothing* for categorical data which is a kind of shrinkage estimator (Lidstone, 1920; Jefreys, 1948) and has the following form:

$$\mathbf{P}_{LP}(y|x_m) = \frac{c(x_m y) + d}{c(x_m) + |V|d}, \text{ with } 0 < d \leq 1. \tag{1.2}$$

where d is chosen arbitrarily. For $d > 0$, all unobserved sequences are assigned some mass but Gale and Church (1994) argued that this is poor choice for smoothing. Alternatively, the *Good-Turning* estimator redistributes the probability mass of sequences that occurred exactly $(k+1)$ times in $\mathcal{C}$ among the sequences with frequency k,

$$\mathbf{P}_{GT}\left(\ y|x_m,\ c(x_m y){=}k\ \right) \propto (k+1)\frac{C_{k+1}}{\mathbf{E}[C_k]}, \tag{1.3}$$

where C_k is the number of sequences with frequency k and $\mathbf{E}[\ \cdot\ ]$ denotes expectation. It redistributes the mass of unit frequency sequences to sequences that we might encounter at test but were never seen in $\mathcal{C}$. This, however, requires us to make an estimate of the number of unseen sequences that we might encounter at testing phase. Good (1953, 2000) explains the details of this estimator.

Back-off smoothing. In estimating $\mathbf{P}(y|x_m)$, the simplest way to avoid zero counts is to lower the order m and look for a smaller context in $\mathcal{C}$. This is the back-off model proposed by Katz (1987) and the order is lowered until $x_m y$ is found in $\mathcal{C}$. In the extreme case, m goes to zero and it uses the unigram probability of predicting y when no context is observed. This has to be done such that the resulting values sum to one over all $y \in V$ for any given context. This requires weighting the lower orders by $\eta(x_m)$. The resulting distribution takes the following recursive form,

$$\mathbf{P}_{KZ}(y|x_m) = \begin{cases} \mathbf{P}(y|x_m) & \text{use highest order model if } c(x_m y) > \tau \\ \eta(x_m)\mathbf{P}_{KZ}(y|x_{m-1}) & \text{otherwise,} \end{cases} \tag{1.4}$$

where τ is a threshold chosen to avoid assigning mass to sequences that occur too few times. The value for τ could be chosen by cross-validating performance on held-out data but it is not very critical and is often set to zero (Chen and Goodman, 1996). The probabilities $\mathbf{P}(y|x_m)$ at the highest order can come from any estimator with uniform distribution for

the null-context ($m = 0$), $\mathbf{P}(y) = 1/|V|$. The weighting function $\eta(x_m)$ is computed by redistributing the left-over mass from the highest order distributions to the lower order distributions,

$$\eta(x_m) = \frac{\beta(x_m)}{\underbrace{\sum_{v \in \{z | c(x_m z) \leq \tau\}} \mathbf{P}_{KZ}(v|x_{m-1})}_{\text{redistributed to lower-order model}}}, \quad \beta(x_m) = \underbrace{1 - \sum_{v \in \{z | c(x_m z) > \tau\}} \mathbf{P}(v|x_m)}_{\text{left-over mass}}.$$

Interpolation smoothing. A natural extension of the back-off model is a weighted combination of the highest order and the lower order distributions. This was proposed by Jelinek and Mercer (1980) in their linear interpolation model that results in a distribution of the following recursive form,

$$\mathbf{P}_{JM}(y|x_m) = \gamma(x_m)\mathbf{P}(y|x_m) + (1 - \gamma(x_m))\mathbf{P}_{JM}(y|x_{m-1}).$$

We can recover the back-off equation in (1.4) for weighting $1 - \gamma(x_m) = \eta(x_m)$ when the corresponding lowest order distribution is $\mathbf{P}(y|x_{m-1})$. The weights $\gamma(x_m)$ are learned using held-out data and they should take values that increase with the frequency of the context x_m. But since there are too many values to be learned, they are pooled into buckets and all weights in a bucket are assigned the same value (Bahl et al., 1983). Witten and Bell (1991) argue that the interpolation weight $\gamma(x_m)$ for a certain context x_m should be proportional to the number of distinct words following the context, $C_{1+}(x_m\bullet) = |\{z \mid c(x_m z) \geq 1\}|$ with the $\bullet$ mapping to any word in V (also see Bell et al., 1990).

Kneser-Ney smoothing. A straightforward alternative to linear interpolation is to explicitly subtract some counts from the highest order distribution and appropriately reweigh the lower order distribution. This method introduced by Ney et al. (1994) is called absolute discounting and has the following form,

$$\mathbf{P}_{AD}(y|x_m) = \frac{\max\{c(x_m) - \delta,\ 0\}}{c(x_{m-1})} + (1 - \gamma(x_m))\mathbf{P}_{AD}(y|x_{m-1}),$$

where δ is the discount parameter tuned on validation dataset. The weighting function for the lower orders takes the form

$$1 - \gamma(x_m) = \frac{\delta C_{1+}(x_m\bullet)}{c(x_m)},$$

so that the mass of the resulting distribution adds to one. Performances of absolute discounting were comparable to that of linear interpolation with fewer parameters to tune (single parameter of δ) and it also agrees with intuition of Witten and Bell (1991) for weighting interpolated models. Kneser and Ney (1995) extended absolute discounting by using modified counts for the lower order distribution so that it is optimized for situations where the higher order counts are small. This modification follows the idea that the lower order probability should be proportional to the number of unique words preceding the

sequence. The resulting Kneser-Ney smoothing formulae are,

$$\mathbf{P}_{KN}(y|x_m) = \begin{cases} \overbrace{\frac{\max\{c(x_m y) - \delta,\ 0\}}{c(x_{m-1})}}^{\text{abs. disc.}} + \overbrace{\frac{\delta C_{1+}(x_m \bullet)}{c(x_m)} \mathbf{P}_{KN}(y|x_{m-1})}^{\text{back-off term}} & \text{largest } m \text{ s.t. } c(x_m y) > 0, \\ \underbrace{\frac{\max\{C_{1+}(\bullet x_{m-1} y) - \delta, 0\}}{C_{1+}(\bullet x_{m-1} \bullet)}}_{\text{modified lower order dist.}} + \underbrace{\frac{\delta C_{1+}(x_{m-1} \bullet)}{C_{1+}(\bullet x_{m-1} \bullet)} \mathbf{P}_{KN}(y|x_{m-1})}_{\text{interpolation term}} & \text{otherwise,} \end{cases}$$

where $C_{1+}(\bullet x_m y) = |\{z \mid c(z\ x_m y) \geq 1\}|$ counts the number of unique words preceding $x_m y$ and $C_{1+}(\bullet x_{m-1} \bullet) = |\{(z_1, z_2) \mid c(z_1\ x_{m-1}\ z_2) \geq 1\}|$ counts the number of distinct pairs sandwiching x_{m-1}. When x_{m-1} is null, $C_{1+}(\bullet\bullet)$ takes the number of bigrams in $\mathcal{C}$ as it value. The discount parameter δ is chosen by tuning on validation dataset or set using heuristics (Ney et al., 1994). This formulation when applied recursively up to unigrams is called interpolated Kneser-Ney. Kneser-Ney smoothing has been shown to consistently outperform other smoothing models (Chen and Goodman, 1996). Chen and Goodman (1998) also observe that varying the discount factor δ with the length of the sequence $|x_m y|$ it is applied on further improves performance.

Discussion on smoothing techniques. Smoothing techniques have long delivered good performance and are still preferred over other models for time intensive applications. Variants of Kneser-Ney are preferred over other methods owing to their superior performance. However, the suitability of a method depends on various factors like the amount of training data available, constraints on response time and so on. For example, Brants et al. (2007) show how they achieve performance close to Kneser-Ney for machine translation with a computationally cheaper and faster back-off model by significantly increasing the amount of data.

On the positive side, smoothing models are very well engineered and thoroughly studied models that are used in numerous applications but they lack certain desirable properties. Chen and Goodman (1996) study various smoothing models and observe that the flexibility offered by models with more parameters is one important aspect that improves performance. These models with parameters that are tuned on held-out data are observed to perform better than other ones. While this is the case with some conventional m-gram models, the flexibility they offer is very limited. The necessity of parametric models that can learn from data goes beyond improvements in predictive performance. They open possibilities to custom design models for specific tasks and fine tune them using data. We further discuss this in Section 1.5. This requirement motivates the modeling of language using parametric models. In the next section we describe such parametric modeling of language using the maximum entropy principle.

1.2.2 Maximum entropy models

The task is to estimate a distribution explaining the given data using a parametric model. 'Explaining data' is used to imply that the distribution must be similar to the empirical distribution observed from the data but not equivalent to avoid overfitting. More precisely, the expectation of random variables under the distribution should equal their empirical

observations from the data. There could be multiple such distributions with expectations matching empirical observations for any given dataset. The principle of maximum entropy states that from among various candidate distributions for a dataset, the one with highest entropy must be chosen. Entropy is the measure of uncertainty in a random process and a uniform distribution has the highest entropy. A distribution chosen by this criterion is closest to uniform distribution that satisfies the above said constraint for that dataset. Maximum entropy methods have been studied extensively in the statistical language processing literature (see Rosenfeld, 1998; Lau, 1994; Lau et al., 1993; Rosenfeld, 1996; Ratnaparkhi, 1998; Berger et al., 1996). Maximum entropy models bear relation to Gibbs and Boltzmann distributions in statistical physics, log-linear models and Markov random fields in probability theory and Boltzmann machines in neural networks. They offer interesting properties as described by Della Pietra and Della Pietra (1993).

Let $\{\delta_i(s)\}_{i=1}^d$ be a set of binary random variables or features defined over sequences s. We use contextual features where $\delta_i(s)$ indicates the occurrance of a certain suffix in the sequence s that we describe shortly. Alternatively, one could use a different feature set and build other models. For example, Lau (1994) uses the occurance of certain words called triggers in s to build a trigger-based language model. In a more generic setting, δ_i could take any non-negative value that quantifies a certain property like, for example, a quantity proportional to the length of s. If $\mathbf{P}_{ME}$ is the maximum entropy distribution then the expectation of $\delta_i(s)$ must satisfy,

$$\sum_{s \in \mathcal{C}} \mathbf{P}_{ME}(s)\delta_i(s) = \sum_{s \in \mathcal{C}} \mathbf{P}_{ML}(s)\delta_i(s), \tag{1.5}$$

where $\mathbf{P}_{ML}$ is the empirical distribution from equation (1.1). The maximum entropy distribution is one that satisfies all such constraints laid down by $\mathcal{C}$ and has the maximum entropy among them. Essentially, these methods choose a distribution that makes no assumptions beyond that of the data, in our case $\mathcal{C}$, and the choice of representation. We now describe a model based on this maximum entropy principle.

Consider a model with one parameter ω_i associated to each feature δ_i. Define an event T associated with a subset of the random variables $\mathcal{S}_T \subset \{\delta_i\}_{i=1}^d$. The maximum entropy model assigns a probability to T that is proportional to the product of the exponentials of all parameters ω_i corresponding to relevant features $\mathcal{S}_T$ or,

$$\mathbf{P}_{ME}(T) \propto \prod_{i \in \mathcal{S}_T} e^{\omega_i}. \tag{1.6}$$

The parameters of the model are estimated by maximizing the probability of observing the training data. We now look at the specific set of features and the estimation method for modeling language.

Maximum entropy language model. Consider a language model of order m that requires estimating $\mathbf{P}_{ME}(y|x_m)$. Let each sequence s of length at most $(m+1)$ in the training corpus $\mathcal{C}$ be a feature,

$$\mathcal{S} = \{s \mid s \in \mathcal{C} \wedge |s| \leq m+1\}.$$

Associate a parameter ω_i to each feature in $\mathcal{S}$ resulting in a model with $d = |\mathcal{S}|$ number of parameters. The feature set gets richer upon increasing the amount of data but so do the number of parameters that need to be estimated.

Given a sequence $(x_m y)$, our features $\{\delta_i(x_m y)\}$ are occurrences of $\mathcal{S}$ as suffixes of $(x_m y)$. This can be described by an indicator function $\mathbb{I}(x_m y, i)$ that assigns one to $\delta_i(x_m y)$ if $\mathcal{S}_i$ and $x_m y$ have a common non-null suffix, i.e., if $x_k y = \mathcal{S}_i$ for some $0 \leq k \leq m$. The indicator function $\mathbb{I}(x_m y, i)$ takes the following form:

$$\delta_i(x_m y) = \mathbb{I}(x_m y, i) = \begin{cases} 1 & \text{if } x_k y = \mathcal{S}_i \text{ for some } 0 \leq k \leq m, \\ 0 & \text{otherwise.} \end{cases}$$

Now, we need to write the probability $\mathbf{P}_{ME}(y|x_m)$ as defined in equation (1.6). This requires us to sum all parameters of features relevant to the event of y following the context x_m. The conditional probability is the exponential of this sum,

$$\mathbf{P}_{ME}(y|x_m;\omega) \propto e^{\sum_{i=1}^{d} \omega_i \mathbb{I}(x_m y, i)}.$$

Actual probabilities are computed by taking ratios with the *normalization factor*, also called the *partition function*, that comes from marginalization of y over all V,

$$Z(x_m;\omega) = \sum_{v \in V} e^{\sum_{i=1}^{d} \omega_i \mathbb{I}(x_m v, i)}. \tag{1.7}$$

The corresponding probability of generating a corpus which is also the likelihood of the corpus $L(\mathcal{C};\omega)$ would be,

$$L(\mathcal{C};\omega) = \mathbf{P}_{ME}(\mathcal{C};\omega) = \prod_{\{x_m y\} \in \mathcal{C}} \mathbf{P}_{ME}(y|x_m;\omega) = \prod_{\{x_m y\} \in \mathcal{C}} \frac{e^{\sum_{i=1}^{d} \omega_i \mathbb{I}(x_m y, i)}}{Z(x_m;\omega)}. \tag{1.8}$$

The optimal set of parameters is recovered by maximizing $L(\mathcal{C};\omega)$ which is usually done by first taking the logarithm. The negative of the log of the likelihood takes the following form:

$$\begin{aligned} \mathcal{L}(\mathcal{C};\omega) &= -\sum_{\{x_m y\} \in \mathcal{C}} \ln \mathbf{P}_{ME}(y|x_m;\omega), \text{ and} \\ \omega^* &= \operatorname*{argmin}_{\omega \in \mathbb{R}^d} \mathcal{L}(\mathcal{C};\omega), \end{aligned} \tag{1.9}$$

where ω^* is the set of optimal parameters in $\mathbb{R}^d$. We briefly review below the methods to solve the optimization in (1.9).

Iterative scaling methods. The optimization described in (1.9) is convex with a unique minimum and any suitable optimization method could be applied depending on the size of the problem. The earliest technique used is the iterative scaling introduced by Darroch and Ratcliff (1972). This method involves minimizing the difference in the likelihood at two different parameter estimates separated by $\Delta\omega$,

$$err = \mathcal{L}(\omega + \Delta\omega) - \mathcal{L}(\omega). \tag{1.10}$$

Method	Update equation $\Delta\omega := U(\omega, \mathcal{C}, i)$
GIS	$\Delta\omega_i := \frac{1}{\mathbb{I}^\#} \ln\left(\frac{\sum_{\mathcal{C}} \mathbb{I}(x_m y, i)}{\sum_{\mathcal{C}} \mathbf{P}_{ME}(y\|x_m)\mathbb{I}(x_m y, i)}\right)$
IIS	$\sum_{\mathcal{C}} \mathbb{I}(x_m y, i) = \sum_{\mathcal{C}} \mathbf{P}_{ME}(y\|x_m)\mathbb{I}(x_m y, i) e^{\Delta\omega_i \mathbb{I}^\#(x_m y)}$
SCIS	$\Delta\omega_i := \frac{1}{\mathbb{I}^M(i)} \ln\left(\frac{\sum_{\mathcal{C}} \mathbb{I}(x_m y, i)}{\sum_{\mathcal{C}} \mathbf{P}_{ME}(y\|x_m)\mathbb{I}(x_m y, i)}\right)$
CD	$\sum_{\mathcal{C}} \mathbb{I}(x_m y, i) = \sum_{\mathcal{C}} \frac{\mathbf{E}_{\mathbf{P}(v\|x_m)}\left[e^{\Delta\omega_i \mathbb{I}(x_m v, i)} \Delta\omega_i \mathbb{I}(x_m v, i)\right]}{\mathbf{E}_{\mathbf{P}(v\|x_m)}\left[e^{\Delta\omega_i \mathbb{I}(x_m v, i)}\right]}$

Table 1.1: Update equations for three different iterative scaling algorithms; generalized iterative scaling (GIS), improved iterative scaling (IIS), sequential conditional generalized iterative scaling (SCIS) and coordinate descent (CD) (see Huang et al., 2010).

Algorithm 1 Minimization in maximum entropy models (see Huang et al., 2010).

A. **Parallel update** (GIS and IIS)
A1 Input: $\mathcal{C}, \omega := 0$
A2 **repeat until termination**
A3 **for** $i = 1$ **to** d
A4 $\Delta\omega_i = U(\omega, \mathcal{C}, i)$ $\#\forall i$ as in Table 1.1
A5 $\omega \leftarrow \omega + \Delta\omega$

B. **Sequential update** (SCIS and coordinate descent)
B1 Input: $\mathcal{C}, \omega := 0$
B2 **repeat until termination**
B3 **for** $i = 1$ **to** d
B4 $\Delta\omega_i = U(\omega, \mathcal{C}, i)$ $\#\forall \mathcal{C}$ as in Table 1.1
B5 $\omega_i \leftarrow \omega_i + \Delta\omega_i$

Typically, the error is not easy to minimize and is replaced by a surrogate function that bounds the error from above and is easier to minimize (see Lange et al., 2000, on using surrogates for optomization). The derivative of the surrogate is then set to zero to derive the update equation. The update is applied iteratively until some termination condition is met. There are different variations of this approach depending how the surrogate is derived from the error function.

The original method proposed by Darroch and Ratcliff (1972) is called generalized iterative scaling (GIS). Della Pietra and Della Pietra (1993) proposed a different surrogate, they refer to as the improved iterative scaling (IIS). Both, GIS and IIS, lead to update equations that can be applied in parallel, meaning all parameters can be updated at each iteration of the optimization. Goodman (2002) derived another variation of the surrogate that results in an algorithm called the sequential conditional generalized iterative scaling (SCIS). This algorithm updates only one parameter at each iteration and cycles over the list of the parameters it updates. Using the following definitions for $\mathbb{I}^\#$, $\mathbb{I}^\#(x_m v)$ and $\mathbb{I}^M(i)$

as,

$$\mathbb{I}^{\#}(x_m v) = \sum_i \mathbb{I}(x_m v, i),$$
$$\mathbb{I}^{\#} = \max_{\mathcal{C}} \ \mathbb{I}^{\#}(x_m v) \text{ and}$$
$$\mathbb{I}^{M}(i) = \max_{\mathcal{C}} \mathbb{I}(x_m v, i),$$

the resulting update equations are tabulated in Table 1.1. The derivations leading to the corresponding surrogates for these methods are described in Appendix A.2. Huang et al. (2010) propose using coordinate descent (CD) as an alternative which directly minimizes the error in (1.10). CD also leads to sequential updates like SCIS. They also show that all these methods (iterative scaling and coordinate descent) converge to the optimum at a linear rate.

These procedures for optimization with both parallel and sequential updates are summarized in Algorithm 1. The parallel algorithm estimates the update for each parameter in the vector ω and applies them together in step (A5) updating all entries at every iteration. On the contrary, the update step (B5) for sequential algorithms is inside the for-loop with only one parameter ω_i updated at each iteration and the update value for the next parameter ω_{i+1} is estimated with the new estimate of ω.

Generic optimization methods. Generic function optimization techniques available in the literature (see Nocedal and Wright, 2006) were later found to be more suitable than iterative scaling to minimize the negative loglikelihood in (1.9). First order techniques use gradients in the parameter space to minimize the objective. The steepest descent method descends along the negative of the gradient with an appropriately chosen step size. In contrast, conjugate gradient method finds directions that are orthogonal to previous steps. Second order methods use the dense Hessian matrix to estimate an appropriately sized descent step. They offer faster convergence at the expense of computational time and memory. The number of entries in the Hessian grows quadratically with the size of the feature set. Further, its inversion is required that is cubic in the number of features.

In quasi-Newton methods, the inverse of the Hessian is approximated efficiently to overcome the computational burden of the Hessian. A rank-one approximation of the inverse of the Hessian is computed by using the last few gradients, a method that is popular as the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm. Malouf (2002) and Minka (2003) compared various numerical optimizers and demonstrated faster convergence of conjugate gradient methods and quasi-Newton methods over iterative scaling methods. However, when the dataset has large number of samples estimating a single gradient itself can be very time consuming. But it is unnecessary to use all available samples to make a reliable estimate of the gradient. In such cases, the gradient at each step is estimated stochastically from a small random subset of the dataset (or just one sample at a time) making it significantly faster. Stochastic methods were found to be more suitable for such optimization involving large data (see, for e.g., Finkel et al., 2008).

Relation to smoothing models. Unlike the smoothing models above, maximum entropy models learn the distributions from the data by fitting an exponential model to it. Chen and Goodman (1996) argue that this parameterization gives them significant advantage

over conventional m-gram models. Despite the difference, models resulting from maximum entropy estimation were shown by Chen and Rosenfeld (2000) to bear resemblance with interpolated and back-off smoothing models described above. Maximum entropy models also suffer from data sparsity and require smoothing of its estimates. Earliest works to smooth the estimates would modify the constraint set (1.5) which are later superseded by regularization of the parameters. We describe this in detail in the next section.

1.2.3 Predictive performance and computational efficiency

Maximum entropy models are simple and intuitive with flexibility to capture information of various types. For example, the model described above captures the information encoded in m-gram models and Lau et al. (1993) show how trigger-based models and cache models can also be absorbed into this framework. These models have been used for various kinds of language applications like disambiguation, sentence boundary identification, part-of-speech tagging, and numerous others. However, there are two major issues with the framework that need to be addressed; one relating predictive performance of the resulting model and the other of its computational efficiency. We discuss these aspects in detail below.

1. *Overfitting.* The parameters recovered by minimizing $\mathcal{L}$ very well fit the data in $\mathcal{C}$ but fitting it too well will cause it to degrade performance on any other sample $\mathcal{C}'$ that deviates from $\mathcal{C}$. This phenomenon of a model fitting the training data too well is called overfitting. Smoothing of parameters, similar to conventional n-gram models is used to avoid overfitting.

   Regularization by constraint modification. Smoothing in maximum entropy models involves modifying the marginal equivalence constraint in (1.5). It is important to note that if the modification leads to an inconsistent set of constraints, the problem becomes infeasible and the solution space will be empty. Different modifications lead to different changes in the feasibility set resulting in different solutions (Chen and Rosenfeld, 2000).

   The earliest choice studied for smoothing of parameter estimates was to drop some of the constraints from the set described by the corpus $\mathcal{C}$ in (1.5). This widely used method is called constraint exclusion and various variants of this idea have been studied by Della Pietra and Della Pietra (1993); Lau (1994); Rosenfeld (1996) and Della Pietra et al. (1997). Typically, constraints corresponding to sequences with very low frequencies are the ones that are excluded. The resulting model is observed to be very close to conventional n-gram models with a positive threshold on the frequency count (Lau, 1994; Rosenfeld, 1996). Replacing the empirical estimate on the right side of (1.5) with a discounted value is called Good-Turing smoothing of maximum entropy models as described by Lau (1994) and Rosenfeld (1996). The resulting model is known to be similar to Jelinek-Mercer interpolation smoothing for conventional n-grams models described above (Lau, 1994). Alternatively, Newman (1977) and Khudanpur (1995) relax the equality conditions in (1.5) to define an upper and a lower bound on the marginals under the resulting distribution $\mathbf{P}_{ME}$ to smoothen the estimates. The resulting inequalities are called fat constraints. An alternative is to optimize the unconstrained objective in (1.9) with an additional penalty term that adjusts for parameter smoothing.

Regularization by penalization. Smoothing by adding a regularization term (also called penalty) to the maximum likelihood formulation was considered as a simpler and efficient alternative to modifying the constraint set. Various regularizers have been tried in the context of maximum entropy models and a survey by Chen and Rosenfeld (2000) reported that they perform better than models smoothed by constraint modifications.

Chen and Rosenfeld (1999) propose using a Gaussian prior on the parameters which is equivalent to imposing an ℓ_2-norm on the parameters (Della Pietra and Della Pietra, 1993; Chen and Rosenfeld, 2000). Penalties similar to the ℓ_1-norm have been also been attempted by Goodman (2004). A combination of the two penalties, ℓ_1 and ℓ_2^2, similar to the elastic net has been tried by Kazama and Tsujii (2003) and Chen (2009). The ℓ_2 penalty and its combination with the ℓ_1-norm have shown improvements (Chen, 2009; Alumae and Kurimo, 2010) while the ℓ_1 penalty alone was less satisfactory (Goodman, 2004) indicating that uniform sparsity is not an appropriate prior for modeling language. These modifications lead to appropriate changes of the update equations in Table 1.1. We discuss the issue of overfitting in detail in Chapter 2 and then in Chapter 3, we propose using structure incorporating penalties to steer the optimization to a more appropriate model.

2. *Computation of the normalization factor.* The update at every step of the optimization in Table 1.1, requires estimating the expectation of some quantity under the distribution $\mathbf{P}_{ME}(\cdot)$ which depends on the current estimate ω. This means, we will need to recompute the expensive normalization factor Z at every iteration of the optimization. Wu and Khudanpur (2000) present a trick that improves the time required for computing Z by storing calculations that are common in multiple terms of Z. The time complexity of Z is still linear in the vocabulary V. Goodman (2001) reduces this dependence on V by clustering words. This breaks into into a two-stage problem of choosing the best cluster followed by picking the best word from the chosen cluster. This idea can be generalized to multiple levels of hierarchy that in the ideal case bring the complexity down to logarithmic in V. The hierarchy, however, should be such that the resulting classifier has predictive performance close to the original one. The hierarchy is usually built ad-hoc (Mnih and Hinton, 2008) or built using the standard Brown clustering (Brown et al., 1992; Goodman, 2001) . In Chapter 4, we propose an alternative to build this hierarchy that is very generic and applicable to build taxonomies for various problems.

1.3 Recent trends

In the recent years, language models have been developed based on Bayesian techniques and distributed representations (recurrent neural network, Boltzmann machine and others). We review some of these methods below.

1.3.1 Bayesian models

Numerous smoothing methods described above have performed well but offer very little insight in terms of the properties of word distributions they generate. It is well known

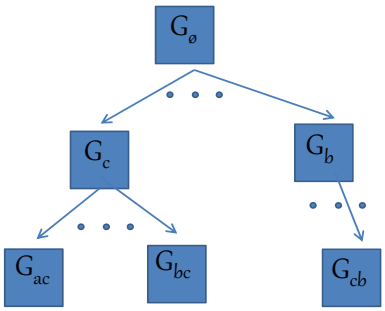


Figure 1.1: Hierarchical Bayesian models for language.

that the distribution of word frequencies in natural language follows a power law with the distribution decaying rather slowly, also called the heavy tail behavior (Zipf, 1932). This is because there are a small number of higher frequency words and a significant number of low frequency words occurring in every sampling of text. A generative model should be able to mimic this behavior of the distribution by putting considerable mass in its tail.

Hierarchical Dirichlet language model. The power-law makes Gaussian-like distributions inappropriate to model language since they decay too quickly. Mackay and Peto (1994) studied the suitability of Dirichlet distribution to generate heavy tails that could match language's behavior. They imposed a Dirichlet prior on the words and hierarchically stacked multiple layers of Dirichlet to model the probability of a word following various lengths of the context. Figure 1.1 shows a hierarchical model with three letter vocabulary $V = \{\texttt{a},\texttt{b},\texttt{c}\}$ where each node indicates a context. The distribution over the words in V following that particular context is given by G_x with $G_x(y) = \mathbf{P}(y|x)$ for $y \in V$. For example, $G_{bc} = [\mathbf{P}(y|\texttt{bc})]_{y \in V}$ is the vector describing the probabilities of words in V following the context bc. Mackay and Peto (1994) proposed this distribution to be drawn from a Dirichlet process, $G_x \sim DP(\alpha_x, B_x)$, with strength parameter α_x controlling the mass in the tail and B_x being the mean or base distribution (Ghosal, 2010). The base distribution at any given node will be the conditional distribution at the parent node, for example, $B_{\texttt{bc}} = G_{\texttt{c}}$ and $G_{\varnothing}(y) = 1/|V|$, the uniform distribution. This was the first work to develop a Bayesian model with an appropriately chosen prior over the words for modeling language and they reported favorable results. It was used develop an input mechanism that relies on touchscreen gestures or eye-tracking (Ward, 2001).

Pitman-Yor's power law. However, the Dirichlet process, despite being an improvement over others, would not satisfactorily match the distribution of word frequencies. Goldwater et al. (2006, 2011) showed that a Pittman-Yor process (Perman et al., 1992) is closer to the empirical distribution of word frequencies. They build a two stage morphological model to learn linguistic structure. The first stage is a generic model following any distribution, a multinomial in their case. This is then followed by the second stage of processing that produces distributions that are closer to natural language. They show that a Pitman-Yor process for the second stage results in better power-law distributions and improved performances. Pitman-Yor, unlike the Dirichlet, has no known closed form representation and the induced distribution is handled using an appropriate Chinese restaurant process or a stick-breaking process (Teh, 2006; Goldwater et al., 2011).

Hierarchical Pitman-Yor language model. Teh (2006) extended the work of Goldwater et al. (2006) to a language model by replacing the Dirichlet process in the hierarchical model of Mackay and Peto (1994) by a Pitman-Yor process. The distributions are now drawn from a Pitman-Yor process $G_x \sim PY(\delta_x, \alpha_x, B_x)$ with discount parameter δ_x, strength parameter α_x and mean distribution B_x. The resulting model with a Hierarchical Pitman-Yor process (HPY process) prior was found to deliver results comparable to the Kneser-Ney and also reveals interesting relations between them. The resulting equation of the HPY process model describing the generative procedure for drawing words was observed to functionally match that of the interpolated Kneser-Ney. This offers a principled derivation and an understanding of the smoothing technique that was otherwise ad-hoc, though effective.

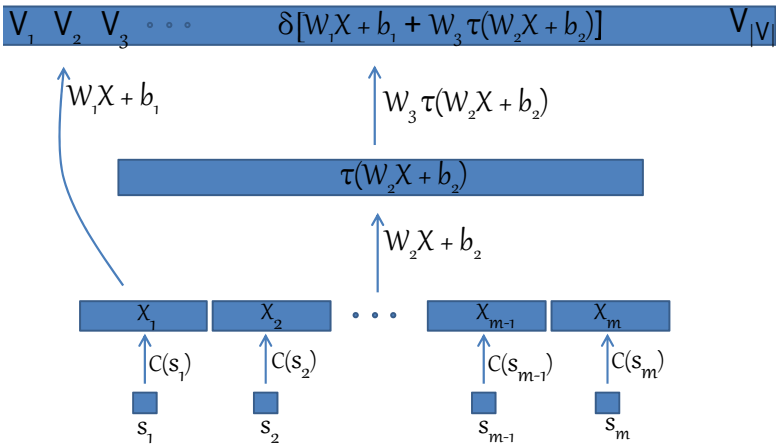
In the hierarchical structure of contexts as shown in Figure 1.1, there are often paths of non-branching nodes. The conditional distribution at each of these nodes follows a Pitman-Yor process. It is known that HPY processes are closed under marginalization and the resulting conditional remains a Pitman-Yor process. This property was used to marginalize out the non-branching nodes in the hierarchy to develop an infinite length Markovian model (or a non-Markovian model) by Wood et al. (2010, 2011), which they refer to as the sequence memoizer.

1.3.2 Distributed representations

Representing words often requires a 1-of-$|V|$ encoding with a long vector of zeros containing a single one indicating the occurrence of the word. This is because finding a continuous space feature representation for words is non-trivial. However, it is a sensible thing to do considering some words are semantically closer than others, something the 1-of-$|V|$ encoding does not reflect. Methods like latent semantic analysis represent words in a vector space (Deerwester et al., 1990) and are widely used in information retrieval and other areas. However, language models learned on such word features were found to be underperforming (Bengio et al., 2003). Models were designed to learn both the feature representation and the probability function jointly. These are referred in the literature as distributed representation models and offer competitive performances to traditional approaches. They minimize a complex energy function of the data and are often implemented with a neural network architecture. We will review three such models below.

Neural probabilistic language model. Bengio et al. (2003) propose the neural probabilistic language model built on feed-forward neural networks. They represent words in a finite d dimensional continuous feature space and express the joint probability of the sequences in the corpus in terms of these feature vectors. The log-likelihood of the training corpus under this model is maximized over the parameters of the feature representation and that of the probability function. Figure 1.2 illustrates the architecture of the network with the notation. It is a three layered network with input to hidden layer connections, hidden to output layer connections and also input to output layer connections.

Consider $s = (x_m y)$ with the context $x_m = (s_1, \cdots, s_m)$. Each word in x_m is mapped to a $\mathbb{R}^{d\times 1}$ as $X_i = C(s_i)$ which are concatenated into $X \in \mathbb{R}^{md\times 1}$ vector. The X vector makes the input that is mapped with W_1 to the output layer and with W_2 to the hidden layer. Offsets defined by vectors b_1 and b_2 are used in the two mappings. The hidden layer has h units and applies a hyperbolic tangent $\tau(\cdot)$ to its values which are then mapped onto the output space with W_3. The output layer has $|V|$ nodes, one per word in the vocabulary



$$
\begin{aligned}
C : V \rightarrow \mathbb{R}^{d\times 1}, X_i &= C(s_i)\\
X = [X_1 X_2 \cdots X_m] &\in \mathbb{R}^{md\times 1}\\
W_2 \in \mathbb{R}^{md\times h}, b_2 &\in \mathbb{R}^{h\times 1}\\
W_3 \in \mathbb{R}^{h\times |V|}, b_3 &\in \mathbb{R}^{|V|\times 1}\\
W_1 \in \mathbb{R}^{d\times |V|}, b_1 &\in \mathbb{R}^{|V|\times 1}\\
\tau(z) \triangleq z_i \leftarrow \tanh(z_i)&\\
\delta(z) \triangleq z_i \leftarrow \frac{e^{z_i}}{\sum_j e^{z_j}}&\\
\theta = (C, W_1, b_1, W_2, b_2, W_3)&
\end{aligned}
$$

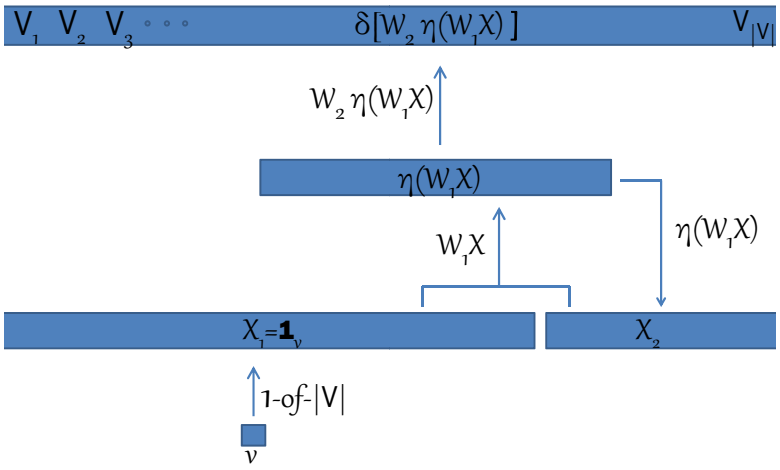
Figure 1.2: Network architecture of the neural probabilistic language model and the corresponding notation.

and contains the unnormalized probabilities of the target words ($\propto \mathbf{P}(y|x_m)$). These scores could be used straight away for some applications (Bengio and Senecal, 2003), but when probabilities are required a softmax $\delta(\cdot)$ is applied to the vector.

The parameters of this model θ are estimated using updates from the back-propagation algorithm. This neural model offered competitive performance to smoothing techniques outperforming Kneser-Ney. Additionally, the model returns a vector space representation for words which is a deviation from standard frequency tables. Smooth representation for words has the advantage of compactness and semantic meaningfulness but becomes challenging with polysemous words. The neural model has significantly more parameters than the traditional methods and has longer training times. Techniques like word clustering or importance sampling could be used to improve training times (see Bengio and Senecal, 2003; Schwenk, 2004; Schwenk and Gauvain, 2005; Morin and Bengio, 2005).

Log-bilinear language model. Mnih and Hinton (2007) proposed a model that minimizes a bilinear energy function similar to that of a restricted Boltzmann machine. They consider continuous space feature representations for words like Bengio et al. (2003) and capture the interactions between the words in their energy function. This function is a similarity score between a mapping of the context x_m onto the feature space and the features of the target word y. It has the following form,

$$
\begin{aligned}
E(x_m, y) &= -\left(\sum_{i=1}^{m} \mathbf{1}_{s_i} C^\top R_i\right) C\mathbf{1}_y^\top,\\
\text{where,}\quad & x_m = (s_1, \cdots, s_m),\\
& \mathbf{1}_v \in \{0,1\}^{1\times|V|} \text{ is a 1-of-}|V| \text{ encoding of the word } v,\\
& C \in \mathbb{R}^{d\times|V|} \text{ is a matrix of } d \text{ dimensional features for the words,}\\
& \text{and } R_i \in \mathbb{R}^{d\times d} \text{ captures the interaction of the } i^{th} \text{ word } s_i \text{ and the target } y.
\end{aligned}
$$



$$\begin{aligned}
\mathbf{1}_v &\in \{0,1\}^{1\times|V|} \\
X = [\mathbf{1}_v\ X_2] &\in \mathbb{R}^{|V|+h\times 1} \\
W_1 &\in \mathbb{R}^{|V|+h\times h} \\
W_2 &\in \mathbb{R}^{h\times|V|} \\
\eta(z) \triangleq z_i &\leftarrow \frac{1}{1+e^{-z_i}} \\
\delta(z) \triangleq z_i &\leftarrow \frac{e^{z_i}}{\sum_j e^{z_j}} \\
\theta &= (W_1, W_2)
\end{aligned}$$

Figure 1.3: Network architecture of recurrent neural network language model and the corresponding notation.

The resulting predictive distribution will be,

$$\mathbf{P}(y \mid x_m) = \frac{e^{E(x_m,y)}}{\sum_{v\in V} e^{E(x_m,v)}},$$

and the log-likelihood of the corpus is maximized to estimate parameters C and R_i. The optimization alternates between the maximization over the feature representations C and the context mappings R_i. The resulting model outperformed back-off Kneser-Ney and also has the advantage of learning a smooth representation for words. This model is similar to that of Bengio et al. (2003) but for the bilinear interactions which helped improve performance over the latter. Learning was further made to work faster using a tree-structured embedding of words in the hierarchical log bilinear model of Mnih and Hinton (2008). Bilinear functions were also be used to model multiple relations in language by Jenatton et al. (2012b).

Recurrent neural network language model. The distributed model of Bengio et al. (2003) is a fixed length Markov models where information in a sequence is processed in chunks limited by its maximal order. However, the recursive nature of language input as opposed to the fixed length input is not sufficiently captured by such models. This motivated models that explicitly characterize the recursive nature. Mnih and Hinton (2007) propose a temporal factored restricted Boltzmann machine that outperforms the log-bilinear model described above for short context lengths. But the temporal model has significantly higher number of parameters restricting its applicability to shorter contexts while the log-bilinear model would easily scale to moderately long lengths and outperform the temporal models. Bengio et al. (2003) suggest accommodating longer contexts by introducing a time-delayed layer in their network. Mikolov et al. (2010) investigate this in their recurrent neural network language model which is considerably simpler than that of Bengio et al. (2003) with fewer parameters to specify and significantly faster training times. It uses a simple three layer network similar to that of Bengio et al. (2003) but with recurrence in the middle layer. Figure 1.3 illustrates the corresponding network architecture with the notation.

Input to the network is 1-of-$|V|$ encoded words X_1 from the training sequence in that order. At any given iteration, the second layer is fed the input word along with its own output in the previous iteration X_2. The sigmoid function $\eta(\cdot)$ is applied at the second layer and the output is fed into the third layer with a mapping W_3. The last layer has as many nodes as there are words and contains unnormalized probabilities of generating the corresponding words. A softmax function gives out actual word probabilities.

In contrast to the architecture of Bengio et al. (2003), the strong difference lies in that the recurrent hidden layer imposes a time-delayed feedback. This time-delayed feedback conveniently takes into account the recursive dependence in the input. This model has outperformed interpolated Kneser-Ney smoothing and was applied to meeting recognition (Stefan et al., 2011) and to capture linguistic patterns (like words `king` and `queen` indicate to positions of similar stature but vary in gender (Mikolov et al., 2013)). The time to train the network was improved further by using techniques like word grouping and other optimizations (such as Mikolov et al., 2011).

1.4 Summary and motivation

In summary, numerous methods were developed following various techniques to model language. There are both advantages and disadvantages for each of these methods and some are more suitable than others for specific tasks. While all methods focus on generating the conditional probability distribution of a word given the preceding context, certain methods make modeling easier for specific problems. A typical language application requires multiple layers of processing. In such cases, numerous factors play role in determining the applicability of a particular model. Factors like ease of integration, or aspects of time and space complexity, or suitability to problem's domain like incorporating additional information and so on. We motivate the purpose of pursuing and improving various models with a particular focus on the class of discriminative models to which maximum entropy models (Section 1.2.2) belong. We use text generation in multimodal problems as an example to present our case. We then describe which aspects of discriminative models we address in this work and summarize our contributions.

Discriminative modeling of multi-modal data. In this broad view of model compatibility to applications, we see discriminative models as those that fit into scenarios that require flexibility to pool additional features to learn combined models easily. Take multi-modal applications with language being one of the modalities, more specifically the target modality. The other modality, being the source modality, could be an audio signal or image features or a video sequence involving both. Generating short descriptive summaries of images or other data is one simple application where this could arise.

Consider the problem of generating human-like captions for images that recently gained significant attention (see, for example, Gupta et al., 2009; Farhadi et al., 2010; Kulkarni et al., 2011; Yang et al., 2011; Mitchell et al., 2012). The first few models that were proposed used template-based construction of sentences from image tags returned by visual classifiers (Kulkarni et al., 2011). Template models are known to be suboptimal when textual variations are necessary (Reiter and Dale, 1997). Template filling has been replaced by models that generate syntactic trees that resulted in varied and well formed text (Mitchell et al., 2012; Yang et al., 2011).

However, these methods deal with the two modalities separately with a one-way information channel transferring the image signal summarized in a few words, mostly image annotations with some additional tags on spatial organization (Mitchell et al., 2012; Yang et al., 2011; Farhadi et al., 2010; Ordonez et al., 2011) sent across to the language model to make its sentence. The language part pools this oversimplified information together with as much textual data of its own to make a meaningful sentence that sometimes leads to completely incorrect sentences (Mitchell et al., 2012).

Making meaningful sentences that explain the source signals requires the model to leverage statistical strength (Li et al., 2011) of text corpora. Such models should also be amenable to accommodate source signals in determining the final distributions without the above over-simplification causing information loss. In tasks like this, we see discriminative models as a natural fit where we build a language model on available text from various sources and reuse that to learn a conditional model for our multi-modal data. Further, discriminative models allow for flexibility over aesthetics of language like its variations in style and tone, characteristics that manifest as terms in the objective function.

Encoding large context for discriminative modeling. Generating complete sentences in a discriminative framework requires handling long contexts, in some cases possibly involving multiple sentences. This could be necessary for various reasons. For example, we might want to suppress information covered in previous sentences to avoid repetitions, or prevent from reusing the same words in the same sentences to maintain good readability, or to control the order in which content is delivered and so on. In such situations, one would need a discriminative language model that can encode large sentences efficiently. This is one aspect that we target in our work.

Large multi-class log-linear models. The other is a much more generic problem involving log-linear models with a large number of classes. The bottleneck is always the estimation of the normalization factor in which the number of vector dot-products directly depends on the number of classes. This slows training since it requires to estimate the gradient and also slows the test time application of the model. A growing number of classes is a challenge in numerous fields especially with language models where the number of classes is the number of words in the vocabulary. Organizing classes into a hierarchy is one standard trick to alleviate this issue and significantly speeds up test time predictions. However, building such a hierarchy without loss of performance in comparison to the standard multi-class classifier is a challenge. We propose various methods to build such a hierarchy from samples and empirically study their behavior.

1.5 Contributions and outline

This work revisits traditional approaches in the light of recent advances in structured penalties and large-scale optimization. In particular, we focus on the two issues of discriminative language models concerning generalization performance and model complexity. Our contributions can be summarized as follows:

1. Discriminative language models are known to overfit for long context lengths even when regularized using standard penalty terms. This clearly reflects in the predictive

performance plots of models involving unstructured regularization on the subsets of AP-news corpus that we considered (see Figure 3.5). We study this issue to understand why simple regularization would not contain overfitting appropriately. To this end, we rewrite this model as a categorical distribution of words leading to a similar likelihood function as in (1.8) but with an extra constraint. The sequential structure of sentences is then interpreted on a suffix tree data structure. Penalties are then designed to exploit this tree structure inducing a structure-aware bias into the optimization. The optimization itself is described as operations on the suffix tree.

Models with structured penalties are evaluated on the same data as the unstructured models above and improvements in generalization clearly show up in predictive performance plots against increasing context lengths (see Figure 3.6). We further analyze the predictions of these models by grouping together words using various criteria to understand performance gains. As the plots reveal (see, e.g., Figure 3.8), there is scope for significant improvements which is a topic for future work.

On the complexity side, increasing context length leads to significant growth in the number of parameters. We study the efficiency of different penalties and some of them can be used to build models that scale efficiently with increasing context length. We propose an algorithm that improves the iteration time complexity for optimizing with such penalties over currently available methods. This work was published in the *Conference on Empirical Methods in Natural Language Processing 2013*.

2. The idea of grouping together words into clusters to make training and prediction time efficient has been used with different models (see (Goodman, 2001) for maximum entropy models, (Morin and Bengio, 2005) for feed-forward networks, (Mikolov et al., 2011) for recurrent networks and (Mnih and Hinton, 2008) for log-bilinear models). However, the method used to cluster words in the above models is either random or ad-hoc with the exception of Brown's clustering (Brown et al., 1992). We propose different methods to group words using their contexts. These methods are generic with applications beyond data of discrete sequences and can also be used with non-sequence data in continuous feature spaces. We study their behavior on image and text datasets and observe how tree-building methods must be chosen carefully to stem performance loss by switching from flat multi-class classifier to a tree-based classifier. Interestingly, in some cases, a suitable tree structure could also lead to improvements in predictive performance.

Outline of the thesis. The outline of the remaining chapters is as follows. Chapter 2 describes risk minimization framework and the necessity of regularization particularly focusing on structure incorporating penalties. It also considers algorithms for optimization with such penalties and studies various methods available. Chapter 3 details our contribution on log-linear models involving structured penalties built on top of the suffix tree data structure. It studies time and space efficiency of optimizing with different penalties. Chapter 4 describes the grouping methods that we propose for building hierarchies or taxonomies. We also present evaluation results on various datasets. We summarize and conclude in Chapter 5 with remarks and directions for future work.

Chapter 2

Learning with structured penalties

Statistical learning theory has been successfully used to develop algorithms for numerous applications including classification and regression tasks (see Vapnik, 1998, 1995, 1982; Hastie et al., 2009; Duda et al., 2001; Anthony and Bartlett, 1999; Breiman et al., 1984; Devroye et al., 1996; McLachlan, 1992; Natarajan, 1991, while there are numerous others dealing with this topic). The theory puts together statistical principles into an algorithmic framework that can be used to process data on machines. One such principle, and probably the most used of them, is the principle of *risk minimization* (see Vapnik, 1998, for details of the risk minimization framework). Consider a parametric model that is designed to achieve a certain task with parameters lying in a defined domain. Risk quantifies the *closeness* of a model to the defined goal in the space of parameters. It is measured by an empirical estimate over a certain sample set and hence the name *empirical risk*. This empirical risk is minimized using a *learning algorithm* to build the necessary classifier or regressor.

However, the effectiveness of this method relies on generating an appropriate and sufficiently large sample set to estimate the parameters of the model. In most scenarios, especially those involving high-dimensional spaces, it is not possible to generate a proper sample set. Minimizing the empirical risk too well is detrimental in such cases. To handle this overfitting, the risk function is modified to sacrifice some empirical risk by biasing the estimates in an appropriately chosen manner. This modification to risk function with a penalty or regularization term that induces the bias is referred to as the *structural risk*. The design of this regularization term is specific to the problem domain. Numerous choices for regularization with different properties have been investigated for various problems. We study some that are more suitable for language modeling than those considered earlier. The other issue in fitting an appropriate model involves designing an algorithm that minimizes this risk. We study algorithms necessary for risk optimization involving our choice of penalties.

In this chapter we detail the theory to achieve the goal of language modeling using structural risk minimization framework. We describe the principles of empirical risk minimization and structural risk minimization. We present various choices for penalty functions and choose the ones appropriate for out model. We briefly survey different algorithms available for their optimization and choose an appropriate method.

2.1 Principle of empirical risk minimization

The principle of *empirical risk minimization* was developed a few decades ago and has since been used successfully in numerous applications. Consider the event space $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ is the feature space and $\mathcal{Y}$ is the output space. We wish to recover the function that maps $\mathcal{X}$ to $\mathcal{Y}$ from a given set of observations. The idea is to design experiments and collect empirical data for testing multiple *hypotheses*, $h : \mathcal{X} \to \mathcal{Y}$ and pick the best fit. Hypothesis is also called the *predictor* that predicts $y \in \mathcal{Y}$ from $x \in \mathcal{X}$. In a more generic case, it could be a mapping to some transformation of the outputs as $\zeta(y) = h(x)$, where ζ is called the link function. Each hypothesis has a certain amount of expressiveness or the kind of relations it can capture. Greater expressiveness comes from more complex hypotheses involving more parameters.

The misfit of a particular hypothesis at any point $(x, y) \in (\mathcal{X} \times \mathcal{Y})$ is measured by a non-negative real valued *loss* function $f(\cdot, \cdot) : \mathcal{Y} \times \mathcal{Y} \to [0, \infty]$. The loss function f takes two arguments and returns a real value representing how different they are. Since our objective is to score the fitness of a hypothesis h for an event (x, y), we pass the predicted value of the hypothesis $h(x)$ and its actual value y to retrieve the score $f(h(x), y)$.

Define a probability density function $\mathbf{P}(\mathcal{X}, \mathcal{Y})$ over the space $\mathcal{X} \times \mathcal{Y}$. To compare multiple hypotheses, we must quantify each one over the complete space. *Risk*, $\mathcal{R}^f_{\mathbf{P}}[h]$, is the expectation of loss from h under the probability distribution $\mathbf{P}(\mathcal{X}, \mathcal{Y})$:

$$\mathcal{R}^f_{\mathbf{P}}[h] = \int_{\mathcal{X}\times\mathcal{Y}} f(h(x), y)\ d\mathbf{P}(x, y). \tag{2.1}$$

It is this risk that we seek to minimize over all possible mappings $\mathcal{Y}^{\mathcal{X}}$ to choose the hypothesis that best models the problem:

$$R^* = \inf_{h\in\mathcal{Y}^{\mathcal{X}}} \mathcal{R}^f_{\mathbf{P}}[h] \tag{2.2}$$

where $\mathcal{R}^*$ is the best achievable risk by the most suitable predictor, commonly referred to as the *Bayes risk*. We define as the excess risk e the difference between R^* and the risk from a hypothesis h, $\mathcal{R}^f_{\mathbf{P}}[h]$, a term quantifying how far we are from the best possible solution:

$$e = \mathcal{R}^f_{\mathbf{P}}[h] - \mathcal{R}^*.$$

Consider a set of hypotheses $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ of varying complexities. The complexity of functions $h \in \mathcal{H}$ defines the *capacity* of the class $\mathcal{H}$. Let the corresponding minimizer of (2.2) in $\mathcal{H}$ be,

$$h_{\mathcal{H}} = \operatorname*{argmin}_{h\in\mathcal{H}} \mathcal{R}^f_{\mathbf{P}}[h].$$

Now, for any $h \in \mathcal{H}$, the excess error e could be split into two components as follows:

$$\begin{aligned} e &= \mathcal{R}^f_{\mathbf{P}}[h] - \mathcal{R}^* \\ &= \left[\mathcal{R}^f_{\mathbf{P}}[h] - \inf_{i\in\mathcal{H}} \mathcal{R}^f_{\mathbf{P}}[i]\right] + \left[\inf_{i\in\mathcal{H}} \mathcal{R}^f_{\mathbf{P}}[i] - \mathcal{R}^*\right] \\ &= \underbrace{\left[\mathcal{R}^f_{\mathbf{P}}[h] - \mathcal{R}^f_{\mathbf{P}}[h_{\mathcal{H}}]\right]}_{e^{est}_h} + \underbrace{\left[\mathcal{R}^f_{\mathbf{P}}[h_{\mathcal{H}}] - \mathcal{R}^*\right]}_{e^{app}_{\mathcal{H}}}, \end{aligned} \tag{2.3}$$

of which $e_{\mathcal{H}}^{app}$ depends only on the choice of the class $\mathcal{H}$ and e_h^{est} depends only on h.

1. Error $e_{\mathcal{H}}^{app}$ is called the *approximation error* and quantifies how much worse $\mathcal{R}_{\mathbf{P}}^f[h_{\mathcal{H}}]$ is in comparison to the Bayes risk $\mathcal{R}^*$. This is due to the possible mismatch of the hypothesis class $\mathcal{H}$ for the problem being considered.

2. The other part of e depends on how far the solution h is from the optimum within $\mathcal{H}$. This component, e_h^{est} is called the *estimation error*.

Consider a feature space $\mathcal{X}$ and an output space $\mathcal{Y}$ with a joint population distribution of $\mathbf{D} = \mathbf{P}(\mathcal{X}, \mathcal{Y})$. However, we do not have access to this true distribution to evaluate quantities involving $\mathbf{D}$. We instead draw samples $\mathbf{S}_n = \{x_i, y_i\}_{i=1}^n \subset (\mathcal{X} \times \mathcal{Y})$ and use the corresponding empirical distribution $\mathbf{S}_n$ ($\sim \mathbf{D}$ as $n \to \infty$) to approximate the unknown distribution $\mathbf{D}$. Quantities involving $\mathbf{D}$ are approximated by an estimate from $\mathbf{S}_n$. Thus, we approximate the risk $\mathcal{R}_{\mathbf{D}}^f[h]$ in minimization (2.2) by the empirical risk defined as:

$$\mathcal{R}_{\mathbf{S}_n}^f[h] = \sum_{i=1}^{n} \frac{1}{n} f(h(x_i), y_i). \tag{2.4}$$

Now, consider a learning algorithm $\mathcal{A} : \mathbf{S}_n \to \mathcal{H}$ that takes observations $\mathbf{S}_n$ and returns a hypothesis from $\mathcal{H}$:

$$h_{\mathbf{S}_n} = \mathcal{A}(\mathbf{S}_n, \mathcal{H}) = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\ \mathcal{R}_{\mathbf{S}_n}^f[h]$$

and the corresponding estimation error as,

$$e_{\mathbf{S}_n}^{est} = \mathcal{R}_{\mathbf{D}}^f[h_{\mathbf{S}_n}] - \mathcal{R}_{\mathbf{D}}^f[h_{\mathcal{H}}].$$

Then we say, $\mathcal{A}$ is

1. *statistically consistent w.r.t.* $\mathbf{D}, f$ if for any $\epsilon > 0$

$$\mathbf{P}(\mathcal{R}_{\mathbf{D}}^f[h_{\mathbf{S}_n}] - \mathcal{R}_{\mathbf{D}}^f[h_{\mathcal{H}}] > \epsilon) \to 0 \text{ as } n \to \infty.$$

2. *Bayes consistent w.r.t.* $\mathbf{D}, f$ if for any $\epsilon > 0$

$$\mathbf{P}(\mathcal{R}_{\mathbf{D}}^f[h_{\mathbf{S}_n}] - \mathcal{R}^* > \epsilon) \to 0 \text{ as } n \to \infty.$$

The idea of consistency in both the above cases shows that for the particular $\mathbf{D}$ and f, the risk can be driven to arbitrary close to the optimal value ($\mathcal{R}_{\mathbf{D}}^f[h_{\mathcal{H}}]$ or $\mathcal{R}^*$ respectively) with high probability by using more samples for $\mathbf{S}_n$. When the above properties hold independent of the distribution $\mathbf{D}$, then they are called *universally consistent* learning methods.

The split of error into approximation and estimation terms can be used to understand an important trade-off between them. The approximation term is solely dependent on the set $\mathcal{H}$ but independent of the specific sampling $\mathbf{S}_n$. It lower bounds the best achievable risk using $\mathcal{H}$. The estimation term, on the other hand, depends on $\mathbf{S}_n$ and $\mathcal{A}$. In the ideal case when $\mathbf{S}_n \sim \mathbf{D}$, $h_{\mathbf{S}_n} = h_{\mathcal{H}}$, but this requires, $n \to \infty$.

Given a finite random sample set, it is fairly possible that $\mathbf{S}_n$ only crudely approximates $\mathbf{D}$. This is often the case in high dimensional space, where there is almost never

enough data to make an accurate estimate. In this situation, the minimizer of $e^{est}_{\mathbf{S}_n}$ and that of $e^{app}_{\mathcal{H}}$ are not equivalent, $h_{\mathbf{S}_n} \neq h_{\mathcal{H}}$. While $h_{\mathbf{S}_n}$ optimizes for the current sample set $\mathbf{S}_n$ and very well minimizes $e^{est}_{\mathbf{S}_n}$ it also leads to generalization error, also called *overfitting* since $\mathbf{S}_n \neq \mathbf{D}$. We could instead trade some $e^{est}_{\mathbf{S}_n}$ for significant reduction in $e^{app}_{\mathcal{H}}$ and, end up reducing the combined error e. This is the *approximation-estimation* trade-off or more classically, when using a square loss, *the bias-variance* trade-off.

The bias-variance split comes from considering the mean-squared error (**MSE**) of the estimator in (2.1) (Hastie et al., 2009):

$$\begin{aligned} \mathbf{MSE}[h_{\mathbf{S}_n} - h_{\mathcal{H}}] &= \mathbf{E}[(h_{\mathbf{S}_n} - h_{\mathcal{H}})^2] \\ &= \underbrace{\mathbf{Var}[h_{\mathbf{S}_n}]}_{\text{variance}} + \underbrace{(\mathbf{E}[h_{\mathbf{S}_n}] - h_{\mathcal{H}})^2}_{\text{square of bias}}. \end{aligned}$$

Approximation error $e^{app}_{\mathcal{H}}$ corresponds to squared bias term and cannot be improved without modifying the hypotheses $\mathcal{H}$. Estimation error e^{est}_{h} corresponds to the variance of this estimator since it depends on the sample size n and the algorithm used for estimation. In this interpretation, we wish to trade a little bias and make a more confident estimate with large reduction in variance.

The trade-off is made by appropriately modifying the class $\mathcal{H}$. Increasing the complexity of the functions in $\mathcal{H}$ results in a lower $e^{app}_{\mathcal{H}}$ and a higher $e^{est}_{\mathbf{S}_n}$ and vice versa. Large capacity of $\mathcal{H}$ shoots up $e^{est}_{\mathbf{S}_n}$ leading to overfitting described above. On the contrary, $\mathcal{H}$ with small capacity will lead to *underfitting* due to high $e^{app}_{\mathcal{H}}$. This trade-off is balanced by controlling the capacity of $\mathcal{H}$ such that the resulting error e is minimal. The balance is achieved using an additional term in the risk function in (2.4) that penalizes complex hypotheses. We discuss specific details of this penalization in the next section.

2.2 Penalized loss and structural risk minimization

In order to ensure good predictive performance, we need to minimize the excess risk e in (2.3). Since, we only have access to a finite sample of the actual distribution, this generalization is achieved by controlling the capacity of $\mathcal{H}$. Define a parametric family of hypotheses $\mathcal{H}(\lambda)$ with the parameter λ controlling the capacity of the resulting class. We modify the empirical estimate of the risk in (2.4) to append the loss f with a penalty term $\Omega : \mathcal{H} \to [0, \infty]$ that depends only on h and not on the sample $\mathbf{S}_n$. The resulting function called the *structural risk* is,

$$\mathcal{R}^{f+\lambda\Omega}_{\mathbf{S}_n}[h] = \sum_{i=1}^{n} \frac{1}{n} f(h(x_i), y_i) + \lambda\Omega(h). \tag{2.5}$$

Parameter λ controls the weighting of Ω in comparison to f and we recover the empirical estimate in (2.4) by setting $\lambda = 0$. The penalty Ω maps the hypotheses to the real number line by order of their increasing complexities. Increasing λ will encourage functions with smaller complexities and lowers the capacity of $\mathcal{H}$. The set of hypotheses that is included in $\mathcal{H}(\lambda)$ for a given λ is easier to understand in the following equivalent form of (2.5):

$$\mathcal{R}^{f,\Omega\leq\delta}_{\mathbf{S}_n}[h] = \begin{cases} \sum_{i=1}^{n} \frac{1}{n} f(h(x_i), y_i) & \text{if } \Omega(h) \leq \delta, \\ \infty & \text{otherwise.} \end{cases} \tag{2.6}$$

Clearly, $\mathcal{H}(\lambda)$ of (2.6) is the set of all hypotheses h for which $\Omega(h)$ is smaller than some δ. There is a one-to-one mapping from λ to δ, however, it is not obvious to compute one from the other for any given Ω. The term $\lambda\Omega$ is also called the *regularization* term and plays a role similar to that of the *prior* in methods involving *maximum a posteriori* estimation. Since the penalty adds some bias in return of variance reduction, the estimator in (2.5) is a biased estimator.

The choice of loss function depends on the specific goal of the model, its suitability to data and ease of optimization. In some cases, the loss might directly describe the problem like square loss for regression or 0-1 loss for binary classification. One might even minimize a complex loss function that is more suitable for the problem (like area under ROC-curve or, MAP, NDCG measures[1] (Chen et al., 2009; Ravikumar et al., 2011), or τ-quantile (Boyd et al., 2012)). However, considering ease of optimization a non-convex loss is often replaced a convex surrogate that is more amenable to optimization like hinge loss, logistic loss (multinomial for multi-class), huber loss, absolute loss, perceptron loss (see Steinwart, 2002; Mannor et al., 2002; Zhang, 2003; Jiang, 2003; Lugosi and Vayatis, 2003; Bartlett et al., 2003, and other references therein for statistical consequences of various loss functions). The other choice to be made for the risk minimization framework is that of the penalty that incorporates our belief into the learning algorithm of what the parameters should look like. We discuss various options for penalty functions and their effect in the following sections.

To simplify notation, let hypothesis set be parameterized by a vector w as $\mathcal{H} = \{h^w\}$. In the case of linear predictors this would be $\zeta(y) = h^w(x) = w^\top x$. The dependence could be more complex with $h^w(x)$ being any function of w and x. Hereafter, we refer to the penalty $\Omega(h^w)$ in terms of the parameter vector as $\Omega(w)$.

2.2.1 Unstructured penalties

Regularization has become a standard component in almost every model and until about a couple of decades ago it was almost always the ℓ_2-norm. It offers additional benefits in some cases like adding stability to ill-conditioned problems which was how it was first introduced by Tikhonov, popularized as Tikhonov regularization. Lasso that was introduced later has been shown to be helpful in practice for which some theoretical results have also been proven like consistency under certain conditions (see for example Zhao and Yu, 2006; Zou, 2006; Wainwright, 2006). Since these penalties treat all parameters equally and regularize them uniformly, we refer to them as *unstructured penalties*. We describe below some unstructured penalties from the literature.

Ridge regularization. The most commonly used penalty is the Euclidean ℓ_2-norm ($\Omega(w) = \|w\|_2$) or its squared value ($\Omega(w) = \|w\|_2^2$) that is called the *Ridge* regularization. It penalizes the magnitude of the weight vectors and encourages vectors closer to the origin. $\mathcal{H}(\lambda)$ in this case is all points lying inside the ℓ_2-ball of δ units as seen from the alternative interpretation in (2.6). Expectedly, the ℓ_2-norm shrinks the weight vectors towards the

[1] ROC is receiver operating characteristic, MAP is min-average precision and NDCG stands for normalized discounted cumulative gain

origin and does so uniformly in all directions. The ℓ_2^2 penalty behaves similarly and the choice among the two is made by convenience of optimization.

Lasso estimator. The ℓ_1-norm is the other common penalty that gained popularity for $\mathcal{H}(\lambda)$ that involves sparse vectors. ℓ_1 penalty encourages sparsity by setting small values to zero that an ℓ_2 penalty would leave with small magnitude. This made ℓ_1 particularly interesting for problems where the actual solution is sparse as is often the case in large feature spaces. The estimator in (2.6) with $\Omega(w) = \|w\|_1$ constrains the hypothesis class to an ℓ_1-ball of δ units. This sparsity inducing penalty leads to solutions similar to that of the popular *Lasso* estimator in statistics (Tibshirani, 1996) and *basis pursuit* (Chen et al., 1998b) in signal processing and is related to compressed sensing (Candes and Wakin, 2008).

ℓ_1-norm has been shown to perform particularly well when sparsity is a valid assumption. In some cases it offers more than just better performance. ℓ_1-norm provides interpretability which is important for some applications involving large feature spaces. It also allows for high-dimensional inference with the number of observations logarithmic in the size of these observations.

Elastic net and fused lasso. Mixed norms were introduced to pool together properties from different penalties. *Elastic net* ($\ell_{En(\gamma)}$) proposed by Zou and Zhang (2009) uses both ℓ_1 and ℓ_2-norms on the features:

$$\Omega(w) = \|w\|_{En(\gamma)} = \|w\|_1 + \gamma\|w\|_2^2.$$

Its advantage comes from both feature selection using ℓ_1-norm and shrinkage of selected features from ℓ_2-norm. *Fused Lasso* ($\ell_{Fl(\gamma_1,\gamma_2)}$) was proposed by Tibshirani et al. (2005) which further adds a term to the elastic net that encourages piece-wise constant behavior by smoothing out minor changes in consecutive entries of w and retaining sufficiently large ones as controlled by γ_2:

$$\Omega(w) = \|w\|_{Fl(\gamma_1,\gamma_2)} = \|w\|_1 + \gamma_1\|w\|_2^2 + \gamma_2 \sum_{i=1}^{|w|-1} |w_{i+1} - w_i|.$$

Non-convex penalties. All penalties considered above are convex functions, however, non-convex penalties are used when more appropriate. The $\ell_0(w)$ penalty, for example, maps a vector to its support (the number of non-zero entries):

$$\ell_0(w) = \mathbf{supp}(w) = \sum_i \mathbb{I}(w_i \neq 0).$$

ℓ_0 is not a proper norm (often called a pseudo-norm) since it is not positive homogeneous, $\alpha||w||_0 \neq ||\alpha w||_0$ for $\alpha \geq 0$ and is non-convex which complicates optimization with troublesome local minima in a combinatorially large space. The ℓ_1-norm which is the tightest convex approximation for the ℓ_0 penalty is used as a surrogate for ℓ_0. However, non-convex ℓ_0 based methods are sometimes preferred and used with approximate greedy algorithms like *orthogonal matching pursuit* (Tropp and Gilbert, 2007) and compressed sensing (Haupt and Nowak, 2006; Baraniuk et al., 2008; Huang et al., 2009) in signal processing.

Matrix penalties. In cases where the hypotheses are matrices, matrix norms are used. The nuclear norm or trace-norm (sum singular values) (Fazel et al., 2001) is used as a convex surrogate to the NP-hard rank minimization problem. Such penalties have been applied to numerous problems like multi-class categorization (Crammer and Singer, 2002), multi-task learning (Agarwal et al., 2008), and matrix completion (Srebro et al., 2005).

Building upon the simpler norms, more theoretically appealing variants have been developed (like adaptive Lasso (Zou, 2006), relaxed Lasso (Meinshausen, 2008), thresholding (Lounici, 2008), Bolasso (Bach, 2008a), stability selection (Meinshausen and Buhlmann, 2008.; Wasserman and Roeder, 2009)). Mixed and hybrid penalties that lie between ℓ_1 and ℓ_2 have also been studied. Apart from the ones mentioned above, various other penalties like the garotte (Yuan and Lin, 2007), bridge (Fu, 1998) and "Berhu" (reverse of Huber function) (Owen, 2006) can also be found in the literature.

Improved generalization from a penalty depends on its suitability (of properties like sparsity) to the data at hand. When suitable, the penalty uses the additional information to reduce the capacity of $\mathcal{A}$ with a more educated estimate of the solution space. While sparsity is the only additional information we have seen with ℓ_1, ℓ_0 penalties, sometimes there is more information available than just sparsity. Information like which groups of features co-occur, or a specific pattern in which zeros occur, or an ordering of its values and so on. This information can be used to further shape-up the solution space by suitably improving the penalty's design. We look at some structured penalties in the next section that help incorporate such additional information by redesigning the penalty.

2.2.2 Structured penalties

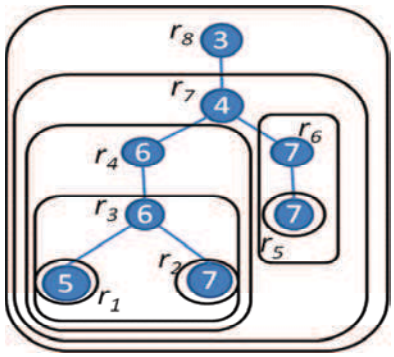
Structured penalties carve the space of hypothesis by using additional information from the problem domain. The earliest structure was in the form of specific patterns in which zeros occur in the solutions and hence the name *structured sparsity*. This was later generalized to other structures involving trees and graphs. We look at some examples in this section that we will use in later chapters.

Group lasso. The simplest form of structure that could be imposed is grouping of variables into non-overlapping sets. This was introduced by Yuan and Lin (2006) which they call the group Lasso. It imposes an ℓ_1-norm on non-overlapping groups of variables and ℓ_2-norm on variables within each group. This was further modified to a norm that uses an ℓ_∞-norm within each group. We define a grouping on the features as follows:

Definition 1. *Let $G^P = \{g_i\}_{i=1}^{|G^P|}$ be a partition (mutually exclusive and collectively exhaustive subsets) of the set of indices $I = \{1, \cdots, |w|\}$ i.e., $\bigcup_{g \in G^P} g = I$ and $g_i \cap g_j = \varnothing$. G^P defines a grouping on the entries of w.*

Let $w_g \in \mathbb{R}^d$ be a vector with entries from w corresponding to g and zeros everywhere else, *i.e.*, $w_g(i) = w_i \mathbb{I}[i \in g]$ where $\mathbb{I}[\,\cdot\,]$ is the indicator function. Now, $\Omega(w)$ takes the following form for group Lasso:

$$\Omega(w) = \ell_p^{G^P}(w) = \sum_{g \in G^P} \|w_g\|_p. \tag{2.7}$$



$$w_{g_8} = [3\ 4\ 6\ 6\ 5\ 7\ 7\ 7]$$
$$w_{g_7} = [4\ 6\ 6\ 5\ 7\ 7\ 7]$$
$$w_{g_5} = [7]\,,\ w_{g_6} = [7\ 7]$$
$$w_{g_3} = [6\ 5\ 7]\,,\ w_{g_4} = [6\ 6\ 5\ 7]$$
$$w_{g_1} = [5]\,,\ w_{g_2} = [7]$$

Figure 2.1: An eight-dimensional vector with a tree structure and the corresponding groupings w_{g_i} corresponding to subtree g_i with root r_i. The numbers $3, 4, 5, 6, 7$ shown are examples of values those components of w can take.

The superscript G^P defines the partitioning of features and the subscript p indicates to the norm imposed on each partition of features w_g. The penalty across various groups is always the ℓ_1-norm also with other group penalties defined below, hence, we do not indicate this with a separate subscript. Such a norm either selects or discards a group completely. This is easy to optimize with non-overlapping feature groups but complicates with overlapping groups. But some overlaps are easier to handle than others.

Overlapping group lasso. A natural generalization of the group Lasso is to have groupings with feature overlap, as studied by Zhao et al. (2008) (see also Jenatton et al., 2011a). We define the grouping with overlaps, G^O as follows.

Definition 2. *Define a grouping with overlaps* $G^O = \{g_i\}_{i=1}^{|G^O|}$ *such that* $g_i \subset I = \{1, \cdots, |w|\}$ *and* $\bigcup_{g \in G^L} g = I$.

An index in I could belong to one or more groups making the intersection of groups possibly non-empty. But each index must belong to at least one group. The group Lasso-like formulation defined on G^O is the overlapping group Lasso:

$$\Omega(w) = \ell_p^{G^O}(w) = \sum_{g \in G^O} \|w_g\|_p. \tag{2.8}$$

The grouping G^O defines a set of overlapping groups that is not a distinct partitioning of features as in group Lasso defined in (2.7). Each group w_g has an ℓ_p-norm imposed on its features.

This formulation also tends to drive groups of variables to zeros together but with a difference due to the overlap. A feature could belong to multiple groups and the feature is excluded from the final solution unless all the groups containing this feature are selected. So, only those features that belong to none of the zeroed groups belong to the final solution. The support of the penalty is the complement of the union of all the discarded groups. If G_0 is the subset of groups with a zero norm, then the support is,

$$\mathbf{supp}(\hat{w}) = \left(\bigcup_{g \in G_0} g \right)^c \text{ for some } G_0 \subset G^O.$$

One special case of the overlapping group Lasso that is of particular interest, is where the overlap is a nested hierarchy, referred to as the tree-structured group Lasso.

Tree-structured group lasso. Tree-structured norms were introduced by Zhao et al. (2008) and further studied by Jenatton et al. (2011b) enforce a tree structure on the variables. This is similar to the group lasso penalty but with overlapping groups organized in a tree structure. Each subtree of the tree forms a feature group. Feature groups in this case have a nested hierarchical structure.

Definition 3. *Let w be a vector with a tree structure $T = \{\mathcal{V}, \mathcal{E}\}$ with $\mathcal{V}$ vertices and $\mathcal{E}$ edges, where each entry w_i belongs to a vertex $v \in \mathcal{V}$. Define grouping $G^T = \{g_v\}_{v \in \mathcal{V}}$ where $g_v \subseteq \{1, \cdots, |w|\}$ is the set indices in the subtree rooted at v.*

Now, $\Omega(w)$ takes the following form for tree-structured penalty:

$$\Omega(w) = \ell_p^{G^T}(w) = \sum_{g \in G^T} \|w_g\|_p. \tag{2.9}$$

Similar to the group lasso, the tree structured penalty is often used with $p = 2, \infty$. Figure 2.1 illustrates an eight dimensional tree-structured vector along with the corresponding groups. The support with this penalty is as defined with the overlapping group Lasso that translates to a property that drives all descendants of a zero valued node to zeros. It does this by penalizing parameters lower in the tree more than those above resulting in solutions that usually have smaller values below. The groups in G_0 are the subtrees lower in the tree all variables of which have been shrunk to zero. Hereafter, we refer to this penalty as the ℓ_p^T-norm with the tree structure T replacing G^T for convenience.

Latent group Lasso. Obozinski et al. (2009a) study a different kind of overlapping group Lasso formulation where the support is the union of a subset of groups rather then their complement. They call this the latent group Lasso since they define latent groups of feature variables, the sum of which gives the final weight vector.

Definition 4. *Define groups with overlaps $G^L = \{g_i\}_{i=1}^{|G^O|}$ such that $g_i \subset I = \{1, \cdots, |w|\}$ and $\bigcup_{g \in G^L} g = I$. For each group $g \in G^L$, define a vector $v_g \in \mathbb{R}^{|w|}$ such that $\mathbf{supp}(v_g) \subset g$ and $w = \sum_{g \in G^L} v_g$.*

The feature grouping here is similar to that of the overlapping Group Lasso above. Additionally, we define vector v_g which is the latent vector corresponding to the feature grouping g. The definition of the latent group Lasso penalty is as follows,

$$\Omega(w) = \ell_p^{G^L}(w) = \sum_{g \in G^L} \|v_g\|_p. \tag{2.10}$$

Since a feature could belong to multiple latent vectors v_g and its contribution from multiple selected groups is summed together to get its value in w, a feature is selected if at least one group containing it has a non-zero norm. This property is in contrast to that of the

overlapping group Lasso above. As a consequence, the support is the union of a subset of groups G_1 that have a non-zero norm:

$$\mathbf{supp}(\hat{w}) = \bigcup_{g \in G_1} g \text{ for some } G_1 \subset G^T.$$

This was then extended to graph Lasso, the case with a graph structure on the parameters by grouping those parameters that share an edge.

Tree-guided group lasso. In a regression model with multiple outputs, it is more appropriate to assume that each output variable depends on a subset of features specific to itself, that varies from feature groups of other output variables. This motivates considering sparsity in features across output variables unlike other penalties for multi-task problems that only consider sparsity among features but not specific to outputs.

Consider the case where we are given a hierarchy defined over the output variables such that each node in the hierarchy groups a subset of the output variables. The leaves of the hierarchy are output variables. Kim and Xing (2010, 2012) describe a method that uses such a hierarchy to define a penalty that at each node considers only those features that are relevant to the output variables belonging to that node in the hierarchy. Let $y_k, k \in [1, \cdots, K]$ be the K output variables. We now have one vector w_k for each of the output variables with I features $\left[w_k^1, \cdots, w_k^i, \cdots, w_k^I\right]$.

Definition 5. *Let $R = \{G^R, E^R\}$ be a tree over output variables $\{y_k\}_1^K$ with nodes G^R and edges E^R such that*

- *for $g \in G^R$, $g \subset [1, \cdots, K]$ and $g = \bigcup_{h \in \chi(g)} h$, where $\chi(g)$ are children of g in R,*
- *and leaves of R are output variables.*

Define w_g^i as the feature set with index $i \in I$ corresponding to the output variables $k \in g$.

For each node in the hierarchy, the penalty defined by tree-guided group Lasso, $\ell_p^{G_R}$ imposes an ℓ_p-norm on all the features of a given index i that belong to output variables at that node and and ℓ_1 across feature indices:

$$\Omega(w) = \ell_p^{G^R}(w) = \sum_i \sum_{g \in G^R} \lambda_g \|w_g^i\|_p. \tag{2.11}$$

There is an additional weight of λ_g for each group to ensure that the penalization of various output variables is balanced when features overlap. This penalty induces sparsity in feature sets across output variables.

All the above mentioned structured penalties are convex functions that are good for optimization. However, they are theoretically harder to analyze and few such results are available (see Bach, 2008b; Nardi and Rinaldo, 2008; Chesneau and Hebiri, 2008; Nardi and Rinaldo, 2012; Percival, 2012).

An alternative approach is to directly solve the non-convex penalties involving ℓ_0 pseudo-norm with appropriate groupings. In this direction, the greedy approach of orthogonal matching pursuit has been extended to incorporate structural properties Huang

et al. (2011); Koltchinskii and Yuan (2008); Huang and Zhang (2010). Sparsity (both structured and unstructured) has also been considered in the Bayesian framework by Wipf and Rao (2005, 2007); Argyriou et al. (2008); Ji et al. (2009) and He and Carin (2009).

Application of structured penalties. Structured penalties have found application in numerous problems like dimensionality reduction (sparse PCA (Zhang et al., 2008; d'Aspremont et al., 2008), CCA (Witten et al., 2009), matrix factorization (Bach et al., 2008)), dictionary learning (sparse decompositions (Olshausen and Field, 1997; Elad and Aharon, 2006; Lee et al., 2007), with online learning (Mairal et al., 2009a), simultaneous sparse approximation (Mairal et al., 2009b; Tropp et al., 2006a,b)), multi-task learning (joint variable selection with group lasso (Obozinski et al., 2009b), joint feature selection with trace-norm (Argyriou et al., 2007), with fewer observations than tasks Obozinski et al. (2008); Lounici et al. (2009), shared dictionary learning with ℓ_1-norm (Mairal et al., 2009c); learning shared structures in multi-class classification with trace-norm (Amit et al., 2007; Harchaoui et al., 2012)), computer vision (image denoising (Elad and Aharon, 2006; Mairal et al., 2009b), image classification with learned code books (Boureau et al., 2010; Yang et al., 2009, 2010; Mairal et al., 2008; Leordeanu et al., 2007) inverse half-toning (Mairal et al., 2012), digital zooming (Couzinie-Devy et al., 2011)), speech processing (source separation Benaroya et al. (2006)), neuroimaging (resting state activity (Varoquaux et al., 2010), M/EEG source localization (Gramfort et al., 2012), multi-scale mining fMRI (Jenatton et al., 2012a)), language processing (chunking, entity recognition, and dependency parsing (Martins et al., 2011)), sociolinguistics (Eisenstein et al., 2011)), bioinformatics (Rapaport et al., 2008; Kim and Xing, 2012), semi-supervised learning (Raina et al., 2007), structure learning (Lee et al., 2007), graphical model selection with ℓ_1-norm (Ravikumar et al., 2010) and time-series prediction (Nardi and Rinaldo, 2011). In the next section we give some definitions and discuss specific tree-structured norms that we will use in later chapters.

2.3 Optimizing penalized loss

We now focus on minimizing the risk in (2.5) with the penalized loss over the observations $\mathbf{S}_n$. The optimization is rewritten in terms of the parameters of the hypothesis space w and the average of loss over samples is replaced by $\mathcal{L} = \sum_{i=1}^{n} \frac{1}{n} f(h(x_i), y_i)$:

$$\widehat{w} = \operatorname*{argmin}_w \mathcal{R}_{\mathbf{S}_n}^{f+\lambda\Omega}[w] = \operatorname*{argmin}_w \mathcal{L}(w, \mathbf{S}_n) + \lambda\Omega(w). \tag{2.12}$$

We briefly review different optimization methods for generic loss functions and detail the specific approach we use for our application. Bach et al. (2012) present a detailed review of various optimization techniques for various sparsity-inducing penalties.

In the case of convex smooth f and Ω, any gradient or Newton based method is appropriate. However, most penalties we considered above are convex but non-smooth, ruling out a simple gradient based algorithm. Similar issues arise with non-smooth loss functions (like the 0-1 or hinge loss) that are not easily amenable for gradient based methods. Various techniques have been adapted to suit specific cases of loss and penalty combinations. Numerous methods are employed to solve the large-scale optimization problems like breaking it down into smaller subproblems or exploiting the problem's structure or using low-rank approximations and so on (see Sra et al., 2011). We restrict our discussion to variations in penalties that we considered above and assume the loss is convex and smooth.

Gradient and Newton methods. If f and Ω are both smooth, the simplest algorithm to use would be steepest descent (or gradient descent) that offers $O(1/t)$ rate of convergence for general convex differentiable functions with Lipschitz continuous gradients (Bertsekas, 2004), with a potential acceleration to $O(1/t^2)$ (Nesterov, 1983). Convergence can be improved further to linear rate, $O(\rho^t)$ for strongly convex objectives. In problems where the Hessian is computable (twice differentiability) and invertible (well-conditioned and computationally feasible), second-order Newton methods would lead up to quadratic rate of convergence of $O(\rho^{2^t})$ (Bertsekas, 2004). Quasi-Newton methods are used to avoid inverting the Hessian that approximates it from successive estimates of the gradient. In cases where storing the dense Hessian is also infeasible, the inverse is locally approximated like using rank-one updates, also called the L-BFGS algorithm (Nocedal and Wright, 2006).

However, these methods are no longer appropriate when the objective is non-smooth. The quasi-Newton method of orthant-wise limited-memory quasi-Newton (OWLQN) (Andrew and Gao, 2007) is an exception that handles non-smooth ℓ_1-norm regularized models. It exploits the fact that origin is the only non-smooth point in the domain. Non-differentiability at the origin is handled by estimating the signs of the variables (by choosing an appropriate orthant) and restricting them to remain unchanged through the current update (make an L-BFGS update and project it back to that orthant). In other non-smooth cases one needs to switch to subgradient methods. However, subgradient methods can be very slow to converge with a convergence rate of $O(1/\sqrt{t})$ (Shor, 1985).

Active set and path-following methods. Alternatively, dedicated algorithms have been proposed for specific penalties on the lines of active set algorithms used in linear programming literature (see Nocedal and Wright, 2006). These methods use the prior knowledge of the solution being sparse to design computationally efficient algorithms. They typically keep track of a set of features, called the working set or the active set, that is likely to be in the support of the solution. This working set is modified appropriately until the solution is obtained.

These algorithms are tailored for specific loss function $\mathcal{L}$ and regularization Ω of the optimization in (2.12) and solve it for a specific value of the regularization parameter λ. A solution found for a specific value of λ can then be used to find the solutions for other λ values close to it. This is done by exploiting the property that for certain penalties the entries in the solution are piecewise linear when plotted against the regularization parameter. These are called homotopy methods or path-following algorithms (Markowitz, 1952; Osborne et al., 1999) that include the popular least angle regression algorithm by Efron et al. (2004). These approaches are known to work well in practice but are restricted to specific penalties.

Coordinate descent. Coordinate descent is another method that is known to perform well for some penalties (Fu, 1998; Friedman et al., 2007). This method minimizes along one coordinate at a time while iterating through the list of free variables of the optimization. In the case of ℓ_1-norm, for example, it results in iteratively soft-thresholding across each coordinate. It leads to similar rules for other penalties where applicable. But for coordinate descent to be applicable, it is necessary that the optimality conditions are separable across different coordinates (Tseng, 2001; Bertsekas, 2004). Fused lasso is one example where such separability is not possible leading to non-optimal points in the domain at which the

Algorithm 2 Generic proximal minimization algorithm (see Parikh and Boyd, 2013)
$w^* := \underset{w}{\text{argmin}} \ \{\mathcal{L}(w, \mathbf{S}_n) + \lambda\Omega(w)\}$

Input: λ regularization parameter , L Lipschitz constant of ∇f , μ coefficient of strong-convexity of $f + \lambda\Omega$, X design matrix, Y label set
repeat until termination
    EstimateGradientPoint()
    GeneralizedGradient()
    UpdateConstants()

Procedure: $w :=$ GeneralizedGradient($\mathbf{S}_n^\vartheta$, $\overline{w}, \lambda, \rho$) # for some $\mathbf{S}_n^\vartheta \subseteq \mathbf{S}_n$
a. $w' = \overline{w} - \frac{1}{\rho}\nabla\mathcal{L}(\mathbf{S}_n^\vartheta, \overline{w})$ #gradient step
b. $w = \mathbf{Prox}_{\lambda\Omega}(w)$ #proximal step (see Table 2.1)

coordinate descent iterations are stuck (Friedman et al., 2007).

Reparameterizing the norm. Another method that applies generically is the *weighted squared norm regularization* formulation (Rakotomamonjy et al., 2008). Any norm can be rewritten as a maximum of quadratic functions which can be minimized using iterative least squares alternating between w and the parameter (Bach et al., 2012). This method has a caveat that might generate a term with zero in the denominator, cases like which need to be handled carefully.

Proximal methods. These are methods that are designed to optimize composite functions that involve sum of a smooth convex term (in our case $\mathcal{L}$) with a convex but possibly non-smooth term ($\lambda\Omega$). There are a few algorithms that tackle such composite functions; Nesterov's method (Nesterov, 2007) with both batch and stochastic optimization algorithms, Forward Looking Subgradient by Duchi and Singer (2009) which is a kind of forward-backward splitting for both batch and stochastic and Regularized Dual Averaging for stochastic by Xiao (2010). We consider Nesterov's method that delivers the best convergence rate among these algorithms.

Nesterov's proximal algorithm is an iterative process that makes a generalized gradient update at each step with gradient update depending only on $\mathcal{L}$ followed by the *proximal step* depending on $\lambda\Omega$. The proximal step is a projection involving a small optimization that can be solved efficiently for numerous penalties (analytically in few cases). We will consider this method in greater detail in the next section.

2.4 Proximal minimization algorithms

We choose proximal methods as they exhibit an optimal convergence rate among first-order techniques combined with the ability to reuse specialized algorithms for different penalties by simply changing the proximal step (see Nesterov (2007); Beck and Teboulle (2009); Parikh and Boyd (2013) for proximal minimization and Bach et al. (2012) for optimization with sparsity inducing penalties).

Proximal methods iteratively update the current estimate by making a generalized gradient update at each iteration. Formally, they are based on a linearization of the

Algorithm 3 Batch proximal minimization following Beck and Teboulle (2009)
$w^* := \underset{w}{\text{argmin}} \{\mathcal{L}(w, \mathbf{S}_n) + \lambda\Omega(w)\}$

1 Initialize: $\overline{w} = Z = 0$, $\tau = \delta = 1$
2 **repeat until termination**
3 $\quad \overline{w} := \frac{\delta+\tau-1}{\delta} w - \frac{\tau-1}{\delta} Z; Z := w$
4a $\quad$ **repeat:** $w = \text{GeneralizedGradient}(\mathbf{S}_n, \overline{w}, \lambda, 1/L)$ #full batch update
4b $\quad\quad$ **if** $\mathcal{L}(w) < \mathcal{L}(\overline{w}) + \nabla\mathcal{L}(w - \overline{w}) + 0.5L||w - \overline{w}||_2^2$ **break else** L := ηL
5 $\quad \tau = \delta; \delta = 1 + \frac{1+\sqrt{1+4\tau^2}}{2}$

Algorithm 4 Stochastic proximal minimization following Hu et al. (2009)
$w^* := \underset{w}{\text{argmin}} \{\mathcal{L}(w, \mathbf{S}_n) + \lambda\Omega(w)\}$

1 Initialize: $\overline{w} = Z = 0$, $\tau = \delta = 1$, $\rho = L + \mu$
2 **repeat until maximum iterations**
3 $\quad \overline{w} = (1 - \tau)w + \tau Z$ #estimate point for gradient update
4 $\quad w = \text{GeneralizedGradient}(\mathbf{S}_n^\vartheta, \overline{w}, \lambda, \rho)$ #use mini-batch $\mathbf{S}_n^\vartheta \subset \mathbf{S}_n$ for update
5a $\quad Z = \frac{1}{\rho\tau+\mu}((1 - \mu)Z + (\mu - \rho)\overline{w} + \rho w)$ #weighted combination of estimates
5b $\quad \rho = L + \mu/\delta$, $\tau = \frac{\sqrt{4\delta+\delta^2}-\delta}{2}$, $\delta = (1 - \tau)\delta$ #update constants

objective function f about the parameter estimate $\overline{w}$, adding a quadratic penalty term to keep the updated estimate in the neighborhood of the current estimate. At iteration t, the update of the parameter w is given by the following equation called the generalized gradient update,

$$w^{t+1} = \underset{w \in \mathcal{K}}{\text{argmin}} \left\{ f(\overline{w}) + (w - \overline{w})^\top \nabla f(\overline{w}) + \lambda\Omega(w) + \frac{L}{2}\|w - \overline{w}\|_2^2 \right\}, \tag{2.13}$$

where $L > 0$ is an upper bound on the Lipschitz constant of the gradient ∇f and $\overline{w}$ is the point at which the gradient is estimated. Equation (2.13) can be rewritten to be solved in two independent steps: a gradient update from the smooth part followed by a projection depending only on the non-smooth penalty:

$$w' = \overline{w} - \frac{1}{L}\nabla f(\overline{w}), \tag{2.14}$$

$$w^{t+1} = \underset{W \in \mathcal{K}}{\text{argmin}} \frac{1}{2}\|w - w'\|_2^2 + \frac{\lambda\Omega(W)}{L}. \tag{2.15}$$

Update (2.15) is called the proximal operator $\mathbf{Prox}_{\frac{\lambda\Omega}{L}}(w')$. Efficiently computing the proximal step is crucial to maintain the fast convergence rate of these methods.

Proximal minimization procedure. The basic steps of a proximal minimization procedure are described in Algorithm 2. These methods iterate over three operations in a loop until termination. The termination usually involves a threshold on the amount of relative progress made in consecutive iterations $\left(\frac{|\mathcal{L}(w^{t+1}) - \mathcal{L}(w^t)|}{\mathcal{L}(w^t)}\right)$ unless the duality gap is easily computable. The first operation involves finding the point at which the gradient needs to

be estimated (Step 3). The second operation in Step 4 solves the update equation in (2.13) for a new estimate of the parameters. This is carried out in two steps with a gradient update for f (equation (2.14) in Step a) followed by the proximal projection for Ω (equation (2.15) in Step b). The third operation updates any necessary constants necessary for the next iteration.

Accelerated batch minimization. The simplest proximal minimization scheme sets $\overline{w} = w^t$, the current estimate of the parameters. This is the iterative soft-thresholding algorithm (ISTA) that is known by other names in signal processing literature (see, e.g., Chambolle et al., 1998; Daubechies et al., 2004) and is known to converge at a rate of $O(1/t)$. However, the rate of convergence can be improved by using a more appropriate choice of $\overline{w}$. These methods are popularly known as accelerated gradient methods originally proposed by Nesterov (1983)(see also Tseng, 2008). One such algorithm is called the Fast Iterative Shrinkage Thresholding algorithm by Beck and Teboulle (2009). It involves computing the gradient at a point between the current parameter estimate and its estimate at the previous iteration. A fixed step-size could be used for the gradient update but it is common to choose a more appropriate step by backtracking. In backtracking, the size of the step is $L\eta^K$ for some $\eta > 1$ with k being the smallest integer satisfying:

$$f(w) < f(\overline{w}) + \nabla f(w - \overline{w}) + 0.5L\eta^k ||w - \overline{w}||_2^2.$$

This algorithm is summarized as the batch method in Algorithm 3 with the three operations within the main loop matching Algorithm 2. Step 3 computes the point ($\overline{w}$) about which gradient is estimated. Step 4 makes the generalized gradient update with a loop for backtracking until an appropriate step is found. Step 5 updates the constants for the next iteration. This technique is shown to improve the rate of convergence to $O(1/t^2)$ that is faster than the $O(1/t)$ rate of the simple algorithm above.

Accelerated stochastic minimization. On the downside, Algorithm 3 requires multiple calls to function evaluation and the full gradient at each iteration. In most applications with large amounts of data it is extremely time consuming to compute the full gradient. A stochastic alternative is to estimate the gradient on sufficiently sized subset called a mini-batch. The size of the mini-batch controls the variance in the gradient estimate that is traded against the computational time. Hence, we use an accelerated stochastic version of the proximal methods by Hu et al. (2009) with mini-batches to process large datasets. At every update, the stochastic algorithm estimates the gradient on a mini-batch. This version of the optimization is summarized as stochastic proximal minimization in Algorithm 4. Steps 3 and 5 are similar to the batch processing algorithm except with a different set of constants. But step 4 only uses a subset $\mathbf{S}_n^\vartheta \subset \mathbf{S}_n$ of the complete set of observations to estimate the gradient. This algorithm converges at a $O(1/t)$ rate for strongly convex objectives but degrades to $O(1/\sqrt{t})$ for general convex objectives.

2.4.1 Proximal operators for penalties

The proximal projection step in equation (2.15) lies at the core of the proximal minimization algorithms. It can be rewritten as follows,

$$\mathbf{Prox}_{\lambda\Omega}(v) \stackrel{\text{def}}{=} \underset{u}{\text{argmin}} \ \frac{1}{2}||u - v||_2 + \lambda\Omega(u). \tag{2.16}$$

Proximal step can be interpreted in numerous ways and related to existing methods (see Parikh and Boyd, 2013, for details). One particular interpretation is to view it as the conjugate of the ℓ_2^2 smoothed conjugate of Ω. This operation is referred to as the Yoshida-Moreau regularization or infimum convolution. This helps us understand how it handles non-smooth functions in optimization. We present proximal projections for unstructured and structured norms and look at specific improvements later.

It is common to iteratively maximize the Fenchel dual of the minimization in (2.15) to solve for u (Jenatton et al., 2011b). However, the minimization can be solved very quickly in closed form for specific choices of Ω. In the case of ℓ_2^2, one can set the derivative of proximal objective in (2.16) to zero to recover $\mathbf{Prox}_{\lambda\ell_2^2}(v) \equiv v/(1+\lambda)$. But it gets tricky with non-differentiable Ω like ℓ_1. $\mathbf{Prox}_{\lambda\ell_1}$ can be rewritten in terms of projection on the dual norm ball using the following property of proximal operators called Moreau decomposition:

$$v = \mathbf{Prox}_{\lambda\Omega}(v) + \mathbf{Prox}_{g}(v), \tag{2.17}$$

where $g(y) = \sup_x\{y^\top x - \lambda\Omega(x)\}$ is the Frenchel-conjugate of $\lambda\Omega$. When $\Omega(v) = ||v||_p$ is the ℓ_p-norm, then the Frenchel-conjugate is the indicator of the dual norm ball $\mathcal{B}_q$ of radius λ. The dual norm ball for the ℓ_p-norm is $\mathcal{B}_q = \{x, ||x||_q \leq \lambda\}$ where $1/p + 1/q = 1$. This makes the corresponding proximal step for a projection onto $\mathcal{B}_q$, and subsequently $\mathbf{Prox}_{\ell_p}$ would be the residual $v - \Pi_{\mathcal{B}_q}(v)$.

Essentially, if we can project onto the dual norm ball efficiently, we make a quick proximal update. The proximal for ℓ_1-norm reduces to residual of projection on ℓ_∞-norm ball; $\mathbf{Prox}_{\lambda\ell_1}(v)$ is element-wise soft-thresholding v at λ.

In the case of group Lasso, the proximal step is separable over the groups leading to individual projection of variables in each of the groups. It can be solved similarly for the elastic net as well. It gets a bit trickier with fused lasso and involves making the projection in a series of two steps following the work of Friedman et al. (2007) (see also Mairal, 2010; Liu et al., 2010).

1. Proximal step is made assuming only the fused part of the regularization exists and that of the elastic net is zero by setting $\Omega(w) = \gamma_2 \sum_{i=1}^{|w|-1} |w_{i+1} - w_i|$. This is performed by a modification of the LARS algorithm.

2. The above projection is then followed by a projection for the elastic net alone corresponding to the other two terms of the penalty.

These are reported in Table 2.1. However, it is much more involved for other structured penalties. We consider the particular case of tree structured penalty in detail.

The $\mathbf{Prox}_{\lambda\Omega}(v)$ for tree-structured norms with Ω defined in (2.9) is complicated due to overlap of groups. This overlap can be untangled by taking the dual and then coordinate ascent could be used to maximize the dual. However, for specific choices of p the maximization of the dual can be performed in closed form. Jenatton et al. (2011b) showed that $\mathbf{Prox}_{\ell_p^T}$ reduces to applying the $\mathbf{Prox}_{\ell_p}$ to the subvectors in a specific order defined by $\mathcal{G}$ below.

Algorithm 5 Proximal update for tree-structured norms following (Jenatton et al., 2011b).
$u^* = \mathbf{Prox}_{\lambda\ell_p^T}(v) \stackrel{\text{def}}{=} \underset{u}{\text{argmin}}\ \frac{1}{2}||u-v||_2 + \lambda||u||_{\ell_p^T}$

Input: Vector u, parameter λ, tree structure T, ordered groups $\mathcal{G}$.
1. **for** $i \in \{1, \cdots, |\mathcal{G}|\}$
2. $\quad w_{g_i} \leftarrow w_{g_i} - \Pi_{||\cdot||_q \leq \lambda}(w_{g_i})$
3. **Return** w

a. ℓ_q is the dual norm of ℓ_p ($\frac{1}{p} + \frac{1}{q} = 1$) and $\Pi_{||\cdot||_q \leq \lambda}$ is projection on the dual norm ball of λ units size.

Definition 6. *Let $\mathcal{G} = \{g_i\}_{i=1}^{|\mathcal{G}|}$ denote the ordered tree structured grouping of indices $I = \{1, \cdots, |w|\}$ of a vector w with tree structure T such that*

- $\bigcup_{g \in \mathcal{G}} g = I$ (grouping as in Definition 3)
- for $r, s \in I,\ r < s$, $\mathcal{G}_r \cap \mathcal{G}_s = \emptyset$ or $\mathcal{G}_r \subset \mathcal{G}_s$. (defines ordering on groups)

The set $\mathcal{G}$ defines an order on the grouping described in G^T (from Definition 3) such that before a group is encountered, all its subsets that are included in G^T are visited. This is equivalent to visiting the nodes of T is a pre-order depth-first search. Given the ordering $\mathcal{G}$, $\mathbf{Prox}_{\ell_p^T}$ takes the following form:

$$\mathbf{Prox}_{\lambda\Omega}(v) = \mathbf{Prox}_{\lambda\Omega}^{g_{|\mathcal{G}|}} \circ \cdots \circ \mathbf{Prox}_{\lambda\Omega}^{g_2} \circ \mathbf{Prox}_{\lambda\Omega}^{g_1}(v) \tag{2.18}$$

where $g_i \in \mathcal{G}$ and $\mathbf{Prox}_{\lambda\Omega}^{g_i}(v) \stackrel{\text{def}}{=} \mathbf{Prox}_{\lambda\Omega}(g_i^v)$. This is to say that the proximal update for the tree-structured norm is equivalent to applying an ordered sequence of projections for its groups with ordering defined by $\mathcal{G}$. The order $\mathcal{G}$ of projections is such that for every $g_r \subsetneq g_s$, $(g_r, g_s) \in \mathcal{G}$, g_r is projected before projecting g_s. In other words, a subtree is projected only after all its subsets in the tree are projected. This order over the groups is generated by a bottom-up sweep of the tree.

Each $\mathbf{Prox}_{\lambda\Omega}^{g_i}$ in the sequence of proximal operators can itself be rewritten as residuals of projections on dual norm ball (projection-residuals) using the decomposition in equation (2.17). This makes $\mathbf{Prox}_{\ell_p^T}$ a sequence of projections on the ℓ_q-norm ball. The resulting procedure is described in Algorithm 5.

The second step of the procedure described in Algorithm 5 performs projection on the dual norm ball. This needs to be done in an iterative fashion and is not guaranteed to converge in finite number of steps. However, for specific norms, namely $p = 2$ and ∞, it can be performed in closed form. ℓ_2-norm being self dual $\mathbf{Prox}_{\lambda\ell_2}(v) = v - \Pi_{\ell_2 \leq \lambda}(v)$ which is a block soft thresholding operator. Similarly, $\mathbf{Prox}_{\lambda\ell_\infty}(v) = v - \Pi_{\ell_1 \leq \lambda}(v)$ requires an element-wise soft thresholding operator.

The projection on the ℓ_2-ball results in scaling operation and has complexity linear in the length of the vector. The resulting $\mathbf{Prox}_{\lambda\ell_2^T}$ when applied naively as described above has quadratic complexity. A clever trick due to Jenatton et al. (2011b) is to compute all necessary scale factors and project the complete vector in one pass rather than projecting each of its subvectors individually. This reduces the complexity of $\mathbf{Prox}_{\lambda\ell_2^T}$ to linear in the

penalty	$\mathbf{Prox}_{\kappa\Omega}(w)$	Operation
ℓ_1	$\mathtt{sgn}(w)\,[\lvert w\rvert - \kappa]_+$	element-wise soft thresholding
ℓ_2	$[\lvert\lvert w\rvert\rvert_2 - \kappa]_+ \frac{w}{\lvert\lvert w\rvert\rvert_2}$	block soft thresholding
ℓ_2^2	$w/(1+\kappa)$	element-wise scaling
ℓ_p	$\Pi_{\lvert\lvert\cdot\rvert\rvert_q \leq \kappa}(w)$	project on κ-unit ball of dual norm (ℓ_q)
$\ell_{En(\kappa,\kappa\gamma_1)}$	$[w-\gamma]_+/(1+\kappa)$	threshold and scale
$\ell_{Fl(\kappa,\kappa\gamma_1,\kappa\gamma_2)}$	$w := \mathbf{Prox}_{\ell_{Fl(0,0,\kappa\gamma_2)}}(w)$ $[w-\kappa\gamma_1]_+/(1+\kappa)$	LARS type method for $\mathbf{Prox}_{\ell_{Fl(0,0,\kappa\gamma_2)}}$ followed by thresholding and scaling
$\ell_p^{G_P}$	$w_g := \mathbf{Prox}_{\ell_p}(w_g)$	group-wise proximal step
$\ell_p^{G_T}$	$\mathbf{Prox}_{\ell_p}^{\mathcal{G}_{\lvert\mathcal{G}\rvert}} \circ \cdots \circ \mathbf{Prox}_{\ell_p}^{\mathcal{G}_1}(w)$	ordered sequence of proximal steps

Table 2.1: Definitions of the penalties and their corresponding proximal operators.

number of dimensions. An adaption of this algorithm to our case is detailed in the next chapter.

Projection on ℓ_1-ball results in soft-thresholding operation and estimating the threshold value has an amortized linear time complexity in number of dimensions due to Bruckner (1984); Maculan and Galdino de Paula (1989); Pardalos and Kovoor (1990) leading to quadratic complexity similar to the case of $\mathbf{Prox}_{\lambda\ell_2^T}$. But the above trick used in $\mathbf{Prox}_{\lambda\ell_2^T}$ does not apply here since a subvector should have been projected for the next threshold to be estimated[2]. So, the sequential structure in projecting the subvectors must necessarily be followed. However, under certain assumption on the distribution of values in the vector, the time complexity can be reduced to deterministic linear in the number of features. We discuss this algorithm with improved complexity for $\mathbf{Prox}_{\lambda\ell_\infty^T}$ in the next chapter.

2.5 Conclusion

In this chapter we reviewed the necessary material for the risk minimization framework. We discussed different penalties available from basic unstructured functions to those that depend on specific grouping of its features. We motivate the choice for incorporating structure specific to the problem's domain into the penalty. Various algorithms available for their minimization were surveyed and the suitable choice of proximal gradient minimization was studied in detail. In the next chapter we will consider the log-linear language model that we propose with suitable penalty functions and their optimization.

[2]When the trees are balanced and short, the amortized complexity is much lower than quadratic at about $O(pD)$ where D is the depth of the tree (Jenatton et al., 2011b).

Chapter 3

Log-linear language model

We now consider the specific model that we propose for language. We describe the problem discussed in Chapter 1 as a generalized linear model and explain the contextual features used in the model. Then, the suffix-tree representation is described which is necessary for tractable computation of the model. We distinguish between the two representations; a trie and the corresponding tree that is made by collapsing non-branching paths in the trie. However, we will see that the tree representation is not suitable for our model and we introduce a partially collapsed trie. We then discuss penalties that are structurally suitable for language sequences and map their proximal projections from Chapter 2 to operations on suffix-trees. We show how specific proximal operators with some nice properties can be computed efficiently on the suffix-tree. We conclude the chapter with an efficient algorithm to optimize the penalized risk with large contexts.

3.1 Introduction to Generalized Linear Models

The hypothesis set we considered earlier in Section 2.1 is restricted to a linear mapping of the form $y = h^{w_y}(x) = w_y^\top x$. However, it is easy to see that numerous applications in real world follow more complex relations that cannot be explained by linear mappings. This has thus been extended to more general class of mappings where some function of the response or dependent variable varies linearly with the predictor, or $\zeta(y) = w_y^\top x$. The link function ζ can be chosen arbitrarily with its domain matching the range of the expectation of y. In our case, the log-odds (log of the ratio of probabilities of two classes) bears a linear relationship with the input variables. This relation is captured by a multinomial logit link function (Hastie et al., 2009).

This generalization offers multiple benefits over the simple linear model. For example, it allows for modeling count data, ordered data, categorical data (multi-way choices), and ordinal data (ratings). It does this by allowing for modeling response variables using skewed distributions (like Poisson, Gamma and others) beyond the standard Normal distribution of linear regression. These models have been dealt in detail by McCullagh and Nelder (1989). We consider the case where ζ is the multinomial logit giving $\ln y = w_y^\top x - \ln Z(x, [w_v]_{\forall v})$ where $[w_v]_{\forall v}$ is the matrix of all parameters and hence the name log-linear model. Log-linear models leave us with an exponential prediction of the form $y \propto e^{w_y^\top x}$. We study a log-linear formulation for modeling language in the next section.

3.2 Log-linear language model

Let V be the set of words or more generally symbols, which we call vocabulary. We are given samples in the form of a text corpus $\mathbf{S}_n \sim \mathcal{C}$ with words from V and are required to assign probabilities to sequences of words of length $m+1$. Similar to the maximum entropy methods considered in Chapter 1, we discriminatively model the probability of predicting a word y from its context x_m, $\mathbf{P}(y|x_m)$. The joint distribution of a sequence s comes from predicting each of its words from the corresponding context.

This is the case of $|V|$-way choice problem where we must choose one word from a list of $|V|$ choices. Such situations are explained using the "categorical" distribution[1] over the vector of probability mass for different words. Probability for each word itself comes from the log-linear model described above as $\pi_v(x) \propto e^{w_v^\top \phi_m(x)}$ leading to the following:

$$y|x \sim \text{Categorical}\left(\pi(x)\right) = \prod_{v \in V} \pi_v(x)^{\mathbb{I}[y=v]}, \qquad \pi_v(x) = \frac{e^{w_v^\top \phi_m(x)}}{\sum_{u \in V} e^{w_u^\top \phi_m(x)}}, \tag{3.1}$$

where $W = [w_v]_{v \in V}$ is the set of parameters and $\phi_m(x)$ is the vector of features extracted from x, which we will describe shortly. Parameters are estimated by minimizing the penalized log-loss over the training corpus $\mathcal{C} = \{(x_m y)_i\}_{i=1}^n$:

$$W^* \in \operatorname*{argmin}_{W \in \mathcal{K}} \mathcal{L}(W, \mathcal{C}) + \lambda \Omega(W), \quad \mathcal{L}(W, \mathcal{C}) := -\frac{1}{n} \sum_{(x_m y) \in \mathcal{C}} \ln \mathbf{P}(y|x_m; W), \tag{3.2}$$

where W^* is the optimal set of parameters and $\mathcal{K}$ is a convex set representing the constraints applied on the parameters (it will be the positive orthant). The convex loss function $\mathcal{L}$ is called the log-loss and the penalty Ω can be non-smooth.

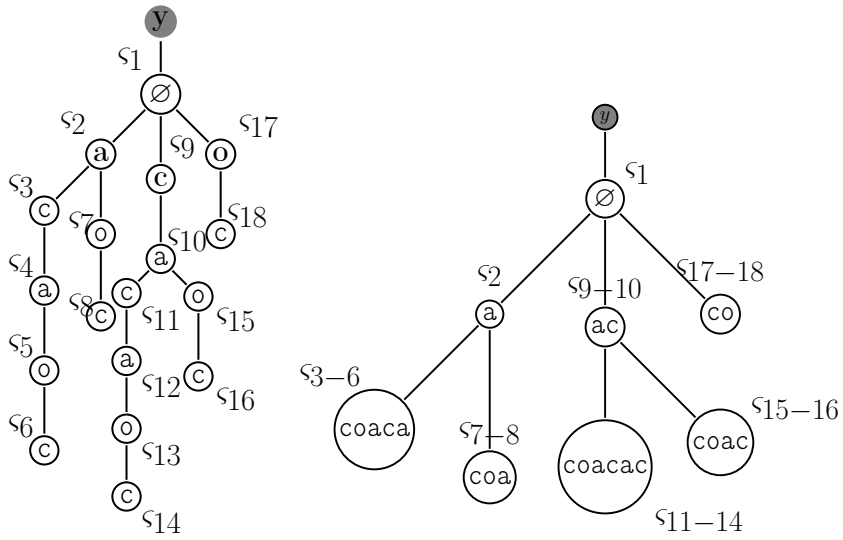
Next, we describe how to map a sequence x to its feature vector $\phi_m(x)$. Our features are scaled indicator vectors that describe the sequence of words. These are the simplest features with no information of syntax or semantics of the language. They merely encode the sequential structure of sentences in vectors. Each non-zero value indicates the occurrence of a particular suffix with a magnitude that could vary with the length of the suffix.

We recall that x_k denotes the suffix of length k of the sequence x, that is, x_k is the sequence of the last k symbols in x. If x is followed by y, then the set of suffixes $\{x_k\}_{k=1}^n$ defines the contexts in which y is observed. Further, let ξ^k denote the lexicographically ordered set of all length k sequences formed with the elements of V. For example, the set ξ^2 for $V = \{\texttt{a}, \texttt{c}, \texttt{o}\}$ is the ordered set $\{\texttt{aa}, \texttt{ac}, \texttt{ao}, \texttt{ca}, \texttt{cc}, \texttt{co}, \texttt{oa}, \texttt{oc}, \texttt{oo}\}$. We define a mapping from sequences x to binary vectors $\psi_k(x)$ indicating the location of the suffix x_k in the set ξ^k:

$$\psi_k(x) = \left[\mathbb{I}(\xi^k(1) = x_k)\ \mathbb{I}(\xi^k(2) = x_k)\ \ldots\ \mathbb{I}(\xi^k(|V|^k) = x_k)\right]^\top, \tag{3.3}$$

where $\mathbb{I}(\cdot)$ is the indicator function. For example, for $x = \texttt{oacac}$ and $x_2 = \texttt{ac}$, we have $\psi_2(x) = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^\top$. The feature vector $\phi_m(x)$ encodes all suffixes observed in x

[1] *Categorical* distribution (a generalization of *Bernoulli*) is equivalent to the *Multinomial* distribution for single observation in discrete space and is sometimes called the *Discrete* distribution. For our purposes, they can be used interchangeably but are not equivalent in general with subtle differences.



(a) Suffix trie $S(X)$. (b) Suffix tree $T(X)$.

Figure 3.1: Diagrams of suffix trie and tree for X=`coacac`.

up to length m. It is obtained by stacking a weighted version of the corresponding binary vectors $\{\psi_k(x)\}_{k=1}^m$:

$$\phi_m(x) = \left[\beta(0)\psi_0(x)^\top \ \beta(1)\psi_1(x)^\top \ \ldots \ \beta(m)\psi_m(x)^\top\right]^\top, \tag{3.4}$$

where the function $\beta(k)$ weighs suffixes depending on their length k. If we choose $\beta(k) = 1$ for all k, then the suffixes of all lengths are equally weighted. The first entry ψ_0 is a non-zero value indicating the null context. Moreover, $\phi_m(x)$ discards all suffixes longer than m, restricting the maximal length of the context. This is equivalent to the Markov assumption used in m-gram language models. Indeed, $\phi_m(x)$ encodes occurrences of various suffixes, while the parameters W provide a weighting of contexts of different lengths. The main difficulty comes from the fact that the dimension of the feature space is $\sum_{k=0}^m |V|^k$, meaning that the number of entries in W becomes rapidly unmanageable using standard data structures when the context length increases (as $|V|$ is typically in the order of 10^4 or larger). This causes storage and computational issues, at training *and* test time. However, in practice $\phi_m(xy)$ has only m non-zeros elements due to the sequential dependencies between successive symbols. In this context we describe the use of suffix trie and tree structures to handle this data efficiently.

3.3 Suffix encoding in tries and trees

Suffix trees provide an efficient way to store and manipulate discrete sequences as they can be constructed in linear time due to Ukkonen (1995) when the vocabulary is fixed (also see Giegerich and Kurtz, 1997). They are extensively used in natural language processing and bioinformatics. Recent examples include language models based on a variable-length Markovian assumption (Kennington et al., 2012) and the sequence memoizer (Wood et al., 2010).

The data structure encodes all the unique suffixes observed in a sequence up to a maximum given length. It is organized as a tree with nodes lower in the tree holding longer suffixes. We distinguish a suffix trie from a suffix tree here. A *trie* stores the suffixes with one suffix per node. A *tree*, however, collapses paths in the trie that do not branch. A suffix tree could have multiple suffixes per node while the hierarchy still exists. Figure 3.1(a) illustrates the suffix trie on a sequence `coacac` and the corresponding suffix tree is shown in Figure 3.1(b). We specifically refer to trie by the symbol $S(X)$ built on sequence X and the corresponding tree by $T(X)$.

This collapsing of such paths into nodes saves memory required to encode the sequence. Each internal node in a suffix tree has at least two children while those in a suffix trie have one or more children. The number of nodes in the tree scales by a significantly lower order than that of the trie with increasing length of the sequence. The exact relation between the sequence length and the node count varies from one application to another. In language, this has been observed to be linear by Wood et al. (2010).

3.3.1 Suffix trie structured vectors

The feature vector $\phi_m(x)$ encodes suffixes (or contexts) of increasing length and it corresponds to one path of length m starting at the root of $S(X)$. Thus, when building a log-linear language model, it is sufficient to consider weight vectors of size $|S(X)|$. In other words, there is one weight parameter per suffix per symbol and the matrix of parameters W is of size $|S(X)||V|$. As illustrated in Figure 3.3(a), each column of W contains the weight vector w_v associated with symbol $v \in V$ and $|S(X)| = 13$. The dimension of the weight vectors is bounded by the length of the training sequence. Hence, like the sequence memoizer (Wood et al., 2010), the model can be used as an infinite length language model, though in practice it makes sense to restrict dependency to the sentence length (or the document length).

Making use of the suffix trie encoding additionally offers the computational benefit of efficient matrix-vector (or vector-vector multiplication), which lies at the core of optimization in (3.2). Indeed, each inner product $w_v^\top \phi_m(x)$ reduces to the following three steps:

a. finding the longest suffix of x of maximal length $m_{\max}$ in $S(X)$,

b. extracting the corresponding entries of w_v and

c. summing them after proper scaling with $\beta(\cdot)$.

Again, this is a substantial gain as the longest suffix of x is much smaller than $|S(x)|$ and we avoid summing over weights that are irrelevant in a specific context. This technique of using an efficient data structure to fetch the few non-zero values of a very sparse large vector has been used earlier in numerous ways (most commonly as feature hashing (Stolcke, 2002; Weinberger et al., 2009) and sometimes using other structures like weighted automaton by Roark et al. (2004)). We refer to these vectors as the trie structured vectors.

3.3.2 Word-specific suffix trie structured vectors

While suffix tree encoding leads to a substantial storage and computational gain it might still cause problems in practice as the number of nodes is large for most real data sequences.

We can make further storage and computational gains by building one context tree for every symbol in the vocabulary and constraining the parameters to be non-negative.

We denote the resulting suffix tries by $\{S_v(x)\}_{v \in V}$ with one trie per word in V. The number of free parameters in W is now $\sum_v |S_v(x)|$, which is substantially smaller than $|S(x)||V|$. Figure 3.2(b) shows the word-specific suffix trees for $X = \texttt{oacac}$ and $V = \{\texttt{a,c,o}\}$ and Figure 3.3(b) the corresponding matrix of parameters W. All other parameters values are fixed to zeros. Note that the number of free variables varies from one vector to another in W.

3.3.3 Constraining to positive orthant

There are two issues that arise from plugging in the above described trie structured vectors into the optimization in (3.2).

1. The gradients of the log-linear model in equation (3.1) are dense and require an update of all entries of W for every single observation making the gradient update very expensive.

2. The trie structured vectors set values of non-occuring suffixes to zeros (see Figure 3.3(b)). However, a gradient with the multinomial loss for optimization in (3.2) requires us to update these variables ruling out possibility of trie structured vectors $\{S_v(X)\}_{v \in V}$.

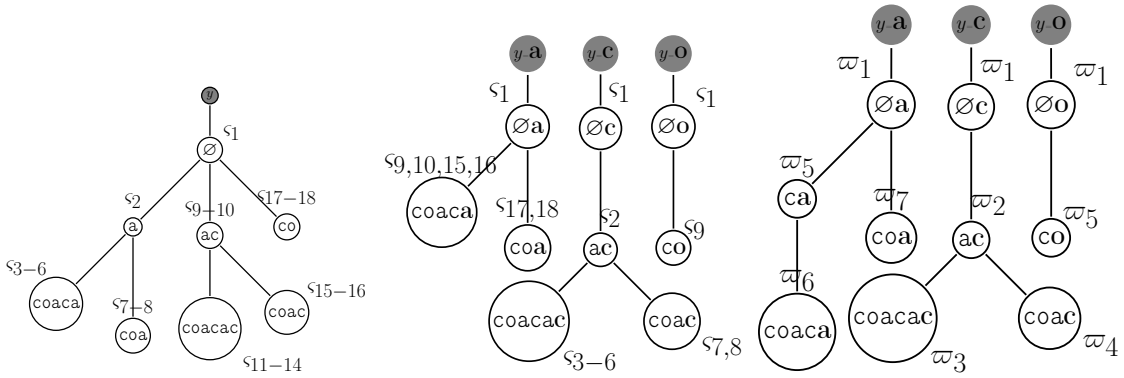
Both these issues are due to the fact that for each positive sample (x, y), all $(x, y'), y' \neq y$ are considered negative examples. This leads to numerous negative entries for parameters in trees of labeled by y'. A simple way to fix this is to exploit the redundancy in the parameterization of the model to add additional constraint that removes the negative entries in the gradient.

The log-linear model described in equation (3.1) with $|V|$ columns in W (one per word) is over-parameterized. Since it only requires parameters from $|V|-1$ columns to be estimated with the left out residual probability mass assigned to remaining $|V|^{th}$ class. A non-negativity constraint is imposed to handle this over-parameterization which is similar to applying a large positive translation to all parameters. The ratio of number of positive entries to the total number of parameters being very small per each sample, this makes the gradient very sparse. This also makes it possible to replace the trie structured vectors (in Figure 3.3(a)) by the word-specific trie structured vectors (shown in Figure 3.3(b)), since the zero entries will remain untouched due to projection into positive orthant.

Fortunately, this constraint can be handled very efficiently by projecting the gradient itself into the non-negative orthant before updating the parameter vector (Appendix B.6 of Jenatton et al., 2011b; Bertsekas, 2004). The constraint effectively reverts parameters corresponding to unobserved suffixes to zeros, which otherwise have small negative values. We solve the optimization problem using the suffix tree representation with this additional constraint of non-negativity of parameters.

3.3.4 Word-specific suffix tree-structured vectors

Model complexity can be reduced further by ensuring that all parameters at a node share a single value. This would impose tree structure on the vectors instead of the trie structure



(a) Complete suffix tree $T(X)$ corresponding to trie $S(X)$.

(b) Trees $T_y(X)$ for different symbols for tries $S_y(X)$.

(c) Trees $M_y(X)$ with only constant value non-branching paths collapsed.

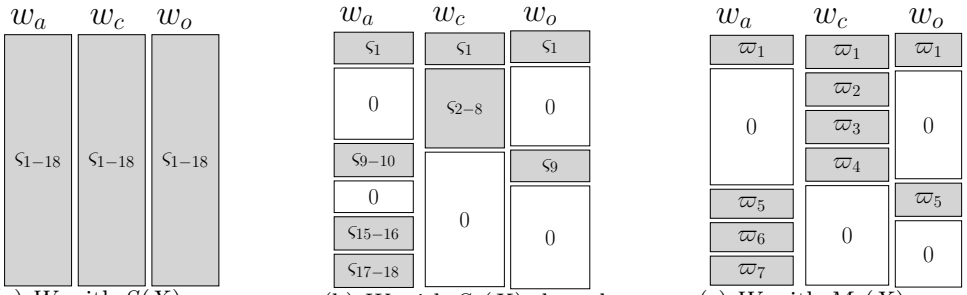
Figure 3.2: Diagrams of suffix trees on X=coacac. Subfigure (a) shows the complete suffix tree with 18 different contexts ς_i. Suffix trees in subfigure (b) are built for each symbol separately with contexts preceding the specific symbols. ς_9 is example of a context that appears in multiple suffix trees and requires to be identified to make $M_y(X)$. Subfigure (c) only collapses constant value non-branching paths of suffix trees in (b). The node corresponding to ς_9 is split to make $M_a(X)$. All contexts in a node are represented by a single parameter.

leading to significant saving in memory. It is not straightforward to see why these parameters should share a single value. But with some modification to the standard suffix tree one can observe that this property will always hold. Then, we can collapse non-branching paths sharing a single value to reduce the number of parameters. We formally define this property as follows.

Definition 7. ***Constant value non-branching paths,*** $\mathcal{P}(S, w)$***,*** *are distinct subsets of nodes $P \subset \mathcal{V}$ of a trie structure $S = \{\mathcal{V}, \mathcal{E}\}$ w.r.t. S-structured vector w such that P has $|P|$ nodes with $|P| - 1$ edges between them and each node has at most one child and all nodes $(i, j) \in P$ have the same value in vector w as $w_i = w_j$.*

Figure 3.11(a) shows an example of a trie with constant value paths before collapsing them and Figure 3.11(b) shows the tree structure for the same example after collapsing them. Rewriting the problem with a single parameter per node requires that along the iterations to solve for the optimization in (3.2), we preserve this property. Assume all values to be initialized with zeros for the optimization. There are two steps in the optimization that update parameters: gradient update and proximal projection. We should collapse nodes from trie $S(X)$ such that the above property is closed under both the gradient and the proximal updates. Considering gradient updates, we wish to have the same gradient values for all parameters at any collapsed node. This is violated by nodes in word-specific trees that have overlapping suffixes with other word-specific suffix trees.

Consider the nodes of coaca and co in Figure 3.2(b). The suffix cy is common to both trees indicated by y=**o**, **a**. After a gradient update for observation co, the parameters at node coaca will no longer share a common value. Since the corresponding gradient will have negative values at $\varsigma_{1,9}$ and zeros for $\varsigma_{10,15,16}$. This can be avoided by splitting nodes that share suffixes with other trees as shown in Figure 3.2(c). Accordingly, the node coaca



(a) W with $S(X)$ shaped columns.

(b) W with $S_y(X)$ shaped columns.

(c) W with $M_y(X)$ shaped columns.

Figure 3.3: Diagrams showing parameter matrices W with different tree structures of Fig. 3.2 for its column vectors. Free variables are shown in grey with the corresponding contexts they represent from Fig. 3.2. Subfigure (a) shows W with $S(X)$ shaped column vectors (from Fig. 3.2(a)). Each column has as many parameters as there are contexts making it 54 parameters. Subfigure (b) shows columns w_y of W with $S_y(X)$ shape (from Fig. 3.2(b)). Each vector only has parameters for relevant contexts making it 17 parameters. No parameters are learned for unobserved suffixes unlike in (a). Subfigure (c) with columns from $M_y(X)$ has 10 parameters with only one parameter per node.

is split into two each with a single parameter. The vectors as shown in Figure 3.3(c) now has 10 parameters against trie structured vectors that uses 17 parameters.

We refer to this as suffix tree-structured vectors with tree structure $M(X)$. However, strictly speaking this is not a proper suffix tree owing to the aspect that some intermediate nodes have a single child. We generically use suffix tree to mean either of the two trees in Figures 3.2(b) and 3.2(c) and distinguish between them by specifying if the vectors are trie-structured $S_v(X)$ or tree-structured $M_v(X)$. In the following section, we look at the effect switching from the trie structure to tree structure has on the number of parameters.

3.3.5 Complexity improvements from trees

The complexity of the model measured by the number of parameters would vary from one task to another and depends on the patterns within the sequences of that data. We measure this for English language to compare the number of nodes in trie structures against that of the tree structures. We consider four distinct subsets of the Associated Press News (AP-news) text corpus with train-test sizes of 100K-20K for evaluation. The corpus was preprocessed as described in Bengio et al. (2003) by replacing proper nouns, numbers and rare words (occurring fewer than three times) with special symbols "⟨proper_noun⟩", "#n" and "⟨unknown⟩" respectively. Sentence boundaries are indicated by manually appending each sentence with a start and end tag at the beginning and the end respectively. Punctuation marks are retained which are treated like other normal words. Vocabulary size for each of the training subsets was around 8,500 words.

Figure (3.4) plots the increase in the number of parameters against increasing context lengths for tries $S_y(X)$ and trees $M_y(X)$ for a sequence of hundred thousand words. Models using the uncollapsed trie structured vectors $S_y(X)$ show a linear increase in the number of parameters as we account for longer contexts to predict the following word. This dependence is only logarithmic when we collapse non-branching paths along the tree that share

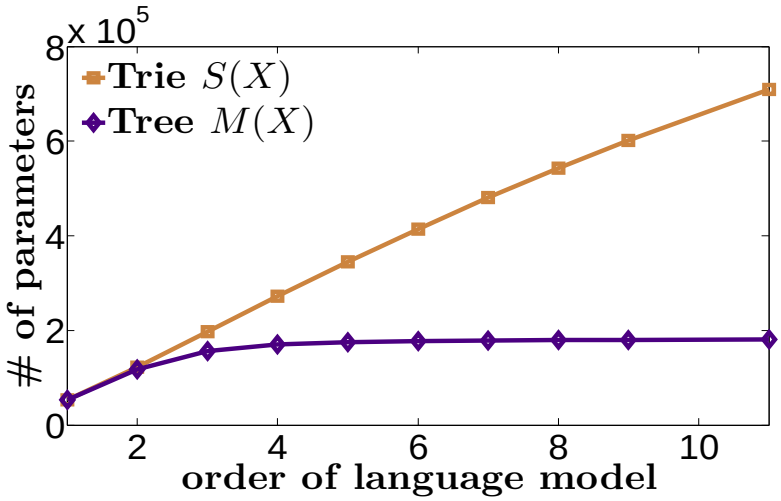


Figure 3.4: Number of parameters for tree and trie structured vectors plotted for increasing orders of the language model. Tree structured vectors have fewer parameters to learn and lead to more compact models.

a value as in $M_y(X)$. Fewer parameters will require fewer updates at each iteration of the optimization over the model and eventually results in a more compact model. This shows how sequences can be encoded efficiently to build models that capture longer dependencies within sequence data.

Next, we consider different models involving structured and unstructured penalties. We could have used different optimization algorithms depending on the choice of the penalty like simple gradient descent with ℓ_2^2, orthant-wise limited memory quasi-Newton for ℓ_1 (Andrew and Gao, 2007) and proximal methods described in Section 2.4 for structured penalties. But we consistently use a proximal algorithm for all penalties including unstructured penalties since it is convenient considering that we only need to change the proximal update to match the penalty. In the following sections, we describe algorithms for proximal updates on suffix trees for different unstructured and structured penalties we consider. These models are then fitted and evaluated on the train and test splits of the dataset described in Section 3.3.5.

3.4 Models with unstructured penalties

In this section and the following one, we fit models with unstructured penalties and evaluate their predictive performance on data. We repeat this evaluation with models of structured penalties in the next section showing how choice of the regularization effects predictive performance.

Evaluation criterion. There are multiple measures to evaluate language models depending on their application. Information theoretic measure of perplexity is one such quantity used in the literature. It is the average of the negative log-likelihood (also referred as the logloss) of the test corpus raised to the power of ten. It upper bounds the number of bits required per word to compress the text using that model. It serves as a generic measure

to compare language models but often, evaluation methods are custom designed for specific tasks. For example, word-error rate (WER (Hunt, 1990)) is used to evaluate speech recognizers, bilingual evaluation understudy (BLEU score (Papineni and Roukos, 2002) or its variants like metric for evaluation of translation with explicit ordering (Banerjee and Lavie, 2005)) is used for translation systems, recall-oriented understudy for gisting evaluation (ROUGE score (Lin, 2004) and it variants) for summarization and so on. These measures are tailored to quantify application-specific properties and are sometimes hard to compute (see Chen et al., 1998a, for WER). There is further work discussing how accurately these measures capture their actual objectives (see, e.g., Wang et al., 2003).

We use the standard perplexity measure that is generic and compares how different methods fare at predicting the test corpus. The perplexity measure is computed as follows:

$$\mathcal{P}(\mathcal{C}_t, W) = 10^{\left\{-\frac{1}{n_V}\sum_{(xy)\in\mathcal{C}_t}\mathbb{I}(y\in V)\log \mathbf{P}(y|x;W)\right\}}, \tag{3.5}$$

where $n_V = \sum_{(xy)\in\mathcal{C}_t}\mathbb{I}(y \in V)$ counts the number of in-vocabulary words and (xy) are sequences from the test set $\mathcal{C}_t$. Tags identifying sentence boundaries are not used in computing perplexity scores. Performance is measured for varying depth of the suffix trie with different norms of regularization suggested in Table 2.1. Perplexity values for unmodified interpolated Kneser-Ney are computed using the openly available SRILM toolkit (Stolcke, 2002) and marked by the **KN** curve in the plots.

We also evaluate space and time complexity of these models. We use the number of free parameters in the resulting model to measure space complexity. A comparison of training time complexities for different models comes out to be a direct comparison of their proximal updates, since the gradient updates are all equivalent. Hence, we use the per iteration cost of making a proximal update to measure the time complexity for each model. In the following section, we describe proximal operations for unstructured penalties on suffix tries and study their behavior on language sequences.

3.4.1 Proximal projection with unstructured penalties

Proximal projections for unstructured penalties are straightforward and independent of the underlying tree structure of the vectors. They give the following update equations for proximal projections:

$$\begin{aligned}\mathbf{Prox}_{\kappa\ell_2^2} &\equiv w_i \leftarrow \frac{w_i}{1+\kappa}\\ \mathbf{Prox}_{\kappa\ell_1} &\equiv w_i \leftarrow \mathtt{sgn}(w_i)[|w_i| - \kappa]_+\end{aligned}$$

Evidently, these operations can be distributed across the entries of the vector and all parameters with the same value will continue to share a single value after the proximal update. ℓ_2^2 and ℓ_1 penalties preserve constant value non-branching paths $\mathcal{P}(S(X), w)$ making it possible to use $M_v(X)$ tree structure. This gives an $O(M(X))$ complexity for the proximal step.

3.4.2 Performance evaluation with unstructured penalties

We evaluate the models by averaging perplexity measure on four distinct subsets 100K words of the AP-news data processed as described in Section 3.3.5. The model is reset at

the start of each sentence, meaning that a word in any given sentence does not depend on any word in the previous sentence. The regularization parameter λ of (3.2) is chosen for each model by cross-validation on a smaller subset of data. Models are fitted to a sequence of 30K words for 20 different logarithmically spaced values of λ from 10^{-8} to 10^{-3} and validated against a sequence of 10K words to choose the best fit. The optimal λ was further fine tuned to find the best performing values of 5.3×10^{-6} and 8.5×10^{-6} for ℓ_2^2 and ℓ_1 penalties respectively. Interestingly, the same regularization parameter was found to work for models with different tree depths for a given penalty.

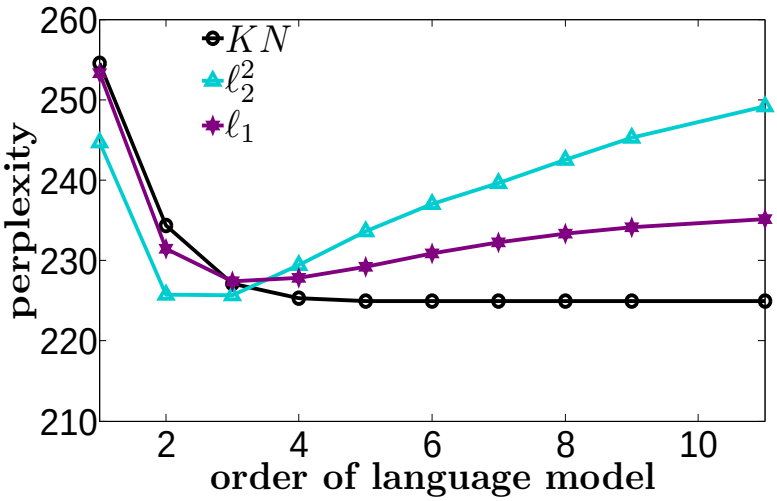


Figure 3.5: Comparison of average perplexity values for unweighted models involving unstructured penalties learned for 2-gram through 12-gram.

Figure (3.5) plots the average perplexity values from four subsets for different penalized models involving unstructured norms. It can be observed that the performance of ℓ_2^2 and ℓ_1 improves until a relatively low order of 3 and then degrades. This is because all features are treated uniformly by these penalties without proper consideration of the space of features. The space of sequences encoded by these features is not uniformly sampled. Features corresponding to longer sequences is sampled much sparsely in comparison to smaller sequences. ℓ_1 and ℓ_2^2 penalties are completely agnostic to this difference in sampling causing them to overfit with features corresponding to longer contexts.

3.5 Models with structured penalties

We next consider models with structured penalties discussed in the earlier chapter. In particular, we choose the tree-structured penalty with hierarchical grouping of features as described in equation (2.9). This penalty naturally fits our data with the suffix trie or tree defining the grouping of features. It perfectly explains the proposition that data observed for features lower in the tree is sparser than that above and hence requires higher amount of regularization.

We restrict our choices of penalty ℓ_p^T to $p = 2, \infty$ owing to the efficient closed form proximal updates as described in Section 2.4.1. In this section, we present algorithms for proximal updates for tree-structured penalties on suffix trees. We then use them to build

Algorithm 6 $w := \mathbf{Prox}_{\kappa\ell_2^T}(w)$ Proximal projection step for ℓ_2^T on grouping $\mathcal{G}$.

Input: T tree structure , w vector to project, κ threshold
Initialize: $\{\gamma_i\} = 0$, $\{\eta_i\} = 1$
η = UpwardPass$(\eta, \gamma, \kappa, w)$
w = DownwardPass(η, w)

Procedure: η := UpwardPass$(\eta, \gamma, \kappa, w)$
for x $\in$ DepthFirstTraversal(T, *PostOrder*)
    $\gamma_x = w_x^2 + \sum_{h \in children(x)} \gamma_h$
    $\eta_x = [1 - \kappa/\sqrt{\gamma_x}]_+$
    $\gamma_x = \eta_x^2 \gamma_x$

Procedure: w := DownwardPass(η, w)
for x $\in$ DepthFirstTraversal(T, *PreOrder*)
    $w_x = \eta_x w_x$
    for $h \in children(x)$
        $\eta_h = \eta_x \eta_h$

a. DepthFirstTraversal(T,*Order*) returns observed suffixes from the suffix tree T by depth-first traversal in the order prescribed by *Order*.
b. w_x is the weights corresponding to the suffix x from the weight vector w and $children(x)$ returns all the immediate children to suffix x in the tree.

models and compare them against those from unstructured penalties.

3.5.1 Proximal projection with ℓ_2^T-norm

The proximal step for ℓ_2^T-norm following the decomposition in (2.18) leads to a sequence of ℓ_2 proximal steps on subvectors. ℓ_2-norm being self-dual, ℓ_2 proximal step comes out to be residual of projection on the ℓ_2-norm ball from the equation in (2.17). Projection of vector w on the ℓ_2-ball of size κ is equivalent to thresholding the magnitude of w by κ units while retaining its direction:

$$w \leftarrow [||w||_2 - \kappa]_+ \frac{w}{||w||_2}. \tag{3.6}$$

This can be performed in time linear in size of w, $O(|w|)$.

In our case the groups for subvectors are defined by the subtrees of the suffix trie $S(X)$. The ordering defined in Definition 6 is an upward pass of the trie. This results in $|S(X)|$ number of $\mathbf{Prox}_{\lambda\ell_2}^{g_i}$ projections. Each $\mathbf{Prox}_{\lambda\ell_2}^{g_i}$ scaling being linear in the number of entries in g_i making the overall $\mathbf{Prox}_{\lambda\ell_2^T}$ quadratic, $O(|S|^2)$.

This can be made efficient by computing scale factors for each subtree g_i as we make an upward pass and later apply them in one single downward pass of $S(X)$. This is possible because the necessary values for computing scale factor for $\mathbf{Prox}_{\lambda\ell_2}^{g_i}$ can be computed from the scaled ℓ_2-norms of child-subtrees if g_i without actually applying the scaling on the children. This technique due to Jenatton et al. (2011b) leads to an algorithm linear in the number of features $|S(X)|$. This algorithm for performing linear $\mathbf{Prox}_{\lambda\ell_2^T}$ in two passes of the suffix trie is presented in Algorithm 6.

Algorithm 7 $w := \mathbf{Prox}_{\kappa\ell_\infty^T}(w)$ Proximal projection step for ℓ_∞^T on grouping $\mathcal{G}$.

Input: T suffix tree, w=[v c] tree-structured vector v with corresponding number of suffixes collapsed at each node in c, κ threshold

for $x \in$ DepthFirstNodeTraversal(T, *PostOrder*)
    $g(v,x) := \pi_{\ell_\infty^T}(\ g(v,x), c_x\kappa\ )$

Procedure: $q := \pi_{\ell_\infty}(q,\kappa)$
Input: $q = [v\ c],\ q_i = [v_i\ c_i],\ i = 1,\cdots,|q|$
Initialize: $U = \{\}, L = \{\}, I = \{1,\cdots,|q|\}$
while $I \neq \varnothing$
  pick random $\rho \in I$ #choose pivot
  $U = \{j | v_j \geq v_\rho\}$ #larger than v_ρ
  $L = \{j | v_j < v_\rho\}$ #smaller than v_ρ
  $\delta S = \sum_{i\in U} v_i \cdot c_i,\ \delta C = \sum_{i\in U} c_i$
  if $(S+\delta S) - (C+\delta C)\rho < \kappa$
    $S := (S+\delta S),\ C := (C+\delta C),\ I := L$
  else $I := U \backslash \{\rho\}$
$r = \frac{S-\kappa}{C},\ v_i := v_i - \max(0, v_i - r)$ #take residuals

a. DepthFirstNodeTraversal(T,*Order*) returns nodes x from the suffix tree T by depth-first traversal in the order prescribed by *Order*.

It is easy to see by example that the $\mathbf{Prox}_{\kappa\ell_2^T}$ operator does not preserve the paths $\mathcal{P}(S(X), w)$ required for efficient tree structure $M_v(X)$. A counter example proving this point is provided in Figure 3.11(c) on an eight-dimensional vector. This makes the method less efficient with increasing contexts eliminating long range dependencies from consideration. We next consider the proximal projection step for ℓ_∞^T-norm.

3.5.2 Proximal projection with ℓ_∞^T-norm

The proximal step for ℓ_∞^T-norm follows a similar decomposition where a sequence of $\mathbf{Prox}_{\lambda\ell_\infty}^{g_i}$ projections are applied in an upward pass of the trie $S(X)$. Again, using the conjugate decomposition in equation (2.17), $\mathbf{Prox}_{\lambda\ell_\infty^T}$ can be performed by taking residuals of projection on to the ℓ_1-norm ball.

Projection of a vector w on the ℓ_1-ball of size κ is more involved and requires thresholding by a value such that the entries in the resulting vector add up to κ, otherwise w remains the same:

$$w \leftarrow \mathtt{sgn}(w)[|w| - \tau]_+ \text{ s.t. } ||w||_1 = \kappa \text{ or } \tau = 0. \tag{3.7}$$

$\tau = 0$ is the case where w lies inside the ℓ_1-ball of size κ with $||w||_1 < \kappa$, leaving w intact. In the other case, the threshold τ is to be computed such that after thresholding, the resulting vector has an ℓ_1-norm of κ. The simplest way to achieve this is to sort in descending order the entries $\overline{w} = \mathtt{sort}(w)$ and pick the k largest values such that the $(k+1)^{th}$ largest entry

is smaller than τ:

$$\sum_{i=1}^{k} \overline{w}_i - \tau = \kappa \text{ and } \tau > \overline{w}_{k+1}. \tag{3.8}$$

We refer to $\overline{w}_k$ as the pivot and are only interested in entries larger than the pivot. Given a sorted vector, it requires looking up to exactly k entries, however, sorting itself take $O(|w| \log |w|)$.

A randomized algorithm to find the pivot that has amortized linear time complexity was proposed by Bruckner (1984) and Pardalos and Kovoor (1990). This method randomly chooses a candidate for the pivot and checks for it correctness using the condition in (3.7). This method is repeated until the right pivot is found. However, a linear time complexity for ℓ_1-ball projection leaves us with an $O(|S|^2)$ quadratic time algorithm for $\mathbf{Prox}_{\lambda\ell_\infty^T}$. The resulting algorithm is summarized in Algorithm 7.

Unfortunately, $\mathbf{Prox}_{\lambda\ell_\infty^T}$ does not simplify into a linear two pass algorithm like $\mathbf{Prox}_{\lambda\ell_2^T}$ because the threshold value depends on the distribution of values unlike the scaling parameter of ℓ_2^T that only depends on the norm of the vector. At each subvector g_i, thresholding needs to be applied and the ℓ_1-norm of the subvector recomputed before moving to its parent node.

We discuss later in Section 3.6.1 two specific aspects of the ℓ_∞^T penalty that lead to significant computational and memory gains over the naive algorithm presented in Algorithm 7. One of them is that the paths $\mathcal{P}(S(X), w)$ required for tree structure $M_v(X)$ are preserved under $\mathbf{Prox}_{\kappa\ell_\infty^T}$ operation. The other involves a single projection at each collapsed node of $M_v(X)$ replacing the series of projections. This comes from an observation that composition of $\mathbf{Prox}_{\kappa\ell_\infty}$ operators over constant value non-branching paths can be replaced by a single $\mathbf{Prox}_{\overline{\kappa}\ell_\infty}$ with κ appropriately modified to $\overline{\kappa}$. We then give a new algorithm for $\mathbf{Prox}_{\kappa\ell_\infty}$ using Fibonacci heaps (Cormen et al., 2009) that has deterministic linear time in the number of entries when some specific condition is true that holds in our case.

3.5.3 Performance evaluation with structured norms

We evaluate models with structured penalties on the same splits of data as described in Section 3.4.2. Structured penalties nicely employ the hierarchy within features that unstructured penalties discussed in Section 3.4.1 fail to capture. This reflects in their proximal steps where features lower in the suffix trie are either scaled or thresholded (depending on the choice of the penalty) more than those higher above in the tree. This property varies the amount of regularization with the depth of the tree. We compare performance of models using structured penalties to that of unstructured models.

Figure 3.6 plots the average perplexity values from the four subsets used in Section 3.4.2 for models involving structured penalties. Regularization parameters of 7.1×10^{-6} and 5.6×10^{-6} were chosen for ℓ_2^T and ℓ_∞^T penalties by cross-validation as in the case of unstructured models. Performance curves for unstructured penalties are also plotted for comparison. It can be observed that unlike ℓ_2^2 and ℓ_1, the ℓ_2^T penalty does not show performance degradation by avoiding overfitting for features of longer contexts. Moreover, the log-linear language model with ℓ_2^T penalty performs quite similar to interpolated Kneser-Ney. Model involving ℓ_∞^T penalty has perplexity values close to Kneser-Ney and performs better at lower depths but starts to overfit for higher ones.

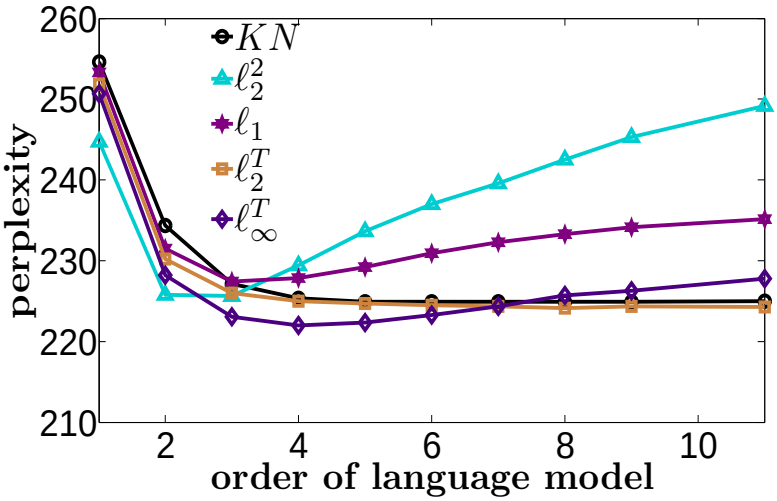


Figure 3.6: Comparison of average perplexity values from models with structured penalties for 2-gram through 12-gram.

3.5.4 Feature weighting to avoid overfitting

Next, to avoid overfitting at larger depths we consider the simpler alternative of weighting features appropriately. By down-weighting a feature we reduce its contribution to performance which is similar to applying regularization on the feature. This method can be used to incorporate structure into unstructured penalties. We apply feature weighting with unstructured penalties ℓ_1, ℓ_2^2 by choosing an exponential decrease of weights varying as $\beta(k) = \alpha^k$.

Parameter α was tuned on a smaller validation set. The best performing values for ℓ_2^2 and ℓ_1 were respectively 0.5 and 0.7. Thus, features deeper in the tree are scaled down. We refer to these weighted models as $w\ell_2^2$ and $w\ell_1$. Performance plots for these models are shown in Figure 3.7.

Further, we applied the scaling with ℓ_2^T, ℓ_∞^T-norm and performance improved for $\alpha = 1.1$ and 0.8. While the model based on $w\ell_1$ performs worse than interpolated Kneser-Ney, we see that the models based on $w\ell_2^2$ and $w\ell_2^T$ peform better. Interestingly, there is observable improvement up to about 9-grams with $w\ell_2^T$ for the data set we considered, indicating that there is more to gain from longer dependencies in natural language sentences than what most standard models extract.

Interestingly, best performing α for $w\ell_2^T$ is greater than one, indicating that the unweighted model based on the ℓ_2^T-norm was overpenalizing features at larger depths. For depths greater than 9, even ℓ_2^T starts to overfit slightly. It is important to note that while feature weighting could be used to verify for improvements in performance by manually fitting different laws of decay for $\beta(k)$, it is not a an appropriate method to handle overfitting for the following reason. On changing the amount of data fed into the algorithm, the amount of regularization must also be varied appropriately. With feature weighting method, this requires retuning of the α appropriately to account for this change in data and hence is not a convenient method to use. However, the same effect of fine tuning the amount of regularization across various depths can be achieved in structured penalties by using group specific weights in the penalty (Jenatton et al., 2011b). This is more conve-

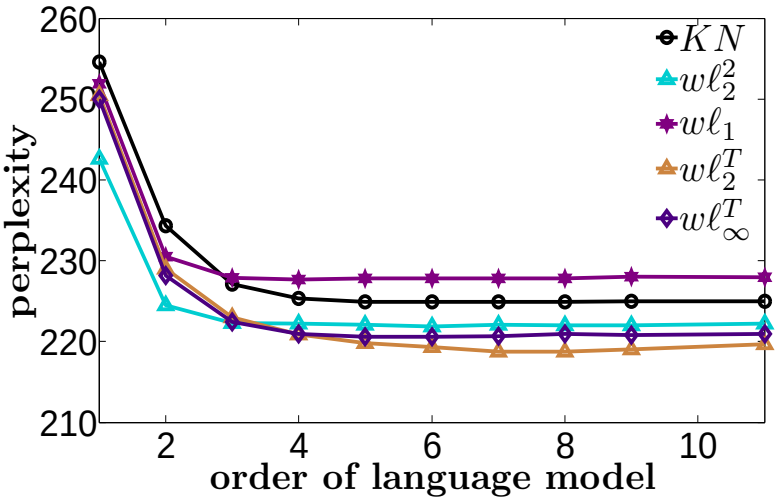


Figure 3.7: Average perplexity values for different penalized models with feature weighting following an exponential decay.

nient with varying amounts of data since the same regularization parameter can be reused by simply scaling it without having to retune it for new data.

The model of exponential decay performed better than a shifted logarithm and linear laws of varying weights from among which it was chosen. However, intuitively speaking, an exponential decay is not the most suitable since, it either starts to discard features completely (for $\alpha < 1$) or gives too much importance further below ($\alpha > 1$) in the tree. Hence, weighing model can be used to verify for possible improvements over the simple structured models but is not a convenient model for practical use. This makes unstructured penalties with feature weighting less useful, further stressing the necessity of structured penalties in handling such problems.

3.5.5 Analysis of perplexity improvements

We now analyze performance of different methods split by various groupings of words. We try to segregate words into groups that would help understand specific cases that are suitable for each of the models.

Figure 3.8 plots how different 7-gram models perform in comparison to Kneser-Ney with the words split by their context length at test time, m_{max}. The value indicated by m_{max} is the length of the longest suffix of a given test sample that was observed in the training corpus (or the suffix tree). All models predict a token (particular occurrence of a word) with the same number of context tokens m_{max}. Hence at each m_{max} value, all models compare average logloss for the same set of words occurring in the same contexts. Points lying above zero are those methods that outperformed Kneser-Ney. The dashed line plotted to the right shows the number of words for a given m_{max} value showing that more words tend to occur in newer or unseen contexts.

Evidently, all penalization methods fare similarly and better than Kneser-Ney when the context length is small. The performance of the models based on $w\ell_1$ and $w\ell_2^2$ degrades significantly with growing context length due to their shortcoming in encoding longer dependencies, while $w\ell_2^T$ maintains performance comparable to that of Kneser-Ney. It can

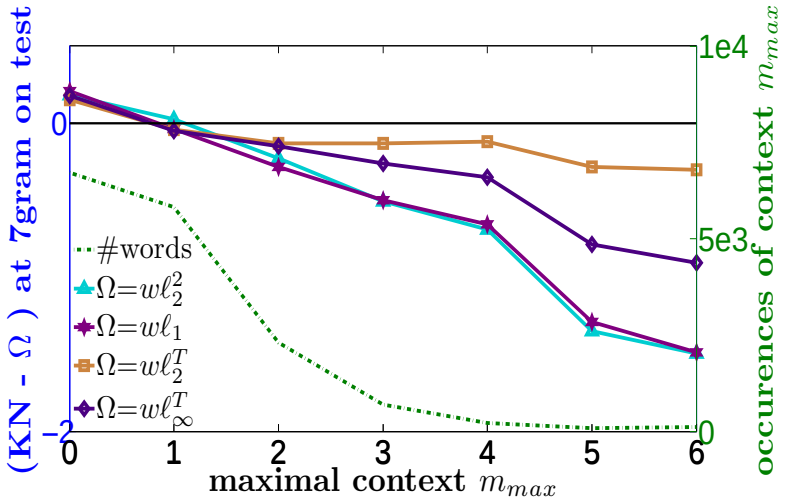


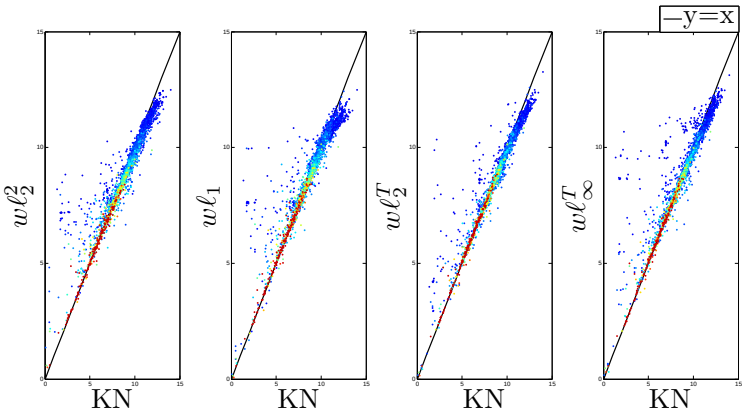
Figure 3.8: Average log-loss is plotted against words grouped by maximal context length used in predicting them at test time. The green dashed line shows the number of words encountered with that m_{max} value plotted on the axis to the right.

be observed that log-linear models are able to learn more accurate weights for the shorter contexts, for which more data is available. The model with $w\ell_2^T$ penalty particularly stems this loss of performance with a gain over Kneser-Ney on average.

This plot explains how structural information helps penalties improve performance. Further, it shows that there is more to gain with a more suitable structural penalty where we perform better than Kneser-Ney for shorter context and at least as well as Kneser-Ney for larger ones. A simple mixed model, for example, where we choose probability from $w\ell_2^T$ for shorter contexts and Kneser-Ney everywhere else will give us a perplexity of 204.1 against 213.6 with only $w\ell_2^T$ model and 217.4 with only Kneser-Ney. This is a significant boost for a simple heuristic from observing the curves in Figure 3.8. However, such a model deviates from our goal to build a principled learning framework for larger tasks. It does not serve the purpose of learning weights through a model that has the various desirable characteristics as described in Section 1.5.

We make a point cloud of the words to visualize how different different methods perform. The average log-loss for each word predicted by the weighted log-linear models of order 6 is plotted against that of Kneser-Ney in Figure 3.9. The color code represents the frequency of the words. There is a large number of low frequency words concentrated on the top-right corner of the plot and fewer high frequencies along the left-bottom of the level set $y = x$. Words lying below the level set are those for which the corresponding log-linear model outperformed Kneser-Ney. The dense blue cloud of words on top-right tilts well below the level set indicating that the log-linear methods gain significantly with less frequent words that occur in large numbers. We can also observe the increasing dispersion of words away from the level set line from $w\ell_2^T$ to $w\ell_2^2$ and $w\ell_1$. These are the fewer high frequency words where performance comparable to Kneser-Ney becomes harder.

Figure 3.10 plots difference in logloss for different models and Kneser-Ney for words in the order of their frequency at train. Each dot represents a word and the mean difference of logloss at a given frequency value is shown in a thick circle. Points to the left in the plots are



(a) Log-loss of different words on the test set.

Figure 3.9: Average log-loss for different words plotted against that of Kneser-Ney (KN). Each dot in the plot is a word. Points are color coded to indicate their frequency with blue being least frequent. Points above the level set $y{=}x$ are better predicted by KN.

concentrated above the zero line. These correspond to words with smaller frequency where learning methods fare better because there is more data for these parameters. Moving rightwards, models with $w\ell_2^T$ and $w\ell_\infty^T$ are packed close to the zero line while that of $w\ell_2^2$ and $w\ell_1$ are scattered with a few points lying well below the zero line. Finally, in Table 3.1 we report a few predictions randomly chosen from the test set that are representative of the different methods. We show predictions for different words with different context lengths m_{max} in the first group. The values reported are logloss values with the best performing model marked in bold-face. We also show logloss of predicting the same word (`police`) under different contexts. We note that the best performing method is either $w\ell_2^T$ or Kneser-Ney.

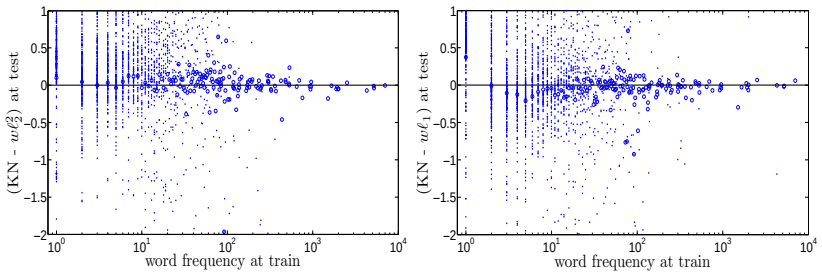
3.6 Time complexity of proximal operators

Proximal updates $\mathbf{Prox}_\Omega$ of unstructured penalties ($\Omega = \ell_1, \ell_2^2$) are linear in the number of nodes in the suffix trees $M_v(X)$ built for each symbol v in the vocabulary V. In the case of structured ℓ_2^T, the time complexity of $\mathbf{Prox}_{\ell_2^T}$ is linear in the number of nodes in $S_v(X)$ for $v \in V$ due to Algorithm 6. However, $\mathbf{Prox}_{\ell_\infty^T}$ has quadratic complexity in the number of nodes in $S_v(X)$ for $v \in V$.

$\mathbf{Prox}_\Omega$	Complexity	
	Naive algorithm	Improved algorithm
$\Omega = \ell_1, \ell_2^2$	$O(\lvert M\rvert)$	
$\Omega = \ell_2^T$	$O(\lvert S\rvert^2)$	$O(\lvert S\rvert)$
$\Omega = \ell_\infty^T$	$O(\lvert S\rvert^2)$	$O(\lvert M\rvert \ln M)$

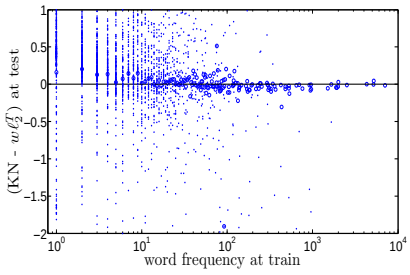
Table 3.2: Complexity of proximal operators for different penalties in terms of the nodes in the corresponding suffix tree (M) and the trie (S).

It can be observed that the complexity of proximal updates for unstructured norms

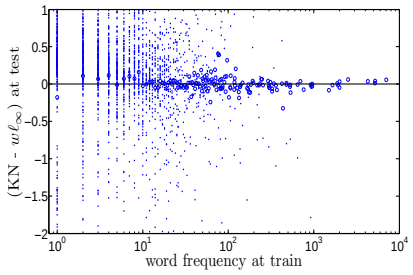


(a) $w\ell_2^2$ penalty.

(b) $w\ell_1$ penalty.



(c) $w\ell_2^T$ penalty.



(d) $w\ell_\infty^T$ penalty.

Figure 3.10: Plots showing logloss difference of tree-based methods and Kneser-Ney for different words grouped by their frequency at train. The larger circles show the mean value. Points lying above the constant zero line correspond to words for which Kneser-Ney was outperformed.

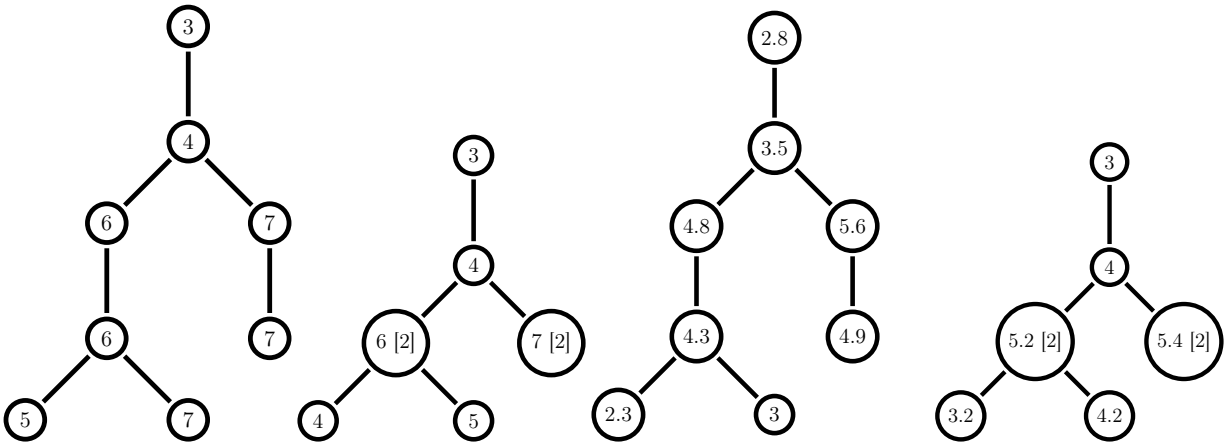
depend on the tree rather than the trie. This is because the corresponding models preserve constant value non-branching paths defined in Definition 7. If this property holds for structured penalties, we could reduce the number of parameters and consequently the number of operations by devising a more appropriate algorithm for proximal projection. It is easy to verify that this property does not hold for $\mathbf{Prox}_{\ell_2^T}$. Figure 3.11(c) shows an example of an eight-dimensional vector that violates this property. However, $\mathbf{Prox}_{\ell_\infty^T}$ on the same example shows how these paths are preserved in Figure 3.11(d). This can be proven for the general case that constant value non-branching paths are indeed closed under $\mathbf{Prox}_{\ell_\infty^T}$ operations.

In the next section we present two properties specific to the ℓ_∞^T-norm. The first among them shows that the suffix tree structure can be used instead of the trie structure reducing the space complexity from $|S|$ to $|M|$. The second property uses this to derive an efficient proximal operation that involves fewer projections on to the ℓ_1-ball. This will then be used to derive an algorithm for $\mathbf{Prox}_{\ell_\infty^T}$ with reduced time complexity in Section 3.6.2. Time complexities for proximal operations with naive projection algorithms and the corresponding improved algorithms are summarized in Table 3.2.

3.6.1 Properties of ℓ_∞^T-norm.

We present two properties of the ℓ_∞^T-norm on trees that will be particularly useful with thin and tall structures. The following proposition shows that paths $\mathcal{P}(S(X), w)$ are preserved under $\mathbf{Prox}_{\kappa\ell_\infty^T}$ operations.

Proposition 1. *Constant value non-branching paths $\mathcal{P}(T, w)$ of T-structured vector w are*



(a) Trie-structured $w = [\,3\ 4\ 6\ 6\ 4\ 5\ 7\ 7\,]^\top$

(b) Tree-structured $w = [\,3\ 4\ 6\ 6\ 4\ 5\ 7\ 7\,]^\top$

(c) $\mathbf{Prox}_{0.8\ell_2^T}(w) = [\,2.8\ 3.5\ 4.8\ 4.3\ 2.3\ 3\ 5.6\ 4.9\,]^\top$

(d) $\mathbf{Prox}_{0.8\ell_\infty^T}(w) = [\,3\ 4\ 5.2\ 5.2\ 3.2\ 4.2\ 5.4\ 5.4\,]^\top$

Figure 3.11: An eight-dimensional trie shaped vector in subfigure (a) and the corresponding tree in (b). Values in the vector after $\mathbf{Prox}_{0.8\ell_2^T}$ are shown in (c) and that after $\mathbf{Prox}_{0.8\ell_\infty^T}$ in (d).

preserved under the proximal projection step $\boldsymbol{Prox}_{\ell_\infty^T}(w, \kappa)$.

Proof. Let T be a tree with constant value non-branching paths $\mathcal{P}(w, T)$ and $P \in \mathcal{P}$ a constant value non-branching path of two nodes $v_i, i = 1, 2$ with v_1 being the child of v_2. Node v_1 could further have a subtree below it. Let the subtree rooted at v_i be denoted g_i. Following the decomposition in (2.18), the proximal operator $\mathbf{Prox}_{\kappa\ell_\infty^T}(w)$ will require applying the following operations

$$\mathbf{Prox}_{\kappa\ell_\infty^T}(w) \equiv \mathbf{Prox}_{\kappa\ell_\infty}^{g_t} \circ \cdots \circ \left(\mathbf{Prox}_{\kappa\ell_\infty}^{g_2} \circ \mathbf{Prox}_{\kappa\ell_\infty}^{g_1}\right) \circ \cdots \circ \mathbf{Prox}_{\kappa\ell_\infty}^{g_s}(w).$$

Consider the operators applied on subtrees g_1 and g_2 of vector v obtained after applying all preceding operators on w,

$$\delta(v) \equiv \mathbf{Prox}_{\kappa\ell_\infty}^{g_2} \circ \mathbf{Prox}_{\kappa\ell_\infty}^{g_1}(v). \tag{3.9}$$

Each of the two $\mathbf{Prox}(\cdot)$ operations above is a residual of projection on the ℓ_1-ball of κ units which is a thresholding operation as mentioned in (3.7). We can equivalently rewrite the same operations as shown below by replacing the thresholding $[\cdot]_+$ with a $\min(\cdot, \cdot)$ operation:

$$g_1^v \leftarrow \mathtt{sgn}(g_1^v)\min(|g_1^v|, \tau_1) \tag{3.10}$$

$$g_2^v \leftarrow \mathtt{sgn}(g_2^v)\min(|g_2^v|, \tau_2) \tag{3.11}$$

These two operations are applied in that order and the g_2^v in (3.11) involves the g_1^v modified by (3.10). τ_1 and τ_2 are computed from vectors g_1^v and g_2^v respectively such that the condition in (3.8) is satisfied. We now look at different possibilities of ordering the pair (τ_1, τ_2) and check if the constant value non-branching path P is closed under (3.9). We us h_i to denote the value at the root variable of g_i^v and h_c to denote the ℓ_1-norm of the subtree below g_1^v (not including its root h_1).

1. $\tau_2 > \tau_1$.
   By using the condition on the pivot for projection in (3.8) on the second proximal step in (3.11) we get,
   $$h_2 - \kappa = \tau_2.$$
   Using the same condition on the first proximal step in (3.10), we have,
   $$h_1 + h_c - \tau_1 = \kappa \implies h_1 - \kappa < \tau_1.$$
   Since, $h_1 = h_2$ because P is a constant value non-branching path,
   $$h_1 - \kappa < \tau_1 \implies h_2 - \kappa < \tau_1 \implies \tau_2 < \tau_1.$$
   This contradicts with our initial assumption of $\tau_2 > \tau_1$ ruling this option out.

2. $\tau_1 \geq \tau_2$.
   The smaller of τ_1 and τ_2 dominates the entries of g_1^v in $\min(\cdot,\cdot)$ operations when applied in (3.10) and (3.11). The entry h_2 is only affected by τ_2. This makes (3.10) redundant if we knew τ_2 and the final solution of (3.7) can be obtained by simply applying $g_v^2 \leftarrow \min(g_v^2, \tau_2)$. This retains the inclusion of $P \in \mathcal{P}$ and the closure property holds in this case. However, note that using (3.7), τ_2 can be computed only after applying τ_1 in (3.10).

This proves that constant value non-branching paths are closed under $\mathbf{Prox}_{\kappa\ell_\infty^T}$ for two node paths. This can be extended to paths of arbitrary length using induction proving the closure property of $\mathcal{P}$ under $\mathbf{Prox}_{\kappa\ell_\infty^T}$ operator. ■

The next proposition shows how composition of $\mathbf{Prox}_{\kappa\ell_\infty}$ operations over constant value non-branching paths in $\mathcal{P}(S(X), w)$ can be simplified. Let $\mathbf{Prox}^P_{\kappa\ell_\infty^T}$ be composition of the following sequence of operations on non-branching path $P \in \mathcal{P}(S(X), w)$:

$$\mathbf{Prox}^P_{\kappa\ell_\infty^T} \equiv \mathbf{Prox}^{g^w_{|P|}}_{\kappa\ell_\infty} \circ \cdots \circ \mathbf{Prox}^{g^w_1}_{\kappa\ell_\infty}, \text{ such that } g_i^w \in \mathcal{G}\ \forall i \in P.$$

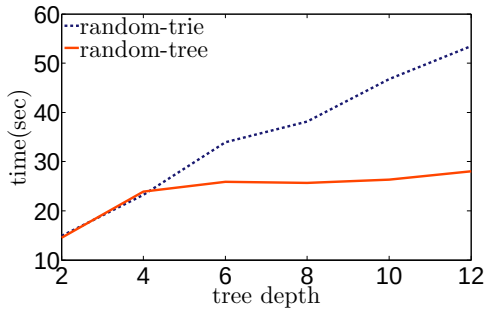
It is those sequence of $\mathbf{Prox}_{\kappa\ell_\infty}$ operations necessary for $\mathbf{Prox}_{\kappa\ell_\infty^T}$ for which the root node of the corresponding subtree lies in the subset of node P with grouping g_1^w through $g_{|P|}^w$ following the bottom-up ordering defined in $\mathcal{G}$. We now have the following simplification of the above proximal sequence over P.

Proposition 2. $\boldsymbol{Prox}^P_{\kappa\ell_\infty^T} \equiv \boldsymbol{Prox}^{g^w_{|P|}}_{\overline{\kappa}\ell_\infty^T}$ *with* $\overline{\kappa} = |P|\kappa$.

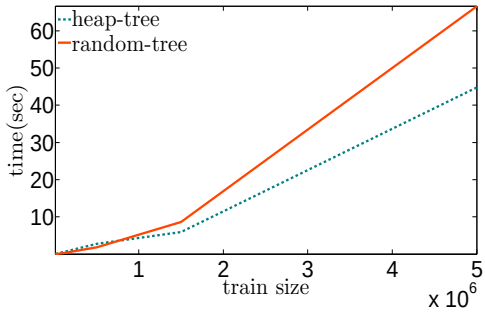
Proof. Consider the proximal operator applied on the two node non-branching path P in equation (3.7). It is clear from the proof of Proposition 1 that if we could directly estimate τ_2, we can make a single projection in place of the two in (3.7).

The case of $||g_2^v||_1 < \kappa$ leads to a zero vector on projection with $\tau_2 = 0$. It is easy to see that this case holds to for this proposition. Consider the other case of $\tau_2 > 0$. Let $z = \mathbf{Prox}^{g_1}_{\kappa\ell_\infty}(v)$ and $u = \mathbf{Prox}^{g_2}_{\kappa\ell_\infty}(z)$. From the condition for projection on ℓ_1-ball in (3.7) we have the following:

$$||g_2^v - g_2^z||_1 = \kappa \tag{3.12}$$
$$||g_2^z - g_2^u||_1 = \kappa \tag{3.13}$$



(a) Time comparison of $\mathbf{Prox}_{\ell_\infty^T}$ with random pivoting on suffix-tree against suffix-trie .



(b) Time comparison of random pivoting against heap-based pivoting on suffix-tree.

Figure 3.12: Comparison of time taken for $\mathbf{Prox}_{\ell_\infty^T}$ projection under various settings.

Since $\texttt{sgn}(v) = \texttt{sgn}(z) = \texttt{sgn}(u)$, we can write the following using the above two relations:

$$||g_2^v - g_2^u||_1 = ||(g_2^v - g_2^z) + (g_2^z - g_2^u)||_1 = ||g_2^v - g_2^z||_1 + ||g_2^z - g_2^u||_1 = 2\kappa$$

This can be extended to P of arbitrary length by induction principle and we recover $\mathbf{Prox}^P_{\kappa\ell_\infty^T} \equiv \mathbf{Prox}^{g^w_{|P|}}_{|P|\kappa\ell_\infty^T}$ proving the proposition. ■

The above propositions show that $M_v(X)$ structure is preserved under the proximal operation for ℓ_∞^T-norm and that it can be performed more efficiently. An immediate consequence of this is that the number of $\mathbf{Prox}_{\kappa\ell_\infty}$ projections required reduces from the number of unique suffixes $|S(X)|$ to the number of nodes in the tree-structured vectors $|M(X)|$. However, the proximal projection from Algorithm 7 will still be quadratic $O(|M(X)|^2)$ due to the linear time complexity of the pivoting method. In the next section we will present a heap-based pivoting method that will improve the complexity of the proximal step to $O(|M(X)| \ln |M(X)|)$.

3.6.2 Fast proximal projection with ℓ_∞^T-norm.

Linear time complexity of computing the threshold value in $\mathbf{Prox}_{\kappa\ell_\infty}$ is the reason for the quadratic complexity of the resulting $\mathbf{Prox}_{\kappa\ell_\infty^T}$ operator. The amortized linear time due to Bruckner (1984) is for projecting generic vectors but if the values were sorted in descending order it would be $O(k)$ where k is the number of entries larger than the pivot. Now if we have an upper bound on the number of largest few values we need to look at, we can bound the time it takes to compute the threshold in a sorted vector.

We verified empirically that in our case of language modeling, the number of largest entries required to compute the threshold is about 3 on average in vectors with as many as 10^5 entries. Depending on the regularization, the maximum number of values it needs to compute the threshold is between 20 and 30. This leads to significantly faster computation of the threshold. However, we need an efficient way to keep the values sorted to exploit this fact.

Algorithm 8 $w := \mathbf{Prox}_{\kappa\ell_\infty^T}(w)$ Proximal step for ℓ_∞^T on $\mathcal{G}$ using heap data structure.

Input: T suffix tree, w=[v c] tree-structured vector v with corresponding number of suffixes collapsed at each node in c, κ threshold
Initialize: $\mathcal{H} = \{\}$ # empty set of heaps
1 **for** $x \in$ DepthFirstNodeTraversal(T, *PostOrder*)
  $g(v, x) := \pi_{\ell_\infty^T}(w, x, c_x\kappa, \mathcal{H})$

Procedure: $q := \pi_{\ell_\infty}(w, x, \kappa, \mathcal{H})$
1 $\mathcal{H}_x = \text{NewHeap}(v_x, c_x, v_x)$
2 **for** $j \in \text{children}(x)$ # merge with child heaps
  $\tau_x = \tau_x + \tau_j$ # update ℓ_1-norm
  $\mathcal{H}_x = \text{Merge}(\mathcal{H}_x, \mathcal{H}_j), \mathcal{H} = \mathcal{H} \backslash \mathcal{H}_j$
3 $\mathcal{H} = \mathcal{H} \cup H_x, S = 0, C = 0, J = \{\}$
4 **if** $\mathcal{H}_x(\tau) < \kappa$, **set** $\mathcal{H}_x = 0$ **return**
5 **for** $j \in \text{OrderedIterator}(\mathcal{H}_x)$ # get max values
  if $v_j > \frac{S+(v_j c_j)-\kappa}{C+c_j}$
    $S = S + (v_j \cdot c_j), C = C + c_j, J = J \cup \{j\}$
  else break
6 $r = \frac{S-\kappa}{C}, \delta = 0$ # compute threshold
7 **for** $j \in J$ # apply threshold
  $\nu = \min(v_j, r), \delta = \delta + (v_j - \nu)$
  $\mathcal{H}_j(v) = \nu$
8 $\mathcal{H}_x(\tau) = \mathcal{H}_j(\tau) - \delta$ # update ℓ_1-norm

a. Heap structure on vector w holds three values (v, c, τ) at each node. v, c being value and its count, τ is the ℓ_1-norm of the sub-vector below. Tuples are ordered by decreasing value of v and $\mathcal{H}_j$ refers to heap with values in sub-tree rooted at j. Merge operation merges the heaps passed. OrderedIterator returns values from the heap in decreasing order of v.

We use a max-heap data structure to have access to values in the vector in a sorted order. A new heap is initialized at each leaf of $M(X)$ with the node's value pushed on the heap. At each intermediate node g_i, a max-heap is made by merging the max-heaps of its children and adding the value at the node itself. Then one can use an ordered iterator to fetch values in the heap in decreasing order.

The exact time complexities of merging and ordered iteration depend on the kind of heap used. We use a Fibonacci heap that allows for constant time merging of heaps (Cormen et al., 2009). The time required to access the top k entries of a Fibonacci heap containing d entries is $O(k \log d)$. This new pivoting method improves the time complexity of $\mathbf{Prox}_{\kappa\ell_\infty^T}$ to $O(k|M(X)| \log |M(X)|)$. The details of the heap-based $\mathbf{Prox}_{\kappa\ell_\infty^T}$ procedure are summarized in Algorithm 8.

Penalty			generalization	space efficiency	time efficiency
unstruc. ℓ_1 and ℓ_2^2			no	yes $O(\|M\|)$	yes $O(\|M\|)$
struc.	ℓ_2^T		yes	no $O(\|S\|)$	no $O(\|S\|)$
	ℓ_∞^T	rand. pivot	yes	yes $O(\|M\|)$	no $O(\|M\|^2)$
	ℓ_∞^T	heap	yes	yes $O(\|M\|)$	yes $O(\|M\| \log \|M\|)$

Table 3.3: Table showing the generalization and efficiency achievable using different penalties.

Figure 3.12 shows the time taken for ℓ_∞^T proximal step under different settings in one of the iterations of our experiments. Plot in Figure 3.12(a) compares the time in seconds for $\mathbf{Prox}_{\ell_\infty^T}$ on suffix-tree and suffix-trie with increasing tree depth. The $\mathbf{Prox}_{\ell_\infty^T}$ operation is computed using the random pivoting method of Bruckner (1984). The curve marked as `random-tree` makes only one projection operation per node in the tree using Proposition 2 as opposed to the `random-trie` method with one projection per node in the trie. We see how the per-iteration time difference between the two methods shows up with increasing depth of the tree.

In Figure 3.12(b) the random pivoting of Bruckner (1984) (curve marked `random-tree`) is compared against the pivoting method based on the Fibonacci heap described in Algorithm 8 (indicated by `heap-tree`). Time of the proximal step is plotted for increasing lengths of the training sequence. These plots show how suffix-tree can be used with the heap-based method to improve the time required for $\mathbf{Prox}_{\ell_\infty^T}$ projection. This allows for efficient use of longer contexts with a structure incorporating penalty.

3.7 Conclusion

In this chapter, we describe log-linear models for language with structured penalties. We show how the suffix tree structure can be used to penalize features at different depths in the tree with varying amount of regularization. Then, we show how ℓ_∞^T penalty allows using the suffix tree instead of the trie leading to a space efficient model. We present an algorithm for optimizing with ℓ_∞^T-norm on trees in a time efficient manner using heap data structure. Properties of different penalties involving their generalization characteristics, time efficiency of proximal updates and space complexity of resulting models are summarized in Table 3.3. In the next chapter we will consider the problem of building a taxonomy of words (or any generic classes) from a text corpus (or samples).

Sequence	$m_{\max}$	KN	wl_2^T	wl_∞^T	wl_2^2	wl_1
... evidence in the trial of a man **charged**	6	**1.8147**	1.9712	2.8732	5.0129	5.6018
... in the trial of a man charged **with**	6	**0.5308**	0.6035	0.6250	0.5673	0.5680
... the trial of a man charged with **killing**	6	1.4865	**1.3437**	2.3570	4.7021	3.2764
... hundreds of thousands of federal **workers**	5	0.2128	**0.1686**	0.1767	0.7049	0.2784
... could face the death_penalty if **convicted**	5	1.5792	**1.2810**	1.8719	2.6345	3.5182
<s> he gave no **details**	4	**2.6791**	**2.6791**	2.6714	5.3761	6.4595
... up_to #n inches of snow **fell**	4	2.6120	**2.4082**	2.0949	5.2029	4.9662
... tons of waste by the year **#n**	3	2.8066	**1.4466**	1.5153	3.7344	3.3907
... he could be sentenced to #n **years**	3	1.7450	**1.3990**	1.7451	2.1379	2.0838
... as evidence in the **trial**	2	6.9694	**6.4189**	6.8333	6.9015	7.2808
... until later in the **trial**	2	7.0168	**6.4274**	6.8338	6.9018	7.2808
... the way ⟨proper_noun⟩ is , i **think**	1	3.7997	**3.5184**	3.4675	3.5108	3.0356
... he returned a_few days later , he told **police**	6	**1.4711**	1.5202	2.8336	2.6429	2.9039
<s> besides recovering guns , **police**	5	1.9928	**1.3217**	2.1992	4.6574	5.5776
the phone for about an hour until **police**	5	**1.6915**	1.7140	1.9780	4.1142	5.4724
<s> no , the **police**	4	**2.6401**	2.9714	5.1949	6.0707	6.8311
<s> the suspects told **police**	4	1.4783	**1.1541**	1.6788	2.2475	2.8880
... , according to a **police**	3	**3.0019**	3.7169	4.3527	5.7334	6.7347
<s> according to **police**	3	4.7113	**4.5544**	5.6588	6.5267	7.8969

Table 3.1: Predictions from the test set illustrating behaviour of various 7-gram methods. The word in bold face is predicted using a context of $m_{\max}$ words preceding it. Values reported are negative of log-probabilities. The lower the value, the better is its performance.

Chapter 4

Efficient taxonomy learning

In this chapter we will consider the problem of building a hierarchy of classes from their observations. This is a problem of generic interest to understand the hierarchical structure within a set of classes but also specifically useful in various learning models involving large number of classes. Large multi-class problems using log-linear problems are slow to train and apply at test time because of the expensive normalization factor involved. Hierarchical grouping can be used to divide and conquer the computation of this expensive factor. The test time complexity per sample of the flat classifier is $O(K)$ where K is the number of classes. In the most optimal hierarchical setting of a complete binary tree, the test time complexity reduces to $O(\ln K)$ which is a significant gain in prediction time complexity. We describe this problem and propose a framework to build such hierarchies to overcome the issue. We particularly focus on the predictive performance of the tree-classifier for various depths of the tree.

4.1 Normalization in multi-class log-linear models

The formulation considered in the language model (3.1) is that of a log-linear model applied to a multi-class problem, that we recall here as,

$$\mathbf{P}(y \mid x) = \frac{e^{w_y^\top x}}{\sum_{k=1}^{K} e^{w_k^\top x}}, \tag{4.1}$$

where w_k is the weight vector of category k and x is the feature vector. There are K categories which in the case of language would be the $|V|$ words. An issue to common to such models is the computation of expensive normalization factor which is the computational bottleneck for large multi-class problems. The normalization factor in the above log-linear model is,

$$Z = \sum_{k=1}^{K} e^{w_k^\top x}$$

During training log-linear models require estimating the expectation of some quantity that requires computing Z with the current estimates of w_k at each iteration. This is also the case with maximum entropy updates in Table 1.1. For discrete sequences, the speed of computing this quantity can be improved by ensuring some calculations common

to multiple terms are not repeated as described by Wu and Khudanpur (2000). However, there is more to gain by using hierarchical organization of classes, a method that is generic and not restricted to discrete sequential data.

The complexity of computing the factor depends directly on the number of classes involved. A simple workaround is to group classes or categories into fewer meta-classes. The problem is then a two-step process of choosing the right cluster followed by picking the best word from the chosen cluster. Each of these two steps have smaller Z factors to compute that are together cheaper than that of the original problem. The grouping process can be repeated to multiple levels to generate a hierarchy of classes (also called a taxonomy) where each non-leaf node in the hierarchy is a meta-class. A classifier is trained at each node for the meta-classes and a sample traverses the hierarchy from its root to a leaf during classification. This hierarchy directly reduces the time required for the normalization factor by reducing the number of choices at each level of the classifier. The use of such a taxonomy for classification is referred to as the hierarchical classification problem and sometimes also called the *Pachinko machine*.

Hierarchical classification and its extensions have been thoroughly studied in the literature. Extensions include problems with multi-label classification, hierarchies where internal nodes could assign labels, and directed acyclic graphs of classes instead of simple hierarchies (see Silla Jr. and Freitas, 2011, for a survey of hierarchical classification methods). However, the focus of these methods is usually on how best to train the hierarchical classifier given the taxonomy. We rather study the problem of building the taxonomy from sample distributions.

The problem of building a taxonomy given data samples has also been studied in the literature. In the particular case of reducing multi-class problems to binary classification, decision trees like CART are the earliest known (see Hastie et al., 2009). CART models use naive decision stumps like thresholds on predictors that are simple for moderately complex data distributions. Vural and Dy (2004) and Tibshirani and Hastie (2007) generalize this idea to more generic cases.

More recent works in this line are that of Griffin and Perona (2008),Weston et al. (2010) and Gao and Koller (2011). Griffin and Perona (2008) and Weston et al. (2010) use a surrogate classifier to build a confusion matrix that is used with spectral clustering methods to build the tree in a top-down manner. Griffin and Perona (2008) also propose simply merging two most confusing categories at each step by looking for the largest entry in the confusion matrix. Alternatively, Gao and Koller (2011) use a max-margin approach extending the idea of Tibshirani and Hastie (2007) to a principled optimization problem. However, their formulation is non-convex and sensitive to initialization. Bart et al. (2011) consider the same in a Bayesian framework using nested Chinese restaurant process and study the usefulness of taxonomic structures of image collections for human browsing and navigation.

Particularly for modeling language, it was tried by Goodman (2001), Mikolov et al. (2011) and Mnih and Hinton (2008). Goodman (2001) uses the agglomerative clustering method called Brown clustering proposed by Brown et al. (1992) to build a two level hierarchical language model of picking the most probable cluster and then the most probable word from the chosen cluster. Brown clustering initializes one cluster per word and greedily merges a pair of clusters while maximizing the average mutual information of adjacent

clusters (clusters with words belonging to them appearing next to each other in the corpus). Mikolov et al. (2011) use a similar hierarchical organization for neural networks. Mnih and Hinton (2008), on the other hand, use a bootstrap method to build a hierarchy over the words.

In most classification tasks, taxonomies are either built manually as in Wordnet (2010) or borrowed from a related ontology (see Deng et al., 2009). In either case the hierarchy might be unsuitable for the task at hand. For example, Mnih and Hinton (2008) prefer to build the tree by bootstrap rather than using the existing hierarchy from Wordnet (2010).

This problem bears relation to parsimony studied in statistics and mathematical biology (Dunn and Everitt, 2004). In phylogenetics one seeks the most likely tree that explains the given data referred to as maximum parsimony (Kolaczkowski and Thornton, 2004; Steel, 2002). Approaches for maximum parsimony include maximum likelihood inference, distance matrix methods and Bayesian inference. While Bayesian inference resorts to sampling, frequentist algorithms often employ greedy methods returning suboptimal trees.

The crux of the problem lies in the fact that despite all samples being labeled with appropriate classes, the grouping of classes is inherently unsupervised. The steps that lead us from observations to a tree-classifier can be split into two parts either of which could effect the overall predictive performance:

1. *Tree construction*: Each leaf of a binary tree can be uniquely encoded by a binary string of 1s and 0s (for left and right edges). Each such string is a series of classifier decisions at the nodes. A classifier makes similar decisions for similar feature vectors and hence classes closer in feature space should have similar codes. This motivates grouping of classes by similarity of features making tree construction dependent on the measure of the proximity of the categories. These details are discussed in Section 4.2.

2. *Classifier training*: Learning parameters for node-specific classifiers of the tree can be formulated in different ways. The choice of the formulation could determine the size of the optimization problem. A simple option is to train one classifier per node independently. This can be any kind of classifier including those that return probabilities like the nested logit (Train, 2003). Alternatively, one can use a classifier where each gradient update depends on a multiple nodes in the tree (Weston et al., 2010). We independently learn multinomial regression coefficient at each of the nodes. Section 4.3 discusses these details

Our goal is to build a hierarchy that speeds computation while trying to retain performance of a simple *flat* model. We study empirically the performance of five different tree-building approaches by measuring predictive log-likelihoods.

4.2 Tree embedding of classes

A *label tree* is a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{Z})$, where $\mathcal{V}$ is the set of nodes or vertices and $\mathcal{E}$ the set of directed edges and $\mathcal{Z} = \{z_v\}_{v=1}^{|\mathcal{V}|}$ is the set of label-sets z_v. Each label-set z_v indicates a subset of the complete list of classes V that are relevant to the vertex v. V

Algorithm 9 Hierarchical organization of categories.

Top-down approach

Input: $X \in \mathbb{R}^{n \times d}, Y \in \mathbb{R}^{n \times K}$
Output: $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{Z})$
Initialize: $\mathcal{V} = \{v\}, \mathcal{E} = \varnothing, \mathcal{Z} = \{z_v\}$ with $z_v = [1, \cdots, K]$
repeat until required depth
  for $v \in \mathcal{V}$, v is leaf and $|z_v| > 1$
    find optimal partition (s, t) such that $z_v = z_s \cup z_t$
    `#update tree`
    $\mathcal{V} \leftarrow \mathcal{V} + \{s, t\}$
    $\mathcal{E} \leftarrow \mathcal{E} + \{e_{v \to s}, e_{v \to t}\}$
    $\mathcal{Z} \leftarrow \mathcal{Z} + \{z_s, z_t\}$
return $\mathcal{T}$

Bottom-up approach

Input: $X \in \mathbb{R}^{n \times d}, Y \in \mathbb{R}^{n \times K}$
Output: $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{Z})$
Initialize: $\mathcal{V} = \{v_1, \cdots, v_K\}, \mathcal{E} = \varnothing, \mathcal{Z} = \{z_1, \cdots, z_K\}$ with $z_k = k$
$A = \mathcal{V}$ #A `is active set of nodes that can be merged`
repeat until required depth
  find optimal pairing $P \in \{0, 1\}^{|A| \times |A|}$ and $P = P^\top$
  for unordered $(s, t) \in A$ such that $P(s, t) = 1$
    `#update tree`
    $\mathcal{V} \leftarrow \mathcal{V} + \{v\}$
    $\mathcal{E} \leftarrow \mathcal{E} + \{e_{v \to s}, e_{v \to t}\}$
    $\mathcal{Z} \leftarrow \mathcal{Z} + \{z_v\}$, where $z_v = z_s \cup z_t$
    $A \leftarrow A - \{s, t\} + v$
`#add root variable with active nodes as children`
$\mathcal{V} \leftarrow \mathcal{V} + \{r\}$
$\mathcal{E} \leftarrow \mathcal{E} + \{e_{r \to A_i}\}_{i=1}^{|A|}$
$\mathcal{Z} \leftarrow \mathcal{Z} + \{z_r\}$, where $z_r = [1, \cdots, K]$
return $\mathcal{T}$

would the vocabulary list for language modeling. Let $K_v = |z_v|$ denote the number of labels at vertex v.

Each vertex except the root has a single parent, but can have multiple children. We are interested in a balanced label tree on top of the K classes based on local optimal decisions. Given a data point, vertex-specific classifiers will then successively decide which branch of the tree to follow (see Figure 4.1). We derive top-down and bottom-up approximations of flat multi-class classifiers by a tree of given depth.

In the top-down approach, the multi-class classifier is greedily approximated by recursively partitioning label-sets. At every level of the tree, each vertex partitions a subset of the classes into two (approximately) balanced meta-classes, meaning that each meta-class contains roughly half the classes originating from the parent vertex. The labeling of the classes at vertex v, denoted z_v, is a vector of size K_v indicating the child to which

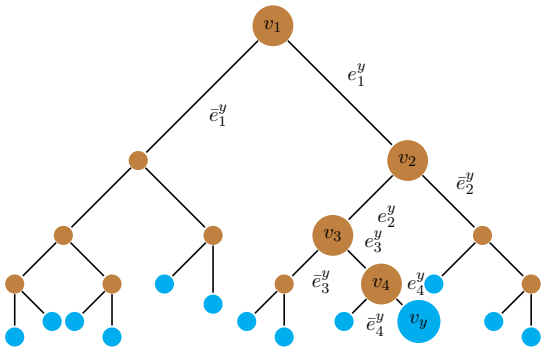


Figure 4.1: Binary classification tree. Leaf nodes are shown in blue and the rest in brown. The path to the leaf y is marked by vertices $\mathcal{P}(y) = \{v_1, v_2, v_3, v_4, v_y\}$ and edges $\{e_1^y, e_2^y, e_3^y, e_4^y\}$. The edges $\{\bar{e}_1^y, \bar{e}_2^y, \bar{e}_3^y, \bar{e}_4^y\}$ are the alternative branches on the path.

the label is pushed. Trees generated by the top-down approaches here are binary with $z_v = \{-1, +1\}^{K_v}$. This process is repeated down to the required depth. Each leaf node then, either corresponds to a single class, or a small number of classes, in which case it contains a multi-class classifier. Hence, in this setting the choice of tree-depth will realize a trade-off between classification accuracy and classification speed.

In the bottom-up approach, classes are successively aggregated up to required level. This procedure creates a tree that could be non-binary at the top. Again, the tree depth determines the trade-off between classification accuracy and classification speed. Given the tree, we learn node-specific classifiers in the same way.

Algorithm 9 summarizes the procedures to build the hierarchy from observations in top-down and bottom-up approaches. Step 4 of both top-down and bottom-up approaches in the algorithm indicates to the grouping of categories. In the top-down case it is the optimal partitioning and in the bottom-up it is the optimal merging. We need a measure of closeness of categories for either of the operations (split or merge). We use the residuals under the square loss for this measure, that we describe in Section 4.2.1. At each step of the iteration, we need to recompute this measure for a particular subset of the categories that we wish to split or merge. This update of the residuals is described in Section 4.2.2. Then, we need an algorithm to carry out the split or merge given the subset of categories. We discuss different methods for this grouping in Sections 4.2.3 and 4.2.4. Finally, we learn the classifiers at each node of the recovered label tree which we describe in Section 4.3. We evaluate these models in Section 4.4 and conclude in Section 4.5.

4.2.1 Measuring the class proximity

Let $X \in \mathbb{R}^{n \times d}$ and $Y \in \mathbb{R}^{n \times K}$ be respectively the d dimensional feature vectors and the corresponding class label indicator vectors. Columns of Y correspond to class labels and rows to observations. Each row of Y contains a single entry equal to 1, all others being equal to zero; if observation i belongs to class j, then $Y_{ij} = 1$. We measure class similarity through the residuals under a ridge regression model. Consider minimization of the ℓ_2-

regularized square loss:

$$W^* = \underset{W}{\operatorname{argmin}} \ \|Y - XW\|^2 + \delta\|W\|^2, \tag{4.2}$$

where $W \in \mathbb{R}^{d\times K}$ with columns $[w_1, \cdots, w_k]$ corresponding to the K categories. Setting the gradient of the above equation (4.2) to zero gives us,

$$2X^\top(Y - XW) - 2\delta W = 0,$$

or equivalently,

$$W^* = (X^\top X + \delta I_d)^{-1} X^\top Y, \tag{4.3}$$

where I_d is a $(d \times d)$ identity matrix.

Residual matrix. Substituting this closed-form solution for W^* from (4.3) back into the objective in (4.2) leads to the following matrix of squared residuals:

$$Q = Y^\top(I_K - X(X^\top X + \delta I_d)^{-1}X^\top)Y. \tag{4.4}$$

The Q matrix contains values that measure how close the corresponding category pairs are under the square loss. The goal for bottom-up or top-down tree building approaches will be to make partitions of the label-sets with a small product of residuals.

The process of partitioning (or merging for bottom-up) will be repeated iteratively until the required depth as described in Algorithm 9. This will require us to recompute the residual matrix for different label-sets. Consider a node $v \in \mathcal{V}$ with $K_v = |z_v|$ categories, $z_v \subset [1, \cdots, K]$. The square residuals Q_v for categories within node v will follow (4.4),

$$Q_v = Y_v^\top(I_{K_v} - X_v(X_v^\top X_v + \delta I_d)^{-1}X_v^\top)Y_v, \tag{4.5}$$

where X_v is the subset of observations belonging to categories in z_v and Y_v is the corresponding class indicator matrix.

Efficient computation of residual matrix. The computation of Q (or any Q_v) can be expensive as it requires the inversion of $d \times d$ matrix, which is $O(d^3)$. Depending on the number of data points n and the number of features d, we can compute the residuals more efficiently as follows:

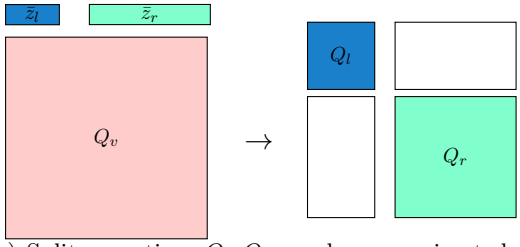
1. In large feature spaces with few observations (large d, small n), we use,

$$Q = Y^\top(I_K - XX^\top(XX^\top + \delta I_d)^{-1})Y,$$

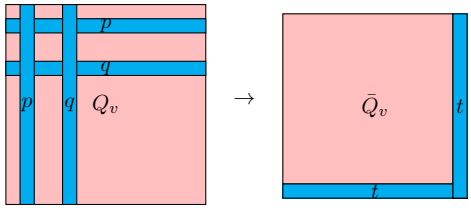
   where the complexity is reduced to $O(n^3)$ by using the matrix inversion lemma.

2. In large feature spaces with many observations (large d, large n), we use,

$$Q = Y^\top(I_K - R(R^\top R + \delta I_d)^{-1}R^\top)Y, \qquad XX^\top = RR^\top, \quad R \in \mathbb{R}^{n\times m},$$



(a) Split operation. Q_l, Q_r can be approximated with submatrices from rows and columns corresponding to labels $\bar{z}_l, \bar{z}_r$ respectively. Such a split operation is only an approximation.



(b) Merge operation. Rows (columns) p, q removed from Q_v and added to make row (column) t which is appended to make the new residual matrix $\bar{Q}_v$.

Figure 4.2: Residual matrix estimation in split and merge operations.

The idea it use matrix inversion lemma and write Q as an inversion of $XX^\top$. The $XX^\top$ is then approximated by a low-rank decomposition such that $XX^\top = RR^\top$ where R has dimensions $m \ll d$. Then we again use the matrix inversion lemma to write it as an inversion of $R^\top R$.

There are various algorithms to achieve this low-rank decomposition like sparse greedy approximations (Smola and Scholkopf, 2000), incomplete Cholesky decompositions (Fine and Scheinberg, 2001), or Nystorm's method (Williams and Seeger., 2000). They approximate $XX^\top$ by exploiting the fact that X can be represented in a lower dimensional subspace and that the rank of the resulting matrix m is small in practice. We use the incomplete Cholesky decomposition in our algorithm (also see Bach and Jordan, 2005; Golub and Loan, 1996). The decomposition is $O(m^2 n)$, i.e., linear in n and the resulting matrix can be inverted in $O(m^3)$ as opposed to $O(d^3)$ or $O(n^3)$.

4.2.2 Split and merge operations

At an given iteration of the tree building process, we need to choose the best split of the classes into two groups for top-down approach and correspondingly the best merge for bottom-up approach. We show below that the update for residual matrix Q_v can be performed efficiently for merge operations but it needs to be recomputed for split operations at every iteration.

Merge operations on residual matrix Consider the situation in Figure 4.2(b) where vertices p, q of matrix Q_v are merged to make vertex t. Merging here implies, labels from sets z_p, z_q both receive the same labeling in vertex t. We show that this can be achieved by merging the columns corresponding to vertices p, q in the indicator matrix Y_v while keeping X_v unchanged. Let Y_v be $(n_v \times k)$ indicator matrix, then the updated indicator matrix $\bar{Y}_v$ is as follows

$$\begin{aligned} \bar{Y}_v &= Y_v U \\ \text{where, } U_{k\times(k-1)} &= \left[u^i_{k\times 1}\right], i \in \{1, \cdots, k\} \cup \{p+q\} \setminus \{p, q\} \\ u^i_{k\times 1}(j) &= \begin{cases} 0 & 1 \le j \le k, j \neq i \\ 1 & j = i \end{cases} \end{aligned}$$

$u^i_{k\times 1}$ is a k-dimensional unit vector with 1 in the ith position. U is a $(k \times (k-1))$ binary matrix made by removing two columns u^p, u^q from I_k and appending the column $(u^p + u^q)$ to it. The residual matrix after merge $\bar{Q}_v$ takes the following form,

$$\begin{aligned}\bar{Q}_v &= \bar{Y}_v^\top [I - X_v(X_v^\top X_v)^{-1}X_v^\top]\bar{Y}_v \\ &= U^\top Y_v^\top [I - X_v(X_v^\top X_v)^{-1}X_v^\top]Y_v U \\ \bar{Q}_v &= U^\top Q_v U\end{aligned}$$

where Q_v is the residual matrix before merge operation. The update U neatly factors out of the residual matrix making it simple to compute the new residuals.

Hence, we can exactly compute the residuals $\bar{Q}_v$ after a merge operation from Q_v by simply summing the appropriate row (or columns) of Q_v. However, in the case of split operation there is no convenient trick to derive the modified residuals which must be recomputed using the new label sets.

Split operations on residual matrix Consider the case where label set z with residual Q is split into z_l and z_r as in Figure 4.2(a). Without loss of generality we can assume that X and Y can be ordered by labels z_l followed by z_r:

$$Y = \begin{bmatrix} Y_l & 0 \\ 0 & Y_r \end{bmatrix}, \qquad X = \begin{bmatrix} X_l \\ X_r \end{bmatrix}. \tag{4.6}$$

Recall that $Q_v = Y_v^\top (I_{K_v} - X_v(X_v^\top X_v + \delta I_d)^{-1}X_v^\top)Y_v$. Rewriting Q_v in terms of X_l, X_r, Y_l, Y_r from (4.6), we get the following,

$$Q_v = \left[\begin{array}{c|c} Y_l^\top Y_l & 0 \\ \hline 0 & Y_r^\top Y_r \end{array}\right] - \left[\begin{array}{c|c} Y_l^\top X_l B^{-1} X_l^\top Y_l & Y_l^\top X_l B^{-1} X_r^\top Y_r \\ \hline Y_r^\top X_r B^{-1} X_l^\top Y_l & Y_r^\top X_r B^{-1} X_r^\top Y_r \end{array}\right],$$

where $B = X^\top X + \delta I$.

If we were to extract the submatrix of Q_v corresponding to the labels in z_l, we would have,

$$Q_l = Y_l^\top \left(I - X_l(X^\top X + \delta I)^{-1}X_l^\top\right) Y_l.$$

But the correct matrix $\bar{Q}_l$ would be $\bar{Q}_l = Y_l^\top \left(I - X_l(X_l^\top X_l + \delta I)^{-1}X_l^\top\right) Y_l$. Note that they are not equivalent and since the matrix inversion makes it impossible to recover $\bar{Q}_l$ that we need by simply modifying Q_l. Hence, we require to recompute the residual after every split operation in the top-down approach.

4.2.3 Top-down procedure

Consider a vertex v with K_v classes and let $\{X_v, Y_v\}$ be the set of observations belonging to one of these K_v classes, where $X_v \in \mathbb{R}^{n_v \times d}$ and $Y_y \in \{0,1\}^{n_v \times K_v}$. At vertex v, we partition the K_v classes into two disjoint subsets corresponding to the two subtrees. This can be done by picking a label vector $z_v \in \{-1, +1\}^{K_v}$, each entry of which entry describe which of the two subtrees the corresponding label belongs. Under a multiple regression model, we obtain the following optimization problem:

$$\min_{z_v} \min_{w_v} \|Y_v z_v - X_v w_v\|^2 + \delta \|w_v\|^2. \tag{4.7}$$

Substituting back β_v from (4.3) into the above equation simplifies it into a form that can be rewritten as the following trace minimization problem:

$$\underset{z_v}{\text{minimize}}\ \mathbf{tr}\left\{z_v z_v^\top Q_v\right\}. \tag{4.8}$$

where Q_v is given by (4.5) and X_v is centered. Equation (4.8) can be recognized as the standard maximum cut problem (Cormen et al., 2009) (also referred in the literature as the Chinese postman problem or the route inspection problem (see also Part VII of Schijver, 2003)). While maximum cut is non-convex and NP-hard, there exist efficient approximation schemes with guaranteed bounds, such as the classical semidefinite programming relaxation known as the 0.878-approximation algorithm by Goemans and Williamson (2005).

Low-rank approximation: Here, we will take a different route by noting that (4.8) is closely related to the clustering formulation DIFFRAC (Bach and Harchaoui, 2007).[1] First, note that $A_v = z_v z_v^\top$ has constant diagonal, is positive definite and has rank-one. As in (Bach and Harchaoui, 2007; Goemans and Williamson, 2005), we relax the rank constraint to obtain a semidefinite program. Second, we add $\kappa 1 1^\top$ to the objective to avoid the trivial solution of a constant vector (Joulin et al., 2010). This leaves us with the following relaxed optimization problem:

$$\begin{aligned}\underset{A_v}{\text{minimize}}\ &\mathbf{tr}\left\{A_v\left(Q_v + \kappa 1 1^\top\right)\right\} \\ \text{subject to}\ &\mathbf{diag}(A_v) = \mathbf{1},\ A_v \succcurlyeq 0.\end{aligned} \tag{4.9}$$

The value of the parameter κ is reasonably set to $\frac{1}{K_v^2}$ (Joulin et al., 2010). We used the solver proposed by Journee et al. (2010), which returns a matrix A_v of rank at least two. This matrix is then projected to the space of rank one matrices by taking the eigenvector corresponding to the largest eigenvalue of A_v. This vector is then thresholded at zero to recover cluster assignments. This problem is referred to as `DIFF` hereafter.

Spectral relaxation: A simpler relaxation to the problem is to impose $z_v^\top z_v = K_v$, which corresponds to relaxing the constraints $\mathbf{diag}(A_v) = 1$ to $\mathbf{tr}\{A_v\} = K_v$. This leads to

$$\begin{aligned}\underset{z_v}{\text{minimize}}\ &\mathbf{tr}\left\{z_v z_v^\top Q_v\right\} \\ \text{subject to}\ &z_v^\top z_v = K_v.\end{aligned} \tag{4.10}$$

The solution to this problem is the eigenvector associated with the smallest eigenvalue of Q_v, which is then thresholded at zero to recover cluster assignments. This method is referred to as `SPEC`.

4.2.4 Bottom-up procedure

The above two methods, `SPEC` and `DIFF`, build the tree in a top-down fashion by partitioning the matrix of squared residuals. They could alternatively be built in a bottom-up fashion by merging vertices agglomeratively. A baseline would be merging vertices one-by-one greedily by picking the pair corresponding to the smallest off-diagonal entry in the residual matrix. We refer to this as `OBYO` which is similar to the bottom-up method in (Griffin and Perona, 2008). Note that trees learned by this method have no reason to be balanced.

[1] Instead of clustering data points like in DIFFRAC, we cluster classes of data points.

Hungarian algorithm: An attractive alternative is to consider it as an optimal assignment problem and use the Hungarian algorithm (Frank, 2004) to pair nodes . The algorithm optimally assigns nodes in polynomial time and every vertex is paired with another. However, the compulsory pairing of nodes causes unwanted merges among them. The residual matrix is modified to avoid such unwanted merges by adding dummy nodes which is classically done in assignment problems:

$$\bar{Q}_v = [Q_v + \eta I_{K_v} \quad \gamma 1_{K_v \times \phi K_v}], \tag{4.11}$$

where η is a large constant added to the diagonal to avoid self-assignments, ϕ is the fraction of real vertices added as dummy vertices and γ is the weights on the edges between real and dummy vertices. The optimization problem for this method, which we refer to as `HUNG`, is defined as

$$\begin{aligned} &\underset{B_v}{\text{minimize}}\ \mathbf{tr}\left\{B_v^\top \bar{Q}_v\right\} \\ &\text{subject to } B_v 1_{(1+\phi)K_v} = 1_{K_v}, \quad B_v^\top 1_{K_v} \leq 1_{(1+\phi)K_v}, \end{aligned} \tag{4.12}$$

where $B_v \in \{0,1\}^{K_v \times (1+\phi)K_v}$.

4.3 Node-specific classifiers

Once we build a tree $\mathcal{T} = \{\mathcal{V}, \mathcal{E}, \mathcal{Z}\}$ using any of the above methods, we need to learn node-specific classifiers. Each of these classifiers is a probabilistic multinomial classifier with independent parameterization. Thus, we are faced with a learning problem in a tree-structured directed graphical model with independent parameterizations and fully observed data. Regularized maximum likelihood thus decouples and we can learn each classifier independently. At test time, we can simply take the product of the individual output probabilities to obtain the joint probabilities (Bishop, 2006).

We consider ℓ_2-regularized linear classifiers, and one important practical issue is the selection of hyper-parameters, i.e., the regularization parameter at each node. We have experimented with different policies in order to choose them at different nodes of the tree:

1. using a single value at all nodes, referred as `single` below,
2. picking the regularization parameter independently for each node referred as `indep` below,
3. using a hierarchical Gamma prior on them, referred as `gamma` below.

Regularization parameters in the first two cases are chosen by cross-validation over a list of candidate values. In the case of Gamma prior, if λ_v denotes the parameters for node v and $\mathcal{L}_v(w_v)$, the negative log-likelihood of the linear classifier represented by w_v at node v, then we assume that λ_v is sampled from the gamma distribution $\mathbf{P}(\lambda) = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda}$, with α controlling the shape and β the rate of decay (these parameters are common to all nodes). The global penalized log conditional-likelihood is given by (up to constants)

$$\mathcal{L}(w, \lambda) = \sum_v \mathcal{L}_v(w_v) + \lambda_v \|w_v\|^2 - \alpha \log \beta - (\alpha - 1) \log \lambda_v + \beta \lambda_v.$$

$\mathcal{L}(w, \lambda)$ is then optimized by alternating between w and λ (Do et al., 2007). This amounts to alternating between regular independent multinomial regression and the update $\lambda_v = \frac{\alpha-1}{\beta+\|w_v\|^2}$. Cross-validation is done over a grid of values for $\alpha > 1$ and $\beta > 0$. This prepares the necessary hierarchy along with the classifier training at each of its nodes.

4.4 Evaluation of taxonomy methods

Detailed experiments are run on a smaller synthetic dataset while only efficient methods are tried on larger real datasets to verify their use in practice. PO50 and WS50 are synthetic datasets with 50 classes of 200 dimensional features with about 200 samples per class. There is partial overlap among samples from different classes in PO50 while classes are well separated in the WS50 dataset. Specific details on how the synthetic data were generated are provided in the Appendix B.

We measure efficiency of a tree by the average number of dot products required on a test dataset. Predictive performance is evaluated in terms of prediction accuracy and negative log-likelihood. The ranking of methods varies with the measure used for performance. The best performing method when considering accuracy might not be as good with logloss. Performance plots show accuracy and loglikelihood of models trained with varying tree depths and also the number of dot-product operations required on average on the test dataset.

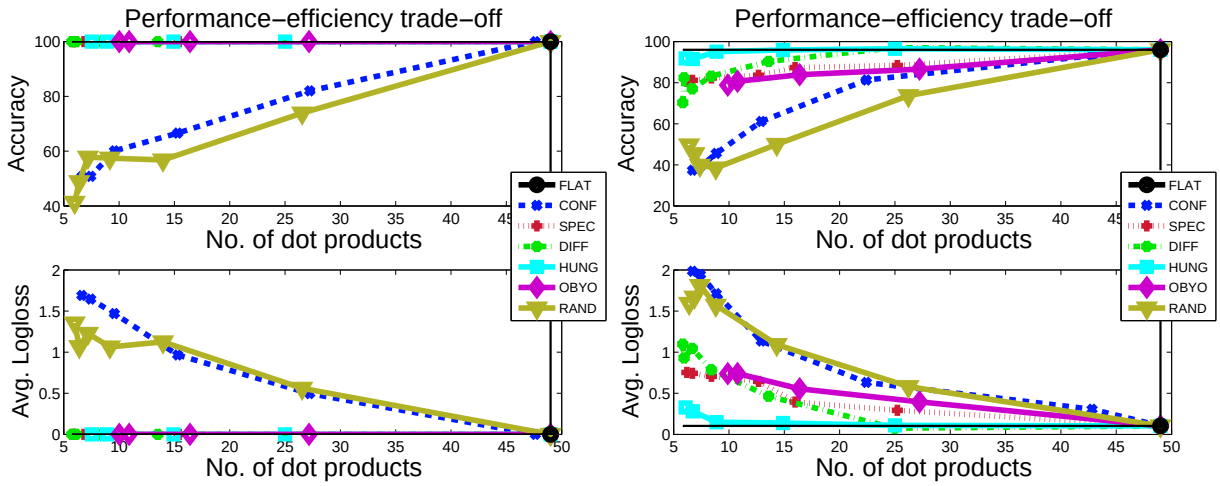
A depth of 1 indicates tree after one split or merge operation. More merge operations are performed in `OBYO` at each level since a single merge in this method does have the same effect on the tree depth as with the other methods. Flat always has a depth of zero and is plotted with a horizontal line for ease of comparison.

We compare top-down approaches (`DIFF` and `SPEC`), bottom-up approaches (`HUNG` and `OBYO`) and random tree building `RAND`, as well as the method `CONF` proposed by Griffin and Perona (2008) and Bengio et al. (2010). `CONF` runs a surrogate classifier and makes a confusion matrix from its predictions on a validation dataset. This confusion matrix is then symmetrized by taking the average with its transpose and used as a measure of the proximity of classes. This is followed by a standard spectral method to cluster classes. Confusion matrix often has zeros when classes are well separated. We observed that this turns out to be detrimental to spectral methods that compute the eigenvectors of this matrix. In cases where the eigenvector cannot be stably estimated, we use a random split of the classes.

Figure 4.3 shows the performance efficiency trade-off for all methods on synthetic dataset. As expected `RAND` performs worse in terms of accuracy and negative log-likelihood on the test. When all classes are well separated, the classification problem is relatively easy. All principled tree building heuristics are able to group similar classes together, eventually leading to performances very close to the flat multi-class classifier.

When there is some overlap, it is more challenging to identify similar classes. The problem becomes more troublesome when the tree is deeper, as the subset of classes are more alike. Despite the overlap, all methods (apart from `RAND`) show performances close to the flat multi-class classifier, with a slight advantage for bottom-up approaches.

It is interesting to note that on both synthetic examples, the method of Bengio et al. (2010) performs similar to the random tree-based classifier when the classes are well sep-



(a) Set of well separated classes (WS50).

(b) Classes with partial overlap (PO50).

Figure 4.3: Performance plots for varying tree depth for various methods on WS50 dataset on the left and on PO50 dataset on the right.

arated. This is because the confusion matrix has too many zeros and almost always a random split is performed to resolve for a clustering. Evidently, in such cases a confusion matrix is not an appropriate measure of proximity of classes. Also, a random grouping should certainly be avoided to organize hierarchies.

Effect of various regularization strategies discussed in the paper are examined in Figure 4.4. The simplest strategy of a single regularization parameter for all nodes worked as well as the other more expensive methods. This single parameter was chosen by cross-validation.

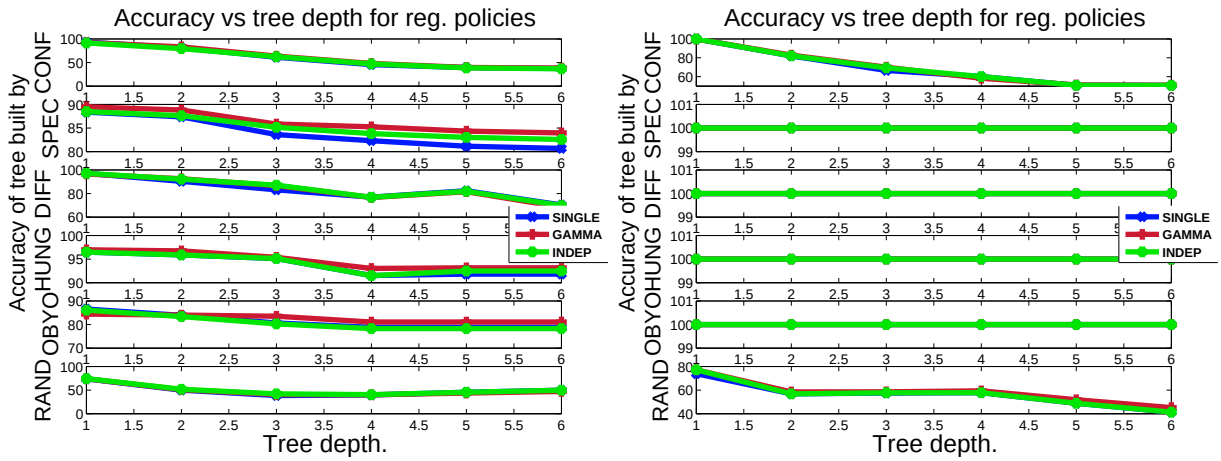
We investigate the performance of different tree building techniques on the following datasets:

1. CIFAR: 100 classes of images in a 384 dimensional space. There are 1000 samples per class which are equally split into for train, cross-validation and test.

2. Vehicle: 263 classes of 262K images. 500 dimensional projections of 4096 dimensional Fisher vectors of the images are used.

3. APnews subset: 2114 classes of 100 dimensional features. Each word is a class from the subset of the original APnews corpus. Features were derived by CCA on bigram co-occurances (Dhillon et al., 2011).

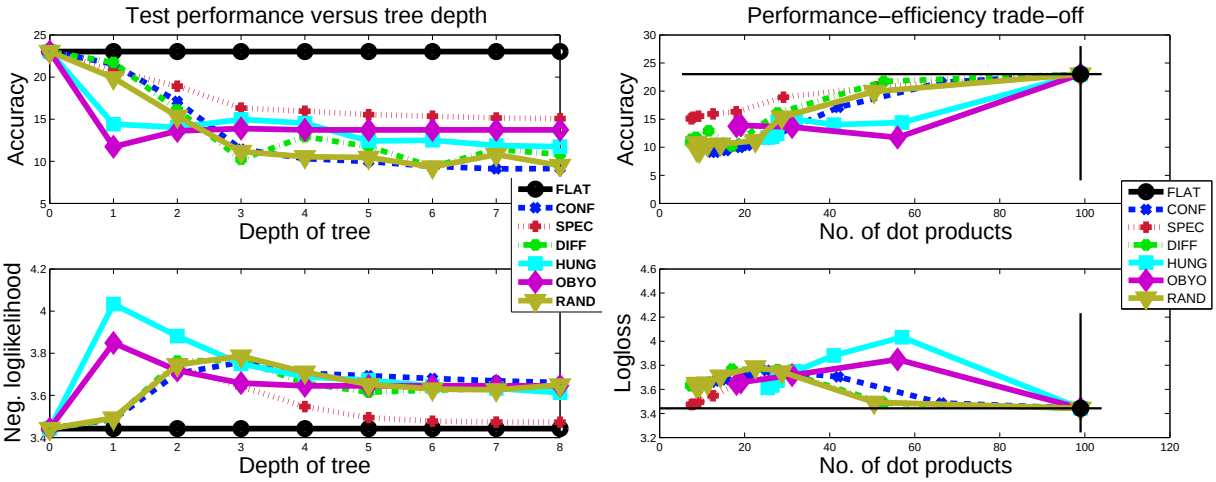
Figure 4.5 shows the results on the CIFAR image classification dataset. `SPEC` performs best among the tree-based methods all of which usually perform better than `RAND`. However, in this case the flat multi-class classifier has a considerable advantage over the approximate methods by accuracy measure. The log-loss plot also reveals a similar trend where `SPEC` outperforms other methods. Strangely, in the log-loss plot , after some initial degradation performance improves slightly with deeper trees. This happens because the most loss performance from simple flat multi-class of happens at the lowest level in the tree that



(a) Predictive performance on PO50 data with overlap of categories.

(b) Predictive performance on WS50 data with separated categories.

Figure 4.4: Performance variation with regularization policy.

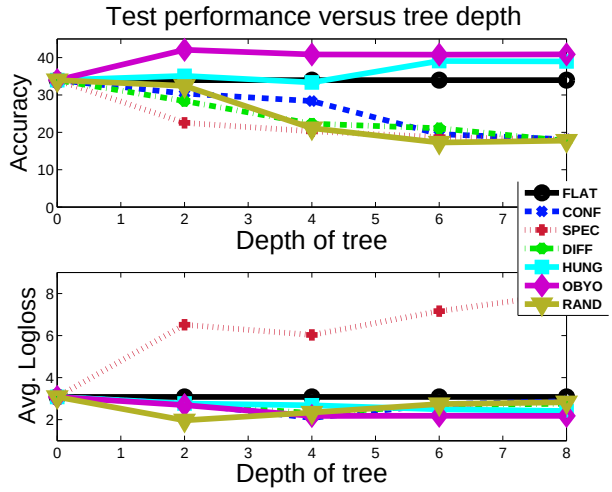


(a) Performance against tree depth.

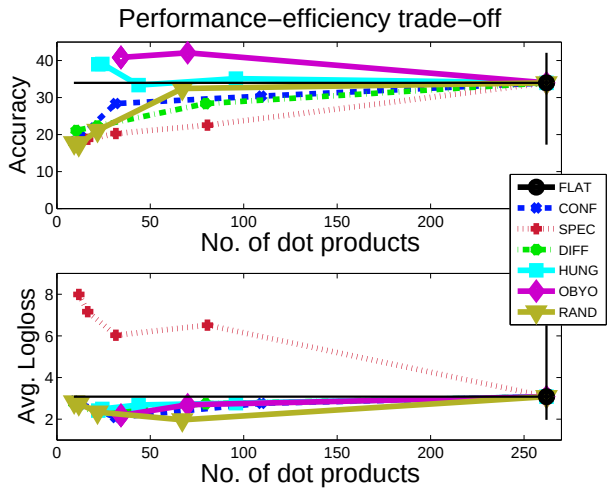
(b) Performance against average number of dot-products on test.

Figure 4.5: Performance variation with depth and performance efficiency trade-off on CIFAR dataset.

involves the multi-class classifier of that lead node. So, it mostly depends on the grouping at the lowest level of the tree which were not very suitable clusterings for a flat multi-class. With larger trees, the clusters got smaller with classifiers being more confident of the correct label improving the log-loss. This, however, does not apply to accuracy as can be seen in the plot because once a sample is misclassified along its true path in the tree it has no recovery from misclassification by this measure. Hence, it might not always be true that smaller trees are closer to flat multi-class than deeper ones.

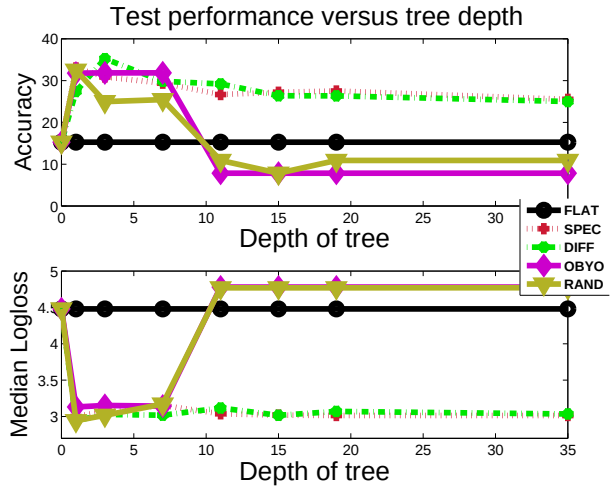


(a) Performance against tree depth.

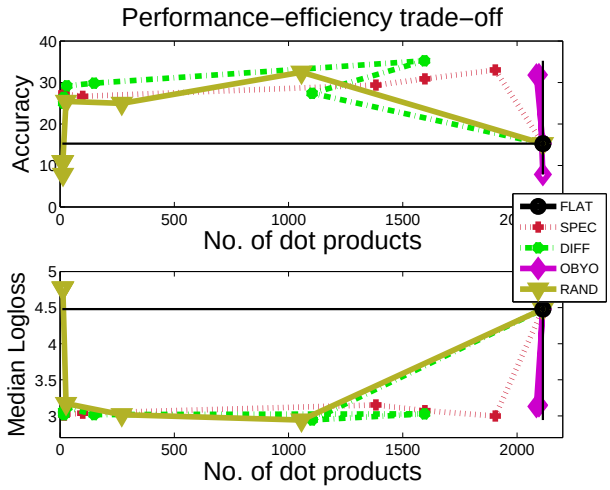


(b) Performance against number of dot-products required at test.

Figure 4.6: Performance variation with depth and performance efficiency trade-off on Vehicle dataset.



(a) Performance against tree depth.



(b) Performance against number of dot-products required at test.

Figure 4.7: Performance variation with depth and performance efficiency trade-off on subset of AP-news dataset.

Contrastingly, plots from Figure 4.6 showing results on the Vehicle dataset has a different trend. Performance of all top-down methods degrades including `SPEC` that performed very well on the CIFAR dataset. Accuracy of bottom-up approaches improves over that of the flat classifier. Logloss values of all methods except for that of `SPEC` are slightly better than that of the flat classifier.

Finally, we consider a bigram language model dataset (AP-news) where we use a CCA-based dimensionality reduction (Dhillon et al., 2011) to reduce the input dimension to 100. We then apply different tree construction methods to the task of predicting a word given the feature representation of the preceding word, a multi-category problem with 2114 classes. These results are reported in Figure 4.7. In this case we use median of log-loss as opposed to average since there were few words that had a very high log-probability leading to a considerably biased depiction of the methods. And also the backward jump in diffrac on second plot. On this dataset, we obtain both improved prediction performance, *i.e.,* improved accuracy and median log-loss and improved prediction efficiency. In this case the bottom-up methods degrade significantly after about depth 7 while the top-down methods continue to perform very well.

Overall, we can conclude from these results that the tree-based methods can improve the performance in some applications when the hierarchy suits the data distribution, but in general the performances are close to optimal at a much smaller computational cost. However, there are two major concerns that remain to be addressed.

1. A fundamental question is to understand the expressive power of tree representation of classification or regression or rather, what distributions can be approximated by tree representations?

2. The best performing tree-based methods varied from one dataset to another. It is not straightforward to determine which method to use for a given dataset for which it is necessary to understand what tree structures are generated by each of these methods?

The above two questions remain to be addressed to make this framework useful in practice. Given a dataset, one can then determine if a tree representation is appropriate and choose a suitable method to build the hierarchy.

An additional concern in dealing with large structured feature spaces like the long and sparse contextual features of language modeling is suitability of regularization. Our regularization scheme must reflect the effect of word groupings on the feature space to maintain predictive performance. This needs to be studied to integrate the language model from the previous chapter with the hierarchical framework presented here.

4.5 Conclusion

We presented a simple framework to learn hierarchical structure within the classes from their samples. This framework involves using a square loss function to rewrite the inter-class residuals in a matrix and using this matrix to group classes appropriately. This could be used with numerous strategies to organize the classes into a hierarchical structure. We follow two kinds of approaches; top-down approach of building the tree from root to leaves and bottom-up approach where classes put in leaves are combined to make meta-classes. Each tree built from the data is an approximation of the conditional-likelihood $P(y \mid x)$ resulting in reduced-classifier complexity for a variety of tasks. The top-down approach involves finding an appropriate partition of the classes repeatedly until the required depth is reached. This reduces to a max-cut problem that is NP-hard of which we consider two different relaxations. In the bottom-up approach, it involves grouping classes agglomeratively to generate the hierarchy. The two approaches generate different kinds of tree structures

that have different properties. We study their performance empirically on different kinds of data.

Chapter 5

Concluding remarks

We conclude the work with a summary and some remarks on certain shortcomings that need to be addressed. We also point to directions worthy of investigation to design more useful models building upon the material presented here to gain a clearer understanding of the methods.

Conclusion and discussion. In this work we address two crucial aspects pertaining to the use of discriminative language models. Maximum entropy and perceptron based discriminative models have only used unstructured penalties in their objective functions. They were agnostic to the structure within the model that led to overfitting at longer context lengths affecting generalization performance. This restricted these models to short context lengths. Our first contribution studies the usefulness of structured penalties to correct for varying levels of sparsity in different contextual features. We observe that overfitting happens due to structural differences in sampling for various features, the solution of which lies in utilizing the same structure to vary the amount of regularization appropriately.

We achieve this using a log-linear model built on suffix tries and using the tree structure to design a suitable penalty. We study performance and complexity with different tree-structured penalties. We further show that in specific cases one could use the suffix-tree structure instead of the suffix-trie structure. This work further adds to the evidence drawn from other fields that structured feature spaces benefit from structure incorporating penalties.

Our second contribution proposes different methods to build taxonomy of classes using a square-loss to measure class proximity in feature space. Numerous algorithms leading to both top-down and bottom-up procedures for growing the tree were studied. Empirical results from these experiments show that building an appropriate tree is important to minimize any loss of performance by switching from flat to tree based classification. Furthermore, in some cases there could be improvements of performance from a well built tree that matches the actual distribution of classes in the feature space. The experiments in Chapter 4 indicate that a top-down approach to build a taxonomy is more suitable for words than bottom-up approaches. The applicability of hierarchical organization of words goes beyond computational gains to tackle problems like out-of-vocabulary words by using other words from the most suitable cluster. Some additional work is required before using the taxonomy algorithms proposed here to build a hierarchical language model. It must

allow for assigning a single word into multiple clusters that was found to be important for deeper trees (Mnih and Hinton, 2008).

A hierarchical language model with classifiers based on suffix-trees at each node that can learn from large data is of huge interest in language processing. As described in Section 1.4, structured penalization in such models can be helpful to discriminatively model task specific requirements (like constraints for good readability) that are hard to handle with other techniques. Beyond applications in language, this model is of interest to various areas handling discrete sequences like molecular biology. Protein sequence alignment, for example, is one problem where large contexts have been found necessary for good performance (see, e.g., Edgar and Sjolander, 2003; Koonin and Galperin, 2003). These algorithms could also be extended to such problems to improve performance by exploiting the structure within their feature spaces. However, there are issues that need to be studied in detail for a more complete understanding that is necessary to tackle some known shortcomings. We describe these with possible directions worthy of exploration for further study.

Further investigation of language modeling. The strength of a model is largely dependent on the amount of data it can crunch, i.e., assuming it is suitably biased with a term tending to zero with growing amount of data fed into the model. One could imagine building an online model for language where nodes are added into the suffix tree with appropriately chosen initial weights as they are observed. Upon observing a new sample, a node is added if necessary and an iteration of the optimization is performed. A tree-structured regularization is most appropriate in such situations. Since adding a new node is equivalent to setting a weight corresponding to a subtree to non-zero that earlier had a zero weight because corresponding samples were observed. Such a model can be used to learn as and when data arrives and can accommodate as much data as we can feed without having to worry about overfitting.

A model with such statistical strength could serve as a backbone to numerous applications and could be used for domain adaptation and transfer learning. Domain adaptation is the problem where the target distribution is known to be different from that of the training observations and transfer learning is where different related tasks are learned from the same set of observations. Under-resourced domains would significantly benefit by borrowing from data available for other domains. The adaptation could be achieved by learning a parametric layer on top of the above model to suit domain specific distributions (like the two-stage model of Goldwater et al. (2011)). Similarly, various tasks can be modeled by parameters specific to the tasks that can be learned jointly to exploit task relatedness. This borrows the statistical strength from the large amounts of text while retaining the flexibility to fine tune it to specific domains and tasks.

An online statistical model that can be customized to address specific tasks is of particular interest to fields like language processing where no amount of data could be considered a sufficiently enough sampling of the feature space. Further, in applications like language generation where annotations are harder to gather borrowing strength from available data could give a significant boost. Consider for example, generating descriptive summaries of medical images or, short explanations to weather plots. We could have easy access to bare images and plots but we cannot expect sufficient annotation to guide a learning algorithm to translate them into meaningful sentences.

Further study of taxonomy learning. In Chapter 4 we proposed different algorithms to hierarchically organize classes. There are two important features these algorithms lack that are desirable for various applications;

1. Some classes might appear at multiple leaves in the tree. For language modeling, these would be polysemous words that have different meanings, and each leaf it appears at would be one concept indicated by that word. The word `bank` is one such example that could either mean a river bank or a financial bank and this problem has been studied extensively in the literature as word sense disambiguation and discovery (see, e.g., Linden, 2005; Navigli, 2009). This replication of classes at multiple leaves was reported to be beneficial by Mnih and Hinton (2008) and was used by Deng et al. (2011) for images with multiple objects but with pre-defined number of clusters at each level of the hierarchy. One could accommodate this within the clustering framework by allowing a class to belong to multiple clusters that the presented framework does not address. Also, it would be interesting to study how one could incorporate information from existing word sense discovery algorithms to improve their hierarchical organization.

2. The space of possible tree structures for a given set of categories is combinatorially large. In some cases, an ontology available from a related task could be used in choosing a suitable tree structure from this large set of choices. For example, Rabani et al. (2008) and Babbar et al. (2013) modify existing taxonomies to make hierarchies for specific tasks. A framework that can exploit side information like existing ontologies is a desirable characteristic that the approaches proposed in Chapter 4 lack.

Additional study is required to address the above issues for more suitable tree structures. Apart from improving proposed algorithms, their relation to other methods developed in the literature also needs to be understood. It would be useful to study how various methods behave in different scenarios. Empirical results suggest that different algorithms tend to perform better with different kinds of class distributions. However, it is unclear what algorithms are suitable for which kinds of distributions. This is necessary to know which algorithm to use given the prior class distribution. The other way round, if we could measure the fitness of a tree to data, it could throw light on what the class distribution of data looks like.

The other issue is that of how many samples do we need to reliably recover the underlying hierarchy, in what cases can it be recovered exactly and how well the corresponding hierarchical classifier performs in comparison to flat classifiers. Some of these questions have been addressed in phylogenetics where certain sample bounds and complexity results on the recovery of hierarchy were derived (Roch, 2007). In discrete choice models distributions that can be recovered under specific assumptions on noise in samples were studied (Ben-Akiva and Lerman, 1985). Recently, Babbar et al. (2013) derived a bound on the generalization error of hierarchical classifiers that reveals a trade-off between hierarchical and flat classifiers. A more complete study could explain the relation between classes of trees and the various distributions they generate to develop algorithms that return taxonomies with theoretical guarantees.

Appendix A

Smoothing and maximum entropy models

A.1 Smoothing models

We detail various methods briefed in Section 1.2.1 relating smoothing techniques for m-gram models. The two aspects of smoothing models relate to redistribution of probability mass to unseen sequences and mixing models of multiple orders using back-off or interpolation. We recall the maximum likelihood estimator as,

$$\mathbf{P}_{ML}(y|x_m) = \frac{c(x_m y)}{c(x_m)}. \tag{A.1}$$

A.1.1 Redistribute mass to handle unseen sequences.

Given that $\mathcal{C}$ only sparsely samples the space of all sequences, it is possible that with more words and larger vocabulary, sequence s will have a smaller probability of being seen at training phase. This motivates setting aside a part of the mass allocated to the sequences in $\mathcal{C}$ for unseen contexts. We briefly describe two methods that tackle this; the Laplace smoothing that uses additive values and the Good-Turing smoothing the uses multiplicative factors to achieve this behavior.

1. *Laplace smoothing.* Laplace smoothing, also called additive smoothing is a generic smoothing technique for models of categorical data (Lidstone, 1920; Jefreys, 1948). It is also called a shrinkage estimator because it brings the distribution closer to uniform distribution. It works by adding some fake samples to original frequencies from data $\mathcal{C}$. This is equivalent to incrementing values of both the numerator and the denominator of $\mathbf{P}_{ML}$ resulting in an estimate of the form:

$$\mathbf{P}_{LP}(y|x_m) = \frac{c(x_m y) + d}{c(x_m) + |V|d}, \tag{A.2}$$

   where $0 < d \leq 1$ is a parameter that is tuned using held-out data. The case of $d = 1$ is called add-one smoothing. Gale and Church (1994, 1990) have argued against this choice of smoothing asserting that is performs rather poorly in general.

2. *Good-Turing.* The method proposed by Good (1953, 2000) uses count of counts (C_k, the number of distinct sequences with frequency k) to make an estimate of the possible amount of unseen data and decide the correction factor. It reallocates the mass of sequences in the C_{k+1} list that occur $(k+1)$ times to sequences with frequency k (the list of C_k sequences):

$$\mathbf{P}_{GT}\left(\ y|x_m,\ c(x_m y)=k\ \right) = \frac{c_k^*}{C}, \tag{A.3}$$

$$\text{where } c_k^* = (k+1)\frac{\mathbf{E}\left[C_{k+1}\right]}{\mathbf{E}\left[C_k\right]} \text{ and } C = \sum_{k=1}^{m} c_k^* \mathbf{E}\left[C_k\right], \tag{A.4}$$

where $\mathbf{E}\left[C_k\right]$ is the expectation of the quantity C_k. The simplest is to use the maximum likelihood estimate of frequency estimates from $\mathcal{C}$, $\mathbf{E}\left[C_k\right] = C_k(\mathcal{C})$. This simply reassigns the probability of unigrams to unseen contexts avoiding the case zero probabilities. However, a caveat of this method is that the values $C_k(\mathcal{C})$ could be zeros, particularly for larger values of k. This requires some modification to appropriately estimate the expectation, the simplest being to use the unmodified counts.

A.1.2 Avoid overfitting using lower order distributions.

Given that contexts of a given length m are not found the most reasonable way to make an estimate is to look for orders lower than m. There are two ways this idea is introduced into the literature, one of which simply reduces order by one at a time until the sequence is found or it backs off until unigram and this method is called the *Katz back-off smoothing*. The other method extends this to a case where probabilities of higher order distributions are interpolated with lower order to make the final estimate. This method is called *Jelenick-Mercer interpolated models*. Jelinek-Mercer has then been improved to better performing methods by appropriately modifying the lower order distributions in the interpolation, namely, Witten-Bell, Absolute Discounting and Kneser-Ney that we describe below.

1. *Katz back-off smoothing.* Katz (1987) proposed a variant of the Good-Turing estimator that uses back-off distribution to redistribute the subtracted mass among unobserved sequences using the following rule:

$$\mathbf{P}_{KZ}(y|x_m) = \begin{cases} \mathbf{P}(y|x_m) & c(x_m) = k > \tau, \\ \eta(x_m)\mathbf{P}_{KZ}(y|x_{m-1}) & \text{otherwise}, \end{cases}$$

where $\mathbf{P}(y|x_m)$ is any estimator. Katz (1987) choose Good-Turing estimator described above for $\mathbf{P}(y|x_m)$. The parameter τ is used as a threshold for backing off to lower distribution is found not be critical for its performance (Chen and Goodman, 1998). The function $\eta(x_m)$ is chosen so that the resulting distribution is a valid probability and sums to one,

$$\eta(x_m) = \underbrace{\frac{\beta(x_m)}{\sum_{\{z|c(x_m z)\leq\tau\}}\mathbf{P}(y|x_{m-1})}}_{\text{redistributed as per lower-order dist.}}, \quad \beta(x_m) = \underbrace{1 - \sum_{\{z|c(x_m z)>\tau\}}\mathbf{P}(y|x_m)}_{\text{subtracted mass}}$$

2. *Jelinek-Mercer interpolated smoothing.* Jelinek and Mercer (1980) proposed a method that interpolates higher order prediction probabilities with that of the lower orders. While avoiding the zero probabilities for unseen sequences, it additionally offers a flexibility in controlling bias by parameters $0 < \gamma(x_m) < 1$ that determines the mixing proportions. The simplest version linearly interpolates the ML estimates of the highest order with its immediate predecessor following:

$$\mathbf{P}_{JM}(y|x_m) = \gamma(x_m)\mathbf{P}_{ML}(y|x_m) + (1 - \gamma(x_m))\mathbf{P}_{ML}(y|x_{m-1}).$$

An improved version uses a recursive formulation that interpolates probabilities from highest order down to unigram:

$$\mathbf{P}_{JM}(y|x_m) = \gamma(x_m)\mathbf{P}_{ML}(y|x_m) + (1 - \gamma(x_m))\mathbf{P}_{JM}(y|x_{m-1}).$$

In both the above versions, a uniform distribution is used for unigrams:

$$\mathbf{P}(y) = \frac{1}{|V|}.$$

Parameters $\gamma(x_m)$ are learned by optimizing predictive performance on validation data. If a single validation dataset is used to estimate the parameters it is called held-out interpolation. If distinct subsets of the training and validation are drawn from the corpus multiple times with replacement to make different estimates of the parameters that are averaged, it is called deleted interpolation. But since there are too many γ values to tune, some of them grouped together into a single bucket and constrained to take the same value.

3. *Witten-Bell smoothing.* The smoothing model of Witten-Bell was published by (Bell et al., 1990; Witten and Bell, 1991) for text compression and is a special case of Jelinek-Mercer interpolation where mixing proportion is chosen cleverly. The weight of the lower order distribution $(1 - \gamma(x_{m-1}))$ indicates the probability of a word following the sequence x_{m-1} and is chosen to be proportional to the number of unique words following x_{m-1}. We define the number of distinct words following the context as, x_{m-1} as

$$C_{1+}(x_m\bullet) = |\{z \mid c(x_m z) \geq 1\}|.$$

Note that the $\bullet$ above maps to unit length sequences $|z| = 1$. The weighting scheme for Witten-Bell takes the following form:

$$1 - \gamma(x_{m-1}) = \frac{C_{1+}(x_m\bullet)}{C_{1+}(x_m\bullet) + c(x_{m-1})}.$$

This weighting scheme leads to Witten-Bell probability distribution of the form:

$$\mathbf{P}_{WB}(y|x_m) = \gamma(x_m)\mathbf{P}_{ML}(y|x_m) + (1 - \gamma(x_m))\mathbf{P}_{WB}(y|x_m).$$

4. *Absolute discounting smoothing.* The method of Absolute Discounting by Ney et al. (1994) is another variant of the Jelinek-Mercer interpolation where a fixed amount of mass (controlled by $0 < \delta < 1$) is subtracted from the highest order term instead of the multiplicative factor γ. The interpolation weight for the lower order distribution,

$(1-\gamma)$ is solved for by ensuring that the resulting values sum to one so that its a valid probability. The formula for the distribution is,

$$\mathbf{P}_{AD}(y|x_m) = \frac{\max\{c(x_m) - \delta,\ 0\}}{c(x_{m-1})} + (1 - \gamma(x_m))\mathbf{P}_{AD}(y|x_{m-1}).$$

Discount parameter δ is tuned using held-out data. Appropriate value of $1-\gamma$ that comes from the probability constraint is,

$$1 - \gamma(x_m) = \frac{\delta C_{1+}(x_m\bullet)}{c(x_m)},$$

where $C_{1+}(x_m\bullet)$ is count of distinct sequences following x_m as defined above.

5. *Kneser-Ney smoothing.* The method proposed by Kneser and Ney (1995) extends Absolute Discounting and is the best performing smoothing technique. It improves the back-off weighting of Absolute Discounting in a very intuitive and effective manner. Lower order distributions, which are significant only when higher order contributions are small, are optimized to perform very well for these cases. This is achieved by modifying the lower order distribution $\mathbf{P}_{KN}(x_{m-1})$ that scales proportional to the number of single word contexts in which it occurs. Similar to the unique succeeding contexts above, we define the number of distinct words preceding a context x_m as,

$$C_{1+}(\bullet x_m) = |\,\{z \mid c(z\ x_m) \geq 1\}\,|$$

and the unique pair of words that sandwich the sequence x_{m-1} in the middle,

$$C_{1+}(\bullet x_{m-1}\bullet) = |\,\{(z_1, z_2) \mid c(z_1\ x_{m-1}\ z_2) \geq 1\}\,|.$$

Note that $|z_i| = 1$ and $C_{1+}(\bullet\bullet)$ is the bigram count in $\mathcal{C}$. The formulae for Kneser-Ney smoothing can now be put down as follows:

$$\mathbf{P}_{KN}(y|x_m) = \begin{cases} \overbrace{\frac{\max\{c(x_m y) - \delta,\ 0\}}{c(x_{m-1})}}^{\text{abs. disc.}} + \overbrace{\frac{\delta C_{1+}(x_m\bullet)}{c(x_m)}\mathbf{P}_{KN}(y|x_{m-1})}^{\text{back-off term}} & \text{largest } m \text{ s.t. } c(x_m y) > 0, \\ \underbrace{\frac{\max\{C_{1+}(\bullet x_{m-1} y) - \delta, 0\}}{C_{1+}(\bullet x_{m-1}\bullet)}}_{\text{modified lower order dist.}} + \underbrace{\frac{\delta C_{1+}(x_{m-1}\bullet)}{C_{1+}(\bullet x_{m-1}\bullet)}\mathbf{P}_{KN}(y|x_{m-1})}_{\text{interpolation term}} & \text{otherwise.} \end{cases}$$

This model is more specifically called the interpolated Kneser-Ney and there is a back-off version of Kneser-Ney that skips the interpolation term (second term of the second part of the equation above) by setting the corresponding δ to zero. The parameter δ is either chosen by held-out data or set using some heuristics. There is a variant of this where δ is set to different values depending on the length of the sequence $|x_m y|$ it is applied on which is called the modified interpolated Kneser-Ney.

It is important to note that while models have been developed separately in the literature as back-off and interpolated, a model of one kind can be rewritten appropriately in the other format (Chen and Goodman, 1998).

A.2 Iterative scaling methods

We derive the surrogate functions for the error defined in (1.10) and the resulting update equations for different iterative scaling methods described in Section 1.2.2. The update equations involved in minimizing $\mathcal{L}$ come from finding a possible $\Delta\omega$ that minimizes the error in (1.10) which, we recall, is:

$$err = \mathcal{L}(\omega + \Delta\omega) - \mathcal{L}(\omega).$$

It measures the difference in $\mathcal{L}$ at two distinct points in the domain that are separated by $\Delta\omega$. Arriving at appropriate update equations comes from the decoupling of ω_i. This is achieved by using an appropriate surrogate function that upper bounds err. The steps below briefly derive iterative scaling algorithm for the minimization of err.

Upon expansion err takes the following form:

$$\begin{aligned} err &= \sum_{\mathcal{C}}\sum_{i}\Delta\omega_i\mathbb{I}(x_m y,i) + \sum_{\mathcal{C}}\left\{\ln\sum_{v\in V}\left[\frac{e^{\sum_i \omega_i\mathbb{I}(x_m v,i)}}{\sum_{z\in V}e^{\sum_i \omega_i\mathbb{I}(x_m z,i)}}e^{\sum_i \Delta\omega_i\mathbb{I}(x_m v,i)}\right]\right\} \\ &= \sum_{\mathcal{C}}\sum_{i}\Delta\omega_i\mathbb{I}(x_m y,i) + \sum_{\mathcal{C}}\left\{\ln\sum_{v\in V}\left[\mathbf{P}_{ME}(v|x_m)e^{\sum_i \Delta\omega_i\mathbb{I}(x_m v,i)}\right]\right\} \end{aligned} \tag{A.5}$$

The inequality of $\ln(x) < x - 1$ is used to bound err from above:

$$err \leq \sum_{\mathcal{C},i}\Delta\omega_i\mathbb{I}(x_m y,i) + \sum_{\mathcal{C}}\left\{\sum_{v\in V}\left[\mathbf{P}_{ME}(v|x_m)\underbrace{e^{\sum_i \Delta\omega_i\mathbb{I}(x_m v,i)}}_{T_e}\right] - 1\right\} \tag{A.6}$$

Three different iterative scaling algorithms were developed in the literature; the generalized iterative scaling (GIS) proposed by Darroch and Ratcliff (1972), the improved iterative scaling (IIS) due to Della Pietra and Della Pietra (1993) and the sequential conditional generalized iterative scaling (SCIS) by Goodman (2002). They follow from the three different surrogate functions that we will derive below. We will make use of the following definitions:

$$\begin{aligned} \mathbb{I}^{\#}(x_m v) &= \sum_i \mathbb{I}(x_m v,i), \\ \mathbb{I}^{\#} &= \max_{\mathcal{C}}\ \mathbb{I}^{\#}(x_m v) \\ \mathbb{I}^{M}(i) &= \max_{\mathcal{C}}\mathbb{I}(x_m v,i) \end{aligned}$$

A.2.1 Generalized Iterative Scaling.

We will derive an upper bound of T_e that splits across the features i using Jensen's inequality. An additional $(d+1)^{th}$ feature $\left(\mathbb{I}^{\#} - \mathbb{I}^{\#}(x_m v)\right)$ with a weight value $\Delta\omega_{d+1} = 0$. This makes the inner fractions involving indicator functions a proper probability mass where

Jensen's inequality is applied to rewrite T_e as a sum over i.

$$\begin{aligned} T_e &= \exp\left\{\frac{\sum_{i=1}^{d+1}\Delta\omega_i\mathbb{I}(x_m v,i)\mathbb{I}^{\#}}{\mathbb{I}^{\#}}\right\} \\ &\leq \sum_{i=1}^{d+1}\frac{\mathbb{I}(x_m v,i)}{\mathbb{I}^{\#}}\exp\left\{\Delta\omega_i\mathbb{I}^{\#}\right\} \\ &\leq \frac{\sum_{i=1}^{d}\mathbb{I}(x_m v,i)\exp\left\{\Delta\omega_i\mathbb{I}^{\#}\right\}+\mathbb{I}^{\#}-\sum_{i=1}^{d}\mathbb{I}^{\#}(x_m v,i)}{\mathbb{I}^{\#}} \\ &\leq \frac{\sum_{i=1}^{d}\mathbb{I}(x_m v,i)\left\{e^{\Delta\omega_i\mathbb{I}^{\#}}-1\right\}}{\mathbb{I}^{\#}}+1. \end{aligned} \tag{A.7}$$

Substituting the upper bound from (A.7) into (A.6) we get the following as the surrogate for GIS,

$$\begin{aligned} err &\leq \sum_i \underbrace{\sum_{\mathcal{C}}\Delta\omega_i\mathbb{I}(x_m y,i)}_{P(\mathcal{C},i)}+\sum_i \underbrace{\frac{\left(e^{\Delta\omega_i\mathbb{I}^{\#}}-1\right)}{\mathbb{I}^{\#}}\sum_{\mathcal{C}}\sum_{v\in V}[\mathbf{P}_{ME}(v|x_m)\mathbb{I}(x_m v,i)]}_{Q(\mathcal{C},i)} \\ err_i &\leq P(\mathcal{C},i)+Q(\mathcal{C},i)=sur_i^{\text{GIS}}. \end{aligned} \tag{A.8}$$

A.2.2 Improved Iterative Scaling.

This method straight away applies Jensen's inequality on the T_e by using $\mathbb{I}^{\#}(x_m v)$:

$$\begin{aligned} T_e &= \exp\left\{\frac{\sum_i\Delta\omega_i\mathbb{I}(x_m v,i)\mathbb{I}^{\#}(x_m v)}{\mathbb{I}^{\#}(x_m v)}\right\} \\ &\leq \sum_i\frac{\mathbb{I}(x_m v,i)}{\mathbb{I}^{\#}(x_m v)}e^{\Delta\omega_i\mathbb{I}^{\#}(x_m v)}. \end{aligned} \tag{A.9}$$

Putting (A.9) into (A.6), we get the surrogate for IIS,

$$\begin{aligned} err &\leq \sum_i \underbrace{\sum_{\mathcal{C}}\Delta\omega_i\mathbb{I}(x_m y,i)}_{P(\mathcal{C},i)}+\sum_i \underbrace{\sum_{\mathcal{C}}\sum_{v\in V}\left[\mathbf{P}_{ME}(v|x_m)\mathbb{I}(x_m v,i)\frac{\left(e^{\Delta\omega_i\mathbb{I}^{\#}(x_m v)}-1\right)}{\mathbb{I}^{\#}(x_m v)}\right]}_{Q(\mathcal{C},i)} \\ err_i &\leq P(\mathcal{C},i)+Q(\mathcal{C},i)=sur_i^{\text{IIS}}. \end{aligned} \tag{A.10}$$

A.2.3 Sequential Conditional Generalized Iterative Scaling.

This method follows a derivation similar to that of GIS above except that $\mathbb{I}^{\#}$ is replaced by $\mathbb{I}^{M}$ gives us the following surrogate:

$$\begin{aligned} err &\leq \underbrace{\sum_i \sum_{\mathcal{C}} \Delta\omega_i \mathbb{I}(x_m y, i)}_{P(\mathcal{C},i)} + \underbrace{\sum_i \frac{\left(e^{\Delta\omega_i \mathbb{I}^M(i)} - 1\right)}{\mathbb{I}^M(i)} \sum_{\mathcal{C}} \sum_{v \in V} [\mathbf{P}_{ME}(v|x_m)\mathbb{I}(x_m v, i)]}_{Q(\mathcal{C},i)} \\ err_i &\leq P(\mathcal{C}, i) + Q(\mathcal{C}, i) = sur_i^{\text{SCIS}}. \end{aligned} \tag{A.11}$$

While the above two methods, GIS and IIS, update all parameters at every iteration, SCIS only updates one parameter at each iteration. It sequentially iterates through the list of parameters until termination. It was shown by Goodman (2002) that this modification could be used to gain about an order of magnitude in computational time over GIS.

The derivatives of surrogate functions for *err* in (A.8), (A.10) and (A.11) are set to zero to get the update equations for each of these methods. These equations are summarized in Table 1.1. Note that the update for IIS unlike the other two is not straightforward with $\Delta\omega$ being implicit and requires solving the equation iteratively to fetch the $\Delta\omega$.

An alternative to above methods is to use coordinate descent to minimize (A.5) directly. Coordinate descent updates variables sequentially as in SCIS but the update step is similar to that of IIS where $\Delta\omega$ is implicit and must be solved for using iterative techniques. The three iterative scaling algorithms above and coordinate descent are all known to converge linearly (Huang et al., 2010).

Appendix B

Synthetic Data Generation

A specified number of clusters are distributed in different orthants. Each cluster has a specified number of data points.

```
dim: dimensionality of features
pntspc: points per class
nclust: number of clusters
disperse: distance of mean of cluster from origin
variance: variance within cluster
```

An orthant is chosen by a binary encoding of decimal number i varying from zero to number of clusters. Disperse is the amount by which is moves in the orthant. It is further moved by a small random vector (`vec` below) in this orthant. Uniformly distributed points are generated with this as the mean. Such clusters are piled to make the data matrix.

```
orth_use = min(nclust,2^dim);  // no. of orthants used
for i = 0 : length(  orth_use )-1
   bcode = 2.*double(bsxfun(@eq, dec2bin(i, dim), '0')); //binary code
   bcode(~bcode) = -1; // replace zeros by -1s

   // no. clusters in a given orthant
   n_orth_clust = floor(nclust ./ length(orth_use) );
   for j = 1 : n_orth_clust
      vec = randn(1,dim);  // generate a random vector
      mu = disperse * bcode + vec; // mean of the cluster

      x = rand( pntspc , dim ) * variance;  //uniformly distributed points
      x = bsxfun(@plus, x, mu); // x + mu
   end
end
```

Appendix C

Effect of parameters

We empirically study the significance of the δ parameter used in estimating the residuals and the effect of approximating residuals in top-down approaches.

Ridge in square loss. There is one free parameter in the measure of proximity of classes, namely the ridge parameter δ. This effects the entries in the residual matrix which is used to determine the shape of the tree. The ridge value showed no significant variations on the final performances of the tree classifiers. They are compared on the synthetic datasets in Figure C.1. The tree structure remained considerably unchanged for a large range of δ which we can safely fix to a value.

Approximatrion of residuals at deeper nodes. Top-down methods, unlike bottom-up, cannot exactly update residuals after a split operation using those before the split. The effect of such approximations is shown in Figure C.2. The difference shows up in the well-separated classes and this was observed to become critical for top-down methods as the number of classes grow.

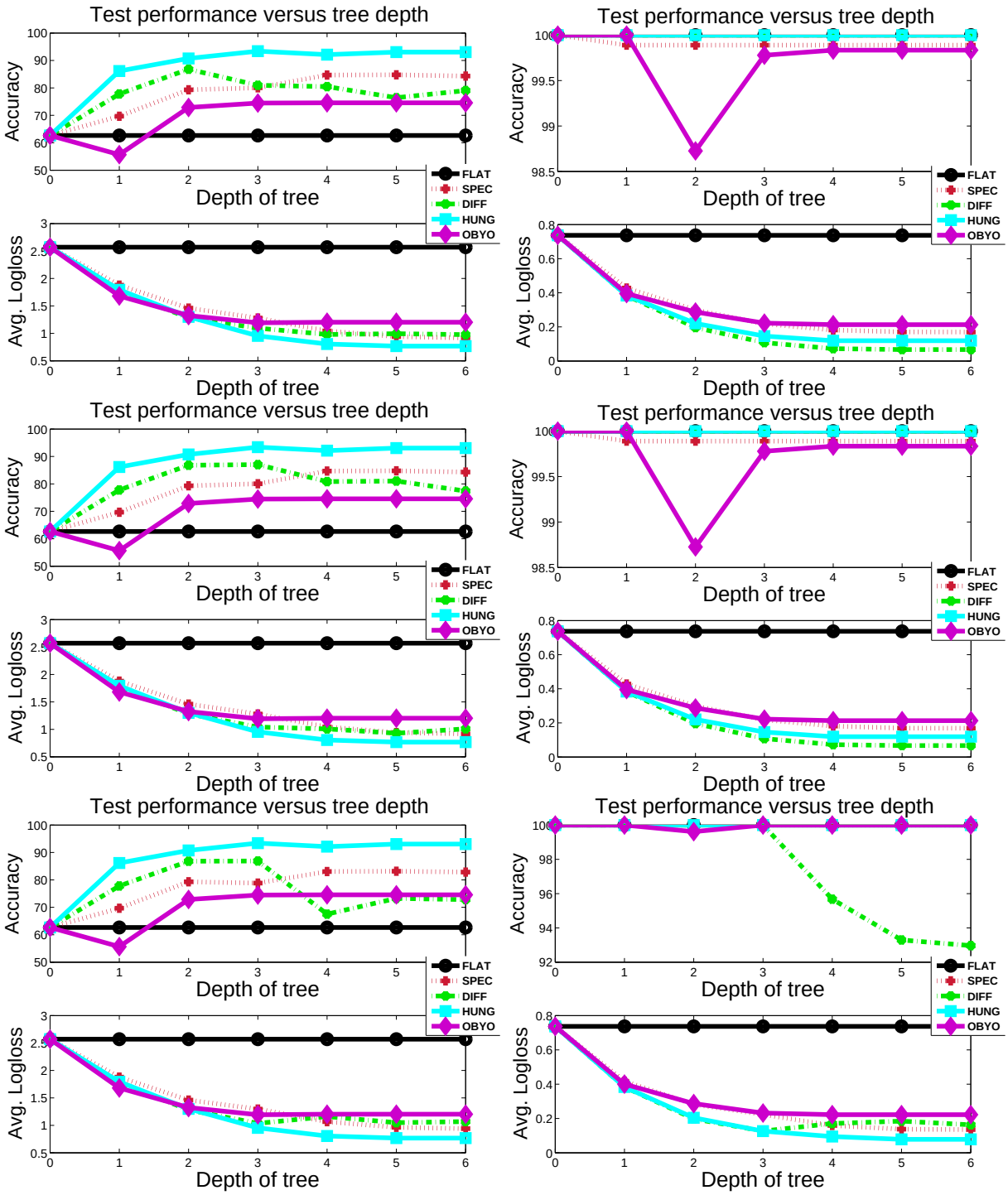


Figure C.1: Performance variation with tree regularization parameter δ in equation (4.8). Three plots on the left are on PO50 dataset and the ones on the right are on WS50. Values of δ vary as $0.001, 1$ and 1000 from top to bottom.

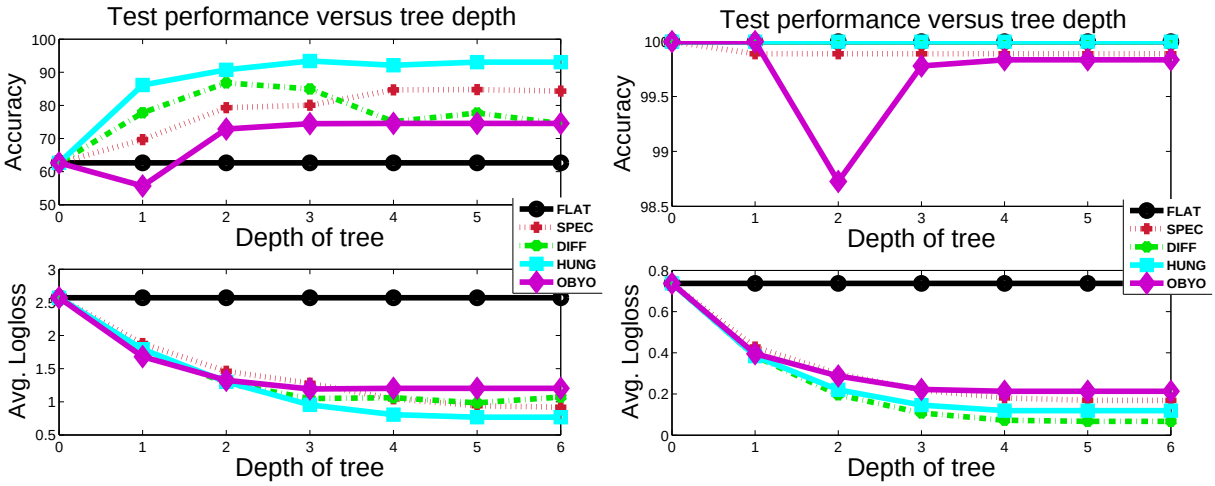


Figure C.2: Effect of residual matrix approximation on top-down methods. Results on dataset PO50 on the left and that on WS50 on the right.

Bibliography

A. Agarwal, A. Rakhlin, and P. Bartlett. Matrix regularization techniques for online multitask learning. *Technical Report EECS-2008-138, EECS Department, University of California, Berkeley*, 2008.

T. Alumae and M. Kurimo. Efficient estimation of maximum entropy language models with N-gram features: an SRILM extension. *Proceedings of the Annual Conference of the International Speech Communication Association*, 2010.

Y. Amit, M. Fink, N. Srebro, and S. Ullman. Uncovering shared structures in multiclass classification. *Proceedings of the International Conference on Machine Learning*, 2007.

G. Andrew and J. Gao. Scalable training of ℓ_1-regularized log-linear models. *Proceedings of the International Conference on Machine Learning*, 2007.

M. Anthony and P. L. Bartlett. Neural network learning: Theoretical foundations. *Cambridge University Press*, 1999.

A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. *Advances in Neural Information Processing Systems*, 2007.

A. Argyriou, T. Evgeniou, and M. Pontil. Convex multi-task feature learning. *Journal of Machine Learning*, 73(3):243–272, 2008.

R. Babbar, I. Partalas, É. Gaussier, and M.-R. Amini. On flat versus hierarchical classification in large-scale taxonomies. *Advances in Neural Information Processing Systems*, 2013.

F. Bach. Bolasso: model consistent lasso estimation through the bootstrap. *Proceedings of the International Conference on Machine Learning*, 2008a.

F. Bach. Consistency of the group lasso and multiple kernel learning. *Journal of Machine Learning Research*, 9:1179–1225, 2008b.

F. Bach and Z. Harchaoui. Diffrac: a discriminative and flexible framework for clustering. *Advances in Neural Information Processing Systems*, 2007.

F. Bach and M. Jordan. Predictive low-rank decomposition for kernel methods. *Proceedings of the International Conference on Machine Learning*, 2005.

F. Bach, J. Mairal, and J. Ponce. Convex sparse matrix factorizations. *Technical Report*, arXiv:0812.1869, 2008.

F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 4(1):1–106, 2012.

L. R. Bahl, F. Jelinek, and R. L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5 (2):179–190, 1983.

S. Banerjee and A. Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. *Proceedings of Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization*, 2005.

R. G. Baraniuk, V. Cevher, M. F. Duarte, and C. Hegde. Model-based compressive sensing. *Technical report*, arXiv:0808.3572, 2008.

E. Bart, M. Welling, and P. Perona. Unsupervised organization of image collections: Taxonomies and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 2302–2315, 2011.

P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe. Large margin classifiers: convex loss, low noise, and convergence rates. *Advances in Neural Information Processing Systems*, 2003.

A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal of Imaging Sciences*, 2009.

T.C. Bell, J.G. Clearly, and H. Witten. Text compression. *Prentice Hall.*, 1990.

M. Ben-Akiva and S. R. Lerman. *Discrete Choice Analysis: Theory and Application to Travel Demand*. The MIT Press, 1985.

L. Benaroya, F. Bimbot, and R. Gribonval. Audio source separation with a single sensor. *IEEE Transactions on Speech and Audio Processing*, 14(1):191–199, 2006.

S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. *Advances in Neural Information Processing Systems*, 2010.

Y. Bengio and J-S. Senecal. Quick training of probabilistic neural nets by importance sampling. *Proceedings of the Conference on Artificial Intelligence and Statistics.*, 2003.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

A. Berger, S. Della Pietra, and V. Della Pietra. A maximum entropy approach to natural language processing. *Proceedings of the Association for Computational Linguistics*, 1996.

D. P. Bertsekas. Nonlinear programming. *Athena Scientific*, 2nd edition., 2004.

C.M. Bishop. Pattern recognition and machine learning. *Springer-Verlag*, 2006.

Y-L. Boureau, F. Bach, Y. Lecun, and J. Ponce. Learning mid-level features for recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

S. Boyd, C. Cortes, M. Mohri, and A. Radovanovic. Accuracy at the top. *Advances in Neural Information Processing Systems*, 2012.

T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. Large language models in machine translation. *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007.

L. Breiman, J. Friedman, and Stone C. Olshen, R. Classification and regression trees. *Wadsworth International*, 1984.

P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. Della Pietra, and J. C. Lai. Class-based n-gram models of natural language. *Journal Computational Linguistics*, 18(4):467–479, 1992.

P. Bruckner. An $O(n)$ algorithm for quadratic knapsack problems. *Operations Research Letters*, 3:163–166, 1984.

E. J. Candes and M. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, pages 21–30, 2008.

A. Chambolle, R. A. DeVore, N. Y. Lee, and B. J. Lucier. Nonlinear wavelet image processing: Variational problems, compression, and noise removal through wavelet shrinkage. *IEEE Transactions on Image Processing*, 7:319–335, 1998.

S. F. Chen. Performance prediction for exponential language models. *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2009.

S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. *Proceedings of the Association for Computational Linguistics*, 1996.

S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. *Technical report, Harvard University.*, 1998.

S. F. Chen and R. Rosenfeld. A gaussian prior for smoothing maximum entropy models. *Technical report, Carniege Mellon University*, 1999.

S. F. Chen and R. Rosenfeld. A survey of smoothing techniques for maximum entropy models. *IEEE Transactions on Speech and Audio Processing*, pages 37–50, 2000.

S.F. Chen, D. Beeferman, and R. Rosenfeld. Evaluation metrics for language models. *DARPA Broadcast News Transcription and Understanding Workshop*, 1998a.

S.S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20:33–61, 1998b.

W. Chen, T. Liu, Y. Lan, Z. Ma, and H. Li. Ranking measures and loss functions in learning to rank. *Advances in Neural Information Processing Systems*, 2009.

C. Chesneau and M. Hebiri. Some theoretical results on the grouped variables lasso. *Mathematical Methods of Statistics*, 17(4):317–326, 2008.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and S. Clifford. Introduction to algorithms. *MIT Press and McGraw-Hill.*, 3rd edition, 2009.

F. Couzinie-Devy, J. Mairal, F. Bach, and J. Ponce. Dictionary learning for deblurring and digital zoom. *Technical report*, 2011.

K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning Journal*, 47(2-3):201–233, 2002.

J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *Annals of Mathematical Statistics*, 43:1470–1480, 1972.

A. d'Aspremont, F. Bach, and L. El Ghaoui. Optimal solutions for sparse principal component analysis. *Journal of Machine Learning Research*, 9:1269–1294, 2008.

I. Daubechies, M. Defrise, and C. D. Mol. Nonlinear wavelet image processing: Variational problems, compression, and noise removal through wavelet shrinkage. *Communications of Puce and Applied Mathematics*, 57:1413–1457, 2004.

S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

S. Della Pietra and V. Della Pietra. Statistical modeling by maximum entropy. *Unpublished Report*, 1993.

S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random variables. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 19(4):380–393, 1997.

J. Deng, W. Dong, R. Socher, L-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

J. Deng, S. Satheesh, A. C. Berg, and L. Fei-Fei. Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition. *Advances in Neural Information Processing Systems*, 2011.

L. Devroye, L. Gyorfi, and G. Lugosi. A probabilistic theory of pattern recognition. *Springer-Verlag*, 1996.

P. S. Dhillon, D. Foster, and L. Ungar. Multi-view learning of word embeddings via cca. *Advances in Neural Information Processing Systems*, 2011.

C. B. Do, C-S. Foo, and A. Y. Ng. Efficient multiple hyperparameter learning for log-linear models. *Advances in Neural Information Processing System*, 2007.

J. Duchi and Y. Singer. Efficient online and batch learning using forward-backward splitting. *Journal of Machine Learning Research*, 10:2873–2898, 2009.

R. O. Duda, P. E. Hart, and D. G. Stork. Pattern classification. *John Wiley*, 2001.

G. Dunn and B. Everitt. An introduction to mathematical taxonomy. *Dover Publications/Cambridge University Press*, 2004.

R. C. Edgar and K. Sjolander. Simultaneous sequence alignment and tree construction using hidden markov models. *Pacific Symposium on Biocomputing*, 8:180–191, 2003.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32(2):407–499, 2004.

J. Eisenstein, N. A. Smith, and E. P. Xing. Discovering sociolinguistic associations with structured sparsity. *Proceedings of the Association for Computational Linguistics*, 2011.

M. Elad and M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing*, 15(12):3736–3745, 2006.

A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth. Every picture tells a story: generating sentences for images. *Proceedings of the European Conference on Computer Vision.*, 2010.

M. Fazel, H. Hindi, and S. P. Boyd. A rank minimization heuristic with application to minimum order system approximation. *Proceedings of the American Control Conference*, 2001.

S. Fine and K. Scheinberg. Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

J. R. Finkel, A. Kleeman, and C. D. Manning. Efficient, feature-based, conditional random field parsing. *Proceedings of the Association for Computational Linguistics*, 2008.

A. Frank. On kuhn's hungarian method - a tribute from hungary. *Technical report, Egervary Research Group, Budapest, www.cs.elte.hu/egres.*, 2004.

J. Friedman, T. Hastie, H. Hofling, and R. Tibshirani. Pathwise coordinate optimization. *Annals of Applied Statistics*, 1(2):302–332, 2007.

W. J. Fu. Penalized regressions: the bridge vs. the lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416, 1998.

W. A. Gale and K. W. Church. Estimationproceduresfor language context: poor estimates are worse than none. *Proceedings of the Symposium on Computational Statistics*, 1990.

W. A. Gale and K. W. Church. What's wrong with adding one? *Proceedings of the Conference on Corpus-Based Research into Language*, 1994.

T. Gao and D. Koller. Discriminative learning of relaxed hierarchy for large-scale visual recognition. *Proceedings of the International Conference on Computer Vision*, 2011.

S. Ghosal. Dirichlet process, related priors and posterior asymptotics. *N. L. Hjort et al., editors, Bayesian Nonparametrics, Cambridge University Press*, pages 36–83, 2010.

R. Giegerich and S. Kurtz. From ukkonen to mccreight and weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 1997.

M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42 (6):1115–1145, 2005.

S. Goldwater, T. L. Griffiths, and Johnson. M. Interpolating between types and tokens by estimating power-law generators. *Advances in Neural Information Processing Systems*, 2006.

S. Goldwater, T. L. Griffiths, and M. Johnson. Producing power-law distributions and damping word frequencies with two-stage language models. *Journal of Machine Learning Research*, 12:2335–2382, 2011.

G. H. Golub and C. F. Van Loan. Matrix computations. *John Hopkins University Press*, 1996.

I. J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3 and 4):237–264, 1953.

I. J. Good. Turing's anticipation of empirical bayes in connection with the cryptanalysis of the naval enigma. *Journal of Statistical Computations and Simulations*, 66:101–111, 2000.

J. Goodman. Classes for fast maximum entropy training. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 2001.

J. Goodman. Sequential conditional generalized iterative scaling. *Proceedings of the Association for Computational Linguistics*, 2002.

J. Goodman. Exponential priors for maximum entropy models. *Proceedings of the Association for Computational Linguistics*, 2004.

A. Gramfort, M. Kowalski, and M. Hamalainen. Mixed-norm estimates for the m/eeg inverse problem using accelerated gradient methods. *Journal of Physics in Medicine and Biology*, 57(7):1937–1961, 2012.

G. Griffin and P. Perona. Learning and using taxonomies for fast visual categorization. *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

A. Gupta, P. Srinivasan, J. Shi, and L. Davis. Understanding videos, constructing plots: Learning a visually grounded storyline model from annotated videos. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2009.

Z. Harchaoui, M. Douze, M. Paulin, M. Dudik, and J. Malick. Large-scale classification with trace-norm regularization. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning: Data mining, inference, and prediction, second edition. *Springer-Verlag*, 2009.

J. Haupt and R. Nowak. Compressive sampling vs. conventional imaging. *Proceedings of the International Conference on Image Processing*, 2006.

L. He and L. Carin. Exploiting structure in wavelet-based bayesian compressive sensing. *IEEE Transaction on Signal Processing*, 57:3488–3497, 2009.

C. Hu, J. T. Kwok, and W. Pan. Accelerated gradient methods for stochastic optimization and online learning. *Advances in Neural Information Processing Systems*, 2009.

F-L. Huang, C-J. Hsieh, K-W. Chang, and C-J. Lin. Iterative scaling and coordinate descent methods for maximum entropy models. *Journal of Machine Learning Research*, 11:815–848, 2010.

J. Huang and T. Zhang. The benefit of group sparsity. *Annals of Statistics*, 38(4):1978–2004, 2010.

J. Huang, T. Zhang, and D. Metaxas. Learning with structured sparsity. *Proceedings of the International Conference on Machine Learning*, 2009.

J. Huang, T. Zhang, and D. Metaxas. Learning with structured sparsity. *Journal of Machine Learning Research*, 12:3371–3412, 2011.

M. J. Hunt. Figures of merit for assessing connected-word recognizers. *Journal of Speech Communication*, 9(4):329–336, 1990.

H. Jefreys. Theory of probability. *Clarendon Press*, second edition, 1948.

F. Jelinek and R. L. Mercer. Interpolated estimation of markov source parameters from sparse data. *Proceedings of the Workshop on Pattern Recognition in Practice.*, 1980.

R. Jenatton, J-Y. Audibert, and F. Bach. Structured variable selection with sparsity-inducing norms. *Journal of Machine Learning Research*, 12:2777–2824, 2011a.

R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal methods for hierarchical sparse coding. *Journal of Machine Learning Research*, 12:2297–2334, 2011b.

R. Jenatton, A. Gramfort, V. Michel, G. Obozinski, E. Eger, F. Bach, and B. Thirion. Multi-scale mining of fmri data with hierarchical structured sparsity. *SIAM Journal of Imaging Sciences*, 5(3):835–856, 2012a.

R. Jenatton, N. Le Roux, A. Bordes, and G. Obozinski. A latent factor model for highly multi-relational data. *Advances in Neural Information Processing Systems*, 2012b.

S. Ji, D. Dunson, and L. Carin. Multitask compressive sensing. *IEE Transactions on Signal Processing*, 57(1):92–106, 2009.

W. Jiang. Process consistency for adaboost. *Annals of Statistics*, 32(1):13–29, 2003.

A. Joulin, F. Bach, and J. Ponce. Efficient optimization for discriminative latent class models. *Advances in Neural Information Processing Systems*, 2010.

M. Journee, F. Bach, P-A. Absil, and R. Sepulchre. Low-rank optimization on the cone of positive semidefinite matrices. *SIAM Journal on Optimization*, 20(5):2327–2351, 2010.

D. Jurafsky and J. H. Martin. *Speech and Language Processing.* Pearson Prentice Hall, 2008.

S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing,*, 35(3):400–401, 1987.

J. Kazama and J. Tsujii. Evaluation and extension of maximum entropy models with inequality constraints. *Proceedings of the Conference on Empirical Methods for Natural Language Processing*, 2003.

C. R. Kennington, M. Kay, and A. Friedrich. Suffix trees as language models. *Language Resources and Evaluation Conference*, 2012.

S. Khudanpur. A method of maximum entropy estimation with relaxed constraints. *Proceedings of the John Hopkins University Language Modeling Workshop*, 1995.

S. Kim and E. P. Xing. Tree-guided group lasso for multi-task regression with structured sparsity. *Proceedings of the International Conference on Machine Learning*, 2010.

S. Kim and E. P. Xing. Tree-guided group lasso for multi-response regression with structured sparsity, with an application to eQTL mapping. *Annals of Applied Statistics*, 6(3): 1095–1117, 2012.

R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1995.

B. Kolaczkowski and J.W. Thornton. Performance of maximum parsimony and likelihood phylogenetics when evolution is heterogeneous. *Nature*, 431(7011):980–984, 2004.

V. Koltchinskii and M. Yuan. Sparse recovery in large ensembles of kernel machines on-line learning and bandits. *Proceedings of the Conference on Learning Theory*, 2008.

E. V. Koonin and M. Y. Galperin. Sequence - evolution - function, computational approaches in comparative genomics. *Kluwer Academic*, 2003.

G. Kulkarni, V. Premraj, S. Dhar, S. Li, Y. Choi, A. C. Berg, and T. L. Berg. Babytalk: Understanding and generating simple image descriptions. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2011.

K. Lange, D. R. Hunter, and I. Yang. Optimization transfer using surrogate objective functions. *Journal of Computation and Graphical Statistics*, 9(1):1–20, 2000.

R. Lau. Adaptive statistical language modeling. *MS thesis, Massachusetts Insitute of Technology*, 1994.

R. Lau, R. Rosenfeld, and S. Roukos. Adaptive language modeling using the maximum entropy principle. *Proceedings of the ARPA Workshop on Human Language Technology*, 1993.

H. Lee, A. Battle, R. Raina, and A. Ng. Efficient sparse coding algorithms. *Advances in Neural Information Processing Systems*, 2007.

M. Leordeanu, M. Hebert, and R. Sukthankar. Beyond local appearance: Category recognition from pairwise interactions of simple features. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

S. Li, G. Kulkarni, T. L. Berg, A. C. Berg, and Y. Choi. Composing simple image descriptions using web-scale n-grams. *Proceedings of the Conference on Computational Natural Language Learning*, 2011.

G. J. Lidstone. Note on the general case of the bayes-laplace formula for inductive or *a posteriori* probabilities. *Transactions of the Faculty of Actuaries*, 8:182–192, 1920.

C-Y. Lin. Rouge: a package for automatic evaluation of summaries. *Proceedings of the Workshop on Text Summarization Branches Out*, 2004.

K. Linden. Word sense discovery and disambiguation. *Department of General Linguistics, Faculty of Arts, University of Helsinki.*, 2005.

J Liu, L Yuan, and J. Ye. An efficient algorithm for a class of fused lasso problems. *Proceedings of the Conference on Knowledge Discovery and Data Mining*, 2010.

K. Lounici. Sup-norm convergence rate and sign concentration property of lasso and dantzig estimators. *Electronic Journal of Statistics*, 2:90–102, 2008.

K. Lounici, A. B. Tsybakov, M. Pontil, and S. A. van de Geer. Taking advantage of sparsity in multi-task. *Proceedings of Conference on Computational Learning Theory*, 2009.

G. Lugosi and N. Vayatis. On the bayes risk consistency of regularized boosting methods. *Annals of Statistics*, 1(30-55), 2003.

D. J. C. Mackay and L. C. B. Peto. A hierarchical dirichlet language model. *Natural Language Engineering*, 1994.

N. Maculan and J. R. G. Galdino de Paula. A linear-time median-finding algorithm for projecting a vector on the simplex of rn. *Operations Research Letters*, 8(4):219–222, 1989.

J. Mairal. Sparse coding for machine learning, image processing and computer vision. *PhD thesis, Ecole Normale Superieure de Cachan*, 2010.

J. Mairal, M. Leordeanu, F. Bach, M. Hebert, and J. Ponce. Discriminative sparse image models for class-specific edge detection and image interpretation. *Proceedings of the European Conference on Computer Vision (ECCV)*, 2008.

J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. *Proceedings of the International Conference on Machine Learning*, 2009a.

J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Non-local sparse models for image restoration. *Proceedings of the International Conference on Computer Vision*, 2009b.

J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Supervised dictionary learning. *Advances in Neural Information Processing Systems*, 2009c.

J. Mairal, F. Bach, and J. Ponce. Task-driven dictionary learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):791–804, 2012.

R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. *Proceedings of the Conference on Natural Language Learning*, 2002.

C. D. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

S. Mannor, R. Meir, and T. Zhang. The consistency of greedy algorithms for classification. *Proceedings of the Annual Conference on Computational Learning Theory*, 2002.

H. Markowitz. Portfolio selection. *Journal of Finance*, 7(1):77–91, 1952.

A. F. T. Martins, N. A. Smith, M. Q. A. Pedro, and M. A. T. Figueiredo. Structured sparsity in structured prediction. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011.

P. McCullagh and J. A. Nelder. *Generalized Linear Models, Second Edition*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability, 1989.

G. McLachlan. Discriminant analysis and statistical pattern recognition. *John Wiley*, 1992.

N. Meinshausen. Relaxed lasso. *Journal of Computational Statistics and Data Analysis*, 52(1):374–393, 2008.

N. Meinshausen and P. Buhlmann. Stability selection. *Technical report*, arXiv: 0809.2932, 2008.

T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur. Recurrent neural network based language model. *Proceedings of the Annual Conference of the International Speech Communication Association*, 2010.

T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. H. Cernocky. Strategies for training large scale neural network language models. *Proceedings of the Automatic Speech Recognition and Understanding Workshop*, 2011.

T. Mikolov, W-T. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013.

T. Minka. A comparison of numerical optimizers for logistic regression. *Unpublished draft.*, 2003.

M. Mitchell, J. Dodge, A. Goyal, K. Yamaguchi, K. Stratos, X. Han, A. Mensch, A. C. Berg, T. L. Berg, and H. Daume III. Midge: Generating image descriptions from computer vision detections. *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics*, 2012.

A. Mnih and G. Hinton. Three new graphical models for statistical language modelling. *Proceedings of the International Conference on Machine Learning*, 2007.

A. Mnih and G. Hinton. A scalable hierarchical distributed language model. *Advances in Neural Information Processing Systems*, 2008.

F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2005.

Y. Nardi and A. Rinaldo. On the asymptotic properties of the group lasso estimator for linear models. *Electronic Journal of Statistics*, 2:605–633, 2008.

Y. Nardi and A. Rinaldo. Autoregressive process modeling via the lasso procedure. *Journal of Multivariate Analysis.*, 102(3):528–549, 2011.

Y. Nardi and A. Rinaldo. The log-linear group lasso and its asymptotic properties. *Bernoulli*, 18(3):945–974, 2012.

B. K. Natarajan. Machine learning: A theoretical approach. *Morgan Kaufmann*, 1991.

R. Navigli. Word sense disambiguation: A survey. *ACM Computing Survey*, 41(2):1–69, 2009.

Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27:372–376, 1983.

Y. Nesterov. Gradient methods for minimizing composite objective function. *Technical report 76, Center for Operations Research and Econometrics, Catholic University of Louvain*, 2007.

W. Newman. Extension to the maximum entropy model. *IEEE Transactions on Information Theory*, 23(1):89–93, 1977.

H. Ney, U. Essen, and R. Kneser. On structuring probabilistic dependences in stochastic language modeling. *Computer, Speech, and Language*, 8(1):1–38, 1994.

J. Nocedal and S. J. Wright. Numerical optimization. *Springer Verlag*, 2nd edition, 2006.

G. Obozinski, M. J. Wainwright, and M. I. Jordan. High-dimensional union support recovery in multivariate regression. *Advances in Neural Information Processing Systems*, 2008.

G. Obozinski, L. Jacob, and J. P. Vert. Group lasso with overlaps: the latent group lasso approach. *Proceedings of the International Conference on Machine Learning*, 2009a.

G. Obozinski, B. Taskar, and M. I. Jordan. Joint covariate selection and joint subspace selection for multiple classification problems. *Journal of Statistics and Computing*, 20 (2):231–252, 2009b.

B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Journal of Vision Research*, 37:3311–3325, 1997.

V. Ordonez, G. Kulkarni, and T. L. Berg. Im2text: Describing images using 1 million captioned photographs. *Advances in Neural Information Processing Systems*, 2011.

M. R. Osborne, B. Presnell, and B. A. Turlach. On the lasso and its dual. *Journal of Computational and Graphical Statistics*, 9(2):319–337, 1999.

A. Owen. A robust hybrid of lasso and ridge regression. *Technical report, Stanford University*, 2006.

K. Papineni and S. Roukos. Bleu: a method for automatic evaluation of machine translation. *Proceedings of the Association for Computational Linguistics*, 2002.

P. M. Pardalos and N. Kovoor. An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds. *Journal of Mathematical Programming*, 46:321–328, 1990.

N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1 (3):123–231, 2013.

D. Percival. Theoretical properties of the overlapping groups lasso. *Electronic Journal of Statistics*, 6:269–288, 2012.

M. Perman, J. Pitman, and M. Yor. Size-biased sampling of poisson point processes and excursions. *Probability Theory and Related Fields*, 25(92):21–39, 1992.

Y. Rabani, L. J. Schulman, and C. Swamy. Approximation algorithms for labeling hierarchical taxonomies. *ACM-SIAM Symposium on Discrete Algorithms*, 2008.

R. Raina, A. Battle, H. Lee, B. Packer, and A.Y. Ng. Self-taught learning: Transfer learning from unlabeled data. *Proceedings of the International Conference on Machine Learning*, 2007.

A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. SimpleMKL. *Journal of Machine Learning Research*, 9:2491–2521, 2008.

F. Rapaport, E. Barillot, and J.-P. Vert. Classification of arraycgh data using fused svm. *Journal of Bioinformatics*, 24(13):375–382, 2008.

A. Ratnaparkhi. Maximum entropy models for natural language ambiguity resolution. *PhD thesis, University of Pennsylvania*, 1998.

P. Ravikumar, M. J. Wainwright, and J. Lafferty. High-dimensional ising model selection using ℓ_1-regularized logistic regression. *Annals of Statistics*, 38(3):1287–1319, 2010.

P. Ravikumar, A. Tewari, and E. Yang. On the ndcg consistency of listwise ranking methods. *Proceedings of the Conference on Artificial Intelligence and Statistics*, 2011.

E. Reiter and R. Dale. Building applied natural language generation systems. *Journal of Natural Language Engineering*, 3(1):57–87, 1997.

B. Roark, M. Saraclar, M. Collins, and M. Johnson. Discriminative language modeling with conditional random fields and the perceptron algorithm. *Proceedings of the Association of Computational Linguistics*, 2004.

S. Roch. Markov models on trees: Reconstruction and applications. *PhD thesis, University of California, Berkeley.*, 2007.

R. Rosenfeld. A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, 10:187–228, 1996.

R. Rosenfeld. Adaptive statistical language modeling: A maximum entropy approach. *PhD thesis, Carnegie Mellon University*, 1998.

A. Schijver. Combinatorial optimization. *Springer*, Volume C, 2003.

H. Schwenk. Efficient training of large neural networks for language modeling. *Proceedings of the International Joint Conference on Neural Networks*, 2004.

H. Schwenk and J-L. Gauvain. Training neural network language models on very large corpora. *Proceedings of the Empirical Methods in Natural Language Processing*, 2005.

N. Z. Shor. Minimization methods for non-differentiable functions. *Springer-Verlag*, 1985.

C. N. Silla Jr. and A. A. Freitas. A survey of hierarchical classification across different application domains. *Journal of Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2011.

A. J. Smola and B. Scholkopf. Sparse greedy matrix approximation for machine learning. *Proceedings of the International Conference on Machine Learning*, 2000.

S. Sra, S. Nowozin, and S.J. Eright. Optimization for machine learning. *MIT Press*, 2011.

N. Srebro, J. D. M. Rennie, and T. S. Jaakkola. Maximum-margin matrix factorization. *Advances in Neural Information Processing Systems*, 2005.

M. Steel. Some statistical aspects of the maximum parsimony method. *Experientia Supplementum*, 92:125–140, 2002.

K. Stefan, M. Tomas, K. Martin, and B. Lukas. Recurrent neural network based language modeling in meeting recognition. *Proceedings of the Annual Conference of the International Speech Communication Association*, 2011.

I. Steinwart. Consistency of support vector machines and other regularized classifiers. *Technical report, University of Jena*, 2002.

A. Stolcke. Srilm- an extensible language modeling toolkit. *Proceedings of International Conference on Spoken Language Processing*, Second Edition, 2002.

Y. W. Teh. A hierarchical bayesian language model based on pitman-yor processes. *Proceedings of the Association for Computation Linguistics*, 2006.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B*, 58(1):267–288, 1996.

R. Tibshirani and T. Hastie. Margin trees for high-dimensional classification. *Journal of Machine Learning Research*, 8:637–652, 2007.

R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *The Journal of the Royal Society of Statistics Series B*, 67(1):91–108, 2005.

K. E. Train. *Discrete Choice Methods with simulation.* The Cambridge University Press, 2003.

J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53:4655–4666, 2007.

J. A. Tropp, A. C. Gilbert, and M. J. Strauss. Algorithms for simultaneous sparse approximation. part i: Greedy pursuit. *Journal of Signal Processing*, 86:572–588, 2006a.

J. A. Tropp, A. C. Gilbert, and M. J. Strauss. Algorithms for simultaneous sparse approximation. part ii: Convex relaxation. *Journal of Signal Processing*, 86:589–602, 2006b.

P. Tseng. Convergence of block coordinate descent method for nondifferentiable maximation. *Journal of Optimization Theory and Applications*, 109:474–494, 2001.

P. Tseng. On accelerated proximal gradient methods for convex-concave optimization. *SIAM Journal on Optimization*, To appear, 2008.

E. Ukkonen. Online construction of suffix trees. *Algorithmica*, 1995.

V. N. Vapnik. Estimation of dependencies based on empirical data. *Springer-Verlag*, 1982.

V. N. Vapnik. The nature of statistical learning theory. *Springer-Verlag*, 1995.

V. N. Vapnik. Statistical learning theory. *John Wiley*, 1998.

G. Varoquaux, R. Jenatton, A. Gramfort, G. Obozinski, B. Thirion, and F. Bach. Sparse structured dictionary learning for brain resting-state activity modeling. *NIPS Workshop on Practical Applications of Sparse Modeling: Open Issues and New Directions.*, 2010.

V. Vural and J. G. Dy. A hierarchical method for multi-class support vector machines. *Proceedings of the International Conference on Machine Learning*, 2004.

M. J. Wainwright. Sharp thresholds for noisy and high-dimensional recovery of sparsity using ℓ_1-constrained quadratic programming. *Technical Report 709, Department of Statistics, UC Berkeley*, 2006.

Y. Wang, A. Acero, and C. Chelba. Is word error rate a good indicator for spoken language understanding accuracy. *IEEE Workshop on Automatic Speech Recognition and Understanding*, 2003.

D. J. Ward. Adaptive computer interfaces. 2001.

L. Wasserman and K. Roeder. High dimensional variable selection. *Annals of statistics*, 37(5(A)):2178–2201, 2009.

K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. *Proceedings of the International Conference on Machine Learning*, 2009.

J. Weston, S. Bengio, and D. Grangier. Label embedding trees for multiclass classification. *Advances in Neural Information Processing Systems*, 2010.

C. K. I. Williams and M. Seeger. Effect of the input density distribution on kernel-based classifiers. *Proceedings of the International Conference on Machine Learning*, 2000.

D. Wipf and B. Rao. ℓ_0-norm minimization for basis selection. *Advances in Neural Information Processing Systems*, 2005.

D. Wipf and B. Rao. An empirical bayesian strategy for solving the simultaneous sparse approximation problem. *IEEE Transactions on Signal Processing*, 55(7):3704–3716, 2007.

D. M. Witten, R. Tibshirani, and T. Hastie. A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. *Journal of Biostatistics,*, 10(3):515–534, 2009.

I. H. Witten and T. C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, 1991.

F. Wood, C. Archambeau, J. Gasthaus, L. James, and Y.W. Teh. A stochastic memoizer for sequence data. *Proceedings of the International Conference on Machine Learning*, 2010.

F. Wood, J. Gasthaus, C. Archambeau, L. James, and Y.W. Teh. The stochastic memoizer. *Communications of the Association for Computing Machinery*, 54(2):91–98, 2011.

About Wordnet. http://wordnet.princeton.edu/. *Princeton University.*, 2010.

J. Wu and S. Khudanpur. Efficient training methods for maximum entropy language modeling. *Proceedings of International Conference on Spoken Language Processing*, 2000.

L. Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11:2543–2596, 2010.

J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

J. Yang, K. Yu, and T. Huang. Supervised translation-invariant sparse coding. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

Y. Yang, C. L. Teo, H. Daume III, and Y. Aloimonos. Corpus-guided sentence generation of natural images. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011.

M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society Series B*, 68(1):49–67, 2006.

M. Yuan and Y. Lin. On the non-negative garrotte estimator. *Journal of the Royal Statistical Society Series B*, 69(2):143–161, 2007.

T. Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *Annals of Statistics*, 32(1):56–85, 2003.

Y. Zhang, A. d'Aspremont, and L. El Ghaoui. Sparse pca: Convex relaxations, algorithms and applications. *Handbook on Semidefinite, Cone and Polynomial Optimization*, pages 915–940, 2008.

P. Zhao and B. Yu. On model selection consistency of lasso. *Journal of Machine Learning Research*, 7:2006, 2006.

P. Zhao, G. Rocha, and B. Yu. The composite absolute penalties family for grouped and hierarchical variable selection. *The Annals of Statistics*, 37(6A):3468–3497, 2008.

G. Zipf. Selective studies and the principle of relative frequency in language. *Harvard University Press*, 1932.

H. Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101:1418–1429, 2006.

H. Zou and H. H. Zhang. On the adaptive elastic-net with a diverging number of parameters. *Annals of Statistics*, 4:1733–1751, 2009.