Table of content

Analysis of subreddit interactions

A project task in Data Science for Digital Humanities 2

Abstract

This document examines how a social graph of link sentiments in subreddits can be used to make predictions on the test data of the graph. In a second analysis, machine learning is used to make predictions.

1 Introduction

This paper explores the potential of subreddit interactions for sentiment analysis. Sentiment analysis is “the process of using algorithms and computer technologies to systematically detect, extract, and classify the subjective information and affective states expressed in a text, such as opinions, attitudes, and emotions regarding a service, product, person, or topic” (Lei et al [1]). The provided data only consists of positive and negative sentiment scores (+1 and -1) and examines the correlation between a source subreddit and a target subreddit. This agrees with the main goal of sentiment analysis as it “is an evaluation mainly in positive vs. negative polarity terms” (Lei et al [1]).

The code used for this work was written in Python with Anaconda, Jupyter Notebooks. The project task consisted of three subtasks: A preliminary data analysis (file final_project_preliminary.ipynb) to analyze the data, building a directed social graph of subreddits (file final_project_subtask1.ipynb) while you had to predict positive and negative sentiments for the test posts based on the graph without numeric descriptors of post properties and the final task applying machine learning algorithms to predict message sentiment (file final_project_subtask2.ipynb). The data investigation will be completed with a statistical comparison of the models.

This data analysis relies on a limited subreddit collection[1] for the training data consisting of an amount of more than 280,000 lines (the data set contains 313,600,538 posts) while the test data only shows 5000 posts. As we will see later on even a compressed training data set will already be computationally intensive.

As soon as the whole data set was analyzed this paper will move on to the next step: building the social directed graph. Building a graph from a training set can become quite a challenge for a low budget computer or a laptop. For this data analysis first a slow four-cores-laptop with 8 GB RAM and Arch Linux installed was used then a desktop computer with a six-core AMD Ryzen 5600X, 32 GB RAM (operating system: Windows 11). As expected, the 5600X performed much better but it still took too much time whether it was subtask 1 or subtask 2. So the whole code execution had to be aborted after running several hours. The solution was the use of Google Colab which showed a much better performance on all executions no matter if it had to deal with less or more intense computations. Personal experience showed that it is not necessary moving to Colab when analyzing small data sets like the Titanic or the Breast Cancer data set. For mid-large or large data sets you are advised to own a fast multi-core processor and graphics card with much tensor cores or you should just use Colab.

[1] http://snap.stanford.edu/data/soc-RedditHyperlinks.html

2 Preliminary data analysis

2.1 Taking a closer look at the training data

Before working with a data set, it is important to know what the data consists of and what it is going to tell the user. You should take a neutral look a all given information that you can define which one is relevant to your goal.

As a first step it was necessary to convert the training data as a pandas data frame via *pd.read_csv()*. This made sense because it had already been stored as a tsv-file. Getting some common information about the data frame *df.info()* was used to display all column names and their types.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 281562 entries, 0 to 281561
Data columns (total 6 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   SOURCE_SUBREDDIT  281562 non-null  object
 1   TARGET_SUBREDDIT  281562 non-null  object
 2   POST_ID           281562 non-null  object
 3   TIMESTAMP         281562 non-null  object
 4   LINK_SENTIMENT    281562 non-null  int64
 5   PROPERTIES        281562 non-null  object
dtypes: int64(1), object(5)
memory usage: 12.9+ MB
```

Figure 1: general information about the data types (columns)
source: final_project_preliminary.ipynb

Figure 1 points out that there is a TIMESTAMP column. Checking this column for the *min()* and *max()* values reveals that the earliest entry is from 2014-01-01 10:08:48, the latest entry from 2017-04-30 16:58:21. We can see that the data have been gathered within three years. There is even more to find out.

The following Screenshot (figure 2) from Jupyter Notebooks offers a view at the main exploratory goal of this project: the LINK_SENTIMENT column. The shape of the data frame points out 281,562 lines and six columns. Furthermore interesting will be the statistical features of the sentiments showing both positive and negative features (maximum value: +1, minimum value: -1). The mean is closer to 1, while the standard deviation is close to 0. If a standard deviation is close to 0 it “indicates that data points are close to the mean, whereas a high or low standard deviation indicates data points are respectively above or below the mean.” (National Library of Medicine [2]).

```
(281562, 6)
1
-1
0.8530483516951861
0.5218328054159836

amount of positive and negatie values in column: LINK_SENTIMENT
 1    260874
-1     20688
Name: count, dtype: int64
```

Figure 2:
source: final_project_preliminary.ipynb

2.2 Data cleaning on the training data

If one deals with data, it is commonly useful, mostly necessary to clean the data. Data can contain empty values (NaN values) duplicate entries or even junk (e.g., someone aged 999 years). A deeper analysis of the provided data set showed that it is not an issue to clean the data. For the observation *isna()* and *duplicated().any()* both stated out a *False* which is good and means that the data has already been clean or cleaned by somebody.

As mentioned before the *duplicated()* command finds all duplicates in a Pandas Data Frame. This command can be used to find duplicate subreddit names in columns, showing that there are more paths leading to one specific node regardless of whether it is a source node or a target node. According to figure 3 there are 253,956 identical source subreddits in contrast to 261,115 target subreddits. Even the TIMESTAMP column confirms 37,489 duplicates.

```
SOURCE_SUBREDDIT
True     253956
False     27606
Name: count, dtype: int64

TARGET_SUBREDDIT
True     261115
False     20447
Name: count, dtype: int64

POST_ID
False    254511
True      27051
Name: count, dtype: int64

TIMESTAMP
False    244073
True      37489
Name: count, dtype: int64

PROPERTIES
False    244180
True      37382
Name: count, dtype: int64
```

Figure 3: statistics of the columns
source: final_project_preliminary.ipynb

	SOURCE_SUBREDDIT				TARGET_SUBREDDIT				POST_ID				TIMESTAMP				PROPERTIES			
	count	unique	top	freq	count	unique	top	freq	count	unique	top	freq	count	unique	top	freq	count	unique	top	freq
LINK_SENTIMENT																				
-1	20688	4119	subredditdrama	1418	20688	3885	askreddit	867	20688	18547	3yj2ee	53	20688	18099	2015-12-27 20:14:14	53	20688	18244	17234.0,16374.0,0.732041313682,0.0276778461181...	53
1	260874	27027	subredditdrama	3183	260874	19959	askreddit	6254	260874	235964	4asjoos	167	260874	227930	2014-08-20 14:40:53	360	260874	226640	39838.0,38917.0,0.656057030975,0.064335559014,...	167

Figure 4: examination of the link sentiments (training data)
source: final_project_preliminary.ipynb

2.3 Statistics for the positive and negative sentiment values

The focus in 2.3 relies on the positive and negative sentiments. Thus the *describe()* command was used on the LINK_SENTIMENT column figuring out all important information. Figure 4 outputs 4,119 negative unique values and 27,027 positive unique values for the source subreddits in contrast to the target subreddit which contains 3,885 negative unique values and 19,959 positive unique values. The most common value for both positive and negative of the source subreddits is the post called *subredditdrama* (frequency: 3183 positive and 1418 negative), for the target subreddits the *askreddit* post (frequency: 6,254 positive and 867 negative).

2.4 Examination of the property column

The property column shows statistical values of characters and word counts of the reddit posts. The Python code offers an analysis of the whole properties. Therefore, the most common statistics were

```
Number of characters (without counting
white space):
Maximum value: 9998.0
Minimum value: 100.0

Number of words:
Maximum value: 999.0
Minimum value: 10.0

Average word length:
Maximum value: 9.96428571429
Minimum value: 1.78751857355

Number of sentences:
Maximum value: 99.0
Minimum value: 1.0

Average number of characters per sen-
tence:
Maximum value: 999.857142857
Minimum value: 10.0680100756

Average number of words per sentence:
Maximum value: 999.0
Minimum value: 0.210084033613
```

Figure 5: selected properties values
source: final_project_preliminary.ipynb

copied and pasted from the result of the code (see figure 5).

2.5 Observing the test data

This paragraph will not be as long as the preceding one because most of the code results like *df.info()* (information about the columns) have already been discussed so far. Furthermore, since data cleaning was not crucial on the training data, it will not be crucial on the test data either. A more relevant statistical exploitation might be regarding statistics like minimum, maximum, mean and standard deviation. Besides the new outcome of the positive and negative values on the LINK_SENTIMENT column or the different information gained from the PROPERTIES column should be taken into consideration.

Taking a look at the *df.shape()* property reveals that the test data now consists of 4999 rows and 6 columns. Certain features and relations were observed in figure 6:

```
shape:(4999, 6)
max value: 1
min value: -1
mean value: 0.8471694338867773
standard deviation value: 0.5313759814588873

amount of positive and negatie values in column: LINK_SENTIMENT
 1    4617
-1     382
Name: count, dtype: int64

amount of positive values (test data): 0.9235847169433887
amount of negative values (test data): 0.07641528305661133
amount of positive values (training data): 0.9265241758475931
amount of negative values (training data): 0.07347582415240693
differences test data/training data (positive values): 0.0029394589042043284
differences test data/training data (negative values): 0.002939458904204398

deviations/relations test data/training data (mean, standard deviation):
mean relation: 0.9931083416353527
mean deviation: 0.006891658364647335
std relation: 0.9820406334198566
std deviation: 0.018287803955322923
```

Figure 6: statistics of the sentiment scores
source: final_project_preliminary.ipynb

As demonstrated in figure 6, there is only a slight difference (approx. 0.294 %) between the relational amount of positive and negative values in both the test and training data. This makes it well-suited for further data analysis, especially for machine learning. While the mean deviates by approximately 0.7 percent, the standard deviation is close to 1.8 percent.

	SOURCE_SUBREDDIT				TARGET_SUBREDDIT				POST_ID				TIMESTAMP				PROPERTIES			
	count	unique	top	freq	count	unique	top	freq	count	unique	top	freq	count	unique	top	freq	count	unique	top	freq
LINK_SENTIMENT																				
-1	382	184	circlebroke	31	382	207	askreddit	25	382	349	1v3273	9	382	344	2014-01-12 20:30:55	9	382	347	9956.0,8601.0,0.787364403375,0.00693049417437,...	13
1	4617	1786	dailydot	140	4617	1458	askreddit	183	4617	4232	1wt4ots	23	4617	4090	2014-01-17 14:40:58	124	4617	4151	3992.0,3774.0,0.740731462926,0.0225450901804,0...	23

Figure 7: examination of the link sentiments (test data)
source: final_project_preliminary.ipynb

As In Section 2.3, we have already analyzed, the LINK_SENTIMENT column on the training data. Now we are doing the same with the test data using the *describe()* command in Python. In comparison to figure 4, figure 7 presents different source subreddits for both the positive and the negative sentiments (in the screenshot from the training data, *subredditdrama* occurred in the rows with +1 and -1 sentiment scores). While there are differences in the source subreddits, the *askreddit* post once again shows the most frequent posts in the training data.

3 Building the social graph of subreddits

3.1 Implementation on the Training Data

After the data analysis in section two, it is time to build the social graph. The PROPERTIES column should be excluded, so we need to work with the selected columns of source, target, link sentiment and name it “df_select”. The command looks like this:

df_select=df[["SOURCE_SUBREDDIT","TARGET_SUBREDDIT","LINK_SENTIMENT"]].

The result is a cleaned data frame (the first five rows are shown in figure 8):

	SOURCE_SUBREDDIT	TARGET_SUBREDDIT	LINK_SENTIMENT
0	israel	palestine	1
1	vertcoin	cryptocurrency	1
2	nofap	explainlikeimfive	1
3	explainlikeimcalvin	pics	1
4	aneros	askgaybros	1

Figure 8: cleaning the data frame
source: final_project_subtask1.ipynb

The LINK_SENTIMENT column now consists of 1 and -1 values. To improve the appearance, the values 1 and -1 were replaced with “positive” and “negative” string assignments (figure 9).

	SOURCE_SUBREDDIT	TARGET_SUBREDDIT	RATING
0	israel	palestine	positive
1	vertcoin	cryptocurrency	positive
2	nofap	explainlikeimfive	positive
3	explainlikeimcalvin	pics	positive
4	aneros	askgaybros	positive

Figure 9: cleaned data frame
source: final_project_subtask1.ipynb

Thanks to these improvements, it is now possible to differentiate between the positive and negative values by creating a bar chart. (figure 10):

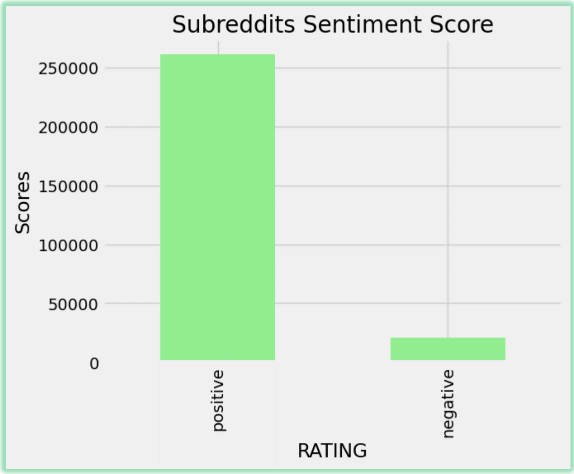

Figure 10: cleaning the data frame (training data)
source: final_project_subtask1.ipynb

According to the bar chart created with matplotlib, the positive sentiments clearly outnumber the negative sentiments. Sometimes, a graphical presentation can be a better choice than looking at numbers, as it can give the spectator a better overview of the data distribution.

A major problem happened after plotting the directed graph. The code for figure 11 is marked as a comment in the Jupyter Notebook file. It took several hours to create the graph, but it was not possible to interpret the plot because the data points were

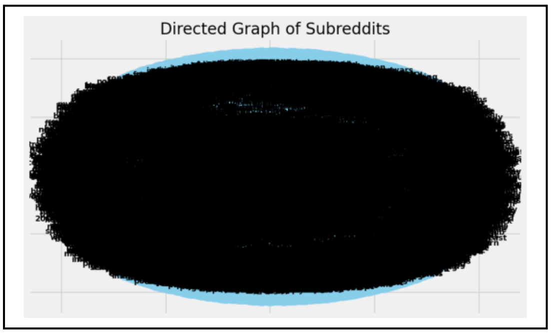

Figure 11: plot of the directed graph (training data)
source: final_project_subtask1.ipynb

overlapping. Reducing the amount of data did not get in any better results.

A more meaningful solution was found by using the t-Distributed Stochastic Neighbor Embedding (t-SNE) which “tries to place the objects in a low-dimensional space so as to optimally preserve neighborhood identity” (Roweis et al [3]). As you can see in figure 11 points are far too close, they overlap. That is why t-SNE should be preferred when dealing with large data sets. Van der Maaten et al [4] points out that “the visualizations produced by t-SNE are significantly better than those pro-

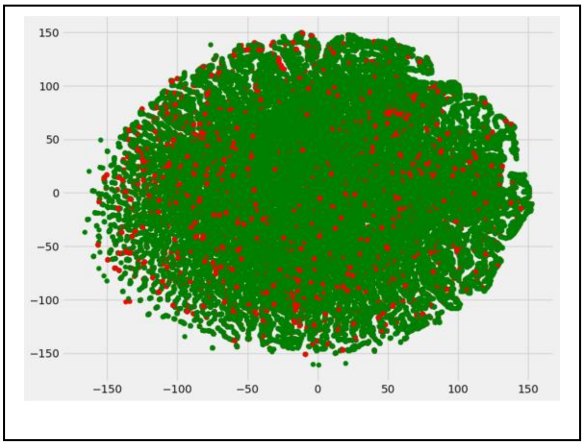

Figure 12: using t-SNE on the training data
scource: final_project_subtask1.ipynb

duced by the other techniques on almost all of the data sets”. A fully directed graph is not advisable. Figure 12 proves the partition of positive and negative sentiments. There is still overlapping but it is not just a black bunch with some blue dots as in figure 11.

To get a better overview of the training data, it was cut to only 1,000 rows using the Python slicing command *df[:1000]*. Figure 13 displays that there

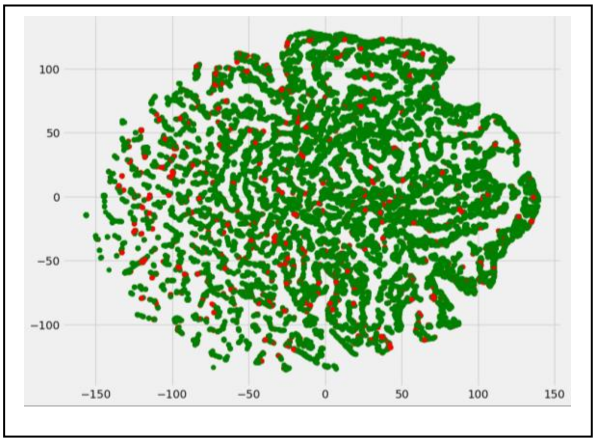

Figure 13: using t-SNE on the training data
scource: final_project_subtask1.ipynb

are more mostly positive sentiments clustered around negative sentiments containing some smaller gaps. The single clusters (red and green dots together) can be interpreted as positive and negative sentiments based on their direct relations. As known from the SOURCE_SUBREDDIT and TARGET_SUBREDDIT columns show that a single name can have multiple relations. The advantage of t-SNE is the fast and strong visualization while a disadvantage might be the missing direction from the source post to the target post.

3.2 Implementation on the test data

The implementation on the test data followed the same steps as in paragraph 3.1, which are not being further explained here. This will lead directly to the comparison of the bar plots.

The bar plot in figure 14 has a similar sentiment distribution to the training data (figure 10). The similarity of the bar plots proves that the amount of data taken from the training set was a good choice.

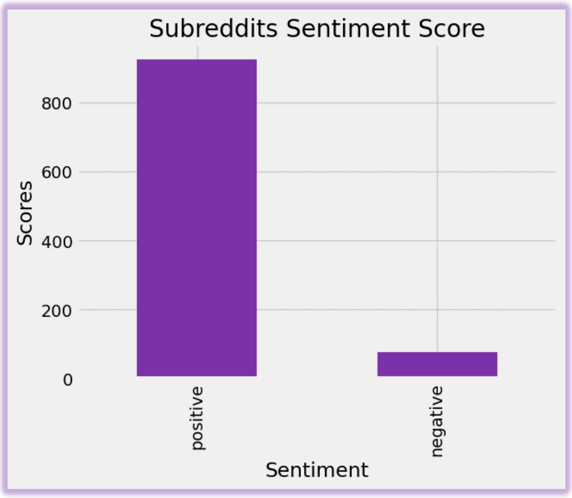

Figure 14: bar plot with negative and positive sentiments (test data)
source: final_project_subtask1.ipynb

The test set only contains 5,000 rows which means the graph should be smaller and easier to interpret. This is because there is less data to process, which makes the graph less crowded and easier to see the trends (figure 15). The clustering in this compressed graph makes it more obvious to see the different clustering of the polarized dots.

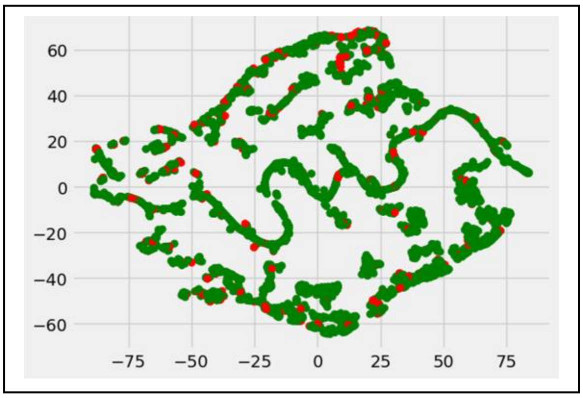

Figure 15: using t-SNE on the test data
scource: final_project_subtask1.ipynb

As a final task the predictions on the graph had to be made. The logistic regression model was utilized for the predictions because it is a common model in machine learning. It performs well with high speed and accuracy, especially on mid-large or large data sets. First, the t-SNE algorithm had to be programmed. Then, user input was used to enter the coordinates of a certain data point which finally made the prediction on the graph (see code above). Entering “3” and “4” for instance lead to a positive prediction.

```
#create a new column with the values of the subreddits
df["SUBREDDIT_ID"] = pd.factorize(df["SOURCE_SUBREDDIT"])[0]
df["TARGET_SUBREDDIT_ID"] = pd.factorize(df["TARGET_SUBREDDIT"])[0]

#make the t-SNE embedding
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
embedding = tsne.fit_transform(df[["SUBREDDIT_ID", "TARGET_SUBREDDIT_ID"]].values)

#define the target variable
target = df["RATING"]

#create the model
model = LogisticRegression()

#fit the model
model.fit(embedding, target)

#make a prediction for a new data point (e.g., new_subreddit = np.array([1, 2]))
#dp = data point
dp1 = int(input("Enter the first data point..."))
dp2 = int(input("Enter the second data point..."))
new_subreddit = np.array([[dp1, dp2]])
prediction = model.predict(new_subreddit)

#print the prediction
print(prediction)
```

Figure 16: using t-SNE on the test data
scource: final_project_subtask1.ipynb

4 Sentiment analysis of messages using machine learning

4.1 Finding the right model

There are many different machine learning models available, but the challenge is to choose the right one for the specific task at hand. Some perform better but are missing some accuracy. This paragraph will help determining the best model for the sentiment analysis of the data provided. For this project the models of logistic regression, Naïve Bayes, decision trees and random forest were applied (source: final_project_subtask2.ipynb).

Logistic regression is simply described as “observations based on quantitative features; predict target class or probabilities of target classes” (Brink et al [5]. It uses binary values 0 and 1 to predict certain properties. In the case of a subreddit sentiment analysis 0 represents negative, 1 positive. Jung [6] depicts logistic regression as “learning a linear hypothesis map to predict a binary label based on numeric features of a data point. The logistic regression of a linear hypothesis map (classifier) is measured using its average logistic loss on some labeled data points (the training set)”.

The Naïve Bayes model is said to be very fast and effective. That was the reason for taking it into consideration. Stuart et al [7] clarifies the model as “it assumes that the attributes are conditionally independent of each other when the class is known [...] one can make a deterministic prediction by choosing the most likely class [...] the method learns relatively well, but not as well as decision tree learning [...] however, it achieves a surprisingly good performance [...] it is among the most effective universal learning algorithms [...] it does not have any difficulties with noisy or erroneous data”.

Decision trees are described by Jung et al [6]: A decision tree “is a flow-chart like representation of a hypothesis map h [. . .] a decision tree is a directed graph which reads in the feature vector x of a data point at its root node. The root node then forwards the data point to one of its children nodes based on some elementary test on the features x. If the receiving children node is not a leaf node, i.e., it has itself children nodes, it [sic!] decision tree another test. Based on the test result, the data point is further pushed to one of its neighbors. This testing and forwarding of the data point is repeated until the data point ends up in a leaf node (having no children nodes). The leaf nodes represent sets (decision regions) constituted by feature vectors x that are mapped to the same function value h(x).” The results can be very accurate nevertheless it can take much longer than the logistic regression or the Naïve Bayes algorithms.

The Random Forest Model (RFM) is a group of decision trees, as the name already provokes. Each decision tree in the RFM is used to make predictions using the classes of the input data. The whole thing works through a ranking system, where the class that contains the most informative data gets the most points and is used for the prediction. The algorithm looks like this (Sing et al [8]):

1. Multiple subsets of attributes from original set of attributes with replacement are created.
2. Each subset is selected one by one from group of subsets and Decision Tree is built.
3. The class predicted by majority of the Decision Trees is considered as output of the model.

4.2 Choosing metrics

Metrics in machine learning evaluate the performance of a model and measure how well the model is able to make predictions on new data. Many different metrics can be used, depending on the type of model and the task that it is being used for. For this project the most common metrics were used: accuracy, precision, recall and F1-score.

Saleh [9] describes accuracy, precision and recall as follows: Accuracy "involves comparing the actual prediction to the real value" which is the ratio between true positives + false negatives and the sum of all other possibilities (TP+TN+FN+FP). Precision is defined "the model's ability to correctly classify positive labels (the label that represents the occurrence of the event) by comparing it to the total number of instances predicted as positive […] the ratio between the true positives and the sum of the true positives and false positives". Recall measures "the number of correctly predicted positive labels against all positive labels. This is represented by the ratio between true positives and the sum of true positives and false negatives".

The F1-score is a metric that provides a good balance between precision and recall. Its use makes sense in many machine learning studies due to its consideration of both precision and recall. If a high F1-score is obtained, it is a "precise solution," while values around 50% represent a random result" (Papp et al [10]).

4.3 Significance Testing

Testing which model performed better different tests like the T-Test, Wilcoxon, Mann-Whitney U and the F-Test were realized. Kaptein et al [11] has written an interesting overview comparing those tests:

1. *The one-sample t-test for testing the hypothesis that the population mean μ is equal to the known value μ0, with y¯ the sample average and s the sample standard deviation.*

   $(\bar{y} - \mu_0)/(s/\sqrt{n})$

2. *The F-test for testing the hypothesis that the variances σ 2 1 and σ22 from two independent populations are equal, with sk the sample standard deviation of population k. A non-parametric version of the F-test, called Levine's test, is also introduced.*

   s_1^2/s_2^2

3. *The Wilcoxon signed rank test for testing the hypothesis that differences from paired samples are on average equal for positive and negative differences.*

   U

4. *The Mann-Whitney U test for testing the hypothesis that the values of one population are not stochastically larger than the values of another population (two independent samples).*

   W^+

4.4 Evaluation with different models

The evaluation was done with all models but not all performed well. The results were collected from the code result of the file *final_project_subtask2.ipynb*.

4.4.1 Logistic Regression

The logistic regression ran very fast (2 minutes 5s, stats see Appendix A) on the training data. The test data did not score perfectly for accuracy and precision, but it was still good. You can see that accuracy and precision are lower than recall and F1-score. For all metrics a cross validation score was applied (results in square brackets), then calculated by its mean:

```
CV Acurracy Scores: [0.913     0.911     0.915     0.905     0.8958959]
CV Precision Scores: [0.9248731  0.92299899 0.92323232 0.92244898 0.92435233]
CV Recall Scores: [0.98593074 0.98593074 0.99024919 0.97941495 0.96641387]
CV F1 score Scores: [0.9544264  0.95342752 0.95556717 0.95007882 0.94491525]

Acurracy: 0.9079791791791791
Precision: 0.9235811435411703
Recall: 0.9815878956724026
F1 score: 0.9516830350485922
```

As mentioned in section 2 the test data consists of 4999 rows. After having checked whether the predictions were correct the test data reached a score of 0.9954 with 4976 correct predictions:

```
4976
0.9953990798159632
```

Generally, models have a predefined setting that can perform quite well. However, sometimes it is necessary to manually adjust the parameters to find the perfect model (the fastest and most accurate model). Entering a threshold value or applying a K-fold cross validation can help improve the performance. The best values were tested and stored as a text file, which is copied into Appendix B. The standard value for the threshold is 0.5, which scored best for values of 0.5 and below. Values above 0.5 lower the precision and accuracy of the predictions. For k-fold cross-validation (standard: 5), a value of 10 worked best.

A grid search was implemented to find the best parameters. The logistic regression performed much faster than the decision tree and random forest models. Nevertheless, it took approximately 36 minutes to finish running the code. The best parameters according to grid search were a C value of 0.1 and a penalty value of 12, which resulted in a score of 0.9257996456984546.

An evaluation of the significance tests concluded:

1. T-test:
```
T-statistic: 9.643142552897022
P-value: 6.545621444045767e-22
```
2. F-test:
```
F-test: T-statistic: 92.99019829549322
P-value: 6.545621444027013e-22
```
3. Wilcoxon test:
```
Wilcoxon signed-rank test: T-statistic: 181.0
P-value: 3.025692422953373e-80
```
4. Mann-Whitney U test:
```
Mann-Whitney U test: T-statistic: 12616441.0
P-value: 0.0641505178871114
```

4.4.2 MultinomialNB (Naïve Bayes)

The Naïve Bayes model ran fastest in only 40 seconds on the training data. However, the test data scored worst for accuracy and precision compared to all models. Only precision scored above 90 percent. Accuracy is close to 80 percent while recall and F1 score are above the mid 80 percent.

```
CV Acurracy Scores: [0.816      0.801      0.795      0.808      0.81081081]
CV Precision Scores: [0.92528736 0.92200233 0.92235294 0.92549476 0.94430993]
CV Recall Scores: [0.87121212 0.85714286 0.84940412 0.86132178 0.84507042]
CV F1 score Scores: [0.8974359  0.88839035 0.88437676 0.89225589 0.89193825]

Acurracy: 0.8061621621621621
Precision: 0.9278894628996399
Recall: 0.8568302589429349
F1 score: 0.8908794312014059
```

The test data reached a score of only 0.915 with 4574 correct predictions:

```
4574
0.9149829965993198
```

The best values for the threshold and the k-fold cross-validation were tested and stored as a text file which is copied into Appendix B (as in paragraph 4.4.1). Regarding the threshold values it is most intriguing that only values close to zero scored best (approx. 0.92). For the k-fold cross-validation score a value of 15 performed best on all metrics.

The grid search on Naïve Bayes did the fastest run of all models (it took only 26 seconds). The grid search on Naïve Bayes completed the fastest run of all models (in 26 seconds). Grid search suggested the best parameters as *alpha=10, class_prior=[0.5, 0.5]*, and *fit_prior=True*, achieving an accuracy of 0.9142035825717251.

Here is what the evaluation of the significance tests showed:

1. T-test:
```
T-statistic: 8.705915124808351
P-value: 3.647995553464554e-18
```
2. F-test:
```
F-test: T-statistic: 75.79295816036671
P-value: 3.647995553452847e-18
```
3. Wilcoxon test:
```
Wilcoxon signed-rank test: T-statistic: 43780.0
P-value: 1.6744310778941882e-30
```
4. Mann-Whitney U test:
```
Mann-Whitney U test: T-statistic: 12533335.0
P-value: 0.5674984510460992
```

4.4.3 Decision Tree

In relation to the Decision Tree model, it took some time on the training data (2 hours and 8 minutes). What we could expect from such a long execution time is a better score on the metrics. The following results will show whether this is the case.

The results show a worse score on the metrics compared to results scored by the linear regression despite the long execution of code. Nevertheless, it is still much better than the Naïve Bayes scores. Accuracy is even below 90 percent.

```
CV Acurracy Scores: [0.888      0.887      0.882      0.889      0.87987988]
CV Precision Scores: [0.92371996 0.92364017 0.92364017 0.92561983 0.92592593]
CV Recall Scores: [0.96536797 0.96536797 0.96641387 0.95991333 0.94799567]
CV F1 score Scores: [0.94379639 0.94664554 0.94211365 0.94199042 0.9374666 ]

Acurracy: 0.8851759759759761
Precision: 0.9245092107134839
Recall: 0.9610117581948566
F1 score: 0.9424025189889825
```

The test data scored 4990 correct prediction which is more than 99.82 percent and thus slightly better than the scores of the logistic regression (4976 and 99.54 percent):

```
4990
0.9981996399279855
```

Concerning the threshold values (Appendix B) only values close to zero scored best (approx. 0.92). We have encountered this phenomenon already in 4.4.2 Naïve Bayes. For the k-fold cross-validation score a value of 11 performed best on all metrics.

The grid search on the Decision Tree model was far too slow, so slow that it was even necessary to downsize the training data to 10,000 rows. Despite that downscaling of data, the code still took 33 minutes to run. It is remarkable that in comparison to the logistic regression which took 36 minutes on the whole data the Decision Tree algorithm almost took the same time on a part of the data of 10,000 rows. The best parameters recommended by grid search were *max_depth=5*, *max_leaf_nodes=10*, and *min_samples_leaf=1* with a score of 0.9218.

Here is what the evaluation of the significance tests showed:

1. T-test:
```
T-statistic: -0.26239479148805606
P-value: 0.7930224950035902
```
2. F-test:
```
F-test: T-statistic: 0.0688510266000634
P-value: 0.7930224950029514
```
3. Wilcoxon test:
```
Wilcoxon signed-rank test: T-statistic: 5.0
P-value: 0.019630657257290667
```
4. Mann-Whitney U test:
```
Mann-Whitney U test: T-statistic: 12477504.0
P-value: 0.7930134476401346
```

4.4.4 Random Forest

The Random Forest model consists of many decision trees, so we could expect the code to run much slower than a single Decision Tree model. This assumption was confirmed by the longest execution time on the training data of all models (5 hours and 46 minutes). As the Decision Tree algorithm could not approve a better score on the metrics it will be interesting to see if the Random Forest makes it different. In fact, Random Forest hit the best scores on all metrics. Especially the recall score is close to 100 percent:

```
CV Acurracy Scores: [0.924      0.924      0.924      0.924      0.92192192]
CV Precision Scores: [0.924      0.924      0.92392392 0.92392392 0.92462312]
CV Recall Scores: [1.         1.         1.         1.         0.99674973]
CV F1 score Scores: [0.96049896 0.96049896 0.96045786 0.96045786 0.95933264]

Acurracy: 0.9235843843843844
Precision: 0.9240941926851475
Recall: 0.9993499458288191
F1 score: 0.9602492543923686
```

Surprisingly the check whether predictions are correct was not as good as the score from the Decision Tree model as the Random Forest proved one value less than the corresponding Tree model, hitting a score of approx. 99.8 percent:

```
4989
0.997999599919984
```

The threshold values (Appendix B) values close to zero scored best as seen for Naïve Bayes and Decision Tree before (approx. 0.92). A good choice for the k-fold cross-validation would be a value of 20, which achieved the best performance.

The grid search on Random Forest model was excessively slow. As applied to the Decision Tree model the training data had to be downsized to 10,000 rows. The execution time took twice as long as the Decision Tree model (1 hour). The best parameter found using grid search was *n_esti=25*, which achieved a score of 0.9224

The significance tests displayed the following outcome:

1. T-test:
```
T-statistic: 0.07537438732209384
P-value: 0.939918345521451
```
2. F-test:
```
F-test: T-statistic: 0.005681298264180369
P-value: 0.9399183455217838
```
3. Wilcoxon test:
```
Wilcoxon signed-rank test: T-statistic: 22.0
P-value: 0.5270892568655381
```
4. Mann-Whitney U test:
```
Mann-Whitney U test: T-statistic: 12499999.5
P-value: 0.939919853072483
```

5 Conclusion

Summarizing all important information, a directed social graph was not the best solution to display such a large amount of link sentiments. The use of t-SNE turned out to be the better way showing all sentiments as clusters clearer and more differentiated.

In sentiment prediction with machine learning the decision tree model was the only model that was significantly better than the others because the p-value of the Wilcoxon test was smaller than 0.5 and finally could reject the null hypothesis. Nevertheless, as we were dealing with a large data set the logistic regression model should be the first choice as it performed quite well and was fast enough dealing with such a massive amount of data.

References

Lei, L., & Liu, D. (2021). *Machine learning for natural language processing***.** Cambridge, UK: Cambridge University Press. pp. 1

National Library of Medicine. (2023, February 1). Standard deviation. In *Statistics tutorial* (Section 2, Module 8). National Institutes of Health. Retrieved from https://www.nlm.nih.gov/nichsr/stats_tutorial/section2/mod8_sd.html.

Roweis*,* Sam; Hinton, Geoffrey (January 2002). Stochastic neighbor embedding (PDF)*.* Neural Information Processing Systems*.* p. 1

van der Maaten, L.J.P.; Hinton, G.E. (Nov 2008). "Visualizing Data Using t-SNE" *(PDF)*. Journal of Machine Learning Research. **9**: 2579–2605.

Brink, H., Richards, J. W., & Fetherolf, M. (2017). *Introduction to machine learning with Python*. Shelter Island, NY: Manning Publications Co. from https://learning.oreilly.com/library/view/real-world-machinelearning/9781617291920/kindle_split_022.html

Jung, A. (2022). Machine learning : The basics. (1st ed.). Singapore: Springer Singapore. pp. 205

Russell, S. J., & Norvig, P. (2012). *Artificial intelligence: a modern approach* (3rd ed.). Upper Saddle River, NJ: Pearson. pp. 934

Singh, P. K., Veselov, G., Vyatkin, V., Pljonkin, A., Dodero, J. M., & Kumar, Y. (2021). Software fault prediction using machine learning models and comparative analysis. In *Futuristic Trends in Network and Communication Technologies* (Vol. 1395, pp. 30-45). Springer International Publishing. p.34

Saleh, H. (2018). Machine learning fundamentals. Packt Publishing. from https://katalog.bibliothek.uni-wuerzburg.de/TouchPoint/singleHit.do?methodToCall=showHit&curPos=2&identifier=2_SOLR_SERVER_898498313

Papp, S., Weidinger, W., Meir-Huber, M., Ortner, B., Langs, G., & Wazir, R. (2019). Handbuch Data Science. Mit Datenanalyse und Machine Learning Wert aus Daten generieren. Hanser. p .9

Kaptein, M., & van den Heuvel, E. (2022). Statistics for Data Scientists: An Introduction to Probability, Statistics, and Data Analysis (1st ed.). Cham: Springer International Publishing. p. xxi

Appendix

Appendix A: Speed stats

(metrics were used on the training data here but
will not be relevant for this paper)

```
Logistic Regression: 2 min. 5s (Grid: ca. 36 min.)
accuracy: 0.9557042498632627
precision: 0.9556597474447306
recall: 0.9985203584872391
f1 score: 0.9766200266942607

Naives Bayes: 40s  (Grid: 26s)
accuracy: 0.9071962835894049
precision: 0.9380663750503859
recall: 0.9634459547521026
f1 score: 0.9505867936445502

Decision Tree Classifier: 2h 8 min. (33 min.; Grid: split10k; best: max_depth=5 max_leaf_nodes=10, min_samples_leaf=1 score=0.9218)
accuracy: 0.9967751330080054
precision: 0.9986993753932567
recall: 0.9978188704125364
f1 score: 0.9982589287426321

Random Forest: 5h 46 min.  (Grid: 1h; split10k; best: n_esti=25 score=0.9224)
accuracy: 0.9967112039266662
precision: 0.9970515124861371
recall: 0.999405843433995
f1 score: 0.9982272897826038
```

Appendix B: Parameters and values for all models

Logistic regression

```
Logistic Regression results

threshold value: 0.5
precision: 0.9233846769353871
recall: 0.9233846769353871

threshold value: 0.6
precision: 0.9223844768953791
recall: 0.9223844768953791

threshold value: 0.7
precision: 0.9215843168633727
recall: 0.9215843168633727

threshold value: 0.4
precision: 0.9233846769353871
recall: 0.9233846769353871

threshold value: 0.56
precision: 0.9223844768953791
recall: 0.9223844768953791

threshold value: 0.1
precision: 0.9233846769353871
recall: 0.9233846769353871
```

```
kfold value: 3
kfold metrics:
accuracy: 0.9185813497564593
precision: 0.9311470814236641
recall: 0.9845952671645257
f1 score: 0.9571230879183054

kfold value: 5
kfold metrics:
accuracy: 0.9199853853853854
precision: 0.9320053551647097
recall: 0.9853261424846697
f1 score: 0.9578483782790578

kfold value: 10
kfold metrics:
accuracy: 0.9223863727454911
precision: 0.9332352651592846
recall: 0.9866170868319782
f1 score: 0.9591311778709507

kfold value: 20
kfold metrics:
accuracy: 0.921387951807229
precision: 0.9330377272586727
recall: 0.9857222597175577
f1 score: 0.958541690525003

kfold value: 15
kfold metrics:
accuracy: 0.9215910521299743
precision: 0.9326461269308051
recall: 0.9863594596237953
f1 score: 0.9586871740882537

kfold value: 8
kfold metrics:
accuracy: 0.9201836538461539
precision: 0.9321529566883097
recall: 0.9852615252575312
f1 score: 0.9579532536165284

kfold value: 11
kfold metrics:
accuracy: 0.920788462638683
precision: 0.9330682325745805
recall: 0.9847979113724624
f1 score: 0.9581609894969229

kfold value: 9
kfold metrics:
accuracy: 0.9213803731789343
precision: 0.9328124434276968
recall: 0.9859384538903283
f1 score: 0.95859681877363
```

Naïve Bayes

```
threshold value: 0.5
precision: 0.8795759151830366
recall: 0.8795759151830366

threshold value: 0.6
precision: 0.8787757551510302
recall: 0.8787757551510302

threshold value: 0.7
precision: 0.8775755151030206
recall: 0.8775755151030206

threshold value: 0.4
precision: 0.8807761552310462
recall: 0.8807761552310462

threshold value: 0.1
precision: 0.8821764352870574
recall: 0.8821764352870574

threshold value: 0.01
precision: 0.8895779155831166
recall: 0.8895779155831166

threshold value: 1e-08
precision: 0.9109821964392879
recall: 0.9109821964392879

threshold value: 1e-20
precision: 0.9207841568313663
recall: 0.9207841568313663

threshold value: 0.0 (input: 1e-1000)
precision: 0.9235847169433887
recall: 0.9235847169433887
```

```
kfold value: 5
kfold metrics:
accuracy: 0.9177829829829831
precision: 0.9320139670858838
recall: 0.9826754498006427
f1 score: 0.9566656505021334

kfold value: 3
kfold metrics:
accuracy: 0.9185818298525169
precision: 0.9320541435120661
recall: 0.9835398604949793
f1 score: 0.9570854394323732

kfold value: 10
kfold metrics:
accuracy: 0.9207831663326653
precision: 0.9330815073389125
recall: 0.9848322860440714
f1 score: 0.9582250535369836

kfold value: 20
kfold metrics:
accuracy: 0.921390361445783
precision: 0.9326173900167797
recall: 0.9861072811793781
f1 score: 0.958512842963918

kfold value: 15
kfold metrics:
accuracy: 0.9229858601116084
precision: 0.9323253263889016
recall: 0.9882840090228877
f1 score: 0.9594174166930027

kfold value: 8
kfold metrics:
accuracy: 0.9189794871794872
precision: 0.9320847571765957
recall: 0.9839832401075492
f1 score: 0.9572842381592295

kfold value: 11
kfold metrics:
accuracy: 0.9187825390468561
precision: 0.9327939024205076
recall: 0.9828830539914167
f1 score: 0.9571216034064587

kfold value: 9
kfold metrics:
accuracy: 0.9179834510769762
precision: 0.9318113786938215
recall: 0.9830695716538221
f1 score: 0.9567472446940548

kfold value: 10
kfold metrics:
accuracy: 0.9179831663326652
precision: 0.9322125060743437
recall: 0.9826794729145479
f1 score: 0.9567310835500498
```

Decision Tree

```
threshold value: 0.5
precision: 0.9219843968793758
recall: 0.9219843968793758

threshold value: 0.6
precision: 0.9219843968793758
recall: 0.9219843968793758

threshold value: 0.7
precision: 0.9215843168633727
recall: 0.9215843168633727

threshold value: 0.4
precision: 0.9233846769353871
recall: 0.9233846769353871

threshold value: 0.1
precision: 0.9233846769353871
recall: 0.9233846769353871

threshold value: 0.01
precision: 0.9235847169433887
recall: 0.9235847169433887

threshold value: 1e-08
precision: 0.9235847169433887
recall: 0.9235847169433887

threshold value: 1e-20
precision: 0.9235847169433887
recall: 0.9235847169433887

threshold value: 0.0 (1e-1000)
precision: 0.9235847169433887
recall: 0.9235847169433887
```

```
kfold value: 5
kfold metrics:
accuracy: 0.9217825825825827
precision: 0.9322781241380381
recall: 0.9869974628204566
f1 score: 0.9588520013204785

kfold value: 3
kfold metrics:
accuracy: 0.9191850705489154
precision: 0.9312375122309425
recall: 0.9852907488296473
f1 score: 0.9574745548040879

kfold value: 10
kfold metrics:
accuracy: 0.9203839679358718
precision: 0.9328871295806465
recall: 0.9846364893306806
f1 score: 0.9579992655840825

kfold value: 20
kfold metrics:
accuracy: 0.9213863453815263
precision: 0.9327648585276382
recall: 0.9858930445909984
f1 score: 0.9585241289063138

kfold value: 15
kfold metrics:
accuracy: 0.920382658107209
precision: 0.9322182076988224
recall: 0.9855097086229051
f1 score: 0.9580538742935751

kfold value: 8
kfold metrics:
accuracy: 0.9215858974358975
precision: 0.9325992955503228
recall: 0.9863318572492853
f1 score: 0.95870409889405

kfold value: 11
kfold metrics:
accuracy: 0.9223802629088973
precision: 0.9330365340116629
recall: 0.9867995563675624
f1 score: 0.9591015853365644

kfold value: 9
kfold metrics:
accuracy: 0.9213882947695898
precision: 0.9329460402779579
recall: 0.9856964909014196
f1 score: 0.9585262915810415

kfold value: 10
kfold metrics:
accuracy: 0.9213831663326653
precision: 0.9327622232378217
recall: 0.9859061862315649
f1 score: 0.9585767278047369
```

Random Forest

```
threshold value: 0.5
precision: 0.9233846769353871
recall: 0.9233846769353871

threshold value: 0.6
precision: 0.9223844768953791
recall: 0.9223844768953791

threshold value: 0.7
precision: 0.9219843968793758
recall: 0.9219843968793758

threshold value: 0.4
precision: 0.9233846769353871
recall: 0.9233846769353871

threshold value: 0.1
precision: 0.9233846769353871
recall: 0.9233846769353871

threshold value: 0.01
precision: 0.9235847169433887
recall: 0.9235847169433887

threshold value: 1e-08
precision: 0.9235847169433887
recall: 0.9235847169433887

threshold value: 1e-20
precision: 0.9235847169433887
recall: 0.9235847169433887

threshold value: 0.0
precision: 0.9235847169433887
recall: 0.9235847169433887
```

```
kfold value: 5
kfold metrics:
accuracy: 0.9195863863863863
precision: 0.9315919307667133
recall: 0.9852619587467887
f1 score: 0.9576680098644512

kfold value: 3
kfold metrics:
accuracy: 0.9209836304047713
precision: 0.9325802466884213
recall: 0.9857048367238371
f1 score: 0.9584046756644606

kfold value: 10
kfold metrics:
accuracy: 0.9205843687374751
precision: 0.9320210264216285
recall: 0.9859314612204255
f1 score: 0.9581836784638827

kfold value: 20
kfold metrics:
accuracy: 0.9211839357429717
precision: 0.9325815728637373
recall: 0.9858907033883636
f1 score: 0.9584465550821655

kfold value: 15
kfold metrics:
accuracy: 0.920382658107209
precision: 0.9327596831943051
recall: 0.9848560904322895
f1 score: 0.9580566966208776

kfold value: 8
kfold metrics:
accuracy: 0.9203823717948717
precision: 0.9326932071806748
recall: 0.9848047640683073
f1 score: 0.9580180152868095

kfold value: 11
kfold metrics:
accuracy: 0.9201833408881868
precision: 0.9328791095213398
recall: 0.9843969085924303
f1 score: 0.9579001023415139

kfold value: 9
kfold metrics:
accuracy: 0.9209842936461643
precision: 0.9324074224561945
recall: 0.9859024419507537
f1 score: 0.9583776714044316

kfold value: 10
kfold metrics:
accuracy: 0.9201843687374749
precision: 0.932900114944586
recall: 0.9844231851420174
f1 score: 0.9579324956573017
```