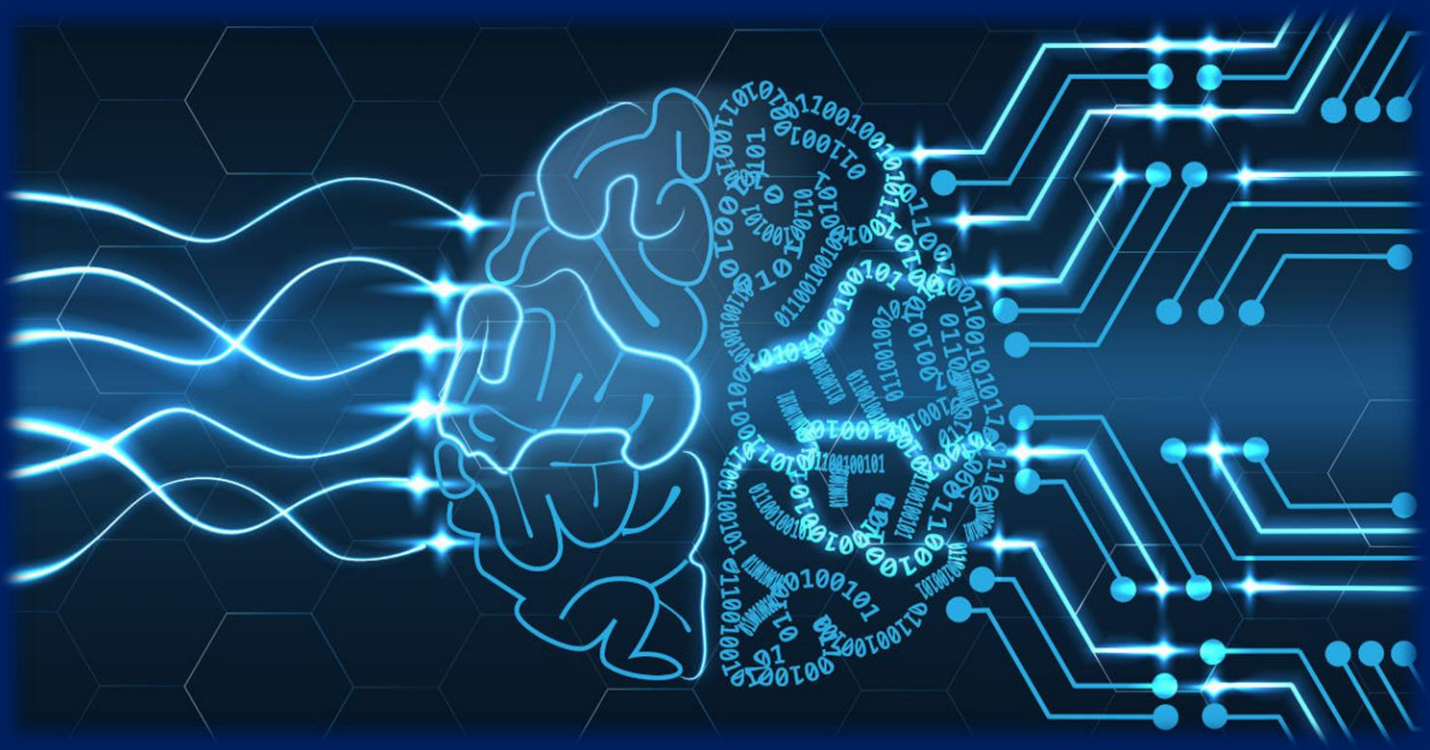
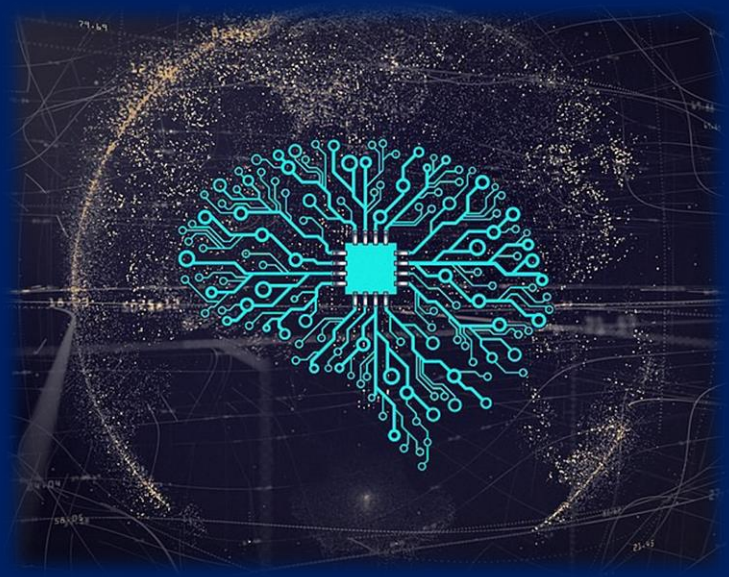
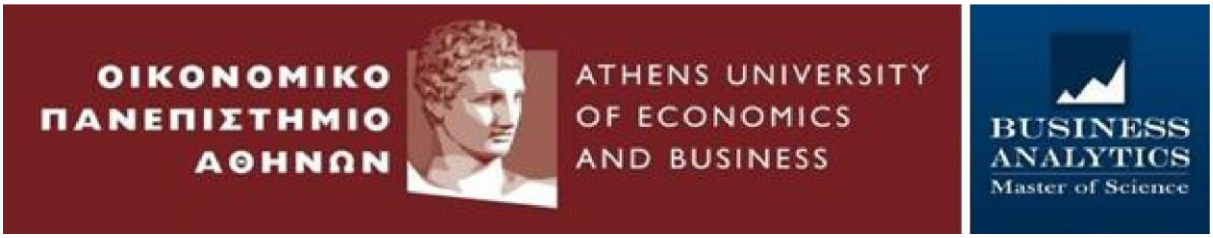




Machine Learning

Neural Networks



Offensive language & hate speech detection

COURSE: MACHINE LEARNING & CONTENT ANALYTICS – PART TIME SPRING 2022

MINI PROJECT 2021-2022 – DUE DATE: SEPTEMBER 4, 2022

Professor: Haris Papageorgiou (xaris@ilsp.gr)

Assistant responsible for this assignment: George Perakis (gperakis@aueb.gr)

Students: Arkoumani Georgia (**p2822104**), Koutsodimitropoulou Anastasia (**p2822119**), Poulou Myrto (**p2822129**) & Zaragka Eftychia (**p2822112**)

Introduction

In the era of social media and communications, it is easier than ever to freely express opinions on a variety of topics. This openness creates a proliferation of useful information for productivity and making the right decisions. With the greatness of social media benefits, unfortunately it also brought up opportunities for harsh discussions that can easily reach uncivilized, hateful, offensive or toxic levels.

Offensive online speech has become a crucial problem nowadays due to an exponential increase in the use of the internet by people from different cultures and educational backgrounds.

Hate speech can be defined as any speech that targets a group of people based on their race, religion, ethnicity, national origin, sexual orientation, or gender identity. It can also be used to intimidate and threaten people. It can make people feel isolated, anxious, and scared or even lead to hate crimes. Thus, detection of hate speech is important because it can help prevent these harmful effects.

In recent times, social media has become a hotbed for hate speech. Hate speech on social media can have harmful effects on society. Machine learning algorithms can be used to detect hate speech by analyzing texts. A variety of methods have been explored for the hate speech detection task, including traditional classifiers, deep learning-based classifiers, or the combination of both approaches. On the other hand, there have been a number of dataset benchmarks introduced and released for the evaluation of the performance of these methods.

Therefore, the following project describes a business case that aims to provide a thorough empirical evaluation and comparison of different types of hate speech detection methods on a variety of datasets.

Business Case

COMPANY DESCRIPTION

WHO WE ARE

Our *start-up* company **QC Greece** was first established in 2021 from a group of friends, who wanted with their product to contribute to the elimination of offensive language in social media. **QC Greece** is an *Information Technology* and *data-driven* company that provides the '**hateless**' app, an application created with machine learning and neural network algorithms.

HOW IT STARTED

As active members on social media, we all came across offensive language, especially in the comments section of a post or a video. Many of these hateful comments have a negative effect on the psychology of the users and there are many reports of victims.

An indicative example of a social media platform whose members are rather offensive, is TikTok. There have been numerous reports of victims who have been negatively affected from hateful comments, and this results to lack of confidence; they are easily manipulated and in many cases, this may result to the victims' psychological issues or even suicide.

28/08/2022, Athens

Considering that a large number of victims are women, we decided to actively contribute to this matter and improve in our way, the experience of the platform's users.

OUR TEAM

Our team is specialized in data analysis and representation with User Interfaces of the actual results in order to determine the corresponding solutions.

MEMBERS

Georgia Arkoumani - Data Engineer & UI Specialist
Myrto Poulou - Statistician & Data Analyst
Anastasia Koutsodimitropoulou - Business & Data Analyst
Eftychia Zaragka - Data Engineer & DB Specialist

LEGAL STRUCTURE

QC Greece is a start-up IT company incorporated in Athens, Greece.

INTELLECTUAL PROPERTY RIGHTS

QC Greece is a trademarked name, located in the city of Athens, and we have filed for protection of our proprietary processes and other intellectual property, such as our logos for both company and product.

MISSION STATEMENT AND TARGETS

OUR MISSION

Our mission is to detect and eliminate offensive language from social media and various platforms through the internet. Also, we provide our services to our clients (smaller or

bigger companies) in order to find effective solutions in social media platforms, such as banning users whose comments are rather offensive to other community members.

OUR TARGET AND CLIENTS

Our company targets to sell the '**hateless**' app to B2B (Business-to-Business) buyers and IT companies.

OUR STRATEGY

QC Greece aims to check a significant number of comments in different platforms through the internet and investigate whether a comment is offensive or not. Thus, the company created an application using machine learning and implementing sentiment analysis in order to detect the aforementioned problem. To begin with, after a detailed research that our company conducted for our clients, we created a dataset with comments from various platforms (e.g. YouTube, Reddit, TikTok, Twitter, hatebase etc) to test the application and start the analysis and the corresponding results were evaluated. Our goal is to maintain the application and add new features to expand the products' functionality by performing monthly deployments.

FUTURE OF OUR COMPANY

In response to industry's climate, QC Greece will offer consulting services in the future by helping businesses understand if their products are engaging or provoke a positive sentiment to the user. Moreover, '**hateless**' application will be extended with new features and addons.

PRODUCT DESCRIPTION

hateless: HOW IT WORKS

Our company implements machine learning and neural network algorithms in order to analyze the corresponding comments and calculate scores that determine if a comment is

offensive or not. QC Greece also implements a sentiment analysis to determine if a comment is positive, negative, or neutral.

Regarding our product, the user logs in the application, writes the specific comment that he/she wants to be analyzed and then he/she gets as an output a list of scores (positive, negative, or neutral), sentence token scores and diagrams that demonstrate how pleasant or subjective the comment is.

PRODUCT SERVICES

Our product provides a sentiment analysis and demonstrates how positive, negative or neutral a comment is.

TECHNICAL PROCEDURES

As per the technical procedures that the company followed in order to design and implement the application, a detailed representation is shown below in the hateless application section.

TECHNICAL REQUIREMENTS

Our company's product 'hateless' is a software as a service (SaaS) platform that makes the application available to end users over the internet. 'hateless' application is easily accessible, easy to use and available to all users and companies. It consists of certain packages depending on the client's necessities. The only requirement for the user is to login after choosing the preferable plan.

PRODUCT LIFECYCLE

All services are ready to be offered to clients, pending approval of contracts.

MARKETING AND FINANCIAL

INITIAL CAPITAL

The initial budget for building our start-up company was gathered from a competition for start-up companies and innovative ideas.

MARKETING STRATEGY AND EXPENSES

As per marketing strategy, in order to grow the company, we will establish a company website that will contain engaging multimedia content about our services. As the business grows, we will advertise in publications that reach our target industries and partners and create social media accounts. As profits increase, QC Greece will look to add an employee to assist with account management and marketing coordination. This individual will also provide company social media and online marketing support in order to obtain a wider target group.

As per company expenses, costs for marketing campaigns in social media will be added and advertisements, as well as additional paid features in order to construct our website and extend the features of our application.

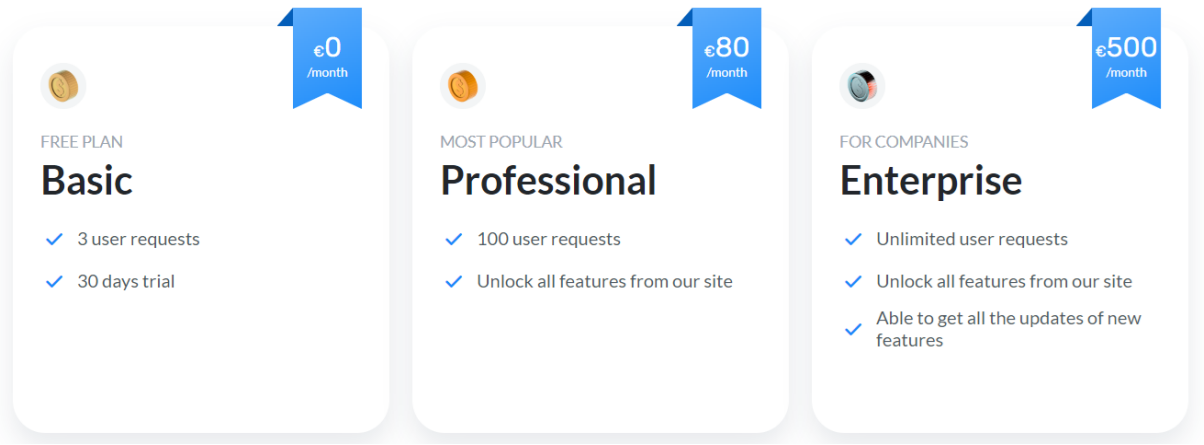
COMMUNICATE WITH CUSTOMERS

Our company will communicate with its customers by meeting managers within targeted companies, using our social media accounts and providing contact information on the company website.

PACKAGES AND COST PLANS

The available packages and cost plans of our company are represented below. We include three subscriptions and packages for companies but we will manage to add more for single users. Generally, our billing policy is pay-as-you-go; payments depending on the use.

28/08/2022, Athens



REGULATIONS AND RESPONSIBILITY

OUR RESPONSIBILITY

For Us it is very important to adapt and tune in with the needs and expectations that the current society anticipates. We want to promote modern values & ethics based on our main goal.

Main Goal:
"Harmonize human social interactions/ relationships and ELIMINATE social offensive language".

Unfortunately, in a variety of platforms, (as mentioned TikTok) harmful speech acts are displayed aiming to hurt and provoke some kind of 'damage' to other groups/ sub-groups or individuals. Via the above, only unfavorable outcomes arise such as excessive bullying acts, attempts of suicide and toxic environments. Our aim is to help ease these issues and to promote a reality where there is freedom of expression and human contact with a purpose of healthy dialogue. The future generations should thrive in a world where toxicity in any form does not dominate and human communications exist to bring people together.

28/08/2022, Athens

Our values:

Peace

Joy

Equity

Respect

From our product, we aim to assist companies, and subsequently the users of social media, to not be afraid and to be freed from the massive online hate speech.

GREEN INITIATIVE

For Us it is important to promote a “GREEN” character with an environmental commitment to zero waste policy (such as no paper use). We believe in the “Go Green Initiative” organization, where all people and especially youth, should learn sustainable ways to conserve our planet by the tools, training and ongoing support they need to create a “culture of conservation” and natural resource stewardship within their community.

Our effort is to adopt the United Nations 2030 agenda, as expressed by the 17 Sustainable Development Goals for 2030 (Sustainable Development Goals). The aim is to actively contribute to their achievement, through the promotion of the well-being and safety of the population, the protection of the environment and the fight against poverty.

28/08/2022, Athens

RESEARCH AND DEVELOPMENT

MARKET RESEARCH

After extensive research on the market, larger consulting firms work with international corporations while smaller consulting firms work with both large corporations and smaller organizations. We are planning to make an improvement in the industry, provide solutions and capitalize on opportunities that are geographically close as we start and grow our business.

DEVELOPMENT

Our company is planning to perform the following actions for development:

- Create a technology solution for businesses that will help them understand the impact that their product has on the market; how pleasant or not the users find it.
- Determine the need for additional consulting services within our market related to tying improved processes to opportunities for increased sales and promotion to potential customers.
- Find trends in software solutions that may provide potentially competitive automated services in order to ensure QC Greece continues to carefully carve its niche in the marketplace.

28/08/2022, Athens

Methodology

The key idea was to combine a variety of datasets from different platforms that contain hate speech and offensive language and categorize their comments depending on if they are offensive or not. Then, a detailed exploratory and text analysis was performed in order to make some basic assumptions for the data.

A variety of methods have been also explored for the hate speech detection task, including traditional classifiers, deep learning-based classifiers, or the combination of both approaches. The result of the aforementioned methods was the best model that describes the problem and predicts whether a comment is offensive or not. Moreover, a sentiment analysis was also included in order to study further the effect that these comments have on people.

After taking into consideration all the above procedures, an application was designed and implemented that contributes to the aforementioned text analysis and demonstrates the actual results to the user.

The analysis particularly focuses on measures of practical performance, including detection accuracy, computational efficiency and capability in using pre-trained models. In doing so the aim is to provide guidance as to the use of hate speech detection in practice, quantify the state-of-the-art, and identify future research directions.

Data Preparation

As per data preparation, 6 *different* datasets were used from official sources in order to obtain real-world data and train the models effectively. In particular, the datasets used are: Berkeley Dataset (consists of 39,565 comments annotated by 7,912 annotators, for 135,556 combined rows and 131 columns. The primary outcome variable is the "hate speech score" but the 10 constituent labels (sentiment, (dis)respect, insult, humiliation, inferior status, violence, dehumanization, genocide, attack/defense, hate speech benchmark) can also be treated as outcomes. Includes 8 target identity groups (race/ethnicity, religion, national origin/citizenship, gender, sexual orientation, age, disability, political ideology) and 42 identity subgroups), ethos Dataset (consists of 998 comments (rows) and 2 columns in the dataset alongside with a label about hate speech *presence* or *absence*. 565 of them do not contain hate speech, while the rest of them, 433, contain), ICWSM18 Dataset (consists of 3,222 comments (rows) and 10 columns: id, title, type, message, class & sub 1 to 5 while it contains approximately 75% of hate speech), All-in-One Jigsaw Dataset (consists of 2,223,065 rows and 11 columns with an id, a comment, the main target, the other toxicity subtypes as well as identity attributes), Davidson Dataset (consists of 24,783 rows and 6 columns with hate and offensive speech counts and classes along with the corresponding comment/tweet) & CONAN Dataset (consists of 5,003 rows and 5 columns with counter narrative pairs covering the multiple hate targets, including DISABLED, JEWS, MIGRANTS etc. Each pair is provided along with its loop information (VERSION), and its target (TARGET). The dataset contains only hate speech.).

The aforementioned datasets contain information from different platforms e.g. YouTube, Reddit, TikTok, Twitter, hatebase etc. In order to proceed with the analysis, all the unnecessary columns were dropped, while several conditions were applied to a part of the available data, so as to create the required information. To be more specific, custom functions were created that classify or merge multiple categories into two; hate or no hate. Therefore, the final processed datasets contained only 2 columns (*comment* and *category*). Thus, the above mentioned 6 datasets were merged into their common columns and the final dataset contained 2,265,516 rows with comments and 2 columns.

Moreover, as a further action, a cleaning procedure with the implementation of a custom function was applied that removed from the dataset: mentions and e-mails, urls, numerics,

punctuations, useless spaces, rt words and other unnecessary characters while the comments were transformed into lower case. A second function was also implemented that removed repeated characters in the dataset.

The final dataset consists of **2,264,714** rows/comments and **2** columns (*comment* and *category*). As a next step, the dataset was prepared for further exploratory data analysis that will be described in detail in the next section.

EDA Text Analysis

Text Analysis (TA) aims to extract machine-readable information from unstructured text in order to manage content. It is about parsing texts in order to extract machine-readable facts from them, focusing on creating structured data out of free text content. The first phase of the above concerns the Exploratory Data Analysis, in which an initial acquaintance is performed with the data, in order to drive intuition and begin to formulate testable hypotheses. Relevant descriptive statistics & visualizations will follow.

The available textual data on online social media are highly unstructured and very often misspelled, complicating the process of detecting offensive content or even evaluating the offensiveness of texts. At this section of the analysis, a research was initiated by trying to understand the data, perform additional cleaning processes and at last, define the form of the dataset on which the future models would rely on.

After the finalization of the dataset structure, used for the underlying product, it was important to detect possible missing data or even duplicated information that would not benefit the further research. A trivial amount of missing values were detected and excluded (concerning the “Non-hateful” category of comments, which were most likely, appeared after the appliance of the cleaning procedures). The duplicated information was larger in terms of volume but it constituted only 3% of the overall one, so it was decided to be removed, as well.

It is worth mentioning that the dataset was considered imbalanced as there was a significant/extreme disproportion among the available comments’ category.

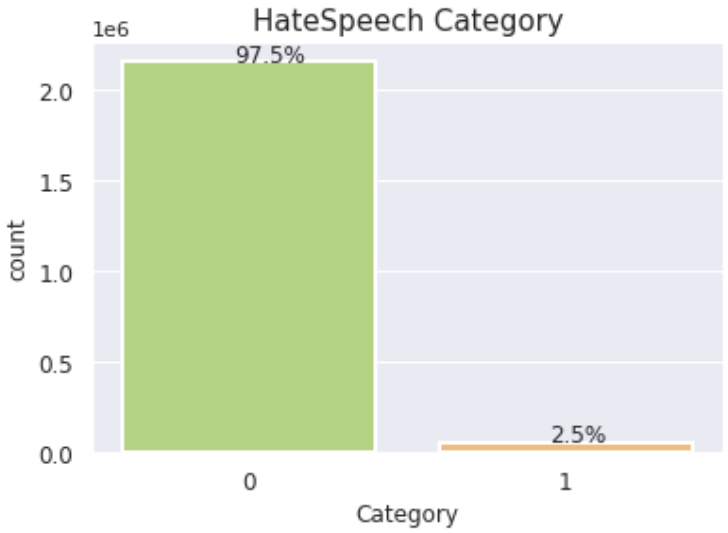


Figure 1: Hate speech Category distribution

A dataset is imbalanced if the classification categories are not approximately equally represented. Imbalance in data, especially in class, always contributes to worse prediction in the modeling process. Further data manipulations for the above, will be explained in the next sections.

Most mentioned words/ Stop words

Moving forward, displaying the most mentioned words in each category, was essential in order to determine any additional corrective acts.

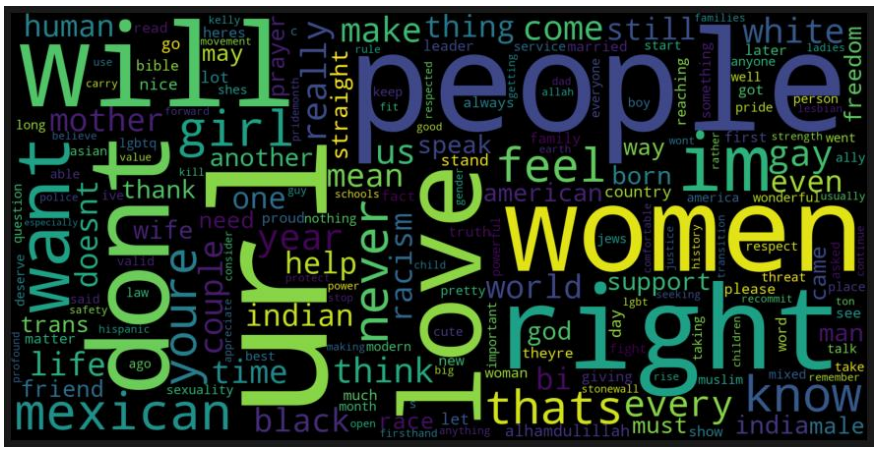


Figure 2: Most mentioned words for “Non-Hateful” comments

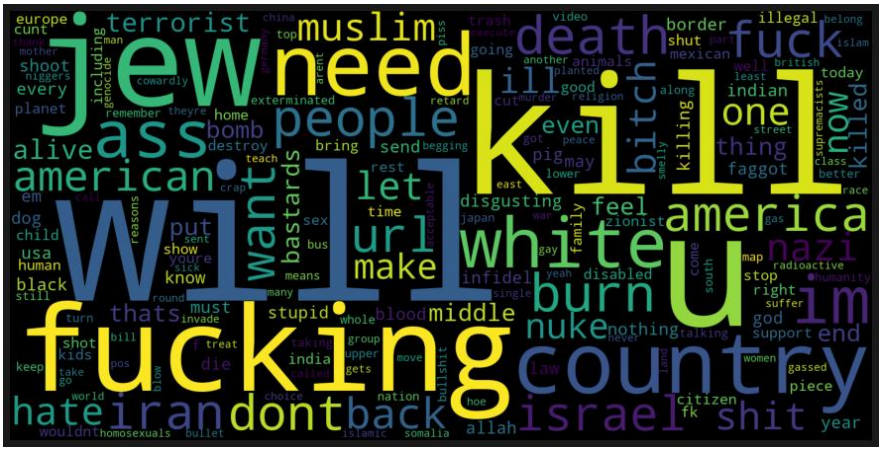


Figure 3: Most mentioned words for “Hateful” comments

Existence of “Stop” words was detected, namely words like articles, prepositions, pronouns, conjunctions, etc. Examples of a few stop words in English are “the”, “a”, “an”, “so”, “what”. The aim was to remove them in order to exclude low-level information and focus on the important information. At the same time, it was needed to accomplish the use of words in their base or dictionary form, so the practice of lemmatization was also applied. Namely, the goal was a word's morphology without inflectional endings.

At this point, a re-examination of missing & duplicated values was performed as it was anticipated that comments with unimportant content (containing only stop words) or small differences with each other (comments that would differ only in existence of stop words or different morphology) would previously exist. The corresponding actions were applied and the updated status was the following:

Figure 4: Most mentioned words for “Non-Hateful” comments

Figure 5: Most mentioned words for “Hateful” comments

A first conclusion drawn, had to do with the fact that words widely accepted as either good or bad seemed to have been correctly characterized for the most part. At the same time, words appeared in both categories implying that some of them have double meaning, while there were, also, words with no semantics (i.e url).

Polarity Score

Our goal is to create an app/product that would analyze the body of given texts, understanding the opinion expressed and quantifying it to a positive or negative value. We aim to create an app that will generate output of 3 different sections: ***Sentiment metrics***, ***Sentiment token metrics*** & ***Sentiment metrics visualization***. Therefore, we proceeded our EDA analysis with the relevant calculations of polarity scores in each comment. As it was expected based on the imbalance of the dataset, the available comments in the analysis seemed to display a neutral sign/meaning on average.

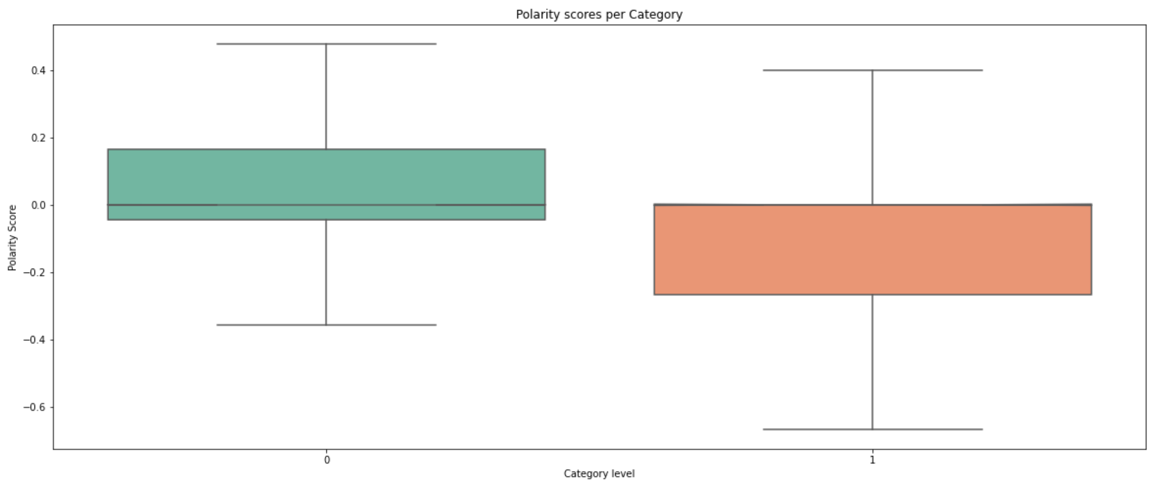


Figure 6: Polarity score contribution among the 2 comments' categories

The 2 categories seemed to share equal average scores with dissimilar distributions. The "Non-Hateful" category included comments that hold a larger variety of scores, higher than the "Hateful" ones. At the same time, the last mentioned, indicated the existence of comments with high negative meaning , namely data with a negative skewed distribution.

Text Readability

The fact that the majority of the texts indicated a neutral status, triggered a search deeper regarding the quality of the data so as to detect other possible issues. By combining the polarity knowledge with the category of scores, what came across was comments with the following form, through which a further dive into the data structure was required.

Comment	Polarity Score
"disagree article old fashion myth ex&le person invite fancy dinner teach fancy dinner etiquette person google table etiquette good idea approach dinner edmonton community require certain portion dedicate affordable housing way live realize similar different design public school work kid similar experience friendship surpass call elite line long health care public education inclusive opioid epidemic fail level prescription drug company market highly addictive drug mds prescribe mass patient take drug dilligence btw get test"	*0.050054*
"retard ed retard ed"	*-0.9*

Table 1: Comments with high complexity structure

Readability is an important factor in producing & ensuring quality of texts' content. It can ease someone to understand the level of difficulty that the text data contain in order to proceed to needed changes for modeling purposes.
The use of 4 different readability scores was made to evaluate how "clear" the comments were.

Flesch Reading Ease : is a score that quantifies the readability of a text. The maximum value that can reach is 121.22, where the lower values do not reach a baseline.

Formula:

$$206.835 - 1.015\left(\frac{\text{total words}}{\text{total sentences}}\right) - 84.6\left(\frac{\text{total syllables}}{\text{total words}}\right)$$

Score	Difficulty
90-100	Very Easy
80-89	Easy
70-79	Fairly Easy
60-69	Standard
50-59	Fairly Difficult
30-49	Difficult
0-29	Very Confusin

Table 2: Classification of scoring

Dale-Chall: is a score different from other tests, since it uses a lookup table of the most commonly used 3,000 English words. Thus, it returns the grade level using the New Dale-Chall Formula.

Formula:

$$0.1579\left(\frac{\text{difficult words}}{\text{words}} \times 100\right) + 0.0496\left(\frac{\text{words}}{\text{sentences}}\right)$$

Score	Understood by
4.9 or lower	average 4th-grade student or lower
5.0–5.9	average 5th or 6th-grade student
6.0–6.9	average 7th or 8th-grade student
7.0–7.9	average 9th or 10th-grade student
8.0–8.9	average 11th or 12th-grade student
9.0–9.9	average 13th to 15th-grade (college) student

Table 3: Classification of scoring

Gunning FOG Formula: is a score that returns the FOG index of the given text. This is a grade formula in that a score of 9.3 means that a ninth grader would be able to read the document.

Formula:

$$0.4\left[\left(\frac{\text{words}}{\text{sentences}}\right)+100\left(\frac{\text{complex words}}{\text{words}}\right)\right]$$

***Text Standard score**:* uses various readability checking formulas, combines the result and returns the grade of education required to understand a particular document completely.

The comments of the dataset, on average, was calculated to have the following scores:

Score category	Mean Value
Dale Chall Score of comments	*12.15671603*
Flesch Reading Score of comments	*36.99411011*
Gunning Fog Index of comments	*15.96875514*
Text Standard of comments	*12th and 13th grade*

Table 4: Readability average scores

Therefore, the content of the text was considered as difficult and was decided to continue with additional cleaning procedures and alterations.

Corrective Acts

Two corrective acts were applied:

1. appliance of additional cleaning, &
2. removal of complex comments.

Regarding the first one, functions were created and applied that would do the following: exclude words that represent one syllable, same multiple characters in a word, repeated words in a row, emoji's or digits that due to the complex data structure may not be removed, and exclude type of word with no sentamatics (i.e "url"). After re-applying the readability scores, comments with high levels of difficulty were dropped (after examining what kind of information was going to be excluded) using as baselines both scores & comments' type of category. For high, readability scores, close to the upper bound of the relevant average one, exclusions were only performed for the majority class in order not to lose samples from the minority one, taking also into consideration, the neutral status of this category comments.

Due to the particular complex data infrastructure and the re-application of cleaning methods, missing values and duplicate text content were resurfaced, which was to be expected since comments with only single letters existed alongside with comments that differed only to repeated words/ letters etc.

The actual final dataset consists of **1,381,262** rows/comments and **2** columns (*lemmatized* and *category*). As a next step, the dataset was used for the final text descriptives and the modeling process, fully explained in the corresponding chapters.

Berkeley Dataset Text Analysis

General matters

In 2017, UC Berkeley's D-Lab started a project aiming to measure 'Hate Speech' in social media. Their main goal was through data science to detect the various forms of hate speech and map its alterations. Via this research, they manage to construct variables that depict not only hate speech but also, insult, respect, humiliation and even more kinds of feelings. However, they focused their research around the constructed variable which was created to score the level of hate speech. They demonstrate their work on data collected from different sources like YouTube, Twitter, and Reddit, for which workers were asked to label them based on their content.

Berkeley Dataset was the 2nd larger sub-dataset with various information available, enabling the expansion of the EDA analysis with more descriptives. There were 131 variables with 135,556 entries. The majority of the variables were in boolean form while there were 22 numerical & 6 nominal.
Missing values were detected in the dataset but corresponding to variables like annotators' education, income, ideology & age. Indicating that personal information may be excluded, especially financial data. Duplicated rows did not exist, however duplicated text/ hateful comments appeared. At the same time, negative values existed only in variables where specialists gave scores to sum up their conclusions, such as the hate speech score.

An annotator performs data annotation which is the process of labeling data in various formats such as video, images, or text so that machines can understand it. For supervised machine learning, labeled datasets are crucial because ML models need to understand input patterns to process them and produce accurate results. For this specific project this information was used in order to proceed to the modeling phase.

Target & outcome variables

In the dataset the target variable corresponded to "hate speech score" but there were, also, other related variables which could be treated as outcomes, such as insult, humiliation, violence etc. Therefore, it was found that the contained comments displayed high emotional depth, insult, moderate humiliation expressions and low levels of potential

violence or other extreme ‘bad’ intentions. Hateful comments did not dominate, giving an average idea that future comments do not tend to be hateful. However, the dataset was considered imbalanced based on the distribution of the non & hateful comments.

Data considered as outliers were detected but in variables that would not be examined at this point (i.e. annotator_outfitms).

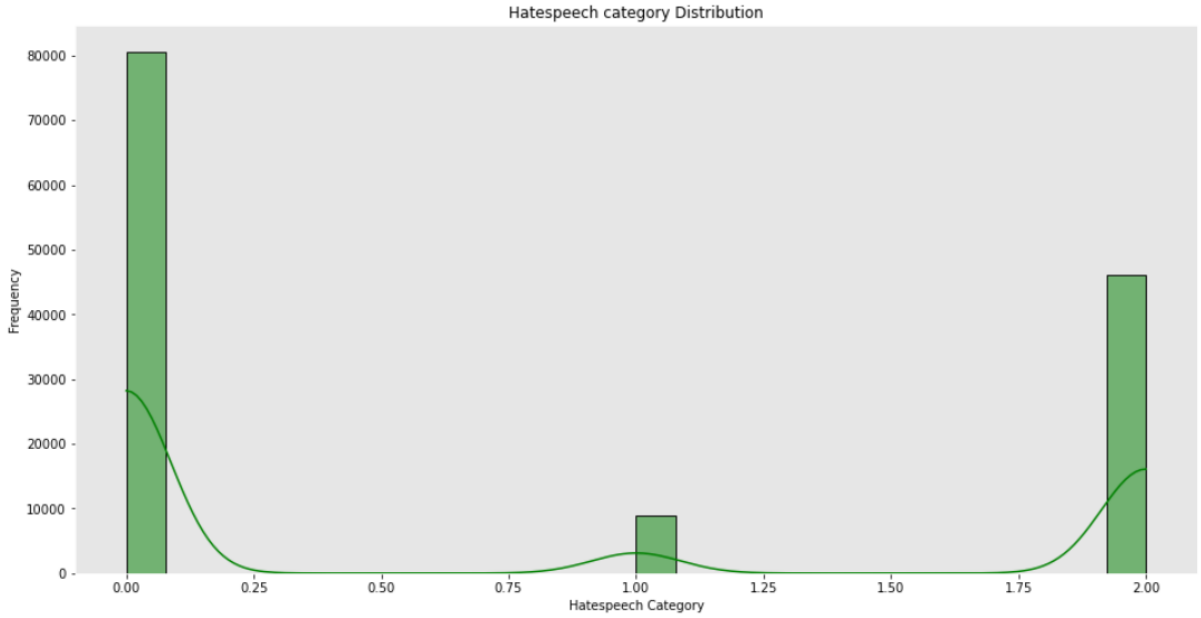


Figure 7: Hate speech distribution

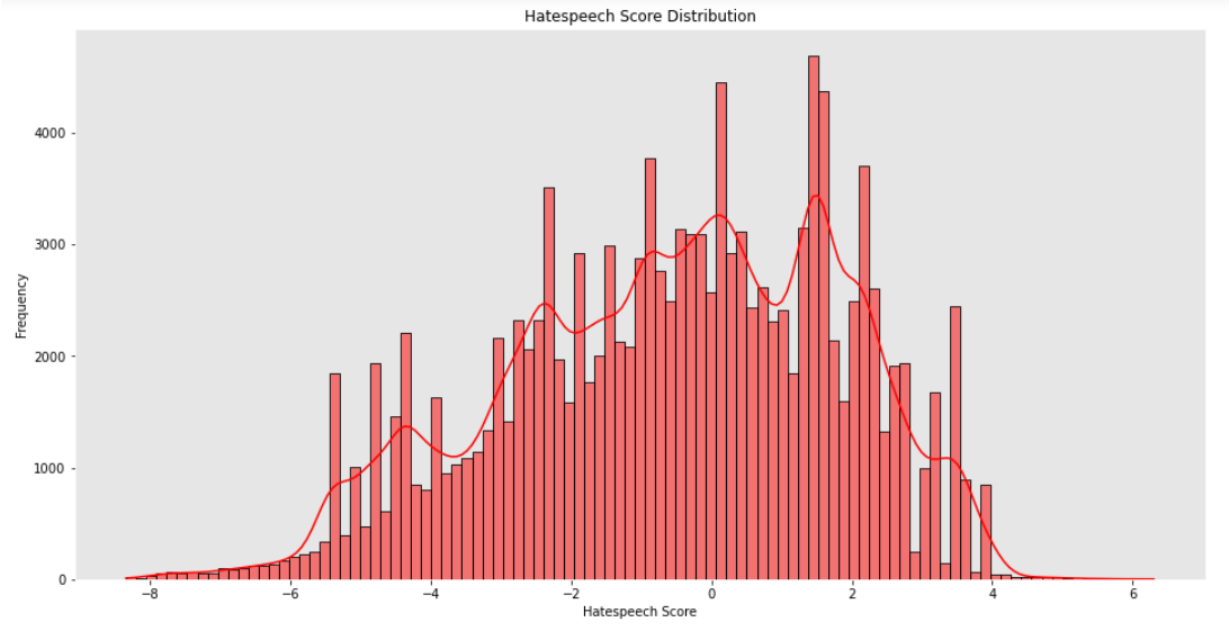


Figure 8: Hate speech score distribution

From the 1st graph the imbalance of the dataset is visible, while for the 2nd one a possible normal distribution of the scores was implied with a left skewness to be detected. Meaning that there were comments with high hateful meaning but on average neutral expressions prevailed.

Annotators & target groups/ sub-groups

Besides the above, from the dataset form, further descriptives were available, corresponding to annotators & the target groups (sub-groups). Over 70% percent of the annotators were college graduates, with few holding a phd degree or even just a highschool one. However, the variety of educational background didn't indicate any difference on how they characterized the data.

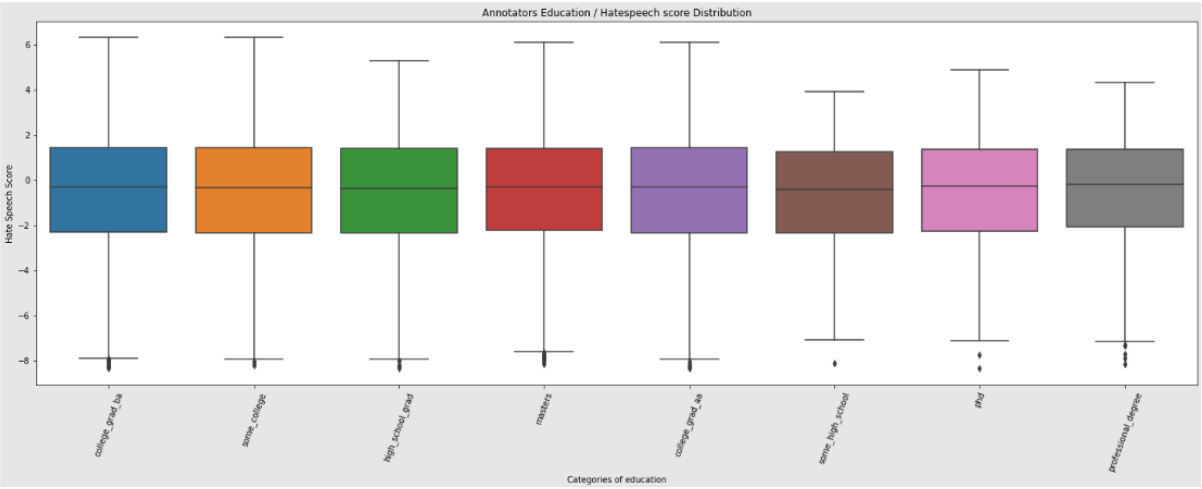


Figure 9: Hate speech score distribution based on education's categories

Their age was estimated to be around 37 on average, but there were, also, annotators in other age groups. The age difference among them didn't affect their comprehension of hate speech, nonetheless annotators of high age seem to grade the levels of hate speech even higher.

Approximately, female annotators were employed from Berkeley Institution, while annotators' incomes could be considered quite high depicting people of wealth or no economical issues. Of course, there were annotators with lower earnings, something that did not alter or affect, in any way, their characterization of hateful comments.

As far as it concerns their political ideologies, liberalism dominated while almost 20% of the annotators showed no political affiliation. Different political beliefs didn't seem to matter.

Severity in testing is a crucial issue. It is defined as the extent to which a particular defect can create an impact on the software. Severity is a parameter to denote the implication and the impact of the defect on the functionality of the software. In this case, it refers to bias that annotators could show. In most cases bias couldn't be fully detected to their work, despite that there were cases where it existed.

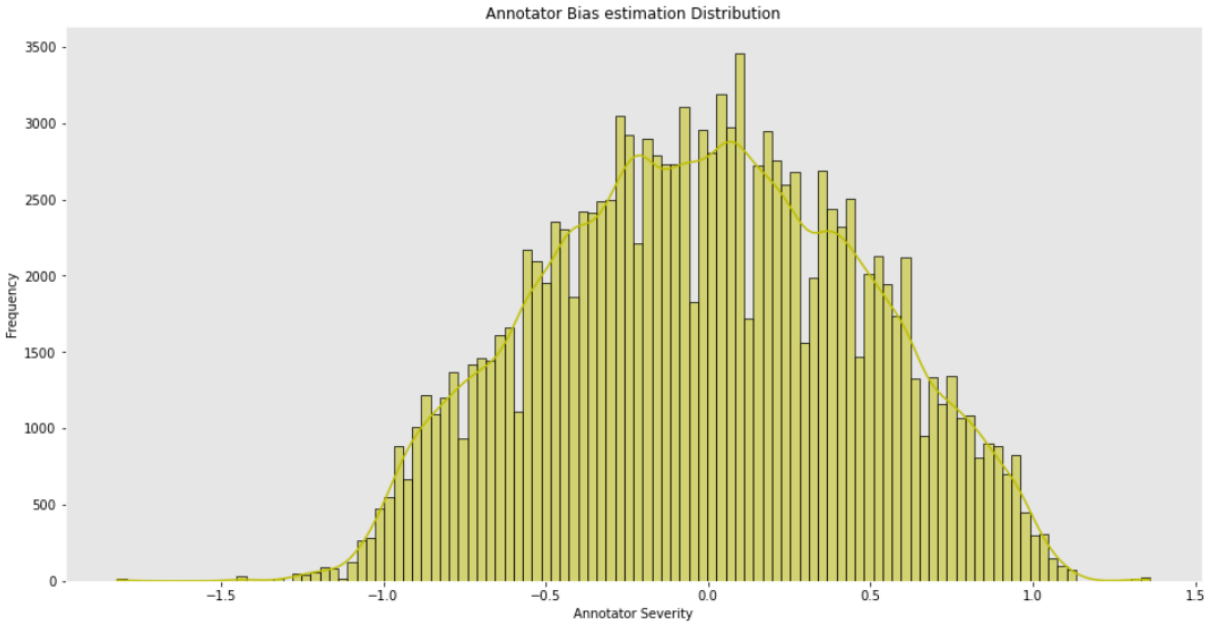


Figure 10: Severity distribution

Regarding the groups that the research referred to, no extreme deviations existed among them in terms of hate speech, however, for race, religion & origin there were indications that could be connected to hateful comments, especially regarding specific sub-groups (the relevant information is detected to the notebook's content).

Dataset Overview

General matters

The final instance of the dataset, regarding comments' distribution and most frequent words in each category ("Non-Hateful"=0 & "Hateful"=1) was the following:

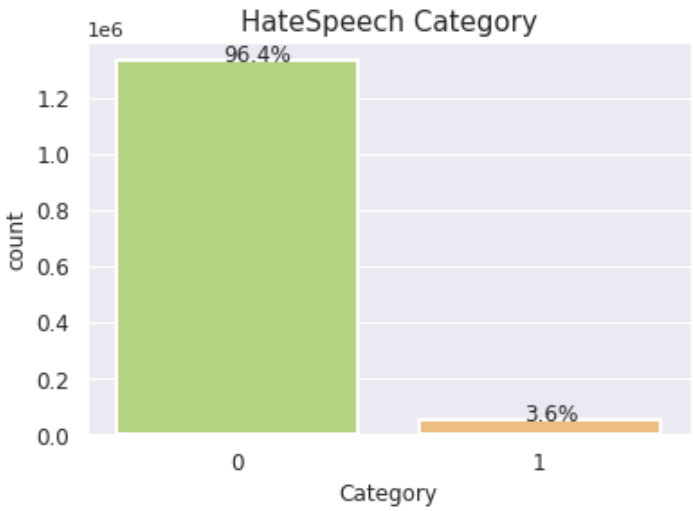


Figure 11: Hate speech Category distribution

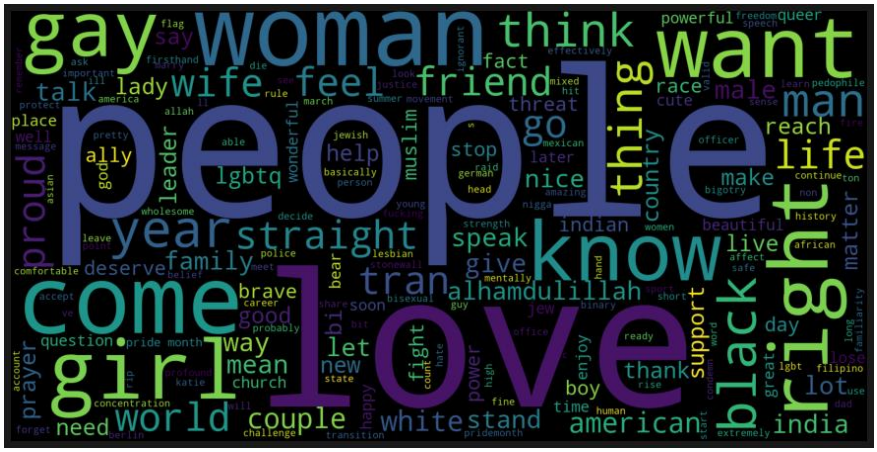


Figure 12: Most mentioned words for "Non-Hateful" comments

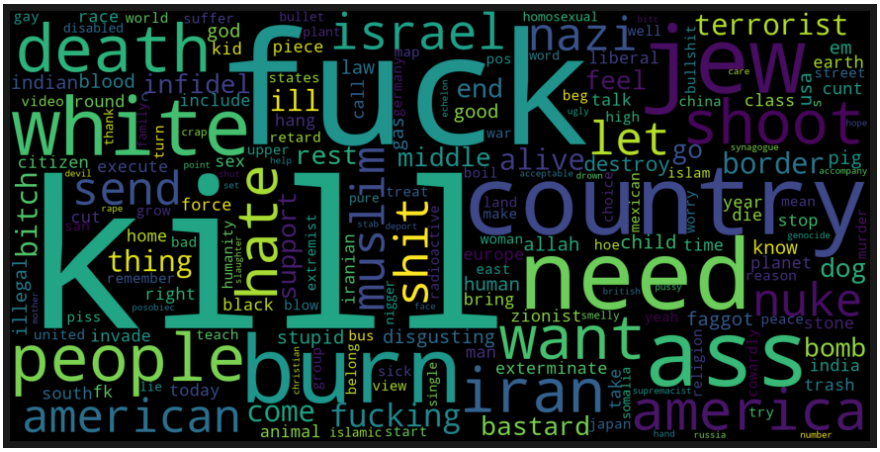


Figure 13: Most mentioned words for “Hateful” comments

A minor improvement to the imbalance of the dataset was held, after the previous corrective acts while polarity & readability scores were equally ameliorated.

Text Descriptives

An additional parsing of texts’ content was applied, in order to obtain a more comprehensive picture of the tokens to be used in the modeling process.

On average the content of the comments consists of 20 - 25 syllabus, 5 to 6 repeated words, and a length of lexicons around 6. The overall English alphabet seems to be used while apparently, difficult words make up 30% of the total words.

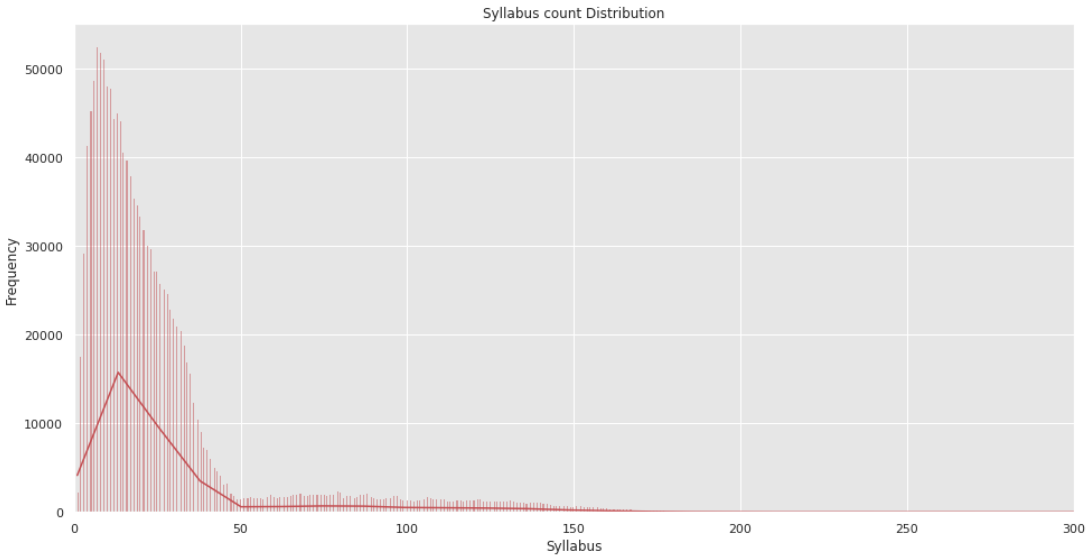


Figure 14: Syllabus Distribution

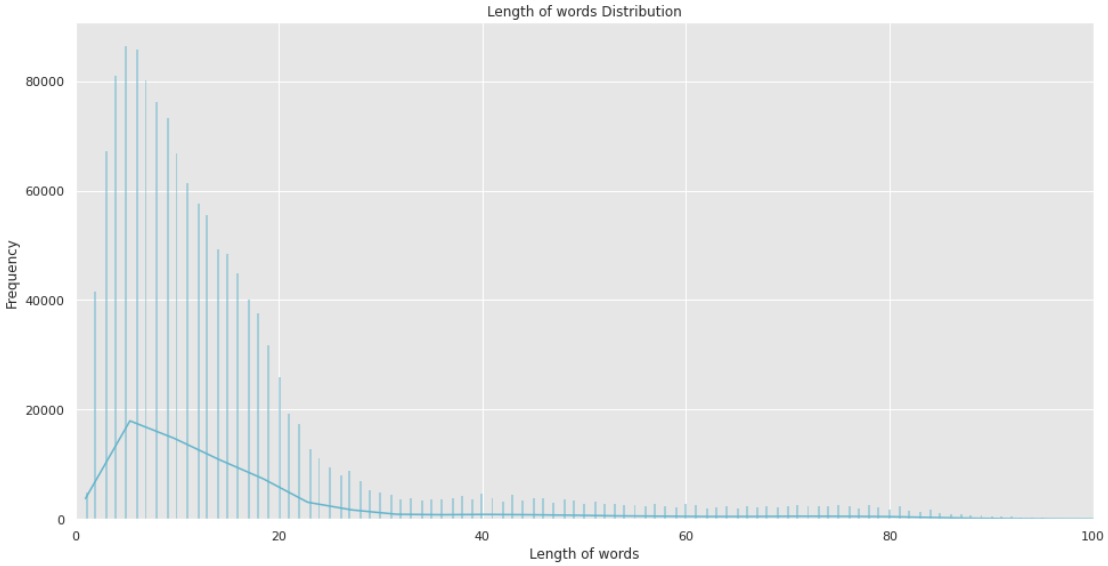


Figure 15: Word length Distribution

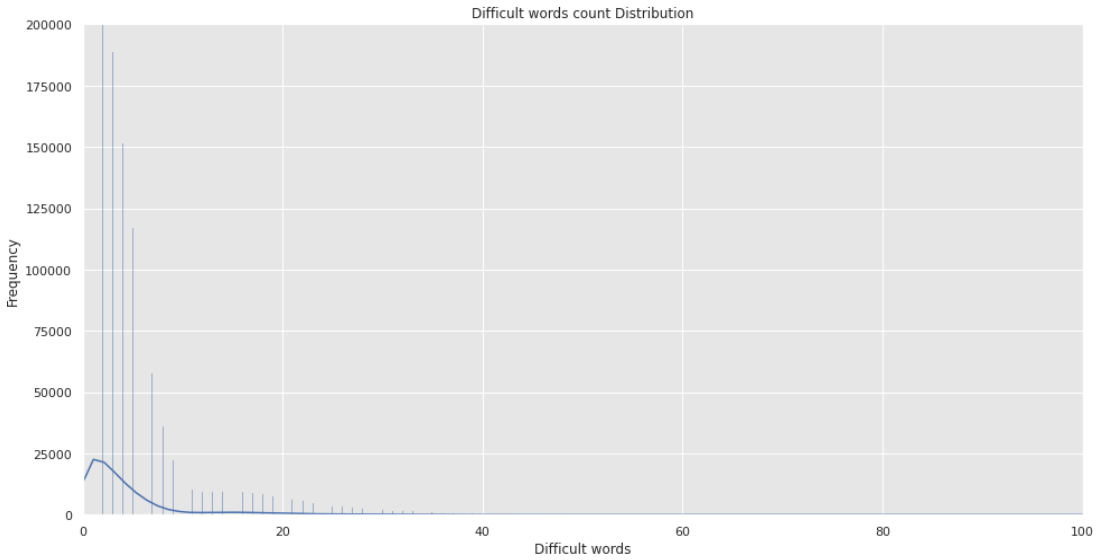


Figure 16: Difficult words Distribution

N-gram approach is considered as an improved approach as it brings words' close to context information to detect offensive contents. They represent subsequences of N continuous words in texts. Bi-gram are the most popular N-grams used in text mining. However, Ngram suffers from difficulty in exploring related words separated by long-distances in texts. Simply increasing N can alleviate the problem but will slow down system processing speed and bring in more false positives. Here, Bi-gram are displayed (in the notebook "2.1 EDA Text Analysis", more N-grams were investigated), for each category, since patterns were investigated.

For the comments that represented Hateness, most used pairs of words had to do with swearing or denial.

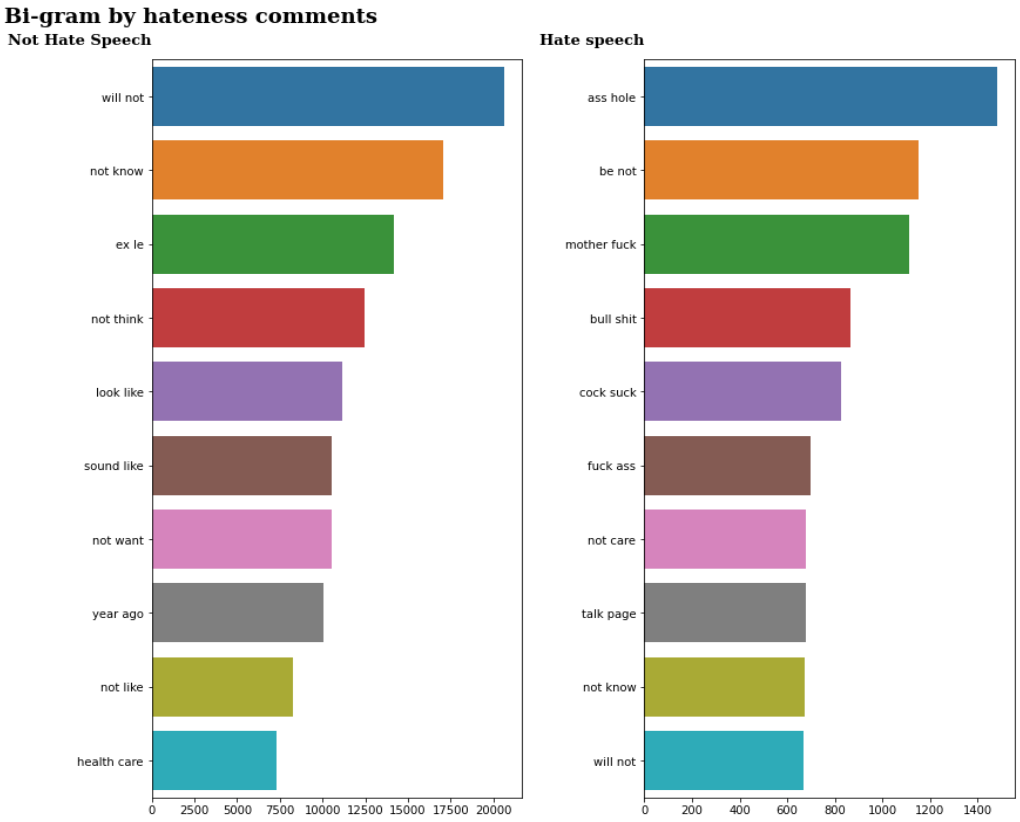


Figure 17: Bi-Gramms

By continuing the analysis around other types of word's combinations, misspellings, possible human errors while writing , were detected. This issue was also detected from the earlier stages of the analysis, but due to algorithms' computational cost, their implementation to the whole dataset wasn't applied, for that time period.

Goal:

#	category	lemmatized	speller
0	0	yes sort remind **eld** lady play movie titanic te...	yes sort remind **old** lady play movie titanic te...
2	0	amazing dad not forget girl **crushs** girl mom as...	amazing dad not forget girl **crush** girl mon ask...

Table 5: Autocorrection's appliance

At this point the data preparation was concluded and the modeling planning had begun.

We approached several techniques through both Machine Learning & AI processes, which will be explained in the next chapters.

Overall the used dataset was considered unstructured and quite complex with the need of several corrective acts. Words' misspellings and errors were detected while in each comment specific words were repeated with a moderate length. The readability context was considered high with the amount of difficult words to be around 30% of the whole one. Autocorrection algorithm is to be examined more and applied for the application matters.

Regarding the modeling phase and the dataset structure , an issue that created several complications with the results and the models' management was the detected imbalance. Therefore, a variety of data manipulations were applied and tested in order to examine the outcomes and determine the future methods to use. Further information will be described in detail in the next sections.

NLP Architectures & Systems

A rather important step in the process was the creation of different types of models and the evaluation of the results.

The models created (and later evaluated) are the following:

- Neural Network Models:
  - Recurrent Neural Networks Model (RNN),
  - Convolutional Neural Network Model (CNN)
  - Bidirectional Encoder Representations from Transformers Model (BERT)
- Logistic Regression Model
- Linear SVC Model
- K-Neighbors Classifier
- Random Forest
- Naive Bayes Multinomial Classifier
- Bag of Words (BoW) and TF-IDF Vectorizers

Also, a sentiment analysis was performed using VADER.

In the following chapters, more details are presented regarding each model, along with the data architecture required in order to appropriately fit the models.

Data Manipulation for Neural Network (RNN-CNN) Models

Before creating the models, the first step was to load the data. Two alternate ways were investigated for the data loading. The first option was to load the data by reading the files locally. The second option was to read the files from Google Drive.
Regarding the second option, a new folder was created, named “MLCA_Datasets”. Over there, the produced file “more_cleaned_df.zip” was stored and uzipped (file produced in a previous step, for more details please check the chapter Data Preparation). This file was later used to load the data in a dataframe.

After successfully loading the data in a dataframe, a random sample of 500,000 records was selected. The augmentation of the data was made, by making sure that the distribution of the classes remained among the two categories with the same instance as it was. At the

same time, the limitation of resources capacity made it necessary to reduce the shape of data. By performing a check in the column category, it seems that the data are imbalanced:

Category	# Data
0	*481,776*
1	*18,224*

Table 6: Dataset Shape (following random sample selection)

As a next step, it was important to set-up the Keras tokenizer, in order to vectorize a text corpus, by turning each text into a sequence of integers. In general tokenization refers to splitting up a larger body of text into smaller lines. The tokens were generated by counting the frequency, and then the text was turned into a sequence of numbers. Lastly, a dictionary named *word_index* was created in order to appropriately map each word to a number.

Additionally, it was rather important to identify the max sequence of all the comments, in order to later proceed with their padding. To achieve this, the distribution of the column 'lemmatized' was calculated and the cumulative percent of text was identified.

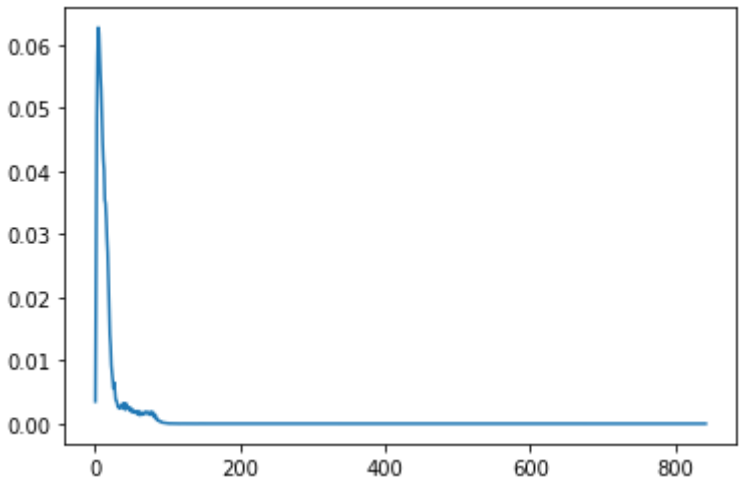


Figure 18: Column 'lemmatized' distribution

By examining the distribution of comments' length, it resulted that the 90 words was the maximum figure. Therefore, it was decided to set all sequences as 90 words long[1] by using the Keras pad sequences tool. The specific tool cuts off sequences that are too long and adds zeros to sequences that are too short. In the end, there were 500,000 and 90 characters sequences.

[1] Note: all text sequences fed into the NN models had the same length.

A rather important step in the process was to perform a shuffle in the data. The reason why this was important is because the shuffling minimizes the training loss and prevents any bias during the training. Also, it prevents the model from learning the order of the training. After successfully performing data shuffling, it was decided to perform training in 400,000 samples, which reflects 80% of the dataset. Then, the dataset was split into train, test and validation.

Something worth mentioning is that any data manipulations have been performed only in the training data for learning techniques; the validation data were treated as it was.

Taking into consideration that the data are imbalanced (there are more records with category 0, rather than category 1), it was decided that a possible solution would be to perform Synthetic Minority Oversampling Technique (SMOTE). SMOTE is an oversampling technique where synthetic samples are generated for the minority class. This algorithm would help to overcome the overfitting problem posed by a random oversampling. To be more precise, it focuses on the feature space to create new instances with the help of interpolation between the positive instances that lie together. SMOTE was performed in order to overcome poor performance issues of neural techniques, related to the knowledge extraction of the minority class. Finally, the distribution of the dataset was determined to 50%-50% per category.

As a next step, it was decided to use pre-trained word embeddings, since training our own embeddings are prone to overfitting. GloVe Embeddings is one of them and for the purpose of the project it was the one used. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

In order to use GloVe Embeddings the folder glove.6B.zip was downloaded from the Stanford website. After the folder was successfully downloaded[2], a function named *glove_embeddings* was created, which loads the embeddings. It is important to note that in the vocabulary not all words might be in the GloVe Embedding. As a result, for the missing

[2] In case the files are read via Google Drive, inside the "Embeddings" folder, the files glove.6B.100d.txt and glove.6B.300d.txt are stored.

words it was decided to use random embeddings with the same mean and standard deviation as the GloVe Embeddings. For this matter, the function *create_embeddings_matrix* was created, This function receives as parameters the embeddings index, the Keras fitted tokenizer and the embeddings dimension and returns a matrix of shape which contains the globe embeddings. From this action the creation of the embedding matrix concluded that the embeddings average is equal to -0.0039 and the standard deviation is equal to 0.3817.

Lastly, it is necessary to mention that since there are 2 classes in 'category' it was needed for the y value for the train, test and validation to be presented as categorical with 2 classes. So, for instance if a comment was negative one would be in the negative class and the number zero in the non-negative class (positive/neutral).

After all the above took place, the next steps were to create the models; Recurrent Neural Networks Model (RNN) and Convolutional Neural Network Model (CNN).

Recurrent Neural Networks Model (RNN)

One of the models created was the Recurrent Neural Networks models - so called RNN model. The Recurrent Neural Networks are a class of neural networks which allow previous outputs to be used as inputs while having hidden states. A typical example of such neural network is the following:

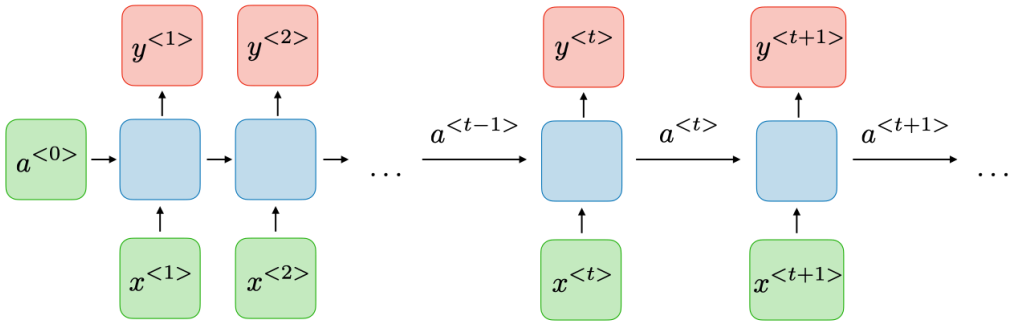


Figure 19: RNN (source: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks)

It was decided to build a RNN model due to the fact that they are mostly used in the fields of natural language processing and speech recognition. Also, one of their advantages is that their weights are shared across time.

The first step in the process was to create a sequential model named *RNN_model*. The sequential class represents a linear stack of layers. The sequential constructor takes an array of keras layers. Then, by using this model, the LSTM model[3] with the embeddings was created. The LSTM model is a special kind of Recurrent Neural Network which is capable of learning long term dependencies in data. This is achieved because the recurring model has a combination of four layers interacting with each other. In general it would be interesting to mention that the used layers in the model were the following:

- **SpatialDropout1D**: promotes independence between feature maps.
- **Bidirectional**: extension of LSTM model. Improves performance on sequence classification.
- **GlobalMaxPooling1D**: receives the max vector over the steps dimension.
- **Dense**: these refer to the hidden layers which were built using 'relu' and 'softmax' as activation rules.
- **Dropout**: randomly sets input units to zero with a frequency of rate at each step during training time, which helps prevent overfitting.

By printing the model summary, it appears that the total number of parameters is equal to 4,596,178. The calculation results by summing the parameters of every layer:
embedding + bidirectional + bidirectional_1 + dense + dense_1 == 4,500,000 + 85,248 + 10,368 + 528 + 24 == 4,596,178
Another interesting thing to mention is that the parameters of the *dense_1* layer are equal to 34, because they are calculated using the formula *output_size * (input_size + 1) == number_parameters* (2 * (16 + 1) == 34)

The next step in the process was to compile the model. During the compilation step, three key factors were used:

- **Adam Optimizer**: Adam is an optimization algorithm which is used instead of the classical stochastic gradient descent procedure to update network weights iterative based on training data.
- **Binary Crossentropy Loss Function:** Taking into consideration that the 'softmax' was used as activation rule in the output layer, the cross-entropy function was used. To be more precise, the binary cross-entropy function was used, since there are two classes in the dataset.

[3] LSTM stands for Long-Short Term Memory

- **Accuracy Metric:** Since the exercise comes down to a classification problem, the accuracy metric was used for reporting.

After all the above actions took place, the next step was to train the model. In order to achieve this, the keras *fit* function was used and a couple of parameters were specified. To be more precise the parameters were the following:

- Training data - *x_train* and *labeltrain* - commonly known as x and y.
- The number of epochs was set equal to 30. Epochs refer to iterations over the entire dataset to train for.
- The batch size was set equal to 256. Batch size refers to the number of samples per gradient update.
- Validation data - *x_val* and *labelval*
- Callbacks, and more precisely the *EarlyStopping* callback was used, with patience equal to two. It means that it checks the lowest validation loss and if that loss increases for two consequent epochs, the training stops.

Then, it was important to check the accuracy metric. It appears that the test accuracy of the model was equal to 0.877; and the binary cross-entropy was equal to 0.281.

In order to better visualize and understand the concept of the training and validation accuracy and loss, two graphs were created; “Training and Validation Accuracy” and “Training and Validation Loss”. In general, there is no relationship between these two metrics. Loss can be defined as the distance between the true values of the problem and the values predicted by the model. Greater the loss is, more huge are the errors made in data level. On the other hand, accuracy is defined as the number of errors made on data level.

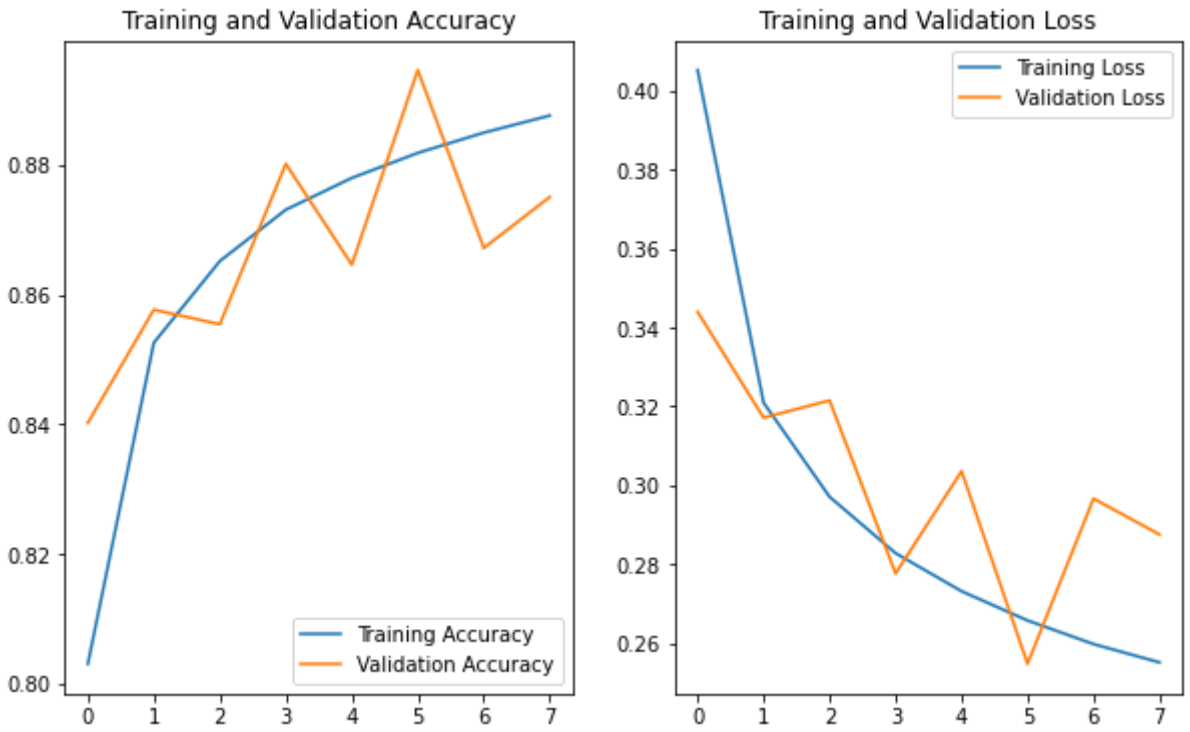


Figure 20: RNN Model Training & Validation Accuracy - Loss

Firstly, by the look on the loss values, it appears that the performance of the model cannot be characterized as poor. In general, it appears that the training data have a more smooth curve, compared to the validation data which have a fluctuation in their values. This exposes overfitting in the model. In general, overfitting occurs when the accuracy of the training set is greater than the testing/validation accuracy. In terms of "loss" overfitting reveals itself when the model has a low error in the training set and a high error in the testing/validation set. In this case, this is an expected behavior, taking into consideration that:

- The data are imbalanced and since SMOTE was used, the data are now partially biased. Also, as mentioned in the previous section, SMOTE was applied only in the training data, and not in the validation data.
- Only part of the original dataset was used and the same dataset was used for train, test and validation.
- The final configuration used in the model (number of epochs, EarlyStopping function, etc.).

Moreover, taking into consideration that the function EarlyStopping was used, and in combination with overfitting, the validation accuracy does not differ significantly among the epochs, and thus the code runs in fewer epochs than the configured ones.

Last but not least, it would be interesting to mention that the weights of the *RNN_model* were saved in a file named 'RNN.hdf5'. Also, the whole model was saved, in a folder named "RNN". The aforementioned files are stored in Google Drive inside the 'Models' folder.

Convolutional Neural Networks Model (CNN)

One other model created was the Convolutional Neural Networks Model - so called CNN model. Convolutional Neural Networks are a powerful image processing deep learning type often used in computer vision that comprises image and video recognition along with a recommender system and Natural Language Processing (NLP). Even though they are widely known for their usage in image/video processing, they can also be used for other data analysis and classification problems. Therefore, they can be applied across a diverse range of sectors to get precise results, covering critical aspects. A typical example of such neural network is the following:

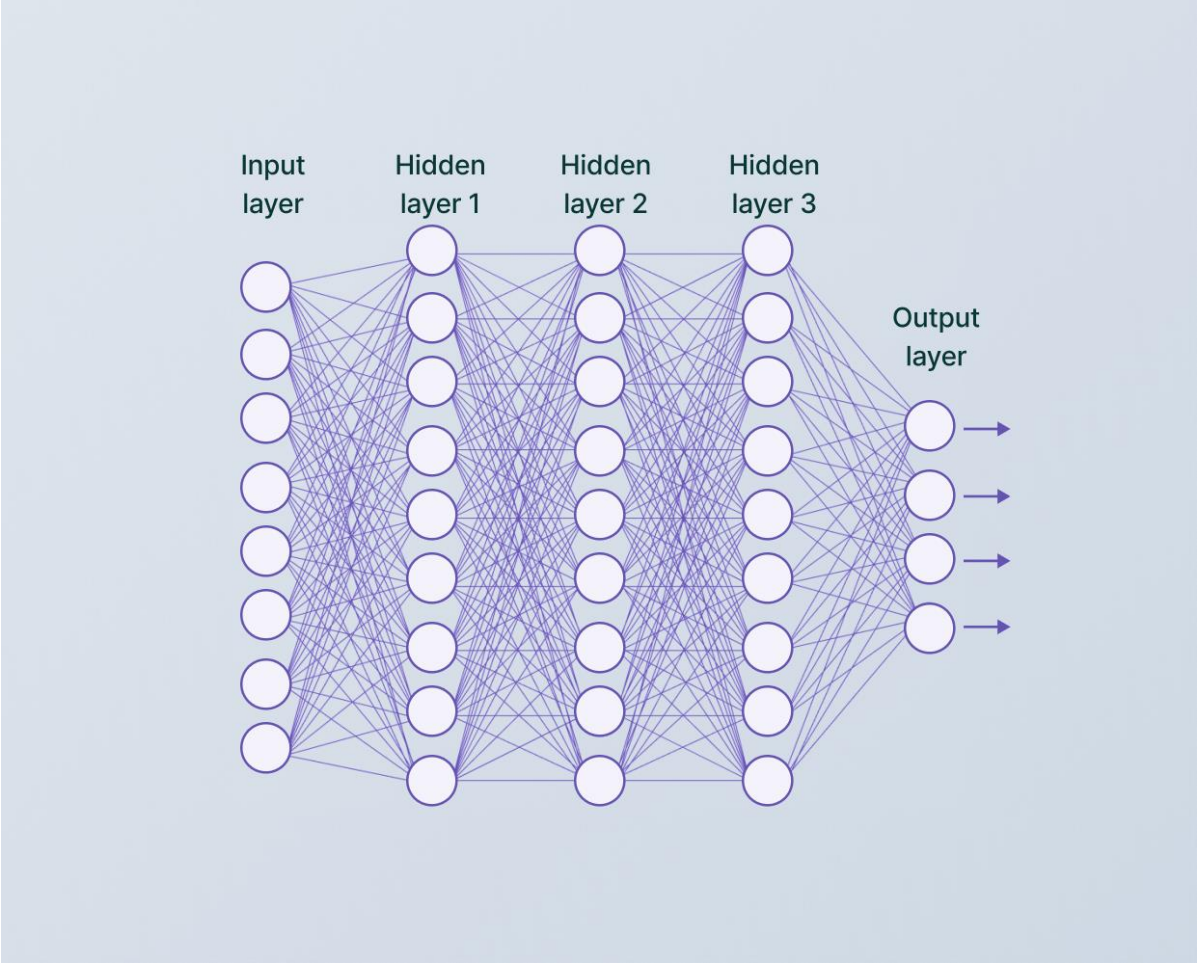


Figure 21: Convolutional Neural Network (source: https://www.v7labs.com/blog/convolutional-neural-networks-guide)

It was decided to build a CNN model because they are rather powerful and popular. Also, it would be very interesting to investigate the behavior of the model in a different classification problem, other than the image/video processing.

The first step in the process was to create a sequential model, called *CNN_model*. As mentioned in the previous chapter, the sequential class represents a linear stack of layers, and the sequential constructor takes an array of keras layers. Then, by using this model first an embedding layer was built, using the embedding matrix as weights. Also, two convolutional layers were added, including a pooling operation. More details about the used layers can be found below:

- **SpatialDropout1D**: promotes independence between feature maps.
- **Conv1D:** each channel in the input and filter is one-dimensional.
- **BatchNormalization:** applies a transformation that maintains the mean output close to zero and the output standard deviation close to one.
- **Dense**: these refer to the hidden layers which were built using 'relu' and 'softmax' as activation rules.
- **Dropout**: randomly sets input units to zero with a frequency of rate at each step during training time, which helps prevent overfitting.

By printing the model summary, it appears that the total number of parameters is equal to 5,095,778. The calculation results by summing the parameters of every layer:
embedding_1 + conv1d + batch_normalization + conv1d_1 + batch_normalization_1 + conv1d_2 + batch_normalization_2 + conv1d_3 + batch_normalization_3 + dense_2 + batch_normalization_4 + dense_3 + batch_normalization_5 + dense_4 + batch_normalization_6 + dense_5 == 4,500,000 + 230,656 + 1,024 + 196,864 + 1,024 + 98,432 + 512 + 49,280 + 512 + 8,256 + 256 + 4,160 + 256 + 4,160 + 256 +130 == 5,095,778
Another interesting thing to mention is that the parameters of the *dense_5* layer are equal to 130, because they are calculated using the formula *output_size * (input_size + 1) == number_parameters* (2 * (64 + 1) == 130). The same rule applies for the rest output layers (dense).

Similar to the RRN model explained in a previous section, the next step in the process was to compile the model. During the compilation step, three key factors were used (same as for RNN model):

- Adam Optimizer
- Binary Crossentropy Loss Function
- Accuracy Metric

After all the above actions took place, the next step was to train the model. In order to achieve this, the keras *fit* function was used and the same parameters as those used for the RNN model were specified:

- Training data.
- The number of epochs was set equal to 30.
- The batch size was set equal to 256.
- Validation data.
- Callbacks, and more precisely the *EarlyStopping* callback.

Then, it was important to check the accuracy metric. It appears that the accuracy of the model was equal to 0.906; and the binary cross-entropy was equal to 0.226.

Similarly to the RNN model, in order to better visualize and understand the model results, two graphs were created; “Training and Validation Accuracy” and “Training and Validation Loss”.

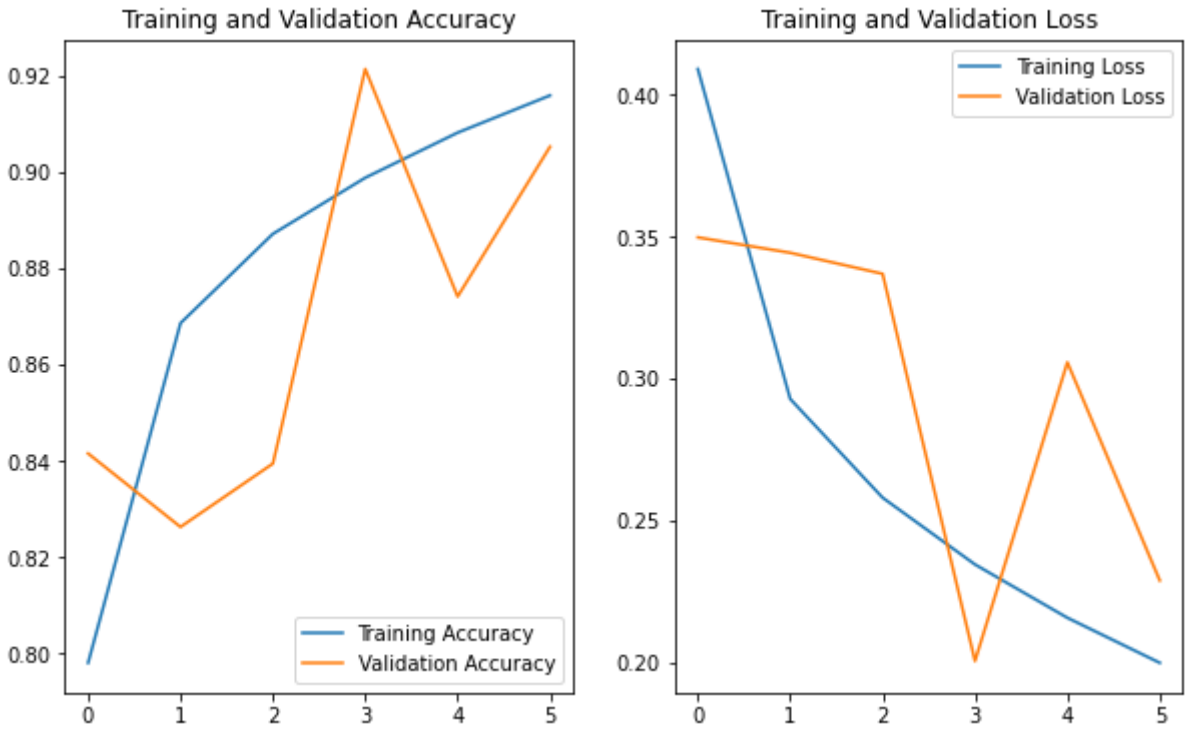


Figure 22: CNN Model Training & Validation Accuracy-Loss

Similarly to the RNN model, the training data curve appears to be more smooth, compared to the validation data which have a variation in their values. This exposes overfitting in the model. The most significant difference here is that fewer epochs have been executed (only 6 epochs). This affects the curves; those that correspond to the validation data, appear to be highly fluctuated compared to the ones presented in the RNN model. Taking into consideration that the function EarlyStopping was also used for the CNN, and in combination with overfitting, the validation accuracy does not differ significantly among the epochs, and thus the code runs in fewer epochs than the configured ones.

Last but not least, it would be interesting to mention that the weights of the *CNN_model* were saved in a file named 'CNN.hdf5'. Also, the whole model was saved, in a folder named "CNN". The aforementioned files are stored in Google Drive inside the 'Models' folder.

BERT Model

BERT (Bidirectional Encoder Representations from Transformers) is a recent paper published by researchers at Google AI Language. It has caused a stir in the Machine Learning community by presenting state-of-the-art results in a wide variety of NLP tasks, including Question Answering (SQuAD v1.1), Natural Language Inference (MNLI), and other.

BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an *encoder* that reads the text input and a *decoder* that produces a prediction for the task. Since BERT's goal is to generate a language model, only the encoder mechanism is necessary. The detailed workings of Transformer are described in the respective paper by Google.

As opposed to *directional* models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore, it is considered *bidirectional*, though it would be more accurate to say that it is non-directional. This characteristic allows the model to learn the context of a word based on all of its surroundings (left and right of the word).

To be more specific, the chart demonstrated below is a high-level description of the Transformer encoder. The input is a sequence of tokens, which are first embedded into vectors and then processed in the neural network. The output is a sequence of vectors of size H, in which each vector corresponds to an input token with the same index.

Nevertheless, when training language models, there is a challenge of defining a prediction goal.

Masked LM (MLM)

Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence. In technical terms, the prediction of the output words requires:

1. Adding a classification layer on top of the encoder output.
2. Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
3. Calculating the probability of each word in the vocabulary with softmax.

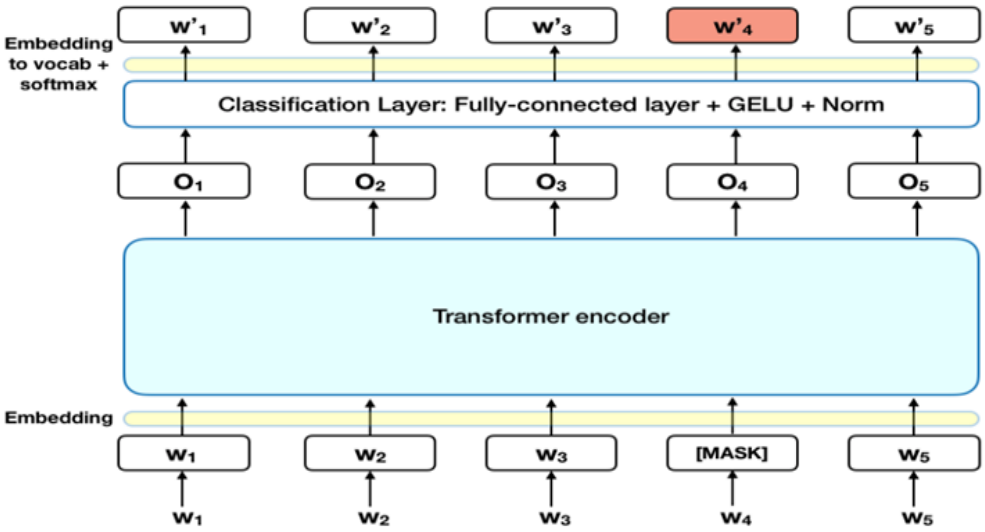


Figure 23: BERT Transformer

The BERT loss function takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked words. As a consequence, the model converges slower than directional models, a characteristic which is offset by its increased context awareness.

Next Sentence Prediction (NSP)

In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original

document. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence.

To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

1. A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
3. A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.

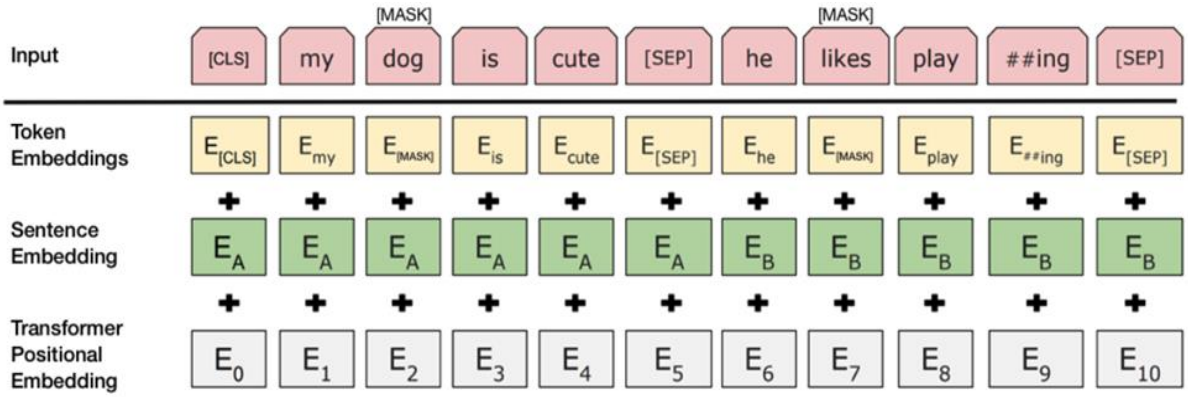


Figure 24: Masking tokens for Embedding layers

To predict if the second sentence is indeed connected to the first, the following steps are performed:

1. The entire input sequence goes through the Transformer model.
2. The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).
3. Calculating the probability of IsNextSequence with SoftMax.

When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

How to use BERT (Fine-tuning)

BERT can be used for a wide variety of language tasks, while only adding a small layer to the core model:

1. **Classification tasks** such as sentiment analysis are done similarly to Next Sentence classification, by adding a classification layer on top of the Transformer output for the [CLS] token.
2. In **Question Answering tasks** (e.g. SQuAD v1.1), the software receives a question regarding a text sequence and is required to mark the answer in the sequence. Using BERT, a Q&A model can be trained by learning two extra vectors that mark the beginning and the end of the answer.
3. In **Named Entity Recognition (NER)**, the software receives a text sequence and is required to mark the various types of entities (Person, Organization, Date, etc) that appear in the text. Using BERT, a NER model can be trained by feeding the output vector of each token into a classification layer that predicts the NER label.

In the fine-tuning training, most hyper-parameters stay the same as in BERT training, and the aforementioned Google paper gives specific guidance (Section 3.5) on the hyper-parameters that require tuning. The BERT team has used this technique to achieve state-of-the-art results on a wide variety of challenging natural language tasks, detailed in Section 4 of the paper.

Takeaways

1. Model size matters, even at a huge scale. BERT large, with 345 million parameters, is the largest model of its kind. It is demonstrably superior on small-scale tasks to BERT base, which uses the same architecture with “only” 110 million parameters.
2. With enough training data, more training steps result to a higher accuracy. For instance, on the MNLI task, the BERT base accuracy improves by 1.0% when trained on 1M steps (128,000 words batch size) compared to 500K steps with the same batch size.
3. BERT’s bidirectional approach (MLM) converges slower than left-to-right approaches (because only 15% of words are predicted in each batch) but bidirectional training still outperforms left-to-right training after a small number of pre-training steps.

BERT is undoubtedly a breakthrough in the use of Machine Learning for Natural Language Processing. The fact that it's approachable and allows fast fine-tuning will likely allow a wide range of practical applications in the future.

Vectorizers

Bag of Words

Bag of words is a Natural Language Processing technique of text modeling. In technical terms, it is a method of feature extraction with text data. This approach is a simple and flexible way of extracting features from documents.

A bag of words is a representation of text that describes the occurrence of words within a document. It keeps track of word counts and disregards the grammatical details and the word order. Moreover, it is called a "bag" of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

However, one of the biggest problems with text is that it is messy and unstructured, and machine learning algorithms prefer structured, well defined fixed-length inputs. By using the Bag-of-Words technique variable-length texts are converted into a fixed-length vector.

Also, at a much granular level, the machine learning models work with numerical data rather than textual data. So, to be more specific, by using the bag-of-words (BoW) technique, a text is converted into its equivalent vector of numbers.

Although Bag-of-Words is quite efficient and easy to implement, there are some disadvantages to this technique which are given below:

1. The model ignores the location information of the word. The location is of high importance in regard to information in the text. For example, "today is off" and "Is today off", have the exact same vector representation in the BoW model.
2. Bag of word models does not respect the semantics of the word. For example, the words 'soccer' and 'football' are often used in the same context. However, the vectors corresponding to these words are quite different in the bag of words model. The problem becomes more serious while modeling sentences. Ex: "Buy used cars"

and “Purchase old automobiles” are represented by totally different vectors in the Bag-of-words model.

3. The range of vocabulary is a big issue faced by the Bag-of-Words model. For example, if the model comes across a new word, it has not encountered yet, but is still of high informative importance, like biblioklept[4]. The BoW model will probably end up ignoring this word as this word has not been seen by the model yet.

TF-IDF

Term frequency-inverse document frequency is a text vectorizer that transforms the text into a usable vector. It combines two concepts, Term Frequency (TF) and Document Frequency (DF).

The term frequency is the number of occurrences of a specific term in a document. It indicates how important a specific term in a document is and represents every text from the data as a matrix whose rows are the number of documents and columns are the number of distinct terms throughout all documents.

Document frequency is the number of documents containing a specific term and indicates how common the term is.

Inverse document frequency (IDF) is the weight of a term that aims to reduce the weight of a term if the term’s occurrences are scattered throughout all the documents.

The higher the DF of a term, the lower the IDF for the term. When the number of DF is equal to n which means that the term appears in all documents, the IDF will be zero, since log(1) is zero.

Limitations:

1. It is only useful as a lexical level feature.
2. Synonymities are neglected.
3. It doesn't capture semantics.
4. The highest TF-IDF score may not make sense with the topic of the document, since IDF gives high weight if the DF of a term is low.

[4] Definition: One who steals books

5. It neglects the sequence of the terms.

ML Models

Logistic Regression

Logistic regression is named for the function used at the core of the method, the logistic function. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

$$1/(1+e^{-value})$$

Where e is the base of the natural logarithms and value is the actual numerical value that is transformed. Below a plot is depicted, of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function.

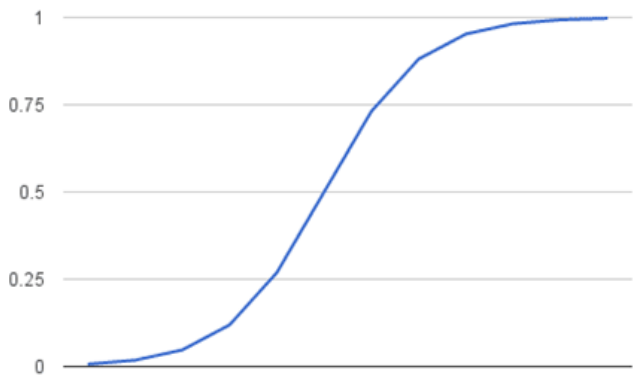


Figure 25: Logistic function (-5,5)

Logistic regression uses an equation as the representation, very much like linear regression.

Input values (x) are combined linearly using weights or coefficient values to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary value (0 or 1) rather than a numeric value.

Below is an example logistic regression equation:

$$y = \frac{e^{b0+b1*x}}{1 + e^{b0+b1*x}}$$

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in the input data has an associated b coefficient (a constant real value) that must be learned from the training data.

The actual representation of the model that would be stored in memory or in a file are the coefficients in the equation (the beta value or b's).

Logistic regression models the probability of the default class (e.g. the first class). The probability that an input (X) belongs to the default class (Y=1) is modeled as:

$$P(X) = P(Y = 1|X)$$

The probability prediction must be transformed into a binary value (0 or 1) in order to actually make a probability prediction.

Logistic regression is a linear method, but the predictions are transformed using the logistic function. The impact of this is that the predictions are no longer understood as a linear combination of the inputs as with linear regression, and the model can be stated as:

$$p(X) = \frac{e^{b0+b1*X}}{1 + e^{b0+b1*X}}$$

The above equation can be turned around as follows:

$$\ln \frac{p(X)}{(1 - p(X))} = b0 + b1 * X$$

This is useful because the calculation of the output on the right is linear again (just like linear regression), and the input on the left is a log of the probability of the default class.

This ratio on the left is called the odds of the default class. Odds are calculated as a ratio of the probability of the event divided by the probability of not the event, e.g. 0.8/(1-0.8) which has the odds of 4. So, it can be written instead as:

$$\ln odds = b0 + b1 * X$$

Because the odds are log transformed, this left-hand side is called the log-odds or the probit. It is possible to use other types of functions for the transform, but as such it is common to refer to the transform that relates the linear regression equation to the probabilities as the link function, e.g. the probit link function.

The exponent can be moved back to the right and the equation is transformed to the following:

$$odds = e^{b0+b1*X}$$

All of the above conclude that indeed the model is still a linear combination of the inputs, but that this linear combination relates to the log-odds of the default class.

Support Vector Machines For Classification

Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. The objective of the SVM algorithm is to find a *hyperplane* in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

Let us consider two independent variables x1, x2 and one dependent variable which is either a blue circle or a red circle.

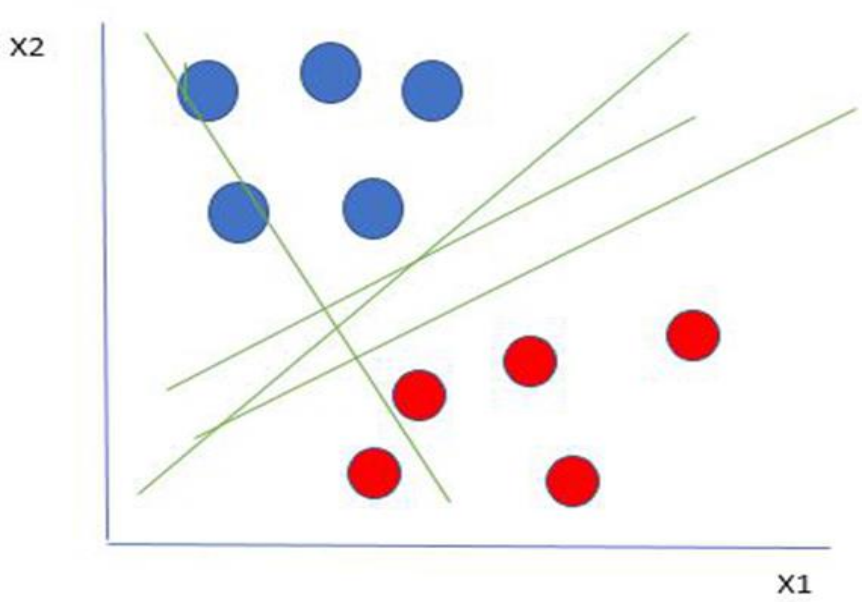


Figure 26: Hyperplane - one line

From the figure above, it is very clear that there are multiple lines (the hyperplane here is a line because only two input features x1, x2 are considered) that segregates the data points or does a classification between red and blue circles.

One reasonable choice as the best hyperplane is the one that represents the largest separation or margin between the two classes.

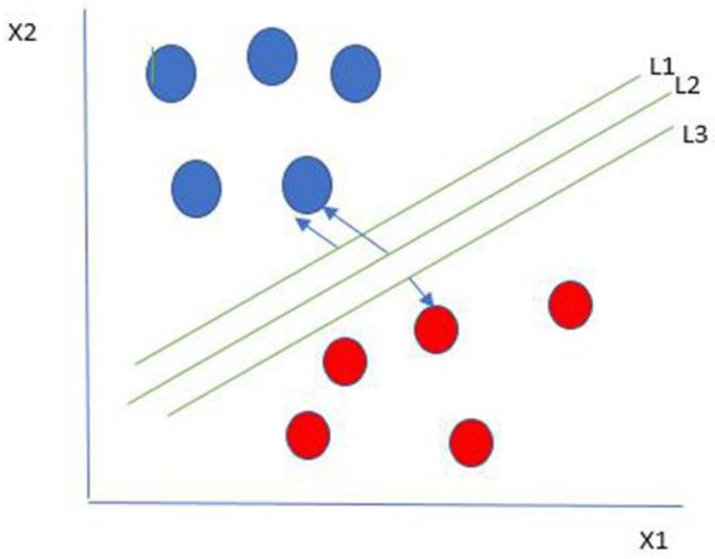


Figure 27: Hyperplane - hard margin

So, the hyperplane whose distance from it to the nearest data point on each side is maximized is chosen. If such a hyperplane exists, it is known as the maximum-margin hyperplane/hard margin. Thus, from the above figure, L2 was selected.

Let us consider a scenario like shown below:

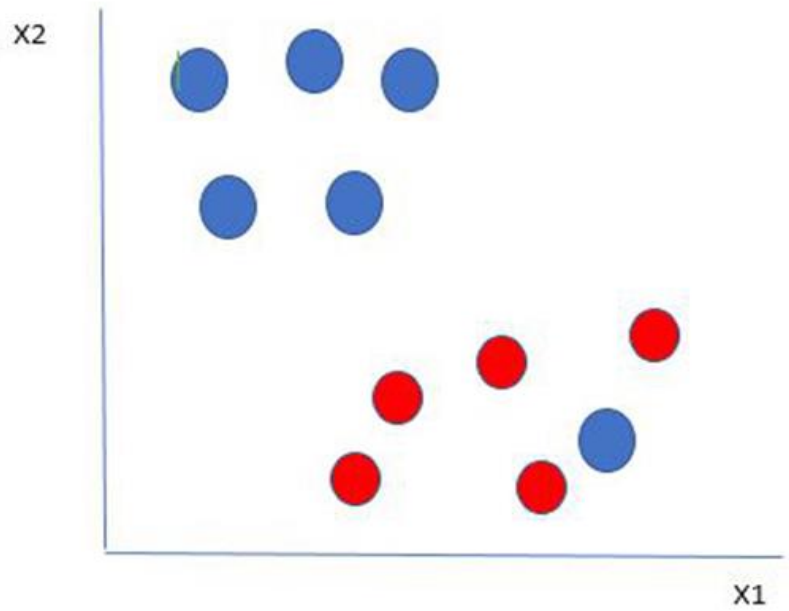


Figure 28: Outlier Example

In this case, one blue ball is observed in the boundary of the red ball. The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM is robust to outliers.

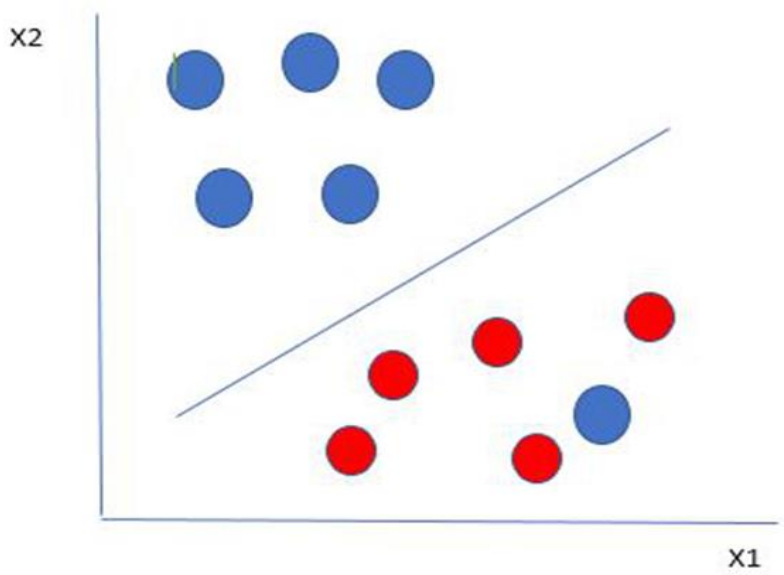


Figure 29: Hyperplane - Ignoring Outliers/Soft Margin

In this type of data points what SVM does is, it finds maximum margin as done with previous data sets along with that it adds a penalty each time a point crosses the margin. The margins in these types of cases are called soft margins. When there is a soft margin to the data set, the SVM tries to minimize $(1/margin + (\sum penalty)$.

Hinge loss is a commonly used penalty. If there are no violations, there is no hinge loss. If there are some violations, the hinge loss is proportional to the distance of said violations.

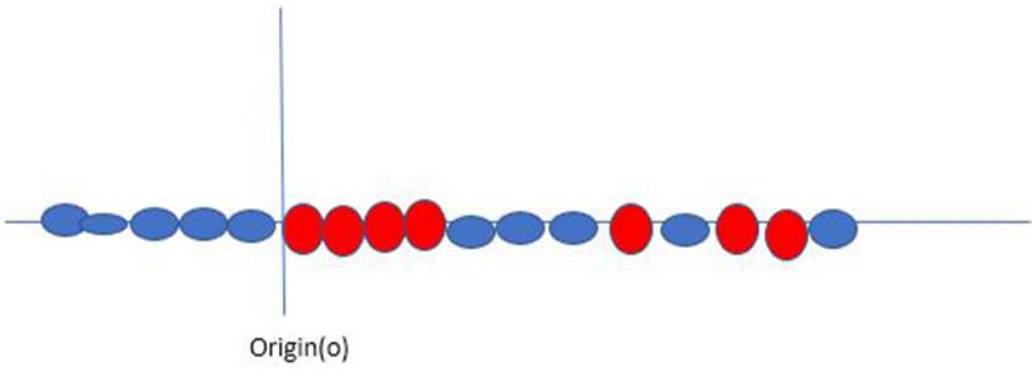


Figure 30: Not Linearly Separable Data

In case the data are not linearly separable, and instead their order resembles the figure above. SVM solves this by creating a new variable using a kernel. SVM calls a point xi on the line, and creates a new variable yi as a function of distance from origin o.

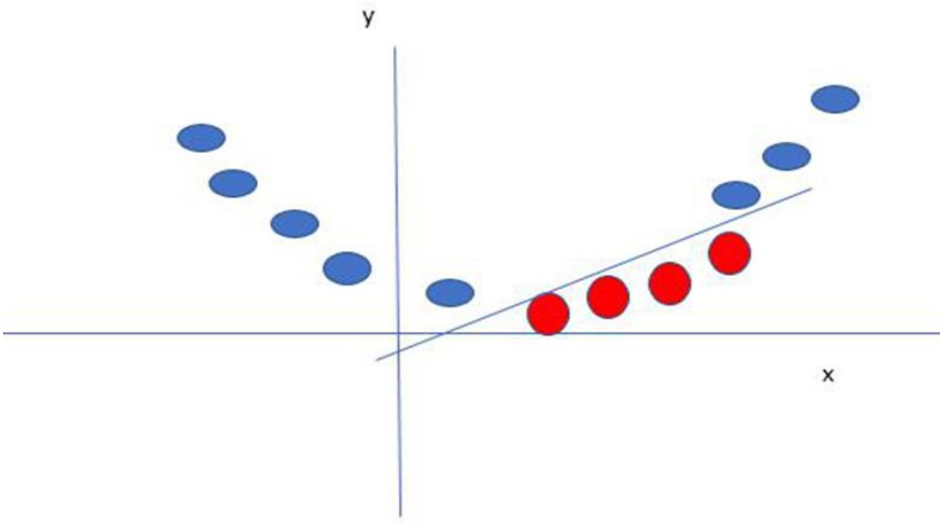


Figure 31: Variable Creation as FoD

In this case, the new variable y is created as a function of distance from the origin. A nonlinear function that creates a new variable is referred to as a kernel.

SVM Kernel

The SVM kernel is a function that takes low-dimensional input space and transforms it into higher-dimensional space, i.e. it converts not separable problems to separable problems. It is mostly useful in non-linear separation problems. Simply put the kernel does some extremely complex data transformations then finds out the process to separate the data based on the labels or outputs defined.

Advantages of SVM

1. Effective in high dimensional cases
2. It is memory efficient as it uses a subset of training points in the decision function called support vectors
3. Different kernel functions can be specified for the decision functions and it is possible to specify custom kernels

K-Nearest Neighbors Classifier

The k-nearest neighbors (KNN) algorithm is a data classification method for estimating the likelihood that a data point becomes a member of one group, or another based on what group the data points nearest to it belong to.

The k-nearest neighbor algorithm is a type of *supervised machine learning* algorithm used to solve *classification* and *regression* problems. However, it is mainly used for classification problems.

It does not perform any training when you supply the training data. Instead, it just stores the data during the training time and does not perform any calculations but it does not build a model until a query is performed on the dataset. This makes KNN ideal for data mining.

It is considered a *non-parametric* method because it does not make any assumptions about the underlying data distribution. Simply put, KNN tries to determine what group a data point belongs to by looking at the data points around it.

Consider there are two groups, A and B. To determine whether a data point is in group A or group B, the algorithm looks at the states of the data points near it. If the majority of data points are in group A, it is very likely that the data point in question is in group A and vice versa. In short, KNN as a supervised clustering algorithm involves classifying a data point by looking at the nearest annotated data point, also known as the nearest neighbor. On the other hand, K-means clustering is an unsupervised clustering algorithm that groups data into a K number of clusters.

As mentioned above, the KNN algorithm is predominantly used as a classifier. Unlike classification using artificial neural networks, k-nearest neighbors' classification is easy to understand and simple to implement. It is ideal in situations where the data points are well defined or nonlinear.

In essence, KNN performs a voting mechanism to determine the class of an unseen observation. This means that the class with the majority vote will become the class of the data point in question. If the value of K is equal to one, then we will use only the nearest neighbor to determine the class of a data point. If the value of K is equal to ten, then we will use the ten nearest neighbors, and so on.

To put that into perspective, consider an unclassified data point X. There are several data points with known categories, A and B, in a scatter plot. Suppose the data point X is placed near group A. A data point is classified by looking at the nearest annotated points. If the value of K is equal to one, then only one nearest neighbor to determine the group of the data point is used. In this case, the data point X belongs to group A as its nearest neighbor is in the same group. If group A has more than ten data points and the value of K is equal to 10, then the data point X will still belong to group A as all its nearest neighbors are in the same group. Then, suppose another unclassified data point Y is placed between group A and group B. If K is equal to 10, the group that gets the most votes is picked, meaning that Y is classified to the group in which it has the greatest number of neighbors. For example, if Y has seven neighbors in group B and three neighbors in group A, it belongs to group B. The fact that the classifier assigns the category with the highest number of votes is true regardless of the number of categories present.

Hence, there are four ways to calculate the distance measure between the data point and its nearest neighbor: **Euclidean distance**, **Manhattan distance**, **Hamming distance**, and **Minkowski distance**. Out of the three, Euclidean distance is the most commonly used distance function or metric.

To validate the accuracy of the KNN classification, a confusion matrix is used. Other statistical methods such as the likelihood-ratio test are also used for validation. There is not a specific way to determine the best K value – in other words – the number of neighbors in KNN. One way to select the value is by considering (or pretending) that a part of the training samples is "unknown". Then, the unknown data is categorized in the test set by using the k-nearest neighbors' algorithm and the categorization is evaluated by comparing it with the information already present in the training data. When dealing with a two-class problem, it's better to choose an odd value for K. Otherwise, a scenario can arise where the number of neighbors in each class is the same. Also, the value of K must not be a multiple of the number of classes present.

Another way to choose the optimal value of K is by calculating the sqrt(N), where N denotes the number of samples in the training data set. However, K with lower values, such as K=1 or K=2, can be noisy and subjected to the effects of outliers. The chance for overfitting is also high in such cases. On the other hand, K with larger values, in most cases, will give rise to smoother decision boundaries, but it should not be too large. Otherwise, groups with a

fewer number of data points will always be outvoted by other groups. Plus, a larger K will be computationally expensive.

Some of the advantages of using the k-nearest neighbors' algorithm are:

1. It is easy to understand and simple to implement
2. It can be used for both classification and regression problems
3. It is ideal for non-linear data since there is no assumption about underlying data
4. It can naturally handle multi-class cases
5. It can perform well with enough representative data

Some of the disadvantages of using the k-nearest neighbors' algorithm are:

1. Associated computation cost is high as it stores all the training data
2. Requires high memory storage
3. Need to determine the value of K
4. Prediction is slow if the value of N is high
5. Sensitive to irrelevant features

Random Forests For Classification

Random forest, like its name implies, consists of a large number of *individual decision trees* that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes the model's prediction (see figure below).

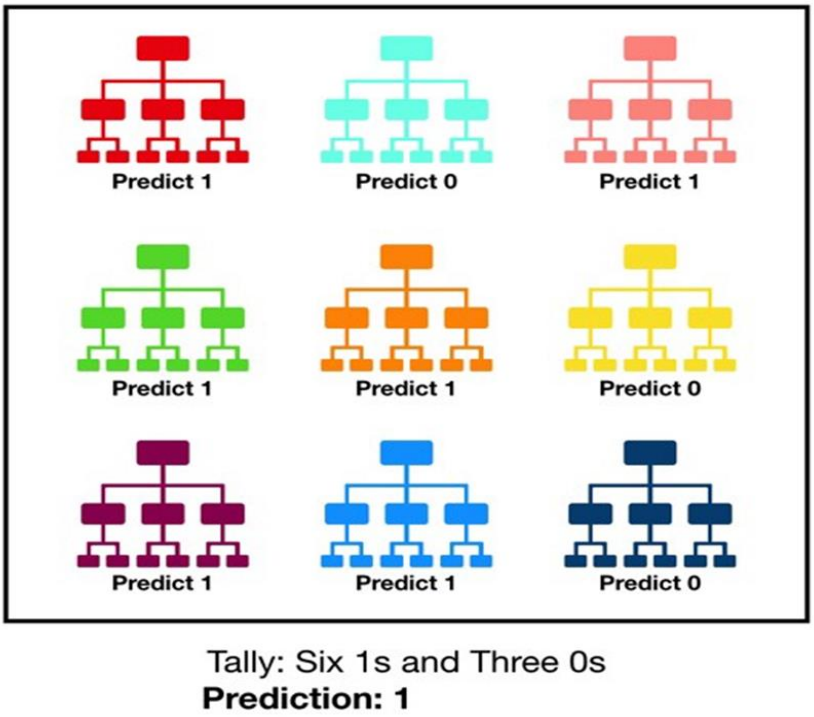


Figure 32: Random Forests Six 1s - Three 0s

The fundamental concept behind random forest is a simple but powerful one - the wisdom of crowds. The reason that the random forest model works so well is the fact that a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they do not constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So, the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.
2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

In order for the random forest to make accurate class predictions:

1. Features that have at least some predictive powers are needed.
2. The trees of the forest and more importantly their predictions need to be uncorrelated (or at least have low correlations with each other). While the algorithm itself via feature randomness tries to engineer these low correlations, the features selected and the hyper-parameters chosen will impact the ultimate correlations as well.

Naive Bayes Classifier

Naïve Bayes — a probabilistic approach for constructing the data classification models. It is formulated as several methods, widely used as an alternative to the *distance-based K-Means* clustering and *decision tree forests* and deals with probability as the "likelihood" that data belongs to a specific class. The *Gaussian* and *Multinomial* models of the naïve Bayes exist.

The multinomial model provides an ability to classify data that cannot be represented numerically. Its main advantage is the significantly reduced complexity. It provides an ability to perform the classification, using small training sets, not requiring to be continuously re-trained.

Multinomial Naïve Bayes Classifiers

The multinomial Naïve Bayes is widely used for assigning documents to classes based on the statistical analysis of their contents. It provides an alternative to the "heavy" AI-based semantic analysis and drastically simplifies textual data classification.

The classification aims to assign fragments of text (i.e. documents) to classes by determining the probability that a document belongs to the class of other documents, having the same subject. Each document consists of multiple words (i.e. terms), that contribute to an understanding of a document's contents. A class is a tag of one or multiple documents, referring to the same subject. The labeling of documents with one of the existing classes is done by performing the statistical analysis, testing the hypothesis that a document's terms already occurred in other documents from a particular class. This

increases the probability that a document is from the same class as the documents, already classified.

As an example, we have the samples $\boldsymbol{S}$ intended to be classified. Each sample in $\boldsymbol{S}$ is defined as a string that occurred in one or multiple documents from a class $\boldsymbol{C}$. The classification is performed, based on the documents D, already classified. To perform the classification, the terms in $\boldsymbol{S}$ are represented by a vector $\boldsymbol{W}$. Each feature $\boldsymbol{w}_{\mathrm{i}}$ of $\boldsymbol{W}$ is an occurrence frequency of the corresponding $\boldsymbol{i}$-th term in the documents from class $\boldsymbol{C}$. Each features vector $\boldsymbol{W}$ — a term/class occurrence frequency histogram:

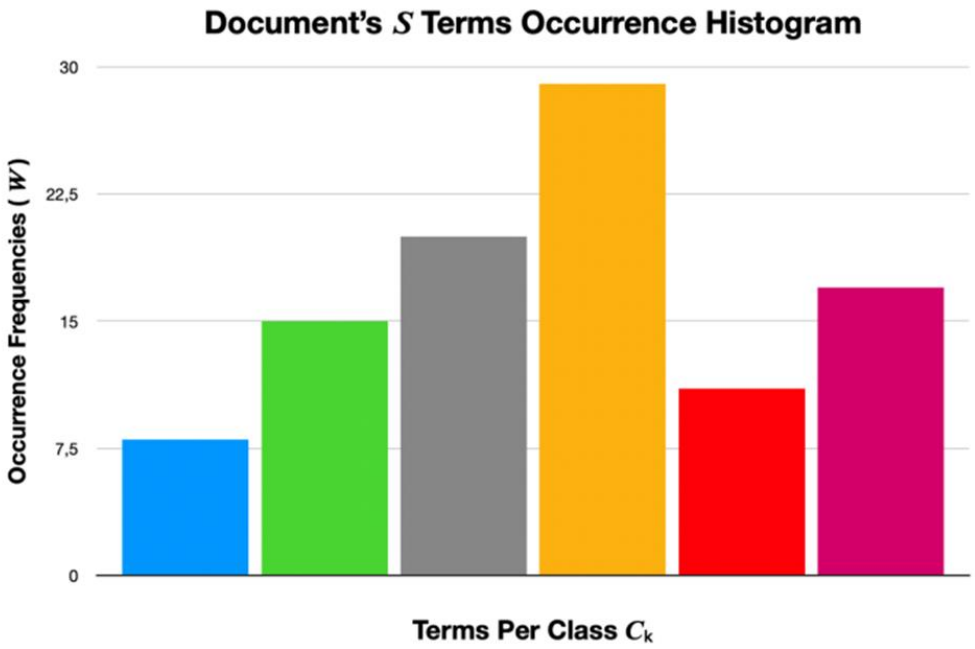


Figure 33: Document's S Terms Occurrence Histogram

The classification is performed by estimating the probability $\boldsymbol{p}(\boldsymbol{w}_{\mathrm{i}} \mid \boldsymbol{C})$ $\boldsymbol{i}$=$\boldsymbol{1}$…$\boldsymbol{n}$, that the terms in $\boldsymbol{S}$ occurred in the documents D from the class $\boldsymbol{C}$.

The multinomial model solely relies on the evaluation of probability-based decision function, concluded from the Bayes theorem. The Bayesian probability $\boldsymbol{p}(\boldsymbol{C}_{\mathrm{k}} \mid \boldsymbol{W})$ is computed as follows:

$$p(C_k \mid \boldsymbol{W}) = \frac{p(C_k) \times p(\boldsymbol{W} \mid C_k)}{p(\boldsymbol{W})}$$

The main idea behind the naïve Bayes is that all features in $\boldsymbol{W}$ independently contribute to the probability that $\boldsymbol{S}$ belongs to $\boldsymbol{C}_{\mathrm{k}}$.

Alternatively, the posterior $\boldsymbol{p}(\boldsymbol{C}_{\mathrm{k}} \mid \boldsymbol{W})$ is expressed as:

$$posterior = \frac{prior \times likelihood}{evidence}$$

The estimated prior $\boldsymbol{p}(C_{\mathrm{k}})$ and likelihood $\boldsymbol{p}(\boldsymbol{W} \mid C_{\mathrm{k}})$ proportionally contribute to an outcome that C_{k} is being the class of $\boldsymbol{S}$.

However, only its numerator affects the estimation of posterior $\mathbf{Pr}(C_{\mathrm{k}} \mid \boldsymbol{W})$. Despite this, the evidence $\boldsymbol{p}(\boldsymbol{W})$ equally scales $\mathbf{Pr}(C_{\mathrm{k}} \mid \boldsymbol{W})$ of each C_{k} and might be simply omitted from the estimate.

The multinomial model is used for document classification with an assumption that $\boldsymbol{S}$ is represented by the features vector $\boldsymbol{W}$. Each feature $\boldsymbol{w}_{\mathrm{i}}$ is the count (i.e. frequency) with which $\boldsymbol{i}$-th term from $\boldsymbol{S}$ occurred in the documents D, already assigned to one of the classes C. Originally, the posterior $\mathbf{Pr}(\boldsymbol{W} \mid C_{\mathrm{k}})$ of each C_{k} is estimated as:

$$\mathbf{Pr}(\boldsymbol{W} \mid C_k) = \frac{(\sum_{i=1}^{n} \boldsymbol{w}_i)!}{\prod_{i=1}^{n} \boldsymbol{w}_i!} \times \prod_{i=1}^{n} p_{ki}^{w_i}$$

The prior $\mathbf{Pr}(\boldsymbol{C}_{\mathrm{k}})$ is a quotient. which numerator is estimated as the factorial of the sum of all features $\forall w_{\mathrm{ki}} \in \boldsymbol{W}$ $\boldsymbol{i}$=$\mathbf{1}$…$\boldsymbol{n}$. In turn, the denominator is obtained as a product of all features' w_{ki} factorials.

The numerator is evaluated as a probability distribution, which is the likelihood of all possible outcomes that $\boldsymbol{S}$ occurred in documents $\boldsymbol{D}$ from class $\boldsymbol{C}_{\mathrm{k}}$. An overall amount of such outcomes is equal to the factorial of the sum of features $\forall w_{\mathrm{ki}} \in \boldsymbol{W}$, mentioned above.

In turn, the denominator of $\boldsymbol{p}(\boldsymbol{C}_{\mathrm{k}})$ is obtained as an amount of all possible outcomes that features of $\boldsymbol{W}$ occurred in the corpus of documents $\boldsymbol{D}$. This is firmed into an arithmetic progression obtained as the sum of each feature's w_{ki}! factorial.

The computations, above, are very similar to *Legendre's* prime factorization of vector $\boldsymbol{W}$. Also, it implies multivariate binomial distribution computation. When the denominator is

large enough, the $\boldsymbol{p}(\boldsymbol{C}_{\mathrm{k}})$ significantly decreases. This normally provides an ability to exclude features, that occurred in all documents from the corpus $\boldsymbol{D}$.

The likelihood of $\boldsymbol{S}$ given $\boldsymbol{C}_{\mathrm{k}}$ is the product of terms' probabilities $\boldsymbol{p}_{\mathrm{ki}}$ in the statistical degree of w_{ki}, rejecting the null hypothesis:

$$p(\boldsymbol{W} \mid C_k) = \prod_{i=1}^{n} p(\boldsymbol{w}_i \mid C_k)$$

The multinomial model, above, is applied to compute the probability $\boldsymbol{p}(\boldsymbol{W} \mid \boldsymbol{C}_{\mathrm{k}})$ that $\boldsymbol{S}$ is from $\boldsymbol{C}_{\mathrm{k}}$. Although, in this case, the goal is to compute the probability that $\boldsymbol{C}_{\mathrm{k}}$ is the class of $\boldsymbol{S}$. The $\mathbf{Pr}(C_{\mathrm{k}} \mid \boldsymbol{W})$ is easily computed by representing the multinomial model in log-space:

$$\log \mathbf{Pr}(C_k \mid \boldsymbol{W}) \propto \log p(C_k) + \sum_{i=1}^{n} \boldsymbol{w}_i * \log p(\boldsymbol{w}_i \mid C_k)$$

Since the likelihood $\boldsymbol{p}(\boldsymbol{W} \mid C_{\mathrm{k}})$ is a product of probabilities $\boldsymbol{p}_{\mathrm{ki}} \equiv \boldsymbol{p}(w_{\mathrm{i}} \mid C_{\mathrm{k}})$, it's represented on the logarithm scale. In this case, $\boldsymbol{p}(\boldsymbol{W} \mid C_{\mathrm{k}})$ is proportional to the maximum possible degree supported by the statistical model.

In the equation above, $\mathbf{log}(\boldsymbol{p})$ is the natural logarithm of probability $\boldsymbol{p}$, evaluated as:

$$\log(\boldsymbol{p}) = \begin{cases} \ln(\boldsymbol{p}), & \boldsymbol{p} > 1 \\ 1.0, & \boldsymbol{p} \leq 1 \end{cases}$$

The $\boldsymbol{p}$'s logarithm is always $\mathbf{1.0}$ when $\boldsymbol{p}$ is less than or equal to $\mathbf{1.0}$, and the natural logarithm $\mathbf{ln}(\boldsymbol{p})$, unless otherwise. The decision about the class C_{s} of S is made, such as that:

$$C_s = \underset{k \in \{1..m\}}{\operatorname{argmax}} \mid \mathbf{Pr}(C_k \mid \boldsymbol{W}) \mid$$

The class C_{s} is determined as one of the existing classes C_{k}, for which an absolute value of $\mathbf{Pr}(C_{\mathrm{k}} \mid \boldsymbol{W})$ is the maximum. Since $\mathbf{Pr}(C_{\mathrm{k}} \mid \boldsymbol{W})$ is always negative, its absolute value is taken.

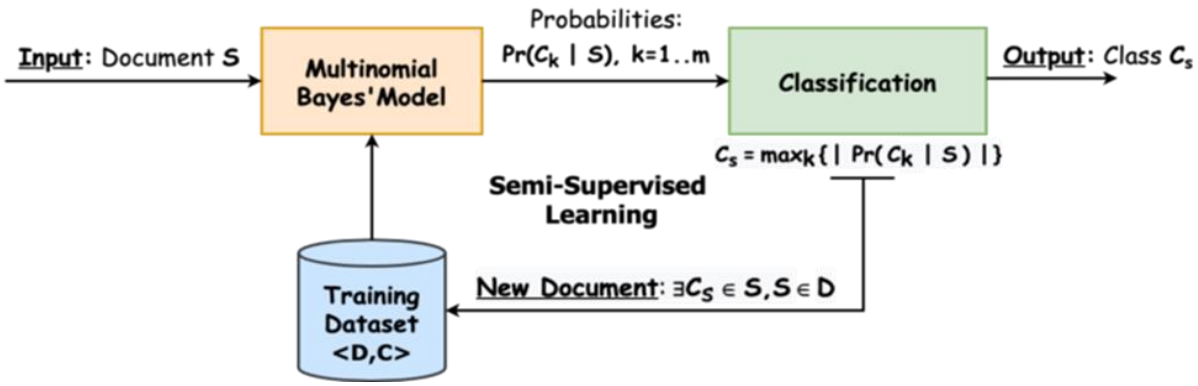


Figure 34: Semi-Supervised Learning - Naive Bayes Model Example

The process, illustrated above, provides an ability to perform classification, assigning samples to a finite set of classes, similar to using the expectation-maximization algorithm (EM).

Experiments – Setup/Configuration

CNN/RNN Model Experiments

Model Training

As described in previous chapters[5] a key point in the process was the training of the model. Before concluding to the 30 epochs and batch size 256, it was first considered to have 30 epochs and batch size equal to 32. Due to the limitation of resources capacity, after a couple of hours the execution would time-out. Apart from these, various experiments were performed with different combinations of batch size and number of epochs, but the issue persisted. The decision of 30 epochs and batch size equal to 256 was the most optimal solution.

An interesting observation is that each time the code was executed (epochs=30, batch_size=256), the number of epochs differed, yet the difference was not significant. Also, minor differences existed in the scores, yet not significant ones.

[5] Reference is made to the RNN and CNN

Fit models with whole dataset

An investigation performed, was to try to fit the model while using the whole dataset (1,381,262 records). This approach was left aside, because each epoch needed 6.30 hours to be executed, and after 2-3 epochs, a time-out error was displayed, and a reconnection was required in order to proceed. It is worth mentioning that regardless of the amount of data used, the accuracy was more or less the same.

Confusion Matrix & Classification Report

Something interesting for analysis would be the creation of the confusion matrix and the classification report. The confusion matrix (for both models) would be created using the confusion_matrix package and the *y_test* were set as the true values and the *RNN_model.predict(x_test)* as the predicted values. Similarly, for the classification report, the package classification_report was used.
The issue with this approach is that when trying to execute the code, an error message was displayed stating that there was a different number of dimensions between true and predicted values. It was decided from our team, to investigate further the root cause, and fix the issues in a future release.

Spell checker & Autocorrection

The goal was, for each sentence of the dataset, an automatic spelling and grammar correction process to precede introduction of several solutions and apply the most related one. The needed actions would have the following results:

- words missing letters, such as “speling,” would be corrected to “spelling”;
- misspelled words, such as “korrect,” would change to “correct.”

Specific libraries & modules were used by trying to accomplish the aforementioned. ‘TextBlob’, ‘autocorrect’ and ‘SpellChecker’ were the most examined ones. Ready - made functions or new created ones from us, were used alongside the several testings.

TextBlob is a Python library for processing data with text structure. It provides a consistent API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, and etc. This library is also a part of

the app structure/ configuration, for which further details will be displayed in the following sections.

Autocorrect is a spelling corrector in python which currently supports English, Polish, Turkish, Russian, Ukrainian, Czech, Portuguese, Greek, Italian, Vietnamese, French and Spanish. New languages can be added from anyone that wants to be involved.

SpellChecker is a spelling algorithm that uses a Levenshtein Distance algorithm to find permutations within an edit distance of 2 from the original word. Then compares all permutations (insertions, deletions, replacements, and transpositions) to known words in a word frequency list. Those words that are found more often in the frequency list are more likely the correct results.

Unfortunately, the above couldn't be applied in time, to the whole dataset due to computational cost and technical difficulties, therefore so as to proceed to the modeling phase only targeted corrections were made. However, its examination and appliance is to be analyzed deeper in the future and tested for better future results.

Machine Learning Models

In order to select the best model that appears to perform better in our data, first we had to take a closer look at them. *Bag of words vectorizer* was used in order to select the **top 10** most frequent words. The results are displayed in the picture below.

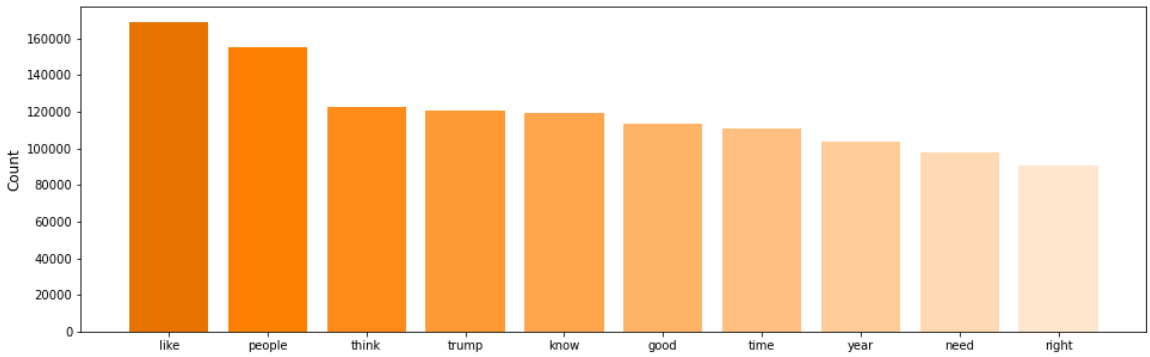


Figure 35: Top 10 most frequent words Bag of Words

Moreover, *tf idf vectorizer* was also used in order to display the **top 10** most frequent words and the respective result can be found in the picture below.

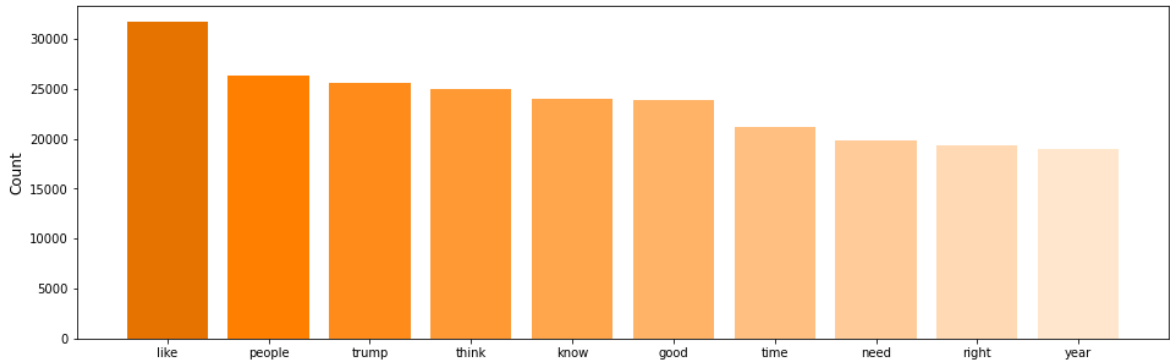


Figure 36: Top 10 most frequent words TF-IDF

It is clear that only a few words have a different order between the two vectorizers and most of the words are in the same order and frequency in both vectorizing ways.

Subsequently, we tested both vectorizers bag of words and tf idf in Logistic Regression and observed the way each vectorizer responded to it. The model that used bag of words vectorizer achieved a test set score of **0.88** in the full dataset. The model that used the tf idf vectorizer achieved a test set score of **0.89** in the whole dataset. Having made this observation, we proceeded with tf idf as the main vectorizer for all the models we tested.

However, due to some hardware limitations that we faced we chose a specific data length, in order to get the largest chunk size possible. If the chunk size exceeded the available positive records, the system retrieved as many positive records as could be found. Then, the data were shuffled and both bag of words features and tf idf features were split into train and test matrices.

Following, we had to get rid of imbalances in the data set. That was achieved by either augmentation or downsampling methods. If there were no records to be created, an *exception* was raised in order to stop the execution of the augmentation process and downsampling was performed, otherwise if there were records to be created the system proceeded to their creation by using the **SMOTE** algorithm.

Having shaped the data in the desired form, we trained our models. Logistic Regression with a score of **0.928**, Multinomial Naïve Bayes with a score of **0.913**, Linear Support Vector

Classifier with a score of **0.929** and K Neighbors Classifier with a score of **0.825** were all applied, and the respective confusion matrices were created.

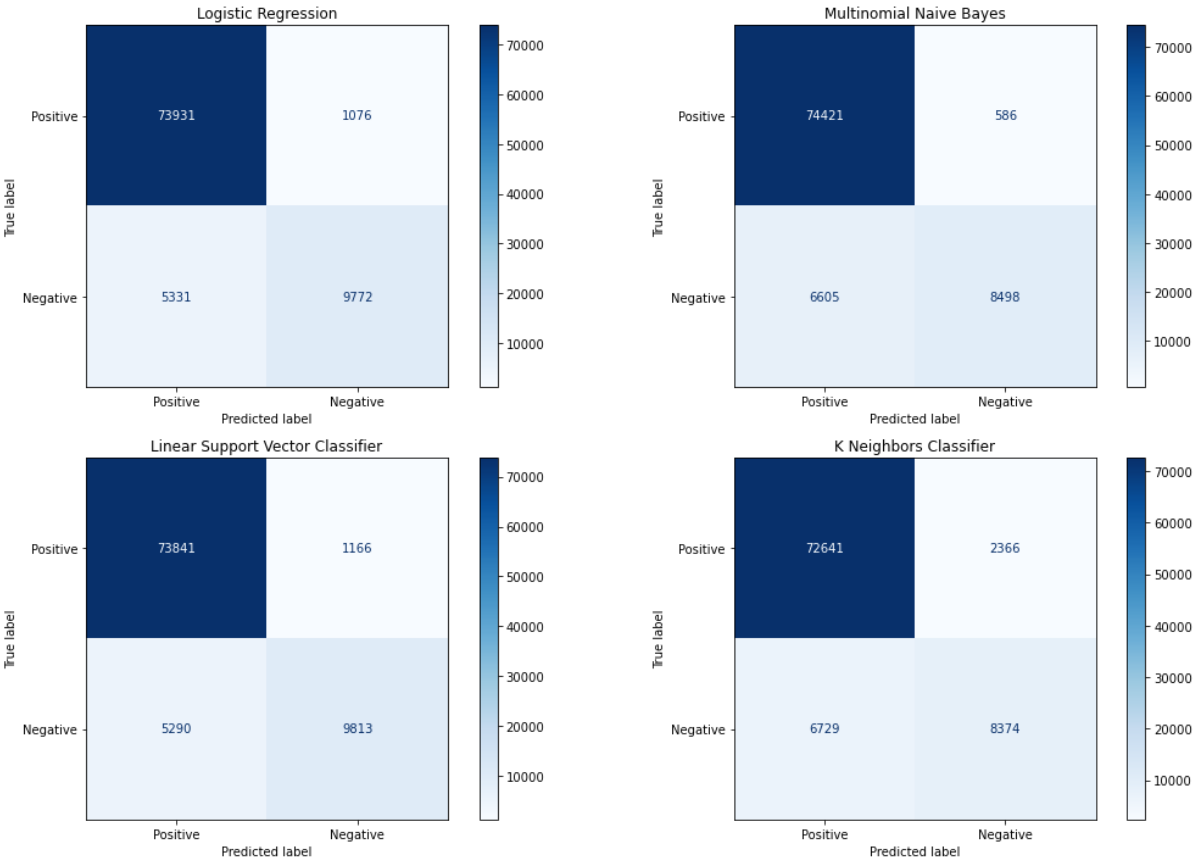


Figure 37: Machine Learning Model's Confusion Matrices

As can be concluded by the confusion matrices, *Logistic Regression*, *Support Vector Classifier* and *Multinomial Naïve Bayes* where the ones to better behave and *K Neighbors Classifier* was the model that was outperformed the most, which is self-explained if we have a look at the disadvantages of the model, since not so well representative data can obstruct the model's effectiveness. Finally, Random Forest Classifier was used, and the prediction score the method produced was also pretty high at **0.923**.

The *Naïve Bayes* model provides high bias and low variance, whereas *Logistic Regression* has low bias and higher variance and that is why in the following section Logistic Regression was applied.

This time we used Logistic Regression both for the tf idf reduced dataset, as well as the bag of words reduced dataset computed with the aforementioned method, again due to hardware limitations, and the respective confusion matrices can be found below.

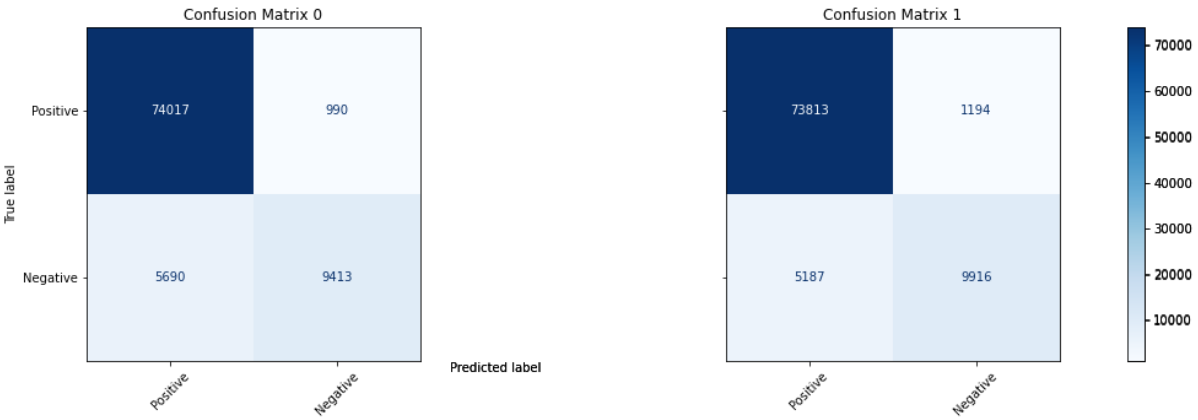


Figure 38: TF-IDF (left) and BoW (right) Confusion Matrices

As expected, the tf idf dataset had slightly better performance with a score of **0.927** than the one of bag of words with a test set score of **0.921**. Finally, the **AUC plots** can be found below for both vectorizers used in Logistic Regression as datasets.

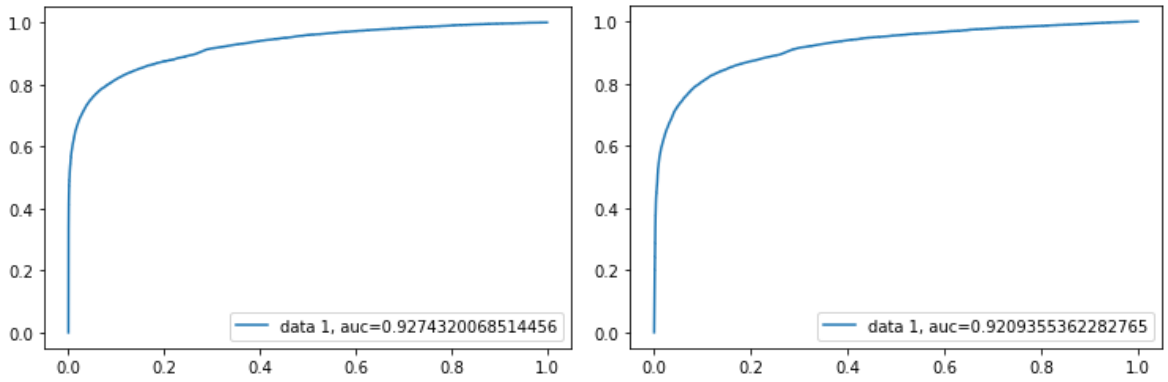


Figure 39: AUC plots for TF-IDF (left) and BoW (right)

BERT Model

The BERT model was created in a way very similar to the machine learning models. Due to hardware limitations, we had to again shuffle the data, and move on to initially separate two sub data frames; one containing all positive records and one containing all negative records. The chunk size of each data frame was initialized.

Respectively to the aforementioned technique, if the chunk size exceeded the available positive records the system retrieved as many positive records as it could find. Since we used only 100,000 records, performing augmentation was not necessary. We proceeded with the downsampling of the data due to the fact that 50,000 records were the positive

ones. It is worth mentioning, that in case of augmentation, we did not use SMOTE as it might lose some of the contextual meanings, which is possibly important for BERT and less so for other simpler models.
Another way of performing class balancing is increasing the number of samples by *random word dropout* or *synonym replacement*. In our case synonym replacement was applied using the python package nlpaug.

We created the *BERT embedding layer* by importing the BERT model from hub.KerasLayer and a BERT vocabulary file in the form of a numpy array. We then set the text to lower case and finally passed our vocabulary file and lowercase variables to the tokenizer object.
We were all set to create our model, called the function and defined our model layers. Three dense neural network layers as well as the respective dropout layers were used. We chose a learning rate of 2e-5.

Relu function was used (with the default values) that returns the element-wise maximum of 0 and the input tensor. The softmax function that converted the real Vector to a vector of categorical probabilities was used, since the elements of the output Vector were in range 0 to 1 and sum to 1. Each Vector was handled independently; the axis argument set with the axis of the input the function was applied along. Finally, categorical cross entropy which is part of binary cross entropy was used and it computes the cross entropy loss between true labels and predicted labels.

We then found the maximum length but sticked to 128 characters for each text due to the great size and again hardware limitations. We trained the model with a batch size of 32 (128*32 ~ 4000) for 2 epochs since we did not want to brainwash the model. We just wanted to give it a taste of our data because the model was already pre-trained. Also, we wanted it to respond well to data close to what we trained it to.

Finally, we saved the model to the disk in order to be able to use it in the future for our application, since the model predicted very accurately considering it is a neural network based model with a prediction accuracy of **0.91**.

Sentiment Analysis

Sentiment analysis refers to a method which detects polarity (e.g. positive or negative opinion) to text corpus. It aims to measure attitude, sentiments and emotions based on computational treatment of subjectivity in a text.

The existence of hate speech in social media has gained much attention as a topic of research. The ability to detect hate speech by analyzing data from social platforms and automatically classify their sentiment polarity has attracted researchers. In this study, we aimed to use the Valence Aware Dictionary for Sentiment Reasoner (VADER) to classify the sentiments expressed in data. We used VADER to classify comments related to Hate speech or not.

VADER is a lexicon and rule-based sentiment analysis tool that is specifically attuned to the sentiments expressed in social platforms. It is an entirely free open-source tool. VADER also takes into consideration word order and degree modifiers. A deeper explanation regarding the tool itself and its use, is explained in the section of Sentiment Analysis implementation.

For the needs of our analysis, the method isalpha() was used in order to include only the words that consisted of letters. Otherwise, our word list might have ended up with punctuation marks as the only words in the list. Moreover, NLTK provided us with nltk.word_tokenize(), a function that splits raw text into individual words. While tokenization is itself a bigger topic, this tokenizer delivered simple word lists really well. We then retrieved the top 3 most frequent words in the dataset, being the words 'like' with **168,937** occurrences, 'people' with **155,027** and 'think' with **122,488** occurrences.

Collocations can be made up of two or more words. NLTK provided us with classes to handle several types of collocations:

1. **Bigrams**: Frequent two-word combinations
2. **Trigrams**: Frequent three-word combinations
3. **Quadgrams**: Frequent four-word combinations

Bigram was used and a threshold of **0.05** in contrast to the threshold used in our app, in order to be able to test both ways to classify our results. Each sentence of our dataset was

characterized as ‘positive’, ‘neutral’ or ‘negative’ depending on its context and the results were pretty accurate with a score of **0.962**.

Conclusions

Taking into consideration all of the above, in the table below, all the models - along with their test accuracy - are presented:

Classifier	Test Accuracy (.%)
Recurrent Neural Networks (RNN)	*87.70%*
Convolution Neural Networks (CNN)	*90.60%*
Linear Support Vector Classifier	*92.50%*
Logistic Regression	*92.40%*
Multinomial Naive Bayes	*91.10%*
K-Neighbors Classifier	*80.90%*
Random Forest Classifier	*92.30%*
Bert Classifier	*91.00%*

Table 7: Classifiers' Test Accuracy

From the above table, it appears that the Linear Support Vector Classifier has the greatest test accuracy among all the classifiers. Logistic Regression comes in second place. Having in mind that the problem investigated in the particular project is a binary problem (0: positive/neutral, 1: negative), the Logistic Regression model fits better, thus it was chosen for further analysis and conclusions. It is important to mention that even though Logistic Regression is more vulnerable to overfitting (compared to Support Vector Classifier), it is based on a statistical approach in opposition to SVC which is based on geometrical properties of the data. In addition, SVM tries to find the "best" margin (distance between the line and the support vectors) that separates the classes, while logistic regression can have different decision boundaries with different weights that are near the optimal point.

As a result, the Logistic Regression model fits better to our needs, and can be considered as the most optimal solution for our business case.

hateless application

hateless is a sentiment analysis application designed for analyzing comments from various platforms through the internet and identifying if they are offensive or not. The user logs in the application and then writes the corresponding comment that he/she wants to analyze in the respective placeholder and by pressing the '*Analyze*' button, starts the analysis. Indicative screenshots are represented below:

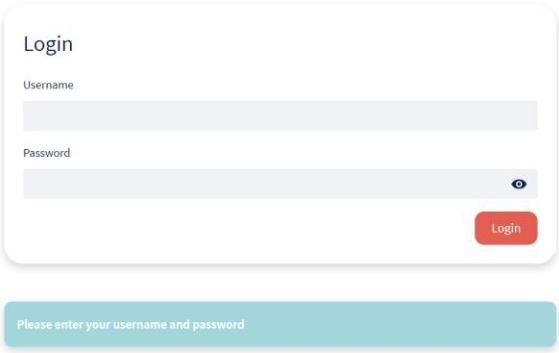
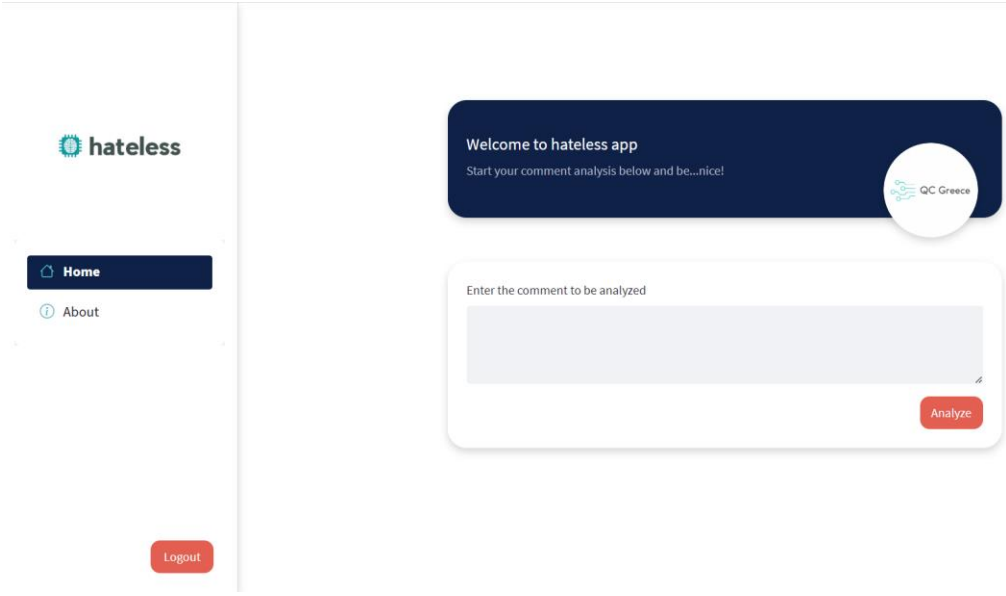
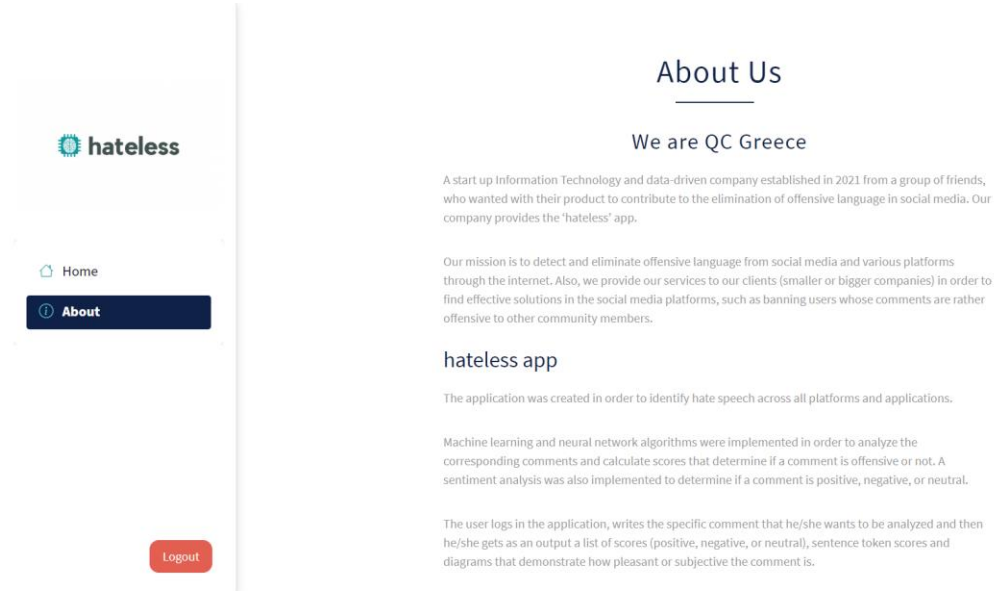


Figure 40: hateless app login page


hateless
Home
About
Logout
Welcome to hateless app
Start your comment analysis below and be...nice!
QC Greece
Enter the comment to be analyzed
Analyze


hateless
Home
About
Logout
About Us
We are QC Greece
A start up Information Technology and data-driven company established in 2021 from a group of friends, who wanted with their product to contribute to the elimination of offensive language in social media. Our company provides the 'hateless' app.
Our mission is to detect and eliminate offensive language from social media and various platforms through the internet. Also, we provide our services to our clients (smaller or bigger companies) in order to find effective solutions in the social media platforms, such as banning users whose comments are rather offensive to other community members.
hateless app
The application was created in order to identify hate speech across all platforms and applications.
Machine learning and neural network algorithms were implemented in order to analyze the corresponding comments and calculate scores that determine if a comment is offensive or not. A sentiment analysis was also implemented to determine if a comment is positive, negative, or neutral.
The user logs in the application, writes the specific comment that he/she wants to be analyzed and then he/she gets as an output a list of scores (positive, negative, or neutral), sentence token scores and diagrams that demonstrate how pleasant or subjective the comment is.

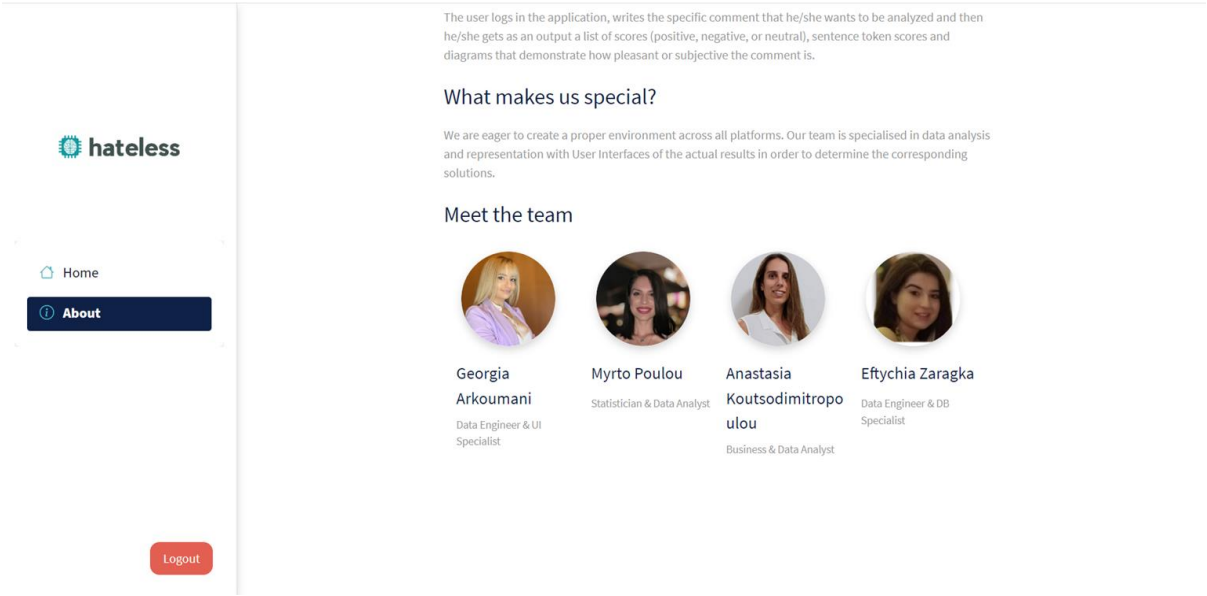


Figure 41: hateless app - home page & About Us page

The output of the analysis consists of 3 different sections: ***Sentiment metrics***, ***Sentiment token metrics*** & ***Sentiment metrics visualization***.

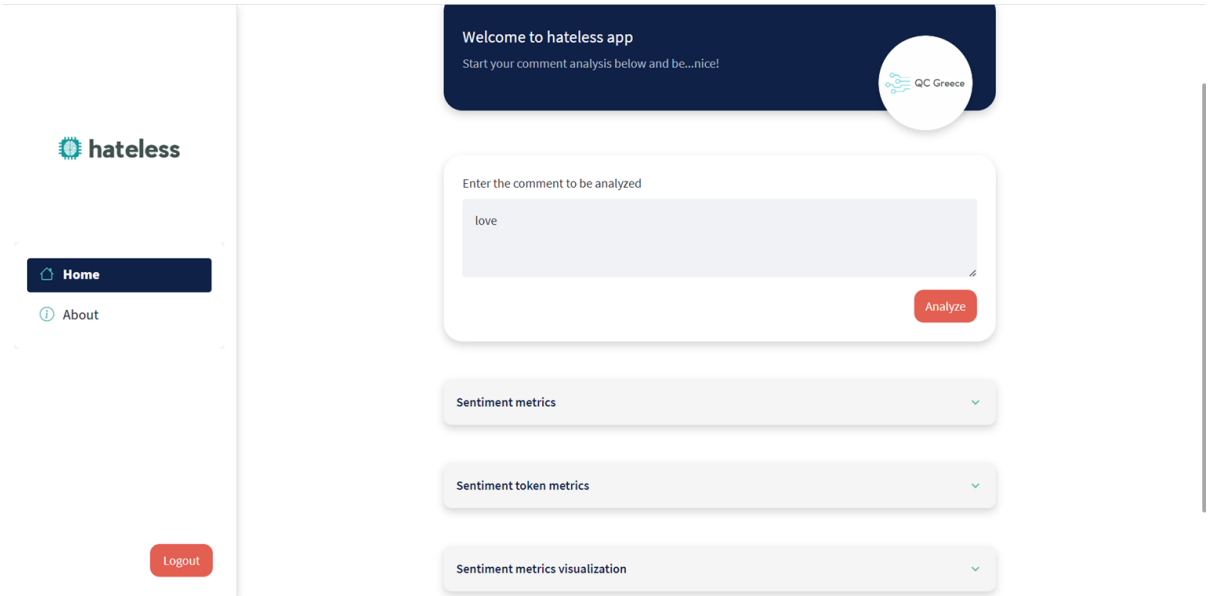


Figure 42: hateless app comment analysis output

In the *first* section, two basic sentiment metrics are represented: polarity (lies between [-1,1], -1 defines a negative sentiment and 1 defines a positive sentiment, negative words reverse the polarity) & subjectivity (quantifies the amount of personal opinion and factual information contained in the text, the higher subjectivity means that the text contains

personal opinion rather than factual information) along with the overall sentiment that the comment provokes to a user with labels: Positive, Negative or Neutral.

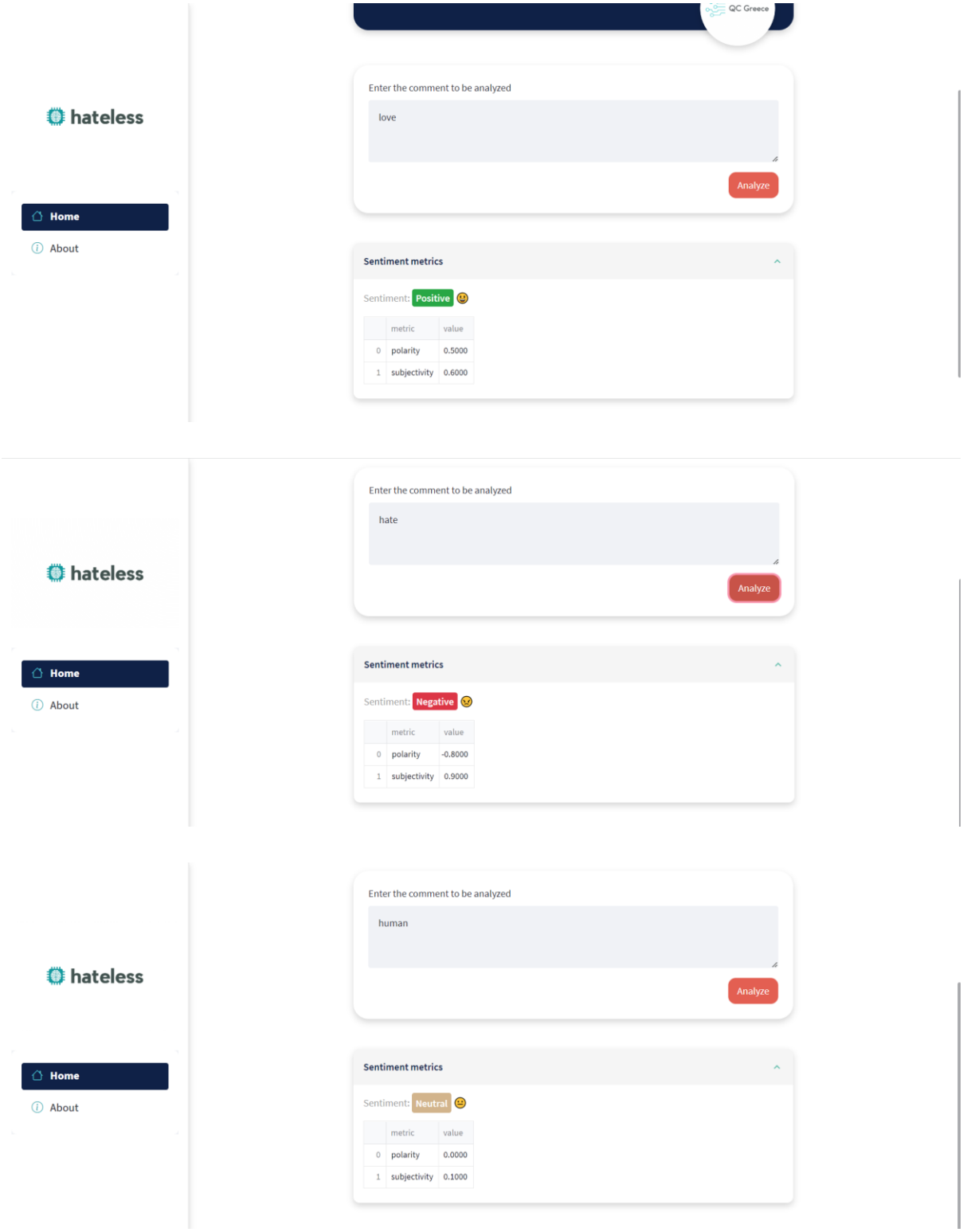


Figure 43: hateless app - Sentiment metrics output (Positive, Negative & Neutral)

In the *second* section, using *SentimentIntensityAnalyzer* from VADER, 3 lists (positives, negatives & neutrals) with sentence tokens are represented, based on their polarity scores. A threshold of 0.1 has been chosen in order to categorize the tokens of the sentence. If the score is > 0.1, then the token will be moved to the positive list, if the score is <= -0.1, then it will be moved to the negative list and otherwise to the neutral list.

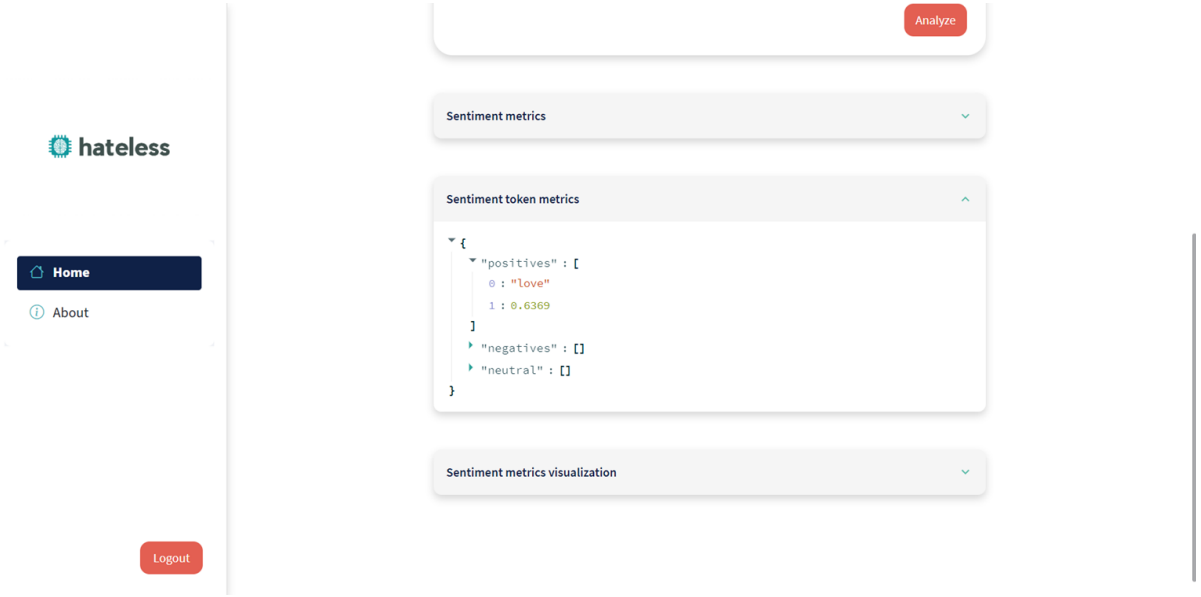


Figure 44: hateless app - Sentiment token metrics output

In the *third* section, a bar chart is represented with the 2 basic metrics mentioned above: **polarity** & **subjectivity**. Also, a variety of actions are available in this section e.g. zoom-in, zoom-out, download the graph etc.

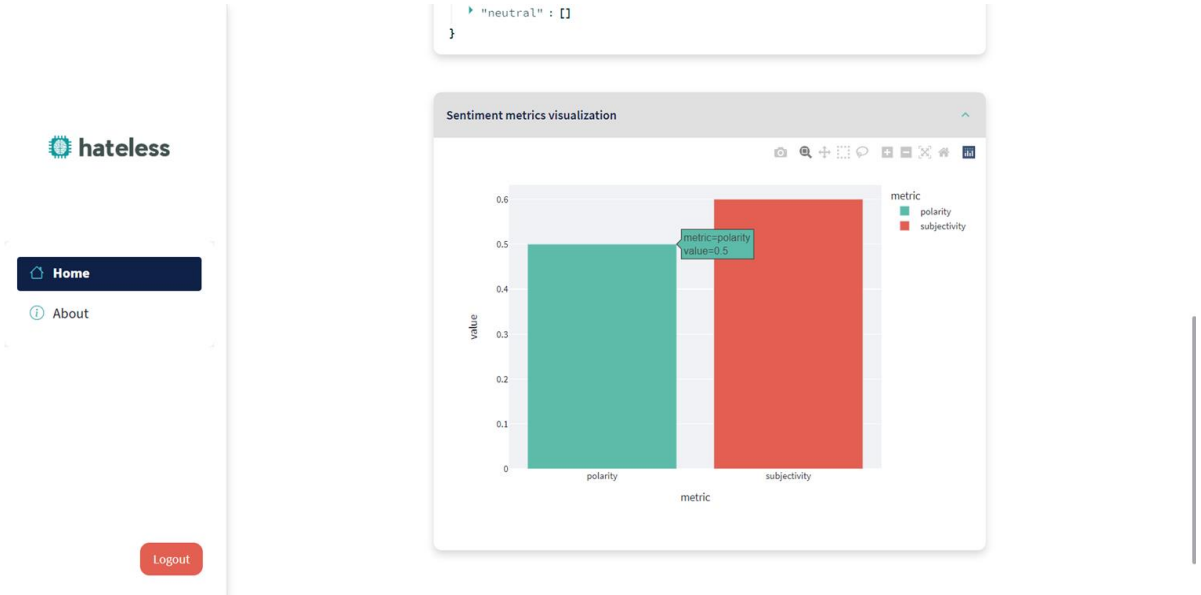


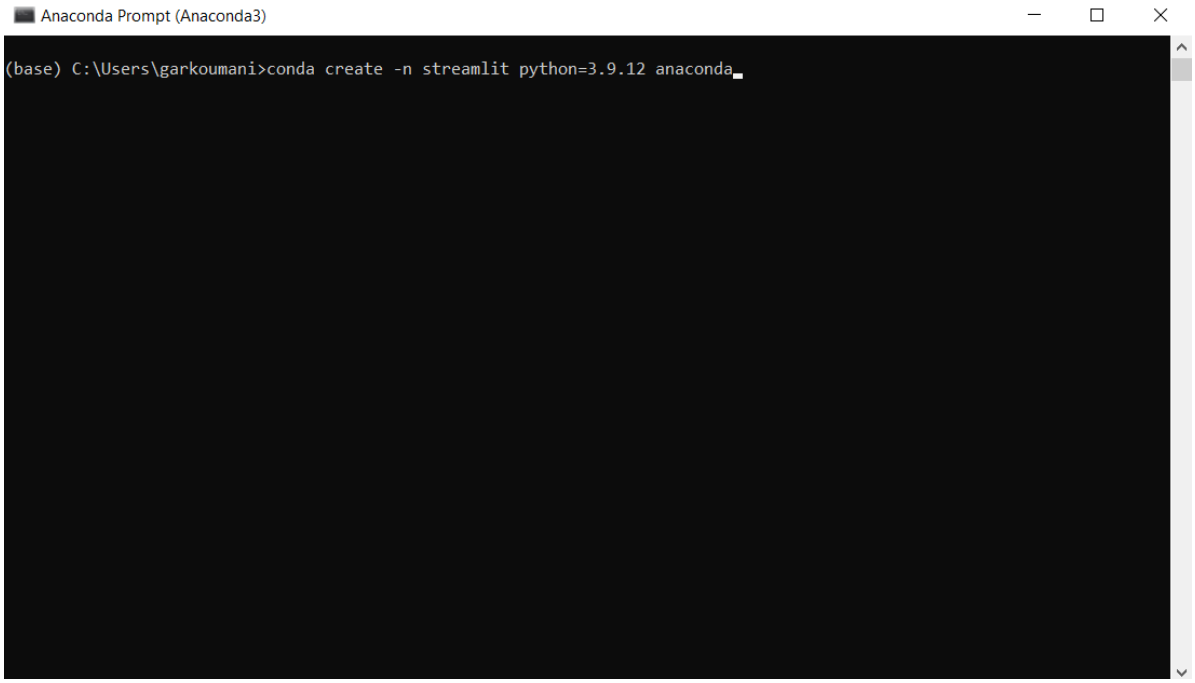
Figure 45: hateless app - Sentiment metrics visualization output

Streamlit configuration

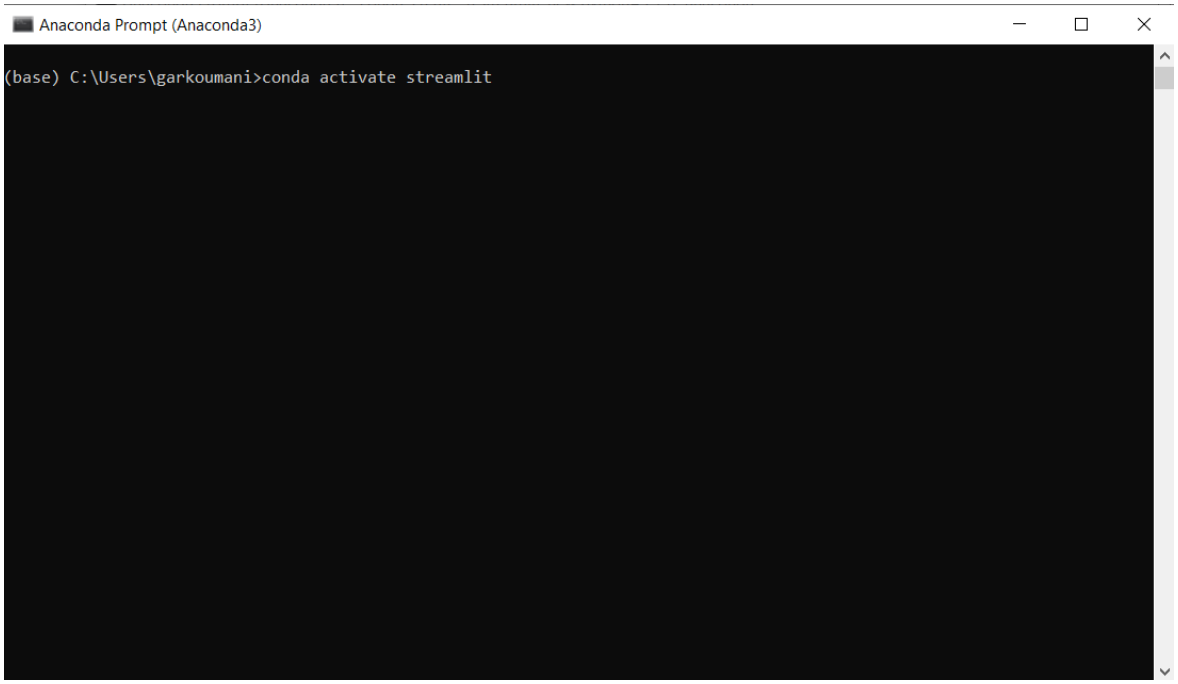
Streamlit is an open-source app framework for Machine Learning and Data Science teams that lets you turn data scripts into shareable web apps using Python. Once the app is created, you can use the cloud platform to deploy, manage, and share it.

For the needs of the aforementioned case for identifying hate and offensive language, Streamlit was a prerequisite in order to build a User Interface that helps the user analyze any preferable comments.

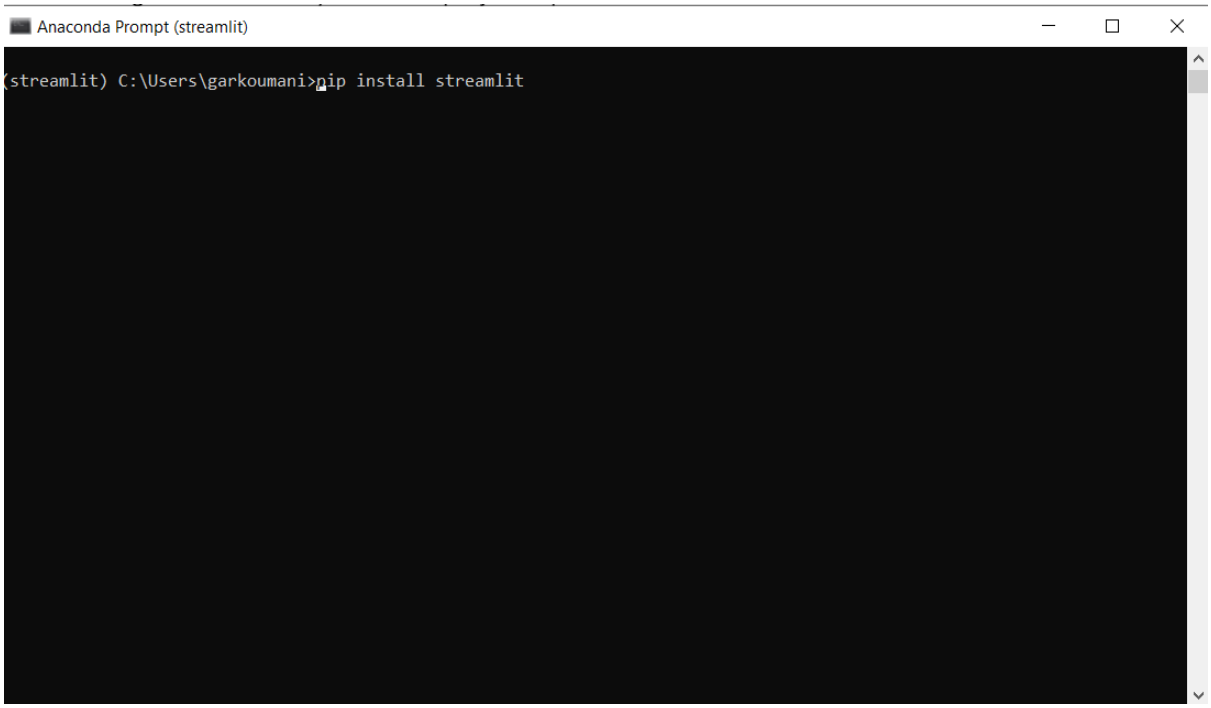
The prerequisites for the Streamlit working environment setup was to create a conda environment, after the installation of Anaconda. For the needs of this application Python 3.9.12 version was used.



As a next step, the new 'streamlit' environment was activated.



After the completion of the aforementioned steps, Streamlit & all the required libraries and packages were installed.



As a final step, the environment was used to run the application.

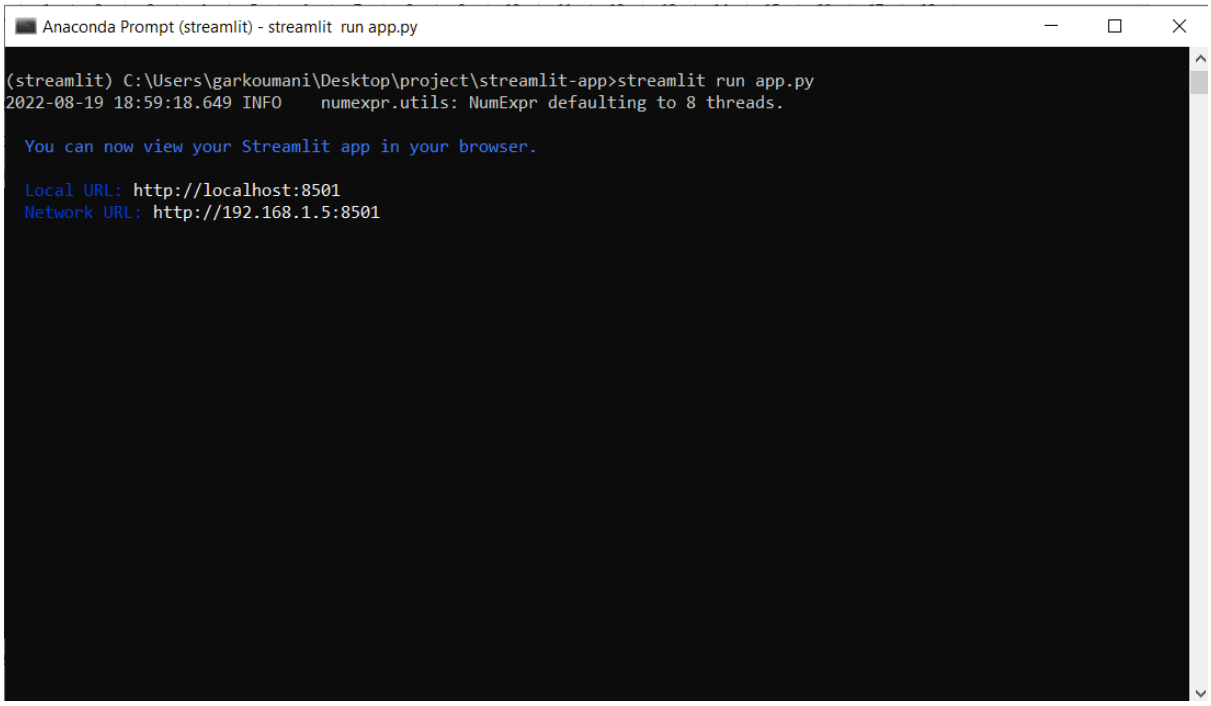


Figure 46: Anaconda Prompt - Create/ Activate environment & Run the application

The application ran successfully on http://localhost:8501/.

Application design & implementation

Sentiment Analysis implementation

As per the implementation of the app, *Streamlit* environment was used with *Python*.
In order to proceed with the sentiment analysis, TextBlob (a library for processing textual data that provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, etc) & vaderSentiment (VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media).

As far as *TextBlob* is concerned, it can be used for complex analysis and working with textual data. When a sentence is passed into Textblob it gives two outputs, which are polarity and subjectivity. Polarity is the output that lies between [-1,1], where -1 refers to negative sentiment and +1 refers to positive sentiment. Subjectivity is the output that lies within [0,1] and refers to personal opinions and judgments.

Regarding *VADER*, it uses a combination of a sentiment lexicon -a list of lexical features (e.g., words) which are generally labeled according to their semantic orientation as either positive or negative- and not only tells us about the positivity and negativity score but also how positive or negative a sentiment is. Moreover, it calculates the *compound score*, a metric that calculates the sum of all the lexicon ratings which have been normalized between -1(most extreme negative) and +1 (most extreme positive). The aforementioned score was used in order to calculate the ***Sentiment token metrics*** that were mentioned in the previous section.

User Interface design & implementation

As per the design of the app, Atomic Design Methodology (a methodology composed of five distinct stages Atoms, Molecules, Organisms, Templates, Pages– working together to create interface design systems in a more deliberate and hierarchical manner) was used for creating a structural, hierarchical and reusable component-based application. As per technologies and frameworks, HTML, CSS and Bootstrap (v4) were used along with all the required packages and libraries needed. In addition, basic streamlit components (e.g. sidebar, markdown, form, title, expander etc) were used in combination with bootstrap.

User Authentication Process

In order to create the applications' login page, Streamlit-Authenticator was used as a secure authentication module to validate user credentials in a Streamlit application. Initially, users' names, usernames, and plain text passwords were defined and then a helper file '**generate_keys.py**' was created using a hasher module to convert the plain text passwords to hashed passwords and store them into the file '**hashed_pw.pkl**'. The aforementioned actions were performed using Python's pickle module.

Subsequently, the hashed passwords were used to create an authentication object that consists of a name for the JWT cookie that will be stored on the client's browser and used to re-authenticate the user without re-entering their credentials. In addition, a random key is provided to be used to hash the cookies' signature as well as the number of days to use the cookie for.

Then, in order to render the login module a name for the login form is provided, and was specified where the form should be located i.e. main body or sidebar (with default to main

body). Finally, the returned name and authentication status allow the verified user to proceed to any restricted content by checking for any missing or wrong passwords.

Indicative screenshots with the login form are shown below.

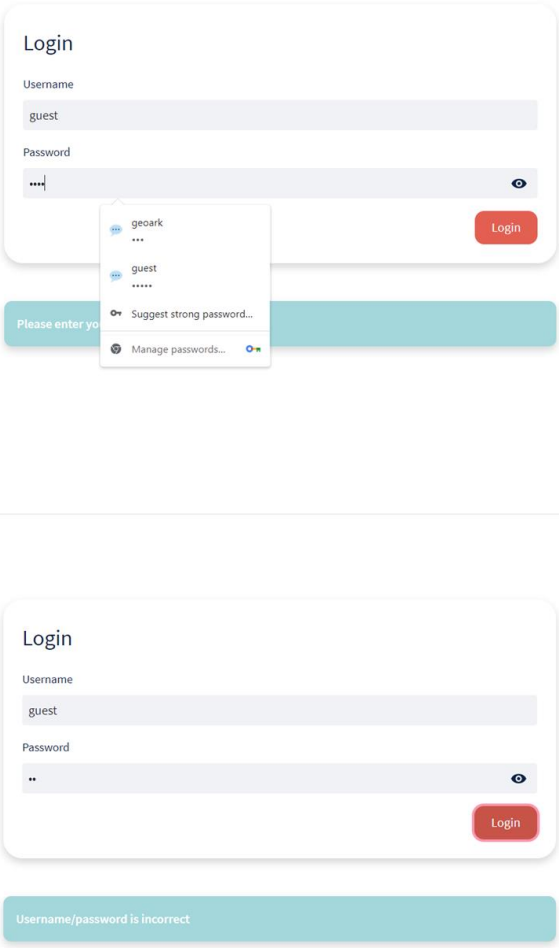


Figure 47: hateless app - Entering user credentials

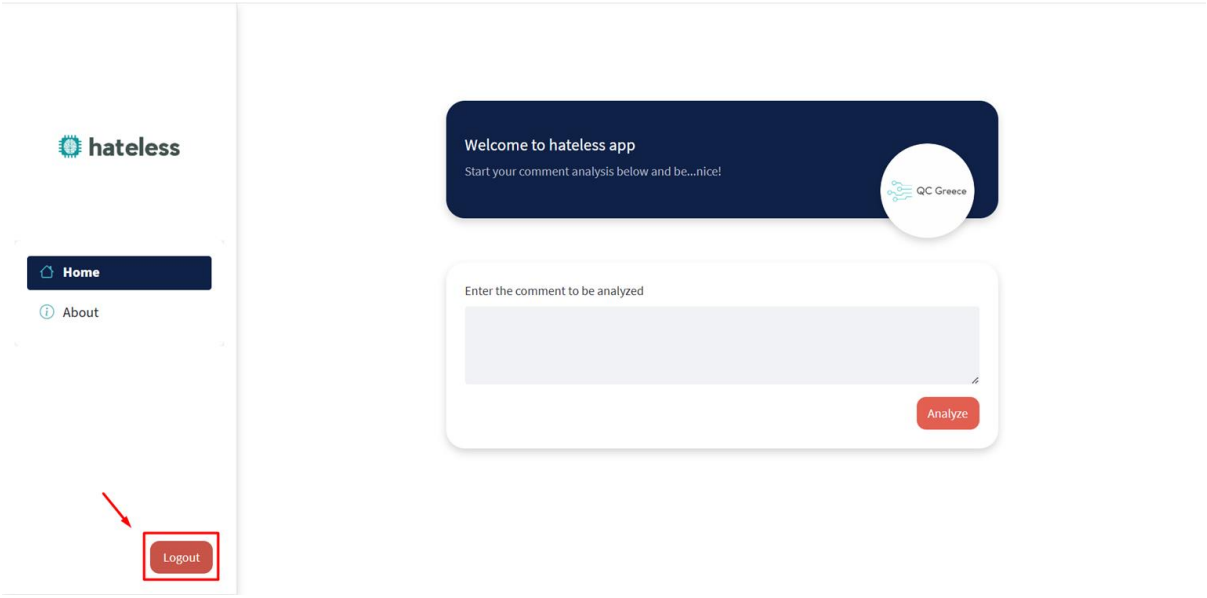


Figure 48: hateless app - Logout in sidebar

Streamlit Cloud

Streamlit applications can be deployed in Streamlit website for free. After the upload of the code on GitHub, a sign in https://share.streamlit.io/ was implemented, connected with the GitHub account and created a new app.

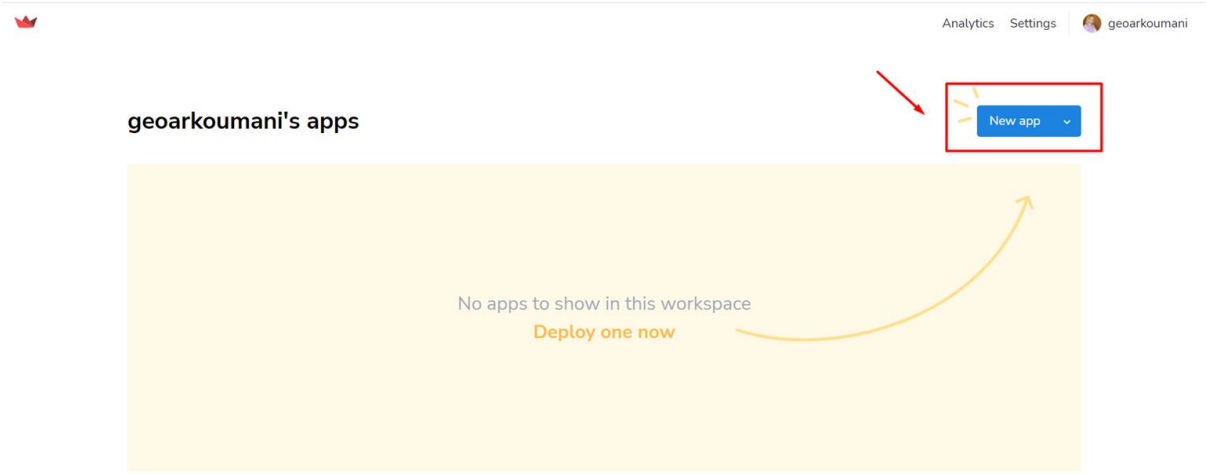


Figure 49: Streamlit Cloud - Create new application

Prerequisite for deploying the app is the creation and addition to the GitHub repository of **requirements.txt** file, containing all the required libraries for the application. Then, GitHub repository and streamlit are connected.

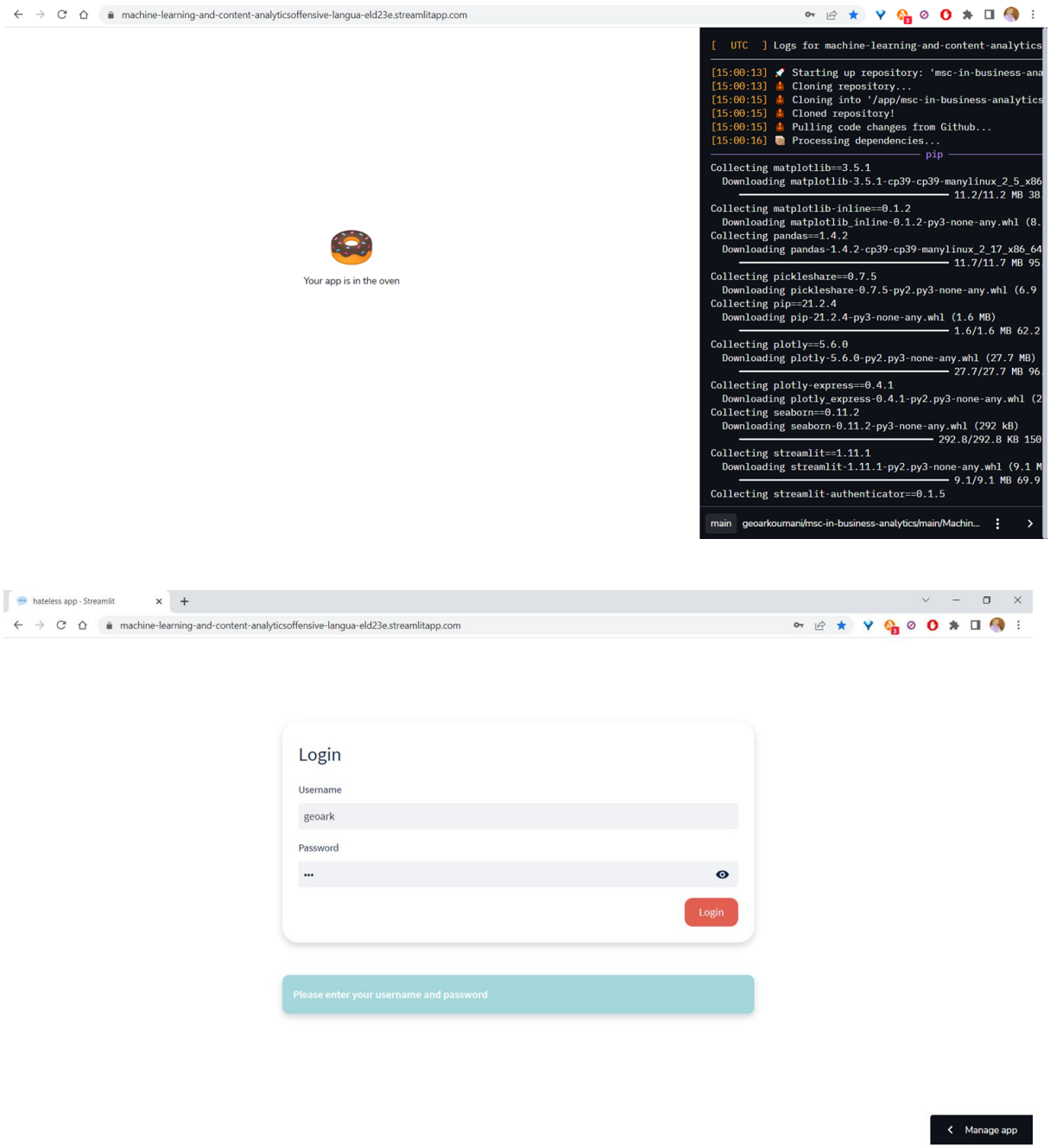


Figure 50: Streamlit Cloud - Deploy application

The application was successfully deployed and can be accessed through Streamlit here.

The application is on a bug fixing mode regarding the deployment phase and some corrective actions via *Streamlit Cloud* will be performed before the production stage.

28/08/2022, Athens

The universal credentials for logging in to the application are: username: guest & password: test.

Discussion & Future Work

Vast online communication forums, including social media, enable users to express themselves freely, at times, anonymously. While the ability to freely express oneself is a human right that should be cherished, inducing and spreading hate towards another group is an abuse of this liberty.

After the aforementioned research and training of different models, we identify and examine challenges faced by various approaches for hate speech detection in texts. Among these difficulties are misspellings, human errors while writing a text/comment, autocorrections and in general imbalanced data that can make detecting hate speech a challenging task. The computational cost of the algorithms used to train the models should also be mentioned as a major difficulty.

However, there are also disagreements on how hate speech should be defined. This means that some content can be considered hate speech to some and not to others, based on their respective definitions. It is noticeable that accuracy has been one of the main offensive language recognition challenges for many years – and a barrier to entry for many businesses.

Our company has set its targets for the future; we are planning to manage the tasks presented below:

COMPANY WEBSITE

We will build a company website with all the available information and financial packages and plans for our customers. In addition to that, social media accounts will also be constructed for supporting the advertising part.

WIDER TARGET GROUP

Our company will offer consulting services by helping businesses understand if their products are engaging or provoke a positive sentiment to the user i.e. analyzing the comments of an application (e.g. food delivery application) and assume if the customers

are happy with the services that the corresponding application provides. Thus, we aim to a wider target group.

AUTOCORRECT ALGORITHM

Our goal for the ongoing time is to build a personalized software which will check each word based on a variety of built-in dictionaries (adjusted to specific purposes & needs), replacing mis-spelled/ incorrect words. If matches cannot be found, alternatives will be used based on the overall meaning of the given comments. At the same time, we aim for an algorithm that automatically will handle large volumes of data by improving all kinds of keyboard-dependent and context-based functions targeting a quick implementation & experience for our work going forward.

NEURAL NETWORKS IMPROVEMENTS

An important goal to achieve in the near future is to overcome the variation in the validation data. More precisely to RNN and CNN models, since there is overfitting in the data, we would like to further experiment with the models' configuration in order to overcome the issues. The first aspects which will be explored are the increase of the Dropout and the batch size. Another improvement that our team wants to achieve, is to investigate further the root cause of the issues introduced in the Confusion Matrix and Classification Report, and fix the issues in a future release.

INTRODUCING NEW FINANCIAL PACKAGES

As it is already described in detail in our Business Case, we include three subscriptions and packages for companies but we will manage to add more for single users in order to add more capabilities for them through the application. The packages concerning industries and businesses may differ in the future because of the addition of new features.

BUILDING A USER BASED APP EXTENSION

As per application, our aim for the future is to add a *Registration page* with a *Forgot Password* functionality and provide the user the capability of connecting social media accounts; to keep track of the comments on his/her posts and videos. In that case, the application will consist of a new *Dashboard page* with charts that will demonstrate to the user the total amount of how many comments are positive/negative or neutral in the respective account. Based on the aforementioned plan, there is a discussion of adding plugins of other applications which provide to the individual the ability to unfollow each user who writes inappropriate and offensive comments to his account.

Regarding the design of the application, there will be an update using React & MUI systems.

As per implementation, the application is on a bug fixing mode regarding the deployment phase and some corrective actions will be performed before the production stage. Moreover, another target for the future will be to load to the application more pretrained ML and NN models in order to obtain more accurate results.

FUTURE GOALS

Our company plans to utilize its propositions and achieve the aforementioned goals in a period of 2 to 5 years in order to increase its growth and profits.

28/08/2022, Athens

Team & Timeplan

Our team members are represented below:

Georgia Arkoumani

Data Engineer & UI Specialist

Myrto Poulou

Statistician & Data Analyst

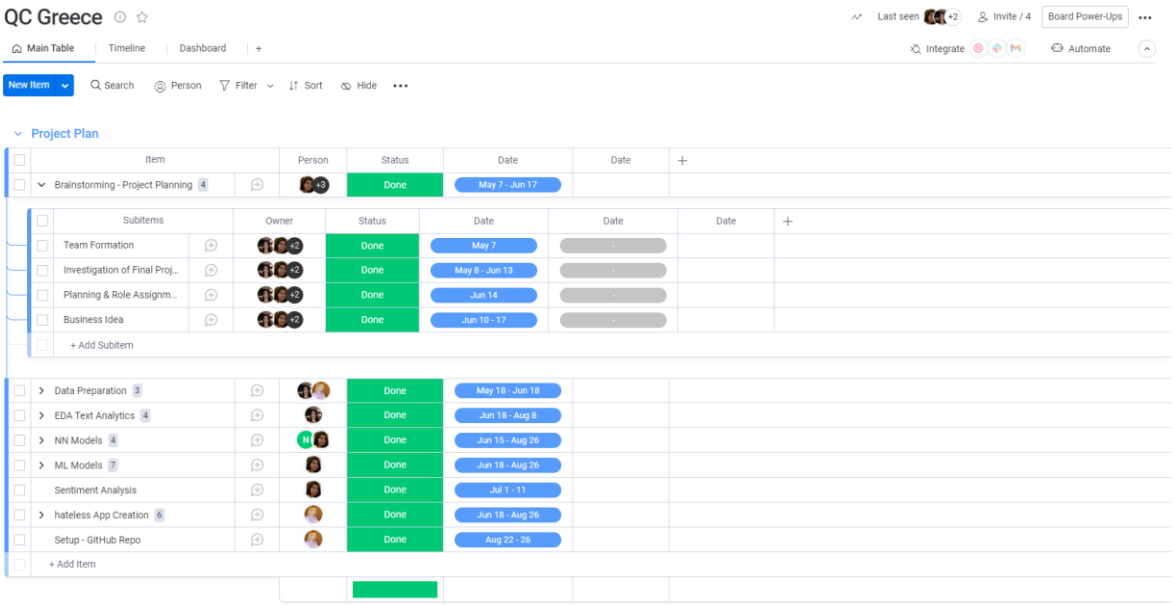
Anastasia Koutsodimitropoulou

Business & Data Analyst

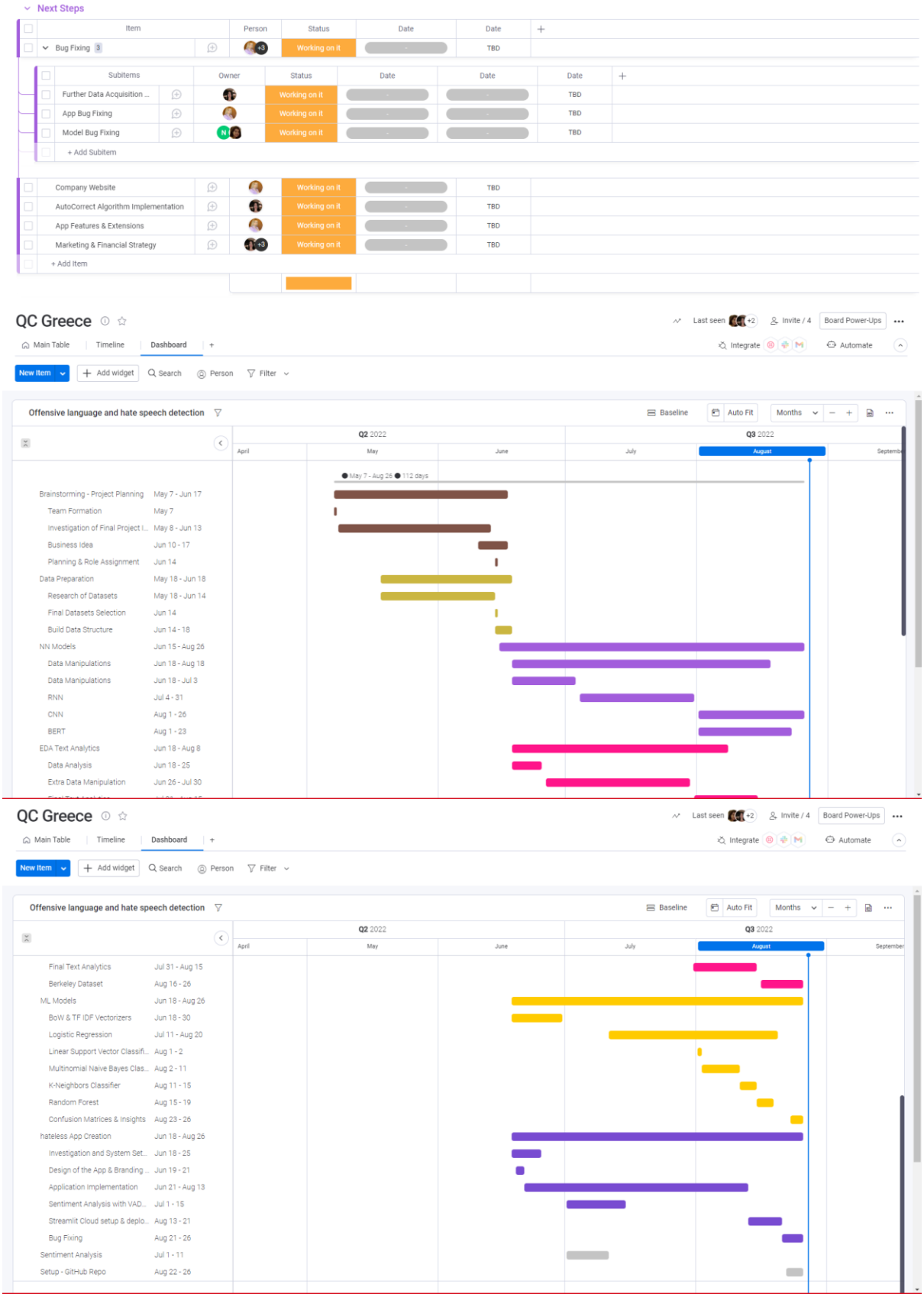
Eftychia Zaragka

Data Engineer & DB Specialist

As per projects' timeplan, a detailed representation is demonstrated below:



28/08/2022, Athens


Next Steps
Item
Person
Status
Date
Date
Bug Fixing 3
Working on it
TBD
Subitems
Owner
Status
Date
Date
Date
Further Data Acquisition ...
App Bug Fixing
Model Bug Fixing
+ Add Subitem
Company Website
AutoCorrect Algorithm Implementation
App Features & Extensions
Marketing & Financial Strategy
+ Add Item
QC Greece
Last seen
Invite / 4
Board Power-Ups
Main Table
Timeline
Dashboard
Integrate
Automate
New Item
Add widget
Search
Person
Filter
Offensive language and hate speech detection
Baseline
Auto Fit
Months
Q2 2022
Q3 2022
April
May
June
July
August
September
May 7 - Aug 26 112 days
Brainstorming - Project Planning May 7 - Jun 17
Team Formation May 7
Investigation of Final Project I... May 8 - Jun 13
Business Idea Jun 10 - 17
Planning & Role Assignment Jun 14
Data Preparation May 18 - Jun 18
Research of Datasets May 18 - Jun 14
Final Datasets Selection Jun 14
Build Data Structure Jun 14 - 18
NN Models Jun 15 - Aug 26
Data Manipulations Jun 18 - Aug 18
Data Manipulations Jun 18 - Jul 3
RNN Jul 4 - 31
CNN Aug 1 - 26
BERT Aug 1 - 23
EDA Text Analytics Jun 18 - Aug 8
Data Analysis Jun 18 - 25
Extra Data Manipulation Jun 26 - Jul 30
QC Greece
Last seen
Invite / 4
Board Power-Ups
Main Table
Timeline
Dashboard
Integrate
Automate
New Item
Add widget
Search
Person
Filter
Offensive language and hate speech detection
Baseline
Auto Fit
Months
Q2 2022
Q3 2022
April
May
June
July
August
September
Final Text Analytics Jul 31 - Aug 15
Berkeley Dataset Aug 16 - 26
ML Models Jun 18 - Aug 26
BoW & TF IDF Vectorizers Jun 18 - 30
Logistic Regression Jul 11 - Aug 20
Linear Support Vector Classifi... Aug 1 - 2
Multinomial Naive Bayes Clas... Aug 2 - 11
K-Neighbors Classifier Aug 11 - 15
Random Forest Aug 15 - 19
Confusion Matrices & Insights Aug 23 - 26
hateless App Creation Jun 18 - Aug 26
Investigation and System Set... Jun 18 - 25
Design of the App & Branding ... Jun 19 - 21
Application Implementation Jun 21 - Aug 13
Sentiment Analysis with VAD... Jul 1 - 15
Streamlit Cloud setup & deplo... Aug 13 - 21
Bug Fixing Aug 21 - 26
Sentiment Analysis Jul 1 - 11
Setup - GitHub Repo Aug 22 - 26

The aforementioned time plan was created via the monday app; a platform providing tools for streamlining business processes and tracking a projects' time plan. A shareable link of the apps' Dashboard is provided here.

Appendix

Bibliography

Offensive Language and Hate Speech Detection with Deep Learning and Transfer Learning, Cornell University, 2021 (https://arxiv.org/abs/2108.03305)

Neural Models for Offensive Language Detection, Ehab Hamdy, 2021 (https://arxiv.org/pdf/2106.14609.pdf)

Constructing interval variables via faceted Rasch measurement and multitask deep learning: a hate speech application, Kennedy, Chris J and Bacon, Geoff and Sahn, Alexander and von Vacano, Claudia, 2020 (https://arxiv.org/abs/2009.10277)

Berkeley dataset (https://huggingface.co/datasets/ucberkeley-dlab/measuring-hate-speech)

ETHOS: an Online Hate Speech Detection Dataset, Ioannis Mollas and Zoe Chrysopoulou and Stamatis Karlos and Grigorios Tsoumakas, 2020 (https://arxiv.org/abs/2006.08328)

ETHOS: an Online Hate Speech Detection Dataset (https://github.com/intelligence-csd-auth-gr/Ethos-Hate-Speech-Dataset) (https://huggingface.co/datasets/ethos)

Hate Speech Detection datasets (https://paperswithcode.com/task/hate-speech-detection?fbclid=IwAR2pDpp5Vt6K0yL1XdtXcSyiYYpzPmv6VrenX640SkgBeUC68ilHmrUknoQ)

Measuring Hate Speech (https://hatespeech.berkeley.edu/?fbclid=IwAR1ZDZ4MiOT5VPP_A0tcAuSxTpVgxZLmFnHMb8ajU11Fwcmk7GCr5Xxxy2A)

Toxic Comment Classification Challenge (https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge/overview)

28/08/2022, Athens

All-in-One Dataset (https://www.kaggle.com/code/adldotori/all-in-one-dataset/data)

ICWSM18 dataset (https://www.icwsm.org/2018/datasets/datasets/)
CONAN dataset (https://paperswithcode.com/dataset/conan)
(https://github.com/marcoguerini/CONAN)

Automated Hate Speech Detection and the Problem of Offensive Language in Proceedings of the 11th International AAAI Conference on Web and Social Media, Davidson, Thomas and Warmsley, Dana and Macy, Michael and Weber, Ingmar, 2017 (https://www.icwsm.org/2017/) (https://github.com/t-davidson/hate-speech-and-offensive-language)

Davidson dataset (https://huggingface.co/datasets/hate_speech_offensive)
(https://data.world/thomasrdavidson/hate-speech-and-offensive-language)

Developing an online hate classifier for multiple social media platforms, Joni Salminen, Maximilian Hopf, Shammur A. Chowdhury, Soon-gyo Jung, Hind Almerekhi & Bernard J. Jansen, 2020 (https://hcis-journal.springeropen.com/articles/10.1186/s13673-019-0205-6)

Hatebase (https://hatebase.org/)

hatespeechdata (https://hatespeechdata.com/)

Getting Started with Sentiment Analysis using Python, Federico Pascual, 2022 (https://huggingface.co/blog/sentiment-analysis-python)

Text pre-processing: Stop words removal using different libraries, Chetna Khanna, 2021 (https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a)

Word embeddings (https://www.tensorflow.org/text/guide/word_embeddings)

TextBlob (https://textblob.readthedocs.io/en/dev/)

28/08/2022, Athens

Autocorrect (https://github.com/filyp/autocorrect)

SpellChecker (https://pypi.org/project/pyspellchecker/)

Textstat (https://pypi.org/project/textstat/)

Readability searches (https://www.kaggle.com/code/ruchi798/commonlit-readability-prize-eda-baseline/notebook
&https://www.kaggle.com/c/commonlitreadabilityprize/discussion/236626)

sklearn.feature_extraction.text (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)
Streamlit (https://streamlit.io/)

Deploy streamlit application
(https://docs.streamlit.io/streamlit-cloud/get-started/deploy-an-app)

How to Add a User Authentication Service in Streamlit,M Khorasani,2021
(https://towardsdatascience.com/how-to-add-a-user-authentication-service-in-streamlit-a8b93bf02031)
(https://github.com/mkhorasani/Streamlit-Authenticator)

JWT Tokens (https://jwt.io/)

Overcoming Class Imbalance using SMOTE Techniques, SWASTIK SATPATHY, 2020
(https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/)

Recurrent Neural Networks cheatsheet, Afshine Amidi, Shervine Amidi
(https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks)

28/08/2022, Athens

Importance of Tokenization in Data Engineering, Shreyak, 2020 (https://medium.com/blocksurvey/importance-of-tokenization-in-data-engineering-2d5b31aab98b)

Time Series - LSTM Model, 2022 (https://www.tutorialspoint.com/time_series/time_series_lstm_model.htm)

Report on Text Classification using CNN, RNN & HAN, Akshat Maheshwari, 2018 (https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f)

Keras model.summary() result - Understanding the # of Parameters (https://stackoverflow.com/questions/36946671/keras-model-summary-result-understanding-the-of-parameters)

Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, Jason Brownlee, 2017 (https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/)
Publication of hate speech (https://rm.coe.int/models-of-governance-of-online-hate-speech/16809e671d)
Go Green Initiative (https://gogreeninitiative.org/about/)

17 Goals (https://sdgs.un.org/goals#goals)
AutoCorrect / Spell Checker (https://techcrunch.com/2015/02/11/big-data-on-the-keyboard/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmdyLw&guce_referrer_sig=AQAAAEvw1ZqcLFRBSApLmVM_mHoAEDA-Pbhlq8obtPuc4vw0ChyjLgxZCoIJeaTds4BlPL6aCOp65cTMYzL5zbD-4WpS_1cQPrhNg46QHpUl_fdaTjSLAheNiK222StpvwwyWehwFZgnSsfesRC19SGUWnzBcuw4jCYAdB9FhRYopGfO)
(https://www.researchgate.net/publication/330109219_Autocomplete_and_Spell_Checking_Levenshtein_Distance_Algorithm_To_Getting_Text_Suggest_Error_Data_Searching_In_Library)
(https://python-bloggers.com/2022/02/spelling-checker-program-in-python/)

28/08/2022, Athens

Information provided by the IT companies about measures taken to counter hate speech, including their actions to automatically detect content, EU Justice and Consumers, 2021 (https://ec.europa.eu/info/sites/default/files/information-provided-by-the-it-companies-about-measures-taken-to-counter-hate-speech_october2021_en.pdf)

Hate Speech Policy of YouTube, YouTube Help (https://support.google.com/youtube/answer/2801939)

Facebook Community Standards, Meta - Transparency Center (https://transparency.fb.com/el-gr/policies/community-standards/?source=https%3A%2F%2Fwww.facebook.com%2Fcommunitystandards%2Fobjectionable_content)

Hateful Conduct Policy, Twitter Help Center (https://help.twitter.com/en/rules-and-policies/hateful-conduct-policy)

A Measurement Study of Hate Speech in Social Media,Mondal M, Silva LA, Benevenuto F.,2017 (https://dl.acm.org/doi/abs/10.1145/3078714.3078723)

A Survey on Automatic Detection of Hate Speech in Text,Fortuna P, Nunes S., 2018 (https://dl.acm.org/doi/10.1145/3232676)

Are You a Racist or Am I Seeing Things? Annotator Influence on Hate Speech Detection on Twitter,Zeerak Waseem,2016 (https://aclanthology.org/W16-5618.pdf)

Hate speech detection: Challenges and solutions, Sean MacAvaney, Hao-Ren Yao, Eugene Yang, Katina Russell, Nazli Goharian, Ophir Frieder, 2019 (https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0221152#ack)

Comprehensive Hands on Guide to Twitter Sentiment Analysis with dataset and code, Prateek Joshi, 2018 (https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/)

Tune Hyperparameters with GridSearchCV, Rahul Shah, 2021 (https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/)

Basics of CountVectorizer, Pratyaksh Jain, 2021 (https://towardsdatascience.com/basics-of-countvectorizer-e26677900f9c)

GridSearchCV for Beginners, Scott Okamura, 2020 (https://towardsdatascience.com/gridsearchcv-for-beginners-db48a90114ee)

5 Natural language processing: classifying social media sentiment (https://livebook.manning.com/book/feature-engineering-bookcamp/chapter-5/v-2/1)

A Simple Walkthrough With Sci-kit Learn's Pipeline, Christopher Lewis, 2021 (https://medium.com/analytics-vidhya/a-simple-walkthrough-with-sci-kit-learns-pipeline-46cdf6e53354)

Applied Text Analysis with Python, Benjamin Bengfort, Rebecca Bilbro, Tony Ojeda (https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/ch04.html)

Support Vector Machines with Scikit-learn Tutorial (https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python)

Hyperparameter Tuning with GridSearchCV, Great Learning Team, 2020 (https://www.mygreatlearning.com/blog/gridsearchcv/)

bert-sklearn (https://github.com/junwang4/bert-sklearn-old)

Tutorial: Fine tuning BERT for Sentiment Analysis, Chris Tran (https://skimai.com/fine-tuning-bert-for-sentiment-analysis/)

Twitter Sentiment Analysis with Deep Learning using BERT and Hugging Face, Bao Tram Duong, 2021 (https://medium.com/mlearning-ai/twitter-sentiment-analysis-with-deep-learning-using-bert-and-hugging-face-830005bcdbbf)

28/08/2022, Athens

Fine-tuning a BERT model (https://www.tensorflow.org/tfmodels/nlp/fine_tune_bert)

Differentiate between Support Vector Machine and Logistic Regression, 2020 (https://www.geeksforgeeks.org/differentiate-between-support-vector-machine-and-logistic-regression/)

Disaster Tweet Classification using BERT & Neural Network, Swati Rajwal, 2021 (https://www.analyticsvidhya.com/blog/2021/12/disaster-tweet-classification-using-bert-neural-network/)

BERT Explained: State of the art language model for NLP,Rani Horev,2018 (https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270)

TF-IDF Simplified, Luthfi Ramadhan, 2021 (https://towardsdatascience.com/tf-idf-simplified-aba19d5f5530)

Efficient Root Searching Algorithms in Python, Louis Chan, 2020 (https://towardsdatascience.com/mastering-root-searching-algorithms-in-python-7120c335a2a8)

Large Batch Optimization for Deep Learning: Training BERT in 76 minutes, Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, Cho-Jui Hsieh, 2020 (https://arxiv.org/abs/1904.00962?context=stat.ML)

Choosing the right parameters for pre-training BERT using TPU, Pooja Aggarwal, 2021 (https://medium.com/analytics-vidhya/choosing-the-right-parameters-for-pre-training-bert-using-tpu-4584a598ca50)

Does Model Size Matter? A Comparison of BERT and DistilBERT, Jack Morris, 2020 (https://wandb.ai/jack-morris/david-vs-goliath/reports/Does-Model-Size-Matter-A-Comparison-of-BERT-and-DistilBERT--VmlldzoxMDUxNzU)

28/08/2022, Athens

How to Configure the Learning Rate When Training Deep Learning Neural Networks, Jason Brownlee, 2019 (https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/)

Difference Between a Batch and an Epoch in a Neural Network, Jason Brownlee, 2021 (https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/)

Precision, Recall and F1 score for different threshold values, 2022 (https://www.researchgate.net/figure/Precision-Recall-and-F1-score-for-different-threshold-values-A-Precision-Recall-and_fig5_340103483)