“Exploring Customer Opinions: Sentiment Analysis and Topic Modeling of Amazon Product Reviews”

by

Fabio Campus, Giuseppe Curridori

Università degli studi di Cagliari - facoltà di Scienze Economiche, Giuridiche e Politiche
corso di Laurea Magistrale in Data Science, Business Analytics e Innovazione

Abstract

The project aims to perform a sentiment analysis on Amazon reviews of various products belonging to different classes, and subsequently, the topic modeling on positive and negative reviews for each product, in order to identify the relevant characteristics, both positive and negative, within each product class.
To gather data for the study, all the reviews were scrapped from Amazon. Different sentiment classifiers were, then, applied on the data to observe the most accurate classifier for the study.
We used the best model to classify the data, and then we applied the LDA method for topic modeling.
The study concluded that the most accurate sentiment classifier was GPT-3.5, tested on a pre-polarized dataset, in order to evaluate its performance. However, the limited use of its free API did not allow us to classify our Amazon reviews dataset. Therefore, the logistic regression classifier with an accuracy of 0.9362 – trained on a sample of 400000 Amazon reviews pre-polarized – was selected for the project to polarize the dataset. Finally, the LDA method for topic modeling was applied to each product for both positive and negative reviews, visualizing the results via the pyLDAvis library, to create an interactive and more accurate visualization.

Chapter 1 – Introduction

1.1 Introduction

In the last decade, data coming from the online platforms are playing a crucial role in helping companies to understand what is good for people and what is not. This, because such data come from people themselves: *personal information* (name, birth, links to social media, etc.), *engagement data* (how consumers interact with a business' website, mobile app, social media pages, etc.), *behavioral data* (purchase histories and product usage information), *attitudinal data* (consumer satisfaction and purchase criteria).

1.2 Problem statement

The relation between sentiments on online platforms and the actions taken from companies, about their own product or services, was thoroughly examined in many studies. Most of these studies focused on the attitudinal data, gathering information from comments, or reviews, about products and performing Sentiment Analysis, followed by the Topic Modeling, in order to understand what is playing a positive role for people and what is negative.

A main step is to choose the source of data. In this project, the study will be performed on Amazon reviews, since most people around the world use Amazon to make purchases and many of them leave reviews as well.

1.3 Project goals

The objective of this study is to examine the Amazon reviews of some product, belonging to a given product class, in order to understand what are the characteristics that contribute to create a positive image of the product and, on the other hand, what is considered bad. Comparing these outputs for each product of a certain product class, we'll be able to understand which actions should be taken to reach higher consumer satisfaction, also with respect of the competitors' ones.

For the project, were chosen two product classes – shoes and protein supplement whey – with different characteristics, in order to point out how this study can be used for every type of product. A third product – Mac M1 – is used to highlight how this type of analysis can be implemented not just for different products (in our case of the same class) but also for the same, for example exploring the feelings about the product in different countries.

Chapter 2 – Literature Review

2.1 Introduction

Sentiment Analysis has been the subject of many promising studies in the past decade. It is also known as *opinion mining*, and is considered to be part of numerous fields such as machine learning, natural language processing (NLP), computational linguistics, psychology and sociology *(Yue et al., 2019 – "Sentiment Analysis and Opinion Mining: A Survey of Deep Learning Approaches").* It deals with building systems for identifying and extracting opinions from text.

On the other hand, Topic Modeling is a technique used in natural language processing to discover the abstract topics that occur in a collection of documents. It is an unsupervised machine learning method that automatically identifies the main themes or subjects discussed in a text. Topic modeling can be useful in many applications, such as organizing large collections of text data, summarizing the content of a collection of documents, and discovering hidden patterns and relationships within the data.

2.2 Sentiment Analysis Approaches

There are several approaches to perform sentiment analysis.

Rule-Based (or Lexicon-based) methods rely on a set of manually crafted rules to determine the sentiment of a text. These rules may be based on the presence of certain keywords or phrases that are indicative of a particular sentiment. For example, a rule-based sentiment analysis system might classify a text as positive if it contains words such as "good," "great," or "excellent".

Machine learning methods, on the other hand, use algorithms to automatically learn to classify text based on examples. These methods require a large dataset of labeled examples to train the algorithm. Once trained, the algorithm can then be used to classify new texts.

Hybrid methods combine elements of both rule-based and machine learning methods. For example, a hybrid system might use a rule-based approach to identify certain keywords or phrases and then use machine learning to classify the text based on these features.

While the lexicon-based approach is widely used for analyzing sentiments on online platforms (mainly social media), it often overlooks important lexical features which are significant to understand the sentiment of a sentence. Furthermore, it sometimes cannot differentiate between different sentiment intensities. For example, the sentences, "The match is exceptional" and "The match is okay", clearly convey different sentiment intensities, the former is more positive than the latter. Yet, the sentiment lexicon, such as Linguistic Inquiry Word Count (LIWC), is unable to differentiate between them and would rate them with the same intensity *(Hutto & Gilbert, 2014 – "VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text")*. Furthermore, new lexicons were developed in last years, more accurate than LIWC – since it uses a binary approach where words are simply classified as either positive or negative – such as: AFINN, SocialSent, EmoLex.

Despite that, most of the literature thinks machine learning approaches are more appropriate for this kind of studies than sentiment lexicon methods. However, machine learning approaches have its drawbacks. To train the model, they often require extensive data which are difficult to obtain. Additionally, they require top performing CPUs and high memory capacities in order to implement the models.

2.3 Topic Modeling Approaches

There are several approaches to topic modeling.

Latent Dirichlet Allocation (LDA) is a generative probabilistic model that assumes that each document in a collection is a mixture of a small number of topics and that each word in a document is attributable to one of the document's topics. LDA uses this assumption to identify the most likely set of topics that could have generated the collection of documents.

Non-negative Matrix Factorization (NMF) is a linear algebraic model that factorizes a non-negative matrix into two lower-rank non-negative matrices. In the context of topic modeling, NMF can be used to decompose a document-term matrix into a document-topic matrix and a topic-term matrix, which can be used to identify the main topics in the collection of documents.

Latent Semantic Analysis (LSA) is another linear algebraic model that uses singular value decomposition (SVD) to decompose a document-term matrix into a set of orthogonal factors. These factors can be used to identify the main topics in the collection of documents.

2.4 Other Approaches

Despite all the approaches – for both Sentiment Analysis and Topic Modeling – seen above, there are more sophisticated models which allow us to acknowledge what is considered good and what bad, from a text.

One of them is *GPT* (Generative Pretrained Transformer). It is a type of language model developed by OpenAI that can be used for a variety of natural language processing tasks, including sentiment analysis. GPT models are trained on large amounts of text data and can generate human-like text, making them well-suited for tasks such as sentiment analysis. GPT models can be used to analyze text data from sources such as social media posts, news articles, and customer feedback to determine the sentiment expressed towards a particular topic or entity. For example, GPT can be used to analyze social media comments to determine the overall sentiment towards a particular brand or product.

GPT models are highly effective in understanding and processing natural language and can better identify nuances and context, resulting in more accurate results.

Chapter 3 – Methodology

Python was used in all stages of the project. It was first used to scrap the reviews from Amazon – different Amazon websites around the world. Then, after a stage of preprocessing and data exploration, it was used to create different sentiment classifiers. These sentiment classifiers were either machine-based, such as *Support Vector Machine (SVM)* or *Random Forest* models, or transformer-based such as and GPT-3.5. Then, it was used to develop Topic Modeling algorithm, such as LDA. Finally, the outputs were analyzed.

All codes were saved inside their relative branches – within the needed packages, in a file *requests.txt,* and the file README.md, which explains what happens in the branch and the functions of the files. For example, the code about scraping was saved inside the branch *data_scraping*, the code about cleaning inside *data_cleaning*, and so on.

Chapter 4 – Project Analysis

4.1 Sources of Data

For the effectiveness of the project, data were scraped from Amazon websites and, in order to have a good machine learning model, we downloaded online a dataset containing Amazon reviews and their sentiment, to train the model.

4.1.1 Amazon Reviews

A central component of the study is to obtain data from Amazon websites – we used Amazon US, Amazon UK, Amazon Canada, Amazon India, Amazon Australia for Mac, in order to have many reviews about a single product and conduct some analysis considering various nations. While we used only Amazon US for running shoes and whey – this website contains most of the reviews about such products.
To do that, a python package called *Selenium* was used to scrap the data. The Selenium package is used to automate web browser interaction from Python. Several browsers/drivers are supported (Firefox, Chrome, Edge), as well as the Remote protocol.
We created a class called "Reviews" which contains the code to extract all the information we want from a given URL, and save them in a json file. Then, we used such class to get all the information for each product review URL (letting the scraper go for all the review pages available) and we transferred all data from the json file to a csv.
During the procedure, we realized that data from Amazon US and Amazon Canada couldn't be tried – our procedure is able to overcome Amazon anti-bot measures, but for data from US and Canada probably the anti-scraper is more effective – so we make people identify, typing e-mail and password of their amazon account (using official Amazon interface), in order to overcome the anti-bot and get all data.

This process returned us around 50000 reviews, taking more or less 8 hours – we set a 4-seconds time sleep in order to make sure to successfully overcome the anti-bot: 2 seconds for the page request, 2 seconds of delay.

4.1.2 Amazon Sentiment

The other dataset is a Kaggle dataset named "Amazon Reviews", and it contains Amazon reviews – it does not gather Amazon reviews with 3 stars, and classify reviews with 1 and 2 stars as negative and reviews with 4 and 5 stars as positive – on hundreds of thousands of products – *title* and *body* – split into two classes: class 1 is the negative and class 2 is the positive. Each class has 1800000 training samples and 200000 testing samples. We chose to use only observations coming from the test file – 200000 of class 1 and 200000 of class 2 – and not from the train file, since this contains around 3600000 observations and overloads the repository.

4.2 Data Cleaning

Since we need cleaned data to perform our study, a class "DataCleaner" was created, using mostly the package *re* – it provides regular expression matching operations. We used that to convert raw scrapped data into more suited ones, making operations such as correcting meaningless and incomprehensible characters in titles and bodies, deleting useless columns, creating a new column "brand" which will be used as a filter for applying models – since the output of the process will be a unique dataset for each product class. At the end, we'll have three cleaned datasets: Mac, shoes, whey – etc.

Furthermore, some comments were removed. In particular, when you scrap comments from a given Amazon website, the ones coming from a nation different from the specific website's nation, yield some conversion error in sentences and characters.

We also created a new dataset, containing all the verified reviews from the three datasets – Mac, shoes, whey – in order to manage all the available information in an easier way among branches. This dataset, called *data*, comes from the file *data_wrangling.py*, and we are able to distinguish product's reviews thanks to two new columns: *pc* ("product class": *m* stands for Mac, *s* stands for shoes, *w* stands for whey) and *dis* ("discriminant", the product discriminant factor: nations for Mac, brands for shoes and whey).

For the Kaggle dataset, since all the operations about cleaning and converting raw data into more suited ones were already implemented, we just extracted 100000 reviews from the dataset *test* – it contains around 400000 reviews – keeping balance between positive and negative comments (50000 each), named columns and switched polarity values (1 -> 0, 2 -> 1).

4.3 Data Dictionary

Two kind of datasets were used in the project. Therefore, it is imperative to have a data dictionary to maintain clarity and avoid confusion in the project.
We'll indicate the column's name followed by its description, for both datasets.

4.3.1 Amazon Reviews data dictionary

1. brand: refers to the company that produces the product.
2. title: refers to the title of a specific Amazon review.
3. body: refers to the comment of a specific Amazon review – its core.
4. rating: refers to the customer satisfaction about the product, in a specific review.
5. helpful: refers to the number of people that found helpful that specific review.
6. location: refers to the nation of that specific reviewer.
7. verified: refers to the case where the reviewer bought that product on Amazon, or not.

4.3.2 *Data*

1. pc: refers to the product class – m = Mac, s = shoes, w = whey.
2. ds: refers to the product discriminant factor – nation for Mac, brand for shoes and whey.
3. title: refers to the title of a specific Amazon review.
4. body: refers to the comment of a specific Amazon review.
5. rating: refers to the customer satisfaction about the product, in a specific review.
6. helpful: refers to the number of people that found helpful that specific review.

4.3.3 Kaggle Amazon Reviews data dictionary

1. polarity: refers to the negativity (0) or the positivity (1) of the body, in a specific review.
2. title: : refers to the title of a specific Amazon review.
3. body: refers to the comment of a specific Amazon review.

4.4 Data Summary and Data Exploration

We conducted descriptive analysis in order to provide a summary of the main features of the datasets, through the use of tables, graphs, and numerical measures. This can help to identify patterns and trends in the data, and can provide a foundation for further statistical analysis.
In the report will appear only some of the graphs – the others, by running the code in the branch 03.*data_exploration.*

4.4.1 Summary

The tables and the graph below summarize all the datasets included in the project. The tables mainly summarize the number of comments per rating, indexed by *location* for "Mac" and by *brand* for "shoes" and "whey", and the total, per columns and rows as well. While the bar blot represents the distribution of reviews' polarity in the Kaggle dataset.
All these plots were realized via Python code, using mostly Pandas and Matplotlib.

Summary of Mac dataset

	1.0	2.0	3.0	4.0	5.0	Total
australia	2	0	0	1	24	27
canada	10	4	0	3	66	83
india	41	5	14	50	430	540
uk	17	6	9	22	347	401
usa	117	34	53	91	1040	1335
Total	187	49	76	167	1907	2386

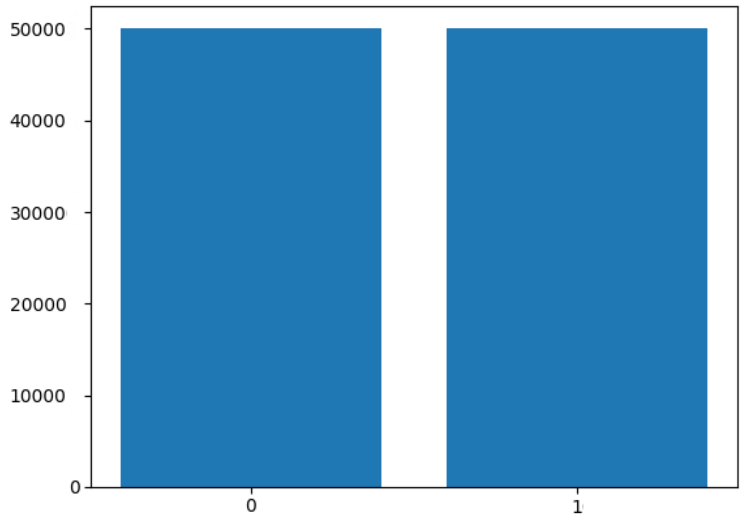
Summary of shoes dataset

	1.0	2.0	3.0	4.0	5.0	Total
adidas	308	118	143	212	1128	1909
asics	195	136	207	273	1430	2241
brooks	100	60	81	123	1239	1603
newbalance	106	63	85	109	860	1223
nike	124	44	70	56	456	750
underarm	350	156	213	321	2115	3155
whitin	222	178	257	483	1764	2904
Total	1405	755	1056	1577	8992	13785

Summary of whey dataset

	1.0	2.0	3.0	4.0	5.0	Total
body	373	225	341	542	2404	3885
levels	240	172	196	288	2542	3438
muscle	158	91	101	190	761	1301
naked	458	277	294	410	2462	3901
optimum	221	82	157	285	1523	2268
premier	576	244	327	285	1587	3019
pure	160	118	150	211	1052	1691
Total	2186	1209	1566	2211	12331	19503

Summary of Kaggle dataset



4.4.2 Exploration

Furthermore, we went deeper exploring data by performing different operations. Using the dataset *data*, we were able to get:

- Barplot representing reviews' distribution per ratings, also indicating: total number of reviews, average rating of review, average length of review, relative and absolute frequency – for all products and nations, via *bar_plot* function;
- 10 most used words in title – for each product and nation – via *words* function. The output, printed in the venv, will come after some operation including tokenization, selection of adjectives and nouns that have more than two characters, exclusion of certain meaningless words and stopwords;
- Word cloud plot representing the 10 most used words in bodies – for each product and nation. All the procedure indicated in the previous point were followed here as well;
- 10 most helpful reviews, for each product and nation – saved in a given dataset, containing 10 rows for the bodies and 19 columns for the product discriminant.

We also created a branch – *04.arc_analysis* – where we explored the change of the average rating for greater values of helpful. This was performed for each product – for Mac, the procedure is inside a Jupyter notebook (containing all computations and conclusions), for the other products, inside a py file named *extended_arc* – and we send the reader to that branch for more insights – especially in the notebook.

Chapter 5 – Data Modeling

In this chapter, the focus will be on comparing the accuracies of different sentiment classifiers. A sentiment classifier is an algorithm that determines, or classifies, a sentence as either positive or negative – it is able to classify as neutral as well, but it isn't relevant for our purpose. There are several different classifiers: machine learning models such as Support Vector Machine (SVM), Random Forest, XGBoost, Logistic Regression, Naive Bayes, KNN, Gradient Boosting, MLP; Generative Pretrained Transformer (GPT). These classifiers will be discussed in detail in this chapter. However, it is important to note that there are many other classifiers that exist in the natural language processing (NLP) domain and are not only limited to the previously mentioned classifiers.

In order to get the best classification model, we implemented a method to optimize the parameters of a custom *tokenizer*, *vectorizer*, *n_gram range* (an n-gram is a contiguous sequence of n items from a given sample of text or speech. For example, in the case of words, a 1-gram is a single word, a 2-gram is a sequence of two words, and so on), *max_df* (to ignore terms that have a document frequency higher than the given threshold. This can be used to exclude terms that are too common and are unlikely to be useful for classification), *max_features* (to limit the number of features, i.e. terms, that the vectorizer will extract from the text. This can be useful when dealing with large datasets, as it can help reduce the dimensionality of the data and improve the efficiency of the model), *norm* (to specify the normalization to apply to the term frequency-inverse document frequency (TF-IDF) values. Normalization can help ensure that all feature vectors have the same scale, which can be important for some machine learning algorithms) within a scikit-learn pipeline. This method is about using the GridSearchCV, which will perform an exhaustive search over the specified parameter values and find the best combination of parameters for our pipeline. Since we looked for the best combination of parameters both in Body and in Title, the amount of interactions checking simultaneously on both of them were more than a million, so, in order to take less time, we checked the best combination of parameters on Body at first, and then the best combination on Title given the best result on Body. It is crucial to specify that to evaluate the performance of each combination of parameters,

GridSearchCV uses cross-validation – we set a 5-fold.
The GridSearchCV was used also to tune the hyperparameters of the machine learning models. Hyperparameters are parameters that are not learned from the data, but are set by the user before training the model. Choosing the right values for the hyperparameters can have a big impact on the performance of the model. GridSearchCV performed an exhaustive search over a specified parameter grid to find the best values for the hyperparameters, for every different model. For each combination of hyperparameter values in the grid, GridSearchCV trained and evaluated the model using cross-validation. The combination of hyperparameter values that gave the best performance according to the specified scoring metric is then selected as the best set of hyperparameters for the model. Every hyperparameter value in indicated in the next paragraphs, next to the relative machine learning model.
These processes are implanted in the branch *05.machine_learning*, in particular files: *backstage.py, hyperP_tuning.py, vectorizer_tuning.py.* It is important to make a note about this last file; in order to get the best model based on the combination of parameters, this code needs to be ran six times – Body with every different vectorizer (commenting Title and the other two vectorizers unused, each time) and the same using Title.

We used some classification metrics to compare precision of classifiers. The first is the *accuracy* – in the tables represented in this chapter, there will be only this metric as a discriminant for each model –, representing the actual quality of the classifier – we used also the *precision*: ratio of true positive predictions to the total number of positive predictions (i.e. the ability of classifier to correctly identify positive instances) –; then, we used the *recall*, representing the ratio of true positive predictions to the total number of actual positive instances. It measures the ability of the classifier to find all positive instances; the third is the *F1-score,* representing the harmonic mean of precision and recall. It provides a single score that balances both precision and recall; the last one is the *support*, representing the number of actual occurrences of each class in the given data.

After the best classifier was found, we used that to classify our Amazon reviews – contained in *data* dataset. Thanks to this binary classification, we were able to perform topic modeling on both positive and negative reviews, for each product. To do that, we used not only the two deep learning methods written before, but also the Latent Dirichlet Allocation. Then, we'll compare the different performances.

5.1 Creating a sentiment classifier

In this section, different machine learning classifiers will be discussed. But before that, it is important to explain how a machine can understand text and assign it its perspective sentiment.
To create a sentiment classifier, it's important to preform data preprocessing. Once data preprocessing is done, the data must be encoded in a form that can be processed by computers and machine learning algorithms. There are several ways to encode the data, but the two most common ones in the NLP domain are Term Frequency – Inverse Document Frequency (TF-IDF) and count vectorizer – there is also Hashing Vectorizer. Count Vectorizer transforms a certain text into a vector by relying on its frequency representation. It creates a matrix where each word in a certain text is a column and each sentence a row. Take for example the two sentences "the student is happy" and "the tourist is happy". A count vectorizer applied to these two sentences will result in the following matrix.

	Happy	is	student	the	tourist
Sentence 1	1	1	1	1	0
Sentence 2	1	1	0	1	1

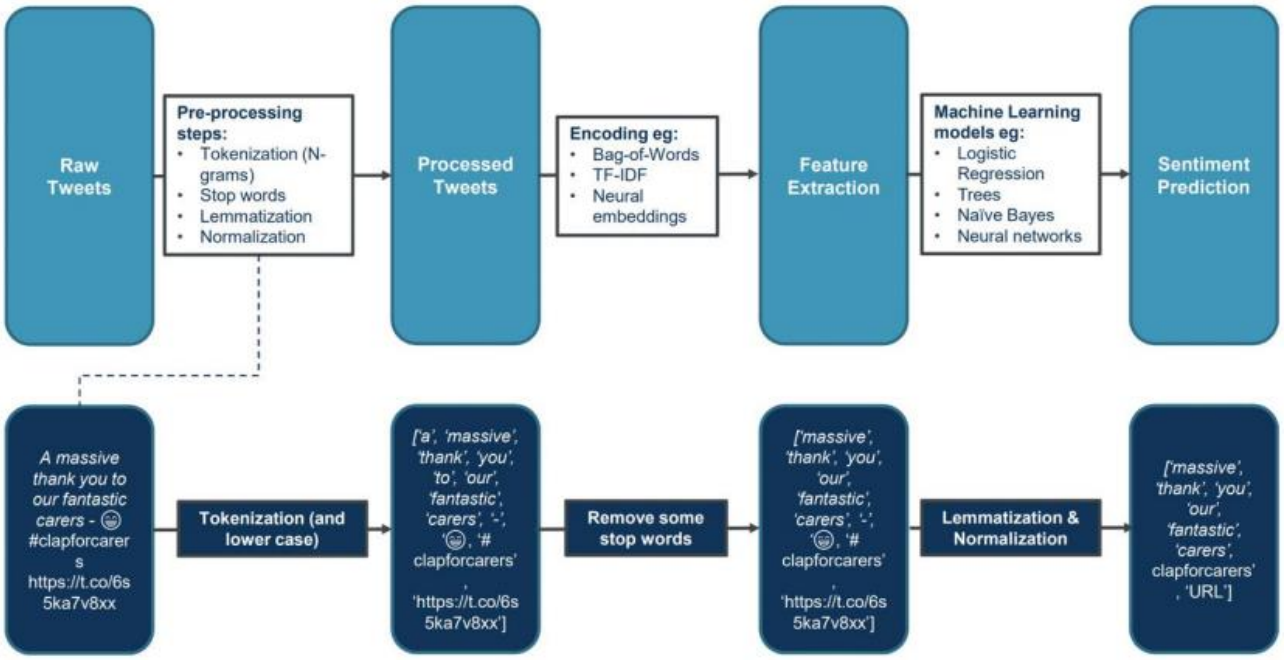
While this matrix does not completely resemble an accurate representation of a what a count vectorized document is, it still serves as a good example for understanding the process of applying it to a document. A typical count vectorized document would be much more complex and wouldn't contain the words of the document; instead, it would contain their indices.
Therefore, to summarize, a count vectorizer converts text into vectors that a computer could understand. It is imperative to mention, however, that the count vectorizer has some disadvantages. These disadvantages stem directly from the simple nature of encoding that the count vectorizer relies on.
A disadvantage of a count vectorizer is that it considers words that occur frequently as important. Following this procedure will result in an incorrect representation of the text. Words that occur frequently do not necessarily mean they are more important.
Term Frequency-Inverse Document Frequency (TF-IDF) is a statical measure that is used to measure how important a word is in a corpus. An advantage of TD-IDF is that it can measure a word's importance and, consequently, less important words can be eliminated from the corpus.
Therefore, when applying TF-IDF on the two previous sentences, "the student is happy" and "the tourist is happy", this matrix is produced.

	Happy	is	student	the	tourist
Sentence 1	1.0	1.0	1.405465	1	0
Sentence 2	1.0	1.0	0	1.0	1.405465

Notice that the TF-IDF stressed the two important parts of both sentences. In sentence 1, the word "student" has a higher TF-IDF value while "tourist" had a TF-IDF value of zero. The TF-IDF equation penalizes words that aren't important to the text and rewards words that are. In sentence 2, the exact opposite occurs.

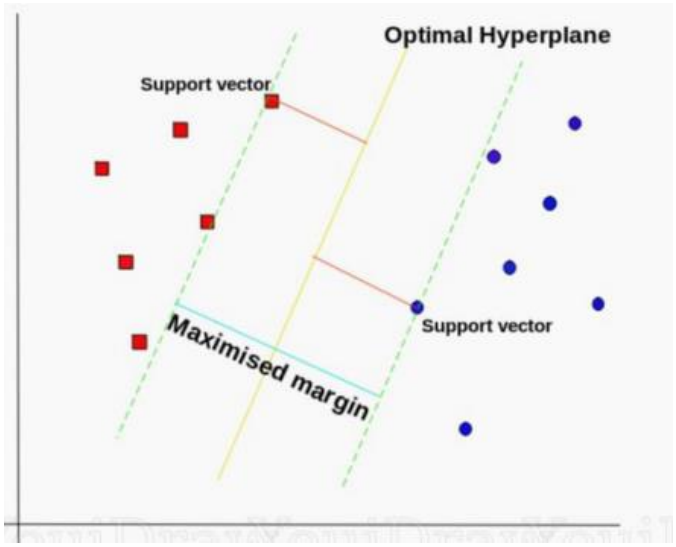
We tried all the vectorizer we named.

So, we said that creating a sentiment classifier requires several steps to be performed. Here, a NLP pipeline - *M. Zhang & Ng (2020) NLP* Pipeline*. In Twitter Sentiment Analysis: What does social media tell us about coronavirus concerns in the UK?* - that can help to sum up all we said.



5.1.1 Support Vector Machine

The first model we tested was the SVM. It works by constructing a hyperplane, or several, in hyperspace and find the optimal way to separate the data points. The SVM model attempts to find the widest separation between those hyperplanes and the data points – in our project: positive label and negative label – which is called a margin. The larger the margin, the lower the classification error, and the more accurate the SVM model will become.



In order to find the highest accuracy for this model, we passed some parameters to the GridSearchCV, performing – as we said before – a 5-fold, in a way they could get the best model trained on the Kaggle dataset, using bodies and titles as features. We split the dataset in *training set* and *test set* (67% - 33%), keeping balance between positive and negative polarities in both subsets – this entire procedure was followed for each model indicated below, so we won't write about it in the next paragraphs.

The tuning parameters we passed to the grid for this model were: *C* (regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. In other words, the C parameter tells the SVM optimization how much you want to avoid misclassifying each training example; we passed: 0, 1, 5, 10, 20), *Kernel* (kernel parameters selects the type of hyperplane used to separate the data. *Linear, poly, sigmoid, rbf*, were passed).

We implemented such steps using only two size (10000 and 50000) of the sample coming from the Kaggle dataset, because the SVM model took much time to be trained – look the figure below.

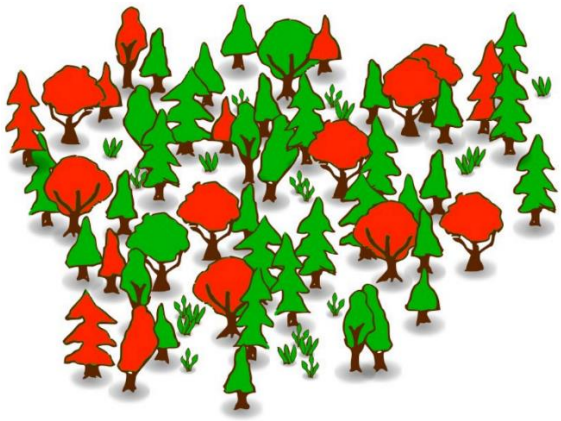
```
SVM
Confusion matrix:
[[1473  177]
 [ 196 1454]]
Accuracy: 0.887
              precision    recall  f1-score   support

           0       0.88      0.89      0.89      1650
           1       0.89      0.88      0.89      1650

    accuracy                           0.89      3300
   macro avg       0.89      0.89      0.89      3300
weighted avg       0.89      0.89      0.89      3300

Tempo di esecuzione: 0 ore e 14 minuti
```

```
SVM
Best parameters:  {'clf__C': 5, 'clf__kernel': 'linear'}
Confusion matrix:
[[7027 1223]
 [1130 7120]]
Accuracy: 0.8574
              precision    recall  f1-score   support

           0       0.86      0.85      0.86      8250
           1       0.85      0.86      0.86      8250

    accuracy                           0.86     16500
   macro avg       0.86      0.86      0.86     16500
weighted avg       0.86      0.86      0.86     16500

Tempo di esecuzione: 7 ore e 53 minuti
```

Furthermore, in order to take less time, we didn't implement the grid procedure on the bigger sample, setting C = 5 and kernel = linear – *right figure* – and probably for that reason the accuracy was lower than the one of the 10k sample – *left figure*.

5.1.2 Random Forest

The second model is the Random Forest. It is an *ensemble learning* method, which means it combines the predictions of multiple decision tree models to improve the overall accuracy and robustness of the model. In a random forest, each decision tree is trained on a different subset of the training set, and at each split in the tree, only a random subset of the features is considered. This introduces randomness into the model, which helps to reduce overfitting and improve generalization.
When making a prediction, each decision tree in the forest makes its own prediction, and the final prediction is determined by taking the majority vote (for classification) or the average (for regression) of all the trees in the forest.



The tuning parameters we passed to the grid for this model were: *N_estimators* (the number of trees in the forest. Each tree in the forest is constructed by considering a random subset of features when splitting a node and a random subset of the training data when building each tree. The final prediction is made by taking the majority vote of all the trees in the forest. Increasing the number of trees can improve the accuracy of the model, but it also increases the computational cost and may lead to overfitting if the number of trees is too large. We considered the following values: 10, 50, 100, 1000), *max_depth* (it controls the maximum depth of each decision tree in the forest. The max_depth parameter can be used to control the complexity of the decision trees in the forest. A higher value of max_depth will allow the trees to grow deeper and capture more complex relationships between the features and the target variable. However, setting max_depth too high can lead to overfitting, where the model memorizes the training data instead of generalizing to new data. We used the following values: None, 5, 10).

We implemented such steps using only two size (10000 and 50000) of the sample coming from the Kaggle dataset – look the figure below.

```
RANDOM_FOREST
Confusion matrix:
[[1363  287]
 [ 261 1389]]
Accuracy: 0.8339
              precision    recall  f1-score   support

           0       0.84      0.83      0.83      1650
           1       0.83      0.84      0.84      1650

    accuracy                           0.83      3300
   macro avg       0.83      0.83      0.83      3300
weighted avg       0.83      0.83      0.83      3300

Tempo di esecuzione: 2 ore e 40 minuti
```

```
RANDOM_FOREST
Best parameters:  {'clf__max_depth': None, 'clf__n_estimators': 100}
Confusion matrix:
[[6762 1488]
 [1411 6839]]
Accuracy: 0.8243
              precision    recall  f1-score   support

           0       0.83      0.82      0.82      8250
           1       0.82      0.83      0.83      8250

    accuracy                           0.82     16500
   macro avg       0.82      0.82      0.82     16500
weighted avg       0.82      0.82      0.82     16500

Tempo di esecuzione: 1 ore e 47 minuti
```

Also in this case, in order to take less time, we didn't implement the grid procedure on the bigger sample, setting max_depth = None and n_estimators = 100 – *right figure* – and probably for that reason the accuracy was lower than the one of the 10k sample – *left figure*.

5.1.3 XGBoost

Another model is the XGBoost – eXtreme Gradient Boosting – which is another case of *ensemble learning* method, that combines the predictions of multiple weak models, typically decision trees.
XGBoost works by adding trees to the model one by one. Each tree tries to fix the mistakes made by the previous trees – and that's the main difference with the Random Forest. The trees are created by finding the best way to split the data using a method called *gradient-based optimization*. XGBoost also uses a technique called *regularization* to prevent the model from becoming too complex and overfitting the data. The final prediction is made by combining the predictions of all the trees.

The tuning parameters we passed to the grid for this model were: *learning_rate* (it controls the step size shrinkage used in update to prevent overfitting. It is a value between 0 and 1 that determines the contribution of each tree to the final prediction. A smaller learning_rate value means that each tree has less impact on the final prediction, and more trees are generally needed to achieve good performance – we chose the following values: 0.01, 0.1, 0.2), *max_depth* (same meaning of the Random Forest case, but we tried different values: 3, 6, 9), *n_estimators* (same meaning of the Random Forest case, but we used different values: 50, 100, 150), *subsample* (it controls the subsample ratio of the training instances and has a value between 0 and 1 that determines the fraction of the training data that will be randomly sampled prior to growing trees. Setting subsample to a value less than 1 can help prevent overfitting by reducing the variance of the model. Subsampling will occur once in every boosting iteration. The values we set were: 0.5, 0.7, 1).

We implemented such steps using three size (10000, 50000 and 100000) of the sample coming from the Kaggle dataset – look the figure below.

```
XGBOOST
Confusion matrix:
[[1401  249]
 [ 276 1374]]
Accuracy: 0.8409
              precision    recall  f1-score   support

           0       0.84      0.85      0.84      1650
           1       0.85      0.83      0.84      1650

    accuracy                           0.84      3300
   macro avg       0.84      0.84      0.84      3300
weighted avg       0.84      0.84      0.84      3300

Tempo di esecuzione: 0 ore e 4 minuti
```

```
XGBOOST
Best parameters:  {'clf__learning_rate': 0.2, 'clf__max_depth': 9, 'clf__n_estimators': 150, 'clf__subsample': 1}
Confusion matrix:
[[6752 1498]
 [1401 6849]]
Accuracy: 0.8243
              precision    recall  f1-score   support

           0       0.83      0.82      0.82      8250
           1       0.82      0.83      0.83      8250

    accuracy                           0.82     16500
   macro avg       0.82      0.82      0.82     16500
weighted avg       0.82      0.82      0.82     16500

Tempo di esecuzione: 0 ore e 3 minuti
```

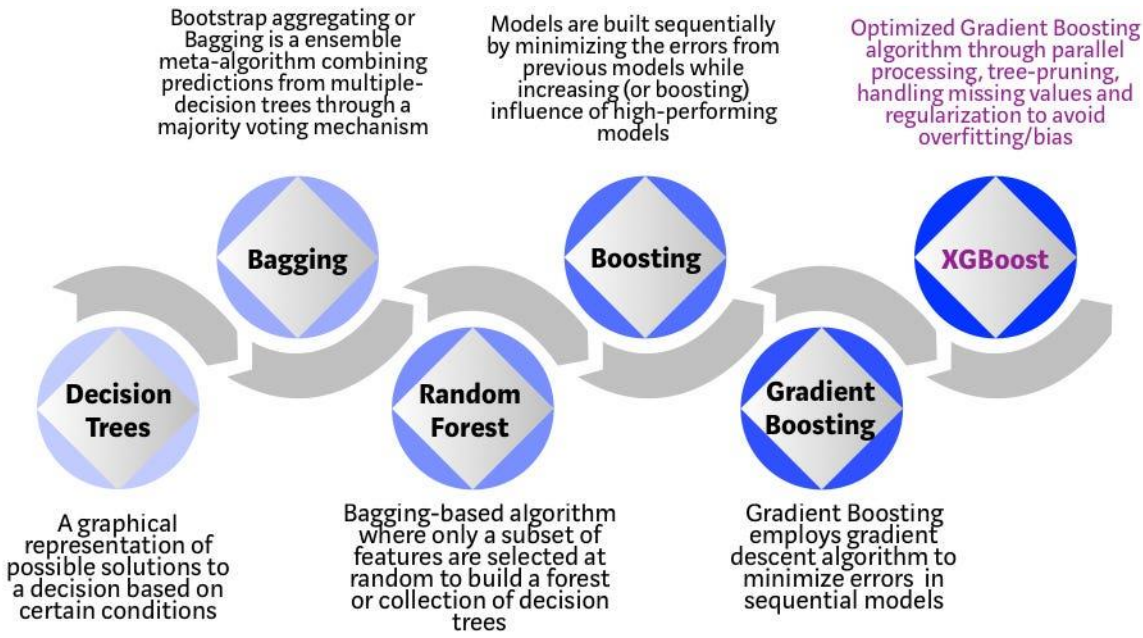
```
XGBOOST
Confusion matrix:
[[14680  1820]
 [ 1916 14584]]
Accuracy: 0.8868
              precision    recall  f1-score   support

           0       0.88      0.89      0.89     16500
           1       0.89      0.88      0.89     16500

    accuracy                           0.89     33000
   macro avg       0.89      0.89      0.89     33000
weighted avg       0.89      0.89      0.89     33000

Tempo di esecuzione: 0 ore e 20 minuti
```

The figures were plotted in order of size – 10k, 50k and 100k – and its speed of learning enabled us to train it using the grid also on a sample of 100000 observations – the relative model showed the highest accuracy.

5.1.4 Gradient Boosting

We tried also the Gradient boosting, which is a technique for regression and classification problems, that produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It works by building simpler prediction models sequentially where each model tries to predict the error left over by the previous model. To improve its predictions, gradient boosting looks at the difference between its current approximation and the known correct target vector, which is called the residual. It then trains a weak model that maps feature vector to that residual vector.
It could seem similar to the XGBoost, but this last model performs better than a normal gradient boosting

algorithm because it uses a regularization technique – to prevent overfitting by adding a penalty term to the loss function – and it is also much faster.



The tuning parameters we passed to the grid for this model were: *learning_rate* (same meaning of the XGBoost case, but we tried different values: 0.1, 0.5, 1), *max_depth* (same meaning of the Random Forest and XGBoost cases, but we tried different values: 3, 5, 10), *n_estimators* (same meaning of the Random Forest and XGBoost cases, but we used different values: 10, 50, 100), *min_samples_split* (it controls the minimum number of samples required to split an internal node. The min_samples_split parameter can be used to control the complexity of the decision trees in the ensemble. A higher value of min_samples_split will prevent the trees from growing too deep and capturing very fine-grained relationships between the features and the target variable. However, setting min_samples_split too high can lead to underfitting, where the model is not able to capture important relationships between the features and the target variable).

We implemented such steps using only two size (10000 and 50000) of the sample coming from the Kaggle dataset – look the figure below.

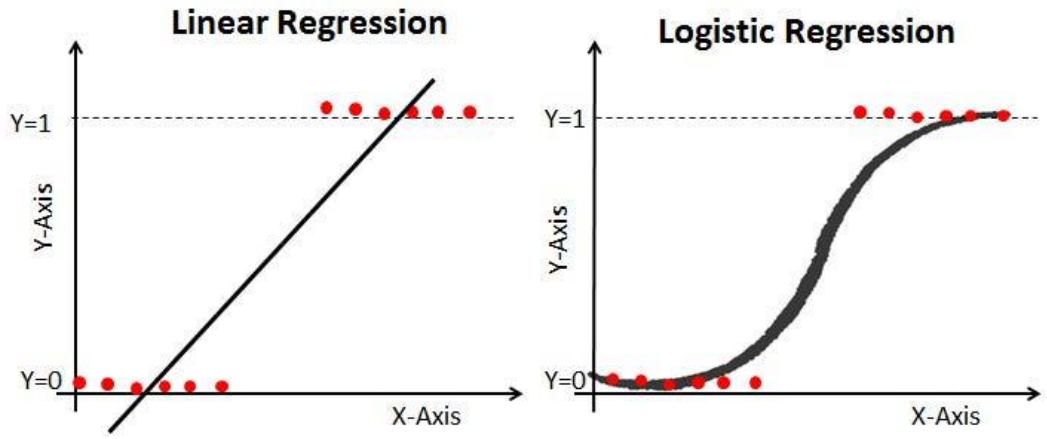
```
GRADIENT_BOOSTING
Confusion matrix:
[[1421  229]
 [ 221 1429]]
Accuracy: 0.8636
              precision    recall  f1-score   support

           0       0.87      0.86      0.86      1650
           1       0.86      0.87      0.86      1650

    accuracy                           0.86      3300
   macro avg       0.86      0.86      0.86      3300
weighted avg       0.86      0.86      0.86      3300

Tempo di esecuzione: 11 ore e 33 minuti
```

```
GRADIENT_BOOSTING
Best parameters:  {'clf__learning_rate': 0.1, 'clf__max_depth': 10, 'clf__min_samples_split': 10, 'clf__n_estimators': 100}
Confusion matrix:
[[6616 1634]
 [1636 6614]]
Accuracy: 0.8018
              precision    recall  f1-score   support

           0       0.80      0.80      0.80      8250
           1       0.80      0.80      0.80      8250

    accuracy                           0.80     16500
   macro avg       0.80      0.80      0.80     16500
weighted avg       0.80      0.80      0.80     16500

Tempo di esecuzione: 2 ore e 1 minuti
```

Also in this case, in order to take less time, we didn't implement the grid procedure on the bigger sample, setting learning_rate = 0.1, max_depth = 10, min_samples_split = 10 and n_estimators = 100 – *lower figure* – and probably for that reason the accuracy was much lower than the one of the 10k sample – *upper figure*.

5.1.5 Logistic Regression

We implemented the logistic regression as well, which is a type of statistical model that is used for classification and predictive analytics. It estimates the probability of an event occurring based on a given dataset of independent variables. It works by modeling the log-odds of an event occurring as a linear combination of one or more independent variables. The log-odds are the natural logarithm of the odds, which is the ratio of the probability of the event occurring to the probability of it not occurring. The logistic function is then used to convert the log-odds to a probability. The logistic function is an S-shaped curve that maps any real-valued number to a value between 0 and 1. This means that the output of the logistic regression model is a probability that can vary between 0 and 1.



The only tuning parameter we passed to the grid for this model was: *C* (it is the inverse of regularization strength; smaller values of C specify stronger regularization – a technique used to prevent overfitting by adding a complexity penalty for more extreme parameters. A high value of C tells the model to give high weight to the training data, while a low value tells the model to give more weight to the complexity penalty at the expense of fitting to the training data. We used the following values: 0.1, 1, 10).

We implemented such steps using different size (10000, 50000, 100000, 300000, 400000) of the sample coming from the Kaggle dataset – look the figure below – and that's because of the extreme learning speed of the Logit and its accuracy values that allow us to think it could play the role of best model – so we conducted a deeper analysis.

```
LOGISTIC_REGRESSION
Confusion matrix:
[[1470  180]
 [ 206 1444]]
Accuracy: 0.883
              precision    recall  f1-score   support

           0       0.88      0.89      0.88      1650
           1       0.89      0.88      0.88      1650

    accuracy                           0.88      3300
   macro avg       0.88      0.88      0.88      3300
weighted avg       0.88      0.88      0.88      3300

Tempo di esecuzione: 0 ore e 1 minuti
```

10k

```
LOGISTIC_REGRESSION
Best parameters:  {'clf__C': 10}
Confusion matrix:
[[7055 1195]
 [1178 7072]]
Accuracy: 0.8562
              precision    recall  f1-score   support

           0       0.86      0.86      0.86      8250
           1       0.86      0.86      0.86      8250

    accuracy                           0.86     16500
   macro avg       0.86      0.86      0.86     16500
weighted avg       0.86      0.86      0.86     16500

Tempo di esecuzione: 0 ore e 1 minuti
```

50k (no grid)

```
LOGISTIC_REGRESSION
Confusion matrix:
[[15258  1242]
 [ 1323 15177]]
Accuracy: 0.9223
              precision    recall  f1-score   support

           0       0.92      0.92      0.92     16500
           1       0.92      0.92      0.92     16500

    accuracy                           0.92     33000
   macro avg       0.92      0.92      0.92     33000
weighted avg       0.92      0.92      0.92     33000

Tempo di esecuzione: 0 ore e 13 minuti
```

100k

```
Confusion matrix:
[[45858  3642]
 [ 3719 45781]]
Accuracy: 0.9256
              precision    recall  f1-score   support

           0       0.92      0.93      0.93     49500
           1       0.93      0.92      0.93     49500

    accuracy                           0.93     99000
   macro avg       0.93      0.93      0.93     99000
weighted avg       0.93      0.93      0.93     99000

Tempo di esecuzione: 0 ore e 57 minuti
```

300k

```
LOGISTIC_REGRESSION
Confusion matrix:
[[61730  4270]
 [ 4150 61850]]
Accuracy: 0.9362
              precision    recall  f1-score   support

           0       0.94      0.94      0.94     66000
           1       0.94      0.94      0.94     66000

    accuracy                           0.94    132000
   macro avg       0.94      0.94      0.94    132000
weighted avg       0.94      0.94      0.94    132000

Tempo di esecuzione: 4 ore e 59 minuti
Tempo totale di esecuzione: 5 ore e 3 minuti
```

400k

Looking at these five figures – each one has the sample below – we found out that learning the model on the entire dataset generate an accuracy of almost 0.94, in just five hours.

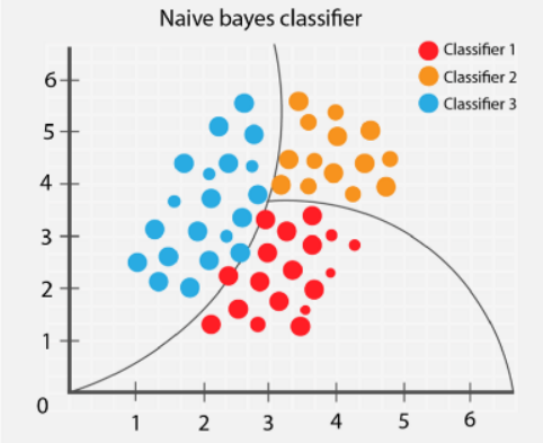
5.1.6 Naive Bayes

Another model we tried is the Naive Bayes, which is a kind of classifier that uses the Bayes Theorem. It predicts membership probabilities for each class such as the probability that given data point belongs to a particular class. The class with the highest probability is considered as the most likely class. It operates under the assumption that predictors in a Naive Bayes model are conditionally independent, or unrelated to any of the other features in the model.

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

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior x likelihood}}{\text{evidence}}$$



The only tuning parameter we passed to the grid for this model was: *alpha* (it is an additive – Laplace/Lidstone – smoothing parameter used to smooth categorical data and it is set to 1.0 by default. This means that if a given class and feature value never occur together in the training data, the probability will be zero when it is computed. This can cause problems when making predictions on new data. To avoid this issue, a small value of alpha can be used to smooth the probabilities and ensure that there are no zero probabilities. We selected values of: 0.1, 1, 10).

We implemented such steps using three size (10000, 50000 and 100000) of the sample coming from the Kaggle dataset – look the figure below.

```
NAIVE_BAYES
Confusion matrix:
[[1531  119]
 [ 275 1375]]
Accuracy: 0.8806
              precision    recall  f1-score   support

           0       0.85      0.93      0.89      1650
           1       0.92      0.83      0.87      1650

    accuracy                           0.88      3300
   macro avg       0.88      0.88      0.88      3300
weighted avg       0.88      0.88      0.88      3300

Tempo di esecuzione: 0 ore e 0 minuti
```

10k

```
NAIVE_BAYES
Best parameters:  {'clf__alpha': 1}
Confusion matrix:
[[7157 1093]
 [1586 6664]]
Accuracy: 0.8376
              precision    recall  f1-score   support

           0       0.82      0.87      0.84      8250
           1       0.86      0.81      0.83      8250

    accuracy                           0.84     16500
   macro avg       0.84      0.84      0.84     16500
weighted avg       0.84      0.84      0.84     16500

Tempo di esecuzione: 0 ore e 0 minuti
```

50k (no grid)

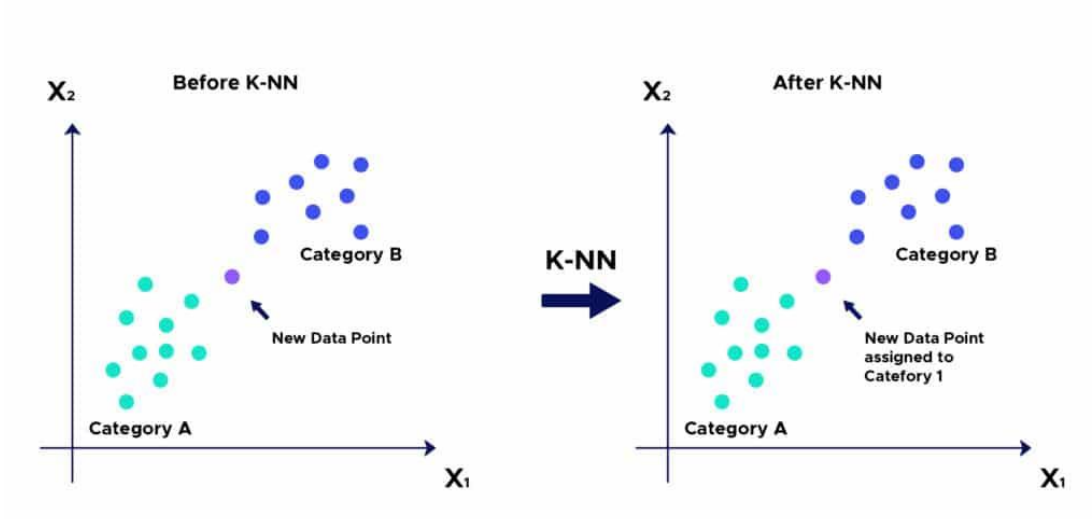
```
NAIVE_BAYES
Confusion matrix:
[[15483  1017]
 [ 1935 14565]]
Accuracy: 0.9105
              precision    recall  f1-score   support

           0       0.89      0.94      0.91     16500
           1       0.93      0.88      0.91     16500

    accuracy                           0.91     33000
   macro avg       0.91      0.91      0.91     33000
weighted avg       0.91      0.91      0.91     33000

Tempo di esecuzione: 0 ore e 7 minuti
```

100k

Even though we noticed the accuracy rate is high enough and its learning speed is high as well, we decided to stop the analysis with the sample of 100000 observations because we saw a classification ability different within the two classes: 0.89 for class 0, 0.93 for class 1 (for sample 100k). Also for this reason we chose to focus deeper on the Logistic Regression – which returned us an higher accuracy, for the same 100k sample, and kept consistent with both classes.

5.1.7 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until function evaluation. It is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The KNN algorithm involves retrieving the K datapoints that are nearest in distance to the original point. It can be used for classification or regression by aggregating the target values of the nearest neighbors to make a prediction.



The only tuning parameter we passed to the grid for this model was: *N_neighbors* (it specifies the number of neighbors to use by default for kneighbors queries. This means that when making predictions on new data, the algorithm will consider the n_neighbors closest points in the training data to determine the class of the new data point and, as neighbors, we considered: 3, 5, 10).

We implemented such steps using only two size (10000 and 50000) of the sample coming from the Kaggle dataset – look the figure below.

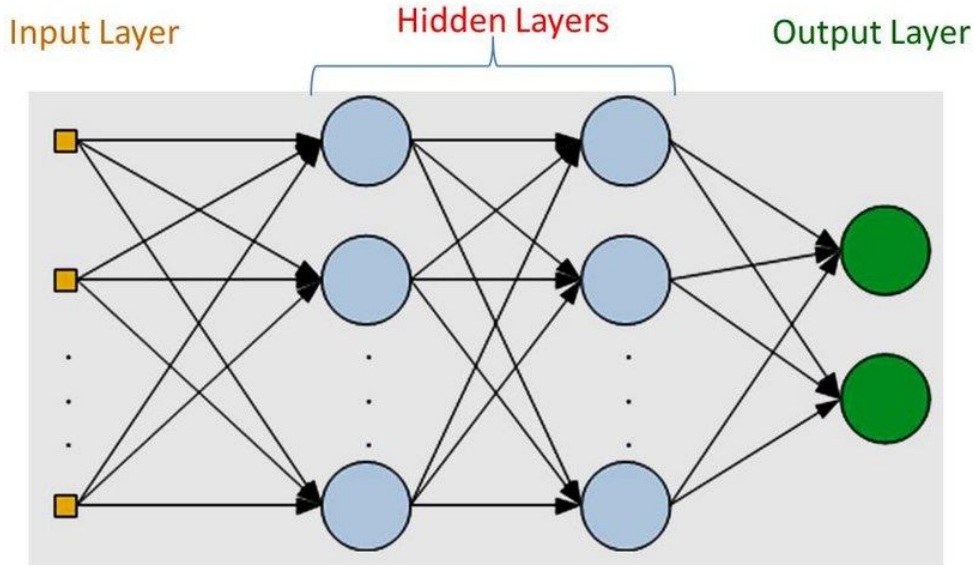
```
K-NN
Confusion matrix:
[[1269  381]
 [ 374 1276]]
Accuracy: 0.7712
              precision    recall  f1-score   support

           0       0.77      0.77      0.77      1650
           1       0.77      0.77      0.77      1650

    accuracy                           0.77      3300
   macro avg       0.77      0.77      0.77      3300
weighted avg       0.77      0.77      0.77      3300

Tempo di esecuzione: 0 ore e 3 minuti
```

```
K-NN
Best parameters:  {'clf__n_neighbors': 10}
Confusion matrix:
[[6333 1917]
 [2653 5597]]
Accuracy: 0.723
              precision    recall  f1-score   support

           0       0.70      0.77      0.73      8250
           1       0.74      0.68      0.71      8250

    accuracy                           0.72     16500
   macro avg       0.72      0.72      0.72     16500
weighted avg       0.72      0.72      0.72     16500

Tempo di esecuzione: 0 ore e 7 minuti
```

Also this model learns fast enough, but we performed analysis on just two sample size because we saw the accuracy rates were not high enough – *left figure* is the sample of 10k, *right figure* is the sample of 50k without the grid – compared with the other models we used.

5.1.8 Multi-Layer Perceptron

We also tried a type of artificial neural network: Multi-Layer Perceptron (MLP), that consists of multiple layers of interconnected nodes. It is a supervised learning algorithm that learns a function by training on a dataset. It can learn non-linear models and is capable of learning in real-time using partial_fit. In an MLP, the input data is fed into the input layer and then passed through one or more hidden layers before reaching the output layer. Each node in a hidden layer transforms the values from the previous layer with a weighted linear summation followed by a non-linear activation function. The output layer receives the values from the last hidden layer and transforms them into output values.



The tuning parameters we passed to the grid for this model were: *hidden_layer_sizes* (it specifies the number of neurons in the hidden layers of the neural network. It is a tuple where the length is equal to the number of hidden layers and each element represents the number of neurons in that hidden layer. We used three tuples: (10,), (50,), (100,)), *activation* (it specifies the activation function for the hidden layer, which is applied to the output of each neuron in the hidden layer and determines the output of that neuron. Scikit-learn provides four options for the activation function: identity, logistic, tanh, and relu, but we tried only the last two).

Since the training activity of this model is much lower than the one of the others – it takes almost four hours with a sample of just 10000 observations – we tested it only on the sample 10K.

```
MLP
Confusion matrix:
[[1511  139]
 [ 188 1462]]
Accuracy: 0.9009
              precision    recall  f1-score   support

           0       0.89      0.92      0.90      1650
           1       0.91      0.89      0.90      1650

    accuracy                           0.90      3300
   macro avg       0.90      0.90      0.90      3300
weighted avg       0.90      0.90      0.90      3300

Tempo di esecuzione: 3 ore e 47 minuti
```

Furthermore, the saved model is not present in the repository because it weights 4GB, exceeding the memory capacity of BitBucket repositories (3.75 GB). So, in order to have also this model available, it will be needed to run the file *main* locally. Of course, this was the main reason that stopped us to conduct further analysis on larger samples, even though we are aware that the accuracy rate could be better than the one of the Logit (the highest so far).

5.2 Generative Pretrained Transformer – GPT-3.5

As we announced in paragraph 2.4, we chose to use the algorithm of GPT-3.5, which is a type of language model developed by OpenAI that can be used for a variety of natural language processing tasks, including sentiment analysis. GPT models are trained on large amounts of text data and can generate human-like text, making them well-suited for tasks such as sentiment analysis.
Thanks to the open API offered by OpenAI, we were able to interact with the algorithm, on our virtual environment, with some prompts that enabled us to classify some reviews from the Kaggle dataset.
Since OpenAI offers only a limited value of requests via its free API, we polarized 1000 reviews, and we used that sample to evaluate a proxy of GPT-3.5 accuracy, by comparing its sentiment outputs with the Kaggle dataset ones. At the end, we got an accuracy equal to 0.946.

5.3 Model Comparison and Selection

The table below summarizes the accuracies of the all the sentiment classifiers we tested.
The columns names represent the size of the sample – the split in train set and test set was performed while testing algorithms, as we said before. The rows names represent the models.

	1 k	10 k	50 k	100 k	300 k	400 k
SVM	-	0.887	0.8574 (no grid)	-	-	-
Random Forest	-	0.8339	0.8243 (no grid)	-	-	-
XGBoost	-	0.8409	0.8243 (no grid)	0.8868	-	-
Gradient Boosting	-	0.8636	0.8018 (no grid)	-	-	-
Logit	-	0.883	0.8562 (no grid)	0.9223	0.9256	0.9362
Naive Bayes	-	0.8806	0.8376 (no grid)	0.9105	-	-
KNN	-	0.7712	0.723 (no grid)	-	-	-
MLP	-	0.9009	-	-	-	-
GPT-3.5	0.946	-	-	-	-	-

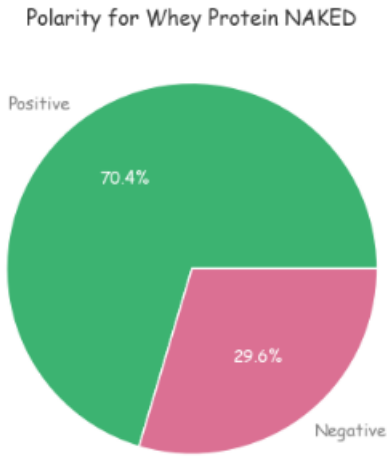
Summarizing the table above, the most accurate sentiment classifier was GPT-3.5 – even though the sample size was just 1000 – but we couldn't use such model to polarize our *data.csv* because the API requests would have exceeded the free tokens (around 2000 requests available per profile). MLP as well presented the best accuracy rate on the 10000-size sample, but it took almost four hours to train, so we didn't have the time to test such model to larger samples.
Then comes the Logit, which presents the third highest values of accuracy among all samples, that we were able to compute thanks to its learning rapidity. For such reasons, we chose to use this model to classify our dataset *data.* The model was saved in both branches *06.sentiment_and_topic* and *05.machine_learning* with the name: *Logistic_Regression400K.joblib*, and was applied via file *polarizing.py* – a new column called *polarity* was then

Chapter 6 – Topic Modeling

In Natural Language Processing (NLP), and more generally in statistical learning, topic modeling algorithms are statistical models that try to associate a topic to a document belonging to a collection of documents. The basic concept is that documents dealing with the same topic have a higher probability of containing similar terms compared to documents dealing with other topics. Topic modeling therefore allows us to find the terms that make up a particular topic and then to automatically group documents with the same topic.

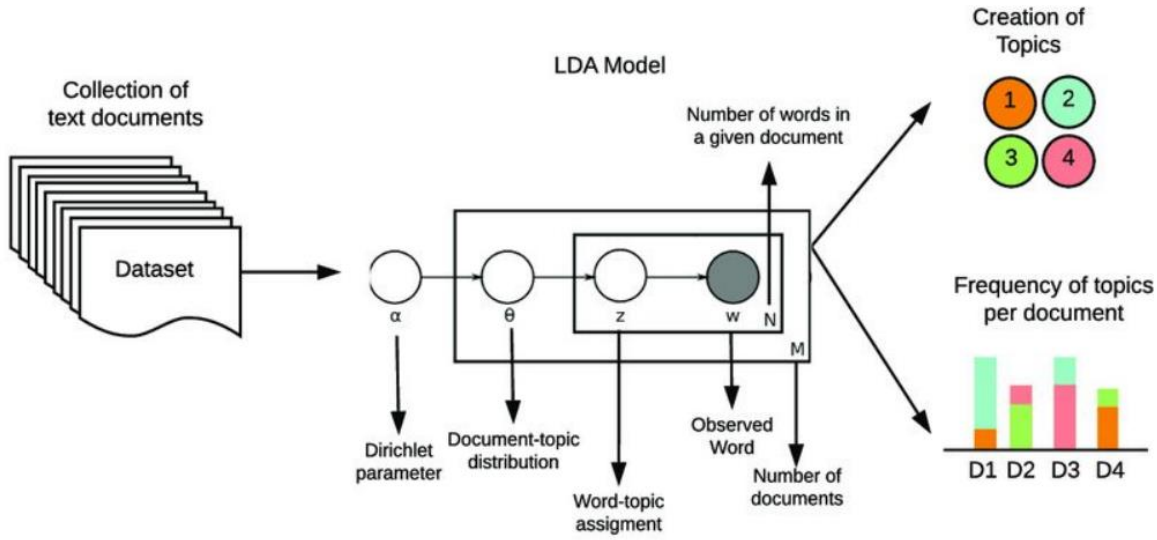
For our purposes, we decided to implement the Latent Dirichlet Allocation (LDA), which is one of the most famous topic modeling algorithms.

All the files about topic modeling are in the *branch 06.sentiment_and_topic* – here, we also plotted a pie plot for each product, representing the proportion of good and bad reviews, which can help to understand the relative dimension where the LDA works to extract topics. Below, an example about *Naked* whey:



6.1 Latent Dirichlet Allocation

We can start talking about LDA by sharing a figure that can help in understanding how it works.



It is a generative probabilistic model that assumes each topic is a mixture over an underlying set of words, and each document is a mixture of over a set of topic probabilities.

The first step while implementing a LDA algorithm consists in performing a simple preprocessing on the content we want to analyze – tokenization, removing stopwords, lemmatization, filtering only adjectives and nouns.

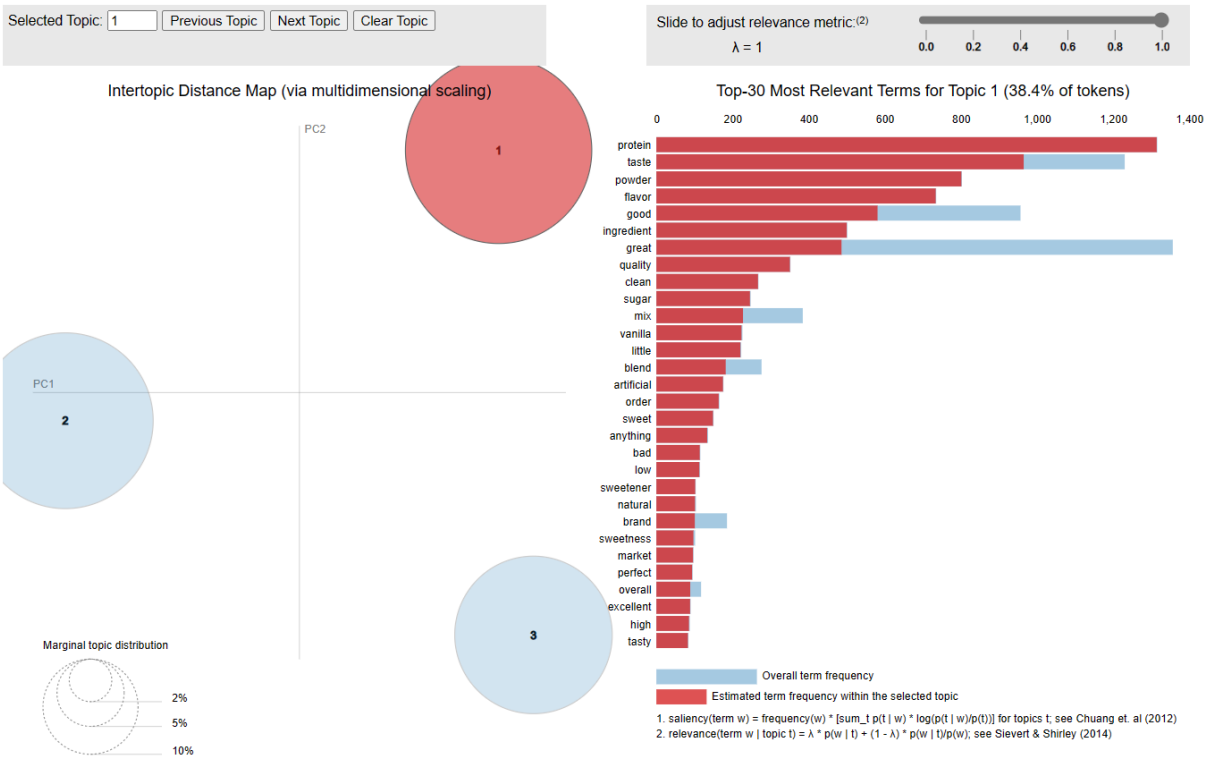
In order to allow the algorithm to perform well, we created a dictionary and corpus – via cleaned text coming from *data.csv* polarized – which will be used to train the LDA. Furthermore, we decided to calculate the coherence score, to evaluate how interpretable the topics are to humans for different numbers of topics and, after is asked user to indicate the desired number of topics to analyze, show the results.
Then, comes the training phase, where the algorithm generates a probabilistic model used to identify groups of topics. This probabilistic model can then be used to classify existing training cases or new cases provided to the model as input. Specifically, in the code we created an instance of the LdaModel class from the *gensim* library and trained it on the corpus variable using some specific parameters – to control various aspects of the training process, such as the number of passes over the data (specifies the number of passes

over the entire corpus during training), the chunk size for processing (controls the number of documents to load into memory at a time and process during the E step of the EM algorithm), and the alpha parameter for the Dirichlet prior (controls the Dirichlet prior on the per-document topic distributions).
After the training phase, the code returns a list of tuples, where each tuple contains a topic ID and its top words.
Furthermore, we decided to use the pyLDAvis library to create an interactive visualization of the LDA model. About that, we report an error that could come out while trying to visualize topics via pyLDAvis, depending on the library version installed. Below, it will be showed a possible solution:

```
If you encounter this error:

    '/LDAvis.css': ["text/css", open(urls.LDAVIS_CSS_URL, 'r').read()],
    OSError: [Errno 22] Invalid argument: 'https://cdn.jsdelivr.net/gh/bmabey/pyLDAvis@3.3.1/pyLDAvis/js/ldavis.v1.0.0.css'

    Don't worry!

    1) Go to the _display.py file and import the following library:

        import requests
        from bs4 import *

    2) look at the source code of the "show" function, in lines 261 to 263:

        files = {'/LDAvis.js': ["text/javascript", open(urls.LDAVIS_LOCAL, 'r').read()],
                 '/LDAvis.css': ["text/css", open(urls.LDAVIS_CSS_URL, 'r').read()],
                 '/d3.js': ["text/javascript", open(urls.D3_URL, 'r').read()]}

    3) and rewrite to the following form:

        files = {'/LDAvis.js': ["text/javascript", open(urls.LDAVIS_LOCAL, 'r').read()],
                 '/LDAvis.css': ["text/css", BeautifulSoup(requests.get(url=urls.LDAVIS_CSS_URL).text, 'lxml').select('p')[0].get_text()],
                 '/d3.js': ["text/javascript", BeautifulSoup(requests.get(url=urls.D3_URL).text, 'lxml').select('p')[0].get_text()]}

    Enjoy!
```

which is also the content of the file *fixing_bug,* inside the repository.

Here, there is an example of such visualization by selecting positive reviews of the brand *Naked* – a whey brand.



Each bubble on the left panel represents a topic, with the size of the bubble indicating the prevalence of the topic in the corpus and the distance between them represents the similarity between topics – bubble number 1 is red because it was selected by clicking on it.
The right panel displays a bar chart of the top terms for the selected bubble.
The relevance metric slider at the top of the right panel allows you to adjust the weight given to the probability of a term, of a certain topic, versus the term's lift (ratio of a term's probability within a topic to its marginal probability across the entire corpus), and in this case it is set to 1.
Via pyLDAvis we are also able to better understand the relationships between the topics. In fact, exploring the *Intertopic Distance Plot* can help the user learn more about how topics relate to each other, including potential higher-level structure between groups of topics.

This visualization will appear for all products – they are 15 – for both positive and negative reviews.

Chapter 7 – Conclusion

With this project, we have explored few thousand reviews directly from Amazon, with a view to understand their sentiment and subsequently indicate what are the main reasons that lead to such perception of the product.

Therefore, after having performed for each product the analysis presented in the previous point, we are able to indicate which are the most appreciated brands for running shoes and whey:

	Most appreciated	**Least appreciated**
Running shoes	Brooks	Nike
Whey	Levels	Premier

also indicating what are the main reasons that, on average – so summing up the topic model outputs for each product – lead to a positive or negative review. When we talk about whey, people who feel good about the product reach such satisfaction thanks to the protein quantity, the flavor (in particular vanilla, chocolate and almond), the powder size and mixing capacity – we could use the previous figure about Naked whey as a match with what we have just said; while, on the other hand, the negative aspects are mainly about difficulty in digestion, taste also in this case (we see that flavors like chocolate and vanilla are source of both good and bad opinions) as well as the powder size, mixing capacity and protein quantity.
If, instead, the analysis is about running shoes, people who appreciate products the most, pay particular attention to the comfort, the feeling of being barefoot, the gym use, the minimalist style and the durability. Conversely, the principal bad aspects are about the sole and the insole, the fact of being plastic shoes or not waterproof, the lace, the discomfort in the heel and toe – being too narrow or too wide –, the bad look and the price.

Since a different perspective of the project was to perform such analysis also discriminating by country, rather than by brand, using the nation discriminant for the Mac M1, we were able to understand which countries appreciate such product the most – in this case, Australia – and the least – Canada. The most important factors to make people appreciate the Mac M1 are its performance, the screen and its light, the video playback capability, the battery, the keyboard, the speaker and the easy-to-use factor. While, on the other hand, people, on average, don't like Mac M1 for its memory system, its camera, the fact that is considered expansive, the chip and the battery – also in this case.

We think that a process like this, in the perspective of single product rather than the whole product class, could be useful for several areas in a company. For example, the marketing department can use the insights gained from the analysis to better understand customer sentiment and tailor their marketing strategies accordingly; the product development team can use the information to identify areas for improvement and

make data-driven decisions about future product updates; the customer service team can use the information to address common customer concerns and improve the overall customer experience. Furthermore, all of these utilities can be seen with regard to the competitors as well.