Leveraging textual embeddings for unsupervised learning

Stanislas Morbieu

Doctoral School of Computer Science, Telecommunication and Electronics (EDITE) Paris

Leveraging Textual Embeddings for Unsupervised Learning

by

Stanislas Morbieu

This dissertation is submitted for the degree of
Doctor of Computer Science
at Université de Paris

Committee:

Pr. Mohamed Nadif	Université de Paris, Supervisor
Dr. François Role	Université de Paris, Co-supervisor
Dr. Ahlame Chouakria	Université de Grenoble Alpes
Pr. Mohamed Quafafou	Aix Marseille Université
Pr. Nadia Ghazzali	Université du Québec à Trois-Rivières
Pr. Pierre-François Marteau	Université de Bretagne Sud
Mr François-Xavier Bois	Kernix

École doctorale Informatique, Télécommunications et Électronique
(EDITE) de Paris

Exploiter les Plongements Textuels pour l'Apprentissage Non Supervisé

par

Stanislas Morbieu

Thèse présentée en vue de l'obtention du titre de
Docteur en Informatique
à Université de Paris

Jury composé de :

Directeur de thèse	:	Mr. Mohamed Nadif	(Professeur, Université de Paris)
Co-encadrant	:	Mr François Role	(MCF, Université de Paris)
Rapporteur	:	Mme Ahlame Chouakria	(MCF, HDR, Université de Grenoble Alpes)
Rapporteur	:	Mr Mohamed Quafafou	(Professeur, Aix Marseille Université)
Examinateur	:	Mme Nadia Ghazzali	(Professeur, Université du Québec à Trois-Rivières)
Examinateur	:	Mr Pierre-François Marteau	(Professeur, Université Bretagne Sud)
Invité	:	Mr François-Xavier Bois	(Kernix)

Résumé

Les données textuelles sont omniprésentes et constituent un vivier d'information exploitable pour de nombreuses entreprises. En particulier, le web fournit une source quasiment inépuisable de données textuelles qui peuvent être utilisées à profit pour des systèmes de recommandation, de veille commerciale ou technologique, de recherche d'information, etc. Les récentes avancées en traitement du langage naturel ont permit de capturer le sens des mots dans leur contexte afin d'améliorer les systèmes de traduction automatique, de résumé de textes, ou encore le regroupement de documents suivant des catégories prédéfinies.

La majorité de ces applications reposent cependant souvent sur une intervention humaine non négligeable pour annoter des corpus : Cette annotation consiste, par exemple dans le cadre de la classification supervisée, à fournir aux algorithmes des exemples d'affectation de catégories à des documents. L'algorithme apprend donc à reproduire le jugement humain afin de l'appliquer pour de nouveaux documents.

L'objet de cette thèse est de tirer profit de ces dernières avancées qui capturent l'information sémantique du texte pour l'appliquer dans un cadre non supervisé. Les contributions de cette thèse s'articulent autour de trois axes principaux.

Dans un premier temps, nous proposons une méthode pour transférer l'information capturée par un réseau neuronal pour de la classification croisée de documents et de mots. La classification croisée consiste à partitionner simultanément les deux dimensions d'une matrice de données, formant ainsi à la fois des groupes de documents similaires et des groupes de mots cohérents. Ceci facilite l'interprétation d'un grand corpus de documents puisqu'on peut caractériser des groupes de documents par des groupes de mots, résumant ainsi une grande volumétrie de texte. Plus précisément nous entraînons l'algorithme Paragraph Vectors sur un jeu de données augmenté en faisant varier les différents hyperparamètres, classifions les documents à partir des différentes représentations vectorielles obtenues et appliquons un algorithme de consensus sur les différentes partitions. Une classification croisée contrainte de la matrice de co-occurrences entre les termes et les documents est ensuite appliquée pour conserver le partitionnement consensus obtenu. Cette méthode se révèle significativement

meilleure en qualité de partitionnement des documents sur des corpus de documents variés et procure l'avantage de l'interprétation offerte par la classification croisée.

Dans un second temps, nous présentons une méthode pour évaluer des algorithmes de classification croisée en exploitant des représentation vectorielles de mots appelées *word embeddings*. Les *word embeddings* sont des vecteurs construits grâce à de gros volumes de textes, dont une caractéristique majeure est que deux mots sémantiquement proches ont des *word embeddings* proches selon une distance cosinus. Notre méthode permet de mesurer l'adéquation entre la partition des documents et la partition des mots, offrant ainsi de manière totalement non supervisée un indice à l'utilisateur de la qualité de la classification croisée.

Dans un troisième temps, nous nous intéressons à la recommandation de petites annonces. Nous proposons un système qui permet de recommander des petites annonces similaires lorsqu'on en consulte une. Les descriptions des petites annonces sont souvent courtes, syntaxiquement incorrectes, et l'utilisation de synonymes font qu'il est difficile pour des systèmes traditionnels de mesurer fidèlement la similarité sémantique. De plus, la fréquence de renouvellement élevé des petites annonces encore valides (produit non vendu) implique des choix permettant de répondre à la contrainte du faible temps de calcul. Notre méthode, simple à implémenter, répond à ce cas d'usage et s'appuie de nouveau sur les *word embeddings*. L'utilisation de ceux-ci présente certains avantages mais impliquent également quelques difficultés : la création de tels vecteurs nécessite de choisir les valeurs de certains paramètres, et la différence entre le corpus sur lequel les *word embeddings* ont été construit en amont et celui sur lequel ils sont utilisés fait émerger le problème de « mots en dehors du vocabulaire », qui n'ont pas de représentation vectorielle. Nous présentons, pour palier ces problèmes, une analyse de l'impact des différents paramètres sur les *word embeddings* ainsi qu'une étude des méthodes permettant de traiter le problème de « mots en dehors du vocabulaire ».

Abstract

Textual data is ubiquitous and is a useful information pool for many companies. In particular, the web provides an almost inexhaustible source of textual data that can be used for recommendation systems, business or technological watch, information retrieval, etc. Recent advances in natural language processing have made possible to capture the meaning of words in their context in order to improve automatic translation systems, text summary, or even the classification of documents according to predefined categories.

However, the majority of these applications often rely on a significant human intervention to annotate corpora: This annotation consists, for example in the context of supervised classification, in providing algorithms with examples of assigning categories to documents. The algorithm therefore learns to reproduce human judgment in order to apply it for new documents.

The object of this thesis is to take advantage of these latest advances which capture the semantic of the text and use it in an unsupervised framework. The contributions of this thesis revolve around three main axes.

First, we propose a method to transfer the information captured by a neural network for co-clustering of documents and words. Co-clustering consists in partitioning the two dimensions of a data matrix simultaneously, thus forming both groups of similar documents and groups of coherent words. This facilitates the interpretation of a large corpus of documents since it is possible to characterize groups of documents by groups of words, thus summarizing a large corpus of text. More precisely, we train the Paragraph Vectors algorithm on an augmented dataset by varying the different hyperparameters, classify the documents from the different vector representations and apply a consensus algorithm on the different partitions. A constrained co-clustering of the co-occurrence matrix between terms and documents is then applied to maintain the consensus partitioning. This method is found to result in significantly better quality of document partitioning on various document corpora and provides the advantage of the interpretation offered by the co-clustering.

Secondly, we present a method for evaluating co-clustering algorithms by exploiting

vector representations of words called *word embeddings*. Word embeddings are vectors constructed using large volumes of text, one major characteristic of which is that two semantically close words have *word embeddings* close by a cosine distance. Our method makes it possible to measure the matching between the partition of the documents and the partition of the words, thus offering in a totally unsupervised setting a measure of the quality of the co-clustering.

Thirdly, we are interested in recommending classified ads. We present a system that allows to recommend similar classified ads when consulting one. The descriptions of classified ads are often short, syntactically incorrect, and the use of synonyms makes it difficult for traditional systems to accurately measure semantic similarity. In addition, the high renewal rate of classified ads that are still valid (product not sold) implies choices that make it possible to have low computation time. Our method, simple to implement, responds to this use case and is again based on word embeddings. The use of these has advantages but also involves some difficulties: the creation of such vectors requires choosing the values of some parameters, and the difference between the corpus on which the word embeddings were built upstream. and the one on which they are used raises the problem of *out-of-vocabulary words*, which have no vector representation. To overcome these problems, we present an analysis of the impact of the different parameters on word embeddings as well as a study of the methods allowing to deal with the problem of out-of-vocabulary words.

Remerciements

Je voudrais témoigner toute ma gratitude aux nombreuses personnes qui ont contribué à la réalisation de cette thèse. J'aimerais tout d'abord remercier mon directeur de thèse, Pr. Mohamed Nadif, qui m'a donné l'envie et l'opportunité de réaliser cette thèse au LIPADE. Ses conseils ont été très précieux ; sa disponibilité, son soutien et son implication m'ont stimulé et permis de mener à terme ce projet de thèse. Je tiens tout autant à remercier mon co-encadrant, Dr. François Role, qui m'a accompagné quotidiennement durant ces années. Sa rigueur et sa curiosité des aspects techniques ont contribué à enrichir ma réflexion et à persévérer pour réaliser de nombreuses expérimentations.

Je tiens à remercier François-Xavier Bois et Dr. Joseph Pellegrino pour la confiance qu'ils m'ont accordée, avoir encadré mon travail en entreprise, et m'avoir permis de réaliser la première thèse CIFRE au sein de Kernix.

Je remercie les rapporteurs de cette thèse, Dr. Ahlame Chouakria et Pr. Mohamed Quafafou pour leur lecture et l'intérêt qu'ils ont porté à mon travail. Je remercie également les autres membres du jury, Pr. Nadia Ghazzali ainsi que Pr. Pierre-François Marteau qui ont accepté de juger ce travail. Les échanges que j'ai eu avec eux sur mon travail me permettent d'envisager d'autres perspectives très intéressantes.

Nombreux sont ceux qui m'ont accompagné à l'université et à l'entreprise : je remercie Mohamad, Emira, Rafika et Mickael pour les bons moments passés ensembles et les discussions stimulantes au LIPADE. À Kernix, merci à tous mes collègues du data lab qui ont rendu ces années si particulières : Joseph, Imen, Jana, Aurélie, Charlotte, Mikael, Cristian, Eelaman, Matthieu, Éloïse, Mohamed, Guillaume et Mohammed. Toujours à Kernix, je remercie Pierre-Emmanuel et Géraldine qui ont contribué au système de recommandations de petites annonces présenté dans cette thèse. À tous mes autres collègues, qui comme Pierre-Edouard, ont montré leur curiosité pour mon travail, merci encore.

À mes amis de toujours, ma famille et à Emna, je tiens à vous remercier chaleureusement pour vos encouragements et d'avoir fait de cette période un moment si exceptionnel.

Contents

List of Figures

List of Tables

Introduction

Motivation

With the growing number of digital resources, textual data has become ubiquitous. For instance, a lot of information is distributed and made available over various pages across the web. This abundance of unstructured data makes it difficult to fully exploit the richness of the content.

Computational power of computers are of great importance to scale the handling of the vast amount of texts we encounter every day. Information extraction can automate and speed-up fastidious tasks by putting events on a calendar and trigger alarms automatically in advance, taking the travelling time into account; search engines can return relevant results against a query to avoid reading unrelated documents; automatic summarization enables insights at a glance, etc. To tackle all these opportunities, texts have first to be stored in an efficient way and then to be processed with adapted methods.

Textual data is usually stored as strings of characters and humans can get the meaning of it by splitting it in sequences of words. Both the sense of each word individually and the combination of them are used to understand the semantics. This enables us to compare two texts and measure intuitively a level of semantic similarity. But, for a computer to process texts in order to make such semantic comparisons, the strings of characters representation is not usable directly. We need to embed the texts into a representation that can be manipulated. A vector representation is of great interest since we can define similarity measures: For instance the inverse of the distance between two vectors can be used as a similarity score between two numerical vectors. Defining a suitable embedding method to transform a string of characters into a vector of real values allows to use operations on vectors to reflect the human notion of semantics.

Traditional methods to embed textual data rely on the representation of a document by a bag of words: a document is represented by a vector of dimensionality equal to the number of words in the whole vocabulary. Each word is assigned to a

dimension of the vector and the value for a dimension is the number of occurrences of the word in the document. A collection of documents is therefore represented by a matrix where each row is the vector representation of a document and each column is a vector representing a word. This so called Vector Space Model (VSM) is at the basis of many algorithms to represent and use textual data. For instance, one can weight this matrix to reduce the weights of the most frequent words in the collection of documents through a weighting scheme named Term Frequency - Inverse Document Frequency (TF-IDF). This whole process, named text vectorization, is a necessary step to use task specific algorithms such as classification, clustering, co-clustering, topic modeling etc: The vectors are easier to handle than raw text since one can manipulate them with mathematical operations.

Although these traditional vectorization methods have proved their practical ability to tackle several tasks involving textual data, we have seen a resurgence of works on textual representations in recent years. New neural network based methods have made breakthroughs in the natural language processing (NLP) field. Some big companies such as Google and Facebook and famous laboratories are competing for constantly new state of the art textual representation models in NLP: ELMo, FastText and BERT for instance are the most recent models which draw attention of the NLP community. They are trained on tremendous amount of textual data in an unsupervised manner to capture the semantics of the text.

The semantic information encoded into these embedding models is exploited in a downstream task of interest. The process of capturing information on a big data set and using the information on another data set is called transfer learning. It also designates the act of training for a specific task (the task of embedding the text in this case) and use captured information for another task (the final downstream task). The above cited models are proven to be particularly good at transfer learning.

These new embedding models are mainly exploited through supervised settings for tasks such as classification on which they work astonishingly well. Their use in unsupervised settings is however not well covered in the scientific literature since it is not trivial to correctly leverage the information contained in the embeddings: For supervised tasks, the task specific model can use parameterized non linear transformations which have to be learned. In the unsupervised setting however, the parameters have to be discovered by human supervision, requiring thus a considerable amount of time and energy which costs too much to be practical.

The aim of this thesis is to contribute to the adoption of the new textual embedding methods on several unsupervised tasks:

- Transfer learning for clustering of documents;
- Evaluation of text co-clustering;

– Semantic similarity based retrieval of short texts for recommendation engines.

Transfer Learning for Clustering of Documents

Clustering techniques play an increasingly important role since they allow users to identify interesting groups of objects from among huge amounts of data. For example, when dealing with textual data, clustering techniques allow to divide a set of documents into groups so that documents assigned to the same group are more similar to each other than to documents assigned to other groups. In information retrieval, the use of clustering relies on the assumption that if a document is relevant to a query, then other documents in the same cluster can also be relevant. This hypothesis can be used at different stages in the information retrieval process, the two most notable being: cluster-based retrieval to speed up search, and search result clustering to help users navigate and understand what is in the search results.

Transfer learning for clustering of documents have practical benefits, particularly in the case of an evolving database of documents. Let's consider two examples:

– Classified ads are published continuously and at some time, some of them might no longer be relevant because the object is sold out.
– News articles are relevant only in a given timeframe. The user who wants to keep up with the actuality needs only fresh articles of the preceding days.

In these cases, we only need to cluster a part of the documents, those which are not outdated. We can however gain from semantic knowledge learned on the outdated documents: the meanings of most of the words often don't change with the context. In this framework, clustering on a given dataset (the documents of interest) but learning from an extended dataset might be relevant.

Evaluation of Text Co-clustering

Co-clustering is an extension of one-sided clustering, and consists in simultaneously clustering the rows and columns of a data matrix. When applied to a document-term matrix, the goal is to identify document-term co-clusters, that is the association between a document cluster and a term cluster containing terms that can serve as good labels for describing the document cluster. Many algorithms for text co-clustering have been proposed over the years. However, whatever the technique used, most studies evaluate the performance of co-clustering algorithms on the quality of the obtained document clusters alone, just like for standard one-sided clustering. This is not satisfactory since co-clustering is at its heart a method for creating co-clusters of words and documents. A first improvement is to consider the quality of the obtained term clusters in addition to that of the document clusters. However, to really do

justice to the two-dimensional nature of co-cluster analysis, it is not enough to assess the quality of the word and document clusters in isolation of each other: measures are needed to evaluate the quality of the matching between the document and term clusters and their potential to form useful co-clusters.

Also, given the extreme difficulty in obtaining labels for both documents and words, it is highly desirable that such measures be *unsupervised*, meaning measures that do not need prior knowledge about the ground-truth classes.

Semantic Similarity Based Retrieval of Short Texts for Recommendation Engines

Short texts are ubiquitous, encountered on microblogs, discussion forums, multimedia sharing sites, product reviews, image captions, personal status messages, classified ads, snippets results of search engines, etc. But because of their main characteristics, they are difficult to handle with traditional document vector representations.

Since they contain few words, they lead to a sparsity issue: if a document is represented as a *one hot vector* in the vocabulary space, the vector is often more sparse for a short document than a longer one. Thus two short documents about the same topic have little chance to use the same words and a similarity measure between the two document vectors gives a value close to zero. Longer documents have a higher probability to use common words if there are about the same topic.

Although it is not directly related to the length of the content, it appears that short texts often contain more mistakes than longer ones. Abbreviations and typos are common due to the facility to create and share texts without quality restriction. Press articles and official documents are indeed often longer than user generated content and contain less typos.

Taking these characteristics into account, a similarity measure which capture the intention of the writers and translate the human notion of semantics is of major importance. Also given the high renewal rate of these types of content, fast computing time is required.

Contributions

The main contributions of this thesis are:

- an end to end unsupervised framework to explore a set of textual documents with co-clustering by leveraging transfer learning and local co-occurrences of words;
- a method to evaluate text co-clustering algorithms by assessing the good fit between document and term clusters;

– a highly scalable method to compute semantic similarity between short texts by leveraging word embeddings along with hints on how to select word embeddings hyper-parameters and deal with the out-of-vocabulary words problem.

The works drove to software contributions:

– an open source and publicly available Python package for co-clustering;
– a highly scalable processing pipeline to compute similarity scores between short texts along with a platform which serves as a backbone to recommender engines at Kernix.

Overview

– **Chapter 1** reviews the major textual embeddings methods. It presents first the embeddings at the word and sub-word levels, then at the phrase and document levels, and last the new emerging methods which treat both word and document levels jointly.

– **Chapter 2** presents co-clustering and its importance in text mining. More specifically, it shows how co-clustering is adapted in the case of sparse and high dimensional data such as textual data. It provides practical illustrations of visualizations that enable the exploration and interpretation of large textual datasets.

– **Chapter 3** is devoted to leveraging document embeddings for transfer learning in the context of clustering and co-clustering.

– **Chapter 4** is about unsupervised evaluation of text co-clustering algorithms using neural word embeddings. More specifically, it presents how to measure the good fit between term and document clusters in an unsupervised fashion by leveraging publicly available word embeddings matrices. Moreover, it explains how to improve text co-clustering by noise identification.

– **Chapter 5** discusses a major challenge of word embeddings: The data on which the word embeddings have been trained differs from the data on which they are used, so the so-called *out-of-vocabulary problem* appears: many words encountered in the downstream task may be missing in the word embeddings vocabulary. It exposes evaluation methods of out-of-vocabulary word embeddings substitution.

– **Chapter 6** demonstrates the advantages word embeddings offer for an application to a scalable recommender system for classified ads. It shows how a simple semantic similarity measure between documents derived from word embeddings

can be implemented. Processing textual data at scale in streaming is at the core of the platform presented here. It also illustrates how this generic platform is used for a diversity of recommender projects at Kernix.

Chapter 1

Textual embeddings

Textual data is usually stored as strings of characters and humans can get the meaning of it by splitting it in sequences of words. Both the sense of each word individually and the combination of them are used to understand the semantics. This enables us to compare two texts and measure intuitively a level of semantic similarity. But, for a computer to process texts in order to make such semantic comparisons, the strings of characters representation is not usable directly. We need to embed the texts into a representation that can be manipulated. A vector representation is of great interest since we can define similarity measures which operate on vectors. Defining a suitable embedding method to transform a string of characters into a vector of real values allows to use operations on vectors to reflect the human notion of semantic.

The outline of this chapter is as follows: First we present how to transform words as vectors. Then we present how to vectorize documents. Then, we present methods that combine the word and document representations.

1.1 Vector Representation of Words

Representing words as vectors facilitates their handling. Indeed, cosine can be used as a way to translate semantic similarity between words; algebraic operations on vectors allow to extract syntactic and semantic regularities (Mikolov et al., 2013d; Levy et al., 2014); topics are discovered by studying the dimensions; composing vectors like summing and averaging can be used to represent documents (Zhu et al., 2016).

This section presents how to represent words as vectors. A special case of vector representation of words, named *word embeddings*, is introduced by emphasizing its specificity and advantages over the other representations. The main word embeddings models are then presented. Last, their characteristics are discussed along with evaluation methods.

1.1.1 From Words to Word Embeddings

This section introduces word embeddings by presenting the data on which the models are based, then by presenting vectors derived from it which are high dimensional, then by presenting the value of low dimensional vectors and last by presenting the added value of word embeddings.

1.1.1.1 Underlying Data

In the Vector Space Model (Salton et al., 1975), a document is represented as a vector in a space where each dimension corresponds to a word. The value for a dimension is the number of occurrences of the word in the document. This (document × word) matrix can be transposed, thus providing a vector representation for a word.

The resulting (word × document) matrix is in fact a special case of a (word × context) matrix were the context is a document. Instead of a document, the context can be a paragraph, a sentence or a phrase. It can also be a window of surrounding words (Lund and Burgess, 1996) (Figure 1.1). Sahlgren (2006) provides a comprehensive study of various contexts.

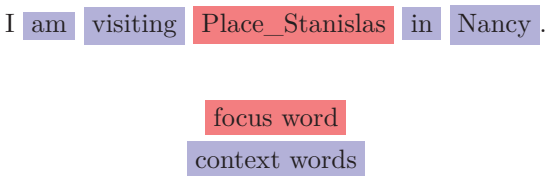


Figure 1.1: Context window of a focus word.

The idea to represent a word by its context follows the *distributional hypothesis* in linguistics (Harris, 1954) which states that words that occur in similar contexts tend to have similar meaning.

The number of occurrences is not the only way to weight the (word × context) matrix: Hyperspace Analogue to Language (Lund and Burgess, 1996) provides a different word-space implementation where the weights depend on the distance between the words.

1.1.1.2 High Dimensional Vectors

High dimensional vectors are derived from the data by weighting the values of the (word × context) matrix. Two families of weighting schemes are mainly used: Term Frequency - Inverse Document Frequency (TF-IDF) and those derived from Pointwise Mutual Information.

The (document × word) matrix is often weighted in information retrieval with TF-IDF. Each entry of the weighted matrix is usually composed of two components (Salton and Yang, 1973):

$$f_{ij} = TF_{ij} \times DF_i$$

where:

- TF is a function of the frequency of term i in document j, so that words which occur less are less discriminant and have a smaller weight.
- DF is a function of the number of documents term i occurs in, so that words which occur in many documents have a smaller weight and thus are less discriminant. The most common version is the *inverse document frequency*: $IDF = \log \frac{D}{df_i}$ with D the number of documents and df_i the number of documents term i occurs in.

In practice, only the TF component is used for the (word × document) matrix in word similarity.

Pointwise Mutual Information (PMI) measures the strength of association between two words. It is used either directly as a similarity measure between two words or indirectly as a weighting scheme for the (word × context) matrix. For this last case, a word has a vector representation.

PMI is first presented and then an often used derivative, Positive Pointwise Mutual Information (PPMI). A comparative study of variants of PMI can be found in (Role and Nadif, 2011).

Pointwise Mutual Information (PMI) is given by:

$$\text{PMI}(w, c) = \log \frac{P(w, c)}{P(w) \times P(c)}$$

where:

- $P(w, c)$ is the joint probability of the word w and the context word c.
- $P(w)$ and $P(c)$ are the marginal probabilities of w and c respectively.

For pairs (w, c) that were never observed, $\text{PMI}(w, c) = \log 0 = -\infty$ so PMI is in practice often replaced with positive PMI (PPMI), which replaces negative values with 0, yielding:

$$\text{PPMI}(w, c) = \max(\text{PMI}(w, c), 0)$$

Other weighting schemes exist, leading to several *Vector Space Models of Semantics* surveyed in (Turney and Pantel, 2010). For example, Rohde et al. (2006) takes the square root of co-occurrence counts before applying Singular Value Decomposition to reduce the dimensionality.

1.1.1.3 Dense Low Dimensional Vectors

In the previous section, the vectors representing the words are high dimensional leading to scalability issues since the computations involving them are expensive and the memory usage is important. Moreover, in a geometric perspective, distances between vectors become relatively uniform in high dimensional spaces. This phenomenon is known as the *curse of dimensionality*.

Several approaches are taken to tackle the *curse of dimensionality* by representing the data in a space with a number of dimensions nearer its intrinsic dimensionality.

Traditional Word Embeddings. Schütze (1993) apply truncated Singular Value Decomposition as a dimensionality reduction on the (word × document) matrix of occurrences in the same manner as in Latent Semantic Analysis (Deerwester et al., 1990). Hellinger Principal Component Analysis on the word co-occurrence matrix is applied in (Lebret and Collobert, 2014).

Neural Word Embeddings. Word embeddings are coined by Bengio et al. (2003) where each word is represented as a continuous real vector with a number of features much smaller than the size of the vocabulary. It has been popularized by Mikolov et al. (2013a) along with the implementation *word2vec*[1] which scales well.

Word embeddings are often associated with the neuronal approach where the underlying data matrix is not explicit. The link between the neuronal models and a factorization of the (word × context) matrix is shown in (Levy and Goldberg, 2014). The key points are that the vectors should be dense, real-valued and low dimensional to tackle the *curse of dimensionality*. They also should scale well in order to train the models on a big amount of data to generalize better. This last requirement leads to consider approaches where the (word × context) matrix is not built beforehand.

1.1.2 Word Embeddings Models

Word embeddings models can be divided in different ways. An often used categorization is to separate the *count* from the *predict* methods (Baroni et al., 2014). This distinction does not hold since models are based on the same underlying data and *predict* methods can be casted as *count* methods (Levy and Goldberg, 2014).

Here we choose to separate the models into:

– methods that operate on the global data at once, suffering thus often from memory issues when dealing with big data;

[1] `https://code.google.com/archive/p/word2vec`

– methods that operate on shallow windows in an incremental manner, thus avoiding memory scalability issues;
– methods that handle data in streaming, thus avoiding heavy updates when the vocabulary changes;
– methods that emphasize on disambiguation by providing topical information and multiple word meaning;
– methods that allow to project words from different languages into a shared embedding space.

This categorization is not perfect since it is fuzzy: global matrix factorization can be computed in a relatively scalable manner (Řehůřek and Sojka, 2010), shallow window-based methods can be casted as global factorization methods (Levy and Goldberg, 2014), and topical information can be extracted from models which are not specifically designed to handle multiple meaning.

This categorization is however meaningful since the first three categories are distinguished by how the data is fed into the model when building it, and the scalability level they are associated with. The last two categories are orthogonal to the others and are thus in distinct sections.

1.1.2.1 Methods Handling a Global Matrix

This section presents methods that need to construct first a global matrix as described in section 1.1.1.1 before actually building the word embeddings model.

Latent Semantic Analysis. Latent Semantic Analysis, named also Latent Semantic Indexing in the context of information retrieval (Deerwester et al., 1990), is designed to provide a similarity measure between documents. A (document × word) matrix is built with the values given by the number of occurrences of the words in the documents. Then a dimensionality reduction of the columns is done through Truncated Singular Value Decomposition. The similarity measure is given by the cosine of the vectors representing the documents (i.e. the rows of the reduced matrix).

Similarly, Schütze (1993) builds vectors to represent the words by applying the dimensionality reduction on the transpose of the (document × word) matrix. Řehůřek (2011) presents detailed explanations on considerations when implementing Singular Value Decomposition: batch implementation, small batches, parallelization, online training with multiple or single pass, approximations. Řehůřek and Sojka (2010) provide a scalable online implementation.

Word Embeddings through Hellinger PCA. Since co-occurence statistics are discrete distributions, Hellinger distance, which is more appropriate than the Euclidean

distance over a discrete distribution space, is used in (Lebret and Collobert, 2014). For two discrete probability distributions $P = (p_1, \ldots, p_k)$ and $Q = (q_1, \ldots, q_k)$, the Hellinger distance is defined by:

$$H(P, Q) = -\frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{k} (\sqrt{p_i} - \sqrt{q_i})^2}$$

The authors perform a principal component analysis (PCA) of the word co-occurrence probability matrix that minimize the reconstruction error according to the Hellinger distance.

GloVe. GloVe (Pennington et al., 2014b) is a model which aims to encode the information contained in the ratio of the co-occurrence probabilities of two words as vector differences. The objective is to minimize the cost function:

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b_j} - \log X_{ij})^2$$

where:

- V is the size of the vocabulary;
- w_i and $\tilde{w}_j$ are the vectors of the focus word and a context word respectively (Figure 1.1);
- b_i and $\tilde{b_j}$ are scalar biases for the focus and context words respectively;
- X_{ij} represents how often word i appears in context of word j;
- f is a weighting function that assigns relatively lower weight to rare and frequent co-occurrences, defined by:

$$f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^{\alpha} & \text{if } x \leq x_{max} \\ 1 & \text{otherwise} \end{cases}$$

  where α is a parameter and x_{max} is a cutoff parameter.

The optimization is done with AdaGrad (Duchi et al., 2011) by training only on the nonzero elements in the co-occurrence matrix.

Singular Value Decomposition of a Pointwise Mutual Information Weighted Matrix. Levy and Goldberg (2014) apply a spectral dimensionality reduction over a shifted Positive Pointwise Mutual Information (PPMI) weighted matrix.

The dimensionality reduction is done through Singular Value Decomposition, discussed previously in 1.1.2.1. The PPMI matrix is discussed in 1.1.1.2.

Eigenwords. Word embeddings based on Canonical Correlation Analysis (CCA) are presented in (Dhillon et al., 2015). CCA computes the directions of maximal correlation between a pair of matrices and is scale invariant. It captures multi-view information: the left and the right contexts. The first method presented consists in computing the CCA of two matrices C and W where:

- $W \in \mathbb{R}^{n \times v}$ is the matrix where each row represents the position of the focus word in the corpus of size n, and each column represents the focus word in the vocabulary of size v. The value w_{ij} in the matrix is 1 if the j^{th} word in the dictionary is at the position i in the corpus.
- $C = [LR] \in \mathbb{R}^{n \times 2vh}$ where $L \in \mathbb{R}^{n \times vh}$ and $R \in \mathbb{R}^{n \times vh}$, h being the context size. Each row of L represents the position of the focus word. Each column of L represents the left context word: a column for each pair (position in the left context, position in the vocabulary). $l_{c_{wkj}} = 1$ if the j^{th} focus word co-occurs with the context word w, w being in the position k in the left context. R is the same as L but for the right context.

The authors present also a two step CCA to have better sample complexity for rare words: it consists in computing CCA between R and L in a first step. In the second step, the resulting projections are concatenated in a matrix and CCA is applied between the resulting matrix and W. The optimization involved in CCA can be solved by Singular Value Decomposition. See 1.1.2.1 for implementation considerations.

WordRank. WordRank (Ji et al., 2015) views word embeddings generation as a ranking problem. The objective is to minimize:

$$J(U, V) = \sum_{w \in W} \sum_{c \in \Omega_w} r_{w,c} \cdot \rho \left(\frac{\overline{\text{rank}}(w, c) + \beta}{\alpha} \right)$$

where:

- $r_{w,c} = f(x_{w,c})$ with f defined as in GloVe (see 1.1.2.1) quantify the association between the word w and the context word c;
- $\alpha > 0$ and $\beta > 0$ are hyperparameters;
- $U = \{u_w\}_{w \in W}$ is the set of embeddings of words and $V = \{v_c\}_{c \in C}$ is the set of embeddings of context words;
- W is the vocabulary;
- Ω_w denotes the set of contexts that co-occurred with a given word w;
- ρ is a monotonically increasing and concave ranking loss function that measures goodness of a rank;
- $\overline{\text{rank}}$ is a convex upper bound on the rank:

$$\overline{\text{rank}}(w, c) = \sum_{c' \in C \setminus \{c\}} l(u_w \cdot (v_c - v_{c'}))$$

where l is a loss function for binary classification such as the hinge loss $l(x) = \max(0, 1 - x)$ or the logistic loss $l(x) = \log_2(1 + 2^{-x})$.

The authors provide an implementation[2] which rely on Message Passing Interface (MPI) to handle large corpus since the computation is expensive.

1.1.2.2 Shallow Window-based Methods

Models that operate on a global matrix require to construct the (word $\times$ context) matrix beforehand. In this section, the models that operate directly on the windows of words are presented.

Feedforward Neural Net Language Model and Recurrent Neural Net Language Model are reviewed for their impact on the raise of the word embedding field but as discussed after, word2vec models or FastText are more scalable and lead to better quality word embeddings.

Feedforward Neural Net Language Model. Feedforward Neural Net Language Model is presented in (Bengio et al., 2003) and is the seminal work on word embeddings. The architecture (Figure 1.2) consists of four layers: input, projection, hidden and output layers. The context consists of n words preceding the focus word. The input is a *one hot vector* for each context word: a vector filled with zeros and with a one at the position given by the index of the context word in the vocabulary. The context words are projected on the projection layer which is a $n \times d$ matrix, d being the embedding size. The third layer is the hidden layer of size 500 to 1000 and the output layer has the same size as the vocabulary. The objective is to maximize:

$$L = \frac{1}{T} \sum_i \log f(w_t, w_{i-1}, \ldots, w_{i-n}; \theta) + R(\theta)$$

where:

- θ is the set of parameters to find;
- $R(\theta)$ is a regularization term;
- f is the function defined by the network;
- T is the number of tokens in the corpus;
- w_i is the word representing the i-th token in the corpus;
- n is the number of words in the context window.

The computational complexity is given in 1.1.3.1. The computation between the projection and the hidden layer is expensive since the values in the projection layer are dense.

[2] `https://github.com/shihaoji/wordrank`

[3] Source: (Bengio et al., 2003)

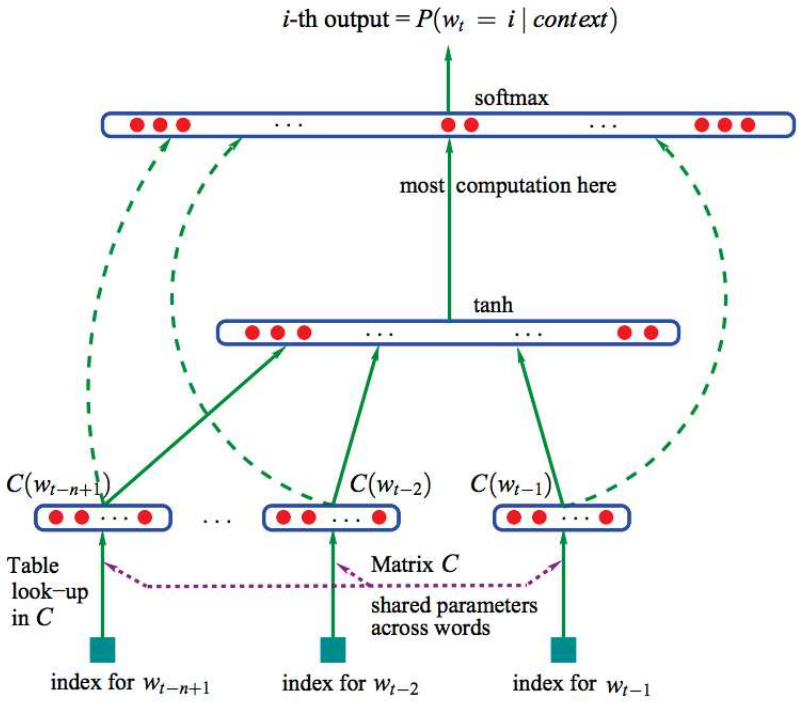


Figure 1.2: Feedforward Neural Net Language Model architecture. [3]

Recurrent Neural Net Language Model. Recurrent Neural Net Language Model is presented in (Kombrink et al., 2011). It overcomes the limitations of the Feedforward Neural Net Language Model by not having a projection layer nor a context size to specify. The *recurrent* term comes from the fact that the hidden layer is connected to itself. The computational complexity is given in 1.1.3.1.

This model is not presented in detail here since it has been superseded by simpler models as state of the art neuronal word embeddings models presented in the following section.

Word2vec. *Word2vec* is the name of the implementation of two models described in (Mikolov et al., 2013a) and (Mikolov et al., 2013c): *Continuous Bag of Words* (CBoW) and *Skip-gram*. The name is also used for the family of these models, with the two architectures and the different training strategies.

Continuous Bag of Words (CBoW) is introduced in (Mikolov et al., 2013a). The principle consists in trying to predict the focus word with its context. The architecture (Figure 1.3) consists in a fully connected neural network with one hidden layer. The input is a *one hot vector* for each context word: a vector filled with zeros and with

a one at the position given by the index of the context word in the vocabulary. The input size is thus the dimensionality of the vocabulary. The output is a vector for the focus word, with the same size as the input vectors since the training is done by comparing the output of the network with the *one hot vector* of the focus word. The activation values of output layer neurons are converted to probabilities using the *softmax* function: $y_j = Pr(word_j|word_{context}) = \frac{e^{activation(j)}}{\sum_{k=1}^{|V|} e^{activation(k)}}$ is the j^{th} value of the output, $|V|$ being the size of the vocabulary. The hidden layer is where the word embeddings are materialized and its size is the one of the word embeddings.

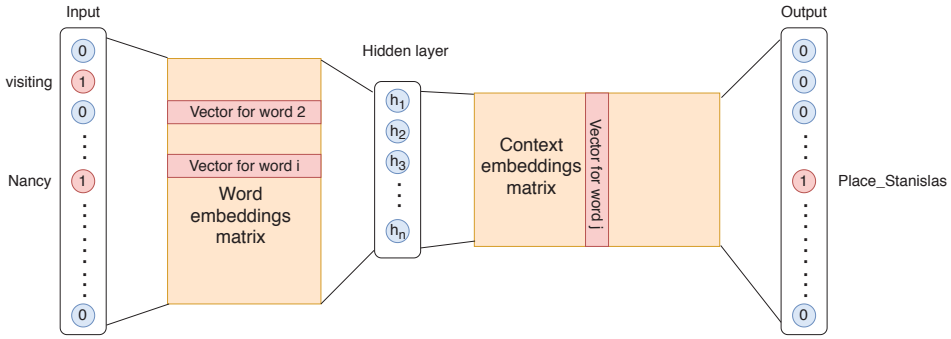


Figure 1.3: Continuous Bag of Words architecture.

The CBoW objective is to find the set of parameters θ that maximize the probability of predicting the focus words given their contexts:

$$\underset{\theta}{\operatorname{argmax}} \prod_{w \in T} \prod_{c \in C(w)} p(w|c;\theta) \tag{1.1}$$

where p corresponds to the *softmax* function, and $C(w)$ is the context words of the word w in the text T. A focus word is presented at the output of the neural network as a *one hot vector* and compared with the output of the network when providing the context vectors as input. The weights are updated with *stochastic gradient descent* and *backpropagation*. Details of the training process of this model are given in (Rong, 2014).

Skip-gram is also introduced in (Mikolov et al., 2013a). The architecture (Figure 1.4) is the reverse of the CBoW, the input being the focus word and the output being the context words. The skip-gram objective is to find the set of parameters θ that maximize the corpus probability:

$$\underset{\theta}{\operatorname{argmax}} \prod_{w \in Text} \prod_{c \in C(w)} p(c|w;\theta) \tag{1.2}$$

To follow the neural networks language models literature, $p(c|w;\theta)$ is modeled as

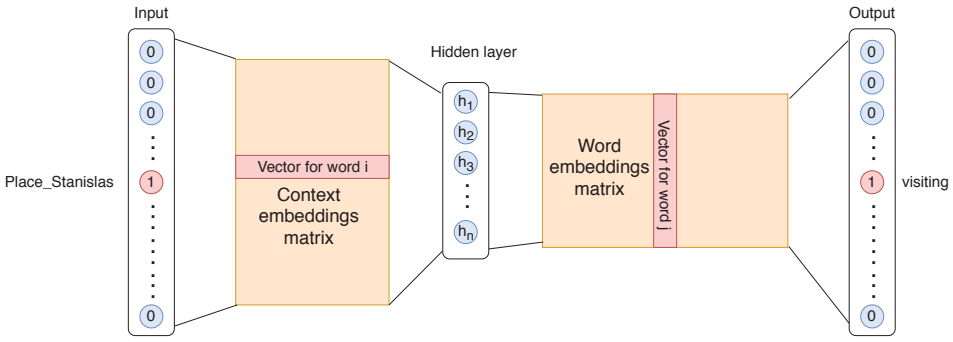


Figure 1.4: Skip-gram architecture.

a *softmax*. The objective is then:

$$\underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} (\log e^{v_c \cdot v_w} - \log \sum_{c'} e^{v_{c'} \cdot v_w}) \tag{1.3}$$

with D the set of all word and context pairs extracted from the text, v_w and v_c are vector representations of the word w and the context word c respectively, and c' is a word selected in the whole context vocabulary.

Computing $\log \sum_{c'} e^{v_{c'} \cdot v_w}$ suffering from scaling issues, *hierarchical softmax* is used instead of *softmax* which results in an approximation. The training is the same as with CBoW except that the input is the focus word and the output is a context word. Mikolov et al. (2013c) proposed negative sampling to speed up skip-gram training resulting in an other objective. The architecture is the same as for the skip-gram model. The objective, of skip-gram with negative sampling is:

$$\underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \sigma(v_c \cdot v_w) + \sum_{(w,c)\in D'} \log \sigma(-v_c \cdot v_w) \tag{1.4}$$

where σ is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

This model is explained in detail in (Goldberg and Levy, 2014). The objective is shown to be the same as a weighted logistic principal component analysis in (Landgraf and Bellay, 2017) and an information-theoretic interpretation is given in (Melamud and Goldberger, 2017). The main idea of negative sampling is to train binary logistic regressions for a true pair (focus word and context word) by maximizing the probability that context words appear, while minimizing the probability that noise words (random words) appear. The optimization is done with stochastic gradient descent. Instead of using stochastic gradient descent, Riemannian optimization framework is applied in (Fonarev et al., 2017) to optimize the objective.

FastText. FastText (Bojanowski et al., 2016) is essentially an extension of the word2vec models which treats each word as a compositon of character *n-grams*. So the vector for a word is made of the sum of its character n-grams. It generates better word embeddings than word2vec for rare words since even if words are rare their character n-grams are still shared with other words. A word embedding being a composition of its character n-grams embeddings, FastText can construct a vector for *out-of-vocabulary* words. The advantages on syntactic tasks and the scalability drawbacks are discussed after.

1.1.2.3 Streaming Word Embeddings

Peng et al. (2017) propose an incremental learning method of hierarchical softmax and provide thus an extension of word2vec models.

May et al. (2017) and Kaji and Kobayashi (2017) propose models based on Skip-gram with negative sampling that handle the training corpus in streaming. They differ on how they maintain an approximate vocabulary and how they estimate the negative sampling distribution.

1.1.2.4 Multi-sense Word Embeddings

Multi-sense word embeddings models are designed to handle the fact that a word can have different senses, also known as *polysemy*. It is noteworthy that the models relying on multiple prototypes have scalability issues since it is required to store a vector per sense. Therefore and since this field knows a proliferation of models, only the most recent ones are briefly presented. The reader interested in this field is invited to follow the state of the art given in the following articles.

Horn (2017) proposes an extension of the CBoW word2vec model with negative sampling called *context encoders (ConEc)*. An implementation is available[4]. It is designed to handle polysemy and also out-of-vocabulary problems. Shi et al. (2017) proposes a model which learn jointly word embeddings and a topic model. Each word has a vector representation for each topic.

1.1.2.5 Cross-lingual Word Embeddings

An extensive survey of cross-lingual word embeddings is available in (Ruder, 2017). Most of the methods consists in training a monolingual embedding model either on a created corpus in the case of *pseudo-cross-lingual* methods, or on two corpora before creating a mapping in the case of *monolingual mapping* methods. The other methods consist in training the models either on parallel on different corpora in order to optimize

[4] `https://github.com/cod3licious/conec`

a combination of monolingual and cross-lingual losses in the case of *joint optimization* methods, or on a parallel corpus in the case of *cross-lingual training* models.

These models will not be further discussed here since they rely on a preprocessing step in the case of *pseudo-cross-lingual* methods, or on a postprocessing step in the case of *monolingual mapping* methods, or are specific to multilingual data in the case of *joint optimization* and *cross-lingual training* models.

1.1.3 Characteristics and Evaluation

Models can be evaluated on two major criteria: the quality of the representation and how they can handle massive data sets. First we will discuss the scalability aspect and then the quality of the produced word embeddings. Last, parameter tuning is discussed because it has a great influence on the quality of a trained model and also have a practical scalability impact since parameter tuning can be time consuming.

1.1.3.1 Scalability

Handling massive data sets leads to consider the memory usage and the processing time. The memory usage should be kept under the available memory and mainly depends on the algorithm itself since the algorithm either should work on the whole data or on a part of it. The time to process the data depends on the computational complexity of the algorithm and also on the implementation since parallelization can lead to drastic improvements.

Word embeddings models have previously been distinguished in function of their handling of the data:

- handling global data at once;
- operating on shallow windows in an incremental manner;
- handling data in streaming.

Methods have been developed to approximate the factorization of the global matrix in order to process it in small batches that fit in memory or in an incremental manner (Řehůřek, 2011) but handling shallow windows of words without computing the whole matrix avoids the storage of the matrix in a slower memory (hard disk or solid state drive).

FastText needs to store character n-grams embeddings so it requires more space than word2vec models. Methods that produce multiple word embeddings for a word in order to represent its different meanings require more space since a vector is stored for each sense.

On-line window-based methods scale like $O(C)$ where C is the size of the corpus. The complexity of the GloVe model is in the worst case $O(V^2)$ where V is the size

of the vocabulary but Pennington et al. (2014b) shows that it scales like $O(C^{0.8})$ in practice. Řehůřek (2011) gives complexities of several implementations of Singular Value Decomposition which is the ground of several word embeddings methods that work on the (word × context) matrix.

Constant factors have a great impact on the training time. Thus, the computational complexity is often given as the number of parameters to estimate. Mikolov et al. (2013a) give the computational complexity per each training example Q for the following models: *Continuous Bag of Words* and *Skip-gram* implemented in *word2vec* and the two language models *Feedforward Neural Net* (Bengio et al., 2003) and *Recurrent Neural Net* (Kombrink et al., 2011).

The *Continuous Bag of Words* model has a computational complexity per each training example given by the formula:

$$Q = N \times D + D \times \log_2(V)$$

where:

- N is the context length;
- D is the dimensionality of the word representation;
- V is the size of the vocabulary.

The term $\log_2(V)$ is given by the hierarchical softmax where the vocabulary is represented as a Huffman binary tree. The hierarchical softmax is an approximation of the softmax that reduces the complexity by avoiding the expensive normalization over all words. Instead, the value for each word is given at the leaves of a tree and the probability of observing a word is decomposed into a sequence of probabilities: following the path from the root to the leaf (i.e. the word).

The *Skip-gram* model has a computational complexity per each training example given by the formula:

$$Q = K \times (D + D \times \log_2(V))$$

where:

- K is the maximum distance of the words (see the following note);
- D is the dimensionality of the word representation;
- V is the size of the vocabulary.

For each training word named the *focus* word, a number R is selected between 1 and K. Then R words are selected in the history and R words in the future of the word (in the context window). The selected $R + R$ words are given as correct output labels with the focus word as input.

The *Feedforward Neural Net Language Model* model has a computational complexity per each training example given by the formula:

$$Q = N \times D + N \times D \times H + H \times V$$

where:

- N is the number of previous words: the context length;
- D is the dimensionality of the word representation;
- H is the size of the hidden layer;
- V is the size of the vocabulary.

V can be reduced to $\log_2(V)$ if using hierarchical softmax. Thus, most of the complexity comes from the term $N \times D \times H$. Since H is typically 500 to 1000 units, this model have a greater complexity than the two *word2vec* models.

The *Recurrent Neural Net Language Model* model has a computational complexity per each training example given by the formula:

$$Q = H \times H + H \times V$$

where:

- H is the size of the hidden layer;
- V is the size of the vocabulary.

V can also be reduced to $\log_2(V)$ if using hierarchical softmax. Thus, most of the complexity comes from the term $H \times H$. Since H is typically 500 to 1000 units, this model have a greater complexity than the two *word2vec* models.

Implementations of methods that use linear algebra can rely on a set of basic linear algebra subprograms (BLAS) (Blackford et al., 2002) that are conceived to implement efficiently operations on vectors and matrices. Those methods are used implicitly when using higher level libraries such as NumPy and SciPy (van der Walt et al., 2011) which rely on a shared BLAS library. The choice of the implementation should however be taken with care in order to gain an order of magnitude in speed. It is noteworthy that some Gensim (Řehůřek and Sojka, 2010) implementations rely on Cython (Behnel et al., 2011) to speed up the computation, and since it is optional, forgetting to install it tears down the performances.

Software libraries such as Tensorflow (Abadi et al., 2015), that were primarily designed for neural networks, leverage graphical processing unit (GPU) in order to parallelize computation and improve drastically the computational time compared to processing in the central processing unit (CPU).

1.1.3.2 Quality

The ability to train models on huge amount of data allows to increase the quality of the embeddings. This section provides an overview of the methods to evaluate the quality of the models.

Perplexity. Perplexity is a way of evaluating language models, i.e. a probability distribution over texts. Word embeddings models derived from language modeling can thus be evaluated with this measure. However, not all word embeddings models are language models. Therefore, perplexity is not a universal quality measure. Moreover, as stated in (Chang et al., 2009b) for topic models, perplexity disagrees with human judgments.

Word Similarity Task. The word similarity task consists in retrieving words that are similar to a given word. The words are ranked according to a similarity measure between two word embeddings. Then, Spearman correlation coefficient between the ranked list given by the model and the one given by human judgment is used as the evaluation metric.

The word similarity is often evaluated using cosine similarity. This is possible when each word has only one vector representation.

For evaluating models for which a word has several vector representations, Shi et al. (2017) proposes two similarity measures:

- AvgSimC is the averaged similarity between two words over the assignments of topics:

$$AvgSimC(w_i, w_j) = \sum_{z_i} \sum_{z_j} p(z_i|w_i, c_{w_i}) p(z_j|w_j, c_{w_j}) \times \cos(U_{w_i,z_i}, U_{w_j,z_j})$$

  where:
  - w_i is the word i;
  - $U(w_i, z_i)$ is the embedding vector of w_i under the topic z_i;
  - $p(z|w, c_w, d)$ is the posterior topic distribution;
  - d is a document.
- MaxSimC measures the similarity between the most probable vectors of each word (i.e. with the higher probability for the word and the context to belong to the same topic):

$$MaxSimC(w_i, w_j) = \cos(U_{w_i,z_i}, U_{w_j,z_j})$$

  where:

$$z = \underset{z}{\operatorname{argmax}}\, p(z|w, c)$$

Stanford's Contextual Word Similarity (SCWS) data set is published and presented in (Huang et al., 2012). It includes 2003 word pairs and their context sentences. The ground truth similarity score was labeled by humans between 0 and 10, according to the semantic meaning of the words in their given contexts.

Faruqui et al. (2016) present problems of the word similarity task:

- This task is subjective since word similarity and relatedness are distinct concepts often confused.
- Some models are task specific so evaluation on word similarity can penalize them.
- There is no standardized splits between train and test sets so there is possibility for overfitting.
- This task has low correlation with extrinsic evaluation.
- Statistical significance is omitted.
- Some pairs of words with similar frequency are found to be closer in the vector space but should be further according to their word meaning.
- The polysemy is often not well handled in word semantic tasks.

Word Analogy Task. The word analogy task consists in answering queries of the form a:b;c:? where ? must be semantically or syntactically related to c in the same way as b is related to a.

Two types of tasks are to be distinguished: the semantic tests and the syntactic tests. An example of semantic query is king:queen;man:? where ? is expected to be "woman". An example of syntactic query is see:saw;return:? where ? is expected to be "returned".

The models behave differently on these two different tests: FastText is better than word2vec for the syntactic tests since it takes advantage of character sequence information but has no advantage on the semantic tests. Skip-gram is better than CBoW on semantic tests but it is the reverse for the syntactic tests.

In order to answer a query, Mikolov et al. (2013d) propose to search the word which normalized to unit norm embedding is closer in the sense of the cosine similarity to $x_b - x_a + x_c$ where x_a, x_b and x_c are the normalized to unit norm embeddings of the words a, b and c respectively:

$$w^* = \underset{x_w}{\text{argmax}} \frac{x_w.y}{||x_w||||y||}$$

with $y = x_b - x_a + x_c$.

Levy et al. (2014) propose to use two objectives:

- 3COSADD which is the objective defined by Mikolov et al. (2013d):

$$\underset{x_w}{\text{argmax}}\, cos(x_w, x_b - x_a + x_c)$$

- 3COSMUL:

$$\underset{x_w}{\text{argmax}} \frac{cos(x_w, x_b)cos(x_w, x_c)}{cos(x_w, x_a) + \epsilon}$$

  where $\epsilon = 0.001$ is used to prevent division by zero. As it requires that all similarities be non-negative, cosine similarities are transformed to $[0, 1]$ using $\frac{x+1}{2}$.

The limitations presented in 1.1.3.2 are applicable also for the word analogy task since Levy et al. (2014) show that the analogy task is equivalent to computing a linear combination of word similarities. Hartmann et al. (2017) show that word analogies results differ from the task-specific evaluations presented hereafter.

Compositional Approaches for Representing Relations between Words. The word analogy task relies on extracting information concerning the relation existing between two words in a word-pair. Examples of semantic relations between words are: synonymy, antonymy, meronymy and hypernymy.

Hakami and Bollegala (2017) go further by comparing compositional operators in order to capture different types of relations. The compositional operators take as input two word embeddings and produce a vector representing the relation. The concatenation of the two vectors, element-wise multiplication between them, difference and addition are considered as compositional operators by the authors. Note that the last three operators are element-wise operators and assume thus that the dimensions of the word representation space are linearly independent. The authors thus considered cross-dimensional operators but did not obtain significant improvement in performance and explain this by the correlation coefficients between two distinct dimensions being close to zero indicating uncorrelated dimensions.

Downstream Tasks. Nayak et al. (2016) present an ensemble of tasks to evaluate word embeddings: part-of-speech tagging and chunking to evaluate syntactic properties; named entity recognition, sentiment classification and question classification, to evaluate the semantic properties; and phrase-level natural language inference to evaluate the encoding of lexical relation information.

Interpretability. An interpretability task for word embeddings is presented in (Murphy et al., 2012b) and is derived from a task used to evaluate topic models in (Chang et al., 2009b). The task consists in taking for each dimension 5 words which have the 5 highest values in the dimension. An intruder word is added to this set and humans should detect the intruder. The intruder is taken to have the value in the second half of the ranked values for the dimension and also in the top 10^{th} percentile of another dimension. Human judgments are collected through *Amazon Turk Task*[5]. Precision (the fraction of retrieved results that are relevant to the query) is used as the evaluation metric.

[5] `http://mturk.amazon.com`

1.1.3.3 Parameter Tuning

The number of parameters is a characteristic that distinguishes the word embeddings models: a model with less parameters, or parameters that have less influence on the quality of the trained model, is easier to tune and is thus often preferred. Indeed, Levy et al. (2015) show that much of the performance gains of word embeddings are due to hyper parameters optimization.

The word embedding models that produce multiple vector representations per word like in (Huang et al., 2012), require to set the number of topics and thus require to have expert knowledge of the corpus used to train the model. A fixed number of topics makes these models not usable in a streaming scenario since the number of topics can evolve.

Setting the parameters consists also in a tradeoff. In terms of quality, smaller windows capture more syntactic information, larger ones more semantic and relational information. For FastText, the minimum and maximum n-gram sizes has a direct bearing on the training time and also on the quality. For word2vec, as noted in (Peng et al., 2017), it is empirically shown that hierarchical softmax performs better for infrequent words while negative sampling performs better for frequent words.

1.1.4 Conclusion

In this section, the use of word embeddings was motivated by presenting their strengths compared to the other vector representations of words: tackling the *curse of dimensionality*.

Moreover, models of word embeddings were presented with a focus on the ones that handle windows of words when training, in order to scale when training on a big corpus, and thus capture better the semantic of the words. Implementation considerations are briefly presented to avoid scalability issues.

While word embeddings capture information on the semantics which is of great interest, some tasks need to encode semantic information at a higher level, namely at phrase, sentence or document level. The following section is about those higher levels.

1.2 Vector Representation of Documents

A document is represented as a string of characters, but in order to apply common machine learning algorithm, it is more convenient to transform the strings into vectors.

1.2.1 Vector Space Model

To represent a document by a vector, we commonly split each document into a list of tokens and build a vocabulary of tokens over all documents of the collection. We can then represent a collection of documents as a two dimensional array: each column represents a word (unique token) and each row represents a document. The value of the cell corresponding to a document d and a word w is the given by the number of occurrences of w in d.

Alternatively, the value can be 0 or 1 if the word is absent or present respectively in the document. The array of word frequencies can also be weighted according to several weighting schemes. The most famous weighting schemes are known as the TF-IDF family: Each entry of the weighted matrix is usually composed of two components (Salton and Yang, 1973):

$$f_{ij} = TF_{ij} \times DF_i$$

where:

- TF is a function of the frequency of term i in document j, so that words which occur less are less discriminant and have a smaller weight.
- DF is a function of the number of documents term i occurs in, so that words which occur in many documents have a smaller weight and thus are less discriminant. The most common version is the *inverse document frequency*: $IDF = \log \frac{D}{df_i}$ with D the number of documents and df_i the number of documents term i occurs in.

1.2.2 Latent Semantic Analysis

Latent Semantic Analysis (LSA), named also Latent Semantic Indexing in the context of information retrieval (Deerwester et al., 1990), is designed to provide a similarity measure between documents.

A (document × word) matrix is built with the values given by the number of occurrences of the words in the documents. Then a dimensionality reduction of the columns is done through Truncated Singular Value Decomposition. The similarity measure is given by the cosine of the vectors representing the documents (i.e. the rows of the reduced matrix). Řehůřek (2011) presents detailed explanations on considerations when implementing Singular Value Decomposition: batch implementation, small batches, parallelization, online training with multiple or single pass, approximations. Řehůřek and Sojka (2010) provide a scalable online implementation.

LSA is a dimensionality reduction of a (document × word) matrix and is to contrast with the following method that uses a different approach.

1.2.3 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) (Blei et al., 2003) is a generative topic model that provides vector representations of documents. It assumes that for each word we want to generate: a topic is selected from a multinomial distribution and a word is generated from that topic which is a distribution over words.

Other probability topic models exist but are not discussed here since LDA is the most cited one and constitutes a comparison element for most topic models.

1.3 Combined Word and Document Levels Textual Embeddings

While the above methods represent documents as semantic vectors, they do not exploit local co-occurrences. This section is about textual embeddings that leverage the local co-occurrences for documents, including phrases, sentences and paragraphs. They combine word level and document level representations.

1.3.1 Methods Leveraging Word Embeddings

The previously cited methods do not rely on external knowledge. Here, we discuss the transfer of external knowledge through the use of word embeddings.

1.3.1.1 Aggregation of Word Embeddings

The most evident method to create a vector representation of a document is to aggregate the vector representations of the words it contains. The aggregation can be done with several aggregation functions.

An example of aggregation function is the element-wise sum of the word vectors, giving a document vector with the same dimensionality as the word vectors. This method is compared to Latent Semantic Analysis for a clustering task in 6.1.3.2.

Boom et al. (2015) and, Kenter and de Rijke (2015) use word embeddings and generate several more complicated new features for the documents. It is noteworthy that they use supervised learning to evaluate their methods so they benefit from new generated features.

1.3.1.2 Paragraph Vector

Paragraph vector (Le and Mikolov, 2014) is an extension of Word2vec where additional input fields are used to represent the document id as a one hot vector (Figure 1.5). They allow to have a vector representation of the document when the document id is projected on the hidden layer.

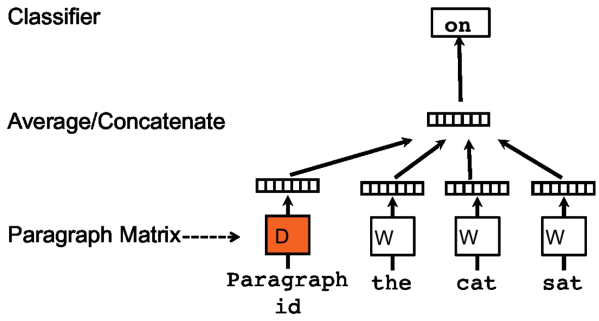


Figure 1.5: Paragraph vector architecture. [6]

1.3.2 ELMo

ELMo (Peters et al., 2018) assigns an embedding for a word depending on the context: each token is assigned a representation that is a function of the entire input sentence. It is a bi-directional LSTM language model which builds the embedding of a token from both the previous tokens and the next ones: A forward language model models the probability of a token given the previous ones. A backward language model is similar to a forward language model except it runs over the sequence in reverse, predicting the previous token given the future context. The bi-directional language model combines both a forward and a backward language models. The bi-directional language model used in ELMo shares some weights between the two directions. To compute the contextualized embedding, ELMo groups together the hidden states by learning a linear function of the internal network states. This weighted average is trained on a specific downstream task. ELMo is thus trained in a semi-supervised setting which requires task-specific architectures where pre-trained representations are included as additional features.

1.3.3 GPT

GPT (Radford, 2018) is a semi-supervised approach for language understanding tasks using a combination of unsupervised pre-training and supervised fine-tuning: The first

[6] Source: (Le and Mikolov, 2014)

stage learns a high-capacity language model on a large corpus of text, and the second stage adapts the model to a discriminative task with labeled data. It aims to capture higher-level semantics than word-level information.

The language model of the first phase is a multi-layer transformer decoder: it is composed by a multi-headed self-attention operation over the input context tokens, followed by position-wise feedforward layers and a softmax layer to produce an output distribution over target tokens. The use of the transformers architecture leverages long-term dependencies in the text.

To train on the discriminative task, the output of the last feedforward layer is fed into a linear layer with a softmax to predict the target label. All pre-trained parameters are updated with this new task specific objective. This architecture works for instance for text classification. For some tasks, since the model of the first phase is trained on contiguous sequences of text, some modifications are proposed by the authors to apply to the textual entailment, similarity, question answering and commonsense reasoning tasks.

GPT-2 (Radford et al., 2019) is a model which shares almost the same architecture than the OpenAI GPT model with a few modifications such as the layers normalizations and a modified initialization. Also the dataset on which it is trained is broader. The authors showed that when trained on this bigger dataset, the model is able to have good results in a zero-shot setting for several NLP tasks: language modeling, reading comprehension, summarization, translation and question answering. It is to note however that all these tasks rely on predicting the following word of a sequence which is at the heart of the training objective.

1.3.4 BERT

BERT (Devlin et al., 2018) improves the fine-tuning based approach (used for instance in GPT) by using a *masked language model*. The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original word based on the context (words on the left and on the right of the masked token).

A second task on which BERT is pre-trained is the next sentence prediction. For training, two sentences A and B are fed as input, and half of the time, B is the actual next sentence that follows A, and half of the time, B is a random sentence from the corpus. To feed the inputs with two sentences, they are packed together into a single sequence. The two sentences are differentiated in two ways: they are separated by a special token, and a learned embedding is added to every token, which indicates if the token belongs to the first or the second sentence of the pair. Thus, the input embeddings are the sum of three token embeddings: an embedding for the word, an embedding which indicates the sentence (called *segment embedding*) and an

embedding for the position of the token in the sequence. This task is trained to output a probability of sentence B to follow sentence A.

The two previously presented tasks constitute the pre-training phase of BERT. The second phase is fine-tuning. Several supervised tasks are used to fine-tune the model. Depending on the task, the correct inputs and outputs are chosen either from the token level part of BERT (the masked language model) or the sentence pair part (next sentence prediction model). All parameters are fine-tuned end-to-end.

XLNet (Yang et al., 2019) integrates ideas from Tranformer-XL (Dai et al., 2019) into the pre-training phase of BERT to improve its ability to leverage longer dependencies. RoBERTa (Liu et al., 2019) improves the pre-training of BERT by carefully selecting the hyper-parameters values. DistilBERT (Sanh et al., 2019) leverages knowledge distillation to produce a more compact model which is of great value to run on mobile devices since it requires less storage space and it runs faster at inference time. Knowledge distillation is a compression technique in which the compact model is trained to reproduce the behaviour of the larger model, BERT in this case.

1.4 Conclusion

In this chapter, we presented an overview of textual representations. Some of them have become mainstream, entering full text search engines of databases like the bag of words approach of the Vector Space Model used in MongoDB or Elasticsearch. Some of the most modern ones like BERT are announced to make breaking changes in the Search Engine Optimization (SEO) world with its use inside Google search engine.

These representations are used as input to other algorithms for task specific use cases. One of such task is co-clustering, which is the topic of the following chapter.

Chapter 2

Co-clustering in Text Mining

Part of this chapter is an adaptation of the paper: François Role, Stanislas Morbieu, and Mohamed Nadif. "CoClust: A Python Package for Co-clustering". Journal of Statistical Software *vol.88, no.7, 2019.*

This chapter is about co-clustering, an ensemble of methods which aim to group together similar objects into clusters and also similar features into feature clusters. For text-mining, they are very handy since they condense information and provide interpretability. We first give some background on text co-clustering, and then focus on the two text co-clustering that are used in the following chapters, namely Coclust-Mod Ailem et al. (2016) referred to as `Mod` and CoclustInfo Govaert and Nadif (2013) referred to as `Info`, both implemented in the Coclust library (Role et al., 2019).

2.1 Co-clustering

In the era of data science, clustering various kinds of objects (documents, genes, customers) has become a key activity and many high quality packaged implementations are provided for this purpose by many popular packages such as the base package **stats** for R (R Core Team, 2018), **skmeans** (Hornik et al., 2012a), **kernlab** (Karatzoglou et al., 2004), **NbClust** (Charrad et al., 2014), **CLUTO** (Karypis, 2003), **scikit-learn** (Pedregosa et al., 2011), **SciPy** (including the **scipy.cluster** module) (Jones et al., 2001–), **nltk** (with the **nltk.cluster** module) (Bird et al., 2009), **Weka** (Hall et al., 2009), etc. A natural extension of standard cluster analysis is co-clustering where objects and features are simultaneously grouped into meaningful blocks called co-clusters or biclusters, thus making large datasets easier to handle and interpret. In fact, since the seminal work of Hartigan (1972), co-clustering has found applications in many areas such as bio-informatics (Cheng and Church, 2000; Madeira and Oliveira, 2004; Van Mechelen et al., 2004; Tanay et al., 2005; Cho and Dhillon, 2008; Gupta and Aggarwal, 2010; Hanczar and Nadif, 2010, 2011, 2012, 2013), web mining (Xu et al., 2010;

Charrad et al., 2009; George and Merugu, 2005; Deodhar and Ghosh, 2010) and text mining (Dhillon, 2001; Dhillon et al., 2003b) and various co-clustering algorithms have been proposed over the years (recent surveys can be found in Freitas et al. 2012; Eren et al. 2013; Henriques et al. 2015).

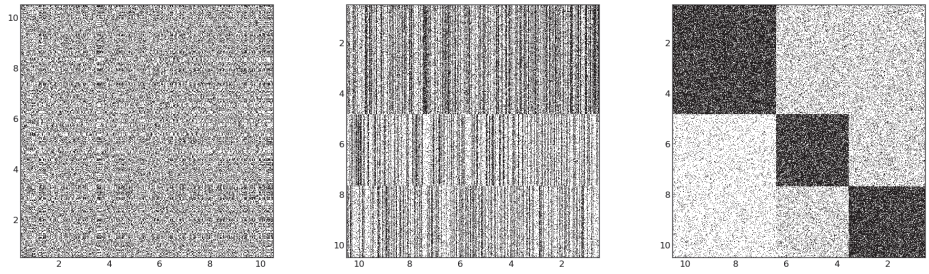


Figure 2.1: Left: original data. Middle: data reorganized according to row clusters. Right: data reorganized according to row and column clusters.

A large number of implementations of co-clustering algorithms[1] have been developed for gene expression data, such as **biclust** (Kaiser and Leisch, 2008), **BicAT** (Barkow et al., 2006) and **bibench** (Eren et al., 2013). In addition, there also exist algorithms designed to efficiently handle co-occurrence matrices such, for example, as document-term matrices used in text mining applications. The already mentioned CoClust package provides implementations of such algorithms and these implementations have been used in our experiments.

2.1.1 Methods for Co-clustering

Depending on the method used, algorithms for co-clustering co-occurrence matrices can broadly be divided into several categories:

Spectral methods: Spectral co-clustering methods treat the input data matrix as a bipartite graph between documents and words, and approximate the normalized cut of this graph using a real relaxation. Currently **scikit-learn** supports two spectral co-clustering algorithms: (1) the well-known "spectral co-clustering" (Dhillon, 2001) and (2) the "spectral biclustering" (Kluger et al., 2003) which is also available in the **biclust** R package.

Model-based methods: With respect to probabilistic co-clustering methods, two model-based co-clustering methods are implemented in the **blockcluster** (Singh Bhatia et al., 2017) and **blockmodels** (Leger, 2016) R packages. The first relies

[1] Also known as biclustering.

on the latent block models (LBM), especially Gaussian, Bernoulli and Poisson LBMs. The derived algorithms are of type expectation-maximization; for details see for instance Govaert and Nadif (2003, 2005b, 2006, 2008); Nadif and Govaert (2010). The second relies on the stochastic block model and the latent block model without or with covariates. Both models have been extended to valued networks with optional covariates on the edges.

Matrix factorization based methods: Matrix factorization based methods are also used in the clustering and co-clustering fields. However while packages exist for document clustering based on non-negative matrix factorization (e.g., the **NMF** R package, Gaujoux and Seoighe 2010, which includes different NMF methods) leading to clustering (see for instance Ding et al. 2006; Ding and Li 2007), there is unfortunately no package on non negative matrix trifactorization factorization for co-clustering.

Information-theoretic based methods: Information-theoretic based methods are used to co-cluster two-way contingency tables. In this approach, a joint probability distribution is first derived from the two-way contingency matrix. The loss function to minimize is then the loss in mutual information between this joint probability distribution and a distribution defined on a reduced contingency table obtained by collapsing the rows and the columns according to the partitions yielded by the co-clustering program. Notable algorithms in this area include those in Dhillon et al. (2003b); Govaert and Nadif (2013).

Modularity-based methods: The use of bipartite graph-modularity as a criterion to co-cluster matrices has been pioneered by Labiod and Nadif (2011) and since further investigated in Ailem et al. (2015, 2016). This method allows to co-cluster binary or contingency matrices by maximizing an adapted version of the modularity measure traditionally used for networks.

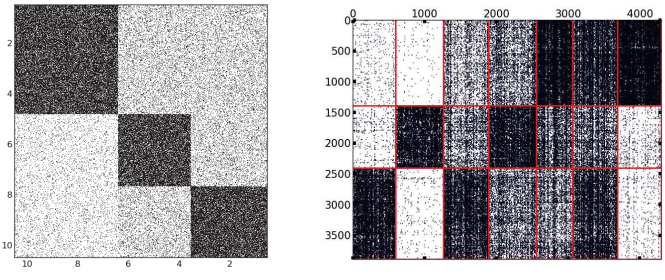


Figure 2.2: Left: diagonal co-clustering. Right: non-diagonal co-clustering.

The CoClust package provides implementations for several of these methods, including the two algorithms that we use in our experiments, namely CoclustMod and CoclustInfo. Before presenting these algorithms in more detail, we clarify the notations used in the following sections.

2.1.2 General Notations

We consider the partition of the sets I of n objects and the set J of d attributes into g non overlapping clusters, where g may be greater or equal to 2. Let us define a $n \times g$ indicator matrix $\mathbf{z} = (z_{ik})$ and a $d \times g$ indicator matrix $\mathbf{w} = (w_{jk})$. The kth row cluster is defined by the set of rows i such that $z_{ik} = 1$. In the same manner, the kth column cluster is defined by the set of rows j such that $w_{jk} = 1$. X is the matrix used as input to all the methods described in this chapter; X can be of any kind provided it is a matrix with non-negative entries (e.g., a graph adjacency matrix, or a document-term matrix, depending on the application domain). $X^{\mathbf{z}}$ and $X^{\mathbf{w}}$ are the matrix X compressed according to the row clusters and the column clusters respectively: $X^{\mathbf{z}} = X^T Z$ and $X^{\mathbf{w}} = XW$.

2.2 CoclustMod: a Modularity-based, Block-diagonal Co-clustering Algorithm

CoClustMod is a modularity-based algorithm that seeks an optimal block-diagonal clustering, meaning that objects and features have the same number of clusters and that, after proper permutation of the rows and columns, the algorithm produces as result a block-diagonal matrix (see Figure 2.1). In the context of document-term matrices, this co-clustering model has the advantage of directly producing interpretable descriptions of the resulting document clusters.

A notable block-diagonal co-clustering algorithm is the bipartite spectral graph partitioning algorithm described in Dhillon (2001). Inspired by previous work on spectral graph clustering, this algorithm finds the optimal minimum cut partitions in a bipartite document-term graph by computing the second left and right singular vector of the normalized document-term matrix, thus using a real relaxation of the discrete optimization problem. In contrast, the CoclustMod algorithm tries to maximize a measure of the concentration of edges within co-clusters compared with the random distribution of edges between all nodes regardless of the co-clusters. This criterion is an adaptation to the bipartite case of the standard "graph modularity". Before describing CoclustMod in more detail, it is therefore useful to review this notion of "bipartite graph modularity".

2.2.1 Bipartite Graph Modularity (BGM)

In this section we first review the standard graph modularity measure, and show how to adapt it so that it can be used in the co-clustering context.

Modularity is a quality criterion often used for detecting communities in graphs, which has received considerable attention in several disciplines since the seminal work by Newman and Girvan (2004). Intuitively, modularity compares the number of edges inside a cluster of nodes with the expected number if the edges in the graph were placed at random.[2]

Given the graph $G = (V, E)$, let X be a binary, symmetric adjacency matrix with (i, i') as entry; and $x_{ii'} = 1$ if there is an edge between the nodes i and i'. If there is no edge between nodes i and i', $x_{ii'}$ is equal to zero. Finding a partition of the set of nodes V into homogeneous subsets leads to the resolution of the following integer linear program: $\max_{\mathbf{c}} Q(X, \mathbf{c})$ where $Q(X, \mathbf{c})$ is the modularity measure:

$$Q(X, \mathbf{c}) = \frac{1}{2|E|} \sum_{i,i'=1}^{n} (x_{ii'} - \frac{x_{i.}x_{i'.}}{2|E|})c_{ii'}. \tag{2.1}$$

In this formula, $\mathbf{c}$ is a binary matrix defined by $c_{ii'} = \sum_{k=1}^{g} z_{ik}z_{i'k}$, meaning that $c_{ii'}$ is 1 when nodes i and i' are in the same group and 0 otherwise. In addition, $|E|$ is the number of edges and $x_{i.} = \sum_{i'} x_{ii'}$ is the degree of i.

Let now $\boldsymbol{\delta} = (\delta_{ii'})$ be the $(n \times n)$ data matrix defined by $\forall i, i'$, $\delta_{ii'} = \frac{x_{i.}x_{i'.}}{2|E|}$. Expression 2.1 then becomes $Q(X, \mathbf{c}) = \frac{1}{2|E|}\text{Trace}[(X - \boldsymbol{\delta})\mathbf{c}]$. In summary, we seek a binary matrix $\mathbf{c}$ which is defined as $\mathbf{z}\mathbf{z}^\top$ and models a partition in a relational space, thus having the properties of an equivalence relation:

$$\begin{cases} c_{ii} = 1, \ \forall i & \text{reflexivity} \\ c_{ii'} - c_{i'i} = 0, \forall (i, i') & \text{symmetry} \\ c_{ii'} + c_{i'i''} - c_{ii''} \leq 1, \forall (i, i', i'') & \text{transitivity} \\ x_{ii'} \in \{0, 1\}, \forall (i, i') & \text{binarity} \end{cases}$$

In a bipartite context, the basic idea is to model the simultaneous row and column partitions using a relation $\mathbf{c}$ defined on $I \times J$. Noting that $\mathbf{c} = \mathbf{z}\mathbf{w}^\top$ and the general term can be expressed as follows: $c_{ij} = 1$ if object i is in the same block as attribute j and $c_{ij} = 0$ otherwise. Then $c_{ij} = \sum_{k=1}^{g} z_{ik}w_{jk}$. Now, given a rectangular matrix X defined on $I \times J$, modularity can be reformulated as follows in the co-clustering context:

$$Q(X, \mathbf{c}) = \frac{1}{x_{..}} \sum_{i=1}^{n} \sum_{j=1}^{d} \sum_{k=1}^{g} (x_{ij} - \frac{x_{i.}x_{.j}}{x_{..}})z_{ik}w_{jk}, \tag{2.2}$$

[2]The standard null model used in the literature also assumes that the nodes keep the degree they have in the original network.

Algorithm 1 CoclustMod.

Input: binary or contingency data X, number of clusters g.

Output: partition matrices $\mathbf{z}$ and $\mathbf{w}$.

1. Initialization of $\mathbf{w}$.

repeat

2. Compute $X^{\mathbf{w}}$.

3. Compute $\mathbf{z}$ maximizing $Q(X^{\mathbf{w}}, \mathbf{z})$ by $z_{ik} = 1$ if $k = \text{argmax}_{1 \leq \ell \leq g} \left(x^{\mathbf{w}}_{i\ell} - \frac{x_{i.} x^{\mathbf{w}}_{.\ell}}{x_{..}} \right)$ and $z_{ik} = 0$ otherwise; $\forall i = 1, \ldots, n$.

4. Compute $X^{\mathbf{z}}$.

5. Compute $\mathbf{w}$ maximizing $Q(X^{\mathbf{z}}, \mathbf{w})$ by $w_{jk} = 1$ if $k = \text{argmax}_{1 \leq \ell \leq g} \left(x^{\mathbf{z}}_{\ell j} - \frac{x^{\mathbf{z}}_{\ell .} x_{.j}}{x_{..}} \right)$ and $w_{jk} = 0$; $\forall j = 1, \ldots, d$.

6. Compute $Q(X, \mathbf{z}\mathbf{w}^\top)$.

until *no change* of $Q(X, \mathbf{z}\mathbf{w}^\top)$.

where $x_{..} = \sum_{i,j} x_{ij} = |E|$ is the total weight of edges and $x_{i.} = \sum_j x_{ij}$ (the degree of i in the binary case and the sum of the weights in the contingency and continuous cases) and $x_{.j} = \sum_i x_{ij}$ (the degree of j in the binary case and the sum of the weights in the contingency and continuous cases). This modularity measure can also take the following form:

$$Q(X, \mathbf{c}) = \frac{1}{x_{..}} \text{Trace}[(X - \boldsymbol{\delta})^\top \mathbf{z}\mathbf{w}^\top] = Q(X, \mathbf{z}\mathbf{w}^\top). \tag{2.3}$$

Using the fact that $Q(X, \mathbf{c})$ can be rewritten as $Q(X^{\mathbf{w}}, \mathbf{z})$ or $Q(X^{\mathbf{z}}, \mathbf{w})$, CoclustMod maximizes the modularity by alternatively maximizing $Q(X^{\mathbf{w}}, \mathbf{z})$ and $Q(X^{\mathbf{z}}, \mathbf{w})$ as shown in Algorithm 1.

2.3 CoclustInfo: an Information-theoretic Co-clustering Algorithm

In our experiments, we also use, CoclustInfo, a text co-clustering algorithm provided by the CoClust package. CoclustInfo takes an information-theoretic approach and uses mutual information to define its criterion (Govaert and Nadif, 2013, Chapter 4). In contrats to CoclustMod, this algorithm does not seek to discover a block-diagonal structure: the requested number of row clusters can be different from the requested number of column clusters. A representative example of the kind of matrix obtained when using CoclustInfo is shown in Figure 2.3.

We now describe the working of the algorithm. Given two variables I and J taking values in the sets $I = \{1, \ldots, i, \ldots, n\}$ of rows and $J = \{1, \ldots, j, \ldots, d\}$ of columns respectively.

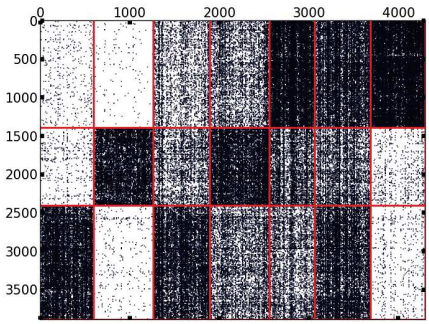


Figure 2.3: Typical matrix obtained when using CoclustInfo to co-cluster a dataset. This matrix i s to be compared to the kind matrices obtained when using lock-diagonal algorithms .

Table 2.1: Example of contingency table and associated joint distribution.

	1	2	3	4	5	
1	5	4	6	1	0	16
2	6	5	4	0	1	16
3	1	0	1	7	5	14
4	1	1	0	6	5	13
5	4	5	3	4	5	21
6	5	4	4	3	4	20
	22	19	18	21	20	100

	1	2	3	4	5	
1	0.05	0.04	0.06	0.01	0.00	0.16
2	0.06	0.05	0.04	0.00	0.01	0.16
3	0.01	0.00	0.01	0.07	0.05	0.14
4	0.01	0.01	0.00	0.06	0.05	0.13
5	0.04	0.05	0.03	0.04	0.05	0.21
6	0.05	0.04	0.04	0.03	0.04	0.20
	0.22	0.19	0.18	0.21	0.20	1.00

Let $\mathbf{X}$ be a contingency table (a *document*$\times$*term* matrix in our case), the associated joint distribution is defined by $P_{IJ} = (p_{ij}) = (\frac{x_{ij}}{N})$ where $N = \sum_{i=1}^{n}\sum_{j=1}^{d} x_{ij}$. We consider a partition $\mathbf{z}$ of the set of rows I into g clusters $A_1, \ldots, A_g$ and a partition $\mathbf{w}$ of the set of columns J into m clusters $B_1, \ldots, B_m$. The partition $\mathbf{z}$ will be represented by the vector of labels $(z_1, \ldots, z_i, \ldots, z_n)$ where $z_i \in \{1, \ldots, g\}$, or by the classification matrix $\{z_{ik}; i = 1, \ldots, n; k = 1, \ldots, g\}$ where $z_{ik} = 1$ if i belongs to cluster k and 0 otherwise. A similar notation will be used for the partition $\mathbf{w}$, which will be represented by the vector $(w_1, \ldots, w_j, \ldots, w_d)$ where $w_j \in \{1, \ldots, m\}$ or the classification matrix $\{w_{j\ell}; j = 1, \ldots, d; \ell = 1, \ldots, m\}$ of size $d \times m$. Note that $z_{ik}w_{j\ell} = 1$ if the couple (i, j) belongs to the co-cluster (k, ℓ) defined by the cartesian product $A_k \times B_\ell$.

An aggregated $g \times m$ two-way contingency table $\mathbf{y^{zw}} := \{y_{k\ell}^{\mathbf{zw}}; k = 1, \ldots, g; \ell = 1, \ldots, m\}$ for two categorical random variables with values from the sets $K = \{1, \ldots, g\}$ and $L = \{1, \ldots, m\}$ can be obtained from the initial table by computing the sum of the rows and columns according to $\mathbf{z}$ and $\mathbf{w}$ by $y_{k\ell}^{\mathbf{zw}} = \sum_{i,j}[z_{ik}w_{j\ell}]x_{ij}\ \forall k \in K$ and $\forall \ell \in L$. The first distribution that can be associated to $\mathbf{z}$ and $\mathbf{w}$ is $P_{KL}^{\mathbf{zw}} = (p_{k\ell}^{\mathbf{zw}})$ defined on

Algorithm 2 CoclustInfo.

Input: X, g, m.
Initialization: $\mathbf{z}$, $\mathbf{w}$, $\gamma_{k\ell} = \frac{p_{k\ell}^{\mathbf{zw}}}{p_{k.}^{\mathbf{z}} p_{.\ell}^{\mathbf{w}}}$.
repeat
  repeat
    Step 1. $z_{ik} = 1$ if $k = \text{argmax}_{1 \leq k' \leq g} \sum_\ell p_{i\ell}^{\mathbf{w}} \log \gamma_{k'\ell}$ and $z_{ik} = 0$ otherwise $\forall i$.
    Step 2. $\gamma_{k\ell} = \frac{p_{k\ell}^{\mathbf{zw}}}{p_{k.}^{\mathbf{z}} p_{.\ell}^{\mathbf{w}}}$.
  until convergence
  repeat
    Step 3. $w_{j\ell} = 1$ if $\ell = \text{argmax}_{1 \leq \ell' \leq m} \sum_k p_{kj}^{\mathbf{z}} \log \gamma_{k\ell'}$ and $w_{j\ell} = 0$ otherwise $\forall j$.
    Step 4. $\gamma_{k\ell} = \frac{p_{k\ell}^{\mathbf{zw}}}{p_{k.}^{\mathbf{z}} p_{.\ell}^{\mathbf{w}}}$.
  until convergence
until convergence
return $\mathbf{z}$ and $\mathbf{w}$.

$K \times L$ by

$$p_{k\ell}^{\mathbf{zw}} = \frac{y_{k\ell}^{\mathbf{zw}}}{N} = \sum_{i,j} [z_{ik} w_{j\ell}] p_{ij} \quad \forall (k,\ell) \in K \times L.$$

It will be remarked that the row margins $\sum_\ell p_{k\ell}^{\mathbf{zw}}$ of this new distribution are respectively equal to $\sum_i z_{ik} p_{i.}$ and consequently do not depend on the partition $\mathbf{w}$ (Govaert and Nadif, 2018). They will be denoted $p_{k.}^{\mathbf{z}}$. Similarly, the column margins $\sum_k p_{k\ell}^{\mathbf{zw}}$ are equal to $\sum_j w_{j\ell} p_{.j}$ and will be denoted $p_{.\ell}^{\mathbf{w}}$. Denoting by

$$q_{ij}^{\mathbf{zw}} = p_{i.} p_{.j} \sum_{k,\ell} z_{ik} w_{j\ell} \frac{p_{k\ell}^{\mathbf{zw}}}{p_{k.}^{\mathbf{z}} p_{.\ell}^{\mathbf{w}}} \qquad \forall (i,j) \in I \times J$$

and using the properties $\sum_{i,j} q_{ij}^{\mathbf{zw}} = 1$, $q_{i.}^{\mathbf{z}} = p_{i.}$ and $q_{.j}^{\mathbf{w}} = p_{.j}$ $\forall i, j$, (Dhillon et al., 2003a; Govaert and Nadif, 2018) we can associate with the partitions $\mathbf{z}$ and $\mathbf{w}$ a second distribution $Q_{IJ}^{\mathbf{zw}} := \{q_{ij}^{\mathbf{zw}}; i \in I; j \in J\}$ defined on $I \times J$ with the same margins as the initial distribution P_{IJ}. Thereby, the co-clustering problem can be viewed as an approximation of the distribution P_{IJ} by a co-clustering distribution termed $Q_{IJ}^{\mathbf{zw}}$, by minimizing the difference between the measure of information between the original distribution and the aggregated distribution

$$\mathcal{I}(P_{IJ}) - \mathcal{I}(Q_{IJ}^{\mathbf{zw}}) = \text{KL}(P_{IJ} || Q_{IJ}^{\mathbf{zw}}).$$

These different steps are summarized in the pseudo-code shown in Algorithm 2.

2.4 Visualization and Interpretability

Co-clustering, at its heart, compress the information into co-clusters. One can therefore describe a cluster of documents by clusters of terms. Visualization enables to understand the structure of a dataset in a glimpse.

2.4.1 Reorganized and Summary Matrices

As presented previously in Figure 2.1, in the case of diagonal co-clustering, a cluster of documents is described by a cluster of terms. In the case of non-diagonal co-clustering, observing the patterns in the reorganized matrix (Figure 2.4) leads to a characterization of document clusters by potentially several clusters of terms.

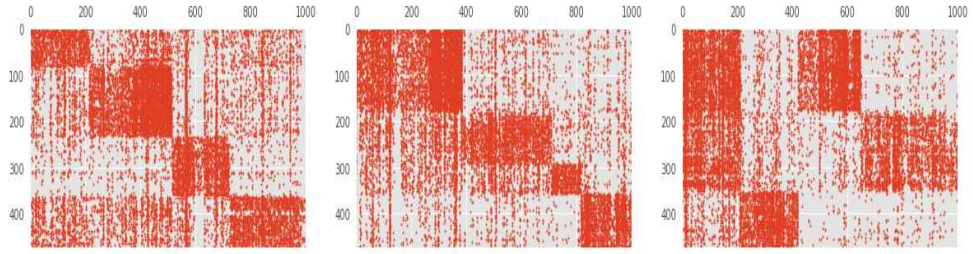


Figure 2.4: Three reorganized matrices for the CSTR dataset obtained with three different algorithms.

Hovering a particular cell of the reorganized matrix in the visualization presented in (Morbieu et al., 2018) shows the term corresponding to the column, so hovering a high density zone give hints to the content of the documents in the corresponding cluster.

The CoclustInfo algorithm provides also a summary matrix given by the γ_{kl} values (Figure 2.5). This summary matrix allows to view in a glimpse the associations between clusters of terms and clusters of documents (a high value indicates a high association).

2.4.2 Representative Terms

Since a set of few words is easier to interpret than a set of documents, taking the associations between clusters of documents and clusters of words given by the summary matrix, and combine them with a summary of the words in the co-cluster enables simple analysis of the clusters of documents. Each co-cluster can be represented by its top terms, *i.e.* the words appearing the most in the co-cluster (Figure 2.6).

Another visualization of the terms in a co-cluster is illustrated by Figure 2.7. The n most frequent terms in a given term cluster are extracted along with the k most

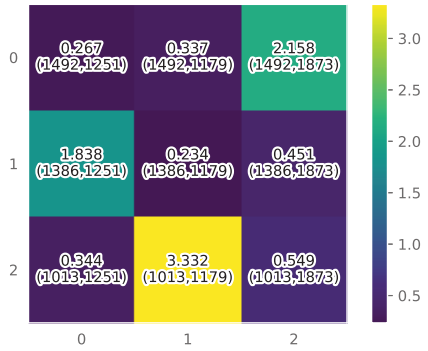


Figure 2.5: CoclustInfo – heatmap showing the final γ_{kl} values obtained for each row cluster k and each column cluster l. This may help to spot the interesting pairs of row and column clusters.

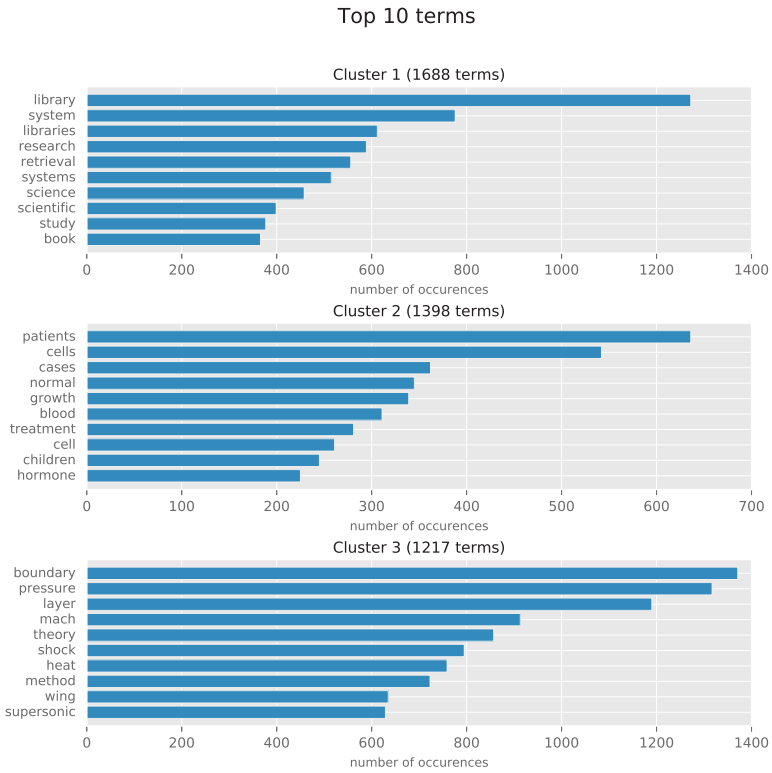


Figure 2.6: CoclustMod – displaying the top terms of each cluster.

similar (in terms of cosine similarity) neighbors of each of these most frequent terms. The shape of the displayed graph using a force layout provides insights of the content:

a dense graph is the result of a thematically focused cluster.

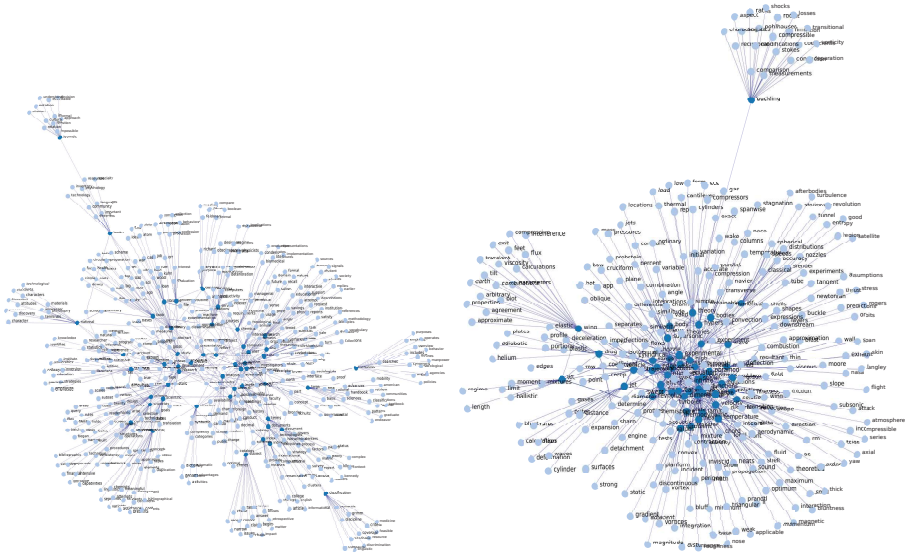


Figure 2.7: CoclustMod – graph representations of two term clusters. We can visually detect that the cluster on the right is dense, and more thematically focused (aerodynamics) than the cluster on the left which is about more general notions (information, knowledge and science in general).

Table 2.2: NMI values.

Dataset	CoclustInfo	CoclustMod	CoclustSpec-Mod	Spectral-Biclustering	Spectral-Coclustering
CLASSIC3	0.934±0.001	0.918±0.003	0.914±0.000	0.729±0.059	0.912±0.001
CSTR	0.684±0.024	0.620±0.043	0.781±0.000	0.444±0.038	0.701±0.012
WEBACE	0.612±0.010	0.595±0.013	0.568±0.010	0.442±0.003	0.524±0.005
CLASSIC4	0.632±0.045	0.712±0.027	0.508±0.020	0.383±0.030	0.530±0.094
REVIEWS	0.593±0.023	0.530±0.032	0.341±0.019	0.291±0.001	0.424±0.003
SPORTS	0.564±0.033	0.547±0.026	0.544±0.010	0.395±0.047	0.435±0.009
RCV1	0.495±0.021	0.469±0.034	0.012±0.003	0.304±0.025	0.012±0.002
NG20	0.565±0.010	0.508±0.012	0.474±0.012	0.075±0.022	0.389±0.006

2.5 Conclusion

Co-clustering is an important technique in the era of so-called *big data* since it allows to compress large, high dimensional matrices.

The two co-clustering algorithms presented in this chapter achieve state-of-the-art performance on many standard text datasets of various balance and sparsity, as shown by experimental results (see Tables 2.2, 2.3 and 2.4).

Table 2.3: Accuracy values.

Dataset	CoclustInfo	CoclustMod	CoclustSpec-Mod	Spectral-Biclustering	Spectral-Coclustering
CLASSIC3	0.987±0.000	0.983±0.001	0.979±0.000	0.884±0.041	0.979±0.000
CSTR	0.814±0.044	0.803±0.044	0.897±0.000	0.560±0.019	0.823±0.007
WEBACE	0.510±0.021	0.583±0.023	0.501±0.020	0.305±0.007	0.389±0.010
CLASSIC4	0.774±0.075	0.888±0.018	0.596±0.011	0.628±0.032	0.629±0.078
REVIEWS	0.716±0.025	0.686±0.035	0.477±0.006	0.462±0.000	0.504±0.001
SPORTS	0.573±0.046	0.674±0.027	0.638±0.028	0.476±0.021	0.550±0.013
RCV1	0.715±0.024	0.710±0.042	0.301±0.000	0.515±0.023	0.301±0.000
NG20	0.492±0.023	0.394±0.021	0.283±0.020	0.078±0.005	0.210±0.005

Table 2.4: ARI values.

Dataset	CoclustInfo	CoclustMod	CoclustSpec-Mod	Spectral-Biclustering	Spectral-Coclustering
CLASSIC3	0.961±0.001	0.948±0.002	0.941±0.000	0.713±0.082	0.940±0.001
CSTR	0.686±0.055	0.642±0.059	0.809±0.000	0.299±0.035	0.721±0.003
WEBACE	0.434±0.039	0.550±0.031	0.334±0.037	0.213±0.013	0.344±0.011
CLASSIC4	0.529±0.101	0.703±0.040	0.299±0.055	0.296±0.044	0.309±0.142
REVIEWS	0.618±0.042	0.529±0.053	0.184±0.022	0.156±0.000	0.320±0.005
SPORTS	0.460±0.053	0.516±0.029	0.390±0.012	0.228±0.033	0.317±0.027
RCV1	0.501±0.029	0.484±0.043	-0.000±0.000	0.238±0.028	-0.000±0.000
NG20	0.380±0.017	0.285±0.019	0.196±0.019	0.008±0.004	0.125±0.008

However, a limitation of these algorithms is that they exclusively rely on the Vector Space Model to represent the documents, *i.e.* documents are described by the number of occurrences of terms in the documents. They therefore do not leverage local co-occurrences of terms. The following chapter aims to fill this gap by exploiting local co-occurrences of terms in sliding windows to better exploit the semantics of texts.

Chapter 3

Transfer learning for co-clustering

Contents

Text clustering aims to group documents into clusters such that similar content lay in the same cluster. Most text clustering algorithms use the standard Vector Space Model (VSM) (Salton et al., 1975) for document representation: a document corpus is represented as a matrix where each row vector represents a document, each column a term, and each cell of the matrix is the number of occurrences of the term in the document. However, the recent years have seen the emergence of new, neural-based ways of representing words and documents that go beyond the above-mentioned VSM-based representations. In particular, neural word embeddings are today widely used as input to text classifiers in the supervised machine learning setting. Representations of larger units of text (sentences, documents) can also be derived using techniques as simple as averaging pretrained word embeddings. While such a simple approach can perform surprisingly well (Ionescu and Butnaru, 2019), better results can even be obtained by using neural networks to directly learn sentence or document level representations using a variety of approaches ranging from unsupervised learning to supervised learning in combination with downstream tasks (Kiros et al., 2015; Le and Mikolov, 2014; Socher et al., 2013; Conneau et al., 2017; Hill et al., 2016; Lin et al., 2017). As already said, these new neural-based representations of text units have been put to good use in the supervised learning field where text embeddings are today routinely used as ini-

tial input to neural text classifiers, leading to significant performance improvements. The question which then naturally arises is what performance gains could be attained if, similarly to what happens in supervised text mining, such neural-based document models were to be used as input to unsupervised tasks such as text clustering or even text co-clustering?

3.1 Motivation

There are many problems to solve when using pre-trained embeddings in the unsupervised learning field.

3.1.1 Problems to Tackle

First, as with many other neural techniques, the training of word or document embeddings is plagued with the problem of hyperparameters setting. Even in the supervised setting where using pre-trained embeddings has been shown to improve results on classification tasks, using such embeddings requires attention since the many hyper-parameters which values greatly impact the quality of the results. This problem is even more acute in the unsupervised field since, in contrast to supervised tasks which can at least leverage cross-validation to set hyper-parameters, this is not possible for unsupervised tasks such as clustering or co-clustering. To deal with this problem, we propose to combine the use of pre-trained document embeddings with a form of consensus clustering; see for instance (Vega-Pons and Ruiz-Shulcloper, 2011), and show in the experimental section that this combination provides a significant improvement over state-of-the-art results on several well-known datasets.

The text co-clustering technique also poses specific problems when it comes to using text embeddings. The goal of co-clustering is to form meaningful co-clusters of words and documents. Therefore, we can't directly use the document embedding vectors since this would only allow to form co-clusters of document and embedding dimensions. In this case, we propose to take a kind of sequential transfer learning approach: First, document embeddings are learned that serve as input to a document clustering program. The so obtained document cluster indicators are then used to constrain a co-clustering algorithm.

Finally, another issue to settle for setting up our experiments is choosing among the many above-mentioned neural-based document-level representations. In this study we favor document models learned without labels, in order to remain in the unsupervised learning since we target downstream tasks (clustering and co-clustering) whose advantage is precisely not to rely on the availability of domain-specific labels. In our experiments, we specifically rely on the fully unsupervised Paragraph Vectors (Le and

Mikolov, 2014) algorithm, which learns representations that have been shown to perform significantly better on document similarity tasks compared to those obtained using traditional document modelling techniques such as LSA or LDA (Dai et al., 2015).

However this good performance level can only be achieved with some specific hyperparameter values. As other neural learning algorithms, Paragraph Vectors is highly sensitive to hyperparameter tuning. For instance when training Paragraph Vectors with several hyperparameter values and clustering a part of the 20 Newsgroups dataset (namely 5ng), we at first obtained Normalized Mutual Information (NMI) (Strehl and Ghosh, 2002) scores in ranging from 0.396 to 0.899. In the course of this study, we discovered that this drawback could be turned into an advantage. Actually, each combination of hyperparameters can be seen as casting a different light on the dataset at hand: combining the results provided by each combination can yield better results. In practical terms, we propose a consensus-based clustering technique which takes the several clustering obtained with the different values of hyperparameters and creates a new partitioning of the documents.

3.1.2 Related Work

One of the most notable examples of using use neural networks for clustering and co-clustering is Deep Embedded Clustering (DEC) (Xie et al., 2016), which simultaneously learns feature representations and cluster assignments. It is initialized with a stacked autoencoder, then the decoder is discarded so the encoder maps the data space to the feature space. K-means is run in the feature space to obtain initial centroids for the clustering part. The clustering part consists in two alternating steps:

- a soft assignment between the embedded points and the cluster centroids;
- the update of the deep mapping by minimizing the Kullback-Leibler divergence between the soft assignment and an auxiliary target distribution.

Another work in this field, is Deep Co-Clustering (DeepCC) (Xu et al., 2019), which uses neural networks for co-clustering. It utilizes a deep autoencoder for dimension reduction, and employs a variant of Gaussian Mixture Model to infer the cluster assignments. DeepCC jointly optimizes the parameters of the deep autoencoder and the mixture model in an end-to-end fashion on both the instance and the feature spaces.

The above-mentioned systems are complex deep models, with the associated learning cost and without real improvements. In fact, the experiments we carried out have clearly shown that DEC at best equals the best standard VSM-based clustering algorithms while DeepCC is even most of the case less effective than these traditional techniques. In contrast, using a far more shallow model such as Paragraph Vectors, we found that one can achieve results that clearly exceed state-of-the art results on a

variety of well-known datasets.

Some work apply consensus clustering in the context of document clustering. Gonzalez and Turmo (2008) make the distinction between the "major" and "minor" ensemble strategies for non-parametric document clustering: Whereas the "minor" strategy uses a small number of different clustering algorithms, the "major" strategy is based on the repetition of a randomly initialized single clustering algorithm. Our method is therefore more in line with the "major" strategy since a single clustering algorithm is used. But since the representation algorithm we choose is parametric, instead of varying the initialization randomly we initialize the representation algorithm with different parameters. Shinnou and Sasaki (2007) generate multiple clustering results by random initialization of Non-negative Matrix Factorization on the VSM representation weighted by Tf-Idf. Greene and Cunningham (2006) discuss prototype reduction to speed-up the consensus process : Prototype reduction produces a minimal set of objects or prototypes to represent the data, while ensuring that a clustering algorithm applied to this set will perform approximately as well as on the original dataset. The prototypes can be produced by two ways: prototype selection and prototype extraction. Prototype selection seeks to identify a subset of representative objects from the original data, while prototype extraction techniques involve the creation of an entirely new set of objects. We do not employ these methods in our work but we suggest that they may also be applied in conjunction with ours. The previously mentioned works use the VSM based representation of documents whereas our method uses neural document embeddings.

3.2 Method

Our method first leverages transfer learning through a clustering step, and then a co-clustering is constrained to keep the document partitions obtained by the clustering step.

3.2.1 Transfer Learning for Text Clustering

The proposed transfer-based clustering is performed in two steps: the embedding step which leverages transfer learning through an augmented dataset, and the clustering itself which clusters only the documents of the dataset of study, directly using the embeddings produced in the previous step.

3.2.1.1 Learning Document Embedding

In this phase, we use a kind of latent feature learning approach where documents are projected into a low-dimensional space computed from the features of both the documents of interest (the documents we want to cluster) and additional documents from a similar domain (Figure 3.1).

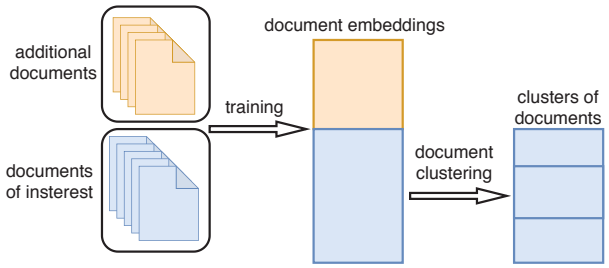


Figure 3.1: Augmented dataset.

We use Paragraph Vector to embed documents into low dimensional real valued dense vectors.

3.2.1.2 Clustering Document Embeddings

We cluster document embeddings corresponding to the dataset of our study. This is done with a directional clustering algorithm: Spherical k-means (Dhillon and Modha, 2001). Spherical k-means is indeed suited to cluster document vectors with cosine similarity (vectors in the same cluster have a high cosine similarity).

Table 3.1: Illustrative example of the construction of $\mathbf{H}$ for $r = 3$ partitions of 7 documents into $g = 3$ clusters.

	$\mathbf{z}^{(1)}$	$\mathbf{z}^{(2)}$	$\mathbf{z}^{(3)}$
d_1	1	2	1
d_2	1	2	1
d_3	1	2	2
d_4	2	3	2
d_5	2	3	3
d_6	3	1	3
d_7	3	1	3

$\Leftrightarrow$

	$\mathbf{Z}^{(1)}$			$\mathbf{Z}^{(2)}$			$\mathbf{Z}^{(3)}$		
d_1	1	0	0	0	1	0	1	0	0
d_2	1	0	0	0	1	0	1	0	0
d_3	1	0	0	0	1	0	0	1	0
d_4	0	1	0	0	0	1	0	1	0
d_5	0	1	0	0	0	1	0	0	1
d_6	0	0	1	1	0	0	0	0	1
d_7	0	0	1	1	0	0	0	0	1

3.2.1.3 Hyper-parameter Setting

Paragraph Vector has several hyper-parameters. The most sensitive are the number of dimensions, the number of epochs and the window size. A grid search over commonly used values in supervised settings is used as a starting point. We then have either to select a single combination of hyper-parameters or to aggregate the results.

For a supervised task, it would be possible to select the hyper-parameters by choosing the setting which results in the highest value of an evaluation metric on a validation set. We however cannot do this in an unsupervised setting. One possible solution is to select them based on the internal criterion of the clustering algorithm. But, since the input data of the clustering algorithm is not the same in each setting (the document vectors depend on the hyper-parameters), we cannot do this. We therefore choose to aggregate the results using consensus clustering.

3.2.1.4 Consensus Clustering

A possible solution to aggregate the results is to combine the vectors before the clustering. As it requires to transform the embeddings, it is hard to keep the good properties of the original embeddings that allow to cluster them well. We therefore prefer to apply an ensemble clustering afterwards: Each combination of hyper-parameters values leads to one clustering and all clustering results are combined into a single clustering. The great challenge in clustering ensemble is the definition of an appropriate consensus function. Many approaches exist (Vega-Pons and Ruiz-Shulcloper, 2011), and we chose to use the graph and hypergraph based methods (Strehl and Ghosh, 2002) which are among the most popular.

Let $\mathbf{X}$ be a data matrix of size $n \times d$ where I is the set of n rows (objects), and J the set of d columns (features). Given a set $\mathcal{S}$ of partitions into g clusters, consensus clustering aims to find a new partition $\mathcal{C}^*$ which maximizes the average of the Normalized Mutual Information (NMI) between $\mathcal{C}^*$ and each partition in $\mathcal{S}$:

$$\mathcal{C}^* = \underset{\mathcal{C}}{\operatorname{argmax}} \sum_{\mathcal{L} \in \mathcal{S}} \frac{1}{|\mathcal{S}|} \mathrm{NMI}(\mathcal{C}, \mathcal{L})$$

where $\mathrm{NMI}(\mathcal{C}, \mathcal{L})$ is estimated by:

$$\mathrm{NMI}(\mathcal{C}, \mathcal{L}) = \frac{\sum_{k,\ell} \frac{N_{k,\ell}}{n} \log \frac{N_{k,\ell}}{N_k \hat{N}_\ell}}{\sqrt{(\sum_k \frac{N_k}{n} \log \frac{N_k}{n})(\sum_\ell \frac{\hat{N}_\ell}{n} \log \frac{\hat{N}_\ell}{n})}},$$

where N_k denotes the number of objects contained in the cluster $\mathcal{L}_k$ ($1 \leq k \leq g$), $\hat{N}_\ell$ is the number of objects belonging to the cluster $\mathcal{C}_\ell$ ($1 \leq \ell \leq g$), $N_{k,\ell}$ denotes the number of objects that are in the intersection between cluster $\mathcal{L}_k$ and cluster $\mathcal{C}_\ell$. Note that the new partition $\mathcal{C}^*$ is not necessarily in the set $\mathcal{S}$ of initial partitions.

Several algorithms such as Cluster-based Similarity Partitioning Algorithm (CSPA), HyperGraph Partitioning Algorithm (HGPA) and Meta-CLustering Algorithm (MCLA) find an approximation of the best consensus clustering (Strehl and Ghosh, 2002).

To do so, an hypergraph is constructed as illustrated in Table 3.1: Given a set of vectors of labels $\mathbf{z}^{(1)} \dots \mathbf{z}^{(r)}$ corresponding to r clustering results, binary membership

indicator matrices $\mathbf{Z}^{(1)} \dots \mathbf{Z}^{(r)}$ are contructed. Then, the matrix $\mathbf{H}$ is constructed as the concatenated block matrix $\mathbf{H} = (\mathbf{Z}^{(1)} \dots \mathbf{Z}^{(r)})$. $\mathbf{H}$ defines the adjacency matrix of a hypergraph which is used to produce an approximation of $\mathcal{C}^*$:

- CSPA creates a similarity matrix $\frac{1}{r}\mathbf{H}\mathbf{H}^T$ and apply a similarity-based clustering algorithm on it;
- HGPA approximates the maximum mutual information objective with a constrained minimum cut objective on the hypergraph;
- MCLA collapses groups of clusters into meta-clusters which are consolidated.

We propose another method for consensus clustering, similar to CSPA: we use CoclustInfo (Algorithm 2) to cluster the matrix $\mathbf{H}\mathbf{H}^T$. The resulting partition of the rows is taken as consensus clustering. Experiments (see table XII, in the discussion section) have shown that this new method outperforms CSPA, HGPA, and MCLA.

3.2.2 Text Co-clustering

Although clustering has been successfully used in a wide range of application domains, clustering approaches may sometimes be challenged by the characteristics, i.e, high dimensionality and sparsity, exhibited by some datasets, such as *document* $\times$ *term* matrices arising in text mining. When we are considering such datasets, co-clustering which, in contrast to clustering, groups objects and features simultaneously, turns out to be more effective (Dhillon et al., 2003a; Govaert and Nadif, 2005a, 2013; Ailem et al., 2016; Salah and Nadif, 2017). The basic idea of co-clustering consists in making permutations of objects and attributes in order to draw a correspondence structure on $I \times J$ making the data easier to handle and interpret.

An information-theoretic based co-clustering, CoclustInfo (Govaert and Nadif, 2018) (Algorithm 2) has been shown in Chapter 2 to handle documents-terms matrices and to result in high quality co-clustering.

3.2.3 Transfer Learning Using a Constrained Text Co-clustering Algorithm

Co-clustering makes the interpretation of clustering easier than one-sided clustering: on a document-term occurrence matrix it forms clusters of documents characterized by clusters of terms. Figure 3.2 shows an example of diagonal co-clustering where each cluster of documents is characterized by a cluster of terms. This section presents how to summarize the data through co-clustering but keeping the clustering of the documents obtained as discussed in the previous section.

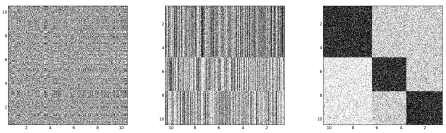


Figure 3.2: Left: original data. Middle: data reorganized according to row clusters. Right: data reorganized according to row and column clusters.

3.2.3.1 Framework

Document clustering is obtained with a clustering of document vectors trained on an extended dataset (Figure 3.1). This high quality document clustering, which leverages local co-occurrences in sliding windows, is kept as the clustering of the documents for a constrained co-clustering algorithm (Figure 3.3). The constrained co-clustering is

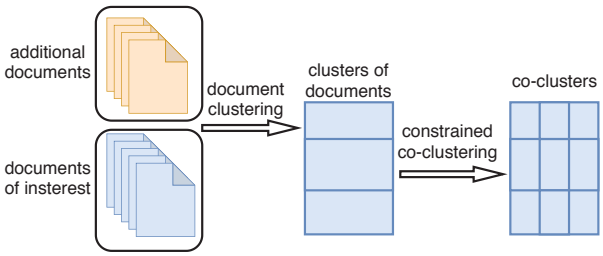


Figure 3.3: Transfer learning for text co-clustering.

applied on the document-term occurrence matrix corresponding to the documents to cluster, enabling thus the observation of the patterns of documents-terms co-clusters. The partitions of the terms make the interpretation possible in contrast to applying a co-clustering on the document embeddings which would not bring the same interpretability since it would cluster the dimensions of the embeddings, which are not associated with understandable labels.

3.2.3.2 Constrained Information-theoretic Co-clustering

The given number of row and column clusters can differ which lets the user a choice over the level of summarization it seeks.

We choose CoclustInfo as the ground for a constrained co-clustering, named ConstrainedCoclustInfo (Algorithm 3). The main difference is the partition of the documents $\mathbf{z}$ which is given as additional input and is not updated.

Algorithm 3 ConstrainedCoclustInfo

Input: $\mathbf{X}$, g, m, $\mathbf{z}$.
Initialization: $\mathbf{w}$, $\gamma_{k\ell} = \frac{p^{\mathbf{zw}}_{k\ell}}{p^{\mathbf{z}}_{k.} p^{\mathbf{w}}_{.\ell}}$.
repeat
  Step 1. $w_{j\ell} = 1$ if $\ell = \text{argmax}_{1 \leq \ell' \leq m} \sum_k p^{\mathbf{z}}_{kj} \log \gamma_{k\ell'}$ and $w_{j\ell} = 0$ otherwise $\forall j$.
  Step 2. $\gamma_{k\ell} = \frac{p^{\mathbf{zw}}_{k\ell}}{p^{\mathbf{z}}_{k.} p^{\mathbf{w}}_{.\ell}}$.
until convergence
return $\mathbf{w}$.

3.3 Experiments

Paragraph Vector is a shallow neural network with one hidden layer. It have two different architectures: Distributed Bag of Words (PV-DBOW) and Distributed Memory (PV-DM). PV-DM is trained to predict a focus word from context words and the id of the paragraph whereas PV-DBOW is trained to predict words randomly sampled from the paragraph from the id of the paragraph.

3.3.1 Datasets

For our experiments, we use four datasets[1] (Cardoso-Cachopo, 2007) of different size and balance[2]:

- **R8** a subset of single topic articles from the Reuters newswire[3].
- **WebKB** a collection of webpages collected by the World Wide Knowledge Base project of the CMU text learning group[4].
- **20ng** the 20 Newsgroups dataset[5].
- **5ng** a subset of five classes (rec.motorcycles, rec.sport.baseball, comp.graphics, sci.space and talk.politics- mideast) from the 20 Newsgroups dataset.

3.3.1.1 Characteristics

Summary statistics of the datasets are given in Table 3.2. We use the test and train sets as given by Cardoso-Cachopo (2007) for the documents to cluster and the additional ones respectively.

[1] `http://ana.cachopo.org/datasets-for-single-label-text-categorization`
[2] The balance is the ratio of the number of documents in the smallest class to the number of documents in the largest class.
[3] `http://www.daviddlewis.com/resources/testcollections/reuters21578/`
[4] `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/`
[5] `http://qwone.com/~jason/20Newsgroups/`

Table 3.2: Characteristics of Datasets.

Dataset	Documents for training	Documents for clustering	Classes	Balance
r8	7674	2189	8	0.013
webkb	4199	1396	4	0.307
5ng	4890	1954	5	0.943
20ng	18821	7528	20	0.628

3.3.1.2 Pre-processing

The following pre-processing is applied for all the datasets:

- Non letter characters (punctuation, numbers, etc.) are transformed to spaces;
- All letters are turned to lowercase;
- Only words longer than two characters are kept;
- Stopwords from the 524 SMART list are removed;
- Stemming is done with Porter's Stemmer.

3.3.2 Evaluation Metrics

In order to evaluate the clustering results, we use the following evaluation metrics:

- Normalized Mutual Information (NMI) (Strehl and Ghosh, 2002)
- Adjusted Rand Index score (ARI) (Steinley, 2004)
- Accuracy (Acc)

The scores are computed against the ground truth labels given by the classes.

3.3.3 Leverage Transfer Learning to Cluster Documents

In order to cluster documents, we conduct experiments on several datasets. First, we set hyper-parameters of Paragraph Vector with a grid search. Then we avoid the selection of hyper-parameters by a consensus clustering step. We evaluate our method on a generalization task.

3.3.3.1 Hyper-parameter Settings

We search through a grid of hyper-parameters in ranges that are known to be effective for supervised tasks. Paragraph Vector is trained on a train set for each value of hyper-parameters and datasets. A spherical k-means is applied on the generated document vectors on a test set.

For the grid search, we choose the window size in the range from 2 to 5, the number of dimensions between 100 and 300 and the number of epochs in [5, 10, 30, 50].

3.3.3.2 Document Clustering

We use Spherical k-means on the document embeddings. For each embeddings matrix (*i.e.* a combination of hyper-parameters), we run 100 clustering with random initialization and take the clustering maximizing the internal criterion of the algorithm.

3.3.3.3 Consensus Clustering

From the results given by clustering the vectors for each combination of hyper-parameters, we create a single consensus clustering using the methods CSPA, HGPA and MCLA of (Strehl and Ghosh, 2002) relying also on (Karypis and Kumar, 1998; Karypis et al., 1999), and our proposed method which applies CoclustInfo on $\mathbf{H}\mathbf{H}^T$. The implementation used for CSPA, HGPA and MCLA is provided by (Giecold et al., 2016). Each method gives an approximation of the best consensus clustering, *i.e.* a clustering that maximizes the average NMI with the input clustering results. The best approximation is selected as our consensus clustering.

3.3.3.4 Generalization from a Subset of Documents

To measure the benefits of our approach, we compare the results found by our method with both clustering and co-clustering algorithms on the rows of the document-term matrix corresponding to the test set. When using the Tf-idf weighting, the weights are computed on the whole matrix, thus can theoretically benefit from a kind of transfer learning.

We apply CoclustInfo and CoclustMod algorithms from the Coclust package (Role et al., 2019), and Spherical k-means on both the document-word co-occurrence matrix (Tf) and the same matrix weighted by Tf-idf. DEC and DeepCC are also considered for the comparison.

3.3.4 Co-clustering with Constraints

For the co-clustering, we fix the document clusters according to the labels of the documents found by the consensus clustering. ConstrainedCoclustInfo is applied with 100 random initializations and the best co-clustering according to the internal criterion is kept as final co-clustering. We set the number of term clusters to be one more than the number of document clusters for better interpretability. This way, the algorithm groups general terms which appear in most of the documents, independently of their class, into a single noisy term cluster.

3.4 Results and Discussion

Clustering quality results in terms of NMI are reported in Table 3.3. One should remember that these results are given for clustering on a subset of the datasets traditionally used as test sets for the supervised task of classification, and not on the whole datasets as clustering are often evaluated, resulting hence in different values than those found in (Role et al., 2019), for instance.

In this context, our method performs better on all four datasets: slightly better than CoclustInfo applied on the document-term co-occurrence matrix (Tf) and Spherical k-means applied on the Tf-Idf matrix for the Reuters8 (r8) dataset, and by a large margin on the other datasets.

One can also see that the co-clustering algorithms applied on document-term matrices do not benefit from the information contained in the additional documents: the inverse document frequency weighting (Tf-idf) is computed on the whole dataset (the documents to cluster and the additional ones) but higher values of NMI are found for the term-frequency (Tf) alternatives. This is in fact coherent with the inner working of the co-clustering algorithms which are supposed to be applied on contingency tables (noted Tf in our experiments).

Table 3.3: NMI values.

		Datasets			
Algorithm	Vectors	r8	webkb	5ng	20ng
Consensus	PV-DBOW	**0.590**	**0.449**	**0.865**	**0.660**
Consensus	PV-DM	0.425	0.340	0.716	0.389
Spherical k-means	Tf-Idf	0.549	0.364	0.782	0.514
	Tf	0.425	0.260	0.070	0.170
CoclustInfo	Tf-Idf	0.351	0.127	0.345	0.216
	Tf	0.383	0.188	0.327	0.217
CoclustMod	Tf-Idf	0.315	0.319	0.215	0.273
	Tf	0.543	0.328	0.472	0.390
DEC		0.409	0.312	0.173	0.234
DeepCC		0.256	0.210	0.323	- [6]

3.4.1 Dataset Augmentation

This section shows the value of using an extended dataset to train the document vectors on.

For each dataset, we randomly sampled 300 documents where 100 constitute the set on which the clustering is done. On one hand clustering is done on the document vectors trained on the whole 300 documents (a), and on the other hand the clustering

[6] We were unable to run DeepCC on this dataset due to the memory consumption. Our configuration has 64GB RAM.

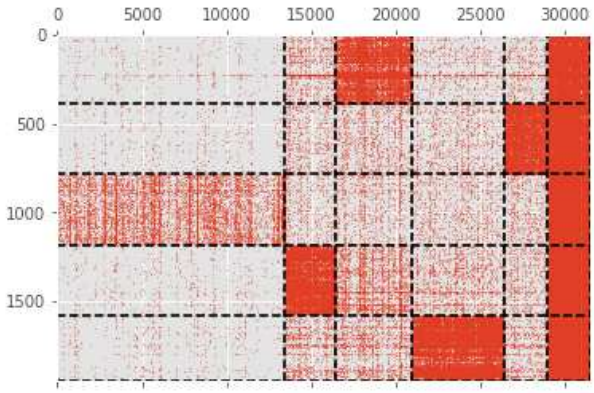


Figure 3.4: Reorganised matrix for ConstrainedCoclustInfo applied on 5ng dataset.

Table 3.4: Top words for each co-cluster formed by ConstrainedCoclustInfo on 5ng dataset with 5 document clusters and 6 word clusters. Terms for dense co-clusters are in bold.

	1	2	3	4	5	6
1	plai, netcom shark, scott	gamma, surfac scienc, design	**bit, file** **graphic, imag**	ran, state unit, di	sale, tek charl, green	**that, you** **for, and**
2	hit, sport clutch, duke	cost, design gov, mile	type, standard mirror, mail	arm, law polic, state	**bmw, ride** **dod, bike**	**for, that** **you, and**
3	**pitcher, hit** **player, game**	star, schedul marin, toronto	sgi, data info, mail	todai, jewish nation, histori	robinson, sabr andrew, hei	**edu, you** **that, and**
4	jack, clark gari, scott	**shuttl, mission** **hst, space**	imag, access rai, system	forc, law govern, right	demon, tank weight, bnr	**you, for** **that, and**
5	lost, offens defens, throw	air, scienc earth, cso	view, set mit, standard	**armenian, arab** **israel, muslim**	demon, dog front, att	**not, you** **that, and**

is done on the document vectors trained on the 100 documents to cluster (b). Quality measures (NMI, ARI and accuracy) are computed on both (a) and (b). This experiment is run 30 times. The difference between the measure evaluated on (a) and on (b) are reported in Table 3.5 along with the p-values corresponding to a Student's t-test where the alternative hypothesis is "True difference in means is greater than 0".

The clustering is run with 100 initializations, and only the best as regards the internal criterion of the clustering algorithm is kept.

The number of epochs for the case (b) is set to be on par with case (a) such that both process the same number of documents in total (number of documents in the training set times the number of epochs).

Number of epochs are set to 30 for case (a). PV-DBOW is used as the document vectorization algorithm with a window of 2 and 300 as the number of dimensions.

Table 3.5: Difference between the scores of evaluation measures for the clustering when the vectorization is done on an augmented dataset and not.

Datasets	Acc diff. (p-value)	ARI diff. (p-value)	NMI diff. (p-value)
20ng	0.0597 (6.6e-12)	0.0754 (4.3e-13)	0.0401 (5.7e-09)
5ng	0.137 (2.7e-07)	0.187 (2.0e-08)	0.207 (8.1e-13)
r8	0.0293 (4.1e-02)	-0.0035 (5.8e-01)	0.0222 (6.2e-02)
webkb	0.0607 (2.7e-03)	0.0910 (3.5e-04)	0.0959 (1.6e-04)

One can see that case (a) consistently results (except for Reuters 8 in terms of ARI) and significantly (except for Reuters 8 in terms of ARI and NMI) in better evaluation scores than case (b). Paragraph Vectors is thus able to leverage transfer learning from additional documents for a clustering task.

3.4.2 Consensus Clustering

Consensus clustering is an essential step since the hyper-parameter values greatly impact the clustering quality. Figure 3.5 shows the distribution of the quality measure scores reported in Tables 3.6 to 3.11 when changing the hyper-parameter values. One can see that no hyper-parameter configuration lead to the highest scores, even increasing the number of epochs do not reliably increases the scores as it is the case for the r8 dataset.

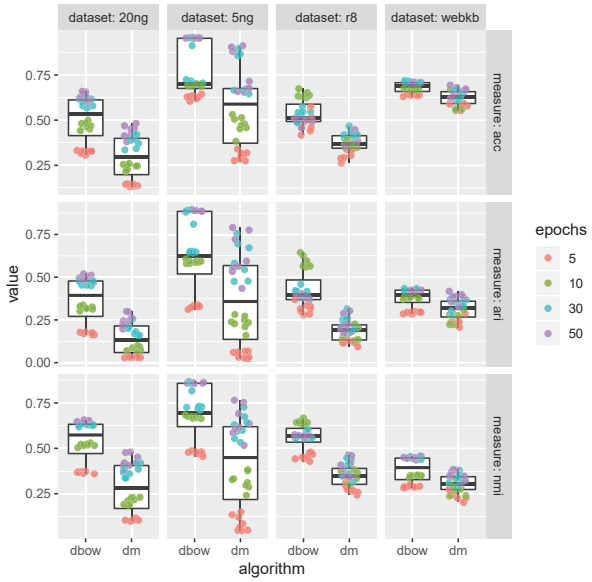


Figure 3.5: Evaluation scores for Paragraph Vectors. NMI, ARI and accuray values vary and depend on the hyper-parameter values.

Table 3.6: NMI values for spherical k-means on PV-DBOW vectors: For each data set, top 3 values are highlighted.

window size	dims	dataset epochs	20ng	5ng	r8	webkb
2	300	5	0.371	0.478	0.454	0.287
		10	0.522	0.673	**0.645**	0.351
		30	0.625	0.719	0.552	0.445
		50	0.648	**0.864**	0.560	**0.456**
	500	5	0.368	0.455	0.443	0.291
		10	0.516	0.665	**0.647**	0.352
		30	0.636	0.710	0.557	0.439
		50	0.645	0.858	0.568	**0.457**
3	300	5	0.373	0.480	0.451	0.282
		10	0.530	0.684	0.642	0.351
		30	0.628	0.729	0.554	0.445
		50	**0.653**	0.863	0.574	0.445
	500	5	0.360	0.471	0.428	0.301
		10	0.519	0.676	0.632	0.352
		30	0.626	0.858	0.557	0.449
		50	0.628	0.862	0.575	0.448
4	300	5	0.369	0.478	0.479	0.287
		10	0.519	0.671	0.642	0.348
		30	0.630	0.816	0.564	0.439
		50	0.634	0.855	0.566	0.451
	500	5	0.367	0.468	0.443	0.293
		10	0.504	0.669	**0.668**	0.348
		30	0.617	0.726	0.598	**0.456**
		50	**0.657**	**0.867**	0.569	0.449
5	300	5	0.370	0.467	0.467	0.285
		10	0.524	0.676	0.642	0.345
		30	0.632	0.731	0.603	0.448
		50	**0.648**	0.859	0.571	0.435
	500	5	0.362	0.487	0.476	0.289
		10	0.514	0.664	0.632	0.337
		30	0.627	**0.867**	0.556	0.446
		50	0.637	0.859	0.567	0.441

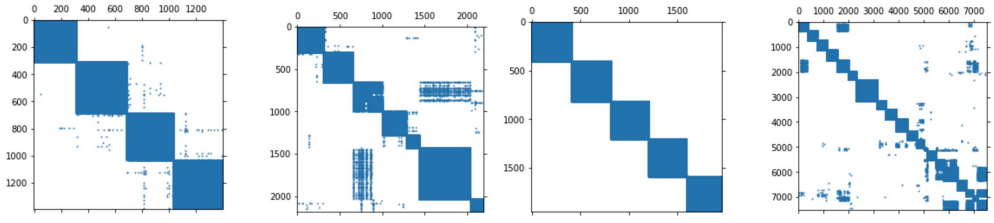
This is also the case when the training is not done on an extended dataset: For instance when training PV-DBOW and clustering the full 5ng dataset, we obtained NMI values in range from 0.396 to 0.899 whereas consensus clustering leads to 0.865. Therefore, consensus clustering is of major importance whether combined with the use of an extended dataset for training or not.

The reported results in Table 3.3 for consensus clustering are the best clustering solution found by CSPA, HGPA, MCLA and CoclustInfo on $\mathbf{H}\mathbf{H}^T$ according to the maximization objective of the average NMI with the input clustering outcomes.

Average NMI found by the different methods for consensus clustering are given in Table 3.12.

Table 3.7: NMI values for spherical k-means on PV-DM vectors: For each data set, top 3 values are highlighted.

window size	dims	dataset epochs	20ng	5ng	r8	webkb
2	300	5	0.107	0.134	0.291	0.276
		10	0.231	0.382	0.366	0.335
		30	0.420	**0.727**	0.440	**0.379**
		50	**0.477**	**0.752**	**0.463**	**0.384**
	500	5	0.117	0.149	0.306	0.274
		10	0.230	0.376	0.382	0.301
		30	0.419	0.630	**0.447**	0.337
		50	**0.479**	**0.764**	**0.460**	**0.380**
3	300	5	0.105	0.121	0.275	0.275
		10	0.227	0.328	0.375	0.281
		30	0.388	0.554	0.416	0.344
		50	**0.452**	0.616	0.417	0.369
	500	5	0.109	0.067	0.257	0.278
		10	0.217	0.332	0.360	0.291
		30	0.385	0.599	0.408	0.343
		50	0.421	0.636	0.408	0.351
4	300	5	0.105	0.084	0.271	0.227
		10	0.191	0.288	0.310	0.273
		30	0.360	0.674	0.360	0.334
		50	0.406	0.606	0.383	0.344
	500	5	0.105	0.046	0.261	0.227
		10	0.211	0.312	0.333	0.259
		30	0.367	0.593	0.347	0.309
		50	0.420	0.586	0.350	0.347
5	300	5	0.098	0.050	0.263	0.241
		10	0.186	0.241	0.302	0.238
		30	0.335	0.642	0.321	0.291
		50	0.405	0.516	0.335	0.310
	500	5	0.102	0.048	0.243	0.203
		10	0.191	0.297	0.300	0.239
		30	0.337	0.532	0.315	0.322
		50	0.380	0.691	0.343	0.322



(a) WebKB (b) Reuters 8 (c) 5ng (d) 20ng

Figure 3.6: Reorganized matrix given by CoclustInfo applied on the matrix $\mathbf{H}\mathbf{H}^T$. The random initial partitions of rows and columns are set to be equal.

Table 3.8: ARI values for spherical k-means on PV-DBOW vectors: For each data set, top 3 values are highlighted.

window size	dims	dataset / epochs	20ng	5ng	r8	webkb
2	300	5	0.174	0.327	0.284	0.296
		10	0.332	0.593	0.577	0.384
		30	0.453	0.642	0.378	0.427
		50	**0.508**	**0.893**	0.382	0.425
	500	5	0.174	0.312	0.329	0.296
		10	0.318	0.581	**0.626**	0.384
		30	0.486	0.652	0.377	0.421
		50	0.496	0.886	0.402	0.424
3	300	5	0.178	0.334	0.334	0.286
		10	0.338	0.607	0.565	0.380
		30	0.454	0.644	0.372	0.421
		50	**0.519**	0.890	0.403	0.423
	500	5	0.164	0.333	0.280	0.300
		10	0.312	0.597	0.559	0.385
		30	0.469	0.880	0.418	**0.434**
		50	0.469	0.890	0.369	0.410
4	300	5	0.170	0.329	0.315	0.284
		10	0.318	0.589	0.566	0.380
		30	0.476	0.810	0.422	0.424
		50	0.478	0.885	0.399	0.425
	500	5	0.170	0.323	0.316	0.296
		10	0.302	0.586	**0.644**	0.378
		30	0.450	0.652	0.459	**0.428**
		50	**0.512**	**0.896**	0.384	0.419
5	300	5	0.178	0.322	0.301	0.294
		10	0.328	0.594	0.565	0.375
		30	0.479	0.647	0.417	**0.434**
		50	0.489	0.889	0.397	0.420
	500	5	0.163	0.337	0.369	0.293
		10	0.326	0.580	**0.595**	0.370
		30	0.475	**0.892**	0.417	0.426
		50	0.486	0.887	0.395	0.416

Table 3.9: ARI values for spherical k-means on PV-DM vectors: For each data set, top 3 values are highlighted.

window size	dims	dataset / epochs	20ng	5ng	r8	webkb
2	300	5	0.032	0.062	0.125	0.246
		10	0.099	0.281	0.171	0.357
		30	0.255	**0.756**	0.260	0.371
		50	**0.297**	**0.776**	**0.289**	**0.418**
	500	5	0.037	0.068	0.147	0.275
		10	0.098	0.272	0.205	0.332
		30	0.258	0.582	**0.315**	0.355
		50	**0.300**	**0.791**	**0.303**	**0.388**
3	300	5	0.029	0.060	0.116	0.287
		10	0.092	0.247	0.226	0.309
		30	0.200	0.476	0.251	0.347
		50	**0.262**	0.564	0.225	**0.398**
	500	5	0.035	0.032	0.116	0.291
		10	0.087	0.239	0.196	0.276
		30	0.211	0.554	0.250	0.367
		50	0.245	0.594	0.220	0.382
4	300	5	0.030	0.067	0.127	0.236
		10	0.072	0.210	0.135	0.282
		30	0.182	0.695	0.208	0.331
		50	0.214	0.558	0.200	0.382
	500	5	0.032	0.023	0.115	0.234
		10	0.085	0.228	0.155	0.235
		30	0.168	0.545	0.200	0.325
		50	0.244	0.533	0.196	0.386
5	300	5	0.031	0.032	0.129	0.230
		10	0.067	0.159	0.133	0.223
		30	0.161	0.672	0.195	0.279
		50	0.217	0.435	0.182	0.356
	500	5	0.030	0.028	0.093	0.206
		10	0.070	0.234	0.132	0.223
		30	0.160	0.477	0.173	0.303
		50	0.197	0.722	0.174	0.328

Table 3.10: Accuracy values for spherical k-means on PV-DBOW vectors: For each data set, top 3 values are highlighted.

window size	dims	dataset / epochs	20ng	5ng	r8	webkb
2	300	5	0.317	0.628	0.440	0.638
		10	0.483	0.696	0.637	0.673
		30	0.566	0.705	0.488	**0.713**
		50	**0.656**	**0.955**	0.499	0.706
	500	5	0.327	0.604	0.500	0.639
		10	0.463	0.686	**0.652**	0.676
		30	0.628	0.724	0.501	0.706
		50	0.628	0.952	0.508	0.705
3	300	5	0.330	0.636	0.516	0.630
		10	0.499	0.702	0.632	0.666
		30	0.579	0.704	0.487	0.705
		50	**0.659**	0.954	0.524	0.708
	500	5	0.306	0.628	0.417	0.644
		10	0.450	0.696	0.624	0.678
		30	0.596	0.950	0.546	**0.716**
		50	0.591	0.954	0.458	0.696
4	300	5	0.332	0.630	0.460	0.636
		10	0.469	0.691	**0.637**	0.671
		30	0.605	0.912	0.543	0.708
		50	0.596	0.952	0.510	0.711
	500	5	0.328	0.624	0.492	0.643
		10	0.442	0.690	**0.673**	0.664
		30	0.579	0.716	0.576	0.707
		50	**0.652**	**0.957**	0.492	0.703
5	300	5	0.327	0.618	0.457	0.635
		10	0.481	0.698	0.628	0.665
		30	0.616	0.701	0.537	**0.718**
		50	0.610	0.954	0.506	0.708
	500	5	0.329	0.643	0.576	0.636
		10	0.467	0.688	**0.637**	0.663
		30	0.616	**0.955**	0.545	0.708
		50	0.626	0.953	0.505	0.698

Table 3.11: Accuracy values for spherical k-means on PV-DM vectors: For each data set, top 3 values are highlighted.

window size	dims	dataset / epochs	20ng	5ng	r8	webkb
2	300	5	0.135	0.340	0.293	0.579
		10	0.246	0.479	0.335	0.671
		30	0.427	**0.896**	0.399	0.668
		50	**0.469**	**0.904**	0.421	**0.693**
	500	5	0.145	0.311	0.338	0.617
		10	0.250	0.472	0.353	0.629
		30	0.421	0.671	**0.468**	0.627
		50	**0.482**	**0.911**	**0.447**	**0.681**
3	300	5	0.132	0.316	0.343	0.632
		10	0.261	0.505	0.415	0.616
		30	0.390	0.663	0.423	0.657
		50	0.451	0.674	0.366	0.667
	500	5	0.145	0.298	0.304	0.630
		10	0.247	0.514	0.398	0.591
		30	0.393	0.663	0.418	0.665
		50	0.413	0.715	0.374	0.658
4	300	5	0.148	0.318	0.338	0.592
		10	0.224	0.452	0.345	0.603
		30	0.387	0.866	0.417	0.648
		50	0.381	0.676	0.373	0.655
	500	5	0.138	0.276	0.289	0.590
		10	0.256	0.459	0.365	0.554
		30	0.372	0.665	0.407	0.606
		50	**0.454**	0.654	0.386	**0.673**
5	300	5	0.144	0.283	0.351	0.583
		10	0.216	0.383	0.349	0.561
		30	0.342	0.856	**0.436**	0.613
		50	0.417	0.659	0.376	0.645
	500	5	0.138	0.275	0.263	0.553
		10	0.227	0.531	0.364	0.561
		30	0.336	0.646	0.413	0.645
		50	0.395	0.880	0.356	0.622

Table 3.12: Average NMI found by consensus algorithms.

vectors	dataset / algorithm	20ng	5ng	r8	webkb
PV-DBOW	CSPA	0.445	0.541	0.441	0.388
	CoclustInfo	**0.652**	**0.815**	**0.691**	**0.512**
	HGPA	0.006	0.229	0.018	0.001
	MCLA	0.452	0.546	0.432	0.396
PV-DM	CSPA	0.199	0.146	0.241	0.351
	CoclustInfo	**0.426**	**0.459**	**0.427**	**0.504**
	HGPA	0.008	0.048	0.003	0.001
	MCLA	0.226	0.151	0.257	0.358

We found that HGPA consistently failed to find a solution near the optimal consensus: Average NMI consistently turned out to be near 0. CSPA, MCLA and CoclustInfo on $\mathbf{H}\mathbf{H}^T$ give way better results: CoclustInfo found better average NMI than both MCLA and CSPA which give similar results in our experiments.

Figure 3.6 gives the reorganized $\mathbf{H}\mathbf{H}^T$ matrix according to the labels returned by CoclustInfo on $\mathbf{H}\mathbf{H}^T$. The documents where all intermediate clustering (when varying the hyperparameter values of Paragraph Vectors) agree is characterized by red blocks with no other points on the same rows or columns.

3.4.3 Constrained Co-clustering

Constraining the co-clustering to respect the document cluster assignments given by consensus clustering on PV-DBOW vectors improves the quality of the document partition over unconstrained co-clustering. The co-clusters are also coherent as one can see on Figure 3.4 and Table 3.4: For instance, the third term cluster (with the following stems in the co-cluster: "bit", "file", "graphic" and "imag") characterizes well the first document cluster which corresponds to the class "comp.graphics". One can also notice the last term cluster which contains general words explaining the high density for the last block of columns on Figure 3.4.

3.5 Conclusion

We proposed a framework to transfer semantic information of an augmented dataset into a co-clustering of a documents-terms matrix. It leverages the semantics learned from local term co-occurrences in sliding windows to provide a high quality clustering of documents. It also simplify the analysis of the clustering through a characterization of the document clusters by clusters of terms.

More specifically, we demonstrated the benefit of Paragraph Vector for a clustering task: It allows to capture information from a large corpus of documents and

encode it into document vectors. Such document vectors, when clustered with Spherical k-means, often result in better clustering quality but highly depend on the hyper-parameters. We showed that consensus clustering created from the clustering obtained with different settings of Paragraph Vector hyper-parameters, consistently leads to high quality document clustering. We presented an information theoretic constrained co-clustering algorithm which provides co-clusters of documents and terms from an initial fixed clustering of the documents. In our framework, it leverages the high quality of the document clustering provided by a consensus clustering over Paragraph Vector document embeddings.

Chapter 4

Unsupervised Evaluation of Text Co-clustering Algorithms Using Neural Word Embeddings

This chapter is an adaptation of the paper: François Role, Stanislas Morbieu and Mohamed Nadif. Unsupervised Evaluation of Text Co-clustering Algorithms Using Neural Word Embeddings. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management. ACM CIKM*, 2018.*

Contents

The previous chapter showed how neural document embeddings can improve the clustering of texts. Also, word embeddings provide semantic similarity measures between words. Considering the ability of neural text embeddings to provide semantic relations between words and between documents, it may be of practical interest to evaluate traditional co-clustering algorithms with large pretrained textual embeddings. This is the matter of this current chapter.

Word embeddings trained on large amount of texts have been evaluated and compared to human judgment on several semantic similarity tasks. Given their high quality, they can thus be used to evaluate the coherence of clusters of terms produced by co-clustering algorithms. Since co-clustering algorithms are most often only evaluated on the document clustering task, evaluating them on such term cluster tasks constitute a first improvement.

However, to really do justice to the two-dimensional nature of co-cluster analysis, it is not enough to assess the quality of the word and document clusters in isolation of each other: measures are needed to evaluate the quality of the matching between the document and term clusters and their potential to form useful co-clusters (Figure 4.1). Also, given the extreme difficulty in obtaining labels for both document and words, it is highly desirable that such measures be "unsupervised", meaning measures that do not need prior knowledge about the ground-truth classes.

To achieve these goals we propose to leverage large, public-domain word embedding matrices such as Word2vec (Mikolov et al., 2013b), GloVe (Pennington et al., 2014a) and FastText (Bojanowski et al., 2017b) which capture the semantics of large sets of words in the form of continuous, low-dimensional representations. These freely-available word embeddings are used to build representations allowing to assess not only how cohesive term clusters are but also to what extend the produced word and

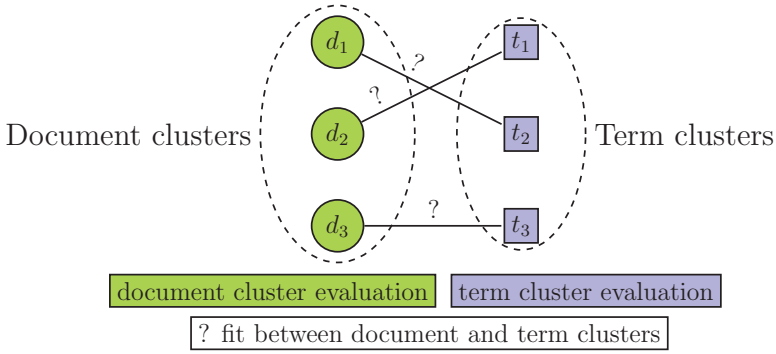


Figure 4.1: Document co-clustering algorithms create document clusters and term clusters. Since co-clustering is a two-dimensional technique, it is necessary not only to assess both term and document clusters in a separate way, but also their potential to form useful co-clusters (pairs associating a document cluster and a term cluster describing the document cluster). To date, most studies only report on the quality of the found document clusters.

document clusters fit well together and have a potential to form interesting co-clusters.

In this chapter, we therefore first show how it is possible to treat term clusters on a par with document clusters as regards evaluation. We then really address the two-dimensional nature of co-cluster analysis by measuring the good fit between term clusters and document clusters and their potential to form useful co-clusters. This is the motivation for a measure we call **FDTC** (Fit between Document and Term Clusters), which has the advantage of being "unsupervised" in the sense that it can be computed in the absence of ground-truth labels. We then show on several real-world datasets how it can help get better insights into the quality of the obtained results and that studying this interaction could even assist in identifying noise in the data so as to achieve better co-clustering performance.

4.1 Evaluating Term Clusters

To validate the results produced by a particular co-clustering algorithm, most studies rely on standard external measures devoted to assessing the quality of document clusters. Such measures include clustering accuracy (Acc[1]) , Normalized Mutual Information (NMI) (Strehl and Ghosh, 2002) and the Adjusted Rand Index (ARI) (Hubert and Arabie, 1985). In addition to being exclusively focused on the document side,

[1] Acc $= \frac{1}{n} \max[\sum_{\mathcal{C}_k, \mathcal{L}_\ell} T(\mathcal{C}_k, \mathcal{L}_\ell)]$where $\mathcal{C}_k$ is an obtained cluster k, $\mathcal{L}_\ell$ is the true ℓ^{th}class and n is the total number of entities. $T(\mathcal{C}_k, \mathcal{L}_\ell)$ is the number of entities which belong to class ℓ and are assigned to cluster k.

these measures also have the drawback of requiring the availability of ground-truth labels.

So, the necessity of evaluating the quality of term clusters in addition to that of document clusters has so far somewhat been neglected by the co-clustering community. The problem of assessing the semantic coherence of a set of terms has been addressed by the topic models community. For example, in (Newman et al., 2009), the following procedure has been proposed for evaluating the quality of a given topic:

1. Select the ten words that score highest on this topic.
2. Form all the 55 possible word pairs from these top-ten words.
3. Compute the PMI score of each pair
4. compute the median of all the PMIs and consider it as the value quantifying the semantic coherence of the topic.

The PMI scores are computed using word-pair co-occurrence statistics from large external data sources such as Wikipedia. Another technique proposed for assessing terms in a topic is the word intrusion task (Chang et al., 2009a), which has then be used for word embeddings (Murphy et al., 2012a). It also relies on a very small number of words (five words and an intruder word per topic) and requires human evaluators.

As concerns the evaluation of word clusters, we propose an alternative solution which differs from the above mentioned ones in several respects. First, although PMI has relatively good agreement with human scoring, it should be kept in mind that it was primarily used by text mining researchers to find collocations between words, what linguists call first-order co-occurrence. Since the goal is to assess to what extent words in a cluster are more similar to each other than to words in different clusters, we propose to measure the similarity between two words as the cosine similarity between the embedding vector representations of these words. Second, restricting the evaluation to a very limited number of words (about ten or so) is problematic from a statistical point of view when handling datasets with a large vocabulary. In contrast, the evaluation measures presented in this chapter are computed from the entire set of words in a cluster and not from small samples.

To be able to evaluate the quality of the term clusters $\mathbf{T}_1, \ldots, \mathbf{T}_k$ produced by a given co-clustering algorithm, we propose to use a large, freely-available word embeddings matrix $\mathbf{E}$. For each possible pair $(\mathbf{T}_i, \mathbf{T}_j), 1 \leq i, j \leq k$ we retrieve an embedding from $\mathbf{E}$ for each word in $\mathbf{T}_i \bigcup \mathbf{T}_j$. After normalizing the embedding vectors we are then able to compute the cosine similarity between every pair $(t_k, t_l), t_k \in \mathbf{T}_i, t_l \in \mathbf{T}_j$. Based on these similarity values we can produce a matrix $\mathbf{S}_t$ of size $k \times k$ where $s_{t_{ij}}$ is the average cosine similarity between a term $t \in \mathbf{T}_i$ and a term $u \in \mathbf{T}_j$. This matrix provides insights into how cohesive and separated term clusters are and can serve as a basis for the computation of various internal measures that do not require

the knowledge of the ground-truth classes. One can take inspiration from measures used in the clustering field. For example, the measure below, which is the ratio between the weighted mean intra-cluster similarity and the weighted mean inter-cluster similarity, is reminiscent of the BetaCV measure with the difference that we are using similarity and not distance, and most importantly that we are dealing with words (not documents) represented by their embeddings:

$$\frac{1}{2}\frac{\sum_{k}\frac{1}{|\{(t_a,t_b)|t_a\in\mathbf{T}_k,t_b\in\mathbf{T}_k,a<b\}|}\sum_{t_a\in\mathbf{T}_k}\sum_{t_b\in\mathbf{T}_k}cos(v_a,v_b)}{\sum_{k}\sum_{l<k}\frac{1}{|\{(t_a,t_b)|t_a\in\mathbf{T}_k,t_b\in\mathbf{T}_l\}|}\sum_{t_a\in\mathbf{T}_k}\sum_{t_b\in\mathbf{T}_l}cos(v_a,v_b)}$$

where t_a and t_b are terms, v_a and v_b are the normalized embedding vectors of terms t_a and t_b resp. The larger the ratio, the better the quality.

4.2 Measuring the Good Fit between Term and Document Clusters

Measuring the quality of document clusters, or of even word clusters, in a separate way, is not enough to judge the quality of the results produced by a co-clustering algorithm. An improvement in the evaluation process is to assess if the produced document and term clusters fit well together. To do so, we leverage large, public domain embedding matrices to compute comparable, centroid representations of both kinds of clusters. An embedding for a document is computed as the mean of the embeddings of the words occurring in it. Then, a document (resp. word) cluster is represented by the centroid of the document (resp. word) embeddings of its members. The centroids of the document clusters are stored in the centroid matrix $\mathbf{C}_d$ and the centroids of the term clusters are stored in the centroid matrix $\mathbf{C}_t$. One can then form the matrix $\mathbf{S} = \mathbf{C}_t\mathbf{C}_d^T$ (Algorithm 4) which gives the similarities between all pairs of term clusters and document clusters.[2]

This matrix provides insights about the compatibility level between the found term clusters and document clusters (Figure 4.4). A global measure can also be derived by first creating a cost matrix $\mathbf{C} = (c_{ij})$ where $c_{ij} = 1 - s_{ij}$. We then can solve the assignment problem by assigning document clusters to term clusters in a way that minimizes the total cost using the Hungarian algorithm. Having thus found the matching $\mathbf{M}$ (set of pairwise nonadjacent edges) between document clusters and term clusters that results in the lowest cost, we can compute the mean value of the edges in $\mathbf{M}$, which can be in $[-1, 1]$, and normalize it to be between 0 to 1. The so computed

[2]Note that the values in $\mathbf{S}$ can be negative since the underlying embedding vectors have mixed-sign components.

value is a measure we call **FDTC** (Fit between Document and Term Clusters), which gives an indication of the level of fit between the k document clusters and l term clusters found by a co-clustering algorithm:

$$\mathbf{FDTC} = \frac{1}{2}\left(1 + \sum_{(d,t)\in\mathbf{M}} \frac{s_{d,t}}{min(k,l)}\right)$$

where k and l are the number of term clusters and document clusters resp.

Algorithm 4

input: set of k term $\mathbf{TC}_1, \ldots, \mathbf{TC}_k$ and l document clusters $\mathbf{DC}_1, \ldots, \mathbf{DC}_l$ produced by a co-clustering algoritm ; matrix $\mathbf{E}$ of word embeddings of size $m \times r$;

output: matrix $\mathbf{S}$ of size $k \times l$ where s_{ij} is the average cosine similarity between term cluster i and document cluster j;

initialization:

- Initialize a zero-valued term centroid matrix $\mathbf{C}_t$ of size $k \times r$;
- Initialize a zero-valued document centroid matrix $\mathbf{C}_d$ of size $l \times r$;

for each term cluster $\mathbf{TC}_i$ **do**

- $V = \varnothing$;

for each term $t \in \mathbf{TC}_i$ **do**

- Search the embedding matrix $\mathbf{E}$ to retrieve the embedding vector e_t for t;
- Add e_t to V;

end for

- Compute and normalize the centroid of all vectors in V and store it as the i-th row of the centroid matrix $\mathbf{C}_t$;

end for

for each document cluster $\mathbf{DC}_i$ **do**

- $V = \varnothing$;

for each term t occurring in a document $\in \mathbf{DC}_i$ **do**

- Search the embedding matrix $\mathbf{E}$ to retrieve the embedding vector e_t for t;
- Add e_t to V;

end for

- Compute and normalize the centroid of all vectors in V and store it as the i-th row of the centroid matrix $\mathbf{C}_d$;

end for

- return $\mathbf{S} = \mathbf{C}_t\mathbf{C}_d^T$;

Figure 4.2 shows that the similarity values in the $\mathbf{S}$ matrix allow to discover the same co-clusters (pairs consisting of a document cluster and a term cluster) as those

directly found by a diagonal algorithm such as the `Mod` co-clustering algorithm. When analyzing the Classic3 dataset using `Mod`, and forming the **S** similarity matrix it can be seen that the pattern of values in **S** allow to identify the same diagonal co-cluster structure as that retrieved by `Mod`.

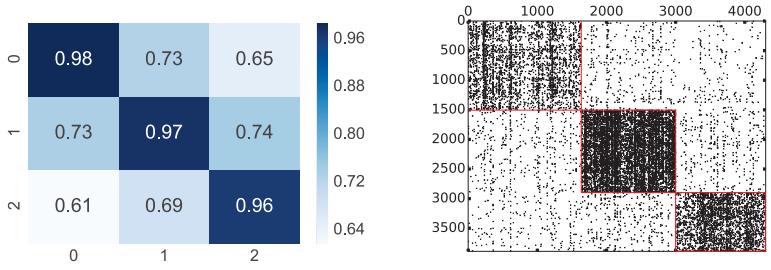


Figure 4.2: Visualization of the similarity matrix **S** for `Mod` on the Classic3 dataset (left). The largest similarity values are on the diagonal, reflecting the co-cluster diagonal structure found by the `Mod` algorithm and exhibited by the document-term matrix where the lines and columns are reorganized according to the labels assigned by the algorithm (right).

It is also possible to use descriptive statistics methods to complement the general performance information provided by **FDTC**. For example this can be done by looking at the **S** matrix and producing boxplots giving indication on the degree of dispersion of the values of **S** corresponding to the best match **M**. From the example in Figure 4.3, it can be seen that, for NG20, `Info` seems to produce more cohesive pairs of document clusters and term clusters than `Mod`. The long tails however show the great variability of the cohesion between term and document clusters for `Info` on NG20. `Mod` produces in contrast less variability since associating pairs of document and term clusters is inherent of diagonal co-clustering. For the other datasets, with a smaller number of classes which are also more separated, the diagonal co-clustering algorithm (`Mod`) finds more cohesive pairs.

4.3 FDTC as an Aid to Noise Identification and Performance Improvement

The similarity **S** matrix between document clusters and term clusters can be used to spot noise words in a dataset. We first analyze the Ng5 dataset using `Info`, keeping the best result over 100 runs, which results in an NMI value of 0.5. The matrix $\mathbf{S} = \mathbf{C}_t\mathbf{C}_d^T$ giving the similarities between all pairs of term and document clusters is then computed, as shown in Figure 4.4 (a). The pattern of values found in the fourth

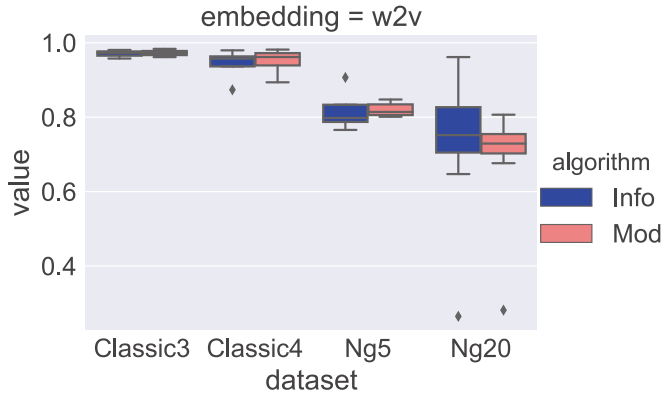


Figure 4.3: Boxplots of the values of **S** corresponding to the best match between document and term clusters.

column (fourth term cluster) makes it stand out as a column (term cluster) having very large values with all rows (document clusters) suggesting that is is a cluster consisting of noisy, very general words. To confirm this intuition we visually inspect the contents of the cluster which indeed mainly consists of dates, adverbs, and general meaning, words such as: "1993apr8" "1993apr9", "1993mar31", "1st", "20", "2000", "2024, "enough", "overall", "see", "seeing", "seeking", "seem", "seemed", "seemingly", "seems", "seen", "sees", etc. Removing the words in this cluster and relaunching the algorithm allows to reach a NMI of 0.6, which is a significant gain, compared to the initial value of 0.5, confirming the value of the information contained in the **S** matrix. Similarly, removing the terms in the three dark columns (term clusters) in Figure 4.4 (b) results in increases of the same order for the NMI, ARI and Acc values, which confirms the ability of the proposed measures to help spot uninteresting clusters.

4.4 Conclusion

We have presented several novel co-clustering evaluation measures which, compared to standard, document oriented ones, better account for the two-dimensional nature of word-document matrices, which is the first contribution of the chapter.

In addition to allowing a fairer evaluation of co-clustering results, these measures also have the advantage of not relying on gold standard labels. In this respect they may be of great use for practitioners working on real-world datasets.

Paths for future research include refining the computation of centroids by using more advanced weighting schemes and using other textual embeddings, in particular those which provide embeddings for both words and documents. We also plan to

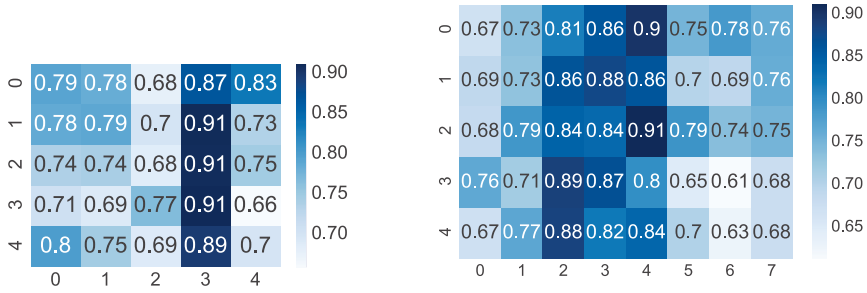


Figure 4.4: (a) Heatmap visualization of the similarity matrix **S** between document clusters (rows) and term clusters (columns) for `Info` on Ng5 with 5 document clusters and 5 term clusters. A noisy term cluster can be spotted (fourth column from the left). (b) `Info` on Ng5 with 5 document clusters and 8 term clusters.

more fully investigate the hints provided by the measures (*e.g.* noise detection) for improving the co-clustering results. Another possible work is to use the FDTC measure as a criterion for a new co-clustering algorithm.

Chapter 5

Handling Out-of-Vocabulary Words

Contents

In the previous chapter, public word embeddings trained on general texts are used to evaluate text co-clustering. A major issue is however the gap that may reside between the vocabulary used on the dataset of study and the one of the word embeddings used for evaluation. Substitution methods are developed to replace unknown word vectors. The topic of this chapter is to evaluate such substitution methods.

Word embedding vectors trained on large amount of publicly available texts commonly serve as input to learning models in numerous domains having each their own particular terminology. Hence, since the data on which the word embeddings have been trained differs from the data on which they are used the so-called "out-of-vocabulary problem" appears: many words encountered in the downstream task may be miss-

ing in the word embeddings vocabulary. These missing words, commonly known as out-of-vocabulary words (OOVs) may very adversely impact the learning process.

For instance, no less than 61% of words in the 20 Newsgroups data set [1] are not found in the well known Word2vec (Mikolov et al., 2013c) model trained on the Google News corpus.

Many practitioners are daily faced with this problem across a large spectrum of applications (text classification, part-of-speech tagging, item recommendation, etc.) and it may harm a system's performance as stated in (Dhingra et al., 2017): "Here we show that seemingly minor choices made on (1) the use of pre-trained word embeddings, and (2) the representation of out-of-vocabulary tokens at test time, can turn out to have a larger impact than architectural choices on the final performance."

In spite of this, there doesn't seem to be any systematic study of the methods used to tackle this problem and how these methods perform. The available information is sketchy and usually spread across papers where the authors briefly mention the often ad-hoc solutions they resorted to. In fact, OOVs are either simply ignored, which is a drastic measure [2], or most often substituted with some kind of replacement vectors built using methods whose quality is not precisely known since it is indirectly measured using downstream tasks that differ from one paper to another.

The main goal of this chapter is therefore to propose generic task-independent evaluation methods allowing to assess to what extent the used replacement vectors are good substitutes for the original ones.

To do this, we propose to compare the substitute and corresponding original vectors from two points of view. On the one hand, we measure how similar in content substitute word vectors are to the original vectors they replace, which is what we call the "word-centric" or "static" point of view. On the other hand, we measure how substitutes behave on word embeddings evaluation tasks, namely similarity and analogy tasks, compared to their corresponding pre-trained word embeddings, which is what we call the "dynamic" point of view.

The outline of the chapter is as follows. After reviewing the methods to create substitute word embeddings for OOVs, we propose new methods to evaluate them, and then experiment to study the quality of the substitutes.

5.1 Handling Out-of-Vocabulary Words

Among the popular methods to create substitute vectors for OOVs, we have

[1] `http://qwone.com/~jason/20Newsgroups/`

[2] Rare words have either poor embeddings in trained word embeddings models, or no embeddings at all: A minimum frequency threshold is indeed often set when selecting the vocabulary for training the word embeddings, as in (Chen et al., 2016) and (Shen et al., 2017).

– **METH1** initializing the unknown embeddings with zeros or random values.
– **METH2** directly averaging a sentence's word vectors. This is the most commonly used method, and we use it as a baseline in our evaluation tasks. For a given out-of-vocabulary word w, we generate a substitute word embedding as follows: Let $C(w)$ be the context of w, *i.e.* the set of words surrounding w in the text. We compute the embedding of w as the centroid of the embeddings of the words in C:

$$Subsitute(w) = \frac{1}{|C(w)|} \sum_{c \in C(w)} \mathbf{E}_c \tag{5.1}$$

where $|C(w)|$ is the number of words in the context of w and $\mathbf{E}_c$ is the embedding of the word c.
– **METH3** computing subword embeddings (character-level (Ling et al., 2015) or n-gram (Bojanowski et al., 2017a) models) and assembling them to create substitute vectors. The best-known example of this approach is FastText (Bojanowski et al., 2017a). For a given pair of target and context words (w_t, w_c), FastText aims to minimize:

$$\log(1 + e^{-s(w_t, w_c)}) + \sum_{n \in N_{t,c}} \log(1 + e^{s(w_t, n)})$$

where $N_{t,c}$ is a set of negative examples sampled from the vocabulary and

$$s(w, c) = \sum_{g \in G_w} z_g^T v_c$$

where G_w is the set of n-grams appearing in w including the word itself, z_g is a vector representation of the n-gram g and v_c is the vector representation of the center word c.
A substitute vector for a word is given by the sum of its n-gram vectors.
– **METH4** relying on auxiliary data (terminologies, thesauri, lexical database such as WordNet, etc.)(Prokhorov et al., 2017; Pilehvar and Collier, 2017). For example, in statistical machine translation, a possible solution, is to find paraphrases of OOV words for which the translations are available, and then aggregate the translations of the found paraphrases (Razmara et al., 2013; Chu and Kurohashi, 2016).

The first of the above-mentioned method (**METH1**) is usually too simplistic to give good results while the last method (**METH4**) may turn out to be effective but heavily depend on the availability of external resources. In the rest of the chapter we focus on the most generic methods, namely **METH2** and **METH3** and use these two methods to demonstrate the possibility of using task-independent evaluation methods.

5.2 Task-independent and Word-centric Evaluation Methods

The main idea is to use as much as possible the same standard methods and datasets that one would use for evaluating "normal" word embeddings. More specifically, we concentrate on the intrinsic and direct quality of the substitute vectors instead of depending on various and constantly changing downstream tasks. Of course, the methods used for evaluating original word vectors can not be used as they are: adjustments are needed, which is the topic of this section.

We propose to measure the quality of a substitute vector along two lines:

- A static point of view: we measure how similar in content generated word vectors are to their originals, how close they are in the embedding space from their known embeddings.
- A dynamic point of view: we measure to what extent they behave like the original on standard similarity and analogy tasks based on common, widely available test datasets.

To evaluate the method that creates a vector for an OOV, we have to compare the substitute with the original vector. For FastText, the substitute embedding of a word is created from the n-grams composing it, so a bias exists when the model is trained on the corpus containing the word we consider as OOV for our experiments. To alleviate this bias, when training the word embeddings, we replace the words used in our tasks by other tokens so that an update for the word does not update the weights of its composing n-grams. The new token is generated by a hash function, so FastText has enough subwords to work with for the given token and also, an update on a token does not impact the embedding of another.

5.2.1 Static Point of View : Measuring the Similarities Between Words and Substitutes Embeddings

5.2.1.1 Word Embeddings and Substitutes

To measure how generated word vectors are close in the embedding space from their known embeddings, we randomly select a set of words and compute for each word the cosine of the two word vectors (original embedding and substitute). For high values, one can consider that word embeddings can be replaced by substitutes.

5.2.1.2 Pair of Word Embeddings and Pair of Substitutes

We also have to check whether a pair of similar words are found to be similar in the substitutes space. We therefore compute the difference between the similarity score

on the word embeddings and the one on the substitutes for a set of pairs of words.

5.2.2 Dynamic Point of View: Compared Behavior on Similarity and Analogy Tasks

Similarity and analogy tasks are often used to evaluate word embeddings: human judgments scores are compared to computations on word embeddings.

5.2.2.1 Similarity Tasks

For similarity tasks, a similarity score is associated to each pair of words by humans, and by computation on word embeddings. Spearman's rank correlation coefficient between the two ranking is then computed.

Human judgment. Similarity tasks measure different levels of similarity or relatedness. We use the following evaluation tasks: wordsim (Finkelstein et al., 2001), mturk (Halawi et al., 2012), rg (Rubenstein and Goodenough, 1965) and mc (Miller and Charles, 1991).

Word Embeddings. Cosine is used as a similarity score between two word embeddings. Since we want to evaluate the computation of embeddings for OOVs, we consider scores obtained with word embeddings for comparison.

Substitutes. We take the substitute embeddings instead of the word embeddings and compare the scores obtained to the scores obtained with word embeddings, and also to the human judgment. The first comparison measures the difference between substitutes and the word embeddings, and the second comparison allows to measure how well they behave on the task word embeddings are used for in the first place.

Word Embeddings and Substitutes. In general, OOVs are rare, so we consider a mix of word embeddings and substitutes to check if some unknown word embeddings can be replaced by substitutes.

5.2.2.2 Analogy Tasks

For analogy tasks, three words are given and a fourth word is to be found by analogy, for instance in "king - man + woman = ?", "queen" is to be found. The proportion of good answers is then computed as a final score. Since the probability to find the substitute for an OOV as the top ranked embedding is low, we compute for each question its rank instead, and report summary statistics. For each question $w_1 - w_2 + w_3 = w_4$, we take

the original embedding for w_1, w_2 and w_3, and compute the rank of the distances of $Substitute(w_4)$ from $\mathbf{E}_{w_1} - \mathbf{E}_{w_2} + \mathbf{E}_{w_3}$ in relation to distances of all embeddings (all the originals and the substitutes for the OOVs) from $\mathbf{E}_{w_1} - \mathbf{E}_{w_2} + \mathbf{E}_{w_3}$.

5.3 Evaluating Two Common Substitution Methods

In this section, we use text8 [3] as training corpora, and also as the corpora on which the substitutes are computed. For training, we use the default parameters of FastText, and the Gensim (Řehůřek and Sojka, 2010) implementation of Word2vec. The context window size to compute the centroids for the substitutes is set to 2.

5.3.1 Static Point of View

5.3.1.1 Word Embeddings and Substitutes

To assess to what extent substitutes are near their corresponding original word embeddings, we randomly select 1000 words from the model trained on the corpus and compute cosine distances between the two vectors (original and substitute). The statistics are given in Figure 5.1. One can see that for Word2vec, computing the centroids with the input embeddings results in higher similarity values than taking the output embeddings. In the second case, the vectors are found to be very different (low value of similarity). For FastText, both centroids and subwords methods result in high similarity values, the centroid method resulting in greater similarity.

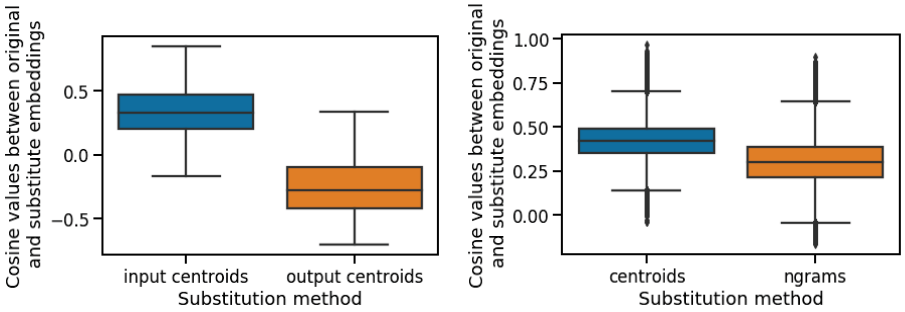


Figure 5.1: Cosine similarity between original and substitute embeddings for Word2vec (left) and FastText (right).

[3] http://mattmahoney.net/dc/textdata

5.3.1.2 Pair of Word Embeddings and Pair of Substitutes

To assess whether two similar (resp. dissimilar) words in the original embeddings are similar (resp. dissimilar) in the substitute embeddings, we compute the differences between their similarity scores in the two embeddings (Figure 5.2). Taking the input embeddings for Word2vec is found to be better than taking the output embeddings. Hence on the two static tasks, it is more appropriate to take the input embeddings. For FastText, the subwords method results in lower differences, and is therefore better than taking the centroids. It is in contrast with the results found in the previous section.

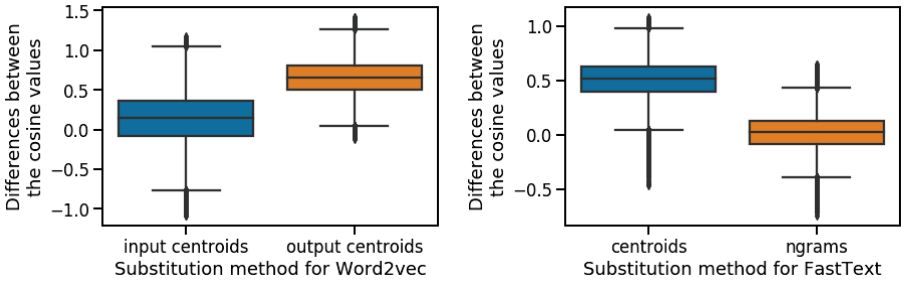


Figure 5.2: Difference between the cosine similarity scores obtained with the original embeddings and the scores obtained with the substitutes, for Word2vec (left) and FastText (right).

5.3.2 Dynamic Point of View

In the following, "w2v" refers to Word2vec and "ft" to FastText. "wc" refers to substitutes as defined in (5.1) when using the centroids of the center word embeddings, and "cc" when using the context word embeddings instead. "ngrams" refers to substitutes as obtained with FastText, as the sum of the embeddings of the n-grams composing the word.

5.3.2.1 Similarity Tasks

The averages of Spearman's rank scores computed between two rankings are given in Tables 5.1, 5.2 and 5.3.

Table 5.1 is given as reference since it gives the scores between the rankings given by the embeddings (Word2vec and FastText) and the human judgement. Table 5.2 gives the scores when taking the substitutes for the first ranking, for both words of the pairs. Taking the input embeddings centroids for Word2vec consistently results in more similar ranking as the originals than taking the output embeddings. Compared

to human judgment taking the output performs better (better on two datasets and slightly worse on two others). These results correspond to the situation where two OOVs are compared. For FastText, the results highly depend on the evaluation dataset. Table 5.3 gives the scores when taking for the first ranking, the substitute for one word and the original embedding for the other word of the pair. In this case, taking the input embeddings centroids as substitutes consistently gives more similar ranking with both original embeddings and human judgment. For FastText, it is the subwords method.

Table 5.1: Similarity tasks.

ranking 1	ranking 2	wordsim	mturk	mc	rg
w2v	human	0.6313	0.5084	0.5507	0.4766
ft	human	0.6954	0.6099	0.6175	0.6200

Table 5.2: Similarity scores when the vectors of each pair are taken from the same embedding: either original or substitute.

ranking 1	ranking 2	wordsim	mturk	mc	rg
w2v_wc	human	0.3501	0.1941	**0.3658**	**0.2654**
w2v_cc	human	**0.3936**	**0.2236**	0.3562	0.2562
w2v_wc	w2v	**0.6551**	**0.4879**	**0.7018**	**0.6479**
w2v_cc	w2v	0.6212	0.4705	0.6155	0.5615
ft_ngrams	human	0.3280	**0.4036**	**0.5000**	**0.4628**
ft_wc	human	**0.3393**	0.2554	0.4760	0.4243
ft_ngrams	ft	0.4701	**0.4962**	0.5452	**0.5779**
ft_wc	ft	**0.5332**	0.4506	**0.7810**	0.5430

Table 5.3: Similarity scores when, for a each pair, the two vectors are originals and substitutes.

ranking 1	ranking 2	wordsim	mturk	mc	rg
w2v_wc / w2v	human	**0.3678**	**0.2904**	**0.2488**	**0.3740**
w2v_cc / w2v	human	0.2652	0.2623	0.1640	0.0992
w2v_wc / w2v	w2v	**0.6009**	**0.5443**	**0.6654**	**0.5996**
w2v_cc / w2v	w2v	0.4855	0.4033	0.5167	0.1142
ft_wc / ft	human	0.3568	0.3535	0.4613	0.4717
ft_ngrams / ft	human	**0.4346**	**0.4673**	**0.5100**	**0.5089**
ft_wc / ft	ft	0.5919	0.6209	0.6275	0.7108
ft_ngrams / ft	ft	**0.6628**	**0.6977**	**0.8251**	**0.7613**

5.3.2.2 Analogy Tasks

In Figure 5.3 are reported the ranks for the Google analogy test set (Mikolov et al., 2013a). There are 14 disparate sets of questions (capital-common-countries (1), capital-world (2), currency (3), city-in-state (4), family (5), gram-adjective-to-adverb (6),

gram-opposite (7), gram-comparative (8), gram-superlative (9), gram-present-participle (10), gram-nationality-adjective (11), gram-past-tense (12), gram-plural (13), gram-plural-verbs (14)) with very different results. For instance, on the first analogy task, the input word embeddings centroids have a lower average of ranks than the output embeddings centroids method but at a higher variance. On the second task, the ranks are low for Word2vec substitutes: $Substitute(w_4)$ are found near the computation of $\mathbf{E}_{w_1} - \mathbf{E}_{w_2} + \mathbf{E}_{w_3}$, particularly with the input embeddings centroids method.

5.4 Conclusion

To the best of our knowledge, this is the first work with a focus on the quality of word vectors meant to serve as substitutes for OOvs. More specifically, we have presented a set of standardized task-independent, word-centric methods to comparatively evaluate the quality of substitute vectors produced using different methods.

Although experimental results are still preliminary, they already give several insights on how some well-known substitution methods behave and compare to each other.

We plan to include more evaluation datasets for future work, and vary the hyper-parameters both for training the word embeddings and the substitutes; for instance by changing the size of the context window.

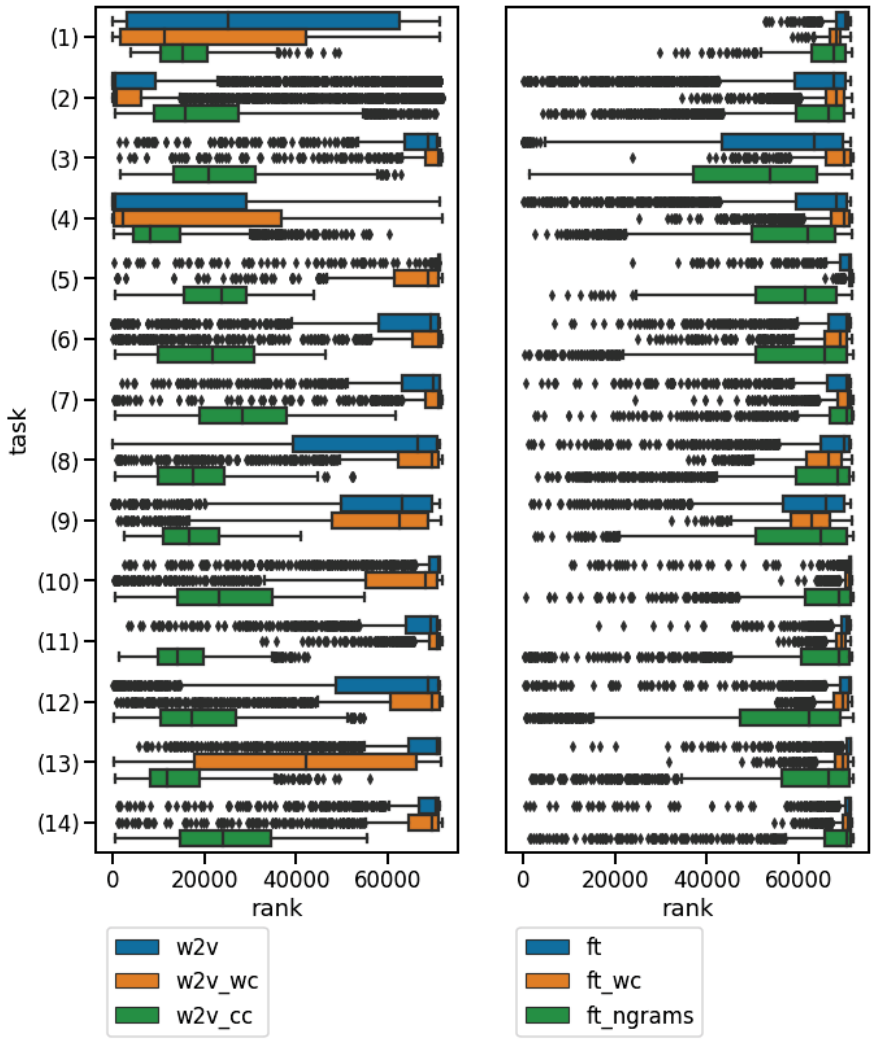


Figure 5.3: Analogy tasks.

Chapter 6

A Scalable Recommender System for Classified Ads

Contents

This chapter covers a use-case where semantic knowledge embedded into word embeddings have been used at Kernix: `http://site-annonce.fr`, a website created by eRows, collects continuously ads from several partners of eRows, such as eBay or LeBonCoin. Around 200 ads per seconds are collected and the database therefore changes frequently. We built a recommender system for this website: for each item, the platform should return a list of recommended ads. The recommender system is based on semantic similarity. The platform is used as the basis of some other projects at Kernix which involve similarity requests between documents, and in particular short ones. It has been used for instance for a recommander system of scientific articles for a pharmaceutical company and a system for monitoring culinary trends for a small domestic appliance company.

The outline of the chapter is as follows: First we present how each document is processed and transformed into a semantic vector, by leveraging word embeddings. Then we discuss an important matter as regards our short text vectorization method, namely how to tune the hyper-parameters for the word embeddings training. Then we discuss how for a given target document we retrieve the most similar ones. Last, technical details about the platform are given.

6.1 Vector Representation of Short Texts

Short texts are nowadays encountered in many applications and dealing with them exposes us to several challenges. This section first presents an overview of short texts sources and their typology. The key differences between short texts and longer ones are then highlighted through several examples. Classical representations of textual documents are explained. The limitations of these representations used for common text mining tasks are exposed in this section when using them on short texts. Similarity measures which are an underlying component common to several methods are last studied.

6.1.1 Typology of Short Texts

In this section, the short texts are described according to three axis:

Figure 6.1: Microblogging messages on Twitter.

- **data sources**: the sources where short texts are encountered;
- **business perspective applications**: the final applications where short texts are encountered;
- **tasks**: the machine learning or data mining tasks where short texts are given as input.

6.1.1.1 Data Sources

Short texts are ubiquitous, encountered on microblogs (Figure 6.1), discussion forums (Figure 6.3), multimedia sharing sites, product reviews, image captions, personal status messages, classified ads, snippets results of search engines (Figure 6.2), etc.

Some *media* impose a maximum number of characters. For instance, on Twitter, the limit is now 280 characters and was only 140 characters when it was launched. Snippet results sizes are also limited by the search engines. For other sources, even if the size is not limited, one can observe a small size in practice (Figure 6.3).

Huge amount of those short texts are posted by individuals, in social media, thus even though the data has the same type, individual styles make texts qualitatively different from each others in the same collection.

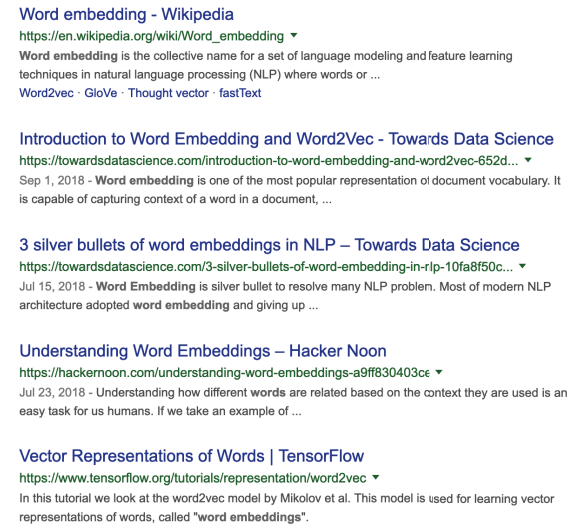


Figure 6.2: Snippets results on Google search engine.

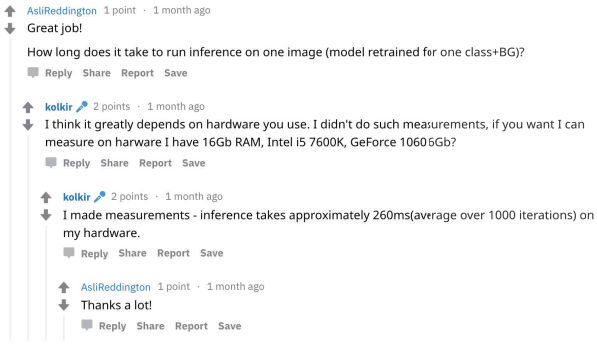


Figure 6.3: Conversation on the reddit discussion forum.

6.1.1.2 Business Perspective Applications

Here is a non exhaustive list of applications where short texts are encountered:

- brand sentiment analysis evolution;
- job recommendations and labor market analytics in online recruitment domain (Zhu et al., 2016);
- music playlist recommendation (Wang et al., 2016);
- item recommendation (Almahairi et al., 2015; He et al., 2017);
- dialog models extraction from call centers conversations.

6.1.1.3 Tasks

Short texts are involved in the same tasks as texts in general:

- **clustering** (Yin and Wang, 2014): given a set of documents, we aim to group documents into clusters so that similar documents are in the same clusters and dissimilar ones are in different clusters;
- **classification** (Song et al., 2014; Zhu et al., 2016): given a list of classes and some examples of document assignations to classes, we aim to classify new documents to the classes;
- **topic modeling** (Li et al., 2016; Qiang et al., 2016; Sridhar, 2015; Yang et al., 2016; Yan et al., 2013; Zuo et al., 2016): given a set of documents, topic modeling aims to create topics as a distribution of words and assign probabilities of topics to each document;
- **information extraction** aims to extract information such as the date and the place of an event from documents;
- **sentiment analysis** (Mostafa, 2013): each document is classified into positive, negative or neutral classes;
- **recommender systems** (Almahairi et al., 2015; He et al., 2017; Wang et al., 2016): for a given item or user, a recommender system provides a recommendation of a new item to the user;
- **information retrieval** (search engines): for a given query, a list of matching documents is retrieved.

6.1.2 Characteristics of Short Texts

In this section, characteristics of short texts are presented along two axis:

- the characteristics which are a consequence of the limited length;
- the quality (*e.g.* misspelling) which is not a consequence of the size of the texts but is often observed in short texts sources.

6.1.2.1 Short Texts *versus* Long Texts

Short texts differ from long ones by two major aspects: the sparsity and the frequencies of the words.

Sparsity. A short text contains a few words leading to a sparsity issue: if a document is represented as a *one hot vector* in the vocabulary space, the vector is often more sparse for a short document than a longer one. Thus two short documents about the same topic have little chance to use the same words and a similarity measure between the two document vectors gives a value close to zero. Longer documents have a higher probability to use common words if there are about the same topic. This sparsity issue lead us to use word embeddings as it provides more information than the presence of the token.

Word Weights. Short texts often contain very few occurrences of words: in general a word appears only once in it. Thus the TF value for the TF-IDF weighting scheme has little meaning.

6.1.2.2 Quality of Short Texts

The quality of texts can be expressed in three ways:

- the informativeness of the content;
- the originality (*e.g.* the diversity of the vocabulary used);
- the correctness (grammar or spelling).

Informativeness. The nature of the medium have a great impact on the quality of the content. For instance, not all messages in an instant messaging system are useful (*i.e.* contain exploitable information). To consider this, Naveed et al. (2011) discuss a static quality measure named *interestingness* which indicates if a content can be of interest as a result to a query. Their score is independent of the query.

Short texts suffer from the lack of information in some documents: for long documents, the uninformative text is only a part of the whole document whereas in short ones, usually in conversations, it appears in distinct messages.

Originality. Another important aspect is the type of text we are dealing with. Finegan-Dollak et al. (2016) study for example the behavior of clustering algorithms on different data sets: captions of photos, cartoon captions and crossword clues. It is important to note that the data sets differ from the creativity point of view: for crossword clues, the authors try to be original and the caption of a photography should be

descriptive. Creative texts on the same subject usually try to not use the same words and this aspect is more important for shorter texts.

Correctness. The most obvious quality aspect of texts is the correctness: spelling or grammar mistakes. Although it is not directly related to the length of the content, it appears that short texts often contain more mistakes than longer ones. Abbreviations and typos are common due to the facility to create and share texts without quality restriction. Press articles and official documents are indeed often longer than user generated content and contain less typos.

6.1.3 Semantic Similarity Measure Between Documents

Semantic similarity measure is at the heart of many tasks such as clustering, classification and recommendation. In this section, we first present how to define semantic similarity measures between documents and then study the behavior of such semantic similarity measures for a clustering task. The clustering task is only used as a proxy to measure the quality of the similarity measures.

6.1.3.1 Similarity Measure

This section discusses the use of document vectors and sets of word vectors in order to provide a semantic similarity measure of two documents.

Comparing Two Vectors. In order to compare two vectors, an often used method is to compute the cosine similarity between the two vectors. This measure is particularly important in the case of high dimensional and sparse vectors (in the Vector Space Model (Salton et al., 1975) for instance). It is also used in the case of Latent Semantic Indexing (LSI) and when comparing word embeddings even if it is less important for those cases. The idea behind this is to capture the orientation of the vector instead of its size as with the euclidean distance. Cosine similarity is the commonly used metric since experiments have shown in a number of tasks with several vector representations that it performs better than euclidean distance.

Comparing Two Sets of Vectors. An other way of comparing documents is to use directly the word embeddings instead of creating a document vector. A document can be represented as a set of word embeddings. The Word Mover's Distance (Kusner et al., 2015) is an instance of such a method. It considers the minimal flow to go from a set of points to the other.

This method is compared, for a clustering task, to Latent Semantic Analysis and to the cosine similarity between the average of word vectors of two documents in 6.1.3.2.

6.1.3.2 Clustering

The study presented here provides a justification of the interest in word embeddings for the similarity of short texts in an unsupervised context. Three previously cited methods, namely Word Mover's Distance, LSI and the cosine similarity between the centroid of the word vectors of two documents are compared.

First the data sets are presented, then the methodology used is presented and last the results are discussed.

Data Sets. The two data sets used for this study are:

- 20 Newsgroups dataset (Ng 20)[1]: each document is a mail in a newsgroup. The 20 Newsgroups dataset contains 11314 documents in 20 classes.
- Web snippets[2]: each document is a web snippet, *i.e.* the text that represents each result of a web search. The web snippets dataset contains 10060 documents of small size (less than 40 words).

Methodology. The methods are compared on a clustering task: grouping documents into semantic clusters. An overview of the methodology is given by the figure 6.4. The figure shows also the retrieval task in the analysis part (Part III): given a document, retrieve the most semantically related documents. This task is useful to see how the methods behave on some queries but since the data sets don't contain information concerning the ground truth for the retrieval task, it is not shown here.

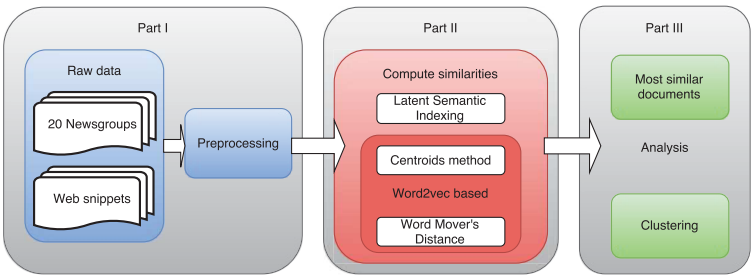


Figure 6.4: Methodology used to compare LSI, word2vec centroid and Word Mover's Distance.

The Ng 20 data set is preprocessed to remove meaningless words called *stopwords*. The headers, footers and quotes are removed since they give too much information about the relationship between mails. After this processing step, some documents are empty so they are not considered in our study. The Web snippets are already bag of words so no preprocessing is done.

[1] `http://qwone.com/~jason/20Newsgroups/`
[2] `http://acube.di.unipi.it/tmn-dataset/`

Since we want to evaluate the methods providing a similarity measure between two documents, we are using a clustering task that takes as input the values of the similarity measures between all pairs of documents. The clustering task is unsupervised since the category labels are only used for evaluation.

The spectral clustering is used since it takes the similarity values as input. It expects as an input a matrix where the values are positives, so we add one to all entries of the similarity matrices (because similarities are by definition lying on the [-1,1] interval). The number of clusters is given by the ground truth.

The Normalized Mutual Information (NMI) (Strehl and Ghosh, 2003) evaluates the quality of the clustering by comparing the result of the clustering algorithm with the ground truth category labels. NMI is estimated by:

$$\text{NMI} = \frac{\sum_{k,\ell} \frac{N_{k,\ell}}{n} \log \frac{nN_{k,\ell}}{N_k \hat{N}_\ell}}{\sqrt{(\sum_k \frac{N_k}{n} \log \frac{N_k}{n})(\sum_\ell \frac{\hat{N}_\ell}{n} \log \frac{\hat{N}_\ell}{n})}},$$

where N_k denotes the number of objects contained in the cluster $\mathcal{C}_k (1 \leq k \leq g)$, $\hat{N}_\ell$ is the number of objects belonging to the class $\mathcal{L}_\ell (1 \leq \ell \leq g)$, and $N_{k,\ell}$ denotes the number of objects that are in the intersection between cluster $\mathcal{C}_k$ and class $\mathcal{L}_\ell$. The larger the NMI, the better the quality of clustering.

Results. The results of this study are informative both in terms of scalability considerations and in terms of quality.

Word Mover's Distance (WMD) is computationally very expensive. The size of the document have a great impact: the best average time complexity of solving the WMD optimization problem scales $O(p^3 \log p)$, where p denotes the number of unique words in the documents.

This method is time consuming: it takes about 8 minutes on a laptop (4 cores and 16 GB of RAM) to compute the distances between one document of the 20 Newsgroups dataset and all the others. When using the *efficient algorithm* called "prefetch and prune" which uses a relaxation of the distance computation problem to prune documents that are not in the top nearest neighbors, so the computation of the true WMD is not done for these documents, this time drops only by a factor two when querying for 20 top documents, so this method is still very slow.

For these sizes of data sets, both LSI and the word embeddings centroid methods are very fast.

The Normalized Mutual Information results are given in Table 6.1. It shows that LSI is the best method for long texts corresponding to the 20 Newsgroups dataset. The intuition is that long texts contain more words that co-occur, so a method like LSI have enough information to compute the similarity between documents.

For the Web snippets dataset, WMD and the Centroid method entail both better clustering than LSI. This corresponds to the lack of co-occurence in small texts. It is noteworthy that WMD is worse than the Centroid method.

Table 6.1: Normalized Mutual Information.

	LSI	*word2vec centroid*	*WMD*
Ng 20	0.40	0.31	too long to process
Web snippets	0.25	0.46	0.39

6.2 Hyper-parameter Tuning and Post-processing

It has been shown in the previous section that using the mean of the word embeddings as a document embedding method has practical advantages when handling short text, both in terms of quality and scalability.

The above study was done on datasets using common words, so the use of large public word embeddings matrices trained on general texts was possible. These public word embeddings are already fine tuned to score high on semantic tasks at the word level. When handling more specific words, we are confronted to out-of-vocabulary words when using these public embeddings. One way to deal with it is to handle out-of-vocabulary words with substitution methods (Chapter 5). These is however not possible when the drift is too important since too many out-of-vocabulary words are present. This section gives therefore indications on what parameters are important when training word embeddings.

6.2.1 Influence of the Number of Occurrences

In this section, the impact of the number of occurrences on the word embeddings models are presented by two ways: the values of preprocessing parameters modify the values of the word similarities and the number of occurrences is related to the main principal component of the embeddings matrix.

6.2.1.1 Preprocessing Parameters Influence on the Word Similarities

First we describe how the models are constructed, then how the analysis of them is done.

Construction of the Models. The corpus consists of the titles of research papers provided by the computer science library DBLP[3].

[3] `http://dblp.uni-trier.de/`

The tokenization is done by lowering the text, splitting on spaces, and stripping dots, colons and commas. The stemming is done using Porter Stemmer (Porter, 1980)[4]. As some terms in computer science are meaningless if they are not grouped together like "unsupervised learning", bigrams are constructed as in (Mikolov et al., 2013a)[5].

The default values of Gensim word2vec implementation are taken for the following parameters:

- training algorithm: CBoW;
- initial learning rate: 0.025;
- the threshold for configuring which higher-frequency words are randomly down-sampled is 10^{-3};
- hierarchical softmax is not used, negative sampling is used instead: 5 "noise words" are drawn;
- the number of iterations over the corpus is 5.

The following parameters are studied, and a model is constructed for each combination of their given values:

- number of training documents (nTitles): 500 000, 1 000 000, 2 000 000, 5 000 000;
- size of the word embeddings (vecSize) (*i.e.* dimension of the word vectors): 100, 200, 500, 1 000, 5 000;
- window size (windowSize): 2, 4, 6;
- minimum total words frequency (minCount): 10, 20, 200, 2 000.

Analysis. For each model, a random sample of pairs of words is taken and the cosine similarities of the words forming the pairs are computed.

We observe that the models with high minimum words frequency have a mean lower than the others. The models contains however not the same words since they are not trained on the same amount of the corpus and some processing is done to prune low frequency words. In the following, models are compared on the same features.

We have m models and a common vocabulary of size n. Here $m = 240$ and $n = 245$.

Each model can be characterized by the similarity values taken by the pairs of words. So, a $m \times n^2$ matrix S is constructed. As $n \times n$ is large, only a sample of the columns is retained (3 000 here).

In order to observe which parameters have the greater influence on the similarity between words, a clustering is done and compared to the clusters induced by the same values of a parameter. For instance, there are 4 clusters of 60 models each, induced by the minCount parameter.

If we cluster the models with 2 clusters using K-means (Figure 6.5) we observe that the clustering match well the clusters induced by the minimum total words frequency

[4]Gensim implementation is used.

[5]The implementation is in the `models.phrases` module of gensim.

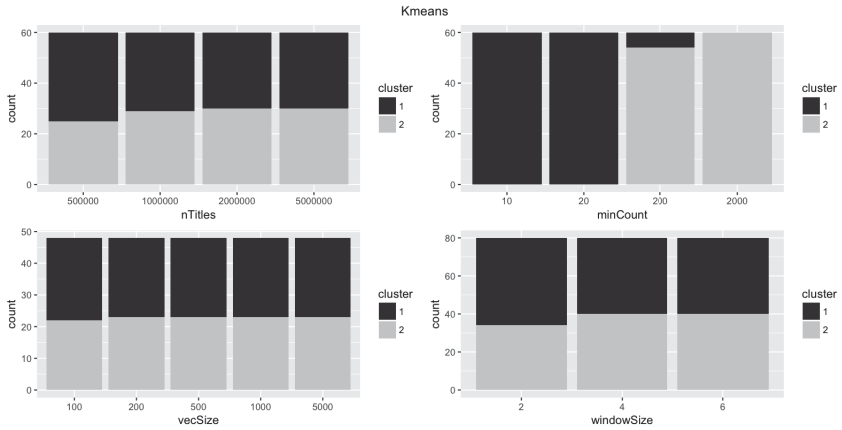


Figure 6.5: Clustering with two clusters of the Word2vec models using K-means.

(minCount parameter). This is shown in the top right corner of the figure where the models with a minCount value equal to 10 and 20 are in a different cluster than the models with a minCount value equal to 200 and 2000. To get a perfect match, the algorithm ClustOfVar (Chavent et al., 2012) (Figure 6.6) is used.

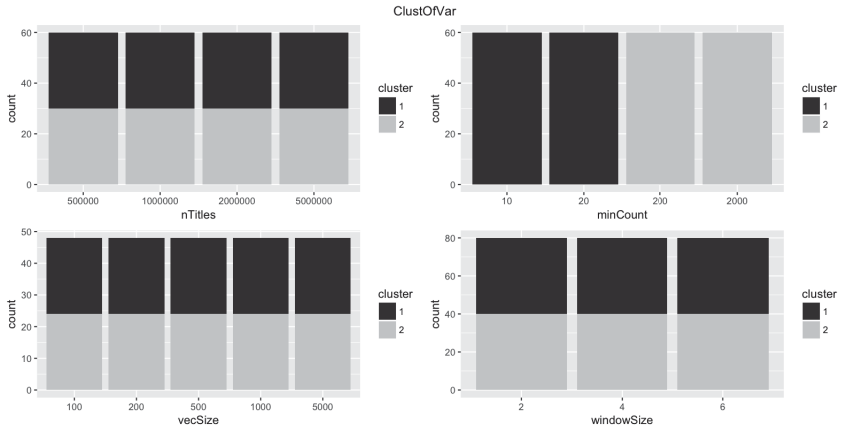


Figure 6.6: Clustering with two clusters of the Word2vec models using ClustOfVar.

When using a clustering with 3 clusters, we can further divide the models (Figure 6.7) into a cluster of models with minCount value equal to 10 and 20, and a cluster with minCount equal to 200 and a third cluster with minCount equal to 2000. The perfect match is obtained using Spherical k-means (Hornik et al., 2012b).

We are unable to get a match for the 4 values of minCount but we can observe (Figure 6.8) when clustering with 4 clusters that only the minCount value and the nTitle parameter have an impact on the clustering. Note that the number of titles has an impact on the frequency of the words since the frequency can only increase when

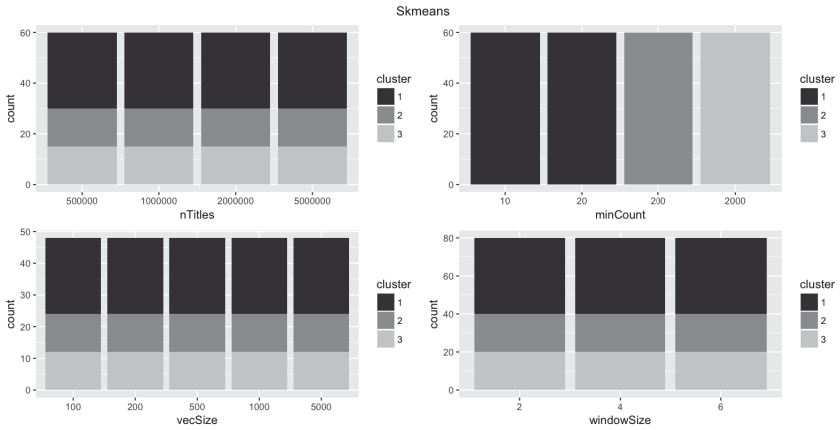


Figure 6.7: Clustering with three clusters of the Word2vec models using Spherical K-means.

nTitle increase.

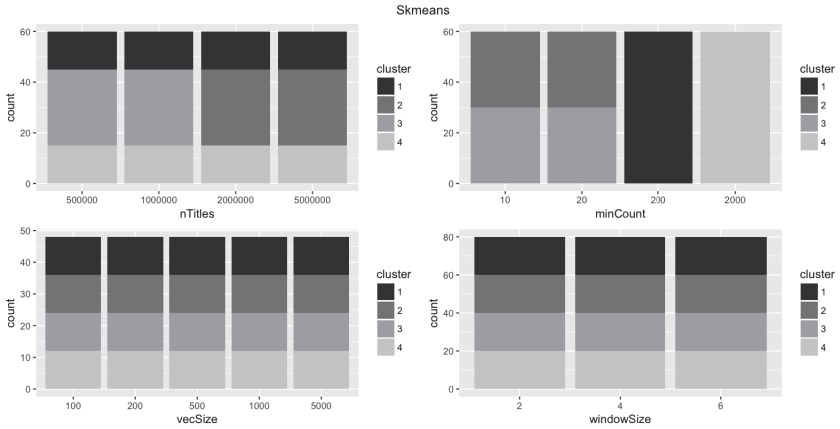


Figure 6.8: Clustering with 4 clusters of the Word2vec models using Spherical K-means.

This study shows that the frequency of the words have a great impact on the values of the similarities of words given by a model.

6.2.1.2 Relations Between the Main Principal Components and the Number of Occurrences

In this section, we take only one model. In order to make the analysis easier, the model is trained on a well known corpus: text8[6].

[6] `http://mattmahoney.net/dc/textdata.html`

When applying the Principal Component Analysis on the word embeddings matrix, we observe a relation between the main principal component and the number of occurrences: Figure 6.9 shows the relation between the contribution of a word to the principal component and its frequency in the corpus. This shows again that the frequency of the words plays a central role in a word embedding model.

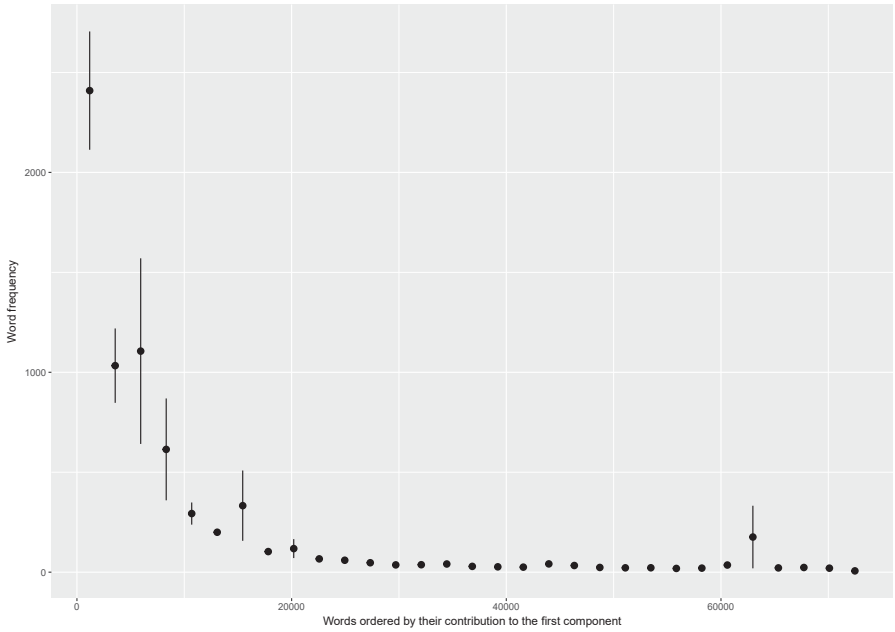


Figure 6.9: Frequency of the words in the corpus in function of their contribution to the main principal component. The values are in bins for clarity of the figure. The mean and the standard error of the bin is plotted.

When looking at the words contributing the most to the main principal component (Table 6.2), we can observe a mixture of topics (actress, molecules, storage). On the contrary, the second principal component is clearly about politics (Table 6.3). The third principal component is about geopolitics (Table 6.4), the fifth is about computers (Table 6.6) and the sixth is about writing (Table 6.7). The fourth is mostly about sport (Table 6.5). The main principal component appears clearly noisier than the others and this noise is to link with the observation on the frequency of the words contributing the most to the main principal component.

6.2.1.3 Conclusion and Perspectives

We provided hints on the parameters which impact the most the word embeddings: the word frequency plays the major role. The minCount parameter of Word2vec should therefore be handled with care.

Table 6.2: Words contributing the most to the main principal component.

Word	Contribution	Frequency
function	0.11	3330
actress	0.11	2031
born	0.10	5246
can	0.10	25519
molecules	0.09	849
properties	0.09	1613
processes	0.09	1007
cells	0.09	1619
earl	0.09	714
components	0.08	849
frequency	0.08	1128
maria	0.08	611
protein	0.08	764
storage	0.08	925
proteins	0.08	639

Table 6.3: Words contributing the most to the second principal component.

Word	Contribution	Frequency
government	0.16	11323
jews	0.14	2620
governments	0.12	1067
policies	0.12	990
religious	0.12	3588
policy	0.12	2278
authority	0.11	2009
independence	0.11	2478
opposition	0.11	1425
political	0.11	6970
leaders	0.11	1430
authorities	0.11	816
citizens	0.11	1194
economic	0.10	4435
reform	0.10	1256

Table 6.4: Words contributing the most to the third principal component.

Word	Contribution	Frequency
km	0.20	4574
airlines	0.14	616
ships	0.13	1352
airport	0.13	1233
rail	0.12	822
bay	0.12	1213
troops	0.12	1724
kilometers	0.12	580
railway	0.12	966
coast	0.12	2013
river	0.12	3984
border	0.11	1531
canal	0.11	606
writings	0.11	1169
pacific	0.11	1406

Table 6.5: Words contributing the most to the 4^{th} principal component.

Word	Contribution	Frequency
him	0.15	10629
languages	0.15	5087
hit	0.14	1676
shot	0.13	1335
ball	0.13	2124
me	0.11	2256
album	0.11	2932
you	0.11	6690
got	0.11	911
season	0.11	2741
dialects	0.11	913
her	0.11	11536
your	0.11	2189
kill	0.11	809
indo	0.11	763

Another approach to take this into account is given by Raunak (2017) and Mu et al. (2017): They improve pre-trained word vectors through post-processing algorithms. Thus, noise can be removed from word embeddings. In particular, they eliminate the common mean vector and a few top dominating directions from the word vectors.

For our purpose, we try some values of the minCount parameter and selected the

value giving the most coherent results on our task: the nearest neighbors retrieval of documents.

Table 6.6: Words contributing the most to the 5^{th} principal component.

Word	Contribution	Frequency
ibm	0.17	1063
software	0.17	3773
corporation	0.16	1104
edition	0.15	1990
digital	0.14	1905
programming	0.13	1989
goddess	0.13	617
intel	0.13	665
microsoft	0.13	1539
communications	0.13	1084
inc	0.12	761
website	0.12	2102
gnu	0.12	627
institute	0.12	1742
program	0.12	3403

Table 6.7: Words contributing the most to the 6^{th} principal component.

Word	Contribution	Frequency
songs	0.19	1605
alphabet	0.18	1088
texts	0.17	1108
testament	0.17	1024
bible	0.16	1627
text	0.16	3098
letters	0.16	2049
stories	0.15	2128
written	0.15	4916
spoken	0.14	1306
languages	0.14	5087
song	0.14	2809
dialects	0.14	913
books	0.13	3992
poems	0.13	618

6.3 Fast Retrieval of the Nearest Classified Ads

In order to inject the semantic similarity relation in a graph, it is useful to only inject the most important relations in order to keep the graph sparse. Only the highest values of similarity are of interest to state that a document is similar to another. A low value of similarity is useless. This leads us to consider algorithms that retrieve only the most similar documents of a considered one: *nearest neighbors algorithms*.

6.3.1 Approximated Nearest Neighbors

Exhaustive nearest neighbors search is expensive regardless of its implementation:

- with scikit-learn (Pedregosa et al., 2011): the matrix of the similarities between documents is a (document × document) matrix that should stay in memory;
- with gensim (Řehůřek and Sojka, 2010): the out-of-core aspect can resolve the memory problem but calculating the values of similarity for all pairs of documents is too slow.

Since the complexity is quadratic in function of the number of documents, the exact method of nearest neighbors is unusable for large data sets. An approximated method is needed.

6.3.2 Locality Sensitive Hashing

Locality Sensitive Hashing (Indyk and Motwani, 1998) and methods using the same principle (Bawa et al., 2005) allow to retrieve the most similar vectors of a considered one.

One drawback is the fact that sometimes, no results are returned since there is no vectors in the same bucket. In order to have enough vectors in the same bucket, a study on the parameters is required. Instead, other approaches are considered.

6.3.3 Annoy

Annoy[7] is an approximated nearest neighbors method that is retained for now for several practical reasons:

- It is easy to install and to use.
- It is fast and precise compared to the other methods with available implementations[8] [9].

6.3.3.1 Algorithm

The Annoy algorithm works as follows:

At training time:

- a binary tree is constructed by dividing at each node of the tree the space in two:
  - sample two points in the space (the subspace if not at the root of the tree);
  - divide the space in two by the hyperplane equidistant of the two selected points;
  - repeat the last two steps for the two subspaces while there is enough points (around 100) in the subspace.
- Repeat the last step in order to produce several trees;

At query time:

- Consider all the vectors that are in the same leaf of the trees;
- Remove duplicated vectors (since there is multiple trees, vectors can appear multiple times);
- Order the considered vectors by exhaustive nearest neighbors search.

Another important thing is to construct a priority queue of the most promising nodes: the nodes are sorted by the maximum distance into the "wrong" side and

[7] https://github.com/spotify/annoy
[8] https://rare-technologies.com/performance-shootout-of-nearest-neighbours-contestants
[9] https://rare-technologies.com/performance-shootout-of-nearest-neighbours-querying

explored in this order. It is useful in order to consider vectors that are close to a split but in the other side of the space than the query vector.

6.3.3.2 Scalability Study

Two parameters drive the use of Annoy: the number of trees to construct and the number of nodes to explore when querying:

- The number of trees affects the build time and the index size. A larger value will give more accurate results, but larger indexes.
- The number of nodes to explore when querying affects the search performance. A larger value will give more accurate results, but will take longer time to return.

Table 6.8: Query time for the approximated nearest neighbors search with Annoy: for each document vector, 100 nearest neighbors are retrieved. 10 trees are used here. The number of nodes to inspect during searching is the default value: the number of trees times the number of neighbors ($10 \times 100 = 1000$).

Number of vectors	Time
135 160	54s
135 160 000	11min 21s

Table 6.8 shows the almost linear time for the approximated nearest neighbors with Annoy. It is to compare with table 6.9 which shows the quadratic query time for the exact nearest neighbors method (gensim (Řehůřek and Sojka, 2010) is used for the implementation). We can note for instance that for 135 160 document vectors in the data set, the approximated method is 104 times faster.

Table 6.9: Query time for the exact nearest neighbors search: for each document vector, 100 nearest neighbors are retrieved.

Number of vectors	Time
10 137	12s
16 895	32s
33 790	2min 15s
67 580	9min 19s
135 160	1h 34min

6.4 Software Architecture

This section describes the software architecture used for the recommender system for classified ads.

First we present the requirements of the platform to fulfill the needs required by the specific use-case, but also the requirements which guide the design of a reusable

platform for other projets. Then several services which compose the overall platform are presented. Last, details are given on the plugin where the text processing is done.

6.4.1 Requirements and Platform Design

Two types of requirements are guiding the design of the platform: a specific use-case at hand and the need for reusable components for other projects developped at Kernix.

6.4.1.1 Recommending Classified Ads at Scale

Classified ads are created continuously by users and their time to live are short: we can consider that an ad is not relevant any more after about three days since its release, because the item could already be sold. The database is thus always rolling. Second, the way we get the ads is something we cannot always control. For instance, about 200 requests per second are received to create or update an ad.

The high frequency of updates drives us to design a system that can first handle requests asynchronously, in order not to drop some updates when the frequency is too high for the platform, and second, to process each ad independently from others. The second point makes the parallelism easier to do since it is parallelism on the data: a cluster of computers or processes can handle only a bunch of requests and we can therefore easily scale the number of computers or processes when the load is more intensive. Also, it is drastically easier to design an algorithm that follows the removal or addition of new data and do not drift. For the above mentioned reason, we decided to design both the main part of the algorithm and the platform as an asynchronously streaming engine.

The nature of the content, small specialized texts written by any user and with lots of typos, drives us to leverage transfer learning through word embeddings.

6.4.1.2 A Scalable Platform

Kernix being a small company which should deal with always specific use-cases, the platform should be versatile enough to handle a large panel of use-cases without major modification in order to drop the cost of both development and maintenance.

We choose a kind of micro services architecture were several parts can be activated or deactivated easily. Two levels of customization for the developer are provided: a separation of concerns through specific containers and plugins.

Each service lives in a distinct container: it is the object of the following section. And the main container, orchestrating almost all the processing, is made of several plugins. One of the plugin, called the "semantic plugin" is responsible of handling the text. It is the object of the last section.

6.4.2 Services

The application is containerized with Docker and the containers are orchestrated with Docker compose. Services are associated with containers. One of the container named app contains the core of the application.

6.4.2.1 App Container

The app container is the main entry point of the application and contains the javascript code. It wraps a web server running on Node.js with Express framework. The web server is launched by PM2.

PM2. PM2 is a process manager. It is used to distribute the workload across several processes. It uses the cluster mode of Node.js which assigns REST calls in a round-robin fashion to the different processes (*i.e.* the first request is send to the first subprocess, the second request to the second subprocess, etc.). Thus, the application uses more effectively the CPUs to handle requests at scale. Most often, its use is transparent since a request is handled completely by only one process. However, care should be taken when using cron jobs: When using cron jobs inside the application, the task is run for all the processes.

6.4.2.2 Databases and Broker Containers

Several databases are synchronized and used for their respective strengths, and each has its own container. A data broker has also its own container.

MongoDB. MongoDB is at the core of the platform. Each entity stored in the platform is stored in MongoDB. An entity of the platform is either a node or a relationship between two nodes.

Neo4j. Neo4j is used as an additional database. It allows fast graph traversals queries and offers a convenient way to manipulate data with a graph structure (i.e. with relations between entities). Although it is not really used for the classified ads use-case, it is convenient for other applications. For instance, graph traversals can be combined with the semantic relations to build some rules provided by expert knowledge.

Reddis. Reddis is an in-memory database used as a cache.

Elasticsearch. Elasticsearch provides fast and convenient queries for textual data.

Data broker. RabbitMq is the data broker used to communicate between the app container and a plugin specific container for which the code can be written in another language such as Python.

6.4.2.3 Plugins Specific Containers

For some plugins, other containers are used to handle some processing outside the app container. This is used for instance by the "com.dbsengine.semantic" plugin in order to process textual data with Python.

6.4.3 Semantic Plugin

The "semantic plugin" allows to retrieve the most similar text documents of a given one. In order to do this:

- a document is represented as a vector;
- the similarity between two documents is given by the cosine similarity of their vector representations.

6.4.3.1 Vectorization

A document is processed by a pipeline described in Figure 6.10 where some alternatives can be selected at each step:

- The string is tokenized, the default behavior is to split on spaces.
- Tokens are transformed: multiple actions are possible, such as lowering the text or stemming.
- Tokens are filtered: multiple actions are possible, such as removing stopwords or names (named entity recognition is used to remove names for some use-cases such as characters in a movie synopsis).
- Tokens are weighted according to their TfIdf score, their number of occurrences or simply by their presence or absence (binary).
- The final step of vectorization involves either Latent Semantic Indexing (truncated Singular Value Decomposition) or averaging the word embeddings.

The final step has been discussed previously: given the type of text given as input (specialized or not, short or longer, diverse or not, with or without spelling errors), and the available physical resources (number of CPUs or memory) or constraints (number of documents, renewal rate, etc.), one of the two possibilities can be chosen or they can also be combined to send different similarity measures in the graph database. The different similarity measures can be selected by expert knowledge at the application level, or combined with other information either on the nodes or between nodes (other similarity measures depending on other characteristics).

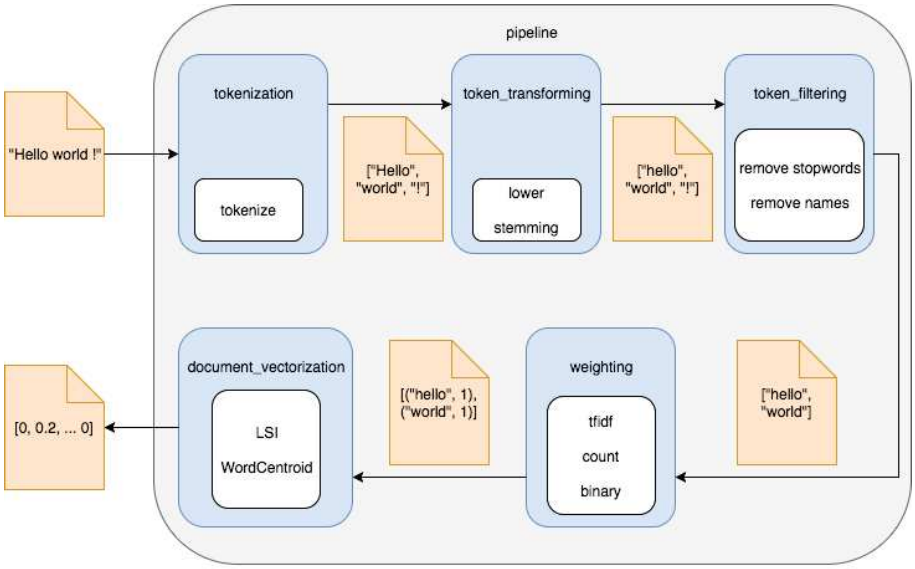


Figure 6.10: Vectorization of a textual document in streaming.

6.4.3.2 Storage

Three storage systems are used:

- a document store (MongoDB) which stores the whole document (*i.e.* several fields, one being the the textual data);
- a graph database (Neo4j) which does not store the textual data (this database is used to navigate through relationships);
- a "similarity model" which stores a vector for each document, and allows fast similarity searches.

6.4.3.3 Similarity Model

The "similarity model" is responsible of storing the vectors and executing retrieval queries of nearest neighbors.

Two kinds of similarity models exist: the brute force algorithm and the Annoy one:

- The brute force algorithm is usable only if the number of vectors is not too much since it needs time to compute the similarity scores for all the pairs of vectors.
- The Annoy algorithm is an approximate nearest neighbors algorithm, so it scales well but at the cost of a potential decrease of accuracy (sometimes the most similar vectors are not retrieved first).

There are two states:

- the "semantic model" is not trained: no retrieval can be done.
- the "semantic model" is trained: for a given input vector, most similar vectors

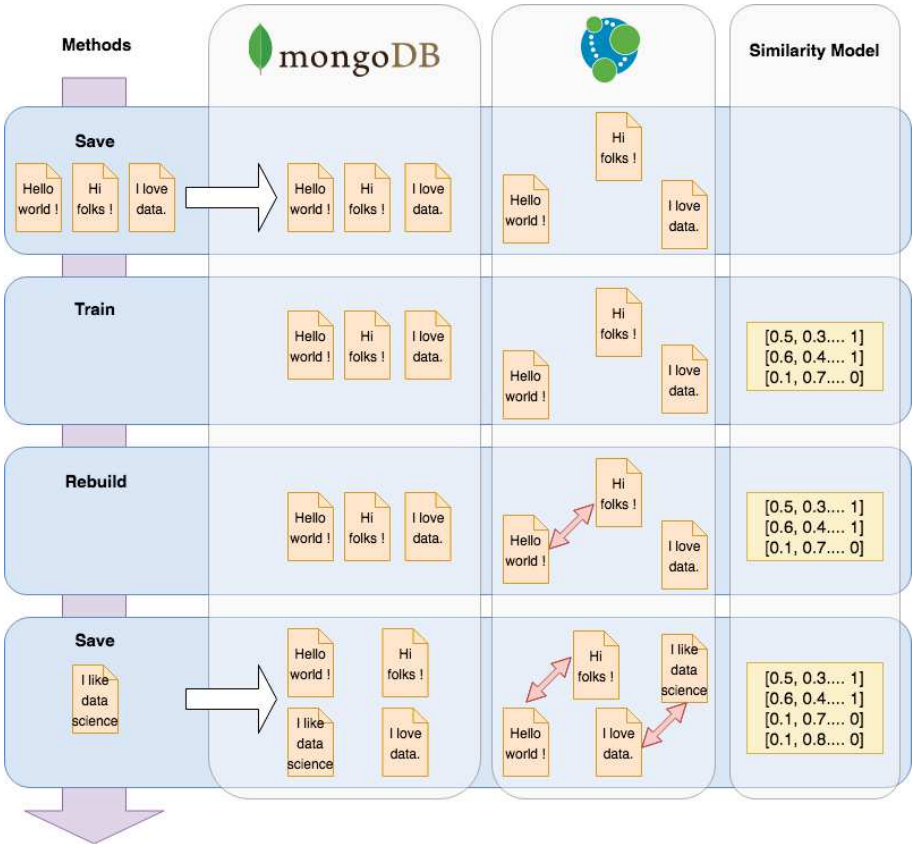


Figure 6.11: Similarity plugin lifecycle.

from the database on which the model has been trained can be retrieved. If some documents are added since the last training, they cannot be retrieved as most similar.

The "semantic model" is controlled by REST requests to the web server (Express on Node.js) and these requests are transmitted to the "semantic container" (running in Python) through a message queue (with RabbitMq). Three requests are exposed (Figure 6.11): save, train and rebuild.

At each *save* request, a textual document is stored in the databases. When the *semantic model* is trained, the vectors are computed for all the documents in the databases. The *rebuild* method train the (approximated) nearest neighbors model and retrieves the most similar documents of each document in the databases, and stores a relationship in the graph database. At each new *save* request, the input document is transformed as vector and stored in the databases, and the similarity relationships between the input document and the previous most similar documents which are already in the databases are stored in the graph database.

6.5 Conclusion

Short texts are ubiquitous and come from many sources, especially on the web. Processing them at scale while capturing their semantics is of a major importance but specific challenges are encountered: their length makes the classical vector representations very sparse and the quality of their content (typos, spelling mistakes, etc.), while not being directly related to their length, makes it even harder.

This chapter presented a study to assess the power of a simple semantic similarity measure leveraging word embeddings in the context of short texts. This method is highly scalable which enables its use in recommender engines of classified ads without the need to use a costly cluster of computers.

Hints are provided to set hyper-parameters for word embeddings training in a totally unsupervised manner.

The software architecture for a recommender system which stores relations in a graph database is shared. The platform enables fast development of new specific use cases where domain knowledge can easily be taken into account through rules on how to combine the characteristics of the entities and the relations between them, one being the semantic relatedness of short texts.

Conclusion and Perspectives

The recent years have seen the emergence of neuronal text embeddings. These new approaches to represent texts as real valued vectors are trained on huge amounts of texts, mainly without the need to label data. However, some hyper-parameters are commonly set with the help of supervised tasks. Through this thesis, we proposed to use neuronal text embeddings for unsupervised tasks. More specifically, we used them to build a semantic similarity measure for short texts, and also for text co-clustering following two lines:

- the evaluation of text co-clustering;
- the transfer of semantic information learned from local co-occurrences of terms to improve text co-clustering.

Text co-clustering is an extension of one-sided clustering which aims to form document-term co-clusters, i.e. the association between a group of documents and a group of terms that describes well the group of documents. The groups of documents are built so that all documents of the same group share the same topics.

Along the way, hints on hyper-parameters influence and how to deal with out-of-vocabulary words, *i.e.* words for which no vectors are found in the word embeddings, are given.

The main contributions and key results of this thesis can be summarized as follows:

- Chapter 1 presents the major textual embeddings methods. A typology of word vectorization methods is highlighted with an emphasis on scalability considerations.

- Chapter 2 exposes a publicly available open source implementation of several co-clustering algorithms which are adapted to textual data. The algorithm implementations come along with visualization utilities to explore co-clustering results in the Coclust Python package. Extensive experimental results on several datasets are presented.

– Chapter 3 proposes a transfer learning framework for text co-clustering. Semantics learned from local co-occurrences of terms on an extended dataset are leveraged to build high quality clusters of documents. Consensus clustering is used to tackle the setting of hyper-parameters for the document embeddings. Co-clustering is constrained to use the high quality partition of documents and build meaningful document-term co-clusters that respect the semantic relationships of documents.

– Chapter 4 presents text co-clustering evaluation methods leveraging large public word embeddings to assess the good fit between documents and terms in co-clusters.

– Chapter 5 presents methods to evaluate out-of-vocabulary word embeddings substitution methods. Since the data on which the word embeddings are trained differs from the data on which they are used, the so called out-of-vocabulary words problem arises: Some words might appear in the dataset of study which are absent in the dataset where the word embeddings are trained. We propose to evaluate the substitution methods which build an embedding for an out-of-vocabulary word.

– Chapter 6 is focussed on short text similarity. A simple semantic similarity method is proposed. Its behavior on short texts is exposed through an experiment in an unsupervised setting. The semantic similarity measure is built on top of word embeddings which makes it process the texts in streaming, allowing thus to scale with huge amount of data. A software architecture based on micro services enables its use for several use cases where semantic relationships between textual entities are needed.

The studies presented in this thesis motivate further investigations that may include:

– Using transformers architectures to assign an embedding for a word depending on the context.

– Use the criterion of the proposed evaluation method for text co-clustering in order to define an objective function for a new co-clustering algorithm.

Publications

Published Articles

1. François Role, Stanislas Morbieu, and Mohamed Nadif. CoClust: A Python Package for Co-clustering. *In Journal of Statistical Software. 2019.*

2. Stanislas Morbieu, François Role, and Mohamed Nadif. Méthodes d'évaluation pour la substitution de vecteurs de mots. *In Société Francophone de Classification. 2019.*

3. François Role, Stanislas Morbieu and Mohamed Nadif. Unsupervised Evaluation of Text Co-clustering Algorithms Using Neural Word Embeddings. *In Proceedings of the 27th ACM International Conference on Information and Knowledge Management. ACM, 2018.*

4. Stanislas Morbieu, François Role and Mohamed Nadif. Le package CoClust : Interface graphique. *In Société Francophone de Classification. 2018.*

5. François Role, Stanislas Morbieu, and Mohamed Nadif. Co-clustering pour la fouille de textes : le package CoClust. *In EGC TextMine. 2017.*

Submitted Articles

1. Stanislas Morbieu, François Role, and Mohamed Nadif. Neural Embedding-based Transfer Learning for Text Clustering and Co-clustering.

Bibliography

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

M. Ailem, F. Role, and M. Nadif. Co-clustering document-term matrices by direct maximization of graph modularity. In *CIKM 2015*, pages 1807–1810, 2015.

M. Ailem, F. Role, and M. Nadif. Graph modularity maximization as an effective method for co-clustering text data. *Knowledge-Based Systems*, 109:160–173, 2016.

A. Almahairi, K. Kastner, K. Cho, and A. Courville. Learning distributed representations from reviews for collaborative filtering. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 147–154, New York, NY, USA, 2015. ACM.

S. Barkow, S. Bleuler, A. Prelić, P. Zimmermann, and E. Zitzler. **BicAT**: A biclustering analysis toolbox. *Bioinformatics*, 22(10):1282–1283, 2006.

M. Baroni, G. Dinu, and G. Kruszewski. Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

M. Bawa, T. Condie, and P. Ganesan. Lsh forest: Self-tuning indexes for similarity search. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 651–660, New York, NY, USA, 2005. ACM.

S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31 –39, march-april 2011.

Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

S. Bird, E. Klein, and E. Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Trans. Math. Softw.*, 28(2):135–151, June 2002.

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017a.

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017b.

C. D. Boom, S. V. Canneyt, S. Bohez, T. Demeester, and B. Dhoedt. Learning semantic similarity for very short texts. *CoRR*, abs/1512.00765, 2015.

A. Cardoso-Cachopo. Improving Methods for Single-label Text Categorization. PdD Thesis, Instituto Superior Tecnico, Universidade Tecnica de Lisboa, 2007.

J. Chang, J. Boyd-Graber, S. Gerrish, C. Wang, and D. M. Blei. Reading tea leaves: How humans interpret topic models. NIPS'09, pages 288–296, 2009a.

J. Chang, S. Gerrish, C. Wang, J. L. Boyd-graber, and D. M. Blei. Reading tea leaves: How humans interpret topic models. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 288–296. Curran Associates, Inc., 2009b.

M. Charrad, Y. Lechevallier, M. B. Ahmed, and G. Saporta. Block clustering for web pages categorization. In *Intelligent Data Engineering and Automated Learning (IDEAL 2009)*, pages 260–267. Springer-Verlag, 2009.

M. Charrad, N. Ghazzali, V. Boiteau, and A. Niknafs. **NbClust**: An R package for determining the relevant number of clusters in a data set. *Journal of Statistical Software*, 61(1):1–36, 2014.

M. Chavent, V. Kuentz-Simonet, B. Liquet, and J. Saracco. Clustofvar: An r package for the clustering of variables. *Journal of Statistical Software, Articles*, 50(13):1–16, 2012.

D. Chen, J. Bolton, and C. D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. In *Association for Computational Linguistics (ACL)*, 2016.

Y. Cheng and G. M. Church. Biclustering of expression data. In *ISMB2000 – The 8th International Conference on Intelligent Systems for Molecular Biology*, volume 8, pages 93–103, 2000.

H. Cho and I. S. Dhillon. Coclustering of human cancer microarrays using minimum sum-squared residue coclustering. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(3):385–400, 2008.

C. Chu and S. Kurohashi. Paraphrasing out-of-vocabulary words with word embeddings and semantic lexicons for low resource statistical machine translation. In *LREC*, 2016.

A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

A. M. Dai, C. Olah, and Q. V. Le. Document embedding with paragraph vectors. In *NIPS Deep Learning Workshop*, 2015.

Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.

S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

M. Deodhar and J. Ghosh. Scoal: A framework for simultaneous co-clustering and learning from complex data. *ACM Transactions on Knowledge Discovery from Data*, 4(3):11, 2010.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, Jan 2001.

I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 89–98. ACM, 2003a.

I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *SIGKDD*, pages 89–98. ACM, 2003b.

I. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'01, pages 269–274, 2001.

P. S. Dhillon, D. P. Foster, and L. H. Ungar. Eigenwords: Spectral word embeddings. *Journal of Machine Learning Research*, 16:3035–3078, 2015.

B. Dhingra, H. Liu, R. Salakhutdinov, and W. W. Cohen. A comparative study of word embeddings for reading comprehension. *CoRR*, abs/1703.00993, 2017.

C. Ding, T. Li, W. Peng, and H. Park. Orthogonal non-negative matrix tri-factorization for clustering. In *KDD'06 – Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 126–135, 2006.

C. Ding and T. Li. Adaptive dimension reduction using discriminant analysis and k-means clustering. In *Proceedings of the 24th international conference on Machine learning*, pages 521–528. ACM, 2007.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

K. Eren, M. Deveci, O. Küçüktunç, and Ü. V. Çatalyürek. A comparative analysis of biclustering algorithms for gene expression data. *Briefings in Bioinformatics*, 14(3): 279–292, 2013.

M. Faruqui, Y. Tsvetkov, P. Rastogi, and C. Dyer. Problems with evaluation of word embeddings using word similarity tasks. In *Proc. of the 1st Workshop on Evaluating Vector Space Representations for NLP*, 2016.

C. Finegan-Dollak, R. Coke, R. Zhang, X. Ye, and D. R. Radev. Effects of creativity and cluster tightness on short text clustering performance. In *ACL*, 2016.

L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. Placing search in context: The concept revisited. In *WWW '01*, WWW '01, pages 406–414, 2001.

A. Fonarev, O. Hrinchuk, G. Gusev, P. Serdyukov, and I. V. Oseledets. Riemannian optimization for skip-gram negative sampling. *CoRR*, abs/1704.08059, 2017.

A. Freitas, W. Ayadi, M. Elloumi, L. J. Oliveira, and J.-K. Hao. Survey on biclustering of gene expression data. In M. Elloumi and A. Y. Zomaya, editors, *Biological Knowledge Discovery Handbook*, pages 591–608. 2012.

R. Gaujoux and C. Seoighe. A flexible R package for nonnegative matrix factorization. *BMC Bioinformatics*, 11(1):1, 2010.

T. George and S. Merugu. A scalable collaborative filtering framework based on coclustering. In *ICDM'05*, pages 625–628. IEEE Computer Society, 2005.

G. Giecold, E. Marco, S. P. Garcia, L. Trippa, and G.-C. Yuan. Robust lineage reconstruction from high-dimensional single-cell data. *Nucleic acids research*, 44 (14):e122–e122, 2016.

Y. Goldberg and O. Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *CoRR*, abs/1402.3722, 2014.

E. Gonzalez and J. Turmo. Comparing non-parametric ensemble methods for document clustering. In *International Conference on Application of Natural Language to Information Systems*, pages 245–256. Springer, 2008.

G. Govaert and M. Nadif. An EM algorithm for the block mixture model. *IEEE Transactions on Pattern Analysis and machine intelligence*, 27(4):643–647, 2005a.

G. Govaert and M. Nadif. Block clustering with bernoulli mixture models: Comparison of different approaches. *Computational Statistics & Data Analysis*, 52(6):3233–3245, 2008.

G. Govaert and M. Nadif. Fuzzy clustering to estimate the parameters of block mixture models. *Soft Computing*, 10(5):415–422, 2006.

G. Govaert and M. Nadif. Clustering with block mixture models. *Pattern Recognition*, 36(2):463–473, 2003.

G. Govaert and M. Nadif. Mutual information, phi-squared and model-based coclustering for contingency tables. *Advances in Data Analysis and Classification*, 12 (3):455–488, 2018.

G. Govaert and M. Nadif. An EM algorithm for the block mixture model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):643–647, 2005b.

G. Govaert and M. Nadif. *Co-Clustering*. John Wiley & Sons, 2013.

D. Greene and P. Cunningham. Efficient ensemble methods for document clustering. Technical report, Department of Computer Science, Trinity College Dublin, 2006.

N. Gupta and S. Aggarwal. Mib: Using mutual information for biclustering gene expression data. *Pattern Recognition*, 43(8):2692–2697, 2010.

H. Hakami and D. Bollegala. Compositional Approaches for Representing Relations Between Words: A Comparative Study. *ArXiv e-prints*, September 2017.

G. Halawi, G. Dror, E. Gabrilovich, and Y. Koren. Large-scale learning of word relatedness with constraints. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 1406–1414, New York, NY, USA, 2012. ACM.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The **Weka** data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

B. Hanczar and M. Nadif. Using the bagging approach for biclustering of gene expression data. *Neurocomputing*, 74(10):1595–1605, 2011.

B. Hanczar and M. Nadif. Ensemble methods for biclustering tasks. *Pattern Recognition*, 45(11):3938–3949, 2012.

B. Hanczar and M. Nadif. Bagging for biclustering: Application to microarray data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 490–505. Springer, 2010.

B. Hanczar and M. Nadif. Precision-recall space to correct external indices for biclustering. In *International Conference on Machine Learning*, pages 136–144, 2013.

Z. Harris. Distributional structure. *Word*, 10:146–162, 1954.

J. A. Hartigan. Direct clustering of a data matrix. *Journal of the American Statistical Association*, 67(337):123–129, 1972.

N. Hartmann, E. Fonseca, C. Shulby, M. Treviso, J. Rodrigues, and S. Aluisio. Portuguese Word Embeddings: Evaluating on Word Analogies and Natural Language Tasks, 2017. arXiv:1708.06025v1.

J. He, H. H. Zhuo, and J. Law. Distributed-representation based hybrid recommender system with short item descriptions. *CoRR*, abs/1703.04854, 2017.

R. Henriques, C. Antunes, and S. C. Madeira. A structured view on pattern mining-based biclustering. *Pattern Recognition*, 48(12):3941–3958, 2015.

F. Hill, K. Cho, and A. Korhonen. Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1367–1377, San Diego, California, June 2016. Association for Computational Linguistics.

F. Horn. Context encoders as a simple but powerful extension of word2vec. *arXiv preprint arXiv:1706.02496*, 2017.

K. Hornik, I. Feinerer, M. Kober, and C. Buchta. Spherical k-means clustering. *Journal of Statistical Software*, 50(10):1–22, 2012a.

K. Hornik, I. Feinerer, M. Kober, and C. Buchta. Spherical k-means clustering. *Journal of Statistical Software*, 50(10):1–22, 2012b.

E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 873–882, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

R. T. Ionescu and A. Butnaru. Vector of locally-aggregated word embeddings (VLAWE): A novel document-level representation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 363–369, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

S. Ji, H. Yun, P. Yanardag, S. Matsushima, and S. V. N. Vishwanathan. Wordrank: Learning word embeddings via robust ranking. *CoRR*, abs/1506.02761, 2015.

E. Jones, T. Oliphant, and P. Peterson. **SciPy**: Open source scientific tools for Python, 2001–. [Online; accessed 2016-06-08].

S. Kaiser and F. Leisch. A toolbox for bicluster analysis in R. In P. Brito, editor, *Compstat 2008 – Proceedings in Computational Statistics*, volume II. Physica Verlag, Heidelberg, Germany, 2008.

N. Kaji and H. Kobayashi. Incremental Skip-gram Model with Negative Sampling. *ArXiv e-prints*, April 2017.

A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. **kernlab** – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004.

G. Karypis. Cluto: A clustering toolkit. Technical Report 02-017, Department of Computer Science, University of Minnesota, 2003.

G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: applications in vlsi domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(1):69–79, 1999.

T. Kenter and M. de Rijke. Short text similarity with word embeddings. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 1411–1420, New York, NY, USA, 2015. ACM.

R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.

Y. Kluger, R. Basri, J. T. Chang, and M. Gerstein. Spectral biclustering of microarray cancer data: Co-clustering genes and conditions. *Genome Research*, 13:703–716, 2003.

S. Kombrink, T. Mikolov, M. Karafiát, and L. Burget. Recurrent neural network based language modeling in meeting recognition. In *INTERSPEECH 2011, 12th Annual Conference of the International Speech Communication Association, Florence, Italy, August 27-31, 2011*, pages 2877–2880, 2011.

M. J. Kusner, Y. Sun, N. I. Kolkin, K. Q. Weinberger, et al. From word embeddings to document distances. In *ICML*, volume 15, pages 957–966, 2015.

L. Labiod and M. Nadif. Co-clustering for binary and categorical data with maximum modularity. In *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, pages 1140–1145, 2011.

A. J. Landgraf and J. Bellay. word2vec Skip-Gram with Negative Sampling is a Weighted Logistic PCA. *ArXiv e-prints*, May 2017.

Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

R. Lebret and R. Collobert. Word embeddings through hellinger pca. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 482–490. Association for Computational Linguistics, 2014.

J. B. Leger. **Blockmodels**: A R-package for estimating in latent block model and stochastic block model, with various probability functions, with or without covariates. arXiv:1602.07587 [stat.CO], 2016.

O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 2177–2185, Cambridge, MA, USA, 2014. MIT Press.

O. Levy, Y. Goldberg, and I. Ramat-Gan. Linguistic regularities in sparse and explicit word representations. In *CoNLL*, pages 171–180, 2014.

O. Levy, Y. Goldberg, and I. Dagan. Improving distributional similarity with lessons learned from word embeddings. *TACL*, 3:211–225, 2015.

C. Li, H. Wang, Z. Zhang, A. Sun, and Z. Ma. Topic modeling for short texts with auxiliary word embeddings. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 165–174, New York, NY, USA, 2016. ACM.

Z. Lin, M. Feng, C. N. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio. A structured self-attentive sentence embedding. *ArXiv*, abs/1703.03130, 2017.

W. Ling, C. Dyer, A. W. Black, I. Trancoso, R. Fermandez, S. Amir, L. Marujo, and T. Luis. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530. Association for Computational Linguistics, 2015.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

K. Lund and C. Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208, 1996.

S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004.

C. May, K. Duh, B. Van Durme, and A. Lall. Streaming Word Embeddings with the Space-Saving Algorithm. *ArXiv e-prints*, April 2017.

O. Melamud and J. Goldberger. Information-theory interpretation of the skip-gram negative-sampling objective function. 2017.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013a.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013b.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013c.

T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *Hlt-naacl*, volume 13, pages 746–751, 2013d.

G. A. Miller and W. G. Charles. Contextual correlates of semantic similarity. *Language & Cognitive Processes*, 6(1):1–28, 1991.

S. Morbieu, F. Role, and M. Nadif. Le package coclust : Interface graphique. 2018.

M. M. Mostafa. More than words: Social networks' text mining for consumer brand sentiments. *Expert Systems with Applications*, 40(10):4241–4251, 2013.

J. Mu, S. Bhat, and P. Viswanath. All-but-the-top: Simple and effective postprocessing for word representations. *CoRR*, abs/1702.01417, 2017.

B. Murphy, P. P. Talukdar, and T. M. Mitchell. Learning effective and interpretable semantic models using non-negative sparse embedding. pages 1933–1950, 2012a.

B. Murphy, P. P. Talukdar, and T. M. Mitchell. Learning effective and interpretable semantic models using non-negative sparse embedding. In M. Kay and C. Boitet, editors, *COLING*, pages 1933–1950. Indian Institute of Technology Bombay, 2012b.

M. Nadif and G. Govaert. Model-based co-clustering for continuous data. In *Ninth International Conference on Machine Learning and Applications (ICMLA)*, pages 175–180, 2010.

N. Naveed, T. Gottron, J. Kunegis, and A. C. Alhadi. Searching microblogs: Coping with sparsity and document quality. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 183–188, New York, NY, USA, 2011. ACM.

N. Nayak, G. Angeli, and C. D. Manning. Evaluating word embeddings using a representative suite of practical tasks. *ACL 2016*, page 19, 2016.

D. Newman, S. Karimi, and L. Cavedon. External evaluation of topic models. In *in Australasian Doc. Comp. Symp., 2009*. Citeseer, 2009.

M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, 2004.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

H. Peng, J. Li, Y. Song, and Y. Liu. Incrementally learning the hierarchical softmax function for neural language models. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 3267–3273, 2017.

J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014a.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014b.

M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference*

of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

M. T. Pilehvar and N. H. Collier. Inducing embeddings for rare and unseen words by leveraging lexical resources. Association for Computational Linguistics, 2017.

M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

V. Prokhorov, M. T. Pilehvar, D. Kartsaklis, P. Lió, and N. Collier. Learning rare word representations using semantic bridging. *arXiv preprint arXiv:1707.07554*, 2017.

J. Qiang, P. Chen, T. Wang, and X. Wu. Topic modeling over short texts by incorporating word embeddings. *CoRR*, abs/1609.08496, 2016.

A. Radford. Improving language understanding by generative pre-training. 2018.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

V. Raunak. Effective Dimensionality Reduction for Word Embeddings. *ArXiv e-prints*, August 2017.

M. Razmara, M. Siahbani, R. Haffari, and A. Sarkar. Graph propagation for paraphrasing out-of-vocabulary words in statistical machine translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1105–1115, 2013.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018.

R. Řehůřek. Scalability of semantic analysis in natural language processing, 2011.

R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

D. L. T. Rohde, L. M. Gonnerman, and D. C. Plaut. An improved model of semantic similarity based on lexical co-occurence. *COMMUNICATIONS OF THE ACM*, 8: 627–633, 2006.

F. Role and M. Nadif. Handling the impact of low frequency events on co-occurrence based measures of word similarity - a case study of pointwise mutual information. In *Proceedings of the International Conference on Knowledge Discovery and Information Retrieval (IC3K 2011)*, pages 218–223, 2011.

F. Role, S. Morbieu, and M. Nadif. Coclust: A python package for co-clustering. *Journal of Statistical Software, Articles*, 88(7):1–29, 2019.

X. Rong. word2vec parameter learning explained. *CoRR*, abs/1411.2738, 2014.

H. Rubenstein and J. B. Goodenough. Contextual correlates of synonymy. *Commun. ACM*, 8(10):627–633, October 1965.

S. Ruder. A survey of cross-lingual embedding models. *ArXiv e-prints*, June 2017.

M. Sahlgren. *The Word-Space Model: Using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces*. PhD thesis, Institutionen för lingvistik, 2006.

A. Salah and M. Nadif. Model-based von mises-fisher co-clustering with a conscience. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 246–254. SIAM, 2017.

G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

G. Salton and C.-S. Yang. On the specification of term values in automatic indexing. *Journal of documentation*, 29(4):351–372, 1973.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

H. Schütze. Word space. In *Advances in Neural Information Processing Systems 5*, pages 895–902. Morgan Kaufmann, 1993.

Y. Shen, P.-S. Huang, J. Gao, and W. Chen. Reasonet: Learning to stop reading in machine comprehension. In *ACM SIGKDD*, pages 1047–1055, 2017.

B. Shi, W. Lam, S. Jameel, S. Schockaert, and K. P. Lai. Jointly Learning Word Embeddings and Latent Topics. *ArXiv e-prints*, June 2017.

H. Shinnou and M. Sasaki. Ensemble document clustering using weighted hypergraph generated by nmf. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 77–80, 2007.

P. Singh Bhatia, S. Iovleff, and G. Govaert. blockcluster: An R package for model-based co-clustering, 2017.

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. 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, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

G. Song, Y. Ye, X. Du, X. Huang, and S. Bie. Short text classification: A survey. *Journal of Multimedia*, 9(5):635–643, 2014.

V. K. R. Sridhar. Unsupervised topic modeling for short texts using distributed representations of words. In *Proceedings of NAACL-HLT*, pages 192–200, 2015.

D. Steinley. Properties of the hubert-arable adjusted rand index. *Psychological methods*, 9(3):386, 2004.

A. Strehl and J. Ghosh. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research*, 3:583–617, 2003.

A. Strehl and J. Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3:583–617, 2002.

A. Tanay, R. Sharan, and R. Shamir. Biclustering algorithms: A survey. In S. Aluru, editor, *Handbook of Computational Molecular Biology*. Chapman and Hall/CRC, 2005.

P. D. Turney and P. Pantel. From frequency to meaning: Vector space models of semantics. *J. Artif. Int. Res.*, 37(1):141–188, January 2010.

S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *CoRR*, abs/1102.1523, 2011.

I. Van Mechelen, H.-H. Bock, and P. De Boeck. Two-mode clustering methods: A structured overview. *Statistical Methods in Medical Research*, 13(5):363–394, 2004.

S. Vega-Pons and J. Ruiz-Shulcloper. A survey of clustering ensemble algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(03):337–372, 2011.

D. Wang, S. Deng, S. Liu, and G. Xu. Improving music recommendation using distributed representation. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 125–126, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.

J. Xie, R. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487, 2016.

D. Xu, W. Cheng, B. Zong, J. Ni, D. Song, W. Yu, Y. Chen, H. Chen, and X. Zhang. Deep co-clustering. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 414–422. SIAM, 2019.

G. Xu, Y. Zong, P. Dolog, and Y. Zhang. Co-clustering analysis of weblogs using bipartite spectral projection approach. In *Knowledge-Based and Intelligent Information and Engineering Systems*, pages 398–407. Springer-Verlag, 2010.

X. Yan, J. Guo, Y. Lan, and X. Cheng. A biterm topic model for short texts. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1445–1456. ACM, 2013.

X. Yang, S. Ying, W. Yu, R. Zhang, and Z. Zhang. Enhancing topic modeling on short texts with crowdsourcing. In R. J. Durrant and K.-E. Kim, editors, *Proceedings of The 8th Asian Conference on Machine Learning*, volume 63 of *Proceedings of Machine Learning Research*, pages 33–48, The University of Waikato, Hamilton, New Zealand, 16–18 Nov 2016. PMLR.

Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

J. Yin and J. Wang. A dirichlet multinomial mixture model-based approach for short text clustering. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 233–242, New York, NY, USA, 2014. ACM.

Y. Zhu, F. Javed, and O. Ozturk. Semantic similarity strategies for job title classification. *CoRR*, abs/1609.06268, 2016.

Y. Zuo, J. Wu, H. Zhang, H. Lin, F. Wang, K. Xu, and H. Xiong. Topic modeling of short texts: A pseudo-document view. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2105–2114. ACM, 2016.