ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ ΠΡΟΓΡΑΜΜΑ ΠΡΟΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάλυση Συναισθήματος σχετικά με την

Μετανάστευση στην Ελληνική Κοινωνία

**Targeted Sentiment & Emotion Analysis**

**On Immigration in Greek Society**

Προπτυχιακή Εργασία

του

Φλεβάρη Θεόδωρου

(ΑΜ: dai16013)

Θεσσαλονίκη, 06/2019

Επιβλέπων Καθηγητής

Μαργαρίτης Κωνσταντίνος

****

Table of Contents

Abstract

1. Introduction …………………………………………………………………………… 1

1.1 Machine Learning…………………………………………………………… 2

1.2 Natural Langauge Processing……………………………………………… 4

1.3 Sentiment Analysis…………………………………………………………. 5

1.4 Social Background on Immigration..…………………………………………7

2. Methodology ………………….………………………………………………………. 9

2.1 Data Collection ……………………………………………………………… 10

2.2 Tweet Preprocessing…………………………………………………………13

2.3 Polarity-based Classification………………………………………………. 21

2.4 Lexicon-based Emotion Extraction………………………………………… 25

3. Deep Learning Approach….………………………………………………………….. 31

3.1 Word Vectorization…………………………………………………………... 32

3.2 Types of Neural Networks for Classification………………………………. 35

3.3 Emotion Extraction using CNN......…………………………………………..39

**Abstract**

The following paper is in regard to a certain sector of Machine Learning, called Natural Language Processing. More specifically, it handles the topic of Sentiment Analysis, a method used in the field of Machine Learning to determine the sentiment one has towards a topic. For this research, data from Twitter have been used to specify the stance Greeks have towards immigration in an effort to generalize whether the society is accepting of the immigrants or not. Instead of focusing on positive & negative sentiment, this thesis also explores a multiclass emotion analysis which classifies the dataset into 8 emotion categories including: anger, anticipation, trust, disgust, joy, sadness, fear and surprise. The data has been accessed through Twitter’s own API, using the Python library Tweepy and has been processed using libraries such as SpaCy and Polyglot with the end result being a model that can accurately classify texts into different sentiment categories.

All code can be found here: [GitHub](https://github.com/DumbRookie/thesis)

Keywords: *Immigration, Sentiment Analysis, Opinion Mining, Machine Learning, Language Models, Text Classifiers, Emotion Extraction, Neural Networks.*

1. **Introduction**

Ever since its creation the Web has been a place where users can exchange information and learn about everything that can be indexed. Having a vast volume of data, people have started turning to automatic ways to interpret and understand it. At first, the available data was collected by organizations and distributed in a structured form. Structured data is data that adheres to a pre-defined data model and is therefore straightforward to analyze, such as SQL databases. After the rise of social media, most of the data has been user-produced, as millions of posts are being uploaded daily, with Twitter alone boasting up to 500 million tweets per day. This shows that social media can be a gold mine for analysts who want to come closer to understanding and making sense of natural language in a manner that is valuable to organizations.

Automatically mining and organizing opinions from heterogeneous information sources is very useful for individuals, social scientists and even companies. From finding hidden knowledge to discovering a customer’s preferences or emotions, text mining has been beneficial to all kinds of sectors. Businesses use data and text mining to analyze customer and competitor data to improve competitiveness; the pharmaceutical industry mines patents and research articles to improve drug discovery; within academic research, mining and analytics of large datasets are delivering efficiencies and new knowledge in areas as diverse as biological science, particle physics and media and communications.

At this point, it is important to distinguish between two main terms, text analytics and sentiment analysis, that stem from the field of text mining. Text Analytics, in a nutshell, can be described as the process of analyzing unstructured text, extracting relevant information, and transforming it into useful business intelligence. Sentiment Analysis, on the other hand, determines if an expression is positive, negative, or neutral, and to what degree. Simply put, text analytics focuses on what is written about most and shows trends. Sentiment Analysis gives meaning to words and enables users to gauge the severity of the feedback based on positive, negative and neutral word usage as well as the sentiment associated with commonly used words.

Pang, Lee, and Vaidyanathan (2002) classified documents by overall sentiments instead of topics. Dave’s (2003) and Hu’s (2004) research works focus on extracting opinions of reviews. However, a document consists of various opinions. Riloff and Wiebe (2003) distinguish subjective sentences from objective ones. Kim and Hovy (2004) propose a sentiment classifier for English words and sentences, which utilizes thesauri. However, template-based approach needs a professionally annotated corpus for learning, and words in thesauri are not always consistent in sentiment. Because no queries are posed beforehand, detecting opinions is similar to the task of topic detection at sentence level. For this paper, the analysis focuses on Tweets generated from random users and are considered sentences.

Below we will examine some basic terms, which are essential to this thesis.

* 1. **Machine Learning**

Whereas Artificial Intelligence (AI) is a broader concept of machines being able to carry out tasks in a manner that would be considered human-like or smart, Machine Learning (ML) is a current application of the concept of AI, where machines have access and are “fed” data from various external sources and are programmed to learn from those data themselves using a variety of training methods.

As technology has progressed, AI and Machine Learning have taken different forms. Devices designed to act intelligently are often classified into two categories, applied or general systems.

Applied AI is the more common of the two. These systems are designed to use knowledge the extract from data to perform action such as maneuvering autonomous vehicles or Google’s DUPLEX system, a system that can mimic human conversation though the telephone. General AIs are system or devices that can supposedly handle any intellectual task, such as reason, use strategy, make judgments under uncertainty and represent commonsense knowledge. These are far less common, but show much promise and this area is where most advancements are happening today.

Historically, Machine Learning emerged in 1959 when a professor named Arthur Samuel noticed that instead of declaring what computers needed to know about the world, it would be possible to teach them to learn for themselves. The Samuel Checkers-playing Program was among the world's first successful self-learning programs, and as such a very early demonstration of the fundamental concept of artificial intelligence (AI). After that realization, many people started researching the topic, as did Mathematician Alan Turing who wrote a paper on the notion that machines can simulate human behaviors and skills such as playing chess. This is a goal which has been set for developers and researchers of Machine Learning and AI called the Turing Test.

The Turing test is a central, long term goal for AI research – to be able to build a computer that can sufficiently imitate a human to the point where a suspicious judge cannot tell the difference between human and machine. From its inception it has followed a path similar to much of the AI research. Initially, it looked to be difficult but possible, once hardware technology reached a certain point, only to reveal itself to be far more complicated than initially thought with progress slowing to the point that some wonder if it will be reached. Despite decades of research and great technological advances the Turing test still sets a goal that AI researchers strive toward.

Along with technological progress come different approaches to the goal of Machine Learning. Modern day Machine Learning has tried to imitate the human brain with the creation of Artificial Neural Networks. Artificial Neural Networks consist of multiple layers of nodes, called neurons. Each neuron is responsible for receiving data from inputs, either external or from another neuron and then executing a function called activation function. The input of the activation function is the weighted sum of all inputs. The activation function can be any function, but the most commonly used ones are the:

* sigmoid ( )
* hyperbolic tangent ( )
* ReLu ( )

The neuron’s goal is to adjust the weights of each input based on lots of examples of inputs and outputs. This means, that the network is being trained with data that have a known output, so it knows what to expect and adjust the weights of the inputs if the prediction deviates from the correct result and then to test it, the network is fed new unseen data. The most common training method is one called k-fold Cross-Validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. The procedure it follows usually is to:

1. Shuffle the dataset randomly
2. Split the dataset into k groups
3. For each unique group:
4. Take the group as a hold out or test data set
5. Take the remaining groups as a training data set
6. Fit a model on the training set and evaluate it on the test set
7. Retain the evaluation score and discard the model
8. Summarize the skill of the model using the sample of model evaluation scores

This form of training using Artificial Neural Networks has been very effective in creating all sorts of smart innovative systems that offer many possibilities based around ML and neural networks. In recent years, though, an idea has also emerged that we should be able to communicate and interact with electronic devices and digital information, as naturally as we would with another human being. To this end, another field of AI – Natural Language Processing (NLP) – has become a source of innovation in recent years, and one which is heavily reliant on ML. NLP applications attempt to understand natural human communication, either written or spoken, and communicate in return with us using similar, natural language. ML is used here to help machines understand the vast nuances in human language, and to learn to respond in a way that can beneficial and easily comprehensible.

* 1. **Natural Language Processing**

Natural Language Processing (NLP) is a cross section a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human languages, in particular how to program computers to process and analyze large amounts of natural language data.

NLP applications have surfaced only recently because of the challenges traditional programming poses. Computers traditionally require humans to express concepts to them in a programming language that are precise, unambiguous and highly structured, or through a limited number of clearly enunciated voice commands. Human speech, however, is not always precise -- it is often ambiguous and the linguistic structure can depend on many complex variables, including slang, regional dialects, social context and vocal nuances.

Early approaches to NLP involved a more rules-based approach, where simpler machine learning algorithms were told what words and phrases to look for in text and give specific responses when those phrases appeared. Current approaches to NLP are based on machine learning, a type of AI that examines and uses patterns in data to improve a program's understanding. Deep learning models require massive amounts of labeled data to train on and identify relevant correlations, and assembling this kind of big data set is one of the main hurdles to NLP currently. Despite the challenges, machine learning is a more flexible, intuitive approach in which algorithms learn to identify speakers' intent from many examples, almost like how a child would learn human language.

This sector has proven to have many applications such as the following:

* Organizations can determine what customers are saying about a service or product by identifying and extracting information in sources like social media. This sentiment analysis can provide a lot of information about customers choices and their decision drivers.
* An inventor at IBM developed a cognitive assistant that works like a personalized search engine by learning all about you and then remind you of a name, a song, or anything you can’t remember the moment you need it to.
* Companies like Yahoo and Google filter and classify your emails with NLP by analyzing text in emails that flow through their servers and stopping spam before they even enter your inbox.
* To help identifying fake news, the NLP Group at MIT developed a new system to determine if a source is accurate or politically biased, detecting if a news source can be trusted or not.

Below we will examine the first bullet which is NLP in the scope of Sentiment Analysis.

* 1. **Sentiment Analysis**

Apart from understanding words, sentiment is also very important. Emotions are essential to effective communication between humans, so if we want machines to handle texts in the same way, we need teach them how to detect emotions and classify text as positive, negative or neutral. That's where sentiment analysis comes into play. It's the automated process of understanding an opinion about a given subject from written or spoken language. For example, by using sentiment analysis companies are able to flag complaints or urgent requests, so they can be dealt with immediately. Other uses of sentiment classifiers include assessing brand reputation, carrying out market research, improving products with customer feedback and analyze text of social interest.

Since the text to be used is unprocessed and unstructured, one common practice is to apply some techniques of preprocessing before starting to create a model for NLP – in this case Sentiment Analysis. These techniques differ depending on the source of data. If the data derives from social media, then the first step would be to clean the text. Cleaning the text includes removing URLs, hashtags (i.e. #sad) or mentions (i.e. @DonaldTrump). Furthermore, tabs and line breaks should be replaced with a blank and quotation marks with apexes. After this step, all the punctuation is removed, except for apexes, because they are part of grammar constructs such as the genitive. The last step is to convert many types of emoticons into tags that express their sentiment (i.e. “:)” → smile happy) or all the text is converted to lower case, and extra blank spaces are removed.

In all cases of texts, dealing with negations (like “not good”) is a critical step in Sentiment Analysis. A negation word can influence the tone of all the words around it, and ignoring negations is one of the main causes of misclassification. In this phase, all negative constructs (can’t, don’t, isn’t, never etc.) are replaced with “not”.

In general, another common operation is to remove the vowels repeated in sequence at least three times, because by doing so the words are normalized: for example, two words written in a different way (i.e. cooooool and cool) will become equals. This method is called normalization and it goes hand-in-hand with stemming. Stemming techniques put word variations like “great”, “greatly”, “greatest”, and “greater” all into one bucket, effectively decreasing entropy and increasing the relevance of the concept of “great”. Entropy, in this case, references to information entropy which is the average rate at which information is produced by a stochastic source of data. In other words, stemming allows us to consider nouns, verbs and adverbs that have the same radix in the same way.

Apart from stemming, another method is lemmatization. Lemmatization has the objective of reducing a word to its base form and grouping together different forms of the same word. For example, verbs in past tense are changed into present (e.g. “went” is changed to “go”) and synonyms are unified (e.g. “best” is changed to “good”), hence standardizing words with similar meaning to their root. Although it seems closely related to the stemming process, lemmatization uses a different approach to reach the root forms of words.

Furthermore, it is important to remove stop words. Stop words are words which are filtered out in the preprocessing step. These words are, for example, pronouns, articles, etc. It is important to avoid having these clutter words within the classifier model, because they can lead to a less accurate classification.

Lastly, there is tokenization, which is the process of segmenting running text into sentences and words. In essence, it’s the task of cutting a text into pieces called tokens, and at the same time throwing away certain characters, such as punctuation.

After having a clean string of words, it is essential to map them into data which can be easily processed. To do that, words are converted into word embeddings. Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. Surely, this representation has drawbacks with one of the main limitations being that words with multiple meanings are conflated into a single representation.

One of the most researched models of word embeddings is Google’s own Word2Vec. Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located close to one another in the space.

Having had processable data, it is easy to apply some models or put the data into a Neural Network to train it to classify text strings into positive, negative or neutral accompanied by a perceptual probability. This paper will undertake the task of building such models regarding the topic of immigration. Immigration has been in the spotlight of news for a long time and is still a debatable topic worth exploring, especially in the country of Greece, which serves as a prominent host country amidst the European migration crisis.

* 1. **Social Background on Immigration**

Humans, as a race, have been migrating to different places ever since their first appearance. From finding viable soil to grow crops, having enough water supply to support basic survival needs, relocating in dire times has always been an essential part of thriving.

Nowadays, we speak more of immigration rather than migration. Firstly, to migrate entails the movement of people from one place to another with the intentions of settling, permanently or temporarily at a new geographic location. Whereas, immigration, as a concept, is the international movement of people to a destination country where they are not natives or where they do not possess citizenship in order to settle or reside there, usually due to better conditions of living.

In modern times, the reasoning behind migrating has evolved and become more perplexed. Research has shown that it is able to divide the reasons of migrating into two main classes. On one hand are the push factors, such as reasons for leaving a place, which is called emigrating, because of certain difficulties like food shortage, war, flood, calamities, etc. On the other hand, there are the pull factors which are reasons for moving into a place, which is called immigrating, because of an aspiration, dream, or something desirable like abundance in food supply, a better climate, more freedom, etc. Due to the circumstances of the present age, most people decide to set a home in a new country mainly, because of war in their own homeland or better living conditions and job opportunities in the destination country.

Lately, there has been a rise in immigrants in the European continent so profound that is has created what is called the European migrant crisis. The European migrant crisis, also known as the refugee crisis is a period beginning in 2015 which is characterized by high numbers of people arriving in the European Union (EU) from across the Mediterranean. Immigrants from outside Europe include asylum seekers and economic migrants. The term "immigrant" is used by the European Commission to describe a person from a non-EU country establishing his or her usual residence in the territory of an EU country for a period that is, or is expected to be, at least twelve months. Most of the migrants came from Muslim-majority countries in regions south and east of Europe, including the Greater Middle East and Africa. Usually they are refugees, meaning that they have no option but to flee their countries of origins, mostly due to chemical or typical warfare, poverty or largescale extremist behaviors.

Due to its geographic location, Greece is usually the receiver of most immigrants and refugees during this crisis. Migrants arrive from the Middle East making the 6-kilometre (4 mi) water crossing to the Greek islands of Chios, Kos, Lesbos, Symi and other islands which are close to Turkey and are thus a quick and easy access border into Europe. As of June 2015, 124,000 migrants had arrived into Greece, a 750 percent increase from 2014, mainly refugees stemming from the wars in Syria, Iraq, and Afghanistan. Since most of the immigrants sail to Greece with rafts or very small boats, many accidents and deaths have occurred and the Greek government and coast guard are making continuous efforts to rescue most of the victims at open waters.

Historically, Greece has been the destination for many immigrants throughout the years, because of the important vantage point of being a member of the EU. This means that there are legislative safeguards such as requests for asylum that are more transparent and ones that focus on human rights. Immigration to Greece percentage of foreign populations in Greece is 7.1% in proportion to the total population of the country. As of 2012, Albanian migrants have constituted some 55–60% of the immigrant population. More recent immigrant groups, from the mid-1990s on, consist of Asian nationalities—especially Pakistani and Bangladeshi—with more recent political asylum or illegal migration flows through Turkey of Afghans, Iraqis, Somali and others.

Since the 1990s, increases in such flows have led to the emergence of immigration as an increasingly important political issue in Greece. Greece has had problems with illegal immigration, many of whom transit through Turkey. Greek authorities believe that 90% of illegal immigrants in the EU enter through Greece, many fleeing because of unrest and poverty in the Middle East and Africa, because it is used as a gateway to the Schengen Area by flows of illegal immigrants. To this day there are more than 600,000 legal immigrants living in Greece and even more undocumented ones.

In 1989, Greeks were presented by the Eurobarometer as the people most tolerant of foreigners in all the EU. However, political instability and warfare in the Balkans in the early 1990s made the Greeks begin to worry about the conflict nearing their own borders alongside the much-debated topic of North Macedonia’s sovereignty over the Greek state of Makedonia. These political developments revived many feelings of nationalism in Greece, and the influx of immigrants in the 1990s challenged the collective image of Greece as an ethnically homogeneous society. The Greek nation is strongly tied to an ethnically based identity that centers on common ancestry, language, and Orthodox religion. The rise of immigration in the 1990, then, was seen as a threat to the cultural and ethnic purity and authenticity of the Greek nation.

Regardless of social unrest, in 2016, Greek volunteers were awarded the Nansen Refugee Award by the United Nations High Commissioner for Refugees (UNHCR). Volunteers of "The Hellenic Rescue Team" were awarded for their tireless voluntary efforts to aid refugees arriving in Greece during the European refugee crisis.

Between the two images of Greek society, on the one side, helping and caring for immigrants and on the other, judging and discarding them as non-Greeks, it is very intriguing to discover what the society actually thinks about the topic through analysis of texts posted by simple citizens. To determine that, I chose to analyze Tweets in the period of September of 2019 up until August 2020 with Sentiment Analysis techniques.

**2. Methodology**

For this thesis the two main research questions posed were:

1. *What do textual data reveal about the viewpoint that Greeks have toward immigration as of late 2019*?
2. *Is an automated system able to efficiently and effectively recognize human emotion and classify it into more distinct categories rather than just positive and negative?*

To answer these questions data has been collected from a famous social media platform, Twitter. This source was chosen as ideal, because of its vast content and its high rate of new daily posts, which would be useful in the effort to create classification software. The tweets posted, also, reflected the unedited, unfiltered and unstructured perception of typical citizens and some news sources about the topic at hand. Therefore, the method used to collect data would be considered quantitative. Quantitative research is expressed in numbers and graphs. It is used to test or confirm theories and assumptions and can be used to establish deduce facts about a topic.

The data will be processed and used to create classification models. The methods used for this will be as follows:

* Preprocessing: tokenization, lemmatization, removal of stop words etc.
* Polarity lexicons & Emotion lexicons for sentence-level analysis
* Vector representation of tokens
* Neural Networks for classification

In the next chapter we will examine the various aspects of the process in detail.

**2.1 Data Collection**

To collect data from Twitter, one must use the API provided by the company. To have access to the API some specific steps must be followed. Firstly, one needs to have a Twitter account, and then go to [developer.twitter.com](http://developer.twitter.com/). Then, an application must be filled to determine the intent one has. Twitter doesn’t let anyone have access to the data. For the purposes of this paper, the application was to use the data for academic purposes, specifying the topic of the research and also the home institution and some personal information. The time of response on Twitter’s end usual is less than two days and, even if the application is lackluster, a representative will request more information. After clearing access, one has to “create an app”, which is a named project that has its unique access keys.

The keys are divided into 4 categories. The consumer key, the consumer secret key, the access token and the secret access token. The consumer key is for your application and access tokens are for end users in your application's context. If you want to call in just the application context, then consumer key is adequate. You'd be rate limited per application and won't be able to access user data that is not public.

One can call the API either from the browser or through a programming language. For the collection of the data of this paper I used Python’s library called tweepy, which is a wrapper of Twitter’s own API. There were two scripts that were developed to scrape Twitter for relevant tweets, the first one did a simple search with a keyword, while the second was opened up a stream for live tweets to be recorded.

To specify here is the snippet of the first script:



***Image 1: Twitter Search Script***

On the first line, we import the necessary library tweepy, to make calls to the API. Then, we specify all four necessary keys, to allow authentication to be successful. The variable “auth” signifies the authentication object which is necessary for signing in to the API. We get prepare the “auth” object by setting its tokens through tweepy’s set\_access\_token method and then we create an API object that will be used as the connecting link between the script and Twitter’s servers.

The main method used was the “search” function. The search method takes many arguments, but for this use case only some were useful. The arguments were specified before calling the method and these include the following variables:

* *query: the keyword used by the method to filter through the tweets.*
* *new\_q: the same keyword but this time ignoring every tweet that is a retweet, to avoid duplicates.*
* *twpp: the number of tweets returned by the search method, usually has a limit posed by the API, but the 200 value signifies to return as many as possible.*
* *language: the specific language which we want results for, usually useful for languages with the same alphabet i.e. gift, which is present in English, but poison is Swedish.*
* *tweet\_mode: is set to ‘extended’ to show full tweets and not some characters followed by “…”.*

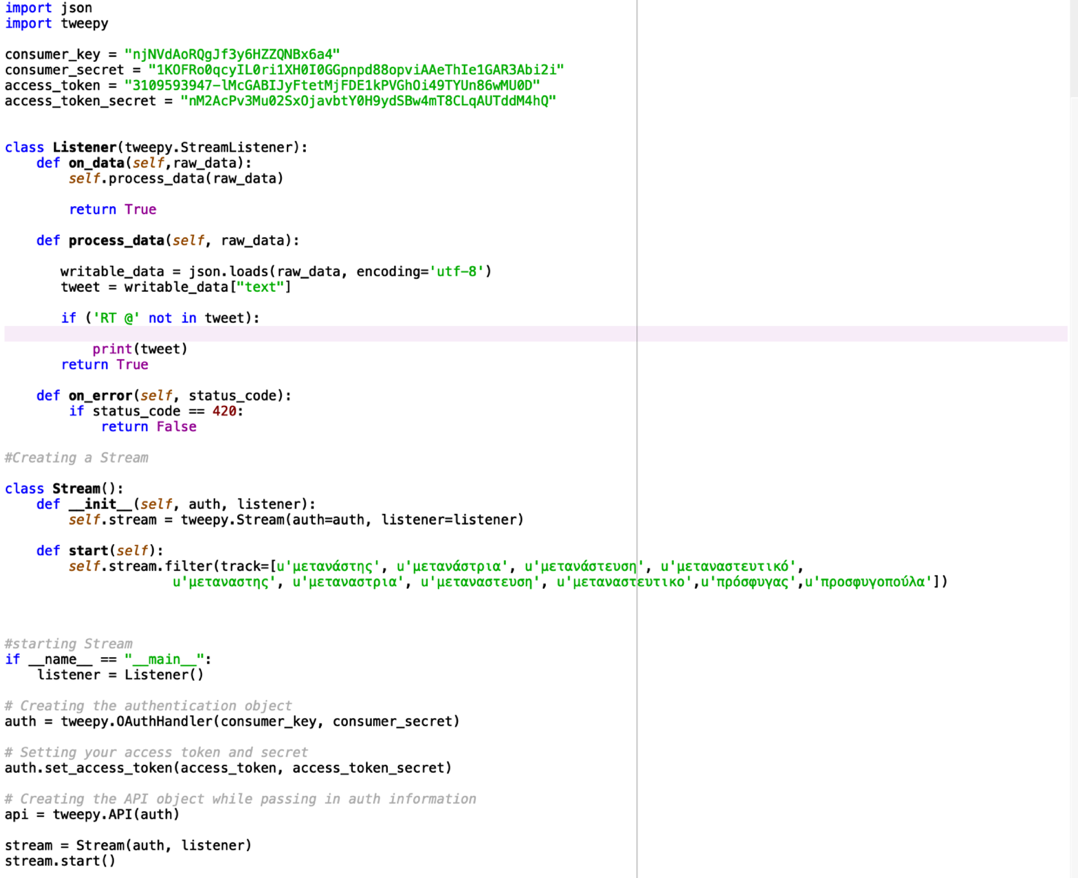
With a for loop we receive the number of tweets we specified in “twpp” and we print it to the console.

The second script can be viewed on the next page.

For the live streaming script, we import the same tweepy library and the json library and we determine the keys as we did on the previous script. We create a new class called Listener that inherits from tweepy’s own *StreamListener* class. We override two of the classes offered by the *StreamListener* class, namely the *on\_data* and the *on\_error* methods. The *on\_data* method exists to determine what to do with the streaming data, here represented by the value of the variable *raw\_data*. The *on\_data* method only calls a user-defined method called *process\_data.* This method takes the data from the stream, encodes them in Unicode UTF-8, to make sure that they can be displayed properly, since ASCII characters don’t show Greek letters well. Each tweet of the data is JSON object that consists of many traits, such as user information, date of creation and a lot more metadata. For this paper we only need the text. Therefore, we have put the line . Lastly, we only display tweets that are not retweets of others, to avoid the duplicates.

The *on\_error* method has the only job of closing the stream under a condition. In our case the condition is to have the status code 420 returned, which is a code of the API that shows that we have exceeded the number of tweets we are allowed to access. If the stream doesn’t quit, then we will have an exponentially growing penalty, which is a timeout of some hours during which we can’t access any data. We, also, create a class called Stream to represent our tweet stream. During its start it is important to filter out some words, to only show live tweets that are relevant. These words are represented by the list called *track,* which is an argument of the *filter* method. Each word in the list is preceded by a “u”, to force the script to recognize it as Unicode and not ASCII.

Lastly, we start the stream and see the printed results as they are posted.



***Image 2: Twitter Live Streaming Script***

**2.2 Tweet Preprocessing**

Text preprocessing is a process where an unstructured text can be normalized into a more processable form. Preprocessing, usually entails a variety of sub-techniques, such as:

* data cleaning
* data editing
* data reduction
* data wrangling

**Data cleaning**

Data cleaning, usually, refers to detecting and correcting (or removing) corrupt or inaccurate records from a record set, database or text. It refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data. Data cleansing may be performed interactively with data wrangling tools, or as batch processing through scripting. When it comes to text cleaning, the goal is to correct words that are misspelled or poorly written. Typically, completeness and consistency are used to measure the success of the process.

**Data editing**

Data editing is defined as the process involving the review and adjustment of collected survey data. The purpose is to control the quality of the collected data. Data editing can be performed manually, with the assistance of a computer or a combination of both. This process tries to remove outliers or other data that would corrupt the end result.

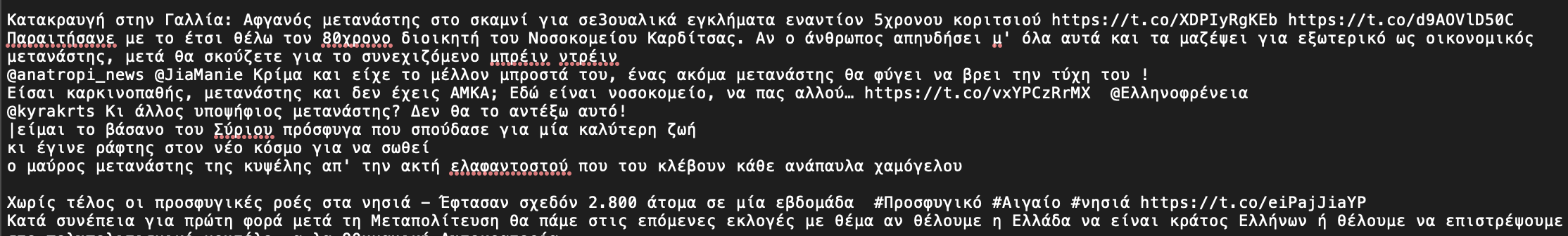
**Data reduction**

Data reduction is the transformation of numerical or alphabetical digital information derived empirically or experimentally into a corrected, ordered, and simplified form. When the data are already in digital form the 'reduction' of the data typically involves some editing, scaling, encoding, sorting, collating, and producing tabular summaries. When the observations are discrete but the underlying phenomenon is continuous then smoothing and interpolation are often needed. For text data reduction, this usually involves removal of stop words and changing number digits into number text.

**Data wrangling**

Data wrangling, sometimes referred to as data munging, is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics. A data wrangler is a person who performs these transformation operations. This may include further munging, data visualization, data aggregation, training a statistical model, as well as many other potential uses. In our case, we save the clean data for further use to our classifier.

Starting off, it’s important to format the data we have in a way that can easily be manipulated. Below, we see a sample of the data that has been collected through the previously-discussed methods in the file with the name “tweets.txt”.



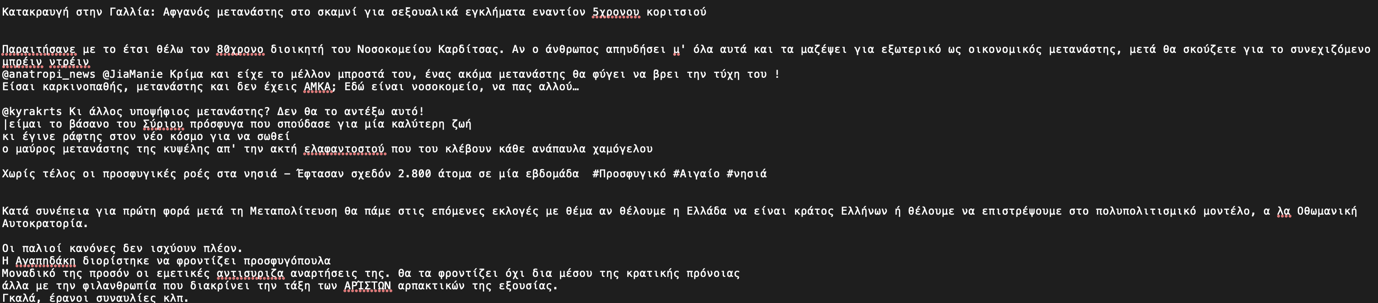
As seen above, the text isn’t properly formatted and needs to be adjusted. First we run the following

code: 

This code takes the file line-by-line removing the line feeds and the carriage returns to have a continuous string of text. Then, the text is formatted by adding a line feed after every link, in order to have a uniform separation of each tweet. All tweets have a link after their text, pointing at the tweet online.

The codes goal is to remove all the duplicate tweets, in case there are any. We do that by adding each tweet to a set, because of its inherent characteristic to not contain duplicates. Before adding the tweet to the set, we remove the URLs, because identical tweets might be posted at different times or users, so the URL would have been different, even though the tweet would not. Tweets that are less than 4 words long aren’t being copied to the new file. This way we avoid having text as tweets that might have been falsely divided.

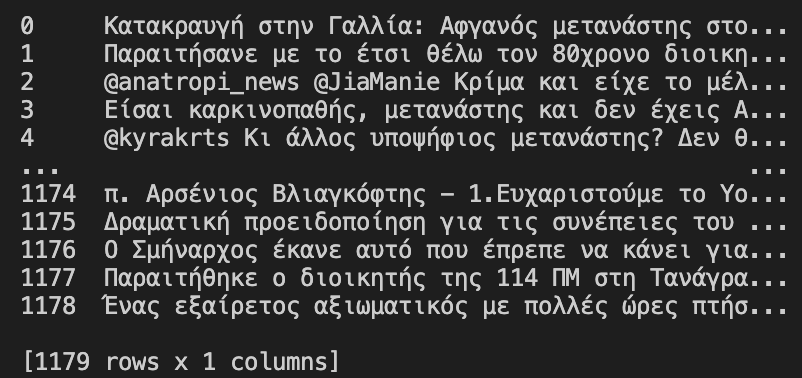
The new file is formatted as such:



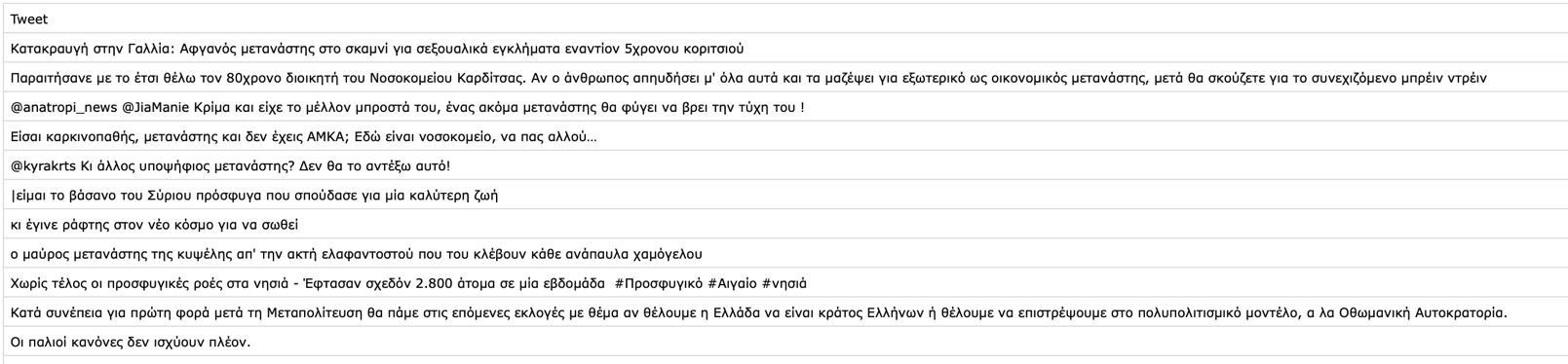
At this point, each tweet occupies some space and is definitely separated by an empty line. This is a vital characteristic of the file and will be useful when we insert the data into another data structure.

Lastly, we create a DataFrame with the help of the “pandas” module. Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A DataFrame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.

The DataFrame is saved into a .csv file for later use. The output in the Terminal is as such:



Whereas, the output of the .csv file is as such:

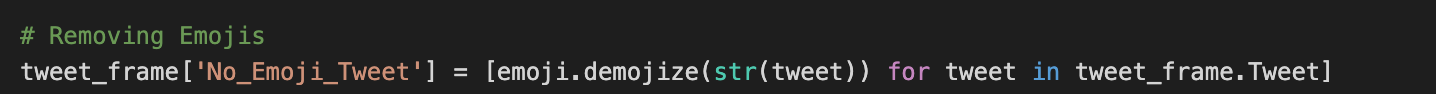


At this point, the data is organized in a useful manner, but any classifier wouldn’t be able to work through them in the format. This is due to the reason that some of the data could be inconsistent, meaning that it might contain words that are written in a different alphabet or contain image-like objects such as emojis mixed in the text. The data might also be misspelled or there might be random small words in some tweets that pose no importance to the analysis. For this reason, the code presented below was developed. Each snippet is followed by the corresponding explanation.



First off, some libraries are imported to facilitate in achieving the functionality expected. We create a set called *punctuation*, which contains all punctuation signs. This way they are easily recognizable and can be easily be identified and removed in later methods. We also set a variable called “dictionary” which uses functionality from the imported library *enchant* and represents a dictionary of American-English words. This, in tandem with the *english\_letters* variable, which is a representation of the Latin alphabet, serve the purpose of recognizing and removing Greeklish characters, which are Greek words written in the Latin alphabet. Lastly, we have a *translator* variable, which takes on the functionality of Google’s own Translate.

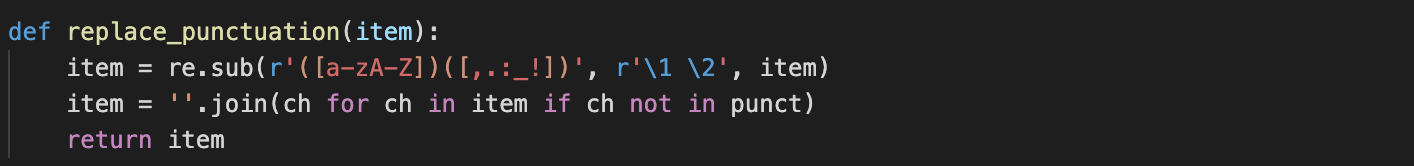
Now that all the variables are set, we move on to loading the data onto a DataFrame variable and creating the vital variable *nlp*, which represents SpaCy’s functionality in understanding Greek words. Once again, we check to remove any other duplicates, this time, which an integrated method.



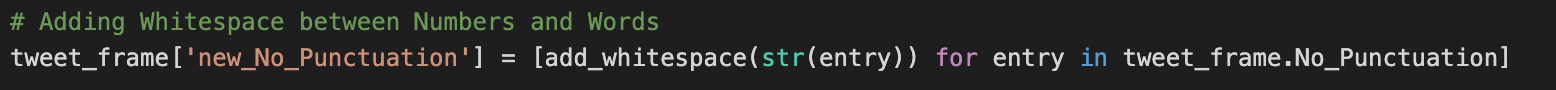
DataFrames are designed to have columns, that contain rows. Each row contains a tuple which is an immutable list with all the data in it. From the previous code, we already have one column in our DataFrame called *Tweet*. We create a new column, called *No Emoji Tweet* that has all the tweets as they are in each row, but every time a tweet would have an emoji, a the place of the emoji there now is a text with the following format: :emoji\_text: . This is achieved with the use of the *emoji* library’s *demojize* method.



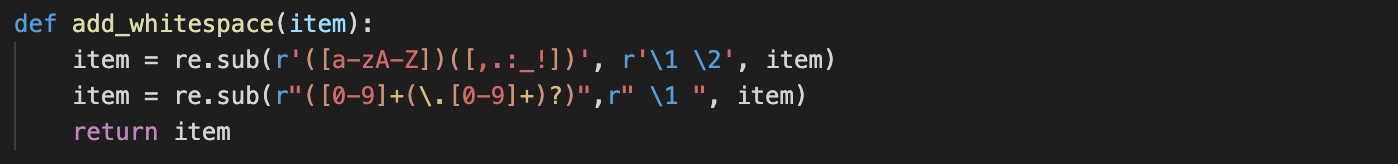
We now create a new column called *No Punctuation* that will contain all the data from the *No Emoji Tweet* column, but this time there will not be any punctuation. For this a new method has been defined and is called *replace\_punctuation(string)*. The code of the method is as seen below:



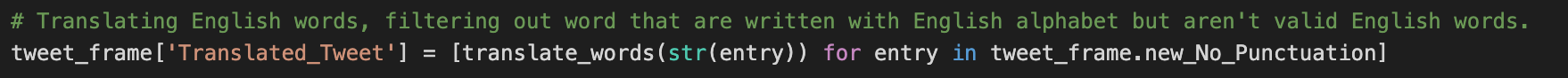
For each tweet, the code looks for letters that have punctuation marks adjacent to them and the adds whitespace between them. Then, the tweet is checked letter for letter, and if there is punctuation, then it is discarded. The first argument of the .sub method is the search term, followed by the replacement for said search term. In this case we would replace the terms found with themselves only adding a comma. \1 references the first part of the term (aka the letters) and the \2 references the second part (aka the punctuation). The r’ is necessary to show that we deal with references and not plain characters.



After removing the punctuation, we create a new column called *new no Punctutation* that contains tweets that are as is, but if there is a number followed by a letter, they are separated. This is useful in cases where there are words like “20χρονος” ( = 20-year-old ) that wouldn’t be understood as a Greek word, where as “ 20” and “χρόνος” would be, separately. The method used here is *add\_whitespacing(string).*

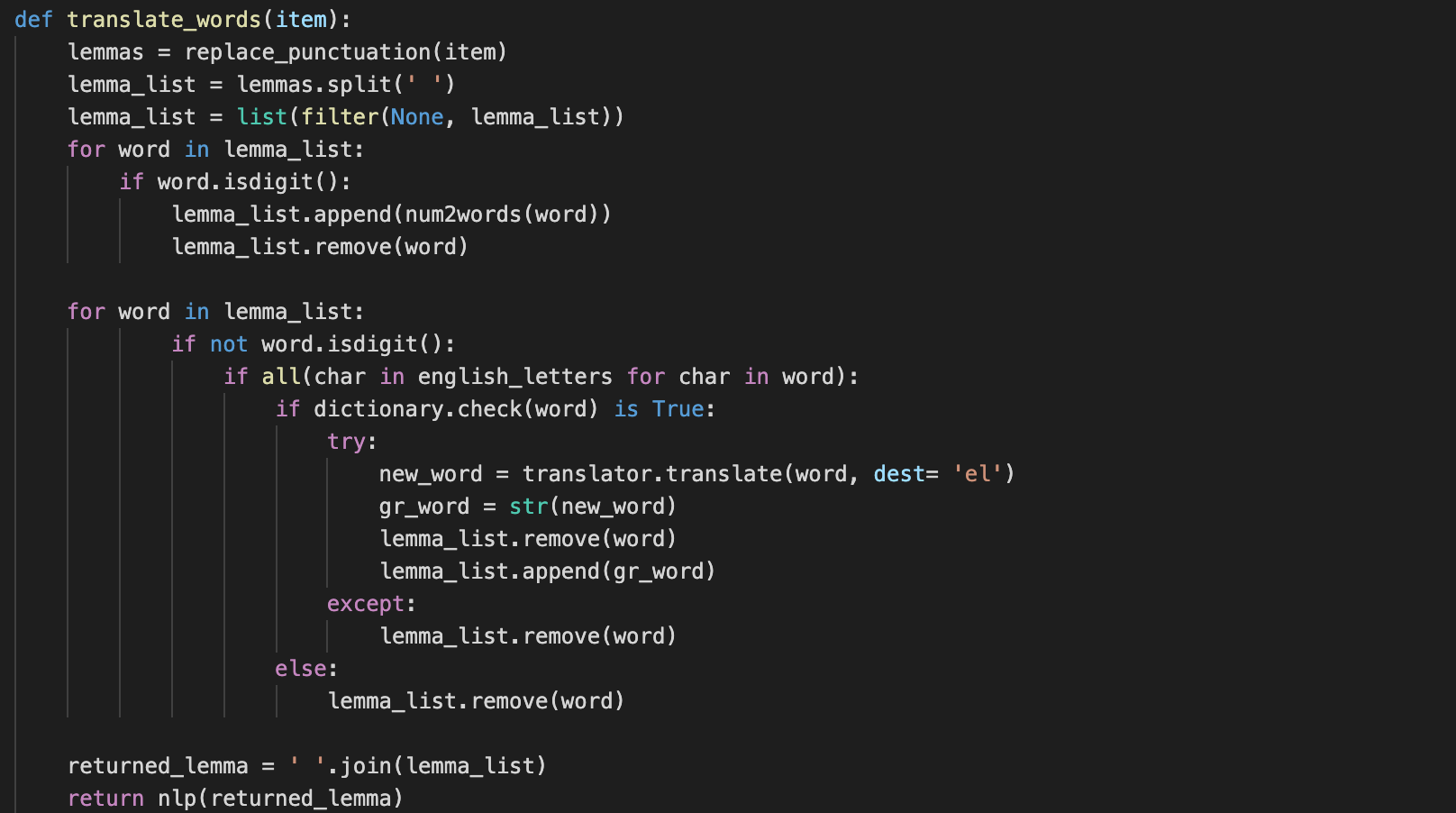


Again, taking advantage of the *re* library’s *substitute* method, we proactively separate punctuation and letters, even though there shouldn’t be any at this point and we do the same with letters and numbers.



At this point, we create a new column called *Translated Tweet* which will contain the tweets but if there are any English words, such as emoji descriptions or simply English words inside an otherwise Greek tweet, they would be translated into Greek. At the same time, tweets that are not English words, but are written in Latin letters, such as usernames or mentions, would be completely removed.

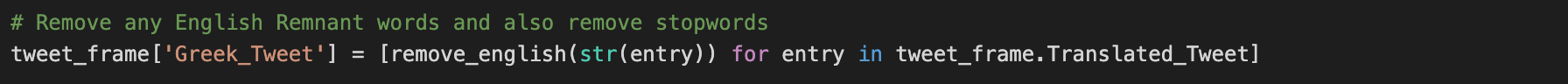
To achieve this, a new method has been created with the name *translate\_words(string).*



Here the method, takes a string as an argument. This string has been casted from the tuple, which is the row of the respective column. We separate the string into words with the native *split* method and we filter out any empty objects from the list of words with the native method filter. For each word in the list, we check if it’s a number and we turn it into text with the method *num2words*. Then, we replace the digit with the text, this way instead of having “20 χρονος” we would have ”twenty χρονος”. Having changed the numbers into words, we enter another loop, were each word written in the Latin alphabet is checked for its validity in the English language. If the word is valid, then the translation of that word is passed into a new variable called *new word.*

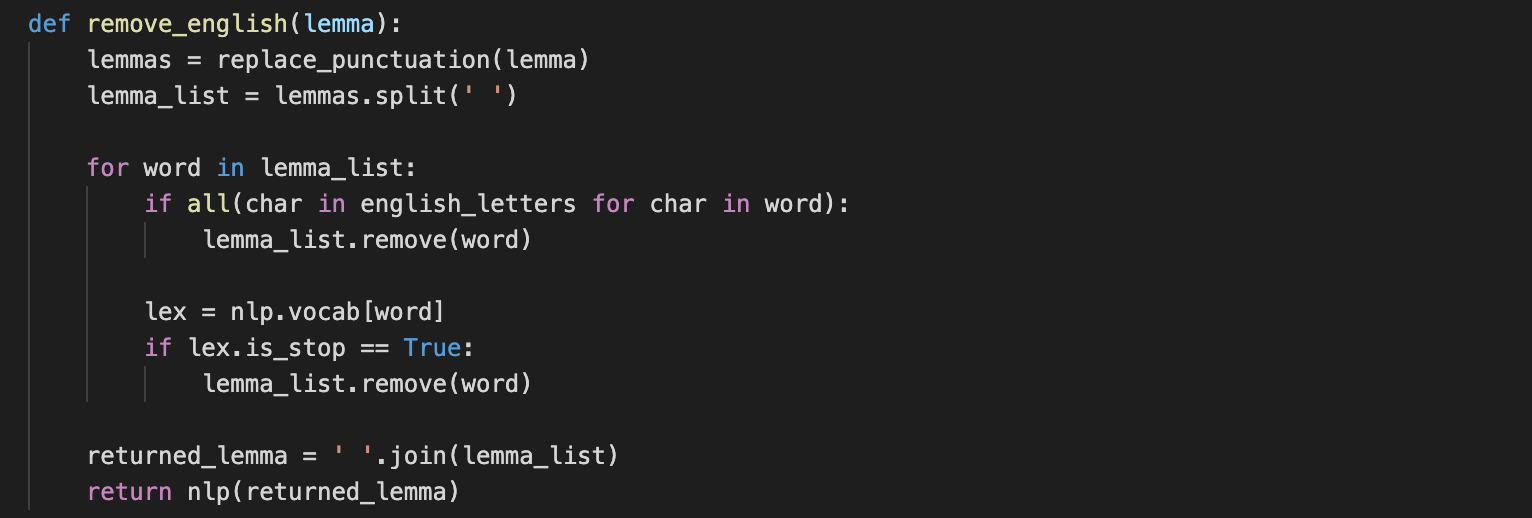
The translated word is then casted as a string and is placed into the word list, while the English word is removed. Since the check happens for each word, the order remains intact.

The code for the translation is in a try – except block, which means that if the translation can’t be completed correctly, then the word is completely removed. In this case the word might have been a slightly misspelled English word or a Greeklish word that somehow passed the dictionary check. In any case, it should be discarded. On the other hand, if the word doesn’t pass the dictionary check, then it is discarded, because it definitely is a Greeklish word or some part of a URL that hadn’t been removed properly. In the end, the new word list is returned. The *nlp* method, tokenizes the newly constructed *returned lemma* string.

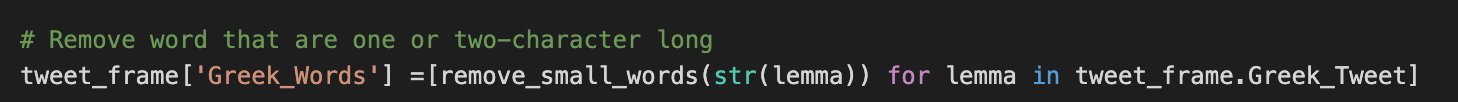


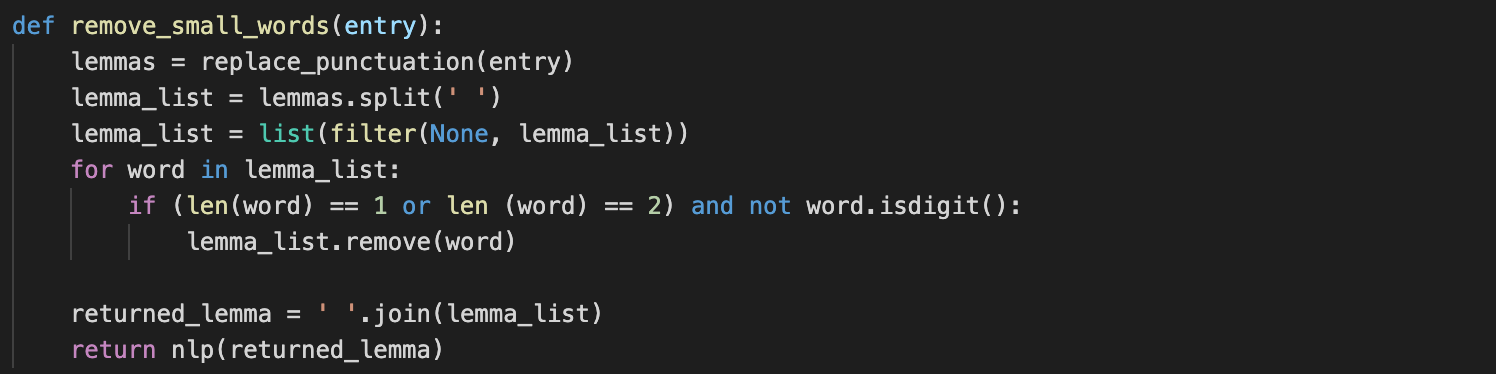
After the translation, we create a new column called *Greek Tweet* which contains the cleaned out version of the translated tweets. At this point we are dealing with tokens, but each argument is still a tuple, even if the tuple is actually a list and not just a string.

The method used to clean the entries is the *remove english(string)* method as is as follows:



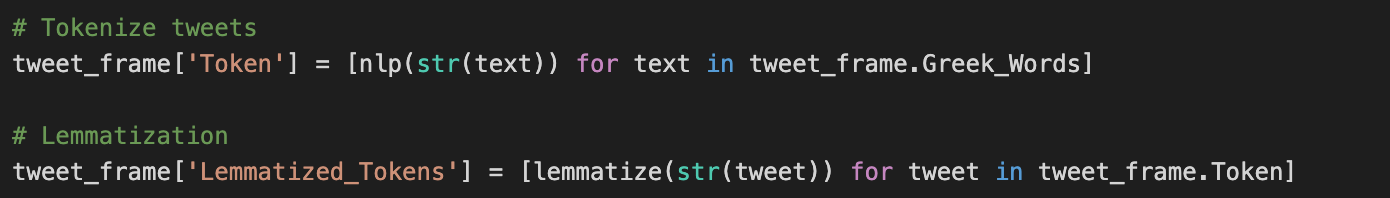
Following the same principle as the last method presented, the function takes the string which is now of the form […,…,…]. Even if it looks like a list, because it has been tokenized, it is still a string and needs to be divided into a new one. Therefore, we call the *replace punctuation* method as used before to clear the brackets and the commas. We split the new string into a word list and for each word we remove every one that is of Latin script, if there are any. Then, we create a new variable called *lex* which tags the word with a SpaCy – specific tag to determine its role. If the word is a stop word, such as “ο, η, οι, το” and so on, it is removed since it gives no particular semantic meaning.





Finishing off the preprocessing, we create a new column that removes all small words with the same principle of creating a word list and removing the words that don’t fulfill the condition posed.

Now that the text has been cleaned the last thing to do is to lemmatize the data. To achieve our goal, we create two more columns: *Token* and *Lemmatized Tokens*.



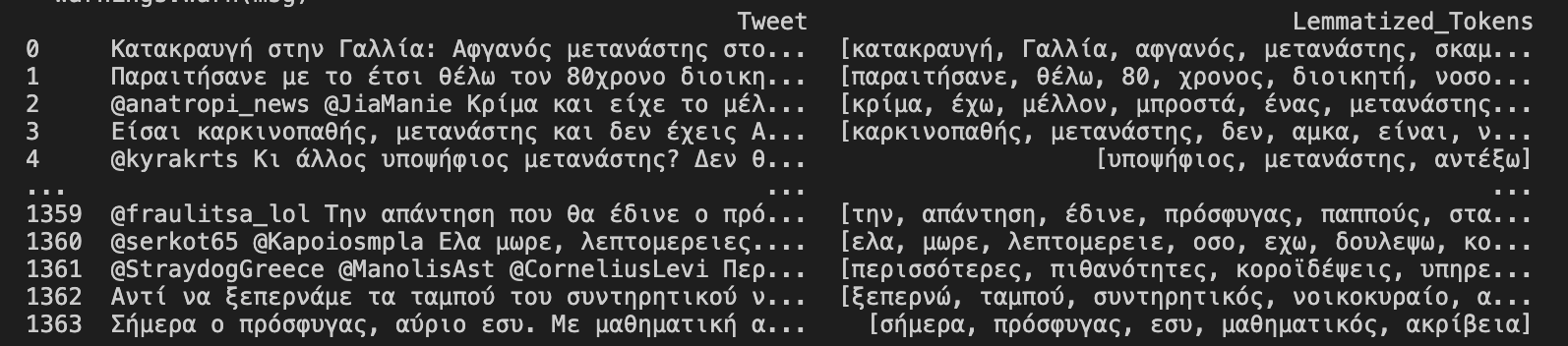
The first column is only created for variable type purposes. It tokenizes anew all the tweets, while the second column contains all the tokens of each tweet but lemmatized, each lemma is the base form of a word. In Greek grammar the base form of verbs is not the infinitive form, but the first singular person form in the present tense. Similarly, Greek nouns have declinations, much like in Latin and, of course, Ancient Greek, in this case their base form is the first person form. The code for lemmatization is a reiteration of an existing method in Spacy called *.lemma\_.*

**

Of course, creating so many columns reduces the calculation time significantly and takes up a lot of storage. This preprocessing code is only made to be run once, so the process to get to the result wouldn’t need to be efficient, but the data that we need for future analysis can’t possibly be a DataFrame of 9 columns. Therefore, we drop all intermediate columns and only keep the original tweet and the lemmatized/tokenized version of it .



The result of the DataFrame now look as such:



In the following chapter we will look at method of labeling the data in a way that is going to prove useful for our further analysis.

**2.3** **Polarity-based Classification**

Sentiment Analysis (SA), a kind of Opinion Mining (OM), is a field of Natural Language Processing (NLP) whose goal is to extract the emotion, sentiment or more generally opinion expressed in a human written text. The text mostly derives from social media, product reviews and blogs. While the term opinion or sentiment is quite generic, the field of study attains a number of tasks. Some of these are, identifying the stance on a target or topic, for instance “Immigration”, extracting the opinion on a product from a review or detecting sentiment polarity in a message. Sentiment analysis is not an easy project. There are issues that can throw the analysis off and need attention. For instance, tweets can be sarcastic or contain ambiguous words, which often lead to misclassifying the polarity of the tweet. In addition, there might be a case of a text, which expresses both a positive and a negative opinion.

A Sentiment Analysis system should also handle negation (i.e., "not good"); perform some kind of word sense disambiguation; and in the case of multiple sentiments and sentiment targets, be able to classify them accordingly. If a message has negative sentiment towards a topic, while expressing positive sentiment towards another topic, then the system should classify the message for each topic accordingly.

In knowledge-based methods, also called Lexicon-based sentiment classification, the target is to construct or use existing sentiment word lexicons with indicated sentiment labels for the words or phrases in the text, also called polarity lexicons. The classification of the text is defined by rules; e.g., a function over the words, such as the sum of word polarities (Taboada et al. 2011). This approach does not require any training, other than forming a lexicon, if required. However, it requires powerful linguistic resources to extract knowledge from words, which are not always available.

Luckily enough, there has been a great effort developing such polarity lexicons by different developers. After extended research, the polarity lexicon of choice will be the one offered by Polyglot, a Python module. Polyglot was made to support all major languages worldwide and offers effective methods of detecting what language is used and other features such as the formation of word embeddings, part-of-speech tagging, morphological analysis and sentiment detection. Out of the features, this thesis will mainly focus on morphological analysis and sentiment detection.

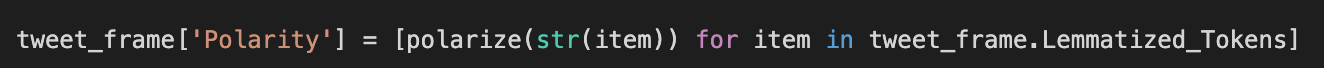
Morphological analysis is useful in cases where words might not have been split properly, such as words that have not been separated by commas and have been taken in as one token, even though they could have been split more effectively. On the other hand, it can offer a better understanding of the already lemmatized words, where the lemmatized words could be converted into morphemes, meaningful morphological units of a language that cannot be further divided. In our case, after trial, it was determined that SpaCy’s lemmatization function and the .morphemes characteristic of words in Polyglot’s library produced the same result, so our lemmas are morphemes as well.

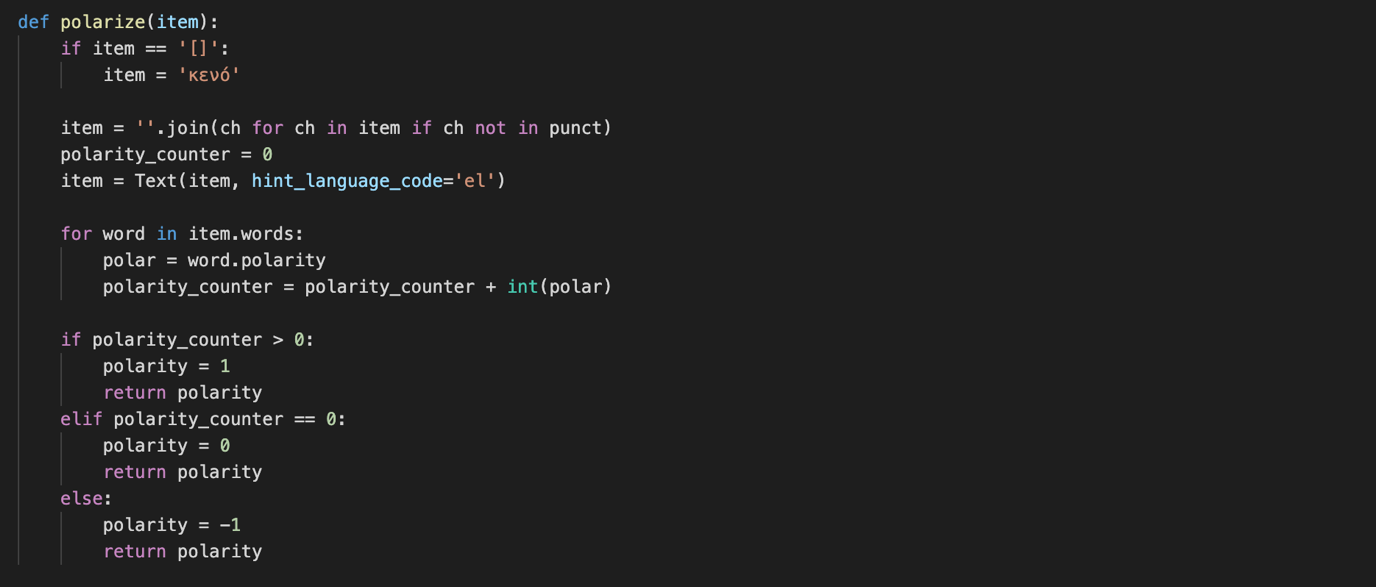
Sentiment detection is useful in cases where we want to label data, in our case tweets, and classify them into coarse categories, such as positive or negative sentiment. Specifically, Polyglot has polarity lexicons for 136 languages. The scale of the words’ polarity consisted of three degrees: +1 for positive words, and -1 for negatives words. Neutral words will have a score of 0. If we add these score up for all word in a tweet, we can have a clear view of the rough sentiment presented in it. Of course, different forms of words should be handled with the same sentiment.

Starting off, we import the polyglot library and access our DataFrame.



Here, we import some libraries which are going to prove useful for determining the polarity of the tweets. We import the method *Text()* from polyglot which converts a string into a special object from that library. The method *Word()* has the same functionality but only for one word. We create anew the set for punctuation, this time adding the Greek quotation marks. Lastly, we access our DataFrame once again, while making sure we have the correct encoding.



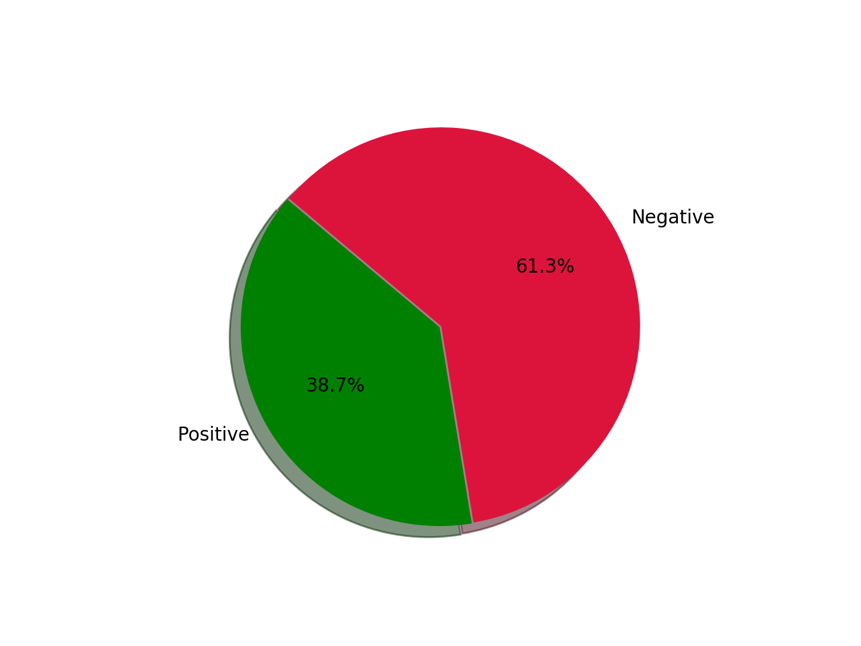


In our DataFrame, we are to add only one more column, which is going to show us the polarity of the tweet. The column will contain the values “ -1, 0 , +1 “, which will indicate *negative, neutral and positive sentiment* respectively. At first, we take each row of the DataFrame’s lemmas column and we use it as input to a user-created *polarize* method. Again, the lemma list is casted into a string, in order to ensure compatibility with some basic string methods.

After preprocessing, the DataFrame had some rows that were empty, as shown by a “[]” symbol. This is due to some separation issue, mainly the addition of some words after the tweet URL, which has been the main separation criterion. Such words were, “via x-y-z news” & similar phrases that posed no importance and were in English, so these had been removed, leaving behind an empty row. If the *Text* method encounters an empty string, it throws an exception and the script crashes. For this reason, the first thing we do is to mask this empty string with the word “κενό”, which means “void” and has by default a neutral (0) polarity. This way, the polyglot library doesn’t find any empty rows and the ones that are empty would change nothing in the evaluation process.

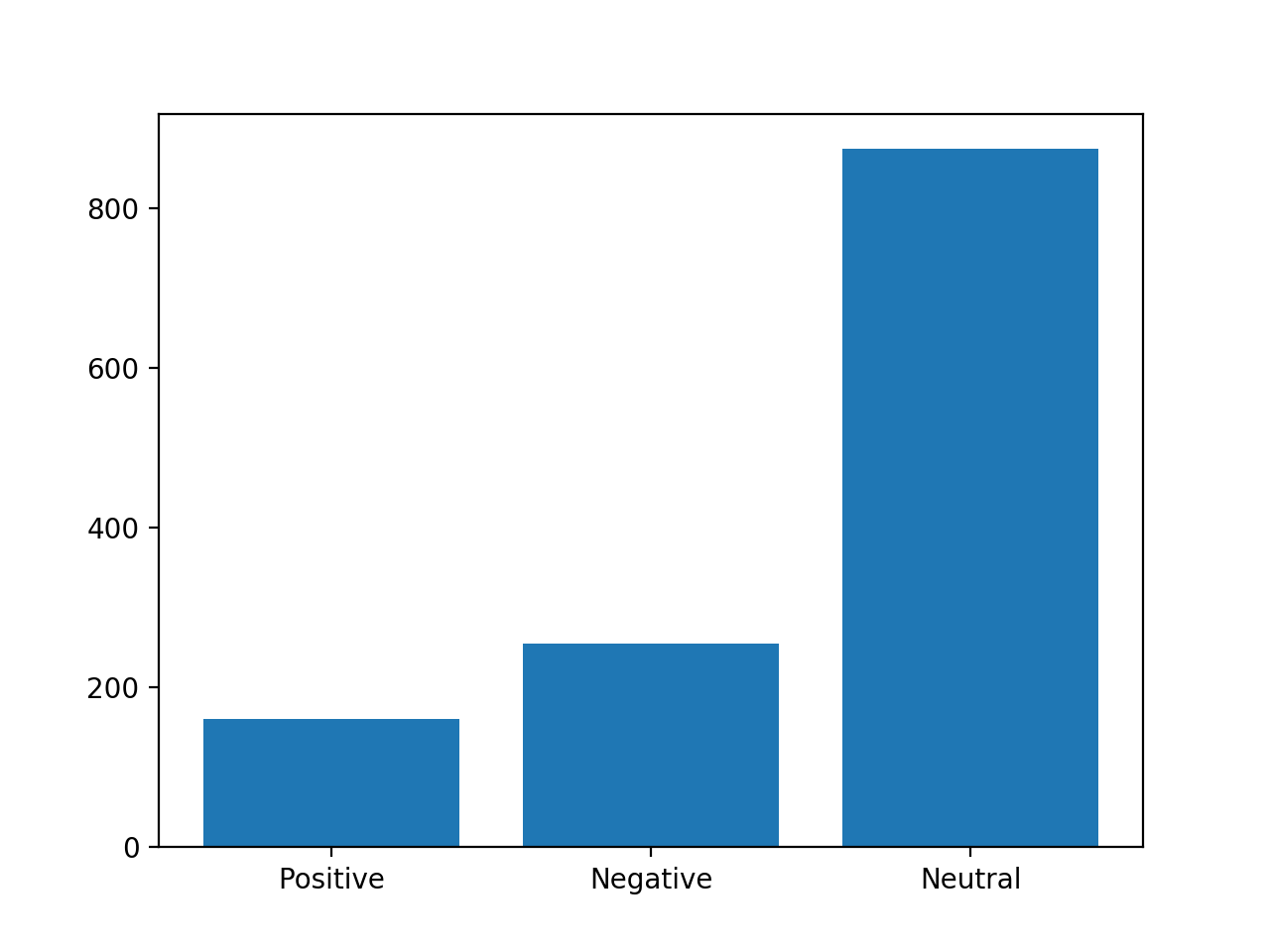
We check the row and remove all punctuation once again, while creating a counter for the polarity of each word. Having created the counter, we change the row into a polyglot.Text type and we divide it into words with a built-in function. Each word has a polarity field by its own, so taking advantage of this feature we add up all word’s polarities in a sentence with the *polarity counter* variable. At this point, it is assumed that if a tweet has both positive and negative words in it, we choose to label it with the prevailing class of words. This functionality is ensured with the *if-elif-else* clause.

Having found the polarity of each tweet we can show the general sentiment Greeks have regarding the topic of Immigration, represented by the following graphs:



**General Sentiment distribution**

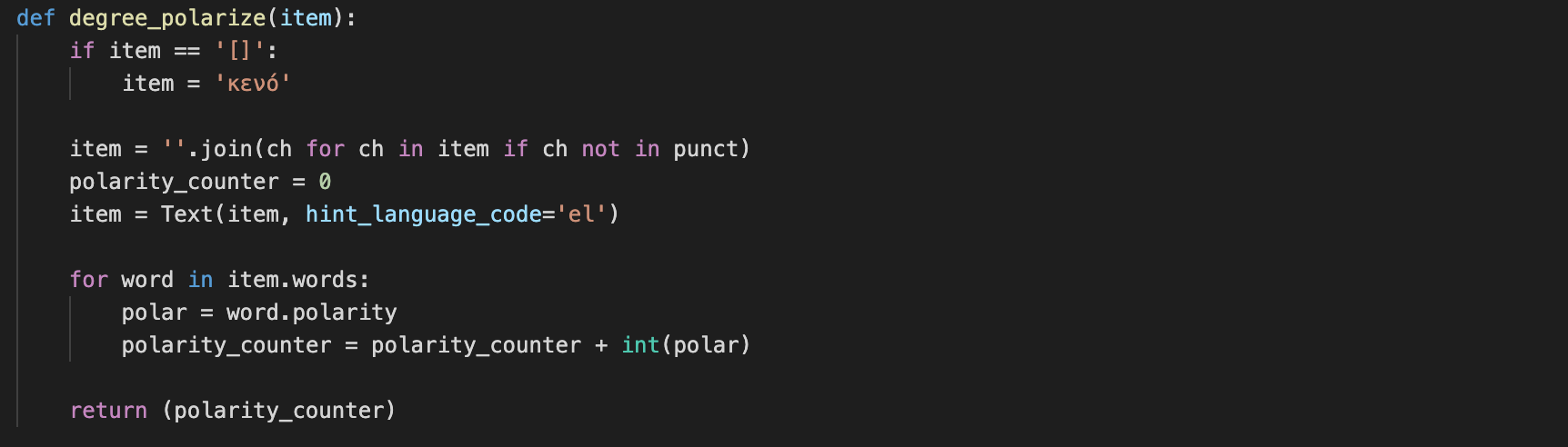
In the pie chart, it is shown that Greeks mainly have a negative disposition towards immigrants and immigration in general. This, on the one hand, might seem a bit discouraging, on the other hand out of the approximately 1300 tweets we have, most of them showed neutral sentiment, as shown below.

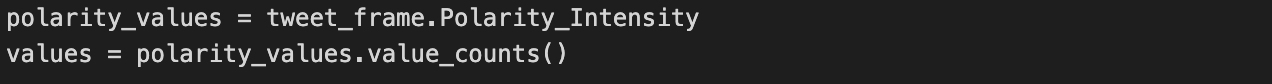


**Number of tweets for each sentiment category**

Even if the tweets are now divided into positive and negative sentiment, in our evaluation we did not leave any room for intensity. Every tweet which had more positive than negative words was placed into the “positive sentiment” class. This is beneficial, if we only plan to show a coarse classification, but it might prove useful to show the degree of the sentiment in each tweet by allowing the polarity to have larger numbers than just those allowed before.







Counting the values of the newly-created column, as shown above we get:



**2.4 Lexicon-based Emotion Extraction**

Emotion extraction in computational linguistics is the process of identifying discrete emotion expressed in text. Emotion analysis can be viewed as a natural evolution of sentiment analysis and its more fine-grained model. In recent years, emotion detection in text has become more popular due to its vast potential applications in marketing, political science, psychology, human-computer interaction, artificial intelligence, etc. That’s because, the amount of useful information which can be gained by moving past the negative and positive sentiments and towards identifying discrete emotions can help improve many already established applications and also open ways to new use-cases.

In other words, not all negative or positive sentiments are created equal. For example, the two emotions Fear and Anger both express negative opinion of a person toward something, but the latter is more relevant in social monitoring of the public sentiment. It has been shown that fearful people tend to have pessimistic view of the future, while angry people tend to have more optimistic view (Lerner and Keltner, 2000). Moreover, fear generally is a passive emotion, while anger is more likely to lead to action (Miller et al., 2009).

**Emotion Models & Theories**

The prerequisite for talking about extracting emotions, is having a general idea about the emotion models and theories in psychology. In spite of the fact that there is no all-around acknowledged model of feelings, the absolute most broadly acknowledged models that have been utilized in emotion detection can be divided based on two viewpoints:

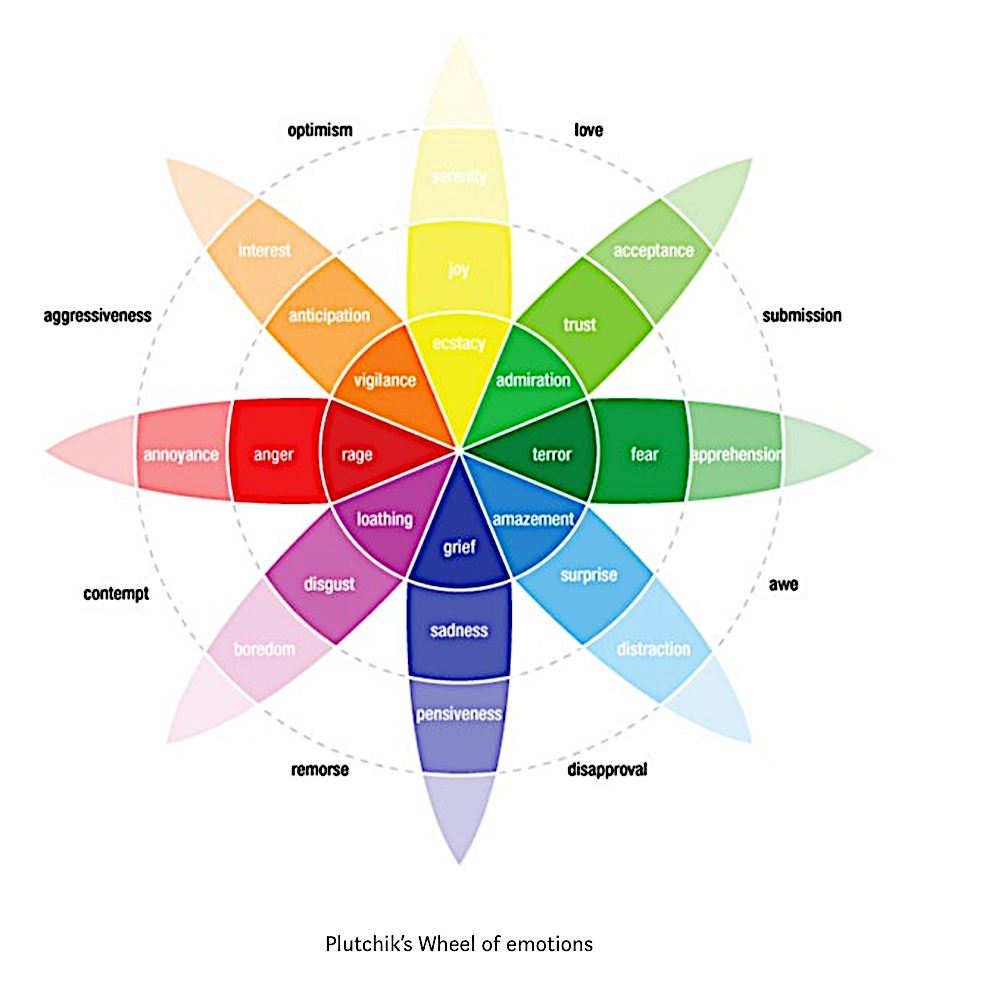
* emotions as discrete categories
* dimensional models of emotions.
* scale of differential emotions

According to Discrete Emotion Theory, some emotions are distinguishable on the basis of neural,

behavioral and expressive features regardless of culture. A well-known and most used example is Ekman’s six basic emotions (Ekman, 1992). Ekman supports that there are six basic emotions which can be found in all cultures: sadness, happiness, anger, fear, disgust, and surprise. Most papers in emotion detection used this model for detecting emotions as a multi-class classification problem.

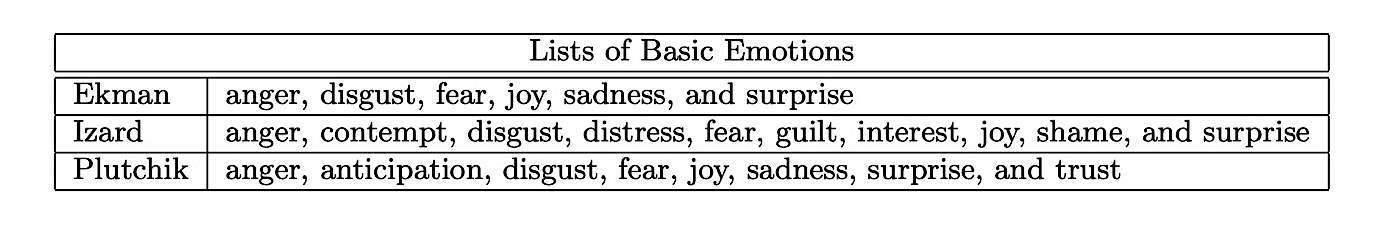
On the other hand, there is Plutchik’s wheel of emotions (Plutchik, 1984) in which he categorized eight basic emotions: joy, trust, fear, surprise, sadness, disgust, anger and anticipation as pairs of opposite emotions. Plutchik's wheel of emotions is an infographic that uses the color wheel to illustrate variations in human affect and the relationship among emotions. Current applications of the wheel of emotion include robotics and sentiment analysis. Intensity of emotion and indicator color increases toward the center of the wheel and decreases outward. At the center, terror, for example, becomes fear and then apprehension; ecstasy becomes joy and then serenity. Secondary emotions are displayed as combinations of the primary ones.

Below, the wheel is presented visually.



There is also one more prevailing theory regarding human emotions, the one of Izard. Discrete emotion theory is the claim that there is a small number of core emotions. In contrast, Carroll Izard analytically delineated 12 discrete emotions labeled: Interest, Joy, Surprise, Sadness, Anger, Disgust, Contempt, Self-Hostility, Fear, Shame, Shyness, and Guilt, as measured via his Differential Emotions Scale or DES-IV.

Summing up the basic emotions, as of each theory can be found in the table below:



For our research we will focus on displaying eight kinds of emotion – namely anger, anticipation, disgust, fear, joy, sadness, surprise, trust – utilizing Plutchik’s model of emotions.

**Lexicon Development**

Emotion expression is very context-sensitive and complex. Ben-Ze’ev (2000) relates this complexity to various reasons: first, its sensitivity to multiple personal and contextual circumstances; secondly, to the fact that these expressions often consist of a cluster of emotions rather than merely a single one; and finally, the confusing linguistic use of emotional terms. Therefore, it is usually very demanding and difficult to analyze texts taking into account all the factors. One solution to automate this task is an emotion lexicon, also called an emotion-labeled dataset.

Emotion-labeled datasets are blocks of text that have been annotated with emotion tags. Manually annotating datasets of text is expensive and time consuming. However, because comparing results to annotated texts is the most stabilized method of checking the accuracy of an algorithm, annotated datasets have been established and consistently used throughout emotion detection studies.

In our research we have utilized 4 different emotion-labeled datasets, namely: 2 CrowdFlower lexicons, the NRC lexicon & the ISEAR lexicon. All of these are open-source attempts to tag sentences or words with emotions, whereas the NRC has been the most used lexicon in the last years, during which emotion extraction has been developing. All lexicons tag whole sentences with one emotion, while the NRC tags specific words with an emotion tag.

All of these have been made into DataFrames and, then, into CSV files with two columns each. The first one being sentiment and the other one being content. As a last step they have been combined into a big emotion lexicon, consisting of over 120k rows, after all irrelevant emotion tags have been filtered out. Emotion tags that were removed include other emotions like guilt, that correspond to a different emotion model.

*The code for this can be found under ./preprocessing tweets/make\_lexicon.py in the repository mentioned in the Abstract page of this dissertation.*

The new lexicon is now almost ready to use. The last step would be to translate the sentiment classes and the content into Greek, so that the tweets can be compared with it directly.

Plutchik’s categories, in the order presented in the table in the previous page, are: θυμός, ανυπομονησία, αηδία, φόβος, χαρά, θλίψη, έκπληξη and εμπιστοσύνη.

Translation was conducted using a python implementation of Cloud Translation, provided by Google. The end result is named *emotion\_lexicon.csv.*

**Emotion Extraction**

Having developed the lexicon necessary to conduct emotion extraction, it is easier of the process to be thought as multinomial classification. In multiclass classification each training point belongs to one of N different classes. The goal is to construct a function which, given a new data point, will correctly predict the class to which the new point belongs. This type of classification makes the assumption that each sample is assigned to one and only one label: a tweet can only convey one emotion at one time, either happiness of anger, not both at the same time.

Based on the general category of Immigration which the tweets adhere to, it’s valuable to start off with an assumption. Having computed the sentiment of the dataset to be mainly negative and basing of the general cultural opinion about the topic, it is safe to assume that the Emotion Extraction process would yield more results that have a negative connotation rather than a positive one. Therefore, the assumption is as follows:

***It is expected to have emotions such as sadness, fear, disgust & anger appear more often than any other emotion during the classification process.***

For the classification we will add a new column to the already manipulated dataset of tweets which we have gathered in section 2.1. The new column will show the prevailing emotion out of Plutchik’s 8 basic emotions. The emotion is calculated looping over each word of a given tweet and trying to match it with words used in the emotion lexicon.





After a match has been found, the script looks at the emotion label of the matched sentence and increments a counter of the corresponding emotion.



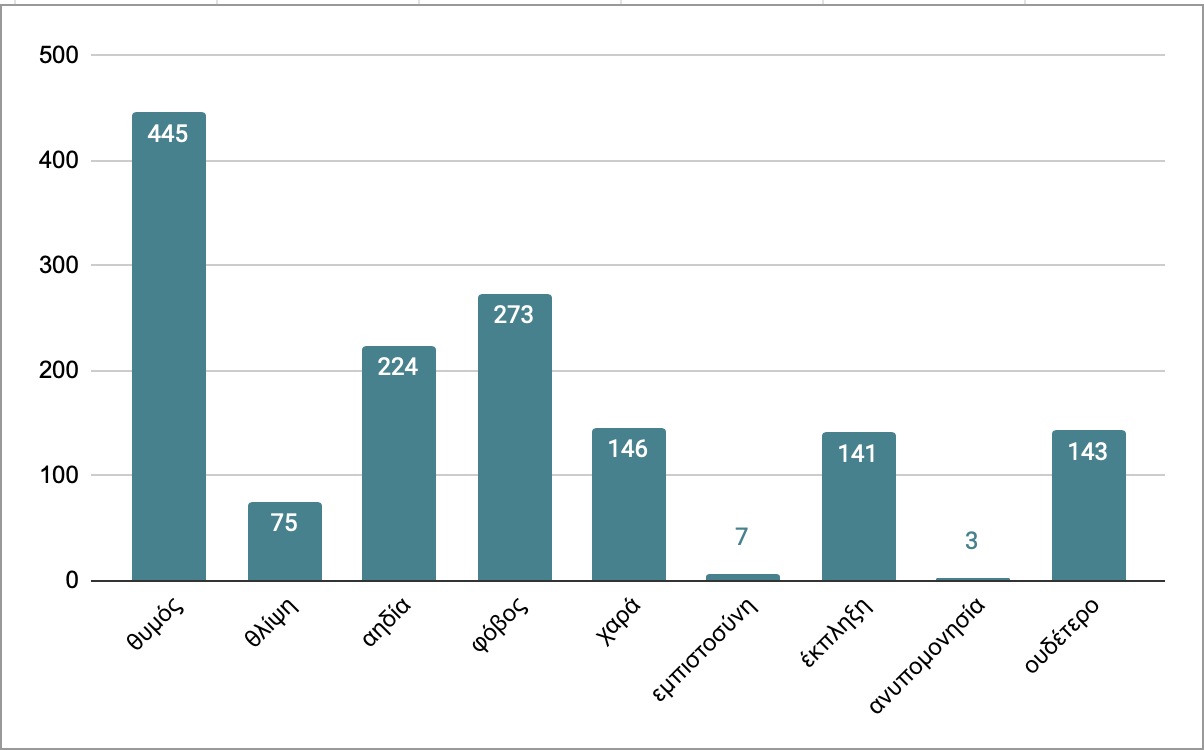
All the counters are then put in a list, where they are correlated with the appropriate emotion tag. This achieved by a zipped list, where each element is a pair of (emotion, frequency in lexicon).

For each tweet, one such list is produced. After the list is sorted based on frequency, the emotion tag is returned and added to the column named ‘Emotion’. For the emotion lexicon all 8 emotions had similar occurrence frequency, meaning that “joyful” sentences appeared roughly as much as any other emotion label to ensure that the result would not lean towards one emotion more often than others.



As mentioned above, we expect that the results will likely be negative emotions. After each tweet has been labeled we will use the resulted DataFrame as a baseline training set to train a Neural Network which will be able to categorize new unseen tweets and determine the emotion they convey.

Below, the results of the script are presented:



**Emotion distribution across all tweets**

These results verify the initial assumption, where negative emotions appear more often than others. More specifically, negative emotions account for 70.7% of all tweets. These include anger (445), sadness (75), disgust (224) and fear (273).

**3. Deep Learning Approach**

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network. It differentiates itself from Machine Learning, in the aspect of training.

Machine Learning can classify or detect a pattern in the data using calculations. In that sense, the detection of emotion in the chapter above can be considered Machine Learning. Another example would be fraud detection, where a model is created around thresholds of money a user sends or receives. Deep Learning takes the results offered by Machine Learning and use it to make prediction of future unlabeled data.

In this chapter, there will be an attempt to create a neural network which will be able to take one unlabeled tweet as input and detect its emotion as effectively as possible. The tweets, also known as test set, will be provided by the script that was created in Chapter 2.1, which listened for tweets with certain keywords and then returned them in real time. This is a guarantee that the tweets will be unseen and unlabeled and will not be part of the training set.

The training set will include all the tweets that have been labeled in Chapter 2.4, as well as part of the emotion lexicon that was created for the task previously mentioned. The emotion lexicon that was created consisted of two parts. The first part included whole sentences that were labeled with emotion tags, whereas the second part of the lexicon only consisted of words that showed emotion. For this training set we will use the tweets labeled and the first part of the lexicon.

To feed the labeled data into the neural network as input, we first need to change them into a format that can be easily understood by the network. Such formats can be achieved with method called word vectorization, with word2vec being one of the most popular ones. After the input is in an understandable form, the network will calculate the weights of each neuron until the outputs are mostly correct. During the epochs of training, there will be metrics to calculate the effectiveness of the process.

**3.1 Word Vectorization**

Word Vectorization is a technique which aims to create word embeddings which are a numerical representation of words in a given text. Since deep learning has taken over the machine learning field, there have been many attempts to change the way text vectorization is done and find better ways to represent text. One of the initial steps that were taken to vectorize words turned out to be the word2vec implementation in 2013.

Word2Vec is an algorithm that uses a shallow neural in order to create embeddings, namely vectors for each word of a text. This algorithm keeps the semantics of the words, meaning that if two words have similar meaning, their vectors will be close in the N-dimensional space which would contain them. These vectors are useful for doing a lot of tasks related to NLP because each of its dimensions encode a different property of the word. For example, you can have one property that describes if the word is a verb or a noun, or if the word is plural or not.

Word2Vec can create vectors using one of two methods, Skip-Grams and Continuous Bag of Words (CBOW). CBOW is learning to predict the word by the context or maximize the probability of the target word by looking at the context. This happens to be a problem for rare words. For example, given the context yesterday was a really [...] day, the CBOW model will tell you that most probably the word is beautiful or nice. Words like delightful will get much less attention of the model, because it is designed to predict the most probable word. This word will be smoothed over a lot of examples with more frequent words.

On the other hand, the skip-gram model is designed to predict the context. Given the word delightful it must understand it and tell us that there is a huge probability that the context is yesterday was really [...] day, or some other relevant context. With skip-gram the word delightful will not try to compete with the word beautiful but instead, delightful & context pairs will be treated as new observations.

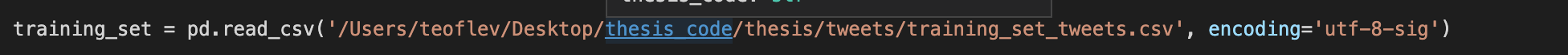
According to the creator of the algorithm CBOW is several times faster to train than the skip-gram and has slightly better accuracy for the frequent words, whereas Skip-Gram works well with small amount of the training data and represents even rare words or phrases well.

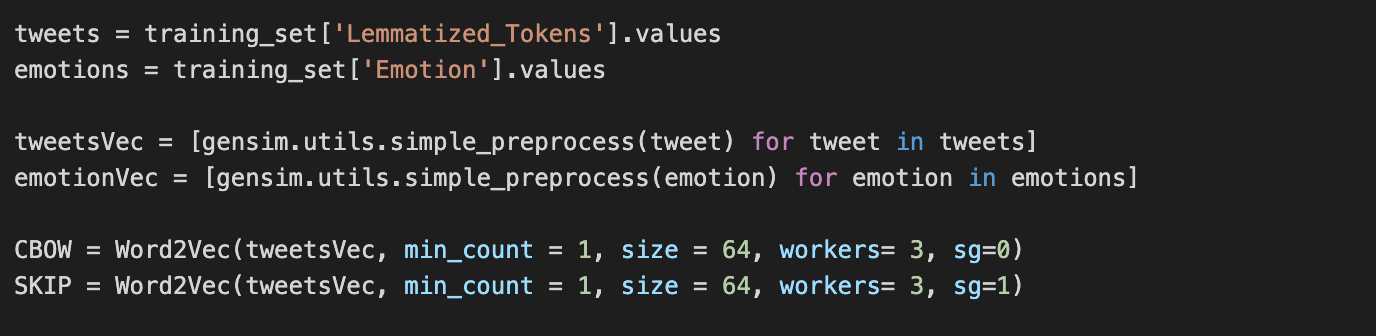
Lastly, a great alternative to the Word2Vec approach would be TF-IDF Vectorization. TF-IDF stands for Term Frequency - Inverse Document Frequency and is a statistical measure, which is used to evaluate how important a word is to a document in a collection or corpus. The measure takes the number of times a term appears in a document and also the inverted value of the number of documents the terms appears in. While computing TF, all terms are considered equally important. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Therefore, we combine the measure with IDF to ensure that more important words are represented in a better manner. Below, there is a coarse formula for each:

By using vast amounts of data, it is possible to have a neural network learn good vector representations of words that have some desirable properties like being able to do math with them. This is the main reason why many NLP systems require this step in order to produce a result using artificial neural networks.

**Implementation**

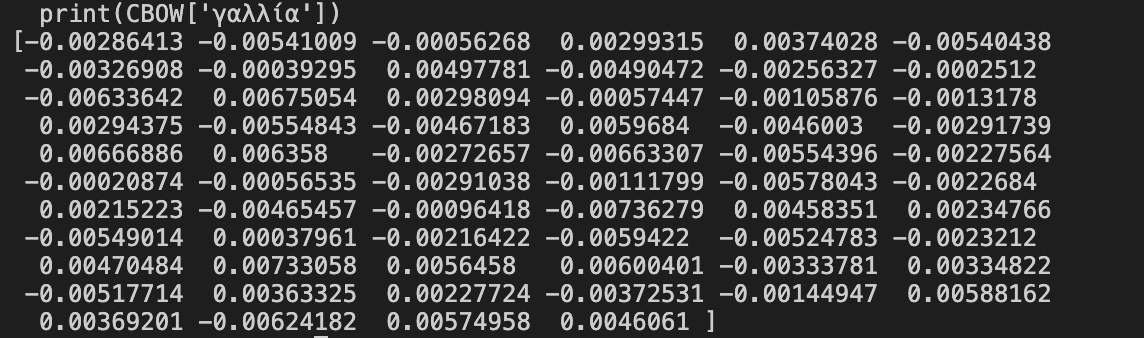
For the Word2Vec approaches the library of choice was Gensim. Below is the code:



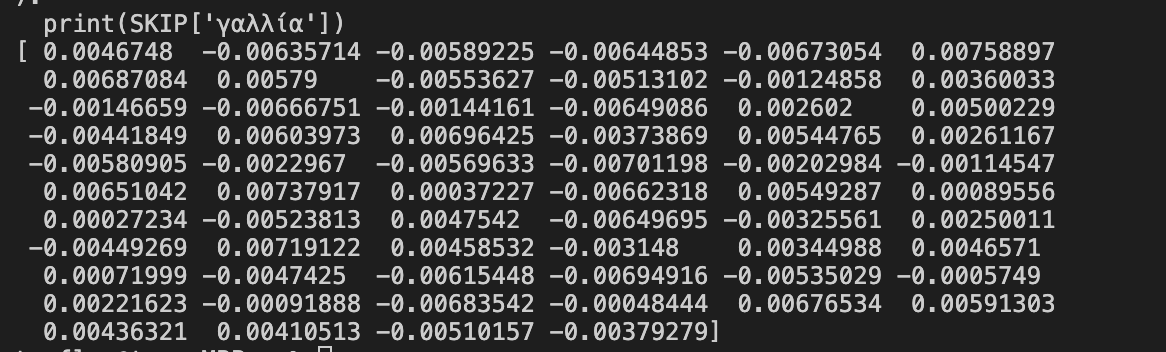


This yielded vectors for each word with each vector having 64 values.

Both models had a resulted vocabulary of 6267 words. For example, a word such as “France” would be represented by the vector below in the CBOW model.

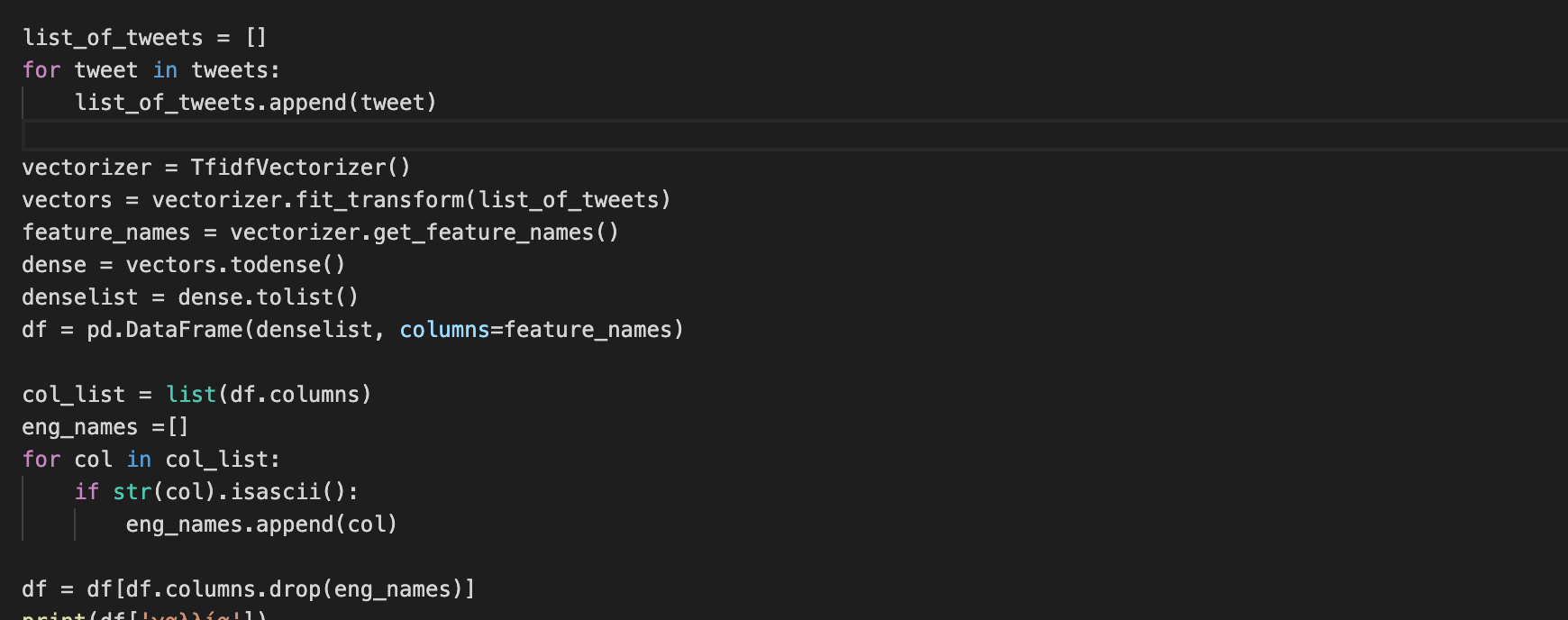


Whereas it would be represented as such in the Skip-Gram model:

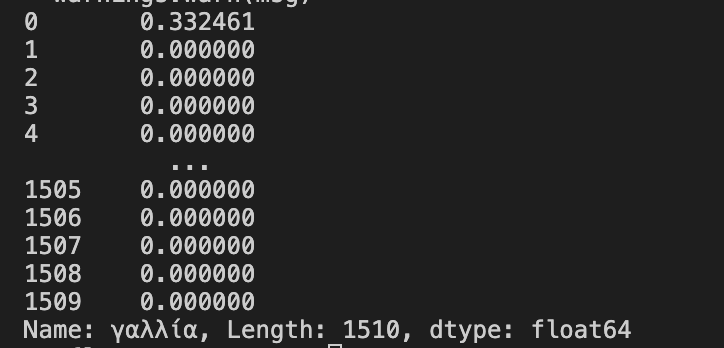


For the TF-IDF Vectorization we will use another library called sklearn. Specifically:





The end result will be an N-dimensional vector for each word. The size of the vector depends on the number of tweets we have. At this point each word is represented by a 1500+ dimensional vector, which consists of many zeros as seen below:



For this reason, it is clear that the use of Word2Vec models will be more practical, because of the significant difference in vector size, as well as the density of the vectors themselves. For the training of the predictive network we will therefore use one of the Word2Vec representations, the Skip-Gram, since the dataset will not be more than 2000-3000 tweets.

**3.2 Types of Neural Networks for Classification**

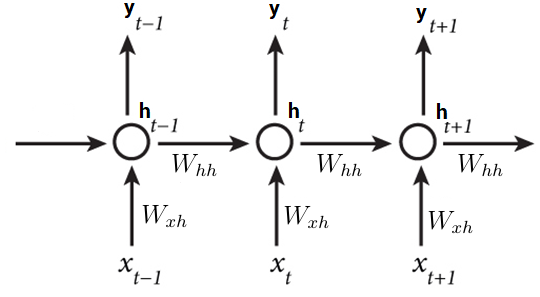
Much of modern technology is based on computational models known as artificial neural networks. There are many different types of neural networks which function on the same principles as the nervous system in the human body.

All in all, there can be argued that there are at least 7 different kinds of Neural Networks. Each kind is mainly differentiated by the type of input it takes in and whether or not it uses other types of inputs for its activation function. The simplest form of Neural Network would be the Feedforward model. In a feedforward neural network, the data passes through the different input nodes till it reaches the output node. This is because the target classes in these applications are hard to classify. A simple feedforward neural network is equipped to deal with data which contains a lot of noise. More complex networks include Recurrent Neural Networks and Convolutional Neural Networks These two are also the most effective in Emotion Extraction.

**Recurrent Neural Networks**

Recurrent Neural Networks (RNNs) are popular models that have shown great promise in many NLP tasks. RNN’s make use of sequential information such as text. In a “traditional” feedforward neural network we assume that all inputs are independent of each other. But for many tasks that’s a very bad idea. A sentence, for example, has a clear grammatical structure and order, where each word depends on the previous word. If you want your neural network to learn the semantics and emotion, the network must know which words came in which order.

RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being dependent on the previous computations. Another way to think about RNNs is that they have a “memory” which captures information about what has been calculated so far. A typical RNN looks as such:

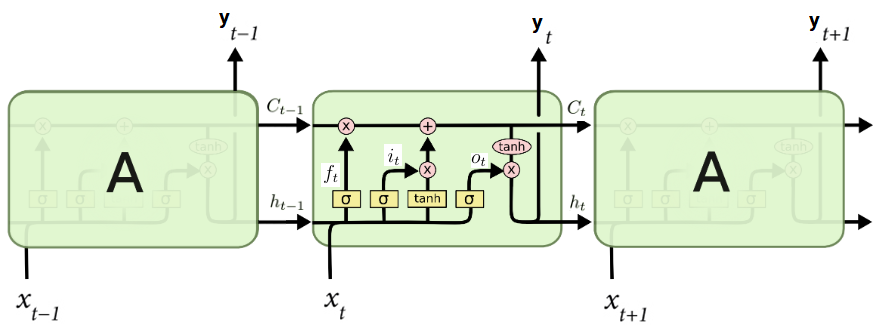


The “x” variables represent words in a sentence, where “y” variables are the output. Unique for RNN is the fact that the calculation of the current hidden state h(t) of the neurons for the input x(t) depends on the previous hidden state h(t-1) for the previous input x(t-1). The hidden state acts as the neural network’s memory. Wxh and Whh are weight matrices that connect the input x(t) with the hidden layer h(t), and h(t) with h(t-1) respectively.

**Long – Short Term Memory Networks**

Recurrent Neural Networks suffer from short-term memory, an issue called vanishing gradient problem. If a sequence is long enough, an RNN will not be able to carry information from earlier time steps to later ones. The solution to this issue is Long Short-Term Memory networks.

Usually just called “LSTMs” they are a special kind of RNN, capable of learning long-term dependencies. LSTMs don’t have a fundamentally different architecture from RNNs, but instead they have some additional modules.



An LSTM offers 4 additional features to the traditional RNN structure. First, there is the cell state C(t), the horizontal line running through the top of the diagram. A cell state is an additional way to store memory, besides just only using the hidden state h(t). Furthermore, LSTMs have the ability to remove or add information to the cell state, regulated by structures called gates, which are able to let information pass through. An LSTM has three of these gates, to protect and control the cell state. These are:

* *Forget Gate:* After getting the hidden state h(t-1 ) of the previous input x(t-1) the forget gate helps make decisions about what must be removed from h(t-1) state and thus keeping only relevant information. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.
* *Input Gate:* The input gates weighs the decision to add new information from the present input x(t) to the present cell state C(t). First, the previous hidden state alongside the current input is passed into a sigmoid function. That decides which values will be updated by transforming the values to be [0, 1], with 0 meaning not important and 1 important. The hidden state and current input are also into the tanh function to transform the values between [-1, 1] to help regulate the network. Then the tanh output is multiplied with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.
* *Output Gate:* The output gate as the name suggests, decides what to output from the current cell state C(t) to the next C(t+1). The current input and the previous hidden state are passed through the sigmoid. The result goes in the tanh function and the tanh output is multiplied with the sigmoid output. The result is the new hidden state. The new cell state and the new hidden state is then carried over to the next time step.

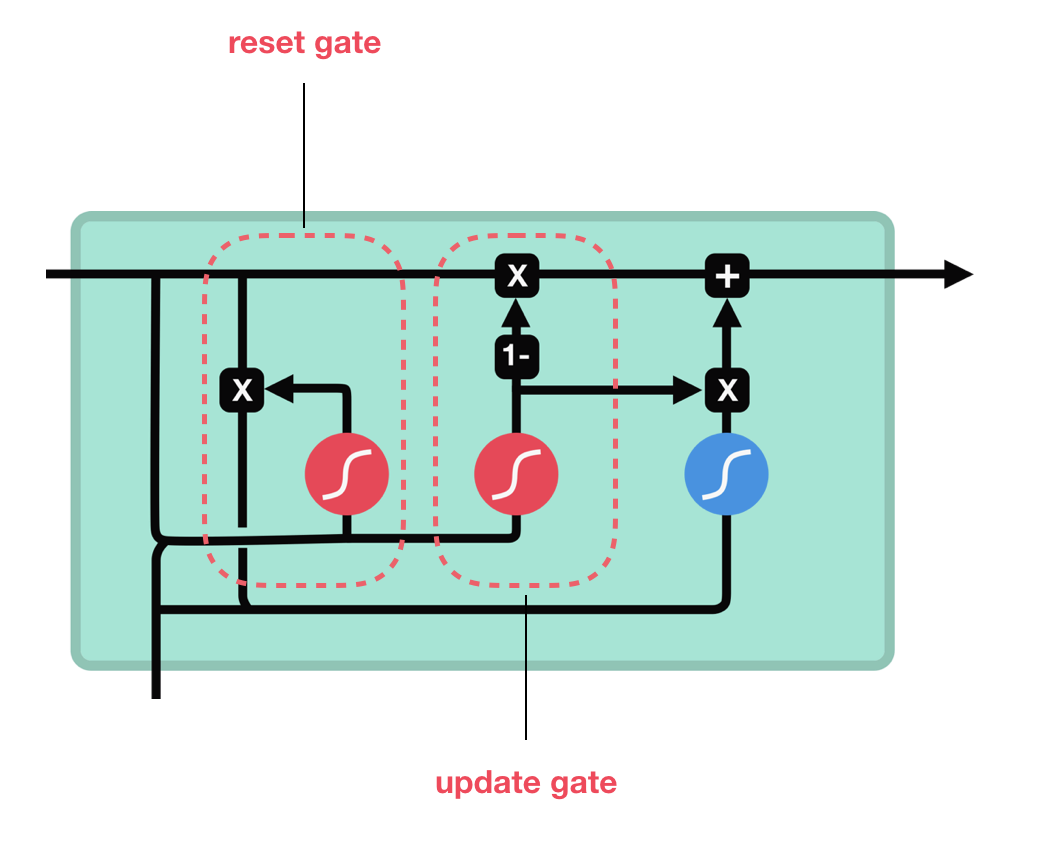
Another variation of the LSTM Networks is the Gated Recurrent Unit.

**Gated Recurrent Units**

When moving from RNN to LSTM, more controlling knobs are used, which control the flow and mixing of inputs and trained weights. Thus, bringing in more flexibility in controlling the outputs. So, LSTM gives us the most controllability and thus, better results. But this also comes with more complexity. A middle ground between RNN and LSTM is the Gated Recurrent Unit, also known as GRU. Due to their structure they are usually faster than a similar LSTM network,

GRUs do not have any cell state, because the hidden state transfers the information. Furthermore it offers only two gates, the Reset and the Update Gate.

* *Update Gate:* The update gate acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add.
* *Reset Gate:* The reset gate is another gate is used to decide how much past information to forget.



**Convolutional Neural Networks**

Convolutional Neural Networks (CNNs) were originally designed for deep learning computer vision tasks, but they have proven highly useful for Natural Language Processing (NLP) tasks as well. A primary use case is sentence classification. Today, CNNs are a state-of-the-art technique helping to automatically classify text strings by emotional sentiment, object categories, urgency, priority, or other characteristics.

CNNs use the concept of a convolution, a sliding window or “filter” that passes over the image, identifying important features and analyzing them one at a time, then reducing them down to their essential characteristics, and repeating the process. In 2014, a paper published proved that CNNs could outperform previous methods of text classification. Even though, most tests were on binomial classification, such as classifying polarity or objectivity, such networks are possible be used for any single- or multi-label classification problem on textual inputs.

A coarse explanation of such uses would be that words, which have been vectorized, are broken up into features and are fed into a convolutional layer. More specifically, the sentence is organized into a matrix, with each row representing a word embedding, a word, or a character. The CNN’s convolutional layer “scans” the text like it would an image, breaks it down into features, and judges whether each feature matches the relevant label or not.

The results of the convolution are “pooled” or aggregated to a representative number. The “pooling” stage, reduces the dimensionality of the word features and retains only a simple probability score that reflects how likely they are to match a label. This number is fed to a fully connected neural structure, which makes a classification decision based on the weights assigned to each feature within the text.

At this point it is important to explain a key feature in Neural Networks, namely backpropagation.

Backpropagation is the computation of the gradient in the weight space of a neural network, with respect to a loss function, such as cross-entropy or mean squared error. The gradient is to be understood as an indicator that shows how the cost changes in the vicinity of the current position respect to the inputs. In other words, it indicates who the output would change if the input was slightly changed.

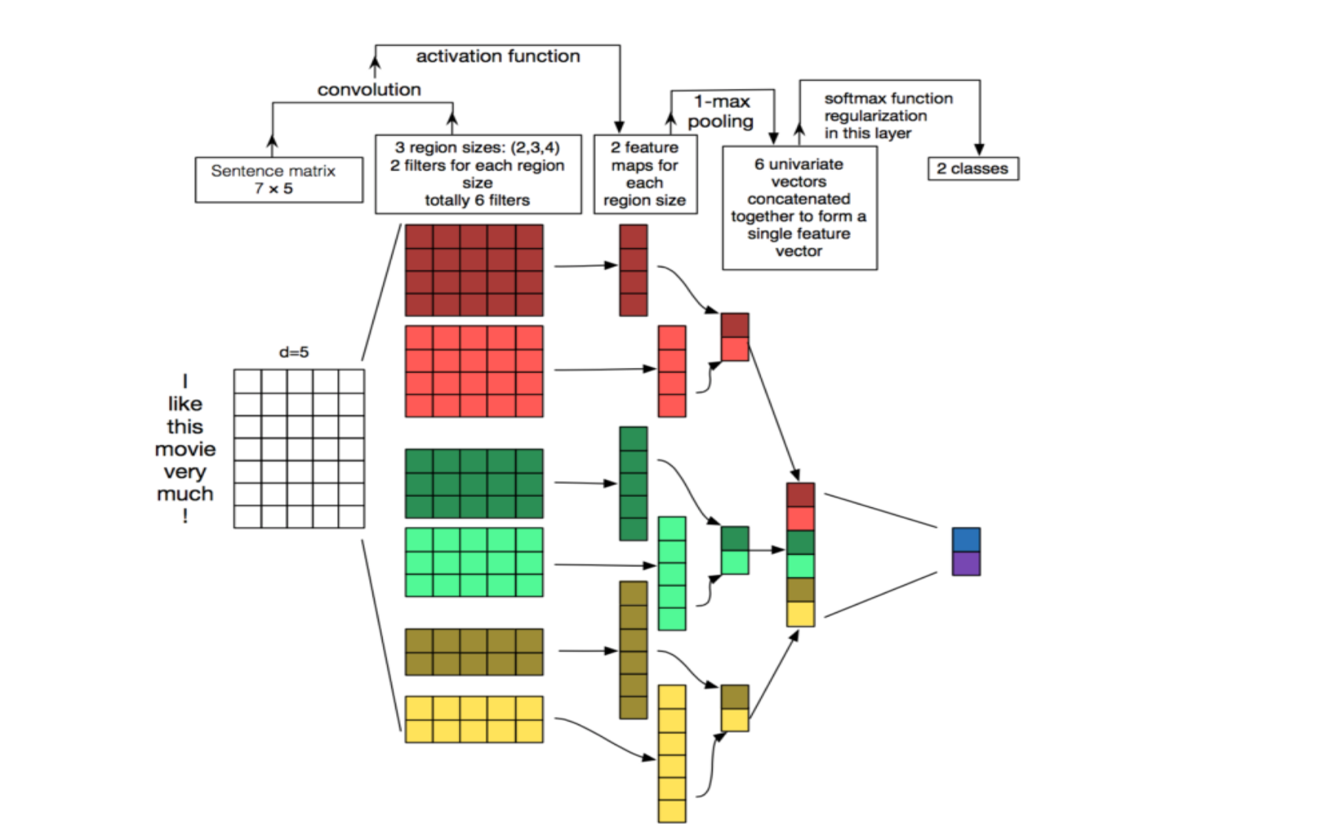
The “fully connected” part of the CNN network goes through its own backpropagation process, to determine the most accurate weights. Each neuron receives weights that prioritize the most appropriate label - for example, “positive sentiment” or “negative sentiment”. Finally, the neurons “vote” on each of the labels, and the winner of that vote is the classification decision.

**3.3 Emotion Extraction using CNN**

As mentioned before a Convolutional Network is an artificial network which has been used for detecting objects in images, whereas other uses such as Sentence Classification, Text Classification, Sentiment Analysis, Text Summarization, Machine Translation and Answer Relations have been recently discovered.

A Convolutional Network is composed of “convolutional” layers and “downsampling” or “subsampling” layers. The convolutional layers comprise neurons that scan their input for patterns, while downsampling layers , or “pooling” layers are often placed after convolutional layers, mainly to reduce the feature map dimensionality for computational efficiency, which can in turn improve actual performance. Typically, the two layers occur in an alternate order , but that’s not necessarily always the case. After this structure there are usually one or more layers of multilayer perceptrons. An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training.

Just like images can be represented as an array of pixel values (float values), similarly we can represent the text as an array of vectors, each word mapped to a specific vector in a vector space composed of the entire vocabulary that can be processed with the help of a CNN. When we are working with sequential data, like text, we work with one-dimensional convolutions, but the idea and the application stays the same. We still want to pick up on patterns in the sequence which become more complex with each added convolutional layer.

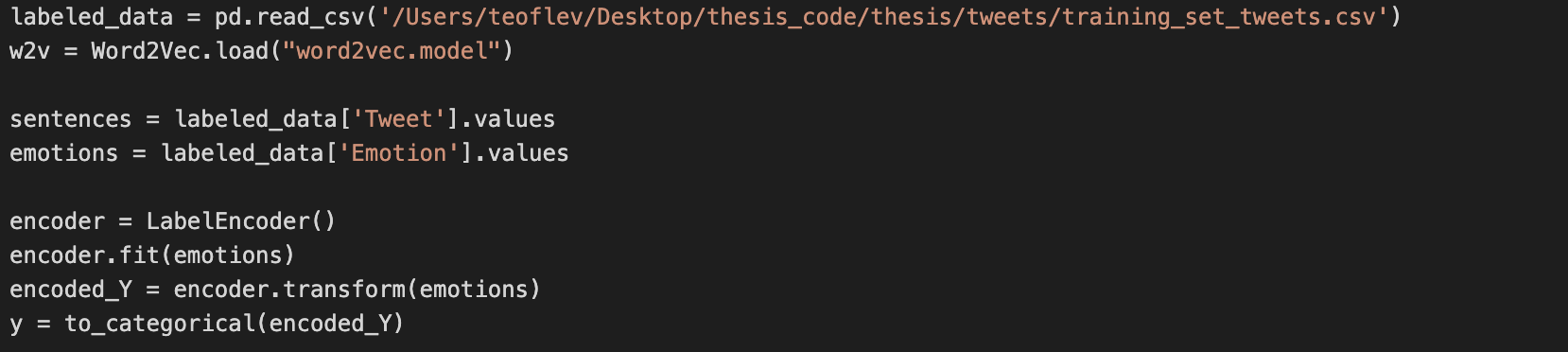


When training a Neural Network there are usually three steps involved.

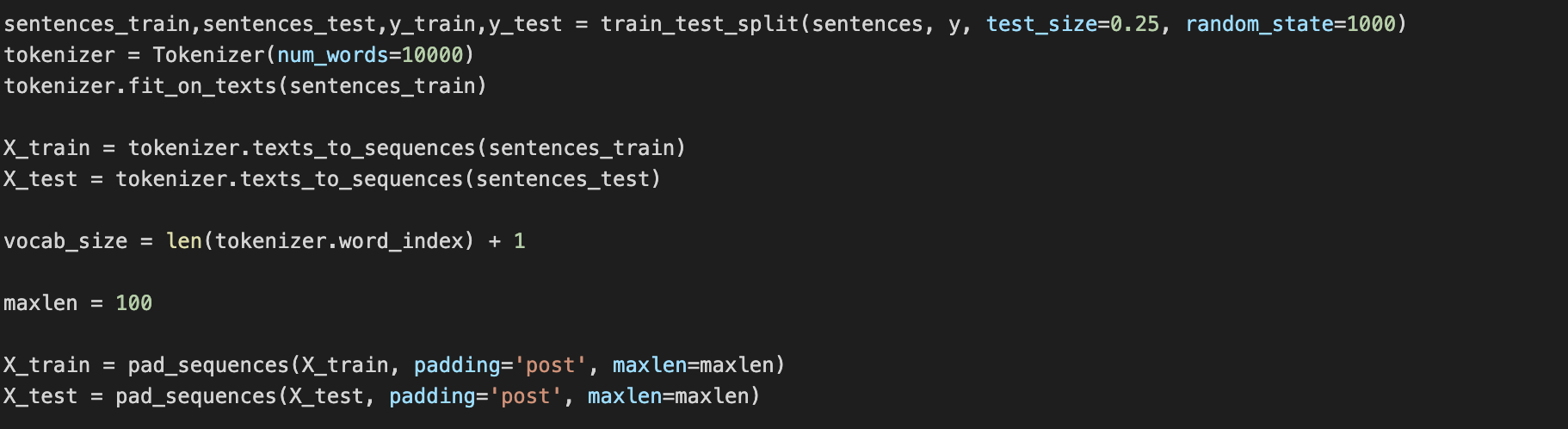
1. Importing the data and preprocessing them into a desirable format
2. Creating word embeddings for our model
3. Loading our data on a CNN architecture and evaluating the accuracy obtained on a validation set.

For the code necessary, there will be heavy use of the Keras library.

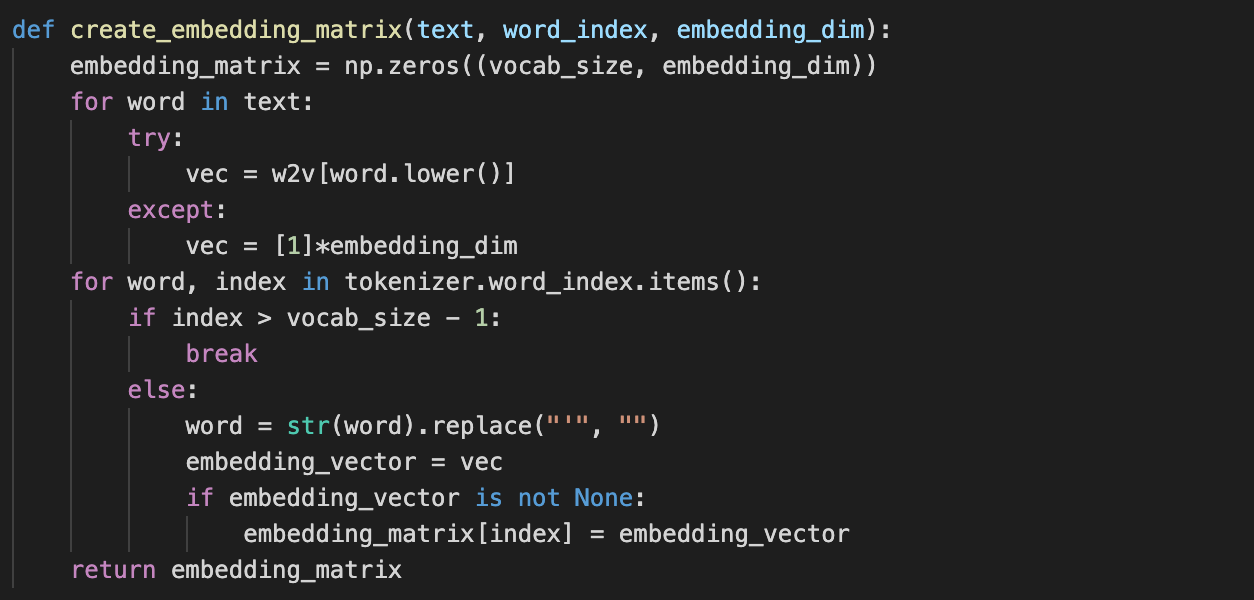
The process starts by preparing the necessary components for the network.

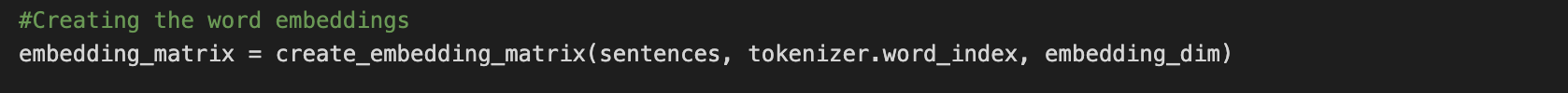


At first, the script reads the .csv file and determines which parts are the actual text and which are the emotion labels. Once the labels are found, the sklearn library’s encoder method encodes the emotion tags in such a way, so that they can be processed by the Convolutional Network. Each emotion label is then formatted to be categorical. A categorical variable is a variable that can take on one of a limited, and usually fixed number of possible values, assigning it to a particular group on the basis of some qualitative property.

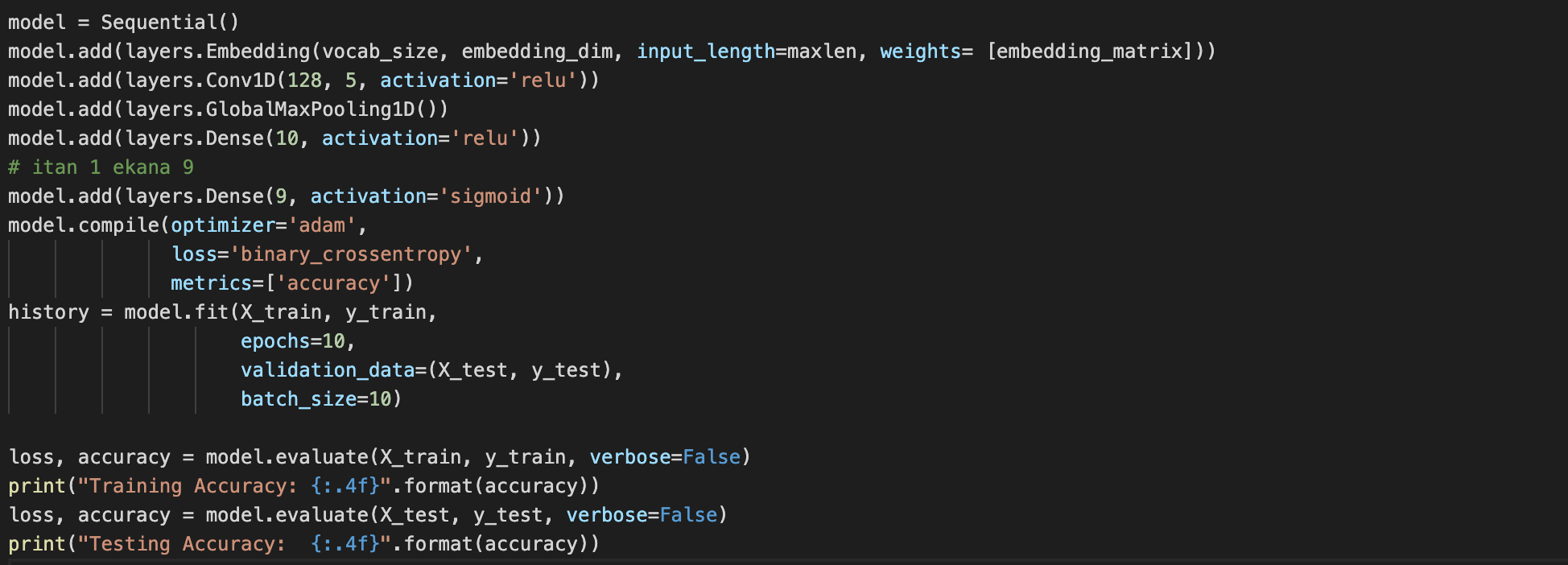


Having completed the preparation of the labels, the script then splits the dataset into training and test data. Sentences of both categories are the are then passed through the *texts\_to\_sequences* method, which transforms each text in texts to a sequence of integers, whereas it takes each word in the text and replaces it with its corresponding integer index value from the dictionary. This way the program can then use the embedding of the corresponding word. Lastly, the newly created sequences are, then, padded to have the same length.





Having already trained the embeddings using the Word2Vec library, we can know take advantage of them by creating a matrix, which contains the vocabulary of the model. This method creates an array where each row represents each unique word, given as a vector from our embeddings model. This works in tandem with the sequences created by the previous part of the script.



At first, the model variable will represent the neural network and will be a sequential model from Keras. The Sequential model is a linear stack of layers, where you can use the large variety of available layers in Keras.

***Layer One:***

The first layer added is the embedding layer. The Keras Embedding layer requires integer inputs where each integer maps to a single token that has a specific real-valued vector representation within the embedding. These vectors are random at the beginning of training, but during training become meaningful to the network. Here we take advantage of the embeddings from the word2vec model.

***Layer Two:***

The second layer is a one-dimensional convolution layer. A 1D CNN is very effective when you expect to derive interesting features from shorter & fixed-length segments of the overall data set and where the location of the feature within the segment is not of high relevance. This layer defines a filter, or also called feature detector, of kernel size 5. Only defining one filter would allow the neural network to learn one single feature in the first layer. This might not be sufficient; therefore, we will define 128 filters. This allows to train 128 different features on the first layer of the network.

***Layer Three:***

The third layer is a max pooling layer. A pooling layer is often used after a CNN layer in order to reduce the complexity of the output and prevent overfitting of the data. The default value is a value of 2, which means that the size of the output matrix of this layer is only half of the input matrix.

***Layer Four & Five:***

A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected. The layer has a weight matrix **W,**a bias vector **b,**and the activations of previous layer **a.**

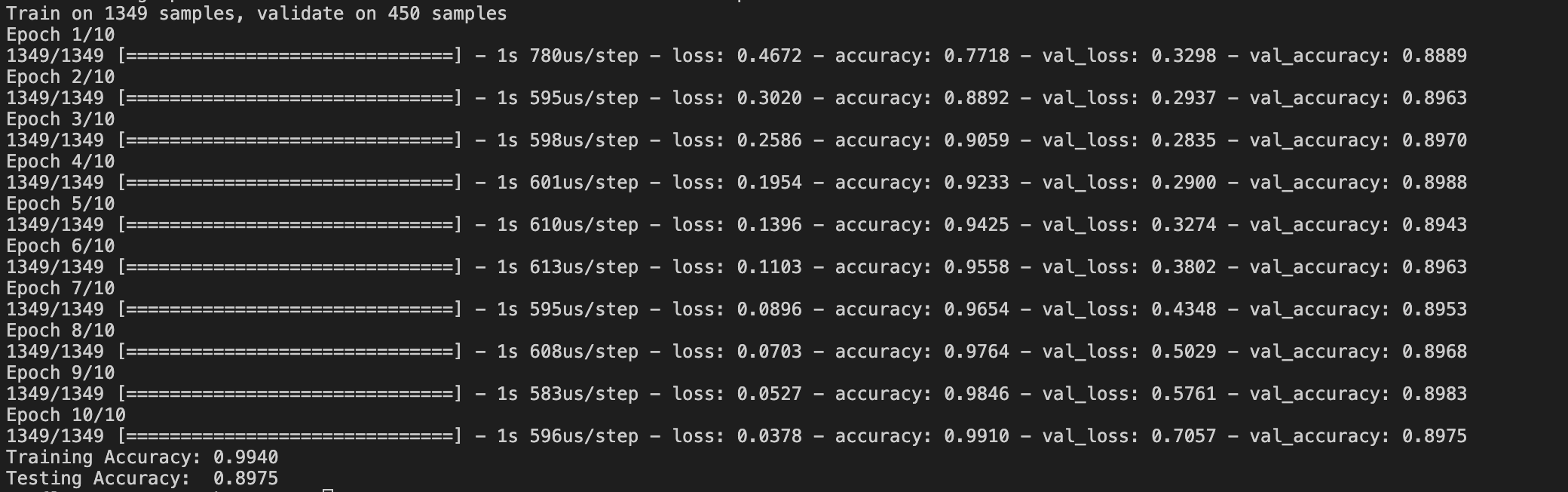
According to the official Keras documentation:

“output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer. ”

The first parameters regard the dimensionality of the result whereas the second parameter is the activation function being used.

***Final Stage:***

After the layers have been placed, the CNN compiles with the loss function being binary cross-entropy and the metric being accuracy, meaning the fraction of correct predictions and the total predictions the model made. The neural network will train for 10 epochs, with the result being displayed in the following image and the **model having an accuracy of approx. 90%.**



**References**

1.     Hu, Minqing and Liu, Bing (2004). “Mining Opinion Features in Customer Reviews”.

2.    Dave, K., Lawrence, S., and Pennock, D.M (2003). “Mining the peanut gallery: Opinion extraction and semantic classification of product reviews”.

3.  Ku, Lun-Wei & Liang, Yu-Ting & Chen, Hsin-Hsi. (2006). “Opinion Extraction, Summarization and Tracking in News and Blog Corpora. AAAI Spring Symposium - Technical Report”.

4.     E. A. Weiss (1992). "Arthur Lee Samuel (1901-90)". *IEEE Annals of the History of Computing*

5.     University of Washington (2006), “The History of Artificial Intelligence”

6.     Giulio Angiani, “A Comparison between Preprocessing Techniques for Sentiment Analysis in Twitter “

7.     Mikolov, Tomas; et al. (2013). "Efficient Estimation of Word Representations in Vector Space".

1. Phillip Cole (2014), “Beyond reason: the philosophy and politics of immigration”
2. Pyle, D. (1999) “*Data Preparation for Data Mining.* Morgan Kaufmann Publishers”
3. Konstantinos Korovesis (2018), “Sentiment Analysis for Tweets”
4. Chen, Yanqing, Skiena (2014), “Building Sentiment Lexicons for all major Languages”
5. Saif M. Mohammad and Peter D. Turney (2013), “Crowdsourcing a Word–Emotion Association Lexicon”
6. Armin Seyeditabari, Narges Tabari, Wlodek Zadrozny (2018), “Emotion Detection in Text: a Review”
7. Ryan Rifkin (2008), “Multiclass Classification”
8. V. Vapnik,(2000), ”The nature of statistical learning theory (2nd ed.)”
9. Stephan Dreiseitl (2002), “Logistic regression and artificial neural network classification models: a methodology review”
10. Anukrati Mehta (2019), “A Comprehensive Guide to Types of Neural Networks”
11. Yoon Kim (2014), “Convolutional Neural Networks for Sentence Classification”
12. C.-C. Jay Kuo (2016), “Understanding Convolutional Neural Networks with A Mathematical Model”