



Master of Science  
<<Artificial Intelligence and Visual Computing>>

# UNIVERSITY OF WEST ATTICA & UNIVERSITY OF LIMOGES

**FACULTY OF ENGINEERING**  
**DEPARTMENT OF INFORMATICS AND COMPUTER ENGINEERING**

**Master Thesis**

**Assessment and comparison of existing methods and  
datasets for sentiment analysis of Greek texts**

**Student: Fragkis Nikolaos  
(aivc20001)**

**Supervisor: Tselenti Panagiota**

**Athens, June 2022**



**ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ  
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ  
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ**

**Τίτλος εργασίας**

**Assessment and comparison of existing methods and datasets for sentiment analysis of Greek texts**

**Μέλη Εξεταστικής Επιτροπής συμπεριλαμβανομένου και του Εισηγητή**

Η πτυχιακή/διπλωματική εργασία εξετάστηκε επιτυχώς από την κάτωθι Εξεταστική Επιτροπή:

<b>A/a</b>	<b>ΟΝΟΜΑ ΕΠΩΝΥΜΟ</b>	<b>ΒΑΘΜΙΔΑ/ΙΔΙΟΤΗΤΑ</b>	<b>ΨΗΦΙΑΚΗ ΥΠΟΓΡΑΦΗ</b>
1	Παναγιώτα Τσελέντη	ΕΔΙΠ	
2	Πάρις Μαστοροκόστας	Καθηγητής	
3	Αναστάσιος Κεσίδης	Αναπληρωτής Καθηγητής	

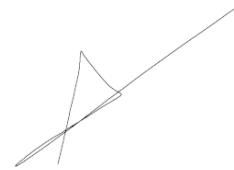
## ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ/ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Φραγκής Νικόλαος του Αναστασίου, με αριθμό μητρώου αίνε20001 φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της πτυχιακής/διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών



[https://www.uniwa.gr/wp-content/uploads/2021/01/%CE%A0%CE%BF%CE%BB%CE%B9%CF%84%CE%B9%CE%BA%CE%B5%CC%81%CF%82\\_%CE%99%CE%B4%CF%81%CF%85%CE%BC%CE%B1%CF%84%CE%B9%CE%BA%CE%BF%CF%85%CC%81\\_%CE%91%CF%80%CE%BF%CE%B8%CE%B5%CF%84%CE%B7%CF%81%CE%B9%CC%81%CE%BF%CF%85\\_final.pdf](https://www.uniwa.gr/wp-content/uploads/2021/01/%CE%A0%CE%BF%CE%BB%CE%B9%CF%84%CE%B9%CE%BA%CE%B5%CC%81%CF%82_%CE%99%CE%B4%CF%81%CF%85%CE%BC%CE%B1%CF%84%CE%B9%CE%BA%CE%BF%CF%85%CC%81_%CE%91%CF%80%CE%BF%CE%B8%CE%B5%CF%84%CE%B7%CF%81%CE%B9%CC%81%CE%BF%CF%85_final.pdf)

## ABSTRACT

*Sentiment Analysis is a well-known field of Natural Language Processing that is concerned with text classification. There is a vast number of papers, especially for the English language, that present state-of-the-art results on many different datasets using a variety of classification models. The aim of this work is to compare machine learning models on different datasets in both Greek and English. In order to achieve this aim, we used the well-known IMDb dataset from Stanford University, which is very often used for the evaluation of new text classification models, and one equivalent new dataset that we created in Greek from the Athinorama website. For our experiments, we used the following models: Logistic Regression, Support Vector Machine, Naïve Bayes, Decision Trees, XGBoost, Convolutional Neural Network, Long Short-Term Memory, Gated Recurrent Units, and Bidirectional Encoder Representations from Transformers (BERT). The first five models were combined with the TF-IDF vectorization technique, while the rest were combined with the Word Embeddings vectorization technique. The results show that the best classifier for sentiment analysis for both English and Greek is the pretrained BERT model. The difference in language does not seem to have a significant impact on the results, whereas the quality, the size, and the level of pre-processing of the data appear to play a significant role in the classification process. The reason we chose to deal with this work is the lack of research for the Greek language and our contribution is the Athinorama Light dataset that could play a significant role in future works for Greek language classification issues.*

# Table of Contents

ABSTRACT .....	3
1. INTRODUCTION.....	6
2. TEXT CLASSIFICATION MODELS.....	8
2.1. Logistic Regression.....	8
2.2. Support Vector Machine.....	9
2.3. Naïve Bayes.....	11
2.4. Decision Tree.....	11
2.5. XGBoost.....	12
2.6. Neural Networks .....	12
2.7. Convolutional Neural Network (CNN) .....	13
2.8. Recurrent Neural Network (RNN) .....	14
2.9. Bidirectional Encoder Representations from Transformers (BERT).....	14
3. RELATED WORK ON SENTIMENT ANALYSIS .....	15
3.1. Machine Learning Methods.....	15
3.2. Deep Learning Methods.....	17
3.3. Methods based on Pre-Trained Models .....	19
3.4. Sentiment Analysis for Greek Language.....	21
4. METHODOLOGY .....	24
4.1. Datasets.....	24
English Dataset.....	24
Greek Dataset.....	24
Data Analysis.....	26
4.2. Preprocessing .....	28
4.3. Vectorization .....	31
4.4. Accuracy Metrics.....	32
4.5. Tools and Libraries .....	33
5. EXPERIMENTS.....	34
6. RESULTS.....	35
6.1. Results for the English Language .....	35
6.2. Results for the Greek Language .....	37
7. CONCLUSIONS.....	41

References.....	42
Appendix .....	45
ML_models.py.....	45
DL_models.py .....	51
BERT_models.py (colab code) .....	58

## List of Figures

Figure 1 Linear Regression Graph (Image source Wikipedia).....	6
Figure 2 Logistic Regression Graph (Image source Wikipedia) .....	7
Figure 3 Support Vector Machines, Decision Boundary (Image source Wikipedia) .....	8
Figure 4 Decision Tree, Boundaries (Xiaoli Fern, 2008).....	10
Figure 5 Convolutional Neural Network Structure (Image source Brilliant.org) .....	12
Figure 6 Mean tokens per review, for IMDb and Athinorama Light .....	24
Figure 7 Most frequent words in IMDb .....	25
Figure 8 Most frequent words in Athinorama Light.....	25
Figure 9 Graph of IMDb experiments – No Preprocessing.....	34
Figure 10 Graph of IMDb experiments – No Preprocessing.....	35
Figure 11 Graph of Athinorama Light experiments – No Preprocessing .....	36
Figure 12 Graph of Athinorama Light experiments – Full Preprocessing .....	37
Figure 13 Graphs of F1-score Comparison .....	38

## List of Tables

Table 1 Classification Results (Tripathi et al., 2020) .....	15
Table 2 Classification Results (Haque et al., 2019).....	17
Table 3 Classification Results (Sousa et al., 2019) .....	18
Table 4 Classification Results (Markopoulos et al., 2015).....	20
Table 5 Classification Results (Spatiotis et al., 2016).....	20
Table 6 Datasets comparison.....	26
Table 7 Greek language example .....	28
Table 8 English language example .....	29
Table 9 Figure 9 Results of IMDb experiments – No Preprocessing.....	34
Table 10 Results of IMDb experiments – Full Preprocessing.....	35
Table 11 Results of Athinorama Light experiments – No Preprocessing.....	36
Table 12 Results of Athinorama Light experiments – Full Preprocessing .....	37
Table 13 F1-score Comparison.....	38

# 1. INTRODUCTION

For thousands of years, people have been working on speech decoding in a way that other people could recognise and understand in order to render something intangible like the human voice into something tangible and easily manageable like a written text. The first samples of this process were found in Mesopotamia and Egypt between 3400 and 3100 BC., with the invention of Cuneiform and Hieroglyphics respectively as the earliest forms of writing. In the following centuries, new ideas and new necessities gave birth to new forms of writing that consisted of alphabets and decimal numeral systems, with each step of the development leading to a higher level of written form of the language. From the Renaissance until the end of the Industrial Revolution, the development of linguistics led to the creation of some less essential parts of writing in order to adapt to new technologies such as the invention of typography which spread the use of fonts. All these developments, along with others exclusively invented for the new digital revolution, proved to be useful tools for Natural Language Processing.

Natural Language Processing (NLP) is the part of computer science that deals with the automatic analysis and representation of the human language (Cambria & White, 2014). It started in 1950 with Alan Turing's paper "Computing Machinery and Intelligence" (Turing, 1950), which led the race that was later started by the scientific community regarding the development of algorithms and hardware in order to make machines solve problems that normally need human-like reasoning to solve, not only in NLP but also in other parts of Computer Science and Artificial Intelligence.

In 1954, the Georgetown–IBM experiment (Hutchins, 2004) achieved the first encouraging results regarding machine translation, where sixty sentences in Russian were translated into English. However, the rule-based approach proved to be inadequate and future development in this area failed to meet expectations.

In the middle of the 1960s, the first chatbot named ELIZA was created at MIT by Joseph Weizenbaum (Amity University et al., 2020). ELIZA was the first attempt at creating a machine that could communicate on a basic level with a human in chat form. Its structure was quite simple, without many capabilities besides answering questions in a generic way using keywords from the user's previous messages. Nevertheless, in 1972 Kenneth Colby created a new implementation of the ELIZA chatbot named PARRY, which was programmed to mimic the behavior of a patient suffering from paranoid schizophrenia. PARRY managed to pass the Turing Test 52% of the time when it was tested during the 1970s (Amity University et al., 2020).

In the 1980s, we witnessed the introduction of machine learning algorithms in NLP due to the increase in the computational power of computers. That meant that the field of NLP was moving away from the rule-based architecture of the programs, which was very restrictive, and was entering a world with unlimited potential based on statistical methods. The first samples of this new approach to NLP appeared in the field of machine translation in 1988 at IBM Research Division (P.Brown et al, 1988), where statistical methods were applied to large multilingual textual corpora created by the Parliament of Canada. The results were a step forward compared to previous approaches; however, the problem was not considered solved due to the non-generalizability of the system.

In the late 1990s, Deep Neural Networks began to appear in real-world problems such as speech recognition, where a new recurrent neural network called Long Short-Term Memory (Hochreiter & Schmidhuber, 1997) managed to avoid the vanishing gradient problem and achieved great results at the time. In the following years, the research was directed towards neural networks with new models such as the Convolutional Neural Networks and Transformers (Vaswani et al., 2017).

Over the last decade, the growth of social media platforms and online encyclopedias such as Twitter and Wikipedia, alongside the creation of many coding tools and frameworks like NLTK, made NLP applications very popular in non-academic fields, especially in the digital industry.

Major research organizations such as Alphabet Inc.<sup>1</sup>, OpenAI<sup>2</sup> and Big Tech companies<sup>3</sup> developed great applications such as Google Translate, which works very well, especially with small texts, Amazon's and Apple's virtual assistants, which can answer questions, make recommendations and perform actions as well as Facebook's Ad Targeting, which can target consumers based on their Facebook posts.

Apart from these technological giants, it is now quite common even for small commercial companies to have their own efficient chatbot on their websites or their own data science team whose job is to analyze data extracted from social media monitoring and customer reviews in order to improve their decision-making skills. Furthermore, in recent years, NLP has emerged in the political field where sentiment analysis techniques are used to predict voters' intentions based on the comments they post on online social networks.

In this work, we focus on sentiment analysis of text data with the use of supervised machine learning algorithms. Sentiment Analysis is a part of Artificial Intelligence that lies in the intersection of NLP and Natural Language Understanding. Its main purpose is to extract and analyze the affective states of a text's content (Cambria et al., 2016). This issue is divided into three main categories: a) the identification of sentiment polarity, where the goal is to classify the content of a given text as positive, negative, or neutral, b) the identification as objective or subjective text, and c) the emotion detection, where the goal is to identify emotions in a text as furious, cheerful, depressed, disgusted etc. (Cambria, 2016; Markopoulos et al., 2015; Nandwani & Verma, 2021; Tsakalidis et al., 2018). The most popular techniques for performing sentiment analysis are the following three: knowledge-based techniques, statistical methods, and hybrid approaches. Knowledge-based techniques are usually applied with the use of a Sentiment Lexicon, which is a list of semantic features for words and phrases that are applied to the texts we want to analyze. Statistical methods are based on machine learning algorithms such as SVM and Logistic Regression, or deep learning models such as CNNs, RNNs, and pretrained models such as BERT. Finally, the hybrid approaches are a combination of both knowledge-based and statistical methods. These techniques encompass a large part of the machine learning spectrum (for sentiment analysis) since knowledge-based techniques and statistical methods are classified as unsupervised and supervised methods, respectively. In addition to politics and social media monitoring, sentiment

---

<sup>1</sup> [https://en.wikipedia.org/wiki/Alphabet\\_Inc.](https://en.wikipedia.org/wiki/Alphabet_Inc.)

<sup>2</sup> <https://en.wikipedia.org/wiki/OpenAI>

<sup>3</sup> [https://en.wikipedia.org/wiki/Big\\_Tech](https://en.wikipedia.org/wiki/Big_Tech)



analysis is used both in product analysis and market & competitor research. The objective of this work is the application of sentiment analysis with supervised statistical methods in movie reviews both in Greek and in English.

In the past, the research on sentiment analysis regarding the English language achieved exceptionally good results in many cases, but for the Greek language the effort was poor due to the absence of labeled datasets that even today are difficult to find as well as the absence of coding tools for the preprocessing of text data in the Greek language. Today, neither of these problems are considered insurmountable as web scraping techniques and the proliferation of websites with reviews in Greek can solve the first problem, while the creation of spaCy, which can support the Greek language, can solve the second one.

## 2. TEXT CLASSIFICATION MODELS

### 2.1. Logistic Regression

Logistic Regression is a statistical model relevant to the Linear Regression model, which depicts the relationship between an independent and a dependent variable with a straight line in the following manner. Given a data set in the following form:

$$\{y_i, x_{i1}, x_{i2}, \dots, x_{ip}\}_{i=1}^n$$

of  $n$  paradigms where  $Y$  is the dependent variable and  $x^t$  is the independent variables, the Linear Regression model assumes that the relationship between  $Y$  and  $x^t$  is linear and is given by the following equation:

$$y_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip} + \varepsilon_i$$

where  $i = 1, 2, 3, \dots, n$  and  $\varepsilon$  is the error or the “noise” of the model ( $\varepsilon$  essentially gives us the parallel shift of the paradigm from the line). This relationship can be visualized as follows:

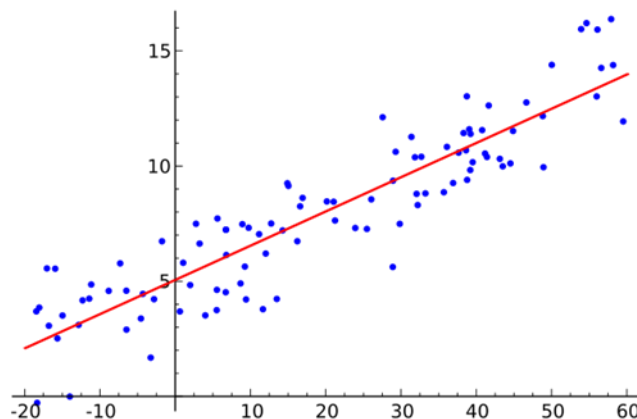


Figure 1 Linear Regression Graph (Image source Wikipedia)

The optimization of the model is given by the minimization of the error. As a result of this process, we have the change of the weights of the mentioned equation. Now the model can predict the Y value of a paradigm by applying  $X^t$  to the resulting equation. This statistical model is very useful when the dependent variable Y is continuous, but it is not effective with a categorical dependent variable (for example True or False). Here enters Logistic Regression which solves the problem with a logistic equation instead of a linear one. The following equation enables us to manage binary problems:

$$\log\left(\frac{Y}{1-Y}\right) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}$$

Now for a certain paradigm  $X^t$  the result is a number between 0 and 1 or, in other words, a probability that gives us the opportunity to classify a paradigm in distinct groups. The logistic equation is visualized in the graph below:

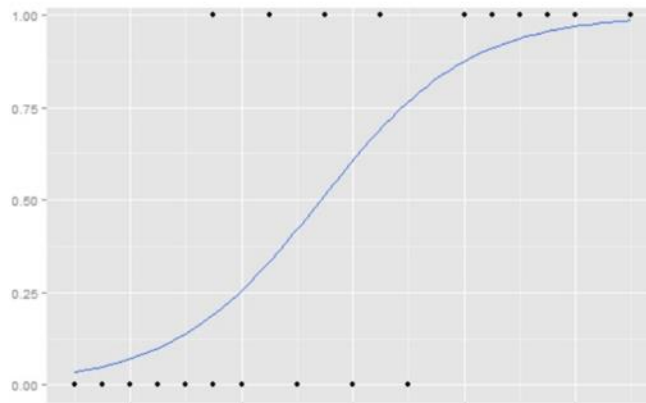


Figure 2 Logistic Regression Graph (Image source Wikipedia)

Here the default threshold is 0.5. Everything above 0.5 is classified as 1 and everything below 0.5 is classified as 0. All these features render this model ideal for classification problems (Jianqiang & Xiaolin, 2017; Tripathi et al., 2020). Other applications that use the Logistic Regression model, are, apart from text classification, image segmentation and handwriting recognition.

## 2.2. Support Vector Machine

Support Vector Machine (SVM) in machine learning is a supervised learning model that uses a decision boundary as an essential element to classify data. In the case of linearly separable data, the model works as follows. Given a dataset in the following form:

$$\{y_i, x_{i1}, x_{i2}, \dots, x_{ip}\}_{i=1}^n$$

of  $n$  paradigms where  $Y$  is the dependent variable (labels,  $y_1 = 1$  and  $y_2 = -1$ ) and  $X^t$  is the independent variables (attributes), SVM creates a separating hyperplane that splits the data into two sets with the largest possible margin. The hyperplane's function has the following form:

$$f(X) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}$$

$$y_i = \begin{cases} 1, \wedge \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip} \geq 0 \\ -1, \wedge \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip} < 0 \end{cases}$$

and the decision boundary is given by the solution of the following equation:

$$0 = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}$$

The correct location of the hyperplane is given by maximizing the margin which is defined by the data of the two different sets. The following graph visualizes the result of this procedure:

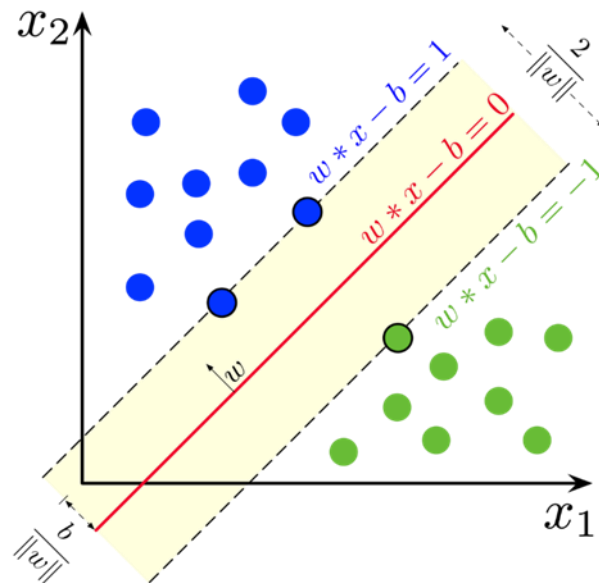


Figure 3 Support Vector Machines, Decision Boundary (Image source Wikipedia)

If the data are not linearly separable, then a nonlinear mapping is used to transform the data in another space of higher dimension, so a linear boundary can separate them. In Sentiment Analysis, SVM is probably one of the most successful machine learning models (Gautam & Yadav, 2014; Markopoulos et al., 2015; Rumelli et al., 2019; Neethu & Rajasree, 2013), but it is also used in several other machine learning problems like face recognition, image classification and handwriting recognition.

## 2.3. Naïve Bayes

Naïve Bayes is another classifier borrowed from statistics based on Bayes theorem that answers the following probabilistic problem: what is the probability of an event happening based on prior knowledge of conditions that might be related to that event? This problem is visualized with the following equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(A \cap B)}{P(B)}$$

where A and B are events and  $P(B) \neq 0$

In simple words, this model demonstrates that if there are two events A and B, then the probability of A happening, given that B has already happened, can be calculated by dividing the probability of the intersection of the two events by the probability of B. In a classification problem that could be translated as follows. Given a data set:

$$\{y_i, x_{i1}, x_{i2}, \dots, x_{ip}\}_{i=1}^n$$

of n paradigms where Y is the dependent variable (set) and  $X^t$  is the independent variables (attributes), then the probability of the paradigm x belonging in the class y is  $P(y|x)$ . Although Naïve Bayes is a well-known classifier for sentiment analysis (Jianqiang & Xiaolin, 2017; Tripathi et al., 2020), it is also often used in medical classification, multi-class classification and real time prediction due to the low computational needs of the algorithm.

## 2.4. Decision Tree

Decision Tree is a decision model used in statistics and machine learning based on observations (or variables) about an item that are represented as the branches of a tree and conclusions (or labels) about the item that are represented as the leaves of the tree. There are two types of decision trees: a) regression trees, where the target variable can take continuous values, and b) classification trees, where the target variable can take distinct values and essentially represents the labels. Classification trees are used in this work because of the nature of the problem which requires a segmentation of the space of the paradigms based on their features in a distinct way. This procedure can be visualized by the following graph:

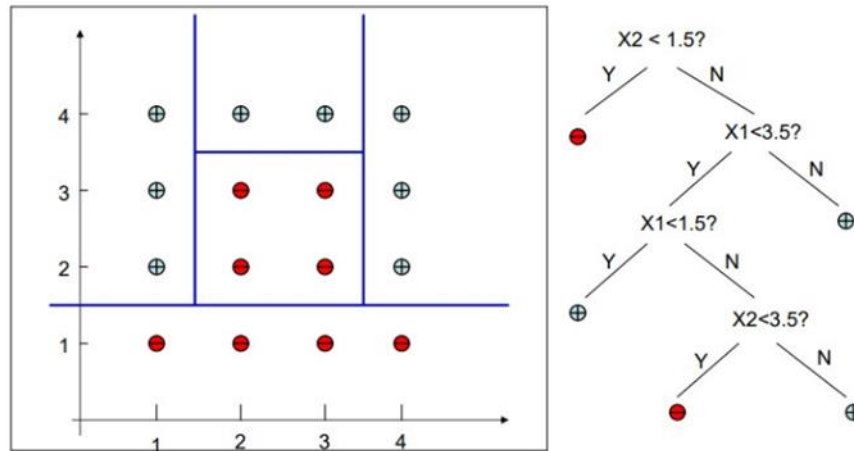


Figure 4 Decision Tree, Boundaries (Xiaoli Fern, 2008)

The benefit of decision trees is that the data do not require a lot of preprocessing as this type of models can handle both numerical and categorical data. Another advantage of decision trees is that the data that we want to classify do not need to be linearly separable, as shown in the graph above. However, the main disadvantage of decision trees is that a large dataset could create a very complex tree which would be very sensitive to overfitting.

## 2.5. XGBoost

XGBoost, or Extreme Gradient Boosting, is a classification and regression algorithm. It is based on the gradient boosting technique which employs a combination of weak prediction models (most of the time, decision trees) in order to create a stronger model with better performance avoiding the above-mentioned overfitting problem. XGBoost was created by Tianqi Chen (Chen, 2016) with the basic goal of the scalability of the algorithm in all scenarios. Experiments showed that the system runs more than ten times faster than existing popular solutions on a single machine (Chen, 2016) and most of the times it achieves state-of-the-art results. Therefore, XGBoost has been a very popular algorithm in recent years (Afifah et al., 2021; Hu et al., 2021) in many fields of machine learning with great success in problems like store sales prediction, web text classification, customer behavior prediction, and malware classification.

## 2.6. Neural Networks

Neural Networks are computing systems inspired by the biological neural networks. A neural network is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron receives a signal, processes it and then signals neurons connected to it. The "signal" at a connection is a number, and the output of each neuron is computed by some

non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer) to the last layer (the output layer), possibly after traversing the layers multiple times. Neural networks are trained by processing examples, each of which contains a known "input" and "result", forming probability-weighted associations between the two which are stored within the data structure of the net itself. The training of a neural network from a given example is usually conducted by determining the difference between the processed output of the network (often a prediction) and a target output. This difference is the error. The network then adjusts its weighted associations according to a learning rule and uses this error value. Successive adjustments will cause the neural network to produce output which is increasingly similar to the target output. After a sufficient number of these adjustments, the training can be terminated based upon certain criteria.

## 2.7. Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is an artificial neural network that specializes in processing data that have a grid-like topology such as images. CNNs are made up of three kinds of layers; the convolutional layers, the pooling layers, and the fully connected layer. The convolutional layer is the first layer of a convolutional network and it is where the majority of computation occurs. The components it requires are input data, a filter, and a feature map. The pooling layer performs dimensionality reduction, reducing the number of parameters in the input. In the fully-connected layer, each node in the output layer connects directly to a node in the previous layer. This layer performs the task of classification based on the features extracted through the previous layers and their different filters. CNNs are very efficient in problems related to image recognition and image classification as they are designed to perform in such problems, but they have also achieved great results in other parts of AI like NLP (Haque et al., 2019; Yenter & Verma, 2017). The following image shows the structure of a simple CNN:

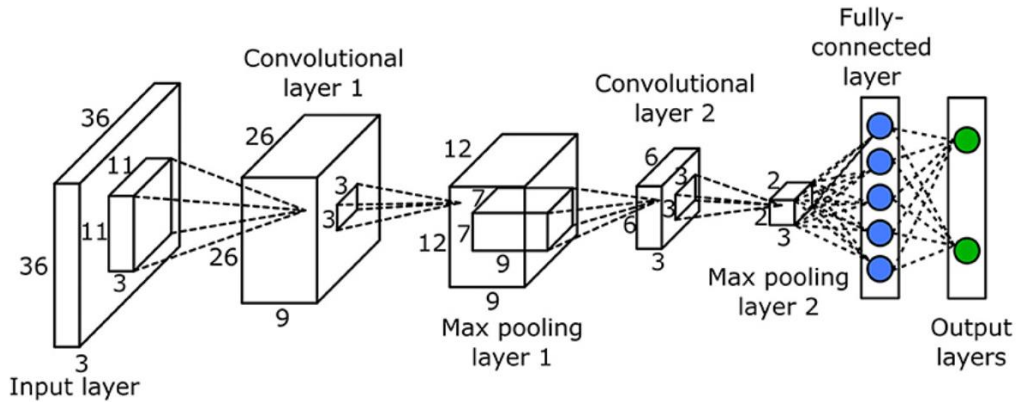


Figure 5 Convolutional Neural Network Structure (Image source Brilliant.org<sup>4</sup>)

## 2.8. Recurrent Neural Network (RNN)

A Recurrent Neural Network, or RNN, is a type of artificial neural network which uses sequential data or time series data. Like feedforward and CNNs, RNNs utilize training data to learn. They are distinguished by their “memory” as they take information from prior inputs to influence the current input and output. Well-known RNNs are Long Short-Term Memory, or LSTM, and Gated Recurrent Units, or GRU. In this work, we used both models in our experiments. Long Short-Term Memory, or LSTM, (Hochreiter & Schmidhuber, 1997) is a recurrent neural network used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections and can process not only single data points such as images, but also entire sequences of data such as speech or video. LSTMs are specifically designed to avoid the long-term dependency problem. A typical feature of an LSTM is that it can retain information for long periods of time, something that is very useful in text classification problems. Gated Recurrent Units, or GRU, (Merri, 2013) is a recurrent neural network that aims to solve the vanishing gradient problem resulting from a standard recurrent neural network. GRU can also be considered a variation of LSTM because both are designed similarly and, in some cases, achieve equally excellent results. RNNs are specially designed for applications related to NLP and they have great success in problems such as text classification, speech recognition and machine translation.

## 2.9. Bidirectional Encoder Representations from Transformers (BERT)

Bidirectional Encoder Representations from Transformers, or BERT, is a transformer-based machine learning model for NLP developed by Google (Devlin et al., 2019). BERT is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-

<sup>4</sup> <https://brilliant.org/wiki/convolutional-neural-network/>

of-the-art models for a wide range of tasks such as question answering and language inference, without substantial task-specific architecture modifications. In this work, we will also use the Greek BERT<sup>5</sup> which is the Greek edition of BERT's model created by the Natural Language Processing Group of Athens University of Economics and Business<sup>6</sup>. BERT is a relatively new algorithm, but it has managed to achieve great results in many cases related to NLP including sentiment analysis on the IMDb dataset that we will use in this work (Gao et al., 2019; Sanh et al., 2019; Sousa et al., 2019; Sun et al., 2019)

### 3. RELATED WORK ON SENTIMENT ANALYSIS

The essential idea behind this work is twofold: first, the application of supervised machine learning models for sentiment analysis on an English dataset that is well established and has produced excellent results in the past, and second, the application of these models on a new Greek dataset that was made with the same principles as the ones employed for the English one in order to evaluate and compare the results and draw useful conclusions for applications in real problems, mainly in the Greek language. Therefore, this part of this work is divided into two sections; the first section is dedicated to research work on sentiment analysis for the English language, and the second one is dedicated to research work on sentiment analysis for the Greek language.

#### 3.1. Machine Learning Methods

In this first part of the related work concerning the English language, we will examine papers that deal with machine learning models for sentiment analysis such as Support-Vector Machines and Naive Bayes. Such techniques were applied in the following paper where the researchers tried to analyze posts about electronic products like mobile phones, laptops, etc. using a machine learning approach (Neethu & Rajasree, 2013) on a dataset that was created from Twitter. Twitter sentiment analysis is very difficult compared to general sentiment analysis because of the nature of the data that come from this platform. These data are very limited in length (only 140 characters each) and usually contain slang and spelling mistakes. The dataset that the researchers created using Twitter's API contained 1,000 tweets for training (500 positive and 500 negative) and 200 tweets for testing (100 positive and 100 negative) which were manually annotated. The pre-processing part of the process included the removal of the URLs, the correction of the spelling mistakes, and the replacement of the slang words. The feature extraction was performed in two steps. First, the researchers gave weights on hashtags and emoticons because these characteristics can provide a lot of information about the sentiment of a tweet, and then they described the tweets as a collection of words using unigrams. After the creation of the feature vectors, they trained four different models: Naive Bayes, Support Vector Machine, Maximum Entropy, and Ensemble classifiers. Ensemble

---

<sup>5</sup> <https://github.com/nlpaueb/greek-bert>

<sup>6</sup> <http://nlp.cs.aueb.gr/>



classifiers, as the name suggests, is the combination of classifiers with the classification results being given by the voting rule. In our case, they used the Naive Bayes, Support Vector Machine, and Maximum Entropy classifiers. The results showed that all these models had a similar performance except for Naïve Bayes which achieved slightly better results in precision but worse results in accuracy and recall. In addition, the results showed that all these models and preprocessing techniques performed well in tweets classification. These algorithms were implemented using built-in Matlab functions except for the Maximum Entropy model that was implemented using the MaxEnt software.

Another paper on sentiment analysis with similar models applied to tweets is the following (Gautam & Yadav, 2014), where the researchers deal with sentiment analysis of customer reviews. The data that were used in this paper concerned labeled tweets that were split into two high polarity categories (positive and negative) using the unigram feature extraction technique. The dataset contained 19,340 tweets; 18,340 for the training of the models and 1,000 for the testing. In the preprocessing stage, the repeated words, stop words and punctuation symbols were removed in order to increase the quality of the dataset. The feature extraction method that the researchers used extracted the aspect (adjective) from the text. Later, this adjective was used to show the positive and negative polarity in a sentence, which is useful for determining the opinion of the individuals, using the unigram model. Finally, the classification process was performed with three models: Naive Bayes, Maximum Entropy, and Support Vector Machine. The results of the experiments were good regarding accuracy, with 88.2% for Naïve Bayes, 85.5% for Support Vector Machine and 83.8% for Maximum Entropy, but things were not as good regarding precision where the results ranged between 33% and 50%. For the preprocessing of the data and the implementation of the models, the researchers used Python and the Natural Language Tool Kit.

The third and final paper regarding machine learning models for sentiment analysis for the English language that we will examine has to do with sentiment analysis of movie reviews using the very famous IMDb dataset (Tripathi et al., 2020). The researchers used text classification techniques in the above-mentioned IMDb dataset that was taken from Kaggle's Challenge called Bag of Words Meets Bag of Popcorn. In the preprocessing stage of the procedure, they cleaned the reviews from HTML tags, removed the stop words and applied text normalization. In the feature extraction stage, the researchers worked towards the limitation of the amount of the text of each review as the number of features would become so big that the models could suffer from overfitting. For the vectorization of the data, they used two techniques based on the Bag of Words model. The first technique is called Count and the second one is called Term Frequency-Inverse Document Frequency, or TF-IDF. These techniques can determine the importance of a word in a review based on its frequency of occurrence on the dataset. In the final step, the researchers applied the classification models that they had chosen, which, for this work, were the following four: Logistic Regression, Naive Bayes, Decision Tree, and Random Forest. The performance of these models was measured with five different metrics, and more specifically, accuracy, Area Under Curve (AUC), F1- score, recall, and precision. The results are presented in the following table:

Table 1 Classification Results (Tripathi et al., 2020)

ML Models	Results With Count (Metrics)					ML Models	Results With TF-IDF (Metrics)				
	Accuracy	Precision	Recall	F-score	AUV		Accuracy	Precision	Recall	F-score	AUV
Logistic Regression	0.8728	0.8708	0.8777	0.8742	0.94	Logistic Regression	0.8914	0.882	0.9055	0.8936	0.96
Naïve Bayes	0.8594	0.8566	0.8658	0.8612	-	Naïve Bayes	0.8228	0.8285	0.8174	0.823	0.85
Decision Tree Classifier	0.7134	0.7218	0.7014	0.7114	0.71	Decision Tree Classifier	0.7066	0.7098	0.7098	0.7114	0.71
Rnandom Forest Classifier	0.8584	0.862	0.8558	0.8589	0.93	Rnandom Forest Classifier	0.8562	0.8597	0.8539	0.8568	0.93

As we can see, Logistic Regression combined with TF-IDF had the best performance in every metric that they used.

### 3.2. Deep Learning Methods

In this second part of the related work for the English language, we will focus on papers that deal with deep learning models for sentiment analysis such as Long Short-Term Memory and Convolution Neural Networks. In the first paper that we will examine, the researchers introduced a word embeddings method obtained by unsupervised learning based on large Twitter corpora (Jianqiang et al., 2018). Sentiment analysis was applied on five different datasets collected from Twitter that had previously been used for sentiment analysis in academic research. More specifically, they used five different datasets: the Stanford Twitter Sentiment Test (STSTd) dataset, which consisted of 359 tweets split in 182 positive and 177 negative ones, the SE2014 dataset, which consisted of 5,892 tweets labeled as positive and negative, the Stanford Twitter Sentiment Gold (STSGd) dataset, which consisted of 2,034 tweets and that were manually labeled as positive and negative, the Sentiment Evaluation Dataset (SED), which consisted of 2,648 tweets split in two categories, 1,658 positive and 990 negative ones, labeled by Mechanical Turk workers, and finally, the Sentiment Strength Twitter dataset (SSTd), which consisted of 2,289 positive and 1,037 negative tweets, also labeled manually. The main goal of this paper was to reach conclusions on the effect of different types of embeddings on different machine learning algorithms such as Support-Vector Machines (SVM), Logistic Regression (LR) and Deep Convolution Neural Networks (DCNN), and the comparison of the results of these algorithms on various datasets as the ones we have already mentioned. The researchers used two types of techniques for word vectorization: the above-mentioned Bag of Words, or BoW, and the GloVe-based technique, which is a model for the pretraining of Word Embeddings. During the data preprocessing stage of the text classification process, the researchers removed all the non-ASCII and non-English characters, URL links, numbers and stop words. Subsequently, they replaced the negative references like the word “can’t” with the original “cannot” as well as the emoticons and emoji icons with their original text form using an emoticon dictionary. Finally, they expanded the acronyms and replaced the slang words with their standard form. After the vectorization stage that we have already mentioned,

the classification was applied to five combinations of models and vectorization techniques. These models were BoW-SVM, BoW-LR (machine learning models combined with Bag of Words), GloVe-SVM, GloVe-LR and GloVe-DCNN (machine learning models combined with GloVe). At this point, we will focus a little more on the latest model GloVe-DCNN, particularly on its architecture. The DCNN model contained one input layer, three hidden layers along with three pooling layers and one output layer that generated a probability value which determined the positive or negative class. On the fully connected layers, dropout regularization was applied to avoid overfitting. The hyperparameters for the DCNN were the following: the batch size was 128, the learning rate was 0.001, the dropout rate was 0.5 and the activation function that they used was the ReLU. In all of the experiments, 10-fold cross validation was applied and the models were evaluated by the average accuracy of each experiment. The results showed that the model with the best performance undoubtedly was GloVe-DCNN, with 87.62% accuracy, which was accomplished on the STSTd dataset. On average, GloVe-DCNN achieved a maximum improvement in accuracy of 19.14%. The conclusion of this paper was that the Deep Neural Networks have a much better performance than other machine learning models, especially when they utilize pre-trained word vectors.

In the second paper that concerns sentiment analysis for the English language with deep learning models, the researchers applied text classification with the use of a Long Short-Term Memory model with the well-known IMDb dataset that we have already discussed (Qaisar, 2020). The researchers chose to use only 10,000 out of the 50,000 reviews. They split these reviews into 80% training data and 20% testing data. Additionally, they used 10-fold cross validation to avoid the potential bias in the results. In the preprocessing stage of the classification workflow, they removed the punctuation symbols, stop words and hybrid links, all letters on the dataset were converted into lowercase and, finally, they applied stemming to the words to convert them into their original form. The vectorization of the data was done by a Python library called Genism, which provided the Doc2Vec tool. For the classification process, the researchers used an LSTM neural network with tree layers. The results have shown an 89.9% accuracy.

In the next paper, the researchers focused on an algorithm that combined a CNN and an LSTM architecture in order to achieve better sentiment analysis performance on the IMDb dataset (Yenter & Verma, 2017). In this paper, the preprocessing of the data was performed with a method from Python's Keras library. For this procedure, only a D number of the most common words of the dataset constituted the dictionary that the researchers used for the vectorization process that followed. The reviews were padded in a certain length with the largest of these being cut to fit this length and the shorter ones being filled with zeros. The hyperparameters they used for the preprocessing and the embedding of the data were the following: the number of the words that constituted the dictionary were 5,000, the length of the vectors was 500, and the dimensions of the embedded words were 32. As we have already mentioned, the architecture of the models consisted of CNN and LSTM characteristics. More specifically, the first layer of this model was the embedding layer with the hyperparameters we have already discussed. The results of the first layer were given to a b number of branches where each one started with a 1-dimensional convolutional layer with a c size kernel. The following layer of each branch was an activation layer with ReLU activation function. The next layers were a max pooling and a dropout layer which protected the model from overfitting. The next and final

layer before the LSTM layer was a batch normalization layer. The LSTM layer was used because it can process sequences of data, a feature which is very important for text data processing as the context of a word depends on other words in a sentence and not only the neighboring ones. After the LSTM layer, what followed was a concatenation layer that simply merged the results of each branch into an array. The last layer was a dense layer with a sigmoid activation function that classified the reviews into two classes; 0 for negative sentiment and 1 for positive sentiment. The researchers performed a lot of experiments with various hyperparameters, but the best results were achieved by a model with a batch size of 128 and a number of branches (of the model) of 4. The optimal number of kernels for the convolutional layer was 3,5,7, and 9 and for the pooling layer was 2, the dropout was set at a rate of 0.5, each LSTM layer had 128 units, and the learning rate was between 0.001 and 0.01 for the Adam and RMSprop optimizers. The best performance of these algorithms achieved 89.5% accuracy, which, at the time, was a great result on the specific dataset. The researchers mentioned that the biggest challenge of this work was to avoid the overfitting of the model.

In the last paper on deep learning models for sentiment analysis for the English language that we will examine, the researchers performed sentiment analysis on the IMDb dataset, with three models which had a similar architecture to models that we have already seen (Haque et al., 2019). The models that they used were CNN, LSTM, and a hybrid LSTM-CNN. Here, we will focus on the results of the models as a reference point for our work. The results are presented in the following table:

*Table 2 Classification Results (Haque et al., 2019)*

<b>Evaluation Measure</b>	<b>CNN</b>	<b>LSTM</b>	<b>LSTM-CNN</b>
<b>Accuracy</b>	0.90	0.88	0.89
<b>Recall</b>	0.95	0.82	0.90
<b>Specificity</b>	0.84	0.90	0.87
<b>Precision</b>	0.87	0.90	0.87
<b>F1-Score</b>	0.91	0.86	0.88

The conclusion that the researchers reached was that the CNN model outperformed the LSTM and LSTM-CNN models on the sentiment analysis classification problem, but the results are not far apart.

### 3.3. Methods based on Pre-Trained Models

Closing this section of papers which deal with sentiment analysis for the English language, we will focus on pretrained models, particularly on Bidirectional Encoder Representations from Transformers (BERT), which is a pretrained model based on Transformers, specifically made for Natural Language Processing. The reason we will mention this model is that there is an equivalent pretrained model that supports the Greek language called GreekBERT and which we will use later in this work. BERT is a relatively new technology, which is why not much work has been done on sentiment analysis with

this algorithm. However, in these few papers that have been written, the results were impressively good and have been state-of-the-art in many cases (Gao et al., 2019).

In the first paper that deals with the BERT model (Sousa et al., 2019), the researchers created a procedure that contained, among others, sentiment analysis to evaluate information from news articles in order to provide relevant information for decision-making in the stock market, especially the Dow Jones Industrial (DJI) Index. This procedure consisted of three stages. Firstly, the researchers collected and preprocessed the texts of the news articles, secondly, they applied sentiment analysis based on the BERT model and, finally, they concluded on the decision-making based on the results of the two previous stages of the procedure. The news articles were collected with the use of the Selenium framework from various websites such as the New York Times, the Washington Post and Business Insider, among others. Selenium is a very popular and powerful tool for web scraping and web browser automation that supports various programming languages like Java and Python. With the use of the tools that we have already mentioned, the researchers collected text data and manually labeled them positive, negative, and neutral. After the labeling process, they tokenized the data with the use of the WordPiece algorithm. For the vectorization process, they used a vocabulary of 30,000 tokens that came up from the dataset. The next step of the procedure was the fine-tuning of the BERT BASE model where they used 10-fold cross-validation with the following parameters: 12 layers, 768 hidden layers and 12 heads. The researchers compared the results of the classification with other algorithms with various vectorization techniques such as SVM combined with Bag of Words, SVM combined with TF-IDF, Naïve Bayes combined with Bag of Words, Naïve Bayes combined with TF-IDF and textCNN combined with fastText. The results are presented in the following table:

*Table 3 Classification Results (Sousa et al., 2019)*

<b>Algorithm</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>NB bow</b>	0.610	0.593	0.557	0.503
<b>SVM bow</b>	0.628	0.627	0.609	0.601
<b>NB tf-idf</b>	0.610	0.607	0.568	0.542
<b>SVM tf-idf</b>	0.624	0.631	0.595	0.578
<b>textCNN</b>	0.739	0.703	0.500	0.569
<b>BERT</b>	0.825	0.750	0.713	0.725

Clearly, BERT outperformed every other algorithm. The final step of the procedure was the analysis of the results in order to predict the stock market prices and trends, but the examination of this topic is beyond the scope of this work.

In the last paper on sentiment analysis for the English language that we will examine, the researchers investigated how to fine-tune BERT for text classification (Sun et al., 2019). The way they approached this task was by performing various experiments with different fine-tuning methods on many different datasets for text classification, with the result of this process being a general solution for BERT fine-tuning. The experiments were performed in two different languages, English and Chinese, on 8 different datasets, with 7 of them being in English and 1 of them being in Chinese. The models that the researchers used for these experiments were the uncased BERT-base model and the

Chinese BERT-base model. The experiments included three different text classification tasks, and more specifically, sentiment analysis, question classification, and topic classification. In the first part of the workflow, they preprocessed the data with the use of the WordPiece embeddings algorithm with a 30,000 token vocabulary. In the hyperparameters area, they used the BERT-base model with 768 hidden layers, 12 Transformer blocks and 12 self-attention heads. They further pre-trained the model, with a batch size of 32, a max sequence length of 128, a learning rate of  $5e-5$ , train steps of 100,000 and warm-up steps of 10,000. The dropout probability was always 0.1, the optimizer that was used was Adam optimizer with  $\beta_1 = 0.9$  and  $\beta_2 = 0.999$ , the base learning rate was  $2e-5$ , and the warm-up proportion was 0.1. Finally, they set the max number of the epoch to 4. The results showed that the most important layer for text classification is the first. Also, further pretraining of the model can boost its performance and with a decreasing learning rate, BERT can overcome the catastrophic interference problem. The researchers reported that they achieved state-of-the-art performances on eight popular text classification datasets.

### 3.4. Sentiment Analysis for Greek Language

In this first part of the related work concerning the Greek language, we will provide a brief overview of the work that has been done on sentiment analysis for the Greek language. There are some papers that deal with this matter, but only a handful of these are concerned with sentiment analysis with supervised methods.

In the first paper that we will examine, the researchers compared two different methods of performing sentiment analysis (Markopoulos et al., 2015). The first method was the one which we have already mentioned several times and that has to do with text classification by machine learning algorithms such as the popular SVM (which they used in this work), and the second one is a technique that employs a sentiment lexicon in order to characterize a text as positive or negative by counting the frequency of specific polarity words. This comparison is very important for this area because it shows the pros and cons of a supervised method such as the machine learning-based one and an unsupervised method like the lexicon-based one. To apply their methods, the researchers created a dataset of hotel reviews that they collected from a travel website which is the Greek equivalent of TripAdvisor. It consisted of 1,800 reviews split into two categories; 900 positive and 900 negative ones. The researchers excluded reviews that were translated into Greek or reviews that were too short or too long in length, and more specifically, with less than 30 or more than 250 words respectively. They manually corrected the punctuation and spelling errors, and finally, they assigned a label (positive or negative) based on their perception and the rate of the reviewer in order to create, as they characteristically stated, 'a proper training set'. The algorithms were applied with the RapidMiner software, and more specifically, with its text mining extension. In the preprocessing stage of the procedure, they tokenized the data, excluded words with less than 4 and more than 25 characters and stop words which did not seem to make a significant contribution to the training stage. In the vectorization stage, for the first method they used TF-IDF, which is a way to vectorize data by giving a weight to each word with a statistical method that reflects the importance of this word to the dataset. For the second method, they used the Term Occurrence approach, or TO, which is a way to classify data

by counting the polarity of each word that the text contains. To implement the TO method, they created a Greek sentiment lexicon of 27,388 positive words and 41,410 negative ones. For the implementation of the machine learning algorithm, they used 10-fold cross-validation, and for the evaluation of the results, they used the measures of accuracy, recall and precision. As expected, the results showed a better performance with the supervised method (accuracy 95.78%) and considerably poorer results with the unsupervised method. In general, the results showed that the supervised method had a much better performance, but it needed time and labeled data in order to work, while the lexicon-based method did not achieve great results, but it was fast and could be applied on the spot without training. The results are presented in the following table:

*Table 4 Classification Results (Markopoulos et al., 2015)*

Results	TF-IDF		TO	
	Predicted negative	Predicted positive	Predicted negative	Predicted positive
Negative cases	880	20	323	481
Positive cases	56	844	0	899
Accuracy	95.78%		71.76%	
Recall	93.78%		100%	
Precision	97.69%		65.14%	

In the next paper (Spatiotis et al., 2016), the researchers applied a multi-level sentiment analysis on a text dataset with five different categories; very negative opinion, negative, neutral, positive and very positive. They used 5 text classification algorithms, and more specifically, J48, IBk, Multilayer Perceptron, PolyKernel and RBFKernel. The dataset that they used consisted of 11,156 user-generated comments, each of which was manually labeled; 133 of these were labeled negative, 584 were labeled unsatisfactory, 3,737 were labeled neutral, 3,217 were labeled satisfactory, and 3,485 were labeled very positive. The following table shows the results which are presented with the accuracy percentage of every algorithm:

*Table 5 Classification Results (Spatiotis et al., 2016)*

Classes	J48	ibk	MLP_N_500	PolyKernel_C_1	PolyKernel_C_10	RBF Kernel_C_1	RBF Kernel_C_10
Class A	6%	7.50%	3.80%	0%	0%	0%	0%
Class B	18.70%	18%	10.80%	1.20%	1%	0%	0%
Class C	60.80%	55.70%	62.20%	62%	63.10%	51.50%	61%
Class D	49.50%	51.40%	57.60%	32.70%	32.80%	12%	31.10%
Class E	65%	65.40%	52.70%	72.70%	72.30%	79.90%	73.40%
Average	56%	54.90%	54.50%	53%	53.20%	45.70%	52.40%

The results showed that the J48 algorithm had the best performance in terms of average accuracy. This group of researchers followed up this work with a new paper (Spatiotis et al., 2019) that aimed to improve the results of the previous one using discretization techniques which are methods that change the values that are in a continuous form into a discrete form in order to execute analysis processes. By applying text preprocessing and by using discretization techniques, they managed to improve the performance of the J48 algorithm on the same dataset but not substantially. Also, they concluded that discretization techniques could reduce the time of making a classification model.

Finally, in the last paper that we will examine, the researchers' goal was to evaluate a vast amount of techniques that consisted of supervised and unsupervised methods (Tsakalidis et al., 2018). More precisely, three experiments were performed on three different sentiment-related tasks. The first task was sentiment analysis, where the goal was to classify data into three sentiment classes; positive sentiment, negative sentiment, and neutral sentiment. The second task was emotion analysis, where the goal was to classify data into six emotion classes such as happy, unhappy, disgusted, etc. The third and final task was sarcasm detection, where the goal was to classify data into two classes; text with sarcastic content and text with non-sarcastic content. For these tasks, many different datasets were used. For the first task, the researchers used three different datasets. The first two were 'TIFF' and 'TDF' that were borrowed from a paper written by Schinas and Herzig (Schinas & Herzig, 2013) and contained tweets in English and Greek from two different festivals held in Thessaloniki - the first one was a film festival and the second was a documentary film festival (for these datasets only the Greek tweets were kept for the implementation of the classification). Finally, the third dataset that they used was 'GRGE' that concerned the 2015 Greek legislative election and consisted of 2,309 tweets which were manually labeled. For the second task, the researchers used a dataset made by Kalamatianos (Kalamatianos et al., 2015) that consisted of 681 tweets which were manually labeled (the 'disgusted' and 'angry' emotions were excluded from the experiment because of the low agreement of the annotators). For the third task, the researchers created a dataset which was manually labeled and that consisted of 3,000 tweets. In the preprocessing stage, they performed procedures such as the lowercasing of the words, replacement of URLs and usernames, tokenization of the text and removal of all non-alphanumeric characters. In these experiments, the researchers did not perform stemming or removal of the stop words because "these steps were found to have no or negative influence on the sentiment analysis tasks" as they noted. For the experiments, they used several algorithms such as Logistic Regression (LR), Random Forest (RF) and Support Vector Machines (SVM) for the first and third task, and LASSO, Random Forest for Regression (RFR), and Support Vector Regression (SVR) for the second task. The results were very encouraging for all tasks, but as the researchers claimed "the major advantage of our resources is highlighted in the cross-domain sentiment analysis task, which is the task that motivates the creation of such resources. Given that it is impossible to have annotated datasets for all domains and purposes, creating lexicons and resources that can be used in a new domain is of the utmost importance in sentiment analysis."



## 4. METHODOLOGY

### 4.1. Datasets

#### English Dataset

Internet Movie Database (IMDb) is a well-known online database that contains information and ratings for movies, TV series, video games, etc. created in 1993 in Wales. In addition to other features of the website, there is a public section where users can review and rate movies, TV series, etc. This means that it is possible to create a dataset of tens of thousands of movie reviews with labels already defined by the users of the website. This is exactly what was done in 2011 by a team from Stanford University (Maas et al., 2011), where a dataset of 50,000 reviews was created. It was split into two categories; 25,000 reviews with a positive label and 25,000 with a negative one.

IMDb has a rating system of ten categories from 1 to 10, so the researchers initially separated the reviews in three categories based on the score of the rate in order to create a set of high polarity data the way they desired. Reviews with a score of less than 4 would be the negative category, reviews with a score greater than 5 would be the positive category and reviews with a score of 4 or 5 would be the neutral category. Of these reviews, they kept only 25.000 of the negative and 25.000 of the positive reviews on condition that the reviews for each movie did not exceed the number 30. Over the next decade to date, this dataset would be considered one of the most popular datasets for testing sentiment analysis algorithms (Haque et al., 2019; Qaisar, 2020; Untawale & Choudhari, 2019; Yenter & Verma, 2017) and the results of these tests would become the benchmark for future applications and the evaluation of new models.

#### Greek Dataset

The aim of this work is to compare the results and the performance of sentiment analysis algorithms in two different languages, English and Greek. In order to achieve that aim, we had to apply these algorithms to two different datasets with similarities not only in the way they were created, but also in the essence of their content. As there are not many labeled datasets in the Greek language, many options were considered for the creation of a dataset that would meet the requirements of this work. Among these options were Skrouz <sup>7</sup>, which is the largest online shopping platform in Greece, the Greek part of Twitter, Insomnia<sup>8</sup>, which is a very popular online Greek forum, and Athinorama <sup>9</sup>, which is the Greek counterpart of IMDb. Each option had its pros and cons. The Greek part of Twitter had millions of Greek messages to choose from, but there were no labels, which was an essential element for this work. Skrouz contained a lot of rated reviews of many different shops and products submitted by customers, but the anti-bot protection of the

---

<sup>7</sup> <https://www.skrouz.gr/>

<sup>8</sup> <https://www.insomnia.gr/>

<sup>9</sup> <https://www.athinorama.gr/>

website would make the creation of the dataset very time-consuming and practically impossible for the size of the dataset we intended to use in this work. Insomnia had various text data, some with labels and some without, but the essence of these text data were unique, and we were not able to find corresponding text data in the English language. That is the reason we came up with the Athinorama dataset, which overcame all these problems as the website did not have anti-bot defense, the reviews had already been rated by the viewers and there was an equivalent dataset in English with movie reviews from IMDb. The IMDb dataset would play a very significant role in this work as it would be the dataset used to apply different algorithms in order to create a benchmark for the experiments on the Greek language dataset. For this reason, our intention was to create the Athinorama dataset based on the same principles employed in the creation of the IMDb dataset.

Following these principles, we created a dataset using a web scraping program made in Python. This program ran for 2 days, from 20/10/2021 to 22/10/2021, and collected all the movie reviews of the website with the following restrictions: a) only reviews in the Greek language were collected (reviews in Greeklish, English or with English majority text were excluded), b) only rated reviews were collected, and c) only reviews with a text length greater than 2 were collected in order to avoid reviews that contained only a rate without text. In this first version of the Athinorama dataset, 15 categories of data were collected for each review as we can see in the following vector ['id number', 'greek title', 'original title', 'category', 'director', 'movie length', 'movie date', 'author', 'review date', 'review', 'stars', 'label', 'mean of stars', 'number of reviews', 'url']. Analyzing each category, 'id number' is an increasing number for each review, 'greek title' is the Greek version of the title of the reviewed movie, 'original title' is the original title of the reviewed movie, 'category' is the genre of film, 'director' is the director's name, 'movie length' is the length of the movie in minutes, 'movie date' is the date of the movie only in years, 'author' is the nickname or the real name of the viewer, 'review date' is the review date in years, 'review' is the raw review in Greek, 'stars' is the rate written in full, 'label' is the rate of the movie on a scale from 0 to 5, 'mean of stars' is the mean of the rates of the reviews for a certain movie, but only of the ones that we used for this dataset (the rates of the reviews that we excluded were not counted), 'number of reviews' is the number of the reviews we extracted from the website for each movie and the last category is the URL of each movie. With this process, we managed to extract about 150,000 reviews of 6,481 different movies. This first version of the Athinorama dataset was full of different data but it was not equivalent to the IMDb one, so we created a second Light version that consisted of only two columns, the 'review' column and the 'label' column, and which met the other conditions of the IMDb dataset, as previously mentioned. The Athinorama website has a rating system with a scale from 0 to 5 which includes the following rates, 0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5 and 5, which means that we had 11 classes instead of 10 – which is how many classes the IMDb website has. In order to create a high polarity dataset, we decided to create three classes; one with the top 4 ratings (ratings > 3), which would be the positive class, one with the bottom 4 ratings (ratings < 2), which would be the negative class, and one with the middle three (2, 2.5 and 3), which would be the neutral class. From these reviews, we randomly kept only 25,000 of the positive and 25,000 of the negative reviews. The restriction of less than 30 reviews for each movie was not applied due to the limited number of reviews. This Athinorama Light dataset was

used for all the experiments on sentiment analysis we performed regarding the Greek language.

### Data Analysis

These two datasets show a lot of commonalities, but they have some differences as well. The biggest structural difference between these two datasets, apart from the different language, is the length of the reviews. The IMDb dataset has reviews with more tokens per review than the Athinorama Light dataset which has shorter reviews in general. More precisely, the IMDb dataset consists of reviews with 267 tokens per review and the Athinorama Light dataset consists of reviews with 51 tokens per review. This is a fact that can affect the results of the classification as it seems that the IMDb reviews contain more information about the sentiment of the message than the respective Athinorama Light reviews.

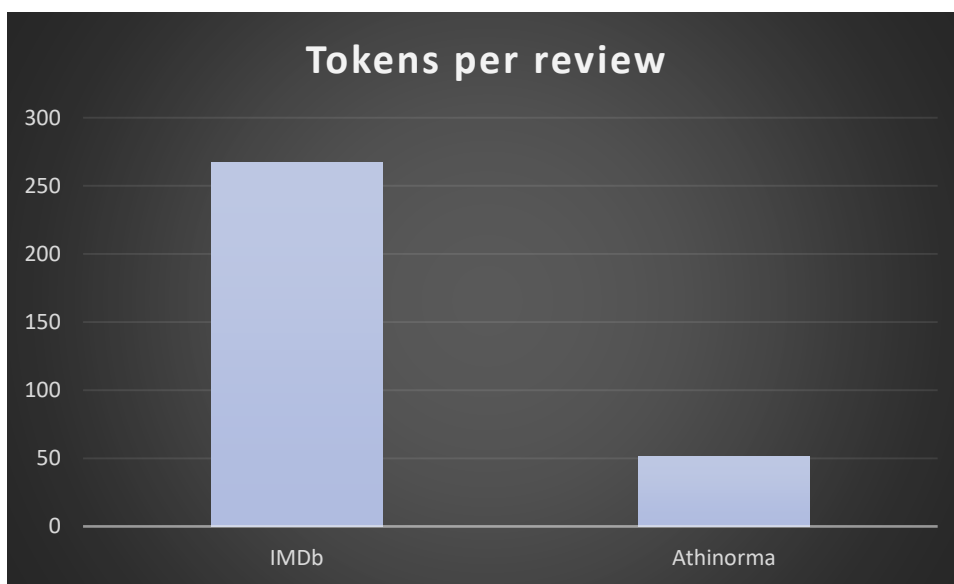


Figure 6 Mean tokens per review, for IMDb and Athinorama Light

The second and most critical difference between these datasets is obviously the languages and, more precisely, the structure of these languages. The English language is a Germanic language with Latin influences that has many similarities to German and French as they share a large part of their vocabulary. The Greek language does not share these commonalities with English. There is only a small part of common vocabulary and it is limited to scientific or artistic words. The Greek language also has other characteristics like accentuation, which does not exist in English, as well as a different inflection, which greatly affects the form of the words. The following image shows the most common words in the IMDb dataset, with the restriction that only words with more than four letters are depicted:

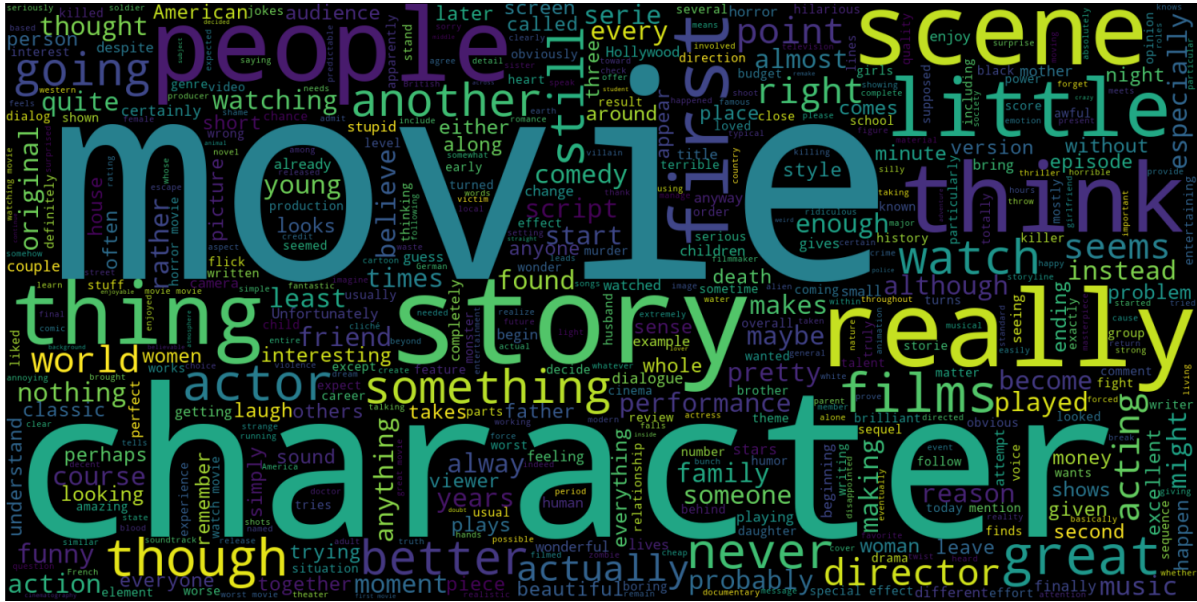


Figure 7 Most frequent words in IMDb

As we can see, words like ‘movie’, ‘character’, ‘story’ and ‘scene’ are really common, as was to be expected. The following image shows the most common words of the Athinorama dataset, with the restriction that only words with more than five letters are depicted:



Figure 8 Most frequent words in Athinorama Light

We see words like ‘ταινία’, ‘σενάριο’, ‘σκηνοθεσία’ και ‘σινεμά’ which are the most common on a movie dataset, but we observed that words like ‘ταινια’, ‘ταινία’, ‘ταινίες’ and ‘ταινιάς’ are also depicted at the same time. The same word appears with different inflections and with or without accent, something that could affect the classification models.

Table 6 Datasets comparison

Dataset name	Athinorama	Athinorama Light	IMDb
Number of Reviews	148,795	50,000	50,000
Number of Labels/ Classes	11	2	2
Tokens per Review	56	51	267
Columns/ Categories	['id number', 'greek title', 'original title', 'category', 'director', 'movie length', 'movie date', 'author', 'review date', 'review', 'stars', 'label', 'mean of stars', 'number of reviews', 'url']	[review, label]	[review, label]
Unique Values (Reviews)	143,912	48,929	49,581
Reviews per class	5.0 33,541 4.0 25,116 3.0 17,687 0.0 15,244 2.0 12,014 1.0 9,810 3.5 9,796 4.5 9,632 2.5 6,514 0.5 5,158 1.5 4,283	Positive 25,000 Negative 25,000	Positive 25,000 Negative 25,000

## 4.2. Preprocessing

Leaving the datasets area and entering the first parts of the sentiment analysis workflow such as the data preprocessing stage, we could face problems like sarcasm detection and metaphor understanding, both of which are concepts that can affect the results of the classification process. In the following paper (Cambria et al., 2017), the

researchers tried to discover the reasons that artificial intelligence did not manage to achieve human-like performance in the above mentioned problems along with a detailed description of the data preprocessing that consists of a) Microtext Normalization, which concerns the nature of short reviews or chat messages that tend to be misspelled and contain slang and emoticons (Xue et al., 2011) - features that make them unsuitable for classification problems, b) Part-of-Speech Tagging, which categorizes each word of a sentence based on its part of speech (e.g. adjective, verb and noun), c) Lemmatization, which converts each word into its base form, and finally d) Subjectivity Detection, which aims to remove content that does not affect the results of the classification (Chaturvedi et al., 2018) in order to make the algorithms more efficient. Application and evaluation of all these techniques can be seen in the work of Jianqiang and Xiaolin (Jianqiang & Xiaolin, 2017) where the researchers tried to evaluate several well-known preprocessing techniques. One of these techniques was the replacement of negative mentions with their corresponding words because these words were important for the definition of the emotion of a message, for example, the replacement of “can’t” with the word ‘cannot’. Other techniques that they used was the removal of URLs, numbers, stop words, and spelling correction where possible, especially in cases where letters were repeated in words. For example, in Twitter messages, it is very common to use words like “coool”. This word can be replaced with the word “coool” and later with the correct word “cool” with a simple algorithm that replaces letters that are repeated three times with only two of the letters that are repeated. One last technique that was used was the expansion of acronyms and the replacement of slang words with their standard forms with the aid of an acronym dictionary. The researchers concluded that of all these techniques, only the expansion of acronyms and the replacement of negative mentions and slang words affect the result of the classification. On the other hand, the removal of URLs, numbers and stop words were proven not to be very important for this kind of problems. Many of these techniques along with the vectorization part of the sentiment analysis workflow that follows will be used in some form in this work.

For this work, we performed experiments that included the preprocessing of the text data, but we also included experiments with text data that were not fully preprocessed. In both cases, we removed the HTML tags and URLs, which are tokens without meaning, and we also removed the emojis from the Athinorama Light dataset because these were tokens that did not exist in the IMDb dataset and we wanted both datasets to have similar content. For the experiments with full preprocessing, we added the following workflow. We applied the lemmatization process, which converted each word to its base form except for the pronouns. Then, we converted each word into its lower-case form to prevent the algorithm from being confused by the same words. Finally, the tokenization process took place at the same time as the exclusion of the stop words and punctuation symbols. In the following table, we can see some examples of the full preprocessing of some reviews both in English and in Greek.

Table 7 Greek language example

Original review	Movie Images	Vectorized review
<p>ανεπανάληπτο 3D καρτούν. Μιλάμε έπαθα πλάκα. Ένα μικρούλικο αθώο κοτοπουλάκι εναντίον μοχθηρών εξωήινων που θέλουν να κατακτήσουν το πλανήτη και να στερήσουν την ελευθερία από τους κατοίκους του. Θα ζήσει ευτυχισμένο μαζί με τους φίλους του ή θα το ψήσουν ζωντανό οι κακοί; Δεν θα σας πω μην καταστρέψω τις πάμπολλες σεναριακές εκπλήξεις που έχει αυτό το μικρούλικο έπος. Μιλάμε για τρελό κλάσιμο! Λέω μάλιστα να πάω να το ξαναδώ αυτή τη φορά με την παρέα μου. Είμαι σίγουρος ότι θα ενθουσιαστούν, θα γελάσουν, θα τραγουδήσουν και θα ξεχάσουν για μιάμιση ώρα τα βάρη της καθημερινότητας. Είναι από τις ταινίες που λες ότι και δεκαπέντε ευρώ να είχε το εισιτήριο θα τα πλήρωνα ευχαρίστως. Μην το χάσετε για τίποτα!</p>	 <p>(Image source IMDb)</p>	<p>[‘ανεπανάληπτος’, ‘3d’, ‘καρτώ’, ‘μιλώ’, ‘παθαίνω’, ‘πλάκα’, ‘ένα’, ‘μικρούλικος’, ‘αθώο’, ‘κοτοπουλάκι’, ‘μοχθηρός’, ‘εξωήινας’, ‘θέλω’, ‘κατακτώ’, ‘πλανήτης’, ‘στερώ’, ‘ελευθερία’, ‘κατοίκος’, ‘ζήσω’, ‘ευτυχισμένο’, ‘φίλος’, ‘ή’, ‘ψήσω’, ‘ζωντανό’, ‘κακός’, ‘λέω’, ‘καταστρέψω’, ‘πάμπολλες’, ‘σεναριακός’, ‘εκπλήξη’, ‘μικρούλικος’, ‘έπος’, ‘μιλάμε’, ‘τρελό’, ‘κλάσιμο’, ‘λέω’, ‘πάω’, ‘ξεναδώ’, ‘φορά’, ‘παρέα’, ‘σίγουρος’, ‘ενθουσιαστώ’, ‘γελάσω’, ‘τραγουδώ’, ‘ξεχάσω’, ‘μιάμιση’, ‘ώρα’, ‘βάρη’, ‘καθημερινότητα’, ‘ταινία’, ‘λες’, ‘δεκαπέντε’, ‘ευρώ’, ‘εισιτήριο’, ‘πλήρωνος’, ‘ευχαρίστως’, ‘χάσω’]</p>

As we can see, some words are not in the form that we expected in the vectorized review. For example, ‘πλήρωνα’ after the preprocessing became ‘πλήρωνος’, which is not the word that we expected. The correct word should have been ‘πληρώνω’. That is a problem resulting from spaCy and, more precisely, from the lemmatization function of the library. The lemma\_ function that we used for the lemmatization process provided the base form of a token, with no inflectional suffixes. spaCy, especially in Greek, was pretrained in a certain amount of data that may not contain some of the words of our dataset. That means that in these cases, spaCy will try to guess the word that has to return with doubtful

results. In English, things are much better probably because of the simpler grammar of the language and the absence of many different inflectional suffixes for each word. Nevertheless, this is a feature that would affect smaller datasets and not so much our case where our datasets contain tens of thousands of reviews.

Table 8 English language example

Original review	Movie Image	Vectorized review
<p>Let me start off by saying I love Japanese cinema, literature and culture generally. I've seen many Japanese movies and enjoyed them, but ""Portrait of Hell"" (aka Jigokuhen) makes itself ridiculous. The two characters who dominate the action -- the ""evil lord"" in his privileged bubble and the ""stubborn, crazy artist"" are pure types with zero subtlety or nuance, and all their actions emanate from cartoonish extremes. The film wants to show horrible scenes of violence and raw emotion but many of these scenes are so over the top they actually become laughable and the overall feeling is that of a made-for-TV movie that went off the rails. If this rarely screened movie falls in your hands or comes to your town, spare yourself and give it a pass.</p>	 <p>(Image source Wikipedia)</p>	<p>['let', 'start', 'love', 'japanese', 'cinema', 'literature', 'culture', 'generally', 'japanese', 'movie', 'enjoy', 'portrait', 'hell', 'aka', 'jigokuhen', 'ridiculous', 'character', 'dominate', 'action', '--', 'evil', 'lord', 'privileged', 'bubble', 'stubborn', 'crazy', 'artist', 'pure', 'type', 'zero', 'subtlety', 'nuance', 'action', 'emanate', 'cartoonish', 'extreme', 'film', 'want', 'horrible', 'scene', 'violence', 'raw', 'emotion', 'scene', 'actually', 'laughable', 'overall', 'feeling', 'tv', 'movie', 'rail', 'rarely', 'screen', 'movie', 'fall', 'hand', 'come', 'town', 'spare', 'pass']</p>

### 4.3. Vectorization

Machine learning models are designed to manipulate numbers and make predictions. For this reason, in order to apply these models, we need numerical data. For



text-related classification problems, a very important step in the classification process is to convert the data we want to use into a format consisting of numbers instead of text. This process is called vectorization and there are many different techniques for doing it. In the next section of this work, we will focus on some of the techniques that we applied in our experiments.

In the vectorization stage we used two different techniques, the first one being Term Frequency – Inverse Document Frequency, or TF-IDF (Markopoulos et al., 2015), and the second one being Word Embeddings. TF-IDF is a vectorization technique that is based on the Bag of Words, or BoW, technique (Harris, 1954), so in order to explain TF-IDF, we had to start with BoW.

In the BoW technique, a vocabulary that contains all the unique words of the dataset is created and the goal is to vectorize each review of the dataset by using that vocabulary. The length of each vector is equal to the number of the unique words of the vocabulary. Every word of the vocabulary has a certain position in that vector; if the word exists in a review, then the number of the appearance of this word in that review is placed in the corresponding position of the vector, otherwise the number 0 is placed. For example, if we have the following vocabulary (you, nice, is, ice, scream, cream, all, that, girl, boy), then the sentence ‘I scream, you scream, we all scream for ice cream’ could be represented as (1,0,0,1,3,1,1,0,0,0). The problem with Bag of Words is that the importance of the words results from the frequency with which they appear in a review, which does not necessarily mean that it is the right criterion for the evaluation of words. The solution to this problem is the TF-IDF technique which offers a better perspective in placing weights on words. In this technique, there are two scores that are combined in order to give the weight of a word. The first one is Term Frequency (TF), which is the number of repetitions of a word in a review divided by the number of the words of that review, and the second one is Inverse Document Frequency, which is the logarithm of the number of the reviews of the dataset divided by the number of the reviews that contain the word we have already mentioned. The key idea behind IDF is that words that appear infrequently in a collection of documents tend to be more informative than the words that appear frequently across many documents. Hence, each term in a document receives a specific weight by multiplying these two scores.

The Word Embeddings technique uses n-dimensional word embedding vectors instead of weights for each word of a review. Each dimension represents a general feature to which a number from 0 to 1 is attached depending on how relevant this feature is to this word. The reason we are doing this with Word Embeddings is because we want to create an embedding space in which the words with similar meaning are relatively close to each other, something that proves to be very efficient when training an algorithm (Tang et al., 2016).

#### 4.4. Accuracy Metrics

For the evaluation of our experiments, we used four different metrics: accuracy, precision, recall, and F1-score. Most of the times, we used the classification ‘accuracy’ to measure the performance of our model; however, accuracy is not enough to truly judge a classification model, so for that reason, we also used the other three metrics.

For the description of these metrics, we will need four rates: the True Positive (TP),

the True Negative (TN), the False Positive (FP) and the False Negative (FN).

Accuracy is the ratio of correctly predicted observations to the total observations.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

$$Precision = \frac{TP}{TP + FP}$$

Recall is the ratio of correctly predicted positive observations to all observations in actual class.

$$Recall = \frac{TP}{TP + FN}$$

F1 Score is the weighted average of Precision and Recall.

$$F1Score = 2 \frac{Recall * Precision}{Recall + Precision}$$

## 4.5. Tools and Libraries

For the web scraping program, we used the **Python** programming language along with the **Beautiful Soup** library, which is capable of pulling data out of HTML files, the **requests** library, which allows us to send HTTP requests, and the **langdetect** library, which was used for the detection of the Greek reviews and the rejection of those written in English or Greeklish. For the preprocessing of the data, we used a relatively new technology called **spaCy**, which is an open-source library written in Python for natural language processing and which supports many languages, including Greek. spaCy uses pipelines that are trained on written web text (blogs, news, comments) that include vocabulary, syntax, and entities. There are many trained pipelines for each language. For example, in Greek there is the **el\_core\_news\_sm** (small), the **el\_core\_news\_md** (medium) and the **el\_core\_news\_lg** (large) which differ in the size of the data that they have been trained with. For this work, we used only the **el\_core\_news\_sm** (small) and the equivalent English one, the **en\_core\_web\_sm** (small), as they are lighter than the other two, and the vectorization techniques that we used for this work do not require the use of more heavily-trained pipelines. For the punctuation symbols for both the Greek and English text, we used the **string** library and its operation **String.punctuation** which includes the following symbols: `!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~`. For the stop words for the English text, we used the **STOP\_WORDS** operation of the spaCy library. For the

Greek text, we used a collection of stop words that were manually collected<sup>10</sup>. The implementation of the models was achieved with Python libraries such as **scikit-learn**, **transformers** and **Keras**, which are created for the development and the evaluation of machine and deep learning models.

## 5. EXPERIMENTS

For this work, we used nine different models: Support Vector Machine (SVM), Logistic Regression (LR), Naïve Bayes (NB), Decision Trees (DT) and XGBoost (XGB), which we combined with the TF-IDF vectorization technique and Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU) and Bidirectional Encoder Representations from Transformers (BERT<sup>11</sup>), which we combined with the Word Embeddings (WE) technique.

As we have already mentioned, the goal of this work is to compare machine learning models on different datasets in Greek and English, so the architecture and hyperparameters of these models must be identical in order for us to draw useful conclusions. For this reason, we created nine different models that we used in both of our datasets and we ran four experiments for each model. Half of these experiments had a full preprocessed workflow for every review that included the lemmatization and the lower casing of the words along with the exclusion of the stop words and punctuation symbols. The other half of the experiments ran without any of these preprocessing steps, with the exception of the lower casing of the words.

The experiments ran on three different programs. In the first program called **ML\_models.py**, five models were deployed using the sklearn and xgb libraries. In most of these models, we used the default hyperparameters of the corresponding classes, the only exception being the XGBoost.

- More precisely, for the **SVM**, we used the LinearSVC class with loss function equal to 'squared\_hinge', tolerance for stopping criteria equal '1e-4' and maximum number of iterations equal to '1000'.
- For the **Logistic Regression**, we used the LogisticRegression class with solver equal to 'lbfgs' and maximum number of iterations taken for the solvers to converge equal to '100'.
- For the **Decision Trees**, we used the DecisionTreeClassifier class with function to measure the quality of a split equal to 'gini' and splitter equal to 'best'.
- For the **Naïve Bayes**, we used the MultinomialNB class with smoothing parameter equal to '1.0', fit\_prior equal to 'True' and class\_prior equal to 'None'.
- Finally, for the **XGBoost**, we used the XGBClassifier class of the xgb library with max\_depth equal to '7', n\_estimators equal to '300' and objective equal to 'binary:logistic'.

---

<sup>10</sup> [https://github.com/explosion/spaCy/blob/master/spacy/lang/el/stop\\_words.py](https://github.com/explosion/spaCy/blob/master/spacy/lang/el/stop_words.py)

<sup>11</sup> <https://towardsdatascience.com/sentiment-analysis-in-10-minutes-with-bert-and-hugging-face-294e8a04b671>

In the second program called **DL\_models.py**, we deployed three different deep learning models using the Keras library. For all these models, we used the same five hyperparameters. More precisely, the embedding vector length was 32, the maximum length of the reviews was 100, the number of words of the vocabulary was 10000, the batches' size was 32 and all models ran for 10 epochs. For each of these models, we used the 'adam' optimizer and the 'binary\_crossentropy' loss function.

Finally, the third program called **BERT\_model.py** concerns the BERT pretrained model. For this program, we used the transformers library and the bert-base-uncased (Kenton et al., 1953) pretrained model for the English experiments and the bert-base-greek-uncased-v1<sup>12</sup> pretrained model for the Greek ones. For hyperparameters, we used learning\_rate equal to '3e-5' and epsilon equal to '1e-08'.

To sum up, we examined four different experiments on two different datasets with nine different models. One experiment on the nine models with the IMDb dataset with no preprocessing of the reviews, a second experiment with the same dataset with full preprocessed reviews, a third experiment on the nine models with the Athinorama Light dataset with no preprocessing of the reviews, and finally, a fourth experiment with the Athinorama Light dataset with full preprocessed reviews. For all our experiments, we split our data into 90% training data and 10% testing data.

## 6. RESULTS

### 6.1. Results for the English Language

Starting with the experiments on IMDb datasets with no preprocessing, most of the models achieved excellent results, ranging between 80 and 91 percent, apart from the Decision Trees model, which did not achieve good results, reaching just 72 to 73 percent in all metrics. As far as the equilibrium between these four metrics is concerned, the most balanced models were SVM+TF-IDF, LG+TFIDF, LSTM+WE and DT+TF-IDF. In this part of the experiments, the model with the best performance was SVM combined with the TF-IDF vectorization technique, which achieved 90.6% accuracy and 90.7% F1-score, with all other metrics giving a result over 90%.

---

<sup>12</sup> <https://huggingface.co/nlpaueb/bert-base-greek-uncased-v1>

Table 9 Results of IMDb experiments – No Preprocessing

IMDb - No Preprocessing				
	Accuracy	Precision	Recall	F1-score
SVM+TF-IDF	0.906	0.901	0.914	0.907
LG+TF-IDF	0.899	0.893	0.909	0.901
DT+TF-IDF	0.724	0.725	0.730	0.728
NB+TF-IDF	0.865	0.889	0.838	0.863
XGB+TF-IDF	0.873	0.865	0.887	0.876
GRU+WE	0.900	0.892	0.910	0.901
LSTM+WE	0.886	0.891	0.880	0.886
CNN+WE	0.821	0.808	0.840	0.824
BERT+WE	0.898	0.889	0.910	0.900

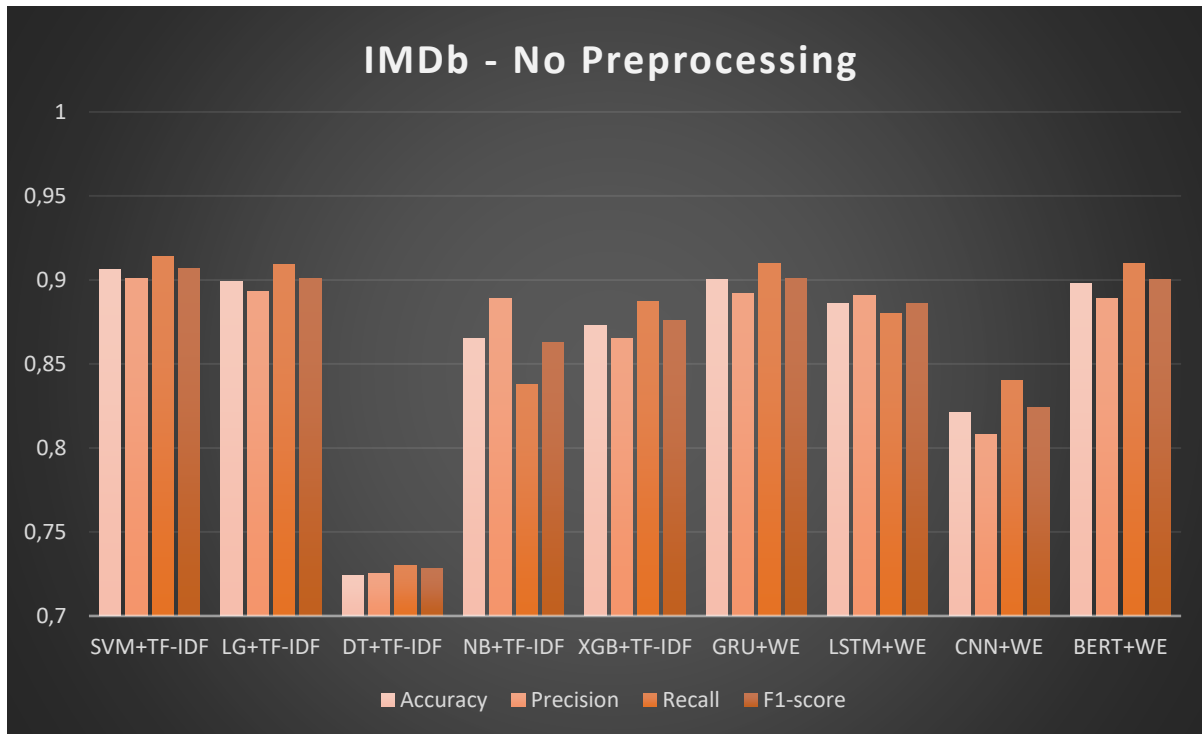


Figure 9 Graph of IMDb experiments – No Preprocessing

In the experiments with the IMDb dataset with full preprocessing, the results are similar, albeit slightly worse. Again, the majority of the models achieved great results with the exception of Decision Trees, but this time the model with the best performance was BERT combined with the Word Embeddings vectorization technique, which achieved 90% accuracy and F1-score. As far as the equilibrium between these four metrics is concerned

the most balanced models were CNN+WE, BERT+WE and DT+TF-IDF. Also, BERT achieved results close to 90% in all other metrics.

Table 10 Results of IMDb experiments – Full Preprocessing

IMDb - Full Preprocessing				
	Accuracy	Precision	Recall	F1-score
SVM+TF-IDF	0.893	0.887	0.904	0.895
LG+TF-IDF	0.888	0.878	0.903	0.891
DT+TF-IDF	0.723	0.726	0.727	0.727
NB+TF-IDF	0.855	0.863	0.847	0.855
XGB+TF-IDF	0.871	0.864	0.885	0.874
GRU+WE	0.865	0.892	0.830	0.860
LSTM+WE	0.862	0.824	0.920	0.869
CNN+WE	0.855	0.858	0.850	0.854
BERT+WE	0.900	0.904	0.895	0.900

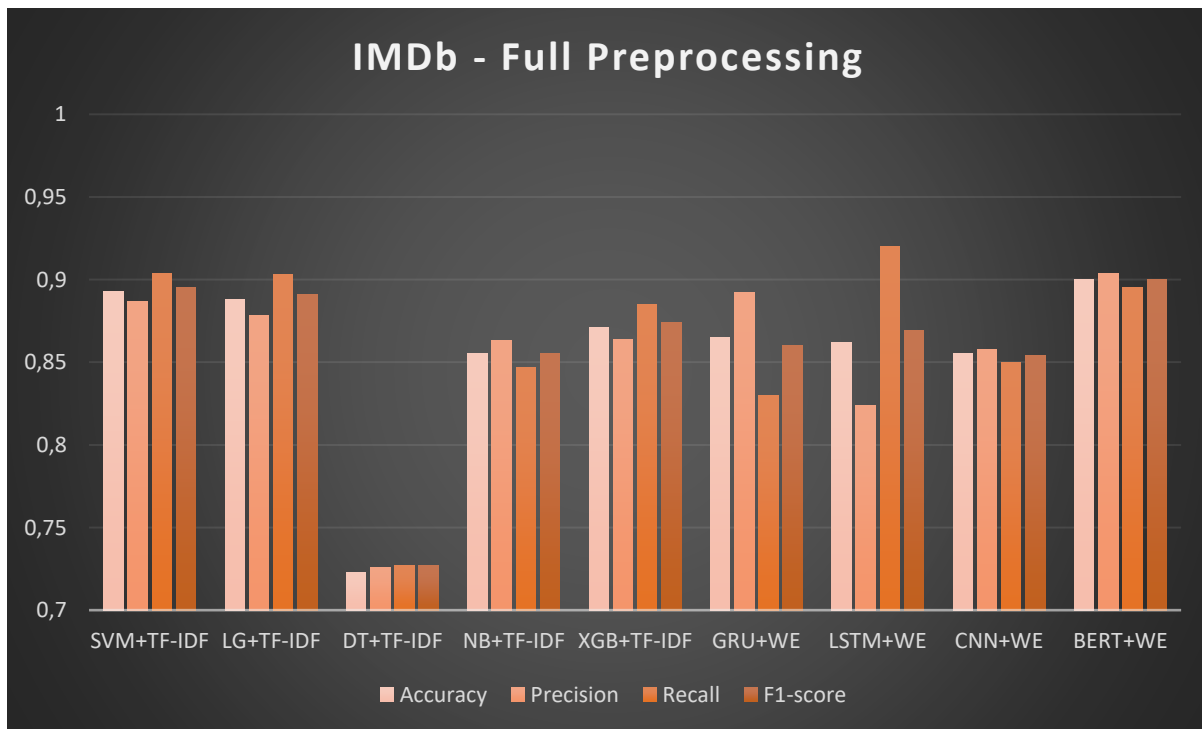


Figure 10 Graph of IMDb experiments – No Preprocessing

## 6.2. Results for the Greek Language

Entering the second part of this section, we can see that, in general, in the

experiments for the Greek language with no preprocessing, the results were not as good as the results for the English language. The majority of the models achieved worse results than the equivalent for the IMDb dataset. The only exception was the BERT model combined with the Word Embeddings vectorization technique, which achieved the best classification results, 92.3% accuracy and 92.2% F1-score, percentages that are the highest of all experiments regardless of language. The rest of the models achieved results between 84% and 87%, with the exception of the Decision Trees model that achieved the worst results, close to 72.2%. As far as the equilibrium between these four metrics is concerned, the most balanced models were NB+TF-IDF, GRU+WE and BERT+WE.

Table 11 Results of Athinorama Light experiments – No Preprocessing

Athinorama Light - No Preprocessing				
	Accuracy	Precision	Recall	F1-score
SVM+TF-IDF	0.869	0.876	0.864	0.870
LG+TF-IDF	0.864	0.878	0.848	0.863
DT+TF-IDF	0.722	0.728	0.717	0.723
NB+TF-IDF	0.871	0.874	0.870	0.872
XGB+TF-IDF	0.839	0.860	0.814	0.836
GRU+WE	0.862	0.858	0.868	0.863
LSTM+WE	0.841	0.807	0.896	0.849
CNN+WE	0.861	0.869	0.850	0.859
BERT+WE	0.923	0.930	0.914	0.922

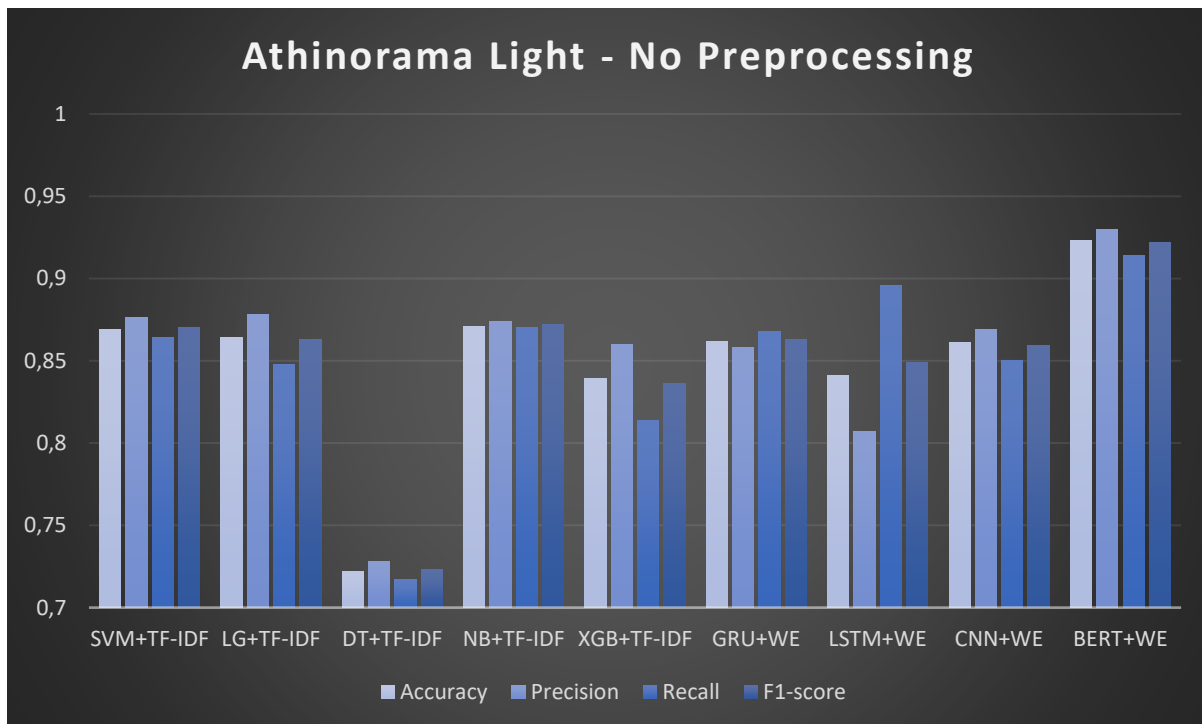


Figure 11 Graph of Athinorama Light experiments – No Preprocessing

Finally, for the last experiments, the results regarding the Greek dataset with full preprocessing were slightly worse than the one with no preprocessing. Again, the best results were a product of the BERT model that was combined with the Word Embeddings vectorization technique with 88.9% accuracy and 88.7 F1-score. The rest of the models achieved results between 82% and 86%, apart from the Decision Trees model that achieved the worst results, close to 72% accuracy. As far as the equilibrium between these four metrics, the most balanced models were SVM+TF-IDF, LG+WE and CNN+WE.

Table 12 Results of Athinorama Light experiments – Full Preprocessing

Athinorama Light – Full Preprocessing				
	Accuracy	Precision	Recall	F1-score
SVM+TF-IDF	0.855	0.855	0.86	0.858
LG+TF-IDF	0.859	0.862	0.858	0.860
DT+TF-IDF	0.720	0.719	0.733	0.726
NB+TF-IDF	0.854	0.844	0.872	0.858
XGB+TF-IDF	0.822	0.832	0.811	0.821
GRU+WE	0.836	0.849	0.818	0.833
LSTM+WE	0.840	0.825	0.862	0.843
CNN+WE	0.837	0.839	0.834	0.837
BERT+WE	0.889	0.903	0.872	0.887

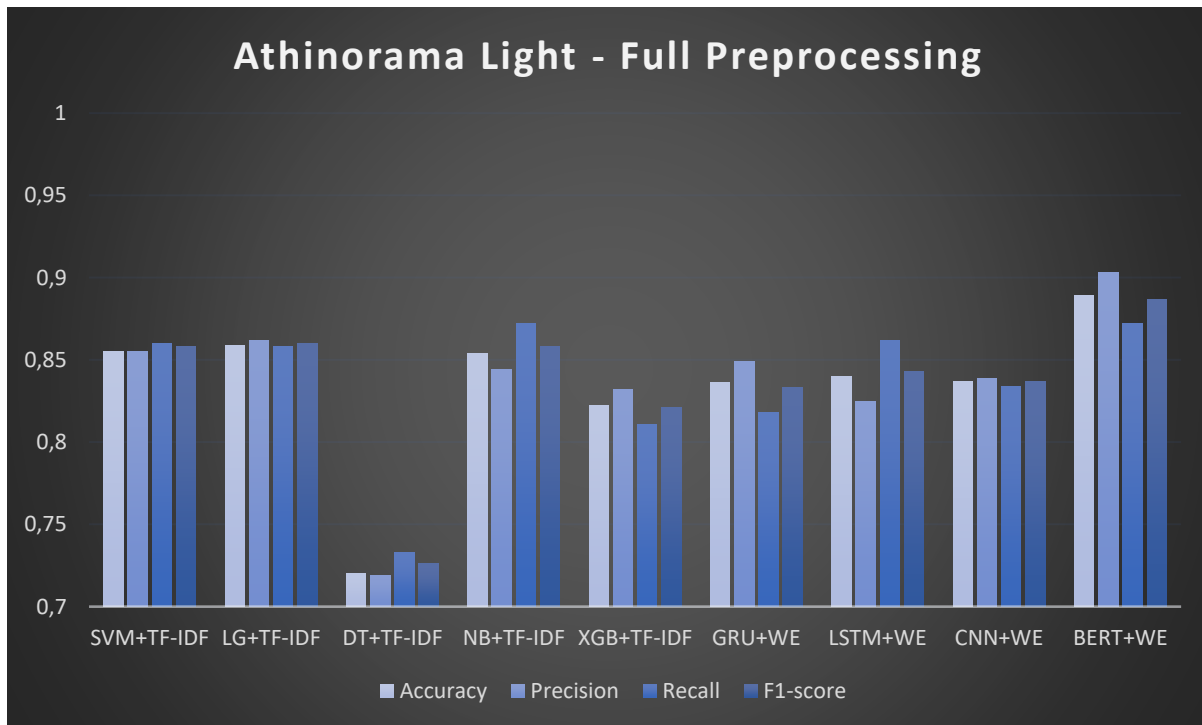


Figure 12 Graph of Athinorama Light experiments – Full Preprocessing



In the last part of the results section of this work, we compared the F1 Score for all our experiments because this metric could provide us with a general evaluation of our models. In the following table, we can see that the model with the best F1 score is, as expected, BERT combined with the Word Embeddings vectorization technique. The only exception is the results for the IMDb dataset without preprocessing as other models such as SVM, LG and GRU achieved slightly better results.

Table 13 F1-score Comparison

Comparison of F1-score				
	Athinorama-NP	Athinorama-FP	IMDb - NP	IMDb - FP
SVM+TF-IDF	0.870	0.858	0.907	0.895
LG+TF-IDF	0.863	0.860	0.901	0.891
DT+TF-IDF	0.723	0.726	0.728	0.727
NB+TF-IDF	0.872	0.858	0.863	0.855
XGB+TF-IDF	0.836	0.821	0.876	0.874
GRU+WE	0.863	0.833	0.901	0.860
LSTM+WE	0.849	0.843	0.886	0.869
CNN+WE	0.859	0.837	0.824	0.854
BERT+WE	0.922	0.887	0.900	0.900

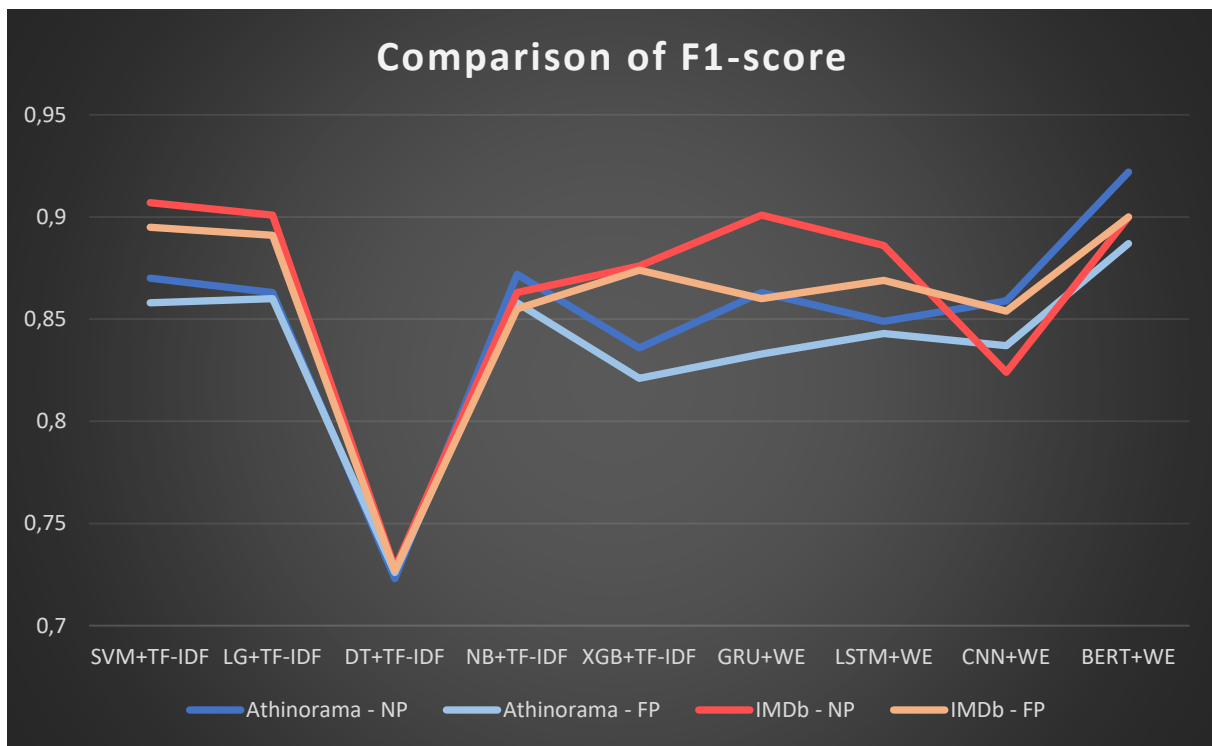


Figure 13 Graphs of F1-score Comparison

## 7. CONCLUSIONS

In this work, we focused on the sentiment analysis of movie reviews for the Greek and English language. We used two different datasets, one in the English language from IMDb and another one in the Greek language from Athinorama which we managed to create with the use of web scraping techniques. The Athinorama dataset was made under the principles of IMDb because the goal was to compare the results of our classification models in Greek and English texts, which meant that the two datasets should have the same structure and content. For the practical part of this work, we used the Python programming language along with various libraries and tools such as BeautifulSoup, requests, spaCy, Keras and scikit-learn. We performed two kinds of experiments; one with texts with limited preprocessing that included only the removal of the HTML tags, URLs, emojis and the lowercasing of the words, and a second one with fully preprocessed texts that included, in addition to the above, the lemmatization of the words and the exclusion of the stop words and punctuation symbols.

The results of the experiments were really encouraging for both English and Greek. In terms of accuracy, most of our models achieved results of between 82 and 92 percent. More precisely, our top two models were the BERT model combined with the Word Embeddings technique and the SVM model combined with the TF-IDF technique. Both models achieved results of around 90 percent for the English dataset in all metrics. For the Greek dataset, BERT achieved results close to or higher than 90 percent in all metrics, and SVM achieved results of between 85.5 and 87 percent depending on the preprocessing of the texts.

Starting with some general conclusions of our work, we can see that the preprocessing of the data (lemmatization, stop words removal, etc.) does not contribute significantly to a problem which contains such a large number of reviews. On the contrary, the results of the experiments without preprocessing were better. It makes sense for problems with limited data to use preprocessing because limited data means embedding space with limited dimensions which is sensitive to frequently occurring words (such as the stop words), but in our case with datasets with tens of thousands of reviews there is no such need because we have the necessary size to train our models successfully. The only thing we can achieve with preprocessing is to lose important information from our data.

A second conclusion is that the Decision Trees model does not perform well in a text classification problem. In all our experiments it was by far the worst classifier. The rest of the models achieved, more or less, the same results with the exception of the BERT model that was the most successful of all and the model that we recommend for text classification problems for both the English and Greek language.

Finally, we observed a constant difference in the results of all of the classification models in favor of the IMDb dataset (compared with the Athinorama Light), with the only exception being the BERT pretrained model. We believe that this difference is not due to the different languages (although this is a factor that influences the results), but mainly to the different number of tokens of the reviews on the datasets we used. In the English dataset we had 267 tokens per review, while in the Athinorama Light dataset we had 51 tokens per review, which meant that the IMDb dataset consisted of much more information in each of the reviews than the Athinorama Light, which made the

classification task easier for it.

In conclusion, we believe that the BERT pretrained model, especially its Greek version that is of interest to us, is a model that can cope with the needs of sentiment analysis for the Greek language and the results are just as good as for their English counterparts.

Future extensions of this work could include the creation of new datasets in the Greek language from different sources with different contents and the implementation of much more supervised and unsupervised classification techniques.

## References

- Khansa Afifah, Intan Nurma Yulita, Indra Sarathan (2021). *Sentiment Analysis on Telemedicine App Reviews using XGBoost Classifier*.
- Siddhant Singh, Hardeo K. Thakur (2020) Amity University. Amity Institute of Information Technology, Institute of Electrical and Electronics Engineers. Uttar Pradesh Section, & Institute of Electrical and Electronics Engineers. (2020). *ICRITO'2020 : IEEE 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) : conference date: 4-5 June 2020 : conference venue: Amity University, Noida, India*.
- Cambria, E. (2016). *Affective computing and sentiment analysis*. [www.sas.com/social](http://www.sas.com/social)
- Cambria, E., Poria, S., Gelbukh, A., & Thelwall, M. (2017). Sentiment Analysis Is a Big Suitcase. *IEEE Intelligent Systems*, 32(6), 74–80. <https://doi.org/10.1109/MIS.2017.4531228>
- Cambria, E., Schuller, B., Xia, Y., & Havasi, C. (2013). *New Avenues in Opinion Mining and Sentiment Analysis*. <http://converseon.com>
- Cambria, E., & White, B. (2014). Jumping NLP curves: A review of natural language processing research. In *IEEE Computational Intelligence Magazine* (Vol. 9, Issue 2, pp. 48–57). Institute of Electrical and Electronics Engineers Inc. <https://doi.org/10.1109/MCI.2014.2307227>
- Chaturvedi, I., Cambria, E., Welsch, R. E., & Herrera, F. (2018). Distinguishing between facts and opinions for sentiment analysis: Survey and challenges. *Information Fusion*, 44(December 2017), 65–77. <https://doi.org/10.1016/j.inffus.2017.12.006>
- Chen, T. (2016). *XGBoost : A Scalable Tree Boosting System*.
- Chomsky, N. (1957). *Chomsky-Syntactic-Structures-2Ed.Pdf*.
- Gao, Z., Feng, A., Song, X., & Wu, X. (2019). Target-dependent sentiment classification with BERT. *IEEE Access*, 7, 154290–154299. <https://doi.org/10.1109/ACCESS.2019.2946594>
- Gautam, G., & Yadav, D. (2014). Sentiment analysis of twitter data using machine learning approaches and semantic analysis. *2014 7th International Conference on Contemporary Computing, IC3 2014*, 437–442. <https://doi.org/10.1109/IC3.2014.6897213>

- Haque, M. R., Akter Lima, S., & Mishu, S. Z. (2019). Performance Analysis of Different Neural Networks for Sentiment Analysis on IMDb Movie Reviews. *3rd International Conference on Electrical, Computer and Telecommunication Engineering, ICECTE 2019*, 161–164. <https://doi.org/10.1109/ICECTE48615.2019.9303573>
- Harris, Z. S. (1954). Distributional Structure. *WORD*, 10(2–3), 146–162. <https://doi.org/10.1080/00437956.1954.11659520>
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
- Hu, Y., Shao, L., La, L., & Hua, H. (2021). *Using Investor and News Sentiment in Tourism Stock Price Prediction based on XGBoost Model*. 20–24.
- Hutchins, W. J. (2004). LNAI 3265 - The Georgetown-IBM Experiment Demonstrated in January 1954. In *LNAI* (Vol. 3265). <http://ourworld.compuserve.com/homepages/WJHutchins>
- Jianqiang, Z., & Xiaolin, G. (2017). Comparison research on text pre-processing methods on twitter sentiment analysis. *IEEE Access*, 5, 2870–2879. <https://doi.org/10.1109/ACCESS.2017.2672677>
- Jianqiang, Z., Xiaolin, G., & Xuejun, Z. (2018). Deep Convolution Neural Networks for Twitter Sentiment Analysis. *IEEE Access*, 6, 23253–23260. <https://doi.org/10.1109/ACCESS.2017.2776930>
- Kalamatianos, G., Mallis, D., Symeonidis, S., & Arampatzis, A. (2015). Sentiment analysis of Greek tweets and hashtags using a sentiment lexicon. *ACM International Conference Proceeding Series, 01-03-Octo(October)*, 63–68. <https://doi.org/10.1145/2801948.2802010>
- Kenton Lee, Ming-Wei Chang, Kristina Toutanova, & Jacob Devlin (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. *Mlm*.
- Markopoulos, G., Mikros, G., Iliadi, A., & Lontos, M. (2015). *Sentiment Analysis of Hotel Reviews in Greek: A Comparison of Unigram Features* (pp. 373–383). [https://doi.org/10.1007/978-3-319-15859-4\\_31](https://doi.org/10.1007/978-3-319-15859-4_31)
- Merri, B. Van. (2013). *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*.
- Nandwani, P., & Verma, R. (2021). A review on sentiment analysis and emotion detection from text. In *Social Network Analysis and Mining* (Vol. 11, Issue 1). Springer. <https://doi.org/10.1007/s13278-021-00776-6>
- Neethu, M. S., & Rajasree, R. (2013). Sentiment analysis in twitter using machine learning techniques. *2013 4th International Conference on Computing, Communications and Networking Technologies, ICCCNT 2013*. <https://doi.org/10.1109/ICCCNT.2013.6726818>
- P. Brown, J. Cocke, S. Deli, A. Pietra, V. Della Pietra, F. Jelinek and P. Roossin (1988) A statistical approach to language translation. IBM Research Division
- Qaisar, S. M. (2020). Sentiment Analysis of IMDb Movie Reviews Using Long Short-Term Memory. *2020 2nd International Conference on Computer and Information*

Sciences, ICCIS 2020, 2020–2023. <https://doi.org/10.1109/IC-CIS49240.2020.9257657>

- Rumelli, M., Akkuş, D., Kart, Ö., & Işık, Z. (2019). Türkçe Metinlerde Makine Öğrenmesi Algoritmaları ile Duygu Analizi Sentiment Analysis in Turkish Text with Machine Learning Algorithms. *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 1–5.
- Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2–6. <http://arxiv.org/abs/1910.01108>
- Schinas, E., & Herzig, J. (2013). *EventSense : capturing the pulse of large-scale events by mining social media EventSense : Capturing the Pulse of Large-scale Events by Mining Social Media Streams. January 2015*. <https://doi.org/10.1145/2491845.2491851>
- Sousa, M. G., Sakiyama, K., Rodrigues, L. D. S., Moraes, P. H., Fernandes, E. R., & Matsubara, E. T. (2019). BERT for stock market sentiment analysis. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI, 2019-Novem*, 1597–1601. <https://doi.org/10.1109/ICTAI.2019.00231>
- Spatiotis, N., Mporas, I., Paraskevas, M., & Perikos, I. (2016). Sentiment analysis for the Greek language. *ACM International Conference Proceeding Series*, 4–7. <https://doi.org/10.1145/3003733.3003769>
- Spatiotis, N., Perikos, I., Mporas, I., & Paraskevas, M. (2019). Examining the Impact of Discretization Technique on Sentiment Analysis for the Greek Language. *10th International Conference on Information, Intelligence, Systems and Applications, IISA 2019*, 1–6. <https://doi.org/10.1109/IISA.2019.8900699>
- Chi Sun, Xipeng Qiu, Yige Xu, Xuanjing Huang (2019). How to Fine-Tune BERT for Text Classification? *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11856 LNAI(2), 194–206. [https://doi.org/10.1007/978-3-030-32381-3\\_16](https://doi.org/10.1007/978-3-030-32381-3_16)
- Tripathi, S., Mehrotra, R., Bansal, V., & Upadhyay, S. (2020). Analyzing Sentiment using IMDb Dataset. *Proceedings - 2020 12th International Conference on Computational Intelligence and Communication Networks, CICN 2020*, 30–33. <https://doi.org/10.1109/CICN49253.2020.9242570>
- Tsakalidis, A., Papadopoulos, S., Voskaki, R., Ioannidou, K., Boididou, C., Cristea, A. I., Liakata, M., & Kompatsiaris, Y. (2018). Building and evaluating resources for sentiment analysis in the Greek language. *Language Resources and Evaluation*, 52(4), 1021–1044. <https://doi.org/10.1007/s10579-018-9420-4>
- Turing, A. M. (1950). COMPUTING MACHINERY AND INTELLIGENCE. In *Computing Machinery and Intelligence. Mind* (Vol. 49).
- Untawale, T. M., & Choudhari, G. (2019). Implementation of sentiment classification of movie reviews by supervised machine learning approaches. *Proceedings of the 3rd International Conference on Computing Methodologies and Communication, ICCMC 2019, Iccmc*, 1197–1200. <https://doi.org/10.1109/ICCMC.2019.8819800>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł.,

& Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems, 2017-Decem(Nips)*, 5999–6009.

Xue, Z., Yin, D., & Davison, B. D. (2011). Normalizing microtext. *AAAI Workshop - Technical Report, WS-11-05*(January), 74–79.

Yenter, A., & Verma, A. (2017). Deep CNN-LSTM with combined kernels from multiple branches for IMDb review sentiment analysis. *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2017, 2018-Janua*, 540–546. <https://doi.org/10.1109/UEMCON.2017.8249013>

## Image source

Xiaoli Fern (2008) Machine Learning and Data Mining lectures, School of Electrical Engineering and Computer Science Oregon State University. <http://web.engr.oregonstate.edu/~xfern/classes/cs434/slides/decisiontree-4.pdf>

## Appendix

ML\_models.py

```
import pandas as pd
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import string

stopwords_english = list(STOP_WORDS)
stopwords_greek = set(
    """
αδιάκοπα αι ακόμα ακόμη ακριβώς άλλα αλλά αλλαχού άλλες άλλη άλλην
άλλης αλλιώς αλλιώς αλλιώς άλλο άλλοι αλλοιώς αλλοιώτικα άλλον άλλος άλλοτε αλλού
άλλους άλλων άμα άμεσα άμέσως αν ανά ανάμεσα αναμεταξύ άνευ αντί αντίπερα αντίς
άνω ανωτέρω άξαφνα απ απέναντι από απόψε άρα άραγε αρκετά αρκετές
αρχικά ας αύριο αυτά αυτές αυτή αυτήν αυτής αυτό αυτοί αυτόν αυτός αυτού αυτούς
αυτών αφότου αφού
βέβαια βεβαιότατα
γι για γιατί γρήγορα γύρω
δα δε δείνα δεν δεξιά δήθεν δηλαδή δι δια διαρκώς δικά δικό δικοί δικός δικού
δικούς διόλου δίπλα δίχως
εάν εαυτό εαυτόν εαυτού εαυτούς εαυτών έγκαίρα εγκαίρως εγώ εδώ ειδημή είθε είμαι
```

είμαστε είναι εις είσαι είσαστε είστε είτε είχα είχαμε είχαν είχατε είχε είχες έκαστα  
έκαστες έκαστη έκαστην έκαστης έκαστο έκαστοι έκαστον έκαστος εκάστου εκάστους εκά-  
στων

εκεί εκείνα εκείνες εκείνη εκείνην εκείνης εκείνο εκείνοι εκείνον εκείνος εκείνου  
εκείνους εκείνων εκτός εμάς εμείς εμένα εμπρός εν ένα έναν ένας ενός εντελώς εντός  
εναντίον εξής εξαιτίας επιπλέον επόμενη εντωμεταξύ ενώ εξ έξαφνα εξήσο εξίσου έξω  
επάνω

επειδή έπειτα επί επίσης επομένως εσάς εσείς εσένα έστω εσύ ετέρα ετέραι ετέρας έτε-  
ρες

έτερη έτερης έτερο έτεροι έτερον έτερος ετέρου έτερους ετέρων ετούτα ετούτες ετούτη  
ετούτην

ετούτης ετούτο ετούτοι ετούτον ετούτος ετούτου ετούτους ετούτων έτσι εύγε ευθύς ευτυ-  
χώς εφεξής

έχει έχεις έχετε έχομε έχουμε έχουν εχτές έχω έως έγιναν έγινε έκανε έξι έχοντας  
η ήδη ήμασταν ήμαστε ήμουν ήσασταν ήσαστε ήσουν ήταν ήτανε ήτοι ήττον  
θα

ι ίδια ίδια ίδιαν ίδιας ίδιες ίδιο ίδιοι ίδιον ίδιου ίδιους ίδιων ιδίως  
ιι ιιι

ίσαμε ίσα ίσως

κάθε καθεμία καθεμίας καθένα καθένας καθενός καθετί καθόλου καθώς και κακά κακώς καλά  
καλώς καμία καμιάν καμίας κάμποσα κάμποσες κάμποση κάμποσην κάμποσης κάμποσο κάμποσοι  
κάμποσον κάμποσος κάμποσου κάμποσους κάμποσων κανείς κάνεν κανένα κανέναν κανέναν  
κανενός κάποια κάποιαν κάποιας κάποιες κάποιο κάποιον κάποιος κάποιου κά-  
ποιους

κάποιων κάποτε κάπου κάπως κατ κατά κάτι κατιτί κατόπιν κάτω κιόλας κλπ κοντά κτλ κυ-  
ρίως

λιγάκι λίγο λιγότερο λόγω λοιπά λοιπόν

μα μαζί μακάρι μακρυά μάλιστα μάλλον μας με μεθαύριο μείον μέλει μέλλεται μεμιάς μεν  
μερικά μερικές μερικοί μερικούς μερικών μέσα μετ μετά μεταξύ μέχρι μη μήδε μην μήπως  
μήτε μια μιαν μιας μόλις μολονότι μονάχα μόνες μόνη μόνην μόνης μόνο μόνον μονομιάς  
μόνος μόνου μόνους μόνων μου μπορεί μπορούν μπρος μέσω μία μεσώ

να ναι νωρίς

ξανά ξαφνικά

ο οι όλα όλες όλη όλην όλης όλο ολόγυρα όλοι όλον ολονέν όλος ολότελα όλου όλους όλων  
όλως ολωσδιόλου όμως όποια οποιαδήποτε οποίαν οποιαδήποτε οποίας οποίος οποιασδήποτε  
οποιδήποτε

όποιες οποιεσδήποτε όποιο οποιοδήποτε όποιοι όποιον οποιονδήποτε όποιος οποιοσδήποτε  
οποίου οποιουδήποτε οποιούς οποιουσδήποτε οποιών οποιωνδήποτε όποτε οποτεδήποτε όπου  
οπουδήποτε όπως ορισμένα ορισμένες ορισμένων ορισμένως όσα οσαδήποτε όσες οσεσδήποτε  
όση οσηδήποτε όσην οσηνήποτε όσης οσησδήποτε όσο οσοδήποτε όσοι οσοιδήποτε όσον ο-  
σονδήποτε

όσος οσοσδήποτε όσου οσουδήποτε όσους οσουσδήποτε όσων οσωνδήποτε όταν ότι οτιδήποτε  
ότου ου ουδέ ούτε όχι οποία οποιές οποίο οποίοι οπότε ος

πάνω παρά περί πολλά πολλές πολλοί πολλούς που πρώτα πρώ-

τες πρώτη πρώτο πρώτος πως

```

πάλι πάντα πάντοτε παντού πάντως πάρα πέρα πέρι περίπου περισσότερο πέρσι πέρυσι πια
πιθανόν
πιο πίσω πλάι πλέον πλην ποιά ποιάν ποιάς ποιές ποιό ποιοί ποιόν ποιός ποιού ποιούς
ποιών πολύ πόσες πόση πόσην πόσης πόσοι πόσος πόσους πότε ποτέ πού πούθε πουθενά πρέ-
πει
πριν προ προκειμένου πρόκειται πρόπερσι προς προτού προχθές προχτές πρωτύτερα πώς
σαν σας σε σεις σου στα στη στην στις στις στο στον στου στους στων συγχρόνως
συν συνάμα συνεπώς συχνάς συχνές συχνή συχνήν συχνής συχνό συχνοί συχνόν
συχνός συχνού συχνούς συχνών συχνώς σχεδόν
τα τάδε ταύτα ταύτες ταύτη ταύτην ταύτης ταύτοταύτον ταύτος ταύτου ταύτων τάχα τάχατε
τελευταία τελευταίο τελευταίος τού τρία τρίτη τρεις τελικά τελικώς τες τέτοια
τέτοιαν
τέτοιας τέτοιες τέτοιο τέτοιοι τέτοιον τέτοιος τέτοιοι
τέτοιους τέτοιων τη την της τι τίποτα τίποτε τις το τοι τον τος τόσα τόσες τόση τόσην
τόσης τόσο τόσοι τόσον τόσοσ τόσοσους τόσων τότε του τουλάχιστο τουλάχιστον τους
τούς τούτα
τούτες τούτη τούτην τούτης τούτο τούτοι τούτοις τούτον τούτος τούτου τούτους τούτων
τυχόν
των τώρα
υπ υπέρ υπό υπόψη υπόψιν ύστερα
χωρίς χωριστά
ω ως ωσάν ωσότου ώσπου ώστε ωστόσο ωχ
"".split()
)

punct = string.punctuation

nlp_greek = spacy.load('el_core_news_sm')
nlp_english = spacy.load('en_core_web_sm')

flag_1 = True
while flag_1 == True:
    dataset_selection = int(input("Select the dataset you want to use: give 0 for the
IMDb and 1 for the Athinorama "))
    if dataset_selection == 0:
        df = pd.read_csv('IMDb_50.000.csv')
        flag_1 =False
    elif dataset_selection == 1:
        df = pd.read_csv('Athinorama_50.000.csv')
        flag_1 =False
    else:
        print("Wrong number")

df['label'].replace({'negative':0, 'positive':1}, inplace=True)

```



```

flag_2 = True
while flag_2 == True:
    num_of_reviews = int(input("Give the number of reviews you want to use for Sentiment Analysis. (The number of reviews must be even and greater than 100) "))
    if num_of_reviews%2==0 and num_of_reviews>=100:
        df_0_labels = df[df['label']==0]
        df_1_labels = df[df['label']==1]
        df_0_sample = df_0_labels.sample(int(num_of_reviews/2))
        df_1_sample = df_1_labels.sample(int(num_of_reviews/2))
        df = df_0_sample.append(df_1_sample)
        print(df['label'].value_counts())
        flag_2 = False
    else:
        print("Remember the number of reviews must be even and greater than 100")

def text_data_cleaning(sentence):
    if dataset_selection == 0:
        doc = nlp_english(sentence)
    elif dataset_selection == 1:
        doc = nlp_greek(sentence)
    tokens = []
    for token in doc:
        if token.lemma_ != "-PRON-":
            temp = token.lemma_.lower().strip()
        else:
            temp = token.lower_
        tokens.append(temp)
    cleaned_tokens = []
    for token in tokens:
        if dataset_selection == 0:
            if token not in stopwords_english and token not in punct:
                cleaned_tokens.append(token)
        elif dataset_selection == 1:
            if token not in stopwords_greek and token not in punct:
                cleaned_tokens.append(token)
    return cleaned_tokens

def text_data_cleaning_no_lemmatization(sentence):
    if dataset_selection == 0:
        doc = nlp_english(sentence)
    elif dataset_selection == 1:
        doc = nlp_greek(sentence)
    tokens = []
    for token in doc:
        if token.lemma_ != "-PRON-":

```

```

        temp = token.lower_
    else:
        temp = token.lower_
    tokens.append(temp)
cleaned_tokens = []
for token in tokens:
    if token not in punct:
        cleaned_tokens.append(token)
return cleaned_tokens

flag_3 = True
while flag_3 == True:
    lemma_no_stop_words = int(input("You want to apply lemmatization and stop words
removal? Give 1 for Yes or 0 for No "))
    if lemma_no_stop_words == 1:
        tfidf = TfidfVectorizer(tokenizer = text_data_cleaning)
        flag_3 = False
    elif lemma_no_stop_words == 0:
        tfidf = TfidfVectorizer(tokenizer = text_data_cleaning_no_lemmatization)
        flag_3 = False
    else:
        print("Give 1 for Yes or 0 for No")

X = df['review']
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, ran-
dom_state = 42)

print('1 SVM')
from sklearn.svm import LinearSVC

classifier = LinearSVC()
clf = Pipeline([('tfidf', tfidf), ('clf', classifier)])
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
accuracy = (tp+tn)/(tp+fp+fn+tn)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1score = 2*((recall*precision)/(recall+precision))
print(confusion_matrix(y_test, y_pred))
print('Accuracy: {:.2f}%'.format(accuracy*100))
print('Precision: {:.2f}%'.format(precision*100))

```

```

print('Recall: {:.2f}%'.format(recall*100))
print('F1-score: {:.2f}%'.format(f1score*100))

print('')
print('2 Logistic Regression')
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression()
logreg = Pipeline([('tfidf',tfidf), ('logreg', classifier)])
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
accuracy = (tp+tn)/(tp+fp+fn+tn)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1score = 2*((recall*precision)/(recall+precision))
print(confusion_matrix(y_test, y_pred))
print('Accuracy: {:.2f}%'.format(accuracy*100))
print('Precision: {:.2f}%'.format(precision*100))
print('Recall: {:.2f}%'.format(recall*100))
print('F1-score: {:.2f}%'.format(f1score*100))

print('')
print('3 Decision Trees')
from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier()
dtt = Pipeline([('tfidf',tfidf), ('dtt', classifier)])
dtt.fit(X_train, y_train)
y_pred = dtt.predict(X_test)

tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
accuracy = (tp+tn)/(tp+fp+fn+tn)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1score = 2*((recall*precision)/(recall+precision))
print(confusion_matrix(y_test, y_pred))
print('Accuracy: {:.2f}%'.format(accuracy*100))
print('Precision: {:.2f}%'.format(precision*100))
print('Recall: {:.2f}%'.format(recall*100))
print('F1-score: {:.2f}%'.format(f1score*100))

print('')
print('4 Naïve Bayes')

```

```

from sklearn.naive_bayes import MultinomialNB

classifier = MultinomialNB()
mnb = Pipeline([('tfidf',tfidf), ('mnb', classifier)])
mnb.fit(X_train, y_train)
y_pred = mnb.predict(X_test)

tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
accuracy = (tp+tn)/(tp+fp+fn+tn)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1score = 2*((recall*precision)/(recall+precision))
print(confusion_matrix(y_test, y_pred))
print('Accuracy: {:.2f}%'.format(accuracy*100))
print('Precision: {:.2f}%'.format(precision*100))
print('Recall: {:.2f}%'.format(recall*100))
print('F1-score: {:.2f}%'.format(f1score*100))

print('')
print('5 XGBoost')
import xgboost as xgb

classifier = xgb.XGBClassifier(max_depth=7, n_estimators=300, objective="binary:lo-
gistic", random_state=42)
XGB = Pipeline([('tfidf',tfidf), ('XGB', classifier)])
XGB.fit(X_train, y_train)
y_pred = XGB.predict(X_test)

tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
accuracy = (tp+tn)/(tp+fp+fn+tn)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1score = 2*((recall*precision)/(recall+precision))
print(confusion_matrix(y_test, y_pred))
print('Accuracy: {:.2f}%'.format(accuracy*100))
print('Precision: {:.2f}%'.format(precision*100))
print('Recall: {:.2f}%'.format(recall*100))
print('F1-score: {:.2f}%'.format(f1score*100))

```

## DL\_models.py

```

import numpy as np
import pandas as pd
import spacy

```

```

from spacy.lang.en.stop_words import STOP_WORDS
from tensorflow.keras.layers import *
from tensorflow.keras.models import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.preprocessing.text import Tokenizer as tk
from keras.models import Sequential
from keras.layers import Dense
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.metrics import confusion_matrix
import string

stopwords_english = list(STOP_WORDS)
stopwords_greek = set(
    """
αδιάκοπα αι ακόμα ακόμη ακριβώς άλλα αλλά αλλαχού άλλες άλλη άλλην
άλλης αλλιώς αλλιώςτικά άλλο άλλοι αλλοιώς αλλοιώςτικά άλλον άλλος άλλοτε αλλού
άλλους άλλων άμα άμεσα αμέσως αν ανά ανάμεσα αναμεταξύ άνευ αντί αντίπερα αντίς
άνω ανωτέρω άξαφνα απ απέναντι από απόψε άρα άραγε αρκετά αρκετές
αρχικά ας αύριο αυτά αυτές αυτή αυτήν αυτής αυτό αυτοί αυτόν αυτός αυτού αυτούς
αυτών αφότου αφού
βέβαια βεβαιότατα
γι για γιατί γρήγορα γύρω
δα δε δείνα δεν δεξιά δήθεν δηλαδή δι δια διαρκώς δικά δικό δικοί δικός δικού
δικούς διόλου δίπλα δίχως
εάν εαυτό εαυτόν εαυτού εαυτούς εαυτών έγκαίρα εγκαίρως εγώ εδώ ειδημή είθε είμαι
είμαστε είναι εις είσαι είσατε είστε είτε είχα είχαμε είχαν είχατε είχε είχες έκαστα
έκαστες έκαστη έκαστην έκαστης έκαστο έκαστοι έκαστον έκαστος εκάστου εκάστους εκά-
στων
εκεί εκείνα εκείνες εκείνη εκείνην εκείνης εκείνο εκείνοι εκείνον εκείνος εκείνου
εκείνους εκείνων εκτός εμάς εμείς εμένα εμπρός εν ένα έναν ένας ενός εντελώς εντός
εναντίον εξής εξαιτίας επιπλέον επόμενη εντωμεταξύ ενώ εξ έξαφνα εξήσο εξίσου έξω
επάνω
επειδή έπειτα επί επίσης επομένως εσάς εσείς εσένα έστω εσύ ετέρα ετέραι ετέρας έτε-
ρες
έτερη έτερης έτερο έτεροι έτερον έτερος ετέρου έτερους ετέρων ετούτα ετούτες ετούτη
ετούτην
ετούτης ετούτο ετούτοι ετούτον ετούτος ετούτου ετούτους ετούτων έτσι εύγε ευθύς ευτυ-
χώς εφεξής
έχει έχεις έχετε έχομε έχουμε έχουν εχτές έχω έως έγιναν έγινε έκανε έξι έχοντας
η ήδη ήμασταν ήμαστε ήμουν ήσασταν ήσαστε ήσουν ήταν ήτανε ήτοι ήττον
θα
ι ιδία ίδια ιδιαν ιδίας ίδιες ίδιο ίδιοι ίδιον ίδιοσ ίδιος ιδίου ίδιους ίδιων ιδίως
ιι ιιι

```

ίσαμε ίσια ίσως  
κάθε καθεμία καθεμίας καθένα καθένας καθενός καθετί καθόλου καθώς και κακά κακώς καλά  
καλώς καμία καμιά καμίας κάμποσα κάμποσες κάμποση κάμποσην κάμποσης κάμποσο κάμποσοι  
κάμποσον κάμποσος κάμποσου κάμποσους κάμποσων κανείς κάνεν κανένα κανέναν κανένας  
κανενός κάποια κάποιαν κάποιας κάποιες κάποιον κάποιοι κάποιον κάποιος κάποιου κά-  
ποιους  
κάποιων κάποτε κάπου κάπως κατ κατά κάτι κατιτί κατόπιν κάτω κιόλας κλπ κοντά κτλ κυ-  
ρίως  
λιγάκι λίγο λιγότερο λόγω λοιπά λοιπόν  
μα μαζί μακάρι μακρυνά μάλιστα μάλλον μας με μεθαύριο μείον μέλει μέλλεται μεμιάς μεν  
μερικά μερικές μερικοί μερικούς μερικών μέσα μετ μετά μεταξύ μέχρι μη μήδε μην μήπως  
μήτε μια μιαν μιας μόλις μολονότι μονάχα μόνες μόνη μόνην μόνης μόνο μόνι μονομιάς  
μόνος μόνου μόνους μόνων μου μπορεί μπορούν μπρος μέσω μία μεσώ  
να ναι νωρίς  
ξανά ξαφνικά  
ο οι όλα όλες όλη όλην όλης όλο ολόγυρα όλοι όλον ολονέν όλος ολότελα όλου όλους όλων  
όλως ολωσδιόλου όμως όποια οποιαδήποτε οποιαν οποιαδήποτε οποίας οποιός οποιασδήποτε  
οποιδήποτε  
όποιες οποιεσδήποτε όποιο οποιοδήποτε όποιοι όποιον οποιονδήποτε όποιος οποιοσδήποτε  
οποίου οποιουδήποτε οποιούς οποιουσδήποτε οποιών οποιωνδήποτε όποτε οποτεδήποτε όπου  
οπουδήποτε όπως ορισμένα ορισμένες ορισμένων ορισμένως όσα οσαδήποτε όσες οσεσδήποτε  
όση οσηδήποτε όσην οσηνδήποτε όσης οσησδήποτε όσο οσοδήποτε όσοι οσοιδήποτε όσον ο-  
σονδήποτε  
όσος οσοσδήποτε όσου οσουδήποτε όσους οσουσδήποτε όσων οσωνδήποτε όταν ότι οτιδήποτε  
ότου ου ουδέ ούτε όχι οποία οποιές οποίο οποίοι οπότε ος  
πάνω παρά περί πολλά πολλές πολλοί πολλούς που πρώτα πρώ-  
τες πρώτη πρώτο πρώτος πως  
πάλι πάντα πάντοτε παντού πάντως πάρα πέρα πέρι περίπου περισσότερο πέρσι πέρυσι πια  
πιθανόν  
πιο πίσω πλάι πλέον πλην ποιά ποιάν ποιιάς ποιές ποιοί ποιόν ποιός ποιού ποιούς  
ποιών πολύ πόσες πόση πόσην πόσης πόσοι πόσος πόσους πότε ποτέ πού πούθε πουθενά πρέ-  
πει  
πριν προ προκειμένου πρόκειται πρόπερσι προς προτού προχθές προχτές πρωτύτερα πώς  
σαν σας σε σεεις σου στα στη στην στις στις στο στον στου στους στων συγχρόνως  
συν συνάμα συνεπώς συχνάς συχνές συχνή συχνήν συχνής συχνό συχνοί συχνόν  
συχνός συχνού συχνούς συχνών συχνώς σχεδόν  
τα τάδε ταύτα ταύτες ταύτη ταύτην ταύτης ταύτοταύτον ταύτος ταύτου ταύτων τάχα τάχατε  
τελευταία τελευταίο τελευταίος τού τρία τρίτη τρεις τελικά τελικώς τες τέτοια  
τέτοια  
τέτοιας τέτοιες τέτοιο τέτοιοι τέτοιον τέτοιος τέτοιου  
τέτοιους τέτοιων τη την της τι τίποτα τίποτε τις το τοι τον τος τόσα τόσες τόση τόσην  
τόσης τόσο τόσοι τόσον τόσος τόσοσ τόσοσους τόσων τότε του τουλάχιστο τουλάχιστον τους  
τούς τούτα  
τούτες τούτη τούτην τούτης τούτο τούτοι τούτοις τούτον τούτος τούτου τούτους τούτων  
τυχόν

```

των τώρα
υπ υπέρ υπό υπόψη υπόψιν ύστερα
χωρίς χωριστά
ω ως ωσάν ωσότου ώσπου ώστε ωστόσο ωχ
""" .split()
)

punct = string.punctuation

encoder = LabelEncoder()
nlp_greek = spacy.load("el_core_news_sm")
nlp_english = spacy.load("en_core_web_sm")

flag_1 = True
while flag_1 == True:
    dataset_selection = int(input("Select the dataset you want to use: give 0 for the
IMDb and 1 for the Athinorama "))
    if dataset_selection == 0:
        df = pd.read_csv('IMDb_50.000.csv')
        flag_1 = False
    elif dataset_selection == 1:
        df = pd.read_csv('Athinorama_50.000.csv')
        flag_1 = False
    else:
        print('Wrong number')

df['label'].replace({'negative':0, 'positive':1}, inplace=True)

flag_2 = True
while flag_2 == True:
    num_of_reviews = int(input("Give the number of reviews you want to use for Senti-
ment Analysis. (The number of reviews must be even and greater than 100) "))
    if num_of_reviews%2==0 and num_of_reviews>=100:
        df_0_labels = df[df['label']==0]
        df_1_labels = df[df['label']==1]
        df_0_sample = df_0_labels.sample(int(num_of_reviews/2))
        df_1_sample = df_1_labels.sample(int(num_of_reviews/2))
        df = df_0_sample.append(df_1_sample)
        print(df['label'].value_counts())
        flag_2 = False
    else:
        print("Remember the number of reviews must be even and greater than 100")

def text_data_cleaning(sentence):
    if dataset_selection == 0:

```

```

    doc = nlp_english(sentence)
elif dataset_selection == 1:
    doc = nlp_greek(sentence)
tokens = []
for token in doc:
    if token.lemma_ != "-PRON-":
        temp = token.lemma_.lower().strip()
    else:
        temp = token.lower_
    tokens.append(temp)
cleaned_tokens = []
for token in tokens:
    if dataset_selection == 0:
        if token not in stopwords_english and token not in punct:
            cleaned_tokens.append(token)
    elif dataset_selection == 1:
        if token not in stopwords_greek and token not in punct:
            cleaned_tokens.append(token)
return cleaned_tokens

flag_3 = True
while flag_3 == True:
    preprocessing = int(input("You want preprocessing? Give 0 for No and 1 for Yes
"))
    if preprocessing == 1:
        X = []
        counter = 0
        for review in df['review']:
            cleaned_review = ''
            if counter<10:
                print(counter,review)
            review = text_data_cleaning(review)
            for word in review:
                cleaned_review += ' '+word
            if counter<10:
                print(counter,cleaned_review)
                counter +=1
            X.append(cleaned_review)
        y = np.array(df['label'])
        flag_3 = False
    elif preprocessing == 0:
        X = df['review']
        y = df['label']
        flag_3 = False
    else:

```



```

    print("Remember! Give True for Yes and False for No ")

X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True, test_size =
0.1, stratify=y, random_state = 42)

#Deep Learning hyperparameters
embedding_vector_length = 32
max_review_length = 100
NUM_WORDS = 10000
batch_size = 32
epochs = 10

tok=tk(num_words=NUM_WORDS, filters='!"#$%&()*+,-./:;<=>@[\\]^_`{|}~\t\n',
lower=True, split=' ', char_level=False, oov_token='Farid')
tok.fit_on_texts(X_train)
X_train1 = tok.texts_to_sequences(X_train)
X_test1 = tok.texts_to_sequences(X_test)
X_train2 = pad_sequences(X_train1, max_review_length, padding='post', truncat-
ing='post')
X_test2 = pad_sequences(X_test1, max_review_length, padding='post', truncat-
ing='post')
X_train3 = np.flip(X_train2, 1)
X_test3 = np.flip(X_test2, 1)

print('')
print('6 GRU')

model_1 = Sequential()
model_1.add(Embedding(input_dim=NUM_WORDS, output_dim=embedding_vector_length, in-
put_length=max_review_length))
model_1.add(Conv1D(filters=100, kernel_size=3, padding='same', activation='relu'))
model_1.add(MaxPooling1D())
model_1.add(GRU(50, return_sequences=True))
model_1.add(GRU(50))
model_1.add(Dropout(0.5))
model_1.add(Dense(100,activation='relu'))
model_1.add(Dense(1, activation='sigmoid'))
model_1.compile(optimizer='adam', loss='binary_crossentropy',metrics='accuracy')
model_1.summary()
model_1.fit(X_train3, y_train, batch_size=batch_size, epochs=epochs)

y_pred_test = (model_1.predict(X_test3) > 0.5).astype('int32').reshape(len(X_test3))
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_test).ravel()
accuracy = (tp+tn)/(tp+fp+fn+tn)
precision = tp/(tp+fp)

```

```

recall = tp/(tp+fn)
f1score = 2*((recall*precision)/(recall+precision))
print(confusion_matrix(y_test, y_pred_test))
print('Accuracy: {:.2f}%'.format(accuracy*100))
print('Precision: {:.2f}%'.format(precision*100))
print('Recall: {:.2f}%'.format(recall*100))
print('F1-score: {:.2f}%'.format(f1score*100))

print('')
print('7 LSTM')

model_2 = Sequential()
model_2.add(Embedding(input_dim=NUM_WORDS, output_dim=embedding_vector_length, input_length=max_review_length))
model_2.add(Dropout(0.2))
model_2.add(LSTM(32))
model_2.add(Dense(units=256, activation='relu'))
model_2.add(Dropout(0.2))
model_2.add(Dense(units=1, activation='sigmoid'))
model_2.summary()
model_2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model_2.fit(X_train3, y_train, batch_size=batch_size, epochs=epochs)

y_pred_test = (model_2.predict(X_test3) > 0.5).astype('int32').reshape(len(X_test3))
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_test).ravel()
accuracy = (tp+tn)/(tp+fp+fn+tn)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1score = 2*((recall*precision)/(recall+precision))
print(confusion_matrix(y_test, y_pred_test))
print('Accuracy: {:.2f}%'.format(accuracy*100))
print('Precision: {:.2f}%'.format(precision*100))
print('Recall: {:.2f}%'.format(recall*100))
print('F1-score: {:.2f}%'.format(f1score*100))

print('')
print('8 CNN')

model_3 = Sequential()
model_3.add(Embedding(NUM_WORDS, 100, input_length=max_review_length))
model_3.add(Conv1D(1024, 3, padding='valid', activation='relu', strides=1))
model_3.add(GlobalMaxPooling1D())
model_3.add(Dropout(0.5))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

```

```

model_3.add(Dense(2048, activation='relu'))
model_3.add(Dropout(0.5))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))
model_3.add(Dense(1, activation='sigmoid'))
model_3.summary()
model_3.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model_3.fit(X_train3, y_train, batch_size=batch_size, epochs=epochs)

y_pred_test = (model_3.predict(X_test3) > 0.5).astype('int32').reshape(len(X_test3))
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_test).ravel()
accuracy = (tp+tn)/(tp+fp+fn+tn)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1score = 2*((recall*precision)/(recall+precision))
print(confusion_matrix(y_test, y_pred_test))
print('Accuracy: {:.2f}%'.format(accuracy*100))
print('Precision: {:.2f}%'.format(precision*100))
print('Recall: {:.2f}%'.format(recall*100))
print('F1-score: {:.2f}%'.format(f1score*100))

```

## BERT\_models.py (colab code)

```

# !pip install transformers
# !pip install --upgrade spacy
# !pip install https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.0.0/en_core_web_sm-3.0.0.tar.gz

import spacy.cli
spacy.cli.download("en_core_news_sm")

import tensorflow as tf
import numpy as np
import pandas as pd
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
import string
from transformers import BertTokenizer, TFBertForSequenceClassification
from transformers import InputExample, InputFeatures
from transformers import AutoTokenizer

from google.colab import drive
drive.mount('/content/drive')

```

```
stopwords_english = list(STOP_WORDS)
stopwords_greek = set(
    ""
```

αδιάκοπα αι ακόμα ακόμη ακριβώς άλλα αλλά αλλαχού άλλες άλλη άλλην  
άλλης αλλιώς αλλιώς αλλιώς άλλο άλλοι αλλοιώς αλλοιώτικα άλλον άλλος άλλοτε αλλού  
άλλους άλλων άμα άμεσα αμέσως αν ανά ανάμεσα αναμεταξύ άνευ αντί αντίπερα αντίς  
άνω ανωτέρω άξαφνα απ απέναντι από απόψε άρα άραγε αρκετά αρκετές  
αρχικά ας αύριο αυτά αυτές αυτή αυτήν αυτής αυτό αυτοί αυτόν αυτός αυτού αυτούς  
αυτών αφότου αφού  
βέβαια βεβαιότατα  
γι για γιατί γρήγορα γύρω  
δα δε δεινά δεν δεξιά δήθεν δηλαδή δι δια διαρκώς δικά δικό δικοί δικός δικού  
δικούς διόλου δίπλα δίχως  
εάν εαυτό εαυτόν εαυτού εαυτούς εαυτών έγκαιρα εγκαίρως εγώ εδώ ειδημή είθε είμαι  
είμαστε είναι εις είσαι είσατε είστε είτε είχα είχαμε είχαν είχατε είχε είχες έκαστα  
έκαστες έκαστη έκαστην έκαστης έκαστο έκαστοι έκαστον έκαστος εκάστου εκάστους εκά-  
στων  
εκεί εκείνα εκείνες εκείνη εκείνην εκείνης εκείνο εκείνοι εκείνον εκείνος εκείνου  
εκείνους εκείνων εκτός εμάς εμείς εμένα εμπρός εν ένα έναν ένας ενός εντελώς εντός  
εναντίον εξής εξαιτίας επιπλέον επόμενη εντωμεταξύ ενώ εξ έξαφνα εξήσο εξίσου έξω  
επάνω  
επειδή έπειτα επί επίσης επομένως εσάς εσείς εσένα έστω εσύ ετέρα ετέραι ετέρας έτε-  
ρες  
έτερη έτερης έτερο έτεροι έτερον έτερος ετέρου έτερους ετέρων ετούτα ετούτες ετούτη  
ετούτην  
ετούτης ετούτο ετούτοι ετούτον ετούτος ετούτου ετούτους ετούτων έτσι εύγε ευθύς ευτυ-  
χώς εφεξής  
έχει έχεις έχετε έχομε έχουμε έχουν εχτές έχω έως έγιναν έγινε έκανε έξι έχοντας  
η ήδη ήμασταν ήμαστε ήμουν ήσασταν ήσαστε ήσουν ήταν ήτανε ήτοι ήττον  
θα  
ι ίδια ίδια ίδιαν ιδίας ίδιες ίδιο ίδιοι ίδιον ίδιουσ ίδιος ιδίου ίδιους ίδιων ιδίως  
ιι ιιι  
ίσαμε ίσια ίσως  
κάθε καθεμία καθεμίας καθένα καθένας καθενός καθετί καθόλου καθώς και κακά κακώς καλά  
καλώς καμία καμιά καμίας κάμποσα κάμποσες κάμποση κάμποσην κάμποσης κάμποσο κάμποσοι  
κάμποσον κάμποσος κάμποσου κάμποσους κάμποσων κανείς κανέν κανένα κανέναν κανέναν  
κανενός κάποια κάποιαν κάποιας κάποιες κάποιο κάποιον κάποιος κάποιου κά-  
ποιους  
κάποιων κάποτε κάπου κάπως κατ κατά κάτι κατιτί κατόπιν κάτω κίολας κλπ κοντά κτλ κυ-  
ρίως  
λιγάκι λίγο λιγότερο λόγω λοιπά λοιπόν  
μα μαζί μακάρι μακρυνά μάλλιστα μάλλον μας με μεθαύριο μείον μέλει μέλλεται μεμιάς μεν  
μερικά μερικές μερικοί μερικούς μερικών μέσα μετ μετά μεταξύ μέχρι μη μήδε μην μήπως  
μήτε μια μιαν μιας μόλις μολονότι μονάχα μόνες μόνη μόνην μόνης μόνο μόνον μονομιάς  
μόνος μόνου μόνους μόνων μου μπορεί μπορούν προς μέσω μία μεσώ

```

να ναι νωρίς
ξανά ξαφνικά
ο οι όλα όλες όλη όλην όλης όλο ολόγυρα όλοι όλον ολονέν όλος ολότελα όλου όλους όλων
όλως ολωσδιόλου όμως όποια οποιαδήποτε οποίαν οποιαδήποτε οποίας οποίος οποιασδήποτε
οποιδήποτε
όποιες οποιεσδήποτε όποιο οποιοδήποτε όποιοι όποιον οποιονδήποτε όποιος οποιοσδήποτε
οποίου οποιουδήποτε οποιούς οποιουσδήποτε οποιών οποιωνδήποτε όποτε οποτεδήποτε όπου
οπουδήποτε όπως ορισμένα ορισμένες ορισμένων ορισμένως όσα οσαδήποτε όσες οσεσδήποτε
όση οσηδήποτε όσην οσηνδήποτε όσης οσησδήποτε όσο οσοδήποτε όσοι οσοιδήποτε όσον ο-
σονδήποτε
όσος οσοσδήποτε όσου οσουδήποτε όσους οσουσδήποτε όσων οσωνδήποτε όταν ότι οτιδήποτε
ότου ου ουδέ ούτε όχι οποία οποίες οποίο οποιίοι οπότε ος
πάνω παρά περί πολλά πολλές πολλοί πολλούς που πρώτα πρώ-
τες πρώτη πρώτο πρώτος πως
πάλι πάντα πάντοτε παντού πάντως πάρα πέρα πέρυι περίπου περισσότερο πέρυι πέρυι πια
πιθανόν
πιο πίσω πλάι πλέον πλην ποιά ποιάν ποιάς ποιές ποιό ποιοί ποιόν ποιός ποιού ποιούς
ποιών πολύ πόσες πόση πόσην πόσης πόσοι πόσος πόσους πότε ποτέ πού πούθε πουθενά πρέ-
πει
πριν προ προκειμένου πρόκειται πρόπερς προς προτού προχθές προχτές πρωτύτερα πώς
σαν σας σε σεις σου στα στη στην στις στις στο στον στου στους στων συγχρόνως
συν συνάμα συνεπώς συχνάς συχνές συχνή συχνήν συχνής συχνό συχνοί συχνών
συχνός συχνού συχνούς συχνών συχνώς σχεδόν
τα τάδε ταύτα ταύτες ταύτη ταύτην ταύτης ταύτοταύτον ταύτος ταύτου ταύτων τάχα τάχατε
τελευταία τελευταίο τελευταίος τού τρία τρίτη τρεις τελικά τελικώς τες τέτοια
τέτοιαν
τέτοιας τέτοιες τέτοιο τέτοιοι τέτοιον τέτοιος τέτοιου
τέτοιους τέτοιων τη την της τι τίποτα τίποτε τις το τοι τον τος τόσα τόσες τόση τόσην
τόσης τόσο τόσοι τόσον τόσος τόσοσ τόσοσους τόσοσων τότε του τουλάχιστο τουλάχιστον τους
τούς τούτα
τούτες τούτη τούτην τούτης τούτο τούτοι τούτοις τούτον τούτος τούτου τούτους τούτων
τυχόν
των τώρα
υπ υπέρ υπό υπόψη υπόψιν ύστερα
χωρίς χωριστά
ω ως ωσάν ωσότου ώσπου ώστε ωστόσο ωχ
"" .split()
)

punct = string.punctuation

nlp_greek = spacy.load("el_core_news_sm")
nlp_english = spacy.load("en_core_web_sm")

flag_1 = True

```

```

while flag_1 == True:
    dataset_selection = int(input("Select the dataset you want to use: give 0 for the
IMDb and 1 for the Athinorama "))
    if dataset_selection == 0:
        df = pd.read_csv('/content/drive/MyDrive/Datasets/IMDb_50.000.csv')
        model = TFBertForSequenceClassification.from_pretrained("bert-base-uncased")
        tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
        flag_1 =False
    elif dataset_selection == 1:
        df = pd.read_csv(r'/content/drive/MyDrive/Datasets/Athinorama_50.000.csv')
        tokenizer = AutoTokenizer.from_pretrained("nlpauueb/bert-base-greek-uncased-
v1")
        model = TFBertForSequenceClassification.from_pretrained("nlpauueb/bert-base-
greek-uncased-v1")
        flag_1 =False
    else:
        print('Wrong number')

df['label'].replace({'negative':0, 'positive':1}, inplace=True)

flag_2 = True
while flag_2 == True:
    num_of_reviews = int(input("Give the number of reviews you want to use for Senti-
ment Analysis. (The number of reviews must be even and greater than 100) "))
    if num_of_reviews%2==0 and num_of_reviews>=100:
        df_0_labels = df[df['label']==0]
        df_1_labels = df[df['label']==1]
        df_0_sample = df_0_labels.sample(int(num_of_reviews/2))
        df_1_sample = df_1_labels.sample(int(num_of_reviews/2))
        df = df_0_sample.append(df_1_sample)
        print(df['label'].value_counts())
        flag_2 = False
    else:
        print("Remember the number of reviews must be even and greater than 100")

def text_data_cleaning(sentence):
    if dataset_selection == 0:
        doc = nlp_english(sentence)
    elif dataset_selection == 1:
        doc = nlp_greek(sentence)
    tokens = []
    for token in doc:
        if token.lemma_ != "-PRON-":
            temp = token.lemma_.lower().strip()
        else:

```

```

        temp = token.lower_
        tokens.append(temp)
    cleaned_tokens = []
    for token in tokens:
        if dataset_selection == 0:
            if token not in stopwords_english and token not in punct:
                cleaned_tokens.append(token)
        elif dataset_selection == 1:
            if token not in stopwords_greek and token not in punct:
                cleaned_tokens.append(token)
    return cleaned_tokens

flag_3 = True
while flag_3 == True:
    preprocessing = int(input("You want preprocessing? Give 0 for No and 1 for Yes
"))
    if preprocessing == 1:
        X = []
        counter = 0
        for review in df['review']:
            cleaned_review = ''
            if counter<10:
                print(counter,review)
            review = text_data_cleaning(review)
            for word in review:
                cleaned_review += ' '+word
            if counter<10:
                print(counter,cleaned_review)
                counter +=1
            X.append(cleaned_review)
        y = np.array(df['label'])
        flag_3 = False
    elif preprocessing == 0:
        X = df['review']
        y = df['label']
        flag_3 = False
    else:
        print("Remember! Give True for Yes and False for No ")

df_review = pd.DataFrame(X)
df_label = pd.DataFrame(y)
df = df_review.assign(label=df_label)

df = df.rename(columns={'review': 'DATA_COLUMN', 'label': 'LABEL_COLUMN'})

```

```

df_0 = df[df['LABEL_COLUMN']==0]
df_1 = df[df['LABEL_COLUMN']==1]

df_train_0 = df_0.iloc[:22500,:]
df_train_1 = df_1.iloc[:22500,:]
train = df_train_0.append(df_train_1)

df_test_0 = df_0.iloc[22500:25000,:]
df_test_1 = df_1.iloc[22500:25000,:]
test = df_test_0.append(df_test_1)

train = train.sample(frac=1)
test = test.sample(frac=1)

def convert_data_to_examples(train, test, DATA_COLUMN, LABEL_COLUMN):
    train_InputExamples = train.apply(lambda x: InputExample(guid=None, # Globally
unique ID for bookkeeping, unused in this case
                                text_a = x[DATA_COLUMN],
                                text_b = None,
                                label = x[LABEL_COLUMN]),
axis = 1)

    test_InputExamples = test.apply(lambda x: InputExample(guid=None, # Globally unique
ID for bookkeeping, unused in this case
                                text_a = x[DATA_COLUMN],
                                text_b = None,
                                label = x[LABEL_COLUMN]),
axis = 1)

    return train_InputExamples, test_InputExamples

train_InputExamples, test_InputExamples = convert_data_to_examples(train,
                                                                    test,
                                                                    'DATA_COLUMN',
                                                                    'LABEL_COLUMN')

def convert_examples_to_tf_dataset(examples, tokenizer, max_length=128):
    features = [] # -> will hold InputFeatures to be converted later

    for e in examples:
        # Documentation is really strong for this method, so please take a look at it
        input_dict = tokenizer.encode_plus(
            e.text_a,

```



```

        add_special_tokens=True,
        max_length=max_length, # truncates if len(s) > max_length
        return_token_type_ids=True,
        return_attention_mask=True,
        pad_to_max_length=True, # pads to the right by default # CHECK THIS for
pad_to_max_length
        truncation=True
    )

    input_ids, token_type_ids, attention_mask = (input_dict["input_ids"],
        input_dict["token_type_ids"], input_dict['attention_mask'])

    features.append(
        InputFeatures(
            input_ids=input_ids, attention_mask=attention_mask, to-
ken_type_ids=token_type_ids, label=e.label
        )
    )

def gen():
    for f in features:
        yield (
            {
                "input_ids": f.input_ids,
                "attention_mask": f.attention_mask,
                "token_type_ids": f.token_type_ids,
            },
            f.label,
        )

return tf.data.Dataset.from_generator(
    gen,
    ({"input_ids": tf.int32, "attention_mask": tf.int32, "token_type_ids":
tf.int32}, tf.int64),
    (
        {
            "input_ids": tf.TensorShape([None]),
            "attention_mask": tf.TensorShape([None]),
            "token_type_ids": tf.TensorShape([None]),
        },
        tf.TensorShape([]),
    ),
)

```

```
DATA_COLUMN = 'DATA_COLUMN'
```

```

LABEL_COLUMN = 'LABEL_COLUMN'

train_InputExamples, test_InputExamples = convert_data_to_examples(train, test,
DATA_COLUMN, LABEL_COLUMN)

train_data = convert_examples_to_tf_dataset(list(train_InputExamples), tokenizer)
train_data = train_data.shuffle(100).batch(32).repeat(2)

test_data = convert_examples_to_tf_dataset(list(test_InputExamples), tokenizer)
test_data = test_data.batch(32)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=3e-5, epsilon=1e-08,
clipnorm=1.0),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))

model.fit(train_data, epochs=1)

tp = 0
fp = 0
tn = 0
fn = 0
for review, label in zip(test['DATA_COLUMN'], test['LABEL_COLUMN']):
    tf_batch = tokenizer(review, max_length=128, padding=True, truncation=True, re-
turn_tensors='tf')
    tf_outputs = model(tf_batch)
    tf_predictions = tf.nn.softmax(tf_outputs[0], axis=-1)
    pred_label = tf.argmax(tf_predictions, axis=1)
    if pred_label == 1 and label == 1:
        tp +=1
    elif pred_label == 1 and label == 0:
        fp +=1
    elif pred_label == 0 and label == 0:
        tn +=1
    else:
        fn +=1

accuracy = (tp+tn)/(tp+fp+fn+tn)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1score = 2*((recall*precision)/(recall+precision))
print([tp,fn])
print([fp,tn])
print('Accuracy: {:.2f}%'.format(accuracy*100))
print('Precision: {:.2f}%'.format(precision*100))
print('Recall: {:.2f}%'.format(recall*100))

```

```
print('F1-score: {:.2f}%'.format(f1score*100))
```