Classification avec des modèles probabilistes génératifs et des réseaux de neurones. Applications au traitement des langues naturelles

Elie Azeraf

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAS011

Thèse de doctorat

Classification avec des Modèles Probabilistes Génératifs et des Réseaux de Neurones. Applications au Traitement des Langues Naturelles

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom SudParis

École doctorale n°574 École doctorale de mathématiques Hadamard (EDMH)
Spécialité de doctorat : Mathématiques aux interfaces

Thèse présentée et soutenue à Paris, le 3 octobre 2022, par

Elie Azeraf

Composition du Jury :

François Roueff Professeur, Telecom Paris (LTCI)	Président
Myriam Maumy-Bertrand Maître de conférence, HDR, Université de Technologie de Troyes (LIST3N)	Rapporteur
Joseph Rynkiewicz Maître de conférence, HDR, Université Paris 1 (SAMM)	Rapporteur
François Yvon Professeur, Université Paris-Saclay (CNRS/LISN)	Examinateur
Wojciech Pieczynski Professeur, Telecom SudParis (SAMOVAR)	Directeur de thèse
Emmanuel Monfrini Professeur, Telecom SudParis (SAMOVAR)	Co-directeur de thèse
Emmanuel Vignon SICARA	Invité

Table des matières

Remerciements

Je tiens à remercier l'ensemble des personnes qui ont cru en moi, m'ont fait confiance, et m'ont donné la force et la motivation pour accomplir ce projet.

Tout d'abord, je tiens à remercier particulièrement mes directeurs de thèse : Pr. Pieczynski et Pr. Monfrini. Leurs patiences, leurs conseils, et leurs visions sur le long terme m'ont été d'une très grande utilité. Ce fut un immense plaisir pour moi d'accomplir ce travail en leur compagnie. Dès 2014 et ma première année d'étude en école d'ingénieur, les cours de probabilités de Pr. Monfrini ont su faire éclore en moi l'appétence pour cette matière, et ceux de Statistiques de Pr. Pieczynski l'année suivante ont su me confirmer cela. C'est ainsi un honneur et un plaisir d'avoir travaillé avec eux durant ces années.

Je remercie également les rapporteurs, Dr. Maumy-Bertrand et Dr. Rynkiewicz, ainsi que Pr. Yvon, qui ont pris le temps d'analyser minutieusement mon travail et d'émettre des remarques très pertinentes, me permettant de le perfectionner et d'en élargir les horizons. Je souhaite également remercier Pr. Roueff qui a accepté de présider mon jury de thèse.

Je remercie également mon entourage professionnel au sein d'IBM France, et plus particulièrement Emmanuel Vignon et Joffrey Martinez. Je tiens à remercier également les autres personnes qui y ont contribué et ont rendu cela possible : Gilles Foinkinos, Laurent Goujon, ainsi que Guilhaume Leroy-Meline.

Enfin, ceux sans qui je ne serais évidemment jamais arrivé aussi loin et qui m'ont toujours encouragé et soutenu tout au long de mes études : ma famille. Une mention spéciale à mes parents, Chantal et Jean-Marc, à mes frères, Kévin, Mendel, et Gamliel, et à mon épouse, Léa.

Liste des symboles et abréviations

Liste des symboles

$\mathbb{N}$	Ensemble des nombres entiers naturels
$\mathbb{N}^*$	Ensemble des nombres entiers naturels strictement positifs
$\mathbb{R}$	Ensemble des nombres réels
$\mathbb{R}^+$	Ensemble des nombres réels strictement positifs
X	Un symbole en italique et majuscule désigne une variable aléatoire, qui peut être un scalaire ou un vecteur
x	Un symbole en italique et minuscule désigne la réalisation d'une variable aléatoire
$X_{1:T}$	Variables aléatoires X_1 à X_T, $T \in \mathbb{N}^*$
$x_{1:T}$	Réalisations des variables aléatoires X_1 à X_T, $T \in \mathbb{N}^*$
$P(.)$ (et $p(.)$)	Distribution de probabilité (et notation raccourcie)
$P(.\|.)$ (et $p(.\|.)$)	Distribution de probabilité conditionnelle (et notation raccourcie)
$\mathbb{E}[.]$	Espérance d'une variable aléatoire

$\mathbb{E}[.\|.]$	Espérance conditionnelle d'une variable aléatoire
$1(.)$	Fonction indicatrice
0_N	Vecteur de taille N composé de $0, N \in \mathbb{N}^*$
$\mathrm{OH}^N(i)$	Vecteur de taille N composé de $0, N \in \mathbb{N}^*$, excepté un unique 1 à la position $i, i \in \{1, ..., N\}$
$\Lambda_X \times \Omega_Y$	Produit cartésien des ensembles Λ_X et Ω_Y
$\mathcal{N}(.,.)$	Densité de la loi gaussienne, le premier argument est la moyenne, le second est la variance

Liste des abréviations

CRF	Champ de Markov aléatoire *(Conditional Random Field)*
EFB	Avant-après entropique *(Entropic Forward-Backward)*
EV	Viterbi entropique *(Entropic Viterbi)*
FNN	Perceptron multicouche *(Feedforward Neural Network)*
GRU	Réseau de neurones récurrents à portes *(Gated Recurrent Unit)*
HMC	Chaîne de Markov cachée *(Hidden Markov Chain)*
HMC-CN	Chaîne de Markov cachée avec bruits complexifiés *(Hidden Markov Chain with Complexified Noise)*
HMC2	Chaîne de Markov cachée d'ordre 2 *(Hidden Markov Chain of order 2)*
HNMC	Chaîne de Markov cachée neuronale *(Hidden Neural Markov Chain)*

HNMC-CN	Chaîne de Markov cachée neuronale avec bruits complexifiés *(Hidden Neural Markov Chain with Complexified Noise)*
HNMC2	Chaîne de Markov cachée neuronale d'ordre 2 *(Hidden Neural Markov Chain of order 2)*
LSTM	Réseaux de neurones récurrents à mémoire courte et longue *(Long-Short Term Memory network)*
MAP	Maximum A Posteriori
MEMM	Modèle de Markov à entropie maximale *(Maximum Entropy Markov Model)*
MPM	Maximum Posterior Mode
Neural NB	*Naive Bayes* neuronal *(Neural Naive Bayes)*
NLP	Traitement des langues naturelles *(Natural Language Processing)*
PCRF	Champ de Markov aléatoire couple *(Pairwise Conditional Random Field)*
PMC	Chaîne de Markov couple *(Pairwise Markov Chain)*
Pooled MC	Chaîne de Markov mutualisée *(Pooled Markov Chain)*
Pooled MC2	Chaîne de Markov mutualisée d'ordre 2 *(Pooled Markov Chain of order 2)*
ReLU	Unité linéaire rectifiée *(Rectified Linear Unit)*
RNN	Réseaux de neurones récurrents *(Recurrent Neural Network)*
TMC	Chaîne de Markov triplet *(Triplet Markov Chain)*
TCRF	Champ de Markov aléatoire triplet *(Triplet Conditional Random Field)*

Introduction générale

Pour les tâches liées au Traitement des Langues Naturelles avec données supervisées, un grand nombre de modèles probabilistes, qualifiés de génératifs, ont été mis de côté. En effet, les classificateurs induits de ces modèles, dont font partie la chaîne de Markov cachée ou encore le *Naive Bayes*, sont considérés comme inefficaces pour traiter les tâches dont les observations ont un grand nombre de *features*, car ils sont supposés devoir modéliser la loi de ces observations. Ce problème peut s'avérer très complexe pour un grand nombre de domaines, notamment en Traitement des Langues Naturelles. Ainsi, d'autres modèles probabilistes tel que la Régression Logistique et le *linear-chain Conditional Random Field*, dits discriminants, sont préférés, car ces derniers n'ont pas à modéliser la loi des observations et peuvent donc appliquer leurs classificateurs induits sans contraintes. Cela poussa à délaisser les modèles génératifs depuis une vingtaine d'années pour ce type d'application.

Au cours de cette thèse, nous montrons qu'il est possible de calculer le classificateur induit de modèles probabilistes génératifs sans prendre en compte la loi des observations. Nous illustrons cette proposition pour de nombreux modèles, notamment le *Naive Bayes* et la chaîne de Markov cachée, ainsi qu'un grand nombre de leurs extensions. De plus, nous actualisons la comparaison entre certains modèles génératifs et discriminants à la lumière de notre proposition, dans la mesure où ces comparaisons étaient souvent effectuées en rapport avec la faculté du classificateur induit à devoir modéliser, ou non, les observations.

En termes d'applications, les modèles les plus utilisés pour la classification avec données supervisées ne sont pas basés sur les modèles probabilistes, mais sur les réseaux de neurones. Ces derniers sont devenues très populaires, atteignant des performances parfois meilleures qu'un humain pour certaines tâches. À partir des résultats précédents, nous présentons une manière originale permettant de paramétrer le classificateur induit d'un modèle génératif, calculé sans prendre en compte la loi des observations, avec des réseaux de neurones. Cela permet de concevoir de nouveaux modèles neuronaux avec une plus grande souplesse en terme de modé-

lisation. De plus, appliqués à la classification et à la segmentation de textes, nos nouveaux modèles neuronaux sont plus performants tout en étant plus rapides que les modèles classiques, soulignant leurs grands intérêts.

Organisation du manuscrit

Ce manuscrit s'articule autour de cinq chapitres. Le premier est introductif et rappelle les concepts de base sur les modèles probabilistes et la classification bayésienne. Une refonte des définitions des termes « génératif » et « discriminant », catégorisant les modèles et les classificateurs probabilistes, est également proposée, ainsi qu'un rappel sur la difficulté qui peut subvenir lorsque nous voulons modéliser la loi des observations. Enfin, nous introduisons les réseaux de neurones et décrivons en détail les différentes tâches de traitement des langues naturelles que nous aborderons au cours de nos travaux.

Le second chapitre est consacré au modèle probabiliste *Naive Bayes*. Après avoir présenté son classificateur induit usuellement utilisé, nous montrons comment le calculer sans devoir prendre en compte la loi des observations. Étant donné ce nouveau résultat, nous comparons ce modèle avec deux modèles discriminants, dont la Régression Logistique. Des applications pour la classification de textes et l'analyse de sentiments illustrent l'intérêt de ce nouveau calcul du classificateur induit.

Le troisième chapitre se consacre à un autre modèle probabiliste génératif très connu : la chaîne de Markov cachée. Nous proposons deux nouveaux algorithmes permettant d'en calculer des classificateurs bayésiens sans prendre en compte la loi des observations. De plus, une actualisation des comparaisons avec deux modèles discriminants, dont notamment le *linear-chain Conditional Random Field*, est proposée. Nous illustrons l'intérêt de nos nouveaux algorithmes en les appliquant à la décomposition syntaxique de textes.

Le quatrième chapitre propose un résultat général montrant comment calculer le classificateur induit d'un modèle probabiliste génératif sans avoir à considérer la loi des observations. Nous illustrons ce résultat pour un grand nombre de modèles probabilistes étendant le *Naive Bayes* et la chaîne de Markov cachée. Ce chapitre se conclut avec la comparaison d'extensions de la chaîne de Markov cachée et du *linear-chain Conditional Random Field*.

Le dernier chapitre montre comment paramétrer les différents classificateurs étudiés au cours de cette thèse avec des réseaux de neurones. Une comparaison

empirique sur diverses tâches de Traitement des Langues Naturelles est effectuée pour montrer l'intérêt de nos nouveaux modèles neuronaux par rapport à ceux existants.

Publications

La majeure partie des travaux présentés dans ce manuscrit font l'objet de publications scientifiques revues par les pairs ou en *preprints* :

— [1] : Elie Azeraf, Emmanuel Monfrini, and Wojciech Pieczynski. Using the Naive Bayes as a Discriminative Model. In *Proceedings of the 13th International Conference on Machine Learning and Computing*, page 106–110, 2021.
Cet article présente comment calculer le classificateur induit du *Naive Bayes* sans prendre en compte la loi des observations et le compare à la Régression Logistique.

— [2] : Elie Azeraf, Emmanuel Monfrini, Emmanuel Vignon, and Wojciech Pieczynski. Hidden Markov Chains, Entropic Forward-Backward, and Part-of-Speech Tagging. *arXiv preprint arXiv :2005.10629*, 2020
Cet article présente comment calculer le classificateur bayésien du Maximum Posterior Mode de la chaîne de Markov cachée sans utiliser la loi des observations.

— [3] : Elie Azeraf, Emmanuel Monfrini, and Wojciech Pieczynski. On Equivalence between Linear-chain Conditional Random Fields and Hidden Markov Chains. In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence - Volume 3 : ICAART*, pages 725–728. INSTICC, SciTePress, 2022.
Cet article montre l'équivalence entre la chaîne de Markov cachée et le *linear-chain Conditional Random Field*.

— [4] : Elie Azeraf, Emmanuel Monfrini, and Wojciech Pieczynski. Deriving Discriminative Classifiers from Generative Models. *arXiv preprint arXiv :2201.00844*, 2022.
Cet article montre comment calculer le classificateur induit d'un modèle probabiliste sans prendre en compte la loi des observations et présente de nombreux exemples sur des extensions du *Naive Bayes* et de la chaîne de Markov cachée.

— [5] : Elie Azeraf, Emmanuel Monfrini, and Wojciech Pieczynski. Improving Usual Naive Bayes Classifier Performances with Neural Naive Bayes based Models. In *Proceedings of the 11th International Conference on Pattern Recognition Applications and Methods - ICPRAM*, pages 315–322. INSTICC, SciTePress, 2022.

Cet article présente comment paramétrer le *Naive Bayes* et deux de ses extensions avec des réseaux de neurones et applique ces nouveaux modèles à l'analyse de sentiments.

— [6] : Elie Azeraf, Emmanuel Monfrini, Emmanuel Vignon, and Wojciech Pieczynski. Introducing the Hidden Neural Markov Chain Framework. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2 : ICAART*, pages 1013–1020. INSTICC, SciTePress, 2021.
Cet article présente comment paramétrer la chaîne de Markov cachée et deux de ses extensions avec des réseaux de neurones et compare ces nouveaux modèles avec les réseaux de neurones récurrents pour la segmentation de textes.

— [7] : Elie Azeraf, Emmanuel Monfrini, Emmanuel Vignon, and Wojciech Pieczynski. Highly Fast Text Segmentation With Pairwise Markov Chains. In *6th IEEE Congress on Information Science and Technology (CiSt)*, pages 361–366, 2020.
Cet article présente une méthode originale pour pouvoir appliquer le classificateur modélisant la loi des observations de la chaîne de Markov couple appliqué à la segmentation de textes.

Chapitre 1

Modèles probabilistes, classification, et réseaux de neurones

Ce chapitre rappelle l'ensemble des notions et concepts abordés au cours de cette thèse. Ainsi, en premier lieu, nous rappelons les définitions de modèles probabilistes et de classificateurs de Bayes. Puis nous analysons la catégorisation usuelle de ces concepts, répartis sous deux classes : générative et discriminante, en détaillant leurs propriétés concernant l'apprentissage des paramètres. Nous présentons par la suite les réseaux de neurones, qui sont les modèles les plus répandus de nos jours pour la classification supervisée de données. Au cours de cette thèse, de nombreuses expériences seront appliquées au traitement des langues naturelles, plus connu sous l'acronyme NLP pour *Natural Language Processing*. Par conséquent, les différentes tâches et les *datasets* utilisés sont également décrits en détail. Ce chapitre se conclut par l'énoncé des différentes problématiques qui seront traitées au cours de cette thèse.

1.1 Les modèles probabilistes

Détecter si un mail est un spam ou non, traduire un texte de l'anglais au français, ou encore trouver les noms propres au sein d'un paragraphe, sont des exemples d'applications liées au traitement des langues naturelles impliquant un grand nombre de variables avec des dépendances complexes les unes aux autres.

Il est possible de modéliser ces tâches et les données impliquées à l'aide de la théorie des probabilités, de variables aléatoires, et des lois de celles-ci. Ainsi, ces diverses tâches peuvent être modélisées comme la prédiction d'un ensemble de variables aléatoires cachées étant donné des variables aléatoires observées.

À titre d'exemple, l'étiquetage morphosyntaxique, consistant à trouver la fonction grammaticale de chaque mot dans une phrase, implique, étant donné $T \in \mathbb{N}^*$, les variables aléatoires cachées $X_{1:T} = (X_1, ..., X_T)$ et les variables aléatoires observées $Y_{1:T} = (Y_1, ..., Y_T)$, avec (X_t, Y_t) le couple composé de la fonction grammaticale et du mot à la position t. Nous pouvons également l'illustrer avec la traduction automatique de textes, qui implique, étant donné T et $T' \in \mathbb{N}^*$, les variables cachées $X_{1:T'} = (X_1, ..., X_{T'})$, modélisant les mots du texte dans la langue cible, avec comme variables observées $Y_{1:T} = (Y_1, ..., Y_T)$, les mots du texte dans la langue source.

Les différentes données, cachées et observées, peuvent être modélisées comme des variables aléatoires avec différentes dépendances entre elles. Cette modélisation complexe des données peut s'effectuer à l'aide de modèles probabilistes. Ceux-ci sont définis comme un triplet $(\Omega, \mathcal{F}, \mathbb{P})$ avec :

— Ω : l'espace d'états ;

— $\mathcal{F}$: un ensemble d'évènements possibles, tribu de Ω ;

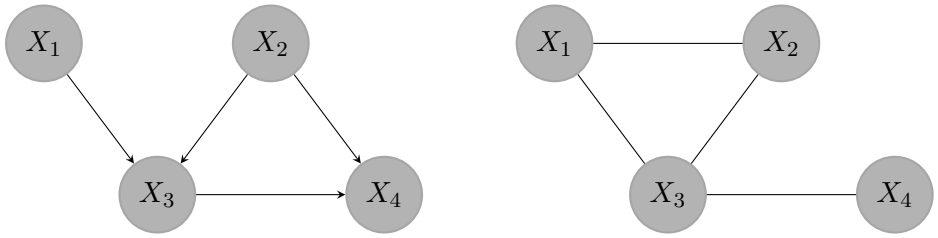
— P : distribution de probabilité définie de $\mathcal{F}$ sur $[0, 1]$.

Dans le cadre de cette thèse, $\mathcal{F}$ sera l'ensemble des parties de Ω, et seuls l'espace d'états et la distribution de probabilité seront énoncés pour définir un modèle probabiliste.

Nous nous intéressons aux modèles probabilistes liant une variable aléatoire observée Y dont la réalisation est à valeurs dans Ω_Y, qui peut être un espace multidimensionnel, discret ou continu, et une variable aléatoire cachée X dont la réalisation est à valeurs dans l'espace fini discret $\Lambda_X = \{\lambda_1, ..., \lambda_N\}$. Ces modèles permettent de traiter les tâches consistant à estimer une réalisation de X à partir d'une réalisation y de Y. Ce type de tâche est appelé une classification et constitue la famille de tâches la plus répandue de nos jours pour les applications liées à l'Intelligence Artificielle.

Exemple 1.1. L'analyse de sentiments consiste à associer à un texte donné le label *Positif* ou *Négatif*. Par conséquent, Λ_X =*(Positif, Négatif)* et nous devons prédire ledit sentiment. Les réalisations des variables observées sont les mots du texte, et nous notons Ω_Y l'ensemble de tous les mots écrits possibles. Nous pouvons définir différentes lois modélisant les liens entre les données, comme à titre d'exemple, pour tout $x \in \Lambda_X, y_{1:T} \in (\Omega_Y)^T$:

$$p(x, y_{1:T}) = p(x)p(y_1|x) \prod_{t=1}^{T-1} p(y_{t+1}|x, y_t).$$



(a) Exemple d'un graphe probabiliste orienté.

(b) Exemple d'un graphe probabiliste non-orienté.

FIGURE 1.1 – Exemples de graphes probabilistes, orienté et non-orienté.

Remarque 1.1. Tout au long de ce manuscrit, comme dans l'exemple précédent, sera parfois utilisé le format de notation raccourcie, pour tout $x \in \Lambda_X, P(X = x) = p(x)$. Dans certains cas, l'espace dans lequel est contenu x ne sera pas énoncé, car implicite.

Certains modèles probabilistes peuvent représenter les dépendances des différentes variables aléatoires sur un graphe et sont dénommés les modèles graphiques probabilistes. Un graphe est défini par un ensemble de sommets et d'arrêtes les reliant. Les variables aléatoires sont représentées par un sommet, et les dépendances sont induites par les arrêtes. Nous ne rentrerons pas dans les détails sur ces notions, et une description exhaustive est disponible dans [8, 9, 10] pour les personnes intéressées. Les notions utiles pour cette thèse concernent les deux catégories de modèles graphiques probabilistes : orientés et non-orientés. Soit $X_{1:T}$ un vecteur de variables aléatoires.

— les graphes orientés permettent de représenter des distributions $p(x_{1:T})$ s'écrivant :

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t|x_1, ..., x_{t-1}),$$

représentant les dépendances par des flèches orientées de $X_1, ..., X_{t-1}$ vers X_t, quand le besoin en est ;

— les graphes non-orientés permettent de représenter des distributions s'écrivant :

$$p(x_{1:T}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \phi_c(x_{1:T}),$$

avec, pour tout $c \in \mathcal{C}, \phi_c$ des fonctions potentielles et Z la constante de normalisation. L'ensemble des variables impliquées dans le calcul de ϕ_c constitue

une clique sur le graphe, qui est un ensemble de sommets tous liés entre eux sur le graphe par des segments non-orientés.

Exemple 1.2. Nous considérons les variables aléatoires $X_{1:4} = (X_1, X_2, X_3, X_4)$. Si leur loi s'écrit $p(x_{1:4}) = p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_2, x_3)$, alors son graphe probabiliste orienté peut être représenté en figure 1.1a. Si elle s'écrit $p(x_{1:4}) = \frac{1}{Z}\phi_1(x_1, x_2, x_3)\phi_2(x_3, x_4)$, alors son graphe probabiliste non-orienté peut être représenté en figure 1.1b.

Le cadre des modèles probabilistes a permis de définir un grand nombre de modèles très connus, tels que le *Naive Bayes* [11, 12], la Régression Logistique [13, 14, 15, 16], la chaîne de Markov cachée [17, 18, 19, 20, 21], le *Maximum Entropy Markov Model* [22], les *Conditional Random Fields* [23, 24], ou encore les machines de Boltzmann [25, 26]. Un grand nombre de ces modèles seront étudiés en détail dans le cadre de cette thèse.

1.2 Classification et approche bayésienne

1.2.1 Classification et classificateurs

Dans le cadre de cette thèse, nous nous intéressons exclusivement à la tâche de classification. Celle-ci consiste à affecter un label à une observation.

À titre d'exemple, la classification de textes permet d'assigner une catégorie à un texte parmi un ensemble donné, telle que *(Spam, Non-Spam)* pour les mails, ou encore *(Sport, Business, Monde, Autre)* pour des actualités. Parmi les tâches de classification, nous pouvons également citer la tâche de segmentation, consistant à affecter une classe à chaque observation. À titre d'exemple, le fait d'affecter à chaque mot d'un texte sa fonction grammaticale constitue une tâche de segmentation.

Une classification est effectuée à l'aide d'un classificateur, également appelée stratégie de classification, qui est une application de l'espace des observations dans celui des classes :

$$\phi : \Omega_Y \to \Lambda_X, y \to \phi(y).$$

Si le calcul de $\phi(y)$ repose sur des probabilités, il est qualifié de probabiliste.

Les tâches de classification occupent aujourd'hui une place prépondérante dans le monde de l'Intelligence Artificielle à travers un très grand nombre d'applications. D'autres types de tâches existent, tels que la régression ou l'apprentissage par renforcement. Dans le cadre de cette thèse, seulement la classification avec apprentissage supervisé, c'est-à-dire avec un échantillon d'apprentissage préalablement annoté pour l'entraînement, sera traitée.

1.2.2 Stratégie bayésienne de classification

Nous considérons les variables aléatoires X et Y et la loi a posteriori $p(x|y)$. Un classificateur ϕ est une application de Ω_Y dans Λ_X. Pour une réalisation $(X, Y) = (x, y), \phi$ peut donner la bonne réponse, $\phi(y) = x$, ou se tromper, $\phi(y) \neq x$. Cela peut être modélisé par une fonction de coût $\mathcal{L}$:

$$\mathcal{L} : \Lambda_X \times \Lambda_X \to \mathbb{R}^+, (x, \hat{x}) \to \mathcal{L}(x, \hat{x}).$$

Cette fonction sert à mesurer le coût d'une classification $\phi(y)$ par rapport à la vérité terrain x, y associant la valeur $\mathcal{L}(x, \phi(y))$. Elle permet de s'adapter à la tâche demandée, notamment lorsque l'erreur de classification n'est pas de gravité équivalente. À titre d'exemple, pour la détection de spam parmi les mails, les erreurs n'ont pas le même coût. En effet, classifier à tort un mail non-spam n'a pas le même impact que considérer un spam comme n'en étant pas un. Ainsi, la définition de la fonction de coût permet d'avoir une souplesse dans l'évaluation du classificateur. En tant que mesure d'erreur, elle doit respecter la condition suivante :

$$\mathcal{L}(x, \hat{x}) = 0 \Leftrightarrow x = \hat{x}.$$

Pour la réalisation $Y = y$, avec une vérité terrain $X = x$, le coût de l'estimation avec le classificateur ϕ vaut donc $\mathcal{L}(x, \phi(y))$. Supposons que nous n'ayons pas un seul couple, mais un ensemble de couples de variables aléatoires $\left(X^{(n)}, Y^{(n)}\right)_{n\in\mathbb{N}}$ de même loi que (X, Y). Sous certaines conditions techniques que nous admettons, la perte moyenne $\frac{\mathcal{L}(X^{(1)},\phi(Y^{(1)}))+\ldots+\mathcal{L}(X^{(n)},\phi(Y^{(n)}))}{n}$ est une variable aléatoire qui tend presque sûrement, en vertu de la loi forte des grands nombres, vers $\mathbb{E}\left[\mathcal{L}(X, \phi(Y))\right]$. Cette dernière est appelée le risque du classificateur ϕ et du coût $\mathcal{L}$:

$$\mathcal{R}(\phi, \mathcal{L}) = \mathbb{E}\left[\mathcal{L}(X, \phi(Y))\right].$$

Par définition, le classificateur bayésien $\phi_B^{\mathcal{L}}$, également appelé classificateur de Bayes, est celui parmi tous les classificateurs qui minimise le risque :

$$\mathcal{R}(\phi_B^{\mathcal{L}}, \mathcal{L}) = \min_{\phi} \mathcal{R}(\phi, \mathcal{L}).$$

Le risque associé au classificateur bayésien est appelé risque bayésien. Plus de détails à propos des classificateurs bayésiens sont disponibles dans [27, 28, 29].

Nous allons notamment nous intéresser à la forme générale de ce classificateur étant donné la loi a posteriori $p(x|y)$. D'après la formule de Fubini, la perte moyenne peut s'écrire :

$$\mathbb{E}\left[\mathcal{L}(X, \phi(Y))\right] = \mathbb{E}\left[\mathbb{E}\left[\mathcal{L}(X, \phi(Y))|Y\right]\right].$$

La minimisation de $\mathbb{E}\left[\mathcal{L}(X, \phi(Y))|Y\right]$ en tout point assure la minimisation de son espérance. Ainsi, nous recherchons le classificateur $\phi_B^{\mathcal{L}}$ à partir de $\mathbb{E}\left[\mathcal{L}(X, \phi(Y))|Y\right]$. Pour la réalisation $Y = y$:

$$\mathbb{E}\left[\mathcal{L}(X, \phi(Y))|Y = y\right] = \sum_{\lambda_j \in \Lambda_X} \mathcal{L}(\lambda_j, \phi(y)) P(X = \lambda_j | Y = y).$$

Ainsi, le classificateur bayésien associé à la perte $\mathcal{L}$, étant donné la réalisation $Y = y$ et la loi a posteriori $p(x|y)$, est donné par la formule :

$$\phi_B^{\mathcal{L}}(y) = \arg\min_{\lambda_i \in \Lambda_X} \sum_{\lambda_j \in \Lambda_X} \mathcal{L}(\lambda_j, \lambda_i) P(X = \lambda_j | Y = y). \tag{1.1}$$

1.2.3 MAP et MPM

Dans cette partie, nous décrivons les classificateurs bayésiens du Maximum A Posteriori (MAP) et du Maximum Posterior Mode (MPM), qui sont parmi les plus couramment utilisés. Nous considérons le cas où X est un vecteur aléatoire : $X_{1:T} = (X_1, ..., X_T)$, avec X_t à valeurs dans Λ_X pour tout $T \in \mathbb{N}^*$. Pour sa part, Y est une variable aléatoire, vecteur ou scalaire, prenant ses valeurs dans Ω_Y.

Le classificateur bayésien du MAP est associé à la fonction de perte suivante :

$$\mathcal{L}_{MAP} : (\Lambda_X)^T \times (\Lambda_X)^T \to \{0, 1\}, (x_{1:T}, \hat{x}_{1:T}) \to \mathbb{I}(x_{1:T} \neq \hat{x}_{1:T}).$$

Ainsi, à partir de (1.1), le classificateur bayésien a la forme :

$$\begin{aligned}
\phi_B^{MAP}(y) &= \arg\min_{\lambda \in (\Lambda_X)^T} \sum_{\lambda' \in (\Lambda_X)^T} \mathbb{I}(\lambda \neq \lambda') P(X_{1:T} = \lambda' | Y = y) \\
&= \arg\min_{\lambda \in (\Lambda_X)^T} \sum_{\substack{\lambda' \in (\Lambda_X)^T, \\ \lambda' \neq \lambda}} P(X_{1:T} = \lambda' | Y = y) \\
&= \arg\min_{\lambda \in (\Lambda_X)^T} 1 - P(X_{1:T} = \lambda | Y = y) \\
&= \arg\max_{\lambda \in (\Lambda_X)^T} P(X_{1:T} = \lambda | Y = y).
\end{aligned} \tag{1.2}$$

Le classificateur bayésien du MAP maximise donc la loi a posteriori $p(x_{1:T}|y)$.

Concernant l'estimateur du MPM, il est associé à la fonction de perte :

$$\mathcal{L}_{MPM} : (\Lambda_X)^T \times (\Lambda_X)^T \to \{0, 1, ..., T\}, (x_{1:T}, \hat{x}_{1:T}) \to \sum_{t=1}^{T} \mathbb{I}(x_t \neq \hat{x}_t).$$

De manière similaire à précédemment pour le MAP, le classificateur servant à définir le MPM a la forme :

$$\begin{aligned}
\phi_B^{MPM}(y) &= \arg \min_{\lambda_{1:T} \in (\Lambda_X)^T} \sum_{\lambda'_{1:T} \in (\Lambda_X)^T} \sum_{t=1}^{T} \mathbb{I}(\lambda_t \neq \lambda'_t) P(X_{1:T} = \lambda'_{1:T} | Y = y) \\
&= \arg \min_{\lambda_{1:T} \in (\Lambda_X)^T} \sum_{t=1}^{T} \sum_{\lambda'_t \in \Lambda_X} \mathbb{I}(\lambda_t \neq \lambda'_t) P(X_t = \lambda'_t | Y = y) \\
&= \arg \min_{\lambda_{1:T} \in (\Lambda_X)^T} \sum_{t=1}^{T} \sum_{\substack{\lambda'_t \in \Lambda_X, \\ \lambda'_t \neq \lambda_t}} P(X_t = \lambda'_t | Y = y) \\
&= \arg \min_{\lambda_{1:T} \in (\Lambda_X)^T} \sum_{t=1}^{T} 1 - P(X_t = \lambda_t | Y = y) \\
&= \arg \max_{\lambda_{1:T} \in (\Lambda_X)^T} \sum_{t=1}^{T} P(X_t = \lambda_t | Y = y).
\end{aligned}$$

Les termes de cette somme étant tous positifs ou nuls, nous la maximisons en maximisant chacun de ses termes. Par conséquent, l'estimateur du MPM consiste à prédire, pour tout t, λ_t tel que $P(X_t = \lambda_t | Y = y)$ soit maximal.

Ainsi, au vu des fonctions de pertes respectives, l'estimateur du MAP cherche à restaurer la séquence cachée entière la plus probable, tandis que celui du MPM se focalise sur une restauration terme-à-terme la plus probable.

1.3 Catégorisation des modèles probabilistes et des classificateurs : génératifs et discriminants

1.3.1 Modèles probabilistes génératifs et discriminants

Le classificateur de Bayes dépend d'une fonction de coût donnée ainsi que d'une loi a posteriori. La donnée de cette loi a posteriori peut s'effectuer de deux manières différentes :

— soit elle est donnée directement ;

— soit la loi jointe $p(x, y)$ est donnée, et nous calculons la loi a posteriori :

$$p(x|y) = \frac{p(x, y)}{\sum_{x'} p(x', y)}. \tag{1.3}$$

Cela amène à catégoriser les modèles probabilistes en deux classes : génératifs et discriminants.

De nombreux ouvrages de littérature traitent de ce sujet. Nous allons passer en revue quelques définitions issues d'ouvrages de référence :

— dans [24] :

> *A generative model [..] is a family of joint distributions that factorizes as $p(x, y) = p(x)p(y|x)$ [..]. A discriminative model is [..] a family of conditional distributions $p(x|y)$, that is, the classification rule is modeled directly. In principle, a discriminative model could also be used to obtain a joint distribution $p(x, y)$ by supplying a marginal distribution $p(y)$ over the inputs, but this is rarely needed.*

— dans [30] :

> *In a discriminative approach we define the conditional distribution $p(x|y)$ [..] on generative models, which define a joint distribution $p(x, y)$ over both input vectors and class labels*

— dans [31] :

> *Approaches that explicitly or implicitly model the distribution of inputs as well as outputs are known as generative models [..] Approaches that model the posterior probabilities directly are called discriminative models.*

— d'autres définitions similaires peuvent être consultées dans [32, 33, 34, 8, 35, 36, 37], parmi d'autres.

Dans le cadre de cette thèse, nous allons définir ces notions avec les définitions suivantes, qui respectent les idées et concepts rencontrés dans la littérature à ce sujet. Elles ne concernent pas la loi $p(x, y)$ en elle-même, mais la manière d'écrire celle-ci. Ces définitions se rapprochent de celles proposées dans [32, 24] notamment.

> **Modèle probabiliste écrit de manière générative :** Un modèle probabiliste écrit de manière générative est un modèle probabiliste dont

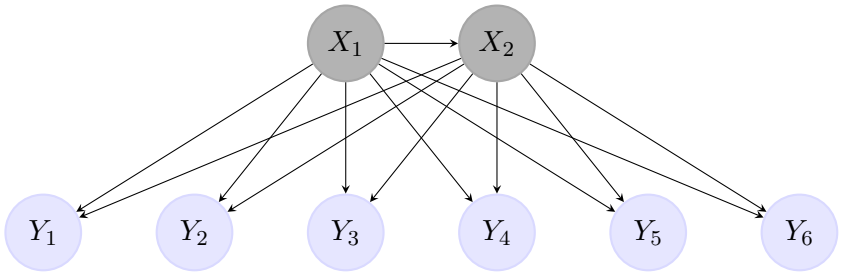


FIGURE 1.2 – Graphe probabiliste orienté du modèle probabiliste défini par (1.4).

l'écriture de la loi jointe $p(x, y)$ possède au moins un facteur $p(y_A|x_B)$, avec y_A et x_B des ensembles non-vides des variables aléatoires observées et cachées.

Modèle probabiliste écrit de manière discriminante : Un modèle probabiliste écrit de manière discriminante est un modèle probabiliste dont l'écriture de la loi jointe $p(x, y)$ ne possède pas de facteur $p(y_A|x_B)$, avec y_A et x_B des ensembles non-vides des variables aléatoires observées et cachées.

Remarque 1.2. L'intérêt de l'une ou l'autre des écritures d'un modèle réside dans la simplification des dépendances que l'une des écritures va apporter. Soit, par exemple, le modèle considérant les variables aléatoires cachées $X_{1:2}$ et observées $Y_{1:6}$, défini par la loi jointe écrite de manière générative suivante :

$$p(x_{1:2}, y_{1:6}) = p(x_1)p(x_2|x_1)\prod_{t=1}^{6} p(y_t|x_1, x_2). \tag{1.4}$$

Il peut être représenté en figure 1.2. Ce même modèle peut s'écrire de manière discriminante de la manière suivante :

$$p(x_{1:2}, y_{1:6}) = \left[\prod_{t=1}^{6} p(y_t)\right] p(x_1|y_{1:6})p(x_2|x_1, y_{1:6}). \tag{1.5}$$

Les deux écritures impliquent des dépendances qui s'avèreront différentes, notamment pour l'apprentissage des lois du classificateur induit défini de manière directe avec la règle de Bayes, comme nous allons le voir dans la section suivante.

1.3.2 Les classificateurs génératifs et discriminants

De même que les modèles probabilistes, les classificateurs qui en sont induits sont également répartis en ces deux catégories [38, 39, 36]. L'un des papiers de

référence sur ces notions [40] les défini comme suit :

> *Generative classifiers learn a model of the joint probability, $p(x, y)$, of the inputs X and the label Y, and make their predictions by using Bayes rules to calculate $p(x|y)$, and then picking the most likely label X. Discriminative classifiers model the posterior $p(x|y)$ directly*

Ainsi, de manière usuelle, un classificateur probabiliste génératif, défini à partir d'une loi jointe $p(x, y)$, calcule en premier lieu cette loi jointe, puis utilise la règle de Bayes pour calculer la loi a posteriori $p(x|y)$. Un classificateur discriminant calcule directement la loi a posteriori $p(x|y)$, sans passer par le calcul de la loi jointe.

Propriété (P) Une propriété fondamentale est mise en avant pour comparer ces deux types de classificateurs : un classificateur génératif se soucie de la loi des observations lors de l'apprentissage des paramètres et l'inférence, ce qui n'est pas le cas des classificateurs discriminants. Cette propriété peut être consultée dans un très grand nombre de travaux [40, 39, 22, 24, 23, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]. Elle est centrale et constitue la frontière entre les classificateurs génératifs et discriminants.

Comme nous allons le voir, la frontière entre ces catégories de classificateurs n'est pas aussi nette que présentée dans la littérature. En effet, notre apport consiste à montrer que certains classificateurs présentés comme génératifs peuvent également vérifier la définition de classificateur discriminant. Ainsi, tout au long de cette thèse, ces notions sont définies comme suit, tout en gardant les mêmes propriétés que les définitions usuelles dans la littérature :

> **Classificateur probabiliste calculé de manière générative :** Un classificateur probabiliste calculé de manière générative est un classificateur probabiliste dont le calcul de la loi a posteriori utilise au moins une loi $p(y_A|x_B)$, avec y_A et x_B des ensembles non-vides des variables aléatoires observées et cachées.
>
> **Classificateur probabiliste calculé de manière discriminante :** Un classificateur probabiliste calculé de manière discriminante est un classificateur probabiliste dont le calcul de la loi a posteriori n'utilise pas de loi de la forme $p(y_A|x_B)$, avec y_A et x_B des ensembles non-vides des variables aléatoires observées et cachées.

Ainsi, nous proposons de nouvelles définitions de ces notions, gardant le même esprit que voulu dans la littérature. Par la suite, nous dénommerons abusivement pour éviter d'alourdir le texte :

— modèle probabiliste écrit de manière générative $\longrightarrow$ modèle génératif ;

— modèle probabiliste écrit de manière discriminante $\longrightarrow$ modèle discriminant.

Remarque 1.3. À la vue de nos définitions, il apparaît qu'il est direct pour un modèle génératif de définir un classificateur calculé de manière générative ; de même pour un modèle discriminant permettant de définir un classificateur calculé de manière discriminante. Ainsi, pour illustrer, le modèle probabiliste introduit en Remarque 1.2., écrit de manière générative (1.4), défini son classificateur calculé de manière générative ainsi :

$$\phi(y_{1:6}) = \arg\max_{x_1,x_2} \left(\frac{p(x_1)p(x_2|x_1)\prod_{t=1}^{6} p(y_t|x_1,x_2)}{\sum_{x'_1,x'_2} p(x'_1)p(x'_2|x_1)\prod_{t=1}^{6} p(y_t|x'_1,x'_2)} \right)$$

En partant de l'écriture discriminante de ce même modèle (1.5), le classificateur induit calculé de manière discriminante vérifie :

$$\phi(y_{1:6}) = \arg\max_{x_1,x_2} \left(\frac{p(x_1|y_{1:6})p(x_2|x_1,y_{1:6})}{\sum_{x'_1,x'_2} p(x'_1|y_{1:6})p(x'_2|x'_1,y_{1:6})} \right)$$

Dans ce dernier cas, il se calcule de la même manière qu'un modèle probabiliste impliquant les mêmes variables aléatoires observées et cachées, mais complètement connectées. Ainsi, le classificateur induit directement à partir de l'écriture discriminante du modèle a « perdu » ses dépendances simplifié, montrant l'intérêt du classificateur induit directement de l'écriture générative dans ce cas.

1.4 Apprentissage des paramètres

Selon le moyen de calculer le classificateur, l'apprentissage des paramètres sera différent. Dans cette partie, nous présentons les deux méthodes d'apprentissage les plus couramment utilisées :

— estimation par maximum de vraisemblance de la loi jointe, pour les classificateurs calculés de manière générative ;

— la descente de gradient sur une fonction de coût convenablement choisie, pour les classificateurs calculés de manière discriminante.

1.4.1 Apprendre les paramètres d'un classificateur calculé de manière générative par maximum de vraisemblance

Étant donné un classificateur calculé de manière générative, celui-ci dépend d'un ensemble de lois de la forme $p(x_A), p(x_B|y_C), p(y_D|x_E)$, avec A, B, C, D, E des sous-ensembles. Il est donc nécessaire d'apprendre ces lois à partir d'un ensemble de N réalisations du couple $\mathcal{D} = (x^{(n)}, y^{(n)})_{1 \leq n \leq N}$, appelé *dataset* d'entraînement. L'estimation par maximum de vraisemblance est une méthode très répandue pour apprendre les paramètres de ces lois. Elle consiste à poser la vraisemblance (à un facteur près) utilisée par le classificateur, puis à la maximiser en fonction des différents paramètres.

Exemple 1.3. Nous considérons la variable aléatoire cachée X avec $\Lambda_X = \{\lambda_1, \lambda_2\}$, les variables aléatoires observées Y_1, Y_2, avec $\Omega_Y = \{\omega_1, \omega_2, \omega_3\}$, et le modèle génératif $p(x, y_1, y_2) = p(x)p(y_1|x)p(y_2|x)$. La vraisemblance des données d'apprentissage $\mathcal{D}$ s'écrit :

$$Li(\mathcal{D}) = \prod_{n=1}^{N} p\left(x^{(n)}\right) p\left(y_1^{(n)}|x^{(n)}\right) p\left(y_2^{(n)}|x^{(n)}\right).$$

Nous passons au logarithme :

$$li(\mathcal{D}) = \sum_{n=1}^{N} \log\left(p\left(x^{(n)}\right)\right) + \log\left(p\left(y_1^{(n)}|x^{(n)}\right)\right) + \log\left(p\left(y_2^{(n)}|x^{(n)}\right)\right).$$

Supposons que nous souhaitons estimer $P(X = \lambda_1)$, que nous notons $\pi(1)$, donc $P(X = \lambda_2) = 1 - \pi(1)$ (usuellement, les paramètres estimés sont notés $\hat{\pi}(1)$, pour alléger les notations et sans perte de généralité, la notation sans chapeau est utilisée tout au long de cette thèse). Nous dérivons la log-vraisemblance par rapport à cette quantité :

$$\begin{aligned}
&\frac{\partial li(\mathcal{D})}{\partial \pi(1)} = 0 \\
\Leftrightarrow &\sum_{n=1}^{N} \mathbb{I}\left(x^{(n)} = \lambda_1\right) \frac{1}{\pi(1)} + \mathbb{I}\left(x^{(n)} = \lambda_2\right) \frac{-1}{1 - \pi(1)} = 0 \\
\Leftrightarrow &\pi(1) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}\left(x^{(n)} = \lambda_1\right) = \frac{N_i}{N},
\end{aligned}$$

avec N_i le nombre de fois où $X = \lambda_i$ dans le jeu d'entraînement. Les autres paramètres sont également estimables par cette méthode de comptage. À titre

d'exemple, en notant $b_i^{(1)}(\omega_j) = P(Y_1 = \omega_j | X = \lambda_i)$, son estimation par maximum de vraisemblance est donnée par la formule :

$$b_i^{(1)}(\omega_j) = \frac{N_{i,j}^{(1)}}{\sum_{k=1}^{3} N_{i,k}},$$

avec $N_{i,j}^{(1)}$ le nombre de fois où $X = \lambda_i$ lorsque $Y_1 = \omega_j$ dans le *dataset* d'entraînement.

Remarque 1.4. L'exemple présente le cas où Ω_Y est un espace discret, mais cette estimation est également possible pour des valeurs continues. Si nous reprenons cet exemple, mais $\Omega_Y = \mathbb{R}$ et $P(Y_t = y_t | X = \lambda_i) \sim \mathcal{N}(\mu_{i,t}, 1)$, l'estimation de $\mu_{i,t}$ par maximum de vraisemblance est égal à la moyenne des observations y_t lorsque $X = \lambda_i$.

Nous avons présenté l'estimation par maximum de vraisemblance pour l'apprentissage d'un classificateur calculé de manière générative, qui est la méthode la plus couramment utilisée, et consistant à estimer les différentes lois par la fréquence des différents *patterns*. Cet algorithme présente l'avantage d'être rapide, avec une complexité linéaire en fonction de la taille du jeu d'apprentissage.

1.4.2 Apprendre les paramètres d'un classificateur calculé de manière discriminante par descente de gradient

Soit une fonction $f : \mathbb{R}^d \to \mathbb{R}^{d'}, d, d' \in \mathbb{N}^*$, dérivable. L'une des méthodes les plus répandues pour trouver l'un des minimums locaux de la fonction f est la descente de gradient.

On se donne un point initial $\theta^{(0)} \in \mathbb{R}^d$. L'algorithme de descente de gradient permet de définir une suite $\theta^{(1)}, \theta^{(2)}, ...$ jusqu'à un critère d'arrêt de la manière suivante :

$$\theta^{(k+1)} = \theta^{(k)} - \alpha \frac{\partial f}{\partial \theta}(\theta^{(k)}),$$

avec α appelé le pas d'apprentissage. Le critère d'arrêt concerne généralement les variations de f sur un certain nombre d'étapes.

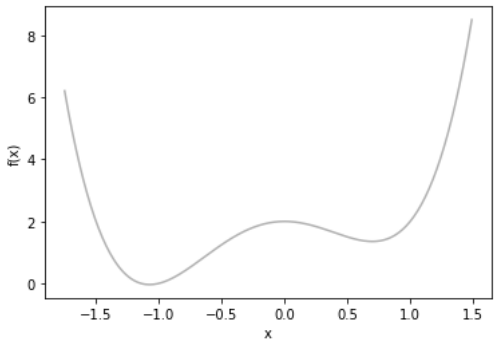
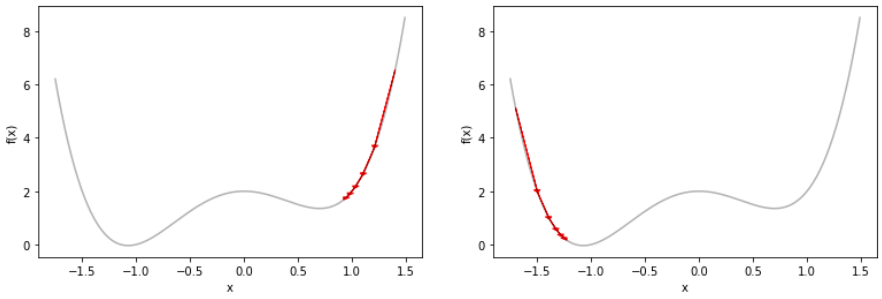


FIGURE 1.3 – Représentation de la fonction $f(x)$ sur $[-1.75; 1.5]$.



(a) En prenant $\theta^{(0)} = 1.4$. (b) En prenant $\theta^{(0)} = -1.7$.

FIGURE 1.4 – Exemples de descente de gradient sur 5 étapes.

Exemple 1.4. Soit la fonction :

$$f : \mathbb{R} \longrightarrow \mathbb{R}, \theta \longrightarrow 2\theta^4 + \theta^3 - 3\theta^2 + 2,$$

représentée en figure 1.3. Nous calculons le gradient de f par rapport à θ :

$$\frac{\partial f}{\partial \theta} = 8\theta^3 + 3\theta^2 - 6\theta.$$

Nous sélectionnons comme point initial $\theta^{(0)} = 1.4$ avec un pas d'apprentissage de 0.01 et comme critère d'arrêt le fait d'effectuer cinq étapes. L'algorithme de descente de gradient aboutit à la valeur finale $f\left(\theta^{(5)}\right) = 1.68$. Le « chemin » suivi est représenté en figure 1.4a. Nous recommençons cet algorithme avec comme point initial $\theta^{(0)} = -1.7$ avec le même pas d'apprentissage et le même critère d'arrêt. Cette fois-ci, $f(\theta^{(5)}) = 0.16$, et le « chemin » suivi est représenté en figure 1.4b.

La descente de gradient est la méthode la plus utilisée pour l'entraînement d'un classificateur calculé de manière discriminante. Soient les données $\mathcal{D} =$

$(x^{(n)}, y^{(n)})_{1 \leq n \leq N}$ et le classificateur probabiliste ϕ, nous choisissons une fonction à minimiser dérivable (appelée fonction de coût dans ce cadre également) convenablement choisie, comme la *cross-entropy* :

$$L_{CE}(\mathcal{D}) = -\sum_{x,y \in \mathcal{D}} \sum_{\lambda_i \in \Lambda_X} \mathbb{I}(x = \lambda_i) \log(P(X = \lambda_i | Y = y)), \tag{1.6}$$

que nous minimisons sur les paramètres de ϕ, qui doit également être dérivable sur ces derniers.

La descente de gradient connaît de nombreuses variations. Nous avons décrit la descente de gradient classique, mais il existe également la descente de gradient ne consistant à calculer le gradient que d'une partie du jeu de données, tirée aléatoirement à chaque étape, et permettant à un modèle de s'entraîner plus rapidement sur de très grands jeux de données.

Des méthodes plus complexes existent, telles que RMSProp [51] ou Adadelta [52]. Dans le cadre de cette thèse, nous utiliserons notamment la méthode d'optimisation Adam [53], considérée comme l'une des plus efficaces. Une revue des principaux algorithmes de descente de gradient est disponible dans [54].

1.4.3 Le problème de la gestion des features des observations pour les classificateurs calculés de manière générative

De par sa construction, un classificateur calculé de manière générative a besoin de calculer numériquement une loi de la forme $p(y_A|x_B)$. Cette étape, nécessitant la prise en compte de la distribution d'une partie des observations, est source de critiques. Nous citons Vapnik [55] cité dans [40] sur ce sujet : *"one should solve the [classification] problem directly and never solve a more general problem as an intermediate step [such as modeling $p(x, y)$]."* En effet, la modélisation de la loi d'une partie des observations est un problème pouvant s'avérer très complexe, notamment concernant la gestion des *features* de ces observations, car il faut en apprendre les lois. Quand l'espace des *features* est « grand », ce qui est fréquemment le cas en NLP par exemple, ce problème s'avère impossible à résoudre étant donné le nombre de paramètres à estimer. Le plus regrettable étant le fait que l'apprentissage de cette loi n'est en rien obligatoire dans la mesure où l'utilisation d'un classificateur calculé de manière discriminante permet de s'en abroger.

Illustrons ce problème par un exemple. Supposons que nous souhaitons faire de la classification de mots : à un mot en français écrit en minuscule donné, nous

souhaitons savoir s'il est féminin, masculin, ou n'a pas de genre. Ainsi, nous avons l'espace des classes de taille 3. Nous considérons la variable cachée X ainsi que la variable observée Y, et nous souhaitons apprendre la loi conditionnelle $p(y|x)$ qui sera utilisé par un classificateur calculé de manière générative. Nous considérons un premier scénario où un mot est représenté par trois *features* :

— son suffixe de taille 1 ;

— son suffixe de taille 3 ;

— son préfixe de taille 3.

La variable observée Y est donc un vecteur aléatoire à trois composantes, $Y = (Y^{(1)}, Y^{(2)}, Y^{(3)})$. L'espace de l'ensemble des suffixes de taille 1 est noté $\Omega_Y^{(1)}$, de cardinal supposé égal à 26, celui de l'ensemble des suffixes de taille 3 est noté $\Omega_Y^{(2)}$, de cardinal 17576, et celui de l'ensemble des préfixes de taille 3 est noté $\Omega_Y^{(3)}$, de cardinal 17576 également. En utilisant la méthode de comptage, pour tout $y^{(1)} \in \Omega_Y^{(1)}, y^{(2)} \in \Omega_Y^{(2)}, y^{(3)} \in \Omega_Y^{(3)}, \lambda_i \in \Lambda_X$, nous estimons la probabilité de la manière suivante :

$$P(Y^{(1)} = y^{(1)}, Y^{(2)} = y^{(2)}, Y^{(3)} = y^{(3)}|X = \lambda_i) = \frac{N(\lambda_i, y^{(1)}, y^{(2)}, y^{(3)})}{N(\lambda_i)}.$$

Avec $N(.)$ la fonction de comptage, comptant le nombre d'occurrence du *pattern* donné dans l'ensemble d'entraînement. Cela implique l'apprentissage de $26 \times 17576 \times 17576 \times 3$ paramètres, imposant la nécessité d'avoir un très grand nombre de données d'entraînement annotées à disposition pour avoir des estimations fiables.

Il est pertinent de noter que nous avons considéré peu de *features*. Le nombre de paramètres aurait été encore bien plus élevé si nous avions pris tous les suffixes et tous les préfixes de taille 10 à 1, à titre d'exemple. Cette observation montre une première limite de la méthode de comptage, liée à un nombre très élevé de paramètres à apprendre.

De plus, ce problème implique l'impossibilité de capter l'influence d'une *feature* prise de manière individuelle pour l'inférence. En effet, supposons pour l'exemple que voulons prédire la classe du mot *gentille*, représenté *(e, lle, gen)*, et supposons que cette combinaison des trois *features* n'a jamais été rencontré dans les données d'entraînement. Alors, pour tout $\lambda_i \in \Lambda_X$, $P(Y^{(1)} = e, Y^{(1)} = lle, Y^{(1)} = gen|X = \lambda_i) = 0$. Pourtant, le suffixe de taille 1 est un e, ce qui a une influence certaine sur la probabilité que le mot soit féminin, mais nous ne pouvons rien en déduire. La méthode de comptage impose un « tout-ou-rien » concernant les *features*. La probabilité estimée d'une observation sera différente de 0 si et seulement s'il existe

une observation dans le *dataset* d'entraînement ayant exactement toutes les *features* identiques. Cela nous montre un autre problème du fait de devoir apprendre la loi des observations ainsi que le besoin d'avoir un jeu d'entraînement très fourni.

Nous avons étudié ce problème pour des *features* discrètes, mais il subsiste dans le cadre continu. Prenons l'exemple de FastText [56], qui représente un mot en un vecteur numérique de taille 300. Si nous modélisons $p(y|x)$ par une gaussienne, il faut alors estimer la moyenne (300 paramètres) et la variance ($301 \times 300/2$ paramètres) par classe. Nous sommes donc confrontés au même problème : il nécessite un *dataset* d'entraînement annoté important pour estimer les paramètres. Nous avons cité FastText qui convertit les mots en vecteurs de taille 300, mais les méthodes les plus récentes le convertissent dans une taille encore plus grande, telles que BERT de taille 784 [57] ou Flair [58, 59] de taille 4096.

Il est possible de supposer les *features* indépendantes conditionnellement à la variable cachée d'intérêt pour simplifier ce problème. Ainsi, le nombre de paramètres à estimer diminuerait et serait traitable. Reprenons l'exemple avec les *features* catégorielles, les supposer indépendantes revient à estimer :

$$\begin{aligned} p\left(y^{(1)}, y^{(2)}, y^{(3)}|\lambda_i\right) &= p\left(y^{(1)}|\lambda_i\right) p\left(y^{(2)}|\lambda_i\right) p\left(y^{(3)}|\lambda_i\right) \\ &= \frac{N(\lambda_i, y^{(1)})}{N(\lambda_i)} \frac{N(\lambda_i, y^{(2)})}{N(\lambda_i)} \frac{N(\lambda_i, y^{(3)})}{N(\lambda_i)}. \end{aligned}$$

Ainsi, une personne traitant ce problème avec cette méthode aurait $(26 + 17576 + 17576) \times 3$ paramètres à estimer. Avec les *features* continues, la matrice de variance-covariance devient diagonale, et le modèle n'a plus que $(300 + 300) \times 3$ paramètres à estimer. En pratique, cette méthode obtient de très faibles résultats. Cela semble cohérent étant donné la très forte hypothèse supposée. Nous pouvons notamment l'illustrer avec l'exemple ci-dessus, qui suppose le suffixe de taille 1 indépendant de celui de taille 3.

Nous avons décrit ce problème d'estimation par comptage, basé sur le maximum de vraisemblance. Néanmoins, le problème subsiste pour toute autre méthode d'estimation pour les classificateurs calculés de manière générative.

Ainsi, nous avons décrit le problème d'estimation de paramètre d'un classificateur calculé de manière générative, qui devient très vite impossible à traiter. De plus, nous ne l'avons illustré qu'avec un cas simple, où nous ne considérions que la loi d'un seul mot. Si nous supposions la loi de deux mots ensemble, le problème se complexifie considérablement, et ainsi de suite.

De leur côté, les classificateurs calculés de manière discriminante ne souffrent pas de ce problème. Ils sont parfaitement capables de gérer les *features* des observations sans supposer l'indépendance entre elles, et peuvent mesurer l'influence d'une combinaison de *features*, même si celle-ci n'a jamais été rencontrée dans le jeu d'entraînement. Le nombre de paramètres à estimer est également raisonnable dans la plupart des cas.

Un grand nombre de travaux, déjà cités dans l'énoncé de la propriété (P), décrivent ce problème incombant aux modèles génératifs. Nous pouvons citer certains d'entre eux, très populaires, comparant des classificateurs calculés de manière générative et discriminante et mettant l'accent sur ce point :

— comparaison du *Naive Bayes* et de la Régression Logistique [40] ;

— comparaison de la chaîne de Markov cachée et du *Maximum Entropy Markov Model* [22] ;

— comparaison de la chaîne de Markov cachée et du *linear-chain Conditional Random Field* [23].

Tous font état de ce problème sur la loi des *features* pour le classificateur calculé de manière générative, et mettent en avant le discriminant, qui s'en affranchit.

De manière unanime, dans un cadre supervisé, les classificateurs calculés de manière discriminante sont considérés comme plus performants que ceux calculés de manière générative. Cela a poussé à mettre de côté ces derniers et les modèles génératifs pour le NLP, car ces tâches impliquent un très grand nombre de *features*, au profit des modèles et classificateurs calculés de manière discriminante. Ainsi, des modèles tels que le *Naive Bayes* et la chaîne de Markov cachée sont mis de côté et peu sollicités depuis un grand nombre d'années pour ces applications.

Remarque 1.5. Il est à noter que les classificateurs calculés de manière générative peuvent toujours se révéler d'une grande utilité, comme cela est le cas dans le cadre non-supervisé. Nous pouvons notamment citer la segmentation d'images avec les modèles de Markov [60, 61, 62].

1.5 Les réseaux de neurones : modèles dominants aujourd'hui pour les tâches de classification

Nous avons décrit les modèles probabilistes, et comment les utiliser pour la classification dans le cadre supervisé via leur classificateur induit. Néanmoins, les modèles dominants aujourd'hui pour ce type de tâche ne reposent pas sur eux,

mais sur une autre catégorie de modèles trouvant ses ressources sur des bases informatiques et computationnelles : les réseaux de neurones. Ils sont très populaires de nos jours, atteignant l'état-de-l'art pour un très grand nombre de tâches, allant du NLP à la vision par ordinateur, en passant par le traitement de signal.

La popularité des réseaux de neurones repose sur leur capacité à s'adapter pour tout type de tâches, ainsi que sur l'apprentissage basé sur la descente de gradient et la *back-propagation* [63, 64]. Il existe de nombreux types de réseaux de neurones, tels que les *Feedforward Neural Networks* (FNN), les réseaux de neurones récurrent (RNN pour *Recurrent Neural Network*), les réseaux de neurones convolutifs, ou encore le *Transformer* [65]. Dans cette section, nous allons décrire brièvement le FNN et le RNN pour les tâches de classification. Plus de détails sur ce domaine sont notamment disponibles dans [66, 67].

1.5.1 Le Feedforward Neural Network

Un FNN est une fonction f de $\mathbb{R}^d$ dans $\mathbb{R}^N$ composée par la succession de transformations linéaires et de fonctions, dites d'activation, dérivables et non-linéaires. Il bénéficie d'un résultat très important : le théorème d'approximation universel [68], démontrant qu'un FNN ayant une seule couche cachée peut approcher n'importe quelle fonction.

Concernant les fonctions d'activation, allant de $\mathbb{R}^{d'}$ dans $\mathbb{R}^{d'}, d' \in \mathbb{N}^*$, nous pouvons notamment citer :

— La fonction *softmax* :

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{d'} e^{x_j}}; \tag{1.7}$$

— la fonction sigmoïde :

$$\sigma(x)_i = \frac{1}{1 + e^{-x_i}}; \tag{1.8}$$

— la fonction *Rectifier Linear Unit* (ReLU) [69] :

$$\text{ReLU}(x)_i = \max(0, x_i); \tag{1.9}$$

— la fonction tangente hyperbolique (tanh) :

$$\tanh(x)_i = \frac{e^{x_i} - e^{-x_i}}{e^{x_i} + e^{-x_i}}; \tag{1.10}$$

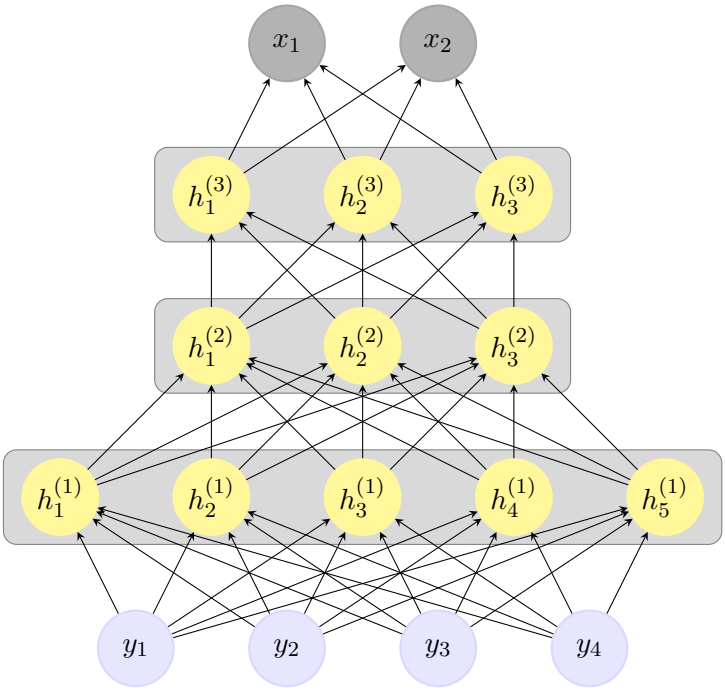


FIGURE 1.5 – Graphe computationnel d'un FNN.

— la fonction *Exponential Linear Unit* (ELU) :

$$ELU(x)_i = \max(e^{x_i} - 1, x_i). \tag{1.11}$$

Exemple 1.5. Soit le FNN suivant :

$$\begin{aligned} f : \mathbb{R}^d \to \mathbb{R}^N, y \to x = \text{softmax}(w^{(4)}h^{(3)} + b^{(4)}) \\ \text{avec } h^{(3)} = \tanh(w^{(3)}h^{(2)} + b^{(3)}), \\ \text{avec } h^{(2)} = \text{ReLU}(w^{(2)}h^{(1)} + b^{(2)}), \\ \text{avec } h^{(1)} = \sigma(w^{(1)}y + b^{(1)}), \end{aligned}$$

avec les paramètres $w^{(1)}, w^{(2)}, w^{(3)}, w^{(4)}$ et $b^{(1)}, b^{(2)}, b^{(3)}, b^{(4)}$ des matrices et vecteurs de tailles adaptés, appelés respectivement poids et biais. Le vecteur y est appelé couche d'entrée, les vecteurs $h^{(1)}, h^{(2)}, h^{(3)}$ sont appelés couches cachées, et x est appelé couche de sortie. Ainsi, le FNN permet de représenter le vecteur d'entrée y sous différentes transformations non-linéaires. Nous supposons que $y \in \mathbb{R}^4, h^{(1)} \in \mathbb{R}^5, h^{(2)} \in \mathbb{R}^3, h^{(3)} \in \mathbb{R}^3, x \in \mathbb{R}^2$. Ce FNN peut être représenté par le graphe computationnel en figure 1.5.

Concernant l'apprentissage des paramètres d'un FNN, il s'effectue dans l'extrême majorité des cas par descente de gradient, utilisant la *back-propagation* et une méthode d'optimisation. Cela est le cas pour toutes les architectures neuronales. La recherche est très active sur ces problématiques visant à optimiser l'apprentissage des paramètres d'un réseau de neurones. Nous pouvons notamment citer les techniques d'annihilation de gradient [70] ou la pondération des poids stochastiques [71].

1.5.2 Les RNN

Le FNN n'est pas capable de gérer les données de tailles variables, comme des textes ou des séries temporelles. Les réseaux de neurones récurrents [72, 73, 74, 75] ont été proposés pour traiter ce type de cas. Pour $T \in \mathbb{N}^*$, un RNN est une fonction considérant comme *input* $y_{1:T}$, avec $y_t \in \mathbb{R}^d$, et produisant comme *output* $h_{1:T}, h_t \in \mathbb{R}^N$, de la manière suivante, pour tout $t \in \{0, ..., T-1\}$:

$$h_{t+1} = f([h_t, y_{t+1}]),$$

avec f un FNN de $\mathbb{R}^{N+d}$ dans $\mathbb{R}^N$, avec h_0 un vecteur initial, souvent tiré aléatoirement ou nul.

Le RNN peut être utilisé pour tout type de tâche impliquant des données de tailles variables, tel que l'étiquetage morphosyntaxique, la traduction de textes, ou encore la reconnaissance d'entités nommées.

Comme tout modèle neuronal, il peut être intégré à une architecture neuronale. Cela signifie que l'*output* de ce modèle peut être l'*input* d'un autre modèle neuronal, ou inversement. À titre d'exemple, il est possible d'avoir l'architecture neuronale suivante :

— les données d'entrée $y_{1:5}$, avec $y_t \in \mathbb{R}^4$;
— $y_{1:5}$ est l'*input* d'un RNN, qui produit en sortie un vecteur $h_{1:5}$, avec $h_t \in \mathbb{R}^6$;
— pour tout t, h_t devient l'*input* d'un même FNN, produisant $x_{1:5}$ avec $x_t \in \mathbb{R}^2$.

Le graphe computationnel de cette architecture est donné en figure 1.6.

Le modèle BiRNN

En utilisant le RNN, le calcul de h_t n'utilise que les données de y_1 à y_t, et non toute la séquence. Pour résoudre ce problème, il est possible d'utiliser le modèle BiRNN [76], pour *Bidirectional* RNN. Ce dernier consiste à appliquer un RNN en considérant la séquence de la gauche vers la droite, et un autre RNN en considérant la séquence de la droite vers la gauche, puis de concaténer les deux *outputs*.

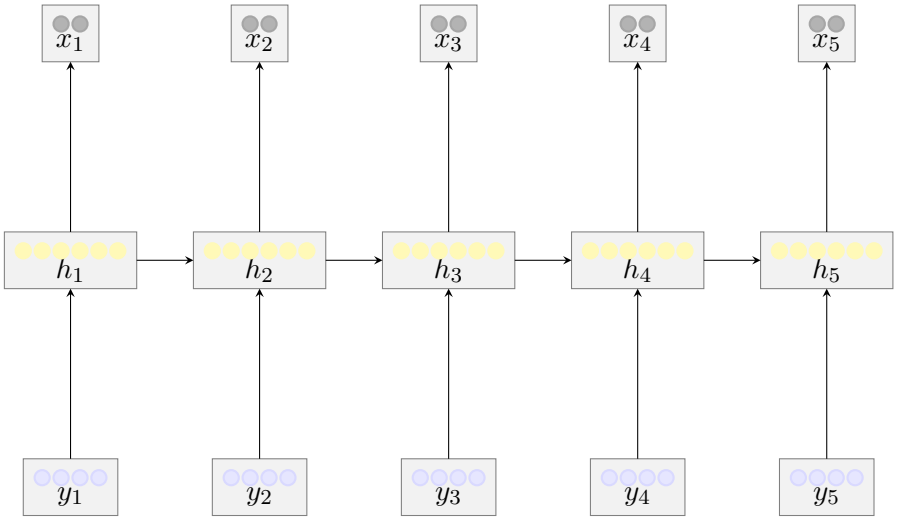


FIGURE 1.6 – Graphe computationnel d'un RNN suivi d'un FNN.

Les modèles LSTM et GRU

Le modèle RNN est également critiqué en raison de sa « mémoire limitée ». En effet, l'influence de y_t sur h_{t+k} décroît de manière exponentielle en fonction de k. Pour résoudre cela, de nombreux modèles ont été proposés. Nous citons les plus connus : le *Long-Short Term Memory* (LSTM) *network* [77], et le *Gated Reccurent Unit* (GRU) [78, 79]. Nous pouvons également citer le mécanisme d'attention [80] qui permet de résoudre le problème de mémoire dans le cadre des RNN, et qui est le précurseur des modèles basés sur le Transformer [65] atteignant l'état-de-l'art pour la plupart des tâches de NLP [57, 81, 82].

1.6 Les différentes tâches du Traitement des Langues Naturelles

Le NLP fait partie des tâches de *Machine Learning* ayant connu le plus grand essor au cours de ces dernières années. Portées notamment par les réseaux de neurones, un grand nombre de tâches liées à ce domaine atteignent des performances supérieures à l'humain aujourd'hui. Dans cette section, nous allons décrire les tâches qui seront rencontrées tout au long de cette thèse, ainsi que les jeux de données utilisées. Ces derniers sont tous disponibles gratuitement avec la librairie *datasets* [83] en Python.

1.6.1 Convertir un mot en vecteur numérique avec les méthodes de Word Embedding

Un grand nombre de modèles, comme ceux basés sur des réseaux de neurones, nécessitent des données numériques en *input*. Dans la mesure où un texte est une chaîne de caractère, l'une des premières étapes consiste à convertir les mots en vecteurs numériques. Ainsi, pour un texte contenant K mots, celui-ci sera converti en K vecteurs de taille d, avec $d \in \mathbb{N}^*$ dépendant de la méthode utilisée. Ces méthodes sont appelés méthodes de *word embedding* [84, 85, 86, 87], et peuvent être séparé en deux catégories : les méthodes non-contextuelles et les méthodes contextuelles.

Les méthodes d'embedding non-contextuelles

Pour les méthodes non-contextuelles, l'algorithme ne prend pas en compte les autres mots du texte pour représenter un mot. Ainsi, un même mot dans deux textes différents sera représenté par le même vecteur. Ces méthodes nécessitent en général moins de puissance de calcul mais entraînent des performances plus faibles.

À titre d'exemple, l'une des méthodes non-contextuelles les plus intuitives est le *one-hot encoding*. Cette méthode consiste à représenter chaque mot par un vecteur de taille V (la taille du vocabulaire) composé de 0, avec un unique 1 dans une position spécifique liée à ce mot. Les mots ne faisant pas partie du vocabulaire, considérés comme inconnus, sont représentés par un vecteur nul de taille V. Il est possible de concaténer des méthodes de *one-hot encoding* pour ajouter des *features* et ainsi considérer plus de mots inconnus.

Exemple 1.6. Soit le vocabulaire *{data, science, machine, learning}*. Le *one-hot encoding* représenterait chaque mot ainsi :

- $data = [1, 0, 0, 0]$;
- $science = [0, 1, 0, 0]$;
- $machine = [0, 0, 1, 0]$;
- $learning = [0, 0, 0, 1]$;

et tout autre mot serait représenté par $[0, 0, 0, 0]$.

Si nous souhaitons ajouter également les suffixes de taille 2, qui seront représentés ainsi : $ta = [1, 0, 0, 0]$, $ce = [0, 1, 0, 0]$, $ne = [0, 0, 1, 0]$, et $ng = [0, 0, 0, 1]$, alors cette méthode représente le vocabulaire ainsi :

- $data = [1, 0, 0, 0, 1, 0, 0, 0]$;
- $science = [0, 1, 0, 0, 0, 1, 0, 0]$;

— *machine* = $[0, 0, 1, 0, 0, 0, 1, 0]$;
— *learning* = $[0, 0, 0, 1, 0, 0, 0, 1]$.

Ainsi, le mot *evening* sera représenté $[0, 0, 0, 0, 0, 0, 0, 1]$ et le mot *computer* $[0, 0, 0, 0, 0, 0, 0, 0]$.

Il existe un grand nombre d'autres méthodes d'*embedding* non-contextuelles. Nous en utiliserons trois parmi les plus connues :

— GloVe [88], qui convertit les mots en vecteurs de taille 100 ;
— FastText [56], qui convertit les mots en vecteurs de taille 300 ;
— ExtVec [89], qui convertit les mots en vecteurs de taille 300, différents de FastText.

Leurs fonctionnements, propre à chaque algorithmes d'apprentissage, dépassent le cadre de la thèse et ne sont pas décrits.

Les méthodes d'embedding contextuelles

Pour les méthodes contextuelles, l'algorithme prend en compte les autres mots du texte pour représenter un mot. Ainsi, un même mot dans deux textes différents sera représenté par deux vecteurs différents, dépendant de ses voisins. Ces méthodes nécessitent en général plus de puissance de calcul mais atteignent des performances plus élevées.

Au cours de cette thèse seront utilisées les méthodes suivantes :

— Flair [58, 59], donnant des vecteurs de taille 4096, basé sur des réseaux de neurones BiLSTM ;
— BERT [57], donnant des vecteurs de taille 784, basé sur le modèle neuronal Transformer [65].

Les modèles basés sur ce type d'*embedding* font l'état-de-l'art pour toutes les tâches de NLP [90, 91, 92, 93]. Malgré leur efficacité, elles sont la cible de critiques dues à leur coût de mise en production et leur impact sur l'environnement [94].

1.6.2 Classification de textes et analyse de sentiments

La classification de textes [101] consiste à affecter un label à un texte. Les classes cibles dépendent de l'objectif et peuvent s'appliquer à tout type de textes. Ainsi, cela regroupe la classification de mail, d'actualités, ou encore d'articles scientifiques. L'un des *dataset* de référence est AG News, décrit en figure 1.7.

Le *dataset* AG News

Le *dataset* AG News fait partie des jeux de données de référence pour la classification de textes. Il affecte à des articles issus de journaux d'informations l'une des quatre classes suivantes : *(World, Sports, Business, Sci/Tech)* [95]. Le *dataset* d'entraînement est composé de 120000 textes, celui de test de 7600 textes. Nous utilisons une portion de son ensemble d'entraînement pour créer le *dataset* de validation quand le besoin en est. Son ensemble de test comporte 0.97% de mots inconnus, répartis dans 25.89% des textes.

FIGURE 1.7 – Description du *dataset* AG News.

Le *dataset* IMDB

L'analyse de sentiments avec le *dataset* IMDB [96] consiste à catégoriser chaque texte issu de revues de films du site internet IMDB selon deux sentiments : *Positive* ou *Negative*. Il est l'un des jeux de données de référence pour cette tâche, avec un jeu d'entraînement composé de 25000 textes, et celui de test de 25000 textes également. Nous utilisons une portion de son ensemble d'entraînement pour créer le *dataset* de validation quand le besoin en est. Son ensemble de test comporte 1.18% de mots inconnus, répartis dans 75.21% des textes.

FIGURE 1.8 – Description du *dataset* IMDB.

L'analyse de sentiments est un cas particulier de classification de textes, où les labels concernent notamment les catégories liées aux sentiments ou aux émotions. À titre d'exemple, nous pouvons citer l'analyse de revues de films, qui peut être positive ou négative, comme avec le *dataset* IMDB décrit en figure 1.8, ou le *dataset* SST-2 décrit en figure 1.9. L'analyse de sentiments s'effectue également sur les tweets, comme avec le *dataset* TweetEval - Emotion, décrit en figure 1.10, ou encore sur des actualités financières avec le *dataset* Financial PhraseBank (FPB) décrit en 1.11.

Ces tâches sont évaluées en calculant le pourcentage de textes correctement prédits sur le jeu de test, appelé *accuracy* :

$$acc = \frac{\text{Nombre de textes correctement prédits}}{\text{Nombre de textes total}}.$$

Le *dataset* SST-2

Le *dataset* Stanford Sentiment Treebank, appelé communément SST-2 [97], est un *dataset* pour l'analyse de sentiment classifiant des revues de films sous les classes *Positive* et *Negative*. Il est constitué d'un ensemble d'entraînement de 67349 textes, de validation de 872 textes, et de test de 1821 textes. Il fait partie de l'ensemble de tâches GLUE [98], considéré comme la référence pour évaluer un modèle de langage sur diverses tâches. Le *dataset* de test de SST-2 comporte 5.49% de mots inconnus, répartis dans 57.17% des textes.

FIGURE 1.9 – Description du *dataset* SST-2.

Le *dataset* TweetEval - Emotion

Le *dataset* TweetEval - Emotion [99] consiste à catégoriser des tweets selon quatre émotions différentes : *Anger, Joy, Sadness, Optimism*. Il est constitué de 3257 tweets pour l'entraînement, 374 pour la validation, et 1421 pour le test. Ce dernier comporte 10.77% de mots inconnus, répartis dans 78.61% des tweets.

FIGURE 1.10 – Description du *dataset* TweetEval - Emotion.

Le taux d'erreur peut également être utilisé, valant $1 - acc$. Le score F_1, que nous décrirons dans la section suivante, peut également être utilisé, notamment pour les *datasets* TweetEval - Emotion et FPB.

1.6.3 La segmentation de textes : Part-Of-Speech Tagging, Chunking, et Named-Entity-Recognition

La segmentation de textes consiste à affecter un label à chaque mot, ou chaque groupe de mots, d'un texte. Il existe trois tâches principales :

— l'étiquetage morphosyntaxique, ou *Part-Of-Speech Tagging* (POS Tagging) ;

— le *Chunking* ;

— la détection d'entités, plus connue sous le nom de *Named-Entity-Recognition* (NER).

Nous allons décrire brièvement ces trois tâches, les personnes intéressées peuvent approfondir ces sujets en consultant les chapitre 5 et 7 du livre *NLTK* [102].

Le *dataset* Financial PhraseBank

Le *dataset* Financial Phrasebank (FPB) [100] consiste à catégoriser des phrases pour savoir si elles ont un effet positif, négatif, ou neutre en termes d'influence sur le marché financier. Il a été annoté par des professionnels du domaine. Nous sélectionnons uniquement les phrases où les annotateurs ont tous convenu du même sentiment, ce qui représente 2264 phrases. Nous décomposons cet ensemble pour créer les *datasets* d'entraînement, de validation, et de test.

FIGURE 1.11 – Description du *dataset* Financial Phrasebank.

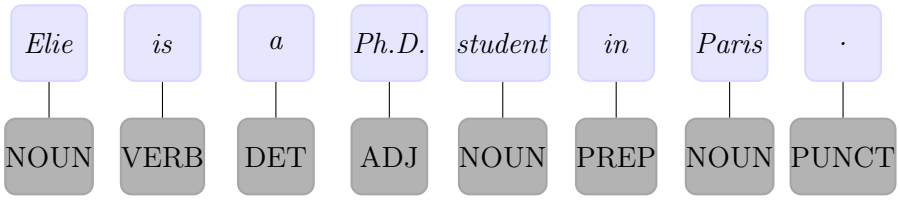
Le *dataset* Universal Dependencies English

Le *dataset* Universal Dependencies English [103] (UD English) est un *dataset* pour le POS Tagging constitué de 12543 phrases d'entraînement, de 2002 phrases de validation, et 2077 phrases de test. Son ensemble cible comporte 17 tags. Les phrases de test sont composées de 8.36% de mots inconnus.

FIGURE 1.12 – Description du *dataset* UD English.

POS Tagging

Le POS Tagging consiste à retrouver la fonction grammaticale de chaque mot au sein d'une phrase. À titre d'exemple :



L'ensemble des labels cible dépend du jeu de données utilisé, comme les *universal pos tags* [103] ou les *Penn tags* [106]. Il est possible d'effectuer cette tâche sur le *dataset* UD English, décrit en figure 1.12. Elle est évaluée en calculant l'*accuracy*, consistant à calculer le pourcentage de mots correctement prédits sur l'ensemble du jeu de test, le taux d'erreur est également parfois utilisé.

Chunking

Le *Chunking* consiste à décomposer une phrase de manière syntaxique, réunissant chaque groupe de mots par leur fonction. À titre d'exemple :

Le *dataset* CoNLL 2000

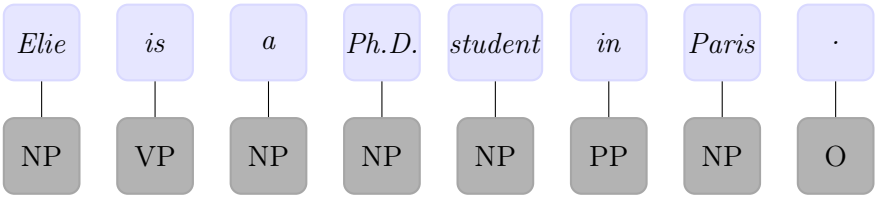
Le *dataset* CoNLL 2000 [104] est le *dataset* de référence pour le *Chunking*, avec un ensemble de *chunk tags* de taille 12. Il fournit également les labels pour le POS Tagging. Il est constitué de 8936 phrases d'entraînement et de 2012 phrases de test. Les phrases de test sont composées de 6.97% de mots inconnus.

FIGURE 1.13 – Description du *dataset* CoNLL 2000.

Le *dataset* CoNLL 2003

Le *dataset* CoNLL 2003 [105] est le *dataset* de référence pour la NER, avec l'ensemble cible *(PER, LOC, ORG, MISC, O)*. Il fournit également les labels pour le POS Tagging et le *Chunking*. Il est constitué de 14986 phrases d'entraînement, de 3465 phrases de validation, et 3683 phrases de test. Les phrases de test sont composées de 11.12% de mots inconnus.

FIGURE 1.14 – Description du *dataset* CoNLL 2003.



Ainsi, dans cet exemple, *a Ph.D. student* est un groupe de mots lié à un nom, et *in* est un groupe de mot, composé d'un seul mot, lié à une préposition. *O* signifie que le mot n'est lié à aucune fonction syntaxique. Le *dataset* de référence pour cette tâche est CoNLL 2000, décrit en 1.13.

Cette tâche est évaluée en calculant le score F_1 sur le jeu de test :

$$F_1 = \frac{2 \times precision \times recall}{precision + recall},$$

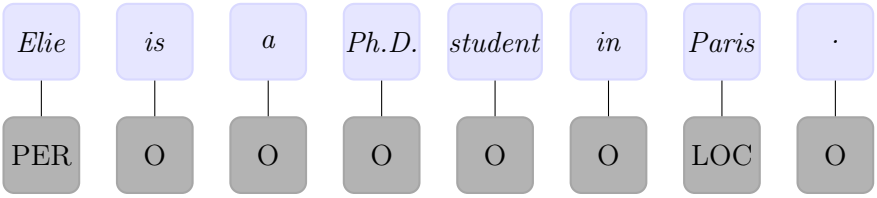
avec :

— $precision = \frac{\text{Nombre de mots différents de } O \text{ correctement prédits}}{\text{Nombre de mots prédits différents de } O}$;

— $recall = \frac{\text{Nombre de mots différents de } O \text{ correctement prédits}}{\text{Nombre de mots différents de } O}$.

Named-Entity-Recognition

La NER consiste à trouver les différentes entités au sein d'un texte. Cela peut être des entités d'ordre général, comme des noms de personnes, de lieux, ou d'entreprises, comme avec le *dataset* CoNLL 2003 décrit en figure 1.14, considéré comme la référence pour cette tâche. Les entités à retrouver peuvent être plus spécifiques, comme des entités médicales [107]. En reprenant les entités générales de CoNLL 2003, nous pouvons avoir comme exemple :



avec *Elie* le nom d'une personne et *Paris* celui d'un lieu, *O* signifiant que ce mot n'est pas une entité. Tout comme le *Chunking*, cette tâche est évaluée avec le score F_1 sur le jeu de test.

Ces tâches sont très étudiées dans la littérature et traitées via des approches variées et pertinentes [108, 109, 110]. Nous avons décrit ces trois familles de tâches liées au NLP, et il en existe bien d'autres, comme le *Question-Answering* [111, 112] ou encore les travaux sur le dialogue des agents conversationnels [113, 114, 115, 116].

1.7 Problématiques

Cette thèse va se structurer autour de deux problématiques que nous traiterons tout au long de ce manuscrit :

Problématique 1 Le problème lié aux *features* a poussé à négliger les modèles probabilistes génératifs pour les tâches de classification avec apprentissage supervisé, notamment pour les tâches de NLP, en raison du fait que le classificateur induit devrait prendre en compte la loi des observations. Ainsi, modéliser des observations issues de vecteurs d'*embedding* de très grandes tailles avec des méthodes récentes ou *one-hot* avec des *features* s'avère contrait à traiter, atteignant des performances limitées. Cependant, ne serait-il pas possible de définir le classificateur issu d'un modèle génératif sans devoir modéliser la loi des observations, c'est-à-dire un classificateur issu d'un modèle génératif peut-il être discriminant, et ce, tout en gardant les dépendances simplifié pouvant survenir dans l'écriture générative d'un modèle ? Et si tel est le cas, qu'en est-il des apports en termes d'applications ? Cette problématique sera traitée dans les chapitres 2, 3, et 4.

Problématique 2 Les réseaux de neurones dominent grandement les principales tâches de classification. À la lumière du résultat précédent, serait-il possible de développer une méthode originales liant modèles probabilistes et réseaux de neurones ? Et que cela va-t-il apporter ? Cette problématique sera traitée au chapitre 5.

Chapitre 2

Le classificateur induit du modèle Naive Bayes

Dans ce chapitre, nous rappelons la définition du modèle *Naive Bayes* et de son classificateur calculé de manière générative, qui est usuellement utilisé. Nous allons montrer comment ce classificateur peut être calculé de manière discriminante. Sous la lumière de ce résultat, nous allons comparer le *Naive Bayes* avec la Régression Logistique, ainsi qu'avec le champ de Markov conditionnel, abrégé CRF pour *Conditional Random Fields*, inspiré par le même graphe probabiliste, mais non-orienté. En fin de chapitre, nous étudions empiriquement l'apport de cette nouvelle forme du classificateur sur la forme générative pour la classification de textes avec le *dataset* AG News et l'analyse de sentiments avec le *dataset* IMDB.

2.1 Présentation du Naive Bayes

Le *Naive Bayes* est un modèle probabiliste liant une unique variable aléatoire cachée X dont la réalisation est à valeurs dans $\Lambda_X = \{\lambda_1, ..., \lambda_N\}$ avec des variables aléatoires observées $Y_{1:T} = (Y_1, ..., Y_T)$, avec pour tout t, Y_t à valeurs dans $\Omega_Y^{(t)}$. Ce modèle est caractérisé par l'indépendance conditionnelle des variables aléatoires observées étant donnée la variable cachée, avec sa loi jointe pouvant s'écrire de la manière suivante :

$$\begin{aligned} p(x, y_{1:T}) &= p(x)p(y_1|x)p(y_2|x)...p(y_T|x) \\ &= p(x)\prod_{t=1}^{T} p(y_t|x). \end{aligned} \tag{2.1}$$

La représentation graphique du *Naive Bayes* est donnée en figure 2.1.

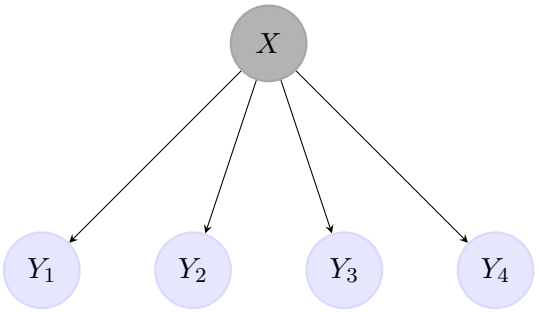


FIGURE 2.1 – Graphe probabiliste orienté du *Naive Bayes*.

Ainsi, pour tout $\lambda_i \in \Lambda_X$, la loi a posteriori du *Naive Bayes*, servant notamment à définir le classificateur bayésien du MAP, peut s'écrire :

$$\begin{aligned} P(X = \lambda_i | Y_{1:T} = y_{1:T}) &= \frac{P(X = \lambda_i, Y_{1:T} = y_{1:T})}{\sum_{\lambda_j \in \Lambda_X} P(X = \lambda_j, Y_{1:T} = y_{1:T})} \\ &= \frac{P(X = \lambda_i) \prod_{t=1}^{T} P(Y_t = y_t | X = \lambda_i)}{\sum_{\lambda_j \in \Lambda_X} P(X = \lambda_j) \prod_{t=1}^{T} P(Y_t = y_t | X = \lambda_j)} \\ &= \frac{\pi(i) \prod_{t=1}^{T} b_i^{(t)}(y_t)}{\sum_{\lambda_j \in \Lambda_X} \pi(j) \prod_{t=1}^{T} b_j^{(t)}(y_t)}. \end{aligned} \tag{2.2}$$

en utilisant les notations :

— $\pi(i) = P(X = \lambda_i)$;

— $b_i^{(t)}(y) = P(Y_t = y | X = \lambda_i)$.

Par conséquent, nous pouvons définir son classificateur calculé de manière générative :

$$\phi^{NB} : (\Omega_Y)^T \to \Lambda_X, y_{1:T} \to \arg\max_{\lambda_i \in \Lambda_X} \left(\frac{\pi(i) \prod_{t=1}^{T} b_i^{(t)}(y_t)}{\sum_{\lambda_j \in \Lambda_X} \pi(j) \prod_{t=1}^{T} b_j^{(t)}(y_t)} \right). \tag{2.3}$$

Dans le cadre du NLP, ce classificateur est notamment utilisé en classification de textes [117] ou encore pour l'analyse de sentiments [39]. Il est également utilisé pour d'autres applications [117, 39, 118, 40] qui l'utilisent toutes via son classificateur calculé de manière générative (2.3). Cela lui vaut un certain nombre de critiques. En effet, ses performances sont considérées comme limitées à cause de la problématique de l'apprentissage de la loi des observations et de la gestion des *features*, empêchant de prendre en compte efficacement les caractéristiques complexes des observations. Ce point est décrit en détail en section 1.4.3.

Outre ce défaut, le *Naive Bayes* est également critiqué car il suppose l'indépendance, conditionnellement sachant X, entre les variables aléatoires $Y_1, ..., Y_T$. Dans le cas du *Naive Bayes* stationnaire, c'est-à-dire lorsque les différents paramètres ne dépendent pas de t, cela implique que le classificateur ne prend pas en compte l'ordre des observations. À titre d'exemple, prenons la tâche consistant à savoir si une personne apprécie ou non un film à partir d'un commentaire, et les deux phrases suivantes :

— *J'adore ce film bien que je n'aime pas l'acteur principal.*

— *Je n'aime pas ce film bien que j'adore l'acteur principal.*

Dans la mesure où elles comportent exactement les mêmes mots, elles sont perçues comme identiques par le *Naive Bayes* stationnaire alors que leur label est différent, illustrant ce défaut inhérent à ce modèle.

2.2 Le classificateur du Naive Bayes calculé de manière discriminante

Pour simplifier les notations, nous posons, pour tout $\lambda_i \in \Lambda_X$ et $y_t \in \Omega_Y$:

$$L_{y_t}^{(t)}(i) = P(X = \lambda_i | Y_t = y_t).$$

Proposition 2.1. Pour tout $\lambda_i \in \Lambda_X$, la loi a posteriori du *Naive Bayes* peut s'écrire de la manière suivante :

$$P(X = \lambda_i | Y_{1:T} = y_{1:T}) = \frac{\pi(i)^{1-T} \prod_{t=1}^{T} L_{y_t}^{(t)}(i)}{\sum_{j=1}^{N} \pi(j)^{1-T} \prod_{t=1}^{T} L_{y_t}^{(t)}(j)}. \tag{2.4}$$

Démonstration Pour tout $\lambda_i \in \Lambda_X$, nous notons :

$$\delta_{y_{1:T}}(i) = \pi(i)^{1-T} \prod_{t=1}^{T} L_{y_t}^{(t)}(i). \tag{2.5}$$

La loi jointe est reliée à cette quantité ainsi :

$$P(X = \lambda_i, Y_{1:T} = y_{1:T}) = \delta_{y_{1:T}}(i) \prod_{t=1}^{T} p(y_t). \tag{2.6}$$

En effet,

$$\begin{aligned} P(X = \lambda_i, Y_{1:T} = y_{1:T}) &= \pi(i) \prod_{t=1}^{T} b_i^{(t)}(y_t) \\ &= \pi(i) \prod_{t=1}^{T} \frac{P(Y_t = y_t, X = \lambda_i)}{P(X = \lambda_i)} \\ &= \pi(i)^{1-T} \left[\prod_{t=1}^{T} p(y_t) L_{y_t}^{(t)}(i) \right] \\ &= \delta_{y_{1:T}}(i) \prod_{t=1}^{T} p(y_t). \end{aligned}$$

Par conséquent

$$P(X = \lambda_i | Y_{1:T} = y_{1:T}) = \frac{P(X = \lambda_i, Y_{1:T} = y_{1:T})}{\sum_{j=1}^{N} P(X = \lambda_j, Y_{1:T} = y_{1:T})} = \frac{\delta_{y_{1:T}}(i)}{\sum_{j=1}^{N} \delta_{y_{1:T}}(j)}, \tag{2.7}$$

achevant la démonstration.

Le classificateur induit du *Naive Bayes* peut donc s'écrire :

$$\phi^{NB} : (\Omega_Y)^T \to \Lambda_X, y_{1:T} \to \arg\max_{\lambda_i \in \Lambda_X} \left(\frac{\pi(i)^{1-T} \prod_{t=1}^{T} L_{y_t}^{(t)}(i)}{\sum_{j=1}^{N} \pi(j)^{1-T} \prod_{t=1}^{T} L_{y_t}^{(t)}(j)} \right). \tag{2.8}$$

Ainsi écrit, ce classificateur ne calcule aucune loi de la forme $p(y_t|x)$, donc correspond à un classificateur calculé de manière discriminante. Il ne connaît plus le problème des *features* et peut considérer n'importe quelle observation sans contraintes.

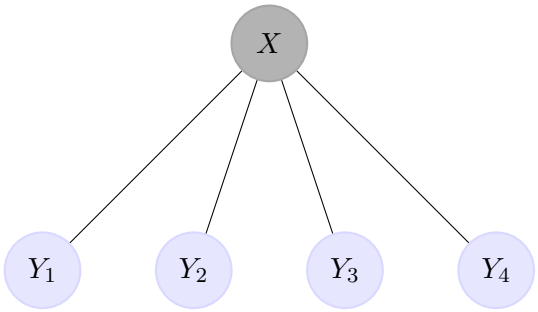


FIGURE 2.2 – Graphe probabiliste de la Régression Logistique.

Ce résultat est d'importance, car nous avons montré que le classificateur du modèle *Naive Bayes* peut être calculé à la fois de manière générative avec (2.3) et de manière discriminante avec (2.8). Cela montre que cette catégorisation des classificateurs, supposée disjointe, n'est pas pertinente dans ce cas.

Remarque 2.1. La loi $p(x, y_{1:T})$ du *Naive Bayes* peut donc s'écrire :

$$p(x, y_{1:T}) = p(x) \prod_{t=1}^{T} p(y_t|x) = p(x)^{1-T} \prod_{t=1}^{T} p(y_t) \prod_{t=1}^{T} p(x|y_t).$$

Ainsi, en notant, pour tout $y \in \Omega_Y, O^{(t)}(y) = P(Y_t = y)$, il est possible de définir un modèle *Naive Bayes* avec deux jeux de paramètres :

1. celui usuellement utilisé : (π, b) ;
2. celui induit par la nouvelle formule du classificateur : (π, O, L).

Si le besoin est d'apprendre ces paramètres pour définir le classificateur de manière supervisée, le premier ensemble souffre du problème de la gestion des *features* car il doit apprendre la loi des observations. Le second jeu, dans la mesure où le paramètre O n'est pas requis pour définir le classificateur, ne connaît pas ce problème.

2.3 Comparaison de la Régression Logistique et du Naive Bayes

La Régression Logistique Multinomiale, que nous désignons par Régression Logistique tout au long de ce manuscrit, est un modèle probabiliste discriminant considérant, tout comme le *Naive Bayes*, une variable aléatoire cachée X et des

variables aléatoires observées $Y_{1:T} = (Y_1, ..., Y_T)$, avec la réalisation de Y_t à valeurs dans $\mathbb{R}$. Elle est définie par la loi a posteriori :

$$P(X = \lambda_i | Y_{1:T} = y_{1:T}) = \frac{\exp\left(w_{1:T}^{(i)} \cdot y_{1:T} + b_i\right)}{\sum_{j=1}^{N} \exp\left(w_{1:T}^{(j)} \cdot y_{1:T} + b_j\right)}, \tag{2.9}$$

avec, pour tout $i \in \{1, \ldots, N\}, w_{1:T}^{(i)} \in \mathbb{R}^T$, $b_i \in \mathbb{R}$, et $\cdot$ désigne le produit scalaire entre deux vecteurs. Son graphe probabiliste non-orienté peut être représenté en figure 2.2.

Ainsi, ce modèle permet de définir le classificateur calculé de manière discriminante :

$$\phi^{LR} : \mathbb{R}^T \to \Lambda_X, y_{1:T} \to \arg\max_{\lambda_i \in \Lambda_X} \left(\frac{\exp\left(w_{1:T}^{(i)} \cdot y_{1:T} + b_i\right)}{\sum_{j=1}^{N} \exp\left(w_{1:T}^{(j)} \cdot y_{1:T} + b_j\right)} \right). \tag{2.10}$$

La Régression Logistique et le *Naive Bayes* sont très comparés dans la littérature, caractérisés comme une *Generative-Discriminative pair* [24, 40, 119, 120, 121, 122, 123, 124, 125, 126]. Cette comparaison est toujours basée sur la nature de leur classificateur : générative pour le *Naive Bayes*, discriminante pour la Régression Logistique. Ainsi, à cause du problème lié aux *features* et au fait d'apprendre la loi des observations, la Régression Logistique est préférée dans la majeure partie des cas. Cependant, nous avons montré que se baser sur la nature du classificateur n'est pas justifié dans la mesure où celui du *Naive Bayes* peut être calculé de manière discriminante. Par conséquent, cela remet en question les comparaisons de ces modèles basées sur leur classificateur, ce que nous allons voir en détail dans la suite cette section.

Proposition 2.2. Soient les variables aléatoires X et $Y_{1:T}$ dont la loi a posteriori est celle d'une Régression Logistique (2.9). Alors (2.9) est également la loi a posteriori d'un *Naive Bayes* $p(x, y_{1:T})$ particulier défini ainsi :

$$P(X = \lambda_i) = \frac{1}{N}, \tag{2.11}$$

$$\text{pour tout } t, p(y_t) \text{ arbitraire}, \tag{2.12}$$

$$\text{pour tout } t, P(X = \lambda_i | Y_t = y_t) = \frac{\exp\left(w_t^{(i)} y_t + c_t^{(i)}\right)}{\sum_{j=1}^{N} \exp\left(w_t^{(j)} y_t + c_t^{(j)}\right)}, \tag{2.13}$$

$$\text{avec } c_1^{(i)} = b_i, \text{ et pour tout } t = 2, ..., T, c_t^{(i)} = 0. \tag{2.14}$$

Démonstration Soit le modèle *Naive Bayes* défini avec (2.11)-(2.14) :

$$\begin{aligned}
P(X = \lambda_i, Y_{1:T} = y_{1:T}) &= P(X = \lambda_i) \prod_{t=1}^{T} P(Y_t = y_t) \frac{P(X = \lambda_i | Y_t = y_t)}{P(X = \lambda_i)} \\
&= \left(\frac{1}{N}\right)^{1-T} \prod_{t=1}^{T} P(Y_t = y_t) \prod_{t=1}^{T} \frac{\exp\left(w_t^{(i)} y_t + c_t^{(i)}\right)}{\sum_{j=1}^{N} \exp\left(w_t^{(j)} y_t + c_t^{(j)}\right)} \\
&= \left(\frac{1}{N}\right)^{1-T} \prod_{t=1}^{T} P(Y_t = y_t) \frac{\exp\left(\sum_{t=1}^{T} w_t^{(i)} y_t + c_1^{(i)}\right)}{\prod_{t=1}^{T} \sum_{j=1}^{N} \exp\left(w_t^{(j)} y_t + c_1^{(j)}\right)} \\
&= \frac{\left(\frac{1}{N}\right)^{1-T} \prod_{t=1}^{T} P(Y_t = y_t)}{\prod_{t=1}^{T} \sum_{j=1}^{N} \exp\left(w_t^{(j)} y_t + c_1^{(j)}\right)} \exp\left(w_{1:T}^{(i)} \cdot y_{1:T} + b_i\right).
\end{aligned}$$

Donc

$$P(X = \lambda_i | Y_{1:T} = y_{1:T}) = \frac{\exp\left(w_{1:T}^{(i)} \cdot y_{1:T} + b_i\right)}{\sum_{j=1}^{N} \exp\left(w_{1:T}^{(j)} \cdot y_{1:T} + b_j\right)},$$

qui est également la loi a posteriori d'une Régression Logistique, achevant la démonstration.

Par conséquent, pour tout classificateur issu d'une Régression Logistique, il existe un modèle *Naive Bayes* particulier dont il peut également être induit. Ainsi, comparer ces deux modèles sur la base de leur classificateur n'est pas pertinent. Cela montre également que, du point de vue de leurs classificateurs respectifs, le *Naive Bayes* est plus général que la Régression Logistique.

2.4 Cas du CRF avec le graphe probabiliste basé sur le Naive Bayes

Après avoir démontré que le classificateur induit de la Régression Logistique est un cas particulier de celui induit du *Naive Bayes*, nous allons plus loin dans la

comparaison entre ce dernier et modèles discriminants. Nous introduisons le CRF supposant les mêmes dépendances entre variables aléatoires que le *Naive Bayes*. L'objectif est de déterminer lequel des deux modèles est le plus général du point de vue de leur classificateur.

Soit le CRF dont le graphe probabiliste est similaire à celui du *Naive Bayes*, mais sans orientation des arcs. Il correspond donc au graphe représenté en figure 2.2. La loi a posteriori de ce CRF, que nous dénommons NB-CRF, s'écrit ainsi :

$$p(x|y_{1:T}) = \frac{\prod_{t=1}^{T} T^{(t)}(x, y_t)}{\sum_{x' \in \Lambda_X} \prod_{t=1}^{T} T^{(t)}(x', y_t)}, \tag{2.15}$$

avec $T^{(t)}$ des fonctions potentielles.

Remarque 2.2. La Régression Logistique en est un cas particulier, dont l'expression des fonctions potentielles est, pour tout t et λ_i :

$$T^{(t)}(X = \lambda_i, Y_t = y_t) = e^{w_t^{(i)} y_t + b_{it}},$$

avec $w_t^{(i)} \in \mathbb{R}$ et $b_{it} \in \mathbb{R}$.

Proposition 2.3. Soit les variables aléatoires X et $Y_{1:T}$ telles que $p(x, y_{1:T})$ soit un *Naive Bayes* (2.1). Alors, sa loi a posteriori est celle d'un NB-CRF défini ainsi :

$$T^{(1)}(x, y_1) = p(x|y_1), \tag{2.16}$$

$$\text{pour tout } 2 \leq t \leq T, T^{(t)}(x, y_t) = \frac{p(x|y_t)}{p(x)}. \tag{2.17}$$

Démonstration La loi a posteriori du *Naive Bayes* peut s'écrire :

$$p(x|y_{1:T}) = \frac{p(x)^{1-T} \prod_{t=1}^{T} p(x|y_t)}{\sum_{x'} p(x')^{1-T} \prod_{t=1}^{T} p(x'|y_t)}.$$

Pour sa part, la loi a posteriori du NB-CRF défini avec (2.16) et (2.17) vaut :

$$p(x|y_{1:T}) = \frac{p(x|y_1) \prod_{t=2}^{T} \frac{p(x|y_t)}{p(x)}}{\sum_{x'} p(x'|y_1) \prod_{t=2}^{T} \frac{p(x'|y_t)}{p(x')}}$$

$$= \frac{p(x)^{1-T} \prod_{t=1}^{T} p(x|y_t)}{\sum_{x'} p(x')^{1-T} \prod_{t=1}^{T} p(x'|y_t)},$$

achevant notre démonstration.

Proposition 2.4. Soient les variables aléatoires X et $Y_{1:T}$ telles que leur loi a posteriori soit un NB-CRF (2.15). Alors (2.15) est la loi a posteriori d'un *Naive Bayes* défini ainsi :

$$p(x) = \frac{1}{N}, \tag{2.18}$$

$$\text{pour tout } t, p(y_t) \text{ arbitraire}, \tag{2.19}$$

$$p(x|y_t) = \frac{T^{(t)}(x, y_t)}{\sum_{x'} T^{(t)}(x', y_t)}. \tag{2.20}$$

Démonstration La loi a posteriori du *Naive Bayes* défini avec (2.18)-(2.20) vaut :

$$p(x|y_{1:T}) = \frac{\left(\frac{1}{N}\right)^{1-T} \prod_{t=1}^{T} T^{(t)}(x, y_t)}{\sum_{x'} \left(\frac{1}{N}\right)^{1-T} \prod_{t=1}^{T} T^{(t)}(x', y_t)}$$

$$= \frac{\prod_{t=1}^{T} T^{(t)}(x, y_t)}{\sum_{x'} \prod_{t=1}^{T} T^{(t)}(x', y_t)},$$

achevant la démonstration.

Par conséquent, nous avons montré qu'étant donné un classificateur issu d'un NB-CRF, celui-ci peut être induit d'un *Naive Bayes*, et vice-versa. Ainsi, aucun de ces deux modèles n'est supérieur à l'autre du point de vue de son classificateur.

Définition 2.1. Nous introduisons une relation entre l'ensemble des modèles probabilistes définis par une même loi jointe, noté $\mathcal{M}_{GEN}$, et l'ensemble des modèles probabilistes définis par une même loi a posteriori, noté $\mathcal{M}_{DIS}$. Ils sont dits équivalents si :

— étant donné la loi a posteriori d'un modèle appartenant à $\mathcal{M}_{DIS}$, il est possible de définir la loi jointe d'un modèle appartenant à $\mathcal{M}_{GEN}$ ayant la même loi a posteriori,

— et si étant donné la loi jointe d'un modèle appartenant à $\mathcal{M}_{GEN}$, sa loi a posteriori permet de définir un modèle appartenant à $\mathcal{M}_{DIS}$.

Exemple 2.1. Les ensembles des modèles *Naive Bayes* et NB-CRF sont équivalents, comme démontré avec les propositions 2.3. et 2.4.

2.5 Applications : classification de textes et analyse de sentiments avec le Naive Bayes

Dans cette partie, nous allons illustrer l'importance de la prise en compte des *features* arbitraires avec le *Naive Bayes* dans le cadre de la classification de textes et l'analyse de sentiments. Nous supposons le modèle *Naive Bayes* stationnaire. Cela signifie que les paramètres ne dépendent pas du temps. Ainsi, l'indice temporel de certains paramètres est omis dans cette section.

2.5.1 Les méthodes d'embedding utilisées

Nous allons utiliser, pour ces expériences, trois méthodes d'*embedding* différentes :

— GloVe, convertissant chaque mot en vecteur de taille 100 ;

— FastText, convertissant chaque mot en vecteur de taille 300 ;

— ExtVec, convertissant chaque mot en vecteur de taille 300.

2.5.2 Apprentissage des paramètres

Concernant le paramètre π défini en paragraphe 2.1, utilisé par les deux moyens de calculer le classificateur, il est estimé par maximum de vraisemblance, décrite en section 1.4.1, pour tout $\lambda_i \in \Lambda_X$:

$$\pi(i) = \frac{N_i}{\sum_j N_j},$$

avec N_i le nombre de fois où la variable X vaut λ_i dans le jeu de données d'entraînement.

AG News		
	Gen	Dis
GloVe	12.79%	**12.30**% $\pm$ 0.20
FastText	11.82%	**10.26**% $\pm$ 0.04
ExtVec	12.38%	**11.29**% $\pm$ 0.16

TABLE 2.1 – Taux d'erreur avec le classificateur du *Naive Bayes* calculé de manière générative et discriminante pour la classification de textes sur AG News.

IMDB		
	Gen	Dis
GloVe	30.46%	**26.76**% $\pm$ 0.50
FastText	27.96%	**16.33**% $\pm$ 0.25
ExtVec	25.28%	**21.37**% $\pm$ 0.34

TABLE 2.2 – Taux d'erreur avec le classificateur du *Naive Bayes* calculé de manière générative et discriminante pour l'analyse de sentiments sur IMDB.

Le paramètre b, défini en paragraphe 2.1 et utilisé par le classificateur calculé de manière générative, est modélisé par une loi gaussienne pour chaque classe. Ainsi, par exemple avec GloVe, où chaque mot du texte est converti en vecteur de taille 100, nous estimons par maximum de vraisemblance la moyenne, qui est un vecteur de taille 100, et la matrice de variance-covariance, de taille 100×100.

La méthode d'apprentissage diffère pour l'estimation de la fonction L définie en section 2.2, qui est utilisée par le classificateur sous sa forme discriminante (2.8). Nous le modélisons par une Régression Logistique (2.9), car il s'agit du modèle le plus simple et le plus populaire pour modéliser ce type de paramètre. Ses paramètres sont estimés par descente de gradient, décrite en section 1.4.2, avec l'optimisation Adam et minimisant la *cross-entropy* (1.6) avec un taux d'apprentissage de 0.001.

2.5.3 Résultats

Nous appliquons le classificateur du *Naive Bayes* calculé des deux manières pour la classification de textes sur AG News et l'analyse de sentiments sur IMDB. Nous reportons les taux d'erreur pour le classificateur calculé de manière générative et discriminante dans les Tables 2.1 et 2.2. Dans la mesure où l'apprentissage du paramètre L est basé sur une méthode stochastique, nous répétons cette expérience à 5 reprises et reportons la moyenne et l'intervalle de confiance gaussien à 95%.

Le classificateur du *Naive Bayes* calculé de manière générative a des performances plus faibles que le classificateur calculé de manière discriminante pour ces deux tâches, peu importe la méthode d'*embedding* utilisée. La réduction de l'erreur s'élève parfois à plus de 40%, comme nous pouvons le voir pour l'analyse de sentiments sur IMDB avec FastText. De plus, il est intéressant de remarquer que ces modèles, même si les scores sont loin de l'état-de-l'art, sont interprétables et ont des temps d'inférence très faibles, inférieurs à 1 milliseconde, ce qui peut s'avérer déterminant dans le cadre industriel.

2.6 Conclusion

Au cours de ce chapitre, nous avons décrit comment calculer le classificateur du *Naive Bayes* de manière discriminante. Nous avons également montré que ce dernier peut être vu comme un cas plus général du classificateur de la Régression Logistique et qu'il existe un CRF équivalent. De plus, des applications pour la classification de textes et l'analyse de sentiments ont montré l'intérêt empirique d'utiliser le classificateur sous sa forme discriminante dans un cadre supervisé.

Chapitre 3

Nouveaux algorithmes pour définir des classificateurs calculés de manière discriminante dans la chaîne de Markov cachée

Après l'étude sur le *Naive Bayes*, nous allons effectuer un travail similaire pour un autre modèle probabiliste très connu : la chaîne de Markov cachée. Après avoir rappelé sa définition et ses classificateurs usuels, qui sont calculés de manière générative, nous allons présenter de nouveaux algorithmes permettant le calcul des classificateurs de manière discriminante : *Entropic Forward-Backward* (EFB) et *Entropic Viterbi* (EV). Sous la lumière de ces nouveaux résultats, nous allons revoir les comparaisons entre la chaîne de Markov cachée et le *Maximum Entropy Markov Model* et le *linear-chain Conditional Random Field*. Nous appliquerons nos nouveaux algorithmes pour le *Chunking* pour illustrer leurs apports en terme de performance.

3.1 Introduction

La chaîne de Markov cachée (HMC, pour *Hidden Markov Chain*) est un modèle probabiliste considérant deux processus stochastiques : le processus caché $X_{1:T} = (X_1, ..., X_T)$ et le processus observé $Y_{1:T} = (Y_1, ..., Y_T)$, avec pour tout $t \in \{1, ..., T\}, X_t$ prenant ses valeurs dans $\Lambda_X = \{\lambda_1, ..., \lambda_N\}$, et Y_t dans Ω_Y, ce dernier pouvant être discret ou continu. Ce modèle est utilisé dans un très grand nombre d'applications, comme en finance [127, 128, 129], image [130, 131, 132], NLP [133, 134, 135, 136], ou encore traitement de la parole [137, 138, 139].

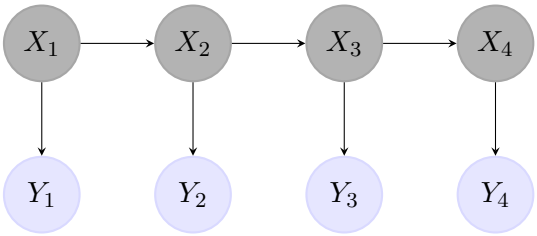


FIGURE 3.1 – Graphe probabiliste orienté de la HMC.

La HMC est définie par la loi jointe suivante :

$$p(x_{1:T}, y_{1:T}) = p(x_1) \prod_{t=1}^{T-1} p(x_{t+1}|x_t) \prod_{t=1}^{T} p(y_t|x_t),$$

et son graphe probabiliste orienté peut être représenté en figure 3.1.

Il existe deux classificateurs bayésiens de la HMC particulièrement connus, dépendant de la fonction de perte utilisée : MPM et MAP. Avant de décrire les algorithmes permettant de calculer ces classificateurs, nous introduisons les notations suivantes, pour tout $\lambda_i, \lambda_j \in \Lambda_X, y_t \in \Omega_Y$, en supposant la HMC stationnaire :

- pour tout $t \in \{1, ..., T\}, \pi(i) = P(X_t = \lambda_i)$;
- pour tout $t \in \{1, ..., T-1\}, a_i(j) = P(X_{t+1} = \lambda_j | X_t = \lambda_i)$;
- pour tout $t \in \{1, ..., T\}, b_i(y_t) = P(Y_t = y_t | X_t = \lambda_i)$.

3.1.1 L'algorithme Forward-Backward

Pour rappel, l'estimateur du MPM, $\hat{x}_{1:T} = (\hat{x}_1, ..., \hat{x}_T)$, est défini tel que, pour tout $t \in \{1, ..., T\}$:

$$\hat{x}_t = \arg \max_{\lambda_i \in \Lambda_X} P(X_t = \lambda_i | Y_{1:T} = y_{1:T})$$

Cet estimateur peut être calculé avec l'algorithme *Forward-Backward* [19], permettant de calculer la loi marginale a posteriori de la HMC :

$$P(X_t = \lambda_i | Y_{1:T} = y_{1:T}) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}, \tag{3.1}$$

avec les probabilités *forward* $\alpha_t(i) = P(X_t = \lambda_i, Y_{1:t} = y_{1:t})$:

— pour $t = 1$:

$$\alpha_1(i) = \pi(i) b_i(y_1); \tag{3.2}$$

— pour $1 \leq t < T$:

$$\alpha_{t+1}(i) = b_i(y_{t+1}) \sum_{j=1}^{N} \alpha_t(j) a_j(i); \tag{3.3}$$

et les probabilités *backward* $\beta_t(i) = P(Y_{t+1:T} = y_{t+1:T} | X_t = \lambda_i)$:

— pour $t = T$:

$$\beta_T(i) = 1; \tag{3.4}$$

— pour $1 \leq t < T$:

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(j) a_i(j) b_j(y_{t+1}). \tag{3.5}$$

Ainsi, l'algorithme *Forward-Backward* permet de calculer un classificateur de manière générative, en s'intéressant en premier lieu à la loi jointe $p(x_t, y_{1:T})$, puis utilisant la règle de Bayes pour le calcul de la loi a posteriori.

3.1.2 L'algorithme de Viterbi

Pour rappel, l'estimateur du MAP $\hat{x}_{1:T}$ est défini tel que :

$$\hat{x}_{1:T} = \arg \max_{\lambda_{1:T} \in (\Lambda_X)^T} P(X_{1:T} = \lambda_{1:T} | Y_{1:T} = y_{1:T}).$$

Celui-ci peut être calculé par l'algorithme de Viterbi [140]. Il consiste à calculer par récurrence la fonction V définie ainsi :

— pour $t = 1$:

$$V_1(i) = \pi(i) b_i(y_1);$$

— pour $1 \leq t < T$:

$$V_{t+1}(i) = \max_{\lambda_j \in \Lambda_X} \left(b_i(y_{t+1}) a_j(i) V_t(j) \right).$$

Puis nous retrouvons la séquence estimée avec un « chemin de Viterbi » :

$$\hat{x}_T = \arg \max_{\lambda_i \in \Lambda_X} \left(V_T(i) \right),$$
$$\hat{x}_{t-1} = \mathrm{Ptr}(\hat{x}_t, t),$$

avec la fonction $\mathrm{Ptr}(\lambda_i, t)$ allant de $\Lambda_X \times \{1, ..., T\}$ dans Λ_X, retournant la valeur de λ utilisée pour calculer $V_t(i)$ si $t > 1$ ou λ_i si $t = 1$.

Remarque 3.1. L'algorithme de Viterbi permet de calculer une approximation du MAP, et non le MAP exact.

3.1.3 La HMC et la gestion des features

Étant donné que les algorithmes *Forward-Backward* et Viterbi prennent en compte la loi des observations, ils définissent des classificateurs calculés de manière générative. Par conséquent, leur incapacité à prendre en compte les *features* complexes des observations fut la source d'un grand nombre de critiques, décrites en section 1.4.3. Nous pouvons notamment cité [39] à propos de ce modèle, et de tous modèles génératifs : *"Although we could try to hack the HMC to find ways to incorporate some of these, in general it's hard for generative models like HMCs to add arbitrary features directly into the model in a clean way."*.

Pour pallier ce problème dans le cadre des HMC, des techniques d'approximation de la probabilité d'émission peuvent être employées, comme dans [133] pour le NLP. Néanmoins, cette technique reste très limitée en termes de nombre de *features* possible, qui ne peut être arbitraires, et se limite souvent à quelques suffixes et si le mot possède une majuscule. D'autres méthodes plus complexes ont été proposées dans le cadre d'apprentissage non-supervisé, mais celles-ci demeurent également limitées. Nous pouvons notamment citer [141], modélisant la probabilité d'émission par une Régression Logistique. Cette méthode pose problème sur le choix des *features* arbitraires et notamment au niveau de la somme du dénominateur de cette Régression Logistique. En effet, l'algorithme ne pourra pas suivre en terme computationnel si cette somme se produit sur des millions de termes, par exemple si nous considérons tous les suffixes et tous les préfixes de taille 10 à 1 qu'il est possible d'écrire. Ces travaux ne prennent pas non plus en compte le cas de *features* continues, qui implique le calcul d'une intégrale, ce qui s'avère également bloquant. Les solutions que nous allons présenter n'ont pas ces problèmes, et peuvent prendre en compte les *features* arbitraires sans contraintes pour la HMC.

Une autre méthode proposée pour résoudre ce problème sans contraintes fut la conception de nouveaux modèles séquentiels définis uniquement à partir de lois a posteriori. Ainsi, le *Maximum Entropy Markov Model* [22], suivi par le *linear-chain Conditional Random Field* [23], ont été proposés et popularisés. Ces modèles probabilistes seront détaillés plus loin. Ils peuvent définir un classificateur calculé de manière discriminante, n'ayant pas de contraintes sur les *features* des observations. Cela a ainsi contribué à l'abandon du modèle HMC, notamment pour le NLP, depuis une vingtaine d'années car les observations peuvent avoir un très grand nombre de *features* pour ce type de tâche, comme vu en section 1.6.

3.2 Nouveaux algorithmes pour calculer des classificateurs de manière discriminante dans le cadre de la HMC

Dans cette section, nous allons voir qu'il est possible de calculer le MPM et le MAP dans le cadre de la HMC sans utiliser la loi des observations. Pour ce faire, nous introduisons le nouveau paramètre suivant, pour tout $t \in \{1, ..., T\}, \lambda_i \in \Lambda_X, y \in \Omega_Y$:

$$L_y(i) = P(X_t = \lambda_i | Y_t = y). \tag{3.6}$$

Ces nouveaux algorithmes n'utiliseront plus le paramètre b modélisant la loi conditionnelle des observations.

3.2.1 L'algorithme Entropic Forward-Backward

Proposition 3.1. Soit $(X_t, Y_t)_{1 \leq t \leq T}$ une HMC. Alors, pour tout $t \in \{1, ..., T\}, \lambda_i \in \Lambda_X$, nous avons :

$$P(X_t = \lambda_i | Y_{1:T} = y_{1:T}) = \frac{\alpha_t^E(i)\beta_t^E(i)}{\sum_{j=1}^{N} \alpha_t^E(j)\beta_t^E(j)} \tag{3.7}$$

avec les fonctions *entropic forward* α^E définies avec la récursion :

— si $t = 1$:

$$\alpha_1^E(i) = L_{y_1}(i); \tag{3.8}$$

— si $1 \leq t < T$:

$$\alpha_{t+1}^E(i) = \frac{L_{y_{t+1}}(i)}{\pi(i)} \sum_{j=1}^{N} \alpha_t^E(j) a_j(i), \tag{3.9}$$

et les fonctions *entropic backward* β^E :

— si $t = T$:

$$\beta_T^E(i) = 1;$$

— si $1 \leq t < T$:

$$\beta_t^E(i) = \sum_{j=1}^{N} \frac{L_{y_{t+1}}(j)}{\pi(j)} \beta_{t+1}^E(j) a_i(j). \tag{3.10}$$

Démonstration Montrons que, pour tout $t \in \{1, ..., T\}$:

$$\alpha_t^E(i) = \frac{\alpha_t(i)}{p(y_1)p(y_2)...p(y_t)}, \tag{3.11}$$

$$\beta_t^E(i) = \frac{\beta_t(i)}{p(y_{t+1})p(y_{t+2})...p(y_T)}, \tag{3.12}$$

où les quantités $\alpha_t(i)$ et $\beta_t(i)$ sont les probabilités *forward* et *backward* définies en section (3.1.1). Pour montrer (3.11), nous utilisons la récursion (3.9). (3.11) est vrai pour $t = 1$ en utilisant la formule de Bayes. Nous supposons (3.11) vrai pour $t \geq 1$, et nous la prouvons à $t + 1$. Nous avons :

$$\begin{aligned} P(X_{t+1} = \lambda_i, Y_{t+1} = y_{t+1}) = L_{y_{t+1}}(i)p(y_{t+1}) = \pi(i)b_i(y_{t+1}) \\ \iff \frac{L_{y_{t+1}}(i)}{\pi(i)} = \frac{b_i(y_{t+1})}{p(y_{t+1})} \end{aligned} \tag{3.13}$$

Associé avec (3.9), cela donne :

$$\begin{aligned} \alpha_{t+1}^E(i) &= \frac{b_i(y_{t+1})}{p(y_{t+1})} \sum_{j=1}^{N} \alpha_t^E(j)a_j(i) \\ &= \frac{b_i(y_{t+1})}{p(y_{t+1})} \sum_{j=1}^{N} \frac{\alpha_t(i)}{p(y_1)p(y_2)...p(y_t)} a_j(i) \\ &= \frac{\alpha_{t+1}(i)}{p(y_1)...p(y_t)p(y_{t+1})}, \end{aligned}$$

ainsi (3.11) est vrai pour $t + 1$.
De manière similaire, nous prouvons (3.12) à partir de la récursion (3.10). Cela est vrai pour $t = T$. Nous la supposons vraie pour $t + 1 < T$, et allons la démontrer pour t.
Comme pour la preuve précédente, nous utilisons (3.13) pour (3.10). Associé avec (3.5), nous avons :

$$\begin{aligned} \beta_t^E(i) &= \frac{1}{p(y_{t+1})} \sum_{j=1}^{N} b_j(y_{t+1})\beta_{t+1}^E(j)a_i(j) \\ &= \frac{1}{p(y_{t+1})...p(y_T)} \sum_{j=1}^{N} b_j(y_{t+1})\beta_{t+1}(j)a_i(j) \\ &= \frac{\beta_t(i)}{p(y_{t+1})...p(y_T)} \end{aligned}$$

Ainsi, (3.12) est vrai à t.
(3.11) et (3.12) impliquent :

$$\alpha_t^E(i)\beta_t^E(i) = \frac{\alpha_t(i)\beta_t(i)}{\prod_{t=1}^{T} p(y_t)}$$

Par conséquent :

$$\frac{\alpha_t^E(i)\beta_t^E(i)}{\sum_{j=1}^{N} \alpha_t^E(j)\beta_t^E(j)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} = P(X_t = \lambda_i | Y_{1:T} = y_{1:T})$$

prouvant (3.7), ce qui achève la démonstration.

3.2.2 L'algorithme Entropic Viterbi

Proposition 3.2. Soit $(X_t, Y_t)_{1\leq t\leq T}$ une HMC. Il est possible de calculer le MAP, $\arg\max_{\lambda_{1:T}\in(\Lambda_X)^T} P(X_{1:T} = \lambda_{1:T}|Y_{1:T} = y_{1:T})$, avec la fonction V^E définie par :

— pour $t = 1$:

$$V_1^E(i) = L_{y_1}(i);$$

— et pour $1 \leq t < T$:

$$V_{t+1}^E(i) = \max_{\lambda_j\in\Lambda_X} \left(V_t^E(j) \frac{a_j(i)}{\pi(i)} L_{y_{t+1}}(i) \right).$$

Cet algorithme imite l'algorithme de Viterbi en retrouvant le même chemin de Viterbi pour calculer le MAP :

$$\hat{x}_T = \arg\max_{\lambda_i\in\Lambda_X} \left(V_T^E(i) \right)$$
$$\hat{x}_{t-1} = \mathrm{Ptr}^E(x_t, t)$$

avec la fonction $\mathrm{Ptr}^E(\lambda_i, t)$ retournant la valeur de λ utilisée pour calculer $V_t^E(i)$ si $t > 1$ et λ_i si $t = 1$.

Démonstration Nous montrons qu'il existe une valeur $\kappa_t(y_{1:t})$ ne dépendant pas des variables cachées $x_{1:t}$ telle que :

$$\kappa_t(y_{1:t}) V_t^E(i) = V_t(i). \tag{3.14}$$

Pour $t = 1$, $V_1(i) = \pi(i)b_i(y_1) = p(y_1)L_{y_1}(i) = \kappa_1(y_1)V_1^E(i)$. Par conséquent, (3.14) est vrai pour $t = 1$.
Nous supposons (3.14) vrai pour t, et nous allons prouver cette assertion pour $t + 1$:

$$\begin{aligned} V_{t+1}(i) &= \max_{\lambda_j \in \Lambda_X} (b_i(y_{t+1})a_j(i)V_t(j)) \\ &= \max_{\lambda_j \in \Lambda_X} \left(\frac{p(y_{t+1})L_{y_{t+1}}(i)}{\pi(i)} a_j(i)\kappa_t(y_{1:t})V_t^E(j) \right) \\ &= \kappa_t(y_{1:t})p(y_{t+1})V_{t+1}^E(i) = \kappa_{t+1}(y_{1:t+1})V_{t+1}^E(i), \end{aligned}$$

concluant la preuve de (3.14). Par conséquent, dans la mesure où $\kappa_t(y_{1:t})$ n'influence pas les fonctions max et arg max, l'algorithme EV retrouve le même chemin que l'algorithme de Viterbi.

Ainsi, les algorithmes EFB et EV donnent le MPM et le MAP dans le cadre de la HMC sans prendre en compte la loi des observations, permettant de calculer des classificateurs de manière discriminante.

3.3 Comparaison entre la HMC et le MEMM

Comme rappeler en section 3.1.3, pour palier au problème de la considération de *features* arbitraires, de nouveaux modèles séquentiels ont été proposés. Ceux-ci sont considérés comme supérieurs à la HMC, dans la mesure où cette dernière a le problème des *features*. Nos nouveaux algorithmes prouvant le contraire, les deux sections suivantes vont mettre à jour la comparaison entre la HMC et deux autres modèles séquentiels connus et conçus pour la remplacer dans le cadre supervisé : Le *Maximum Entropy Markov Model* (MEMM) [22] et le *linear-chain Conditional Random Field* [23].

3.3.1 Le modèle MEMM

Le *Maximum Entropy Markov Model* (MEMM) est un modèle probabiliste impliquant, comme la HMC, des variables aléatoire cachées $X_{1:T}$ et observées $Y_{1:T}$. Il est défini à partir de la loi conditionnelle suivante :

$$p(x_{1:T}|y_{1:T}) = p(x_1|y_1) \prod_{t=1}^{T-1} p(x_{t+1}|x_t, y_{t+1}),$$

et peut être représenté avec le graphe en figure 3.2.

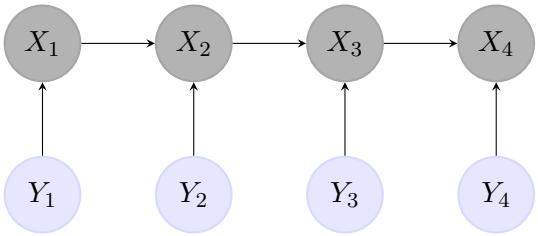


FIGURE 3.2 – Graphe probabiliste orienté du MEMM.

Le MEMM a été proposé pour concurrencer la HMC dans la mesure où les classificateurs de cette dernière étaient supposés ne pas pouvoir prendre en compte les *features* arbitraires des observations. En tant que modèle discriminant, il peut définir directement un classificateur calculé de manière discriminante, qui ne rencontre pas les problèmes des *features*. Il est important de noter que le graphe des dépendances du MEMM implique l'indépendance, pour tout t, entre X_t et les variables $Y_{t+1:T}$.

Pour comparer ces deux modèles, nous présentons la restauration du MPM avec le MEMM stationnaire. Nous posons, dans le cadre du MEMM :

$$\alpha_t^M(i) = P(X_t = \lambda_i | Y_{1:T} = y_{1:T}) = P(X_t = \lambda_i | Y_{1:t} = y_{1:t}).$$

Étant donné la réalisation des observations $y_{1:T}$, les probabilités α^M se calculent de la manière suivante :

$$\begin{aligned}
\alpha_1^M(i) &= P(X_1 = \lambda_i | Y_1 = y_1); \\
\alpha_{t+1}^M(i) &= \sum_{j=1}^{N} P(X_{t+1} = \lambda_i | X_t = \lambda_j, Y_{t+1} = y_{t+1}) \alpha_t^M(j).
\end{aligned}$$

3.3.2 Différences entre la HMC et le MEMM

Avant les algorithmes EFB et EV, il était supposé impossible de considérer les *features* arbitraires avec la HMC, donc le MEMM était considéré comme plus adapté pour les tâches impliquant des observations avec beaucoup de *features*, notamment la segmentation de textes. Cependant, avec ces nouveaux algorithmes, les *features* peuvent être considérés exactement de la même manière, dépendant de la modélisation de la probabilité $p(x_t|y_t)$ pour la HMC et $p(x_t|y_t), p(x_{t+1}|x_t, y_{t+1})$ pour le MEMM. Ainsi, les éléments de comparaison entre ces deux modèles sont à revoir. La principale différence repose sur la « direction » des modèles durant la restauration. D'une part, pour estimer la réalisation de X_t, le MEMM n'utilise que

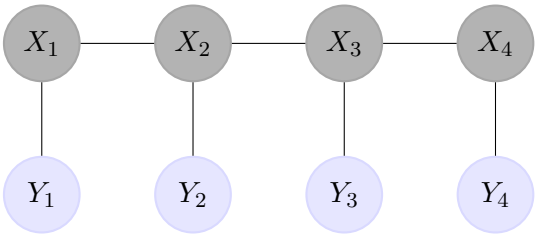


FIGURE 3.3 – Graphe probabiliste du *linear-chain* CRF.

les observations $y_{1:t}$, ne prenant jamais en compte les observations à partir de $t+1$. Le MEMM est alors qualifié d'unidirectionnel. D'autre part, la HMC utilise toutes les observations $y_{1:T}$ pour estimer la réalisation de X_t. Elle est ainsi qualifiée de bidirectionnelle. Par conséquent, nous pouvons espérer de meilleures performances de la HMC en vertu de cette propriété. Nous pouvons tout de même remarquer que l'influence de (X_t, Y_{t+1}) sur X_{t+1} a une forme plus générale dans le MEMM que dans la HMC, donc il peut exister des cas où le MEMM sera meilleur.

3.4 Comparaison entre la HMC et le linear-chain CRF

3.4.1 Le linear-chain CRF

La HMC étant bidirectionnelle, mais initialement considérée comme incapable de prendre en compte les *features* arbitraires, là où le MEMM peut, mais est unidirectionnel, un autre modèle a été proposé : le *linear-chain Conditional Random Field*. C'est un modèle probabiliste considérant toujours les mêmes variables aléatoires $X_{1:T}$ et $Y_{1:T}$, bidirectionnel, et capable de prendre en compte les *features* sans contraintes. Il peut être représenté en figure 3.3 et est défini par la loi a posteriori :

$$p(x_{1:T}|y_{1:T}) = \frac{U^{(1)}(x_1, y_1) \prod_{t=1}^{T-1} V^{(t)}(x_t, x_{t+1}) U^{(t+1)}(x_{t+1}, y_{t+1})}{\sum_{x'_{1:T}} U^{(1)}(x'_1, y_1) \prod_{t=1}^{T-1} V^{(t)}(x'_t, x'_{t+1}) U^{(t+1)}(x'_{t+1}, y_{t+1})} \tag{3.15}$$

avec $U^{(t)}$ et $V^{(t)}$ des fonctions potentielles mesurant l'influence entre les différentes variables.

Le *linear-chain* CRF se présente comme l'équivalent CRF de la HMC. Ces deux modèles sont très largement comparés et pour un grand nombre d'applications

[22, 24, 23, 42, 43, 44, 45, 41, 46, 47, 48, 49, 50]. Le *linear-chain* CRF est toujours préféré pour sa capacité à prendre en compte les *features* complexes sans contraintes. Ainsi que nous venons de le voir, cette préférence n'est pas justifiée car la HMC permet la même prise en compte des *features* complexes.

3.4.2 Équivalence entre la HMC et le linear-chain CRF

À la lumière de nos nouveaux algorithmes, *Entropic Forward-Backward* et *Entropic Viterbi*, les modèles HMC et *linear-chain* CRF ont les mêmes propriétés :

— ils sont bidirectionnels ;

— leurs classificateurs peuvent s'affranchir de la modélisation de la loi des observations.

Dans cette section, nous allons plus loin dans la comparaison de ces deux modèles en montrant qu'ils sont équivalents, au sens de la définition 2.1 introduite en section 2.4.

Pour montrer cette équivalence, nous allons utiliser le lemme suivant, définissant une chaîne de Markov [142].

Lemme 3.1. Soit $X_{1:T} = (X_1, ..., X_T)$ un processus stochastique à valeurs dans un ensemble fini Λ_X. Alors

1. $X_{1:T}$ est une chaîne de Markov si et seulement s'il existe $T-1$ fonctions $\varphi_1, ..., \varphi_{T-1}$ de $(\Lambda_X)^2$ dans $\mathbb{R}^+$ telles que
$$p(x_1, ..., x_T) \propto \varphi_1(x_1, x_2)...\varphi_{T-1}(x_{T-1}, x_T), \tag{3.16}$$
avec « $\propto$ » signifiant « proportionnel à » ;
2. Pour la chaîne de Markov définie avec $\varphi_1, ..., \varphi_{T-1}$ vérifiant (3.16), $p(x_1)$ et $p(x_{t+1}|x_t)$ sont définies par
$$\begin{aligned} p(x_1) &= \frac{\beta_1(x_1)}{\sum\limits_{x'_1} \beta_1(x'_1)}; \\ p(x_{t+1}|x_t) &= \frac{\varphi_t(x_t, x_{t+1})\beta_{t+1}(x_{t+1})}{\beta_t(x_t)}, \end{aligned} \tag{3.17}$$
avec $\beta_1(x_1), ..., \beta_T(x_T)$ définies avec la récursion *backward* suivante :
$$\begin{aligned} \beta_T(x_T) &= 1; \\ \beta_t(x_t) &= \sum_{x_{t+1}} \varphi_t(x_t, x_{t+1})\beta_{t+1}(x_{t+1}), \text{ pour } 1 \leq t < T. \end{aligned} \tag{3.18}$$

Pour la preuve, voir [142], Lemma 2.1, page 6.

Cette caractérisation de la HMC permet d'émettre la proposition suivante :

Proposition 3.3. Soit $(X_t, Y_t)_{1\leq t\leq T}$ un processus stochastique dont la loi a posteriori est un *linear-chain* CRF (3.15). Alors (3.15) est la loi a posteriori d'une HMC $p(x_{1:T}, y_{1:T})$ définie ainsi :

$$p(x_1) = \frac{\beta_1(x_1)}{\sum_{x'_1} \beta_1(x'_1)}; \tag{3.19}$$

$$p(x_{t+1}|x_t) = \frac{\varphi_t(x_t, x_{t+1})\beta_{t+1}(x_{t+1})}{\beta_t(x_t)} \tag{3.20}$$

$$p(y_t|x_t) = \frac{U^{(t)}(x_t, y_t)}{\psi_t(x_t)}, \tag{3.21}$$

avec :

$$\psi_t(x_t) = \sum_{y_t} U^{(t)}(x_t, y_t); \tag{3.22}$$

$$\varphi_1(x_1, x_2) = V^{(1)}(x_1, x_2)\psi_1(x_1)\psi_2(x_2), \tag{3.23}$$

$$\text{et, pour } t = 2, ...,T-1, \quad \varphi_t(x_t, x_{t+1}) = V^{(t)}(x_t, x_{t+1})\psi_{t+1}(x_{t+1}). \tag{3.24}$$

et les fonctions *backward* :

$$\begin{gathered}\beta_T(x_T) = 1, \\ \text{et pour } t = T-1, ..., 2 : \beta_t(x_t) = \sum_{x_{t+1}} \varphi_t(x_t, x_{t+1})\beta_{t+1}(x_{t+1}).\end{gathered} \tag{3.25}$$

Démonstration En premier lieu, nous montrons que la loi a posteriori du modèle défini avec (3.19)-(3.21) est bien égale à celle d'un *linear-chain* CRF (3.15) :

$$\begin{aligned}
p(x_{1:T}, y_{1:T}) &= p(x_1) \prod_{t=1}^{T-1} p(x_{t+1}|x_t) \prod_{t=1}^{T} p(y_t|x_t) \\
&= \frac{\beta_1(x_1)}{\sum_{x'_1} \beta_1(x'_1)} \prod_{t=1}^{T-1} \frac{\varphi_t(x_t, x_{t+1})\beta_{t+1}(x_{t+1})}{\beta_t(x_t)} \prod_{t=1}^{T} \frac{U^{(t)}(x_t, y_t)}{\psi_t(x_t)} \\
&= \frac{1}{\sum_{x'_1} \beta_1(x'_1)} \prod_{t=1}^{T-1} \varphi_t(x_t, x_{t+1})\beta_T(x_T) \prod_{t=1}^{T} \frac{U^{(t)}(x_t, y_t)}{\psi_t(x_t)} \\
&= \frac{1}{\sum_{x'_1} \beta_1(x'_1)} \prod_{t=1}^{T-1} V^{(t)}(x_t, x_{t+1}) \prod_{t=1}^{T} \psi_t(x_t) \prod_{t=1}^{T} \frac{U^{(t)}(x_t, y_t)}{\psi_t(x_t)}
\end{aligned}$$

$$= \frac{1}{\sum\limits_{x'_1} \beta_1(x'_1)} \prod_{t=1}^{T-1} V^{(t)}(x_t, x_{t+1}) \prod_{t=1}^{T} U^{(t)}(x_t, y_t),$$

la loi a posteriori est bien égale à (3.15). De plus, nous vérifions bien que :

$$p(x_{1:T}) = \frac{1}{\sum\limits_{x'_1} \beta_1(x'_1)} \prod_{t=1}^{T-1} \varphi_t(x_t, x_{t+1}) \propto \prod_{t=1}^{T-1} \varphi_t(x_t, x_{t+1}),$$

prouvant, avec le Lemme 3.1, que $X_{1:T}$ est une chaîne de Markov, donc le modèle défini avec (3.19)-(3.21) est bien une HMC, achevant la démonstration.

Remarque 3.2. Le résultat montrant qu'à partir d'une HMC, sa loi a posteriori permet de définir celle d'un *linear-chain* CRF est connu et nous le rappelons. Soient les variables aléatoires $X_{1:T}$ et $Y_{1:T}$ définissant une HMC. Alors sa loi a posteriori permet de définir le *linear-chain* CRF suivant :

$$\begin{gathered}\text{pour tout } t, 1 \leq t \leq T, U^{(t)}(x_t, y_t) = p(y_t|x_t) \\ V^{(1)}(x_1, x_2) = p(x_1)p(x_2|x_1) \\ \text{pour tout } t, 2 \leq t \leq T-1, V^{(t)}(x_t, x_{t+1}) = p(x_{t+1}|x_t).\end{gathered}$$

Nous avons donc montré l'équivalence entre la HMC et le *linear-chain* CRF. Ainsi, nous étendons le résultat soutenant que le *linear-chain* CRF et la HMC sont équivalents si l'espace des valeurs de Y_t est de dimension 1 [24], autrement dit si les observations n'ont qu'une seule *feature*. Nous étendons également le résultat de [143], qui a prouvé cette équivalence, mais avec une condition sur la chaîne cachée et sur la loi de transition des états cachés.

3.5 Application : HMC pour le Chunking

Dans cette partie, nous allons appliquer les nouveaux algorithmes EFB et EV pour le *Chunking*. Le but étant de mettre en évidence la supériorité de ces algorithmes sur ceux usuellement utilisés, *Forward-Backward* et Viterbi, notamment dans le cas où les observations comportent beaucoup de *features*.

Nous appliquons les quatre algorithmes : *Forward-Backward*, Viterbi, EFB, et EV pour le *Chunking* sur les *datasets* CoNLL 2000 et CoNLL 2003. Nous employons quatre méthodes d'*embedding* différentes :

— le mot lui-même, dénoté *One-Hot Word* ;

	One-Hot Word	One-Hot Handcrafted	FastText	Flair
FB	92.63	92.62	68.46	95.09
EFB	92.73 ± 0.03	92.82 ± 0.12	92.41 ± 0.03	$\mathbf{97.94} \pm 0.01$
Viterbi	92.47	92.48	67.89	95.09
EV	92.31 ± 0.06	92.65 ± 0.16	91.83 ± 0.20	97.82 ± 0.04

TABLE 3.1 – Score F_1 du *Chunking* sur le *dataset* CoNLL 2000 avec les quatre algorithmes de la HMC et différents *embedding*.

	One-Hot Word	One-Hot Handcrafted	FastText	Flair
FB	94.22	94.30	78.58	94.38
EFB	94.27 ± 0.10	94.55 ± 0.06	94.02 ± 0.08	$\mathbf{96.44} \pm 0.06$
Viterbi	94.12	94.18	78.59	94.39
EV	94.17 ± 0.08	94.40 ± 0.11	93.51 ± 0.10	96.37 ± 0.08

TABLE 3.2 – Score F_1 du *Chunking* sur le *dataset* CoNLL 2003 avec les quatre algorithmes de la HMC et différents *embedding*.

— le mot lui-même, les suffixes de taille 1 à 3, si la première lettre est en majuscule, s'il s'agit du premier mot de la phrase, si le mot possède un chiffre, dénoté *One-Hot Handcrafted* ;

— FastText, convertissant les mots en vecteurs de taille 300 ;

— Flair, convertissant les mots en vecteurs de taille 4096.

Les paramètres π, a, et b, introduits en section 3.1, sont appris par maximum de vraisemblance, avec b modélisé par une loi gaussienne avec les méthodes d'*embedding* continues. De son côté, le paramètre L, introduit en section 3.2, est modélisé par une Régression Logistique et est appris par descente de gradient avec la méthode Adam, un mini-batch de taille 50, et un taux d'apprentissage de 0.01. De par la nature stochastique de cet apprentissage, nous le réalisons à cinq reprises et nous reportons la moyenne et l'intervalle de confiance gaussien à 95% pour chaque expérience.

Les résultats sur CoNLL 2000 sont en tableau 3.1, et CoNLL 2003 en tableau 3.2. Tout d'abord, nous pouvons observer que les algorithmes classiques sont presque équivalents à EFB et EV concernant les méthodes *One-Hot Word* et *One-Hot Handcrafted*, avec des résultats légèrement meilleurs pour ces derniers. Ce résultat est attendu pour *One-Hot Word*, car dans la mesure où nous ne considérons qu'une seule *feature*, les algorithmes sont strictement équivalents. Pour *One-Hot Handcrafted*, cela montre que les *features* choisies ne sont pas d'une grande im-

portance pour le *Chunking*, confirmé par le fait que l'amélioration par rapport à *One-Hot Word* est très faible.

La comparaison la plus pertinente concerne les méthodes d'*embedding* FastText et Flair, car ces méthodes sont parmi les plus utilisées de nos jours en NLP. Les algorithmes *Forward-Backward* et Viterbi sont confrontés au problème des *features*, tandis que EFB et EV les gèrent sans contraintes. Ainsi, les résultats avec ces derniers sont meilleurs qu'avec les algorithmes calculant le classificateur de manière générative. Cela se remarque notamment lors de l'utilisation de la méthode d'*embedding* FastText et confirme l'intérêt de nos nouveaux algorithmes, permettant à la HMC d'être pleinement utilisée pour le NLP.

De plus, nous pouvons même remarquer que la HMC avec EFB associée avec la méthode d'*embedding* Flair atteint l'état-de-l'art pour le *Chunking* sur CoNLL 2000. En effet, il est meilleur que la méthode ACE [93] atteignant un score F_1 de 97.30, que l'algorithme basé sur ELMO et une méthode de *masked adversarial training* atteignant 97.03 [144], ou encore que le BiLSTM-CRF [145] associé à Flair [58], atteignant un score de 96.83 [1]. Cette observation est d'autant plus intéressante que les besoins computationnels de la HMC sont bien moindres que les algorithmes que nous venons de citer, tout en atteignant de meilleures performances.

3.6 Conclusion

Dans ce chapitre, nous avons présenté les nouveaux algorithmes *Entropic Forward-Backward* et *Entropic Viterbi* permettant de définir les classificateurs du MAP et du MPM de la HMC sans modéliser la loi des observations. Ainsi, celui-ci peut-être justement comparé au MEMM et au *linear-chain* CRF, montrant notamment l'équivalence entre ce dernier et la HMC. Nous avons montré l'intérêt de nos nouveaux algorithmes pour le *Chunking*, où ils permettent de largement surpasser ceux utilisés habituellement, atteignant même l'état-de-l'art.

1. Le classement des algorithmes pour le Chunking sur le dataset de référence, CoNLL 2000, est disponible sur le lien suivant : https://paperswithcode.com/sota/chunking-on-conll-2000

Chapitre 4

Définir un classificateur calculé de manière discriminante dans un modèle probabiliste génératif

Dans un premier temps, nous avons étudié le *Naive Bayes*, qui est l'un des modèles probabilistes génératifs les plus connus, et nous avons montré qu'il est possible de calculer son classificateur de manière discriminante. Pour ce faire, nous avons multiplié le numérateur et le dénominateur dans (2.2) par une valeur $\kappa^{NB}(y_{1:T})$ telle que la loi des observations ne soit plus requise pour calculer le classificateur :

$$\begin{aligned} P(X = \lambda_i | Y_{1:T} = y_{1:T}) &= \frac{P(X = \lambda_i, Y_{1:T} = y_{1:T})}{\sum_{\lambda_j \in \Lambda_X} P(X = \lambda_j, Y_{1:T} = y_{1:T})} \\ &= \frac{\kappa^{NB}(y_{1:T}) P(X = \lambda_i, Y_{1:T} = y_{1:T})}{\sum_{\lambda_j \in \Lambda_X} \kappa^{NB}(y_{1:T}) P(X = \lambda_j, Y_{1:T} = y_{1:T})} \end{aligned}$$

Au vu de la démonstration de 2.2, nous avons $\kappa^{NB}(y_{1:T}) = \left[\prod_{t=1}^{T} p(y_t)\right]^{-1}$.

Nous avons fait de même avec la HMC. À titre d'exemple, pour calculer $P(X_t = \lambda_i | Y_{1:T} = y_{1:T})$, nécessaire pour définir le classificateur de Bayes du MPM, nous avons également multiplié le numérateur et le dénominateur dans (3.1) :

$$\begin{aligned} P(X_t = \lambda_i | Y_{1:T} = y_{1:T}) &= \frac{P(X_t = \lambda_i, Y_{1:T} = y_{1:T})}{\sum_{\lambda_j \in \Lambda_X} P(X_t = \lambda_j, Y_{1:T} = y_{1:T})} \\ &= \frac{\kappa^{HMC}(y_{1:T}) P(X_t = \lambda_i, Y_{1:T} = y_{1:T})}{\sum_{\lambda_j \in \Lambda_X} \kappa^{HMC}(y_{1:T}) P(X_t = \lambda_j, Y_{1:T} = y_{1:T})} \end{aligned}$$

avec $\kappa^{HMC}(y_{1:T}) = \left[\prod_{t=1}^{T} p(y_t)\right]^{-1}$, permettant de ne plus avoir besoin à modéliser la loi des observations pour définir le classificateur.

Dans ce chapitre, nous allons montrer qu'il est possible de construire un classificateur calculé de manière discriminante à partir de tout modèle probabiliste génératif. Cela signifie que pour toute loi jointe donnée, il est possible de calculer le classificateur induit sans utiliser la loi des observations. Cela va impliquer notamment une nouvelle paramétrisation des modèles, différente de celles utilisées usuellement, comme nous l'avons vu concernant le *Naive Bayes* et la HMC.

4.1 Cadre

Nous considérons une, ou plusieurs, variable aléatoire cachée X et les variables aléatoires observées $Y_{1:T}, T \in \mathbb{N}^*$. X est à valeurs dans l'espace fini discret Λ_X, et Y_t dans $\Omega_Y^{(t)}$, qui peut être discret ou continu.

Nous notons X_H les variables cachées d'intérêt, que nous cherchons à estimer, avec Λ_+ l'espace de toutes ses valeurs possibles, et X_O le vecteur des composantes restantes, avec Λ_- l'espace de toutes ses valeurs possibles. Donc $X = (X_H, X_O)$. De plus, nous posons, pour tout t, A^t l'ensemble minimal des composantes de $Y_{1:t-1}$ et X tel que $p(y_t|x, y_{1:t-1}) = p(y_t|A^t)$. Nous posons également $A_y^t = \{Y_{t'} \text{ telle que } Y_{t'} \in A^t\}$ et $A_x^t = \{X_{t'} \text{ telle que } X_{t'} \in A^t\}$. Par conséquent, $A^t = A_y^t \cup A_x^t$, et $A_y^t \cap A_x^t = \emptyset$.

À titre d'exemple, pour le *Naive Bayes*, nous avons pour tout $t, A^t = \{x\}$ car $p(y_t|x, y_{1:t-1}) = p(y_t|x)$, ainsi $A_x^t = \{x\}$ et $A_y^t = \emptyset$.

4.2 Résultat général

Proposition 4.1. Soit le modèle génératif $p(x, y_{1:T}) = p(x_H, x_O, y_{1:T})$, et le classificateur bayésien :

$$\phi(y_{1:T}) = \arg\max_{x_H \in \Lambda_+} p(x_H|y_{1:T}). \tag{4.1}$$

Alors $\phi(y_{1:T})$ vérifie :

$$\phi(y_{1:T}) = \arg\max_{x_H \in \Lambda_+} \left[\sum_{x_O \in \Lambda_-} p(x_O, x_H) \prod_{t=1}^{T} \frac{p(A_x^t|y_t, A_y^t)}{p(A_x^t|A_y^t)}\right]. \tag{4.2}$$

Ainsi, ϕ peut être défini comme un classificateur calculé de manière discriminante dans la mesure où (4.2) n'implique aucune loi de la forme $p(y_A|x_B)$, avec A, B tels que y_A et x_B sont des ensembles non-vides des variables aléatoires observées et cachées, respectivement. Cela est vrai si les différentes lois dans (4.2) sont calculables sans utiliser de lois de cette forme, sachant qu'il est possible de les modéliser pour faire en sorte que cela soit le cas.

Démonstration Soit

$$\kappa(y_{1:T}) = \left(\prod_{t=1}^{T} p(y_t|A_y^t) \right)^{-1} \tag{4.3}$$

Nous avons

$$\begin{aligned}
p(y_{1:T}|x_O, x_H) &= \prod_{t=1}^{T} p(y_t|x_O, x_H, y_{1:t-1}) \\
&= \prod_{t=1}^{T} p(y_t|A^t) \\
&= \prod_{t=1}^{T} \frac{p(y_t, A^t)}{p(A^t)} \\
&= \prod_{t=1}^{T} \frac{p(y_t, A_y^t, A_x^t)}{p(A_y^t, A_x^t)} \\
&= \prod_{t=1}^{T} \frac{p(y_t|A_y^t)p(A_x^t|y_t, A_y^t)}{p(A_x^t|A_y^t)} \\
&= \frac{1}{\kappa(y_{1:T})} \prod_{t=1}^{T} \frac{p(A_x^t|y_t, A_y^t)}{p(A_x^t|A_y^t)}.
\end{aligned}$$

Il en résulte

$$\kappa(y_{1:T})p(x_O, x_H, y_{1:T}) = p(x_O, x_H) \prod_{t=1}^{T} \frac{p(A_x^t|y_t, A_y^t)}{p(A_x^t|A_y^t)},$$

ce qui implique

$$\kappa(y_{1:T})p(x_H, y_{1:T}) = \sum_{x_O \in \Lambda_-} p(x_O, x_H) \prod_{t=1}^{T} \frac{p(A_x^t|y_t, A_y^t)}{p(A_x^t|A_y^t)}.$$

Étant donné $y_{1:T}$, maximiser $p(x_H|y_{1:T})$ est équivalent à maximiser $\kappa(y_{1:T})p(x_H, y_{1:T})$, donc (4.1) et (4.2) sont équivalents, concluant la démonstration.

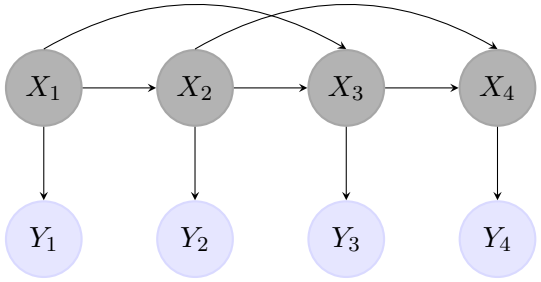


FIGURE 4.1 – Graphe probabiliste orienté de la HMC2.

Remarque 4.1. Un grand nombre de modèles probabilistes génératifs ont une loi de la forme $p(x, y_{1:T}) = p(x) \prod_{t=1}^{T} p(y_t|x)$, comme notamment le *Naive Bayes* et la HMC. Dans ce cas, la détermination de (4.2) est aisée, avec $A_x^t = \{x\}, A_y^t = \emptyset, \kappa(y_{1:T}) = \left[\prod_{t=1}^{T} p(y_t)\right]^{-1}$, et $\frac{p(A_x^t|y_t, A_y^t)}{p(A_x^t|A_y^t)} = \frac{p(x|y_t)}{p(x)}$. Cela permet de retrouver rapidement les expressions des classificateurs calculés de manière discriminante montrées précédemment. La formule (4.2) est plus générale et permet de traiter des cas plus complexes, comme nous allons le voir par la suite.

Exemple 4.1. Nous considérons la chaîne de Markov cachée d'ordre 2 (HMC2), représentée en Figure 4.1, dont la loi est :

$$p(x_{1:T}, y_{1:T}) = p(x_1)p(x_2|x_1) \prod_{t=1}^{T-2} p(x_{t+2}|x_t, x_{t+1}) \prod_{t=1}^{T} p(y_t|x_t). \tag{4.4}$$

À partir de la Remarque 4.1. et la Proposition 4.1., nous pouvons déterminer l'algorithme EFB pour en calculer les MPM :

— pour $t = 1$:

$$p(x_1|y_{1:T}) = \frac{\sum_{x_2} \alpha_2^{E,2}(x_1, x_2)\beta_2^{E,2}(x_1, x_2)}{\sum_{x_1'} \sum_{x_2} \alpha_2^{E,2}(x_1', x_2)\beta_2^{E,2}(x_1', x_2)};$$

— pour $t > 1$:

$$p(x_t|y_{1:T}) = \frac{\sum_{x_{t-1}} \alpha_t^{E,2}(x_{t-1}, x_t)\beta_t^{E,2}(x_{t-1}, x_t)}{\sum_{x_t'} \sum_{x_{t-1}} \alpha_t^{E,2}(x_{t-1}, x_t')\beta_t^{E,2}(x_{t-1}, x_t')};$$

avec $\alpha^{E,2}$ calculé de manière récursive :

$$\alpha_2^{E,2}(x_1, x_2) = p(x_1|y_1)p(x_2|x_1)\frac{p(x_2|y_2)}{p(x_2)};$$
$$\alpha_{t+1}^{E,2}(x_t, x_{t+1}) = \sum_{x_{t-1}} \alpha_t^{E,2}(x_{t-1}, x_t)p(x_{t+1}|x_{t-1}, x_t)\frac{p(x_{t+1}|y_{t+1})}{p(x_{t+1})};$$

ainsi que $\beta^{E,2}$:

$$\beta_T^{E,2}(x_{T-1}, x_T) = 1;$$
$$\beta_t^{E,2}(x_{t-1}, x_t) = \sum_{x_{t+1}} \beta_{t+1}^{E,2}(x_t, x_{t+1})p(x_{t+1}|x_{t-1}, x_t)\frac{p(x_{t+1}|y_{t+1})}{p(x_{t+1})}.$$

Selon la Proposition 4.1., il est possible de calculer le classificateur de tout modèle génératif de manière discriminante. Cela nécessite néanmoins un changement des paramètres utilisés, comme nous l'avons vu pour les cas du *Naive Bayes* et de la HMC. Ce changement de paramètres est d'importance, car il permet la gestion des *features* complexes pendant l'apprentissage.

4.3 Classificateurs calculés de manière discriminante dans les modèles basés sur le Naive Bayes

Dans cette partie, nous allons introduire des modèles probabilistes génératifs originaux étendant le modèle *Naive Bayes* et présenter le classificateur du MAP calculé de manière discriminante de chacun d'eux. L'intérêt de ces extensions est que leur classificateur calculé de manière générative aurait eu des performances très faibles dû au problème de la considération de *features*, là où la version discriminante ne connaît pas ce problème et permet d'améliorer les résultats du *Naive Bayes*.

Dans cette section, nous considérons la variable aléatoire cachée X dont la réalisation est à valeurs dans Λ_X, et les variables aléatoires observées $Y_{1:T}$, avec Y_t à valeurs dans Ω_Y.

4.3.1 Le modèle Pooled Markov Chain

Présentation du Pooled MC

Le *Pooled Markov Chain* (Pooled MC) est un modèle probabiliste génératif original inspiré du *Naive Bayes*, considérant une variable aléatoire cachée X et des

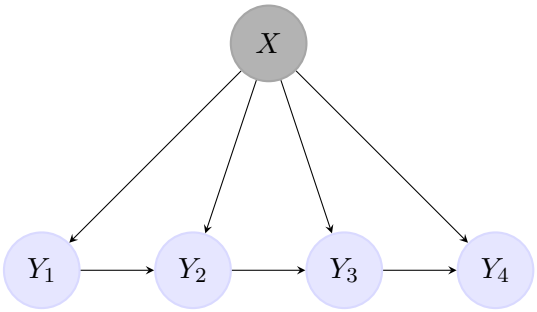


FIGURE 4.2 – Graphe probabiliste orienté du Pooled MC.

variables aléatoires observées $Y_{1:T}$. Sa loi jointe s'écrit :

$$p(x, y_{1:T}) = p(x)p(y_1|x) \prod_{t=1}^{T-1} p(y_{t+1}|x, y_t). \tag{4.5}$$

Le graphe orienté de sa loi est représenté en figure 4.2.

L'une des principales critiques du *Naive Bayes* stationnaire réside dans le fait qu'il ne se soucie pas de l'ordre des observations conditionnellement à X, comme décrit en section 2.1. Le modèle Pooled MC stationnaire corrige cela, car les variables observées ne sont plus indépendantes conditionnellement à X, suivant ici la loi d'une chaîne de Markov.

Le classificateur du MAP du Pooled MC

Nous considérons le calcul du MAP défini par la formule (1.2). Usuellement, dans la mesure où le Pooled MC est un modèle génératif, le classificateur du MAP de ce modèle est considéré comme génératif, calculant en premier lieu la loi jointe puis utilisant la formule de Bayes. Ainsi, des lois de la forme $p(y_{t+1}|x, y_t)$ seraient à apprendre, ce qui peut s'avérer très difficile à cause du problème des *features* décrit en section 1.4.3.

Nous présentons ce classificateur calculé de manière discriminante en utilisant la Proposition 4.1. En premier lieu, nous déterminons, au vu de la loi et du graphe, que $A^t = \{x, y_{t-1}\}$, donc $A^t_x = \{x\}$ et $A^t_y = \{y_{t-1}\}$. Ainsi, avec (4.2), nous avons :

$$\phi^{PooledMC}(y_{1:T}) = \arg\max_x \left[p(x|y_1) \prod_{t=1}^{T-1} \frac{p(x|y_t, y_{t+1})}{p(x|y_t)} \right], \tag{4.6}$$

qui est la formule du classificateur induit du Pooled MC calculée de manière discriminante.

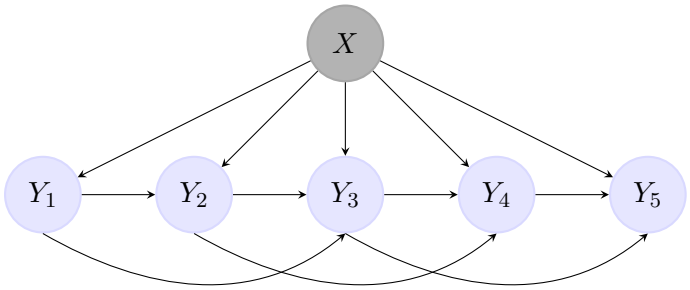


FIGURE 4.3 – Graphe probabiliste orienté du Pooled MC2.

4.3.2 Le modèle Pooled Markov Chain d'ordre 2

Présentation du Pooled MC2

Dans la même idée que précédemment, nous pouvons considérer que les variables aléatoires observées $Y_{1:T}$ suivent la loi d'une chaîne de Markov d'ordre 2 étant donné la variable cachée X. C'est ainsi que nous définissons le *Pooled Markov Chain* d'ordre 2 (Pooled MC2), qui étend les modèles précédents.

Sa loi s'écrit :

$$p(x, y_{1:T}) = p(x)p(y_1|x)p(y_2|x, y_1) \prod_{t=1}^{T-2} p(y_{t+2}|x, y_t, y_{t+1}), \tag{4.7}$$

et son graphe probabiliste orienté est donné en figure 4.3.

Le classificateur du MAP du Pooled MC2

Nous considérons toujours l'estimateur du MAP. Le classificateur calculé de manière générative du Pooled MC2 consisterait donc à calculer la loi jointe, puis utiliser la règle de Bayes, comme usuellement. Il serait encore plus impacté par le problème des *features* avec des lois telles que $p(y_{t+2}|x, y_t, y_{t+1})$ à apprendre.

Après avoir déterminé que $A^t = \{x, y_{t-2}, y_{t-1}\}, A^t_x = \{x\}, A^t_y = \{y_{t-2}, y_{t-1}\}$, le classificateur calculé de manière discriminante en utilisant la Proposition 4.1. s'écrit :

$$\phi^{PooledMC2}(y_{1:T}) = \arg\max_x \left[p(x|y_1, y_2) \prod_{t=1}^{T-2} \frac{p(x|y_t, y_{t+1}, y_{t+2})}{p(x|y_t, y_{t+1})} \right]. \tag{4.8}$$

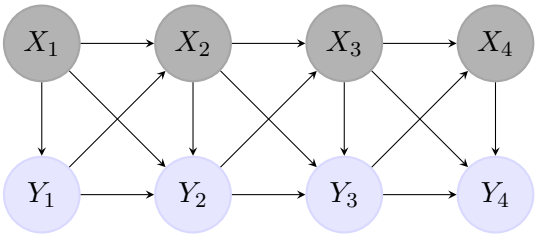


FIGURE 4.4 – Graphe probabiliste orienté de la PMC.

Comme indiqué en section 1.4.3, les formes discriminantes des classificateurs présentent plus d'intérêt car l'apprentissage des paramètres n'est pas mis à mal par les *features* des observations en cas de grande dimension.

4.4 Modèles de Markov couples et triplets

De même que pour le *Naive Bayes*, dans cette section nous allons étendre le modèle HMC et définir les classificateurs induits calculés de manière discriminante.

4.4.1 Classificateurs calculés de manière discriminante dans les modèles de Markov couples

Présentation du modèle PMC

En premier lieu, nous allons rappeler la définition de la *Pairwise Markov Chain* (PMC) [146], un modèle probabiliste considérant les variables aléatoires cachées $X_{1:T}$, avec X_t à valeurs dans Λ_X et les variables aléatoires observées $Y_{1:T}$, avec Y_t à valeurs dans Ω_Y, telles que le processus $(X_t, Y_t)_{1\leq t\leq T}$ soit une chaîne de Markov :

$$p(x_{1:T}, y_{1:T}) = p(x_1, y_1) \prod_{t=1}^{T-1} p(x_{t+1}, y_{t+1}|x_t, y_t). \tag{4.9}$$

Sa loi peut être représentée par le graphe orienté en figure 4.4.

Remarque 4.2. Pour la représentation de la PMC en figure 4.4, pour tout t, les flèches sont issues de X_t et pointent vers Y_t. Il est également possible de représenter avec les flèches pointant dans l'autre sens.

Ce modèle est une extension de la HMC, disposant de nombreux résultats théoriques [147, 148, 149, 150, 151, 152] et utilisé dans un grand nombre d'applications comme le filtrage [153, 154, 155, 156, 157, 158, 159, 160, 161] et notamment la

segmentation d'image [162, 163, 164, 165, 166] ou encore en finance [167, 168]. Dans chacune de ces applications, il est utilisé avec son classificateur calculé de manière générative, sensible à la gestion des *features* complexes, qui peut être dommageable pour certains domaines. Un exemple d'application classique de la PMC pour le NLP, dans ce cadre, est disponible en annexe A pour illustrer ce point.

Algorithme EFB pour la PMC

Proposition 4.2. Les lois marginales a posteriori d'une PMC, donnant l'estimateur du MPM, peuvent être calculées de manière discriminante. Nous avons, pour tout $t \in \{1, ..., T\}, x_t \in \Lambda_X$:

$$p(x_t|y_{1:T}) = \frac{\alpha_t^{E,PMC}(x_t)\beta_t^{E,PMC}(x_t)}{\sum\limits_{x'_t} \alpha_t^{E,PMC}(x'_t)\beta_t^{E,PMC}(x'_t)},$$

avec $\alpha^{E,PMC}$ calculé de manière récursive :

— pour $t = 1$:

$$\alpha_1^{E,PMC}(x_1) = p(x_1|y_1);$$

— pour $1 \leq t < T$:

$$\alpha_{t+1}^{E,PMC}(x_{t+1}) = \sum_{x_t} \alpha_t^{E,PMC}(x_t) \frac{p(x_t, x_{t+1}|y_t, y_{t+1})}{p(x_t|y_t)}.$$

De même pour $\beta^{E,PMC}$:

— pour $t = T$:

$$\beta_T^{E,PMC}(x_T) = 1;$$

— pour $1 \leq t < T$:

$$\beta_t^{E,PMC}(x_t) = \sum_{x_{t+1}} \beta_{t+1}^{E,PMC}(x_{t+1}) \frac{p(x_t, x_{t+1}|y_t, y_{t+1})}{p(x_t|y_t)}.$$

Démonstration La Proposition 4.1. permet de démontrer l'existence du classificateur calculé de manière discriminante du MPM, et également de spécifier l'algorithme EFB ci-dessus. Néanmoins, nous allons présenter une autre démonstration, ne reposant pas sur celle-ci, et demeurant plus concise :

$$p(x_t|y_{1:T}) = \frac{p(x_t, y_{1:T})}{\sum\limits_{x'_t} p(x'_t, y_{1:T})} = \frac{p(x_t, y_{1:t})p(y_{t+1:T}|x_t, y_t)}{\sum\limits_{x'_t} p(x'_t, y_{1:t})p(y_{t+1:T}|x'_t, y_t)}.$$

Or,

$$\begin{aligned}
p(x_1, y_1) &= p(y_1)p(x_1|y_1) \\
p(x_{t+1}, y_{1:t+1}) &= \sum_{x_t} p(x_t, y_{1:t})p(x_{t+1}|x_t, y_t)p(y_{t+1}|x_t, y_t, x_{t+1}) \\
&= \sum_{x_t} p(y_{1:t})p(x_t|y_{1:t})p(x_{t+1}|x_t, y_t)\frac{p(y_t, y_{t+1})p(x_t, x_{t+1}|y_t, y_{t+1})}{p(y_t)p(x_t|y_t)p(x_{t+1}|x_t, y_t)} \\
&= p(y_{1:t})p(y_{t+1}|y_t)\sum_{x_t} p(x_t|y_{1:t})\frac{p(x_t, x_{t+1}|y_t, y_{t+1})}{p(x_t|y_t)},
\end{aligned}$$

et :

$$\begin{aligned}
p(y_{t+1:T}|x_t, y_t) &= \sum_{x_{t+1}} p(x_{t+1}|x_t, y_t)p(y_{t+1}|x_t, y_t, x_{t+1})p(y_{t+2:T}|x_{t+1}, y_{t+1}) \\
&= \sum_{x_{t+1}} \frac{p(y_t, y_{t+1})p(x_t, x_{t+1}|y_t, y_{t+1})}{p(y_t)p(x_t|y_t)}\frac{p(y_{t+1:T})p(x_{t+1}|y_{t+1:T})}{p(y_{t+1})p(x_{t+1}|y_{t+1})} \\
&= p(y_{t+1}|y_t)p(y_{t+2:T}|y_{t+1})\sum_{x_{t+1}} \frac{p(x_t, x_{t+1}|y_t, y_{t+1})}{p(x_t|y_t)}\frac{p(x_{t+1}|y_{t+1:T})}{p(x_{t+1}|y_{t+1})}.
\end{aligned}$$

Par conséquent, nous retrouvons bien l'algorithme EFB de la PMC en posant :

$$\begin{aligned}
\alpha_t^{E,PMC}(x_t) &= p(x_t|y_{1:t}); \\
\beta_{t+1}^{E,PMC}(x_{t+1}) &= \frac{p(x_{t+1}|y_{t+1:T})}{p(x_{t+1}|y_{t+1})},
\end{aligned}$$

achevant la démonstration.

4.4.2 Algorithme Entropic Forward-Backward pour des cas particuliers de la chaîne de Markov couple

Le modèle HMC+

À partir de l'algorithme EFB de la PMC, il est possible de définir son expression pour des cas particuliers de la chaîne de Markov couple. C'est en effet le cas pour le modèle que nous dénommons HMC+, consistant à ajouter une dépendance directe entre la variable cachée au temps t et la variable observée au temps $t+1$ dans le modèle HMC classique. Sa loi s'écrit :

$$p(x_{1:T}, y_{1:T}) = p(x_1)p(y_1|x_1)\prod_{t=1}^{T-1} p(x_{t+1}|x_t)p(y_{t+1}|x_t, x_{t+1}),$$

et son graphe orienté est représenté en figure 4.5.

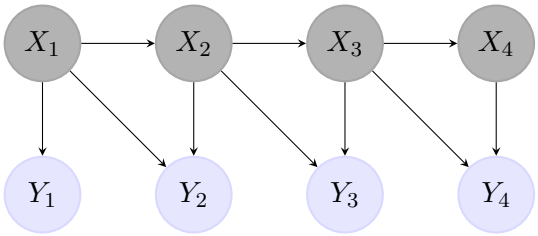


FIGURE 4.5 – Graphe probabiliste orienté de la HMC+.

Remarque 4.3. Dans le cadre de la HMC+, nous pouvons remarquer que :

$$\begin{aligned}\frac{p(x_t, x_{t+1}|y_t, y_{t+1})}{p(x_t|y_t)} &= \frac{p(x_t, y_t)p(x_{t+1}|x_t)p(y_{t+1}|x_t, x_{t+1})p(y_t)}{p(y_t, y_{t+1})p(x_t, y_t)} \\ &= \frac{p(y_t)p(y_{t+1})}{p(y_t, y_{t+1})}\frac{p(x_t, x_{t+1}|y_{t+1})}{p(x_t)},\end{aligned}$$

nous permettant d'émettre le corollaire suivant :

Corollaire 4.1. L'algorithme EFB de la HMC+ consiste à calculer, pour tout $t \in \{1, ..., T\}, x_t \in \Lambda_X$:

$$p(x_t|y_{1:T}) = \frac{\alpha_t^{E,+}(x_t)\beta_t^{E,+}(x_t)}{\sum\limits_{x'_t} \alpha_t^{E,+}(x'_t)\beta_t^{E,+}(x'_t)},$$

avec $\alpha^{E,+}$ calculé de manière récursive :

$$\alpha_1^{E,+}(x_1) = p(x_1|y_1); \qquad \alpha_{t+1}^{E,+}(x_{t+1}) = \sum_{x_t} \alpha_t^{E,+}(x_t)\frac{p(x_t, x_{t+1}|y_{t+1})}{p(x_t)}.$$

De même pour $\beta^{E,+}$:

$$\beta_T^{E,+}(x_T) = 1; \qquad \beta_t^{E,+}(x_t) = \sum_{x_{t+1}} \beta_t^{E,+}(x_{t+1})\frac{p(x_t, x_{t+1}|y_{t+1})}{p(x_t)}.$$

Le modèle HMC-CN

Nous appliquons également cela à un autre modèle, que nous appelons HMC-CN (pour *Hidden Markov Chain with Complexified Noise*), rajoutant une nouvelle dépendance entre la variable observée au temps t et la variable cachée au temps $t+1$ par rapport à la HMC+. Sa loi s'écrit :

$$p(x_{1:T}, y_{1:T}) = p(x_1)p(y_1|x_1)\prod_{t=1}^{T-1} p(x_{t+1}|x_t, y_t)p(y_{t+1}|x_t, x_{t+1})$$

Et son graphe orienté est représenté en figure 4.6.

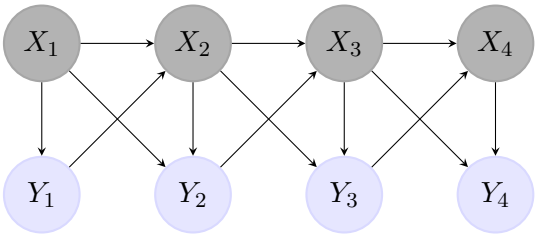


FIGURE 4.6 – Graphe probabiliste orienté de la HMC-CN.

Remarque 4.4. Dans le cadre de la HMC-CN, nous pouvons remarquer que :

$$\begin{aligned}\frac{p(x_t, x_{t+1}|y_t, y_{t+1})}{p(x_t|y_t)} &= \frac{p(x_t, y_t)p(x_{t+1}|x_t, y_t)p(y_{t+1}|x_t, x_{t+1})p(y_t)}{p(y_t, y_{t+1})p(x_t, y_t)} \\ &= \frac{p(y_t)p(y_{t+1})}{p(y_t, y_{t+1})}p(x_{t+1}|x_t, y_t)\frac{p(x_t, x_{t+1}|y_{t+1})}{p(x_t, x_{t+1})},\end{aligned}$$

nous permettant d'émettre le corollaire suivant :

Corollaire 4.2. L'algorithme EFB de la HMC-CN calcule, pour tout $t \in \{1, ..., T\}, x_t \in \Lambda_X$:

$$p(x_t|y_{1:T}) = \frac{\alpha_t^{E,CN}(x_t)\beta_t^{E,CN}(x_t)}{\sum\limits_{x'_t} \alpha_t^{E,CN}(x'_t)\beta_t^{E,CN}(x'_t)},$$

avec $\alpha^{E,CN}$ calculé de manière récursive :

$$\alpha_1^{E,CN}(x_1) = p(x_1|y_1); \quad \alpha_{t+1}^{E,CN}(x_{t+1}) = \sum_{x_t} \alpha_t^{E,CN}(x_t)p(x_{t+1}|x_t, y_t)\frac{p(x_t, x_{t+1}|y_{t+1})}{p(x_t, x_{t+1})}.$$

De même pour $\beta^{E,CN}$:

$$\beta_T^{E,CN}(x_T) = 1; \quad \beta_t^{E,CN}(x_t) = \sum_{x_{t+1}} \beta_{t+1}^{E,CN}(x_{t+1})p(x_{t+1}|x_t, y_t)\frac{p(x_t, x_{t+1}|y_{t+1})}{p(x_t, x_{t+1})}.$$

4.4.3 Classificateurs calculés de manière discriminante dans les modèles de Markov triplets

Présentation du modèle TMC

À leur tour, les PMC ont été étendues aux chaînes de Markov triplets (TMC pour *Triplet Markov Chain*) [169, 170, 171, 172], dont l'intérêt repose notamment

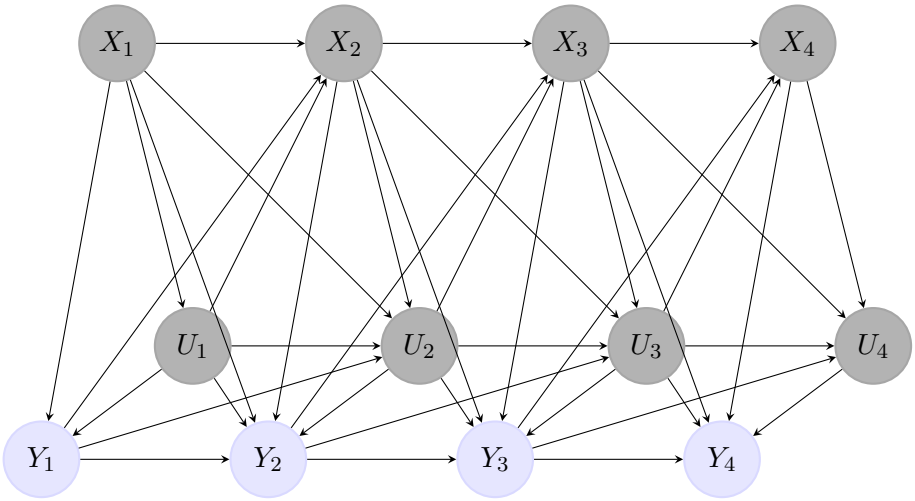


FIGURE 4.7 – Graphe probabiliste orienté de la TMC.

sur le fait qu'ils permettent de modéliser des PMC non-stationnaires. Elles sont également appliquées pour de nombreuses applications [173, 174], telles qu'en segmentation d'image [175, 176], filtrage [177], ou finance [178]. Une TMC considère un processus caché $X_{1:T}$, avec X_t à valeurs dans Λ_X, un processus observé $Y_{1:T}, Y_t$ à valeurs dans Ω_Y, ainsi qu'un troisième processus caché $U_{1:T}, U_t$ à valeurs dans Λ_U. Il ne suppose plus la markovianité du processus $(X_t, Y_t)_{1\leq t\leq T}$ comme précédemment, mais celle du triplet $(X_t, U_t, Y_t)_{1\leq t\leq T}$. La loi jointe de la TMC s'écrit :

$$p(x_{1:T}, u_{1:T}, y_{1:T}) = p(x_1, u_1, y_1) \prod_{t=1}^{T} p(x_{t+1}, u_{t+1}, y_{t+1}|x_t, u_t, y_t). \tag{4.10}$$

Le graphe orienté de la TMC peut être représenté en figure 4.7.

Algorithme EFB pour la TMC

Proposition 4.3. De manière similaire à la PMC, nous pouvons définir l'algorithme EFB pour la TMC permettant de définir le classificateur de Bayes du MPM, pour tout $t \in \{1, ..., T\}, x_t \in \Lambda_X$:

$$p(x_t|y_{1:T}) = \frac{\sum_{u_t} \alpha_t^{E,TMC}(x_t, u_t)\beta_t^{E,TMC}(x_t, u_t)}{\sum_{x'_t}\sum_{u_t} \alpha_t^{E,TMC}(x'_t, u_t)\beta_t^{E,TMC}(x'_t, u_t)},$$

avec $\alpha^{E,TMC}$ calculé de manière récursive :

— pour $t = 1$:

$$\alpha_1^{E,TMC}(x_1, u_1) = p(x_1, u_1|y_1);$$

— pour $1 \leq t < T$:

$$\alpha_{t+1}^{E,TMC}(x_{t+1}, u_{t+1}) = \sum_{x_t, u_t} \alpha_t^{E,TMC}(x_t, u_t) \frac{p(x_t, u_t, x_{t+1}, u_{t+1}|y_t, y_{t+1})}{p(x_t, u_t|y_t)}.$$

De même pour $\beta^{E,TMC}$:

— pour $t = T$:

$$\beta_T^{E,TMC}(x_T, u_T) = 1;$$

— pour $1 \leq t < T$:

$$\beta_t^{E,TMC}(x_t, u_t) = \sum_{x_{t+1}, u_{t+1}} \beta_{t+1}^{E,TMC}(x_{t+1}, u_{t+1}) \frac{p(x_t, u_t, x_{t+1}, u_{t+1}|y_t, y_{t+1})}{p(x_t, u_t|y_t)}.$$

Démonstration La démonstration est identique à celle de la PMC, mais en considérant (x_t, u_t) comme variables cachées et non seulement x_t, puis en sommant sur les u_t pour ne considérer que la loi a posteriori de x_t.

Ainsi, nous avons défini les algorithmes EFB pour des extensions de la HMC permettant de calculer l'estimateur du MPM de manière discriminante.

4.5 Équivalence entre des extensions du linear-chain CRF et les extensions de la HMC

D'une part, nous avons étudié les PMC et les TMC, qui étendent les HMC. D'autre part, nous avons prouvé que les *linear-chain* CRF sont équivalents aux HMC. Nous nous intéressons donc à l'équivalence entre les PMC et les TMC avec les CRF inspirés par leurs graphes probabilistes respectifs. Nous proposons ainsi deux résultats :

— l'extension des *linear-chain* CRF aux *linear-chain Pairwise Conditional Random Fields*, et nous montrons l'équivalence entre ce nouveau modèle et la PMC ;

— nous faisons de même en étendant de nouveau ce modèle aux *linear-chain Triplet Conditional Random Fields*, et nous montrons l'équivalence avec la TMC.

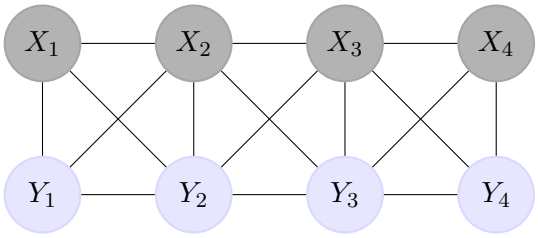


FIGURE 4.8 – Graphe probabiliste de la *linear-chain* PCRF.

4.5.1 Linear-chain Pairwise Conditional Random Field

Nous introduisons l'équivalent CRF de la PMC, que nous dénommons *linear-chain Pairwise Conditional Random Field*, abrégé *linear-chain* PCRF. Il considère les variables aléatoires $X_{1:T}$, X_t à valeurs dans Λ_X, et $Y_{1:T}$, Y_t à valeurs dans Ω_Y, et est défini par la loi a posteriori suivante :

$$p(x_{1:T}|y_{1:T}) = \frac{\prod_{t=1}^{T-1} \phi_t(x_t, x_{t+1}, y_t, y_{t+1})}{\sum_{x'_{1:T}} \prod_{t=1}^{T-1} \phi_t(x'_t, x'_{t+1}, y_t, y_{t+1})}, \tag{4.11}$$

et dont le graphe probabiliste est représenté en figure 4.8. Ce modèle est bien est extension de la *linear-chain* CRF de par ses cliques plus générales.

Proposition 4.4. Soit $(X_t, Y_t)_{1\leq t\leq T}$ un processus stochastique dont la loi a posteriori est un *linear-chain* PCRF (4.11). Alors (4.11) est la loi a posteriori d'une PMC avec $p(x_{1:T}, y_{1:T})$ définie ainsi, en posant $Z_t = (X_t, Y_t)$:

$$p(z_1) = \frac{\beta_1(z_1)}{\sum_{z'_1} \beta_1(z'_1)}; \tag{4.12}$$

$$p(z_{t+1}|z_t) = \frac{\phi_t(z_t, z_{t+1})\beta_{t+1}(z_{t+1})}{\beta_t(z_t)}; \tag{4.13}$$

avec :

$$\begin{gathered} \beta_T(z_T) = 1, \\ \text{et pour } t = T-1, ..., 2 : \beta_t(z_t) = \sum_{z_{t+1}} \phi_t(z_t, z_{t+1})\beta_{t+1}(z_{t+1}). \end{gathered} \tag{4.14}$$

Démonstration La loi jointe du modèle défini avec (4.12) et (4.13) est :

$$p(z_{1:T}) = p(z_1) \prod_{t=1}^{T-1} p(z_{t+1}|z_t)$$

$$= \frac{\beta_1(z_1)}{\sum_{z'_1} \beta_1(z'_1)} \prod_{t=1}^{T-1} \frac{\phi_t(z_t, z_{t+1})\beta_{t+1}(z_{t+1})}{\beta_t(z_t)}$$

$$= \frac{1}{\sum_{z'_1} \beta_1(z'_1)} \prod_{t=1}^{T-1} \phi_t(z_t, z_{t+1})$$

Donc ce modèle a bien la même loi a posteriori que le *linear-chain* PCRF. De plus, dans la mesure où :

$$p(z_{1:T}) \propto \prod_{t=1}^{T-1} \phi_t(z_t, z_{t+1})$$

alors $Z_{1:T}$, en utilisant le Lemme 3.1. énoncé en section 3.4.2, est bien une chaîne de Markov, prouvant que le modèle défini avec (4.12) et (4.13) est bien une PMC, achevant la démonstration.

Remarque 4.5. Soit $(X_t, Y_t)_{1 \leq t \leq T}$ un processus stochastique dont la loi jointe est une PMC (4.9). Sa loi a posteriori peut définir celle d'un *linear-chain* PCRF (4.11) en posant :

$$\phi_1(x_1, x_2, y_1, y_2) = p(x_1, y_1)p(x_2, y_2|x_1, y_1),$$
$$\text{pour tout } t, 1 < t \leq T-1, \phi_t(x_t, x_{t+1}, y_t, y_{t+1}) = p(x_{t+1}, y_{t+1}|x_t, y_t).$$

Ainsi, nous avons introduit le modèle *linear-chain* PCRF, étendant le *linear-chain* CRF, et nous avons montré son équivalence avec le modèle PMC. Nous pouvons donc espérer les mêmes améliorations qu'a amenées l'utilisation de la PMC à la place de la HMC en utilisant un *linear-chain* PCRF à la place d'un *linear-chain* CRF.

4.5.2 Linear-chain Triplet Conditional Random Field

Après avoir introduit l'équivalent CRF de la PMC, nous faisons de même pour la TMC, que nous dénommons le *linear-chain Triplet Conditional Random Field*, abrégé *linear-chain* TCRF. Il considère les variables aléatoires $X_{1:T}$, X_t à valeurs dans Λ_X, $Y_{1:T}$, Y_t à valeurs dans Ω_Y, et $U_{1:T}$, U_t à valeurs dans Λ_U, est défini par la loi a posteriori suivante :

$$p(x_{1:T}, u_{1:T}|y_{1:T}) = \frac{\prod_{t=1}^{T-1} \phi_t(x_t, x_{t+1}, u_t, u_{t+1}, y_t, y_{t+1})}{\sum_{x'_{1:T}, u'_{1:T}} \prod_{t=1}^{T-1} \phi_t(x'_t, x'_{t+1}, u'_t, u'_{t+1}, y_t, y_{t+1})}, \tag{4.15}$$

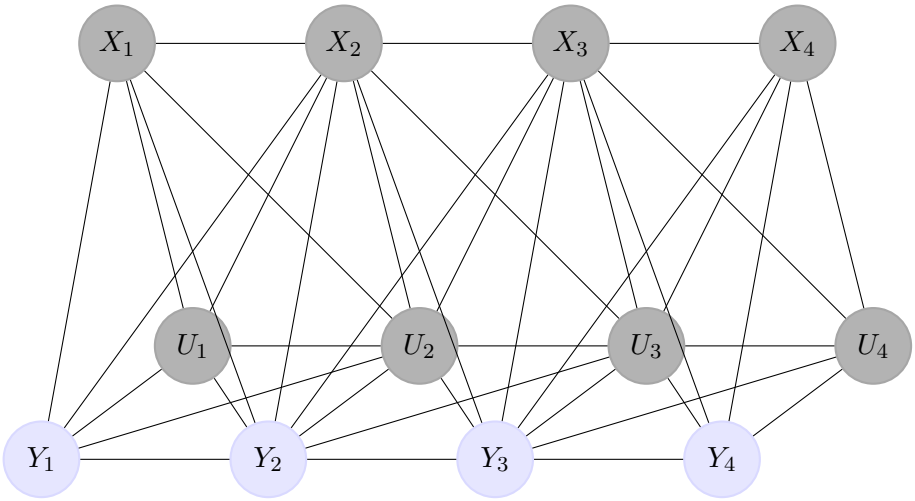


FIGURE 4.9 – Graphe probabiliste de la *linear-chain* TCRF.

et dont le graphe probabiliste est représenté en figure 4.9. Ce modèle se veut très général car $U_{1:T}$ est quelconque. Il peut, par exemple, modéliser la non-stationnarité du modèle couple. Par ailleurs, les *linear-chain* TCRF sont plus généraux que tous les *linear-chain* CRF classiques car $X_{1:T}$ n'est plus de Markov conditionnellement à $Y_{1:T}$, c'est une marginale d'un Markov $(U_t, X_t)_{1\leq t\leq T}$.

Proposition 4.5. Soit $(X_t, U_t, Y_t)_{1\leq t\leq T}$ un processus stochastique dont la loi a posteriori est un *linear-chain* TCRF (4.15). Alors (4.15) est la loi a posteriori d'une TMC avec $p(x_{1:T}, u_{1:T}, y_{1:T})$ définie ainsi, en posant $Z_t = (X_t, U_t, Y_t)$:

$$p(z_1) = \frac{\beta_1(z_1)}{\sum_{z_1'} \beta_1(z_1')}; \tag{4.16}$$

$$p(z_{t+1}|z_t) = \frac{\phi_t(z_t, z_{t+1})\beta_{t+1}(z_{t+1})}{\beta_t(z_t)} \tag{4.17}$$

avec :

$$\begin{gathered} \beta_T(z_T) = 1, \\ \text{et pour } t = T-1, ..., 2 : \beta_t(z_t) = \sum_{z_{t+1}} \phi_t(z_t, z_{t+1})\beta_{t+1}(z_{t+1}). \end{gathered} \tag{4.18}$$

Démonstration La démonstration est similaire à celle de la Proposition 4.4., en considérant $Z_t = (X_t, U_t, Y_t)$.

Remarque 4.6. Soit $(X_t, U_t, Y_t)_{1\leq t\leq T}$ un processus stochastique dont la loi jointe est une TMC (4.10). Sa loi a posteriori peut définir celle d'un *linear-chain* TCRF (4.15) en posant :

$$\begin{aligned}\phi_1(x_1, x_2, u_1, u_2, y_1, y_2) &= p(x_1, u_1, y_1)p(x_2, u_2, y_2|x_1, u_1, y_1),\\ &\text{pour tout } t, 1 < t \leq T-1,\\ \phi_t(x_t, x_{t+1}, u_t, u_{t+1}, y_t, y_{t+1}) &= p(x_{t+1}, u_{t+1}, y_{t+1}|x_t, u_t, y_t).\end{aligned}$$

Ainsi, nous avons également montré l'équivalence entre la *linear-chain* TCRF et la TMC, prouvant que ces champs de Markov conditionnels ne sont pas plus généraux.

4.6 Conclusion

La Proposition 4.1. a permis de démontrer qu'il est possible de calculer le classificateur induit d'un modèle génératif de manière discriminante, présentant l'avantage de ne pas se soucier de la loi des observations. Nous avons par la suite passé en revue un certain nombre de modèles sur lesquels nous avons appliqué cette proposition.

Ces divers résultats viennent confirmer le fait que la catégorisation des classificateurs avec les adjectifs « générative » et « discriminante » n'est pas justifiée, dans la mesure où tout classificateur peut être considéré comme discriminant. Cette proposition vient remettre en question les comparaisons usuelles entre modèles probabilistes basés sur la nature de leur classificateur dans le cadre supervisé, comme cela a pu être le cas entre la Régression Logistique et le *Naive Bayes* ou la HMC et le MEMM, dans la mesure où utiliser le classificateur calculé de manière générative pour l'un d'eux n'est pas légitime.

De plus, nous avons montré des équivalences entre des CRF et des extensions de la chaîne de Markov cachée. Une perspective intéressante consisterait à approfondir plus en détails les équivalences entre des CRF et des modèles génératifs, comme entre des extensions du *linear-chain CRF* [179, 45, 180, 181, 182, 183, 184] et leurs penchants génératifs.

Chapitre 5

Classificateurs induits de modèles probabilistes génératifs paramétrés avec des réseaux de neurones

Les réseaux de neurones sont aujourd'hui parmi les modèles les plus performants pour le *Machine Learning*, atteignant l'état-de-l'art dans un grand nombre de tâches, notamment en NLP. Les modèles ne reposant pas dessus ont souvent été mis de côté ces dernières années tant leurs performances sont satisfaisantes. Les modèles probabilistes génératifs, souffrant en plus de la difficulté de prendre en compte les *features* arbitraires, ne font pas exception.

Nous considérons un modèle probabiliste génératif. En vertu de la nouvelle manière de calculer son classificateur induit de manière discriminante décrite en section 4.2, le problème de considération des *features* peut être contourné grâce à l'utilisation d'un nouveau jeu de paramètres, comme nous l'avons remarqué pour le *Naive Bayes* et la HMC. Ainsi, le problème d'estimation des paramètres, qui pouvait devenir impossible à gérer avec le classificateur calculé de manière générative, devient considérablement modifié en calculant ce dernier de manière discriminante et avec un nouveau jeu de paramètres.

Au cours de ce chapitre, nous estimons ce nouveau jeu de paramètres avec des réseaux de neurones, profitant de leurs propriétés permettant d'approcher n'importe quelle fonction. Ainsi, l'estimation des paramètres de notre classificateur est effectuée par descente de gradient, et les problématiques usuelles spécifiques aux modèles génératifs lors de cette étape ne sont plus rencontrées. Cela permet donc d'estimer le classificateur induit d'un modèle probabiliste avec des réseaux de neurones et d'effectuer des classifications telles que décrites tout au long de ce

manuscrit.

Dans cette partie, nous allons étudier ce procédé avec les modèles présentés tout au long de cette thèse, basés sur le *Naive Bayes* et la HMC, et étudier leurs apports empiriques. Cela ouvre la voie à de nouveaux modèles combinant les modèles probabilistes génératifs et les réseaux de neurones pour la classification avec données supervisées.

5.1 Classificateurs des modèles basés sur le Naive Bayes définis par des réseaux de neurones

Dans les chapitres précédents, nous avons présenté les modèles *Naive Bayes* (2.1), Pooled MC (4.5), et Pooled MC2 (4.7), ainsi que comment calculer leur classificateur de manière discriminante. Dans cette section, nous allons étudier une méthode permettant d'estimer ces classificateurs avec des fonctions neuronales. Cela permet ainsi de proposer de nouveaux modèles neuronaux que nous allons appliquer à l'analyse de sentiments sur divers jeux de données

5.1.1 Neural Naive Bayes

Nous considérons le modèle *Naive Bayes* stationnaire, avec X prenant ses valeurs dans $\Lambda_X = \{\lambda_1, ..., \lambda_N\}$, et les variables aléatoires observés $Y_{1:T}$, avec $\Omega_Y = \mathbb{R}^d$. Nous développons le calcul du classificateur calculé de manière discriminante du *Naive Bayes* (2.4), où nous supposons que les fonctions π et L, introduites en section 2.1 et 2.2, sont strictement positives :

$$\begin{aligned}
\phi^{NB}(y_{1:T}) &= \arg\max_{\lambda_i \in \Lambda_X} \left(\frac{\pi(i)^{1-T} \prod_{t=1}^{T} L_{y_t}(i)}{\sum_{j=1}^{N} \pi(j)^{1-T} \prod_{t=1}^{T} L_{y_t}(j)} \right) \\
&= \arg\max_{\lambda_i \in \Lambda_X} \text{softmax} \left(\log \left(\pi(i) \prod_{t=1}^{T} \frac{L_{y_t}(i)}{\pi(i)} \right) \right) \\
&= \arg\max_{\lambda_i \in \Lambda_X} \text{softmax} \left(\log(\pi(i)) + \sum_{t=1}^{T} \log \left(\frac{L_{y_t}(i)}{\pi(i)} \right) \right) \qquad (5.1)
\end{aligned}$$

Nous supposons que, pour tout $i \in \{1, ..., N\}, \log(\pi(i))$ est négligeable. Cela est notamment le cas lorsque les données sont équilibrées entre les différents labels, c'est-à-dire lorsque, pour tout $i, \pi(i) \approx \frac{1}{N}$. C'est également le cas lorsque T est suffisamment grand.

Soit NN^{NB} une fonction neuronale ayant pour *input* un vecteur de taille d et pour *output* un vecteur de taille N estimant les paramètres du *Naive Bayes* comme ci-après :

$$NN^{NB}(y_t)_i = \log\left(\frac{L_{y_t}(i)}{\pi(i)}\right)$$

Par conséquent, (5.1) est estimé ainsi :

$$\phi^{NB}(y_{1:T}) \approx \arg\max_{\lambda_i \in \Lambda_X} \text{softmax}\left(\sum_{t=1}^{T} NN^{NB}(y_t)_i\right) \tag{5.2}$$

Dans la mesure où (5.2) est le classificateur du *Naive Bayes* estimé uniquement avec des fonctions neuronales, nous le nommons *Neural Naive Bayes* (Neural NB).

5.1.2 Neural Pooled Markov Chains

Nous considérons à présent le modèle Pooled MC stationnaire. Nous faisons les mêmes hypothèses que précédemment concernant les variables aléatoires, et nous supposons que toutes les probabilités impliquées dans le calcul de (4.6) sont strictement positives. Ainsi, le classificateur calculé de manière discriminante du Pooled MC peut s'écrire de la manière suivante :

$$\phi^{PooledMC}(y_{1:T}) = \arg\max_{x \in \Lambda_X} \text{softmax}\left(\log\left(p(x|y_1)\right) + \sum_{t=1}^{T-1} \log\left(\frac{p(x|y_t, y_{t+1})}{p(x|y_t)}\right)\right). \tag{5.3}$$

Dans la même idée que précédemment, nous supposons $\log\left(p(x|y_1)\right)$ négligeable quel que soit x et y_1, et introduisons NN^{MC} une fonction neuronale ayant comme *input* y_t et y_{t+1} et un *output* de taille N estimant les fonctions suivantes :

$$NN^{MC}(y_t, y_{t+1})_i = \log\left(\frac{P(X = \lambda_i | Y_t = y_t, Y_{t+1} = y_{t+1})}{P(X = \lambda_i | Y_t = y_t)}\right)$$

Par conséquent, nous estimons le classificateur du Pooled MC ainsi, que nous nommons *Neural Pooled Markov Chain* (Neural Pooled MC) :

$$\phi^{PooledMC}(y_{1:T}) \approx \arg\max_{\lambda_i \in \Lambda_X} \text{softmax}\left(\sum_{t=1}^{T-1} NN^{MC}(y_t, y_{t+1})_i\right) \tag{5.4}$$

IMDB		
	FastText	ExtVec
Naive Bayes « génératif »	27.96%	25.28%
Neural NB	$12.46\% \pm 0.19$	$13.40\% \pm 0.34$
Neural Pooled MC	$11.37\% \pm 0.24$	$12.68\% \pm 0.17$
Neural Pooled MC2	$\mathbf{11.01}\% \pm 0.05$	$\mathbf{12.34}\% \pm 0.14$

TABLE 5.1 – Taux d'erreur du classificateur usuel du *Naive Bayes* et des nouveaux modèles neuronaux sur le *dataset* IMDB avec les *embedding* FastText et ExtVec.

SST-2		
	FastText	ExtVec
Naive Bayes « génératif »	38.00%	38.00%
Neural NB	$15.32\% \pm 0.20$	$17.22\% \pm 0.22$
Neural Pooled MC	$14.50\% \pm 0.17$	$16.14\% \pm 0.29$
Neural Pooled MC2	$\mathbf{14.10}\% \pm 0.26$	$\mathbf{15.88}\% \pm 0.14$

TABLE 5.2 – Taux d'erreur du classificateur usuel du *Naive Bayes* et des nouveaux modèles neuronaux sur le *dataset* SST-2 avec les *embedding* FastText et ExtVec.

Dans le même ordre d'idée, en suivant le même raisonnement, nous introduisons le *Neural Pooled Markov Chain* d'ordre 2 (Neural Pooled MC2) à partir de (4.8) :

$$\phi^{PooledMC2}(y_{1:T}) \approx \arg\max_{\lambda_i \in \Lambda_X} \text{softmax}\left(\sum_{t=1}^{T-2} NN^{MC2}(y_t, y_{t+1}, y_{t+2})_i\right) \tag{5.5}$$

avec NN^{MC2} une fonction neuronale ayant la concaténation de y_t, y_{t+1}, et y_{t+2} comme *input*, et comme *output* un vecteur de taille N.

Pour aller plus loin, pour tout $k \in \mathbb{N}^*$, nous introduisons le *Neural Pooled Markov Chain* d'ordre k (Neural Pooled MC(k)) :

$$\phi^{MCk}(y_{1:T}) \approx \arg\max_{\lambda_i \in \Lambda_X} \text{softmax}\left(\sum_{t=1}^{T-k} NN^{MCk}(y_t, y_{t+1}, ..., y_{t+k})_i\right) \tag{5.6}$$

avec NN^{MCk} une fonction neuronale ayant pour *input* la concaténation de y_t à y_{t+k}, et un vecteur d'*output* de taille N.

5.1.3 Application pour l'analyse de sentiments

Dans cette partie, nous allons appliquer les modèles Neural NB, Neural Pooled MC, et Neural Pooled MC2 pour l'analyse de sentiments. Nous allons également

TweetEval - Emotion		
	FastText	ExtVec
Naive Bayes « génératif »	61.91	60.18
Neural NB	71.15 ± 0.16	68.17 ± 0.37
Neural Pooled MC	71.32 ± 0.20	69.07 ± 0.46
Neural Pooled MC2	$\mathbf{72.00} \pm 0.28$	$\mathbf{69.28} \pm 0.45$

TABLE 5.3 – Scores F_1 du classificateur usuel du *Naive Bayes* et des nouveaux modèles neuronaux sur le *dataset* TweetEval - Emotion avec les *embedding* FastText et ExtVec.

FPB		
	FastText	ExtVec
Naive Bayes « génératif »	78.48 ± 0.94	80.22 ± 0.84
Neural NB	82.87 ± 0.53	82.60 ± 0.58
Neural Pooled MC	84.61 ± 0.91	85.01 ± 0.77
Neural Pooled MC2	$\mathbf{86.02} \pm 0.49$	$\mathbf{85.14} \pm 1.06$

TABLE 5.4 – Scores F_1 du classificateur usuel du *Naive Bayes* et des nouveaux modèles neuronaux sur le *dataset* Financial PhraseBank avec les *embedding* FastText et ExtVec.

appliquer le classificateur usuel du *Naive Bayes*, calculé de manière générative (2.3), à titre de comparaison.

Nous utilisons deux méthodes d'*embedding* différentes : FastText [56] et ExtVec [89], permettant de convertir chaque mot en un vecteur de taille 300. Par conséquent, $\Omega_Y = \mathbb{R}^{300}$ pour ces expériences. Concernant les fonctions neuronales, NN^{NB}, NN^{MC}, et NN^{MC2}, chacune d'elle est un FNN avec la taille d'*input* adaptée, une couche cachée de taille 64, et une couche de sortie de taille N. Les fonctions d'activation utilisées sont des ReLU.

Concernant l'apprentissage des paramètres, NN^{NB}, NN^{MC}, et NN^{MC2} sont estimées avec l'algorithme de descente de gradient stochastique avec la méthode d'optimisation Adam [53]. Nous minimisons comme fonction de perte la *cross-entropy* avec un taux d'apprentissage de 0.001 et un mini-batch de taille 64. Pour chaque expérience, nous entraînons un modèle sur 10 époques, et conservons celui atteignant la meilleure performance sur le jeu de validation pour le test. Dans la mesure où l'algorithme d'apprentissage utilisé est stochastique, chaque expérience est réalisée cinq fois, et nous reportons la moyenne et l'intervalle de confiance gaus-

sien à 95%. Concernant le classificateur usuel du *Naive Bayes* (2.3), les paramètres π et b, introduit en section 2.1, sont estimés par maximum de vraisemblance, avec le paramètre b modélisé par une loi gaussienne.

Nous appliquons les modèles aux *datasets* IMDB, SST-2, TweetEval - Emotion, et FPB, dont les résultats sont disponibles en Tables 5.1, 5.2, 5.3, et 5.4, respectivement. Pour rappel, nous reportons le taux d'erreur pour les *datasets* IMDB et SST-2, et les scores F_1 pour TweetEval - Emotion et FPB. Le jeu de test pour FPB est également tiré de manière aléatoire, donc nous effectuons plusieurs expériences, une dizaine, pour ce jeu de donnée également avec le classificateur usuel du *Naive Bayes*.

D'une part, nous pouvons observer que les modèles neuronaux permettent une amélioration des résultats par rapport au classificateur calculé de manière générative du *Naive Bayes*, qui est pourtant celui utilisé usuellement. De plus, nous observons une amélioration des résultats plus le modèle neuronal est général. Cela est attendu, dans la mesure où le Pooled MC d'ordre k est un cas particulier du Pooled MC d'ordre $k+1$, montrant ainsi empiriquement l'impact de supposer une dépendance plus complexe des variables observées. Ainsi, l'erreur se réduit d'environ 10% en moyenne en utilisant le Neural Pooled MC2 plutôt que le Neural NB. Nous avons également réalisé ces expériences avec le Neural Pooled MC(k), pour $k \in \{3, 5, 7, 10\}$. Ces modèles ont des résultats équivalent au Neural Pooled MC2, montrant les limites de ces extensions.

Nous comparons également les résultats de nos modèles avec les modèles populaires. En premier lieu, il est important de préciser que chacun de nos modèles est très léger, avec des temps d'entraînement d'environ 5 minutes et d'inférence d'environ 1.5 milliseconde. Par conséquent, la comparaison n'est pertinente qu'avec des modèles légers, et non avec des modèles « lourds » possédant des millions de paramètres et nécessitant une très grande quantité de données pour s'entraîner tels que ceux basés sur le Transformer [65]. La légèreté de nos modèles est d'importance, notamment pour des besoins industriels. À propos de IMDB, nos modèles sont largement meilleurs que le SVM combiné avec TF-IDF, les réseaux de neurones convolutifs [185], et même l'algorithme FastText classique pour la classification de textes [186], atteignant des taux d'erreurs de 59.5%, 62.5%, et 54.8%, respectivement. Sur TweetEval - Emotion, nos modèles sont meilleurs que le BiLSTM (66.0), FastText (65.2), SVM (64.7), et ayant des résultats équivalents à RoBERTa [99, 187] (72.0), ce dernier étant un modèle « lourd » basé sur le Transformer, nécessitant environ 2 secondes pour l'inférence. Nous pouvons faire les mêmes observations avec FPB, avec le LSTM (74), LSTM avec ELMO (77), et

d'autres modèles disponibles dans [188]. Ces résultats montrent le grand intérêt de nos modèles. Ils constituent donc une alternative très intéressante tout en ayant des temps d'entraînement et d'inférence faibles.

5.1.4 Conclusion

Dans cette partie, nous avons introduit le Neural NB ainsi que deux de ces extensions : Neural Pooled MC et Neural Pooled MC2. Ces modèles neuronaux sont un premier exemple de modèle neuronal inspiré de modèle graphique probabiliste génératif et permettent de mettre en relation ces deux mondes. De plus, ils atteignent des performances très intéressantes pour l'analyse de sentiments, notamment au vu des résultats des autres modèles populaires, pouvant égaler des modèles nécessitant une puissance de calcul bien plus importante.

5.2 Hidden Neural Markov Chains et comparaison avec les RNN

Après l'étude de modèles neuronaux basés sur le *Naive Bayes*, nous allons à présent étudier, dans le même ordre d'idée, ceux basés sur la HMC et deux de ses dérivées : la HMC2 et la HMC-CN, définis en section 4.2 et 4.4.2, respectivement. Tout au long de cette section, nous supposons $\Omega_Y = \mathbb{R}^d, d \in \mathbb{N}^*$

5.2.1 Hidden Neural Markov Chain

Nous considérons le classificateur calculé de manière discriminante de la HMC défini à l'aide de l'algorithme EFB décrit en section 3.2.1. Nous supposons que les trois fonctions définissant ce classificateur, π, a, et L, introduits en section 3.1 et 3.2, sont strictement positives. Soit NN^{HMC} un réseau de neurones ayant comme *input* un vecteur de taille $N + d$ et pour *output* un vecteur de taille N positif, défini pour estimer les paramètres de la HMC de la manière suivante :

$$NN^{HMC}\left(\left[\mathrm{OH}^N(j), y_{t+1}\right]\right)_i = \frac{L_{y_{t+1}}(i)}{\pi(i)} a_j(i)$$

avec la fonction $\mathrm{OH}^N(j)$ retournant un vecteur de taille N composé de 0 excepté un unique 1 à la position j.

Ainsi, l'algorithme EFB devient :

— pour les fonctions *entropic forward* α^E :

$$\alpha^E_{t+1}(i) = \sum_{j=1}^{N} \alpha^E_t(j) NN^{HMC}([\mathrm{OH}^N(j), y_{t+1}])_i;$$

— pour les fonctions *entropic backward* β^E :

$$\beta^E_t(i) = \sum_{j=1}^{N} \beta^E_{t+1}(j) NN^{HMC}([\mathrm{OH}^N(i), y_{t+1}])_j;$$

en supposant que, pour tout $i \in \{1, ..., N\}, \alpha^E_0(i) = \beta^E_T(i) = 1_N$.

L'algorithme EFB de la HMC étant paramétré uniquement avec une fonction neuronale, nous appelons ce modèle ainsi estimé le *Hidden Neural Markov Chain* (HNMC).

5.2.2 HNMC2 et HNMC-CN

Nous considérons le classificateur du MPM de la HMC2 défini en section 4.2, et nous supposons toutes les probabilités le définissant strictement positives. De manière similaire à précédemment, nous introduisons un réseau de neurones NN^{HMC2} ayant un *input* de taille $N + N + d$ et un *output* de taille N strictement positif, tel qu'il estime les probabilités suivantes de la HMC2 :

$$NN^{HMC2}\left([\mathrm{OH}^N(i), \mathrm{OH}^N(j), y_{t+1}]\right)_k = P(X_{t+1} = \lambda_k | X_t = \lambda_i, X_{t+1} = \lambda_j) \frac{P(X_{t+1} = \lambda_k | Y_{t+1} = y_{t+1})}{P(X_{t+1} = \lambda_k)}$$

Ainsi, l'algorithme EFB pour la HMC2 devient :

— Pour les fonctions $\alpha^{E,2}$:

$$\alpha^{E,2}_{t+1}(j,k) = \sum_i \alpha^{E,2}_t(i,j) NN^{HMC2}\left([\mathrm{OH}^N(i), \mathrm{OH}^N(j), y_{t+1}]\right)_k;$$

— Pour les fonctions $\beta^{E,2}$:

$$\beta^{E,2}_t(i,j) = \sum_k \beta^{E,2}_{t+1}(j,k) NN^{HMC2}\left([\mathrm{OH}^N(i), \mathrm{OH}^N(j), y_{t+1}]\right)_k;$$

en supposant que pour tout $i, j \in \{1, ..., N\}, \alpha^{E,2}_1(i,j) = \beta^{E,2}_T(i,j) = 1_N$. Nous appelons ce modèle ainsi estimé le *Hidden Neural Markov Chain* d'ordre 2 (HNMC2).

Nous considérons également le classificateur du MPM de la HMC-CN défini en section 4.4.2, toujours en supposant les probabilités impliquées strictement positives. Nous introduisons deux réseaux de neurones NN_1^{HMC-CN} et NN_2^{HMC-CN} avec un *input* de taille $N+d$ et un *output* de taille N strictement positif estimant les probabilités suivantes de la HMC-CN :

$$NN_1^{HMC-CN}(\left[\mathrm{OH}^N(j), y_{t+1}\right])_i = \frac{P(X_t = \lambda_i | X_{t+1} = \lambda_j, Y_{t+1} = y_{t+1})}{P(X_t = \lambda_i)}$$

$$NN_2^{HMC-CN}(\left[\mathrm{OH}^N(i), y_t, y_{t+1}\right])_j = P(X_{t+1} = \lambda_j | X_t = \lambda_i, Y_t = y_t) \frac{P(X_{t+1} = \lambda_j | Y_{t=1} = y_{t+1})}{P(X_{t+1} = \lambda_j | X_t = \lambda_i)}$$

Ainsi, l'algorithme EFB pour la HMC-CN devient :

— pour les fonctions $\alpha^{E,CN}$:

$$\alpha_{t+1}^{E,2}(j) = \sum_i \alpha_t^{E,CN}(i) NN_1^{HMC-CN}(\left[\mathrm{OH}^N(j), y_{t+1}\right])_i \times NN_2^{HMC-CN}(\left[\mathrm{OH}^N(i), y_t, y_{t+1}\right])_j;$$

— pour les fonctions $\beta^{E,CN}$:

$$\beta_t^{E,CN}(i) = \sum_j \beta_{t+1}^{E,CN}(j) NN_1^{HMC-CN}(\left[\mathrm{OH}^N(j), y_{t+1}\right])_i \times NN_2^{HMC-CN}(\left[\mathrm{OH}^N(i), y_t, y_{t+1}\right])_j;$$

en supposant que pour tout $i \in \{1, ..., N\}, \alpha_0^{E,CN}(i) = \beta_T^{E,CN}(i) = 1_N$. Nous appelons ce modèle ainsi estimé le *Hidden Neural Markov Chain with Complexified Noise* (HNMC-CN).

5.2.3 Comparaison de la HNMC, HNMC2, et HNMC-CN avec RNN et BiRNN

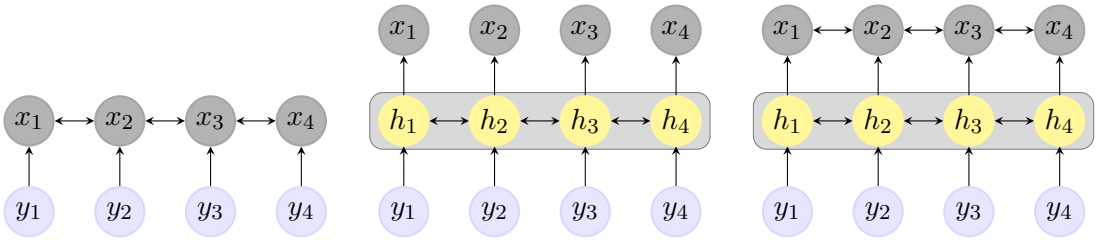
Nous allons comparer empiriquement la HNMC, la HNMC2, et la HNMC-CN avec le RNN et le BiRNN pour différentes tâches de segmentation de textes :

— POS Tagging avec le *dataset* UD English et l'*embedding* ExtVec, évalué par l'*accuracy* ;

— *Chunking* avec le *dataset* CoNLL 2000 et l'*embedding* GloVe, évalué par le score F_1 ;

— NER avec le *dataset* CoNLL 2003 et l'*embedding* FastText, évalué par le score F_1.

Pour avoir une comparaison des plus exhaustive, nous utilisons différentes architectures neuronales.



(a) Architecture 1 (b) Architecture 2 (c) Architecture 3

FIGURE 5.1 – Les graphes computationnels des différentes architectures utilisées avec la HNMC.

Architecture 1			
	POS ExtVec UD	Chunking GloVe 00	NER FastText 03
RNN	$88.40\% \pm 0.02$	86.68 ± 0.08	81.91 ± 0.14
BiRNN	$91.38\% \pm 0.04$	90.76 ± 0.55	82.62 ± 0.56
HNMC	$90.98\% \pm 0.03$	87.77 ± 0.13	83.41 ± 0.10
HNMC2	$91.33\% \pm 0.04$	88.18 ± 0.04	83.49 ± 0.06
HNMC-CN	$\mathbf{92.62}\% \pm 0.04$	$\mathbf{92.02} \pm 0.03$	$\mathbf{87.49} \pm 0.19$

TABLE 5.5 – Résultats des différents modèles pour le POS Tagging, *Chunking*, et NER, avec l'architecture 1 - uniquement le modèle.

Les différentes architectures neuronales

L'un des avantages de l'utilisation de modèles neuronaux est leur capacité à pouvoir « s'empiler » les uns aux autres. Pour une comparaison empirique des plus complètes, pour chacun des modèles, nous allons étudier les trois architectures suivantes :

— Architecture 1 : seulement le modèle ;

— Architecture 2 : le modèle suivi d'un FNN ;

— Architecture 3 : deux modèles empilés.

L'exemple des graphe computationnels de ces architectures pour la HNMC est donnée en figure 5.1.

Expériences et résultats

Nous appliquons nos différents modèles neuronaux pour les trois tâches citées. Les paramètres des réseaux de neurones sont estimés par descente de gradient avec la méthode d'optimisation Adam et un *mini-batch* de taille 32, en utilisant les librairies Flair et PyTorch. Le taux d'apprentissage vaut 0.005 pour l'architecture

Architecture 2			
	POS ExtVec UD	Chunking GloVe 00	NER FastText 03
RNN	89.84% ± 0.04	93.85 ± 0.06	84.53 ± 0.21
BiRNN	93.07% ± 0.05	95.02 ± 0.11	87.52 ± 0.13
HNMC	92.77% ± 0.06	95.43 ± 0.09	88.22 ± 0.13
HNMC2	93.01% ± 0.04	**95.59** ± 0.13	88.47 ± 0.05
HNMC-CN	**93.29**% ± 0.05	95.36 ± 0.07	**89.40** ± 0.03

TABLE 5.6 – Résultats des différents modèles pour le POS Tagging, *Chunking*, et NER, avec l'architecture 2 - le modèle suivi d'un FNN.

Architecture 3			
	POS ExtVec UD	Chunking GloVe 00	NER FastText 03
RNN	89.20% ± 0.09	93.13 ± 0.14	85.10 ± 0.12
BiRNN	92.80% ± 0.21	94.91 ± 0.09	88.68 ± 0.31
HNMC	92.73% ± 0.12	95.53 ± 0.13	88.02 ± 0.19
HNMC2	92.97% ± 0.08	**95.59** ± 0.06	88.66 ± 0.33
HNMC-CN	**93.36**% ± 0.03	95.40 ± 0.14	**89.37** ± 0.12

TABLE 5.7 – Résultats des différents modèles pour le POS Tagging, *Chunking*, et NER, avec l'architecture 3 - deux modèles empilés.

1, et 0.05 pour la première couche et 0.005 pour la deuxième pour les autres architectures. Chaque réseau de neurones utilisé n'a pas de couche cachée, les fonctions d'activation liées au RNN et au BiRNN sont des tangentes hyperboliques (1.10) comme cela est fait usuellement, et celles liées aux modèles HNMC, HNMC2, et HNMC-CN sont $1 + ELU$ (1.11), car devant être strictement positives.

Les résultats obtenus sont en Tables 5.5, 5.6, et 5.7 pour les trois architectures [1]. D'une part, nous pouvons observer que la HNMC a de meilleures performances que le RNN pour chaque tâche et chaque architecture, alors que les deux modèles ont le même nombre de paramètres. De plus, en terme computationnel, la HNMC est légèrement plus rapide que le RNN. En effet, étant donné une séquence, à chaque étape temporelle le RNN effectue une opération neuronale (une multiplication matricielle et une fonction d'activation). De son côté, la HNMC effectue une

1. À la vue de ces scores, les améliorations proposées peuvent sembler limitées. Il faut mettre en lumière que le coût supplémentaire de ces opérations est quasiment nul, et que ces améliorations sont significatives étant donné ce qui se fait dans ce domaine. Cela peut être illustré avec le *Question-Answering*, où chaque nouveau modèle a un gain pouvant sembler modeste sur le précédent : https://rajpurkar.github.io/SQuAD-explorer/. Cela est fréquent concernant le NLP.

	Taille 100	Taille 250	Taille 500
BERT	$0.43s \pm 0.01$	$1.08s \pm 0.02$	$2.34s \pm 0.01$
ALBERT	$0.45s \pm 0.01$	$1.10s \pm 0.01$	$2.37s \pm 0.03$
DistilBERT	$0.22s \pm 0.00$	$0.54s \pm 0.00$	$1.15s \pm 0.02$
RoBERTa	$0.41s \pm 0.01$	$1.06s \pm 0.04$	$2.23s \pm 0.03$

TABLE 5.8 – Temps d'inférence pour une séquence de différentes tailles des modèles basés sur le Transformers sur un CPU de 16Go.

	Taille 100	Taille 250	Taille 500
BiLSTM-CRF(300, 64)	$0.03s$	$0.07s$	$0.15s$
BiLSTM-CRF(300, 256)	$0.04s$	$0.09s$	$0.18s$
BiLSTM-CRF(4096, 64)	$0.04s$	$0.08s$	$0.18s$
BiLSTM-CRF(4096, 256)	$0.05s$	$0.13s$	$0.26s$

TABLE 5.9 – Temps d'inférence pour une séquence de différentes tailles des modèles BiLSTM-CRF sur un CPU de 16Go selon la taille d'entrée de celle de la couche intermédiaire.

opération neuronale en début, puis une multiplication matricielle à chaque étape, justifiant sa rapidité sur le RNN. Nous observons également que pour chaque architecture et chaque tâche, l'un des modèles basés sur la chaîne de Markov cachée atteint les meilleures performances, surpassant notamment le BiRNN, remarquant notamment que la HNMC-CN est plus rapide que ce dernier pour l'inférence.

5.2.4 Conclusion

Dans cette partie, nous avons introduit de nouveaux modèles neuronaux basés sur la HMC : les *Hidden Neural Markov Chains*. Ces modèles bénéficient d'une grande flexibilité de modélisation des dépendances des différentes variables, basés sur les modèles probabilistes. Ils atteignent des performances dépassant ceux du RNN et du BiRNN, qui sont la brique de bases des modèles neuronaux pour gérer des séquences, tout en étant plus rapide. Ces résultats sont prometteurs dans l'objectif de mettre au point des modèles neuronaux plus performants sans que cela n'impacte la vitesse d'exécution de manière significative.

5.3 BiLSTM associé avec la PMC et la TMC

De nos jours, deux catégories de modèles se révèlent particulièrement utilisées pour la segmentation de textes, et plus précisément les tâches de POS Tagging,

Chunking, et NER. D'une part, il y a les modèles basés sur le Transformer [65] tels que BERT [57], ALBERT [82], DistilBERT [189], RoBERTa [187], et encore bien d'autres [190, 191, 192, 81]. Ces modèles atteignent l'état-de-l'art pour ces tâches avec des performances parfois supérieures à celles d'un humain. Néanmoins, une critique sévère peut leur être faite : ils sont très lourds et nécessitent une très grande puissance de calcul, que cela soit pour l'inférence ou l'entraînement. De plus, ils ne s'adaptent pas à la taille de la phrase considérée, ce qui peut se révéler problématique pour des phrases trop longues. D'autres critiques sont également émises concernant l'impact écologique de leur utilisation [94]. Par conséquent, bien que très performants, leur utilisation se révèle limitée pour des applications industrielles, principalement dû au coût requis par leur architecture. Le tableau 5.8 prodigue les temps d'inférence sur un ordinateur muni d'un CPU de 16Go sur des séquences de taille 100, 250, et 500 pour les modèles cités plus haut avec la librairie de référence fournie par HuggingFace [193]. Nous prenons la moyenne sur 50 inférences, et nous réalisons cette expérience à cinq reprises, nous présentons la moyenne et l'intervalle de confiance gaussien à 95%.

D'autre part, les modèles neuronaux dits « séquentiels », c'est-à-dire étant conçus pour pouvoir traiter des données de taille variable, sont également très appréciés pour la segmentation de textes, et notamment le modèle BiLSTM-CRF [145]. Ce modèle, que nous décrirons en détail par la suite, consiste en la concaténation d'un BiLSTM et d'un *linear-chain* CRF (que nous dénommons CRF dans cette section), l'*output* du BiLSTM devenant l'*input* du CRF. Bien que légèrement moins performant, il est très apprécié pour ses temps d'inférence bien plus faibles, sa flexibilité en termes de puissance, et son adaptation à la taille de la séquence. Nous illustrons ses temps d'inférence en tableau 5.9, avec différentes tailles de la couche d'entrée du BiLSTM et de la couche intermédiaire entre le BiLSTM et le CRF. Les intervalles de confiances, toujours inférieur à ± 0.01, ne sont pas précisés pour plus de clarté. Ces temps d'inférence sont bien moins élevés que pour les modèles basés sur le Transformer cités plus haut.

Nous allons donc nous concentrer sur les modèles BiLSTM-CRF. En section 3.4, nous avons montré l'équivalence entre les modèles HMC et *linear-chain* CRF. Donc le BiLSTM-CRF peut être vu comme un BiLSTM-HMC. Or, nous avons étudié en section 4.4 des extensions de la HMC dont les travaux sur le sujet ont mis en évidence une amélioration importante de ses performances : la PMC et la TMC. Ainsi, dans cette partie, nous allons développer les modèles BiLSTM-PMC et BiLSTM-TMC. Ces modèles devraient atteindre des performances supérieures au BiLSTM-CRF sans pour autant en augmenter les temps d'inférence de manière significative.

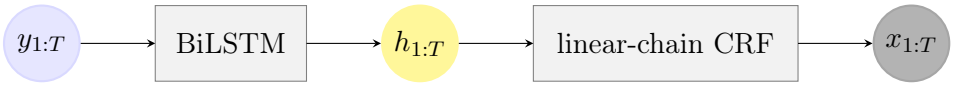


FIGURE 5.2 – Architecture du BiLSTM-CRF.

Remarque 5.1. Nous aurions pu utiliser les extensions du CRF introduits en section 4.5, mais étant donné l'équivalence avec les modèles PMC et TMC, les mêmes résultats sont attendus.

5.3.1 Le modèle BiLSTM-CRF

Le modèle BiLSTM-CRF est un modèle combinant le modèle BiLSTM, avec un modèle probabiliste, le CRF. Il considère comme *input* la séquence de données $y_{1:T} = (y_1, ..., y_T), y_t \in \mathbb{R}^d$. Cette séquence est l'*input* d'un BiLSTM, qui produit la séquence $h_{1:T} = (h_1, ..., h_T), h_t \in \mathbb{R}^{d'}$. Cette nouvelle séquence est considérée comme la réalisation de variables aléatoires observées, devenant l'*input* d'un CRF, et produisant ainsi la séquence $x_{1:T} = (x_1, ..., x_T), x_t \in \mathbb{R}^{d''}$, considérée comme l'estimation d'une réalisation des variables aléatoires cachées du CRF. Cette architecture est représentée en figure 5.2.

Le BiLSTM

Nous allons à présent décrire cette procédure en détail. Pour ce faire, nous décrivons la fonction LSTM :

$$LSTM : \mathbb{R}^d \longrightarrow \mathbb{R}^{\frac{d'}{2}}, y_{1:T} \longrightarrow h_{1:T} = LSTM(y_{1:T})$$

avec :

$$\begin{aligned}
i_t &= \sigma(w_{yi} y_t + w_{hi} h_{t-1} + w_{ci} c_{t-1} + b_i) \\
f_t &= \sigma(w_{yf} y_t + w_{hf} h_{t-1} + w_{cf} c_{t-1} + b_f) \\
c_t &= f_t c_{t-1} + i_t \tanh(w_{yc} y_t + w_{hc} h_{t-1} + b_c) \\
o_t &= \sigma(w_{yo} y_t + w_{ho} h_{t-1} + w_{co} c_t + b_o) \\
h_t &= o_t \tanh(c_t)
\end{aligned}$$

i, f, o et c sont appelés les *input gate, forget gate, output gate* et *cell vectors*, ils ont tous la même taille que le vecteur h_t. Les matrices de poids ont la signification que leur nom suggère. Il suppose des vecteurs initiaux h_0 et c_0, souvent nulles, ou tirés aléatoirement.

Le LSTM est un modèle neuronal très populaire, ayant l'avantage de bénéficier d'une mémoire longue [2]. Néanmoins, ce modèle présente l'inconvénient d'être unidirectionnel, tout comme le MEMM. Cela signifie que le calcul de h_t n'utilise que $y_{1:t}$, et ignore $y_{t+1:T}$. Pour résoudre cela, le BiLSTM a été proposé et connaît un franc succès. Ce dernier consiste à concaténer les *outputs* de deux fonctions LSTM : l'une appliquée à $y_{1:T}$, et l'autre appliquée à $(y_T, y_{T-1}, ..., y_2, y_1)$. Ainsi, le calcul de h_t avec un BiLSTM utilise bien toute la séquence $y_{1:T}$. Nous notons cette fonction ainsi :

$$BiLSTM : \mathbb{R}^d \longrightarrow \mathbb{R}^{d'}, y_{1:T} \longrightarrow h_{1:T} = BiLSTM(y_{1:T})$$

Le CRF

Le CRF est un modèle probabiliste décrit en section 3.4.1. Pour rappel, il considère les variables cachées $X_{1:T}$ avec X_t à valeurs dans Λ_X, et les variables observées $H_{1:T}, H_t$ prenant ses valeurs dans Θ_H. Il est défini par la loi a posteriori suivante, pour un CRF stationnaire :

$$\begin{aligned} p(x_{1:T}|h_{1:T}) &= \frac{U(x_1, h_1) \prod_{t=1}^{T-1} V(x_t, x_{t+1}) U(x_{t+1}, h_{t+1})}{\sum_{x'_{1:T}} U(x'_1, h_1) \prod_{t=1}^{T-1} V(x'_t, x'_{t+1}) U(x'_{t+1}, h_{t+1})} \\ &= \frac{1}{Z(h_{1:T})} U(x_1, h_1) \prod_{t=1}^{T-1} V(x_t, x_{t+1}) U(x_{t+1}, h_{t+1}) \end{aligned}$$

avec U et V des fonctions potentielles mesurant l'influence entre les différentes variables, et $Z(h_{1:T})$ la fonction de normalisation.

Tout comme la HMC, il est possible de définir les classificateurs du MAP et du MPM pour le CRF. Ainsi, nous allons présenter l'algorithme *Forward-Backward* et l'algorithme de Viterbi pour ce modèle [24]. Comme attendu, ces algorithmes sont très similaires à ceux de la HMC, décrits au chapitre 3.

L'algorithme *Forward-Backward* du CRF permet de définir son classificateur de Bayes du MPM. Il a donc pour objectif de calculer pour tout $t \in \{1, ..., T\}, x_t \in \Lambda_X$, avec les réalisations $H_{1:T} = h_{1:T}, p(x_t|h_{1:T})$. La procédure est la suivante :

$$p(x_t|h_{1:T}) = \frac{\alpha_t^{CRF}(x_t)\beta_t^{CRF}(x_t)}{\sum_{x'_t} \alpha_t^{CRF}(x'_t)\beta_t^{CRF}(x'_t)},$$

2. Le tutoriel suivant décrit en détail le fonctionnement du LSTM : https://colah.github.io/posts/2015-08-Understanding-LSTMs/

avec :

$$\begin{aligned}
\alpha_1^{CRF}(x_1) &= U(x_1, h_1), \\
\alpha_{t+1}^{CRF}(x_{t+1}) &= U(x_{t+1}, h_{t+1}) \sum_{x_t} V(x_t, x_{t+1}) \alpha_t^{CRF}(x_t), \\
\beta_T^{CRF}(x_T) &= 1, \\
\beta_t^{CRF}(x_t) &= \sum_{x_{t+1}} V(x_t, x_{t+1}) U(x_{t+1}, h_{t+1}) \beta_{t+1}^{CRF}(x_{t+1}).
\end{aligned}$$

L'algorithme de Viterbi permet d'approcher l'estimateur de Bayes du MAP, que nous allons décrire pour le CRF. Il est de nouveau très similaire à celui de la HMC. Nous calculons :

$$\begin{aligned}
V_1^{CRF}(x_1) &= T(x_1, y_1) \\
V_{t+1}^{CRF}(x_{t+1}) &= \max_{x_t} \left(U(x_t, x_{t+1}) T(x_{t+1}, h_{t+1}) V_t^{CRF}(x_t) \right)
\end{aligned}$$

Puis il suffit de retrouver, comme usuellement, le chemin de Viterbi :

$$\begin{aligned}
x_T &= \arg \max_{\lambda_i \in \Lambda_X} \left(V_T^{CRF}(\lambda_i) \right) \\
x_{t-1} &= \mathrm{Ptr}^{CRF}(x_t, t)
\end{aligned}$$

avec $\mathrm{Ptr}^{CRF}(\lambda_i, t)$ la fonction retournant la valeur de λ utilisée pour calculer $V_t^{CRF}(\lambda_i)$ si $t > 1$ et λ_i si $t = 1$.

Remarque 5.2. Nous notons bien les très grandes similarités entre ces algorithmes et ceux de la HMC, notamment les versions dites *« Entropic »*.

Remarque 5.3. Dans le cadre du BiLSTM-CRF, un grand nombre de travaux combine un BiLSTM avec l'algorithme de Viterbi du CRF. Cependant, l'utilisation de l'algorithme *Forward-Backward* est plus appropriée. En effet, les méthodes d'évaluations, le score F_1 et l'*accuracy*, sont basées sur une évaluation terme-à-terme, plus adaptée au critère du MPM. Par conséquent, l'algorithme *Forward-Backward* sera donc utilisé pour ce modèle lors de nos expériences.

5.3.2 BiLSTM-PMC et BiLSTM-TMC

De manière similaire au fait que la concaténation du BiLSTM et du CRF permet de définir le BiLSTM-CRF, nous définissons les modèles BiLSTM-PMC et BiLSTM-TMC. Ainsi, le BiLSTM-PMC est la concaténation d'un BiLSTM et

	Taille 100	Taille 250	Taille 500
BiLSTM-PMC(300, 64)	$0.03s$	$0.07s$	$0.14s$
BiLSTM-PMC(300, 256)	$0.04s$	$0.08s$	$0.16s$
BiLSTM-PMC(4096, 64)	$0.04s$	$0.08s$	$0.17s$
BiLSTM-PMC(4096, 256)	$0.05s$	$0.12s$	$0.24s$
BiLSTM-TMC(300, 64)	$0.03s$	$0.07s$	$0.14s$
BiLSTM-TMC(300, 256)	$0.04s$	$0.08s$	$0.16s$
BiLSTM-TMC(4096, 64)	$0.04s$	$0.08s$	$0.17s$
BiLSTM-TMC(4096, 256)	$0.05s$	$0.12s$	$0.25s$

TABLE 5.10 – Temps d'inférence de modèles BiLSTM-PMC et BiLSTM-TMC sur un CPU de 16Go selon la taille d'entrée de celle de la couche intermédiaire.

d'une PMC, avec l'*output* du BiLSTM considéré comme la réalisation de la variable en *input* de la PMC. Il en est de même pour le BiLSTM-TMC. Nous utilisons les algorithmes *Entropic Forward-Backward* de nos modèles markoviens définis en section 4.4, où chaque paramètre de la PMC et de la TMC sont modélisés par une Régression Logistique, comme c'est le cas pour le CRF. Concernant la dimension de la variable latente pour la TMC, nous la prenons égale à la dimension de Λ_X.

5.3.3 Expériences

Nous souhaitons mesurer empiriquement l'apport des modèles BiLSTM-PMC et BiLSTM-TMC sur le BiLSTM-CRF. Nous allons donc appliquer ces modèles pour les tâches suivantes :

— POS Tagging avec le *dataset* UD English ;

— NER avec le *dataset* CoNLL 2003.

Nous allons utiliser deux méthodes d'*embedding* : FastText et Flair. Pour la première, la dimension de la couche intermédiaire vaut 64, et 256 pour la seconde.

Tous les paramètres de ces modèles sont estimés par descente de gradient en utilisant la méthode d'optimisation AdamW [194] avec un taux d'apprentissage de 0.01 ainsi que la méthode d'annihilation de gradient Cosine [70]. Ils sont entraînés sur 50 époques, et nous gardons le modèle ayant le meilleur score sur les données de validation. Les codes sont réalisés en Python avec les librairies PyTorch et Flair. Les temps d'inférence des modèles sont indiqués en Tableau 5.10, avec une méthodologie similaire à celle présentée plus haut. Les résultats du POS Tagging et du NER sont en Tables 5.11 et 5.12 pour les méthodes d'*embedding* FastText et Flair, respectivement. Nous affichons l'*accuracy* pour le POS Tagging et le score F_1 pour la NER.

FastText Embedding		
	POS	NER
BiLSTM-CRF	94.24 ± 0.08	89.56 ± 0.08
BiLSTM-PMC	94.38 ± 0.06	89.95 ± 0.10
BiLSTM-TMC	$\mathbf{94.48} \pm 0.05$	$\mathbf{90.17} \pm 0.17$

TABLE 5.11 – Performances des différents modèles pour le POS Tagging et la NER avec la méthode d'*embedding* FastText.

Flair Embedding		
	POS	NER
BiLSTM-CRF	96.14 ± 0.04	91.29 ± 0.13
BiLSTM-PMC	96.37 ± 0.05	91.42 ± 0.12
BiLSTM-TMC	$\mathbf{96.44} \pm 0.05$	$\mathbf{91.54} \pm 0.07$

TABLE 5.12 – Performances des différents modèles pour le POS Tagging et la NER avec la méthode d'*embedding* Flair.

En premier lieu, il est très intéressant et pertinent de noter que nos modèles ont des temps d'inférence égaux à ceux du BiLSTM-CRF. En effet, en analysant les algorithmes de restauration du MPM du CRF, de la PMC, et de la TMC, nous pouvons remarquer que les mêmes opérations sont effectuées, excepté une somme supplémentaire en fin d'algorithme pour la TMC. La seule différence étant la mémoire allouée pour effectuer ces opérations qui sera plus importante pour la PMC, et encore plus pour la TMC. Cela résulte ainsi en des temps d'inférence équivalents pour nos trois algorithmes. Cette observation est d'importance, car l'utilisation d'un modèle plus complexe n'a pas de coût outre celui d'utiliser plus de mémoire.

Comme attendu, nous observons que pour chaque tâche et chaque méthode d'*embedding*, le BiLSTM-TMC est meilleur que le BiLSTM-PMC, lui-même meilleur que le BiLSTM-CRF. Nous avons ainsi mis au point des modèles meilleurs que le très populaire BiLSTM-CRF sans occasionner de coût supplémentaire important et sans influence sur les temps d'inférence.

5.3.4 Conclusion

À partir de nos résultats théoriques, notamment l'équivalence entre le CRF et la HMC et la capacité à tout modèle génératif de définir un classificateur calculé de manière discriminante, nous avons mis au point deux nouveaux modèles permettant de surpasser les performances du BiLSTM-CRF : le BiLSTM-PMC et le

BiLSTM-TMC. Ces résultats illustrent de nouveau l'intérêt des modèles génératifs paramétrés avec des réseaux de neurones.

5.4 Conclusion sur les modèles génératifs paramétrés par des réseaux de neurones

Ce chapitre a permis de mettre en application un grand nombre des résultats théoriques établis au cours de cette thèse. Il a permis d'introduire les modèles suivants : le *Neural Naive Bayes*, le Neural Pooled MC, le Neural Pooled MC2, la HNMC, la HNMC2, la HNMC-CN, le BiLSTM-PMC, et le BiLSTM-TMC. Tous ont montré un grand intérêt dans les tâches sur lesquelles ils ont été appliqués, que ce soit pour l'analyse de sentiments ou la segmentation de textes.

Cela permet d'illustrer l'intérêt applicatif de nos résultats théoriques : peu importe la complexité de la modélisation probabiliste, nous sommes capables de calculer le classificateur induit de manière discriminante, et de le paramétrer avec des réseaux de neurones. Cette méthodologie permet de faire le pont les modèles probabilistes et les réseaux de neurones, avec des premiers résultats d'ores et déjà prometteurs.

En termes de perspectives, après avoir exploré l'analyse de sentiments de la segmentation de textes, il serait intéressant d'appliquer les mécanismes d'attention [80] sur les modèles markoviens neuronaux pour en évaluer empiriquement l'intérêt. Une autre perspective serait également d'appliquer ces modèles pour développer de nouvelles méthodes de *word embedding* permettant d'avoir un embedding performant et entraînable rapidement sur des *datasets* de taille humaine.

Conclusion et perspectives

Plusieurs résultats ont été montré et présenté au cours de cette thèse. Cela commença par un apport avec de nouvelles définitions de modèles probabilistes et classificateurs génératifs et discriminants. Puis nous avons montré qu'il est possible de définir le classificateurs calculé de manière discriminante du *Naive Bayes*. Ce résultat a permis de mettre à jour la comparaison entre le *Naive Bayes* et la Régression Logistique : longtemps perçu comme supérieur dû au problème des *features*, nous avons montré que le classificateur de la Régression Logistique n'est qu'un cas particulier de celui du *Naive Bayes*. De plus, nous avons montré l'équivalence entre le *Naive Bayes* et le CRF inspiré par son graphe.

Ce travail n'a pas été uniquement effectué avec le *Naive Bayes*, mais également pour un autre modèle probabiliste très connu : la chaîne de Markov cachée. Nous avons montré que les classificateurs du MAP et du MPM de ce modèle peuvent être calculés de manière discriminante. De plus, comme pour le *Naive Bayes*, nous avons l'équivalence avec son CRF défini par le même graphe, connu sous le nom du *linear-chain* CRF. La littérature considérait pourtant ce dernier comme supérieur à la HMC, en raison du problème des *features*.

Enfin, pour conclure sur cet axe, après avoir montré qu'il était possible de définir le classificateur calculé de manière discriminante pour le *Naive Bayes* et la HMC, nous avons montré ce résultat pour tout modèle probabiliste génératif, ainsi que son expression. Ce résultat est d'importance et permet une grande flexibilité de modélisation qui n'est plus contrainte par la complexité des dépendances entre les différentes variables comme cela pouvait être le cas auparavant. Ainsi, nous avons pu définir ce classificateur pour un grand nombre de modèles, étendant notamment le *Naive Bayes* et la HMC.

D'un point de vue pratique, ces résultats ont permis de mettre en évidence le potentiel des modèles *Naive Bayes* et HMC pour le NLP, alors qu'ils étaient considéré comme inadapté. Nous sommes allés plus loin en définissant les classificateurs issus de ces modèles avec des fonctions neuronales. Ainsi, nous avons développé les

modèles *Neural Naive Bayes*, *Hidden Neural Markov Chain*, mais également leurs extensions. Cet ensemble de modèles obtient des résultats des plus prometteurs. En effet, nous atteignons l'état-de-l'art pour le *Chunking* avec la HMC, nous avons mis au point des modèles pour l'analyse de sentiments 1000 fois plus rapides tout en étant proches des meilleurs résultats possibles avec le Neural Pooled MC2, et nous avons modélisés et implémentés les modèles BiLSTM-PMC et BiLSTM-TMC, meilleurs en tout point que le BiLSTM-CRF, pourtant considéré comme la référence des modèles neuronaux séquentiels. Ces nombreuses applications permettent de mettre en évidence les applications pratiques de nos résultats théoriques.

Un nombre important de perspectives s'offre à nous pour aller plus loin et continuer nos recherches. D'un point de vue théoriques, nous avons montré tout au long de ce manuscrit des équivalences entre des modèles génératifs et des CRF, ces derniers étant définis par une loi a posteriori. Il serait intéressant d'explorer si cette équivalence existe pour tout modèle génératif et tout CRF, même très complexes.

De plus, nous avons introduit la *Hidden Neural Markov Chain*, et montré sa supériorité sur le RNN en terme de performances et de rapidité. Or, les RNN ont été étendus au LSTM, qui est le modèle séquentiel neuronal le plus utilisé. Ainsi, concevoir l'équivalent LSTM mais à partir de la *Hidden Neural Markov Chain* peut également être une recherche prometteuse.

Enfin, appliquer cette nouvelle méthode consistant à paramétrer des modèles probabilistes génératifs avec des réseaux de neurones pourrait avoir des performances intéressantes pour d'autre tâches, qu'elles soient liées au Traitement des Langues Naturelles ou non, comme par exemple l'élaboration de nouvelles méthodes d'*embedding* ou la classification d'images. Ces applications sont également des perspectives intéressantes à explorer.

Annexe A

Application du classificateur calculé de manière générative de la PMC pour la segmentation de textes

Dans cette annexe, nous présentons une application de la PMC avec son classificateur calculé de manière générative pour la segmentation de textes. Comme décrit en section 4.4, ce modèle considère les variables aléatoires cachées $X_{1:T}$, avec X_t prenant ses valeurs dans Λ_X et des variables aléatoires observées $Y_{1:T}, Y_t$ prenant ses valeurs dans Ω_Y telles que le couple (X_t, Y_t) soit une chaîne de Markov :

$$p(x_{1:T}, y_{1:T}) = p(x_1, y_1) \prod_{t=1}^{T-1} p(x_{t+1}, y_{t+1} | x_t, y_t).$$

Son classificateur calculé de manière générative souffre donc du problème des *features* décrit en section 1.4.3 lors de l'estimation des paramètres. Néanmoins, nous allons voir comment l'appliquer afin d'obtenir des résultats pertinents. Cette méthode bénéficie notamment d'une très grande rapidité pour l'entraînement et l'inférence. Elle reste néanmoins très limitée en comparaison aux meilleurs classificateurs calculés de manière discriminante pouvant gérer des séquences de taille variable.

A.1 Algorithme Forward-Backward de la PMC

Dans la mesure où nous étudions l'application de la PMC pour la segmentation de textes, évaluée terme-à-terme, nous considérons le critère du MPM. Ainsi, nous utilisons l'algorithme *Forward-Backward* de la PMC. Il est défini de la manière

suivante, pour tout $t \in \{1, ..., T\}, x_t \in \Lambda_X$:

$$p(x_t|y_{1:T}) = \frac{\alpha_t^{PMC}(x_t)\beta_t^{PMC}(x_t)}{\sum_{x'_t} \alpha_t^{PMC}(x'_t)\beta_t^{PMC}(x'_t)},$$

avec $\alpha_t^{PMC}(x_t) = p(y_{1:t}, x_t)$ calculé de manière récursive :

— pour $t = 1$:

$$\alpha_1^{PMC}(x_1) = p(x_1, y_1);$$

— pour $1 \leq t < T$:

$$\alpha_{t+1}^{PMC}(x_{t+1}) = \sum_{x_t} \alpha_t^{PMC}(x_t)p(x_{t+1}, y_{t+1}|x_t, y_t).$$

De même pour $\beta_t^{PMC}(x_t) = p(y_{t+1:T}|x_t, y_t)$:

— pour $t = T$:

$$\beta_T^{PMC}(x_T) = 1;$$

— pour $1 \leq t < T$:

$$\beta_t^{PMC}(x_t) = \sum_{x_{t+1}} \beta_{t+1}^{PMC}(x_{t+1})p(x_{t+1}, y_{t+1}|x_t, y_t).$$

Les fonctions α^{PMC} et β^{PMC} sont normalisées à chaque étape afin d'éviter les erreurs d'*underflows*.

A.2 Estimation des paramètres

L'algorithme *Forward-Backward* nécessite la connaissance des probabilités $p(x_1, y_1), p(x_{t+1}|x_t, y_t)$ et $p(y_{t+1}|x_t, y_t, x_{t+1})$. Nous estimons ces différentes lois par le compte des différents *patterns*, comme décrit en section 1.4.1. Nous les estimons ainsi :

$$p(x_1, y_1) = \frac{N(x_1, y_1)}{L}$$

$$p(x_{t+1}|x_t, y_t) = \frac{N(x_t, y_t, x_{t+1})}{\sum_{x'_{t+1}} N(x_t, y_t, x'_{t+1})},$$

$$p(y_{t+1}|x_t, y_t, x_{t+1}) = \frac{N(x_t, y_t, x_{t+1}, y_{t+1})}{\sum_{y'_{t+1}} N(x_t, y_t, x_{t+1}, y_{t+1})},$$

avec $N(.)$ la fonction de comptage, comptant le nombre d'occurrence du *pattern* donné dans l'ensemble d'entraînement et L le nombre de phrases du jeu d'entraînement.

A.3 Rétrogradation de la PMC vers la HMC pour le NLP

Nous souhaitons appliquer la PMC avec l'algorithme *Forward-Backward* pour des tâches de NLP. Nous n'utilisons pas de méthodes d'*embedding* en raison du problème des *features*, et considérons comme observation le mot lui-même, donc Ω_Y est l'ensemble de tous les mots écrits possibles.

Cet ensemble Ω_Y étant de cardinal très grand, si ce n'est infini, nous sommes limités par la taille du *dataset* d'entraînement. Ainsi, pour un grand nombre de réalisations, les paramètres estimés vont être égal à 0. Cela est particulièrement dommageable pour la PMC. Ainsi, si y_t n'a jamais été observé dans le jeu d'entraînement, $p(x_{t+1}|x_t, y_t)$ sera égal à 0. $p(y_{t+1}|x_t, y_t, x_{t+1})$ sera également égal à 0 si y_t suivi y_{t+1} n'est pas dans le jeu d'entraînement, donc ce paramètre est très couramment égal à 0.

Cela limite ainsi les performances de la PMC avec son classificateur calculé de manière générative du MPM. Ce problème survient bien moins souvent avec la HMC avec son algorithme *Forward-Backward*. En effet, la PMC requiert que la suite de deux mots soit connue via $p(x_{t+1}, y_{t+1}|x_t, y_t)$, tandis que la HMC uniquement le mot au temps t, via $p(y_t|x_t)$.

Par conséquent, pour alléger ce problème relatif à la PMC, une solution consiste à approcher ses paramètres par ceux de la HMC en cas de mots inconnus :

(i) $p(x_1, y_1)$ est estimé par $p(x_1)$ si y_1 est inconnu ;

(ii) $p(x_{t+1}|x_t, y_t)$ est estimé par $p(x_{t+1}|x_t)$ si y_t est inconnu ;

(iii) $p(y_{t+1}|x_t, y_t, x_{t+1})$ est estimé par $p(y_{t+1}|x_{t+1})$ si y_t suivi de y_{t+1} est inconnu.

De plus, dans le cas (iii), il est possible que y_{t+1} soit inconnu. Une solution consisterait à le remplacer par une constante. Nous pouvons également utiliser la méthode d'approximation suivante :

$$p(y_{t+1}|x_{t+1}) \approx p(u(y_{t+1}), h(y_{t+1}), f(y_{t+1}), d(y_{t+1}), s_3(y_{t+1})|x_{t+1})$$

avec les fonctions :

— $u(y) = 1$ si la première lettre de y est capitale, 0 sinon ;

— $h(y) = 1$ si y possède un tiret, 0 sinon ;

— $f(y) = 1$ si y est le premier mot de la phrase, 0 sinon ;

	HMC-FB	CRF	PMC-FB
CoNLL 2000	2.96%	2.47%	**2.32**%
CoNLL 2000 KW	1.94%	1.72%	**1.27**%
CoNLL 2000 UW	16.54%	**12.47**%	16.41%
CoNLL 2003	5.29%	**4.28**%	4.71%
CoNLL 2003 KW	4.03%	**3.21**%	3.40%
CoNLL 2003 UW	15.30%	**12.79**%	15.16%
UD English	8.13%	**6.95**%	7.16%
UD English KW	6.07%	5.46%	**5.00**%
UD English UW	30.73%	**23.25**%	30.78%

TABLE A.1 – Taux d'erreur de la HMC-FB, du *linear-chain* CRF, et de la PMC-FB pour le POS Tagging.

— $d(y) = 1$ si y possède un chiffre, 0 sinon ;

— $s_m(y)$ correspond au suffixe de taille m de y.

Si $s_3(y_{t+1})$ est également inconnu, nous utilisons $s_2(y_{t+1})$, puis sinon $s_1(y_{t+1})$, ou encore $s_0(y_{t+1})$. Il est important de préciser que le choix de ces *features* est très délicat, car nous sommes limités par le problème de *features* de l'estimation de la loi des observations. Ainsi, prendre plus de suffixes, ou même les préfixes nous désavantagerait très rapidement car nous aurions plus de chances d'avoir un ensemble de *features* inconnu.

Nous avons ainsi présenté une méthode simple et originale d'utilisation de la PMC avec l'algorithme *Forward-Backward* consistant à le « rétrograder » en HMC lorsque le besoin en est. L'ensemble des probabilités présentées dans cette section est également estimé par la fréquence des différents *patterns*.

A.4 Expériences

Nous appliquons la PMC avec l'algorithme *Forward-Backward* (PMC-FB) pour le POS Tagging sur les *datasets* CoNLL 2000, CoNLL 2003, et UD English, pour la NER sur CoNLL 2003, et pour le *Chunking* sur CoNLL 2000 et CoNLL 2003. Nous affichons, à titre de comparaison et pour montrer l'utilité de cette méthode, les résultats de la HMC (HMC-FB). Nous comparons également les performances de la PMC avec le *linear-chain* CRF avec les mêmes *features* que décrites en section précédente, en y ajoutant les suffixes et les préfixes de taille 1 à 4. Tous les codes sont réalisés en Python, et le *linear-chain* CRF avec la librairie CRF Suite [195].

	HMC-FB	CRF	PMC-FB
CoNLL 2003	78.44	79.16	**79.52**
CoNLL 2003 KW	87.07	87.41	**88.47**
CoNLL 2003 UW	56.67	**58.67**	56.87

TABLE A.2 – Scores F_1 de la HMC-FB, du *linear-chain* CRF, et de la PMC-FB pour la NER.

	HMC-FB	CRF	PMC-FB
CoNLL 2000	92.72	93.05	**94.49**
CoNLL 2000 KW	93.18	93.35	**95.09**
CoNLL 2000 UW	87.45	**89.63**	87.58
CoNLL 2003	94.30	94.79	**95.61**
CoNLL 2003 KW	94.65	94.92	**96.17**
CoNLL 2003 UW	91.95	**93.88**	91.85

TABLE A.3 – Scores F_1 de la HMC-FB, du *linear-chain* CRF, et de la PMC-FB pour le *Chunking*.

Les résultats sont présentés en Table A.1 pour le POS Tagging, en Table A.2 pour la NER, et en Table A.3 pour le *Chunking*. Nous présentons également les performances sur les mots connus (KW) et inconnus (UW). Les temps d'entraînement sont présentés en Table A.4.

La HMC-FB est, comme attendu, le modèle le plus rapide à entraîner, mais ses performances sont inférieures aux deux autres. La PMC-FB améliore la HMC-FB de 20% environ du point de vue de l'erreur, ce qui montre l'utilité de notre méthode. De plus, nous voyons que la PMC-FB et le *linear-chain* CRF ont des performances équivalentes, mais la PMC-FB est beaucoup plus rapide à entraîner, montrant l'intérêt de celle-ci. L'analyse des scores sur les mots inconnus nous montre bien, comme attendu, que cette méthode est limitée pour les mots inconnus. En effet, même la méthode d'approximation que nous proposons ne permet pas de tirer pleinement partie des *features*, comme tel est le cas pour un classificateur calculé de manière discriminante comme illustré avec le *linear-chain* CRF.

	HMC-FB	CRF	PMC-FB
CoNLL 2000 POS	1.3s	140s	5s
CoNLL 2000 Chunk	1.3s	140s	5s
CoNLL 2003 POS	1.5S	180s	5.5s
CoNLL 2003 NER	1.5s	260s	5.5s
CoNLL 2003 Chunk	1.5s	160s	5.5s
UD English POS	1.7s	185s	6.3s

TABLE A.4 – Temps d'entraînement des différents modèles sur un CPU de 8Go.

A.5 Conclusion

Nous avons présenté une application de classificateur du MPM calculé de manière générative de la PMC pour la segmentation de textes. Cela a permis de mettre en évidence les limites de cette méthode tout en y montrant des intérêts. En effet, la PMC-FB atteint des performances équivalentes au *linear-chain* CRF tout en étant bien plus rapide à entraîner.

Bibliographie

[1] Elie Azeraf, Emmanuel Monfrini, and Wojciech Pieczynski. Using the Naive Bayes as a Discriminative Model. In *Proceedings of the 13th International Conference on Machine Learning and Computing*, page 106–110, 2021.

[2] Elie Azeraf, Emmanuel Monfrini, Emmanuel Vignon, and Wojciech Pieczynski. Hidden Markov Chains, Entropic Forward-Backward, and Part-of-Speech Tagging. *arXiv preprint arXiv :2005.10629*, 2020.

[3] Elie Azeraf, Emmanuel Monfrini, and Wojciech Pieczynski. On Equivalence between Linear-chain Conditional Random Fields and Hidden Markov Chains. In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence - Volume 3 : ICAART*, pages 725–728. INSTICC, SciTePress, 2022.

[4] Elie Azeraf, Emmanuel Monfrini, and Wojciech Pieczynski. Deriving Discriminative Classifiers from Generative Models. *arXiv preprint arXiv :2201.00844*, 2022.

[5] Elie Azeraf, Emmanuel Monfrini, and Wojciech Pieczynski. Improving Usual Naive Bayes Classifier Performances with Neural Naive Bayes based Models. In *Proceedings of the 11th International Conference on Pattern Recognition Applications and Methods - ICPRAM*, pages 315–322. INSTICC, SciTePress, 2022.

[6] Elie Azeraf, Emmanuel Monfrini, Emmanuel Vignon, and Wojciech Pieczynski. Introducing the Hidden Neural Markov Chain Framework. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2 : ICAART*, pages 1013–1020. INSTICC, SciTePress, 2021.

[7] Elie Azeraf, Emmanuel Monfrini, Emmanuel Vignon, and Wojciech Pieczynski. Highly Fast Text Segmentation With Pairwise Markov Chains. In *6th IEEE Congress on Information Science and Technology (CiSt)*, pages 361–366, 2020.

[8] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models : Principles and Techniques*. MIT press, 2009.

[9] Michael I Jordan. Graphical Models. *Statistical Science*, 19(1) :140–155, 2004.

[10] Martin J Wainwright and Michael I Jordan. *Graphical Models, Exponential Families, and Variational Inference*. Now Publishers Inc, 2008.

[11] David D Lewis. Naive (Bayes) at Forty : The Independence Assumption in Information Retrieval. In *European Conference on Machine Learning*, pages 4–15, 1998.

[12] Irina Rish. An Empirical Study of the Naive Bayes Classifier. In *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, pages 41–46, 2001.

[13] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied Logistic Regression*, volume 398. John Wiley & Sons, 2013.

[14] Scott Menard. *Applied Logistic Regression Analysis*, volume 106. Sage, 2002.

[15] David G Kleinbaum, K Dietz, M Gail, Mitchel Klein, and Mitchell Klein. *Logistic Regression*. Springer, 2002.

[16] Chao-Ying Joanne Peng, Kuk Lida Lee, and Gary M Ingersoll. An Introduction to Logistic Regression Analysis and Reporting. *The Journal of Educational Research*, 96(1) :3–14, 2002.

[17] Ruslan Leont'evich Stratonovich. Conditional Markov Processes. In *Non-Linear Transformations of Stochastic Processes*, pages 427–453. Elsevier, 1965.

[18] Leonard E Baum and Ted Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6) :1554–1563, 1966.

[19] Lawrence Rabiner and B Juang. An Introduction to Hidden Markov Models. *IEEE ASSP Magazine*, 3(1) :4–16, 1986.

[20] Yariv Ephraim and Neri Merhav. Hidden Markov Processes. *IEEE Transactions on Information Theory*, 48(6) :1518–1569, 2002.

[21] Olivier Cappé, Eric Moulines, and Tobias Rydén. *Inference in Hidden Markov Models*. Springer Science & Business Media, 2006.

[22] Andrew McCallum, Dayne Freitag, and Fernando CN Pereira. Maximum Entropy Markov Models for Information Extraction and Segmentation. In *Proceedings of the International Conference on Machine Learning*, volume 17, pages 591–598, 2000.

[23] John D Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional Random Fields : Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the International Conference on Machine Learning*, 2001.

[24] Charles Sutton and Andrew McCallum. An Introduction to Conditional Random Fields for Relational Learning. *Introduction to Statistical Relational Learning*, 2 :93–128, 2006.

[25] Ruslan Salakhutdinov and Geoffrey Hinton. Deep Boltzmann Machines. In *Artificial Intelligence and Statistics*, pages 448–455. PMLR, 2009.

[26] Geoffrey E Hinton. Boltzmann Machine. *Scholarpedia*, 2(5) :1668, 2007.

[27] Luc Devroye, László Györfi, and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition*, volume 31. Springer Science & Business Media, 2013.

[28] Peter E Hart, David G Stork, and Richard O Duda. *Pattern Classification*. John Wiley & Sons, 2006.

[29] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition*. Elsevier, 2013.

[30] Julia A Lasserre, Christopher M Bishop, and Thomas P Minka. Principled Hybrids of Generative and Discriminative Models. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 87–94, 2006.

[31] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[32] Tom Minka. Discriminative Models, not Discriminative Training. Technical report, Technical Report MSR-TR-2005-144, Microsoft Research, 2005.

[33] Wolfgang Roth, Robert Peharz, Sebastian Tschiatschek, and Franz Pernkopf. Hybrid Generative-Discriminative Training of Gaussian Mixture Models. *Pattern Recognition Letters*, 112 :131–137, 2018.

[34] Ying Nian Wu, Ruiqi Gao, Tian Han, and Song-Chun Zhu. A Tale of Three Probabilistic Families : Discriminative, Descriptive, and Generative Models. *Quarterly of Applied Mathematics*, 77(2) :423–465, 2019.

[35] Ilkay Ulusoy and Christopher M Bishop. Generative versus Discriminative Methods for Object Recognition. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 258–265, 2005.

[36] Koffi Eddy Ihou, Nizar Bouguila, and Wassim Bouachir. Efficient Integration of Generative Topic Models into Discriminative Classifiers using Robust Probabilistic Kernels. *Pattern Analysis and Applications*, pages 1–25, 2020.

[37] Oksana Yakhnenko, Adrian Silvescu, and Vasant Honavar. Discriminatively Trained Markov Model for Sequence Classification. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp. IEEE, 2005.

[38] Guillaume Bouchard and Bill Triggs. The Tradeoff Between Generative and Discriminative Classifiers. In *16th IASC International Symposium on Computational Statistics (COMPSTAT'04)*, pages 721–728, 2004.

[39] Dan Jurafsky and James H. Martin. *Speech and Language Processing : an Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition (3rd Edition)*. Pearson Prentice Hall, 2021.

[40] Andrew Y Ng and Michael I Jordan. On Discriminative vs. Generative Classifiers : A Comparison of Logistic Regression and Naive Bayes. In *Advances in Neural Information Processing Systems*, pages 841–848, 2002.

[41] Ikechukwu I Ayogu, Adebayo O Adetunmbi, Bolanle A Ojokoh, and Samuel A Oluwadare. A Comparative Study of Hidden Markov Model and Conditional Random Fields on a Yoruba Part-Of-Speech Tagging Task. In *2017 International Conference on Computing Networking and Informatics (ICCNI)*, pages 1–6, 2017.

[42] Wahab Khan, Ali Daud, Jamal Abdul Nasir, Tehmina Amjad, Sachi Arafat, Naif Aljohani, and Fahd S Alotaibi. Urdu Part-Of-Speech Tagging using Conditional Random Fields. *Language Resources and Evaluation*, 53(3) :331–362, 2019.

[43] Dewi Yanti Liliana and Chan Basaruddin. A Review on Conditional Random Fields as a Sequential Classifier in Machine Learning. In *2017 International Conference on Electrical Engineering and Computer Science (ICECOS)*, pages 143–148, 2017.

[44] Muhammad Hameed Siddiqi, Madallah Alruwaili, Amjad Ali, Saad Alanazi, and Furkh Zeshan. Human Activity Recognition using Gaussian Mixture Hidden Conditional Random Fields. *Computational intelligence and neuroscience*, 2019, 2019.

[45] Shengli Song, Nan Zhang, and Haitao Huang. Named Entity Recognition based on Conditional Random Fields. *Cluster Computing*, 22(3) :5195–5206, 2019.

[46] Romansha Chopra, Nivedita Singh, Yang Zhenning, and N Ch SN Iyengar. Sequence Labeling using Conditional Random Fields. *International Journal of u-and e-Service, Science and Technology*, 10(9) :101–108, 2017.

[47] Mengqi Fang, Hariprasad Kodamana, Biao Huang, and Nima Sammaknejad. A Novel Approach to Process Operating Mode Diagnosis using Conditional Random Fields in the Presence of Missing Data. *Computers & Chemical Engineering*, 111 :149–163, 2018.

[48] Jerry Chun-Wei Lin, Yinan Shao, Ji Zhang, and Unil Yun. Enhanced Sequence Labeling based on Latent Variable Conditional Random Fields. *Neurocomputing*, 403 :431–440, 2020.

[49] Joan Santoso, Esther Irawati Setiawan, Eko Mulyanto Yuniarno, Mochamad Hariadi, and Mauridhi Hery Purnomo. Hybrid Conditional Random Fields and K-Means for Named Entity Recognition on Indonesian News Documents. *International Journal of Intelligent Engineering and Systems*, 13(3) :233–245, 2020.

[50] Xuli Sun, Shiliang Sun, Minzhi Yin, and Hao Yang. Hybrid Neural Conditional Random Fields for Multi-View Sequence Labeling. *Knowledge-Based Systems*, 189 :105151, 2020.

[51] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural Networks for Machine Learning - Lecture 6a Overview of Mini-Batch Gradient Descent. *Cited on*, 14(8) :2, 2012.

[52] Matthew D Zeiler. Adadelta : an Adaptive Learning Rate Method. *arXiv preprint arXiv :1212.5701*, 2012.

[53] Diederik P Kingma and Jimmy Ba. Adam : A Method for Stochastic Optimization. *arXiv preprint arXiv :1412.6980*, 2014.

[54] Sebastian Ruder. An Overview of Gradient Descent Optimization Algorithms. *arXiv preprint arXiv :1609.04747*, 2016.

[55] Vladimir Vapnik. Statistical Learning Theory - Vol. 3, 1998.

[56] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5 :135–146, 2017.

[57] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT : Pre-Training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv :1810.04805*, 2018.

[58] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual String Embeddings for Sequence Labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649, 2018.

[59] Alan Akbik, Tanja Bergmann, and Roland Vollgraf. Pooled Contextualized Embeddings for Named Entity Recognition. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, page 724–728, 2019.

[60] Roger Fjortoft, Yves Delignon, Wojciech Pieczynski, Marc Sigelle, and Florence Tupin. Unsupervised Classification of Radar Images using Hidden Markov Chains and Hidden Markov Random Fields. *IEEE Transactions on geoscience and remote sensing*, 41(3) :675–686, 2003.

[61] E Monfrini, J Lecomte, F Desbouvries, and W Pieczynski. Image and Signal Restoration using Pairwise Markov Trees. In *IEEE Workshop on Statistical Signal Processing, 2003*, pages 174–177. IEEE, 2003.

[62] Clément Fernandes, Thomas Monti, Emmanuel Monfrini, and Wojciech Pieczynski. Fast Image Segmentation with Contextual Scan and Markov Chains. In *2021 29th European Signal Processing Conference (EUSIPCO)*, pages 626–630, 2021.

[63] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A Theoretical Framework for Back-Propagation. In *Proceedings of the 1988 Connectionist Models Summer School*, volume 1, pages 21–28, 1988.

[64] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural computation*, 1(4) :541–551, 1989.

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[66] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[67] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521(7553) :436–444, 2015.

[68] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5) :359–366, 1989.

[69] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML*, 2010.

[70] Ilya Loshchilov and Frank Hutter. SGDR : Stochastic Gradient Descent with Warm Restarts. *arXiv preprint arXiv :1608.03983*, 2016.

[71] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging Weights Leads to Wider Optima and Better Generalization. *arXiv preprint arXiv :1803.05407*, 2018.

[72] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning Internal Representations by Error Propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[73] Michael I. Jordan. *Attractor Dynamics and Parallelism in a Connectionist Sequential Machine*, page 112–127. IEEE Press, 1990.

[74] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An Empirical Exploration of Recurrent Network Architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.

[75] Zachary C Lipton, John Berkowitz, and Charles Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv preprint arXiv :1506.00019*, 2015.

[76] Mike Schuster and Kuldip K Paliwal. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing*, 45(11) :2673–2681, 1997.

[77] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8) :1735–1780, 1997.

[78] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint arXiv :1412.3555*, 2014.

[79] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation : Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, 2014.

[80] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv :1409.0473*, 2014.

[81] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. XLNet : Generalized Autoregressive Pretraining for Language Understanding. In *Advances in Neural Information Processing Systems*, pages 5753–5763, 2019.

[82] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT : A Lite Bert for Self-Supervised Learning of Language Representations. *arXiv preprint arXiv :1909.11942*, 2019.

[83] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander M. Rush, and Thomas Wolf. Datasets : A community library for natural language processing, 2021.

[84] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3(Feb) :1137–1155, 2003.

[85] Ronan Collobert and Jason Weston. A Unified architecture for Natural Language Processing : Deep Neural Networks with Multitask Learning. In

Proceedings of the 25th International Conference on Machine Learning, pages 160–167, 2008.

[86] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv :1301.3781*, 2013.

[87] Felipe Almeida and Geraldo Xexéo. Word Embeddings : A Survey. *arXiv preprint arXiv :1901.09069*, 2019.

[88] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe : Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014.

[89] Alexandros Komninos and Suresh Manandhar. Dependency Based Embeddings for Sentence Classification Tasks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, pages 1490–1500, 2016.

[90] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. LUKE : Deep Contextualized Entity Representations with Entity-Aware Self-Attention. *arXiv preprint arXiv :2010.01057*, 2020.

[91] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. Smart : Robust and Efficient Fine-Tuning for Pre-Trained Natural Language Models Through Principled Regularized Optimization. *arXiv preprint arXiv :1911.03437*, 2019.

[92] Xiaodong Liu, Kevin Duh, Liyuan Liu, and Jianfeng Gao. Very Deep Transformers for Neural Machine Translation. *arXiv preprint arXiv :2008.07772*, 2020.

[93] Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. Automated Concatenation of Embeddings for Structured Prediction. *arXiv preprint arXiv :2010.05006*, 2020.

[94] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv :1906.02243*, 2019.

[95] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level Convolutional Networks for Text Classification. In *NIPS*, 2015.

[96] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics : Human Language Technologies*, pages 142–150, 2011.

[97] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[98] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE : A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP : Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[99] Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa-Anke, and Leonardo Neves. TweetEval : Unified Benchmark and Comparative Evaluation for Tweet Classification. In *Proceedings of Findings of EMNLP*, 2020.

[100] P. Malo, A. Sinha, P. Korhonen, J. Wallenius, and P. Takala. Good Debt or Bad Debt : Detecting Semantic Orientations in conomic Texts. *Journal of the Association for Information Science and Technology*, 65, 2014.

[101] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text Classification Algorithms : A Survey. *Information*, 10(4) :150, 2019.

[102] Edward Loper and Steven Bird. NLTK : the Natural Language ToolKit. *arXiv preprint cs/0205028*, 2002.

[103] Joakim Nivre, Marie-Catherine De Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Tsarfaty Reut, and Zeman Daniel. Universal Dependencies v1 : A Multilingual Treebank Collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation*, pages 1659–1666, 2016.

[104] Erik F. Tjong Kim Sang and Sabine Buchholz. Introduction to the CoNLL-2000 Shared Task Chunking. In *4th Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop*, 2000.

[105] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 Shared Task : Language-Independent Named Entity Recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003.

[106] Mitchell P Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English : The Penn Treebank. In *COMPUTATIONAL LINGUISTICS*, 1993.

[107] J-D Kim, Tomoko Ohta, Yuka Tateisi, and Jun'ichi Tsujii. GENIA Corpus — A Semantically Annotated Corpus for Bio-Textmining. *Bioinformatics*, 19 :i180–i182, 2003.

[108] Hamid Jalalzai, Pierre Colombo, Chloé Clavel, Éric Gaussier, Giovanna Varni, Emmanuel Vignon, and Anne Sabourin. Heavy-Tailed Representations, Text Polarity Classification & Data Augmentation. *Advances in Neural Information Processing Systems*, 33 :4295–4307, 2020.

[109] Wojciech Witon, Pierre Colombo, Ashutosh Modi, and Mubbasir Kapadia. Disney at IEST 2018 : Predicting Emotions using an Ensemble. In *Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 248–253, 2018.

[110] Alexandre Garcia, Pierre Colombo, Slim Essid, Florence d'Alché Buc, and Chloé Clavel. From the Token to the Review : A Hierarchical Multimodal Approach to Opinion Mining. *arXiv preprint arXiv :1908.11216*, 2019.

[111] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD : 100,000+ Questions for Machine Comprehension of Text. *arXiv preprint arXiv :1606.05250*, 2016.

[112] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know What you Don't Know : Unanswerable Questions for SQuAD. *arXiv preprint arXiv :1806.03822*, 2018.

[113] Pierre Colombo, Wojciech Witon, Ashutosh Modi, James Kennedy, and Mubbasir Kapadia. Affect-Driven Dialog Generation. *arXiv preprint arXiv :1904.02793*, 2019.

[114] Pierre Colombo, Emile Chapuis, Matteo Manica, Emmanuel Vignon, Giovanna Varni, and Chloe Clavel. Guiding Attention in Sequence-to-Sequence Models for Dialogue Act Prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7594–7601, 2020.

[115] Emile Chapuis, Pierre Colombo, Matteo Manica, Matthieu Labeau, and Chloe Clavel. Hierarchical Pre-Training for Sequence Labelling in Spoken Dialog. *arXiv preprint arXiv :2009.11152*, 2020.

[116] Tanvi Dinkar, Pierre Colombo, Matthieu Labeau, and Chloé Clavel. The Importance of Fillers for Text Representations of Speech Transcripts. *arXiv preprint arXiv :2009.11340*, 2020.

[117] Sebastian Raschka. Naive Bayes and Text Classification I - Introduction and Theory. *arXiv preprint arXiv :1410.5329*, 2014.

[118] Kevin P Murphy. Naive Bayes Classifiers. *University of British Columbia*, 18(60), 2006.

[119] Jing-Hao Xue and D Michael Titterington. Comment on "On Discriminative vs. Generative Classifiers : A Comparison of Logistic Regression and Naive Bayes". *Neural processing letters*, 28(3) :169–187, 2008.

[120] Paraskevas Tsangaratos and Ioanna Ilia. Comparison of a Logistic Regression and Naive Bayes Classifier in Landslide Susceptibility Assessments : The Influence of Models Complexity and Training Dataset Size. *Catena*, 145 :164–179, 2016.

[121] Anjuman Prabhat and Vikas Khullar. Sentiment Classification on Big Data using Naive Bayes and Logistic Regression. In *2017 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–5. IEEE, 2017.

[122] Teemu Roos, Hannes Wettig, Peter Grünwald, Petri Myllymäki, and Henry Tirri. On Discriminative Bayesian Network Classifiers and Logistic Regression. *Machine Learning*, 59(3) :267–296, 2005.

[123] Wei Chen, Xusheng Yan, Zhou Zhao, Haoyuan Hong, Dieu Tien Bui, and Biswajeet Pradhan. Spatial Prediction of Landslide Susceptibility using Data Mining-Based Kernel Logistic Regression, Naive Bayes and RBFNetwork Models for the Long County area (China). *Bulletin of Engineering Geology and the Environment*, 78(1) :247–266, 2019.

[124] L Mary Gladence, M Karthi, and V Maria Anu. A Statistical Comparison of Logistic Regression and Different Bayes Classification Methods for Machine Learning. *ARPN Journal of Engineering and Applied Sciences*, 10(14) :5947–5953, 2015.

[125] Longjun Dong, Johan Wesseloo, Yves Potvin, and Xibing Li. Discrimination of Mine Seismic Events and Blasts using the Fisher Classifier, Naive Bayesian Classifier and Logistic Regression. *Rock Mechanics and Rock Engineering*, 49(1) :183–211, 2016.

[126] D Seka, BS Bonny, AN Yoboué, SR Sié, and BA Adopo-Gourène. Identification of Maize (Zea Mays L.) Progeny Genotypes based on two Probabilistic Approaches : Logistic Regression and Naive Bayes. *Artificial Intelligence in Agriculture*, 1 :9–13, 2019.

[127] Poonam Somani, Shreyas Talele, and Suraj Sawant. Stock Market Prediction Using Hidden Markov Model. In *2014 IEEE 7th Joint International Information Technology and Artificial Intelligence Conference*, pages 89–92. IEEE, 2014.

[128] Md Rafiul Hassan and Baikunth Nath. Stock Market Forecasting Using Hidden Markov Model : a New Approach. In *5th International Conference on Intelligent Systems Design and Applications (ISDA'05)*, pages 192–196. IEEE, 2005.

[129] Md Rafiul Hassan. A Combination of Hidden Markov Model and Fuzzy Model for Stock Market Forecasting. *Neurocomputing*, 72(16-18) :3439–3446, 2009.

[130] Jia Li, Amir Najmi, and Robert M Gray. Image Classification by a Two-Dimensional Hidden Markov Model. *IEEE Transactions on Signal Processing*, 48(2) :517–533, 2000.

[131] Guoliang Fan and Xiang-Gen Xia. Image Denoising using a Local Contextual Hidden Markov Model in the Wavelet Domain. *IEEE Signal Processing Letters*, 8(5) :125–128, 2001.

[132] Gunasekaran Manogaran, V Vijayakumar, R Varatharajan, Priyan Malarvizhi Kumar, Revathi Sundarasekar, and Ching-Hsien Hsu. Machine Learning Based Big Data Processing Framework for Cancer Diagnosis using Hidden Markov Model and GM Clustering. *Wireless Personal Communications*, 102(3) :2099–2116, 2018.

[133] Thorsten Brants. TnT – A Statistical Part-of-Speech Tagger. In *Sixth Applied Natural Language Processing Conference*, pages 224–231, Seattle, Washington, USA, 2000. Association for Computational Linguistics.

[134] Shai Fine, Yoram Singer, and Naftali Tishby. The Hierarchical Hidden Markov Model : Analysis and Applications. *Machine learning*, 32(1) :41–62, 1998.

[135] Sudha Morwal, Nusrat Jahan, and Deepti Chopra. Named Entity Recognition Using Hidden Markov Model. *International Journal on Natural Language Computing*, 1(4) :15–23, 2012.

[136] Asif Ekbal, S Mondal, and Sivaji Bandyopadhyay. POS Tagging using HMM and Rule-based Chunking. *The Proceedings of SPSAL*, 8(1) :25–28, 2007.

[137] Lawrence R Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2) :257–286, 1989.

[138] Xian Tang. Hybrid Hidden Markov Model and Artificial Neural Network for Automatic Speech Recognition. In *2009 Pacific-Asia Conference on Circuits, Communications and Systems*, pages 682–685. IEEE, 2009.

[139] AP Varga and RK Moore. Hidden Markov Model Decomposition of Speech and Noise. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 845–848. IEEE, 1990.

[140] Andrew Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, 13(2) :260–269, 1967.

[141] Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. Painless unsupervised learning with features. In *Human Language Technologies : The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, page 582–590, USA, 2010. Association for Computational Linguistics.

[142] Pierre Lanchantin, Jérôme Lapuyade-Lahorgue, and Wojciech Pieczynski. Unsupervised Segmentation of Randomly Switching Data Hidden with non-Gaussian Correlated Noise. *Signal Processing*, 91(2) :163–175, 2011.

[143] Georg Heigold, Hermann Ney, Patrick Lehnen, Tobias Gass, and Ralf Schluter. Equivalence of Generative and Log-Linear Models. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(5) :1138–1148, 2010.

[144] Luoxin Chen, Xinyue Liu, Weitong Ruan, and Jianhua Lu. Enhance Robustness of Sequence Labelling with Masked Adversarial Training. In *Findings of the Association for Computational Linguistics : EMNLP 2020*, pages 297–302, 2020.

[145] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF Models for Sequence Tagging. *arXiv preprint arXiv :1508.01991*, 2015.

[146] Wojciech Pieczynski. Pairwise Markov Chains. *IEEE Transactions. Pattern Anal. Mach. Intell.*, 25(5) :634–639, 2003.

[147] Jüri Lember and Joonas Sova. Regenerativity of Viterbi Process for Pairwise Markov Models. *Journal of Theoretical Probability*, 34(1) :1–33, 2021.

[148] Jüri Lember and Joonas Sova. Exponential Forgetting of Smoothing Distributions for Pairwise Markov Models. *Electronic Journal of Probability*, 26 :1–30, 2021.

[149] Maria V Kulikova, Julia V Tsyganova, and G Yu Kulikov. SVD-Based State and Parameter Estimation Approach for Generalized Kalman Filtering with Application to GARCH-in-Mean Estimation. *Journal of Computational and Applied Mathematics*, 387 :112487, 2021.

[150] Jüri Lember, Heinrich Matzinger, Joonas Sova, and Fabio Zucca. Lower Bounds for Moments of Global Scores of Pairwise Markov Chains. *Stochastic Processes and their Applications*, 128(5) :1678–1710, 2018.

[151] Mohamed El Yazid Boudaren and Wojciech Pieczynski. Dempster–Shafer Fusion of Evidential Pairwise Markov Chains. *IEEE Transactions on Fuzzy Systems*, 24(6) :1598–1610, 2016.

[152] Jüri Lember and Joonas Sova. Existence of Infinite Viterbi Path for Pairwise Markov Models. *Stochastic Processes and their Applications*, 130(3) :1388–1425, 2020.

[153] Guanghua Zhang, Jian Lan, Le Zhang, Fengshou He, and Shaomin Li. Filtering in Pairwise Markov Model With Student's t Non-Stationary Noise With Application to Target Tracking. *IEEE Transactions on Signal Processing*, 69 :1627–1641, 2021.

[154] Frederic Lehmann and Wojciech Pieczynski. State Estimation in Pairwise Markov Models with Improved Robustness using Unbiased FIR Filtering. *Signal Processing*, 172 :107568, 2020.

[155] Ronald Mahler. The Pairwise-Markov Bernoulli Filter. *IEEE Access*, 8 :168229–168245, 2020.

[156] LIU Jiangyi, WANG Chunping, and WANG Wei. SMC-CBMeMBer Filter Based on Pairwise Markov Chains. *Systems Engineering and Electronics*, 41(8) :1686–1691, 2019.

[157] Ivan Gorynin, Elie Azeraf, Wiem Sabbagh, Emmanuel Monfrini, and Wojciech Pieczynski. Optimal Filtering in Hidden and Pairwise Gaussian Markov Systems. *International Journal of Mathematical and Computational Methods*, 1, 2016.

[158] Yohan Petetin and François Desbouvries. Bayesian Multi-Object Filtering for Pairwise Markov Chains. *IEEE Transactions on Signal Processing*, 61(18) :4481–4490, 2013.

[159] Jiangyi Liu, Chunping Wang, Wei Wang, and Zheng Li. Particle Probability Hypothesis Density Filter based on Pairwise Markov Chains. *Algorithms*, 12(2) :31, 2019.

[160] Maria V Kulikova, Julia V Tsyganova, and Gennady Yu Kulikov. UD-based Pairwise and MIMO Kalman-like Filtering for Estimation of Econometric Model Structures. *IEEE Transactions on Automatic Control*, 65(10) :4472–4479, 2020.

[161] Guang-Hua Zhang, Chong-Zhao Han, Feng Lian, and Ling-Hao Zeng. Cardinality Balanced Multi-Target Multi-Bernoulli Filter for Pairwise Markov Model. *ACTA Automat. Sin*, 43(12) :2100–2108, 2017.

[162] Stéphane Derrode and Wojciech Pieczynski. Signal and Image Segmentation Using Pairwise Markov Chains. *IEEE Transactions on Signal Processing*, 52(9) :2477–2489, 2004.

[163] Armand Kodjo Atiampo and Georges Laussane Loum. Unsupervised Image Segmentation with Pairwise Markov Chains based on Nonparametric Estimation of Copula using Orthogonal Polynomials. *International Journal of Image and Graphics*, 16(04) :1650020, 2016.

[164] Stéphane Derrode and Wojciech Pieczynski. Unsupervised Data Classification using Pairwise Markov Chains with Automatic Copulas Selection. *Computational Statistics & Data Analysis*, 63 :81–98, 2013.

[165] Steven Le Cam, Fabien Salzenstein, and Ch Collet. Fuzzy Pairwise Markov Chain to Segment Correlated Noisy Data. *Signal processing*, 88(10) :2526–2541, 2008.

[166] Nicolas Brunel and Wojciech Pieczynski. Unsupervised Signal Restoration using Hidden Markov Chains with Copulas. *Signal processing*, 85(12) :2304–2315, 2005.

[167] Ivan Gorynin, Hugo Gangloff, Emmanuel Monfrini, and Wojciech Pieczynski. Assessing the Segmentation Performance of Pairwise and Triplet Markov Models. *Signal Processing*, 145 :183–192, 2018.

[168] Ivan Gorynin, Emmanuel Monfrini, and Wojciech Pieczynski. Pairwise Markov Models for Stock Index Forecasting. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 2041–2045, 2017.

[169] Wojciech Pieczynski. Chaınes de Markov Triplet. *Comptes Rendus Mathematique*, 335(3) :275–278, 2002.

[170] Mohamed El Yazid Boudaren, Emmanuel Monfrini, Wojciech Pieczynski, and Amar Aissani. Phasic Triplet Markov Chains. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11) :2310–2316, 2014.

[171] Fabien Salzenstein, Christophe Collet, Steven Lecam, and Mathieu Hatt. Non-Stationary Fuzzy Markov Chain. *Pattern Recognition Letters*, 28(16) :2201–2208, 2007.

[172] Wojciech Pieczynski. Modeling non Stationary Hidden semi-Markov Chains with Triplet Markov Chains and Theory of Evidence. In *IEEE/SP 13th Workshop on Statistical Signal Processing, 2005*, pages 727–732. IEEE, 2005.

[173] Haoyu Li, Stéphane Derrode, and Wojciech Pieczynski. An Adaptive and online IMU-Based Locomotion Activity Classification Method Using a Triplet Markov Model. *Neurocomputing*, 362 :94–105, 2019.

[174] Ali Ben Abbes, Mohamed Farah, Imed Riadh Farah, and Vincent Barra. A Non-Stationary NDVI Time Series Modelling using Triplet Markov Chain. *International Journal of Information and Decision Sciences*, 11(2) :163–179, 2019.

[175] Wojciech Pieczynski, Cédric Hulard, and Thomas Veit. Triplet Markov Chains in Hidden Signal Restoration. In *Image and Signal Processing for Remote Sensing VIII*, volume 4885, pages 58–68. International Society for Optics and Photonics, 2003.

[176] Stephanie Bricq, Christophe Collet, and J-P Armspach. Triplet Markov Chain for 3D MRI Brain Segmentation using a Probabilistic Atlas. In *3rd IEEE International Symposium on Biomedical Imaging : Nano to Macro, 2006.*, pages 386–389. IEEE, 2006.

[177] Boujemaa Ait-El-Fquih and François Desbouvries. Kalman Filtering in Triplet Markov Chains. *IEEE Transactions on Signal Processing*, 54(8) :2957–2963, 2006.

[178] Shou Chen and Xiangqian Jiang. Modeling Repayment Behavior of Consumer Loan in Portfolio across Business Cycle : A Triplet Markov Model Approach. *Complexity*, 2020, 2020.

[179] Muhammad Hameed Siddiqi. An Improved Gaussian Mixture Hidden Conditional Random Fields Model for Audio-Based Emotions Classification. *Egyptian Informatics Journal*, 22(1) :45–51, 2021.

[180] Ariadna Quattoni, Sybor Wang, Louis-Philippe Morency, Morency Collins, and Trevor Darrell. Hidden Conditional Random Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(10) :1848–1852, 2007.

[181] Sanjiv Kumar. Discriminative Random Fields : A Discriminative Framework for Contextual Interaction in Classification. In *Proceedings of the 9th IEEE International Conference on Computer Vision*, pages 1150–1157, 2003.

[182] Jaime F Delgado Saa and Müjdat Çetin. A Latent Discriminative Model-Based Approach for Classification of Imaginary Motor tasks from EEG Data. *Journal of Neural Engineering*, 9(2), 2012.

[183] Mohamed El Yazid Boudaren, Lin An, and Wojciech Pieczynski. Unsupervised Segmentation of SAR Images using Gaussian Mixture-Hidden Evidential Markov Fields. *IEEE Geoscience and Remote Sensing Letters*, 13(12) :1865–1869, 2016.

[184] Thomas Lavergne, Olivier Cappé, and François Yvon. Practical very large scale CRFs. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 504–513, 2010.

[185] Duyu Tang, Bing Qin, and Ting Liu. Document Modeling with Gated Recurrent Neural Network for Sentiment Classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1422–1432, September 2015.

[186] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics : Volume 2, Short Papers*, pages 427–431, Valencia, Spain, April 2017.

[187] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta : A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv :1907.11692*, 2019.

[188] Dogu Araci. FinBERT : Financial Sentiment Analysis with Pre-Trained Language Models. *arXiv preprint arXiv :1908.10063*, 2019.

[189] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a Distilled Version of BERT : Smaller, Faster, Cheaper and Lighter. *arXiv preprint arXiv :1910.01108*, 2019.

[190] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. *OpenAI blog*, 1(8) :9, 2019.

[191] Hang Le, Loïc Vial, Jibril Frej, Vincent Segonne, Maximin Coavoux, Benjamin Lecouteux, Alexandre Allauzen, Benoît Crabbé, Laurent Besacier, and Didier Schwab. FlauBERT : Unsupervised Language Model Pre-Training for French. *arXiv preprint arXiv :1912.05372*, 2019.

[192] Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric Villemonte de La Clergerie, Djamé Seddah, and Benoît Sagot. CamemBERT : a Tasty French Language Model. *arXiv preprint arXiv :1911.03894*, 2019.

[193] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Davison Joe, Shleifer Sam, von Platen Patrick, Ma Clara, Jernite Yacine, Plu Julien, Xu Canwen, Le Scao Teven, Gugger Sylvain, Drame Mariama, Lhoest Quentin, and M. Rush Alexander. Huggingface's Transformers : State-of-the-Art Natural Language Processing. *arXiv preprint arXiv :1910.03771*, 2019.

[194] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *arXiv preprint arXiv :1711.05101*, 2017.

[195] Naoaki Okazaki. CRFsuite : a Fast Implementation of Conditional Random Fields (CRFs), 2007.

ECOLE
DOCTORALE
DE MATHEMATIQUES
HADAMARD

Titre : Classification avec des Modèles Probabilistes Génératifs et des Réseaux de Neurones. Applications au Traitement des Langues Naturelles

Mots clés : Modèles Probabilistes Génératifs, Classificateurs calculés de manière discriminante, Réseaux de Neurones, Traitement des Langues Naturelles, Chaîne de Markov Cachée, *Naive Bayes*

Résumé : Un nombre important de modèles probabilistes connaissent une grande perte d'intérêt pour la classification avec apprentissage supervisé depuis un certain nombre d'années, tels que le *Naive Bayes* ou la chaîne de Markov cachée. Ces modèles, qualifiés de génératifs, sont critiqués car leur classificateur induit doit prendre en compte la loi des observations, qui peut s'avérer très complexe à apprendre quand le nombre de *features* de ces derniers est élevé. C'est notamment le cas en Traitement des Langues Naturelles, où les récents algorithmes convertissent des mots en vecteurs numériques de grande taille pour atteindre de meilleures performances.

Au cours de cette thèse, nous montrons que tout modèle génératif peut définir son classificateur sans prendre en compte la loi des observations. Cette proposition remet en question la catégorisation connue des modèles probabilistes et leurs classificateurs induits - en classes générative et discriminante - et ouvre la voie à un grand nombre d'applications possibles. Ainsi, la chaîne de Markov cachée peut être appliquée sans contraintes à la décomposition syntaxique de textes, ou encore le *Naive Bayes* à l'analyse de sentiments.

Nous allons plus loin, puisque cette proposition permet de calculer le classificateur d'un modèle probabiliste génératif avec des réseaux de neurones. Par conséquent, nous « neuralisons » les modèles cités plus haut ainsi qu'un grand nombre de leurs extensions. Les modèles ainsi obtenus permettant d'atteindre des scores pertinents pour diverses tâches de Traitement des Langues Naturelles tout en étant interprétable, nécessitant peu de données d'entraînement, et étant simple à mettre en production.

Title : Classification with Generative Probabilistic Models and Neural Networks. Applications to Natural Language Processing

Keywords : Probabilistic Generative Models, Classifier computed discriminatively, Neural Networks, Natural Language Processing, Hidden Markov Chain, Naive Bayes

Abstract : Many probabilistic models have been neglected for classification tasks with supervised learning for several years, as the Naive Bayes or the Hidden Markov Chain. These models, called generative, are criticized because the induced classifier must learn the observations' law. This problem is too complex when the number of observations' features is too large. It is especially the case with Natural Language Processing tasks, as the recent embedding algorithms convert words in large numerical vectors to achieve better scores.

This thesis shows that every generative model can define its induced classifier without using the observations' law. This proposition questions the usual categorization of the probabilistic models and classifiers and allows many new applications. Therefore, Hidden Markov Chain can be efficiently applied to Chunking and Naive Bayes to sentiment analysis.

We go further, as this proposition allows to define the classifier induced from a generative model with neural network functions. We "neuralize" the models mentioned above and many of their extensions. Models so obtained allow to achieve relevant scores for many Natural Language Processing tasks while being interpretable, able to require little training data, and easy to serve.

Institut Polytechnique de Paris
91120 Palaiseau, France