

ONBOARDING ASSIGNMENT



Hogeschool van Amsterdam

ANH THU BUI

STUDENTNUMBER: 500908730

BIG DATA MINOR WS

SEPTEMBER 2022

Table of Contents

Abstract	1
Introduction	1
Background	1
Sentiment analysis	1
Algorithms	2
Method.....	2
Preprocessing data	2
Generating the custom test data set.....	5
Modelbuilding	5
Logistic Regression	6
Naive Bayes	6
Visualization	7
Results	9
Logistic Regression results.....	10
Naive Bayes results.....	12
Comparison	12
Conclusion	13
Bibliography	15

Abstract

What is the meaning behind “sentiment analysis”? In our daily life, media and therefore, an extensive amount of data plays a significant role in our life. It is not always obvious, but wherever one goes, we produce data, which is saved and processed in a way to enhance not only the user experience of a certain website or application, but also helps to adjust daily processes in a company and therefore influences one’s way to work. But what does it mean, that we “always” produce data? Think of that when someone orders something online, write reviews, recommendations or even a single comment under a YouTube video. These are, to be exact, written as texts and contain data, which can be processed and analyzed. To do this, a method called “sentiment analysis” is used. This report will focus on this matter and present some techniques and procedures to analyze a given data set.

Introduction

This report is part of the onboarding assignment of the “Big Data” minor at the university of applied sciences in Amsterdam and combines techniques and knowledge that were learned and adopted in the first weeks at university. The task was to clean and preprocess twitter data, which was provided from Sentiment140, so that it can be used in at least two machine learning models to predict, whether a tweet should be classified as negative or positive. In addition, it was mandatory to create a custom test data set, which should at least contain ten handwritten tweets. This test set should also be used to evaluate the model. While working on the assignment, other topics like “data storage” in a database and “visualization of data” needed to be covered as well. The whole process of data cleaning and preprocessing, building, and training the models, the theory behind them and methods used are explained on the following pages. Lastly, both models are evaluated and compared to each other on accuracy and other suitable measures.

The Jupyter notebook “preprocessing.ipynb” contains all code lines regarding the preprocessing and cleaning data procedure. After that, the data is used to train the models defined in “models.ipynb” and lastly evaluated. The “dash_app.ipynb” contains the implementation for a dashboard with Dash, where relevant plots and images are included to get a detailed insight into the data set. Since the data used is always saved in a local database, I included a couple of .csv files, where the needed data is stored as well. The “assets” folder contains the png files of the results such as confusion matrix’s and classification reports.

Background

Sentiment analysis

As already mentioned before, the aim in this assignment was to build and train two models, which can classify tweets, or short texts, as positive or negative. This is also described as “sentiment analysis”, a natural language processing (NLP) method and used to detect sentiment in texts, which helps to process customer feedback and therefore, detect their needs. Not only that, but sentiment analysis also comes into action in the customer service. By organizing incoming support queries by topic and directly routing them to the right department, the most urgent tasks can be handled immediately.

Sentiment analysis can be divided into different types. One is “graded sentiment analysis”, which allows leveling the positive and negative classification into e.g. “very positive”, “positive”, “neutral”

and so on. Besides that, not only polarity, but also specific emotions like happiness or sadness can be detected [1]. This report will just focus on a basic polarity detection.

Algorithms

There are three different approaches to implement a sentiment analysis. The first one is rule-based, and sentiment analysis is realized by a predefined set of principles. Words are sorted into different lists, such as “negative” and “positive”. By counting the number of appearances of these words in a text, the overall document’s sentiment can either be classified as positive or negative. Another approach, which is also used in this assignment, is the machine learning based method, where a model or to be more specific, a classifier is built and trained with already classified, or “labeled” data. While training the model, the model learns to link an input to the correct output by extracting the most relevant features, which are in this case positive or negative sentiment in words. This is realized by transforming text into “bag-of-words”, which will later be discussed in more detail. The third approach, called “hybrid approach” combines the two methods above.

Method

Preprocessing data

In order to use text as input data for a machine learning model, it needs to be cleaned and processed first. Since text data is unstructured and contain many words, which do not hold any information regarding its sentiment, they should be removed before feeding them to the model. This is an essential step since inaccurate information could strongly influence the performance of the model. The provided data were saved into two csv-files, named as “test- ” and “training data”, which are loaded into pandas data frames for easy data manipulation.

```
[270]: colnames=['polarity', 'id', 'date', 'query', 'username', 'text']
df_testdata = pd.read_csv("trainingandtestdata/testdata.manual.2009.06.14.csv", names=colnames, header=None)

[271]: df_testdata.head()
```

	polarity	id	date	query	username	text
0	4	3	Mon May 11 03:17:40 UTC 2009	kindle2	tpryan	@stellargirl I loooooooooovvvvvveee my Kindle2. Not that the DX is cool, but the 2 is fantastic in its own right.
1	4	4	Mon May 11 03:18:03 UTC 2009	kindle2	vcu451	Reading my kindle2... Love it... Lee childs is good read.
2	4	5	Mon May 11 03:18:54 UTC 2009	kindle2	chadfu	Ok, first assesment of the #kindle2 ...it fucking rocks!!!
3	4	6	Mon May 11 03:19:04 UTC 2009	kindle2	SIX15	@kenburbary You'll love your Kindle2. I've had mine for a few months and never looked back. The new big one is huge! No need for remorse! :)

Figure 1: Load Sentiment140's dataset into a pandas data frame

As the screenshot shows, each tweet is described as one row in the data set. Besides information on polarity, where the number “0” represents negative, “2” neutral and “4” positive tweets, and the actual tweet in the “text” column, the data set also contains information on the date where the tweet was published, its query and the user, who posted the tweet, as well as an id. Both data sets were combined into a single data set afterwards, because it would make the cleaning process faster.

The first step of data cleaning was to check whether the data set was imbalanced, meaning that the number of tweets labeled as one class, is much greater than the other class. This could influence the

model in that way, that it would be able to detect one class very well, but not the other one, since there was not enough data to learn from. By grouping both data frames by the column “polarity” and performing the `count()` method afterwards, the number of observations of each polarity, or “target label” is returned. In this case, the classes “0” and “1” are equally balanced with around 800.0000 observations each. The third class, which represents the neutral tweets only contains 139 tweets, which would be not enough to properly train the model to detect neutral tweets. Because of that, all tweets classified as neutral were dropped, so that the model can focus on distinguishing between negative and positive tweets.

▼ Check if there is data imbalance ↕

```
[10]: df_data.groupby('polarity')['polarity'].count()
```

```
[10]: polarity
0      800177
2         139
4      800182
Name: polarity, dtype: int64
```

Remove tweets with polarity of 2 from data

```
[11]: df_data.drop(df_data[df_data.polarity == 2].index, inplace=True)
```

Figure 3: Drop tweets with polarity "2"

Figure 2: Distribution of negative, positive and neutral tweets

After that, the columns “date”, “query” and “username”, which held irrelevant information for training the model were removed and the label for positive tweets “4” was changed to “1”. Another important step of preprocessing data is to find duplicate data in the data set:

Check uniqueness

```
print('number of unique ids ', df_data['text'].nunique())
print('total number of tweets ', len(df_data))
```

```
number of unique ids 1581825
total number of tweets 1600359
```

Figure 4: Check uniqueness

```
[288]: df_data[df_data['id'] == 1467863684]
```

```
[288]:
```

	polarity	id	text
213	0	1467863684	Awwh babs... you look so sad underneath that shop entrance of "Yesterday's Musik" O- I like the look of the new transformer movie
800261	1	1467863684	Awwh babs... you look so sad underneath that shop entrance of "Yesterday's Musik" O- I like the look of the new transformer movie

Figure 5: Duplicate tweet, where it is classified as 0 and 1

The screenshots show that there are 18534 duplicate tweets. In the right screenshot is a duplicate, where the same tweet is classified as negative, but also positive as well. These duplicates were also deleted from the data set as well as the column “id”.

After that, the actual text of each tweet was cleaned from unrelated letter and words. This includes twitter handles (mentions such as @username), punctuations, numbers and other special characters and hyperlinks. In addition, also stop words were removed. Stop words are common words such as “I”, “we”, “but”, “my” etc., which do not hold any relevant information and can be classified as “neutral”. Lastly, all tweets were checked if any words contain less than three letters. These were removed as well.

After going through all tweets, the finished and preprocessed dataset is saved locally on a MySQL database so that the data can easily be accessed and used again for different models or other purposes.

```

•[49]: words = stopwords.words('english')
words.append("i'm")
words_set = set(words)
tr_table = str.maketrans('', '', string.punctuation)
def cleanTweets(tweet):
    # remove mentions
    tweet = re.sub(r'@[A-Za-z0-9]+', '', tweet)
    # remove hashtags
    tweet = re.sub(r'#', '', tweet)
    # remove links
    tweet = re.sub(r'https?:\V/\S+', '', tweet)
    tweet = re.sub(r'http?:\V/\S+', '', tweet)
    tweet = tweet.lower()
    # remove stop words
    text_tokens = word_tokenize(tweet)
    tokens_without_sw = [word for word in text_tokens if not word in words_set]
    new_tweet = (" ").join(tokens_without_sw)
    # remove punctuation
    new_tweet = new_tweet.translate(tr_table)
    # remove words with only 2 or less characters
    text_tokens = word_tokenize(new_tweet)
    tweet_with_words_longer_than_3 = [word for word in text_tokens if len(word) >= 3]
    new_tweet = (" ").join(tweet_with_words_longer_than_3)
    if new_tweet == '' or new_tweet == ' ':
        return tweet
    else:
        return new_tweet

Save cleaned data to database ¶
•[53]: # create db first in MySQL
engine = create_engine('mysql+pymysql://root:{myPassword}@localhost:3306/{myDatabase}')

[55]: # save sentiment140's cleaned data to database
df_data.to_sql(name='twitter_data', con=engine, if_exists='fail', index=False)

[55]: 1581689

```

Figure 7: Save cleaned data to local database

Figure 6: Method to clean the tweets

The data set is now ready to be split into training and test data. The models will be trained on one part of the data, and then tested on the other remaining part, the test data. It is important to strictly divide the data set and only test the model on new and unseen data to properly evaluate the model on accuracy and other measures. In this case, the whole data set is split into four variables, where the train data set takes up to 75% of the data set, while the remaining 25% are test data.

```

[10]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_train_cleaned['text'], df_train_cleaned['polarity'], test_size = 0.25, random_state=24)

```

Figure 8: Split data into train and test set

Create Bag of Words

```

[44]: cv = CountVectorizer()
X_train = cv.fit_transform(X_train)
X_test = cv.transform(X_test)

[45]: X_train.shape

[45]: (1186266, 280888)

[46]: print("Number of words in vocabulary: ", len(cv.vocabulary_))

Number of words in vocabulary: 280888

```

Figure 9: Create bag of words with CountVectorizer

One last step before feeding the models with training data is to convert it to numerical form, since machine learning models cannot cope with words. This is achieved with the “bag of words” method, a NLP method, that considers the vocabulary of words and the frequency of a word in the given text while ignoring the order and also the grammar. Therefore, scikit-learn’s “CountVectorizer” is used. By calling

the `fit_transform()` method on the training data, a matrix is created on the given text where each unique word is represented by a column and each sample of text a row [2][3].

The screenshot above shows, that the training data is now a matrix of 1186266x280888, meaning that there are 1186266 tweets or rows, which hold the occurrence of each word, and 280888 (columns) unique words in the vocabulary. These represent the “features” the models need to learn. Another important aspect is, that the step of vectorizing of the data set always needs to be done after it was already split into train and test data. Otherwise, information of the test data would be leaked into the training process, which could lead to overfitting. This is also the reason, that the method `fit_transform()` is only called on the train data, while on the test data, only `transform()` is used [4].

Generating the custom test data set

In addition to the already given data set, creating a custom test data set was also part of the assignment. To do this, a handwritten dictionary containing the mandatory columns needed to train the model, were added as keys. Based on the sentiment of the tweet, the associated polarity, which were originally labeled as “0” for negative and “4” for positive tweets, were also added. Moreover, duplicates, stop words and punctuation were integrated into the `preprocess()` method, which I wrote to combine all preprocessing steps.

```
list_of_tweets = [
    {"polarity": 4, "id" : 1, "text" : "This is so fun"}, # create duplicates
    {"polarity": 4, "id" : 1, "text" : "This is so fun"}, # create duplicates
    {"polarity": 4, "id" : 1, "text" : "I like Machine Learning!!"},
    {"polarity": 4, "id" : 1, "text" : "Ohhh, I love this movie! I can't wait to see it in the cinema. The actors are great!"},
    {"polarity": 4, "id" : 1, "text" : "I liked this book so much, I am a huge fan of the author."},
    {"polarity": 4, "id" : 1, "text" : "This is brilliant"},
    {"polarity": 4, "id" : 1, "text" : "Cannot wait to see them live in concert. They are my favorite band #"},
    {"polarity": 4, "id" : 1, "text" : "Loved his new song. His album is also great. :)"},
    {"polarity": 4, "id" : 1, "text" : "The weather is nice today"},
    {"polarity": 4, "id" : 1, "text" : "Amsterdam is a beautiful city"},
    {"polarity": 0, "id" : 1, "text" : "I did not like his new song. It sounds boring"},
    {"polarity": 0, "id" : 1, "text" : "I am so sad, the weather is bad today"},
    {"polarity": 0, "id" : 1, "text" : "I think it is stupid"},
    {"polarity": 0, "id" : 1, "text" : "I don't like brokkoli, it's disgusting."},
    {"polarity": 0, "id" : 1, "text" : "This movie is too scary for me."},
    {"polarity": 0, "id" : 1, "text" : "That is terrifying."},
    {"polarity": 0, "id" : 1, "text" : "I am crying, I failed my test today"},
    {"polarity": 0, "id" : 1, "text" : "That's so sad, I am feeling lonely :("},
    {"polarity": 0, "id" : 1, "text" : "I don't understand how people do not hate him"},
    {"polarity": 0, "id" : 1, "text" : "I am feeling so sick today. I am gonna skip class"}]
```

Figure 10: Custom data set

After defining the dictionary, a pandas data frame was generated and the already mentioned preprocess method was called on the data set. The following screenshots show the data set before and after the preprocessing steps.

	polarity	id	text
0	4	1	This is so fun
1	4	1	This is so fun
2	4	1	I like Machine Learning!!
3	4	1	Ohhh, I love this movie! I can't wait to see it in the cinema. The actors are great!
4	4	1	I liked this book so much, I am a huge fan of the author.

Figure 11: Custom data set before preprocessing

	polarity	text
0	1	fun
2	1	like machine learning
3	1	ohhh love movie wait see cinema actors great
4	1	liked book much huge fan author
5	1	brilliant
6	1	wait see live concert favorite band

Figure 12: Custom data set after preprocessing

Modelbuilding

As already mentioned in the introduction, the task was to build and train at least two classifiers. After doing some research, I decided to look more into “Logistic Regression” and “Naive Bayes” models, since these two were common in document analysis. Another reason was that there are two types of classification models. Generative models like Naive Bayes and discriminative models, which include models such as Logistic Regression and SVM. The difference of these two is, that generative models computes the joint probability distribution $p(x,y)$ first, and after that, the conditional probability $p(y|x)$, which can be calculated using the Bayes Theorem. On the other hand, discriminative models model the conditional probability $p(y|x)$ directly [5]. Seeing how both approaches evaluate on the same task, would be interesting to see. In the following paragraphs, both will be presented on how they were implemented and how they work to detect sentiment in a text document.

Logistic Regression

Logistic Regression can be used for classification problems and is originated in the statistical field. It is used for supervised learning, meaning that by analyzing a dataset, where a set of independent variables determine a specific or dependent outcome, logistic regression aims for finding the best fitting model to describe the relationship between these variables [6].

As the name already suggests, its core function is the logistic function, which is also known as the sigmoid function and used to map predicted values to probabilities. It is S-shaped and can take any number and map it into a value between 0 and 1. Every input value (x) is combined while using weights and coefficients to predict the desired outcome (y). Therefore, logistic regression uses just like linear regression, an equation. A possible equation could look like this:

$$y = \frac{e^{b_0 + b_1 * x}}{1 + e^{b_0 + b_1 * x}}$$

“Y” represents the predicted label, “b0” is the bias and “b1” the coefficient for the input value. In the data set, each column has a related coefficient, which are learned from the training data, using maximum-likelihood estimation. While training the model, the goal is to find the best coefficients, which would be able to predict a value either close to 1, which would be in our case the “positive tweets” class, or 0, representing the “negative tweets” class. The maximum-likelihood estimation tries to find values for the coefficients, which minimize the errors while predicting input values [7].

To build a Logistic Regression model, scikit-learn’s “LogisticRegression” was imported, where a model can be defined in just a few lines of codes:

```
[ ]: #Train the model
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)

#Predicting the labels for test data
y_pred = lr.predict(X_test)
```

Figure 13: Define logistic regression model and train

After defining a LogisticRegression object, it is fitted to the training data, divided into input (X_train) and the corresponding output class (y_train). The model is now trained and has learned the before mentioned best coefficients on basis of the given training data. To evaluate the model, the input values of the test data are passed to the model in the predict method. The output are the predictions, which will later be compared to the actual outputs. The evaluation and comparison of the models will be covered in the last section of this chapter.

Naive Bayes

The second approach I followed for sentiment analysis is naive Bayes, or to be specific, multinomial naive Bayes, which is one of the most common supervised learning classification methods to analyze text documents. As mentioned before, naive Bayes is based on the Bayes theorem and computes based on the given input, the probability of each class. The class with the highest probability is returned. The following formula shows the Bayes theorem:

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

$P(A|B)$ represents the probability of A or in this case, the probability of a positive or negative tweet when “B”, or a specific text, is provided. Other components are:

$P(B)$ = Prior probability of B or the input text

$P(A)$ = Prior probability of class A or "positive"

$P(B|A)$ = Probability of B, when class A is provided

All these formulas are needed to be able to calculate $P(A|B)$ [8]. To compute $P(B|A)$, or the probability of a text when class A is given, the conjoint probability of all included words in the text needs to be calculated. To do this, all conditional probabilities of the words are ignored and the approach, that the appearance of a word is completely independent of other words, is followed. This is also the reason why the Bayes theorem is called naive. The probability of $P(A)$ is calculated by the number of tweets of class A divided by the number of all tweets. $P(B)$ can be left out since it is the same for every category [9].

Implementing a Bayes model is similar to a logistic regression model. For this, “MultinomialNB” from the scikit library is imported. After creating an object of the imported class, the model is fitted with the input values “X_train” and the associated labels “y_train”. After that, “y_pred_nb” is generated by calling `predict()` on the test data once again to evaluate the trained model.

```
•[17]: from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
```

Figure 14: Define naive Bayes model and train

Visualization

Another part of the onboarding assignment was to visualize the data in an interactive way. For this, I implemented a dashboard with the help of JupyterDash, which is a version of the library Dash, but for Jupyter notebooks. On this dashboard, a more detailed insight into the Sentiment140's dataset before and after the preprocessing procedure is given, as well as the results and relevant plots of training and testing the models are also included on this dashboard. All plots are created using plotly, except for the confusion matrix and the word cloud, which was created using the library “WordCloud”.

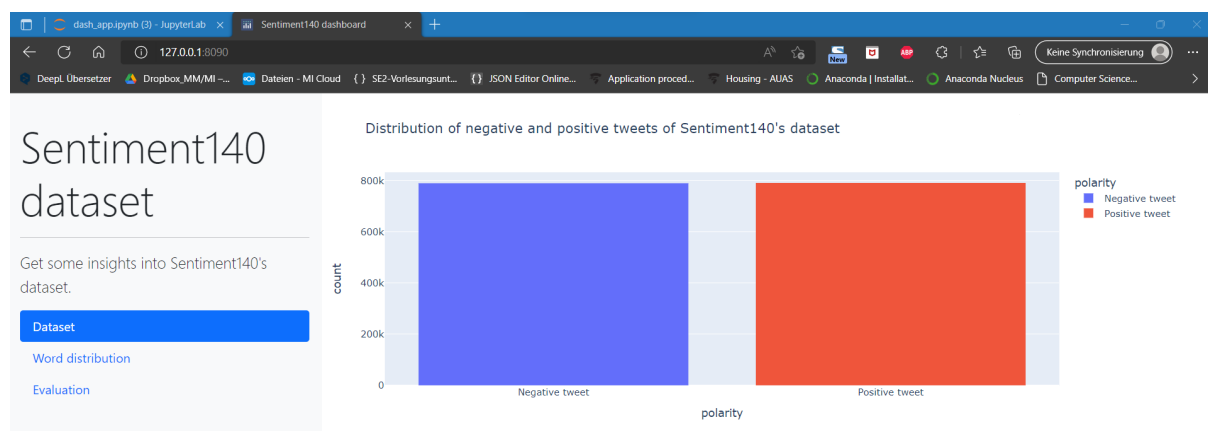


Figure 15: Home page of dash app

As the screenshot shows, there is a menu on the left, where the user can navigate through the page and find the information needed. On the home screen, an overall insight into Sentiment140's dataset is given. The figure shows the distribution of all negative and positive tweets in the dataset before preprocessing. When hovering over the bars, the exact number of tweets is shown in a small textbox, making the figure interactive. By plotting the distribution of negative and positive tweets it gets clearer, that the dataset is equally balanced and therefore, suitable to use for machine learning algorithms.

The next screenshot shows the next page, where the user can navigate to by clicking "Word distribution" in the navigation box. On this page, the user gets more insight into the train data set. The first interactive figure shows the distribution of the top 50 most used words in positive tweets. To make the differences between all words clearer, the y-axis is scaled using logarithmic scale. As the figure shows, the word "good" is the most used word in positive tweets, followed by the word "love". To highlight these results, I created a word cloud based on the frequency dictionary of the used words in positive tweets. The words "good" and "love" are also the biggest portrayed, which supports the results seen in the figure above it.

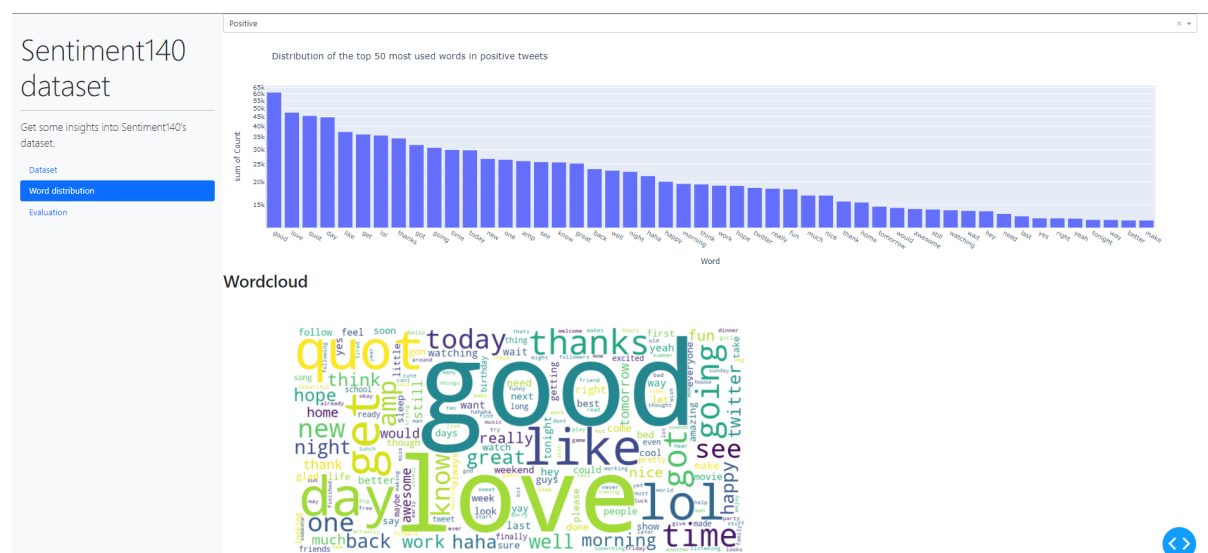
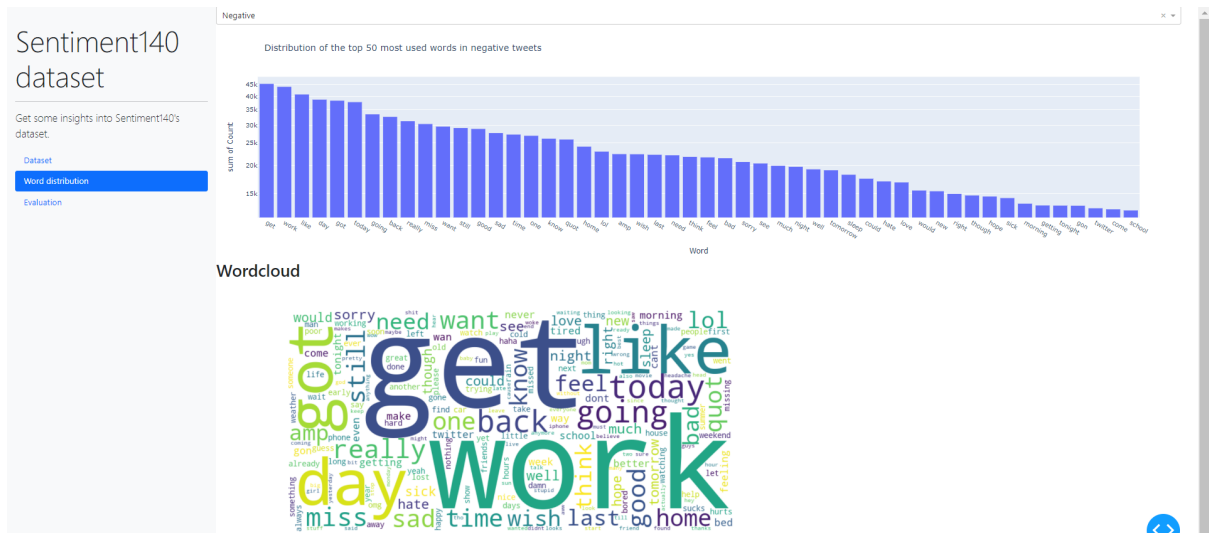


Figure 16: "Word distribution" page showing results for positive tweets

A dropdown menu at the top of the page allows replacing the plots with the plots showing the results for the negative tweets. Looking at the results, the two most used words in negative tweets are "get" and "word".



The last page “Evaluation” contains all plots and results of the two implemented models: Logistic regression and naive Bayes. Since the results are discussed in detail in chapter “Results” I will not go into them here.

Results

To analyze and compare both trained models, a classification report using the scikit-learn library is computed. Since classification reports show different evaluation metrics, the following paragraph gives a short description of each one of them.

Precision

The precision of a model represents the percentage of the correct precisions. It is calculated by the ratio of the true positive (correctly predicted positives) to the sum of true positives and false positives (predicted positives, which are actually negative) [10].

Recall

The recall shows, how many of the positives the model found. It is the capability of the model to find all positives. It is computed by the ratio of true positives divided by the sum of true positives and false negatives (predicted negatives, which are actually positive) [10].

F1 Score

The F1 score is the harmonic mean of precision and recall, where a score of “1” represents the best and “0” the worst score. The F1 score is calculated by multiplying recall and precision then dividing by the sum of both. Lastly, the score is multiplied by two [10].

Accuracy

The accuracy of a model is the ratio of correctly predicted classes divided by all prediction.

Support

The support is not an evaluation metric but represents the number of occurrences of a class in the dataset [10].

Logistic Regression results

	precision	recall	f1-score	support
0	0.79	0.75	0.77	197264
1	0.76	0.80	0.78	198159
accuracy			0.78	395423
macro avg	0.78	0.78	0.78	395423
weighted avg	0.78	0.78	0.78	395423

Figure 19: Classification report of the logistic regression model

As the screenshot shows, for both classes “0”, which represent the negative tweets, and “1”, which therefore represents the positive tweets, are an equal number of test data given. These are around 198000 observations for each class. This means, that the data set is equally balanced, and the results shown in the classification report are meaningful. Overall, most of the metrics are around a percentage of 78%. Looking at the negative class, the model is able to find all the negative tweets with a percentage of 75%, which is represented by the recall, while the precision is 79%. The model performs a bit better finding the positive tweets with 80%. Nevertheless, the precision score is slightly lower with 76%.

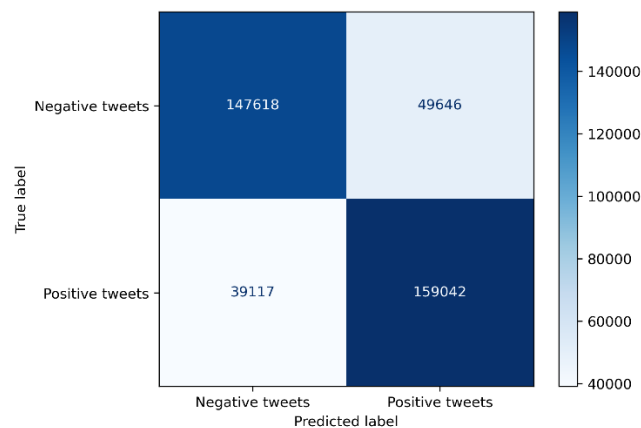


Figure 20: Confusion matrix of the logistic regression model

The corresponding confusion matrix reflects the values in the classification report. On the left, the true labels divided into negative and positive tweets are represented by the vertical axis. The horizontal axis at the bottom shows the predicted labels. As the figure above shows, there is a diagonal line going through the matrix, which is highlighted by the intensity of the colors. This means, that the model predicts the correct labels for the input values well. Out of 197.264 negative tweets, which can be computed by summing the first two values at the top (147.618 and 49.646), 147.618 were correctly predicted as negative, while 49.646 were incorrectly classified as positive tweets. The model predicts positive tweets slightly better. It classified 159.042 tweets correctly as positive and 39.117 incorrectly as negative. All in all, it can be said that the model performs good since there were no hyperparameter tuning involved.

After testing the model on the custom data set, which contained only 20 tweets, where also one tweet was also duplicated and therefore dropped, the associated classification report was also generated. In this classification report, the accuracy is much higher with a value of 95%. Another striking value is the precision for the negative class with 100%. The corresponding confusion matrix emphasizes the results given in the classification report as well, since the diagonal line can be easily recognized. There is only one observation, which was predicted as a positive tweet but in fact was a negative tweet.

	precision	recall	f1-score	support
0	1.00	0.90	0.95	10
1	0.90	1.00	0.95	9
accuracy			0.95	19
macro avg	0.95	0.95	0.95	19
weighted avg	0.95	0.95	0.95	19

Figure 21: Classification report of the logistic regression model using custom data set

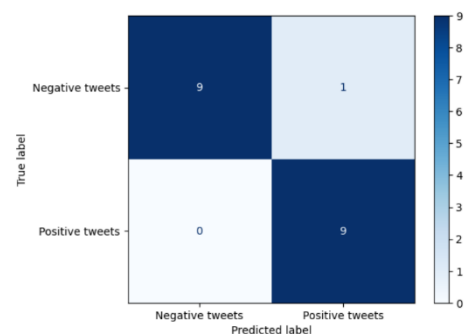


Figure 22: Confusion matrix of logistic regression model using custom data set

Naive Bayes results

	precision	recall	f1-score	support
0	0.76	0.78	0.77	197264
1	0.77	0.76	0.77	198159
accuracy			0.77	395423
macro avg	0.77	0.77	0.77	395423
weighted avg	0.77	0.77	0.77	395423

Figure 23: Classification report of naive Bayes model

Looking at the classification report of the naive Bayes model, it is clear, that both models perform similar on the test set. The naive Bayes model has a slightly lower accuracy with 77%. Nevertheless, the Bayes model seems to recognize negative tweets better than the logistic regression model, which is portrayed by the recall with 78%. On the other hand, the recall of the positive tweets is only 76%, which is 4% lower than the logistic regression model. However, it is hard to tell, which model performs better. Only the F1 score, which is the harmonic mean of recall and precision, is in both classes 77%, which is in fact, lower than the percentages of the logistic model. These were 78% for the negative and 79% for the positive class.

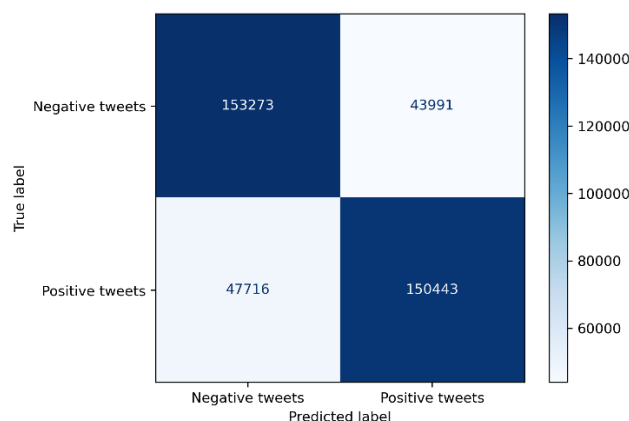


Figure 24: Confusion matrix of naive Bayes model

Just like the regression model, the confusion matrix of the Bayes model represents the values in the classification report and its values look similar to the values of the logistic regression model. Here, the diagonal line is also very clear. The model predicts 43.991 tweets as positive, whose true label are negative and incorrectly classifies 47.716 tweets as negative.

The Bayes model was tested on the custom data set as well, but since the results were exactly the same as the logistic regression model, both classification report and confusion matrix are not further discussed in this chapter.

Comparison

To see, which model performs better, another evaluation metric can be analyzed. In this case, I chose the ROC AUC. ROC stands for “receiver operating characteristic curve” and plots the true positive rate against the false positive rate at different thresholds in a graph, representing the performance of a classification model. AUC is short for “area under the ROC curve” and measures the whole area

underneath the calculated ROC curve. This is usually computed to compare different models and see, which one is performing better [11].

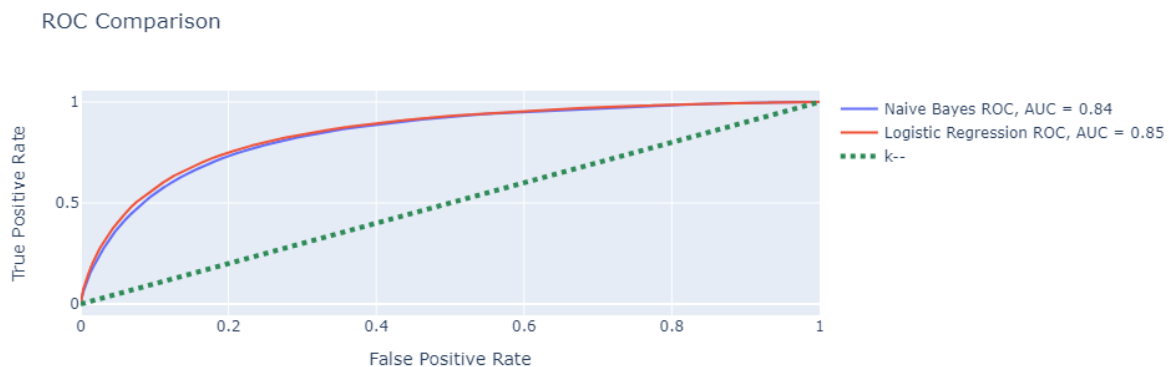


Figure 25: ROC comparison of naive Bayes and logistic regression model

In the figure above, both roc curves, one of the naive Bayes and one of the logistic regression model, which are represented by the red and the blue line, are plotted together. In addition, a chance model, which randomly guesses labels, is added represented by the green dotted line. This helps, to see whether our own model is performing better or not. As the blue and red lines show, there is only a minimal difference between both. Nevertheless, the values of the red line, the logistic regression roc, is slightly above the blue line at a false positive rate with around 0.1. In addition, the AUC value of the logistic regression model is 0.01 higher than the naïve bayes', which has an AUC value of 0.84. Therefore, the logistic regression model performs slightly better than the naive Bayes model.

Conclusion

As the results show, both models perform almost equally on the test data set with accuracies of 77% and 78%, which is good for the beginning. Nevertheless, both models can be improved to get a higher accuracy. Regarding this, I played with the library "RandomSearchCV", which allows, trying different hyperparameters for a model and find the most suitable, where the accuracy is maximized.

Tuning hyperparameters

```
[27]: kf = KFold(n_splits=5, shuffle=True, random_state=42)

param_grid = [
    {
        'penalty' : ['l1', 'l2'],
        'C' : np.logspace(-4, 4, 20),
        'solver' : ['liblinear']}

# Instantiate the RandomizedSearchCV object
logreg_cv = RandomizedSearchCV(lr, param_grid, cv=kf)

# Fit the data to the model
logreg_cv.fit(X_train, y_train)

# Print the tuned parameters and score
print("Tuned Logistic Regression Parameters: {}".format(logreg_cv.best_params_))
print("Tuned Logistic Regression Best Accuracy Score: {}".format(logreg_cv.best_score_))
```

Figure 26: Parameter tuning of the logistic regression model

As the screenshot shows, a variable called "param_grid" is defined, where different values for the hyperparameters "penalty", "C" and "solver" are set. While training the model,

“RandomizedSearchCV” goes through the specified hyperparameters using cross-validation, which is defined in the first line of code and finds the best parameters as a result. Nevertheless, after two attempts using different param_grids, the accuracy of the logistic regression model remained the same with 78%.

Therefore, I would recommend going back to the data preprocessing step. As the plots in the dashboard, which was presented in the chapter “Visualization” show, there are still many words after preprocessing, which do not hold any information regarding the sentiment of a text. Looking at the plot, which shows the distribution of the most frequent used words in negative tweets, “get” scores the highest, although it does not have any sentiment. The data used to train the models is therefore not cleaned “enough” and could be improved. This also includes integrating additional methods used in document analysis such as “Lemmatization”, where the “Lemma” or the base form of a word is extracted, and “Stemming”, where the last characters of a word are cut off in order to extract the stem [12]. In addition, a method could be implemented, which looks for typos in a text, preventing the model to recognize and therefore classify the text correctly. Another idea would be to take the context or combination of words into consideration. Just because a text contains negative words, does not mean that the overall sentiment must be negative as well.

Bibliography

- [1] *Sentiment Analysis Guide*. (n.d.-b). MonkeyLearn. Retrieved September 23, 2022, from <https://monkeylearn.com/sentiment-analysis/>
- [2] GeeksforGeeks. (2022, July 7). *Using CountVectorizer to Extracting Features from Text*. Retrieved September 23, 2022, from <https://www.geeksforgeeks.org/using-countvectorizer-to-extracting-features-from-text/>
- [3] Yadla Jaswanth, Ravilla Muni Sandeep Kumar, Ravilla Madhu Sudhan,, Mr. Vijaya Kumar S and Mr. Rajagopalam D, "Sentiment analysis using logistic regression algorithm", European Journal of Molecular & Clinical Medicine, Volume 7, Issue 4, 2020
- [4] *Sentiment Analysis Using Bag-of-Words — ENC2045 Computational Linguistics*. (n.d.). Retrieved September 23, 2022, from https://alvinntnu.github.io/NTNU_ENC2045_LECTURES/nlp/ml-sklearn-classification.html
- [5] Zhang, D. (2021, December 15). *Sentiment Classification with Logistic Regression — Analyzing Yelp Reviews*. Medium. Retrieved September 23, 2022, from <https://towardsdatascience.com/sentiment-classification-with-logistic-regression-analyzing-yelp-reviews-3981678c3b44>
- [6] Raj, A. (2021, December 16). *Perfect Recipe for Classification Using Logistic Regression*. Medium. Retrieved September 23, 2022, from <https://towardsdatascience.com/the-perfect-recipe-for-classification-using-logistic-regression-f8648e267592>
- [7] Jason Brownlee. (2016, April 1). *Logistic Regression for Machine Learning* . Retrieved September 23, 2022, from <https://machinelearningmastery.com/logistic-regression-for-machine-learning/>
- [8] *Multinomial Naive Bayes Explained: Function, Advantages & Disadvantages, Applications in 2022*. (2022, September 16). upGrad Blog. Retrieved September 23, 2022, from <https://www.upgrad.com/blog/multinomial-naive-bayes-explained/>
- [9] Sheong, S. (2020, June 5). *Sentiment analysis with a simple naive Bayes classifier in Go*. Medium. Retrieved September 23, 2022, from <https://towardsdatascience.com/sentiment-analysis-with-a-simple-naive-bayes-classifier-in-go-6c18a0134a1c>
- [10] Kohli, S. (2022, May 3). *Understanding a Classification Report For Your Machine Learning Model*. Medium. Retrieved September 23, 2022, from <https://medium.com/@kohlishivam5522/understanding-a-classification-report-for-your-machine-learning-model-88815e2ce397>

[11] *Classification: ROC Curve and AUC | Machine Learning* /. (n.d.). Google Developers. Retrieved September 23, 2022, from <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>

[12] *Stemming vs Lemmatization in NLP: Must-Know Differences*. (2022, July 1). Analytics Vidhya. Retrieved September 23, 2022, from <https://www.analyticsvidhya.com/blog/2022/06/stemming-vs-lemmatization-in-nlp-must-know-differences/#:%7E:text=Stemming%20is%20a%20process%20that,%20would%20return%20'Car'>.