Mining Massive Datasets

Sentiment Analysis Project

Clezia Costanzo

A.Y. 2021/2022

Contents

1 Introduction

> "Sentiment analysis and opinion mining is the field of study that analyzes people's opinions, sentiments, evaluations, attitudes, and emotions from written language." [1]

The field of linguistics and natural language processing (NLP) had a major development after the year 2000. Nowadays, it is considered a prevalent field of study in different applications, mainly for commercial uses. It is also deemed as a challenging problem, which is fairly recent and found its way into in the machine learning and artificial intelligence research. This is possible because of the huge amount of data that is available to us in the recent years: the notion of big data is crucial, in the sense that we have opinionated information about any topic one can think of. The growth of sentiment analysis coincides with those of social media, representing a goldmine of data that one can extract, analyze, and gather information from. For this reason, not only sentiment analysis is important for NLP, but also for political and sociological studies, marketing research and economics.
The core concept of sentiment analysis is the opinion. They are central to every action a human can perform and they influence our behavior on a daily basis. For example, when a consumer wants to buy a product, they will almost certainly go online and read reviews and/or ratings for a specific good (with the term good I identify products, services, and ideas) and its substitutes. Not only customers want to be able to take into account the opinion of their peers, but organizations and businesses gather data to find specific market segments, to improve their products, and to gather information about the public opinion on what they provide.

2 Problem definition

Let's formally define what a sentiment analysis problem entails.

2.1 Opinion definition

An opinion is a quintuple,

$$(e_i, a_{ij}, s_{ijkl}, h_k, t_l),$$

where e_i is the name of an entity, a_{ij} is an aspect of e_i, s_{ijkl} is the sentiment on aspect a_{ij} of an entity e_i, h_k is the opinion holder, and t_l is the time when the opinion is expressed by h_k. The orientation (or polarity) s_{ijkl} is positive, negative, or neutral, or expressed with different strength/intensity levels.

2.2 Entity definition

An entity e is a product, service, topic, issue, person, organization, or event. It is described with a pair,

$$\text{e: (T, W)},$$

where T is a hierarchy of parts, sub-parts, and so on, and W is a set of attributes of e.
As one can imagine, it's sometimes difficult to discern the polarity of a piece of text, given its intrinsic subjectivity: a sentence can have a positive, negative, or neutral connotation depending on the reader's values. At the same time, a given author can write a sentence with an intention which can be read in a completely different way from a reader.

From a formal standpoint, one can have a direct or indirect opinion. The former case is when an opinion is explicitly stated. An example is the following sentence:

The new iPhone's battery doesn't last long.

In this example, one can identify all the elements which make up an opinion, namely:

- e_i: the iPhone;
- a_{ij}: the battery;
- h_k: the author;
- t_l: today;
- s_{ijkl}: negative.

This particular definition can't be applied to indirect opinions, or opinions which make comparisons between different objects. In the following example:

The iPhone's new camera lasts longer than the new Samsung's one.

The above sentence is a personal opinion held by the author, and it's difficult to infer a particular sentiment to it.

2.3 Limits of sentiment analysis

As mentioned above, one of the major limit of this technique is to infer meaning to a subjective opinion. In addition, opinions are sometimes expressed through particular idioms, sarcasm, informal expression, and memes/slang popular at a particular moment in time in a particular reference group.

Oftentimes, adverbs and adjectives can be used to emphasize a word that can carry a meaning opposite to the one expressed in the sentence.

An example of this is the following:

This new car is the best. It broke down in two days.

As one can imagine, the above sarcastic sentence expresses disappointment towards the car. On the other hand, the use of the word *best* carries a inherently positive meaning which will then turn the entire sentence into a positive opinion, even though it clearly isn't. Finally, one should consider the different points of views of the people involved: an author might want to convey an opinion, but the readers could find another meaning to it, given their values, ethics and general views. On this note, it should be highlighted that, because a sentence is highly dependant on the context in which it was written, one needs to take this characteristic into account.

An example of this is the following:

I am happy that cryptocurrencies are crashing.

On the one hand, a person who invested in crypto stocks might be disappointed, while a person who is against them or indifferent to them might find this a good news or be entirely indifferent to it. The sentiment one might discern from this, by feeding the sentence to a machine learning algorithm or a neural network, is probably going to be neutral, but a person reading this text might beg to differ on its overall polarity.

3 Different approaches

A different number of techniques might be employed to solve the sentiment analysis problem.

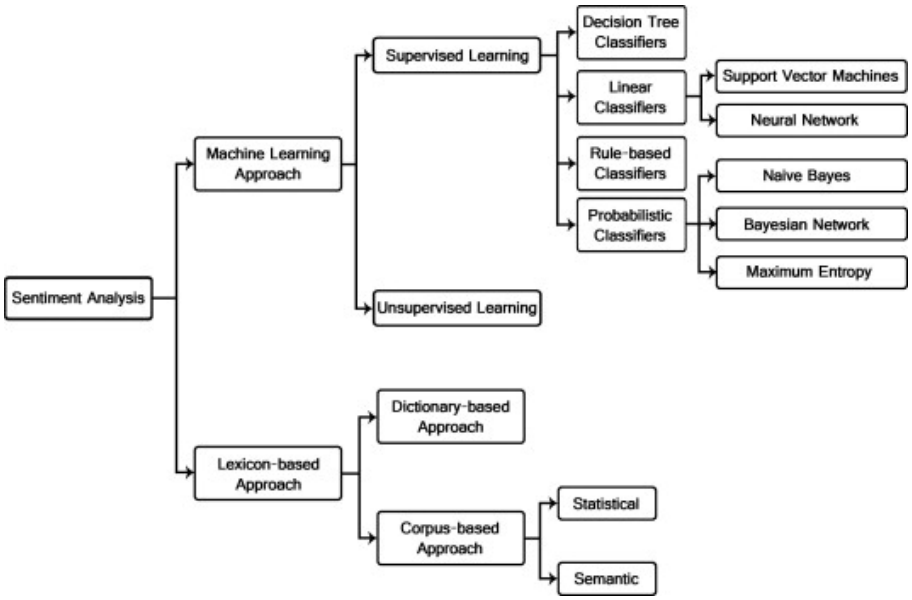

Figure 1: Different approaches to tackle sentiment analysis

In general, the current literature identifies three major approaches to the sentiment analysis problem, namely:

- The machine learning approach: this entails both supervised and unsupervised methods. In the supervised case, the most common algorithms are: decision tree classifiers, linear classifiers (SVM, NN), rule-based classifiers, and probabilistic classifiers (Naive Bayes, Bayesian Network, Maximum entropy);
- The lexicon-base approach further employs two different methodologies, namely the dictionary-based approach and the corpus-based approach (both statistical and semantic).
- The hybrid approach, which utilizes mixed solutions.

Generally speaking, assigning a sentiment to a specific sentiment can be treated as a classification problem. In particular, one wants to classify the opinion behind a sentence into two (binary classification) or more classes (multi-class classification). Most implementation I actually found prefer the binary classification, so they only take into consideration positive and negative sentiments. It's possible to adjust the sentiment threshold scores so to have three classes or

more. In this project, the sentiment analysis will be treated as a classification problem because a part of the data set is split to be used as a validation set. I also applied two unsupervised approaches to test the accuracy scores on the validation set. Before giving details about the implementation, I will provide some general definitions about the different approaches.

3.1 Lexicon-based approach

The lexicon approach is one of the main two used to perform sentiment analysis and it involves computing the sentiment from the semantic orientation of words or phrases that occur in a text. [2]

In a dictionary-based sentiment analysis, words in texts are labeled as positive or negative based on a so-called valence dictionary. A valence (or polarity) association lexicon might have entries such as the ones reported below:

- good - positive valence;
- bad - negative valence;
- joy - positive valence;
- death - negative valence.

The lexicons can be created by manual annotation for each word considered, or though automatic means. Automatically built lexicon dictionary include a lot more entries than the ones manually created. The score associated to each word may vary depending on the implementation, but usually there is a lower bound given by the worst possible sentiment and an upper bound that gives the score for the best one. Taking into account the example above, a fair assumption would be to assign 0 to the word *death* and 1 to the word *joy*, considering a scale in the range $[0, 1]$. When dealing with lexicon-based approaches, the analyzed words are part of a so-called bag of words. The analysis itself can then be performed at two levels, namely sentence and document. At a sentence-level valence, the classification system assigns a positive or negative label to the entire sentence. The overall score is then computed according to some formula that is dependent on the different implementation. At a document-level valence, on the other hand, the system assigns an overall sentiment score to the entire corpus of words. This system is generally less used, but it could be useful if applied to text that have one subject only, to avoid conflicting scoring during the process: as stated in the previous section, one of the limits of sentiment analysis is the fact that it can't distinguish different facts referring to different entities. The corpus-based approach solves the issue of finding the bag of words that are important in the specific context one takes into consideration by applying syntactic patterns or patterns that occur together with a list of opinion words which might appear in a large corpus. There are two main methods in the corpus-based approach:

- Statistical approach: if a specific word appears frequently amid positive texts, its polarity is positive. Contrarily, if a word appears frequently in

negative texts, its valence will be negative. If the frequencies are equal, it will result in a neutral polarity. The main idea is that two words appear together frequently within the same context, there is a high probability that they have the same polarity.This implies that the sentiment of a new word can be computed by determining the frequency of its co-occurrence with another word.

- Semantic approach: the classifier assigns similar sentiment scores to semantically similar words. These words are obtained by retrieving an initial set of sentiment words and iteratively expanding on it by adding synonyms and antonyms. Then, the polarity of the new word is given by the relative count of positive and negative synonyms for this word. [3]

3.2 Machine learning approach

Machine learning (ML) is a branch of artificial intelligence (AI) that enables machines to self-learn from training data and improve over time. ML algorithms detect patterns in data and learn from them, in order to compute predictions on new data. When applying these algorithms, a distinction has to be made between supervised and unsupervised learning. Supervised learning is defined by its use of labeled data sets to train algorithms to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross validation process. Supervised learning helps organizations solve for a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox. Unlike supervised learning, unsupervised learning uses unlabeled data. From that data, it discovers patterns that help solve for clustering or association problems. This is particularly useful when subject matter experts are unsure of common properties within a data set. [4] Two unsupervised techniques are used in the project: the LSTM (*long-short term memory*) approach, which is a deep learning tool, and VADER, a library specifically created to analyze social media texts. Other than those two, all systems are supervised and therefore use labels to train the model.

4 Project outline and implementation

In this project, I analyze a data set that is taken from the social network Twitter. Tweets, what the users on the platform post, have a few important characteristics:

- Tweets are short, they were at most 140 characters long. The character limit has been increased to 280 in 2017. Given that the tweets in the data set were scraped from 2009, the length of each string is 140 at most;
- The fact that they need to convey a message or a piece of information in a small number of characters leads to the fact that they are not rich in punctuation, contain abbreviations, and usually present poor grammatical structure, similarly to the old SMS.
- Tweets posted by corporations and news outlets only provide information, so they are mostly neutral. On the other hand, user-generated tweets provide a distinct sentiment/opinion.
- Tweets are sometimes hard to analyze because users write in a specific *internet slang*, or make use of emojis/emoticons.
- The use of hashtags is very helpful, because usually a person who writes about something will utilize an hashtag for that specific topic, making it easier to emphasize if the opinion held is positive or negative.
- Tweets are posted in more than 80 languages, even though Twitter provides support for only 30. This is useful to gather information about events happening in real-time in different countries in the world. In order to do this, though, one needs to develop a processing system capable of handling different languages.

The aim of this project is to provide a general introduction to the concept of sentiment analysis, by implementing and comparing different methodologies studied in other courses, namely statistical learning and programming, to see the advantages and disadvantages of each one. In addition, the comparison is done on the same same data set, containing 1.6 million rows of data extracted from Twitter. The data set was taken from Kaggle [5].

The original data set contains six columns, namely:

- *target*: the feature which will be predicted, representing the polarity of the tweet;
- *ids*: the unique identifier of the tweet;
- *date*: the date at which the tweet was posted online;
- *flag*: a boolean flag set to 'NO_QUERY' if there is no query;
- *user*: the user who tweeted;

- *text*: the actual text of the tweet.

Given the fact that only two columns are useful for this analysis, all columns except for *target* and *text* are removed. Some exploratory data analysis (EDA) will be performed to display some basic characteristics of the tweets, but this is just to check basic facts and display simple graphs.

This paper provides additional information about each technique used, while the code itself will be linked, together with the trained models. The aim is to compare different techniques and see how well they perform on the same data set. The performance is measured in terms of the accuracy on the test set.

Project phases and implementation:

1. Pre-processing of the tweets;
2. Removing stopwords, tokenization and lemmatization;
3. Feature extraction;
4. Training and evaluation of the different models.

Stopwords are the words that are commonly used in a language that they carry very little information. Example: 'a', 'the', 'is', 'are' etc. While speaking a language they are useful since they carry information about 'tense', for machine learning models these words do not add much information.

Tokenization is the process of converting text into tokens before transforming it into vectors. This process splits each sentence into the single words that compose it.

Stemming is a process that converts a word to its root meaning. This is important because in a language words can be written in either tense (present, past, or future) but what we need is the essence of a word so that it can be used to identify positive or negative reviews.

Lemmatization is a process conceptually similar to stemming, with the main difference that, instead of being rule-based, it's dictionary-based. The main idea is the same, so it brings each word to its root form, with the added feature that it takes into account the actual context the word is used in. This entails that the lemmatization process is much slower when applied.

An example of the difference between stemming and lemmatization is the following:

- Stemming: *Studied* $\rightarrow$ *studi*
- Lemmatization: *Studied* $\rightarrow$ *Study*

Tests performed include the following techniques:

1. Naive Bayes, in the multinomial and Gaussian variants;
2. K-Nearest Neighbors;

3. Support Vector Machines;
4. Decision Tree classifier;
5. Random forest classifier;
6. Logistic regression;
7. Multilayer perceptron;
8. VADER;
9. LSTM;
10. WordNet.

After I imported the data set and started working on it, I realized that its dimension was limiting to perform different tests, given the fact that oftentimes the memory would fill up quickly. To avoid this issue, I ultimately decided to keep 1/4 of the whole data set, meaning that I randomly sampled the original one to keep tweets balanced between the negative and positive ones but with a reasonable sized sample.

After this, I performed some simple EDA (Exploratory Data Analysis), to check whether missing values were present, find out the data types, dropping the unused columns, and compute the Jaccard similarity. To fulfill this, I split the whole data set into different parts: two training sets containing the positive and negative tweets, respectively, and two test sets with the same rule.

The Jaccard similarity is given by: $I \cap J / I \cup J$, which represents the ratio between the intersection of the set of words and its union.

Both Jaccard similarities resulted in a 0.99 score, which is not surprising given the fact that many words are ultimately used in both the negative and positive context. This metric is not useful to mine text similarities when the bag of words contain too many words in common.

Most of the pre-processing is done with the NLTK library, which is implemented specifically to perform NLP analysis. Spacy could have been a good alternative, because of the fact that it has GPU support, contrary to NLTK. Ultimately, though, the operations were quick because I slimmed down the dimensions of the data set. In terms of pre-processing, most of my efforts were on cleaning the strings using the regex library, perform tokenization, stemming and lemmatizing. After this phase is done, one important thing is to transform the bag of words resulting from all the tweets into a vector that one can use to model the data. This process results into a sparse matrix where each word is transformed into a numerical feature. Simply computing the frequency of each term is not enough to gather how important a word is in its context: to overcome this issue, one computes the term frequency - inverse document frequency (FT-IDF) score, to provide context for each word in each tweet. In addition, sometimes the sparse matrix resulting from this process was scaled using a StandardScaler function to reduce its dimension.

After performing the pre-processing, I went on to build different ML models. To do this, I split the data set into train and test set, keeping a 0.8-0.2 ratio. Let's briefly go over each method before giving accuracy results:

4.1 Naive Bayes

A Naive Bayes classifier is among the simplest form of Bayesian network systems, and they are based off the Bayes theorem. They are probabilistic classification models that assign class labels to data points, represented as a vector of feature values, where the class labels are drawn from a finite set. These classifiers are based on one assumption: the value of a given feature is independent of the value of any other feature, given a class variable. Using mathematical notation, the algorithm applies the following rule:

$$p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)} \tag{1}$$

where C_k represents the k classes and x is the vector of n features we want to classify. Two different implementations of this method are provided, namely the Gaussian and the multinomial. In the Gaussian variant, one additional assumption is that the continuous data is distributed with as a normal. By finding the mean and variance within each label, one can easily fit the model. In the multinomial case, we take into account the frequency that a particular feature has in its entire corpus. The assumption made is that the data points have a multivariate distribution, which is a fair assumption to make especially with data such as counts of words in a document.

4.2 K-nearest neighbors

K-nearest neighbors (KNN) is a statistical supervised algorithm used for classification and regression. In the classification setting, such as this one, the input is the set of feature one wishes to classify and the output is a class membership. Given a data point, KNN picks the closest K points to it and computes the conditional probability of that specific point belonging to one class or another. The object will be assigned to the class most common among its k nearest neighbors. Choosing the best K is the critical part of the algorithm. With $K = 1$ the algorithm applied is the one of the Bayes classifier, in which the error rate is never worse than twice the Bayes error rate, which is the lowest possible error a classification system can reach. Choosing the right K can be done by cross-validation: we test the accuracy score and the error rate for a given number of K by fitting K models, the results are the plotted. The resulting plot has an elbow shape and we want to pick the K that will minimize the error and maximize the accuracy. Doing this in practice is feasible and fast only with a small data set, given that fitting a model to a large number of feature is a fairly small process and we don't have the certainty of cycling though the right K that will guarantee the optimal solution. In my case, with a 400k data set split in two with 0.8-0.2 ratio for training and validation, it took my machine around 12

minutes of computation to fit the model with this algorithm. Cycling through a large list of K would just take too long, so I ultimately decided to use a K=20.

4.3 Support Vector Machines

Support Vector Machines (SVM) are a supervised learning model. SVM can be used for both classification and regression. SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. In the project the Linear SVM is implemented. The main idea behind the algorithm is to find the "maximum-margin hyperplane" that maximizes the distance between two classes of objects in a two-dimensional space. The greater the distance between the classes, the lower the possible error the model makes. The original plane in which the data resides is mapped into another plane with more dimensions, making it easier to find a good separation line within each class. To keep the computational load feasible, the mappings used are computed such that the scalar product of the vectors of two points one uses as inputs are easily computed through a kernel function k(x, y), chosen based on the problem one needs to solve.

4.4 Decision tree classifier

A decision tree classifier is another example of supervised learning models. It can be used for both classification and regression. In this case I built a classification tree, where the output is the predicted class in which a given data point belongs. A Decision tree is a tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. A tree is fit by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. It is called a decision tree because it starts with the root node, which expands on further branches and constructs a tree-like structure. In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm. From a graphical perspective, one can visualize the data set into a Cartesian plane in which splits are performed where it makes most sense to keep classes separated. A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into sub trees, or sub planes in the Cartesian plane. In the default implementation provided by scikit-learn, the splitting is performed based on the Gini index, a measure of purity used while creating a decision tree in the CART algorithm. An attribute with the low Gini index should be preferred as compared to the high Gini index (a low Gini index implies a purer split in the branch). The CART algorithm uses the Gini index

to create binary splits. Gini index can be calculated using the below formula:

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i)) \tag{2}$$

where C are the classes and p(i) is the probability of picking a data point with the class i. A parameter that I set in the actual implementation is the maximum depth of the tree equal to 20, which results in a bushier tree with a lot of leaves. Without setting this parameter, the default algorithm will minimize the Gini index as much as it can, meaning that some nodes will only contain one data point. This optimal solution can be reached by bagging different trees or by pruning the original one to obtain a cleaner version of the previous tree.

4.5 Random forest classifier

A random forest classifier works similarly to the decision tree one. The main difference is that in this case we create a multitude of decision trees on the training data, hence the forest in the name. The main issue with decision trees is that when they are grown very deep, they tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model. Though not quite similar, forests give the effects of a k-fold cross validation.

4.6 Logistic regression

Logistic regression is another statistical supervised learning method. The method estimates the probability of an event occurring, based on a given data set of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied on the odds—that is, the probability of success divided by the probability of failure. This is also commonly known as the log odds, or the natural logarithm of odds, and this logistic function is represented by the following formulae:

$$Logit(Pi) = \frac{1}{(1 + e^{(-Pi)})} \tag{3}$$

$$ln\frac{P_i}{1 - P_i} = \beta_0 + \beta_1 * X_1 + \ldots + \beta_k * X_k \tag{4}$$

where P_i is the probability of a given element belonging to class i. In this logistic regression equation, logit(pi) is the dependent or response variable and x is the independent variable. The β parameter, or coefficient, in this model is commonly estimated via maximum likelihood estimation (MLE). This method tests

different values of beta through multiple iterations to optimize for the best fit of log odds. All of these iterations produce the log likelihood function, and logistic regression seeks to maximize this function to find the best parameter estimate. Once the optimal coefficient (or coefficients if there is more than one independent variable) is found, the conditional probabilities for each observation can be calculated, logged, and summed together to yield a predicted probability. For binary classification, a probability less than .5 will predict 0 while a probability greater than 0 will predict 1.

4.7 Multilayer perceptron

A multilayer perceptron (MLP) is a fully connected class of feedforward artificial neural network (ANN). An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. The default activation function provided is the relu, which is the one used, and the default solver is adam, which works pretty well on fairly large data sets. In addition, the amount of hidden layer in the default method is 100. I also increased the maximum number of iteration to 1000, while the default is 200, to attempt an increase in accuracy.

4.8 VADER

VADER (Valence Aware Dictionary and sEntiment Reasoner)[6], is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media. The main difference between all the other methods and this one is that VADER does not require any pre-processing of the text one feeds to it. The compound score is computed by summing the valence scores of each word in the lexicon, adjusted according to the rules, and then normalized to be between -1 (most extreme negative) and +1 (most extreme positive). This is the most useful metric if you want a single unidimensional measure of sentiment for a given sentence. Calling it a 'normalized, weighted composite score' is accurate. Typical threshold values (used in the literature cited on this page) are:

- Positive sentiment: compound score $\geq$ 0.05;
- Neutral sentiment: (compound score $>$ -0.05) and (compound score $<$ 0.05);
- Negative sentiment: compound score $\leq$ -0.05.

The compound score is the one most commonly used for sentiment analysis by most researchers, including the authors. The pos, neu, and neg scores are ratios for proportions of text that fall in each category (so these should all add up to be 1, or close to it with float operation). These are the most useful metrics if

you want to analyze the context and presentation of how sentiment is conveyed or embedded in rhetoric for a given sentence. These proportions represent the "raw categorization" of each lexical item (e.g., words, emoticons/emojis, or initialisms) into positive, negative, or neutral classes; they do not account for the VADER rule-based enhancements such as word-order sensitivity for sentiment-laden multi-word phrases, degree modifiers, word-shape amplifiers, punctuation amplifiers, negation polarity switches, or contrastive conjunction sensitivity. Given that the final scores are divided into three classes and the ground-truth sentiments in the data set were purely binary, the neutral sentences will be divided into two set equally to compute the final accuracy. The implementation is black-box like and does not provide a flag or a parameter to remove the neutral score, so I averaged the final scores.

4.9 LSTM

A LSTM (Long Short-Term Memory) is an artificial neural network used in the fields of artificial intelligence and deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. Long Short Term Memory networks are a special kind of RNN (Recurrent Neural Networks), capable of learning long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior. [7] In order to implement the LSTM architecture to analyze sentiments, one needs to pre-process the data: namely tokenize, lemmatize and vectorize it, similarly to what one does in the machine learning scenario. In particular, the implementations I found online were similar in terms of architecture; in this case the model in itself is sequential, with four layers, namely an embedding, a dropout, a LSTM, and a dense one. The activation function in use is the sigmoid, the accuracy metric is the accuracy, and the optimizer is adam. The training phase is by far the longest of all models tested for this project, but with a GPU available it doesn't take too long to finish the process.

4.10 WordNet

ordNet is the only method that utilizes a purely lexicon based approach. WordNet is a lexical database composing English words, grouped as synonyms into what is known as synsets. SentiWordNet operates on the database provided by WordNet. The additional functionality that it provides is the measure of positivity, negativity or neutrality as is required for Sentiment Analysis. [8]

The library that provides this method is NLTK, in particular the SentiWordNet method. Again, we pre-process the data in the same way, and feed the result as input to the method that provides as output a score that defines the polarity of the sentence. The polarity final score is a float number that is not set in a specific range like VADER, but it changes. In order to make sense of the results, the final sentiment score was computed by giving a positive sentiment to a sentence with a score greater or equal than the average of all

scores, negative otherwise. A third threshold might be added if we want to infer a neutral polarity, too. Similarly to the VADER case, the final scores are normalized with a threshold given by the average: of course this assumes that one has the ground-truth scores so I inferred a supervised method myself.

5 Final results and considerations

n the following table I reported the results of the different tests.

System	Accuracy Score
Multinomial Naive Bayes	76.72%
Bernoulli Naive Bayes	77.48%
Complement Naive Bayes	76.70%
Support Vector Machine	78.72%
K-nearest neighbors (K=3)	68.43%
K-nearest neighbors (K=21)	72.82%
Decision tree classifier	65.76%
Random forest classifier	74.23%
Logistic regression	72.97%
Multilayer perceptron classifier	79.17%
VADER classifier	75.79%
WordNet classifier	72.79%
LSTM	78.39%

It's worth noting that, for what concerns VADER and WordNet, the scoring given by the output includes the neutral scores. To normalize the final accuracy, I split the neutral sentiment equally between the negative and positive set, and then proceeded to compute the percentage of correct assignments. The overall scores do not go over 80% accuracy, so they work pretty well but are not optimal. The best systems are the SVM, known to work well in this scenario, so I expected them to fit the model well. KNN is highly dependant on the K chosen, as one can see I performed two tests with K=3 and K=21 and the accuracy has increased with the larger K. It's possible that by cross-validating more Ks, one could have increased the overall accuracy more. The VADER classifier displayed an average performance, but I'd like to point out that it was probably the most straightforward to implement and it's worth noting that it's the only system that which does not require pre-processing the tweets. LSTM has a worse accuracy than the multilayer perceptron classifier, which I found interesting, given that the training time to fit the model are way longer on the former. The worst model is the decision tree classifier, which is known to overfit the data and it's not entirely precise on text analysis, such as this case. By the use of bagging, provided with the random forest classifier, the score increases significantly because the overfitting problem is partly overcome.

References

[1] Bing Liu. *Sentiment Analysis and Opinion Mining*. URL: https://www.cs.uic.edu/~liub/FBS/SentimentAnalysis-and-OpinionMining.pdf.

[2] Maite Taboada et al. "Lexicon-Based Methods for Sentiment Analysis". In: *Computational Linguistics* 37 (June 2011), pp. 267–307. DOI: 10.1162/COLI_a_00049.

[3] Hovy E. Kim S. "Determining the sentiment of opinions". In: *Proceedings of interntional conference on Computational Linguistics (COLING04)* (2004).

[4] IBM. *Supervised and unsupervised learning definition*. URL: https://www.ibm.com/cloud/learn/supervised-learning.

[5] *Dataset used*. URL: https://www.kaggle.com/code/paoloripamonti/twitter-sentiment-analysis/data.

[6] E.E. Hutto C.J. Gilbert. "VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media". In: (2014).

[7] *Understanding LSTM*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[8] *Sentiment analysis using the SentiWordNet lexicon*. URL: https://srish6.medium.com/sentiment-analysis-using-the-sentiwordnet-lexicon-1a3d8d856a10.