UNIVERSITAT POLITÈCNICA DE CATALUNYA(UPC) - BARCELONATECH

BARCELONA SCHOOL OF INFORMATICS(FIB)

DEGREE THESIS

Topical Web-page Classification with Similarity Neural Networks

Author:
GUILLEM GILI I BUENO

Project Director:
DR. LUIS ANTONIO BELANCHE MUÑOZ
from the deparment:
Department of Computer Science

A thesis submitted in fulfillment of the requirements for the degree of Bachelor's degree in Informatics Engineering, major in Computing

Delivered April 16, 2019, Defended on April 23rd,2019

Declaration of Authorship

I, Guillem Gili i Bueno, declare that this thesis titled, "Topical Web-page Classification with Similarity Neural Networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a degree at UPC.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

Date:

UNIVERSITAT POLITÈCNICA DE CATALUNYA(UPC) - BARCELONATECH

Abstract

Barcelona School of Informatics(FIB)
Department of Computer Science

Bachelor's degree in Informatics Engineering, major in Computing

Topical Web-page Classification with Similarity Neural Networks

by GUILLEM GILI I BUENO

The purpose of this project is to solve the problem of website topic categorization. To be able to discernish the topic of the contents of a website offers, given a discrete amount of categories. To achieve this goal, we will be using Neural Networks and Word Embeddings together with a Crawler. Word Embeddings have become a very popular method to represent the meaning a word has by analyzing its context. There is a variety of techniques and principles that define each Word Embedding, but the common denominator is the fact that they allow to represent words in a low-dimensional space vector where words that are more distant tend to appear less together. This allows these group of technique to effectively encode meaning and to be capable of getting an approximation of the substraction of meaning between them. This could work very well with Neural Networks, where what we receive is an n-sized input of values, if we find some way to codify whole websites into fixed size vectors. Throughout this project we will cover all the necessary steps to be able to obtain a website, extract its defining topic features and obtain its topic category using the forementioned tools.

Acknowledgements

To my parents, for supporting me throughout my studies and believing in what I can achieve. I have not been an easy person to teach and they have tried their best during my life to teach me how to live it.

I cannot thank enough the friends that I have made throughout my studies in FIB, those who stayed and kept studying with me and those who had to move for one or another reason, Jordi Mayta and Juan David Ramirez. The support and help from them has been invaluable, and I will always keep fond memories of speaking with them in class and meeting casually.

I must give a special mention to those that stayed with me in Computer Science, and adamantly persisted in getting the specialy. I will never forget staying for long in some of the lab classes while trying to fix our code, or trying to understand some abstract concept. I will even remember cheerly preparing some project delivery at the last day, working the whole day nonstop in it and somehow manashing to rush a decent project. Thank you Enrique Reyes, Ferran Noguera, Yeray Bosoms.

I have to also thank Claudia Martinez for supporting me throughout the project, and assisting me and trying to help me in whatever she could.

I am also grateful to Luis Belanche, for his guidance and recommendations during the project, specially in moments where I felt completely blocked and uncertain. I also must thank him for the knowledge taught during his classes, and having a good sense of humor about my chaotic naming in some documents.

I should also thank JEDI, for allowing me to get some extracurricular information and practice that has helped me form myself. Preparing the courses was fun, and while I have learned a lot of theory from classes, JEDI prepared me for real life work.

Contents

List of Figures

List of Tables

List of Abbreviations

GloVe	Global Vectors for Word Representation
URL	Uniform Resource Locator
B2B	Business to Business
NLP	Natural Language Processing
DMOZ	Directory Mozzila
ODP	Open Directory Project
RDF	Resource Description Framework
CBOW	Continuous Bag of Words
NN	Neural Network
PPMI	Positive Point-wise Mutual Information
PCA	Principal Component Analysis

Chapter 1

Introduction

1.1 Crawling a website

Displaying content from a website has been actively changing ever since internet expanded worldwide. Browsers have steadily become more complex over the years, and so have websites [1]. Weight has steadily increased in the websites[2], due to the need to fit more appearance-focused content [3]. New needs had to be addressed, such as fitting the functionalities to smartphone screens or limit the data. Dinamically loaded websites have become a necessity to balance the workload of servers.

For all those reasons crawling a website has also become a more difficult task and costly tax. A simple *httprequest* to get the HTML document is nowadays not enough to get all the information in most pages. While HTML is widespread used, webpages may choose to use some script that loads the entire website as their *httpresponse*. Modern crawlers need some kind of assisting software to obtain the websites, such as a browser. Headless browsers [4] are the most common kind of browsers for website crawling, as they render the HTML document without visually loading it. Websites also can specify some rules for automated programs that read them, and a polite (not ignoring these rules) crawling is strongly advised for all the programmers doing it.

1.2 Problem of web topic categorization

The problem of topic categorization is the process of assigning a document to one of many predetermined categories. It is a problem typically approached by representing the specific document in some quantifiable way and then attempting to find out its topic using the representation. Both steps are equally important: when converting the document the information from the document could be misrepresented; when evaluating the document representations our classification may be biased to more common categories.

In particular, web topic categorization has been heavily studied throughout history[5] [6] [7]. This is due to the diversity of needs when it comes to the task and how widespread the use of web resources is. A quick categorization may be required when we have huge amounts of information and little time for each document [8], as in filtering the web access in a network. Alternatively we could be interested in the reliability of our classifier, as in when we want to see websites of a specific topic personally.

Artificial intelligence has shown profficience in text categorization in various quantifiable ways[9]. This has been noticed using simple methods such as Naive Bayes [10] and Linear Regressions [11] as well as with using more complex algorithms such as Neural Networks [12] and SVMs [10].

Specifically, we want to use deep neural networks combined with word embeddings. Word embeddings allow us to represent the words within a text in limited size vectors, such that the meaning extracted from the context of that word is preserved. In practice this means that the differences of the vectors between man and woman should be similar to the differences of the vectors between king and queen. This will hopefully improve the quality results of previous works in web classification by using more complex tools, as some studies [13] have done in other text classification problems.

Chapter 2

Context

2.1 Areas of Interest

In order to determine which areas are of interest to the project, we will have think about all the processes it will do. We will not go over them in depth in this section, as we will do so in the scope of the project. Nonetheless, going over the whole list should help us build a list of areas that it will cover. Besides these tasks, we will also require some extra work to get a dataset in order to train our neural network. This is due to the lack of existing datasets of such kind: websites with their category indicated.

Our program has to be able to access a page, load it, transform its contents and evaluate its category. To access and load a page we will need a **crawler** and a **headless browser** respectively. The **crawler** will perform the *httprequest* and handle everything needed with that request, and the **headless browser** will make sure the resources from within a page are properly loaded by the time we save the HTML page. Then, the page will have to be transformed so that we can feed it to a neural network. We will begin by **parsing HTML** and separating its text from the rest, then we will perform some pre-processsing tasks to make our text more meaningful. Afterwards we will use our **word embeddings** to turn the text into a numerical values that we can directly feed to a **neural network** . Finally we will generate the NN, fine tune it and evaluate the statistics we obtain.

To sum up, the areas of interest of our project are **Crawlers**, **Headless Browsers**, **HTML Parsing**, **Word Embeddings** and **Neural Networks** (deep learning specifically). We will not cover HTML parsing in our State-of-the-art section, as it is a very simple and homogenized task.

2.1.1 Tangential areas of interest

Operative Systems

This project challenged our knowledge of the OS to make it work correctly. First of all, all OS allow a limited amount of open files at the same time. This is in order to protect the user, but it will make our Broad Crawl stop working, so we had to change some parameters in specific configuration files. We also had to change the swappiness parameter of our OS, to prevent it as much as possible to use swap memory. Swap memory is used when our OS is running out of physical memory(RAM), the main issue is that it makes the execution notably slower, as it is basically disk memory.

Hardware, Memory Transfer and speeds

Another field we had to work on was a correct usage of my physical hardware resources and how to make my program faster. As any computer scientist knows, one of the fastest ways to execute a program is loading everything into memory (RAM has a transfer speed 10-20 times faster than disk usually), execute the changes then save the result. We were reluctant to do this since the beginning with Python, as we thought it may not be necessary. We also thought this could complicate saving the progress in most tasks. Our first version of all programs simply accessed the disk as they processed and worked within an acceptable time.

By the end of the project, when we had to execute some of my programs with all the websites, they required a really long time to execute (20 days or more). This meant we had to use the forementioned strategy. We attempted to do so with Python but it completely overwhelmed my memory, and began using the swap partition (and was once again too slow). This was a surprise, as we thought loading a 5.7GB file would not take so much memory from Python. This may be due to how Python handles its data structures. We had to rewrite the code in C++, as it has a much more transparent memory usage and would likely be able load the necessarily files into memory. This also gave us an intuition that we would have an even lower execution time, as C++ is closer to C (C being the most efficient language). This reduced the execution from an estimate of 20-30 days (Python without loading the data to memory) to 4 days (C++ loading all data to memory).

2.2 State-of-the-art

2.2.1 Crawlers

A variety of crawlers exist nowadays, most multipurpose languages have them. Javascript has Node's crawler[14], Python has Scrapy [15], Java has (among others) Apache Nutch [16]. Node is a well-renowned framework for Javascript and is one of the most popular technologies when it comes to high scalability. Apache Nutch could also work for us since it allows us to use external plugins and has the advantage of being in Java (where almost any library has been implemented). Scrapy has the advantage of being easily customizable, and having an extensive and accessible documentation.

In the end, we chose Scrapy for various reasons: easy customization, good documentation, familiarity with the language and homogeneity of the project. The parameters for the crawl can be easily set up[17] and have recommendations for when we are massively obtaining websites[18]. The developer has also worked more with Python than in Java or Javascript. Finally, Python is a programming language with a broad spectrum of capabilities. The main 2 languages for the most modern Deep Learning Library are Python and R, and Python also has tokenizers, HTML processors, and a relatively efficient input output. This means that we could ideally do our whole project in the same language, which should make our work easier.

Headless Browsers

The main drawback of Scrapy is that it cannot load dinamically sites and has no included solution. While this may seem a drawback it won't necessarily be. Since we will only attempt to render the websites that respond our *httprequest*, this may improve the efficacy of our crawl. We will need to use a headless browser for this process. We will use PhantomJs[19], as it is a relatively fast renderer.

2.2.2 Word Embeddings

A word embedding is a model that provides a geometrical encoding for a word. This allows us to transform a word into a vector of numerical values, which is necessary for most document representation algorithms. This exists to solve the problems of computing with natural language: what we would want ideally would be an incredibly big square matrix where each word has some measure of similitude with another word. Such matrix would be incredibly complex to compute and hard to work with, so we use a more limited number of dimensions hoping that they can represent each word's meaning. WE may have various approaches, such as using a predictive model(word2vec) or a count-based model(Glove). They also may have a variety of implementations. Regardless, the common denominator in WE is an attempt to minimize the value of a cost function that takes into account a word and the context it appears into. We should also mention that some experiments [20] show that using WE subject to different preprocessing may lower our accuracy.

To turn our documents into something we can feed to a NN we will test a variety of word embeddings. We will be using various in order to find out which behaves best when used with NN. One of them will be GloVe [21], a count-based parallellizable model from 2014 using a least squares objective. GloVe is an open-source project developed by the Stanford NLP group[22]. We will also use a predictive-based model, Word2Vec[23], from 2013 that uses a continuous bag of words model or continuous skip-gram. Finally we will use another model that combines both predictive and count-based models, LexVec [24] from 2016. Like predictive-based models it uses negative sampling and gradient descent to minimize a cost function, but it also takes into account negative cooccurrance.

Word2Vec

This model, proposed by Mikolov[23] in 2013, is also known as Skipgram or CBOW and appeared in an attempt to try and get a better representation of words based on their context. Conversely to most of the methods that were based in a statistical analysis of the corpus, this paper suggested a predictive model that encoded each word's meaning in a vector based on the context of that word. This model computes a cooccurrence matrix and applies a Singular Value Decomposition. The computational cost of calculating this model is $\theta(mn^2)$ which with large corpus can scale very badly. The other issue this model has it adding new documents: each time we want to incorporate a new document to our training the Singular Value Decomposition must be done from scratch .The cost function for word2vec is the following:

$$J(\theta) = \frac{-1}{T} \sum_{t=1}^{T} \sum_{-m<=j<=m, j\neq 0} logP(w_{t+j} \mid w_t)$$

Where P is the probability, and m defines the threshold of our window; how many words we would like to consider context. We must also mention that for each word we will have a context vector v and a center vector u with different values, to try and separate both concepts. To calculate the final vector of a word we will simply sum them as is recommended by Richard Socher[25].

This proved to be very effective with word analogy tasks and capture more complex patterns based on locality. However, this model did not scale well with huge amounts of corpus and the usage of its statistics (the formula above needs some to make it work within a reasonable time), and it did not take into account the overall statistics for a corpus. It also does not consider the fact that many non-informative words are likely to be common. These are likely reasons for its lower performance compared to the rest of techniques when tested.

Glove

Glove[26] was a WE designed in an attempt to combine the best of both the classic statistical methods (for example LSA) and the more modern predictive models (Skipgram and CBOW). Glove deals with the issue of scalability by having to collect the word coccurrance matrix only the first time the model is build and allowing for its updates to be reflected in the final model easily. It also allows to capture more complex linguistic patterns than LSA (semantics, syntax) due to its objective function, which is the following:

$$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - logP_{ij})^2$$

Where P refers to the cooccurance matrix and f is a function that aims to weight lower the most common words for our cost function, and thus take them less into consideration. The 2nd parenthesis is the part Glove really optimizes, as it measures the distance between the real values of the two vectors and its cooccurance. We must also mention that for each word we will have a context vector v and a center vector u with different values again. To combine them we get the sum of them.

What this function allows us to do is optimize cooccurrances one parameter at a time, so instead of $\theta(mn^2)$ our computational cost is $\theta(max(n^2, mn))$ for the optimization.

The results of Glove suggest that while it does not focus as much as Word2Vec on context and on knowing when a word can replace another word (they have the same syntactic function) it does allow to see capture relations and can also capture semantic meaning. Glove usually allows working out the meaning of words through arithmetic operations on other words. This means that it encodes a lot of word analogies and it is capable to answer question such as the following (using cosine similarity with the vector resulting from the arithmetic operation):

- *man is to woman, the same as ???? is to queen.* man-woman +queen = king
- *Cu is to copper, the same as ???? is to silver.*Cu - copper + silver = Ag
- *Japan is to sushi, the same as ???? is to bratwurst.* Japan - sushi + bratwurst= Germany

We can see a couple of other relationships that Glove is able to capture in the pictures obtained from the Glove official website[26]. These graphics were obtained doing a PCA on the selected words. As we can see, Glove is able to capture beyond simple context relations in some cases.

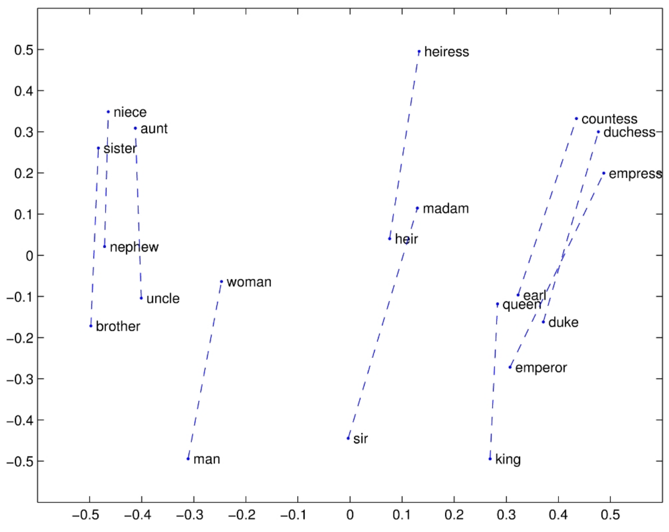


FIGURE 2.1: Encoding of man-woman relation in Glove. Extracted from the official website[26]

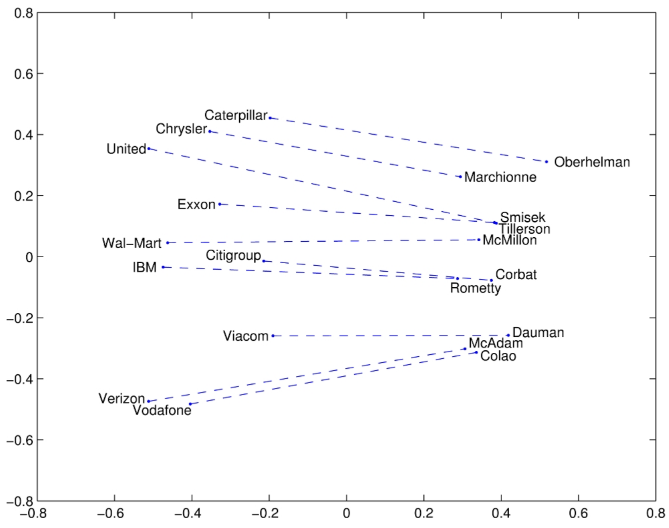


FIGURE 2.2: Encoding of company-CEO relation in Glove. Extracted from the official website[26]

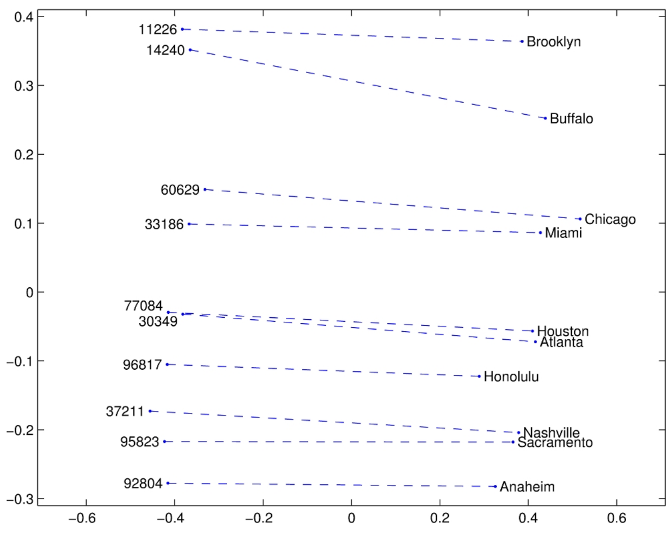


FIGURE 2.3: Encoding of city-zip relation in Glove. Extracted from the official website[26]

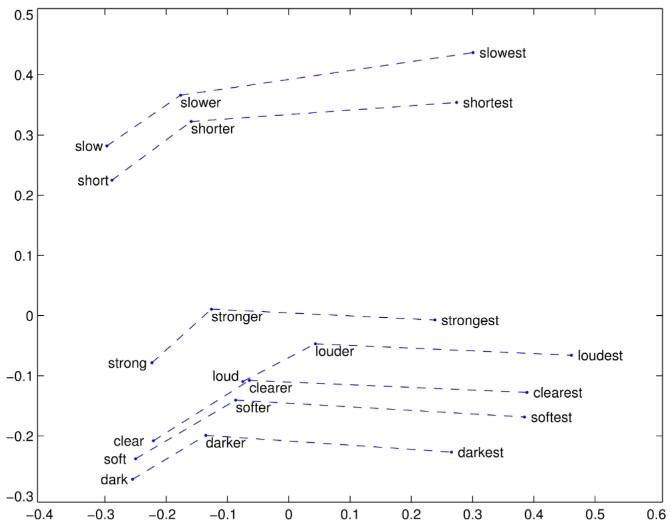


FIGURE 2.4: Encoding of comparative-superlative relation in Glove. Extracted from the official website[26]

LexVec

LexVec is a method strongly based on the idea of factorizing a PPMI matrix using a reconstruction loss function that penalizes frequent cooccurrence errors more heavily[27]. Unlike Glove, this model takes into account negative cooccurrance. The loss function for LexVec has two terms which are later combined:

$$J(\theta)_{wc} = \frac{1}{2}(W_w \tilde{W}_c^T - PPMI^*_{wc})^2$$

$$J(\theta)_w = \frac{1}{2}\sum_{i=1}^{k} E_{w_i \sim P_n}(W_w \tilde{W}_{w_i}^T - PPMI^*_{ww_i})^2$$

Where w represents a word and c represents a context. There are two ways of combining these two terms: multiply the sum of both for the times (w, c) is observed or multiplying $J(\theta)_{wc}$ for the amount of times (w, c) is observed and adding it to $J(\theta)_w$ times the amount of times w appears. Basically this method takes the approach of considering the cost function for context and center word separately, which is somewhat uncommon, and can be interesting to compare to the other methods.

Comparison through word analogies

To evaluate a WE a common test is to give some word analogy tasks, similarly to the words from the examples mentioned in Glove (man is to woman what king is to ????). Both Glove and LexVex seem to perform generally better than Word2Vec, or that is at least from what we see in the tables. Here we can see the performance each model has across various tests:

TABLE 2.1: Glove's comparison with various word analogy tasks.[21]

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	**72.7**	75.1	56.5	37.0
CBOW	6B	57.2	65.6	68.2	57.0	32.5
SG	6B	62.8	65.2	69.7	**58.1**	37.2
GloVe	6B	**65.8**	**72.7**	**77.8**	53.9	**38.1**
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW	100B	68.4	79.6	75.4	59.4	45.5

TABLE 2.2: LexVec's performance comparison with various word analogy tasks.[28]

Model	GSem	Gsyn	MSR	RW	SimLex	SCWS	WS-S	WS-R	MEN	MTurk
LV,W	76.4	71.3	70.6	.508	**.444**	**.667**	.762	.668	.802	.716
LV,WC	80.4	66.6	65.1	.496	.419	.644	**.775**	**.702**	**.813**	**.712**
W2V	73.3	**75.1**	**75.1**	**.515**	.436	.655	.741	.610	.699	.680
Glove	**81.8**	72.4	74.3	.384	.374	.540	.698	.571	.743	.645

Other information that we can infer from Glove's comparison is that the more data we have the better our model will perform. In Richard Socher's lectures[25] we can also see him comment that the quality of the data is important, Wikipedia being a prime example. Wikipedia's entries in a country will likely mention the country's capital, its currency, etcetera, where if we use a news' page entry about that country its likely to limit to some aspect. The fact that more data improves performance is generally true for AI techniques, so we can assume it as a fact with WE. Because of that we will use the Common Crawl pretrained WE whenever we can, and also because the corpus we want to categorize is precisely websites. Another interesting thing to know from his lectures is that more dimensions do not necessarily correlate to better performance at a word analogy task.

2.2.3 Deep Learning

Deep learning refers to the subset of machine learning that attempts to use a cascade of multiple layers of non-linear processing units in order to infer various levels of representation. Deep learning has been applied to multiple fields such as computer vision, speech recognition, natural language processing, social network filtering,etc where it has produced comparable results to human experts, even shown a better performance in some cases. Most of Deep Learning models are made using NN, although some other architectures are also considered Deep Learning[29]. The main idea behind this approach is that each level of representation will be able to transform the data into a more abstract and composite representation, and thus will allow us to identify complex patterns in the input data.

We can see some previous attempts to use NN to identify the topics in texts [12] [30] [31]. We can also see other alternative recent approaches for semantics analysis, such as association rules [32]. Others focus only in the social media content [33]. More recently, CNNs have been used while working with NLP [34] and shown notable results. CNNs have been the main choice for NLP tasks due to their effectiveness in Computer Vision[35].

Evaluation metrics

There are various ways to evaluate a classifier model's performance, we will only mention 2 of them, the Accuracy, which is the most common, and the F1 measure, which is another popular alternative. Both have a very basic form that is only usable in binary classification, and with some tweaking it is extendable to multiclass classification. The formula for binary accuracy is the following:

$$Accuracy = \frac{TruePositives+TrueNegatives}{TruePositives+TrueNegatives+FalsePositives+FalseNegatives}$$

This is the easiest to implement and simpler formula to calculate. However this formula has an issue: it performs poorly when a dataset is unbalanced. Suppose we have a problem where 95% of our data is a positive case. If we evaluate using accuracy there is a possibility our model decides it is better to classify all the cases as positive rather than evaluating them. A "always true" classifier guarantees a 0.95 accuracy, where another model might actually try to generalize the function that separates true and false may get a 0.90 accuracy. The other measure, F1, does not have this same issue with binary classification. The formula for binary F1 is the following:

$$Recall = \frac{TruePositives}{TruePositives+FalseNegatives}$$

$$Precision = \frac{TruePositives}{TruePositives+FalsePositives}$$

$$F1 = 2\frac{Precision*Recall}{Precision+Recall}$$

This formula does not have the same issues as accuracy, and instead focuses on balancing all the classes' performance. We should also mention that there are two ways of calculating the F1 score for non-binary classification: Macro averaging and Micro averaging. Macro averaging calculates the individual precision and recall for each class, then averages them before before returning the F1 score, Micro averaging calculates recall and precision for all instances and then returns the F1. While Macro averaging treats all classes equally (once again in a very unbalanced dataset will not work) micro focuses on the overall precision and recall statistics.

2.3 Available resources

In order to develop our project a series of libraries, software and data must be obtained. We will be using open-source resources as a way to support them.

2.3.1 Data availability

Dataset

For our dataset we will be using DMOZ[36], the ODP project. This is a discontinued project that has not been updated since 2013 (DMOZtools, the web hosting ODP, is a static mirror). It contains more than 1.9 million urls and is codified in in RDF format. Although being old this is the only dataset that exists for website topic clasification. Currently Curlie[37] exists in an attempt to recontinue ODP, but at the time this project began the updated dataset from Curlie was not available and thus this option was discarded.

There are other open datasets available for text classification[38]. But using them is not ideal for our problem, as they are not websites and they are quite old (Reuter's [39] is from 1987 and was last updated in 1996, 20Newsgroups[40] is from 1995 and has not been updated). Text from modern websites may not ressemble the documents in those datasets. We would also run into the risk of concept drifting.

An alternative possible dataset could be using the Common Crawl[41] files, but we would have to attempt to get the category for each website through some specific statistic obtained from the website. There is also the issue that Common Crawl Files are a couple TiB, which we have no way to store.

Word Embeddings

For our word embeddings we will be using pre-trained examples found on official sources. We must mention that most of them are trained on the Common Crawl dumps[42], the biggest open dataset of crawled webs (since 2011, continued up till now). As the comparative tables from Glove suggest, we will be using the word embeddings that utilize the biggest amount of data, and using a 300-dimensional vector. The Glove word embedding can be obtained from the official website[26] and is trained with a Common Crawl dump. The LexVec word embedding used for this project can be obtained from the official github repository [28], and is also trained with a Common Crawl dump. Finally the Word2Vec word embedding is available in Google's official Word2Vec page [43] and has been trained on the Newsgroups dataset.

We will use the word embeddings for each dataset that use a 300-vector space. This will simplify comparing them as each NN will receive the same amount of data from each document. Glove's WE weights 5.6Gb, LexVec's weights 5.7GB and Word2Vec's model weights 3.6GB.

2.3.2 Software

All of the resources used are use open-source code and are available online. We will use them to support open software. The only exception is Github, which is free for us since we qualify as students. We will be using the following software and libraries:

- Github[44] for managing the software version and stability in development.
- PhantomJs[19] used as a headless browser.

- Selenium[45] used for web browsing automation(while scrapping).
- Scrapy[15] used for scrapping website.
- BeautifulSoup [46] used to parse HTML and extract its text.
- Gensim [47] used to load a word embeddings model (word2vec's WE comes in a binary format model).
- NLTK [48] used to process natural language.
- Keras[49] used to implement NNs with state of the art technology.

The rest of WE come in a simple text format, where each line has one word and 300 floating point numbers that represent that word. Thus no specific software library will be needed to use them.

2.4 Actors

2.4.1 Student

This project has me as its one and only developer. This means I will be responsible for accomplishing the deadlines and project management, as well as writing the report and documentation for the results obtained. I will also have to communicate and check with the project director when needed for technique or technology choices.

2.4.2 Director

The director for this project is Lluís Belanche[50], associate tenured professor at the Faculty of Computer Science(FIB) and part of the Soft Computing Research Group(SOCO). His role in this project will be project manager: detecting errors in the proposal or execution, guiding, giving advice and helping the developer.

2.4.3 UPC

UPC[51] will be an actor, the main benefitiary of this project. All intellectual and industrial rights of this project will belong to UPC as is established in the TFG normative approved by Consell de Govern in 10/10/2008.

Chapter 3

Scope

The aim of this project is to create a model that can correctly classfiy an url through using WE, and to evaluate the performance of various WE. This project aims only to cover the webpages in the English language. We want to do this with the maximum accuracy possible, and in a relatively reasonable time. The project is divided into two big processes: Data Extraction and Model Training. Data Extraction includes everything between acquiring the URLs dataset DMOZ and getting a raw text document. On the other hand, Model Training is meant to focus on the parts of the process that involved how we would preprocess the simple text documents, feed them to a model and make the model effective enough for our standards.

There is also the final segment of this project: building a prototype. The model that we will obtain from Model Training will just be a file that when loaded is capable of classifying a Word Embedding representation to a category. Thus, we will have to create a simplified version of all the project that is capable, given a URL, to scrap it, obtain its text, preprocess it, generate its word embedding representation and finally feed it to the model and obtain the category. That is what we will call our prototype.

3.1 Data Extraction

The first part of the project is to both obtain data as a reference to train our project and to create a web extraction process for when we have to analyze new websites. These two processes will be considered together since the ODP dataset mentioned earlier did not have the html documents, only website links, thus both tasks need to cover the same necessities and will use the same programs. We must also mention the fact that both the crawler and the Feature Extractor check whether a webpage exists in the dataset they output to and ignore that webpage/document if it exists, that way we are not repeating our work.

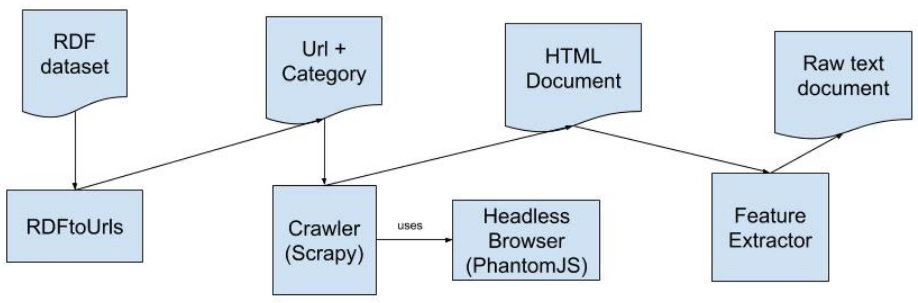


FIGURE 3.1: Data Extraction Proces

This part will also include a program to preprocess the DMOZ original dataset[36]. This dataset's format is RDF, so we will need a program that parses RDF, locates the URLs together with the categories and then saves the pairs in a file. Along with this program we will add another program that groups the URLS by 5,000, in order to make our scrapping work easier. This is so that we can split the workload between different computers, and also so that in case of a crash, power outage, or internet cut we do not lose all our progress. This will also allow us to have an estimation on the amount of websites we go through in a day. The splitting program will not be included in the project planning, as it is a rather easy and simple implementation that will cost us almost not time.

Another part of this process is a Scrapy project (the Crawler), that will query those urls and attempt to render the page and save the HTML. This program will follow a standard Scrapy structure, and it will receive as input the files with URLs and store the HTML document. Furthermore, the Scrapy project itself will be using the headless browser PhantomJS along with the Selenium library: it will do an http request to an url, then if it receives a response it will load PhantomJS and attempt to render the websites. While this may seem inefficient our dataset has a huge amount of unresponsive urls, by only attempting to render the pages that do respond we will dedicate more resources to the websites that guarantee us results. Besides, we will set each spider to have a time limit of 2 hours and allowing 5,000 concurrent requests, since we do not want our scrapping to get stuck in crawling a single group of URLs.

Taking into account Scrapy's recommendations for broad crawls [18] altogether with the results of experimenting on our own and our priorities we have set a series of parameters that are relevant for our crawl:

- The amount of **concurrent requests** has been set to 5,000. Both of our computers were able to run at this parameter without any CPU issues(which is mentioned[18] to be the bottleneck), and also this allows us to work with the all of the urls in a url group concurrently. The reason this worked for this particular datset is probably the fact that many of the urls did not work anymore and gave an inmediate 404 response. This meant that all these requests had an inmediate response and easy resolution.

- **REACTOR_THREADPOOL_MAXSIZE** was increased to 20, to prevent DNS resolver timeouts. This parameter has to do with how Scrapy handles DNS requests.
- We have enabled cookies. Initially we did not have them enabled, and this resulted in part the text extracted containing the cookies disclaimer message which is noise for our Model. The main reason this was not enabled at the begining was to try and lower the amount of personal information that websites try to create from our connections, thus lowering the amount of work our Crawler
- We enabled the Ajax Crawl, for the few websites that are loaded using some kind of Ajax script.
- We enabled Autothrottle, this is an algorithm that estimates and updates the download delay dinamically, according to the latency on our machine.
- We set the spider timeout to 7,200 seconds, which is two hours. We do not want our crawl to get stuck in any group of 5,000 urls, if it takes too long to crawl proceed to the next group.

Finally, the project will have another program that will parse HTML documents and extract the text content within them, along with some meta information. Meta information is information we can find in the head section (this section is not visible when a document is rendered and it would not appear while extracting the text) of an HTML document, and it gives some extra information on the webpage. To simplify our problem we will only get the text metas that may have some information on the content of our webpage[52]. We will attempt to obtain the text from the following OG metas: *title, type, description,locale,locale:alternate,site_name* and *image:alt*. We will save the words obtained from these metas together with the text outside the tags in the HTML, thus giving us a simple text file that we can preprocess.

3.2 Model Training

Model training deals with all the preprocessing and transformations needed to feed our basic text documents into a neural network, training neural networks and getting data from their performance. While the preprocessing will be the same for all the webpages, we will begin to diverge once we get to WE generator. We will have two programs that work with the 3 WE, one of them will be used for two WE. For each WE we will generate a different version of the dataset. Finally, for each version of the dataset we will try to generate various NN arquitectures, to see which behaves best.

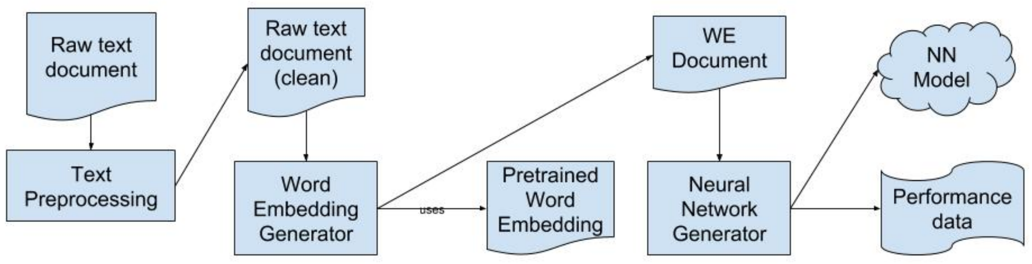


FIGURE 3.2: Model Training Proces

First of all a program will preprocess the text-only documents obtained from the Data Extraction. The text preprocessing reads a raw text document and removes all the "weird" words (words with strange characters; ">","<" and such). It will tokenize them identifying each word as a permutation of letters and numbers, and any other character as a separator. We will not include words with a single character or above 15 characters since they are likely to be typos or cases our tokenization does not solve well. This program will be using the NLTK library[48] from Python.

Next we will have a program that, for Glove and LexVec (and a separate program for Word2Vec) will receive the preprocessed text and transform it to a single word vector. Each will load the pretrained WE and preprocessed documents into memory, then it will turn each document's words to vectors and sum these vectors (without mixing separate dimentions). The intuition here is that if substraction can be applied when working with words (queen-woman= something similar to king) then representing a document as the addition of its words should also work. The result of this program will be that each document will be represented by a 300-dimensional vector.

The implementation of the Word2Vec in this case is separate, since the pretrained word embedding is only available as a binary we will have to load it using Gensim[47], in Python. The other two WE generators (Glove and Lexvec) will be done in C++ as they are very heavy and cannot be loaded into memory otherwise (Python cannot load them even with 16GB of physical RAM). It is a necessity to load them into memory, since otherwise it could take up more than 20 days to process all of them, loading them on memory using C++ allows us to process them in approximately 4 days(each). Although the programming language may be different this program will also obtain each word in a document, transform it to its vectors and save the sum of its vectors (once again without mixing separate dimensions). The result of this program will also be that each document will be represented by a 300-dimensional vector.

To try and facilitate the work of our algorithm, we have also implemented 3 different programs that transform the values of the 300 vectors that represent each document. Each program will generate a new dataset for each of our already existing datasets, thus resulting in us having to train 12 NN at the beginning of the training step. Once these networks have been trained and we have a general idea of how these modifications work, we will work on tunning the datasets that gave us the best result, and let this transformation be a part of the final prototype. The first program does a simple normalization where it picks the highest positive value and lowest negative value in the whole dataset, then divides each value in the dataset for the number with the same sign. The second program picks the highest and lowest values for each dimension of the whole dataset, then divides the values in each dimension for the highest or lowest value in the dataset's dimension, once again according to their sign. Finally, the last program turns each document's representation into a unit vector.

To conclude, this part will include a program (NNGenerator) that is able to generate each of our NNs and train them onto a specified dataset. This program will read all the documens (represented by a single vector) in a folder and generate a NN according to the parameters we specify for its arquitecture and training (depth, amount of units, epochs). NN generator should also save some performance measures while training and validating, and save the model.

3.3 Generate our prototype

The final part to this project is a simplified combination of the other two, where a script will read some urls as input and return the category it predicts for each. This will use the scrapping and text extraction mentioned in the first part, then proceed to preprocess the text, turn it to the most accurate WE representation and feed it to the best model obtained for that WE so that we get the topic category the website has.

This part should be very simple, since by the time we work on our prototype we will have gone through all the individual programs and be strongly familiarized with the libraries.

3.4 Other modifications

3.4.1 Operative Systems

As mentioned in subsection *Tangential areas of interest* some OS modifications are required for this project to run. The file limits can be checked with 'ulimit -Hn' and can be set the same way. We did these changes in an alternative way, but it will be covered in the *Obstacles and Solutions* more thoroughly.

Another parameter I had to change in my OS was the 'swappiness'. The swap partition is used when an operative system runs out of physical memory, this makes any program execution notably slower. The swappines value adjusts how much the operative system loads the files to the swap memory (disk) instead of using memory, I reduced it from the default 60 to 10.

Chapter 4

Possibles obstacles and solutions

4.1 Issues encountered during the project

4.1.1 Dataset is too big to handle

This happened during the project while generating the word embeddings documents. Initially we saved the documents as the vector of individual word vectors. This made a dataset of 40,000 websites weight around 40GB (and we had to generate 3), so the situation was scaling out of control. While 120GB may not seem that overwhelming storage-wise if we think of them in terms of processing and loading into memory they are enormous. In the end we saved each document as the addition of those vectors, which is what we would feed our neural network directly. This meant that we could not alter how we combined the word vectors (we were commited to use just addition) and that we had to work with a static dataset each time it was generated with WE. We had to sacrifice some versatility in exchange of physical space and speed.

4.1.2 Slow Broad Crawl

The task of performing a Broad Crawl (crawling DMOZ) took longer than we initially estimated. Initially (after beggining the Broad crawl) we had estimated that it would take a month. To accelerate this proces, we tweaked with the amount of concurrent connections that were allowed and used multiple computers and access points. The crawling has proven to be slower than our expectations due to the latency on the pages, even after ensuring we are getting the most out of our resources (two computers in two access points) and making sure to check for revisited websites (and not crawl them again).

During the final sprints in the project we considered it may not be possible to crawl the whole url dataset in time. To make sure the dataset we have before the deadline was representative of all the categories, we handpicked some of the URLs groups and changed them to be crawled with priority. This was done in order to ensure we get enough representation from each category to be able to generalize on them. In the end it was possible to crawl the whole dataset but this could have left some of the categories we had empty so it was an issue to be addressed.

4.1.3 Consequences of an slow Broad Crawl

Due to being behind on the amount of websites that we had collected our schedule got notably delayed. For this reason precisely we made sure that all the programs that took a decent amount of time were able to progress with the work even with an incomplete dataset. That means we changed their implementation in order to check for every file that they were going to generate, in order to not repeat work. What this effectively meant was a way to save our progress, and made it possible to have a lot of the work done in advance (specially in the case of generating the WE documents).

4.1.4 Unintended cookies notification

Initially when we were working in our crawler, we decided that it would be better to not accept the cookies in the websites. The reasons for this were two: to exchange less data from the websites (and make the scrapper's task less heavy) and to attempt to get the "base" page (the page the average user gets). The issue with this choice is that it has made the crawled websites include in their text the cookies notification, and sometimes load the page partially only. This is a difficult part to remove from the HTML and it will add some noise to the training of our Neural Network. Ideally we would have changed the scrapper and restarted the crawl from 0, but this is not viable due to our time constraints. The best we could do to solve this issue was changing the crawler to accept the cookies as soon as this was noticed.

4.1.5 Reimplementation of WE conversor

The program meant to do the conversion from preprocessed raw text to documents has been reimplemented a couple times. Initially it was a Python3 script that loaded into memory all the website documents, then iteratively accessed each word in the WE files while updating each document's representation and finally wrote its representation (with each word separated). This task would take several days, and the total dataset from only 50,000 websites at the time weighted 100GB. This meant that the representation could not save each word's vector separately and that a power outage could compromise the entire work done. Thus, the program was redesigned with a focus on being able to save progress and in an attempt to improve its performance. After some modifications, it saved the documents in bulks of 5,000 and it was outputting daily 5,000 processed websites, and now saving the documents as the sum of their word vectors.

Reading the big file from the disk was clearly still too slow. So the only viable option was to load everything it required into memory before the execution began. This was attempted to do with Python, but after attempting several attempts and implementations the script would always crash: it would run out of memory (on a 16GB machine). Since memory management and basic object structures are a bit opaque in Python, we deemed the best solution to reimplement the program in C++. While the program occupies 8GB on memory, it can run, and is also capable of going through the entire dataset (which by the end are almost 200,000 files) in much less time (4-5 days compared to the estimated 40). The program was also implemented so that no work is repeated regarding files.

4.1.6 OS configuration changes

As mentioned in the the chapter *Context* section *Areas of interest*, subsection *Tangential areas of interest* we had to change some specific operative system parameters in order to make our programs able to run correctly.

We changed */etc/security/limits.conf* and added the lines "* hard nofiles 10000000" and "* soft nofiles 10000000", to increase both the soft limits and hard limits. Soft limits control the amount of files open in a session, hard limits give a ceiling to the soft limits of a user (soft limits can be modified while that session is happening). Both limits can be checked using "ulimit -Hn" (hard limit) and "ulimit -Sn"(soft limit).

Besides, we had to modify */etc/sysctl.conf* with the line "fs.file-max = 10000000". This modification changes the amount of files that can be open system-wide. It can be checked using "cat /proc/sys/fs/file-max".

Finally, another parameter I had to change in my OS was the 'swappiness'. The swap partition is used when an operative system runs out of physical (RAM) memory, this makes any program execution notably slower. The swappines value adjusts how much the operative system loads the files to the swap memory (disk) instead of using memory, I reduced it from the default 60 to 10 (less swap usage), since we value highly our efficiency, and 16GB of memory should be enough for executing most of our programs completely in memory.

4.1.7 Lack of experimentation with WE

Within the plans I had for this project was the possibility of trying various approaches to how we used the WE to codify documents. We had to limit ourselves to sum, as transforming the whole dataset took 4-5 days of execution in our 16GB machine (where it did not have to use swap memory). Ideally we could have attempted to generate more variety of techniques to combine them. Specifically, we could have tried to make each document's representation as the averaged value of each component of its word vectors.

After informing myself more in WE, I have also found out that some researchers recommend the usage of pretrained models in Wikipedia. The reason for that is that Wikipedia has very precise and thorough explanations, that may help to capture a word's meaning better than the Common Crawl pretrained embeddings. While Common Crawl is the exact context we are working in, if a word appears in the news it will not necessarily capture all the information that word encodes. If Barcelona is mentioned, tourism and more modern topics are likely to appear in a website about it. Alternatively, the Wikipedia articles are likely to have more historic, cultural and territorial information.

This lack of experimentation also comes with the NN, in our approach we attempting to match each document vector (300 variables) to a categorical value among the 13 categories we have. This means we have 13 neurons and we expect that our neural network can, for each example make that neuron be the only one that activates for the category. What if instead of that we had used 300 variables as output as well, and we gave just the word embedding representation for each category's title?

Chapter 5

Project Planning

5.1 Planning and scheduling

The project duration is 5 months and half, starting at November 19th and finishing at April 12th. The duration of this project has extended due to issues while scrapping the websites.

The planning includes me getting knowledgeable on the topic, which is unusual for a project in a company environment. This is due to me being still a student of the topic, as we do not have that much experience in it . This is not an uncommon practice in scientific investigation.

The general structure of the project has been thoroughly explained in the *Scope* chapter, so we will not go over it again. We will not go into details either of what each individual implementation requires, as that has also been covered in the same section. We will focus solely on which facts are relevant to the time each task covers.

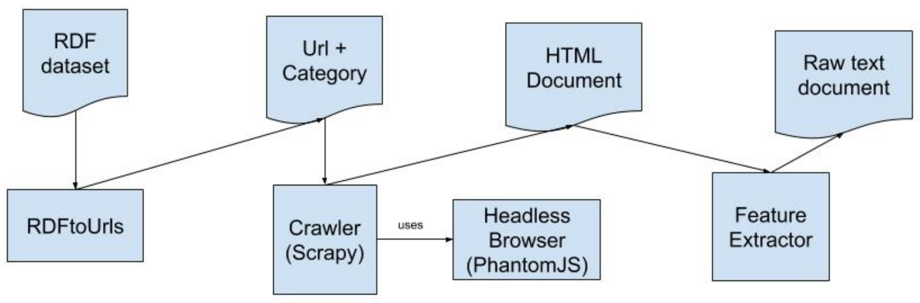


FIGURE 5.1: Data Extraction Proces

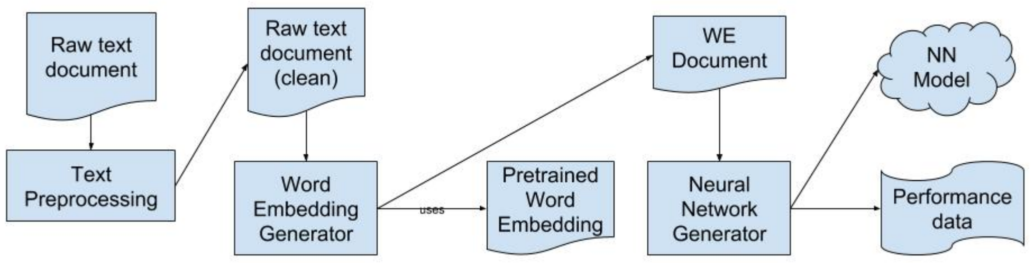


FIGURE 5.2: Model Training Proces

5.2 Task description

This section will cover the main tasks that have been considered to make this project. We will not cover tasks that imply a single execution that takes less than 3 hours, to simplify this document. Namely we will not be covering the execution of the feature extractor, the execution of the text preprocesser and both the execution and implementation of the 3 programs used to generate new datasets.

5.2.1 Planning and scheduling

The project starts by planifying which tasks will be needed and our schedule for doing them. Due to how big this project is, it is important to have a solid foundation. We have to also think about these tasks in terms of which tasks block each other and which do not. For example one may think that until we have extracted some pages we would be unable to start working on our word embeddings, but once we have a few documents we can already get to work on them. We could even have done some experiments on NNs to see whether their generator program worked as expected untill the end.

5.2.2 Acquire background in Deep Learning techniques

When we started my project, our knowledge in Deep Learning was a bit shallow. We knew the basic theory and principles behind it, but our experience with deep learning projects and Keras was very low. We had to familiarize ourselves with the libraries used in this project and the specific Python framework. We enrolled in Automated Learning and Data mining subjects from FIB the past term, and plan on using the contents we learned on the project.

5.2.3 Download dataset and Implement RDF parser

During this task we had to download the rdf file content.rdf.u8.gz from the DMOZ archive[36] and manage to obtain the urls together with their categories. While the compressed file weights only 250MB, the site has a very high latency and unstability, to the point which the browser cancelled the download a few times. Downloading the file took approximately 6 hours. Then, we had to program a Python script that simplifed the data on the RDF file, to alleviate the work of our crawler and make it easier to diagnose errors that affected the crawler.

5.2.4 Implement Crawler

During this task we had to program the crawler spider. We started with a simple implementation, then realized that most of websites were dinamically loaded and integrated the use of Selenium and PhantomJS in it. We also had to customize the parameters (according to my hardware limitations) and into specific recommendations for the kind of crawling we did[18]. The implemented crawler had to be able (with minimal changes) to also scrap newly specified pages (for the final version of our classifier).

During this step we programmed the crawler, extracted the raw pages and learned what information from the page is relevant (such as metas). We had to investigate page structures and how we may be able to get some extra topic information from non-visible parts of webpages.

5.2.5 Broad Crawling

This task is simply to execute the crawler with the urls obtained from the DMOZ dataset. This will take quite a bit of time, since we must attempt to crawl almost 2 milllion pages. This process will also have to be semi-monitored since we must remove the URL groups of 5,000 once it has been crawled so that we do not lose progress. We must also take into consideration that some of the access points used for this project may be too busy at times to catch most of the websites. We will have to make sure that when one of the URL groups finishes most of its (available) URLs have been crawled; if it has not we will not remove it and run again our crawler on it.

Another thing to consider is to prioritize the minoritary categories. Our dataset is very extensive; given the time limitations of the project this will ensure we get each category well represented. While this task's workload is not very high, it will take considerable time.

5.2.6 Implement feature extractor

This task consists in the implementation of the program that will get the text from the html documents using Beautiful Soup. The specifications regarding this implementation have been mentioned in the scope.

5.2.7 Implement text preprocessor

This task consists in the implementation of the program that will preprocess our text documents using NLTK. The specifications regarding this implementation have been mentioned in the scope.

5.2.8 Implement conversor to WE

This task consists in the implementation of the programs that will use each word embedding and a document to generate each document's representation. The specifications regarding this implementation have been mentioned in the *Obstacles and Solutions* chapter, taking only the time into consideration for the very first version that worked.

The prolonged time needed for this task has been included in Testing Text Transformations.

5.2.9 Transform text to WE

This task consists in the execution of the programs that will use each word embedding and a document to generate the document's representation.

This task has been very slow due to having to work with very big files(between 3GB and 6GB) and many small files(almost 200,000). The strain of this task was initially on the disk, but in the final version of this project all the file requirements have been loaded to memory. These makes the task much faster, but it adds a constraint to our execution. This makes it so that only the machine with 16GB of memory is able to run it(without using the swap partition), so we cannot execute two of the programs concurrently. While we will use a lot of the computer's resources, this task should is not monitored so it should not put a lot of strain on the developer. The only issue is that while one of these tasks is running memory should be monitored so that we do not have to use the swap partition.

5.2.10 Testing all the text transformations

This is a task that we will consider separated from the implementations, as we will have to do it concurrently to our crawl. This implies we are likely to encounter new problems due to new webs, and we will have to address and correct them.

What this task includes: detecting errors during the executions of text transformations (feature extractor,text processor, WE conversor) and correct them. When we implement the former programs we will attempt to make a very simple version of them ensuring it works with a very limited number of pages. As we progress in our project, it is likely that a simple implementation is not enough to handle all of the dataset, so we will have to check weekly and correct the new errors that appear.

This task includes the multiple reimplimentations of the WE conversors. Due to fact that we had to implement these programs multiple times to fit our time and hardware constraints, this task prolonged a lot, as in explained in chapter *Obstacles and Solutions*.

While this task should progressively occupy less time, we cannot consider it complete until we have seen how it behaves with our whole dataset. This task has some workload but not all of the time while it is active will be dedicated to it.

5.2.11 Implement NN generator and trainer

This task consists in the implementation of the programs that will read each document's representation and feed it to a NN, then save said NN with some statistics. The specifications regarding this implementation have been mentioned in the scope.

5.2.12 NN Training and Tuning

This task implies analysing which parameters work best for our NN and feeding it with our dataset, then saving the results. This means trying to use different architectures and parameters to solve the problem of classifying correctly. NN are known for having many parameters to tune such as width, depth, learning rate, activation functions, decay, epochs on training, etc. This implies doing iterative experiments and solving high bias and variance, while ensuring our model does not overfit.

5.2.13 Result analysis

This tasks consist in visualizing the results, analyzing them, drawing conclusions from them and making a final choice for our model. We will have to understand their weaknesses and strengths, and choose which adequates best for our objective.

5.2.14 Implement prototype

Given the conclusions drawn from the final model, we must build a prototype. That is to implement a relatively simple program that can read an url and obtain the category it predicts for this website. This implies joining all the other parts of the project and using the trained model.

5.3 Time estimations

TABLE 5.1: Time estimations for each of the project's active tasks

Developer Task	**Hours**
Planning and Scheduling	70
Acquire background in Deep Learning techniques	60
Download dataset and Implement RDF parser	15
Implement Crawler	45
Implement feature extractor	30
Implement text preprocessor	25
Implement conversor to WE	35
Testing all the text transformations	100
Implement NN generator and trainer	30
NN Training and Tuning	60
Result analysis	20
Implement prototype	8
Total	498

Estimations based on a 730 hours per month basis.

TABLE 5.2: Time estimations for each of the project's semi-automated tasks

Semi-Monitored Automated Task	**Hours**
Broad Crawling	2555
Transform text to WE	288
Total	2843

5.4 Gantt chart

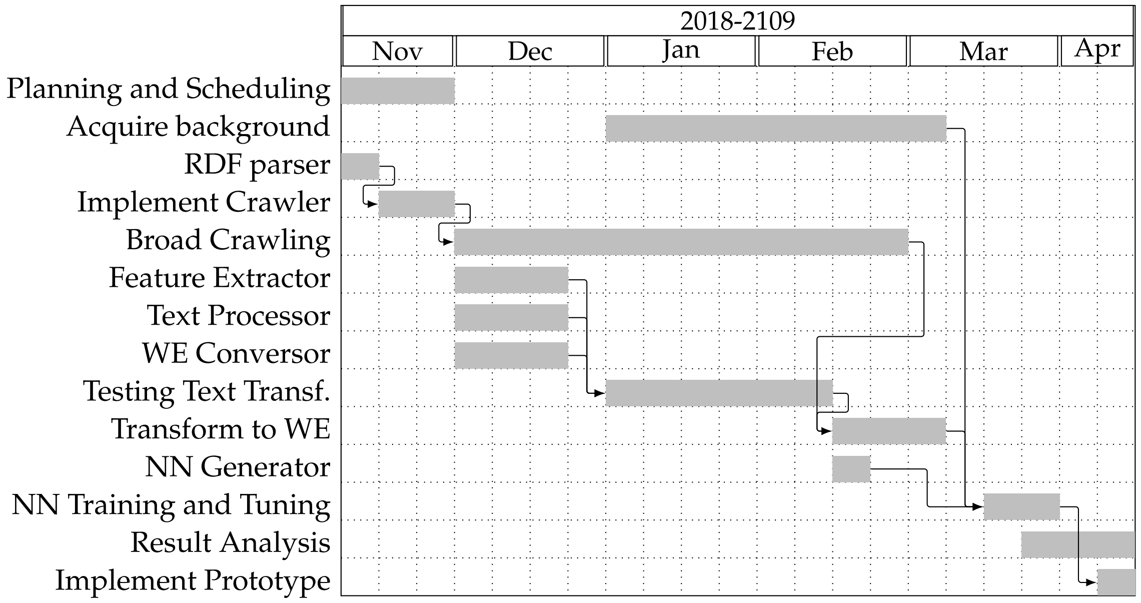


FIGURE 5.3: Gantt Chart

When looking at this Gannt chart we must remember that the length of a task does not necessarily correlate to its workload. What that means is that the tasks Broad Crawling, Transform to WE and Testing Text Transf. do not require someone to be paying attention to them all the time, thus doing them concurrently is possible.

We must also take into account the fact that the time writing this project's memory was spent concurrently to this project, and documenting each of the tasks and getting information from them has also been a part of the work that is not reflected in the Gannt.

5.5 Action Plan

We can see that some of the weeks, tasks that are planified do not cover all of the hours that should be in a week's working time. This is done intentionally in order to leave some time for the developer to properly document the project done and write this document. This also was done in order to have some mechanism to be able to prepare for obstacles and allow us to at least alleviate the work in the last weeks.

Consecutively, we could leverage how prepared we were to be to execute a lot of transformations in the last weeks and how much documentation and background we had. This greatly helped in regards to not letting our work be blocked by the Broad Crawling. During the time that we were blocked we had time to reimplement the WE conversor, have an estimation of how long it would take to execute, modify all the programs so that progress was saved, etc. Besides, once our schedule began being delayed due to not having completely crawled the websites, many of the url groups were delayed for the crawling and we left only the ones with underrepresented topic categories.

Chapter 6

Budget

6.1 Hardware Budget

For this project two personal computers were required, other than that no other harward cost should arise. We will use a total of 5 months and a half, 0.46 years. Estimating the useful life of the computer at 4 years (as is established per law), amortization translates to approximately 11.5% of time, thus that same percentage of the product price.

TABLE 6.1: Hardware budget

Product	Price(€)	Units	Useful Life	Amortization(€)
Desktop PC 16GB	1100	1	4 years	126
MSI - GL62 6QF 8GB	956	1	4 years	110
Total	**2056**	**-**	**-**	**226**

6.2 Software Budget

Since we are only using open software we will not have any additional budget costs for our software. The only of our expenses should be the use of a Github professional account, but we are verified by Github as a student it is free regardless the amount of repositories.

TABLE 6.2: Software budget

Product	Price(€)	Units	Useful Life	Amortization(€)
Github Student account	0	1	-	0
PhantomJS	0	1	-	0
Selenium	0	1	-	0
Scrapy	0	1	-	0
BeautifulSoup	0	1	-	0
Gensim	0	1	-	0
NLTK	0	1	-	0
Keras	0	1	-	0
Total	**0**	**-**	**-**	**0**

6.3 Human Resources Budget

This project will be developed by single person, which will carry out all the roles required. As mentioned before, the project manager will be the developer as well. The extra hours are considered for the semi-monitored tasks.

TABLE 6.3: Human resources budget

Role	**Hours**	**€/Hour**	**Salary(€)**
Project Manager	150	50	7,500
Developer	350	30	10,500
Total	**150**	**-**	**18,000**

6.4 Indirect costs

Our project will have other non-related to the project itself costs, such as electricity and internet connectivity. Electricity is based on the estimated cost of a 330W per hour, taking into account the fact that our computer will be open for all the Broad Crawl. We will also take into account the fact that our two computers will be running concurrently for this project. This table reflects an estimate of them:

TABLE 6.4: Indirect costs budget

Product	**Cost**	**Units**	**Estimated cost(€)**
Electricity	0.14€/kWh	1700kW	238
Internet supplier fee	35€/month	5.5months x 2AP	385
Total	**150**	**-**	**623**

6.5 Total cost

Given the estimations we have for each kind of cost, we can calculate the total project budget:

TABLE 6.5: Total cost budget

Expense type	**Cost**
Hardware	226
Software	0
Human Resources	18,000
Indirect	623
Total	**18,849**

Chapter 7

Sustainability

7.1 Sustainability Matrix

TABLE 7.1: Sustainability matrix

-	**PPP**	**Useful life**	**Risks**
Environmental	Only hardware and electricity consumption, very low impact. 9/10	Cost of keeping the deep learning system running, very low impact. 10/10	May not be as energy efficient as other techniques, may not work as well as we expect it to. -2/-20
Economical	Big human resources investment, high investigative value. May help give some insight into word semantic analysis. 6/10	Can help approach people to the content they are looking for. Applicable to improve search engines. 9/10	May not give a competitive result. -7/-20
Social	Investigation on modern key technologies, combination of them . 8/10	Companies can use this project's results. This can be useful for companies that need to provide any content through web. 9/10	Technology proving to be limited, ineffective or just not applicable the combination of fields it studies. -3/-20
Sustainability	**23/30**	**28/30**	**-12/-60**

7.2 Environmental

Our project's resources are limited to a computer and its electricity and connectivity costs, which implies our environmental impact will be minimal. Therefore during the PPP the impact is limited to the 1700 kW that will be spent for creating this project. Since few resources will be used, there is very little room for the reutilization of them. We will also reuse as many libraries as possible that are already developed and reliable (tensorflow and nltk to give an example).

During its useful life this project may save up time for a lot of companies requiring information in the subject. The main saving for companies comes to having a good resource in order to recommend content to their own employees or clients. As an example, this may save time to a client that wants information on a specific service from said company. The results of this project can easily be adapted to process a simple text query, then we could attempt to classify this query into a category and provide only the urls from the various subdomains the company's page has that get classified under that subdomain. This may save a lot of energy simply from having to load more and more pages until that client reaches the information he wanted.

The results of this investigation should reduce the time an user needs to spend in front of a computer to find about a certain topic, thus reducing the overall ecological footprint.

Finally, the main risk is that this project determines that NNs together with W Eare not adequate to classify by web topic. Currently refining methods and machine learning algorithms are used for solving this kind problem, but other combinations of them may prove to be more effective and thus save up costs in human resources and kW spent. The way we are facing this problem may provide a better way of solving web-topic classification. Alternatively we may get inconclusive results, in which case our results will discourage people from spending the resources for this case again.

7.3 Economical

Economically, this project's main cost is in Human Resources. As mentioned in the environmental aspect of sustainability, the energy cost this project has should be balanced by the benefits it will bring. 18,850 euros is a lot of money, but since the benefit it can bring is precisely lessen cost of human resources for a company it should be worth it. By that what I mean is that having a good content recommendation algorithm should save up a lot of employee's work in giving support to a client, and thus save a lot of money in salaries while providing the same service.

The main concern economically are the resources dedicated to have a project manager and a developer working in this project. It is clear that for both the experience obtained will be useful, as the techniques the project will use have proven to be applicable to many fields. There is also the fact that the project may discover a better technology for an interesting field. The risks of this project turning being unsuccessful in improving the current solutions are there, but having better information of whether they can work together has its value, even in the worst case scenario. Once again, economically it can be a better method than the ones that are used (which with a 70-90% efficacy leave room for improvement).

7.4 Social

During this project, the developer will get more experience in modern key technologies and how one may combine them together. Natural language analysis is a field that is notably growing nowadays, we can see many tech companies aiming each time more at creating robots that humans can communicate to in a human way (Google Assistant, Amazon Alexa,etc). Having a bit more information on how to numerically represent knowledge is without any doubt a valuable skill.

Once the project is complete, the benefit will be another field where we have tested Word Embeddings combined with Neural Networks and the results obtained in it. This information may help a lot of companies (companies searching websites of competition with similar product, searching engines,etc) and investigators. Among the direct social risks this project may bring none are significant. The only risk is the already mentioned fact in economical and environmental aspect: that this may be a resource waste. That is that the work of the developer does not bring a conclusive response, which may diminish his motivation.

Chapter 8

Results

8.1 Data Extraction results

8.1.1 Feature extraction results

After running the broad crawl over various months, we obtained a total of 201,535 documents. From these, we were able to extract text features and get a total of 195,218 text-only documents. After that we were able preprocess and tokenize all the text-only documents except one, and we generated the following amount from each WE:

TABLE 8.1: Amount of documents generated on each WE.

WE	Amount
Glove	195,217
word2vec	190,927
lexvec	195,217

8.1.2 Results per category

The following table shows us the amount per category obtained. As we can see all the categories have at least 1,000 documents, and our dataset is very unbalanced. In general, this is a good amount of data, and our NN should have enough instances for most classes, ideally a few more of the News category should be helpful. Seemingly, the few websites that Word2Vec was not able to transform into vectors did not alter significantly how our categories look, this is not a surprise as it is only a 2.5-3%, but was worth checking just in case there was some pattern.

TABLE 8.2: Distribution per Category of the documents used with Word2Vec.

Topic	**Amount**	**Percentage**
Business	33711	17.66
Arts	31153	16.32
Society	27,640	14.48
Science	20,698	10.84
Sports	15,903	8.33
Computers	15,124	7.92
Shopping	12,579	6.59
Recreation	11,987	6.28
Health	7,989	4.18
Reference	6,310	3.3
Games	3,401	1.78
Home	3,324	1.74
News	1,108	0.58
Total	**190,927**	**100**

TABLE 8.3: Distribution per Category of the documents used with Glove and LexVex.

Topic	**Amount**	**Percentage**
Business	34664	17.75
Arts	32097	16.44
Society	28086	14.38
Science	21158	10.84
Sports	16147	8.27
Computers	15380	7.88
Shopping	12,834	6.57
Recreation	12324	6.31
Health	8085	4.14
Reference	6,410	3.28
Games	3,519	1.80
Home	3,348	1.72
News	1,165	0.6
Total	**195,217**	**100**

8.2 Neural Networks

8.2.1 First Experiments

We will begin by training each model on each of the datasets we have generated. We want to see which datasets are the best option to generate our model. In order to achieve that, we will do a decent amount of training while using each, without focusing too much on optimizing them. As explained in the *Scope* section, we have a total of 12 datasets. Usually it is recommended to have a test dataset separate from the beginning, to be able to have a reliable estimation of how our model performs in real life. Since we are using approximately 20,000 files as validation and this is just a step to discard badly performing datasets, we will base our choices solely on the validation results. We must also take into account that while we will show the results of F1 are the only real performance statistic, and the Loss function is just some extra information on how our model progresses.

To compare them fairly we will be using the same architecture on all of them: 5 hidden layers, 300 units per hidden layer, and 80 epochs of training. The input layer will also have 300 neurons, and the output layer will have 13, one per category. Our activation function will be relu[53] for all the layers except the last, where it will be softmax[54]. We will use 90% of our data for training, and the rest for validation. Another thing worth mentioning is that we will be using F1'Micro as our measure instead of accuracy. Accuracy can be a really bad measure for datasets that are unbalanced, since it may make the model ignore the minoritary categories, thus we will use F1-Micro. F1-Macro is a reasonable option, since it focuses much more on making the individual classes work in a balanced way. However, we will use F1 Micro, since our dataset is not that horribly unbalanced and it is computationally cheaper.

Simple normalization

This normalization picks the maximum and minimum across all the values in all the vectors that represent our words. Then for every value in every vector, it divides that value by the maximum if they are positive, and by the positive version of the minimum if the value is negative. This is the most simple method of turning all the values between -1 and 1. The results are the following:

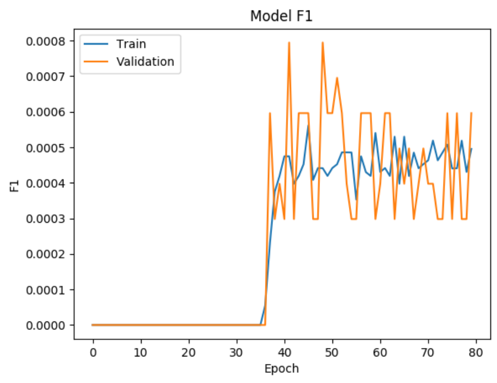


FIGURE 8.1: Glove's F1 score over 80 epochs simple normalization

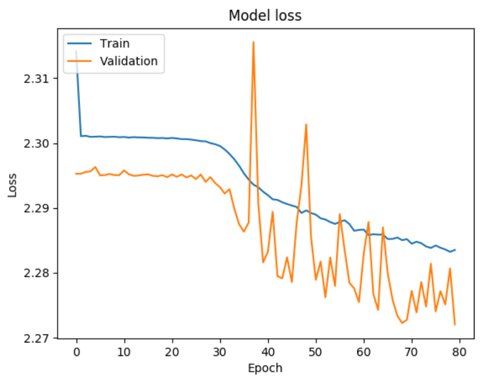


FIGURE 8.2: Glove's Loss function over 80 epochs simple normalization

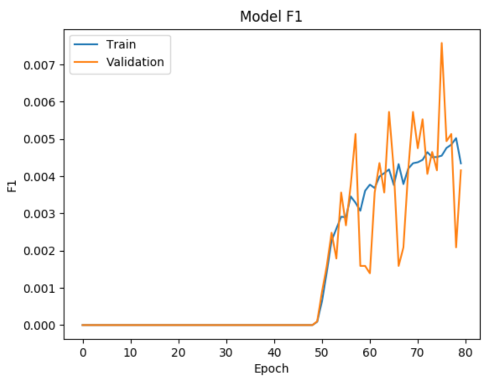


FIGURE 8.3: LexVec's F1 score over 80 epochs simple normalization

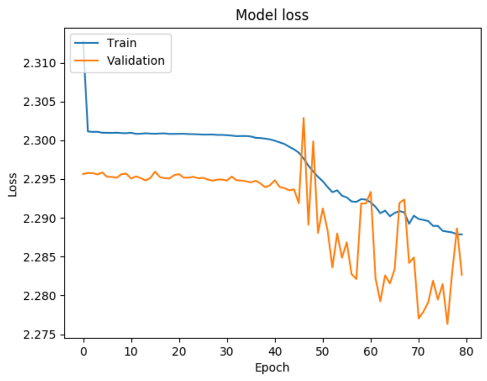


FIGURE 8.4: LexVec's Loss function over 80 epochs simple normalization

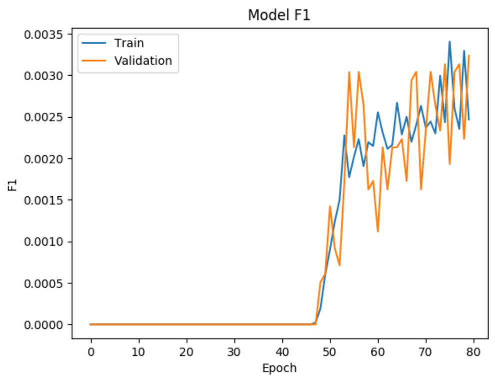


FIGURE 8.5: Word2Vec's F1 score over 80 epochs simple normalization

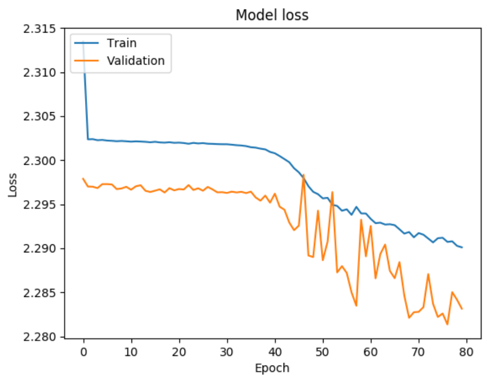


FIGURE 8.6: Word2Vec's Loss function over 80 epochs simple normalization

The results with this normalization are beyond bad: We do not even reach a 0.1 score F1. After looking at the dataset itself instead of the results, turns out most of our values get turned to 0. This means most of our dimensions are useless and our dataset barely has any information. As we can see from the graphics our score does not even take off until at least 50 iterations.

Normalization by dimensions

The next dataset we tested was by applying a normalization to each dimention separately, that way if some dimension has unusually high values for some reason we may be able to correct them. We once again treat positive and negative values separately. This is a risky transformation to do, as we are be altering the original representation behind the word vectors. They following are the performance metrics on the results:

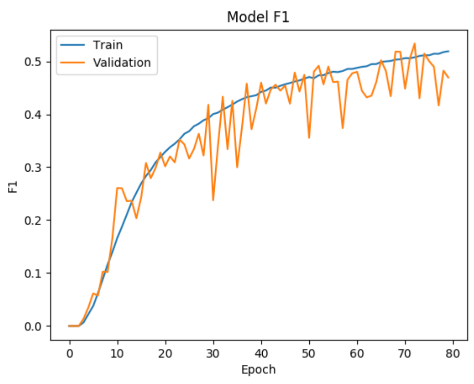


FIGURE 8.7: Glove's F1 score over 80 epochs normalizing per dimension

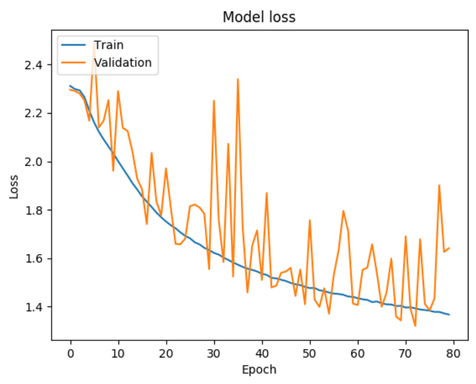


FIGURE 8.8: Glove's Loss function over 80 epochs normalizing per dimension

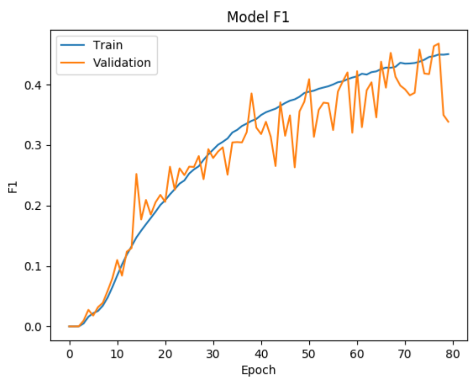


FIGURE 8.9: LexVec's F1 score over 80 epochs normalizing per dimension

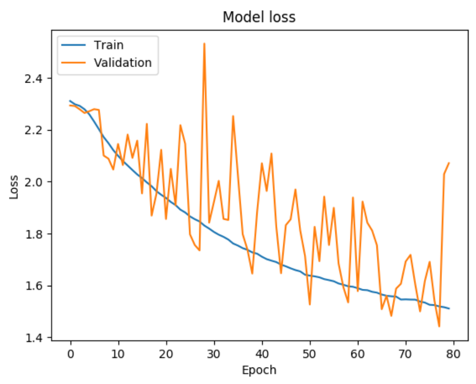


FIGURE 8.10: LexVec's Loss function over 80 epochs normalizing per dimension

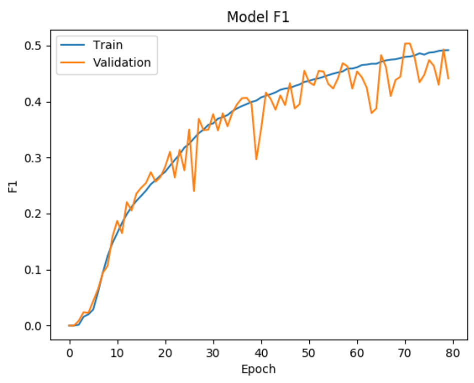


FIGURE 8.11: Word2Vec's F1 score over 80 epochs normalizing per dimension

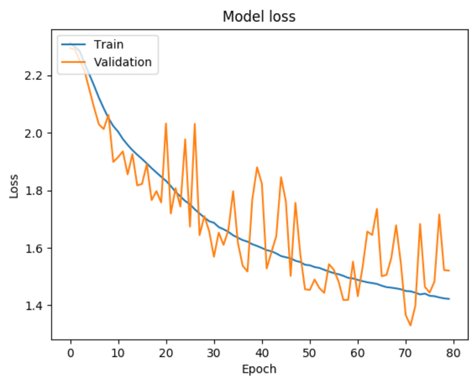


FIGURE 8.12: Word2Vec's Loss function over 80 epochs normalizing per dimension

We can see quite an improvement from the simple normalization model, so this model likely does not erase as much relevant information. We can also see a very smooth F1 and Loss improvement throughout training, and it seems like even after 80 epochs we did not reach a point where we began overfitting. We should also notice that lexvec seems to have a slower progression compared to the other two WE. The F1 value is still pretty low, but if none of the other datases perform any better this could be a starting point (although not a very good one).

Transform to unit vectors

In this dataset we changed every document's vector modulum to 1. This was done simply dividing the every value in a vector by the modulum of that vector (calculated as root of the sum of the vector's values). The intuition here is that longer documents are likely to have longer vectors due to how we combine the values of words. Ideally we should do this while summing the word vectors for each word and dividing by the amount of them, that should give us the "average word" the word embedding represents (which should be easy to relate to a topic). As we have mentioned in the *Obstacles and Solutions* section due to time constraints we were not able to do that.

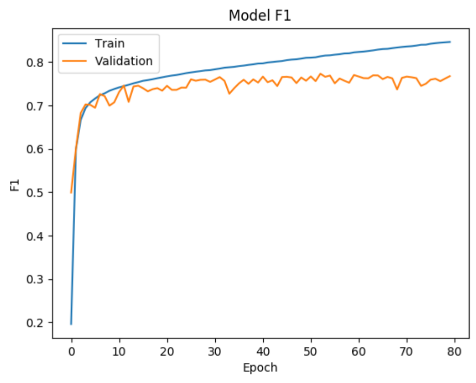


FIGURE 8.13: Glove's F1 score over 200 epochs normalizing to unit vectors

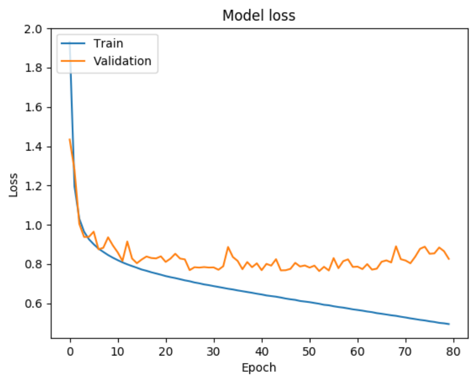


FIGURE 8.14: Glove's Loss function over 200 epochs normalizing to unit vectors

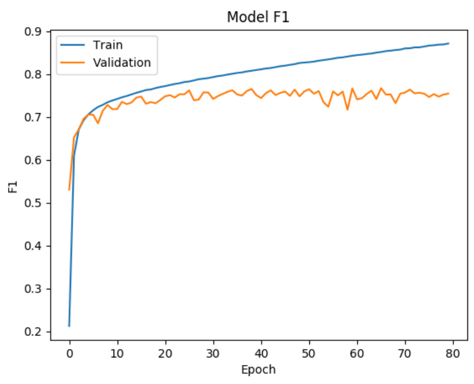


FIGURE 8.15: LexVec's F1 score over 200 epochs normalizing to unit vectors

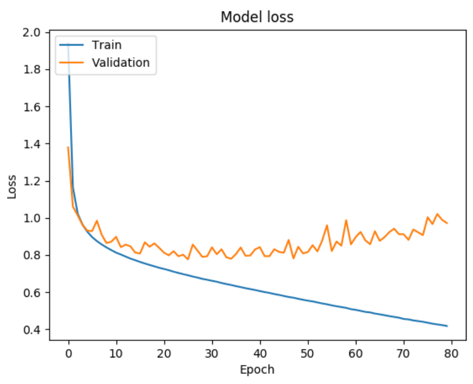


FIGURE 8.16: LexVec's Loss function over 200 epochs normalizing to unit vectors

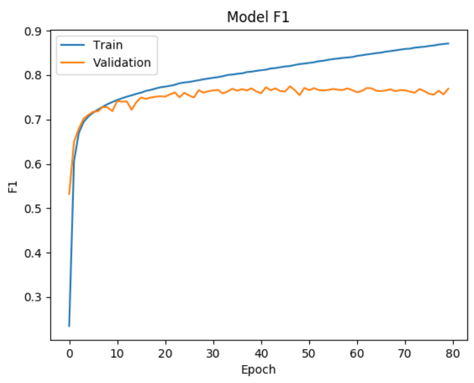


FIGURE 8.17: Word2Vec's F1 score over 200 epochs normalizing to unit vectors

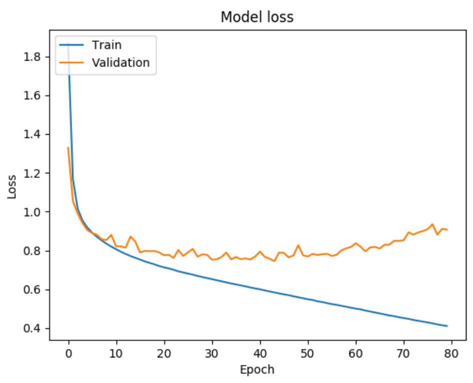


FIGURE 8.18: Word2Vec's Loss function over 200 epochs normalizing to unit vectors

We are finally getting some decent results, ending up with an F1 of approximately 0.75 before we overfit. Once again we see the that during the training the F1 and loss values create a very smooth line. This is probably due to the fact that we are using 90% of our data for training, and thus its fitting can improve progressively. We can also see that in validation our F1 stays relatively stable even once we start overfitting and our loss function is also stable and barely increases. This is surprising as we would expect our validation loss to increase at the rate our training loss decreases. We should also notice the fact that Word2Vec seems to have a more stable validation line, maybe due to being a simpler WE.

Base dataset

Here we attempted to plan and simply feed the dataset to the specified NN. This experimentation is necessary to see if some kind of normalization is even applicable to our dataset. Theoretically, it should make the work of our NN more difficult since it will have to normalize by itself, but by normalizing we may be simplifying dimensions that gave relevant information. The results are in the following page.

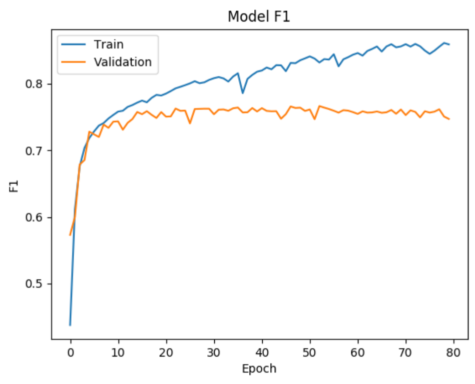


FIGURE 8.19: Glove's F1 score over 80 epochs

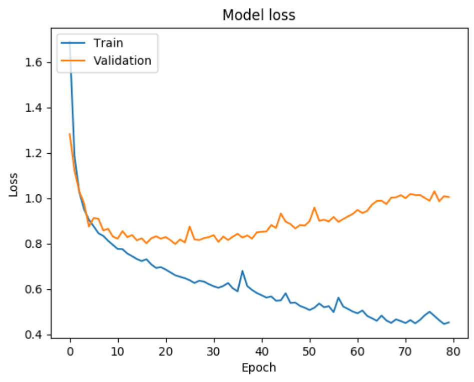


FIGURE 8.20: Glove's Loss function over 80 epochs

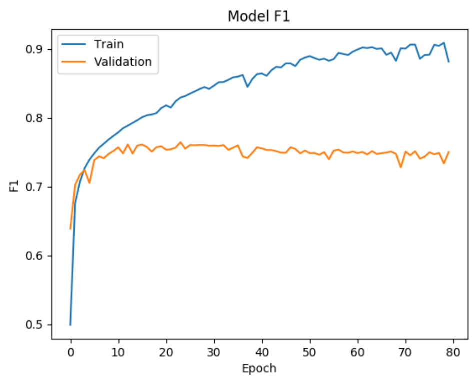


FIGURE 8.21: LexVec's F1 score over 80 epochs

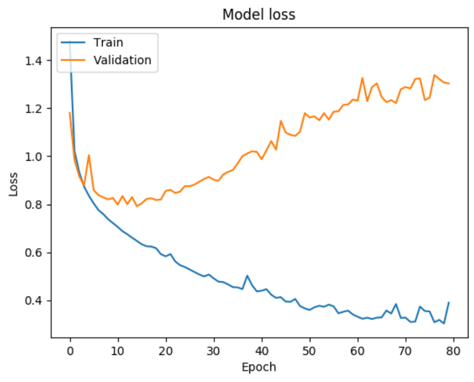


FIGURE 8.22: LexVec's Loss function over 80 epochs

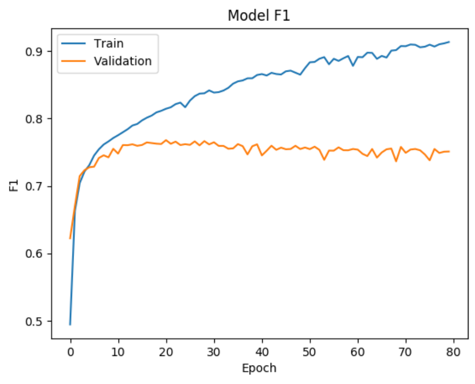


FIGURE 8.23: Word2Vec's F1 score over 80 epochs

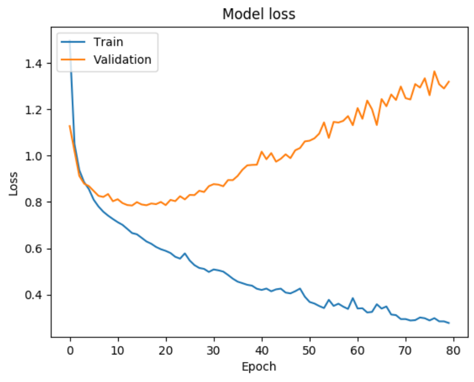


FIGURE 8.24: Word2Vec's Loss function over 80 epochs

We can see results that are pretty similar to transforming to unit vectors. That is not a huge surprise, as we are not intrinsecally transforming the data (like when we normalized by dimensions) nor we are losing a lot of important information (like when we did a simple normalization). We have once again an F1 of approximately 0.75 across all word embeddings, which is somewhat decent. Unlike while normalizing by dimensions and turning to unit vectors, we have a non smooth training curve, probably due to the fact that the values in our vectors can be above 1,000. This makes small optimizations risky and thus the more unstable curve. The main difference compared to our other best model(unit vectors) is the fact that our loss function diverges a lot more. The validation loss and the training loss grow apart as epochs pass. We can also see that our models do a much better prediction on first iteration.

Final choice

In the end, the real choice is between the unit vectors dataset and the base dataset, since the other two do not even reach a F1 score of 0.5. The real differences between the other two are very basic: the base dataset's loss function diverges much more (in training and validation) when overfitting and the base dataset's training F1 is much less smooth.

For our next experiments we will work only using the unit vectors dataset. A smoother training curve means a steadier progress for our model. This should translate to having an easier time when generalizing the function, thus having less work to do by our model. Regarding convergence, we are not very sure about whether less convergence is positive or not. Less convergence means our model overfits less, but it also means we will not have such a clear picture about whether our model overfits. Given that they have more or less reached the same peak performance, we think that both things could be useful.

8.2.2 Second Experiments

Here, we have used only the unit vectors dataset and attempted to experiment with some of the parameters to try and tune our network. Since we are using big amount of data we will not be able to do too much experimentation, but we expect to hopefully be able to increase the performance shown in the first experiments.

Most of the parameters used will be the same, but we focused on experimenting with the learning rate, lambda for L2 regularization and Dropout. The reason for this is that we observed that if we kept training on the data the training F1 got bigger although our validation F1 became stagnant. The most similar to this is overfitting, which is addressed with L2 regularization and Dropout. In general, these are methods to prevent a model from not generalizing well. The reason for experimenting with learning rate is that it is easily that most alters the performance of a model. We also chose to use a single dataset for this experimentation (lexVec), as all WE seem to behave similarly.

Learning rate

For the results in this experimentation we will not show the Loss function, as it gives us no extra information. The results are the following:

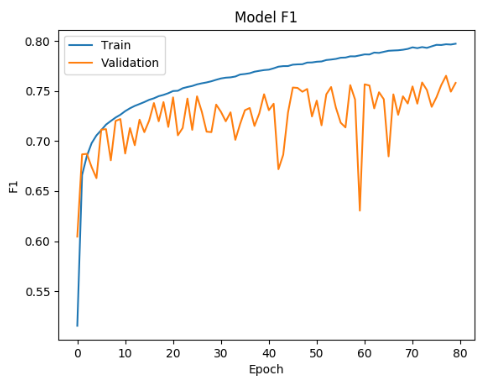


FIGURE 8.25: Unit vector LexVec's F1 score over 80 epochs with lr=0.1

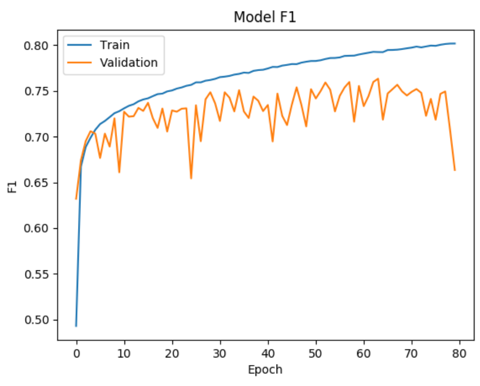


FIGURE 8.26: Unit vector LexVec's F1 score over 80 epochs with lr=0.05

We can observe that values this high make our F1 fluctuate a lot; they are probably not the best values to slowly train our NN.

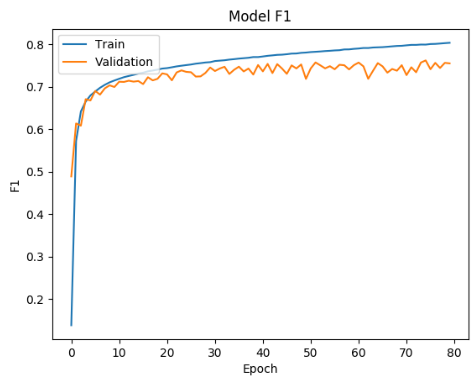


FIGURE 8.27: Unit vector LexVec's F1 score over 80 epochs with lr=0.01

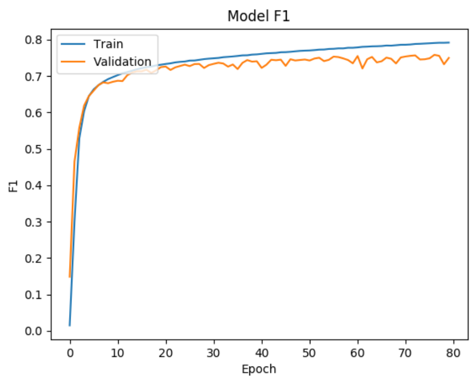


FIGURE 8.28: Unit vector LexVec's F1 score over 80 epochs with lr=0.005

We can see that values around this range make our NN change slowlier, and thus allow it to progress more steadily. They both manage to get to an F1 of 0.7 and the validation fluctuates a bit without getting too low, while our training increases. They both are fine values, but we will use 0.01 as we seem to have issues getting stuck on the 0.7-0.75 validation. At this point decreasing more the learning rate will only make our results stagnant, it guarantees our validation will get stuck on local minima and our network will take longer to reach 0.7-0.75, as we can see with a learning rate of 0.001 and 0.0005:

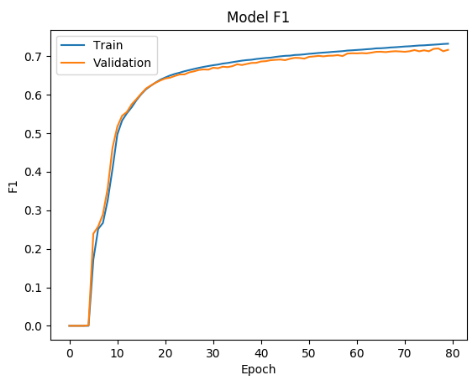


FIGURE 8.29: Unit vector LexVec's F1 score over 80 epochs with lr=0.001

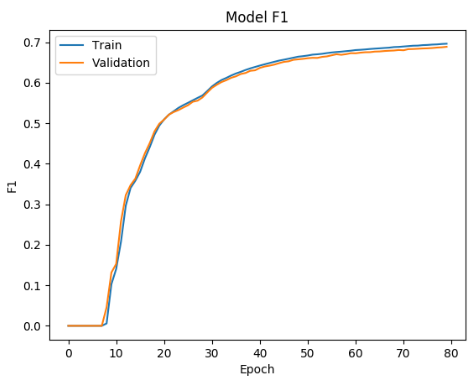


FIGURE 8.30: Unit vector LexVec's F1 score over 80 epochs with lr=0.0005

Lambda with L2 regularization

Initially, we wanted to try and guess what the correct range of the lambda was, so we used some very spaced points. We tried 0.1, 0.01, 0.001, 0.0001 and 0.00001, but the two first parameter values proved to be too high for our network and it was not able to get above an F1 of 0.5. Thus, we will only show the others:

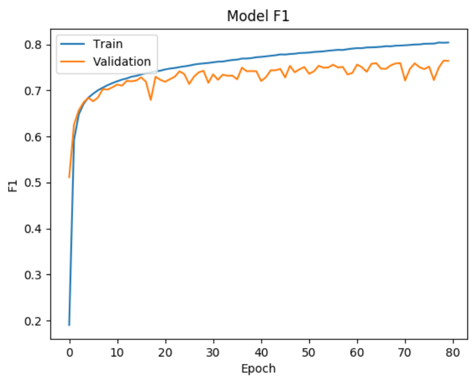


FIGURE 8.31: Unit vector LexVec's F1 score over 80 epochs with lambda=0.001

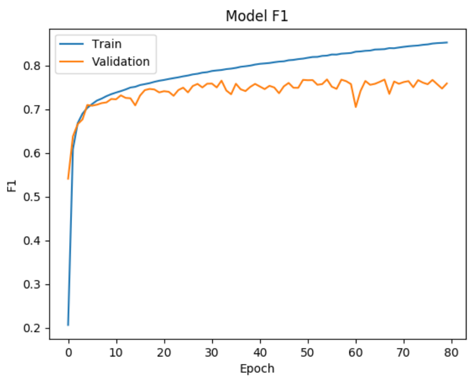


FIGURE 8.32: Unit vector LexVec's F1 score over 80 epochs with lambda=0.0001

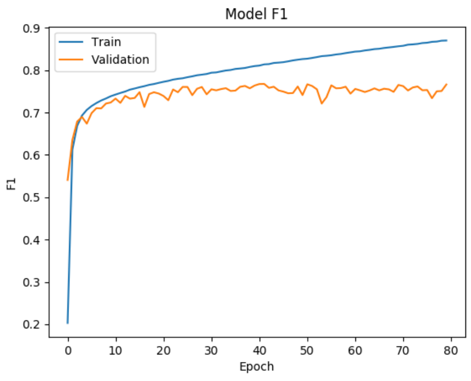


FIGURE 8.33: Unit vector LexVec's F1 score over 80 epochs with lambda=0.00001

As we can see the only difference is how fast our network is able to memorize the training dataset. We can see that in validation there is barely any difference. At this point, we even tried to use some dropout to try and fit better our validation set, but it gave us no bigger success than playing with lambda.

Visualizing our results

Since all the changes and experimentation we did to our NN barely had any impact to its performance, we displayed some of the statistics from the model to try and see if some specific category was having issues. These were the statistics obtained while atttempting to use the model to classify the whole dataset (training and validation):

TABLE 8.4: Precision,Recall, F1 per category of the classifier using Glove.

Topic	Amount	Precision	Recall	F1
Business	34664	0.7	0.81	0.75
Arts	32097	0.82	0.81	0.82
Society	28086	0.81	0.79	0.80
Science	21158	0.82	0.75	0.78
Sports	16147	0.88	0.86	0.87
Computers	15380	0.90	0.71	0.79
Shopping	12834	0.57	0.77	0.66
Recreation	12324	0.60	0.73	0.66
Health	8085	0.83	0.77	0.80
Reference	6410	0.76	0.38	0.51
Games	3519	0.83	0.62	0.71
Home	3348	0.89	0.77	0.83
News	1165	0.00	0.00	0.00
Micro-Avg	**195,217**	**0.77**	**0.77**	**0.77**
Macro-Avg	**195,217**	**0.72**	**0.67**	**0.69**

The problems in this table are pretty obvious. The model does not have the slightest clue on how to generalize on the News category. One of the reasons is probably due to news not having a specific lexic, but this is probably heavy aggravated by the fact that they represent less than 1 percent of our dataset. After looking at some of the documents in news manually, there seems to be no other reason for this. It seems like in general it is very difficult to identify the kind of words News pages use.

Since we cannot get our model to categorize News webpages at all, we should try and see what results we get from using the exact same parameters for the network but remove all the News webpages. Besides, we can see that both References and Shopping have a very low F1 as well. Looking at the rest of categories, they also have the issue of not having a specific lexic that revolves around the word. Shopping webpages may have some ocasional word such as 'discount' and 'offer', but that is not that likely to appear on the websites (it is more likely that we will see description of the product and produc names). The references category will have the same issue but for different reasons: it will either contain some non-textual results (such as maps) or it will contain textual results that do no necessarily have any relation with the category as they are titles from some specific domain.

We should also see how the results change once we remove them, since all of our word embeddings try to define a word according to its context, but they do not give a word vector for a whole context.

8.2.3 Final Results

We will do new round of experiments, this time attempting to remove the category our NN struggles with the most and also trying to remove all the categories that have no specific lexic them. We will keep the best values found in the last experiments: lambda=0.001 and lr=0.01 and train them through 100 epochs.

No News Results

These are the results obtained by removing all the documents that belonged to the News category from our dataset. In general, our overall results do not change much. The main benefit we get is a tiny boost in the F1 obtained, of around 0.01-0.02 in the f1 score. There are no huge differences between various WE regarding the F1 and loss:

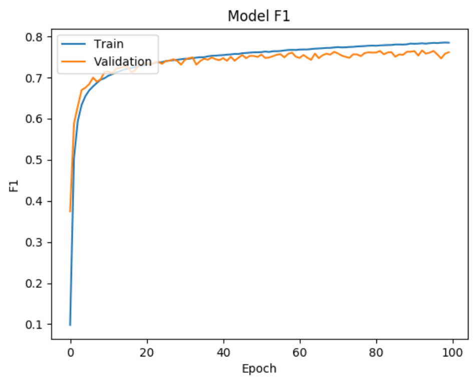


FIGURE 8.34: Unit vector Glove's F1 score over 100 epochs excluding News

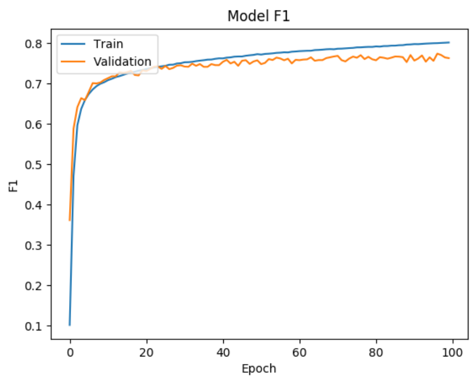


FIGURE 8.35: Unit vector Word2Vec's F1 score over 100 epochs excluding News

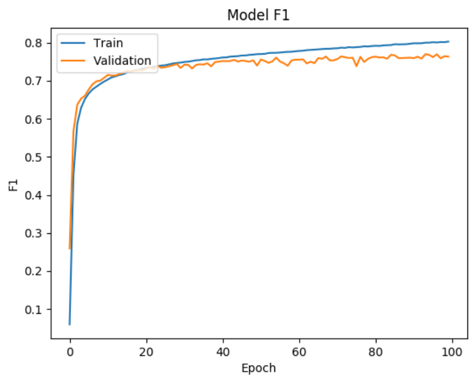
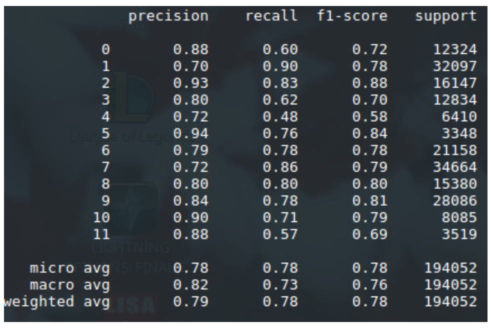


FIGURE 8.36: Unit vector LexVec's F1 score over 100 epochs excluding News

We also get the following statistics by trying to see how our algorithm performs on the whole dataset:

TABLE 8.5: Precision,Recall, F1 per category of the classifier using Glove(no News).

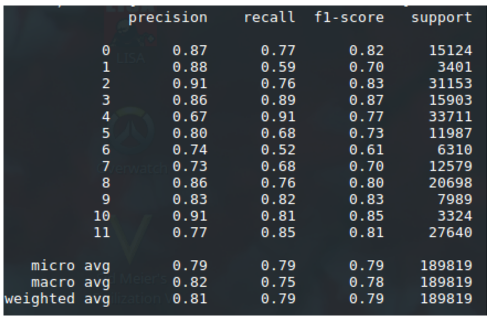


```
              precision    recall  f1-score   support

           0       0.88      0.60      0.72     12324
           1       0.70      0.90      0.78     32097
           2       0.93      0.83      0.88     16147
           3       0.80      0.62      0.70     12834
           4       0.72      0.48      0.58      6410
           5       0.94      0.76      0.84      3348
           6       0.79      0.78      0.78     21158
           7       0.72      0.86      0.79     34664
           8       0.80      0.80      0.80     15380
           9       0.84      0.78      0.81     28086
          10       0.90      0.71      0.79      8085
          11       0.88      0.57      0.69      3519

   micro avg       0.78      0.78      0.78    194052
   macro avg       0.82      0.73      0.76    194052
weighted avg       0.79      0.78      0.78    194052
```

FIGURE 8.37: Unit vector Glove's F1 per category excluding News

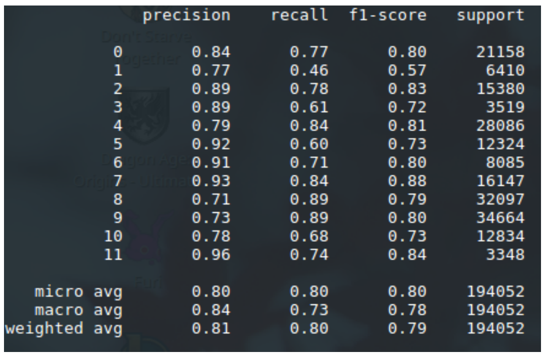


```
              precision    recall  f1-score   support

           0       0.87      0.77      0.82     15124
           1       0.88      0.59      0.70      3401
           2       0.91      0.76      0.83     31153
           3       0.86      0.89      0.87     15903
           4       0.67      0.91      0.77     33711
           5       0.80      0.68      0.73     11987
           6       0.74      0.52      0.61      6310
           7       0.73      0.68      0.70     12579
           8       0.86      0.76      0.80     20698
           9       0.83      0.82      0.83      7989
          10       0.91      0.81      0.85      3324
          11       0.77      0.85      0.81     27640

   micro avg       0.79      0.79      0.79    189819
   macro avg       0.82      0.75      0.78    189819
weighted avg       0.81      0.79      0.79    189819
```

FIGURE 8.38: Unit vector Word2Vec's F1 per category excluding News



```
              precision    recall  f1-score   support

           0       0.84      0.77      0.80     21158
           1       0.77      0.46      0.57      6410
           2       0.89      0.78      0.83     15380
           3       0.89      0.61      0.72      3519
           4       0.79      0.84      0.81     28086
           5       0.92      0.60      0.73     12324
           6       0.91      0.71      0.80      8085
           7       0.93      0.84      0.88     16147
           8       0.71      0.89      0.79     32097
           9       0.73      0.89      0.80     34664
          10       0.78      0.68      0.73     12834
          11       0.96      0.74      0.84      3348

   micro avg       0.80      0.80      0.80    194052
   macro avg       0.84      0.73      0.78    194052
weighted avg       0.81      0.80      0.79    194052
```

FIGURE 8.39: Unit vector LexVec's F1 per category excluding News

As we can see there are no major differences other than Glove doing slightly worse, probably due to the other models memorizing more the Dataset(since their F1 is roughly equal). We can also observe that the Reference category is working badly as we thought, but the Shopping category is getting somewhat well classified.

No News, References or Shopping Results

We once again tried to generate a NN for each model but this time removing as well the other two categories that we speculated had no specific lexic. We also printed the statistics for each individual category. These are the F1 results on training and validation:

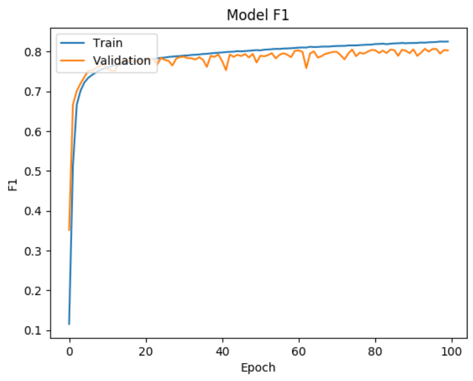


FIGURE 8.40: Unit vector Glove's F1 score over 100 epochs excluding News,Ref and Shop

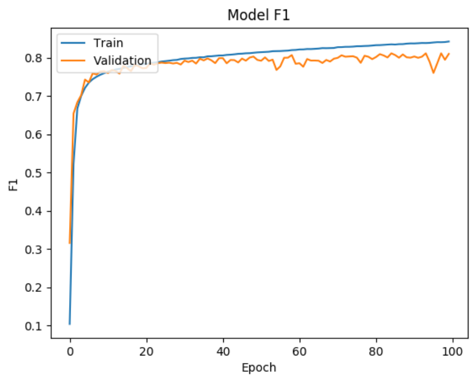


FIGURE 8.41: Unit vector Word2Vec's F1 score over 100 epochs excluding News,Ref and Shop

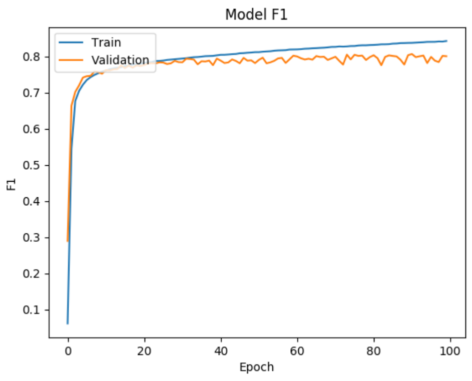
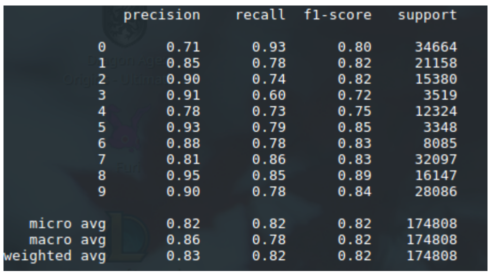


FIGURE 8.42: Unit vector LexVec's F1 score over 100 epochs excluding News,Ref and Shop

We can see that we are reaching 0.8 which is a great improvement. This is no surprise as we handpicked and eliminated the worst performing categories. We basically are making the problem our NN faces easier. Other than that, Glove seems to be slower on memorizing the training dataset.

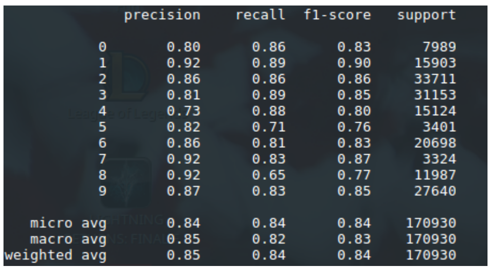


```
              precision    recall  f1-score   support

           0       0.71      0.93      0.80     34664
           1       0.85      0.78      0.82     21158
           2       0.90      0.74      0.82     15380
           3       0.91      0.60      0.72      3519
           4       0.78      0.73      0.75     12324
           5       0.93      0.79      0.85      3348
           6       0.88      0.78      0.83      8085
           7       0.81      0.86      0.83     32097
           8       0.95      0.85      0.89     16147
           9       0.90      0.78      0.84     28086

   micro avg       0.82      0.82      0.82    174808
   macro avg       0.86      0.78      0.82    174808
weighted avg       0.83      0.82      0.82    174808
```

FIGURE 8.43: Unit vector Glove's F1 score across categories excluding News,Ref and Shop

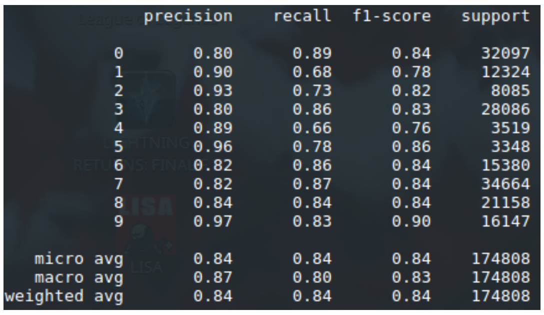


```
              precision    recall  f1-score   support

           0       0.80      0.86      0.83      7989
           1       0.92      0.89      0.90     15903
           2       0.86      0.86      0.86     33711
           3       0.81      0.89      0.85     31153
           4       0.73      0.88      0.80     15124
           5       0.82      0.71      0.76      3401
           6       0.86      0.81      0.83     20698
           7       0.92      0.83      0.87      3324
           8       0.92      0.65      0.77     11987
           9       0.87      0.83      0.85     27640

   micro avg       0.84      0.84      0.84    170930
   macro avg       0.85      0.82      0.83    170930
weighted avg       0.85      0.84      0.84    170930
```

FIGURE 8.44: Unit vector Word2Vec's F1 score across categories excluding News,Ref and Shop



```
              precision    recall  f1-score   support

           0       0.80      0.89      0.84     32097
           1       0.90      0.68      0.78     12324
           2       0.93      0.73      0.82      8085
           3       0.80      0.86      0.83     28086
           4       0.89      0.66      0.76      3519
           5       0.96      0.78      0.86      3348
           6       0.82      0.86      0.84     15380
           7       0.82      0.87      0.84     34664
           8       0.84      0.84      0.84     21158
           9       0.97      0.83      0.90     16147

   micro avg       0.84      0.84      0.84    174808
   macro avg       0.87      0.80      0.83    174808
weighted avg       0.84      0.84      0.84    174808
```

FIGURE 8.45: Unit vector LexVec's F1 score across categories excluding News,Ref and Shop

We can see that all categories in general get improved, compared to when we only removed News. Once again we are giving our model a much easier problem so it makes sense.

Chapter 9

Conclusions

9.1 About a broad crawl

We have seen that our Broad Crawl has been a very time consuming task, with many variables to take into account. Crawling a websites is nowhere as simple as it used to be and this forced us to take care of all those variables. Our results will be greatly altered depending on them as we have seen with accepting cookies, where a page may not show information at all if cookies are not accepted.

On the other hand, you have all of the other parameters that matter more to determine how efficient our crawl is rather than how we interact with these webpages. They have to be set empirically, depend of each architecture (network, computer itself,etc) and greatly impact our efficiency and efficacy. Not only that but they tend to intertwine and override the behaviour of others, so it takes a good amount of time to really learn how to work with them. They are easy to use but hard to master.

We must also take into account that even at this step we had to pay attention and modify the urls that were crawled; that is to ignore some of them. The reason being many of them belonged to the same website and subdomain and were the different articles. While this may seem not that bad and technically correct it would have run the risk of fooling our AI with that websites's signature text, specially when there were more than 5,000 websites of that same kind. An interesting conclusion is that is it very difficult to fully automatize a broad crawl.

As a side observation, looking at the results from our Broad Crawl we can see that the amount of websites per category greatly varies. The biggest surprise is how few News pages we were able to crawl. It would be interesting to compare them to some statistics on the raw dataset urls, but I am pretty sure they were not even the lowest category. It seems like News pages have been changing a lot their urls, they have bankrupted, they were not added commonly to the dataset or they are hard to find.

9.2 About word preprocessing

Word preprocessing can be an issue due to the many small decisions that can be made. The biggest problem is different pages may use different encodings, and being able to unify them can be a lot of work. Seeing some of the websites after obtaining only the raw words that composed them we saw a lot of separates *n*'s, probably endlines that were not properly translated from an encoding to another. The main reasons we dismissed redoing the encoding from zero were the fact that WE usually already take into account some of this weird mistranslations(and we speculate that give them very low values) or they do not appear in the WE at all and thus will not alter the document representation at all. This is still sure to cause some noise, but we hoped it would be negligible.

The other important choice when it comes to word embeddings is what do we consider a token. To elaborate, we want to consider a token whatever we would consider a word, the trick is that some words contain within them characters that are not alphabet characters. As an example we have words like *Word2Vec* and *sugar-free* that contain hyphens and numbers. We could also consider words that have apostrophes. In the end we decided to only consider as tokens any chain of numbers and words. We excluded hyphen-containing words for two reasons: they are not that common and the meaning they tend to encode is heavily related to the words composing them, so getting the words composing them separately is also good.

9.3 Neural Network results

We experimented quite a bit with different ways to normalize our original preprocessing, in hopes of giving an easier work to our NNs. We discovered that the small values in different dimensions mattered a lot, and that the difference in size between the vectors of various representations was significant (thus we could not normalize with a single value). We also noticed that making all the document vectors into the same length prevented overfitting, although the best results with them was more or less equally good to the one obtained using the base dataset without any alterations. We could also see that overall normalizing our data smoothed our training.

After that, we had some serious issues improving our NN, and no matter which parameters we changed we were unable to improve them. Upon further inspection we have seen that the News category gets almost no improvement no matter how much we vary our parameters. The obvious conclusion here is that it is caused by two reasons: lack of examples on the class and being a class intrinsecally hard to classificate. The combination of both proved devastating for this class and our algorithm was completely unable to generalize anything on it.

Knowing what the issue was, we attempted to remove the News class, which gave a small but noticeable improvement to how well our dataset classified. We were also able to notice that one of the classes we thought was going to be difficult to classify did not perform poorly, probably due to the fact our dataset had many instances of it. This gives the intuition that Reference pages could also get well classified, given enough data.

We then decided to try and remove all the categories that should give us difficulties when classifying due to being intrinsecally difficult for word embeddings: Reference, Shopping and News. The results showed a noticeable improvement, but this is probably due to removing Reference as Shopping had already a decent score.

After a lot of experimentation, the main conclusion is that using one of the WE over another does not give a Neural Network much improvement. That is, all the WE bring approximately the same information when used together with a Neural Network. This strikes me as surprising, as Glove does take into account some facts that Word2Vec did not, such as penalizing very common words. I was also expecting Word2Vec to perform slightly worse due to having been trained with less information and having a much more small-sized base file. While being more complex to update and doing much worse at word analogy tests, it seems equally capable to LexVec and Glove. It even used a much more limited dataset as a base for being generated, that was notably older.

In the end, they all seem to more or less be able to encode overall the same information. Contrarily to most of the tests I have seen, Word2Vec reached peak F1 at the same time as Glove or LexVec. The most surprising part is that Glove did not perform better than either, universally Glove tends to score higher in all analogy tests. We can deduce from this that if we want to be able to run this code into a somewhat limited machine Word2Vec is the way to go. The reason for this is that Word2Vec's pretrained model weights less than 4GB; we can load it into memory with any OS and we should still be well under 8GB threshold, thus working using only memory. Glove and LexVec's WE cannot do that, as they need 5.6 GB of memory at least, that altogether with an OS and a desktop can easily surpass 8GB. Thus Word2Vec is the smallest model that would allow us to classify a web in reasonable time.

If we wanted to build our classifier in an even more limited system we would have to switch to some of the other WE again, as they allow to be read line by line, contrarily to Word2Vec's model. It would be nowhere as fast as loading the WE to memory, but it would be able to run.

Finally, if we wanted adaptability and to be able to train our own model we should still use Glove or Lexvec above Word2Vec. The bad scaling of Word2Vec in big corpuses would penalize us greatly otherwise. Not only that, but being able to quickly update your WE is a really valuable skill. The most common for a successful business is to be able to bring more data as it grows. A WE that is capable of growing as your data does has a huge value.

Chapter 10

Technical Competences

10.1 CCO1

Having a deep knowledge of the fundamental principles and computational models and being capable to apply, interpret, select, evaluate, model and create new concepts, theories, uses and technological developments related to computing.

10.1.1 CCO1.1

Evaluate the computational complexity of a problem, knowing algorithmic strategies to solve it and recommend, develop and implement the one that gives the best performance according to the established requirements.

This had to be done while reading the documentation of the WE. It was important to understand the computational complexity of the algorithms and the problems that they attempted so solves, and understanding why some of them were more expensive to generate than others. We also had to deduce that if the representations could somewhat handle simple substraction while keeping some meaning, they would be able to hold the meaning as well when combining all the words in a document. Not only that but we had to interpret and evaluate various types of normalization that made us have a better understanding of how these representations worked.

This also had to be done while choosing how to improve the WE conversion, as we had to go through our data as quick as possible, being a matter of days.

10.2 CCO2

Effectively and efficiently develop adequate algorithms and software to solve computational complex problems.

10.2.1 CCO2.1

Show knowledge in the fundamentals, paradigms and techniques typical from intelligent systems, and analyze, design and build systems, services and computer apps that use these tools in any applicable kind of field.

While building our NN models, we had to use some specific layer kinds for our specific problem. Basically we used a softmax function for our last layer and relu functions for the rest, since we had a continuous input and wanted a discrete output. We also tried to address some of the issues that we saw in our problem (overfitting) and tried to correct them as best we could. We also used specific measurements to try and make our graphics the most meaningful.

10.2.2 CCO2.2

Capacity to obtain, formalize and represent human knowledge in a computable way in order to solve problems while using an information system in any applicable field, particularly in the ones related to computer science aspects, perception and response in intelligent environments.

Through the use of our WE we were able to effectively encode and combine all the words into a 300-dimensional vector. Given the results we have obtained from our NN, we were successful at capturing the overall topic in a websites and managing to quantify it.

10.2.3 CCO2.4

Show knowledge and develop computational learning techniques: design and implement applications and systems that use them, including the ones dedicated to automatically extract information and knowledge from big amounts of data.

Our NN needed to be designed and implemented, and we carefully picked the parameters we needed for them and their architecture. Since we wanted to also experiment with our designs varying the parameters that tend to alter the most a NN performance, and had to build them with small variations to see which was the best design

10.2.4 CCO2.5

Implement information retrieval software.

We had to implement a crawler capable of extracting websites with the most efficiency possible. We also had to be able to extract from these websites the maximum amount of information, and even retrieve some that was not present in the page. From that we had to be able to properly parse the text and manipulate it a little bit to fit our needs.

Bibliography

[1] *Http archive*. [Online]. Available: https://httparchive.org/.

[2] *Page weight*. [Online]. Available: https://httparchive.org/reports/page-weight.

[3] *The average web page is 3mb. how much should we care?* [Online]. Available: https://speedcurve.com/blog/web-performance-page-bloat/.

[4] *Headless browser*, Dec. 2018. [Online]. Available: https://en.wikipedia.org/wiki/Headless_browser.

[5] H. Kazawa, T. Izumitani, H. Taira, and E. Maeda, "Maximal margin labeling for multi-topic text categorization", in *Advances in neural information processing systems*, 2005, pp. 649–656.

[6] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen, "Web content categorization using link information", Stanford, Tech. Rep., 2006.

[7] G. Attardi, A. Gullì, and F. Sebastiani, "Automatic web page categorization by link and context analysis", in *Proceedings of THAI*, Citeseer, vol. 99, 1999, pp. 105–119.

[8] R Rajalakshmi and C. Aravindan, "Web page classification using n-gram based url features", Dec. 2013, pp. 15–21. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6921920.

[9] Y. Yang, X. Liu, *et al.*, "A re-examination of text categorization methods", in *Sigir*, vol. 99, 1999, p. 99.

[10] D. D. Lewis and M. Ringuette, "A comparison of two learning algorithms for text categorization", in *Third annual symposium on document analysis and information retrieval*, vol. 33, 1994, pp. 81–93.

[11] M. Garcia, H. Hidalgo, and E. Chavez, "Contextual entropy and text categorization", in *2006 Fourth Latin American Web Congress*, IEEE, 2006, pp. 147–153.

[12] E. Wiener, J. O. Pedersen, A. S. Weigend, *et al.*, "A neural network approach to topic spotting", in *Proceedings of SDAIR-95, 4th annual symposium on document analysis and information retrieval*, Las Vegas, NV, vol. 317, 1995, p. 332.

[13] M. Naili, A. H. Chaibi, and H. H. B. Ghezala, "Comparative study of word embedding methods in topic segmentation", *Procedia Computer Science*, vol. 112, pp. 340–349, 2017.

[14] *Crawler*. [Online]. Available: https://www.npmjs.com/package/crawler.

[15] *Scrapy | a fast and powerful scraping and web crawling framework*. [Online]. Available: https://scrapy.org/.

[16] dev@Nutch.apache.org, *Highly extensible, highly scalable web crawler*. [Online]. Available: http://nutch.apache.org/.

[17] [Online]. Available: https://docs.scrapy.org/en/latest/topics/settings.html.

[18] *Broad crawls¶*. [Online]. Available: https://docs.scrapy.org/en/latest/topics/broad-crawls.html.

[19] PhantomJS contributors. (2010-2018). Phantomjs - scriptable headless browser, [Online]. Available: http://phantomjs.org/.

[20] J. Camacho-Collados and M. T. Pilehvar, "On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis", *arXiv preprint arXiv:1707.01780*, 2017.

[21] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation", in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162.

[22] *The stanford nlp group*. [Online]. Available: https://nlp.stanford.edu/.

[23] T. Mikolov, K. Chen, G. S. Corrado, and J. A. Dean, *Computing numeric representations of words in a high-dimensional space*, US Patent 9,037,464, May 2013.

[24] A. Salle, M. Idiart, and A. Villavicencio, "Matrix factorization using window sampling and negative sampling for improved word representations", 2016.

[25] S. U.S. o. Engineering, *Lecture 3 | glove: Global vectors for word representation*, 2017. [Online]. Available: https://www.youtube.com/watch?v=ASn7ExxLZws.

[26] Pennington, Jeffrey. [Online]. Available: https://nlp.stanford.edu/projects/glove/.

[27] A. Salle, M. Idiart, and A. Villavicencio, "Matrix factorization using window sampling and negative sampling for improved word representations", *arXiv preprint arXiv:1606.00819*, 2016.

[28] Alexandres, *Alexandres/lexvec*, Oct. 2018. [Online]. Available: https://github.com/alexandres/lexvec#pre-trained-vectors.

[29] Y. Bengio *et al.*, "Learning deep architectures for ai", *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[30] F. Walls, H. Jin, S. Sista, and R. Schwartz, "Topic detection in broadcast news", in *Proceedings of the DARPA broadcast news workshop*, Morgan Kaufmann Publishers, Inc., 1999, pp. 193–198.

[31] J. Liu and T.-S. Chua, "Building semantic perceptron net for topic spotting", in *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, Association for Computational Linguistics, 2001, pp. 378–385.

[32] Z. Xu, S. Zhang, K.-K. R. Choo, L. Mei, X. Wei, X. Luo, C. Hu, and Y. Liu, "Hierarchy-cutting model based association semantic for analyzing domain topic on the web", *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1941–1950, 2017.

[33] J. Pang, F. Jia, C. Zhang, W. Zhang, Q. Huang, and B. Yin, "Unsupervised web topic detection using a ranked clustering-like pattern across similarity cascades", *IEEE Transactions on Multimedia*, vol. 17, no. 6, pp. 843–853, 2015.

[34] M. M. Lopez and J. Kalita, "Deep learning applied to nlp", *arXiv preprint arXiv:1703.03091*, 2017.

[35] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing", *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[36] AOL Inc., *Dmoz - rdf data*, 2016. [Online]. Available: `https://web.archive.org/web/20170303013959/http://dmoz.org/rdf.html` (visited on 11/15/2016).

[37] Curlie.org. (2018). Curlie - rdf data, [Online]. Available: `http://curlie.org/docs/en/rdf.html` (visited on 07/18/2018).

[38] *Datasets for single-label text categorization*. [Online]. Available: `http://ana.cachopo.org/datasets-for-single-label-text-categorization`.

[39] *Reuters-21578*. [Online]. Available: `http://www.daviddlewis.com/resources/testcollections/reuters21578/`.

[40] *20newsgroups*. [Online]. Available: `http://qwone.com/~jason/20Newsgroups/`.

[41] *Web data commons*. [Online]. Available: `http://webdatacommons.org/`.

[42] [Online]. Available: `http://commoncrawl.org/the-data/get-started/`.

[43] Google, *Google code archive - long-term storage for google code project hosting*. [Online]. Available: `https://code.google.com/archive/p/word2vec/`.

[44] *Build software better, together*. [Online]. Available: `https://github.com/`.

[45] *Selenium with python¶*. [Online]. Available: `https://selenium-python.readthedocs.io/`.

[46] L. Richardson, *Beautiful soup*. [Online]. Available: `https://www.crummy.com/software/BeautifulSoup/`.

[47] *Gensim: Topic modelling for humans*. [Online]. Available: `https://radimrehurek.com/gensim/`.

[48] *Natural language toolkit¶*. [Online]. Available: `https://www.nltk.org/`.

[49] *Keras: The python deep learning library*. [Online]. Available: `https://keras.io/`.

[50] L. A. Belanche-Muñoz. [Online]. Available: `https://www.cs.upc.edu/~belanche/`.

[51] *Upc*. [Online]. Available: `https://www.upc.edu/ca`.

[52] *Open graph protocol*. [Online]. Available: `http://ogp.me/`.

[53] *Rectifier (neural networks)*, 2019. [Online]. Available: `https://en.wikipedia.org/wiki/Rectifier_(neural_networks)`.

[54] *Softmax function*, 2019. [Online]. Available: `https://en.wikipedia.org/wiki/Softmax_function`.