RESUME SHORTLISTING

NEHAL M
Roll No. 16PD22

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

FIVE YEAR INTEGRATED
M.Sc. DATA SCIENCE
OF ANNA UNIVERSITY

NOVEMBER 2019

DEPARTMENT OF APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCES

PSG COLLEGE OF TECHNOLOGY
(Autonomous Institution)
COIMBATORE – 641 004.

PSG COLLEGE OF TECHNOLOGY
(Autonomous Institution)
COIMBATORE – 641 004.

Seventh Semester
Project work

RESUME SHORTLISTING

Bona fide record of work done by

NEHAL M
Roll No. 16PD22

Submitted in partial fulfillment of the requirements for the degree of

FIVE YEAR INTEGRATED
M.Sc. DATA SCIENCE
Of Anna University

NOVEMBER 2019

_______________ _______________________
Academic Guide **Head of the Department**

Submitted for the Viva-Voce Examination held on _________________

________________ _________________
Internal Examiner **External Examiner**

Contents

ACKNOWLEDGEMENT

I am extremely grateful to **Dr. K. Prakasan**, Principal In Charge, PSG College of Technology, Coimbatore for giving me this opportunity to do this project at Indian Institute of Management, Bengaluru.

I am very much thankful to **Dr. R. Nadarajan**, Professor and Head, Department of Applied Mathematics and Computational Sciences, PSG College of Technology, Coimbatore, for his continual support and guidance throughout the project tenure.

I extend my heartfelt gratitude to my internal guide, **Dr. N.K. Sreeja**, Assistant Professor, Department of Applied Mathematics and Computational Sciences, PSG College of Technology, Coimbatore, for her exemplary guidance, ardent motivation and constant support throughout this project.

I am deeply indebted to my external guide **Dr.U. Dinesh Kumar**, Professor and Head, Department of Decision Sciences, Indian Institute Of Management, Bengaluru, for his constant motivation, guidance, continuous support and help in various aspects to complete this project work.

I am indebted and thankful to my external guide **Mr. Naveen Kumar Bhansali**, Advisory Consultant, Dell EMC, Bengaluru and Visiting Faculty , Indian Institute Of Management, Bengaluru for his constant motivation, guidance, continuous support and help in various aspects to complete this project work.

Finally, I express my gratefulness to all the staff members of the Department of Applied Mathematics and Computational Sciences, PSG College of Technology, Coimbatore, and my family and friends for their encouragement and support.

SYNOPSIS

Finding a solution to automatically recruit workers for a job is one of the major business analytics problem in the industry. The same scenario can be compared to that of the admission process in an academic institution. Going through each resume manually and shortlisting them is an arduous process. This manual process can be automated by employing Natural Language Processing and Deep Learning techniques.

The aim of this project is to improve the efficiency of the recruiter team. This report focuses on building a pipeline for evaluating a pool of resumes and rank order them against a given academic requirement description.

Complete automation of the recruitment may not be plausible as it requires domain expertise but the laborious process of going through all resumes at the initial stage can be automated so that the evaluator just need to recruit from a small pool of desirable resumes.

The pipeline includes four major blocks namely pre-processing , embedding ,extracting and short-listing. Data pre-processing is done by Natural Language Processing techniques. Glove word embedding is employed to vectorize the corpus of word. Feature extraction is done using the concept of Named Entity Recognition. Finally the candidates are shortlisted by the Word Movers Distance method.

CHAPTER 1

INTRODUCTION

1.1 PROBLEM OVERVIEW

Every year Business Analytics and Intelligence course under the department of Decision Sciences at IIM, Bengaluru receives nearly thousands of applications. Out of these only 60 to 70 applicants are selected. IIMB being one of the top B-schools in the country has acceptance rate of less than two percent. Decision science which is currently ruling the industry is becoming popular and the number of applications for admission, exponentially increases every year.

This bottle neck shortlisting process requires a great deal of manual work. Since the selectors are the Professors of the same institution their valuable time is wasted. This project aims at creating a pipeline which will automatically shortlist candidates to the interview round thereby saving a lot of time, effort and human errors.

1.2 LITERATURE REVIEW

In [1], a toolkit named as "Learning Pinocchio (LP)2", was applied on resumes to learn Information Extraction rules from resumes written in English. The information identied in their task includes a at structure of Name, Street, City, Province, Email, Telephone, Fax and Zip code. Rules are learned by generalizing over a set of examples marked via XML tags in a training corpus.

An approach has been proposed in [2] for resume information extraction to support automatic resume management and routing. A cascaded information extraction (IE) framework is designed. In the rst pass, a resume is segmented into consecutive blocks attached with labels indicating the information types. Then, in the second pass, the detailed information, such as Name and Address, are identied in certain blocks (e.g.

blocks labelled with Personal Information), instead of searching in the entire resume. Based on the requirements of an ongoing recruitment management system which incorporates database construction with IE technologies and resume recommendation (routing), general information elds like Personal Information, Education etc. are dened.

In [3], an approach has been proposed in which each object is assigned a degree of being an outlier, which is called local outlier factor. It is local as the degree depends on how isolated the object is with respect to the surrounding neighborhood. In [4], an approach has been proposed to mine the distance based outliers. The notion of K-nearest neighbour has been used to identify outliers in [5].

1.3 SYSTEM CONFIGURATION

1.3.1 Hardware Configuration

- **Processor** - Intel® Core™ i7-4900MQ CPU@2.80GHz
- **Main Memory** - 16 GB
- **Secondary Memory** - 1TB

1.3.2 Software Configuration

- **Operating System** - Windows 10 Professional
- **Programming Language** - Python 3.5
- **Deep Learning Framework** - TensorFlow 2.0

CHAPTER 2

NATURAL LANGUAGE PROCESSING

Natural Language Processing, is the sub-field of Artificial Intelligence that is focused on enabling computers to understand and process human languages. In certain limited areas, what we can do with NLP already seems like magic. Doing anything complicated in machine learning usually means building a pipeline. The idea is to break up the problem into very small pieces and then use machine learning to solve each smaller piece separately. Then by chaining together several machine learning models that feed into each other, complicated things can be done. There are few common steps that are applied to almost all NLP problems which are mentioned in this chapter.

2.1 SENTENCE SEGMENTATION

The first step is to break the text apart into separate sentences. Basic assumption is that each sentence is a separate thought or idea. It is a lot easier to write a program to understand a single sentence than to understand a whole paragraph. Coding a Sentence Segmentation model can be as simple as splitting apart sentences whenever a punctuation mark is encountered.

2.2 WORD TOKENIZATION

Now that the document is split into sentences, it can be processed one at a time. The next step in the pipeline is to break this sentence into separate words or tokens. This is called tokenization. Tokenization is easy to do in English. Whenever a space or a punctuation mark is encountered in the sentence, it is splitted into separate words.

2.3 PREDICTING PARTS OF SPEECH

Next, each tokens's part of speech is guessed, whether it is a noun, a verb, an adjective and so on. By knowing the role of each word in the sentence, context of the sentence can be figured out. This can be done by feeding each word (and some extra words around it for context) into a pre-trained part-of-speech classification model as depicted in Figure 2.1

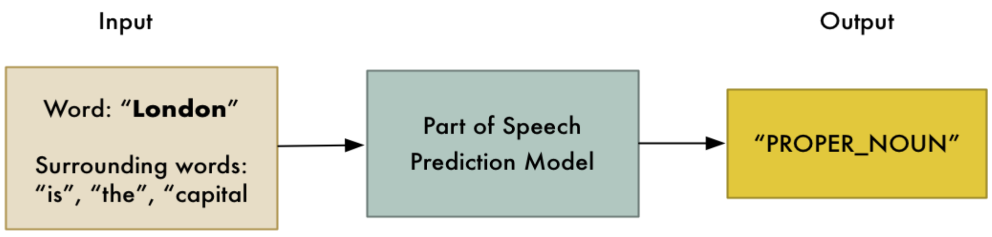

Input
Output
Word: "London"
Surrounding words: "is", "the", "capital
Part of Speech Prediction Model
"PROPER_NOUN"

Figure 2.1: POS MODEL

The part-of-speech model was originally trained by feeding it millions of English sentences with each word's part of speech already tagged and having it learn to replicate that behaviour. It does not actually understand what the words mean in the same way that humans do. It just knows how to guess a part of speech based on similar sentences and words it has seen before. Figure 2.2 shows an example of POS tagging after processing the sentence.

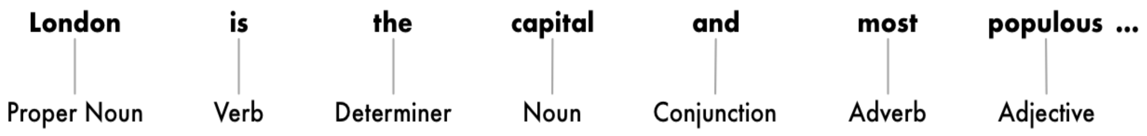


Figure 2.2: POS TAGGING

With this information itself, already some very basic meaning can be derived. For example, in Figure 2.2 the nouns in the sentence include "London" and "capital", so the sentence is probably talking about London.

2.4 TEXT LEMMATIZATION

In English (and most languages), words appear in different forms. Lemmatization is the process of figuring out the most basic form or lemma of each word in the sentence. For example consider the sentences

I had a pen.

I had two pens.

Both pen and pens denotes the same token ”pen”.

“I had two pens” becomes “I [have] two [pen].”

2.5 IDENTIFYING STOP WORDS

Next, step is to consider the importance of each words in the sentence. English has a lot of filler words that appear very frequently like “and”, “the”, and “a”. When doing statistics on text, these words introduce a lot of noise since they appear way more frequently than other words. Some NLP pipelines will flag them as stop words ,that is, words that should filtered out before doing any statistical analysis.

Stop words are usually identified by just by checking a hardcoded list of known stop words. But there is no standard list of stop words that is appropriate for all applications. The list of words to ignore can vary depending on your application.

2.6 DEPENDENCY PARSING

Sometimes it is important to figure out how all the words in our sentence are related to each other. This is called dependency parsing. The goal is to build a tree that assigns a single parent word to each word in the sentence. The root of the tree will be the main verb in the sentence. Figure 2.3 depicts the beginning of the parse tree for the sentence: ”London is the capital and most populous city”.

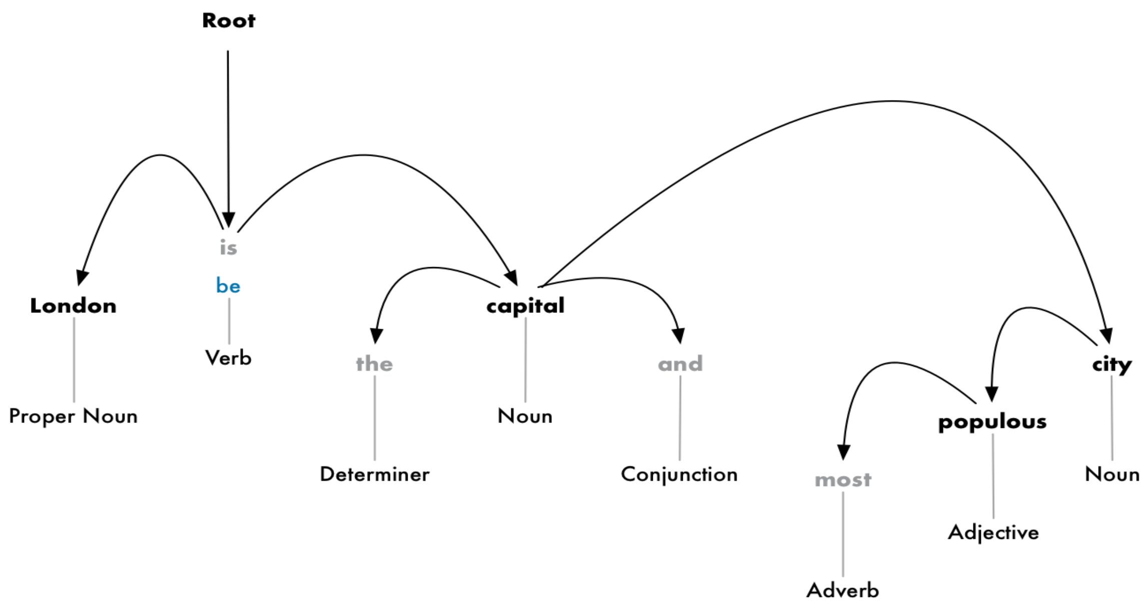


Figure 2.3: Parser Tree

In addition to identifying the parent word of each word, the type of relationship that exists between two words can also be predicted as shown in Figure 2.4.

This parse tree shows us that the subject of the sentence is the noun "London" and it has a "be" relationship with "capital".Finally something useful can be derived : London is a capital. If the tree is parsed further, more information can be derived like "London is the capital of the United Kingdom".

Just like how the parts of speech are predicted using a machine learning model, dependency parsing also works by feeding words into a machine learning model and outputting a result.

It is also important to remember that many English sentences are ambiguous and just really hard to parse. In those cases, the model will make a guess based on what parsed version of the sentence seems most likely, but it's not perfect and sometimes the model will be embarrassingly wrong. But over time NLP models will continue to get better at parsing text in a sensible way.

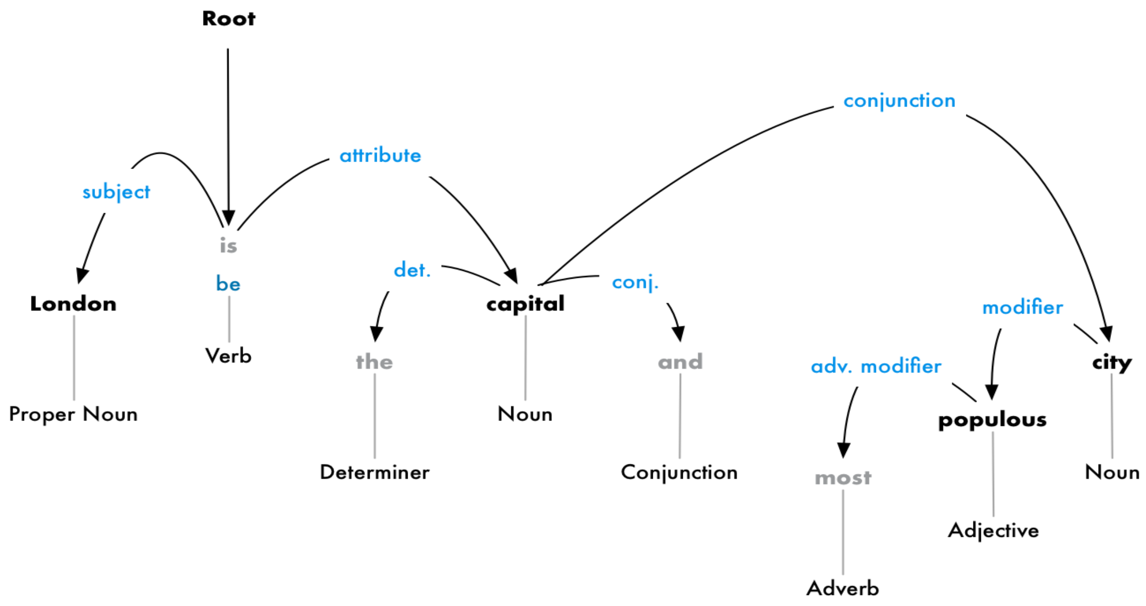


Figure 2.4: Identifying relationship from the parser tree

2.6.1 Finding Noun Phrases

So far, every word in our sentence is treated as a separate entity. But sometimes it makes more sense to group together the words that represent a single idea or thing. The information from the dependency parse tree can be used to automatically group together words that are all talking about the same thing. For example, instead of Figure 2.5

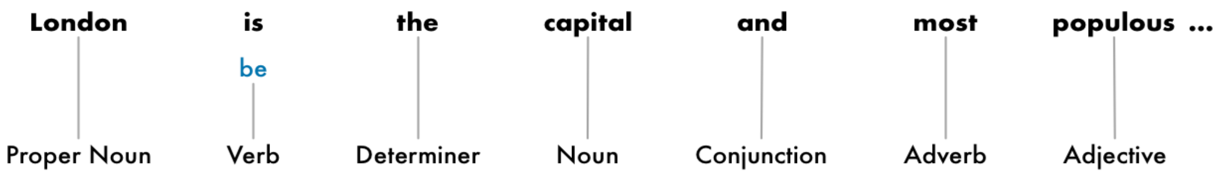


Figure 2.5: Before grouping noun phrases

The noun phrases can be grouped to generate Figure 2.6

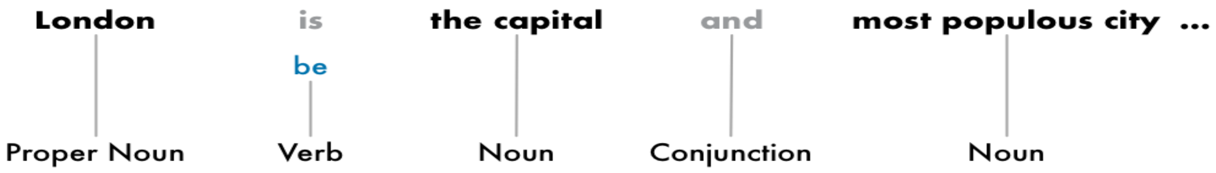


Figure 2.6: After grouping noun phrases

Whether or not to do this step depends on our end goal. But it is often a quick and easy way to simplify the sentence if extra detail about which words are adjectives are not needed and instead extracting complete ideas is our primary goal.

2.7 NAMED ENTITY RECOGNITION

Named Entity Recognition is the most useful NLP technique for extracting ideas. NER highlights nouns in a sentence as in Figure 2.7

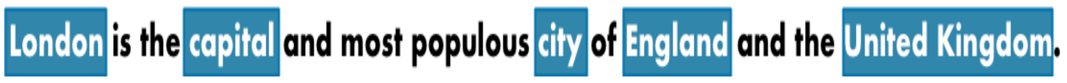

London is the capital and most populous city of England and the United Kingdom.

Figure 2.7: NER highlighting important words in a sentence

Some of these nouns present real things in the world. For example, “London”, “England” and “United Kingdom” represent physical places on a map. With that information, a list of real-world places can be extracted automatically from a document using NLP.

The goal of Named Entity Recognition, or NER, is to detect and label these nouns with the real-world concepts that they represent. Figure 2.8 depicts an example of how a sentence looks like after running each token through our NER tagging model:

But NER systems aren’t just doing a simple dictionary lookup. Instead, they are using the context of how a word appears in the sentence and a statistical model to guess which type of noun a word represents. A good NER system can tell the

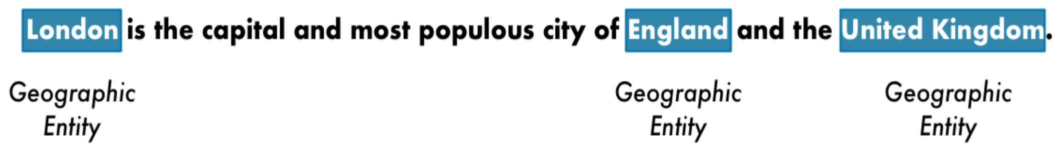

London is the capital and most populous city of England and the United Kingdom.
Geographic Entity
Geographic Entity
Geographic Entity

Figure 2.8: NER tagging

difference between "Brooklyn Decker" the person and the place "Brooklyn" using context clues.

Here are just some of the kinds of objects that a typical NER system can tag:

- People's names
- Company names
- Geographic locations
- Product names
- Dates and times
- Amounts of money
- Names of events

NER has tons of uses since it makes it so easy to grab structured data out of text. It is one of the easiest ways to quickly get value out of an NLP pipeline.

2.8 COREFERENCE RESOLUTION

At this point, already some useful representations of a sentence are derived like : Parts of speech for each word, how the words relate to each other and which words are talking about named entities .

However, there is still one big problem. English is full of pronouns, words like he, she, and it. These are shortcuts that are used instead of writing out names over and over in each sentence. Humans can keep track of what these words represent based on context. But a NLP model doesn't know what pronouns mean because it only examines one sentence at a time.

The goal of coreference resolution is to figure out mapping words by tracking pronouns across sentences. The idea is to figure out all the words that are referring to the same entity. Figure 2.9 shows an example of running coreference resolution on a document for the word "London":

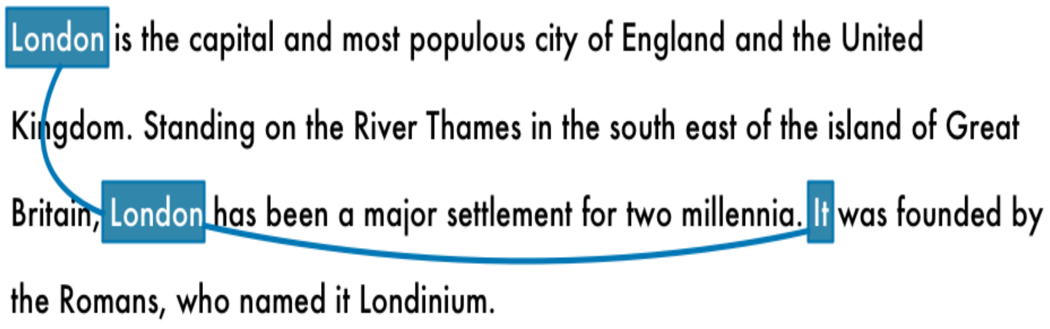

London is the capital and most populous city of England and the United Kingdom. Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia. It was founded by the Romans, who named it Londinium.

Figure 2.9: Coreference mapping

With coreference information combined with the parse tree and named entity information, a lot of information can be extracted out of a document.

2.9 NLP PIPELINE

Figure 2.10 shows the steps in a typical NLP pipeline, but these steps can be skipped or re-ordered depending on the application and how the NLP library is implemented.

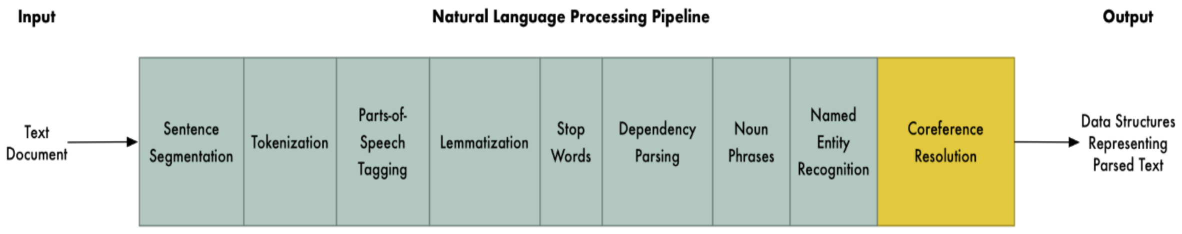


Figure 2.10: Generic NLP pipeline

CHAPTER 3

WORD EMBEDDING

3.1 WHAT ARE WORD EMBEDDINGS?

In very simplistic terms, Word Embeddings are the texts converted into numbers and there may be different numerical representations of the same text. Formally, a Word Embedding format generally tries to map a word to a vector.

3.2 NEED FOR WORD EMBEDDINGS?

As it turns out, many Machine Learning algorithms and almost all Deep Learning architectures are incapable of processing strings or plain text in their raw form. They require numbers as inputs to perform any sort of job, be it classification, regression, etc. in broad terms. With the huge amount of data that is present in the text format, it is imperative to extract knowledge out of it and build applications.

3.3 DIFFERENT TYPES OF WORD EMBEDDINGS

The different types of word embeddings can be broadly classified into two categories:

- Frequency based Embedding
- Prediction based Embedding

3.4 FREQUENCY BASED EMBEDDING

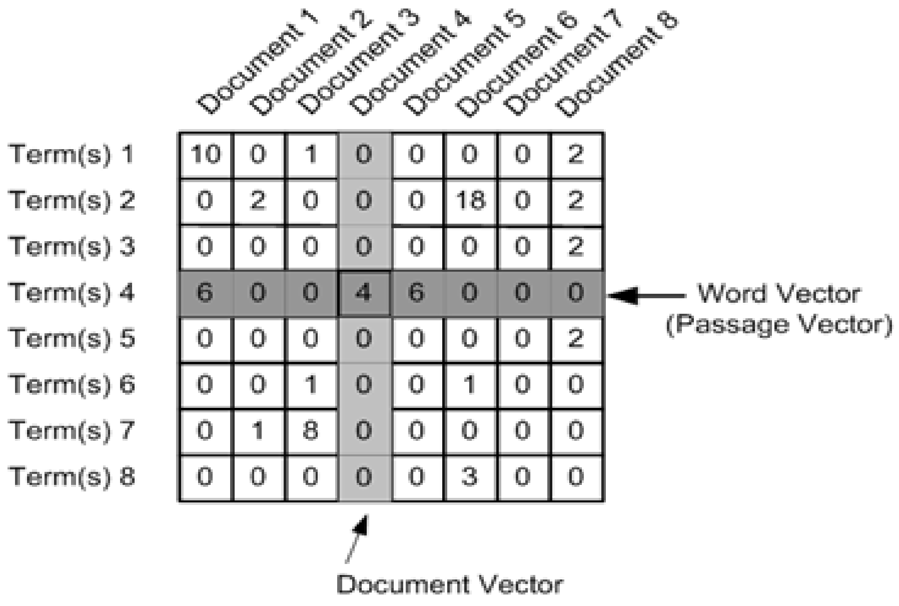
There are generally three types of vectors under this category.

- Count Vector
- TF-IDF Vector
- Co-Occurrence Vector

3.4.1 Count Vector

Consider a Corpus C of D documents d1,d2. . . . .d_D and N unique tokens extracted out of the corpus C. The N tokens will form our dictionary and the size of the Count Vector matrix M will be given by D X N. Each row in the matrix M contains the frequency of tokens in document D(i).

Figure 3.1 a representational image of the matrix M for easy understanding.



	Document 1	Document 2	Document 3	Document 4	Document 5	Document 6	Document 7	Document 8
Term(s) 1	10	0	1	0	0	0	0	2
Term(s) 2	0	2	0	0	0	18	0	2
Term(s) 3	0	0	0	0	0	0	0	2
Term(s) 4	6	0	0	4	6	0	0	0
Term(s) 5	0	0	0	0	0	0	0	2
Term(s) 6	0	0	1	0	0	1	0	0
Term(s) 7	0	1	8	0	0	0	0	0
Term(s) 8	0	0	0	0	0	3	0	0

Word Vector
(Passage Vector)
Document Vector

Figure 3.1: Count vector matrix

Now there may be quite a few variations while preparing the above matrix M. The variations will be generally in:

- the way dictionary is prepared. In real world applications, a corpus may contain millions of documents and with millions of document, hundreds of millions of unique words can be extracted. So basically, the matrix that is prepared will be a very sparse one and inefficient for any computation. An alternative to using every unique word as a dictionary element would be to pick say top 10,000 words based on frequency and then prepare a dictionary.

- the way count is taken for each word. either the frequency (number of times a word has appeared in the document) or the presence (has the word appeared in the document?)is taken to be the entry in the count matrix M. But generally, frequency method is preferred over the latter.

3.4.2 TF-IDF Vectorization

This is another method which is based on the frequency method but it is different to the count vectorization in the sense that it takes into account not just the occurrence of a word in a single document but in the entire corpus. Common words like 'is', 'the', 'a' etc. tend to appear quite frequently in comparison to the words which are important to a document. For example, a document A on "Messi" is going to contain more occurences of the word "Messi" in comparison to other documents. But common words like "the" etc. are also going to be present in higher frequency in almost every document.

Ideally, the common words occurring in almost all documents are down-weighed and more importance is given to the words that appear in a subset of documents. TF-IDF works by penalising these common words by assigning them lower weights while giving importance to words like "Messi" in a particular document.

TF-IDF definition and formulation:

TF = (Number of times term t appears in a document)/(Number of terms in the document)
It denotes the contribution of the word to the document i.e words relevant to the document should be frequent.

IDF = log(N/n),
where, N is the number of documents and n is the number of documents a term t has appeared in.

The reasoning behind IDF is deally if a word has appeared in all the document, then probably that word is not relevant to a particular document. But if it has appeared in a subset of documents then probably the word is of some relevance to the documents it is present in.

3.4.3 Co-Occurrence Matrix with a Fixed Context Window

Basic idea is, similar words tend to occur together and will have similar context.

There are two concepts that are needed to be clarified:

Co-occurrence – For a given corpus, the co-occurrence of a pair of words say w1 and w2 is the number of times they have appeared together in a Context Window.

Context Window – Context window is specified by a number and the direction. It specifies the number of words taken at a time to count the co-occurence.

Variations of Co-occurrence Matrix

Let V be the number of unique words in the corpus. So the Vocabulary size is equal to V. The columns of the Co-occurrence matrix form the context words. The different variations of Co-Occurrence Matrix are:

- A co-occurrence matrix of size V X V. Now, for even a decent corpus V gets very large and difficult to handle. So generally, this architecture is never preferred in practice.

- A co-occurrence matrix of size V X N where N is a subset of V and can be obtained by removing irrelevant words like stopwords etc. This is still very large and presents computational difficulties.

This co-occurrence matrix is not the word vector representation that is generally used. Instead, this Co-occurrence matrix is decomposed using techniques like PCA, SVD etc. into factors and combination of these factors forms the word vector representation.

Advantages of Co-occurrence Matrix

It preserves the semantic relationship between words. i.e man and woman tend to be closer than man and apple. It uses factorization which is a well-defined problem and can be efficiently solved. It has to be computed once and can be used anytime once computed. In this sense, it is faster in comparison to others.

Disadvantages of Co-Occurrence Matrix

It requires huge memory to store the co-occurrence matrix. But, this problem can be circumvented by factorizing the matrix out of the system for example in Hadoop clusters etc. and can be saved.

3.5 PREDICTION BASED VECTOR

So far, deterministic methods to determine word vectors had been discussed. But these methods proved to be limited in their word representations until Mitolov et.al introduced [6] to the NLP community. These methods were prediction based, in the sense that they provided probabilities to the words and proved to be state of the art for tasks like word analogies and word similarities. They were also able to achieve tasks like King -man +woman = Queen, which was considered a result almost magical. Word2vec is one such model that is used to generate word vectors.

Word2vec is not a single algorithm but a combination of two techniques – CBOW(Continuous bag of words) and Skip-gram model. Both of these are shallow neural networks which map word(s) to the target variable which is also a word(s). Both of these techniques learn weights which act as word vector representations.Both these methods are discussed below.

3.5.1 Continuous Bag of Words

The way CBOW work is that it tends to predict the probability of a word given a context. A context may be a single word or a group of words

Diagrammatic representation of CBOW for a singe word is depicted in Figure 3.2

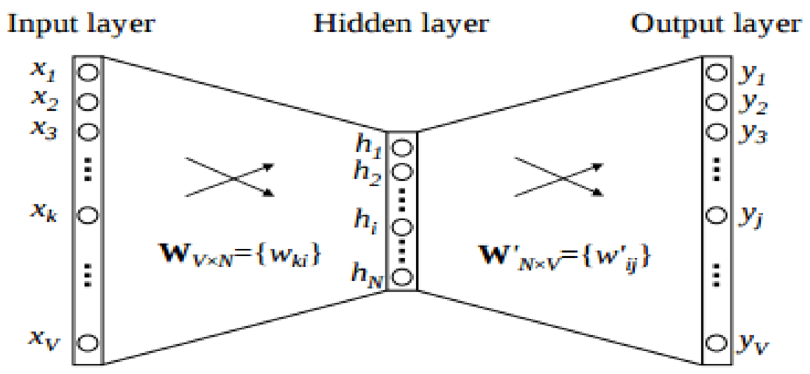


Figure 3.2: CBOW model

The flow of CBOW is as follows:

- The input layer and the target, both are one- hot encoded of size [1 X V].
- There are two sets of weights. One is between the input and the hidden layer and second between hidden and output layer. Input-Hidden layer matrix size = [V X N] , hidden-Output layer matrix size = [N X V] : Where N is the number of dimensions we choose to represent the word in. It is arbitary and a hyper-parameter for a Neural Network. Also, N is the number of neurons in the hidden layer.
- There is a no activation function between any layers.
- The input is multiplied by the input-hidden weights and called hidden activation. It is simply the corresponding row in the input-hidden matrix copied.
- The hidden input gets multiplied by hidden-output weights and output is calculated.
- Error between output and target is calculated and propagated back to re-adjust the weights.

- The weight between the hidden layer and the output layer is taken as the word vector representation of the word.

The above steps are for a single context word.

Figure 3.3 depicts the CBOW architecture for multiple context words.

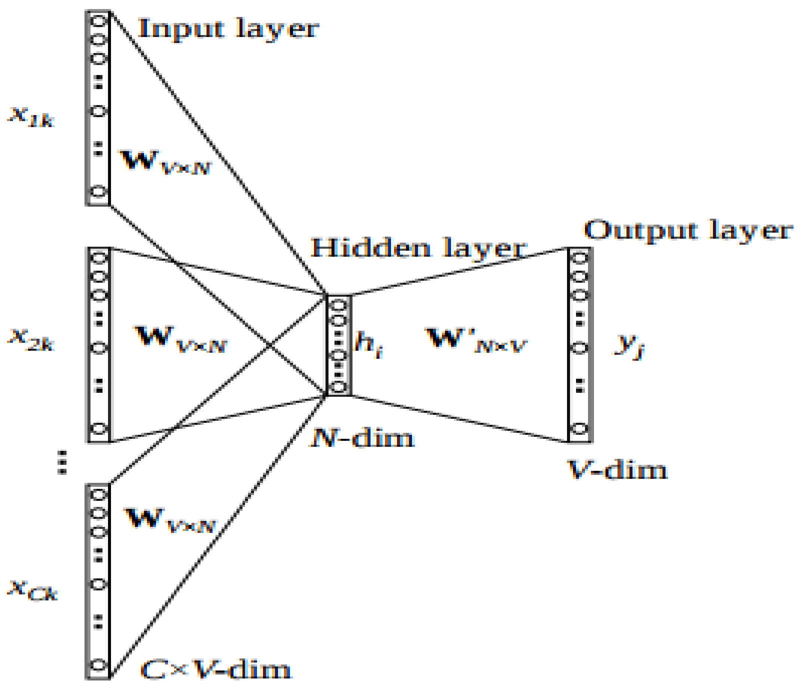


Figure 3.3: CBOW for multiple words

Being probabilistic is nature, it is supposed to perform superior to deterministic methods.It is low on memory. It does not need to have huge RAM requirements like that of co-occurrence matrix where it needs to store three huge matrices.

CBOW takes the average of the context of a word . For example, Apple can be both a fruit and a company but CBOW takes an average of both the contexts and places it in between a cluster for fruits and companies. Training a CBOW from scratch can take forever if not properly optimized.

3.5.2 Skip–Gram Model

Skip – gram follows the same topology as of CBOW. It just flips CBOW's architecture on its head. The aim of skip-gram is to predict the context given a word.

The input vector for skip-gram is going to be similar to a one-context CBOW model. Also, the calculations up to hidden layer activations are going to be the same. The difference will be in the target variable. If context window of one is defined on both the sides, there will be "two" one hot encoded target variables and "two" corresponding outputs.

Two separate errors are calculated with respect to the two target variables and the two error vectors obtained are added element-wise to obtain a final error vector which is propagated back to update the weights. The weights between the input and the hidden layer are taken as the word vector representation after training. The loss function or the objective is of the same type as of the CBOW model.

The skip-gram architecture is shown in Figure 3.4

Input layer size – [1 X V], Input hidden weight matrix size – [V X N], Number of neurons in hidden layer – N, Hidden-Output weight matrix size – [N X V], Output layer size – C [1 X V]

Skip-gram model can capture two semantics for a single word. i.e it will have two vector representations of Apple. One for the company and other for the fruit. Skip-gram with negative sub-sampling outperforms every other method generally.

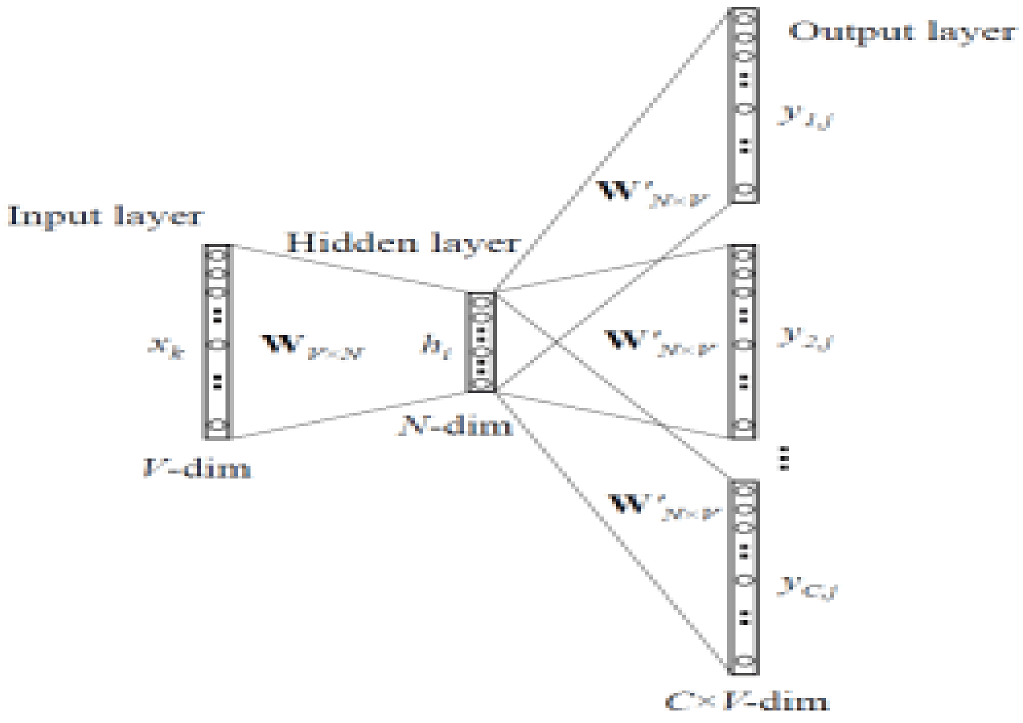


Figure 3.4: Skip-Gram architecture

3.6 GLOVE EMBEDDING

GloVe is another commonly used method of obtaining pre-trained embeddings. GloVe stands for Global Vectors for Word Representation In reality, GloVe is a much more principled approach to word embeddings that provides deep insights into word embeddings in general.

GloVe aims to achieve two goals:

- Create word vectors that capture meaning in vector space
- Takes advantage of global count statistics instead of only local information.

Unlike word2vec which learns by streaming sentences, GloVe learns based on a co-occurrence matrix and trains word vectors so their differences predict co-occurrence ratios. GloVe weights the loss based on word frequency

3.6.1 Why GloVe?

In word2vec, the loss function is computed by measuring how well a certain word can predict its surroundings words. Word2vec tries to either predict the word in focus from the context words (this is called the CBOW model) or the context words using the word in focus (this is called the Skip-gram model). The important thing to note is that word2vec only takes local contexts into account. It does not take advantage of global count statistics. So word2vec does not know if stopwords like "the" is a common word or they have some relationship with other words in the document.

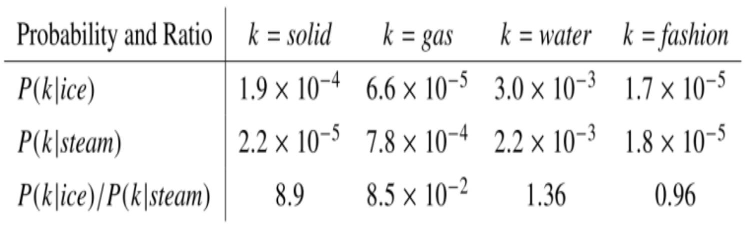
Using matrices that contain global count information is one of the major ways of converting words to vectors. Though these methods take advantage of global information, the obtained vectors do not show the same behavior as those obtained by word2vec. For instance, in word2vec, word analogies can be expressed in terms of simple (vector) arithmetic such as in the case of "king – man + woman = queen". This behavior is desirable because it shows that the vectors capture dimensions of meaning. Some dimensions of the vectors might capture whether the word is male or female, present tense or past tense, plural or singular, etc. This means that downstream models can easily extract the meaning from these vectors, which is what we really want to achieve when training word vectors.

GloVe aims to take the best of both worlds: take global information into account while learning dimensions of meaning. From this initial goal, GloVe builds up a principled method of training word vectors.

3.6.2 Building GloVe

In word2vec, the vectors were learned so that they could achieve the task of predicting surrounding words in a sentence. During training, word2vec streams the sentences and computes the loss for each batch of words. GloVe, is a more principled approach. The first step is to build a co-occurrence matrix. GloVe also takes local context into account by computing the co-occurrence matrix using a fixed window size

The underlying principle behind GloVe can be stated as follows: the co-occurrence ratios between two words in a context are strongly connected to meaning. An example table is given in Figure 3.5



Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 3.5: co-occurence ratio table

X refers to the co-occurrence matrix and X_{ij} refers to the i, j th element in X which is equal to the number of times word j appears in the context of word i. Also $X_i = \sum_l \mathrm{X}_{il}$ denotes the total number of words that have appeared in the context of word i . Other notations will be defined as we go along.

3.6.3 GloVe formulation of Loss Function

The aim is to predict the co-occurrence ratios using the word vectors. The relation between the ratios can be expressed with the following equation:

$$F(W_i, W_j, \overline{W_k}) = P_{ij}/P_{jk} \tag{3.1}$$

Here, P_{ij} refers to the probability of the word j appearing in the context of i , and can be computed as

P_{ij}= (number of times j appeared in context of i)/(number of words appeared in context of i)

which is nothing but X_{ij}/X_i

F is some unknown function that takes the embeddings for the words i, k, j as input. Notice that there are two kinds of embeddings: input and output (expressed as W and $\tilde{W}$) for the context and focus words. This is a relatively minor detail but is important to keep in mind.

Now, the question becomes what F should be ?. One of the goals of GloVe is to create vectors with meaningful dimensions that express meaning using simple arithmetic (addition and subtraction).F is chosen such that the vectors it leads to meet this property.

Now aim is form a simple arithmetic between the vectors to have meaning, so it's only natural that the input to the function F also should be the result of arithmetic between vectors. The simplest way to do this is by making the input to F the difference between the vectors we are comparing:

$$F(W_i - W_j, \overline{W_k}) = P_{ij}/P_{jk} \tag{3.2}$$

Now , a linear relation between w_i - w_j and $\overline{w_k}$ has to be created. This can be accomplished by using the dot product:

$$F(dot(W_i - W_j, \overline{W_k})) = P_{ij}/P_{jk} \tag{3.3}$$

Now, two tricks are used to determine F and simplify this equation:

- By taking the log of the probability ratios, the ratio is converted into a subtraction between probabilities.
- By adding a bias term for each word, the fact that some words just occur more often than others, can be captured.

These two tricks leads to the following equation:

$$dot(W_i - W_j, \overline{W}_k) + b_i - b_j = \log(P_{ik}) - \log(P_{jk}) \tag{3.4}$$

This equation can be converted into an equation over a single entry in the co-occurrence matrix.

$$dot(W_i, \overline{W_k}) + b_i = \log(P_{ik}) = \log(X_{ik}) - \log(X_i) \tag{3.5}$$

By absorbing the final term on the right-hand side into the bias term, and adding an output bias for symmetry, we get

$$dot(W_i, \overline{W_k}) + b_i + \overline{b_k} = \log(X_{ik}) \tag{3.6}$$

This is the core equation behind GloVe.

There is one problem with the equation above: it weights all co-occurrences equally. Unfortunately, not all co-occurrences have the same quality of information. Co-occurrences that are infrequent will tend to be noisy and unreliable, so the frequent co-occurrences has to be weighted more heavily. On the other hand, co-occurrences like "it is" dominates the loss, so these should not be weighted heavily based on frequency.

Through experimentation, the authors of the paper found the following weighting function to perform relatively well:

$$weight(x) = min(1, (x/x_{max})^{3/4}) \tag{3.7}$$

The function is depicted in Figure 3.6 :

Essentially, the function gradually increases with x but does not become any larger than 1. Using this function, the loss becomes

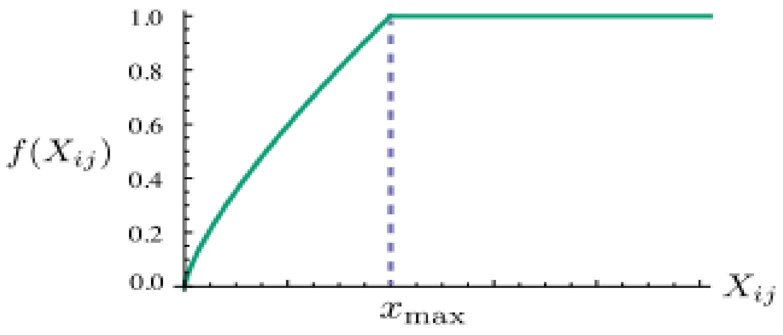


Figure 3.6: weight fuction

$$\sum_{ij} weight(X_{ij})(dot(W_i, \overline{W_j}) + b_i + \overline{b_j} - \log(X_{ij}))^2 \tag{3.8}$$

The objective is to train the weight matrix which minimizes the above cost function.

CHAPTER 4
DEEP LEARNING

The concepts used in the project are based on Deep Learning. The concepts of neural networks, a basic overview of RNNs and transfer learning are explained in this chapter.

4.1 NEURAL NETWORK

Neural Networks form the backbone of deep learning. The goal of a neural network is to find an approximation of an unknown function. It is formed by interconnected neurons. These neurons have weights, and bias which is updated during the network training depending upon the error. The activation function puts a nonlinear transformation to the linear combination which then generates the output. The combinations of the activated neurons give the output.

A neural network is best defined by “Liping Yang” as - “Neural networks are made up of numerous interconnected conceptualized artificial neurons, which pass data between themselves, and which have associated weights which are tuned based upon the network’s “experience.” Neurons have activation thresholds which, if met by a combination of their associated weights and data passed to them, are fired; combinations of fired neurons result in “learning”.

4.1.1 Input / Output / Hidden Layer

The input layer is the one which receives the input and is essentially the first layer of the network. The output layer is the one which generates the output or is the final layer of the network. The hidden layers are the ones which perform specific tasks on the incoming data and pass on the output generated by them to the next layer. The input and output layers are the ones visible to us, while the intermediate layers are hidden. Figure 4.1 depicts a neural network with one hidden layer.

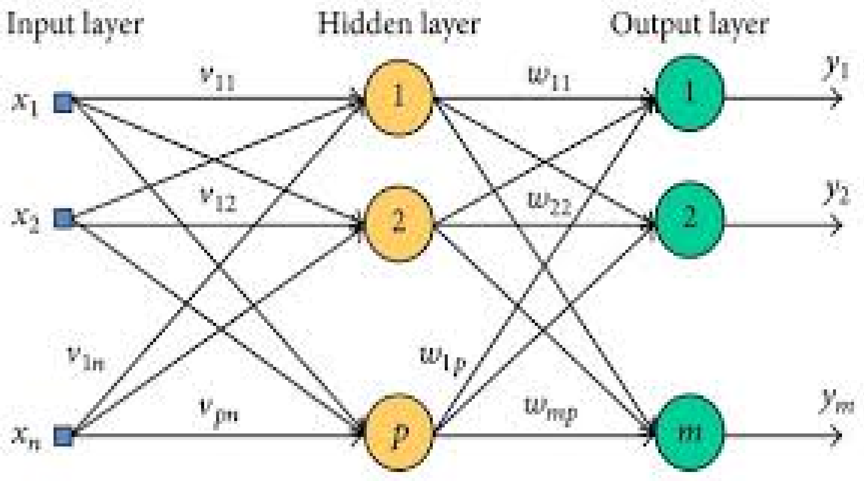


Figure 4.1: Neural Network with one hidden layer

4.1.2 Forward Propagation

Forward Propagation refers to the movement of the input through the hidden layers to the output layers. In forward propagation, the information travels in a single direction FORWARD. The input layer supplies the input to the hidden layers and then the output is generated. There is no backward movement.

4.1.3 Cost Function

When building a network, the network tries to predict the output as close as possible to the actual value. Measuring this accuracy of the network using the cost/loss function. The cost or loss function tries to penalize the network when it makes errors. The objective while running the network is to increase the prediction accuracy and to reduce the error, hence minimizing the cost function. The most optimized output is the one with least value of the cost or loss function. The learning process revolves around minimizing the cost.

4.1.4 Gradient Descent

Gradient descent is an optimization algorithm for minimizing the cost. To think of it intuitively, while climbing down a hill, usual thing to do is take small steps and walk down instead of just jumping down at once. Therefore, considering the starting point to be x, moving down a little i.e. delta h, and updating the position to x-δh and doing the same till the bottom is reached. Consider bottom to be the minimum cost point.

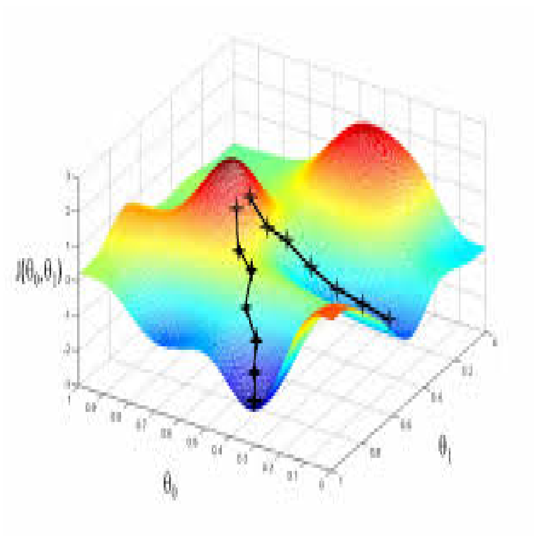


Figure 4.2: Gradient Descend

Figure 4.2 depicts Gradient Descent showing two paths that can be taken to reach local minimum. Mathematically, to find the local minimum of a function one takes steps proportional to the negative of the gradient of the function.

4.1.5 Backpropagation

When defining a neural network, assign random weights and bias values to nodes. On receiving the output for a single iteration, the error of the network is calculated. This error is then fed back to the network along with the gradient of the cost function to update the weights of the network. These weights are then updated so that the errors in the subsequent iterations is reduced. This updating of weights using the gradient of the cost function is known as back-propagation.In back-propagation the movement of the network is backwards, the error along with the gradient flows back from the out layer through the hidden layers and the weights are updated.

4.2 RNN

The idea behind RNNs is to make use of sequential information. In a traditional neural network the assumption is that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If the next word in a sentence have to be predicted, the words which came before it had to be known. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few step.

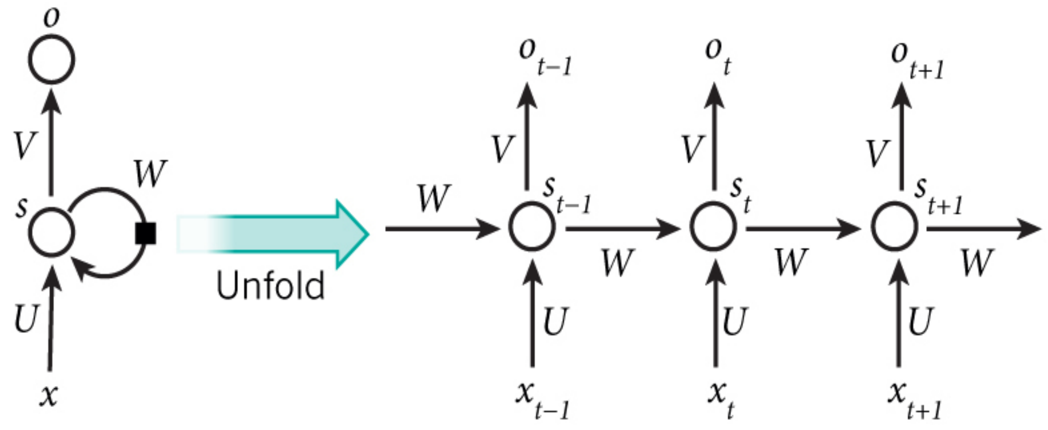


Figure 4.3: RNN unfolding in time of the its forward computation

Figure 4.3 shows a RNN being unrolled (or unfolded) into a full network. Unrolling means that the network for the complete sequence is written.The formulas that govern the computation happening in a RNN are as follows:

- x_t is the input at time step t. x_1 could be a one-hot vector corresponding to a word in a sentence.
- S_t is the hidden state at time step t. It is the "memory" of the network. S_t is calculated based on the previous hidden state and the input at the current step:

  $$S_t = f(U_{xt} + W_{St-1}) \tag{4.1}$$

  . The function f usually is a non-linearity such as tanh or ReLU. S_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
- o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary.

  $$o_t = \text{softmax}(V_{St}). \tag{4.2}$$

  .

There are a few things to note here:

- You can think of the hidden state S_t as the memory of the network. S_t captures information about what happened in all the previous time steps. The output at step o_t is calculated solely based on the memory at time t. As briefly mentioned above, it's a bit more complicated in practice because S_t typically can't capture information from too many time steps ago.
- Unlike a traditional deep neural network, which uses different parameters at each layer, a RNN shares the same parameters (U, V, W above) across all steps. This reflects the fact that we are performing the same task at each step,

just with different inputs. This greatly reduces the total number of parameters we need to learn.

- Figure 4.3 has outputs at each time step, but depending on the task this may not be necessary. For example, when predicting the sentiment of a sentence we may only care about the final output, not the sentiment after each word. Similarly, we may not need inputs at each time step. The main feature of an RNN is its hidden state, which captures some information about a sequence.

4.3 BIDIRECTIONAL RECURRENT NEURAL NETWORK

Bidirectional Recurrent Neural Network is basically put two independent Recurrent Neural Network together. The input is fed in normal order for one network and in reverse order for the other network. The output of the two networks are concatenated at each time steps. This structure allows the network to have both the forward (future) and backward (past) information at each time steps. The figure 4.4 shows the pictorial representation of Bidirectional Recurrent Neural Network.

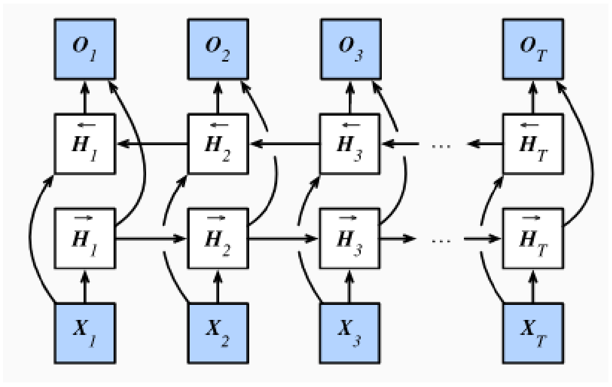


Figure 4.4: Bidirectional Recurrent Neural Network

4.3.1 Loss function

Loss fuction is defined based on the model and task. By finding the minimum point for the loss function the model learns optimal parameters for the given task. (eg., cross entropy loss). Loss function should be differentiable.

Cross-entropy loss (4.4) is defined below where p is model predicted probability for a class to occur and y ϵ {-1,+1} specifiying ground truth classes.

$$CE(p,y) = \begin{cases} -\log(p) & if\ y = 1 \\ -\log(1-p) & otherwise \end{cases}, \ p_t = \begin{cases} p & if\ y = 1 \\ 1-p & otherwise \end{cases} \tag{4.3}$$

$$CE(p,y) = CE(p_t) = -\log(p_t) \tag{4.4}$$

4.4 TRANSFER LEARNING

4.4.1 OVERVIEW

The general idea of transfer learning is to use knowledge learned from tasks for which a lot of labelled data is available in settings where only little labelled data is available. Creating labelled data is expensive, so optimally leveraging existing datasets is key.

In a traditional machine learning model, the primary goal is to generalize to unseen data based on patterns learned from the training data. Transfer learning is an attempt to kickstart this generalization process by starting from patterns that have been learned for a different task. Essentially, instead of starting the learning process from a (often randomly initialized) blank sheet, it starts from patterns that have been learned to solve a different task.

Transfer of knowledge and patterns is possible in a wide variety of domains. The goal is to incentivize data scientists to experiment with transfer learning in their machine learning projects and to make them aware of the advantages and disadvantages.

Transfer learning is key to ensure the breakthrough of deep learning techniques in a large number of small-data settings. Deep learning is pretty much everywhere in research, but a lot of real-life scenarios typically do not have millions of labelled data points to train a model. Deep learning techniques require massive amounts of data in order to tune the millions of parameters in a neural network. Especially in the case of supervised learning, this means that there is a need for a lot of (highly expensive) labelled data. Labelling images sounds trivial, but for example in Natural Language Processing (NLP), expert knowledge is required to create a large labelled dataset. The Penn treebank for example, a Part-of-Speech tagging corpus, was 7 years in the making and required close cooperation of trained linguists. Transfer learning is one way of reducing the required size of datasets in order for neural networks to be a viable option. Other viable options are moving towards more probabilistically inspired models, which typically are better suited to deal with limited data sets.

Transfer learning has significant advantages as well as drawbacks. Understanding these drawbacks is vital for successful machine learning applications. Transfer of knowledge is only possible when it is 'appropriate'. Exactly defining what appropriate means in this context is not easy, and experimentation is typically required. One should not trust a toddler that drives around in a toy car to be able to ride a Ferrari. The same principle holds for transfer learning: although hard to quantify, there is an upper limit to transfer learning. It is not a solution that fits all problem cases.

Typically, the word vector variants differ in the corpus they originate from, such as Wikipedia, news articles, etc., and the differences in the embedding models. It is important to understand the background of these models and corpuses in order to know whether transfer learning with word embeddings is sensible. People typically wouldn't call the use of word embeddings transfer learning, given the similarity with transfer learning with computer vision. Essentially, using word embeddings means using a featuriser or the embedding network to convert words to vectors.

CHAPTER 5

RESEARCH PLAN

This chapter talks about how the pipeline is created for the shortlisting process. Pipeline is mainly seperated into 4 different parts as shown in Figure 5.1.

- Formatting
- Data Preprocessing
- Information Extraction
- Shortlisting

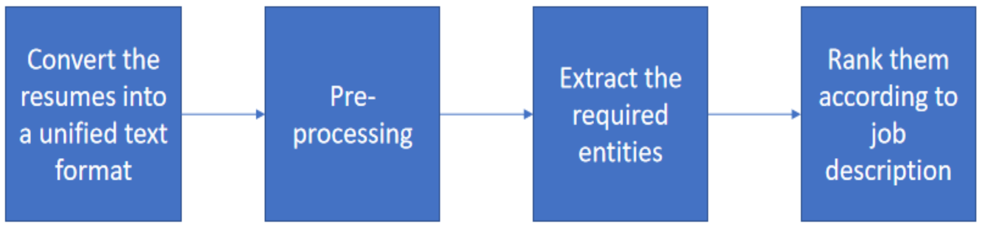

Convert the resumes into a unified text format
Pre-processing
Extract the required entities
Rank them according to job description

Figure 5.1: Resume Ranker Pipeline

5.1 DATA SOURCE

Resumes of the students who applied for the past two batches of Business Analytics course at IIM ,Bengaluru is taken for training. The validated batches are placed into train, validation and test. Seventy percent of the data are placed in the training folder and fifteen percent of the data are placed under the validation and testing folder. The data was annotated to train the NER model.

5.2 FORMATTING

Most of the resmes were in pdf format. So the initial task was to convert the pdf files to text files in order to get the raw data from the resumes. PdfMiner.Six library in python platform was used for this extraction of raw data.

5.3 PRE PROCESSING

Since the format was converted there were some encoding issues involved. Apart from the encoding issue the regular pre-processing steps as mentioned in chapter 2 is done to clean and make sense of the data.

5.4 INFORMATION EXTRACTION

This is the important step in the whole pipeline. Extracted fields are:

- Companies worked at
- Skills
- Graduation Year
- College Name
- Degree
- Designation
- Email Address
- Location
- Name

Words from a resume are vectorized using GloVe embedding. These vectors goes as input to the NER model.

5.4.1 NER Architecture

Bidirectional RNN with 4 layers was used in the NER architecture which is shown in Figure 5.2

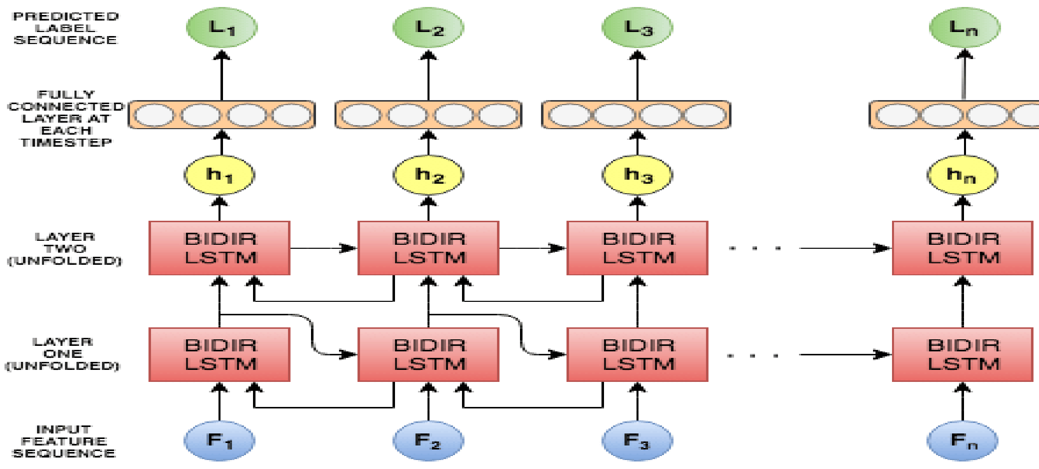


Figure 5.2: NER architecture

5.5 SHORTLISTING

Once information is extracted for each resume, it is shorlisted using WMD and top 500 candidates are shortlisted for the next round. WMD was derived from the EMD.

5.5.1 Earth Mover Distance

Before introducing WMD, I have to share the idea of Earth Mover Distance (EMD) first because the core part of WMD is EMD. EM solves transportation problem. For instance, let m and n denote a set of suppliers and warehouses. The target is going to minimize transportation cost such that shipping all goods from m to n. Given that there are constraints:

$$f_{i,j} >= 0, 1 <= i <= m, 1 <= j <= n \tag{5.1}$$

$$\sum_{j=1}^{n} f_{i,j} <= w_{pi}, 1 <= i <= m \tag{5.2}$$

$$\sum_{i=1}^{m} f_{i,j} <= w_{qj}, 1 <= j <= n \tag{5.3}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} f_{i,j} = min(\sum_{i=1}^{m} w_{pi}, \sum_{j=1}^{n} w_{qj}) \tag{5.4}$$

- Only allowing transport from m to n. Not allowing transport from n to m
- Total number of sending cargoes cannot exceed total capacity of m
- Total number of receiving cargoes cannot exceed total capacity of n
- Maximum number of transportation is the minimum between total cargoes in m and total cargoes in n

The denotations are:

- p: Set of origin
- q: Set of destination
- f(i,j): flow from i to j
- m: Number of origin
- n: Number of destination
- w(i, j): Number of cargo transport from i to j

To optimal flow F, the linear formula is

$$EMP(P,Q) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{i,j}d_{i,j}}{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{i,j}} \tag{5.5}$$

5.5.2 Word Mover's Distance

WMD is designed to overcome synonym problem.

The typical example is

Sentence 1: Obama speaks to the media in Illinois

Sentence 2: The president greets the press in Chicago

Except the stop words, there is no common words among two sentences but both of them are taking about same topic as see in Figure 5.3.

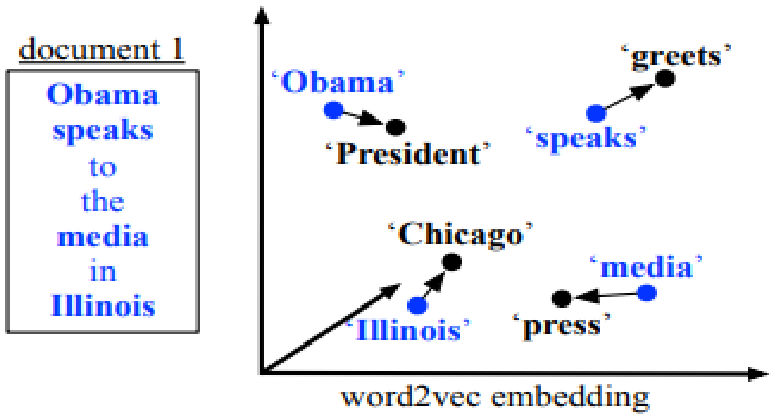


Figure 5.3: WMD

WMD use word embeddings to calculate the distance so that it can calculate even though there is no common word. The assumption is that similar words should have similar vectors.

Retrieve vectors from any pre-trained word embeddings models. It can be GloVe, word2vec, fasttext or custom vectors. After that it uses normalized bag-of-words (nBOW) to represent the weight or importance. It assumes that higher frequency implies that it is more important.

$$\begin{aligned} \underset{T>=0}{\text{minimize}} \quad & \sum_{i,j=1}^{n} T_{ij} C(i,j) \\ \text{subject to:} \quad & \sum_{j=1}^{n} T_{ij} = d_i, \ i = 1, \dots, n. \\ & \sum_{i=1}^{n} T_{ij} = \overline{d_i}, \ i = 1, \dots, n. \end{aligned}$$

It allows transfer every word from sentence 1 to sentence 2 because algorithm does not know “obama” should transfer to “president”. At the end it will choose the minimum transportation cost to transport every word from sentence 1 to sentence 2.

The advantage of WMD are hyper-parameter free and overcoming synonym problem. Same as those simple approaches, WMD does not consider ordering. The time complexity is an issue. The original version is O(p^3 log p) while the enhanced version is still O(p^2).

CHAPTER 6

CONCLUSION AND FUTURE WORKS

In this work a pipeline is created to shortlist the candidates in an interview process. The objective of the project is to bring in some automation to reduce the human effort and save time which is achieved.

Future work may include employing more powerful hierarchical ranking algorithms. Including other data sources like github and linkedin for verifying their profile might increase the candidates reliability. Capture complete recruitment funnel right from Resume Review process to Final shortlisting and Hiring can be done. This model can be further refined by capturing more structured data of candidates like Gender, Total experience, Number of organizations worked at etc.

Bibliography

[1] Ciravegna F and Lavelli A , ”Learning Pinocchio: adaptive information extraction for real world applications”,Nat. Lang. Eng. 10(2) 145–165, 2004.

[2] Yu K, Guan G and Zhou M, ”Resume information extraction with cascaded hybrid model”, ACL 2005: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, Morristown, NJ, USA, pp. 499–506. Association for Computational Linguistics, 2005.

[3] Breunig M.M, Kriegel H.P, Ng R.T, Sander J, ”Lof: identifying density-based local outliers”, SIGMOD Rec. 29(2), 93–104, 2000

[4] Knorr E.M., Ng R.T, ”Algorithms for mining distance-based outliers in large datasets”, VLDB 1998: Proceedings of the 24rd International Conference on Very Large Data Bases, pp. 392–403. Morgan Kaufmann Publishers Inc., San Francisco, 1998

[5] Ramaswamy S, Rastogi R, Shim K, ”Efficient algorithms for mining outliers from large data sets”, SIGMOD Rec. 29(2), 427–438, 2000.

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean, ”Distributed representations of words and phrases and their compositionality”, NIPS, pages 3111–3119, 2013

[7] Jeffrey Pennington, Richard Socher, Christopher D.Manning,”GloVe: Global Vectors for Word Representation”, Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, 2014