

UNIVERSITY “POLITEHNICA” OF BUCHAREST
FACULTY OF ENGINEERING IN FOREIGN LANGUAGES
ELECTRONIC ENGINEERING, TELECOMMUNICATIONS AND
INFORMATION TECHNOLOGIES - APPLIED ELECTRONICS

DIPLOMA PROJECT

Project coordinator:

Conf. dr. ing. Bujor PĂVĂLOIU

Graduate:

Nicolae-Costin TANASE

Bucharest

2021

**UNIVERSITY “POLITEHNICA” OF BUCHAREST
FACULTY OF ENGINEERING IN FOREIGN LANGUAGES
ELECTRONIC ENGINEERING, TELECOMMUNICATIONS AND
INFORMATION TECHNOLOGIES - APPLIED ELECTRONICS**

Deep learning: Sentiment Analysis from text

Project coordinator:

Conf. dr. ing. Bujor PĂVĂLOIU

Graduate:

Nicolae-Costin TANASE

Bucharest

2021

**UNIVERSITY “POLITEHNICA” OF BUCHAREST
FACULTY OF ENGINEERING IN FOREIGN LANGUAGES
ELECTRONIC ENGINEERING, TELECOMMUNICATIONS AND
INFORMATION TECHNOLOGIES**

Approved

Director of department:

Prof. dr. ing. George DRAGOI

**DIPLOMA PROJECT THEME FOR:
Nicolae-Costin TANASE**

1. Theme title:

EN: *Deep Learning – Sentiment Analysis from Text*

RO: Învățare profundă – analiza sentimentelor din text

2. Initial design data:

The goal of this project is to research the functionalities and possible applications of natural language processing in internet text hotspots such as blogs, social media, review sites. For this purpose in this project there will be shown the potential of powerful python libraries in the realm of the sentiment analysis industry.

3. Student contribution:

- *search for relevant bibliography*
- *design application logic*
- *design user interface*
- *implement required software*

4. Compulsory graphical material:

Diagrams, learning curves

5. The paper is based on the knowledge obtained at the following study courses:

- *Neural Networks and genetic algorithms, Digital Signal Processing, Systems Engineering, Databases*

6. Paper development environment:

Python (Anaconda IDE)

7. The project serves as:

Research

8. Paper preparation date:

June 2021

Project coordinator

Conf. Bujor PĂVĂLOIU



Student:

Nicolae-Costin TANASE



Academic Honesty Statement

I, Nicolae-Costin TANASE, hereby declare that the work with the title “Deep Learning: Sentiment Analysis from text”, to be openly defended in front of the diploma theses examination commission at the Faculty of Engineering in Foreign Languages, University "Politehnica" of Bucharest, as partial requirement for obtaining the title of Engineer is the result of my own work, based on my work.

The thesis, simulations, experiments and measurements that are presented are made entirely by me under the guidance of the scientific adviser, without the implication of persons that are not cited by name and contribution in the Acknowledgements part.

The thesis has never been presented to a higher education institution or research board in the country or abroad.

All the information used, including the Internet, is obtained from sources that were cited and indicated in the notes and in the bibliography, according to ethical standards. I understand that plagiarism is an offense and is punishable under law.

The results from the simulations, experiments and measurements are genuine. I understand that the falsification of data and results constitutes fraud and is punished according to regulations.

Nicolae-Costin TANASE



23.06.2021

Table of Contents

1.Introduction	9
1.1 The basics	9
1.2 Motives	9
1.3 Advantages	9
1.4 Amazon as a study case	10
1.5 Amazon business model	12
1.6 Language and deep learning	12
1.7 Sentiment analysis as a solution	16
1.8 Customer feedback	16
 2.Presentation of the domain	 19
2.1 Neural Networks and Deep Learning	19
2.1.1 History of Deep Learning	20
2.1.2 Neural network classification	21
2.2 Advantages of deep learning	27
2.3 Limits and disadvantages of deep learning	28
 3.State of the art.....	 29
3.1 Needs and technologies	29
3.1.1 Essential text classifier features	30
3.2 Similar applications	31
3.2.1 Packaging.....	31
3.3 Relevant research in the domain of sentiment analysis for e-commerce.....	37
 4. Project development	 41
4.1 Introduction	41
4.2 The dataset	42
4.2.1 The fastText library	42
4.2.2 The content	44
4.3 Python for deep learning.....	45
4.4 The environment.....	45
4.5 Description.....	47
4.6 Data preprocessing.....	48
4.7 Deep learning models	50
4.8 Data visualization	60
 5. Future improvements	 66
6. Conclusions	69
7. References	71
8. Annexes.....	75

Table of Figures

Figure 1.1 Companies bought by Amazon by years.....	11
Figure 1.2 Pie chart of Amazon's revenue streams	12
Figure 1.3 English phonemes with Examples [4].....	13
Figure 1.4 Pie chart of Amazon's revenue streams.....	15
Figure 1.5 Pie chart representing customer opinions on retailer efficiency	17
Figure 1.6 Pie chart representing consumer opinion on intelligent product selection systems	17
Figure 2.1 A simple and a deep neural network.....	19
Figure 2.2 Frank Rosenblatt and his invention, the first perceptron	21
Figure 2.3 Deep Learning classification.....	22
Figure 2.4 Basic Convolutional Neural Network Architecture [14]	22
Figure 2.5 Example of a filter applied to a two-dimensional input to create a feature map [15]	23
Figure 2.6 Basic Recurrent Neural Network Architecture [16]	24
Figure 2.7 Inside a LSTM memory cell [17].....	24
Figure 2.8 Inside a GRU memory cell [18].....	25
Figure 2.9 Graphical representation of a SOM model [20].....	26
Figure 2.10 Parts of an autoencoder [22]	27
Figure 3.1 komprehend.io sentiment analysis API [23].....	32
Figure 3.2 Different types of APIs for different text classification needs from komprehend.io [23]	33
Figure 3.3 Power-interest grid of scikit-learn stakeholders [25].....	34
Figure 3.4 Diagram representing the keras sequential model topology	37
Figure 4.1 The word index dictionary – first 15 words.....	50
Figure 4.2 Layer structure of the 1st deep learning model.....	51
Figure 4.3 Training the model.....	53
Figure 4.4 Accuracy graph	53
Figure 4.5 Loss graph.....	54
Figure 4.6 Testing the accuracy	54
Figure 4.7 Layer structure of the RNN model.....	55
Figure 4.8 Training the RNN model	56
Figure 4.9 Accuracy graph	57
Figure 4.10 Loss graph.....	57
Figure 4.11 Evaluation of the RNN model.....	57
Figure 4.12 Layer structure of the CNN model.....	58
Figure 4.13 Training of the CNN model	58
Figure 4.14 Accuracy graph	59
Figure 4.15 Loss graph.....	59
Figure 4.16 Evaluation of the CNN model.....	59
Figure 4.17 Pie chart representing the rapport between positives and negative for the training dataset	60
Figure 4.18 Count plot representing the rapport between positives and negatives for the training dataset	61
Figure 4.19 Count plot representing the rapport between positives and negatives for the testing dataset.....	61
Figure 4.20 A frequency dictionary (shows the word and the number of times it appears for each sentiment category)	61

Figure 4.21 A word counter made using the Collections library	61
Figure 4.22 A sentiment dictionary, build from the frequency dictionary	61
Figure 4.23 Plotting the sentiment of word (reduced dataset)	62
Figure 4.24 Word frequency distribution in the reviews.....	62
Figure 4.25 Box plot showing the number of words for each sentiment category.....	63
Figure 4.26 CNN range of characteristic curve.....	63
Figure 4.27 RNN range of characteristic curve.....	64
Figure 4.28 Confusion matrix for the RNN model	64
Figure 4.29 Confusion matrix for the CNN model	65
Figure 5.1 Android text classification app using Tensorflow Lite.....	67
Figure 5.2 Simple web app for text classification using flask.....	68

List of acronyms

GRU – Gate Recurrent Unit

LSTM – Long-short Term Memory

RNN – Recurrent Neural Network

AI – Artificial intelligence

GPU – Graphical processing unit

CNN- Convolutional Neural Network

ANN – Artificial Neural Network

BiGRU – BiDirectional GRU

BiLSTM – BiDirectional LSTM

TPU – Tensor processing unit

1. INTRODUCTION

1.1-The basics

Deep learning is a family of machine learning algorithms where the learning happens through different kinds of multilayered neural network architectures. Over the past few years, it has shown remarkable improvements on standard machine learning tasks, such as image classification, speech recognition, and machine translation. This has resulted in widespread interest in using deep learning for various tasks, including text classification.

Two of the most commonly used neural network architectures for text classification are convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

Each of the aforementioned model types, along with a basic multilayer perceptron, have been implemented as solutions for classifying the sentiment in the Amazon Reviews dataset from Kaggle. In the interest of practicality, the dataset has a few million reviews that have been scraped from the internet, thus being a business oriented dataset rather than a dataset for learning. The data comes in string format, containing both the review and the corresponding label. The data is then preprocessed in different ways in order to be fed to the model for training. Once the models are trained they can be evaluated and ranked based on their accuracy. The models can then be deployed to predict future reviews.

1.2-The motives

I have chosen this particular project because, even though deep learning is not yet the silver bullet for natural language processing, it is still good enough to be implemented in retail and commerce, and it's still a young science with untapped potential, the only thing holding it back now is computational power, which will continue to make deep learning more and more useful for different industry tasks, especially with new emerging technologies such as quantum computing. I have picked the Amazon dataset because it is one of the biggest multi-industry companies, and because, compared to product-oriented companies like Apple, Tesla, Microsoft and others, which have a definitive fanbase that buy their products and usually have at worst mixed reviews, Amazon has a more diverse sentiment pool to pick from.

1.3-Advantages

This kind of models can aid businesses with their text classifying needs. They can cut down costs on business analysis, can increase performance because of the around-the-clock advantage that software in general delivers and with the proper maintenance they are more easily adaptable and can work with other technologies to predict trends at a pace human ability cannot achieve. They can receive noisy or missing data, work with large amounts of variables, and can be programmed to self-improve. Also, a human can look at data through different preprocessing techniques, in natural language processing it's more like turning the volume up and down rather than finding a right answer: there is data that you want to get rid of, data that you want to keep based on the language you are interested in and the words that are deemed uninteresting in the company's line of business. Also, the visualization of data is important so that the human analyst can get a better feel of the data the model is being fed, to know if it is balanced or not.

1.4-Amazon as a study case

Amazon is nowadays the largest e-commerce platform, because it acts more as a delivery and marketing system rather than a store. It is sometimes referred to as "The Everything store", and its sheer size combined with the small producers creating directly tailored to customer needs is what got it to 49% of the US online sales.

The main protagonist to this whole story is the famous Jeff Bezos. After he graduated university, he was in and out of different jobs, more or less related to tech. The prequel of the Amazon story started when he got a job at the hedge fund D.E Shaw., where he was recognized as a valuable employee and promoted at a fast pace, becoming Vice President in just four years. At that level, his job consisted of searching the internet for business opportunities, a time where the internet was still this shady, untrusted-by-many technology. He identified at the time 20 items that had the potential of being sold on the internet and presented the opportunity to his fellow staff members. He was ridiculed for daring to think that the internet is something people took seriously, and, since he held great conviction that the internet had at least a part to play in the future of commerce, he decided to try it out on his own.

"I knew that I might sincerely regret not having participated in this thing called the internet that I thought was going to be a revolutionizing event. When I thought about it that way... it was incredibly easy to make the decision." – Jeff Bezos

The first item he decided to list on his newly developed and deployed platform were books. He chose them because of their relatively low price, because of the sheer diversity and the

universal demand.

How did he manage to switch the narrative of shopping more to the online realm?
Through focusing on the convenience aspect that the internet had the capacity to provide.

When we refer to Amazon, we still think of this huge online store with tons of different products from suppliers all over the globe, but Amazon has acquired some very important names in the business realm while on the ascension.

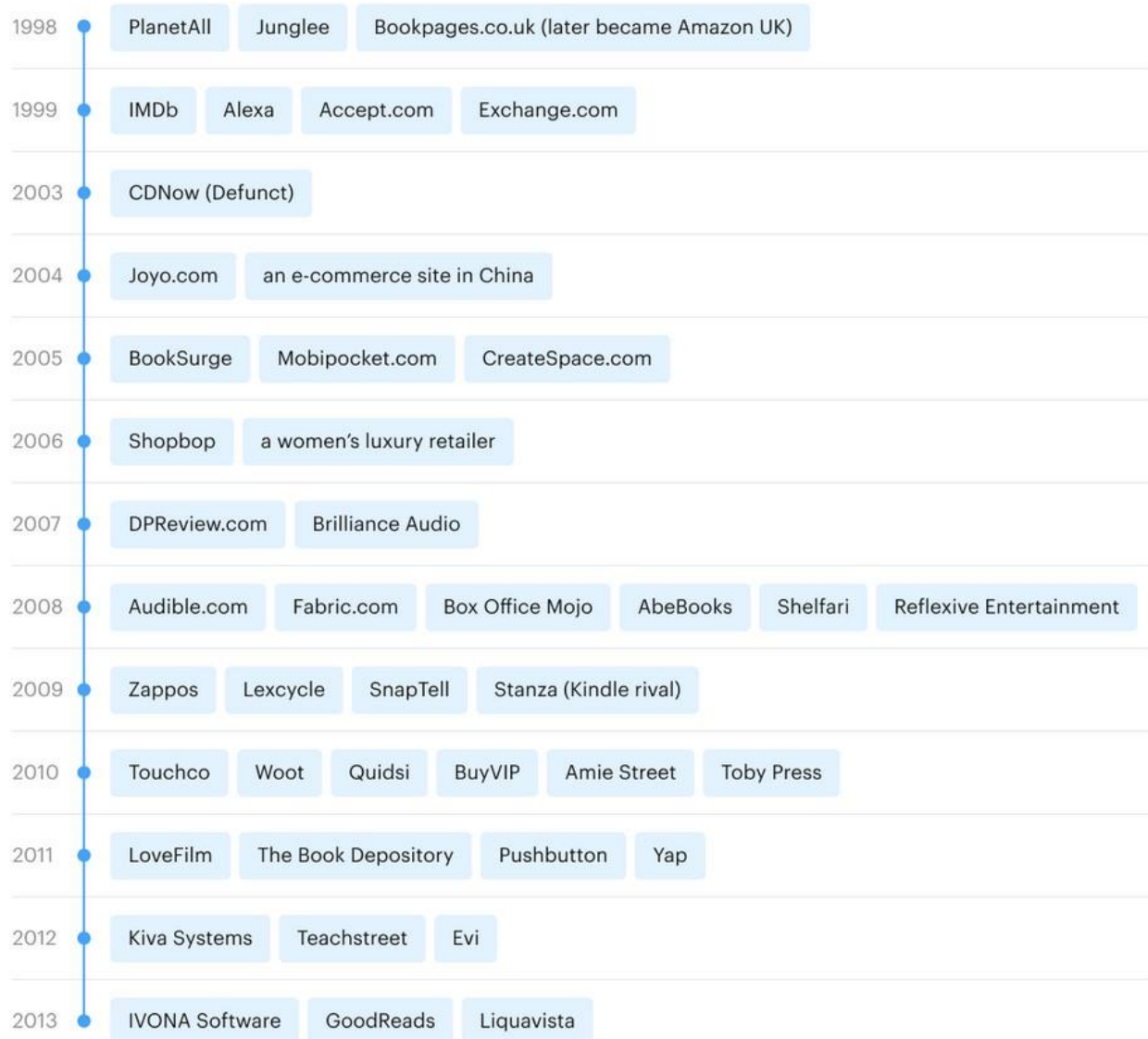


Figure 1.1 Companies bought by Amazon by years

There are so many products that many of us use that we never think who actually owns the companies. Amazon has become one of the biggest “providers of everything” and have shown great interest in adapting and keeping this giant colossus of a company still in touch with the consumers at the lowest level [1] [2].

1.5-Amazon business model

Amazon's business model, although very diversified (advertising, cloud, subscription services, AWS, physical stores), still has at its core its creation, the e-commerce platform.

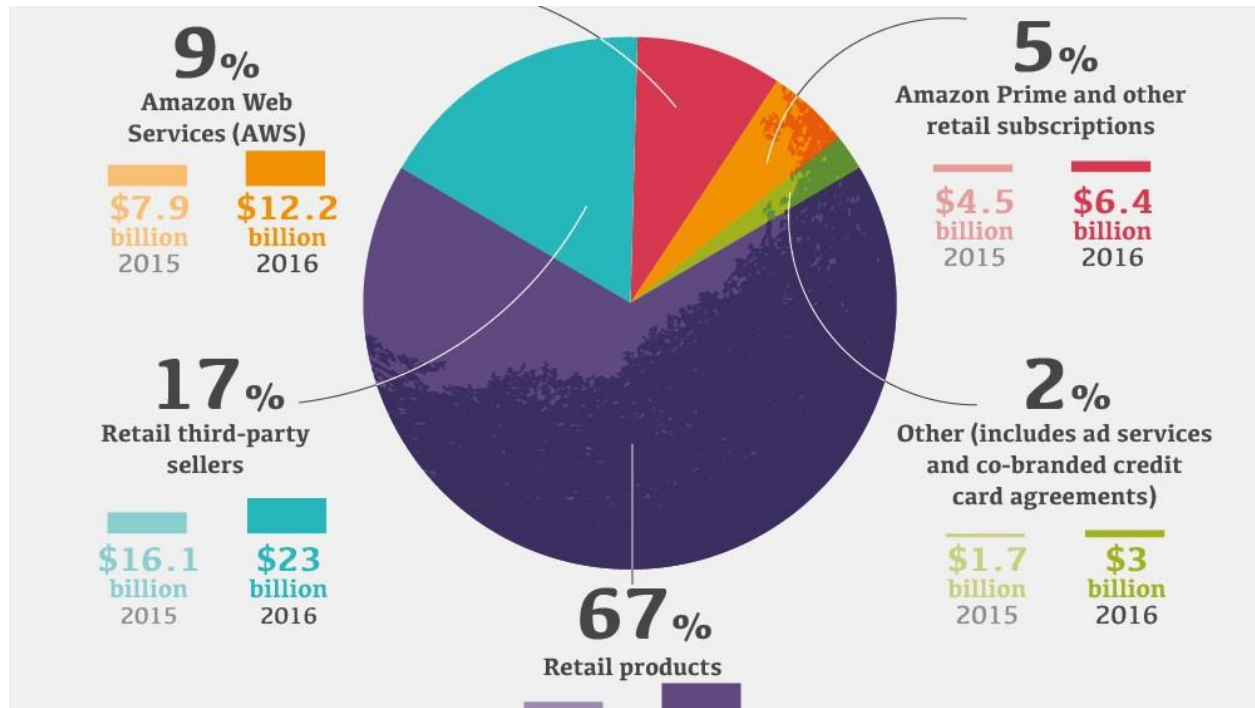


Figure 1.2 Pie chart of Amazon's revenue streams

Retail products still remain their core driver of profit, allowing them to pursue a wide array of other niches and compete against giants like Netflix, Walmart, Google, etc. without implying a great amount of risk from their part [3].

1.6-Language and deep learning

Language is a structured system of communication that consists of complex combinations of its subtypes (words, sentences, characters, etc). In order to understand better the needs deep learning has when it comes to data amounts and quality, we must first understand some concepts of linguistics and how language works.

According to the reference [4] we can think of human language as composed of four major building blocks: phonemes, morphemes and lexemes, syntax, and context. When building a deep learning model for text classification, it needs knowledge of different levels of this building blocks, starting from the phonemes (the sounds of the language) to context (meaningful

expression of a text). Let me now examine each language building blocks:

- Phonemes

Phonemes are the smallest units of sound in a language. They may not have any meaning by themselves but can induce meanings when uttered in combination with other phonemes. For example, standard English has 44 phonemes, which are either single letters or a combination of letters. In Romania, phonemes are even more important, since, compared to English, we possess grammatical gender in our language, which for us makes phonemes all the more important, since, at least in most of the feminine gendered nouns, the noun changes its inner spelling when converted to plural. Moreover, the number of phonemes the Romanian language possesses has not yet been clearly determined by any relevant literature authority: it is simply too hard for the Romanian literature experts to settle on a number given the complexity of the language. In English, which is the language with which this paper is mainly concerned, has its phonemes split between vowels and consonants.

Consonant phonemes, with sample words		Vowel phonemes, with sample words	
1. /b/ - bat	13. /s/ - sun	1. /a/ - ant	13. /oi/ - coin
2. /k/ - cat	14. /t/ - tap	2. /e/ - egg	14. /ar/ - farm
3. /d/ - dog	15. /v/ - van	3. /i/ - in	15. /or/ - for
4. /f/ - fan	16. /w/ - wig	4. /o/ - on	16. /ur/ - hurt
5. /g/ - go	17. /y/ - yes	5. /u/ - up	17. /air/ - fair
6. /h/ - hen	18. /z/ - zip	6. /ai/ - rain	18. /ear/ - dear
7. /j/ - jet	19. /sh/ - shop	7. /ee/ - feet	19. /ure/ ⁴ - sure
8. /l/ - leg	20. /ch/ - chip	8. /igh/ - night	20. /ə/ - corner (the 'schwa' - an unstressed vowel sound which is close to /u/)
9. /m/ - map	21. /th/ - thin	9. /oa/ - boat	
10. /n/ - net	22. /th/ - then	10. /oo/ - boot	
11. /p/ - pen	23. /ng/ - ring	11. /oo/ - look	
12. /r/ - rat	24. /zh/ ³ - vision	12. /ow/ - cow	

Figure 1.3 English phonemes with Examples [4]

- Morphemes and Lexemes

A morpheme is the smallest unit in a language that has actual meaning. It is made out of one or more phonemes. In order for a sequence of characters to classify as a morpheme, it must be a word or a part of a word, cannot be divided any further into smaller segments with meaning and, no matter the context, it holds a stable meaning.

Morphemes can be categorized into two classes: bases (or roots) and affixes (prefixes or suffixes). A "base" is a morpheme in a word that gives the word its principle meaning. The base morphemes can be further subcategorized into free-base morphemes or bound-base morphemes. For example, in the word gladly, glad is the free base morpheme, but in the word "dissent", "-sent" is a bound-base morpheme.

The affixes are what we intuitively think of as morphemes. As previously mentioned, they are either prefixes ("**un**healthy", "**ante**date", "**pre**historic", "**dish**onest") or suffixes ("social**ism**", "bake**r**", "fond**ness**", "spiritual**ism**").

Lexemes are the structural variations of morphemes related to one another by meaning. For example, "run" and "running" belong to the same lexeme form.

Morphological analysis, which analyzes the structure of words by studying its morphemes and lexemes, is a foundational block for many text classification tasks, such as tokenization, stemming, learning word embeddings, and part-of-speech tagging [4] [5] [6].

- Syntax

Syntax represents the rules for constructing grammatically correct sentences out of words in a language. Syntactic structure in linguistics is represented in many different ways. A common approach to representing sentences is a parse tree.

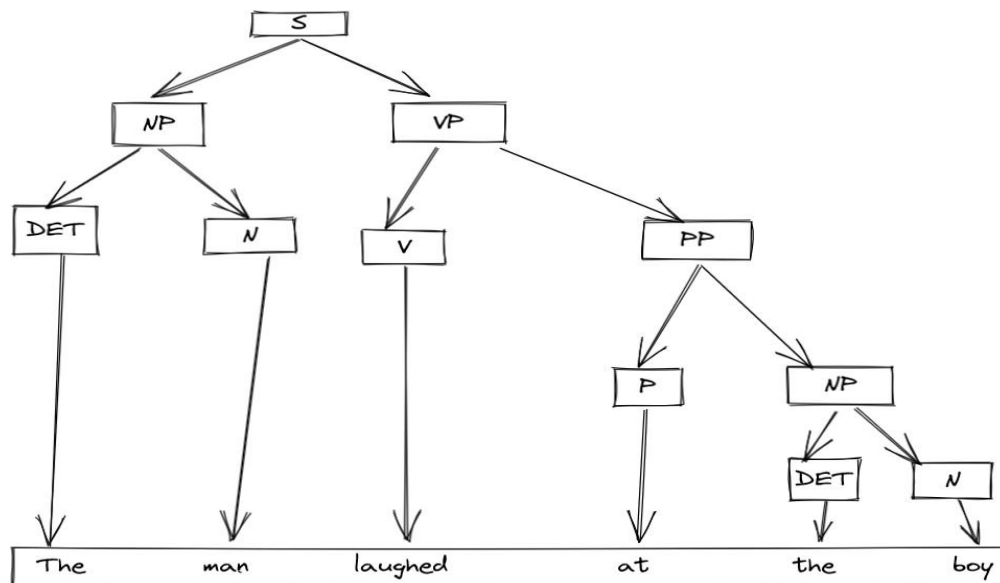


Figure 1.4 Pie chart of Amazon's revenue streams

This is how you represent a sentence as a parse tree. S stands for sentence, NP stands for noun phrase, VP stands for verb phrase, N for noun, V for verb, PP for preposition phrase, P for preposition and DET for determiner [7] [4].

An important note: this kind of syntax described above is only specific to the English language. Other languages require a modified approach, but the tools necessary for processing already support many of the languages around the world, especially the ones that are spoken internationally.

- Context

Context shows how various pieces of the language come together to convey meaning. Context includes references, word knowledge, and common sense, something AI sometimes struggles with, along with the literal meaning of the words used. The meaning of the sentence can be entirely switched just because of context, and can sometimes even have multiple meanings. Generally, context is composed from semantics and pragmatics. Semantics is the direct meaning of the words and sentences without external context. Pragmatics adds world knowledge and external context of the conversation to enable us to infer implied meaning.

Complex text classification models used for tasks such as sarcasm detection, summarization, and sentiment analysis are some of tasks that use context heavily [4].

As a conclusion, language in the reality of text classification is, maybe in some instances, even as important as the tools use to process it. Since deep learning models struggle with the complexity of human speech and text, you as a deep learning enthusiast or professional need to have a specific task in mind when designing the model: if it performs well on sentiment analysis you have no guarantees it will perform well on sarcasm or emotion detection.

1.7-Sentiment analysis as a solution

Every business spends a considerable amount of money in analysis of consumer opinion and preference. Sentiment analysis, or opinion mining, as it is sometimes referred to, can not only help enhance the products diversity and customer satisfaction, but it also has the massive advantage of contributing to stock market predictions by prediction of market trends, which can help common people make wise investments and become shareholders of successful businesses. Costs of advertising and business analysis will go down significantly, so businesses can now focus more on the creative and production parts.

Usually, e-commerce sites contain huge amounts of data that can sometimes be vague and distorted. The average human reading speed is quite small, and their ability to summarize the information and opinions is quite limited. But humans have other advantages that deep learning and AI in general still struggles with, such as sarcasm detection or frequent incorrect spelling.

1.8-Customer feedback

To check if people are satisfied with the current state of the e-commerce stores, I have performed a modest survey which was taken by people of various ages, with different backgrounds and different expectations, a group consisting of relatives, friends and classmates. Although people generally considered big e-commerce stores own and utilize some kind of powerful software in order to assess the reviews and change their product range accordingly, they also felt that their needs are not taken into account fast enough.

For the question: “Do you ever feel that the reviews you offer on the items you bought have no impact and do not shape the retailers' range of products fast enough?”, the answers were the following:

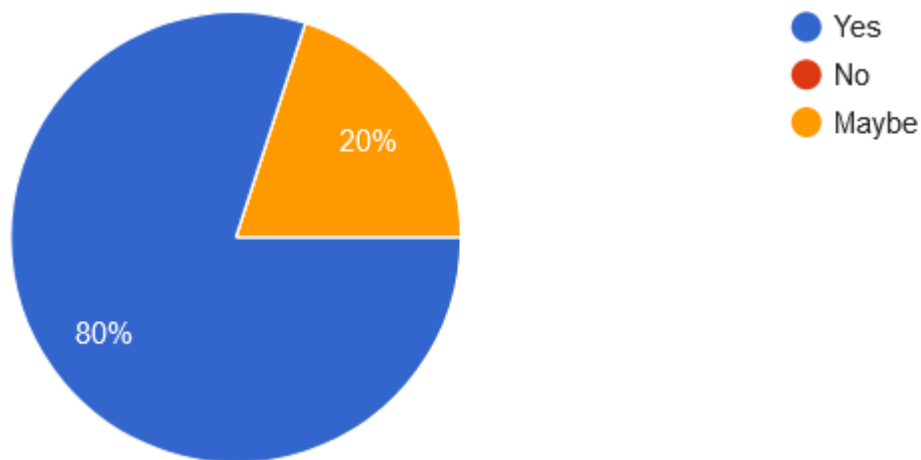


Figure 1.5 Pie chart representing customer opinions on retailer efficiency

These percentages show that there is a need for intelligent decisions when it comes to software, since many feel the change created by customer feedback comes at a slow and agonizing pace.

For the question: “Do you think most of the big retailers have an intelligent product selection system in place based on customer satisfaction and needs?” the answers were the following:

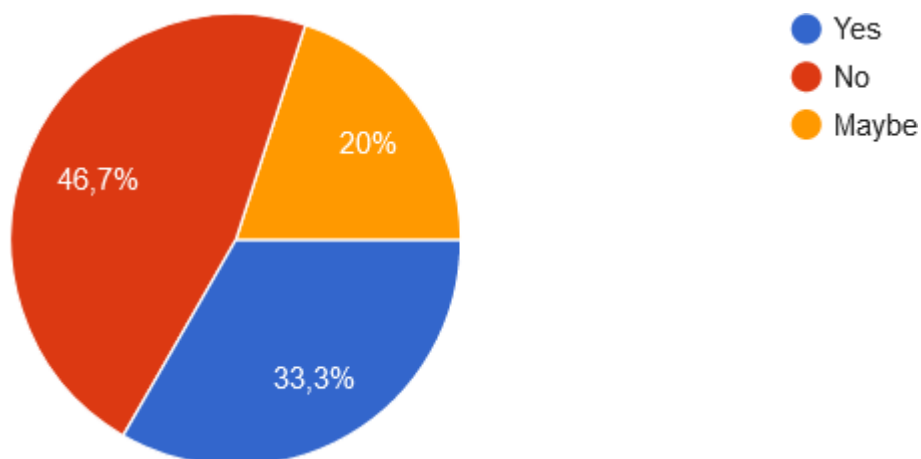


Figure 1.6 Pie chart representing consumer opinion on intelligent product selection systems

More of the people that were unsatisfied before, at this point show that they think or they might be inclined to believe that their reviews are being taken as input by the big e-stores and processed into decision making data, but the real world results beg to differ. The technology exists, it continues to improve, but the deployment of such technologies has just recently got the attention it truly deserves. As for the smaller e-stores, there is still the preconception of the sheer difficulty of putting such software together and the belief that deep learning models are a thing better left to the big sharks of the industry, which can afford to lose capital in the pursuit of this kinds of software.

2. PRESENTATION OF THE DOMAIN

2.1-Neural Networks and deep learning

Neural networks are a small part of AI. As they currently exist, neural networks carry out miniscule, highly specific tasks. Unlike the human brain, computer-based neural networks are not general-purpose computational devices. Furthermore, the term neural network can create confusion because the brain is a network of neurons just as AI uses neural networks. To avoid this problem, we must make an important distinction.

Programmers design neural networks to execute one small task. A full application will likely use neural networks to accomplish certain segments of a whole. However, the entire application will not be implemented as a neural network. It may consist of several neural networks of which each has a specific task.

Pattern recognition is a task that neural networks can easily accomplish. For this task, you can communicate a pattern to a neural network, and it communicates a pattern back to you. At the highest level, a typical neural network can perform only this function. Although some network architectures might achieve more, the vast majority of neural networks work this way [8].

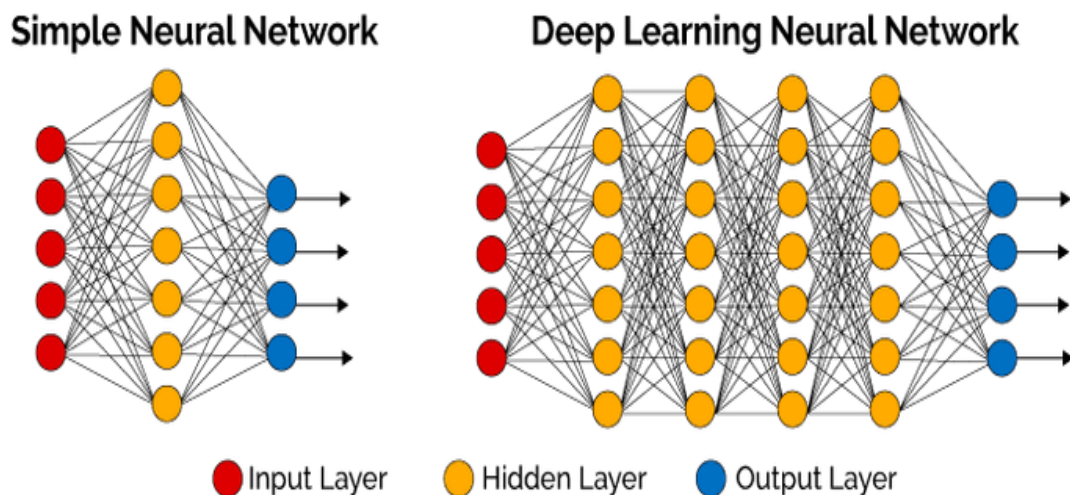


Figure 2.1 A simple and a deep neural network

As you can see, the above neural network accepts a pattern and returns a pattern. Neural networks operate synchronously and will only output when it has input. This behavior is not like that of a human brain, which does not operate synchronously. The human brain responds to input, but an output is not mandatory, or at least does not emerge necessarily in a predictable manner [9].

2.1.1-History of deep learning

The world right now is seeing a global AI revolution across all industry. There was thought that artificial intelligence will find its place only in commerce, both online and physical, in marketing and other industries with higher profit margins. But now we are witnesses to the first steps of the fourth industrial revolution, where different AI models are built and deployed in domains like aviation, medicine, domains in which was considered to be impossible to replace a significant amount of the human labor, or where precision was of the essence. And one of the driving factors of this AI revolution is Deep Learning. Thanks to giants like Google and Facebook, Deep Learning has now become a popular term and people might think that it is a recent discovery, but its history dates back to the 1940s.

Indeed, deep learning has not appeared overnight, rather it has evolved slowly and gradually over seven decades. And behind this evolution, there are many machine learning researchers who worked with great determination even when no one believed that neural networks have any future.

Neural networks have risen from the ashes of discredit several times in their history. Warren McCulloch and Walter Pitts (1943) first introduced the idea of a neural network. However, they had no method to train these neural networks. Programmers had to craft by hand the weight matrices of these early networks. Because this process was tedious, neural networks were not pursued any further.

Frank Rosenblatt (1958) provided a much-needed training algorithm called backpropagation, which automatically creates the weight matrices of neural networks. In fact, backpropagation has many layers of neurons that simulate the architecture of animal brains. However, backpropagation is slow, and, as the layers increase, it continues to get slower. It appeared as if the addition of computational power in the late 20th century helped neural

networks perform tasks, but the hardware and training algorithms of this era could not effectively train neural networks with many layers, and again neural networks faded as an academic or industry interest.

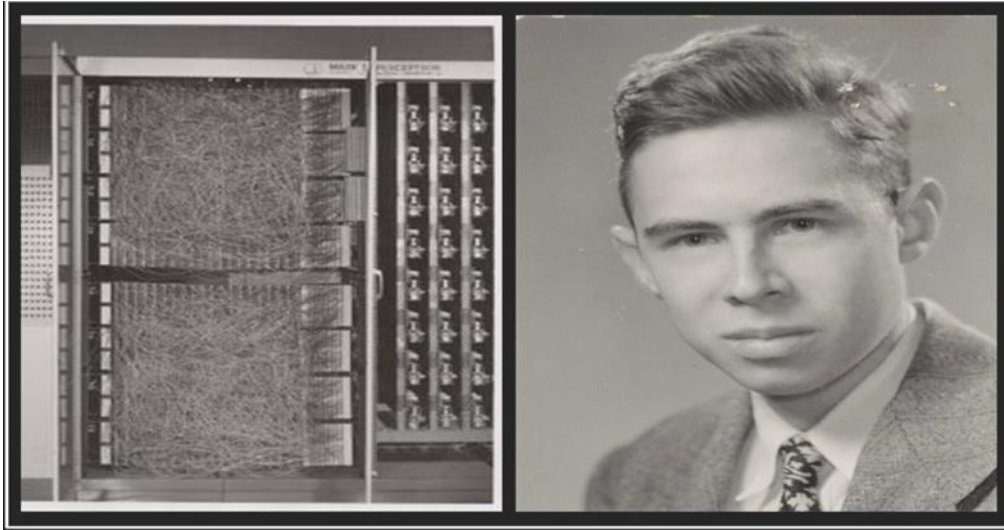


Figure 2.2 Frank Rosenblatt and his invention, the first perceptron

The third rise of neural networks occurred when Geoffrey Hinton, a cognitive psychologist and computer scientist who currently works for Google provided a radical new way to train deep neural networks without a human teacher. The recent advances in high-speed graphics processing units (GPU) allowed programmers to train neural networks with three or more layers and led to a resurgence in this technology as programmers realized the benefits of deep neural networks [10] [11] [12] [13].

2.1.2-Neural Network Classification

Connectionist architectures have been around for 7 decades, but without the GPU power to back them up they were not feasible. Also new architectures, along with the rise of GPU power, contributed to the increased interest in these kinds of solutions. Deep learning is not a one size fits all solution, but a group of algorithms and topologies that can be applied depending on the situation. The Artificial Neural Network (ANN) is the underlying architecture that holds the foundation for the other, more specialized approaches.

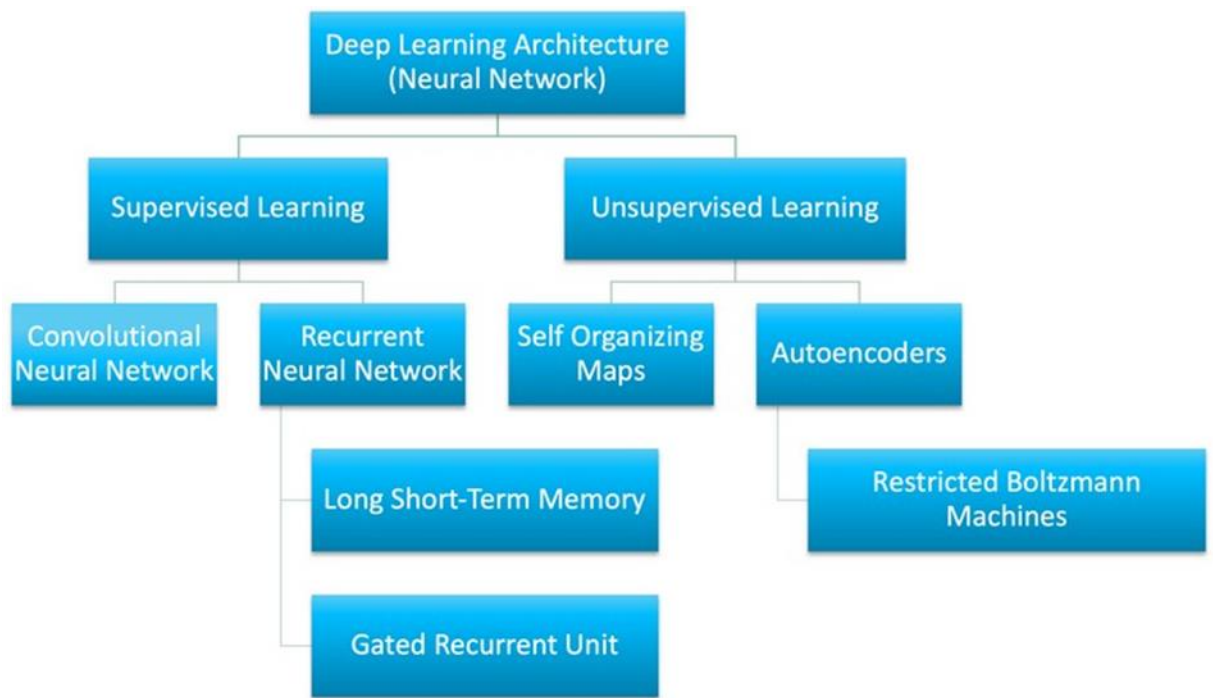


Figure 2.3 Deep Learning classification

A. Supervised deep learning

1. Convolutional Neural Networks

This architecture is considered to have as a standard use computer vision applications. The first CNN was created by Yann LeCun, and at its time, the task which was supposed to solve was the recognition of handwriting.

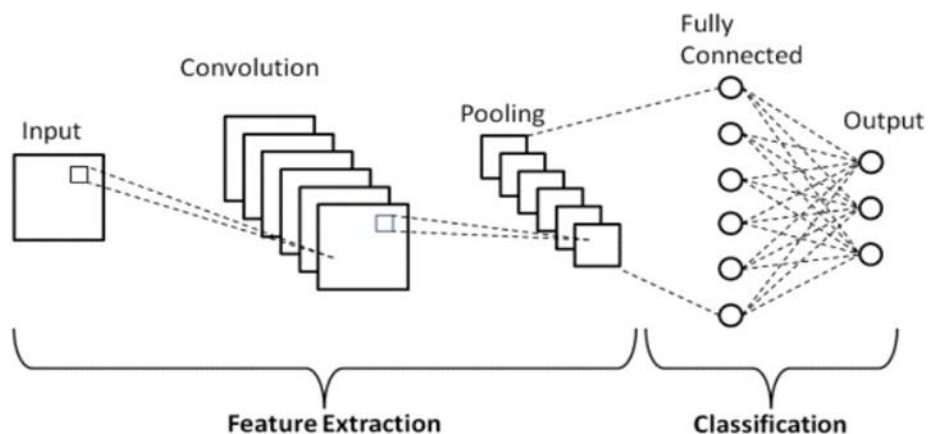


Figure 2.4 Basic Convolutional Neural Network Architecture [14]

The convolution is an operation that is done with the purpose of filtering, where weights are multiplied with the inputs which are, as a standard, pixels. The weights usually come in the

form of a smaller matrix, called a filter.

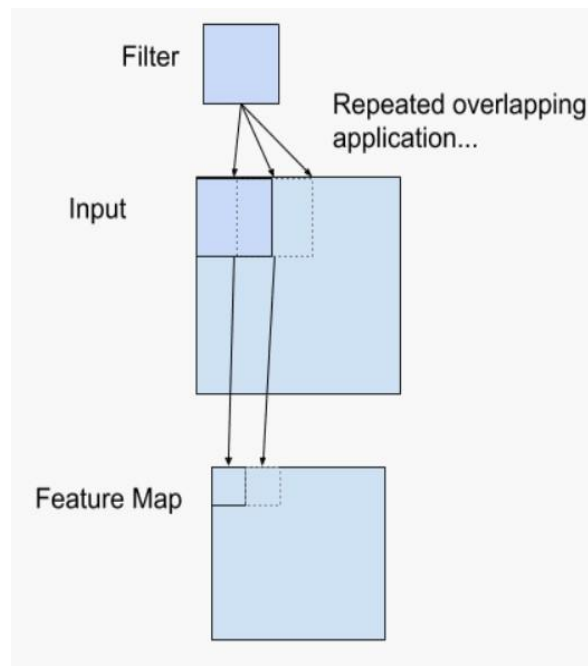


Figure 2.5 Example of a filter applied to a two-dimensional input to create a feature map [15]

The output of the operation is called a feature map because the filter, depending on its type, is designed to search for particular patterns.

A convolutional layer can be applied to both raw input, or the output of other layers, which is usually the case. Applied systematically on more and more layers can isolate features more clearly and can extract things such as cars, faces, types of pets and so on.

The pooling layer is used to reduce the size of the feature map, while maintaining the features extracted by the convolution. Pooling, in general, can be either done by taking the maximum or the average out of a cluster of pixels.

Instead of flattening the data and sending it as an output, a fully connected layer is used, which is the part that actually “learns” and helps deal with the non linearity of the task performed.

2. Recurrent Neural Networks

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data. These deep learning architecture is commonly employed in working with text rather than images. Examples of technologies that use this kind of neural network are Siri, Google Translate and voice searching.

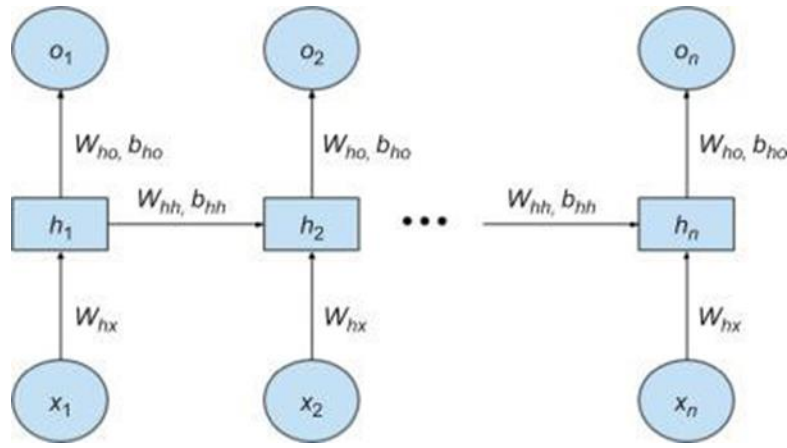


Figure 2.6 Basic Recurrent Neural Network Architecture [16]

The difference between a feed-forward basic neural network and using a recurrent neural network is that a RNN can feed back into previous layers or into the same layer. The idea behind RNNs is that they examine individual elements of the sequence once and retain memory of it so they can use it for the next element. There are two common types of RNNs that are used depending on the dimensions of the dataset used, the processing power and the requirements of the task performed:

1. LSTM (Long-Short Term Memory)

The LSTM was created in 1997 by Hochreiter and Schmidhuber, but it has since become popular for applications that work with large sets of data. The LSTM is where the memory cell become a concept. The long-short term memory can either ignore, forget or add preexisting data to the current action, this being a big aid in tasks like text suggestions, where the suggestion has to take into account entire sequences of words rather than the last ones, in other words, if working properly it can retain what's important and drop what's not.

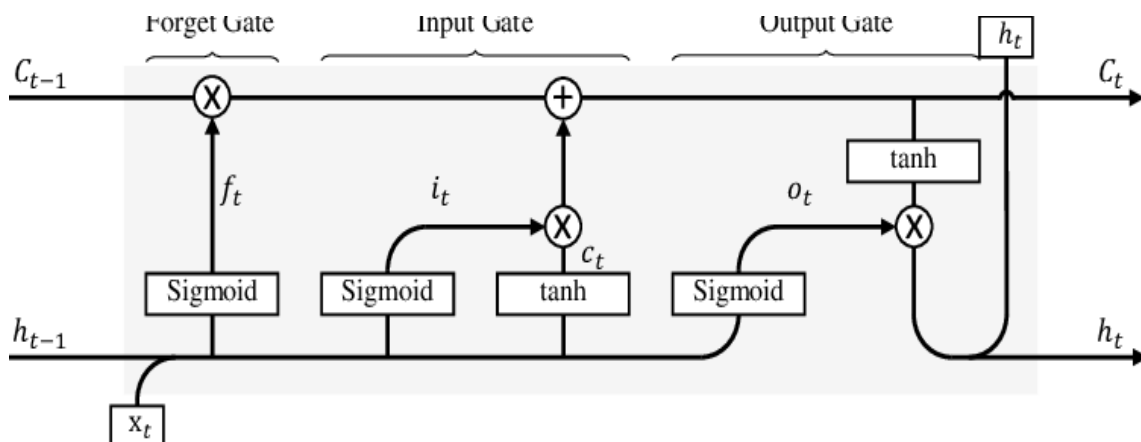


Figure 2.7 Inside a LSTM memory cell [17]

As it can be observed in the previous image, the basic topology of the memory cell has three gates: forget, input and output. They all work using either a sigmoid or a hyperbolic tangent activation, which are threshold functions that block or allow information.

2. GRU (Gate Recurrent Units)

GRU is a simplified version of the LSTM that was introduced in 2014. This model has only two gates with no output gate, the only gates being reset and update. The update gate is used in dosing how much of the previous cell to retain. The reset gate relates the new input to the previous cells. A simple RNN can be a GRU that has the reset gate set to 1 and the update gate set to 0.

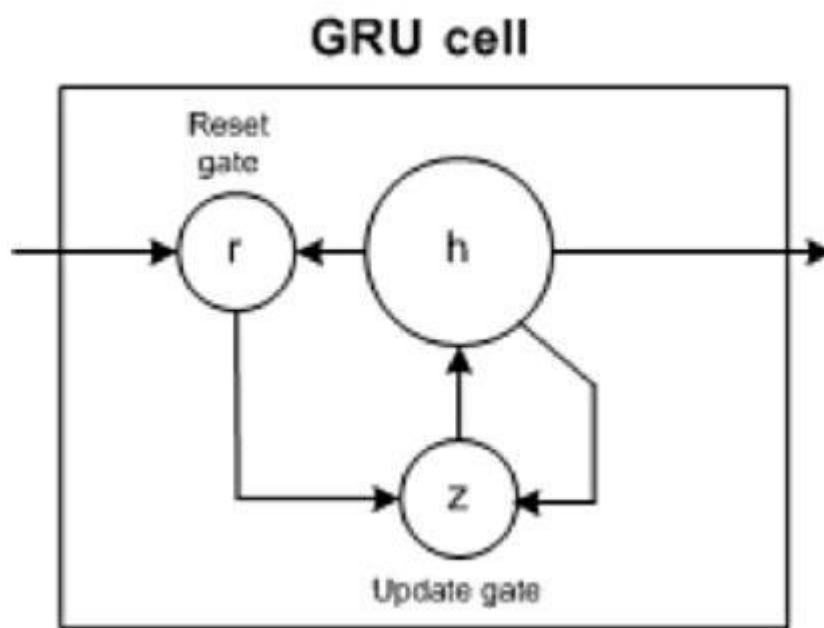


Figure 2.8 Inside a GRU memory cell [18]

Depending on the task you are trying to perform is important to pick the right kind of cell. For large and diverse datasets, when coupled with a corresponding GPU power can yield better results than the GRU, but the GRU is easier to train and requires less resources and time [19].

B. Unsupervised deep learning

Unsupervised learning refers to learning without the labels, which, in supervised learning, help with minimizing the loss. There are two important subcategories: Self Organizing Maps and Autoencoders.

1. Self Organizing Maps

Self-organizing maps are usually used in the reduction of dimensions of data, to turn a data sample of multiple dimension into only two dimensions, which is then considered a discrete map. “Self-organizing maps differ from other artificial neural networks as they apply competitive learning as opposed to error-correction learning (such as backpropagation with gradient descent), and in the sense that they use a neighborhood function to preserve the topological properties of the input space. **SOM** was introduced by Finnish professor Teuvo Kohonen in the 1980s is sometimes called a **Kohonen map**. “(Source – medium.com)

Basically, the data is “fighting” to be represented, and once a reference vector is chosen, the one that is most alike “wins” a modification of the weights, this algorithm being called the **Best Matching Unit**. This process repeats for the best matching unit node, so in the grand scheme of things, the Self Organizing Map aims to draw “alike-ness” on a 2D space.

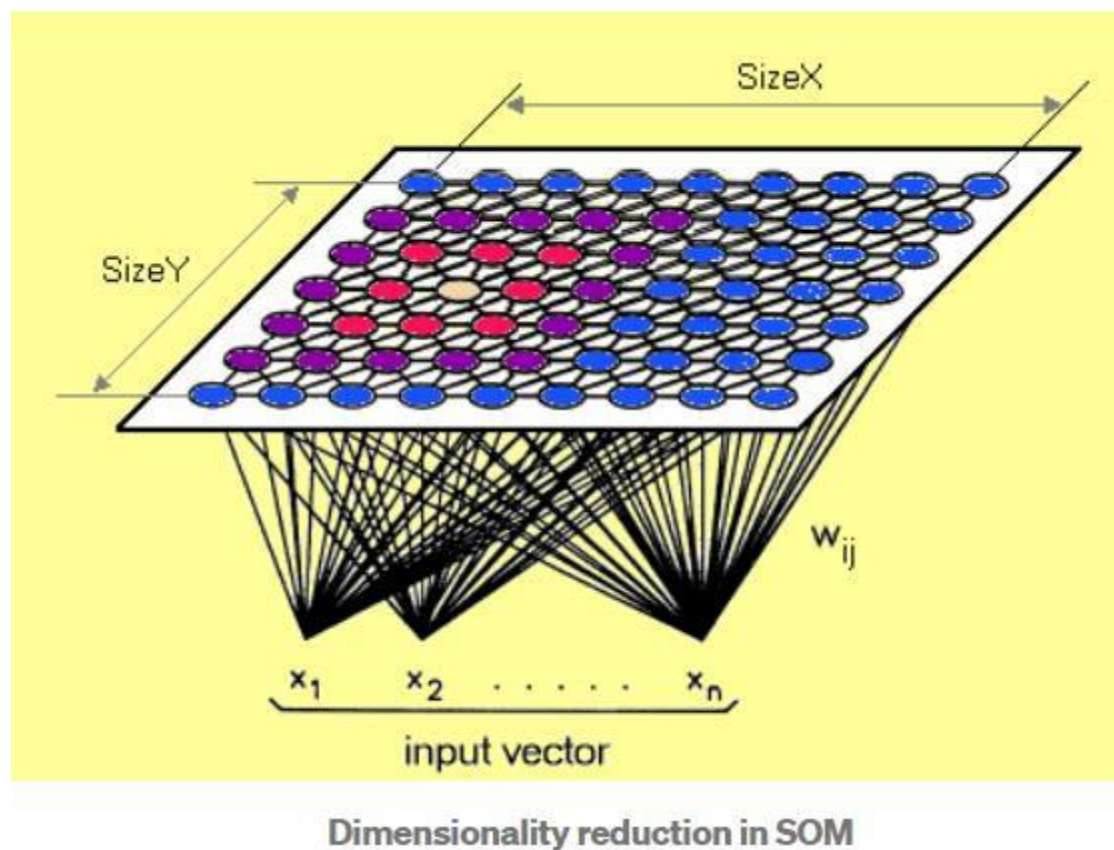


Figure 2.9 Graphical representation of a SOM model [20]

2. Autoencoders

In their essence, autoencoders are trained to create a set of features (encoding), that can then be transformed back to their original form (decoding). They share some similarities with their supervised learning counterparts (an autoencoder, if used for images, can have, for example, convolutional layers. If used for text, the text must be converted to sequences or vectors, which is not uncommon in supervised learning either) [21].

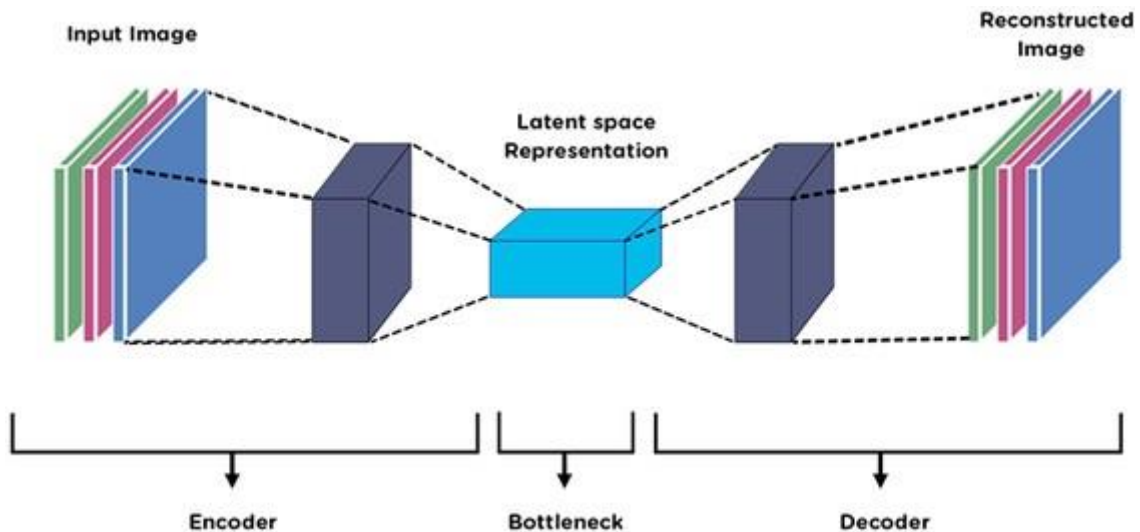


Figure 2.10 Parts of an autoencoder [22]

2.2-Advantages of Deep Learning

The advantages of deep learning are what led people to call these kind of shift the fourth industrial revolution. Based on its diverse set of architectures, there are many applications in which deep learning can provide useful results:

1. Different chatbots have been created using RNNs to improve customer experience.
2. Google translate has been improved by the use of LSTMs and autoencoders.
3. Paraphrasing tools and other text generation apps of different kinds have been implemented using deep learning.
4. Colorizing old photographs is now no longer needing human intervention.
5. Autonomous vehicles have become a reality with the help of deep learning.
6. Sentiment analysis aggregates news based on each individual's preference.
7. Robotics are seeing rapid advancements because of deep learning.

2.3- Limits and Disadvantages of Deep Learning

1. The need for training data: no matter if the training is supervised or unsupervised, there are problems where gathering data is troublesome (as an example: try and predict how long a human can live without food, there's simply not enough data to train for a decent level of accuracy).
2. Lack of trust of the general population: unless a trivial task is performed, like colorizing a photography, people would rather have a human performing the task.
3. The transparency problem: these kinds of programs can get so complex that debugging becomes a serious challenge.

3. STATE OF THE ART

There was a time when expensive statistical and predictive software was available only for large enterprises while small business and ordinary people were one step behind, where automation of tasks was a luxury and software aided flexibility was a thing for corporations and large store chains. Many still think that in order to employ automation and computer decision in your business you still need either a lot of financial aid, or a lot of processing power. But human analysts, without the aid of software of this kind, cannot compete in a world where other software does thousands of decisions a second with no decision fatigue, no breaks and no remuneration.

For this reason, I created my diploma project to prove that using deep learning for sentiment analysis is neither expensive nor does it require massive amounts of processing power. Usually, when people think about deep learning, and especially when think about implementing automation in a sector of their business, they think that it is still far out of reach. The fault is not on them though, this technology has emerged at a colossal speed, and it still will be a long time before people dare to use these powerful tools. This diploma project will show how different types of models, a feed forward network, a RNN and a CNN perform on a dataset that was created for business applications, respectively the Amazon Reviews Dataset from Kaggle, and it will prove that even training such a model on a modest laptop inside your own home can yield performance above expectations.

3.1- Needs and technologies

The reason I chose this kind of project for my thesis because the deep learning approach of the AI domain is in continuous development and there are thousands of ways to make your life easier by using it. And with so many possibilities, I've decided to pick something that can aid businesses become more responsive to consumer opinion.

With all these being said, as I've previously stated, deep learning models can be powerful and don't demand a lot of computation power. But a deep learning model is only as strong as the data you're working with. And by data I don't refer only to the dataset used, but how that data is manipulated so that only the significant pieces remain, which is far from being a straightforward task, it's more like tuning a guitar rather than adding 2+2. Before the deep learning model is

implemented, it is great for the human user to be able to visualize the data, then different parts of the data must be removed. Also, in order for the model to be able to train with the data, it cannot take literal words as input, so words must be transformed into something readable. Finally, a trained model should have a decent accuracy, should train in a reasonable amount of time, and the output can be used as it is or it can be merged with other software.

3.1.1- Essential text classifier features

- Data cleaning

Having in place a preprocessing code can save a lot of time and frustration when you're building models. It's the low maintenance part of the project that can transfer easily to different kinds of other project that use the same data. Since sentences can contain different variations of the same word, known as its lexical family, can contain typos, slang or words that have no business in deciding the sentiment of a piece of text. Also, keeping emojis while removing irrelevant punctuation is important.

- Data visualization

In the beginning, before altering the data, it's important to see things like the balance between negative and positive elements, how long the sentences are, how many of each word does the dataset contain, check the words for positive and negative frequencies and plot them to visually assess how positive or how negative a word is.

- Presentation of the dataset

Datasets can come in all shapes and sizes, and labels are not always separated from the text, because some datasets are made with only one specific approach in mind. In this case, the Amazon Dataset was created in a fasttext format, which means labels are merged with the text. Placing your dataset into a dataframe can be easier on the eyes and can aid further in using your data.

- Word Embeddings

Word embeddings is what makes the magic happen. The process is pretty

straightforward, but essential. The model still must work with numbers, it does not understand words. This way, every word has a number assigned and a dictionary called a word index is created. Not only is a number assigned to each word, sentences now have become sequences of numbers, which “map” the sentiment in the grand scheme of things.

- A proper deep learning framework

In the field of deep learning there are two main frameworks that are being used for developing models and even data preprocessing or visualization, removing extra work and allowing to build without a deep understanding of the whole process. The two frameworks were released and are maintained by two tech giants, Google and Facebook. These two frameworks are quite similar in performance, but have different coding styles.

3.2- Similar Applications

3.2.1- Packaging

It is common for business solutions of this sort to come as SaaS for businesses to integrate into their own apps. SaaS APIs for text classification are much easier and faster to implement, require fewer resources and less technical skill. It is probable that text classification APIs will be along for the ride in the continuous ascension of the SaaS industry.

- What is SaaS

SaaS is a software distribution model in which a provider in the cloud hosts the solutions and offers them to clients over the internet. Depending on the size and resources of the vendor, the software can have an independent cloud provider, or, as in the case of Microsoft for example, it can possess its own cloud.

- How does SaaS work

In the software-on-demand SaaS model, the provider gives customers network-based access to a single copy of an application that the provider created specifically for SaaS distribution. The application's source code is the same for all customers, and when new features

or functionalities are released, they are rolled out to all customers. Depending on the service-level agreement, the data doesn't have to be specifically stored in the cloud, it can also be stored local or a combination of both.

- APIs vs open libraries

Even though APIs sound like the clear choice, open libraries to build your own products should not be dismissed, especially when you have highly specific needs, as in having a store with either a single or few item categories. Moreover, using APIs is little to no work and is not fit for a thesis project, although its popularity among different organizations is understandable. Open libraries are the ones used for developing the actual APIs that are after that monetized. There is also the safety and reliability of owning the products you are importing in your own software.

Moreover, even buying API solution to your text classification needs can be an unwise investment, since deep learning solutions are created for highly specific tasks and do not perform well when generalized.


[Try The Demo](#)


Try our free demo now by typing a sentence or choose from the options in the drop-down menu.

Select A Language English ▼

What a great day to have your broken TV delivered. I am really happy i get to send it back.

Analyze


Positive
58.70 %


Neutral
13.40 %



Negative
28.00 %

Figure 3.1 comprehend.io sentiment analysis API [23]

An API that is specialized on detecting sentiment is not at all fit for detecting sarcasm, so buying multiple solutions for different kinds of raw text can get expensive real quick, while implementing the solutions can drive down on costs in the long run, while at the same time allowing for debugging, optimization and overall freedom in improving your own tools.

Since this API is not enough for any kind of text, the company had to come up with different APIs that focus on specific text aspects, charging for every solution separately.



Figure 3.2 Different types of APIs for different text classification needs from komprehend.io [23]

Top SaaS APIs for Text Classification [24]

These tools offer pre-trained models or trial versions to decide if they are the right fit for your needs:

1. MonkeyLearn
2. Google Cloud NLP
3. Aylien
4. IBM Watson
5. Meaning Cloud
6. Lexalytics
7. Amazon Comprehend

These tools offer pre-trained models or trial versions to decide if they are the right fit for your needs:

Top Open Libraries for Text Classification

Most of these tools offer trial versions or pre-trained models, so you can try them out to see if they fit your business model:

1. Scikit-learn

This library is usually used for beginners for multiple reasons: first of all, because it is very well documented, secondly, because it is based on well known libraries such as matplotlib, numpy or scipy (libraries for advanced mathematics, statistics and engineering). Scikit-learn possesses multiple supervised and unsupervised learning algorithms and also has tools for preprocessing the data and statistical modelling. One of the companies that publicly revealed their use of the scikit-learn library is Spotify.

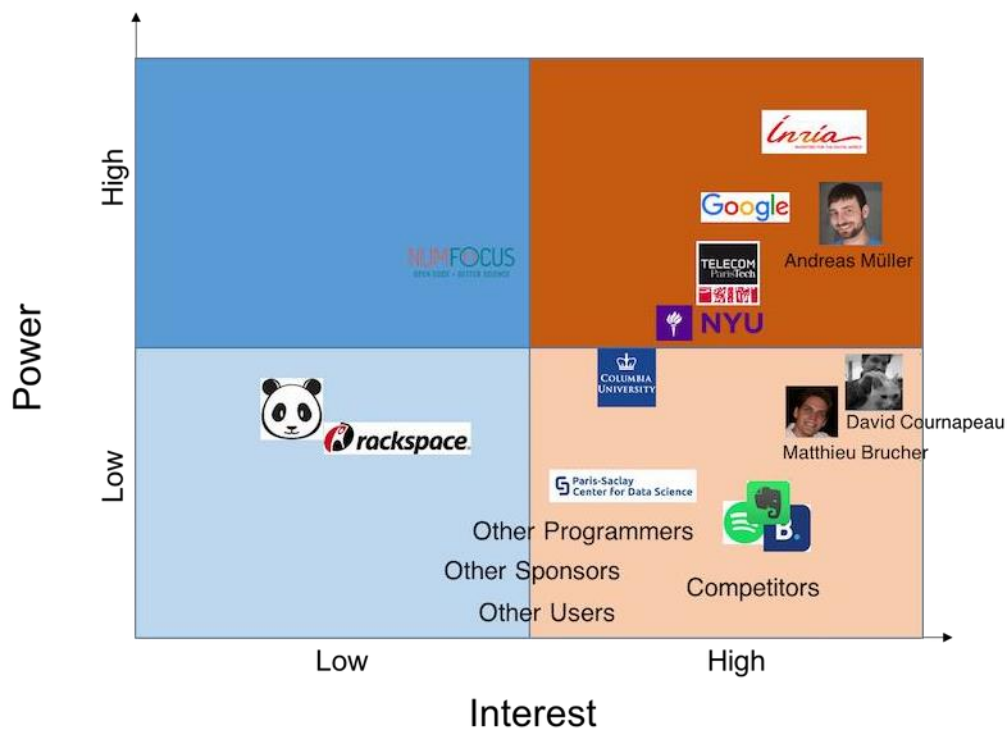


Figure 3.3 Power-interest grid of scikit-learn stakeholders [25]

The power-interest grid of scikit-learn, on the x axis, shows how willing one is in contributing to the project and using the library in their private endeavors, and the y axis represents the size of the shareholder and its influence of the library's future. That being said,

there is a clear interest in this library and big businesses that have classification or generation needs are considering it.

2. Natural Language Toolkit

This library is the main python package used for natural language processing. It is needed because of its high number of packages for tasks like tokenization, stemming, removing different stopwords, and other functionalities that will be discussed in the practical part of this paper.

Similar to scikit-learn, this library is documented fairly well, having both its own site and a guide written by the creators of the library. This guide introduces topics in computational linguistics and contains lessons for comprehending the basics of python, such that the NLTK library can be used not only by data science and AI professionals, but by people with no specific knowledge of this topics, such as engineers and researchers that require a high level library for developing their own tools. It is particularly of great use compared to other libraries when it comes to working with multiple languages, since it has over 50 languages supported [26] [27].

3. SpaCy

SpaCy is the direct competitor for NLTK, although, depending on the purpose of your project, one might be more suited than the other. SpaCy is better than NLTK for development of applications, while for research and education NLTK is simpler and more straightforward.

4. TensorFlow

Tensorflow has become more than just a library, it is now an ecosystem for creating and deploying machine and deep learning models. It has been created in 2015 by the Google Brain team for internal use, but is now an open source framework. It uses Python code because of its convenience, while running the applications in C++ for speed. Tensorflow applications can run on platforms no matter their operating system, being it Android, IOS, Windows or Linux. It can even run in the cloud when using Google Colab, although their cloud can be used for a limited amount of time, making the preprocessing and training of large amounts of data impossible, yet it supports the developers and the new learners by giving them access to Google machines, so they can train their models without using local resources. In version 2.x Tensorflow incorporated the Keras API for model training, which was a standalone product before that.

Advantages of Tensorflow are the abstraction it provides. Deep learning is, under the hood, math heavy and has a steep learning curve, but using the Tensorflow framework allows the developer to focus more on the overall logic of the project and improve it rather than work to reinvent the wheel all over again. Moreover, Tensorflow is very well documented, and also has a repository of trained machine learning models, which can be very handy in development [28] [29].

5. PyTorch

Pytorch is the equivalent of Tensorflow, but created and maintained by Facebook. Although the syntax is still in Python, they are very different in their approach. Pytorch is very good for someone with a coding background that works in an app developing work field rather than someone who is into research and education. Other than that, the performance is similar to its counterpart, but one advantage is the native support for Open Neural Network Exchange and can allow the developer to export the models directly into ONNX format [30].

6. Keras

Keras is probably the most used high-level neural network API. It has been greatly appreciated for tearing down the barriers that made developers reluctant to explore use cases for neural network technology. The API was “designed for human beings, not machines,” and “follows best practices for reducing cognitive load.” Keras has the great advantage of being widely adopted and therefore possessing significant amounts of quality documentation and also receives strong support from Google, Microsoft, Amazon, Apple, Nvidia and Uber. Keras is still just an API, it needs a back-end engine to perform its low level calculations, and it has plenty, Tensorflow being only one and the most popular, it also is integrated with CNTK, Theano, MXNet, and PlaidML. What I mean by saying it cannot perform its low-level calculations? Since Tensorflow has as its data structure the tensor, which is in its most basic definition a generalized array, Keras has as its data structure the model [31].

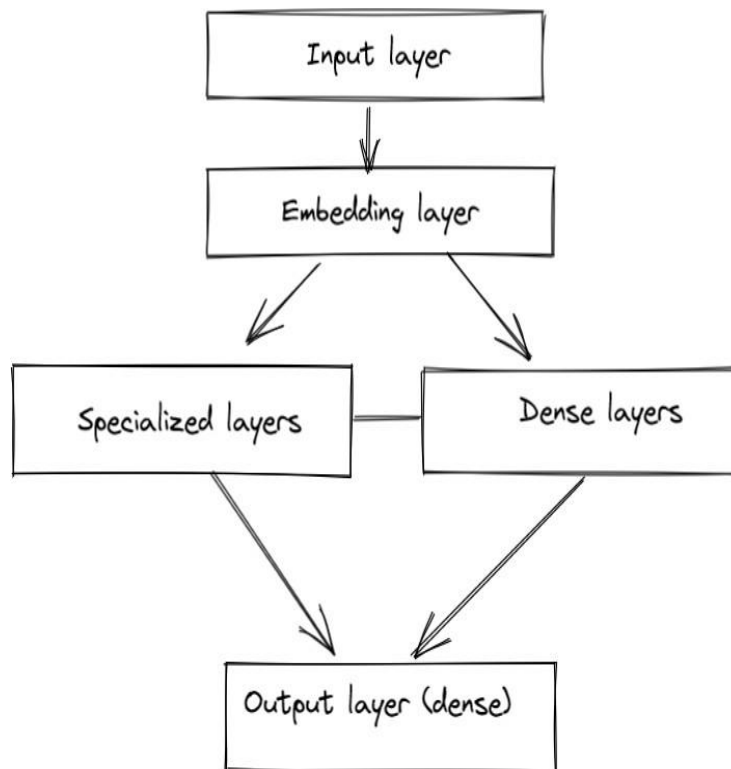


Figure 3.4 Diagram representing the keras sequential model topology

The keras sequential model in NLP usually starts with an input layer, then has an embedding layer, which takes the data creates a matrix of weights, allowing for different types of layers to continue the process. The layers in between the embedding and the output layer are either specialized network cells, like LSTMs or GRUs, or can be simple Dense layers with an activation function. Finally, an output layer with the shape according to the wanted result is given. [4]

3.3- Relevant research in the domain of sentiment analysis for e-commerce

My diploma project has as its aim presenting the possibilities of using the aforementioned deep learning open source tools for creating powerful tools that can help different kinds of entities, from Wall Street trading giants, music platforms with playlist generating capabilities, retail chains, big research institutes or big marketing firms to the average Joe that might be interested to trade stocks, to analyze the current trends on his online shop, meaning that you no longer need large amounts of capital to make good decisions and you can create different tools by using the same building blocks, cutting the learning curve you have to invest in all the domains that you are trying to get involved in.

There are numerous academic papers that analyze the efficacy of using deep learning in

sentiment analysis for e-commerce purposes. Different researchers are inspecting different types of cells, like Gate Recurrent Unit, Long-short Term Memory, even Convolutional Layers and a combination of these three for achieving greater and greater accuracy. Research work in this particular niche is happening in China more than in other parts of the world. The interest over traditional machine learning comes with the desire of reducing human intervention, yet the cost is the sheer requirement of data.

1. "Sentiment analysis of agricultural product ecommerce review data based on deep learning" - H. Zikang, Y. Yong, Y. Guofeng and Z. Xinyu

“The relevant comment texts of agricultural products sold on the e-commerce platform contain various and complex emotions of consumers, and reflect customers' consumer experience and views on those products. Compared with the features and advantages of English text segmentation structure, the analysis of Chinese sentiment polarity in a complex context has always been a problem in deep learning. This article explores the issue of sentiment polarity analysis of short texts of Chinese reviews of agricultural products by constructing Text-CNN, Bi-LSTM and BERT to fine-tune three deep learning models. The research object adopted is the review data of agricultural products on the e-commerce platform obtained based on crawler technology. The experimental results show that, compared with the Bi-LSTM and BERT fine-tuning models, the Text-CNN model's accuracy, precision, recall, and F1 value are all about 3 to 8 percentage points higher. The classification effect gradient gap between the three models is obvious. The accuracy of the Text-CNN convolutional neural network reached 99.92%. The results confirmed that Text-CNN has high sentiment analysis accuracy and good sentiment polarity classification effect in processing agricultural short text comment data, as an agricultural product e-commerce The emotional classification model of sales review data can provide an important reference for manufacturers to accurately distinguish user feedback and enhance market competitiveness [32].”

2. “Sentiment Analysis for E-Commerce Product Reviews in Chinese Based on Sentiment Lexicon and Deep Learning” - L. Yang, Y. Li, J. Wang and R. S. Sherratt

“In recent years, with the rapid development of Internet technology, online shopping has become a mainstream way for users to purchase and consume. Sentiment analysis of a large number of user reviews on e-commerce platforms can effectively improve user satisfaction. This paper proposes a new sentiment analysis model- SLCABG, which is based on the sentiment

lexicon and combines Convolutional Neural Network (CNN) and attention-based Bidirectional Gated Recurrent Unit (BiGRU). In terms of methods, the SLCABG model combines the advantages of sentiment lexicon and deep learning technology, and overcomes the shortcomings of existing sentiment analysis model of product reviews. The SLCABG model combines the advantages of the sentiment lexicon and deep learning techniques. First, the sentiment lexicon is used to enhance the sentiment features in the reviews. Then the CNN and the Gated Recurrent Unit (GRU) network are used to extract the main sentiment features and context features in the reviews and use the attention mechanism to weight. And finally classify the weighted sentiment features. In terms of data, this paper crawls and cleans the real book evaluation of dangdang.com, a famous Chinese e-commerce website, for training and testing, all of which are based on Chinese. The scale of the data has reached 100000 orders of magnitude, which can be widely used in the field of Chinese sentiment analysis. The experimental results show that the model can effectively improve the performance of text sentiment analysis [33].”

3. “Sentiment analysis for e-commerce product reviews by deep learning model of Bert-BiGRU-Softmax” - Liu Y, Lu J, Yang J, Mao F.

“Sentiment analysis of e-commerce reviews is the hot topic in the e-commerce product quality management, from which manufacturers are able to learn the public sentiment about products being sold on e-commerce websites. Meanwhile, customers can know other people's attitudes about the same products. This paper proposes the deep learning model of Bert-BiGRU-Softmax with hybrid masking, review extraction and attention mechanism, which applies sentiment Bert model as the input layer to extract multi-dimensional product feature from e-commerce reviews, Bidirectional GRU model as the hidden layer to obtain semantic codes and calculate sentiment weights of reviews, and Softmax with attention mechanism as the output layer to classify the positive or negative nuance. A series of experiments are conducted on the large-scale dataset involving over 500 thousand product reviews. The results show that the proposed model outperforms the other deep learning models, including RNN, BiGRU, and Bert-BiLSTM, which can reach over 95.5% of accuracy and retain a lower loss for the e-commerce reviews [34].”

4. “Research on Text Sentiment Analysis of E-Commerce Comments Based on Deep Learning”

“For the huge amount of comment data, the high cost of emotions is analyzed manually,

and the LSTM is used to analyze the emotions of business comment. In order to train the model, including text de-duplication, removal of stop words, Chinese word segmentation, and a series of data processing, the data are coded into word vectors. Finally, the crawling data is used to verify the accuracy of the method. The result shows that the accuracy of the method is 75% [35].”

5. “Sentiment Analysis For Product Reviews Based on Deep Learning” -
Shaozhang Xiao, Hexiang Wang, Zhi Ling, Lanfang Wang and Zhaoxia Tang

“With the popularity of the Internet and e-commerce, the sentiment analysis of text can help users to quickly and accurately obtain effective information they are interested in from massive product reviews to purchase satisfactory products. In this paper, a sentiment analysis system for product reviews was designed based on deep learning, and the digital and electronic products on Jingdong Mall with at least 100,000 reviews were crawled as the training data set. After data pre-processing operations such as word segmentation and removal of stop words for product reviews to remove useless features, the feature vectors were constructed based on the bag of words model and word2vec method, and then three classification algorithms, namely LSTM, Naive Bayes and logistic regression were used to model reviews. The LSTM algorithm is significantly superior to Naive Bayes and logistic regression algorithm in the training stage, and provides a reliable reference for the analysis of product reviews [36].”

As a conclusion, it can be observed that deep learning as a tool for sentiment analysis is of interest in the research and development communities. Studies like the ones above and others suggest that deep learning has become more efficient than traditional machine learning methods and the efficiency of the models developed by these specialists are somewhere in the range of 70-95% efficiency, depending on multiple variables. The processing power used, the dataset that was available for their own specific task, even the language and culture can make a great difference, since Chinese reviews tend to be more pragmatic and less sarcastic or emotional, which can lower the need for different very specific preprocessing steps and enhance the results of their models. Nevertheless, one thing is clear: deep learning is here to stay, especially in the realm of text mining, and the performance of the models is only going up, especially with this very interconnected internet community that not only shares their code, but imports and builds upon it, leading to an advanced acceleration and democratization of the field.

4. PROJECT DEVELOPMENT

4.1- Introduction

Now to the more practical part of this paper, to utilize the tools and technologies mentioned, I have decided that showing just the performance of a model is too high-level for a person to draw any conclusions and to understand the possibilities that these deep learning models possess. What I am about to present can be modified to fit another purpose, showing how versatile such software can be and how rewarding it is to learn the ins and outs of building models for classification, and ultimately, predictions.

Although there are many technologies coming into play, they are very well integrated and the simplicity and elegance with which you can harness the power of deep learning is astounding. In many other niches, having so many technologies interplay requires a lot of time dedicated to debugging, creating a lot of frustration and discouragement, but here it is not the case. This great advantage was not a mistake, behind deep learning there is a great community, from the ones that build the tools to the ones that utilize them. They are created with the human in mind, leaving the focus on actually building and optimizing your models rather than spending time putting it together. And the platforms like Kaggle, but also DrivenData, CROWDANALYTIX, Signate, Zindi, Alibaba Cloud Tianchi, Analytics Vidhya, CodaLab are responsible for the great hype that this field has received in the last years. Not only do they provide a hub for people with this common interest to discuss, some of them actually organize competitions, offer rewards, and offer free courses that can get a novice started on the right path. Moreover, there are ranking systems which can be then used in a CV, making people more willing to get better, since now they can prove to the employer their experience before even getting into the interview. Besides the learning benefits of such platforms, there is one essential component that aids the learning process and the development of the field, both in research and development, tremendously: data. Data is by far one of the hardest pieces of the puzzle to attain, probably the reason why there is so much effort put in getting people involved with the field. Kaggle provides numerous datasets, many of them cleaned of impurities and carefully selected so the aspiring data scientist can work with it and get good results. Highly optimized deep learning models yield high results usually when it comes to sentiment analysis tasks, yet the problem that developers and, especially, researchers face is finding data that contains only the right things and is also abundant enough for the training to occur properly.

As for my motives, I aim that the words below should serve as a guide for anyone that

would like to create a classifier for sentiment analysis using the tools I am about to present and that have been mentioned or introduced in the previous chapters.

4.2- The dataset

The dataset I have chosen for this project is hosted on Kaggle and is called “Amazon Reviews for Sentiment Analysis”. This dataset comes in fasttext format, which is very good if your plan is to train a model quick, but not fit for a thesis project in which you try and show the inner workings and concepts behind the hood. The dataset was initially posted in .csv format, which meant you could open it in Excel and look at the reviews for yourself, but the .csv file no longer exists and now the data in fasttext format requires extra preprocessing to bring it to what .csv could have offered us. The dataset does not require splitting for training and testing, since the author has beforehand considered this requirement and provides the dataset already separated. Even if the dataset was not separated, this wouldn’t have been an issue, there are tools that can take care of this issue with no effort, yet it saves some computational power.

The Kaggle rating for this particular dataset is 6.9, since it has no file descriptions, license, provenance or update frequency specified.

As for the community around this particular dataset, it has 416k views, yet only 9% download rate, amounting to a little under 40k views. This shows that this dataset is not a toy, it is not supposed to be used by a beginner, because this data is real business input on a reasonable scale, and using fastText it can be trained in minutes. Since fastText is so much faster than doing the preprocessing and training on your own, implementing all the required functions and optimizing it accordingly, this only means it will take some time for my models to train. [37]

4.2.1- The fastText library

As previously mentioned, the raw data comes in fasttext format. This format allows the person working with the dataset to take it as input when working with the fasttext library. This library was created by Facebook and initially used internally, but was later open-sourced and now it is available for the world, allowing people with modest means in terms of computing power to train their models much faster compared to using traditional methods. Since it has been open-sourced, the Github community approved of it, receiving votes of confidence from tens of thousands of developers who have actively used it.

FastText is a library built with efficiency in mind, used for learning of word

representations and classification of sentences. It is originally coded in C++, hence the improvement in speed. Moreover, it support multiprocessing, further improving the performance. Fasttext is used both for training supervised or unsupervised (labeled or unlabeled) sets of data. The applications of this library are numerous, starting from data compression, to predictions, to transfer learning.

- How fasttext works

To be efficient on datasets with very large number of categories, it uses a hierarchical classifier instead of a flat structure, in which the different categories are organized in a tree (think binary tree instead of list). This reduces the time complexities of training and testing text classifiers from linear to logarithmic with respect to the number of classes. FastText also exploits the fact that classes are imbalanced (some classes appearing more often than other) by using the Huffman algorithm to build the tree used to represent categories. The depth in the tree of very frequent categories is therefore smaller than for infrequent ones, leading to further computational efficiency.

In order to be efficient on datasets that hold a large category set, it uses a hierarchical classifier instead of a flat structure, in which the different categories are arranged in a tree rather than a list. This reduces the time complexity of training and testing since the complexity is brought down from logarithmic to linear with respect to the number of classes. Fasttext also helps when the classes are inbalanced (classes don't appear for an equal amount each) by using the Huffman algorithm used to build the tree to represent categories. Huffman algorithm is used for lossless data compression. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code. The variable-length codes assigned to the input characters are called prefix codes, meaning that the code assigned to one character will never be a prefix of another code of another character, thus reducing ambiguity. By this principle a tree is build made of input characters, and then numerical codes are assigned to each of them. Since this algorithm is very low-level and requires another set of skills to fully comprehend, it is only mentioned with the purpose of reference rather than to dissect its inner mechanics. Therefore, the idea here is that the deeper into the tree, the rarer the input, thus leaving the most used inputs at the top, aiding efficiency.

FastText is able to represent text by a low dimensional vectors, obtained by adding vectors corresponding to the words appearing in the text. In fastText, every word in the vocabulary such a low dimensional vectors associated to it. This kind of representation can then

be shared between classifiers for different categories, allowing the information regarding the data to be used by indiscriminately among the other categories. These kind of representations are called bag of words.

Moreover, in fastText local word order is represented using word ngrams, which, depending on the classification problem you are engaged in, can make a great deal of difference[38] [39] [40] [41].

As a conclusion, fastText is a truly robust data science and machine learning tool, that is, and probably will be for some time, a work in progress. It has taken very complex tasks and managed to compress them into just a couple of lines of code. Not only is it efficient in terms of saving tons of lines of code and cutting the learning curve of any text classification project by a lot, it also is very time-efficient! The team at Facebook Research fastText to the test by comparing it with two types of convolutional neural networks, achieving remarkable results: it achieved more or less similar accuracy rates as the two convolutional neural networks, but the time needed was a matter of seconds, instead of a matter of hours or even days. This is truly a breakthrough, because it allowed the use of datasets that were previously impossible to train on. As an example fastText trained on a billion word dataset in less than 10 minutes. Compare that to the regular previously mentioned CNN that would probably take at best months to train on that amount of data, and at worst even tens of years. Although it is received with applause in the development community, in education it does not make for a good tool, since it does not allow for unfolding of different mechanisms that happen behind the library and will leave students with no real understanding of anything besides library calls[40].

4.2.2- The content

Since this dataset comes in a fastText format, the data inside has this pattern: `__label_<X>_label_<Y> ... <Text>`. A review can have only one of the two labels, X or Y. There are no quotes and the labels are merged with the review. In this case the classes are `__label_1_label_2`. Since on Amazon there is a 5-star rating, the ones with 4 or 5 stars have been deemed to be positive, and the ones with 1 or 2 stars negative. The ones with 3 stars have been excluded for this dataset, because of their ambiguity. They were not included in the original.csv dataset either. At the beginning of the string, there is always the label, followed immediately by the review title. The review title is then differentiated from the rest of the text by a separation made with a : and a space.

Most of the reviews are in English, but there are other languages among those, since most of the reviews come from US customers, which can have very diverse backgrounds. The other

language most used in the dataset is Spanish, so for the sake of simplicity these are the only two languages I am going to take into consideration when implementing the software for this paper (the others shouldn't make a significant difference, and since the training will be done on a reduced dataset it should be more than enough to bypass any foreign words) [37].

4.3-Python for deep learning

Python is steadily heading to being one of the most used programming languages on the planet. Not only is it the go-to programming language for AI developers and data scientists, it holds his own in other software fields, like web programming or cybersecurity.

“Both machine learning and deep learning rely on extremely complex algorithms and multi-stage workflows, so the less a developer has to worry about the intricacies of coding, the more they can focus on finding solutions to problems, and achieving the goals of the project.” – Jakub Protasiewics, Python Deputy Engineering Manager. This means that, due to its simple syntax, a developer can focus more on testing and optimizing the algorithms rather than having to implement them.

Besides the convenience it brings to the table, Python is surrounded by a vivid community with tons of high-quality resources, the reason for that being the readability of the code and the incentive to share in the AI community. There are many free and paid courses and full career paths one can take using the internet.

Most importantly, Python is the go-to choice because of the sheer number of libraries designed for implementing a vast array of applications, making it both easy for the developers to focus on the creative part rather than technicalities and also making it accessible for students to engage in this kind of work without getting stuck [42] [43].

4.4-The environment

When one embarks on a deep learning project, among the first questions he needs to ask himself is what environment should I use, and the answer is: it depends. When one works in a research lab or in a enterprise of large proportions, possessing great computational power and also has a lot of experience regarding certain inconveniences that may come along the way through conflicts in the code and being concerned with security matters, one may choose to use local computing. On the other end, a student or any learner, armed with a laptop and his limited free time, might not consider training a model locally as being a good idea. Therefore, these are the main two environments that can be used based on your needs:

- Anaconda IDE

This package is installed on a local machine and has great benefits if running it on your local power is for you. One of the main benefits is it offers access to software like Spyder, Jupyter Notebook, R, QT code, etc. Besides that, the community is very fond of this whole IDE because it bundled all this different softwares that were standalones until Anaconda came along (you had to use pip for installing packages, virtualenv to work with pip, and had to add extensions to the whole mix). Anaconda is much more stable, comes as a complete solution, and developers using it consider it handles the libraries way better than its predecessors. Since in conda (the package installer), everything is installed as a package instead of being a compiled resource like in pip, it creates conveniency in the long run, since everything can be updated easier and the developers won't have errors because the version of package X is too old to work with package Y in order to install package Z. Everything runs smoothly to let developers tackle the real issues.

- Google Colab

Google Colab is a great initiative that aimed to give people the computational means to run and test their own models, at least on an educational scale rather than business. This environment is the right tool if you do not own a great CPU or GPU, but that is not the only reason people choose it. It has the massive advantage of pre-installed libraries, so the only concern one might have is importing what one needs. Secondly, the portability feature is not to be taken lightly: since everything is stored in the cloud (your google account with your google drive directories), you can access your work on any machine by logging with your google account. Also, if storage space is a concern for you, the dataset used can be imported from google drive, further dropping the resource demand of the user. Thirdly, collaborating on a project with more people has never been easier until now: Google Colab allows multiple developer to work on the same notebook in the same manner many people work on the same Google Docs at the same time, which can greatly improve efficiency in a project.

Google colab lets you choose between using the local resources or using the cloud. Using the Cloud lets you use Google's GPUs and TPUs. GPU stands for Graphical Processing Units, which are most often present on local machines too, but TPUs are tensor processing units developed by Google to accelerate operations on a Tensorflow Graph. Each TPU packs up to 180 teraflops of floating-point performance and 64 GB of high-bandwidth memory onto a single board. In other words, Google created a whole ecosystem, having its own cloud IDE (Google

Colab), deep learning frameworks (Tensorflow), its own storage space (Google Drive) merged, hence the hype in the data science and AI community [44] [45].

For the task of classifying reviews for Amazon I have decided to use Google Colab, since the thesis has as its focus building three different kinds of models and analyzing their performance, seeing how fast each trains, how accurate they are and how they operate. This task does not require the whole dataset, because we are aiming for comparison, not peak performance. If this was not an educational project and should be deployed for an actual business, the larger the dataset utilized the better. But again, training on the whole dataset locally would probably take days on end, and Google Colab, with all its features, has one limit: the code cannot run for more than 12 hours and the state of your machine cannot be idle for more than 90 minutes, thus it not only limits the time but compels you to stay at your computer for a prolonged period of time.

4.5-Description

The program in this thesis aims to classify Amazon reviews as either positive or negative, using different methods and putting them to the test for comparison. In order to do this, the dataset is imported from Google Drive, then it is reduced to only 100k reviews for training and 100k for testing, because, as previously mentioned, Google Colab does not allow for more than 12 hours of continuous use, therefore training on 100k reviews will take around two to three hours to complete, which is reasonable when you seek to test and modify for optimization rather than chase the most precision you can get out of your current model. After the dataset is limited to the desired number of reviews, the training set is split into two smaller sets: training and validation. The test dataset comes in a separated file, so a split for testing the model is not required. Once this is done, all the newly divided raw data subsets have to be preprocessed. Preprocessing in the natural language processing domain means to keep only what is essential and will help the model learn the right patterns. Then, once we obtain the final data, it is ready to be transformed into word embeddings, which basically create a matrix of weights from text. After all this, the data is ready to be fed into the desired model. The models are then training with the corresponding set of data, also using the validation to tune the parameters while at the same time evaluating the model. Finally, the model can be evaluated with the testing data and it can make predictions, which can be compared with the actual truth labels. Moreover, visualization can be performed not only for model performance, but also to get a feel of the data. In the following chapters I will break down the previously mentioned steps, examine our options as developers and implement the methods deemed fit for our task.

4.6-Data preprocessing

Data preprocessing is used because of the dirty nature of data, it has inconsistencies, it has noise and missing values, and that is because it is coming from lots of different sources. If the data comes in the form of images, it can be blurry, it can be edited to something that doesn't bring value to our prediction model. In the case of text, words can have typos, languages can be mixed (Romanians use English words that convey a meaning indescribable by common words in the Romanian vocabulary), and most importantly, context can mess with the relations a deep learning model can find between words. When you are applying data preprocessing, it is probably because you have a great amount of data on your hands, since this cuts into the size of your dataset, and the amount of data is important in deep learning. Data preprocessing in text has the following steps: first of all, since the reviews come in a fastText format (labels tied to the text review), going through the reviews and using string operations to remove the label and store it in another array is a must before doing anything else. Secondly, we must lemmatize or stem the words in the reviews, also known as the text normalization step. Both have as aim to reduce all the words and their derived forms to their common base form. Usually lemmatization is preferred over stemming because it is more precise. Stemming is a heuristic solution that means chopping off ends of a word, getting derived words to a same base, which may or may not be the parent word (for example: axes can be the plural for both axe and axis, but the words convey a clear different meaning and should not be all cut to the root ax). Moreover, stemming has issues with getting the infinitive of a verb (for example: getting the root "be" from "am", "are", "is" is not possible when using stemming in the preprocessing stage). Its only advantage is its speed, but on our reduced dataset the waiting time will not be massively improved by choosing a less costly approach in this case. Lemmatization converts a word to its root by a more costly and more precise approach: using a vocabulary and morphological analysis to convert a word to its original form, also known as the word's "lemma". When using lemmatization you can be sure that your time was worth the wait, since most of the words will turn into their corresponding base form, but it can also make mistakes. For examples, using verbs as nouns (ex: "the next coming") might trick the lemmatizer into creating contextual mistakes.

Lemmatization can be imported from the NLTK library and you can apply it while going through all the reviews. Another great thing to add while lemmatizing is to remove the words with less than three characters, because more often than not they do not convey too much meaning, although that can change depending on the dataset and what you wish to accomplish. In this case, after creating a function that takes the raw data as input and passes through it, when lemmatizing you can remove the words you deem too short as you go along.

After this step, where every or most of the words have been brought to their lemma, removing the stopwords is the next step. Stopwords are the words in a language that are deemed not to add any sentiment to any sentence (connector words). Also making sure that the words you are actually checking for exist in the language (or whatever language your dataset might contain). In this case, the NLTK corpus contains an English and a Spanish dictionary and the stopwords for both the languages, and you can check every word to see if it is found in one of the languages, while at the same time removing the stopwords. After this step, your dataset should contain most of the words with meaning from your language of choice.

Note: If you would like, for data visualization purposes (otherwise it is not a necessity) to split the reviews into negative and positive, you can do that by having a function search a string for the first number that appears. Then from a list of reviews you can create two, one for each sentiment.

After the text is preprocessed, we must now take a look at the labels. In fasttext format, the label comes with the whole string in the form “_label<X>_”. In our case, our two initial labels are __label__1 and __label__2. Amazon uses the five star rating for its reviews, therefore the reviews with one or two stars have been assigned the label 1 and the ones with 4 or 5 stars label 2. The ones with 3 stars have been deemed by the creator of the dataset as not being helpful to the purpose of text classification, therefore they will not be used in this thesis also. You can do them either by using basic python syntax or you can use the label binarizer from scikit-learn. The label binarizer turn the labels which are currently 1s and 2s into 0s, respectively 1s. I have mentioned in previous chapters that scikit-learn is a tool for beginners mostly, but for this preprocessing step it gets the job done. The point here is to never get stuck on a certain technology or library, see what each has to offer and combine them for efficiency without sacrificing the performance. After these steps have been completed, keras has a to_categorical function, which can create a binary matrix out of your labels. [46]

Once you have your preprocessed reviews in a list, with their corresponding labels in another, it is time to create the word embedding so that the text can be fed into deep learning models. There have been tools like Word2Vec created for turning text into a vector space with weights. Word2Vec uses a small neural network to find relations between the words and provide you with what is, basically, a matrix representation. But that is not helpful when the aim is also to understand how it works. Therefore the word embeddings have been created with a more low-level approach, although even this can be considered high-level. First of all, the now preprocessed text must be tokenized. Tokenization is a way of separating small pieces of text into individual objects called “tokens”. Tokens can be anything you need them to be, either words, subwords, or characters, but the mainstream thing to do is to separate each word

individually. And since we have preprocessed them beforehand, this is the best thing to do.

Once we have our sentences tokenized, each token (word), has to be assigned a number. This way, not only they can be used in the embedding vector space, a word gets its meaning “mapped” by the numbers it has around him. This is called turning the text to sequences. Fortunately, both the NLTK and the keras library possess a tokenizer, for this thesis project I have chosen the one from keras.

You should by now have the words separated as individual strings, and sentences turned into one dimension vectors. The tokenizer also creates a word index dictionary for you that should look like this:

```
{ '<OOV>': 1, 'the': 2, 'and': 3, 'this': 4, 'for': 5, 'that': 6, 'not': 7, 'you': 8, 'book': 9, 'but': 10, 'are': 11, 'with': 12, 'have': 13, 'one': 14, 'all': 15,
```

Figure 4.1 The word index dictionary – 15 word example

Note: The oov token means “out of vocabulary” and that’s what the tokenizer assigns to a word it doesn’t know.

Now, all the reviews clearly do not have the same length. In order to be fed to a deep learning model, those sequences that now represent your text must be truncated (cut off) or padded (added length). Truncating is pretty straightforward, you can choose whether the cutting takes place from the beginning or the end of the sequence. Padding is done by adding zeros at the beginning or the end of the sentence. Also, at this point you have to decide on how long you want the sequences to be, shorter intuitively means less data, but might affect the performance. In this case, I have chosen the sequences to all be of length 100 (they can be found in different sizes depending of the task, but can be as small as a few words), also the padding and truncating was done at the end of the sequence. For the basic feed-forward model I have not given a maximum length to the sequences, therefore it will automatically pick as the maximum length the review with the most words.

Finally, once you have reached this point, you should have data that can be fed into a deep learning model. The next chapters will examine the needs and options for the deep learning models.

4.7-Deep learning models

For this task, I have decided to use three kinds of deep learning models. They have in common the input type, all of them receive sentences of length 100. The embedding layer

requires to have already have the text integer encoded (which we did in the preprocessing stage). The embedding layer is initialized with random weights and aims to tune itself to your text. In order for it to work, you have to decide what is the maximum number of features you want to have: in this case, I have decided that the best fit is take the whole vocabulary size.

Note: The embedding layer can have as maximum features the whole array of tokens, or it can have less than that, its job is to tune its weights to your text. Optimizing in this step is not so important in the first model, since this model is not going to outperform the more powerful ones, but I have attempted to optimize the embedding layer's input for the specialized ones. Usually, it is best to have as an embedding size the size of the array which has been created in the preprocessing stage.

1. A basic, multi-layer perceptron model

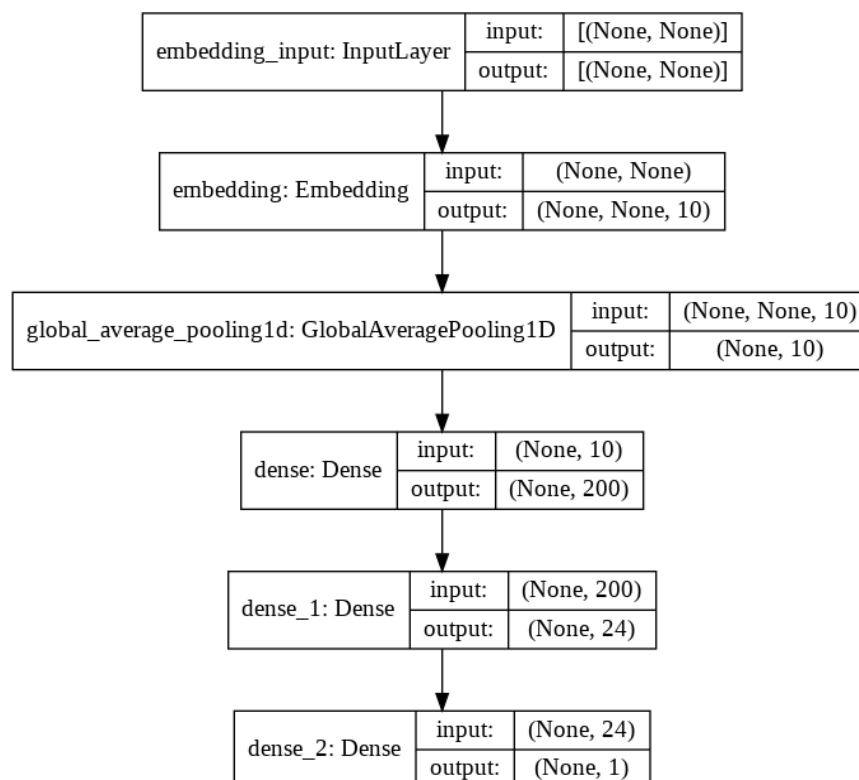


Figure 4.2 Layer structure of the 1st deep learning model

This is the basic model that performs the worst out of all the three models. As for all of them, this one contains the embedding layer, a layer for reducing dimensions, and then layers with activation functions, with only one neuron at the output, since that's what shape we want our output to be (the same shape as the labels used).

- The embedding layer

When given an embedding matrix to the embedding layer, it first of all initializes an embedding tensor, with random weights, which are then adjusted through training. It returns as output a three-dimensional tensor. Although the embedding layer does not have a “right size”, some sizes are better than others. In these examples, as indicated by the Kaggle community [47], one can use a set word length for his embedding dimensions. One can pick either the whole vocabulary or a set of maximum features (equal to the number of words), which is usually the number that is used to initialize the tokenizer.

- The GlobalAveragePooling1D layer

In order to be fed to the Dense layers, it must be flattened (brought to a singular dimension). This is what the GlobalAveragePooling1D layer is for. It is called global because it flattens all the extra dimension and averages the values into only an one dimensional vector.

- The Dense layers

First of all, what is a dense layer? They are called dense layers because they are “densely” connected, meaning that all the neurons in a layer have to be connected to the neurons in the next and previous layer. These layers use activation functions in order to attempt learning the data patterns. Basically, an activation function attempts to decide what to give to the next neuron. We can alter the output by changing the activation function. If an activation function is not manually selected, the basic linear activation will be used ($a \cdot x + b$, where a is the weight and b is the bias, which can be random or none). The thing is, in most cases you don’t want an activation function to be linear, because any combination of linear functions would yield a linear function. If your aim is to have a shallow neural network (2 layers or less), then the linear activation would not stunt your performance, but when using more layers, a non-linear activation is necessary, otherwise adding depth to the neural network model is futile. The most used non-linear activation functions are sigmoid, tanh (hyperbolic tangent) and ReLu (rectified linear unit). For the inner layer, I have chosen ReLu, because sigmoid and tanh saturate (since they represent values in the $[0,1]$ respectively $[-1,1]$ intervals, their lack of sensitivity for very high or very low values creates error, that can then propagate back to the other layers and misguide the updating of the weights). ReLu is a function that takes all the positive values but assigns the value 0 to all the negative ones, thus it might appear linear at the first glance but it’s in fact non- linear. This

activation function was a milestone that allowed the training of deep neural networks. Finally, the output dense layer will take only one neuron with the sigmoid activation function, meaning that a value above 0.5 will be labeled as 1 and below 0.5 as 0. As for the accuracy, it is decent all things considered (it has nothing but activation function layers and is trained on a small dataset). The model trained quicker than the others and gave an accuracy value of 52% on testing.

```

Epoch 90/100
2344/2344 [=====] - 21s 9ms/step - loss: 0.2396 - accuracy: 0.9251
Epoch 91/100
2344/2344 [=====] - 21s 9ms/step - loss: 0.2396 - accuracy: 0.9251
Epoch 92/100
2344/2344 [=====] - 21s 9ms/step - loss: 0.2396 - accuracy: 0.9251
Epoch 93/100
2344/2344 [=====] - 22s 9ms/step - loss: 0.2396 - accuracy: 0.9251
Epoch 94/100
2344/2344 [=====] - 22s 9ms/step - loss: 0.2396 - accuracy: 0.9251
Epoch 95/100
2344/2344 [=====] - 19s 8ms/step - loss: 0.2396 - accuracy: 0.9251
Epoch 96/100
2344/2344 [=====] - 20s 9ms/step - loss: 0.2396 - accuracy: 0.9251
Epoch 97/100
2344/2344 [=====] - 20s 9ms/step - loss: 0.2396 - accuracy: 0.9251
Epoch 98/100
2344/2344 [=====] - 20s 9ms/step - loss: 0.2396 - accuracy: 0.9251
Epoch 99/100
2344/2344 [=====] - 21s 9ms/step - loss: 0.2396 - accuracy: 0.9251
Epoch 100/100
2344/2344 [=====] - 20s 9ms/step - loss: 0.2396 - accuracy: 0.9251

```

Figure 4.3 Training the model

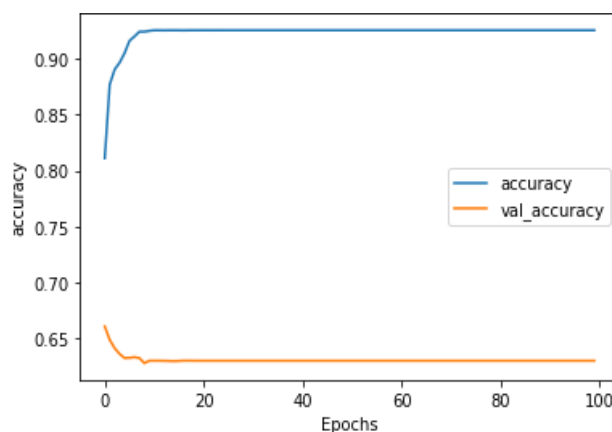


Figure 4.4 Accuracy graph

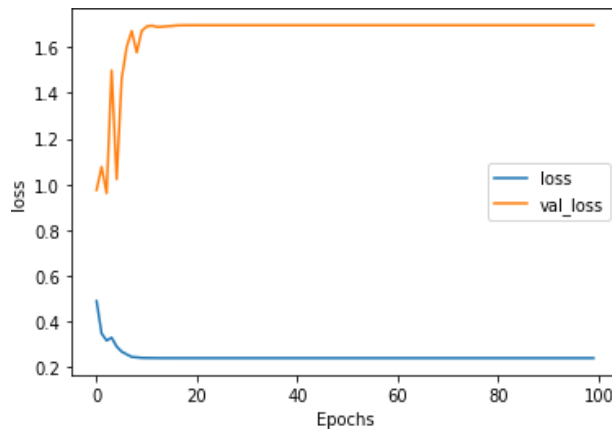


Figure 4.5 Loss graph

The data from the previous graph and also the graphs below for the RNN model, respectively the CNN model, have on their X axis the number of epochs (all of them have been trained for 100 epoch, so that the difference in epoch cannot be interpreted as trying to favor a model at the cost of all others, but the batch size can differ), and on the Y axis the loss or accuracy reached. This model has clearly been trained for too long and suffers from overfitting, since the training accuracy is so much higher than the testing value, which reflects its true performance.

```
3125/3125 [=====] - 6s 2ms/step - loss: 2.4103 - accuracy: 0.5263
```

Figure 4.6 Testing the accuracy

2. A recurrent neural network model

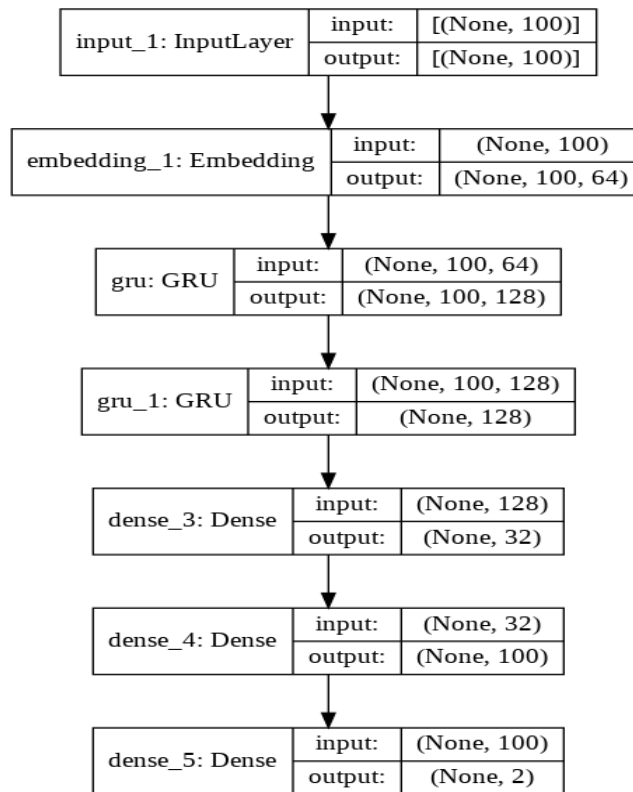


Figure 4.7 Layer structure of the RNN model

This is the most powerful of all the three models. RNNs are naturally better working on text tasks compared to other models. The cornerstone of this RNN are the Gate Recurrent Units.

When it comes to RNNs supervised text classification, you have to decide if your purpose would be better served by a GRU or a LSTM. These two are the evolution of the standard recurrent neural network that came as a solution to the “vanishing gradient” problem: when trying to update the weights of cells, the further away you get from the current cell, the smaller the gradient gets (an intuitive example would be word suggestions that are not taking the whole phrase into consideration, only the last words). As a solution, the GRU and the LSTM both contain a cell pipeline that is responsible for long term memory, yet the way they are organized differ. An LSTM cell has three gates: forget, input and output, that work using the activation function I’ve described earlier (sigmoid and hyperbolic tangent). A GRU cell has two gates, reset and update, again using sigmoid and tanh. For this project I have decided to go with GRU cells for the RNN network for many reasons. First of all, having a less complex cell structure and a lower number of activation functions, it is more effective computationally speaking. Moreover, it

performs better on smaller datasets, and since the dataset had to be reduced for working in the cloud, this came in handy. In the end, certainly there are tasks out there for which LSTMs would perform better, it is a matter of what fits your problem, and I have obtained an improvement in accuracy from using GRUs instead of LSTMs in this specific case.

- The embedding layer

Since I tried to get this one to higher accuracy, for set the embedding layer shape to be exactly 100k rows with 100 columns, to match the way we preprocessed the text (100k word sequences with 100 numbers each). The embedding layer shape can be changed and it will try and create the proper weights using the shape given, but performance might suffer because of it.

- The GRU layers

For this model I have used two GRU layers with 128 units each. It is important, both for LSTMs or GRUs, when stacking them, to select `return_sequences` equal to `true`. By doing this, one output will be returned for every input, and the total output will have 3 dimensions (the same as the output of the embedding layer), which we need for stacking the GRU or the LSTM layers together. The final one, the one before the Dense layers, does not require it to be set to `true`.

- The Dense layers

The dense layers here are simple, the difference between the output of this model and that of the first one is the number of neurons of the output layer (which is now 2 instead of 1), because I have used the `to_categorical` keras function on the labels, and created a 2 dimensional matrix of them. The output layer must have the same number of neurons as the dimension of your layer vector, otherwise you will run into errors.

```
Epoch 1/2
782/782 [=====] - 458s 580ms/step - loss: 0.5666 - binary_accuracy: 0.6637
Epoch 2/2
782/782 [=====] - 453s 579ms/step - loss: 0.2813 - binary_accuracy: 0.8833
```

Figure 4.8 Training the RNN model

The metric used in the model's training is `binary_accuracy`, and for this metric, it is recommended to use either a Range Of Characteristic curve or a Precision-Recall Curve. The

first one is more fit for small imbalances between the classes of the dataset, and the latter for large ones. They will be presented in the following chapter, of data visualization.

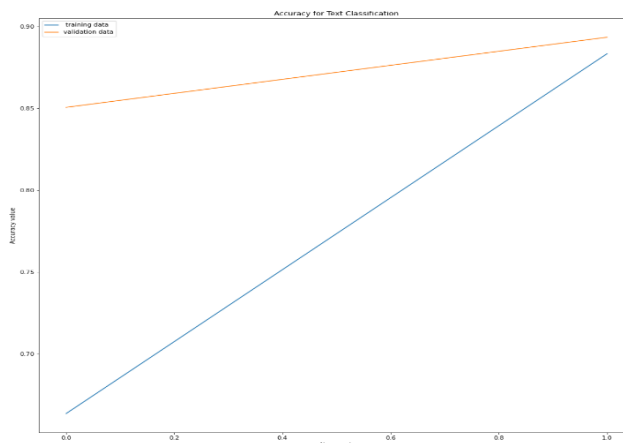


Figure 4.9 Accuracy graph

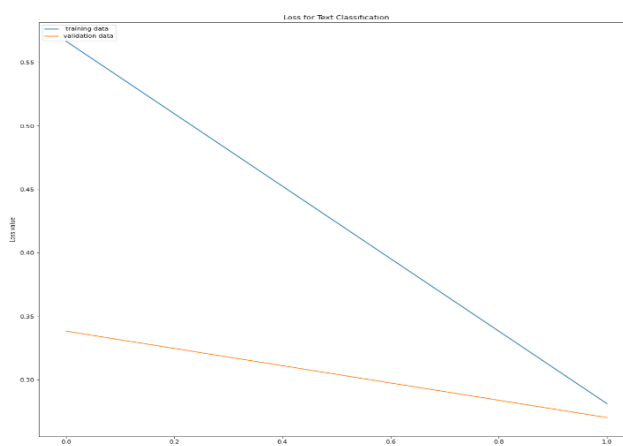


Figure 4.10 Loss graph

```
3125/3125 [=====] - 127s 41ms/step - loss: 0.2788 - binary_accuracy: 0.8890
```

Figure 4.11 Evaluation of the RNN model

The testing phase showed a 88.90% accuracy, which is very good given the time it took to train and the relatively small dataset it was trained on. Overfitting does not occur compared to the previous feed-forward neural network, or at least not a degree of significance. This model takes the most to train but performs the best in less than optimal conditions (have attempted to train it with an embedding layer of less than the vocabulary size and it performed better, which is a idea to consider when tackling larger projects in the future).

3. A convolutional neural network model

This is a model that, although not the most intuitive approach (RNNs are the common use in text), Convolutional Neural Networks can now be applied to text using word embeddings.[47] Convolutional layer apply a lot of filters to the data, their number and their size being given as input, which can impact performance either positively or negatively. A convolution is an operation that takes a two-dimensional set of weights and applies it to a two

dimensional input. Clearly, text in its natural form is not a two-dimensional input, or cannot be in any meaningful manner by itself. That's when word embeddings come to help. They turn the words into a meaningful two-dimensional form.

Note: This definition of a convolution stands in the realm of deep learning, it is a different operation in other fields (here the kernel does not need flipping). [15]

The inspiration came from a famous paper written by Yoon Kim, which had useful information for how to apply CNNs for text classification. [48] Many Kaggle users developed CNNs based on the architecture presented in this paper. Word embeddings create vectors for every word, thus making the sentence a matrix. The convolutions extract the features, which are then flattened, with the last layer being dense, with 2 neurons and a softmax activation function.

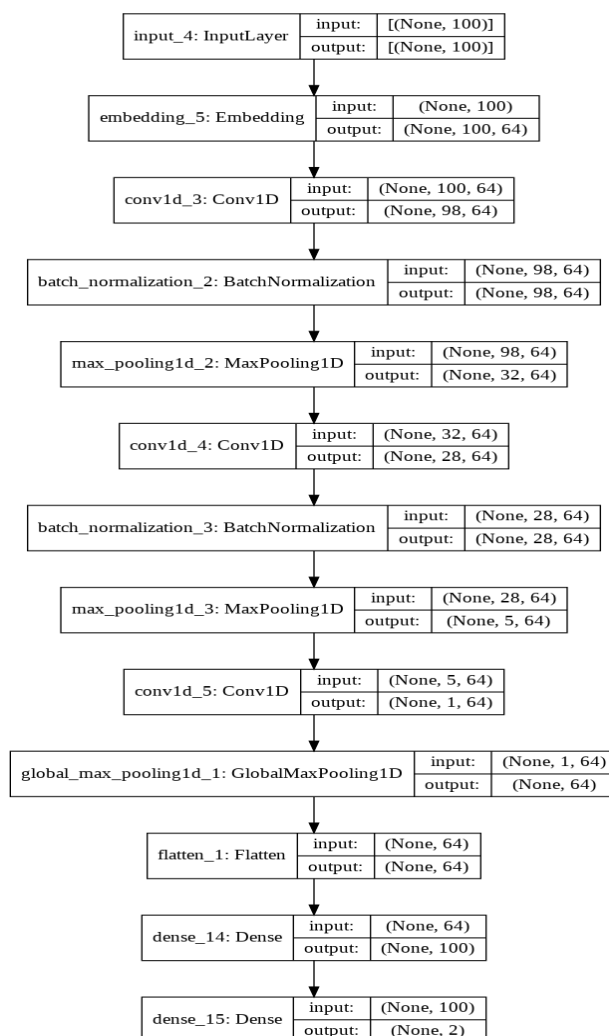


Figure 4.12 Layer structure of the CNN model

```

Epoch 1/2
782/782 [=====] - 68s 84ms/step - loss: 0.3184 - binary_accuracy: 0.8584
Epoch 2/2
782/782 [=====] - 65s 83ms/step - loss: 0.1996 - binary_accuracy: 0.9221
  
```

Figure 4.13 Training of the CNN model

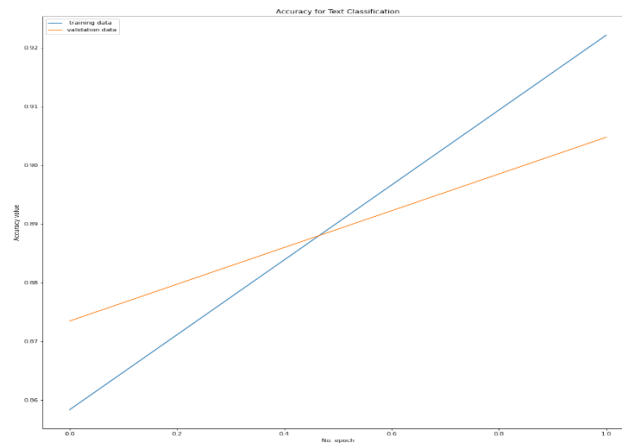


Figure 4.14 Accuracy graph

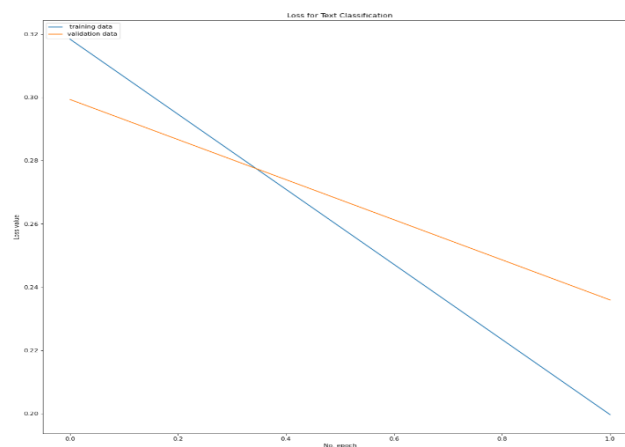


Figure 4.15 Loss graph

```
[ ] cnn_model.evaluate(X_test,Y_test)
```

```
3125/3125 [=====] - 18s 6ms/step - loss: 0.2456 - binary_accuracy: 0.9008
[0.24556736648082733, 0.9008200168609619]
```

Figure 4.16 Evaluation of the CNN model

Here, compared to the basic feed-forward neural network previously presented, it is clear that the model is evaluated at about the same accuracy rate than its training, with no signs of significant overfitting. In less than optimal conditions for the embedding layer (where the embedding layer does not have as input dimension the whole vocabulary size), this model might perform worse than the RNN one, but here it held its ground as a viable method for text classification.

4.8-Data visualization

Data visualization aims to graphically describe the information of interest. Even though the accuracy and loss graphs for the neural networks have already been presented in the previous chapter for the purpose of clarity, there is more to data visualization than those. When working on larger projects, and especially when a team with diverse fields of expertise aims to build a deep learning model, data visualization comes in as an essential tool. When working with data, in the case of sentiment analysis and many other deep learning tasks, one of the challenges before optimizing the model is knowing how much and what kind of data one might need. In the case of sentiment analysis for Amazon reviews, or when using a dataset posted on Kaggle, the community is usually going to help in this matter of cutting out pieces of data, which makes the job easier for a data scientist or a machine learning engineer. But, when working in education or research, and your team aims to build a model for a custom dataset that has not been implemented in other instances, the scientists that are on the team might provide great insight into what should be kept and what should disappear. A scientist without experience in deep learning cannot understand the data unless presented in a clear and meaningful manner, hence the need for data visualization: the stage where the developer presents the data to an expert in order to receive feedback that will help him work on further optimization. Below I will present different ways of displaying your data using the seaborn and matplotlib libraries:

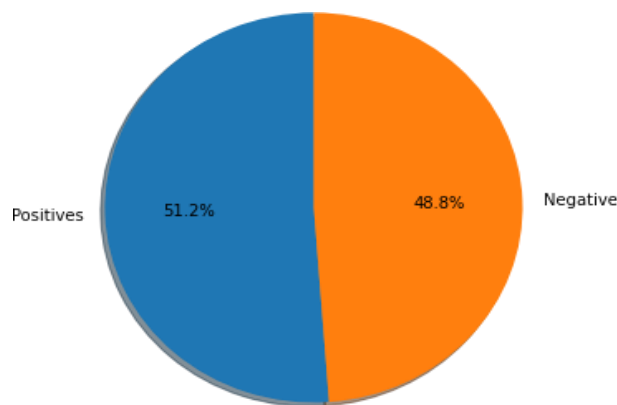


Figure 4.17 Pie chart representing the rapport between positives and negative for the training dataset

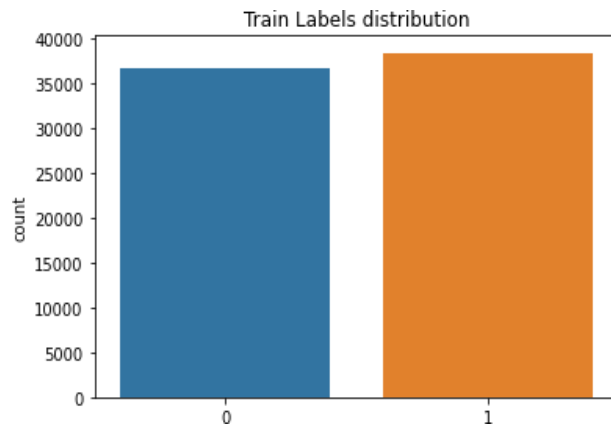


Figure 4.18 Count plot representing the rapport between positives and negatives for the training dataset

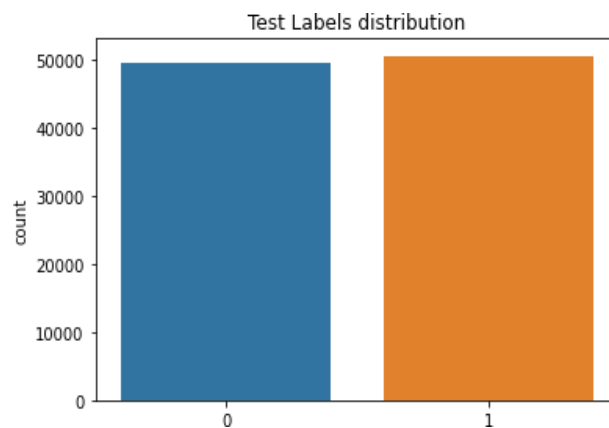


Figure 4.19 Count plot representing the rapport between positives and negatives for the testing dataset

```
{('solid', 1): 526, ('oyster', 1): 6, ('knife', 1): 121, ('made', 1): 2735, ('quick', 1): 506, ('work', 1): 5003, ('and', 1): 87981, ('left', 1): 649,
```

Figure 4.20 A frequency dictionary (shows the word and the number of times it appears for each sentiment category)

```
Counter({'the': 706949, 'and': 376797, 'this': 262943, 'for': 145044, 'that': 139061, 'you': 108567, 'not': 108476, 'but': 100876, 'book': 100197, 'with': 99027,
```

Figure 4.21 A word counter made using the Collections library

```
[['_PAD', 0, 0], ['_UNK', 0, 0], ['the', 149974, 156626], ['and', 87981, 74433], ['this', 55082, 58463], ['for', 31800, 29714], ['that', 28893, 31625],
```

Figure 4.22 A sentiment dictionary, build from the frequency dictionary

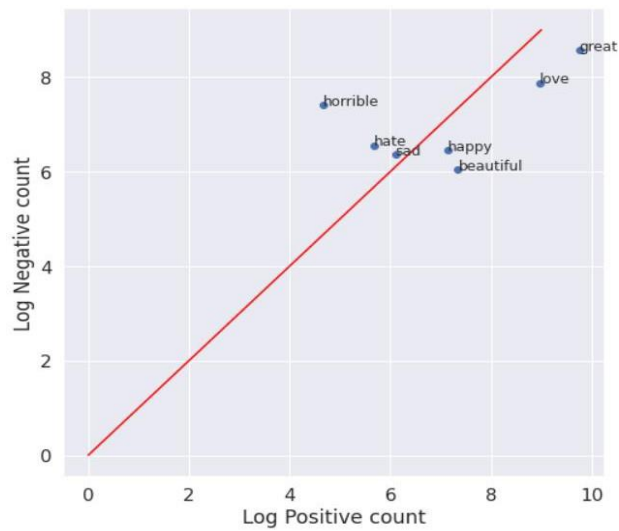


Figure 4.23 Plotting the sentiment of word (reduced dataset)

Note: The choice for taking the logarithmic values of the frequencies when representing the words was to help with the discrepancies between the sentiment numbers (a lot of apparitions of the word in reviews classified as positive while very few in negative ones).

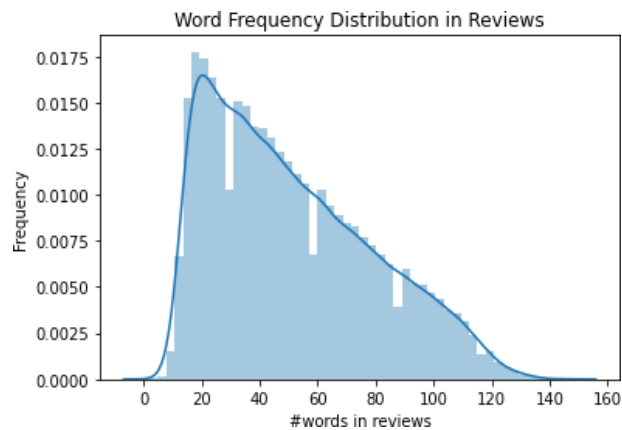


Figure 4.24 Word frequency distribution in the reviews

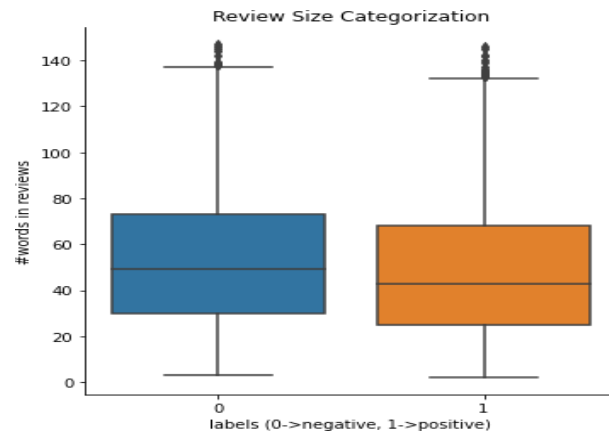


Figure 4.25 Box plot showing the number of words for each sentiment category

A box plot not only shows the average number of words in the reviews for every sentiment category, it also shows how many sentences have a word count above or below the average, which can be useful when deciding the max length of the sentences for word embeddings. In more scientific terms, the representation below the median is called the first quartile, the one above the third quartile (what percentage of reviews have words above and below the median line), and the whiskers at the top represent the maximum of words for every sentiment category.

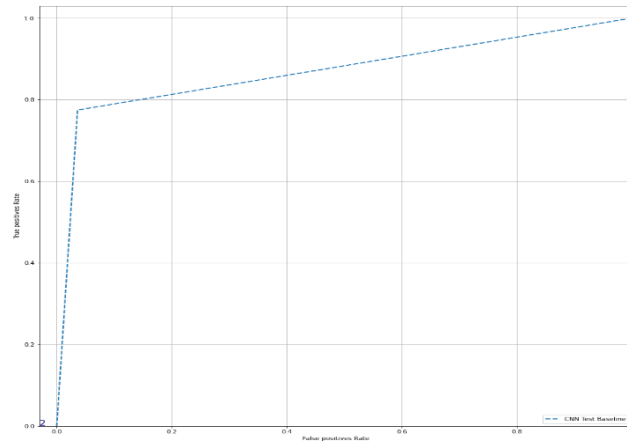


Figure 4.26 CNN range of characteristic curve

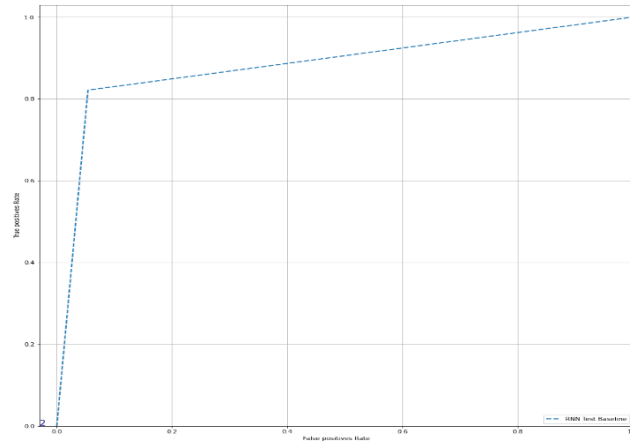


Figure 4.27 RNN range of characteristic curve

As previously stated, when using the binary accuracy metric, a way to analyze the performance of a model is to see the false and true positives is the range of characteristic curve. The model is better the closer the proximity of the curve to the Y axis. A model is not efficient at all if the curve has the function $Y=X$ (a center line). The Y axis is the True Positives axis and the X axis represents the False Positives axis.

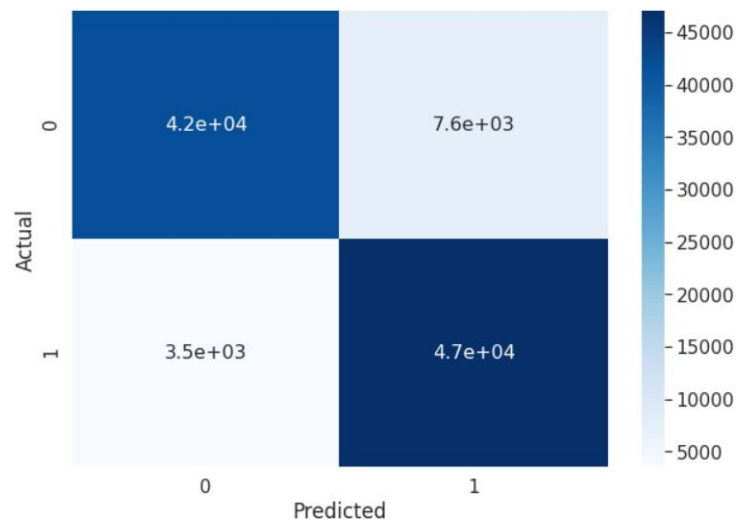


Figure 4.28 Confusion matrix for the RNN model

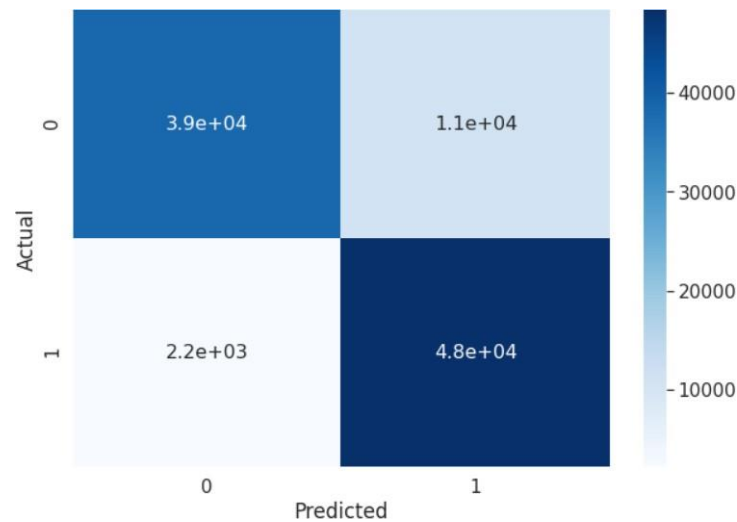


Figure 4.29 Confusion matrix for the CNN model

Besides the ROC curve, another way to verify the veracity of the results for the models is to compute the confusion matrix. This is easily performed using the scikit-learn library, combined with the plotting capabilities of the seaborn library. According to the confusion matrices, the RNN performed somewhat better than the CNN model, which shows that only evaluating the model is not enough in assessing your model's performance. Simply put, the diagonal represents the True positives and negatives, while the secondary diagonal represents the False positives and negatives. The confusion matrix, in this case, has been represented using a heatmap in shades of blue, the darker the shade of blue the more cases that particular category contains.

5. FUTURE IMPROVEMENTS

Although I feel confident that I have squeezed everything i could in terms of the models (obtained a good performance and analyzed the data in different ways), I plan on utilizing these models or at least the CNN and RNN ones for different applications. For some time now the models were used solely without interface, but nowadays models can be integrated with different types of applications for the purpose of deploying it for users. Moreover, not only can the models be saved for being used with apps, they can be saved for being used with other models, which is being extensively used in AI developed by Facebook, Google and other tech giants with the means of creating such colossal projects.

In text classification, some popular examples of apps that can be created with these models are:

- Android apps

Tensorflow has offered to the public a text classification demo app. With the right knowledge of android studio a model like this can be integrated using the tflite format. Models can either be saved as tflite when made by hand or can be converted to tflite using keras (therefore models can be taken from external sources).

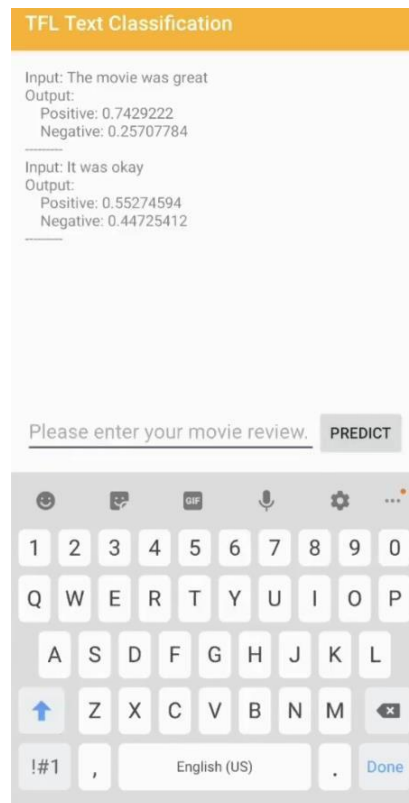


Figure 5.1 Android text classification app using Tensorflow Lite

Note: Here, we can observe that the output of the pretrained model has two dimensions, one for the positive label and one for the negative one. This can be done by using the scikit-learn library, more exactly the label binarizer function. Therefore, by making you output two dimensional your positive will have the form [0, 1] and the negative [1, 0] or the other way around, depending on your configurations.

- Web apps

Here, I have attempted to implement, to satisfy my curiosity, a simple flask app with the aim of integrating the RNN with a simple frontend (aesthetics was not a goal in this case). The app still needs debugging, but a connection to the model was clearly established. I have checked by modifying the labels with the label_binarizer to give a two-dimensional prediction, and then I went back to a one dimensional prediction, in order to see if the flask app actually catches the output from the model.

Text Classification

I hate this TV

Submit

Clear

[[0.4833170473575592]]

Figure 5.2 Simple web app for text classification using flask

Note: The output here is the output from the sigmoid activation function in the output layer, it gives a result between 0 and 1, above 0.5 meaning a positive sentiment and below a negative one, but in some cases one might need to consider another threshold for the positive and negative sentiments. Using flask and saving the model is not the only way to make a web app (i.e. using python only for development), tensorflow released tensorflow.js, which allows for integrating the models with javascript code. It is still experimented with and not a fully deployed technology.

6. CONCLUSIONS

In my opinion, this kind of thesis project is great for understanding not only sentiment analysis, but for understanding the difficulties and challenges that deep learning has when tackling a certain task. Having a proper dataset helps a lot especially with learning and education, and having access to so many communities eager to help is of great importance.

First of all, the theoretical part is not to be taken lightly, since it can aid the reader into understanding one's options, which in deep learning is quite important. There is nowadays an abundance of great tools that can aid a developer, some easier to use than others and some providing more performance. Besides understanding the concepts and the technicalities of deep learning (which are very important once one gets past the beginner stage), the tools you use can impact the size and the actual use of a project. In this thesis I have given the example of implementing your own functions as much as I possibly could, since understanding the concepts is mandatory before moving on to more advanced tasks which undeniably would require the use of more libraries. In this thesis the fastText library has been mentioned, which, once you understand the concepts and are interested in developing very fast models, is something to take into consideration.

Secondly, the practical part allows the reader to see for himself how the dataset is imported, cleaned and used for making predictions for text classification by using one implementation from each of the supervised deep learning approaches. As it has been shown, there are many ways to achieve a certain goal, but when the aim is maximum performance, there will always be a method better than all the others. The reader should not concern itself with achieving maximum performance in this case, on the contrary, the reader should observe how the models perform in different scenarios. For example, the CNN and the RNN models yielded more or less similar accuracy, yet the RNN, in the early stages of implementation, before optimization, might still yield some decent results, compared to a CNN that requires a lot of work for optimization.

As for myself, this thesis project has made me much more interested in this field than I was before. The beauty of getting into deep learning is understanding what can be achieved before knowing how to implement the solution. I can say with certainty that I will continue and improve my deep learning and data science skills because the independence that this field offers the learner is maybe unmatched in any other domain, you can literally become an "one man army" when building prediction tools for yourself, or even helping businesses as a freelancer. I highly recommend to anyone to use this still untapped potential that the deep learning field has

and improve their lives and others’.

References

- [1] E. Martin, „Jeff Bezos hasn’t always had the golden touch: Here’s what the Amazon founder was doing in his 20s,” <https://www.cnbc.com>, 2017. [Interactiv].
- [2] „History of Amazon: From Garage Startup to The Largest E-Commerce Marketplace,” [capitalism.com](https://www.capitalism.com/history-of-amazon/), [Interactiv]. Available: <https://www.capitalism.com/history-of-amazon/>.
- [3] J. Desjardins, Breaking Down How Amazon Makes Money, 2017.
- [4] B. M. A. G. H. S. Sowmya Vajjala, Practical Natural Language Processing, 2020.
- [5] „What are Morphemes?,” SEA - Supporting English Acquisition, [Interactiv]. Available: [https://www.rit.edu/ntid/sea/processes/wordknowledge/grammatical/what are](https://www.rit.edu/ntid/sea/processes/wordknowledge/grammatical/what%20are).
- [6] C. Español, „What are the different levels of NLP?,” [medium.com](https://medium.com/@CKEspanol/what-are-the-different-levels-of-nlp-how-do-these-integrate-with-information-retrieval-c0de6b9ebf61), 2017. [Interactiv]. Available: <https://medium.com/@CKEspanol/what-are-the-different-levels-of-nlp-how-do-these-integrate-with-information-retrieval-c0de6b9ebf61>.
- [7] „What Are Noun Phrases? (with Examples),” [grammar_monster.com](https://www.grammar-monster.com/glossary/noun-phrases.htm), [Interactiv]. Available: <https://www.grammar-monster.com/glossary/noun-phrases.htm>.
- [8] L. Hardesty, „Explained: Neural networks,” 2017. [Interactiv].
- [9] S. (. Kampakis, <https://thedata scientist.com>.
- [10] K. D. Foote, „A Brief History of Deep Learning,” dataversity.net. [Interactiv].
- [11] V. Bansal, „The Evolution of Deep Learning,” towardsdatascience.com. [Interactiv].
- [12] MLK, „Brief History of Deep Learning from 1943-2019,” machinelearningknowledge.ai, 2019. [Interactiv].
- [13] V. Bansal, „The Evolution of Deep Learning,” April 2005. [Interactiv].
- [14] M. Gurucharan, „Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network,” [upgrad.com](https://www.upgrad.com/blog/basic-cnn-architecture/), 2020. [Interactiv]. Available: <https://www.upgrad.com/blog/basic-cnn-architecture/>.
- [15] J. Brownlee, „How Do Convolutional Layers Work in Deep Learning Neural Networks?,” [machinelearningmastery.com](https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/), [Interactiv]. Available: <https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/>.
- [16] K. Srinivasa, Hybrid Computational Intelligence, Elsevier Inc., 2020.
- [17] Y. A. L. D. J. H. V. O. K. L. James J.Q., „Delay Aware Intelligent Transient Stability Assessment System,” www.researchgate.net, 2017. [Interactiv].
- [18] M. Phi, „Illustrated Guide to LSTM’s and GRU’s: A step by step explanation,”

towardsdatascience.com, 2018. [Interactiv].

[19] „when to use gru over lstm,” datascience.stackexchange.com. [Interactiv].

[20] A. Ralhan, „Self Organizing Maps,” medium.com, 2018. [Interactiv].

[21] D. Birla, „Basics of Autoencoders,” 12 March 2019. [Interactiv]. Available: <https://medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f>.

[22] M. T. J. Samaya Madhavan, „Deep learning architectures,” developer.ibm.com, 25 January 2021. [Interactiv].

[23] „Sentiment analysis,” komprehend.io, [Interactiv]. Available: <https://komprehend.io/sentiment-analysis>.

[24] F. Pascual, „Top Sentiment Analysis APIs (SaaS & Open Source),” monkeylearn.com, 2019. [Interactiv].

[25] „scikit-learn-architecture.md,” 2017. [Interactiv]. Available: <https://github.com/gopala-kr/10-weeks/blob/master/Projects-Blogs/06-ml-dl-frameworks/scikit-learn-architecture.md>.

[26] „nltk wiki,” [Interactiv]. Available: <https://github.com/nltk/nltk/wiki>.

[27] N. K. P. D. Krishna Bhavsar, Natural Language Processing with Python Cookbook: Over 60 recipes to implement text analytics solutions using deep learning principles, 2017.

[28] K. B. L. M. Alexia Audevart, Machine Learning Using TensorFlow Cookbook: Over 60 recipes on machine learning using deep learning solutions from Kaggle Masters and Google Developer Experts, 2020.

[29] A. Bansal, Advanced Natural Language Processing with TensorFlow 2, 2021.

[30] K. Dubovikov, „PyTorch vs TensorFlow — spotting the difference,” towardsdatascience.com, [Interactiv]. Available: <https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b>.

[31] F. Chollet, „Introduction to Keras,” stanford.edu, 2018. [Interactiv].

[32] H. Zikang, Y. Yong, Y. Guofeng și Z. Xinyu, „Sentiment analysis of agricultural product ecommerce review data based on deep learning,”

International Conference on Internet of Things and Intelligent Applications (ITIA), 2020.

[33] Y. L. J. W. a. R. S. S. L. Yang, „Sentiment Analysis for E-Commerce Product Reviews in Chinese Based on Sentiment Lexicon and Deep Learning,” IEEE Access, vol. 8, pp. 23522-23530, 2020, doi: 10.1109/ACCESS.2020.2969854., 2020.

[34] L. J. Y. J. M. F. Liu Y, „Sentiment analysis for e-commerce product reviews by deep learning model of Bert-BiGRU-Softmax,” Mathematical Biosciences and Engineering : MBE, 2020.

- [35] X. L. Qiwei Chen, „Research on Text Sentiment Analysis of E-Commerce Comments Based on Deep Learning,” International Journal of Social Science and Education Research.
- [36] H. W. Z. L. L. W. a. Z. T. Shaozhang Xiao, „Sentiment Analysis For Product Reviews Based on Deep Learning,” Journal of Physics: Conference series.
- [37] A. Bittlingmayer, „Amazon Reviews for Sentiment Analysis,” Kaggle, 2017. [Interactiv]. Available: <https://www.kaggle.com/bittlingmayer/amazonreviews>.
- [38] N. Subedi, „fastText under the hood,” towardsdatascience.com, 2018. [Interactiv]. Available: <https://towardsdatascience.com/fasttext-under-the-hood-11efc57b2b3>.
- [39] „Huffman Coding,” geeksforgeeks, [Interactiv]. Available: <https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/>.
- [40] E. G. A. J. T. M. Piotr Bojanowski, „Faster, better text classification!,” Facebook , 2016. [Interactiv]. Available: <https://research.fb.com/blog/2016/08/fasttext/>.
- [41] Facebook, [Interactiv]. Available: <https://ai.facebook.com/tools/fasttext/>.
- [42] J. Protasiewicz, „Python AI: Why Is Python So Good for Machine Learning?,” netguru.com, 2017. [Interactiv]. Available: <https://www.netguru.com/blog/python-machine-learning>.
- [43] A. Luashchuk, „Why I Think Python is Perfect for Machine Learning and Artificial Intelligence,” towardsdatascience.com, 2019. [Interactiv]. Available: <https://towardsdatascience.com/8-reasons-why-python-is-good-for-artificial-intelligence-and-machine-learning-4a23f6bed2e6>.
- [44] A. Kızrak, „Step-by-Step Use of Google Colab’s Free TPU,” HEARTBEAT, 2019. [Interactiv]. Available: <https://heartbeat.fritz.ai/step-by-step-use-of-google-colab-free-tpu-75f8629492b3>.
- [45] U. Malapaka, „Using TPUs on Google Colab with Keras,” towardsdatascience.com, 2020. [Interactiv]. Available: <https://towardsdatascience.com/using-tpus-on-google-colab-966239d24573>.
- [46] A. Geron, Hands-On Machine Learning with scikit-learn & Tensorflow, 2017.
- [47] „Ch. 17 - NLP and Word Embeddings,” kaggle.com, [Interactiv]. Available: <https://www.kaggle.com/jannesklaas/17-nlp-and-word-embeddings>.

- [48] J. Brownlee, „Deep Convolutional Neural Network for Sentiment Analysis (Text Classification),” machinelearningmastery.com, 2017. [Interactiv]. Available: <https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/>.
- [49] Y. Kim, „Convolutional Neural Networks for Sentence Classification,” Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014. [Interactiv]. Available: <https://www.aclweb.org/anthology/D14-1181>.
- [50] P. Patidar, „Introduction to Keras — Deep Learning Library,” 2019. [Interactiv].
- [51] K. D. Foote, „A Brief History of Deep Learning,” 2017. [Interactiv].
- [52] V. Gladchuk, „The History of Machine Learning: How Did It All Start?,” July 2016. [Interactiv].
- [53] A. Chaudhuri, Visual and Text Sentiment Analysis through Hierarchical Deep Learning Networks, 2019.
- [54] P. K. J.F.Pagel, Machine Dreaming and Consciousness, 2017.
- [55] P. Sarang, Artificial Neural Networks with TensorFlow 2: ANN Architecture Machine Learning Projects, 2021.
- [56] D. Rothman, Transformers for Natural Language Processing: Build innovative deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and more, 2021.
- [57] J. Brownlee, „How Do Convolutional Layers Work in Deep Learning Neural Networks?,” machinelearningmastery.com, 2019. [Interactiv]. Available: <https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/>.
- [58] M. Gungor, „Various Confusion Matrix Plots,” kaggle.com, [Interactiv]. Available: <https://www.kaggle.com/agungor2/various-confusion-matrix-plots>.
- [59] <https://www.kaggle.com/nabamitachakraborty/amazon-review-sentiment-analysis-reduced-dataset>

Annexes

Note: The code for data visualization was mostly inspired from Coursera's Natural Language processing, the parameters for the models were inspired from the contributions on Kaggle, and the data preprocessing from the book "Practical NLP"

1. Annex for Google colab code

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import bz2 # To open zipped files
import re # regular expressions
import os
import gc #for data extraction
import random
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download("wordnet")
lemmatizer = WordNetLemmatizer()
nltk.download("stopwords")
from nltk.corpus import stopwords
from nltk.classify import SklearnClassifier
import itertools
import codecs
from nltk.tokenize import word_tokenize
nltk.download('punkt')
import string
from collections import Counter
from torchtext.vocab import Vocab
import seaborn as sns
from nltk import pos_tag
from gensim.utils import simple_preprocess
nltk.download('cess_esp')
```

```

nltk.download('words')
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.python.keras.utils.np_utils import to_categorical

```

Note 1: Not all libraries have been used, they are here because since I was working in the cloud importing everything you think you might need is a good idea

The first model

```

train_file = bz2.BZ2File('/content/drive/MyDrive/dataset/train.ft.txt.bz2')
test_file = bz2.BZ2File('/content/drive/MyDrive/dataset/test.ft.txt.bz2')
train_file_lines = train_file.readlines()
#train_file_lines=random.sample(train_file_lines, 1000000)
test_file_lines = test_file.readlines()
#test_file_lines=random.sample(test_file_lines, 100000)
train_file_lines=train_file_lines[:100000]
test_file_lines = test_file_lines[:100000]
num_train = 10000
num_test = 10000

```

Note 2: 100k reviews for testing plus 100k for training and validation

```

train_file_lines = [x.decode('utf-8') for x in train_file_lines]
test_file_lines = [x.decode('utf-8') for x in test_file_lines]

```

The preprocessing part was done with the help of a notebook from NABAMITA CHAKRABORTY, with the contribution of preprocessing the reviews in Spanish too (Kaggle) [59]

```

def remove_underscores(sentence):
    sentence= sentence.replace("_", " ")
    return sentence

```

```

panish_words = set(nltk.corpus.cess_esp.words())
english_words = set(nltk.corpus.words.words())
stopwords_english = stopwords.words('english')
stopwords_spanish = stopwords.words('spanish')
stopwords = stopwords_english + stopwords_spanish

```

```

def remove_STOPWORDS(sentence):    #remove stop words and meaningless words
    new_sentence=""
    for word in sentence.split():
        word=word.lower()
        if word in english_words or word in spanish_words and word not in stopwords and word.isalpha():
            new_sentence=new_sentence+" "+word
    word_tokenize(new_sentence)
    return new_sentence

```

```

def lemmatize(sentence, min_word_length
    sent = ""
    for word in sentence.split():
        if len(lemmatizer.lemmatize(word))>=min_word_length:
            sent= sent+" "+lemmatizer.lemmatize(word)
    return(sent)

```

```

def split_label_review(dataa):
    #takes in list of raw data and returns list of string sentences and the list of their corresponding labels.
    labels = [0 if x.split(' ')[0] == '__label__1' else 1 for x in dataa]

```

```

reviews = [re.split("__label__[1|2]",str(lines))[1].strip().lower() for lines in dataa]
return(reviews,labels)

```

```
def preprocessed_data(raw_text):    #takes in list of raw data and returns list of processed string sentences and the list of their corresponding labels.
```

```
    reviews = [lemmatize(remove_stop_words(BeautifulSoup(re.sub(r'^\w\s|^https?:\w.*[\r\n]*\d+', "", remove_underscores(str(lines)).strip().lower()), "xml").text),3) for lines in raw_text]
```

```
    return(reviews)
```

```
def convert_labels(labels):
```

```
    refined_labels = [1 if x == '2' else 0 for x in labels]
```

```
    refined_labels = [0 if x == '1' else 1 for x in labels]
```

```
    return refined_labels
```

```
def split_train_into_train_validate(data_,validation_ratio):    #Takes in list of (raw or processed string) data and splits in train validation and returns in same format
```

```
    data=data_
```

```
    train_size = int(len(data)*(1-validation_ratio))
```

```
    random.shuffle(data)
```

```
    validate_samples= data[train_size:]
```

```
    data = data[:train_size]
```

```
    return(data,validate_samples)
```

```
def split_into_negative_positive(dataa):
```

```
    negative_reviews=[]    # list of all negative reviews
```

```
    positive_reviews=[]    # list of all positive reviews
```

```
    for lines in dataa:
```

```
        lines= str(lines).lower()
```

```
        x=re.findall("1|2", lines)[0]
```

```
        if x=="1":
```

```
            negative_reviews.append(lines)
```

```

elif x=="2":
    positive_reviews.append(lines)
return(negative_reviews, positive_reviews)

train_file_lines, validation_file_lines = split_train_into_train_validate(train_file_lines, 0.25)
negative_reviews, positive_reviews = split_into_negative_positive(train_file_lines)
negative_reviews, negative_labels = split_label_review(negative_reviews)
positive_reviews, positive_labels = split_label_review(positive_reviews)
negative_reviews = preprocessed_data(negative_reviews)
positive_reviews = preprocessed_data(positive_reviews)
#training_sentences, training_labels = split_label_review(train_file_lines)
final_labels = np.concatenate([positive_labels, negative_labels], 0)
final_reviews = positive_reviews + negative_reviews

negative_reviews_val, positive_reviews_val = split_into_negative_positive(validation_file_lines
)
negative_reviews_val, negative_labels_val = split_label_review(negative_reviews_val)
positive_reviews_val, positive_labels_val = split_label_review(positive_reviews_val)
negative_reviews_val = preprocessed_data(negative_reviews_val)
positive_reviews_val = preprocessed_data(positive_reviews_val)
#training_sentences, training_labels = split_label_review(train_file_lines)
final_labels_val = np.concatenate([positive_labels_val, negative_labels_val], 0)
final_reviews_val = positive_reviews_val + negative_reviews_val

negative_reviews_test, positive_reviews_test = split_into_negative_positive(test_file_lines)
negative_reviews_test, negative_labels_test = split_label_review(negative_reviews_test)
positive_reviews_test, positive_labels_test = split_label_review(positive_reviews_test)
negative_reviews_test = preprocessed_data(negative_reviews_test)
positive_reviews_test = preprocessed_data(positive_reviews_test)
final_labels_test = np.concatenate([positive_labels_test, negative_labels_test], 0)
final_reviews_test = positive_reviews_test + negative_reviews_test

```

```

def build_sentence_final_label(raw_data):

    refined_labels = [0 if x.split(' ')[0] == '__label__1' else 1 for x in raw_data]
    refined_labels = [1 if x.split(' ')[0] == '__label__2' else 0 for x in raw_data]
    refined_sentences = [x.split(' ', 1)[1][:-1].lower() for x in raw_data]

    labels_list = np.squeeze(refined_labels).tolist()
    freqs = []
    for sentence, labels in zip(refined_sentences, labels_list):

        pair = (sentence, labels)
        freqs += pair
    return freqs

print(build_sentence_final_label(train_file_lines[0:5]))

```

Note 2: The code for visualization was inspired by Coursera's Natural Language Processing specialization

```

words = Counter() #Dictionary that will map a word to the number of times it appeared in all the
training sentences
for sentence in final_reviews:
    #The sentences will be stored as a list of words/tokens

    for word in sentence.split():
        words.update([word.lower()]) #Converting all the words to lower case
for sentence in final_reviews_test:
    #The sentences will be stored as a list of words/tokens

    for word in sentence.split():

```



```
words.update([word.lower()]) #Converting all the words to lower case
```

```
print(words)
```

```
# Sorting the words according to the number of appearances, with the most common word being first
```

```
notok_words = sorted(words, key=words.get, reverse=True)
```

```
# Adding padding and unknown to our vocabulary so that they will be assigned an index
```

```
words = ['_PAD', '_UNK'] + notok_words
```

```
# Dictionaries to store the word to index mappings and vice versa
```

```
#word2idx = {o:i for i,o in enumerate(words)}
```

```
#idx2word = {i:o for i,o in enumerate(words)}
```

```
final_data = []
```

```
# loop through our selected words
```

```
for word in words:
```

```
    # initialize positive and negative counts
```

```
    pos = 0
```

```
    neg = 0
```

```
    # retrieve number of positive counts
```

```
    if (word, 1) in freqs:
```

```
        pos = freqs[(word, 1)]
```

```
    # retrieve number of negative counts
```

```
    if (word, 0) in freqs:
```

```
        neg = freqs[(word, 0)]
```

```

# append the word counts to the table
final_data.append([word, pos, neg])

print(final_data)

# select some words to appear in the report. we will assume that each word is unique (i.e. no duplicates)
keys = ['sad', 'happy', 'horrible', 'great', 'beautiful', 'hate', 'love']

# list representing our table of word counts.
# each element consist of a sublist with this pattern: [<word>, <positive_count>, <negative_count>]
data = []

# loop through our selected words
for word in keys:

    # initialize positive and negative counts
    pos = 0
    neg = 0

    # retrieve number of positive counts
    if (word, 1) in freqs:
        pos = freqs[(word, 1)]

    # retrieve number of negative counts
    if (word, 0) in freqs:
        neg = freqs[(word, 0)]

    # append the word counts to the table
    data.append([word, pos, neg])

data

```

```
fig, ax = plt.subplots(figsize = (8, 8))
```

```
# convert positive raw counts to logarithmic scale. we add 1 to avoid log(0)
```

```
x = np.log([x[1] + 1 for x in data])
```

```
# do the same for the negative counts
```

```
y = np.log([x[2] + 1 for x in data])
```

```
# Plot a dot for each pair of words
```

```
ax.scatter(x, y)
```

```
# assign axis labels
```

```
plt.xlabel("Log Positive count")
```

```
plt.ylabel("Log Negative count")
```

```
# Add the word as the label at the same position as you added the points just before
```

```
for i in range(0, len(data)):
```

```
    ax.annotate(data[i][0], (x[i], y[i]), fontsize=12)
```

```
ax.plot([0, 9], [0, 9], color = 'red') # Plot the red line that divides the 2 areas.
```

```
plt.show()
```

```
fig, ax = plt.subplots(figsize = (8, 8))
```

```
# convert positive raw counts to logarithmic scale. we add 1 to avoid log(0)
```

```
x = np.log([x[1] + 1 for x in final_data])
```

```
# do the same for the negative counts
```

```
y = np.log([x[2] + 1 for x in final_data])
```

```
# Plot a dot for each pair of words
```

```
ax.scatter(x, y)
```

```
# assign axis labels
```

```

plt.xlabel("Log Positive count")
plt.ylabel("Log Negative count")

# Add the word as the label at the same position as you added the points just before
for i in range(0, len(final_data)):
    ax.annotate(final_data[i][0], (x[i], y[i]), fontsize=12)

fig = plt.figure(figsize=(5, 5))

# labels for the two classes
labels = 'Positives', 'Negative'

# Sizes for each slide
sizes = [len(positive_reviews), len(negative_reviews)]

# Declare pie chart, where the slices will be ordered and plotted counter-clockwise:
plt.pie(sizes, labels=labels, autopct='% 1.1f%%',
        shadow=True, startangle=90)

# Equal aspect ratio ensures that pie is drawn as a circle.
plt.axis('equal')

# Display the chart
plt.show()

plot_1 = sns.countplot(final_labels)
plt.title("Train Labels distribution")
plt.show(plot_1)
plt.close()
plt.clf()

plot_2 = sns.countplot(final_labels_test)
plt.title("Test Labels distribution")
plt.show(plot_2)

```

```
plt.close()
```

```
plt.clf()
```

```
training_sentences, training_labels = split_label_review(train_file_lines)
```

```
training_sentences = preprocessed_data(training_sentences)
```

```
validation_sentences, validation_labels = split_label_review(validation_file_lines)
```

```
validation_sentences = preprocessed_data(validation_sentences)
```

```
testing_sentences, testing_labels = split_label_review(test_file_lines)
```

```
testing_sentences = preprocessed_data(testing_sentences)
```

```
train_sentences_size = list(map(lambda x: len(x.split()), training_sentences))
```

```
sns.distplot(train_sentences_size)
```

```
plt.xlabel("#words in reviews")
```

```
plt.ylabel("Frequency")
```

```
plt.title("Word Frequency Distribution in Reviews")
```

```
neg_mean_len = len(negative_labels)
```

```
pos_mean_len = len(positive_labels)
```

```
print(f"Negative mean length: {neg_mean_len:.2f}")
```

```
print(f"Positive mean length: {pos_mean_len:.2f}")
```

```
print(f"Mean Difference: {neg_mean_len - pos_mean_len:.2f}")
```

```
sns.catplot(x='labels', y='len', data=train_label_len, kind='box')
```

```
plt.xlabel("labels (0->negative, 1->positive)")
```

```
plt.ylabel("#words in reviews")
```

```
plt.title("Review Size Categorization")
```

```
tokenizer = Tokenizer(oov_token="<OOV>", num_words = len(words))
```

```
tokenizer.fit_on_texts(training_sentences)
```

```
word_index = tokenizer.word_index
```

```
print(word_index)
```

```
training_sequences = tokenizer.texts_to_sequences(training_sentences)
training_padded = pad_sequences(training_sequences, padding = 'post', truncating = 'post')
```

```
import numpy as np
training_padded = np.array(training_padded)
training_labels = np.array(training_labels)

tokenizer.fit_on_texts(validation_sentences)
validation_sequences = tokenizer.texts_to_sequences(validation_sentences)
validation_padded = pad_sequences(validation_sequences, padding = 'post', truncating = 'post')
```

```
del validation_sentences, validation_sequences
```

```
import numpy as np
validation_padded = np.array(validation_padded)
validation_labels = np.array(validation_labels)
vocab_size = len(words)
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, 10),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(200, activation='relu'),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='relu')])
```

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
model.summary()
```

```
tf.keras.utils.plot_model(model, "multi_input_and_output_model.png", show_shapes=True)
history = model.fit(training_padded, training_labels, epochs=100, validation_data=(validation_padded, validation_labels), verbose=1, callbacks = [tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=.25, patience=2, cooldown=0)])
```

)

```
import matplotlib.pyplot as plt
```

```
def plot_graphs(history, string):  
    plt.plot(history.history[string])  
    plt.plot(history.history['val_'+string])  
    plt.xlabel("Epochs")  
    plt.ylabel(string)  
    plt.legend([string, 'val_'+string])  
    plt.show()
```

```
plot_graphs(history, "accuracy")  
plot_graphs(history, "loss")
```

```
import tensorflow as tf  
from tensorflow import keras  
from tensorflow.keras.layers import *  
from sklearn.model_selection import train_test_split  
from tensorflow.keras.callbacks import ModelCheckpoint
```

```
tokenizer.fit_on_texts(testing_sentences)  
testing_sequences = tokenizer.texts_to_sequences(testing_sentences)  
testing_padded = pad_sequences(testing_sequences)
```

```
testing_padded = np.array(testing_padded)  
testing_labels = np.array(testing_labels)
```

```
del testing_sentences, testing_sequences
```

```
model_testing = model.evaluate(testing_padded, testing_labels)
```

```
prediction = model.predict(testing_padded)
```

```

print(prediction[0:15])
testing_labels[0:15]

n_train_samples = training_padded.shape[0]
n_val_samples = validation_padded.shape[0]
n_test_samples = testing_padded.shape[0]

print('We have %d TRAINING samples' % n_train_samples)
print('We have %d VALIDATION samples' % n_val_samples)
print('We have %d TEST samples' % n_test_samples)

```

Then the other two models:

```

train_file = bz2.BZ2File('/content/drive/MyDrive/dataset/train.ft.txt.bz2')
test_file = bz2.BZ2File('/content/drive/MyDrive/dataset/test.ft.txt.bz2')
train_file_lines = train_file.readlines()
#train_file_lines=random.sample(train_file_lines, 100000)
test_file_lines = test_file.readlines()
#test_file_lines=random.sample(test_file_lines, 100000)
##num_train = 10000
#num_test = 10000

train_file_lines = [x.decode('utf-8') for x in train_file_lines]
test_file_lines = [x.decode('utf-8') for x in test_file_lines]

train_file_lines,validation_file_lines = split_train_into_train_validate(train_file_lines, 0.25)

def getDocumentSentimentList(docs,splitStr='__label__'):
    docSentimentList=[]
    for i in range(len(docs)):
        #print('Processing doc ',i,' of ',len(docs))
        text=str(docs[i])

```



```

    #print(text)
    splitText=text.split(splitStr)
    secHalf=splitText[1]
    text=secHalf[2:len(secHalf)-1]
    sentiment=secHalf[0]
    #print('First half:',secHalf[0],'\nsecond half:',secHalf[2:len(secHalf)-1])
    docSentimentList.append([text,sentiment])

print('Done!!')
return docSentimentList

```

```

from sklearn.preprocessing import LabelBinarizer

```

```

docSentimentList=getDocumentSentimentList(train_file_lines[:100000],splitStr='__label__')
train_df = pd.DataFrame(docSentimentList,columns=['Text','Sentiment'])
train_df['Sentiment'][train_df['Sentiment']=='1'] = 0
train_df['Sentiment'][train_df['Sentiment']=='2'] = 1
#train_df['Sentiment'] = [1 if x == '2' else 0 for x in train_df['Sentiment']]
#train_df['Sentiment'] = [0 if x == '1' else 1 for x in train_df['Sentiment']]
print(train_df.head())

```

```

train_df['Sentiment'].value_counts()

```

```

train_df['Text'] = preprocessed_data(train_df['Text'])

```

```

print(train_df.head())

```

```

X_train=train_df['Text']
#X_train = preprocessed_data(X_train)
Y_train = train_df['Sentiment']
#Y_train=np.asarray(train_df['Sentiment'])#.tolist()
Y_train=Y_train.astype('int32')
#lb=LabelBinarizer(pos_label=1,neg_label=0)

```

```

#Y_train=lb.fit_transform(Y_train)

#Y_train=to_categorical(num_classes=2,y=Y_train)
#Y_train.shape

docSentimentList=getDocumentSentimentList(validation_file_lines[:100000],splitStr='__label__')
')
val_df = pd.DataFrame(docSentimentList,columns=['Text','Sentiment'])
val_df['Sentiment'][val_df['Sentiment']=='1'] = 0
val_df['Sentiment'][val_df['Sentiment']=='2'] = 1
#val_df['Sentiment'] = [1 if x == '2' else 0 for x in val_df['Sentiment']]
#val_df['Sentiment'] = [0 if x == '1' else 1 for x in val_df['Sentiment']]
print(val_df.head())

val_df['Text'] = preprocessed_data(val_df['Text'])
print(val_df.head())

val_df['Sentiment'].value_counts()

X_val=val_df['Text']

#Y_val=np.asarray(val_df['Sentiment'])#.tolist())
Y_val = val_df['Sentiment']
Y_val=Y_val.astype('int32')
#lb=LabelBinarizer(pos_label=1,neg_label=0)
#Y_val=lb.fit_transform(Y_val)

docSentimentList=getDocumentSentimentList(test_file_lines[:100000],splitStr='__label__')
test_df = pd.DataFrame(docSentimentList,columns=['Text','Sentiment'])
test_df['Sentiment'][test_df['Sentiment']=='1'] = 0
test_df['Sentiment'][test_df['Sentiment']=='2'] = 1
print(test_df.head())

test_df['Text'] = preprocessed_data(test_df['Text'])

```

```

print(test_df.head())
test_df['Sentiment'].value_counts()

X_test=test_df['Text']
#X_test = preprocessed_data(X_test)
Y_test = test_df['Sentiment']
#Y_test=np.asarray(test_df['Sentiment'])#.tolist()

Y_test=Y_test.astype('int32')
#lb=LabelBinarizer(pos_label=1,neg_label=0)
#Y_test=lb.fit_transform(Y_test)

#Y_test=to_categorical(num_classes=2,y=Y_test)
#Y_test.shape

max_length=100
vocab_size=10000
#embedding_dim=64
oov_tok="<OOV>"

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tok=Tokenizer(num_words=vocab_size,oov_token=oov_tok)
#vocab_size = len(words)

tok.fit_on_texts(X_train)
print('Toekizing...done')
seqs=np.array(tok.texts_to_sequences(X_train))
print('Sequencing...done')

```

```
X_train=pad_sequences(seqs,maxlen=100, padding = 'post', truncating= 'post')
print('Padding sequences...done')
```

```
word_index = tok.word_index
print(word_index)
```

```
print('Toekizing...done')
seqs=np.array(tok.texts_to_sequences(X_val))
print('Sequencing...done')
X_val=pad_sequences(seqs,maxlen=100, padding = 'post', truncating= 'post')
print('Padding sequences...done')
```

```
#tok.fit_on_texts(X_test)
print('Toekizing...done')
seqs=np.array(tok.texts_to_sequences(X_test))
print('Sequencing...done')
X_test=pad_sequences(seqs,maxlen=100, padding = 'post', truncating= 'post')
#X_test=pad_sequences(seqs,maxlen=100, padding = 'post', truncating = 'post')
print('Padding sequences...done')
```

```
def build_rnn_model():
    sequences = layers.Input(shape=(100,))
    embedded = layers.Embedding(vocab_size, 64)(sequences)
    x = layers.GRU(128, return_sequences=True)(embedded)
    x = layers.GRU(128)(x)
    x = layers.Dense(32, activation='relu')(x)
    x = layers.Dense(100, activation='relu')(x)
    predictions = layers.Dense(1, activation='sigmoid')(x)
    model = models.Model(inputs=sequences, outputs=predictions)
    model.compile(
        optimizer='rmsprop',
        loss='binary_crossentropy',
```

```

        metrics=["binary_accuracy"]
    )
    return model

rnn_model = build_rnn_model()

tf.keras.utils.plot_model(rnn_model, "multi_input_and_output_model.png", show_shapes=True)
adam=Adam(lr=0.0001)
earlyStopping=EarlyStopping(patience=3,monitor='binary_accuracy',min_delta=0.0001,verbose
=1)
callbackslist=[earlyStopping]
history = rnn_model.fit(X_train,Y_train,batch_size=128,epochs=2,verbose=1,callbacks=callback
slist, validation_data=(X_val,Y_val))

def plot_roc(name, labels, predictions, **kwargs):
    fp, tp, thresholds = sklearn.metrics.roc_curve(labels, predictions)
    plt.plot(fp, tp, label=name, linewidth=2, **kwargs)
    plt.xlabel('False positives Rate')
    plt.ylabel('True positives Rate')
    plt.xlim([-0.03, 1.0])
    plt.ylim([0.0, 1.03])
    plt.grid(True)
    thresholdsLength = len(thresholds)
    thresholds_every = 1000
    colorMap = plt.get_cmap('jet', thresholdsLength)
    for i in range(0, thresholdsLength, thresholds_every):
        threshold_value_with_max_four_decimals = str(thresholds[i])[:5]
        plt.text(fp[i] - 0.03, tp[i] + 0.001, threshold_value_with_max_four_decimals, fontdict={'siz
e': 15}, color=colorMap(i/thresholdsLength));

    ax = plt.gca()
    ax.set_aspect('equal')

```

```

import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (16, 16)

colors = plt.rcParams['axes.prop_cycle'].by_key()['color']
import sklearn
plot_roc("RNN Test Baseline", Y_test.argmax(axis=1), (rnn_model.predict(X_test)).argmax(axis
=1), color=colors[0], linestyle='--')
plt.legend(loc='lower right')

plt.plot(history.history['loss'], label='training data')
plt.plot(history.history['val_loss'], label='validation data')
plt.title('Loss for Text Classification')
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.legend(loc="upper left")
plt.show()

plt.plot(history.history['binary_accuracy'], label='training data')
plt.plot(history.history['val_binary_accuracy'], label='validation data')
plt.title('Accuracy for Text Classification')
plt.ylabel('Accuracy value')
plt.xlabel('No. epoch')
plt.legend(loc="upper left")
plt.show()

def build_cnn_model():
    sequences = layers.Input(shape=(100,))
    embedded = layers.Embedding(vocab_size, 64)(sequences)
    x = layers.Conv1D(64, 3, activation='relu')(embedded)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPool1D(3)(x)
    x = layers.Conv1D(64, 5, activation='relu')(x)
    x = layers.BatchNormalization()(x)

```

```

x = layers.MaxPool1D(5)(x)
x = layers.Conv1D(64, 5, activation='relu')(x)
x = layers.GlobalMaxPool1D()(x)
x = layers.Flatten()(x)
x = layers.Dense(100, activation='relu')(x)
predictions = layers.Dense(1, activation='sigmoid')(x)
model = models.Model(inputs=sequences, outputs=predictions)
model.compile(
    optimizer='rmsprop',
    loss='binary_crossentropy',
    metrics=['binary_accuracy']
)
return model

```

```

cnn_model = build_cnn_model()

```

```

tf.keras.utils.plot_model(cnn_model, "multi_input_and_output_model.png", show_shapes=True)
earlyStopping=EarlyStopping(patience=3,monitor='binary_accuracy',min_delta=0.0001,verbose
=1)
callbackslist=[earlyStopping]
history = cnn_model.fit(
    X_train,
    Y_train,
    batch_size=128,
    epochs=2,verbose=1,callbacks=callbackslist, validation_data=(X_val, Y_val)
)
cnn_model.evaluate(X_test,Y_test)

```

```

mpl.rcParams['figure.figsize'] = (16, 16)

```

```

colors = plt.rcParams['axes.prop_cycle'].by_key()['color']
plot_roc("CNN Test Baseline", Y_test.argmax(axis=1), (cnn_model.predict(X_test)).argmax(axi
s=1), color=colors[0], linestyle='--')
plt.legend(loc='lower right')
plt.plot(history.history['loss'], label=' training data')

```

```
plt.plot(history.history['val_loss'], label='validation data')
plt.title('Loss for Text Classification')
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.legend(loc="upper left")
plt.show()
```

```
plt.plot(history.history['binary_accuracy'], label=' training data')
plt.plot(history.history['val_binary_accuracy'], label='validation data')
plt.title('Accuracy for Text Classification')
plt.ylabel('Accuracy value')
plt.xlabel('No. epoch')
plt.legend(loc="upper left")
plt.show()
```

```
print(Y_test[:15])
rnn_model.predict(X_test[:15])
cnn_model.predict(X_test[:15])
print("Generate rnn predictions")
predictions_rnn = rnn_model.predict(X_test)
data_rnn = {'y_Actual': predictions_rnn,
            'y_Predicted': Y_test
            }
data_cnn = {'y_Actual': predictions_cnn,
            'y_Predicted': Y_test
            }
rnn_df = pd.DataFrame(data_rnn, columns=['Real', 'Predicted'])
cnn_df = pd.DataFrame(data_cnn, columns=['Real', 'Predicted'])
rnn_df['Predicted'] = np.where((rnn_df.Predicted >= 0.5), 1, rnn_df.Predicted)
rnn_df['Predicted'] = np.where((rnn_df.Predicted < 0.5), 0, rnn_df.Predicted)
cnn_df['Predicted'] = np.where((cnn_df.Predicted >= 0.5), 1, cnn_df.Predicted)
cnn_df['Predicted'] = np.where((cnn_df.Predicted < 0.5), 0, cnn_df.Predicted)

predictions_rnn = [1 if x >= 0.5 else 0 for x in predictions_rnn]
print(predictions_rnn[:15])
```



```

from sklearn.metrics import confusion_matrix
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

data = confusion_matrix(Y_test, predictions_rnn)
df_cm = pd.DataFrame(data, columns=np.unique(Y_test), index = np.unique(Y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16})# font size

predictions_cnn = [1 if x >= 0.5 else 0 for x in predictions_cnn]
print(predictions_cnn[:15])
data = confusion_matrix(Y_test, predictions_cnn)
df_cm = pd.DataFrame(data, columns=np.unique(Y_test), index = np.unique(Y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16})# font size

```

2. Annex for Flask

Html file

```

<!DOCTYPE html>
<html lang="en">

```

```

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <title>Title</title>
  <link rel="stylesheet" href="../static/tailwind.css">
  <script src="https://cdn.jsdelivr.net/gh/alpinejs/alpine@v2.x.x/dist/alpine.min.js"
defer></script>
</head>
<body class="min-h-screen flex flex-col">
  <header class="p-24 bg-gray-300 flex justify-center items-center">
    <h1 class="text-3xl font-bold">Text Classification App</h1>
  </header>
  <main class="p-10" x-data="{ ...data() }">
    <div class="container items-center px-5 py-12 text-blueGray-500 lg:px-20">
      <div class="p-2 mx-auto my-6 bg-white border rounded-lg shadow-xl lg:w-1/2">
        <div class="flex-grow p-6 py-2 rounded-lg">
          <form @submit.prevent="classifyText(text)" method="POST">
            <div class="my-6">
              <label>
                <span class="font-semibold text-xl">Text Classification</span>
                <textarea
                  x-model="text"
                  class="w-full px-4 py-2 mt-2 text-base text-black transition duration-
500 ease-in-out transform border-gray-100 rounded-lg bg-gray-50 focus:border-gray-500
focus:bg-white focus:outline-none focus:shadow-outline focus:ring-1 ring-offset-current ring-
offset-1 appearance-none autoexpand"
                  name="text"
                  placeholder="Enter text to classify..."
                  cols="30"
                  rows="5"
                >
              </textarea>
            </div>
          </div>
        </div>
      </div>
    </div>
  </main>

```

```

    <div class="space-x-4 flex">
      <button type="submit"
        class="px-16 py-2 text-center text-white transition duration-500 ease-in-
out transform bg-blue-600 border-blue-600 rounded-md focus:shadow-outline focus:outline-
none focus:ring-2 ring-offset-current ring-offset-2 hover:bg-blue-800">
        Submit
      </button>
      <button class="px-16 py-2 transition text-center duration-500 ease-in-out
transform bg-gray-200 border-gray-200 rounded-md focus:shadow-outline focus:outline-none
focus:ring-2 ring-offset-current ring-offset-2 hover:bg-gray-300"
        @click="text = ''
        type="button"
      >
        Clear
      </button>
    </div>
  </form>
  <template x-if="classification">
    <div class="pt-5" x-text="classification"></div>
  </template>
</div>
</div>
</div>
</main>
<script>
  function data() {
    return {
      text: "",
      classification: "",
      classifyText
    }
  }

  function classifyText(text) {
    fetch('/classify', { method: 'POST', body: JSON.stringify({ text }), headers: {

```

```

        'Content-Type': 'application/json'
    }, })
    .then(response => response.json())
    .then(data => {
        this.classification = data.response;
    });
}
</script>
</body>
</html>

```

APP file

```

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from flask import Flask
from flask import render_template
from flask import jsonify
from flask import request
from keras.models import load_model
import json

```

```

app = Flask(__name__)
app.config['TEMPLATES_AUTO_RELOAD'] = True

```

```

@app.route('/')
def index():
    return render_template('index.html')

```

```

@app.route('/classify', methods=['POST'])

```

```
def classify():
    if request.method == "POST":
        text = request.json['text']

        tokenizer = Tokenizer(num_words=len(text))
        maxlen = 100
        instance = tokenizer.texts_to_sequences(text)

        flat_list = []
        for sublist in instance:
            for item in sublist:
                flat_list.append(item)

        flat_list = [flat_list]

        instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)

        model = load_model('./models/rnn_model.h5')
        response = model.predict(instance)

        return jsonify({'response': json.dumps(response.tolist())})
    return render_template('index.html')
```

