# Twitter Dataset Sentiment Analysis

West University of Timișoara, Master Study Program: Big Data - Data Science, Analytics and Technologies

Author: Adrian-Ioan Tuns
Supervisor: Lect. Spătaru Adrian

---

## Abstract

Sentiment Analysis, also known as opinion mining, represents a process used to determine if a piece of writing has a positive, neutral or negative connotation. The importance of this process lays in the applications of its results: for example, the results can be used to better understand a customer's feelings towards a product, and to develop a better marketing strategy afterwards, or they can be used to make predictions, such as predicting the stock market or even the result of a political debate.

Throughout this paper, two sentiment classification models are created and tested on two Twitter datasets. The classification models are created with one of the simplest classification algorithms, Naïve Bayes, firstly using a sequential algorithm implementation, and then with a distributed implementation, using Apache Spark engine, trying to confirm the fact that for large datasets the distributed algorithm implementation can solve the problem in a much better time.

## Introduction

Sentiment Analysis has multiple applications in the world, making it a process of great importance. The particular case of Twitter [8] data was chosen to be analyzed because it represents a large source of data, on a daily basis being generated more than 12 Terabytes of data with millions of tweets being written every day, because it is a valuable source of people opinions and also because is offers a variability in social, regional and interest groups.

The research field of Sentiment Analysis represents one of the faster growing research areas in computer science, being a field that entered the computer-based analysis at the beginning of the 21th century, after subjective texts have become more available on the Web. This problem was usually tackled with two main approaches: Machine Learning and Lexicon-Based [4]. In the Machine Learning approach, this is considered a simple classification problem, which can be solved with supervised learning, with weakly, semi or unsupervised supervised learning and with meta classifiers. The Lexicon-Based approach on the other hand, focuses more on finding

an opinion lexicon used to analyze the text, either by finding the opinion seed words for which the synonyms and antonyms are then considered, or by beginning with a seed list of opinion words, that together with a corpus allow the finding of context specific orientations.

As previously mentioned, having to work with datasets of possibly millions of tweets, the traditional machine learning methods would lack the capacity to efficiently analyze the datasets. This is the point where big data technologies can shine, by making use of distributed environments to better optimize the analysis of such big volumes of data.

# Related Work

The area of sentiment analysis was tackled by several people, for the particular case of Twitter and for many other cases too. For example, in the paper “Thumbs up? Sentiment Classification using Machine Learning” [1], the authors analyze a dataset containing several movie reviews using 3 machine learning methods (Naïve Bayes, maximum entropy classification, and Support Vector Machines), and they concluded that the problem is more complicated than it seems, and that performance wise Naïve Bayes tended to do the worst and SVM the best, although at a very small difference. It was also concluded that taking Unigrams as a criterion was the most effective decision.

In “Large Scale Sentiment Analysis on Twitter with Spark” [2], the authors analyzed the hashtags and the emoticons present in the tweets together with the text, using the MapReduce model, Spark Framework, Bloom Filters and K-Nearest Neighbors classification algorithm, and they obtained an efficient, robust and scalable system created in a parallel and distributed manner. They concluded that the decision to use hashtags, emoticons, punctuation and n-gram words, together with using Bloom Filters had a great impact on the binary classification performance and that it also reduced the total processing cost.

In “Twitter sentiment classification using distant supervision” [3], a solution using distant supervision was proposed, in which the training data contained tweets with emoticons. The emoticons served as noisy labels, and they built models using Naïve Bayes, Maximum Entropy and Support Vector Machines. Their feature space consisted of Unigrams, Bigrams and Parts of Speech. In the end, the authors concluded that the SVM outperformed other models and that Unigrams were most effective as features.

# Algorithms

Before going into the sequential and distributed implementation of the machine learning algorithm, the dataset goes firstly through a sequential preprocessing phase:

1. Every tweet is converted to lower case and the Twitter usernames, the punctuations, numbers, special characters, extra spaces, single letters and URLs are removed.
2. The repetitions of more than 2 letters were reduced to 2 letters and some of the most common emojis were analyzed and converted to positive and negative emojis.
3. The contractions of the words are replaced (e.g., can't => can not).
4. The text is normalized with lemmatization, which transforms the inflected forms of a word to a root form (e.g., playing => play, played => play).
5. The stop words, which represent very common words (e.g., me, was, to), are removed.
6. The newly obtained processed tweets are saved in a new column, called "processed_tweet".

Afterwards, both implementations take as an input the preprocessed data, and work on the "processed_tweet" column in the following way:

1. Separate (tokenize) the words of each processed tweet.
2. Create a count vector of the words appearing in the text, by taking into consideration the Unigrams (single words) only.
3. Compute the Inverse Document Frequency (IDF), which is a measure of the importance of a term, that gives a larger weight for the terms which are less common in the corpus.
4. Split the dataset in training and test sets (with 80% of the datasets as training, and 20% as test).
5. Train the multinomial Naïve Bayes classifier.
6. Predict the sentiment of the test subsets.
7. Measure the accuracy of the model by comparing the predictions with the actual values and by using specific evaluators.

The accuracy measurement reflects the evaluation capacity of the model, and it is computed using specific metrics, already implemented as functions in the related libraries.

The classifier algorithm used was the Naïve Bayes classifier, which is a supervised learning algorithm, based on the Bayes Theorem, algorithm highly suited for text classification. The algorithm assumes that the occurrence of a certain feature is independent of the occurrence of other features and then goes into applying Bayes Rule:

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

Where,

- P(A|B) is the probability of hypothesis A on the observed event B.
- P(B|A) is the probability of the evidence given that the probability of a hypothesis is true.

- P(A) is the probability of hypothesis before observing the evidence.
- P(B) is the probability of evidence.

# Implementation

The algorithms were implemented in Python 3.9.6, using the JetBrain's PyCharm IDE [6]. A part of the experiments was run locally, directly from PyCharm, while another part was run from a Google Cloud Dataproc [5] cluster, that had the python script and the datasets stored in HDFS.
For the implementation of the algorithms, it must be noted that the following libraries were needed:

- Preprocess Part:
    - Numpy
        - Used for its power in scientific computing, especially when working with arrays
    - Pandas
        - Used for its power in data science/data analysis and machine learning tasks
    - WordNetLemmatizer, from nltk.stem.wordnet
        - The lemmatization function from this Natural Language Toolkit was needed for switching any kind of a word to its base root mode
- Sequential Algorithm
    - Heavy usage of scikit-learn (sklearn) [7] was needed in this part. Sklearn is an open-source project that offers simple and efficient tools for predictive data analysis
    - Numpy
    - Pandas
    - TfidfVectorizer, from sklearn.feature_extraction.text
        - Used to compute the term frequency, followed by the Inverse Document Frequency, which determines how important a word is in a corpus
    - train_test_split, from sklearn.model_selection
        - It is a simple function that splits a dataset into training and test subsets
    - MultinomialNB, from sklearn.naive_bayes
        - It is an implementation for a classifier made with Naïve Bayes formula, used for multinomial models
    - accuracy_score, from sklearn.metrics
        - Function used for computing the accuracy score of a classification model
- Distributed Algorithm

- SparkSession, from pyspark.sql
  - This represents the entry-point for programming with python, in Spark
- Tokenizer, from pyspark.ml.feature
  - Function used for separating(tokenizing) the words of a text, in the Spark distributed environment
- CountVectorizer, IDF, from pyspark.ml.feature
  - The two functions were used to compute the term frequency and to compute to Inverse Document Frequency
- Pipeline, from pyspark.ml
  - Used to specify the stages in which a set of functions should be applied
- NaiveBayes, from pyspark.ml.classification
  - It is an implementation for a classifier made with Naïve Bayes formula
- MulticlassClassificationEvaluator, from pyspark.ml.evaluation
  - Function used for computing the accuracy score of a classification model

# Algorithm implementation

Before the preprocessing part, it must be mentioned that a short utility function was used to convert the second dataset to a form more similar to the first one.

The preprocess part of the algorithm was made as a common step for both algorithms, being therefore independent of the environment. For the work on the processed data, the implementation of both algorithms follows the same steps, as mentioned in the Algorithms section, with little adaptations needed.

The sequential preprocessing starts by reading a .json file with a list of contractions, followed by reading the original .csv file as a pandas DataFrame. The column containing the tweet's text is converted to a numpy array, and has a function to process the text applied for each entry. The process function lowercases the string, removes unneeded information, converts the emojis to positive or negative tags, and replaces the contractions. After this, a lemmatization is performed on each word of the tweets, the stop words are removed and the newly obtained data is saved as a new column in the original dataset.

The sequential algorithm starts by reading the .csv file as a pandas DataFrame, dropping the rows with null values, and then it initializes the function that computes both the term frequency and the Inverse Document Frequency, on which the processed data is applied (after ensuring that the data is of string type). The newly obtained data together with the "Sentiment" column values are split into test and train subsets, and the Naïve Bayes multinomial model is trained on the train subsets. A classification is performed on the model with the test subset and then its accuracy is measured with the evaluator function.

The distributed algorithm starts by reading the .csv file as a Spark DataFrame, dropping the rows with null values, and then three functions are prepared to tokenize the text, to compute the term frequency and to compute the Inverse Document Frequency, functions which are put as stages in pipeline, on which the DataFrame is fitted and transformed, in order to obtain the rescaled data. The newly obtained data is split into test and train subsets, and the Naïve Bayes multinomial model is trained on the train subsets. A classification is performed on the model with the test subset and then its accuracy is measured with the evaluator function. The pipeline represents a needed adaptation, used to ensure that the functions will be called in the specific order needed, considering the distributed environment on which they are to be run on.

# Dataset

Two datasets of different sizes were used for the experiments in this paper.

The first dataset was taken from a Kaggle community competition [https://www.kaggle.com/competitions/twitter-sentiment-analysis2], which was published with the goal of building a model able to determine the sentiment of the tweets. The dataset has a size of 8.559 KB, with 100.000 rows and 3 columns: “ItemID”, “Sentiment”, “SentimentText”. The relevant columns are “Sentiment”, which can have the values 0 – negative, 1 – positive and “SentimentText” which represents the actual tweet.

The second dataset was also taken from Kaggle [https://www.kaggle.com/datasets/kazanova/sentiment140?datasetId=2477], created by a user with the purpose of also detecting the sentiment of a tweet. This dataset has a size of 233.307 KB, with 1.600.000 rows and 6 columns: “target”, “id”, “date”, “flag”, “user”, “text”. The relevant columns are “target”, which can have the values 0 – negative, 4 – positive and “text” which represents the actual tweet.

# Experiments

Before making the experiments, the two datasets were preprocessed with a sequentially implemented algorithm, with the purpose to prepare the data for the machine learning part.

In order to analyze the machine learning algorithm, 3 experiments were made on each preprocessed dataset, in the exact same way. The first two experiments were performed locally, on one physical machine (laptop), while having only the IDE process open, with the following configuration:

- Operating System: Windows 10
- IDE: PyCharm

- Processor: Intel Core i7-7700 HQ, with 4 physical cores of 2.80 GHZ
- RAM: 16 GB
- Disk type: SSD

The third experiment was performed on Google Cloud, using a Dataproc cluster on Computer Engine, with the following configuration:

- Cluster type: 1 master, 2 workers
- CPU: N1 series, with 4 vCPU
- RAM: 15 GB
- Disk Type: Standard Persistent Disk
- Primary Disk Size: 500 GB

For the first experiment, the sequential algorithm was run from inside PyCharm and the time was measured with the “time” library, by subtracting the time after the .csv file was read from the time when the classification on the test set was done.

For the second experiment, the distributed algorithm was also run from inside PyCharm, after having configured PySpark locally. The algorithm read the data from a local .csv file and the time was measured in the same way, with the “time” library.

For the third experiment, the distributed algorithm was run as a PySpark job inside the Dataproc cluster, taking as parameter the HDFS path to the python script. Before that, the local .csv file and the script file were firstly uploaded to Google Cloud, in a Google Storage Bucket and then to the HDFS inside the cluster. The python script was slightly modified to use the path towards the .csv situated in the HDFS.

# Results

After running the preprocessing algorithm, in which all the necessary steps for optimizing the unstructured data were made, the following execution times were obtained:

100.000 entries dataset execution time: 9.780968427658081 seconds.

1.600.000 entries dataset execution time: 146.53297233581543 seconds.

As for the machine learning experiments made on the datasets, the sequential implementation of the algorithm obtained an execution time of 0.8160052299499512 seconds for the first dataset, and 15.19605541229248 seconds on the second one. The accuracy of the created model was 73.73569303685776% and respectively 76.99894218424828%, for the dataset with more data available.

The second experiment with the locally run distributed algorithm had the following results for the first dataset: execution time: 8.445001363754272 seconds and accuracy of Naïve Bayes model: 70.96452272840676%, and for the second dataset: execution time: 31.253251552581787 seconds and accuracy of Naïve Bayes: 75.74977289101902%

The third and last experiment with the distributed algorithm run on Google Dataproc, with the script and the .csv files stored in HDFS, had the following results for the first dataset: execution time: 11.353847980499268 seconds and accuracy of Naïve Bayes: 70.99830457764037%, and for the second dataset: execution time: 48.59863781929016 seconds and accuracy of Naïve Bayes: 75.70725957876325%.

The accuracy of the models created by the two implementations had a similar value, with a difference of approximatively 1%, the sequential one being better in this case. As for execution time, the sequential implementation had a better result, regardless of the size of the dataset.

# Conclusion

A comparison of the Naïve Bayes classifier was made between a sequential algorithm implementation and a distributed algorithm implementation, with the purpose of performing sentiment analysis on a Twitter dataset. The premise was not met in end, regarding the time execution improvement in a distributed environment. This may be due to several factors, such as: the size of the datasets may have been too small to benefit from Apache Spark distributed engine perks, an inefficient implementation may have been made for the algorithm, or the fact that the preprocessing step was made separately greatly affected the measured time.

As future work, the previously mentioned cases will be investigated: the preprocessing will be implemented in the distributed algorithm too and tests will be made on the original datasets, and the distributed algorithm will be fine-tuned as best as possible.

# Bibliography

[1] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? SentimentClassification using Machine Learning Techniques. In Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002), pages 79–86. Association for Computational Linguistics.

[2] Nodarakis, Nikolaos & Sioutas, Spyros & Tsakalidis, Athanasios & Tzimas, Giannis. (2016). Large Scale Sentiment Analysis on Twitter with Spark.

[3] Go, Alec & Bhayani, Richa & Huang, Lei. (2009). Twitter sentiment classification using distant supervision. Processing. 150.

[4] Walaa Medhat, Ahmed Hassan, Hoda Korashy, Sentiment analysis algorithms and applications: A survey, Ain Shams Engineering Journal,

[5] Dataproc | Google Cloud. (2022, August 25). Retrieved from https://cloud.google.com/dataproc

[6] PyCharm. (2022, August 31). Retrieved from https://www.jetbrains.com/pycharm

[7] scikit-learn: machine learning in Python — scikit-learn 1.1.2 documentation. (2022, August 31). Retrieved from https://scikit-learn.org/stable

[8] About Twitter | Our company and priorities. (2022, August 31). Retrieved from https://about.twitter.com/en