A study of similarity measures for natural language processing as applied to candidate-project matching

Matthew Perreault-Jenkins

Department of Electrical & Computer Engineering

McGill University

Montreal, Canada

May, 2020

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Engineering.

Preface

This thesis contains original work done by the author under the supervision of Professor Ioannis Psaromiligkos.

Abstract

Natural language processing is a discipline rooted in both linguistics and computer science. It incorporates syntactic problems (grammatical category, word segmentation, name entity recognition, etc.), semantics (sentiments analysis, texts categorization, translation, questions answering, etc.), vocal signals generation from texts or texts generation from vocal signals, etc. In the last few years, some scientists and companies have been able to create algorithms capable of achieving very high levels of performance for some of these tasks such as translation or sentiment classification, in part, by using big data. The fact that some algorithms perform so well with such a large amount of data gives a significant business advantage to large companies with large databases over smaller businesses or start-ups. The purpose of this thesis is to find algorithms or methods that can be effective in solving some natural language processing problems on small databases. For our research, we built a content-based recommendation system. We tested similarity measures such as Latent Dirichlet Allocation, cosine similarity, long-short term memory neural network, and the RV coefficient. We also compared the efficiency of the term frequency-inverse document frequency versus the mutual information to give a weighting scheme for the cosine similarity. We also compared the effectiveness of mutual information versus using raw word count as thresholds to remove words from a dictionary for the other similarity measures. We also used external databases, one containing documents related to our problem and another having Wikipedia documents. We also used a pre-trained GLOVE word embedding vector for our neural networks and the RV coefficient. We concluded that the simplest algorithms generally work best when

there is little data. We also proposed several possible solutions to improve the algorithms we tested.

Abrégé

Le traitement du langage naturel est une discipline ayant racine autant dans la linguistique que dans les sciences informatiques. Elle incorpore un bon nombre de problèmes syntaxiques (catégorie grammaticale, segmentation de mots, détection de noms propres, etc), sémantiques (classification de sentiments, catégorisation de textes, traduction, répondre à des questions, etc), transformation de signaux vocaux en textes ou textes en signaux vocaux, etc. Dans les dernières années certains scientifiques et compagnies ont été capable de créer des algorithmes capable d'atteindre de très haut niveaux de performances pour certaines de ces tâches tel que la traduction ou la classification de sentiment, en partie, grâce aux mégadonnées. Le fait que certains algorithmes performent aussi bien avec une aussi grande quantité de données donne un avantage commercial significatif aux grandes compagnies ayant de grande base de données sur les plus petits commerçants ou les entreprises en démarrages. Le but de cette thèse est du trouvé des algorithmes ou méthodes qui peuvent être efficaces pour résoudre certains problèmes de traitement du langage naturel sur de petites base de données. Pour notre recherche, nous avons construit un système de recommandation à base de contenus. Nous avons testé des mesures de similarités tel que le Latent Dirichlet Allocation, la similarité du cosinus, un réseau de neurones profond long-short term memory, et le coefficient RV. Nous avons aussi comparé l'efficacité du term frequency-inverse document frequency versus l'information mutuelle pour donnée un poids aux mots pour la similarité du cosinus. Nous avons aussi comparé l'efficacité de l'information mutuelle versus utilisé le nombre de mots comme seuil d'épuration de dictionnaires dans l'application des autres mesures

de similarités. Nous avons aussi utilisé des bases de données externes, une contenant des documents reliés à notre problème et une autre ayant des documents de wikipedia. Nous avons aussi utilisé un vecteur de prolongement de mot GLOVE pré entraîné pour notre réseaux de neurones et le coefficient RV. Nous avons conclu que les algorithmes les plus simple fonctionne généralement mieux lorsqu'il y a peu de données. Nous avons aussi proposé plusieurs piste de solution pour amélioré les algorithmes que nous avons testés.

Acknowledgements

I would like to tank the natural sciences and engineering research council of Canada (NSERC) who provided the fund that was required to pursue this study trough the Engage grant program.

I would like to thank my supervisor Dr. Ioannis Psaromiligkos for his support during my studies at McGill University. He's mentoring have been a great source of motivation and he always gave me excellent constructive criticism. His guidance has been a key factor in the success of this project.

I would like to thank Fleexer who gave me the opportunity to contribute to their recommender system during my master's degree. Alexandre Laberge, the president and co-fonder of Fleexer, was always available for me and always did what he could to provide me all the resources that I needed to complete my work.

I would like to thank the City of Quebec, Longueuil and the Canadian Ministry of public service who accepted to share their data on their website with us. Without these resources, this work would have been very different.

I would like to thank Sainte-Justine Hospital, they provided exceptional medical care to my newborn son Adriam who was born at only 25 weeks of gestation. He was hospitalized for five months. Their state of the art facilities, devoted care and professionalism

made this experience much more bearable. Today, my son is a very healthy toddler who walks, smile, giggle and make every single senior citizen turn their head.

I would like to thank my girlfriend Ana-Lidia who supported me during my Master's Degree. We had a difficult time when our son was born, we supported each other during those times. This hardship was hard on our relationship but today it made us and our love stronger. She is now carrying our second child, a girl.

I would like to thank my Mother and Father, they supported me through all my life. At an early age, they interested me in science especially in mechanical and electrical engineering. They invested lots of time and money in my education and for that, I am forever grateful.

Table of Contents

List of Acronyms

ML	Machine Learning
TF-IDF	Term Frequency-Inverse Document Frequency
KNN	K-Nearest Neighbour
CBOW	Continuous Bag Of Words
CF	Collaborative filtering
CB	Content base
TP	True Positive
TF	True Negative
FP	False Positive
FN	False Negative
Doc2Vec	Documents to Vector
GLOVE	Global Vector
LDA	Latent Dirichlet Allocation
DNN	Deep Neural Network
RNN	Recurent Neural Network
NLP	Natural Language Processing
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
MLP	Multi-Layered Perceptron
TL	Transfer Learning
MSE	Mean Square Error

Chapter 1

Introduction

The coming of information technologies has changed our societies, economies and social interactions. We now have web platforms that can help us to find a job or a romantic partner, buy used or new objects, etc. Those platforms can have massive catalogs, and it can be difficult for a user to navigate through so much data. For instance, the job recruiting website indeed.com claims to add 9.8 jobs every second to its global catalog. A user could not possibly read every new job opening to find the best job for them. As another example, according to Statistica.com [33], Facebook had 2.27 billion users during the second quarter of 2018. Facebook cannot manually select which ad will be shown to which user. In those cases, Machine Learning (ML) can help users and platforms to make sense of all the data and facilitate the recommendation or search process. An algorithm that can produce automatic recommendations could be used to propose jobs to applicants, items to users, etc. Some platforms such as Amazon, LinkedIn or Google have acquired significant amounts of data over the years. Using that much data, it is possible to create an efficient and accurate matching algorithm for various tasks such as: target advertising, job recommendation, automatic text translation, social recommendation. Some ML algorithms, such as deep learning neural networks (DLNN), perform extremely well given vast amounts of data. This gives a powerful competitive advantage to large internet companies. Unfortunately, most of those algorithms do not perform

very well with smaller databases.

Most companies, organizations or government entities do not have such large amounts of data; therefore, there is a need for ML algorithms that can be trained and perform well using small databases. For instance, according to a survey from the firm Clutch [32] published in March 2017, 71% of small businesses in the United States of America own a website. These businesses could benefit from a recommender system to propose items to their customers. There is also a need for chatbots to guide users on a website. These chatbots could help customers with their purchases, explain a public policy to a citizen or converse with citizens to gather data about their concerns. Large government entities might have enough data on their databases to train such algorithms but smaller entities such as small towns might need a new way to train useful algorithms with their limited resources. Other similar applications of ML include recruitment, automated report generation, sentiment analysis for a product or a public policy, etc.

The general objective of this thesis is to find ways to train efficient ML algorithms for natural language processing (NLP) problems. To train a ML algorithm, we need samples of input-output pairs. In general, it is much simpler to train an algorithm when the inputs are the same size across all samples. For instance, images are set to a fixed number of pixels along the width and height of the image. In natural language problems, the sentence length and the number of sentences in a document vary which prohibit or complicate the use of ML algorithms. Another challenge in NLP is to choose appropriate features for the task at hand. The simplest feature is the words themselves. Words can be represented by a one-hot vector or as a word embedded vector. Words can be lemmatized (i.e., converted to their basic form; singular and infinitive) to reduce the size of a dictionary. Properties of words can also be used as features such as part of speech, synonyms, antonyms, the co-occurrence of other words, etc. For instance, the MRC psycholinguistic database [34] claims to have 150,837 English words and 26 different linguistic properties assigned to them on their database's website. The domain of application of NLP algorithms is quite diverse as it can be used for automatic

translation, search engine optimization, part of speech tagging, similarity measurement between documents, question and answer, chatbots, sentiment analysis, etc. However, considering how many feature and application domains there are, relevant papers in the NLP literature for a particular task can be limited.

Our specific objective is to build a recommender system that can provide good recommendations when trained on small textual databases. Our target application is to improve Fleexer.com's applicant-project matching system. Fleexer.com's database contains a list of candidates and project descriptions. The database contains textual data, but is small and contains very few labels, that is, in only a few cases, appropriate applicant-project matches have been manually identified. Our approach is to first perform a comparative study of existing algorithms and then identify ways to improve these algorithms by augmenting the training set using external databases and incorporating information measures of words.

We will train and test several algorithms as a similarity measure for our CB such as: the cosine similarity, the RV coefficient, a long short-term memory (LSTM) deep neural network and latent Dirichlet allocation (LDA). We will use the one-hot vector and the word embedding for word encoding. We will use the term-frequency inverse document frequency (TF-IDF), raw word count and mutual information (MI) as information measure. We will also use three different databases, Fleexer's database, Fleexer's database and other job-related documents gathered from government entities and a corpus made from all previously mentioned databases plus a Wikipedia dataset build by the Westbury Lab [68]. We will test the effect of using the different databases to train the different similarity measures and on both information measures. We will also test the effect that information measure have on each similarity algorithm.

In this thesis, we build a content based recommender system based on natural language processing. The contributions of our work are centered around similarity measures for NLP applications:

- First, we compared how the MI performs against the TF-IDF and raw word count. MI and TF-IDF has been compared as weighting schemes and MI and raw word count has been compared as a word removing schemes.

- Secondly, we tested several similarity measures: LDA, the cosine similarity, the RV coefficient and a LSTM network. We chose four different similarity measures that are very different from each other. Since most NLP research is done on large databases, we did not know how well most algorithms should perform under these circumstances.

- Thirdly, we have investigated the effect of using external databases to improve the performances of recommender systems on small databases. We used an external database that is somewhat relevant to the task and another that contains unrelated documents to the task. We tested the effect of those external databases on information measures and similarity measures. This helped us understand how sensitive this task is on the choice of external databases.

This thesis is organized as follows. In Chapter 2, we present a literature review that focuses on recommender systems and provides descriptions of three mainly used recommender system types: the collaborative filtering (CF), the content based (CB) and the hybrid recommender system. In Chapter 3, we provide some background on natural language processing topics. In this chapter, we discuss performance measures, feature representation, some challenges in NLP, and we define overfitting. Then, we define the information measures that we used in our experiments: MI and TF-IDF. The last section of this chapter is where we define the similarity measures that we use in our experiment, namelt, LDA, the cosine similarity, the RV coefficient and the LSTM network. In Chapter 4, we provide a detailed explanation of our architecture and methodology. In this chapter, we presente in detail our databases, how we used our different information and similarity measures and how we evaluated our results. In Chapter 5, we present and

discuss our results. In chapter 6, we conclude this thesi, sharing what we learned in this work.

Chapter 2

Literature Review

In this section, we will talk about recommender systems. First, we will describe three types of a recommender system: collaborative filtering (CF), content base (CB) and hybrid. We will finish this chapter by presenting several research papers about recommender systems and also some that used external databases to improve their algorithms. Recommender systems are designed to offer recommendations to users by ”matching” them with items (other users, objects, jobs, etc.). Recommender systems can be used by a wide variety of web services (dating websites, marketplaces, job recruiting websites, social networks, etc.). There are three predominant types of algorithms found in the recommender systems literature: CF, CB and hybrid. CF systems work in two steps. First, they find similarities between users and then, they recommend items that similar users liked or bought. They use recorded interactions between users and items (clicks, items rating, purchases, messages send, etc.) to compute similarities. The motivation behind CF is that a user might get better recommendations from users with similar opinion rather than the general rating from all users for an item. Collaborative filtering algorithms are used for movies and television shows recommendation, for examples, Zhou et al. [22] and Koren et al. [23] used them for the Netflix prize challenge [21]. For online shopping recommendations, Lenden et al. [24] used CF for Amazon.com item recommendations. The disadvantage of collaborative filtering is the cold start problem, that

happens when a new item or user is added to the database. Since the algorithm bases its recommendation on user-item interactions, it is impossible to give recommendations to a user or for an item with no history of interactions.

In CB systems, the idea is to match users with items using a similarity measure between them. Mamadou et al. [26] used what they called interactions data from social networks profiles (Facebook.com and LinkedIn.com) to recommend jobs opportunities to users. They define interaction data as every interaction that a user has with the social network, e.g., user's posts, likes, comments, publications, time spent reading a publication, etc. They used a semantic similarity algorithm to compute how similar a job's description and the textual interaction data of a user profile are, and then based their recommendation on this measurement of similarity. Using only textual data, they were able to make predictions using an unsupervised similarity algorithm, cosine similarity, and a supervised one, support vector machine (SVM). The unsupervised learning approach does not use any labeled data; in contrast, in the supervised learning approach, the labels are used during training. Mooney et al. [27] used a slightly different approach for book recommendations. They used textual data about books such as: title, authors, synopses, published reviews, costumer comments and more. Then, they asked new users to rate books. Finally, they recommended a book based on a similarity algorithm between users using book's textual data. In content-based recommendation, there is no cold start problem since the algorithm bases its recommendation on data in the user's or item's profile instead of interactions.

There is also the possibility to combine the CF and CB approaches to create hybrid recommender systems which take advantage of the knowledge of other users experience and are less affected by the cold start problem. A hybrid recommender system was proposed by Lu et al. [28] for a job recruiting website. They achieve better results than their CF and CB algorithms individually. Ghazanfar et al. [29] also used a hybrid recommender system for a general purpose recommendation task. The CB part uses a naive Bayes classifier as a similarity measurement. For the CF part, they used the cosine simi-

larity between items that the users liked and the other items in the database multiplied by the rating of other users. Finally, an algorithm determines the confidence level of the CB. If it is high enough, it will recommend the items to the users without using the CF parts. If the confidence level of the CB is not high enough, the algorithm will combine the prediction from the CB and the CF part. Ghazanfar et al. reported better results than using their CB or CF individually, a user-based CF, the algorithm IDemo4 [79] and the content-boosted algorithm proposed in [80].

Most recommender systems are trained and tested using huge amounts of data. Najafabadi et al. [47] did a survey of over 131 recommender system papers between 2000 and 2016. These papers used public databases such as: MovieLens 100k [51] (100k movie ratings), MovieLens 1M [51] (1 million movie ratings), MovieLens 10M [51] (10 millions movie ratings), Netflix [21] (100 millions television and movie ratings), Jester [52] (4.1 millions joke ratings), BookCrossing [53] (1.1 millions book ratings), delicious [54] (104 833 recipe ratings), and the MSD [55] (8 598 630 music track - tag pairs). As we can see, most published methods use a lot of data, MovieLens 100k being the smallest, which may indicate a lack in interest in recommender systems using small databases. However, some papers have been published on using transfer learning for recommender systems. The motivation and basic concept behind transfer learning is to improve an algorithm when the dataset of the target task is small, by "transferring" what the algorithm has learned from a larger but similar (in some sense) dataste. Zhao et al. [48] developed a framework to train a recommender system using a large database by mapping items to the other, smaller, database. Specifically, they used the Netflix database to train a CF recommender system for Douban which is a Chinese recommendation website for books, movies and music (however, they didn't use the music data in their experiment). They achieved better results than using only Douban database's. The limitation of their algorithm is the need for both databases to be very similar since they are using entity-correspondence mappings (items with the same titles on Netflix and Douban) between both systems. Zhang et al. [49] also used information from external databases to im-

prove recommender systems. They developed a five steps framework to extract useful data from one domain to another. For instance, they claim that they can use books ratings to improve a recommender system that proposes movies.

After looking at what we found in the literature, we concluded that the reported research on recommender systems that use small datasets is very sparse. The few papers that we found that used external datasets to improve their algorithm on a task with a smaller dataset used external datasets that are very similar to the smaller dataset's target task. There has been no study, and therefore, we cannot have any expectation of the effect of using unrelated data on a recommender system for a small dataset. Since a small dataset might have few labels the cold start problem of a CF algorithm might be a big issue. For this reason, for this thesis we decided to build a CB recommender system.

Chapter 3

Methodology: Mathematical Background

In this chapter, we will discuss several topics related to NLP. We will first define several performance measures that are typically used in machine learning. Then, we will discuss feature representations that are used in NLP such as: word embedding, one-hot vector and others. We will then talk about the challenges and ambiguities frequently encountered in NLP tasks.

After discussing overfitting, we will address the topic of information measurement and explain the terms frequency-inverse document frequency and mutual information. Then, since we aim to build a content-based recommender system, we will talk about several similarity measures that we can use for this task such as: Latent Dirichlet allocation, cosine, Jaccard, Sorensen-Dice, RV coefficient and long-short-term memory (LSTM) deep neural network (DNN).

3.1 Performance Measures

In machine learning, we need metrics to evaluate the performance of an algorithm. In the binary classification (positive/negative) problem, such as identifying in which one of two classes an example belongs, several metrics are often used such as the accuracy, the precision, the recall and the F1-score [81]. If what we want to measure is not binary,

e.g., when we want to predict a real number such as temperature forecast or stock price prediction, the Pearson correlation coefficient and the Spearman correlation coefficient can be used to compare the output of an algorithm to the ground truth.

In binary classifier evaluation, we first need to define four terms: true positive (TP), true negative (TN), false positive (FP) and false-negative (FN). TP is the number of examples that the algorithm correctly identified as positive. TN is the number of examples that the algorithm correctly identified as negative. FP is the number of examples that the algorithm falsely identified as positive. Finally, FN is the number of examples that the algorithm falsely identified as negative. Using these terms we can define the following performance metrics: The accuracy which is the ratio of correctly identified examples over the total number of examples or $\frac{TP+TN}{TP+TN+FP+FN}$, the precision which is the ratio of correctly identified positive examples over the total number of predicted positive example or $\frac{TP}{TP+FP}$, and the recall is the ratio of correctly identified positive examples over the total number of true positive examples or $\frac{TP}{TP+FN}$. Finally, the F1-score is the harmonic mean of the recall and the precision or $2\frac{Precision \times Recall}{Precision+Recall}$.

The Pearson correlation coefficient is a measure of how well two vectors, say Y and $\tilde{Y}$, linearly correlate to each other. It is defined as the ratio of the covariance between two vectors over the product of their standard variation

$$\rho_p = \frac{cov(Y, \tilde{Y})}{\sigma_Y \sigma_{\tilde{Y}}}. \tag{3.1}$$

with the covariance defined as:

$$cov(Y, \tilde{Y}) = E[(Y - E[Y])(\tilde{Y} - E[\tilde{Y}])] = \frac{1}{N}\sum_{i=1}^{N}(y_i - E[Y])(\tilde{y}_i - E[\tilde{Y}]) \tag{3.2}$$

and the standard variation defined as:

$$\sigma_\nu = \sqrt{E[\nu^2] - [E[\nu]]^2} = \sqrt{\sum_{i=1}^{N}(\nu_i - E[\nu])^2} \tag{3.3}$$

for vectors of lenght of N and the expected value of a vector E being the mean of the vector.

The Spearman correlation coefficient, first described by Charles Spearman [75], is also a measurement of how well two vectors correlate to each other for any monotonic relation between two vectors Y and $\tilde{Y}$. The difference between the Pearson and Spearman correlation coefficient is that the Spearman coefficient uses a ranks vector rather than the values of the vectors. For instance, in a vector of N values, transforming it to a rank vector will result in the smallest value to have a rank of 1 and the biggest value to have a rank of N. If we wanted to transform the vector $Y = \{1.2, 5, -1, 2\}$ into a rank vector, it will become $Rank_Y = \{2, 4, 1, 3\}$. The Spearman correlation coefficient is defined by $\rho_s = \frac{cov(Rank_Y, Rank_{\tilde{Y}})}{\sigma_{Rank_Y}\sigma_{Rank_{\tilde{Y}}}}$.

As a performance measure, the Pearson and Spearman coefficients are used with the predicted values $\tilde{Y}$ and the ground truth (or labels) Y. Suppose that we have a ground truth vector Y and we want to test two algorithms that try to predict it by producing the output vectors $\tilde{Y}_1$ and $\tilde{Y}_2$, respectively, which can be real numbers or integers. Then we can find the Spearman correlation coefficient ρ_{s1} and ρ_{s2} between the two predictions $\tilde{Y}_1$ and $\tilde{Y}_2$ and the ground truth Y. When the values of ρ_{s1} and ρ_{s2} are similar, we need to do a statistical significance analysis to determine how confident we are that one algorithm is better than the other. According to Myers et al. [78] we first need to compute the Fisher's Z-transformation using:

$$Z_r = 0.5 \ln \frac{1+\rho_s}{1-\rho_s} = \operatorname{arctanh}(\rho_s) \tag{3.4}$$

Then, using eq. (3.4) on the two results that we want to compare, we can find the z-score:

$$z = \frac{Zr_1 - Zr_2}{\sqrt{(N_1 - 3)^{-1} + (N_2 - 3)^{-1}}} \tag{3.5}$$

where Zr_1, Zr_2, N_1 and N_2 are respectively the Fisher's Z-transformation from ρ_{s1} and ρ_{s2} and the size of $\tilde{Y}_1$ and $\tilde{Y}_2$. Then, with the z-score, we can do a simple statistical test

to know the significance of the result or in another words, the probability that the first algorithm produces a more accurate prediction then the second:

$$P(\rho_{s1} > \rho_{s2}) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) dx \tag{3.6}$$

3.2 Feature Representation

As mentioned in the introduction, feature representation is a challenge in NLP problems. In most NLP algorithms, there are two ways to represent words: one-hot vectors / bag of words [82] or word embeddings [16] [18]. A one-hot vector is a vector of length equal to the vocabulary size where a word is represented by a one in its assigned position. For instance, if the word *cat* is the 100th word in a 1000-word vocabulary, the one-hot vector that will represent *cat* will be a 1000-word long vector where the value at the index 100 is one while the other 999 values of the vector are zeros. This encoding is very simple to implement and does not require any particular knowledge about words but it also has several limitations. Its main weakness is the sparsity of the vector which can cause computational problems and inefficiency. Also, adding new words to the vocabulary will change the length of the feature vector which might be problematic for certain algorithms. If we want to represent a document, we simply sum up all the one-hot vectors representing each word in the document and the resulting vector is what is called a bag of words vector.

Another problem with one-hot vector encoding is that the length of the vector can be very large. In NLP problems, the vocabulary can be very large, thus the need for a method to reduce it. One popular such method is word lemmatization [83] which is the removal of plurals and the deconjugation of verbs. It can be advantageous for some problems such as classification since words in their singular or plural form keep the same meaning. On the other hand, for automatic text translation, it could be useful to keep the verbs' tense and the plural form since this information should appear in the

final result. Lemmatization is a complicated NLP problem in itself. In most cases, simple rules can be used. Examples of possible rules are removing the apostrophe at the end of a word, replacing suffixes such as *sses* by *ss*, *ies* by y, or removing suffixes such as *ed* or *s*. A rule like replacing *ies* by y would eventually fail for plural words where their singular form ends with *ie* such as the word *lie*. Irregular verbs will also need special rules such as replacing *am*, *are*, *is* by *be*.

In the case of word embedding, the goal is to encode each word by a vector such that the distance between two vectors represents a semantic or syntactic relation between these words. According to Mikolov et al. [16], their word embedding algorithm, the continuous bag of words (CBOW), can capture semantic similarity in a way where a similar relationship between words will result in a similar vector encoding. Word2vec [18], created by Mikolov et al., is also a very popular word embedded algorithm. Their methods overcome the limitation mentioned above in the one-hot vector approach. On the other hand, to capture enough information, those algorithms usually need a lot of data and a lot of computation to be properly trained.

In NLP problems, there are more than just words that can be used as features. Bojanowski et al. [17] additionally used the internal structure of words and trained an embedded vector for words and sub-words. For each word, they added a $\langle end \rangle$ character at the beginning and at the end, and then they created sub-words of a certain number of letters. For instance, for sub-words of 3 letters, the word *where* becomes $\langle end \rangle wh$ + *whe* + *her* + *ere* + $re\langle end \rangle$ and the special sequence *where*. They then created vector representation for each word and its sub-words using the word2vec training algorithm [18]. According to Bojanowski and his team, this model allows for better representation of rare words than other baseline methods such as CBOW [2] and word2vec [18] in a similarity judgment task on the English Rare Words database [59]. On the other hand, they achieve worse results than word2vec on the words analogy task on a frequently used words dataset [58]. They also reported better results than their baseline on words analogy tasks in French, English, Spanish and German.

It is also possible to use dictionaries to add features to an NLP problem. Dictionaries can be used to find synonyms, antonyms, part of speech tags (noun, verb, adverb), etc. For instance, Sultan et al. [12] used the Paraphrase Database (PPDB) to find similar words to improve a longest common subsequence (LCSS) algorithm in a sentence similarity task. LCSS is an algorithm that finds the longest subsequence that is present in two sequences; the length of the longest subsequence is indicative of how similar the two sequences are. Instead of only using a subsequence that matches perfectly with both sequences, Sultan's LCSS uses the longest subsequence of similar words between two sentences. Gao et al. [15] used the Wordnet's [50] taxonomy database to compute similarities between pairs of words. They reported achieving a Pearson correlation coefficient of 0.885 in a word similarity task on the RG dataset [60].

3.3 Challenges in NLP

In NLP, there are different cases of ambiguity that can appear. Some examples are described next. One of the most common causes of ambiguity are homographs [84] or words with multiple possible meanings. For instance, the word *grenade* can be a fruit or a weapon. A word can also have multiple meanings and part-of-speech ambiguity; for instance, the word *ski* can be a noun, e.g., "I got a pair of *skis*", or a verb, e.g., "I want to *ski*." In sentiment analysis [86], some words can be more significant since they carry more information about the general sentiment in a sentence. Such words can be adjectives like *good*, *bad*, *beautiful* and *ugly*. Those words, based on the circumstances and context, can carry an ambiguous connotation. For instance, in the phrases "the ice cream was *cold*" and "the waiter was *cold*", we can see that the adjective *cold* can be used to describe the state of ice cream as well as saying that the waiter was unpleasant. Capturing the semantics of a joke can be really difficult since jokes tend to talk about something that is incoherent without telling it explicitly. Sarcasm [85] is another challenge in NLP since the author says literally the opposite as what is meant.

3.4 Overfitting

The overfitting problem occurs when the decision boundaries ”fit” too closely to the training data. In machine learning, for a classification problem, the decision boundaries are hyperplanes: classification decisions are made depending on which side of decision boundaries a data point falls on and overfitting will cause the classifier to not create good boundaries. In a function approximation problem, overfitting will cause the approximation to worsen when increasing the complexity of the approximation function. For a function approximation example, see Figure 3.1. All 6 data points were created using the function $y = x + \epsilon$ where ϵ is Gaussian noise. We approximated the original linear model using 5 different degrees of polynomial regression, linear to fifth degree. We can see that the linear, second-degree and third-degree polynomial regression estimations of the original function were not too far off but that the fifth-degree polynomial regression, even though it perfectly fits the original data points, is a really bad approximation of the original function.

In supervised machine learning, overfitting occurs when the loss function of a model during training is getting lower while the performance decreases when testing with new samples (validation phase). This happens when the ratio of the number of parameters over the number of labeled samples is high. Gordon F. Hugues [77] noted the existence of an optimal complexity (number of parameters) for a binary classifier. For instance, in a binary image classification task using a Bayesian classifier, using $1,000$ samples, the optimal number of parameters was 23 and for 100 samples, the optimal number of parameters was 8.

3.5 Definitions

First, let us define some symbols that we will use throughout the thesis.

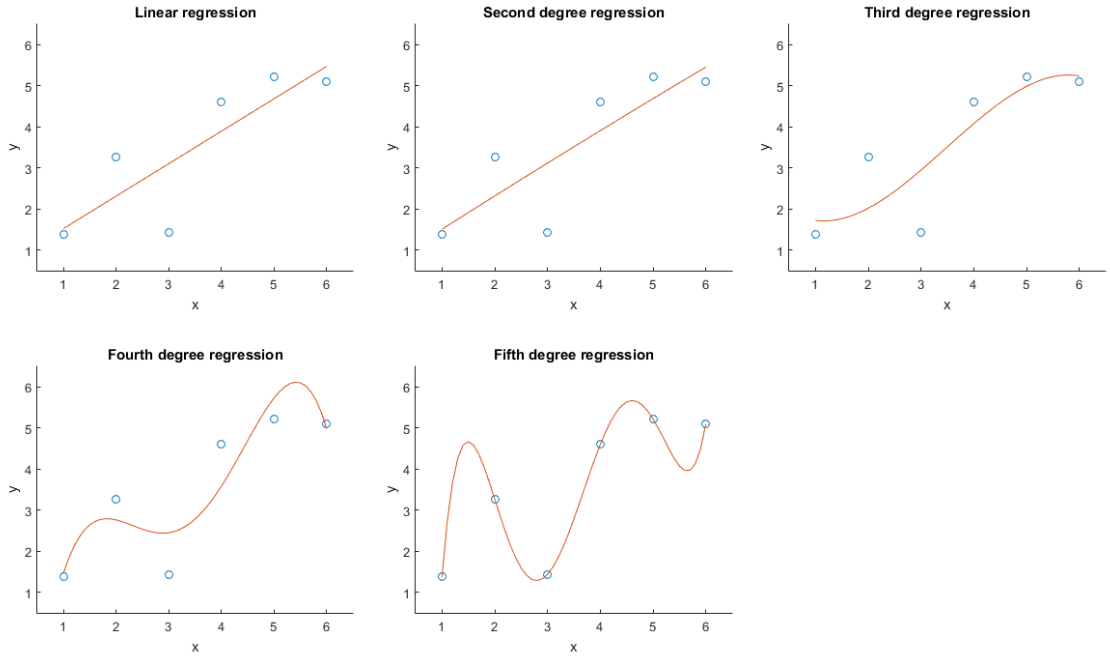

Figure 3.1: Overfitting example. Six data points (shown by circles) were created using the function $y = x + \epsilon$ where ϵ is Gaussian noise. The curves on the graphs were obtained by polynomial regression of degree 1 (top left graph) to 5 (bottom center graph).

$S \in \mathbb{N}$	number of words in a dictionary
$w \in \mathbb{N}$	index of a word from 1 to S
$N \in \mathbb{N}^M$	number of words for each document
$M \in \mathbb{N}$	number of documents in a corpus
$d_m \subseteq D$	document m in the corpus D
$V \in \mathbb{R}^{S \times K}$	set of K-dimensional vector representation for all words
$v_w \in V$	K-dimensional vector representation of word w in the set V

3.6 Word Embedding

"You shall know a word by the company it keeps" John Rupert Firth, 1957.

Word embedding is a way to map words to vectors. The idea is that words that are commonly found near each other must have some kind of semantic or syntactic relation. The goal of word embedding algorithms is to capture this relation between all words in a corpus and represent it in a vector space. In that vector space, vectors of words that share similar meanings should be clustered together. For instance, food-related words should be close to each other. The distance between two word vectors also carries information about their relationship; for instance, the resulting vector of $(Paris - France)$ should be similar of $(London - England)$ since they share a common relationship: the first word is the capital city of the second word, a country . There are many algorithms that can be used to create a model for word embedding, such as: global matrix factorization [35], skip-gram model [18] or a combination of multiple models [2].

3.6.1 Skip-Gram and Word2Vec

Although the Word2Vec algorithm of Mikolov et al. [18] [37] was originally trained using a skip-gram model it has been shown to be an explicit matrix factorization of the words co-occurrence matrix by Li et al. [36]. A skip-gram model is a model where the objective is to predict values in a series surrounding one known value or more concretely, it models the relation between one value and values surrounding it . In NLP, this means predicting words surrounding one known word or more concretely, word2vec is a modal of the relation between a word and it's surrounding. The known word is called the center word and the surrounding words are called the context window. This model supposes a softmax distribution of the context window knowing the centered word and is defined by:

$$\prod_{w \in D} P(l|w, \tilde{V}, V) \tag{3.7}$$

where:

$$P(l|w, \tilde{V}, V) = \frac{\exp(\tilde{v}_l^\intercal v_w)}{\sum_{l'=1}^{L} \exp(\tilde{v}_{l'}^\intercal v_w)} \tag{3.8}$$

where l is the identification of an individual context window, L is the number of possible context windows, w is the identity of a center word, $\tilde{v}_l \in \tilde{V}$ and $v_w \in V$ are respectively vector representations for l and w and $\tilde{V}$ is the set of vector for every possible context windows.

Maximizing the probability distribution in eq. (3.7) with respect to the sets of vectors V and $\tilde{V}$ bring word vectors with similar centered word close to each other. However, this approach is impractical since it requires computing the term $\exp(\tilde{v}_{l'}^{\mathsf{T}} v_w)$ for every context window and every word in the dataset or corpus which can be extremely computationally expensive. For instance, if the context window is composed of 4 words around the centered word and the vocabulary contains 10000 words, the number of possible context windows would be equal to $L = 10000^4 = 10^{16}$. This is why Mikolov et al., used a much more efficient negative sampling approach. Instead of maximizing the probability of every word over every context window, the objective is now to maximize the probability that every pair of a centered word and a context window (w, l) are observed in the corpus D. The objective function is:

$$\arg\max_{\tilde{V},V} \prod_{(w,l)\in E} P((w,l) \in D|\tilde{V}, V) = \arg\max_{\tilde{V},V} \prod_{(w,l)\in E} \frac{1}{1 + e^{-\tilde{v}_l^{\mathsf{T}} v_w}} \tag{3.9}$$

where E is the set of all pairs of center words and context windows observed in corpus D. We can see that a trivial solution exists for the problem in eq. (3.9) obtained by setting $\tilde{v}_l = v_w$ and $\tilde{v}_l^{\mathsf{T}} v_w$ to a large enough value for every l and w. According to Goldberg et al. [37], setting $\tilde{v}_l^{\mathsf{T}} v_w$ to a value greater or equal to 40 result of a probability very close to 1 in eq. (3.9). There is also the problem that maximizing the probability distribution in eq. (3.9) will make every vector in V converge to a similar value. . To prevent those issues, Mikolov used a new set E' of pairs of centered word and context windows that do not belong in the corpus D, which is where the term negative sampling comes from.

The new objective function is now:

$$\arg\max_{\tilde{V},V} \prod_{(w,l)\in E} P((w,l) \in D|\tilde{V},V) \prod_{(w,l)\in E'} P((w,l) \notin D|\tilde{V},V) \tag{3.10}$$

By considering the log of both sides of eq. (3.10) the optimization problem becomes:

$$\arg\max_{\tilde{V},V} \sum_{(w,l)\in E} \log \frac{1}{1+e^{-\tilde{v}_l^\intercal v_w}} + \sum_{(w,l)\in E'} \log \frac{1}{1+e^{\tilde{v}_l^\intercal v_w}} \tag{3.11}$$

By optimizing eq. (3.11), we make good (or existing in E) pairs of context windows and centered words vector representation scalar product $\tilde{v}_l^\intercal v_w$ high and inversely, pairs drew from E' will make this scalar product of their vector representation small. According to Goldberg and Levy [37] this means that words that share many context windows will have similar word vectors. This affirmation makes intuitive sense considering that group of words that are often seen together may not have similar meanings but rather add complementary information to the group such as in: deep learning, Canadian maple syrup, special Olympics.

3.6.2 Glove

The global vector for word representation (Glove) [2] is a very popular word embedding algorithm created by Jeffrey Pennington and his team at Stanford University. Glove combines a context window-based method such as word2vec and global matrix factorization. They define a matrix X for co-occurring words whose (i,j)th element, X_{ij}, is the number of times the word w_j occurs in a context window around the centered word w_i, and $X_i = \sum_k X_{ik}$ is the number of times w_i appear in the corpus. The probability of seeing a word in a context window is $P_{ij} = P(j|i) = X_{ij}/X_i$. The context window is composed of words that appear either before, after or around a center word and can be of variable length or fixed in size. First, they suppose that the relationship between two words w_i and w_j can be quantified by using the ratio of co-occurrence of a "probe" word

w_k. Using this intuition, their starting point was to define a function F which took the form of:

$$F(v_{w_i}, v_{w_j}, \tilde{v}_{w_k}) = \frac{P_{ik}}{P_{jk}} \tag{3.12}$$

where $v_w \in V$ is a vector representation of word w when w is a centered word and $\tilde{v}_w \in \tilde{V}$ is a vector representation of word w when w is in the context window. In their paper, they give an example using w_i = ice, w_j = steam and using probe words w_k: solid, gas, water and fashion. They demonstrated that words that are related to ice but not steam such as solid would have a very high co-occurrence ratio, words that are related to steam but not at ice would have a small co-occurrence ratio and words that are related to both word like water or to none of then like fashion would have a co-occurrence ratio close to 1.

They defined a least square cost function to minimize by optimizing both set of vector v and $\tilde{v}$ and biases b and $\tilde{b}$ as:

$$J = \sum_{i=1}^{S} \sum_{j=1}^{S} f(X_{ij})(v_{w_i}^{\intercal} \tilde{v}_{w_j} + b_{w_i} + \tilde{b}_{w_j} - \log(X_{ij}))^2 \tag{3.13}$$

with:

$$f(X_{ij}) = \begin{cases} (\frac{X_{ij}}{x_{\max}})^{\alpha} & if\ X_{ij} < X_{\max} \\ 1 & otherwise \end{cases} \tag{3.14}$$

where $b_{w_i} \in B$ and $\tilde{b}_{w_j} \in \tilde{B}$ are biases respectively associated with the centered word and a word on a context window. The general idea of eq. (3.13) is very similar to the word2vec eq. (3.11), the more often a center word w_i is close to a word w_j inside the context window, the more the term $v_{w_i}^{\intercal} \tilde{v}_{w_j}$ is going to be large, therefore, if the center words w_{ii} and w_i have similar word concurrence, their vector representation will converge to similar values. The goal of the weighting function $f(X_{ij})$ is to limit the weight of very frequent co-occurrence by choosing $x_{\max}$ arbitrarily,they chooses to fix this number to 100. Choosing an α parameter smaller than one put more weight on rare co-occurrence,

they reported better result using $\alpha = 3/4$ than $\alpha = 1$.

They reported better results than word2vec and other baselines in word similarity tasks on dataset such as WordSim353 [72], MC [73] and The Stanford Rare Word [59] and on the name recognition dataset CoNLL-2003 [74]. They also demonstrate that they can easily train on a large corpus of 42 billion words.

3.7 Information Measurement

Information measurement is useful in semantic similarity tasks since it can tell us how important a word is in a sentence. We can use this information either in a weighting scheme or to choose which words are relevant enough to be added in a dictionary. For instance, we used information measures as a weighting scheme in the cosine similarity where we simply multiplied each word represented in a bag of words vector with their associate weight. This is done before using the cosine similarity between the bag of words of the documents that we want to compare. Next, we describe two of the most commonly used information measures, namely, Term Frequency - Inverse Document Frequency (TF-IDF) and Mutual Information (MI).

3.7.1 Term Frequency - Inverse Document Frequency (TF-IDF)

TF-IDF is composed of two parts, the term frequency (TF) which is a by-document term and the inverse document frequency (IDF) which is a per-corpus term. TF-IDF is defined by $TF\text{-}IDF(w, d, D) = TF(f_{w,d})IDF(f_{w,D})$ where $f_{w,d}$ is the frequency or count of the word w in a document d and $f_{w,D}$ is the frequency of the word w in a corpus D. There is a wide variety of TF and IDF schemes that can be used together.

Some TF schemes include:

binary	1 if the term w exists in d , 0 otherwise	(3.15)
raw term frequency	$f_{w,d}$	(3.16)
term frequency	$\frac{f_{w,d}}{\sum_{w' \in W} f_{w',d}}$	(3.17)
augmented normalized term frequency	$0.5 + 0.5\frac{f_{w,d}}{\underset{w' \in W}{max}(f_{w',d})}$	(3.18)
log term frequency	$\log(1 + f_{w,d})$	(3.19)

Some IDF schemes include:

unary	1	(3.20)
inverse document frequency	$\log(\frac{M}{f_{w,D}})$	(3.21)
probabilistic inverse frequency	$\log(\frac{M - f_{w,D}}{f_{w,D}})$	(3.22)

where M is the number of document in the corpus and W is the set of all word in the corpus.

Salton & Buckley [30] did a comparative study comparing several TF and IDF measures as weighting schemes on five datasets of automatic text retrieval systems, in another word, they tried to find documents using textual queries. . They reported better precision using raw term frequency or augmented normalized term frequency for TF and inverse document frequency or probabilistic inverse frequency for IDF.

3.7.2 Mutual Information (MI)

The MI between two discrete random variables X and X' is defined by:

$$I(X;X') = \sum_{x}\sum_{x'} P(x,x') \log(\frac{P(x,x')}{P(x)P(x')}). \tag{3.23}$$

where x and x' are the possible values of X and X'. MI was first described by Claude Shannon [38] in his 1959 paper "Coding theorems for a discrete source with a fidelity criterion." MI is a measure of how much information is contained in a random variable X about another random variable X'. We can see that the term inside the log in Eq. (3.23) should be equal to 1 if both variables are independent which would result in no mutual information at all.

To use MI for text classification as a weighting scheme [31], we can use the MI between the word $X = w_i$ and the random variable X' being the classes. The amount of information in w_i can be computed with:

$$I(w_i) = I(X = w_i; X') = \sum_{x'} P(x,x') \log(\frac{P(x,x')}{P(x)P(x')}). \tag{3.24}$$

MI can also be used in words relatedness tasks [39] with $X = w_i$ and $x' = w_j$. In this case, $MI(w_i, w_j)$ is the probability that the words w_i and w_j are related.

3.8 Semantic Similarities

In this section, we will describe some techniques that can be used to compute how semantically related two documents are. We will talk about the latent Dirichlet allocation, the cosine similarity, the Jaccard distance, the Sorensen-Dice distance, the Rv coefficient and the long short term neural network. Detail of how we used those techniques as similarity measures can be found in the architecture and methodology chapter with the exception of the Jaccard and Sorensen-Dice distances that we did not use.

3.8.1 Latent Dirichlet Allocation (LDA)

LDA is a generative probabilistic model for a collection of discrete data created by Jordan et al. [10] which is very appropriate for modeling text corpora. According to Ng et al. [61], generative models make predictions by learning the joint distribution $P(x, x')$ between the input x and the label x' while discriminative models base their predictions on the posterior probability $P(x'|x)$ or map directly an input x to a label x'. A common example of a generative model would be the naive Bayes classifier and for a discriminative model would be the logistic regression classifier. Since LDA is an unsupervised algorithm and therefore doesn't use labels during the training phase, the joint distribution that we need to compute is between the input and the latent variables.

Before we describe LDA, we will first define a few variables.

K	number of topics
$\alpha \in \mathbb{R}^K$	Dirichlet prior for the topic distribution per document
$\Omega \subseteq \mathbb{N}^M$	sparse bag of words representation for a document
$\omega \in \mathbb{N}^{M \times S}$	dense representation of bag of words for each document
$Z \subseteq \mathbb{N}^M$	latent categorical variable of words in a document
$\beta \in \mathbb{R}^{S \times K}$	probability distribution of every word over each category
$\theta \in \mathbb{R}^{M \times K}$	latent variable, topic distribution of each document

Jordan et al. [10], summarized LDA as a 3-level (corpus, document and word) hierarchical Bayesian model, where α and β are corpus-level parameters, θ is a document-level parameter and Ω, ω and Z are word-level parameters. The general idea of LDA is that each document and each word are associated to a finite mixture of K topics. The model assumes a Dirichlet distribution of those topics θ per documents in D.

LDA is a generative model which means that the assumption of this model is that documents are created according to a set of rules. To generate a new document the length or number of word N is decided based on a Poisson distribution. Then the distribution of topics for this document is chosen using a Dirichlet prior $\theta \sim Dir(\alpha)$. For each word in document d, a topic z is assigned based on the probability distribution in

θ. Finally, a words w is chosen according to $P(w|z,\beta)$ which is a multinomial probability conditioned on the topics in z, or in another word, the probability of a word being chosen in the document is proportional to the probability of that word existing multiplied by the probability of that word being associated with topic z. We can make the assumption that the length of the document d is not critical and that the parameter K is known and fixed. In original LDA paper, the following probability distribution for a document was proposed:

$$P(\theta, z, w|\alpha, \beta) = P(\theta|\alpha) \prod_{n=1}^{N} P(z_{n|}\theta) P(w_n|z_n, \beta) \tag{3.25}$$

where $P(z_n|\theta)$ is the probability of z_n given θ or simply θ_{z_n}. $P(\theta|\alpha)$ is a Dirichlet distribution given by:

$$P(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^{K} \alpha_i)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \theta^{\alpha_1 - 1} ... \theta^{\alpha_K - 1} \tag{3.26}$$

where $\Gamma(\cdot)$ is the gamma function.

We can also visualize the LDA model using the plate notation shown at figure 3.2. In this representation the plate represent sets (set of all documents from 1 to M, set of all words in a document from 1 to N), the circles represent variable and the arrows (or edges) represent the direction of the dependency between the variable.

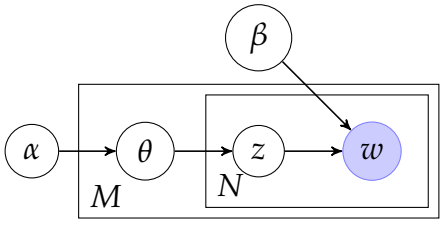

Figure 3.2: Plate Diagram of LDA.

The Dirichlet distribution is convenient to create such a generative model since it is in the exponential family, has finite dimensional, sufficient statistic and is conjugate to the multinomial distribution which facilitates parameters estimation using variational inference according to the original paper [10]. Variational inference is a mathematical

tool that allows the optimization of parameters for a convex probability distribution function p when one or more parameters are impossible or impractical to compute. This is needed in this context since evaluating the latent variables θ and z using Bayesian inference would require to compute:

$$P(\theta, z|w, \alpha, \beta) = \frac{P(\theta, z, w|\alpha, \beta)}{P(w|\alpha, \beta)}. \tag{3.27}$$

Therefore, we would need to marginalize the latent variable in eq. (3.27) using:

$$P(w|\alpha, \beta) = \frac{\Gamma(\sum_{i=1}^{K} \alpha_i)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \int (\prod_{i=1}^{K} \theta^{\alpha_i - 1})(\prod_{n=1}^{M} \sum_{i=1}^{K} \prod_{j=1}^{S} (\theta_i \beta_{ij}) \omega_{nj}) d\theta \tag{3.28}$$

As we can see in eq. (3.28), using vanilla Bayesian inference would be computationally impractical for large S and M.

The idea of variational inference is to find a simpler function q that approximates the function p then minimize the Kullback–Leibler divergence $D(q\|p)$ between q and p with respect to the parameters in the model. To create the function q, the author Jordan et al. [10] proposed to remove the edges $\theta - z$, $z - w$ and $\beta - w$ and the node w due to the problematic of coupling θ and β, see figure 3.2. The new distribution becomes:

$$q(\theta, z|\gamma, \phi) = q(\theta|\gamma) \prod_{n=1}^{M} q(z_n|\phi) \tag{3.29}$$

where $\gamma \in \mathbb{R}^{K \times M}$ is a Dirichlet parameter over the topic distribution for each document and $\phi \subseteq \mathbb{R}^{M}$ is the topic distribution for each word in each document, γ and ϕ are the free variational parameters. Then, we minimize the Kulback-Leibler divergence with respect to the free variational parameters. To minimize $D(q\|p)$, Jordan et al. [10] proposed an expectation-maximization (EM) algorithm. In the E-step, we optimize the free variational parameters γ and ϕ and in the M-step, we optimize α and β. To find the optimal γ and ϕ, we can use the fixed-point method to set the derivative of $D(q\|p)$ to zero. This will give us the pair of update equations that need to be repeated until

convergence below:

$$\phi_{ni} \propto \beta_{iw_n} \exp(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{K} \gamma_j)) \tag{3.30}$$

and

$$\gamma_i = \alpha_i + \sum_{n=1}^{M} \phi_{ni} \tag{3.31}$$

where $\Psi(\cdot)$ is the digamma function or the integral of the log of the gamma function and the $\propto$ sing mean proportional to. The algorithm to estimate both parameter ϕ and γ is:

for $d = 1$ **to** M **do**
  initialize $\phi_{idn} = 1/k$ for all i and n
  initialize $\gamma_{id} = \alpha_i + M/K$ for all i
  while *converging* **do**
    for $n = 1$ **to** N_d **do**
      for $i = 1$ **to** K **do**
        $\phi_{idn}^{t+1} = \beta_{iw_n} \exp(\Psi(\gamma_{id}^t))$
      normalize ϕ_{dn}^{t+1} to sum to 1
    $\gamma_d^{t+1} = \alpha + \sum_{n=1}^{N_d} \phi_{dn}^{t+1}$

Algorithm 1: E-step parameter estimation

The next step, M-step, is to find the equation that minimizes $D(q||p)$ with respect to α and β. To find β, it is possible to use a Lagrange multiplier since the sum of the probability that a word belongs in all topics is equal to 1 or $\sum_{i=1}^{K} \beta_{wi} = 1$ for a word w, the original author [10] found this equation which is a closed-form solution to the optimization problem:

$$\beta \propto \sum_{d=1}^{M} \sum_{n=1}^{N_d} \phi_{dn} \Omega_{dn} \tag{3.32}$$

The next step is to normalize every row so that all words have a cumulative probability of being in every category equal to one.

Then, using Newton-Raphson's method Jordan et al. [10] update the α as follows:

$$\alpha_{t+1} = \alpha_t - H(\alpha_t)^{-1} g(\alpha_t) \tag{3.33}$$

using

$$
\begin{aligned}
g(\alpha_i) &= M(\Psi(\sum_{j=1}^{K} a_j) - \Psi(a_i)) + \sum_{d=1}^{M}(\Psi(\gamma_{di}) - \Psi(\sum_{j=1}^{K}\gamma_{dj})) \\
H(\alpha_i) &= \delta(i,j)M\Psi^{'}(\alpha_i) - \Psi^{'}(\sum_{j=1}^{K}\alpha_j)
\end{aligned}
\tag{3.34}
$$

where $\Psi^{'}$ is the trigamma function or the first derivative of the digamma function. Eq. (3.33) need to be repeated until convergence.

The final algorithm can be summarized as follows. We first need to initialize all values in β to random positive numbers and normalize β's rows to one. Then initialize all values in α to one arbitrary positive value. Then repeat the E and M steps until global convergence.

3.8.2 Cosine Similarity, Jaccard Distance, Sorensen-Dice

The Cosine Similarity, Jaccard Distance, Sorensen-Dice are similarity measures that use simple bag of words of documents that are compared. We will denote by A and B the bags of words representing the two strings under comparison.

The cosine similarity is the cosine of the angle θ between two vectors given by:

$$
Cosine(A,B) = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} \tag{3.35}
$$

where the $A \cdot B$ is the scalar product between vector A and B.The Jaccard distance is defined by the ratio between the intersection and the union of two vectors. The intersection of two vectors is the number of members (in this case words) that are simultaneously present in both vectors. The union of two vectors is the total number of members (in our case, the total number of different words) present in either both vectors.

$$
Jaccard(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{A \cdot B}{\|A\| + \|B\| - A \cdot B} \tag{3.36}
$$

The Sorensen-Dice similarity is the ratio between the intersection and the sum of the length of the two vectors which is defined by:

$$Sorensen - Dice(A, B) = \frac{2|A \cap B|}{|A| + |B|} = \frac{2|A \cdot B|}{\|A\| + \|B\|} \tag{3.37}$$

Thada et al. [11] tested the cosine similarity, the Jaccard distance and the Sorensen-Dice coefficient for document retrieval using queries. In their paper, they found that the cosine similarity gives better results for their problem.

3.8.3 RV Coefficient

The RV coefficient or correlation of vectors coefficient, developed by Robert and Escoufier [19], is a measure of the similarity of two sets of points represented in two matrices. Let's define two matrices F and G build from two sets of points of dimensionality K and of respectively P and Q rows, where each row are the coordinate of a point, or $F \in \mathbb{R}^{P \times K}$ and $G \in \mathbb{R}^{Q \times K}$.The RV coefficient uses the ratio of the covariance over the square root of the product of the variances of F and G. The RV coefficient takes values between 0 and 1. The RV coefficient is equal to

$$RV(F, G) = \frac{\sum_{i=1}^{K} \sum_{j=1}^{K} f'_{ij} g'_{ij}}{\sqrt{(\sum_{i=1}^{N} \sum_{j=1}^{K} f'^{2}_{ij})(\sum_{i=1}^{K} \sum_{j=1}^{K} g'^{2}_{ij})}} = \frac{tr(F'^{\intercal} G')}{\sqrt{tr(F'^{\intercal} F') tr(G'^{\intercal} G')}} \tag{3.38}$$

where F' and G' are defined as the square positive semi-definite matrices $F^{\intercal}F$ and $G^{\intercal}G$, respectibely, where tr is the trace operation. In the experiment section we will describe how we used the RV coefficient as a similarity measure for NLP task.

3.8.4 Feed-Forward Neural Networks

Let's define a few more variable relevant for feed-forward neural networks.

K length of feature vector x_t

M number of label ed example in a dataset

$X \in \mathbb{R}^{M \times K}$ set of all feature vectors

$b \in \mathbb{R}$ bias

$R \in \mathbb{R}^K$ weight vector

$Y \in \mathbb{R}^M$ set of all labels (on classification task Y is usually a natural number)

To understand how feed-forward neural networks work, we first need to define the perceptron. A perceptron is a single neuron in an artificial neural network, and as such, it can be viewed as the simplest neural network. It was invented by Frank Rosenblatt [40] in 1957. A perceptron is a linear binary classifier. It computes the inner product between a weight vector $R \in \mathbb{R}^K$ and the input $x \in \mathbb{R}^K$. A bias b is added to the result and if the result is greater than zero, the perceptron's output is 1, otherwise, it is 0. Therefore, a perceptron can be defined by the equations (3.39) and (3.40), below.

$$f(x) = \Phi(net(x, R, b)) \tag{3.39}$$

$$net(x, R, b) = x^\intercal R + b \tag{3.40}$$

where $\Phi(\cdot)$ is the activation function, specifically, the step function Φ_{step}:

$$\Phi_{step}(x) = \begin{cases} 1 & \text{if } net(x, R, b) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.41}$$

A graphical representation of a perceptron is shown in Figure 3.3.

To train the perceptron for a classification task, we have to find the optimal weight vector R and bias b that minimize the squared error on the equation:

$$Y = [b||R][1||X]^\intercal \tag{3.42}$$

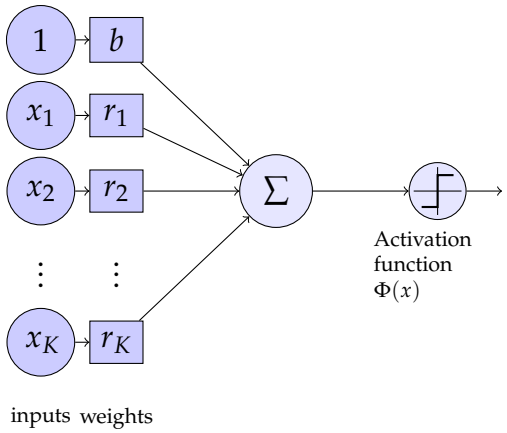

Figure 3.3: Perceptron structure

where the $||$ operand is a concatenation and 1 is the "all-ones" vector of length M. We can solve this equation by using a linear regression. Before performing the linear regression, values in Y have to be set to either -1 (or negative example of this class) and 1 (positive example of this class). The close form solution of this problem is:

$$[b||R] = ([1||X]^\intercal [1||X])^{-1} [1||X]^\intercal Y \tag{3.43}$$

An MLP is a fully connected feed-forward neural network comprising several layers of neurons where all neurons of one layer are connected to all neurons in the next layer. The first layer is the input layer, the last is the output layer, while all the other layers are called intermediate layers. A neural network with a small number of intermediate layers is called shallow as opposed to a deep neural network that contains a large number of intermediate layers. Examples of possibles deep neural network can be convolutional neural network (CNN), long short term memory (LSTM), gated recurrent unit (GRU), deep belief network, etc. The complexity of those networks made it impractical (sometimes impossible) to use a closed-form solution to train them. This is why most neural networks are trained using backpropagation or gradient descent describe by the equation:

$$[R_{t+1} + b_{t||1}] = [R_t||b_t] - \alpha(\Delta[R_t||b_t]) \tag{3.44}$$

where $\Delta[R_t||b_t] = \frac{\partial E}{\partial[R_t||b_t]}$ is the partial derivative of the mean square error (MSE) with respect to the weight for the neuron t and α is a learning parameter $0 < \alpha \leq 1$. The MSE of a neural network, also call the loss function, can be compute using :

$$E = \frac{1}{2M}||Y' - Y||^2 \tag{3.45}$$

where Y' is the actual output vector of the neural network. Since the derivative of the step function is zero everywhere except at the origin where it is not defined, other activation functions such as the hyperbolic tangent ($\tau(x) = Tanh(x)$) or the logistic function ($\sigma(x) = \frac{1}{1+\exp(-x)}$) are commonly used in neural networks trained using backpropagation. Next, we will derive the backpropagation function in equation (3.44) for one neuron, say neuron j, using equations (3.39) and (3.40) and the derivative chain rule . For simplification, the bias b_j is included in the vector R_j:

$$\frac{\partial E}{\partial R_j} = \frac{\partial E}{\partial \Phi(o_j)} \frac{\partial \Phi(o_j)}{\partial o_j} \frac{\partial o_j}{\partial R_j} \tag{3.46}$$

with $o_j = net(x, R_j)$.

Further, computing the derivative of the MSE

$$\frac{\partial E}{\partial \Phi(o_j)} = \frac{\partial E}{\partial y'} = \frac{\partial}{\partial y'} \frac{1}{2M}(y - y')^2 = \frac{1}{M}(y' - y) \tag{3.47}$$

where y and y' are respectively the objective output (or labels for the last layer) and the actual output of the neuron. Finally,

$$\frac{\partial \Phi(o_j)}{\partial o_j} = \frac{\partial}{\partial o_j}\Phi(o_j) = \begin{cases} \sigma(o_j)(1 - \sigma(o_j)) & if\ \Phi(o_j) = \frac{1}{1+\exp(-o_j)} \\ 1 - \tau(o_j)^2 & if\ \Phi(o_j) = \tanh(o_j) \end{cases} \tag{3.48}$$

$$\frac{\delta o_j}{\delta R_j} = \frac{\delta}{\delta R_j} R_j^\intercal x = x \tag{3.49}$$

These equations are used to update the weights of neurons in R during the backward pass. The calculation of the loss function is called the forward pass. Since the number of training examples can be very large, it is possible to speed up the training phase by using subsets or batches of all training examples at once instead of training the neural network one example at the time. Weight also needs to be updated on multiple iterations. During training, each time a neural network sees all the training examples once, it is called an epoch, neural networks usually require to be trained on multiple epochs.

Backpropagation is commonly used to train shallow neural networks. A problem arises when the error has to backpropagate through multiple layers of neurons as, of course, in the case of deep neural networks. In a neural network, as demonstrated by Bengio et al. in [42] the magnitude of the gradient can decrease exponentially and become extremely small when it reaches the first layers; this problem is called the vanishing gradient problem. Bengio et al. also demonstrated in another paper [41] the effect of an exponentially increasing gradient or exploding gradient.

There are several methods to deal with the problems of vanishing and exploding gradient. One of them, the auto-encoding learning method, is to train a subset of layers at the time, starting from the earliest layer to the latest. In each subset of layers, the objective is to ”encode” the input in a small layer of neurons then reproduce the input in a larger layer. This technique can work without any labels. Auto encoding technique can be used in CNN [44] and deep belief network [43]. A deep belief network is a neural network with an architecture that can reassemble a MLP. The difference is that they are trained using auto-encoder so they can produce good results with many layers.

3.8.5 Long Short Term Memory (LSTM) Neural Networks

To understand how LSTM work, we will first start by defining several variables.

H hidden state size

$x_t \in \mathbb{R}^K$ feature vector at step t

$f_t \in \mathbb{R}^H$ output vector of the forget gate at time t

$i_t \in \mathbb{R}^H$ output vector of the input gate at time t

$o_t \in \mathbb{R}^H$ output vector of the output gate at time t

$l_t \in \mathbb{R}^H$ cell state vector at time t

$h_t \in \mathbb{R}^H$ hidden state vector at time t

$R_f, R_i, R_o, R_l \in \mathbb{R}^{H \times K}$ input gate weight matrices

$U_f, U_i, U_o, U_l \in \mathbb{R}^{H \times H}$ output gate weight matrices

$b_f, b_i, b_o, b_l \in \mathbb{R}^H$ bias vector

To process series (sequence of events temporally related such as stock prices, meteorological measures, sequences of words in a document, etc) , recurrent neural networks (RNN) are very popular. RNN is a class of neural networks that have a memory cell whose output is also called internal state. Each entry in a series is processed one by one as individual inputs. When processing the input at step t in a series, RNN uses their internal states l from step $t-1$ as input. Since RNNs are able to convey information from every time step before t, they can make decisions (classification, estimation) based on a complete sequence of events. Since time series can be very long, for an RNN to work properly in those circumstances, it needs a mechanism to keep the information when it is relevant to the task or otherwise discard it and, of course, also avoid the vanishing/exploding gradient problem.

Gating is the most used keeper/discarder information mechanism in RNNs. The general idea of gating is to use a gate, which is a layer of neurons, that control the "flow" of information in the network. It is used, for instance, in gated recurrent unit (GRU) [45] and LSTM neural network [46]. GRUs and LSTMs are similar variations of gated RNN, there is two main difference between then. The first being that the LSTM have three gates, input gate, output gate and forget gate and the GRUs have two gates

the update gate (similar to the LSTM's input gate) and the reset gate (similar to the LSTM's forget gate). The second difference being that the LSTMs have two memory cells, the cell state and the hidden state while the GRUs only have one memory cell, the output vector.

There are several possible LSTM schemes. In a blog post, Microsoft researcher James McCaffrey [57] describes a basic LSTM with three gates: a forget gate, an input gate and an output gate. He describe an LSTM using equation (3.50) to (3.54).

$$f_t = \sigma(R_f x_t + U_f h_{t-1} + b_f) \tag{3.50}$$

$$i_t = \sigma(R_i x_t + U_i h_{t-1} + b_i) \tag{3.51}$$

$$o_t = \sigma(R_o x_t + U_o h_{t-1} + b_o) \tag{3.52}$$

$$l_t = f_t \circ l_{t-1} + i_t \circ \tau(R_c x_t + U_c h_{t-1} + b_c) \tag{3.53}$$

$$h_t = o_t \circ \tau(l_t) \tag{3.54}$$

Where the $\circ$ operation is a point-wise multiplication, τ is the hyperbolic tangent function, σ is the sigmoid function, eq. (3.50) is the equation of the forget gate, eq. (3.51) is the equation of the input gate, eq. (3.52) is the equation of the output gate, eq. (3.53) is the equation of the cell state and eq. (3.54) is the equation of the hidden state. The total number of parameters is equal to $4HK + 4H^2 + 4H$.

A diagram of an LSTM can be found in figure (5.15), it might help clarify how LSTM's equation work together. As we can see in eq. (3.50), (3.51) and (3.52), the output of the three gates have their range limited between 0 and 1 by a sigmoid function. Then, the outputs of those gates are pointwise multiplied with a "signal" or vector such as in eq.

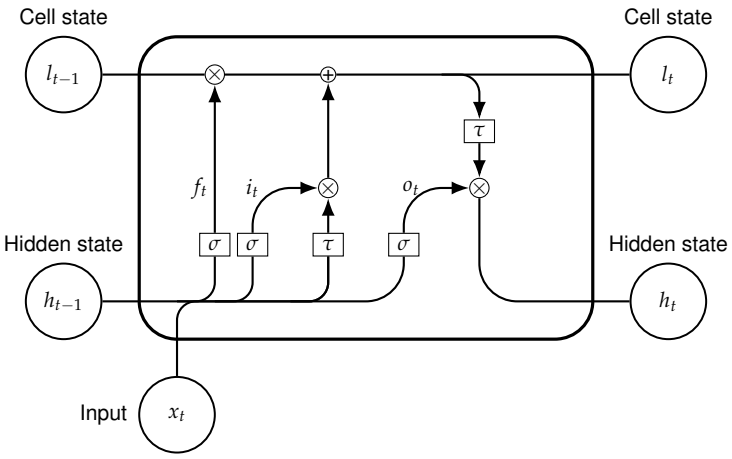

Figure 3.4: Graph representation of an LSTM

(3.53) and (3.54). By doing this pointwise multiplication with values ranging from 0 to 1, gates are able to control how much information is transmitted to the current cell state and the hidden state. In a LSTM, the role of the forget gate is to control how much information will be convey from the previous cell state to the actual cell state. The input gate control how much information is added from the previous hidden state and the current input to the cell state. Finally, the output gate control how much information is transmitted from the cell state to the hidden state. The difference between the cell state and the hidden state is very subtle. Since the cell state depends on the previous cell state plus the information the input gate let trough, the cell state can be seen as the accumulation of information at every time step. Since the hidden state values depend on the cell state and the output gate, the hidden state can be seen as the relevant information from every time step.

Chung et al. [45] did a comparative study of LSTM, GRU and normal RNN for different tasks. They concluded that both LSTM and GRU are superior to the normal RNN due to their gated unit. However, between the GRUs and LTSMs, they could not conclude that one is better than the other and they did not conduct their research specifically on a NLP task. Irie et al. [56] reported better results, in general, using LSTM than GRU for several NLP tasks but not by a wide margin.

To prevent overfitting, one common solution is to use dropout. The dropout regularization technique is used by deactivating a randomly chosen set of connections in a neural network during training. For each trained batch , a percentage of connections will not be used during the forward and backward pass.

3.8.6 Skip-Through Vector

Skip-through vector, developed by Kiros et al. [20], is a good example of how we can use deep learning for various NLP tasks. The goal of this algorithm is to encode sentences for a general-purpose semantic task. They used a GRU neural network to predict a sentence using the previous and next sentences as inputs. They then used the output vector of the GRU as the feature vector for various tasks such as semantic-relatedness, paraphrase detection, image-sentence ranking and classification.. They used a pre-trained word2vec [18] word-embedded representation to encode words. They trained their model on a large corpus of books (BookCorpus dataset). They show that skip-thoughts vectors, or the output vector of their GRU, learn to accurately capture semantic and syntactic elements of encoded sentences. They achieve very good overall results on several datasets such as: the SICK dataset [62], the Microsoft Paraphrase Corpus dataset [63], subjectivity/objectivity classification on the SUBJ dataset [64], opinion polarity on the MPQA dataset [65], question-type classification on the TREC dataset [66] and image caption on the COCO dataset [67].

3.8.7 NLP and DNN Transfer Learning (TL)

The general idea of TL is to use a large dataset to increase the accuracy of a neural network for a task where a smaller dataset is available. The intuition behind TL is that some patterns can be learned in a dataset that will be useful for a task on another dataset. For instance, in computer vision, a neural network can learn low-level features (on few pixels) such as edges, corners, etc. Those features are present in all images thus can be used on a wide variety of computer vision tasks. In TL, it is also common to ”freeze”

layers of neurons, this means to not allow those layers to be trained or fine-tune on the smaller dataset.

Mou et al. [69] did a comparative case study on TL of DNN for NLP applications. Their motivation was the realization that although multiple papers reported good results OF? on image processing tasks, the reported results on NLP tasks were not promising.

They defined two distinct categories of TL training techniques. The first one, which they called (INIT), consists of two phases. The first phase being pre-training, or initialize, a DNN on the large dataset then use the parameter found as an initial point for the second phase. The second phase being fine-tuning or using the smaller dataset to train the DNN. The second technique, which they called multi-task-learning (MULT), is to train a model on both databases simultaneously. In their experiments, they used two distinct neural network configurations. For their first experiment, they used an LSTM and for their second experiment, they used a CNN.

Their study focused on three questions. The first one is, whether we can transfer knowledge from one neural network trained on one task to another in the cases where the tasks are similar or different. The second is, which layers of the neural network are transferable? And finally, what is the difference in transferability between INIT and MULT and what is the effect of combining those two methods? They tested TL on several sentence classification tasks.

For their first question, they found out that the transferability of a neural network depends largely on how semantically similar the tasks are. In another paper, Mueller et al. [70] reached the same conclusion. Wei et al. [71] also concluded that using pre-trained word vectors that are trained on corpus semantically similars to the task result in better performance in TL that word vectors trained on unrelated tasks. For their second question, they concluded that only the last layer of the neural network is not transferable. Wei et al. [71] concluded that freezing excessive or too many layers impacts negatively the performance of TL. For their final question, they found that MULT and INIT achieve generally similar results and combining these two approaches does not

result in any significant improvement. It is also worth noting that they observed, in the case of the INIT configuration, that the accuracy on the smaller dataset can worsen when the number of trained epoch on the original problem (bigger dataset) gets higher than the optimal number of training epochs.

Chapter 4

Methodology: Architecture and Databases

In this chapter, we will describe the architecture of the proposed system. First, we will describe in detail the problem that we consider. Then, we will present the different databases that we used. As we will see, the nature of these databases was the main deciding factor behind our choice to build a content-based recommender system over a collaborative filtering algorithm. Finally, we will discuss several other design choices, such as, the pre-trained word embedding vector that we chose and the different information measures and similarity measures that we tested.

4.1 Problem Description

For this project, the goal is to match vacant positions in projects with applicants. The data is provided by Fleexer and is only textual. To test and train our recommender system, we manually assigned scores for 600 pairs of position and applicant. Our method must be able to take a pair of strings as input and output a score that represents how good of a match the position is to the applicant.

4.2 Architecture Overview

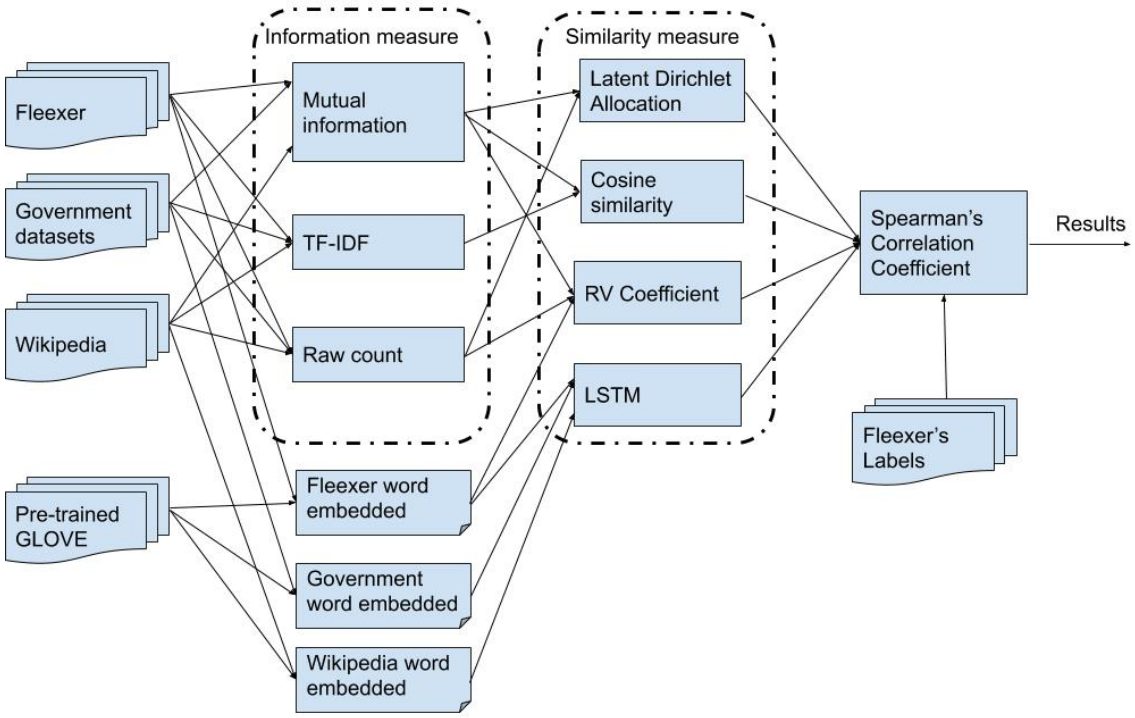

Figure 4.1: Architecture overview

A schematic of the proposed architecture is shown in Figure 4.1. In our project, we used three different databases, Fleexer's database, a database created from government entities website and a Wikipedia database from the Westbury Lab [68]. All those databases are use in our information measures. Then, using our similarity measures, we find scores for our position-applicant matches. Finally, we compare the output of our recommender system against our 600 manually assigned scores using the Spearman's correlation coefficient to evaluate our performances.

In the following, we will describe all the above components in detail, starting with the databases used.

4.3 Databases

For this project, we used databases from three different sources: Fleexer's own databases, government agencies' databases and Wesbury Lab's Wikipedia corpus (2010) [68].

4.3.1 Fleexer's Database

The goal of Fleexer's website is to let people create new projects or to contribute to existing projects. Multiple projects can also be clustered in an organization. Organizations, however, were not considered in this research. For each project, there are fields for the project name, start and end date, description, one or more business activities (which is a categorical field, e.g., science, business, art, car, etc) and contact information. Each project also has one or more available positions that need to be filled. For each position, there are fields for the title, category (also categorical, e.g., design, chemestry, security, etc), the number of applicants needed to fulfill that position, description and spoken languages. For each applicant, there are fields for name, short bio, education level, spoken languages (categorical, e.g., French, English, Spanish, etc) and contact information. Also, each applicant has one or more skills that they can advertise. For each skill, there are fields for title, category (categorical, e.g., art, design, education, etc.), short description and an objective (categorical, e.g., finding an internship, job, personal development).

A survey of the profiles, reveals that this database does not contain a lot of the ambiguities described in section 3.3 such as humor or sarcasm and, therefore, there is no need to implement a sentiment analysis algorithm. However, using this data we might encounter several homographs such as *"bar"* which could be used in sentences such as *"I passed the bar exam"* or *"I worked in a bar"*.

At the time of writing this thesis, Fleexer's database was populated by 389 positions and 552 skills from applicants for a total of 941 examples. However, around one-third of the database contains incomplete profiles. Fleexer's also owns a human-made dataset of 600 labels. To create those labels, we manually took 60 project's positions and 60

applicant's skills and found for all of them five matches rated from 1 (very good) to 5 (poor) according to the following scale: 1 meant a very good match (in other words, the applicant should be working on this project), 2 meant a good match (or the applicant might be a good fit but someone more qualified could be found), 3 corresponded to a medium match (or the domain of expertise of the applicant and the required skills for the project had some overlap but the applicant wouldn't be a good fit), 4 meant a bad match (or the domain of expertise of the applicant and the required skills for the project had some small overlap but the applicant was not qualified for the position) and, finally, 5 corresponded to a very bad match (there was no overlap between the domain of expertise of the applicant and the required skills for the project).

To build our content-based recommender system, we chose to use only textual information, thus, we transformed any categorical data into text. We also thought that some information was irrelevant for the algorithm such as the name of the applicant or their contact information. On the applicant skills side, we chose to keep the applicant's description, and the skill's category, description and title. On the position side, we chose to keep: the project's title, description, and the position's category, description, and title.

4.3.2 Government Databases

Since Fleexer's database was of limited size, we decided to use external databases to see if we could improve on the accuracy of certain algorithms such as LDA, LSTM, and our information measures. In other words, we wanted to know if using external databases could remove certain limitations that comes with working on small databases. Ideally, we would have used databases of job postings, applicant resumes and relational data between those databases. For obvious reasons, public resume databases don't exist let alone relational data between job postings and resumes. Another problem is that all data from third-party recruitment websites is proprietary, thus cannot be legally used for commercial purposes.

For the above reasons, we inquired at different government entities about allowing us to use their job posting data on their websites. The city of Quebec, the town of Longueuil and the Canadian Ministry of the public service gladly let us use their data for our research. During summer 2018, When we gathered that data, the city of Quebec had 161 job postings, the town of Longueuil had 69 job postings and the ministry of public service had 937 job postings for a total of 1167 job postings.

Since these job postings were done according to government standards, some words were repeated very often which might misguide some algorithms to conclude that those words are not relevant to the task. For instance, since Canada is an officially bilingual country (French and English), almost all job posting from the Canadian ministry of the public service contains terms like ”Bilingual Imperative”, ”French” or ”English”. There are also some terms that are just very usual in most job postings such as ”ability to communicate”, ”experience in”, ”level of education” or ”competitive salaries and benefits package”. Those words can potentially impact negatively our results.

4.3.3 Westbury Lab's Wikipedia Corpus (2010)

The Westbury lab is a psycho-linguistic laboratory at the University of Alberta located in Edmonton Canada. To create this database [68], they used the April 2010 English Wikipedia Dump. From every article, they removed all links, navigation text and other irrelevant material to obtain only textual data. They then removed every article less than 2000 characters long.

The final wikipidia database contains more than 2 million documents and 990,248,478 words. Since we have far less than 2 million job-related documents (government datasets and Fleexer's dataset) we only used 2500 documents to train our LSTM. We choose to used this database to see if we can still improve our results even if this database is unrelated to our problem.

4.4 Data pre-Processing

Before using it, we first needed to pre-process the raw data from the three databases. Since Fleexer, at the time of this writing, operates mostly around the region of Montreal, their database's textual data are in French and English. The city of Quebec and the town of Longueuil had only french job postings, however, most job postings from the Canadian ministry of the public service were in English. For this reason, we had to translate all of our data into one language. We chose to translate everything into English for two main reasons. First, so that we wouldn't have to deal with all the special character present in French such as é,è,ê,à,ç, etc. Secondly, there are more available NLP resources in English such as lemmatizers or pre-trained word embedded vectors. To do the translation, We used the Azure translator application programming interface (API) from Microsoft [87].

We also used a lemmatizer called LemmaGenerator available on Nugget [88]. This allowed us to reduce our vocabulary. We then removed all special characters or all characters that were not a letter (a to z, A to Z) or a number (0 to 9).

4.5 Word Embedding

We chose to use the pre-trained 300-dimensional Glove word embedding developed by Pennington et Al. [2]. This pre-trained word vector embedding is available under the Public Domain Dedication and License which gives us the right to use it for commercial purposes. It is also widely used in the literature [3], [36], [89], [90]. The creators of the algorithm trained it using the common crawl database containing 840 billion tokens (words). The word embedded vector was trained over a dictionary of 2.2 million words. No special tag was put on homographs (words with multiple meanings) such as "address" (is it a location or the verb to address?) or "ring" (is it a sound or a jewel?).

4.6 Information Measures

For this project, we used information measures as a weighting scheme and to reduce our dictionary. For weighting purposes, we used the mutual information (MI) and the term frequency-inverse document frequency (TF-IDF). When using TF-IDF, we chose the raw term frequency in eq. (3.16) as TF and we chose the inverse document frequency in eq. (3.21) as IDF. We chose those TF-IDF schemes since they provided good results in document query applications according to Salton & Buckley [30].

To choose which word to keep in the dictionary, we used MI and raw word count. We chose to remove words using two thresholds, a lower threshold to only remove the words with low amount of MI or only remove the more frequent words using raw count and an upper threshold to only remove words with high MI or only remove the rarest words using raw count . For every experiment that used words removal, we tested several values for lower thresholds ($\{0\%, 1\%, 5\%, 10\%, 15\%\}$) as well as for the upper threshold at values ($\{100\%, 80\%, 60\%, 40\%\}$). For instance, if we were testing with a lower threshold of 5% and an upper threshold of 60% using raw word count, that would mean that we removed from the dictionary the top 5% most common words and the top 40% rarest words. We did our tests by using every combination of both thresholds.

4.6.1 TF-IDF and Raw Word Count

For both TF-IDF and raw word count, we tested a combination of Fleexer's data and the government's jobs databases as well as a combination of all three databases. The reason why we tested two sets of databases is to determine if we could gather more beneficial information using general data compared to only job-related data.

4.6.2 MI

The idea behind using MI is that words that appear more often in job-related texts than in general texts might hold more information than words that are equally likely to appear in the two datasets. To train MI, we used two different classes, job-related text and general text. We used our Fleexer and government's jobs databases as one job-related database and Westbury's Wikipedia databases as general texts.

4.7 Similarity Measures

We tested several similarity measures which have been presented previously in the background in the similarity measures section. In this subsection, we will describe how we used them.

4.7.1 Latent Dirichlet Allocation (LDA)

For LDA, we have several parameters or and architectural choices that can impact our results such as the number of hidden topic K, which databases to choose from to train LDA, how many words are we removing from our dictionary and which information measure is the most appropriate to do so. We trained LDA using three training datasets: Fleexer's data only, job-related documents only (Fleexer's and government entity's) and from all available databases (FLeexer's, government entity's and Westbury Wikipedia). We tested LDA with several values for the number of topics $K \in \{25, 50, 75, 100, 150\}$. We also tested MI and raw word count to remove words from the dictionary. We tested raw word count from job-related documents only and from all available documents. We used the probability distribution of all topics θ for each document as a feature vector. Finally, using this feature vector, to measure how similar two documents are, we used the cosine similarity.

4.7.2 Cosine Similarity

To use the cosine similarity as a similarity measure, both texts that are compared need to be transformed to vectors. The vector that represents a text is the sum of the one-hot vector representation for each word in the text. We can multiply each element of the vector with the associated word's amount of information (MI or TF-IDF) when we want to use information measure as weight. We tested the cosine similarity with: no information measure, with MI, with TF-IDF from job-related documents only and TF-IDF from all sources.

4.7.3 RV Coefficient

The RV coefficient is a similarity measure between two sets of points . Using the pre-trained GLOVE word embedded vector, we created for each document a matrix representation for a document i of dimension $N_i \times K$ where N_i is the number of words in documents i and K the word vector representation length. Each row, or each point in the matrix, is a vector representation of a word. The idea is that since similar word vectors will cluster in the vector space and the distance between words vector carries information, that the matrix representation of two similar documents even when using different words should have a high normalized correlation coefficient. For instance, the RV coefficient of the matrix representation of the document "*tiny dog*" should be very similar as the one for "*small canine*" since "*small*" and "*tiny*" should be, in the vector space, close to each other which should also be true for "*dog*" and "*canine*", thus both documents should have a high RV coefficient. As a brief illustration of this hypothesis we compare the phrases $A =$"*i ran to the store*" to $B_1 =$"*i ran to lose weight*" and $B_2 =$"*i sprinted to the shop*". Using the cosine similarity at eq. (3.35) without weighting we get, $Cosine(A, B_1) = Cosine(A, B_2) = \frac{vec_A^\intercal vec_{B1}}{||vec_A||^2||vec_{B1}||^2} = \frac{vec_A^\intercal vec_{B2}}{||vec_A||^2||vec_{B2}||^2} = \frac{3}{\sqrt{5}\sqrt{5}} = 0.6$. Using the RV coefficient with GLOVE word embedding, we have $RV(A, B_1) = 0.5678$ and $RV(A, B_2) = 0.8960$. This is a desired result since the meaning of the phrases in A

and B_2 are much more similar than A and B_1. We also used the raw word count and MI to reduce our dictionary. According to Josse et al. [76], the value of the RV coefficient also depends on the sample size. For this reason, we also tested the RV coefficient using a maximum of W_{max} words by documents. For those tests, we still used MI and raw word count to remove words from the dictionary. From those remaining words, for each documents, using MI, we choose the W_{max} words with the most information still in the dictionary and using raw word count we choose to keep the W_{max} words with the rarest occurrences.We tested W_{max} with values $10, 25, 50, 100$.

4.7.4 Long-Short Term Memory (LSTM)

Since for this project we only had 600 labels, we could not train an LSTM on this data alone. Thus, we chose to use a TL approach. First, we needed a dataset to train an LSTM. We chose to use all our datasets and train the LSTM to recognize if a document is from the Westbury lab's Wikipedia corpus or if it is job-related. The final classifier is a perceptron with a softmax classifier and we used the hidden state of the LSTM as the input to the perceptron. We chose to use the INIT approach since the LSTM architecture has to change from the initial training phase to the fine-tuning phase and according to Mou et al. [69], there is no significant performance difference between INIT and MULT. We also used the pre-trained 300-dimensions GLOVE word embedding vector [2]. A graphical representation of the architecture of the LSTM during the initial training phase can be found in figure (4.2) where x_t is the input at step t and h_t is the hidden vector of the LSTM at step t.

We used two different approaches for the task of rating matches between applicant and project position. One was to use the cosine similarity between the hidden state vector of the LSTMs for the applicant's document and for the project position's document. The second approach was to use a linear regression on the hidden state vector of the LSTMs. For the fine-tuning phase, we cloned our LSTM then used the hidden state vector as input for a linear regression model. Cloning a neural network mean to create a ,complete

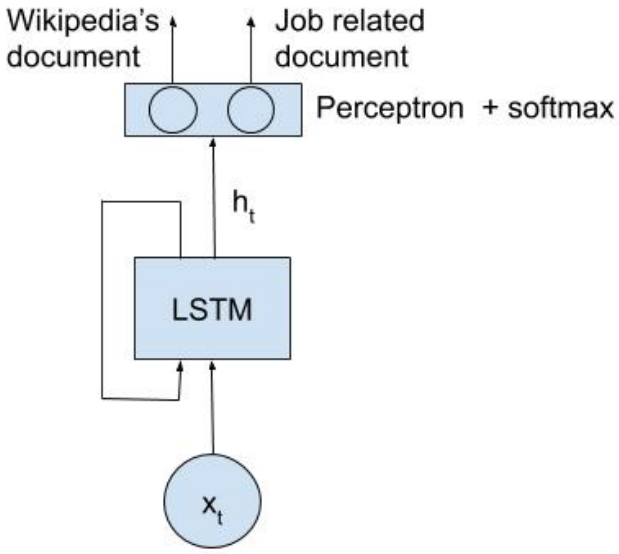

Figure 4.2: LSTM architecture, initial training

or partial, copy of the network, both the architecture and the parameters are exact copy. We trained our new neural network using the match rating labels as the target. This allowed our networks to train using both documents as input at the same time. The LSTM parameters were shared during cloning, i.e., all the parameters of both LSTM's were updated with the same values thus, they remained copies of each other. We did our test using 10-fold cross-validation. Then, after the fine-tuning phase, we tested our LSTM with both approach, keeping our linear regression as last layer and using the cosine similarity as the last layer. The learning rate was set to 0.005 and we used 30% dropout during both learning phases. A graphical representation of the architecture of the LSTM during the fine-tuning phase can be found in figure (4.3).

For the linear regression model, after the fine-tuning, we decided to use the hidden state vectors from both LSTMs and compute the closed-form solution of the linear regression from there. It is common practice to use the hidden vector of the LSTM as a feature vector [69], [70], [71], [89]

We tested our LSTMs over several cell sizes, trained epochs and fine-tuned training epochs. The cell size varied over $\{50, 100, 200, 400\}$ which give respectively 61200,

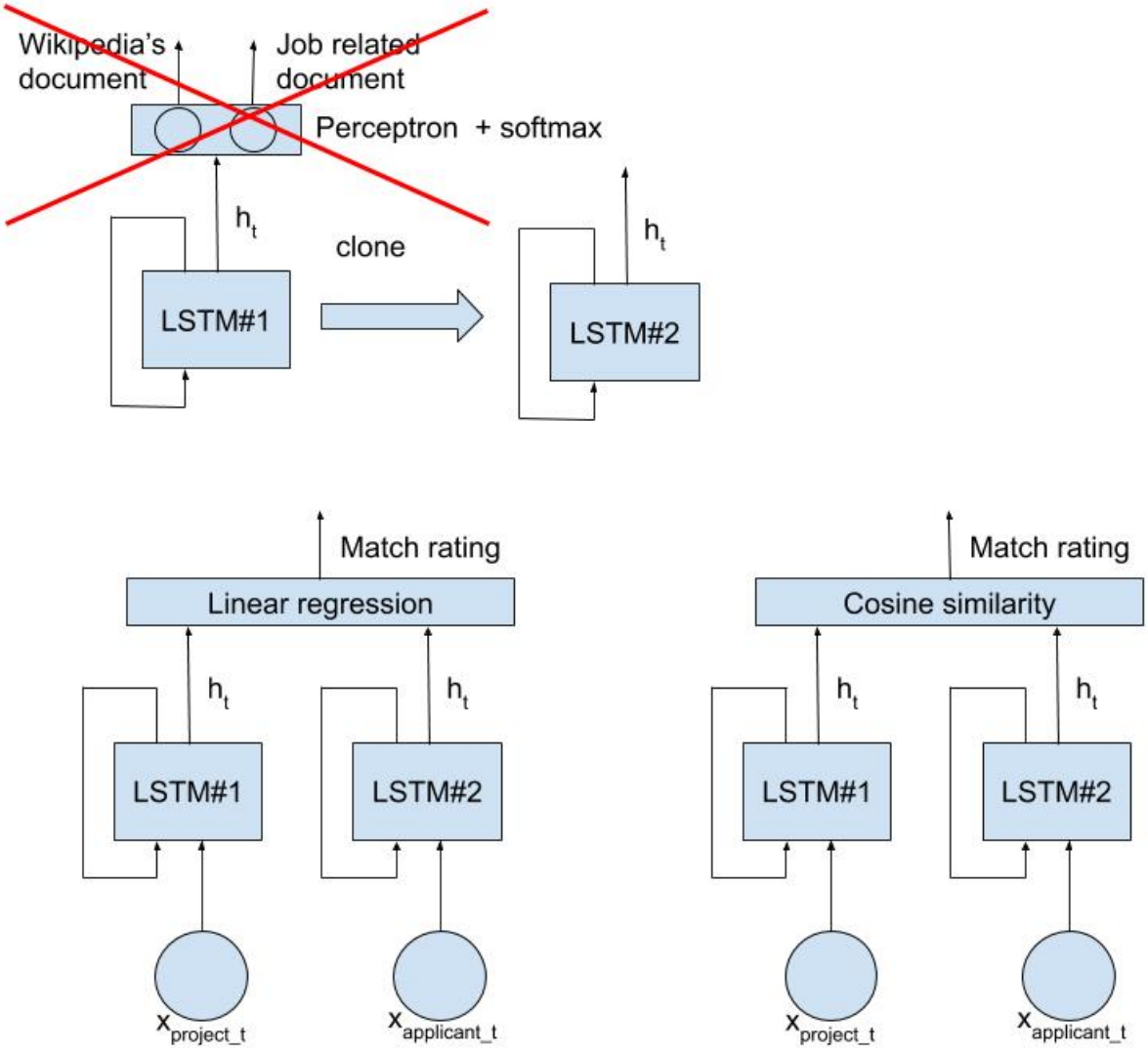

Figure 4.3: The LSTM architecture in fine tuning: top left is the original LSTM cloned without the last layer, bottom left is our final LSTM architecture with a linear regression as last layer, bottom right is our final LSTM architecture with the cosine similarity as last layer

160400, 400800 and 1121600 parameters. The initial number of training epochs varied over $\{2, 5, 10, 30\}$. The number of fine-tuned training epoch varied over $\{0, 5, 10, 30, 50, 100, 200\}$ where using 0 fine-tune training epochs corresponds to using the original LSTM as-is. The number of initial training epochs may seem low but we observed that the LSTM cell rapidly overfitted which necessitated early termination of training.

4.8 Evaluation

To evaluate our algorithms, we chose to use the Spearman correlation coefficient. This evaluation criterion makes sense since it is more practical to measure if the algorithm rank matches similarly to a human rather then measure if the algorithm's output is linearly similar to the human labels. This is true because, for some labels, good matches were hard to find due to the small database, therefore, some labels classified as very good matches (or 1's) might not be so good after all.

Chapter 5

Results

In this section, we will show the results of our experiments. For each parameter (hyper-parameter, the dataset used, information measure used) that we evaluated, we will only display the best results obtained for each value. We used Fisher's transformation that transforms the Spearman's correlation coefficient to a Z-score in eq. (3.4), we then got a z-score with eq. (3.5) and performed a z-test with eq. (3.6). Using those equations, we performed several statistical significance tests on our results to add perspective. A summary and discussion of these results will be given in the conclusion section.

5.1 LDA

For LDA, we first analyze the effect of changing the information measure to discriminate words. We can see in Table 5.1 that using raw word count on all documents gave us the best result with a Spearman's correlation coefficient (ρ_s) of 0.415. Using raw word count on job-related documents did slightly worse with a best ρ_s of 0.392 and finally, the best ρ_s for MI was 0.307. The columns dataset used, Min inf thresh, Max inf thresh and K refer to the parameter that gave us the best result with respect to the information measure used. A statistical significance test on those results reveals that the probability of those

information measures being the best is: raw count word on all documents at 68%, raw word count on job-related documents only at 31.5% and MI at 0.5%.

Information Measure	ρ_s	dataset used	Min inf thresh	Max inf thresh	K
MI	0.307	Fleexer	1%	100%	50
Raw word count (Job related documents only)	0.392	Fleexer	0%	80%	75
Raw word count (All documents)	**0.415**	All documents	0%	40%	100

Figure 5.1: Best results using LDA by information measure

We then analyze the effect of using different datasets to train the LDA algorithm. As we can see in Table 5.2, the best result was obtained when we trained LDA on all documents with a ρ_s of 0.415, on only Fleexer's documents we obtained a ρ_s of 0.401 and only on job-related documents we obtained a ρ_s of 0.360. In addition, the best result using job-related documents was obtained using all words in the corpus. A statistical-significance z-test on those results reveals that the probability of these corpora being the best choice to train on is: all documents at 58%, Fleexer's documents at 36% and job-related documents at 6%. It is hard to explain why training LDA on job-related documents performed worst. Since those documents were generated mostly by government entities, it is possible that the repetition of some words and terms (such as *French*, *English*, *benefit* or *education level*) made the algorithm think that those words were likely to appear in most topics.

Training dataset	ρ_s	information measure used	Min inf thresh	Max inf thresh	K
Fleexer	0.401	Raw word count (All documents)	5%	40%	100
Job related documents	0.360	Using all words	–	–	75
All documents	**0.415**	Raw word count (All documents)	0%	40%	100

Figure 5.2: Best LDA results by dataset used

In Figures 5.3 and 5.4 we see the effect on LDA of varying the number of topics for different the datasets (Fig. 5.3) and information measures (Fig. 5.4). In Figure 5.3, we can see that LDA's performance peaks at 100 topics when it is trained on Fleexer's dataset or

on all documents but peaks at 75 topics when trained on job-related documents. We can see that adding more topics increased the performance of LDA more significantly when trained on all documents; this might be because the Wikipedia's corpus has a more diverse vocabulary, therefore, LDA requires a larger number of topics to distinguish between job-related topics.

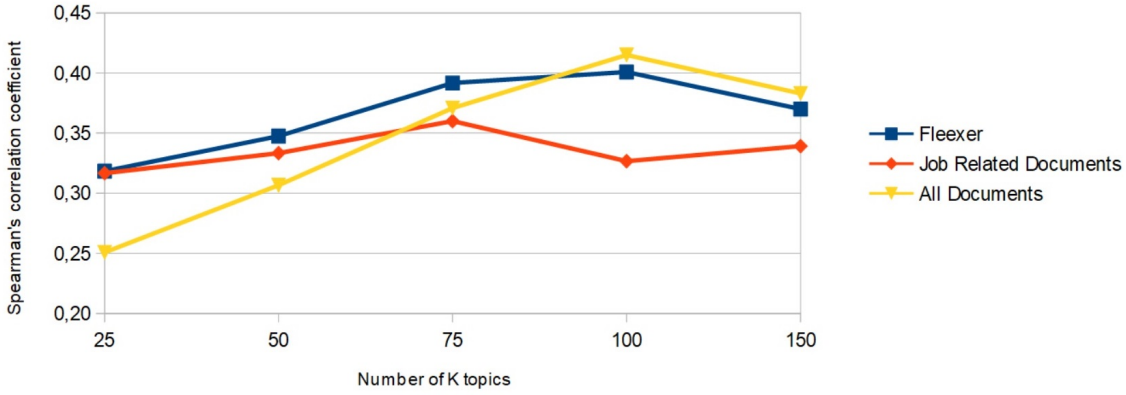

Figure 5.3: Best results by number of K topics for different datasets used for training for LDA

In Figure 5.4 we can see that the MI performance peaks at 50 topics, raw word count on only job-related documents peaks at 75 topics and raw word count with all documents peaks at 100 topics. We can also see that on this graph the difference between the results increases with the number of topics. Therefore, the higher the number of topics, the more important it is to choose the best scheme to discriminate words.

In Figure 5.5 and 5.6 we can see the effect of varying the minimum amount of information threshold. For raw word count, words with low amounts of information are words that appear very often such as prepositions, pronouns or articles. For MI, words with low amounts of information are words that are equally likely to appear in job-related documents and in any other documents, again, words such as prepositions, pronouns, articles. Therefore we expected those words to be irrelevant to the task but when we look at the results, in general, we can tell that removing them decreased the performances of LDA. It seems that LDA properly assigns the probability distribution

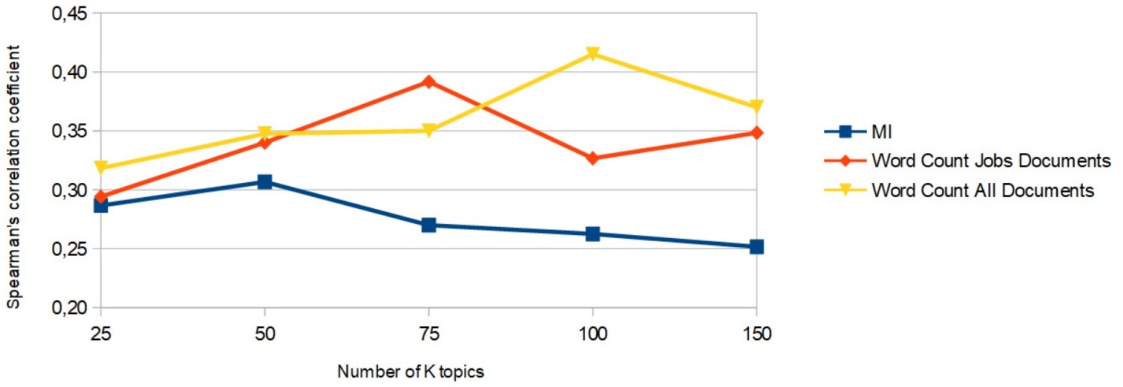

Figure 5.4: Best results by number of K topics for different information measures for LDA

of the topics of those words. It is also possible that some of them are in fact relevant to the task and since they appear often in the corpus, they help the LDA model to make better word-topic associations.

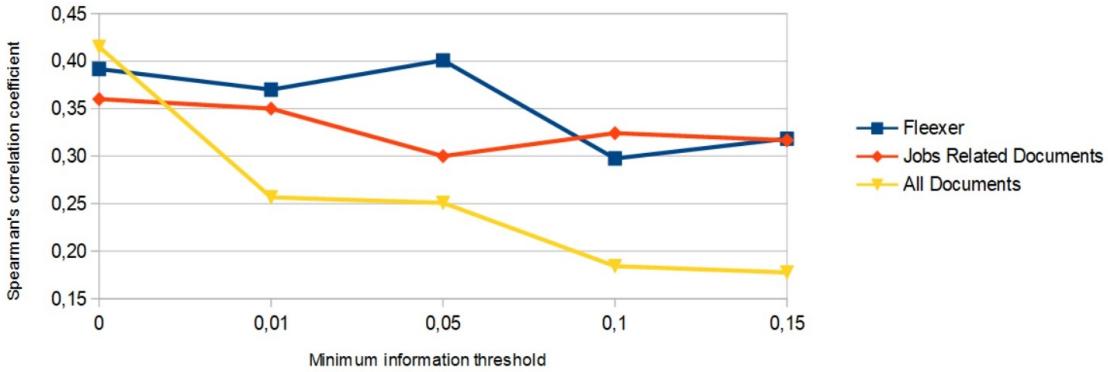

Figure 5.5: Best results by minimum information threshold for each dataset used for LDA

In Figures 5.5 and 5.6 we can see the effect of varying the maximum amount of information threshold. For raw word count, words that we consider having a high amount of information are uncommon words while for MI, they are words that are more likely to appear in a job-related document. We expected that removing those words would decrease the performance of our LDA models but, in fact, we could not saw such a trend. The performance decreased slightly by removing words for MI and raw

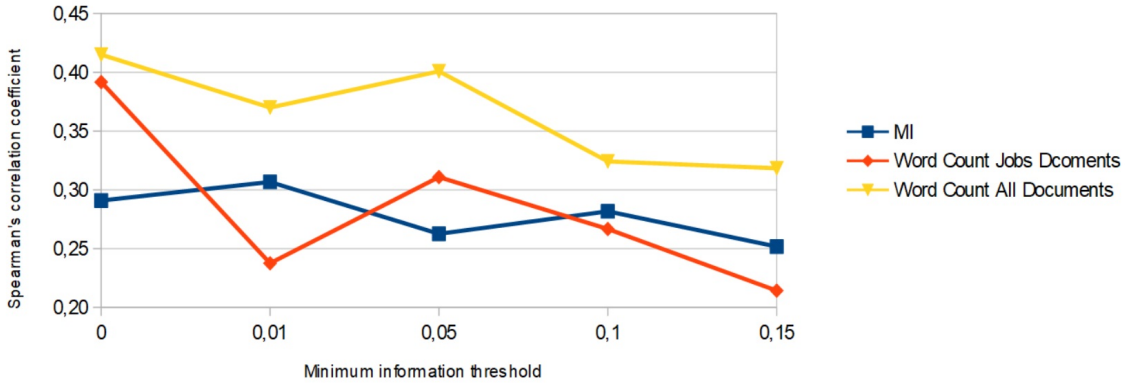

Figure 5.6: Best results by minimum information threshold by information measure for LDA

word count on job-related documents but increased on raw word count trained on all documents.

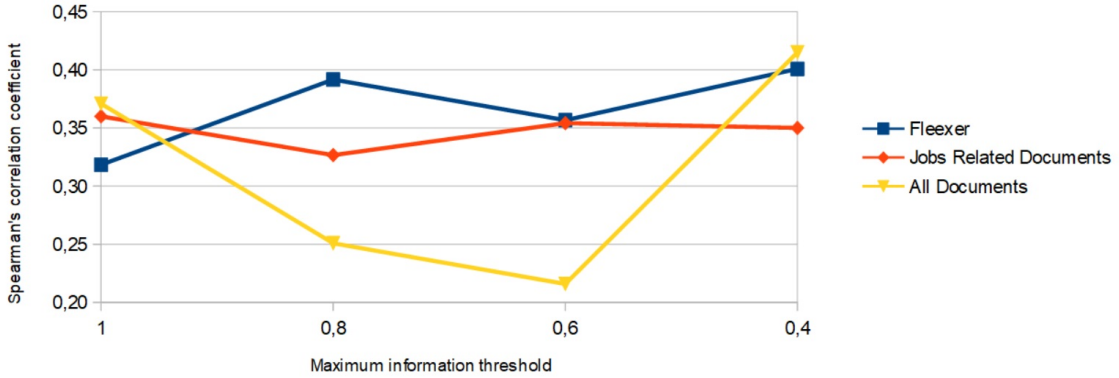

Figure 5.7: Best results by maximum information threshold for each dataset used for LDA

5.2 Cosine Similarity

As we can see in Figure 5.9, using the MI as weight decreases the performances of the cosine similarity compared to not using any weight on words. The TF-IDF clearly improved our results. We can see that the TF-IDF is slightly more efficient using job-related documents rather than all documents. We calculated using a z-test eq. (3.6) from a z-

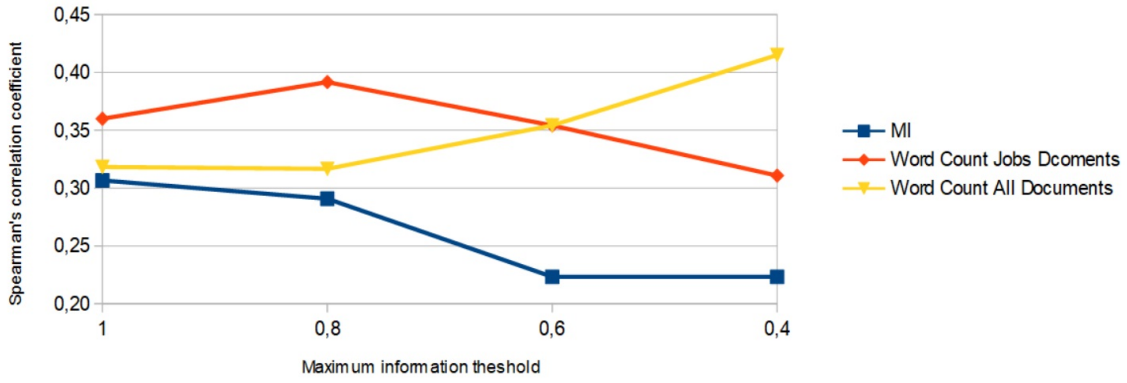

Figure 5.8: Best results by maximum information threshold by information measure for LDA

Information Measure	ρ_s
None	0.356
MI	0.168
TF-IDF (Job related documents only)	**0.511**
TF-IDF (All documents)	0.500

Figure 5.9: Cosine similarity results

score obtain with eq. (3.5) from the Fisher's Z-transform in eq. (3.4) the significance of these results and found out that the probability of TD-IDF using job-related documents being the best algorithm to be at 60%, the probability of TD-IDF using all documents being the best algorithm to be at 40% and the probability that the other two are the best algorithm to be negligible (less than 0.1%).

5.3 RV Coefficient

We first present the results of the experiment done with no limit on the maximum number of words kept per document.

As we can see in Figure 5.10, raw word count still outperforms MI as a scheme to discriminate words. Also, using all documents outperforms using only job-related documents in raw word count. We could also improve our result using MI to discriminate word over using all word . A statistical-significance z-test using eqs. (3.4), (3.5) and (3.6)

Information Measure	ρ_s	Min inf thresh	Max inf thresh
None	0.303	na	na
MI	0.415	0%	40%
Raw word count (Job related documents only)	0.428	10%	100% tie 80%
Raw word count (All documents)	**0.469**	15%	100%

Figure 5.10: RV coefficient results

reveals that the probability that the best words removing scheme is: raw word count on all documents being at 75%, raw word count using job-related documents at 16%, using MI at 9% and using all words is negligible.

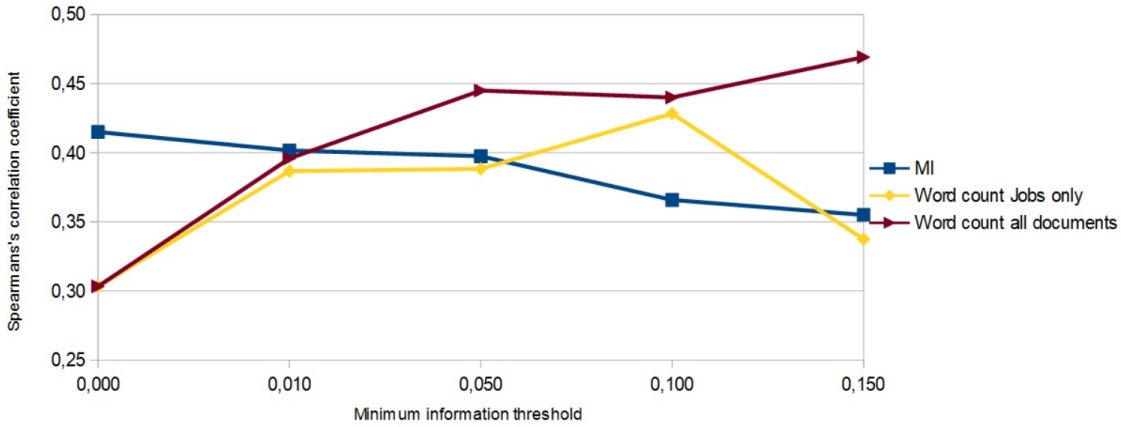

Figure 5.11: Best results varying the minimum words selector threshold for RV coefficient

As we can see in Figure 5.11, when we increase the minimum information threshold, the results are getting better using raw word count and worsen using MI. We can also see in Figure 5.12 that diminishing the maximum threshold impacts the results using word count only slightly negatively while improving the results using MI.

Now, we will discuss the results of the RV coefficient using a maximum number of words per document. If we compare the results in Figure 5.10 with those in Figure 5.13, we can see little to no difference in the results. Using MI to discriminate words did not improve our results at all from that experiment. Using raw word count on job-related documents only improved the performance from 0.428 to 0.430 and using raw word

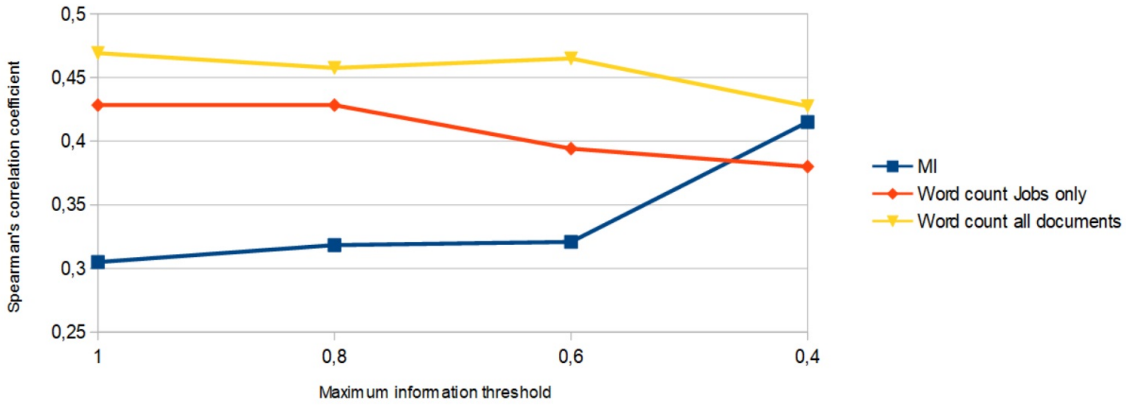

Figure 5.12: Best results varying the maximum words selector threshold for RV coefficient

Information Measure	ρ_s	Min inf thresh	Max inf thresh	max nb. of words
MI	0.415	0%	40%	100
Raw word count (Job related documents only)	0.430	10%	100% tie 80%	100
Raw word count (All documents)	**0.472**	15%	60%	50

Figure 5.13: Best results using a maximum number of words on RV coefficient

count on all documents improved the performance from 0.469 to 0.472. The chances that putting a limit on the number of words used will improve our results from raw word count using job-related documents and all documents are 52% and 53%, respectively.

We can see in Figure 5.14 that the RV coefficient performs better when given more words. These results are surprising for us since we know that the results of the RV coefficient depend on the number of rows of the two matrices under comparison. It might be due to the fact that our word embedded vector is quite large (300 values) thus the resulting matrices that are compared in eq. (3.38) are also very large (300×300) which might make our implementation of the RV coefficient sensible to a higher number of words that the vast majority of our compared documents contain.

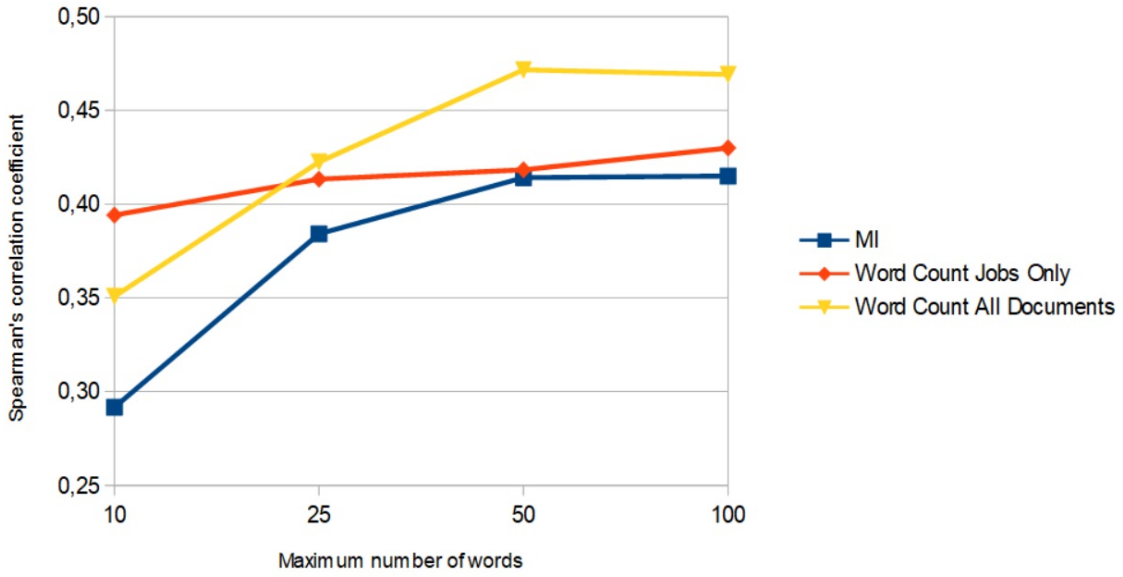

Figure 5.14: Best results varying the maximum number of words by documents for RV coefficient

5.4 LSTM

Last layer	ρ_s	NB pre-trained epoch	NB fine tuning epoch	cell dimension
Cosine similarity	**0.320**	2	5	400
Linear regression	0.225	2	100	200

Figure 5.15: Best results LSTM

In Figure 5.15 we can see that using the cosine similarity on the hidden layers gave us a ρ_s of 0.320 and using a linear regression on the hidden layers gave us a ρ_s of 0.225. In this experiment, we are confident that the cosine similarity outperforms the linear regression with a statistical confidence of 96% using a z-test from eqs. (3.4), (3.5) and (3.6).

In Figure 5.16 we can see that using larger LSTMs cells improved our result. It is also worth noting that the performance gain is much bigger on the cosine similarity than for the linear regression as the last layer. This might be due to the fact that since we have so little labeled data, the linear regression tends to overfit our data.

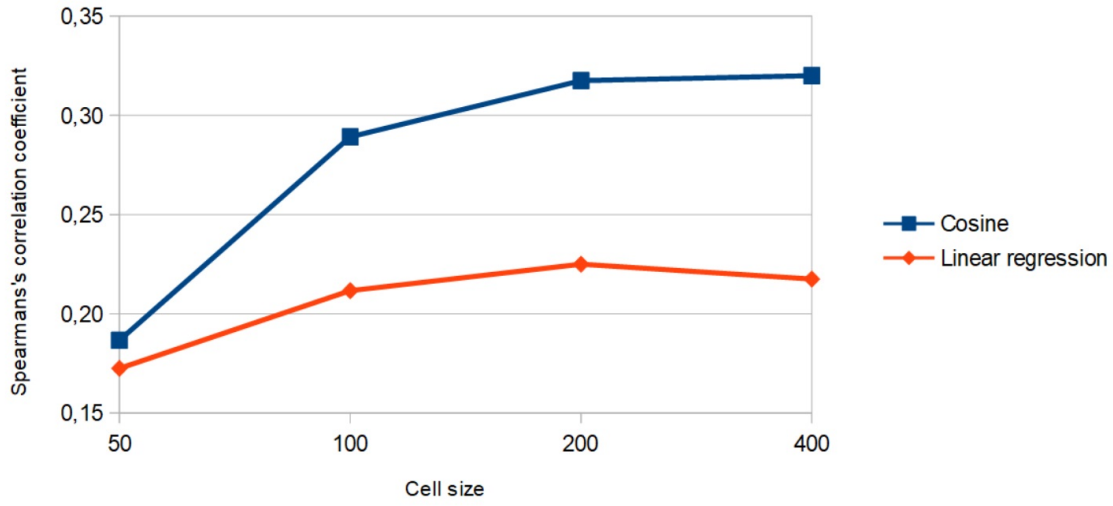

Figure 5.16: Best results by cell size for LSTM

In Figure 5.17 we can see that the performance decreased when we added more pre-trained epoch. There are two possible explanations for this behavior. First, that the task that we trained our LSTMs on is not suitable for pre-training LSTMs for our final task. Second, the LSTMs are overfitting to our initial task.

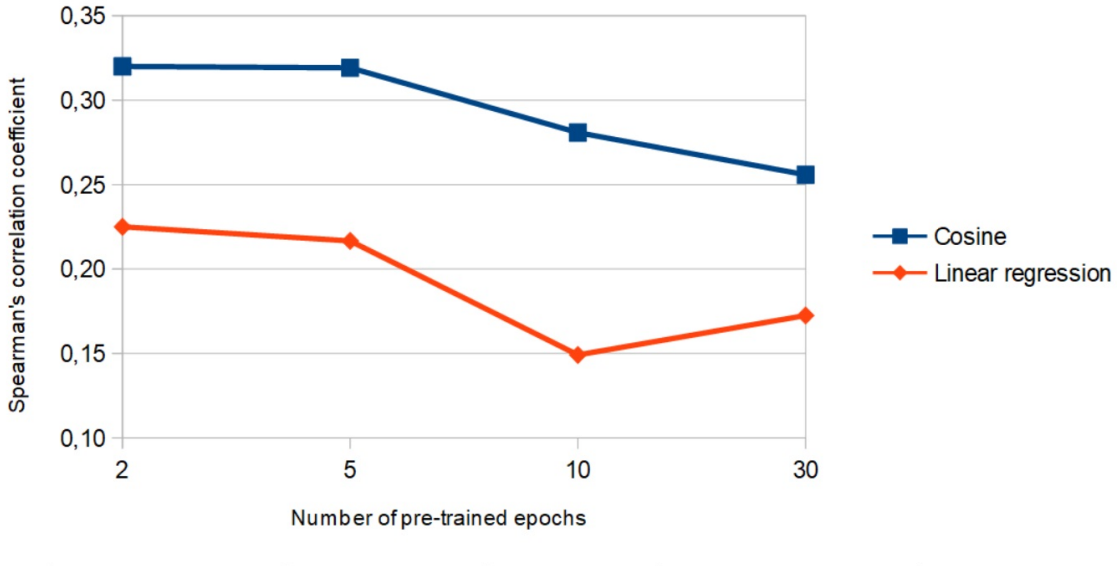

Figure 5.17: Best results by number of pre-trained epochs

To test the effect of fine-tuning, we looked at the average performance gain for a given number of epochs. We can see in Figure 5.18 that fine-tuning does not have a significant

impact on the performance. When using the cosine similarity as the last layer, we can see that the fine-tuning bring little gain with 5 or 10 epochs then the results get generally worse. We can see the opposite behavior using a linear regression as the last layer.

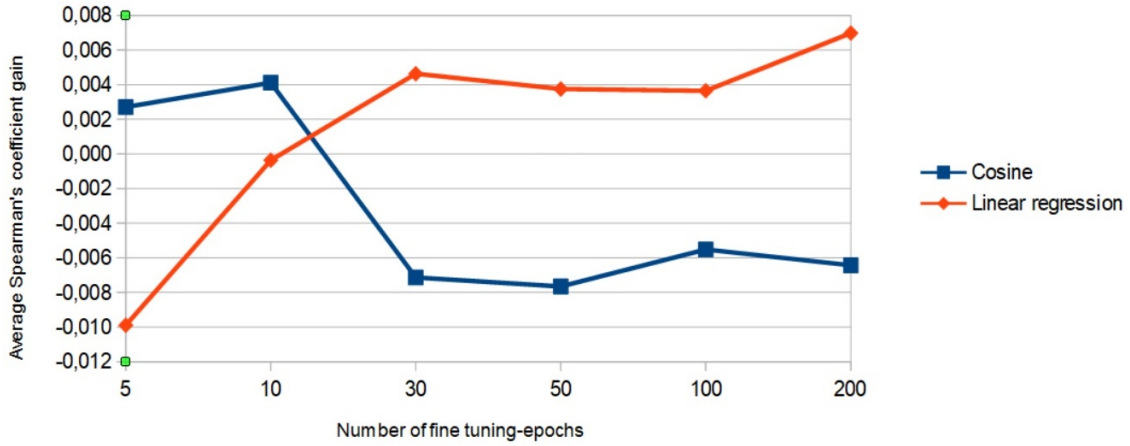

Figure 5.18: Average performance gain by number of fine-tuning epoch

5.5 Result Summary

Finally, we compare here all our results. The best ρ_s obtained by each similarity measure are:

Cosine Similarity	**0.511**
RV Coefficient	0.472
LDA	0.415
LSTM	0.320

We can see that the cosine similarity gave us the best performance followed by the RV coefficient, LDA and finally our LSTM. As we can see, the simplest the similarity measure is, the better our results are which makes sense since we worked with so little data. A statistical significance test reveals that the chances that the cosine similarity gives the best result are 81%, 18% for the RV coefficient, 1% for LDA and is negligible for the LSTM.

Chapter 6

Conclusion and Future Work

In this thesis, we built a CB recommender system. As information measures, we have used TF-IDF and MI as weighting schemes as well as raw word count and MI as word discriminators. To build our CB recommender system, we tested four similarity measures: LDA, the cosine similarity, the RV coefficient and LSTMs. We have used external databases to improve performance on several similarity and information measures. In this section, we will share our findings and discuss approaches for future work.

As a weighting scheme, we have shown that using MI didn't work very well. In fact, it decreased the performance of the cosine similarity. MI put more weight on words that are more likely to appear in job-related documents than in general documents. Those words are not necessarily useful to match a project with an applicant. For instance, as expected, words such as *salary*, *position*, *federal* and *paperwork* are some of the words that get the highest values. We also observed that in a job-related document, the author is more likely to write in the first person and in the plural form such as word as: *us*, *we* or *our*. Therefore, some very common words might be given a high weight even if they do not provide any knowledge about the general meaning of a document. Since TF-IDF improved performance by adding weight on the words that occur less often, we might presume that the term $p(x, x')$ in the MI equation (3.23) would negatively impact our result. Without that term, very rare words that only appear in one class of document will

have a disproportionately high value. We tested that hypothesis by removing that term and the ρ_s fell from 0.168 to 0.110 using the cosine similarity. MI also systematically and significantly underperformed raw word count as a word discriminant scheme on LDA and RV coefficient. In the end, it is always hard to analyze the results when using MI since useful words can end up with either a high or low value just as non-useful words. We think that MI should be avoided as a weighting scheme and word discriminant scheme in a document-document similarity measure.

As a weighting scheme, we have shown that TF-IDF performs very well. TF-IDF improved the cosine similarity performance from a ρ_s of 0.356 to 0.511, a significant change. If we look at the effect of using all documents versus using only the job-related documents, the difference is not that significant anymore. The ρ_s is 0.500 using all documents against 0.511 using job-related documents. These results prove that putting more weight on rare words works. We think that other TF and IDF schemes have the potential of improving the performance of the cosine similarity.

Raw word count was used in LDA and in the RV coefficient as a word removal scheme. Contrary to TF-IDF, we saw that raw word count performed better using all documents rather than job-related documents. On LDA, our best result using all documents gave us a ρ_s of 0.415 against 0.392 using the job-related document. On the RV coefficient, our best result using all documents was a ρ_s of 0.472 against 0.430 using job-related documents.

The best similarity is the cosine similarity. Our best run was achieved using TF-IDF on job-related documents as a weighting scheme which gave us a ρ_s of 0.511. Our results suggest that the cosine similarity is quite sensitive to the quality of the weighting scheme used.

The next best similarity measure was the RV coefficient with Stanford's pre-trained GLOVE word embedding [2]. Using all words, we achieved a ρ_s of 0.303. Removing words using raw word count on all documents and limiting the number of words per document to 50 gave the best result with a ρ_s of 0.472. For the RV coefficient, we saw

that the higher the minimum information threshold on raw word count is, the more the performance increased. We also saw that the more we decreased the maximum information threshold, the more the performances decreased. We also saw the opposite trend using MI. These results suggest that the RV coefficient is particularly robust when comparing documents using the rarest occurring words on those documents. Since the result of the RV coefficient depends on the number of rows of each matrix, we thought that limiting the number of words per documents would have a positive impact on performances. We found out that it barely increased the performance using raw word count (increased the ρ_s by 0.02 and 0.03 using respectively the job-related documents and all documents) and made no difference using MI. Since the RV coefficient is relatively robust on rare occurring words, it would be interesting to see the effect using other word embedded vectors that are tuned to be accurate on rare occurring words such as the word2vec trained on words and sub-words of Bojanowski et al. [17].

The third best similarity measure is the LDA. For the LDA, we tested the effect of: discriminating words using MI and raw word count, training on three different corpora and varying the number of topics. Our best result was found when we trained on all documents, we used raw word count on all document to discriminate words, we kept only the 40% most common words and used 100 topics. On this configuration, we obtained a ρ_s of 0.415. To discriminate words, just as for the RV coefficient, we found out that MI performed poorly and raw word count performed slightly better when using all documents against using job-related documents for a best performance of 0.307, 0.415 and 0.392, respectively, using the Spearman's correlation coefficient. We also found out that training LDA on all documents performed best followed by training LDA on Fleexer's dataset then training LDA on job-related datasets. We cannot explain why training LDA on all documents increased the performance while training LDA on job-related documents decreased our performance. Using a statistical significance analysis revealed that the chance that training on Fleexer's database gives the best result is 36%. LDA may perform better without being trained on external datasets. Contrary to the

RV coefficient, we found out that LDA performed better when trained using the most occurring words and removing the rarest words. This result suggests that the LDA needs a word to be in a multitude of documents to accurately assign a topic probability distribution to it. Finally, we also found out that the LDA performance peaks when using 100 topics. In the literature, other versions of LDA exist that could improve our result such as adding a Dirichlet prior to the topic distribution per words which should be considered for future work [10].

Finally, the worst similarity measure was the LSTM. Our best score was achieved using the cosine similarity as the last layer, 2 training epochs on the initialization task and 5 fine-tuning epochs and the LSTM cells had 400 dimensions. On our tests, the cosine similarity clearly outperformed the linear regression as the last layer of our LSTMs. We think that the linear regression overfitted and therefore we would need more labeled data to make it work properly. We observed that bigger cell sizes tend to improve our results on both architectures. We also observed that our results worsen when we added more initial training epochs. This might be the case because our initial task is too different from our final task, thus the network overfits to the initial task. Fine-tuning did not improve the performance of our network significantly, this might be occurring because our labeled dataset is too small therefore the network is overfitting. We think that the LSTM can be used for such a task but we would need more labeled data and also a better initial task.

To summarize, in this thesis, we demonstrated that we can use external databases to improve the performance of a similarity measure for an NLP application. As a weighting scheme, we observed slightly better performance using the job-related dataset than all our datasets using TF-IDF on the cosine similarity. Using all our datasets to train LDA improved our result against using only Fleexer's data but using job-related document performed worst. When we used raw word count as a word discriminant scheme, in all our tests, using all documents performed better than using the job-related dataset. In

most of our tests, using all documents resulted in better performances than using only the job-related documents.

Finally, we demonstrated that the simplest similarity measure works best on small labeled datasets. The cosine similarity paired with TF-IDF as a weighting scheme, even if it as been known for a while, works best in our case. We also demonstrated that we can use external datasets to improve our result. Since LDA works best using the most commonly occurring words and the RV coefficient works best using the rarest words, it is possible that combining their prediction could improve our result. We also think that our LSTM architectures can work but we would need either a much bigger labeled dataset or a better initial task for pre-training.

Bibliography

[1] K. Liu, et al., "Temporal learning and sequence modeling for a job recommender system," arXiv:1608.03333 [cs.LG], 2016.

[2] Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

[3] Tai, Kai Sheng, Richard Socher, and Christopher D. Manning. "Improved semantic representations from tree-structured long short-term memory networks." arXiv preprint arXiv:1503.00075 (2015).

[4] Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." International Conference on Machine Learning. 2014.

[5] Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXiv:1408.5882 (2014).

[6] Maron, Yariv, Michael Lamar, and Elie Bienenstock. "Sphere embedding: An application to part-of-speech induction." Advances in Neural Information Processing Systems. 2010.

[7] Bengio, Yoshua, et al. "A neural probabilistic language model." Journal of machine learning research 3.Feb (2003): 1137-1155.

[8] Islam, Aminul, and Diana Inkpen. "Semantic text similarity using corpus-based word similarity and string similarity." ACM Transactions on Knowledge Discovery from Data (TKDD) 2.2 (2008): 10.

[9] Griffiths, Thomas L., and Mark Steyvers. "Finding scientific topics." Proceedings of the National academy of Sciences 101.suppl 1 (2004): 5228-5235.

[10] Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." Journal of machine Learning research 3.Jan (2003): 993-1022.

[11] Thada, Vikas, and Vivek Jaglan. "Comparison of jaccard, dice, cosine similarity coefficient to find best fitness value for web retrieved documents using genetic algorithm." International Journal of Innovations in Engineering and Technology 2.4 (2013): 202-205.

[12] Sultan, Md Arafat, Steven Bethard, and Tamara Sumner. "DLS CU: Sentence Similarity from Word Alignment and Semantic Vector Composition." Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015). 2015.

[13] Marco Baroni, Georgiana Dinu, and German Kruszewski 2014. Dont 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, ACL 14, pages 238-247, Baltimore, Maryland, USA.

[14] Baskaya, Osman. "AI-KU: Using Co-Occurrence Modeling for Semantic Similarity." Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014). 2014.

[15] Gao, Jian-Bo, Bao-Wen Zhang, and Xiao-Hua Chen. "A WordNet-based semantic similarity measurement combining edge-counting and information content theory." Engineering Applications of Artificial Intelligence 39 (2015): 80-88.

[16] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[17] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 5, 135-146.

[18] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).

[19] Robert, Paul, and Yves Escoufier. "A unifying tool for linear multivariate statistical methods: the RV-coefficient." Applied statistics (1976): 257-265.

[20] Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Skip-thought vectors. In Advances in neural information processing systems (pp. 3294-3302).

[21] Bennett, James, and Stan Lanning. "The netflix prize." Proceedings of KDD cup and workshop. Vol. 2007. 2007.

[22] Zhou, Yunhong, et al. "Large-scale parallel collaborative filtering for the netflix prize." International Conference on Algorithmic Applications in Management. Springer, Berlin, Heidelberg, 2008.

[23] Koren, Yehuda. "Factorization meets the neighborhood: a multifaceted collaborative filtering model." Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2008.

[24] Linden, Greg, Brent Smith, and Jeremy York. "Amazon. com recommendations: Item-to-item collaborative filtering." IEEE Internet computing 1 (2003): 76-80.

[25] Abel, Fabian, et al. "Recsys challenge 2016: Job recommendations." Proceedings of the 10th ACM Conference on Recommender Systems. ACM, 2016.

[26] Diaby, Mamadou, Emmanuel Viennet, and Tristan Launay. "Toward the next generation of recruitment tools: an online social network-based job recommender system." Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on. IEEE, 2013.

[27] Mooney, Raymond J., and Loriene Roy. "Content-based book recommending using learning for text categorization." Proceedings of the fifth ACM conference on Digital libraries. ACM, 2000.

[28] Lu, Yao, Sandy El Helou, and Denis Gillet. "A recommender system for job seeking and recruiting website." Proceedings of the 22nd International Conference on World Wide Web. ACM, 2013.

[29] Ghazanfar, Mustansar, and Adam Prugel-Bennett. "An improved switching hybrid recommender system using Naive Bayes classifier and collaborative filtering." (2010).

[30] Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information processing & management, 24(5), 513-523.

[31] McCallum, Andrew, and Kamal Nigam. "A comparison of event models for naive bayes text classification." AAAI-98 workshop on learning for text categorization. Vol. 752. No. 1. 1998.

[32] Amelia Peacock, March 14, 2017, https://clutch.co/web-designers/resources/small-business-2017-website-survey

[33] statistica.com 2018-12-19 https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide

[34] Wilson, M. (1988). MRC psycholinguistic database: Machine-usable dictionary, version 2.00. Behavior Research Methods, Instruments, & Computers, 20(1), 6-10.

[35] Lebret, R., & Collobert, R. (2013). Word embeddings through hellinger PCA. arXiv preprint arXiv:1312.5542.

[36] Li, Y., Xu, L., Tian, F., Jiang, L., Zhong, X., & Chen, E. (2015, June). Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In Twenty-Fourth International Joint Conference on Artificial Intelligence.

[37] Goldberg, Y., & Levy, O. (2014). word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722.

[38] Shannon, Claude E. "Coding theorems for a discrete source with a fidelity criterion." IRE Nat. Conv. Rec 4.142-163 (1959): 1.

[39] Church, Kenneth Ward, and Patrick Hanks. "Word association norms, mutual information, and lexicography." Computational linguistics 16.1 (1990): 22-29.

[40] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

[41] Pascanu, R., Mikolov, T., & Bengio, Y. (2012). Understanding the exploding gradient problem. CoRR, abs/1211.5063, 2.

[42] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), 157-166.

[43] Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), 1527-1554.

[44] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P. A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of machine learning research, 11(Dec), 3371-3408.

[45] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

[46] Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.

[47] Najafabadi, Maryam Khanian, Azlinah Hj Mohamed, and Mohd Naz'ri Mahrin. "A survey on data mining techniques in recommender systems." Soft Computing (2017): 1-28.

[48] Zhao, Lili, Sinno Jialin Pan, and Qiang Yang. "A unified framework of active transfer learning for cross-system recommendation." Artificial Intelligence 245 (2017): 38-55.

[49] Zhang, Qian, et al. "A cross-domain recommender system with consistent information transfer." Decision Support Systems 104 (2017): 49-63.

[50] Christiane Fellbaum (1998, ed.) WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press.

[51] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

[52] Eigentaste: A Constant Time Collaborative Filtering Algorithm. Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Information Retrieval, 4(2), 133-151. July 2001.

[53] Improving Recommendation Lists Through Topic Diversification, Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, Georg Lausen; Proceedings of the 14th International World Wide Web Conference (WWW '05), May 10-14, 2005, Chiba, Japan.

[54] Cantador, I., Brusilovsky, P. L., & Kuflik, T. (2011). Second workshop on information heterogeneity and fusion in recommender systems (HetRec2011).

[55] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011), 2011.

[56] Irie, K., Tüske, Z., Alkhouli, T., Schlüter, R., & Ney, H. (2016, September). LSTM, GRU, Highway and a Bit of Attention: An Empirical Overview for Language Modeling in Speech Recognition. In Interspeech (pp. 3519-3523).

[57] McCAffrey, James "Understanding LSTM Cells Using C#" April 2018, Volume 33 Number 4 msdn.microsoft.com/en-us/magazine/mt846470.aspx

[58] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin "Placing Search in Context: The Concept Revisited" ACM Transactions on Information Systems, 20(1):116-131, January 2002

[59] Minh-Thang Luong, Richard Socher, and Christopher D. Manning. 2013. Better word representations with recursive neural networks for morphology. In Proceedings of the Seventeenth Conference on Computational Natural Language Learning, pages 104–113. Association for Computational Linguistics.

[60] Rubenstein and Goodenough, 1965, H. Rubenstein, J.B. Goodenough Contextual correlates of synonymy Commun. ACM, 8 (1965), pp. 627-633

[61] Ng, Andrew Y., and Michael I. Jordan. "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes." Advances in neural information processing systems. 2002.

[62] M. Marelli, S. Menini, M. Baroni, L. Bentivogli, R. Bernardi and R. Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. Proceedings of LREC 2014, Reykjavik (Iceland): ELRA.

[63] Dolan, William B., and Chris Brockett. "Automatically constructing a corpus of sentential paraphrases." Proceedings of the Third International Workshop on Paraphrasing (IWP2005). 2005.

[64] A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts", Proceedings of the ACL, 2004.

[65] Janyce Wiebe, Theresa Wilson , and Claire Cardie (2005). Annotating expressions of opinions and emotions in language. Language Resources and Evaluation, volume 39, issue 2-3, pp. 165-210.

[66] Diaz, Fernando, trec-data, 2018, https://github.com/diazf/trec-data

[67] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár,and C Lawrence Zitnick. Microsoft coco: Common objects in context. InECCV, pages 740–755. 2014

[68] Shaoul, C. & Westbury C. (2010) The Westbury Lab Wikipedia Corpus, Edmonton, AB: University of Alberta (downloaded from http://www.psych.ualberta.ca/ westburylab/downloads/westburylab.wikicorp.download.html)

[69] Mou, Lili, et al. "How transferable are neural networks in nlp applications?." arXiv preprint arXiv:1603.06111 (2016).

[70] Mueller, Jonas, and Aditya Thyagarajan. "Siamese recurrent architectures for learning sentence similarity." Thirtieth AAAI Conference on Artificial Intelligence. 2016.

[71] Wei, Xiaocong, et al. "A convolution-LSTM-based deep neural network for cross-domain MOOC forum post classification." Information 8.3 (2017): 92.

[72] Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Pasca, Aitor Soroa, A Study on Similarity and Relatedness Using Distributional and WordNet-based Approaches, In Proceedings of NAACL-HLT 2009.

[73] A. Miller, George & G. Charles, Walter. (1991). Contextual Correlates of Semantic Similarity. Language and Cognitive Processes. 6. 1-28. 10.1080/01690969108406936.

[74] Ekbal, Asif, et al. "Language independent named entity recognition in indian languages." Proceedings of the IJCNLP-08 Workshop on Named Entity Recognition for South and South East Asian Languages. 2008.

[75] Spearman, C. (1904). The proof and measurement of association between two things. American journal of Psychology, 15(1), 72-101.

[76] Josse, Julie, Jérome Pagès, and François Husson. "Testing the significance of the RV coefficient." Computational Statistics & Data Analysis 53.1 (2008): 82-91.

[77] Hughes, Gordon. "On the mean accuracy of statistical pattern recognizers." IEEE transactions on information theory 14.1 (1968): 55-63.

[78] Myers, Leann, and Maria J. Sirois. "Spearman correlation coefficients, differences between." Encyclopedia of statistical sciences 12 (2004).

[79] Vozalis, Manolis, and Konstantinos G. Margaritis. "On the enhancement of collaborative filtering by demographic data." Web Intelligence and Agent Systems: An International Journal 4.2 (2006): 117-138.

[80] Melville, Prem, Raymond J. Mooney, and Ramadass Nagarajan. "Content-boosted collaborative filtering for improved recommendations." Aaai/iaai 23 (2002): 187-192.

[81] Flach, Peter, and Meelis Kull. "Precision-recall-gain curves: PR analysis done right." Advances in neural information processing systems. 2015.

[82] Uriarte-Arcia, Abril Valeria, Itzamá López-Yáñez, and Cornelio Yáñez-Márquez. "One-hot vector hybrid associative classifier for medical data classification." PloS one 9.4 (2014): e95715.

[83] Plisson, Joël, Nada Lavrac, and Dunja Mladenic. "A rule based approach to word lemmatization." Proceedings of IS-2004 (2004): 83-86.

[84] Hearst, Marti. "Noun homograph disambiguation using local context in large text corpora." Using Corpora (1991): 185-188.

[85] Mukherjee, Shubhadeep, and Pradip Kumar Bala. "Detecting sarcasm in customer tweets: an NLP based approach." Industrial Management & Data Systems 117.6 (2017): 1109-1126.

[86] Cambria, Erik, et al. "New avenues in opinion mining and sentiment analysis." IEEE Intelligent systems 28.2 (2013): 15-21.

[87] Microsoft, (2019, August 07) Microsoft Translator Speech API, Retrieved from https://azure.microsoft.com/en-ca/services/cognitive-services/translator-speech-api/

[88] AlexPoint, Ayushoriginal, (2019, August 07) LemmaGenerator repository, Retrieved from https://github.com/AlexPoint/LemmaGenerator

[89] Peters, Matthew E., et al. "Deep contextualized word representations." arXiv preprint arXiv:1802.05365 (2018).

[90] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2016.