Sentiment Analysis of Tweets using multiple Algorithms

Project Report for "Data Mining"

Olivia Saukonoja, David Riemer

December 2023

Abstract

In this Project Report we will explain in detail the process of our Python Project, in which we use multiple algorithms discussed in class to find relationships between words used in tweets. The Dataset provided by the kaggle competition "Twitter sentiment analysis" includes 100000 tweets as Training Data, with their given Sentiment. 1 as a representation for a positive tweet and 0 for a negative one. The project utilizes various algorithms, including k-means clustering, Hashing, Dimension Reduction by Singular Vector Decomposition, and A-priori. By applying these algorithms, we aim to gain insights into patterns and connections within the datasets' content. This analysis will contribute to a better understanding of the relationships between words and their context in the realm of social media data. Throughout this report, we will delve into the methodology, results, and implications of the algorithms, highlighting the steps necessary for a successful interpretation of the results and also which obstacles had to be overcome on the way but also what room for improvement is available. In the end we will build a Logistic Regression-model in order to classify other tweets to their respective Sentiment value and give a conclusion about our Project.

Contents

1 Introduction

The importance of Natural Language Processing (NLP) has grown significantly due to the overabundance of data on the internet. With the vast amount of text-based information available online, it has become increasingly challenging for humans to manually process and extract meaningful insights from this data. NLP techniques, such as text classification, sentiment analysis, and named entity recognition, enable us to automate the analysis and understanding of large volumes of text data. One of the motivations of our project is the fact, that tweets provide a short form of information singlehandedly, however in bulk they reveal hidden trends and connections between various users. In the first section of this report we will explain the methods used in our data mining process, which include Word2Vec, Dimension Reduction, k-means Clustering, A-priori as well as Logistic Regression, in order to build a solid fundation of knowledge, for the purpose of comprehending the following chapters better. This is also the section where we describe the given data. After that, we will present the results from our algorithms to provide more detailed information about the outcome of our project.

2 Methods

2.1 Architecture and Environment

For our project enviroment we opted to use Visual Studio Code 1.84 on Windows 11. In order to be able to work better with our large dataset we decided to include Pyspark, which ran locally on my RazerBook 13. On top of that for some of the visualization we also used Knime with its respective JavaScript extension.

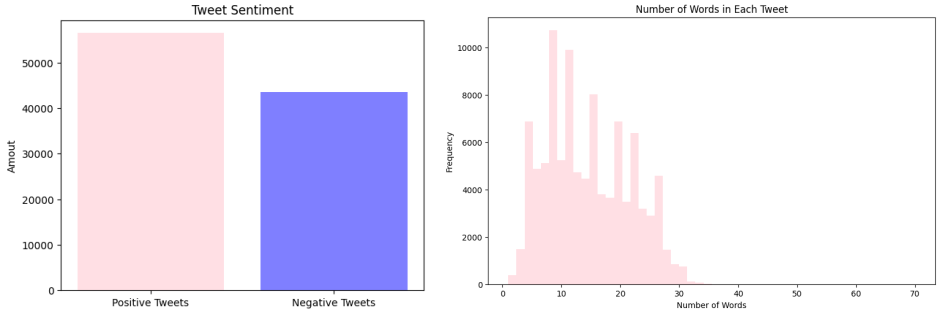
2.2 The data

The dataset utilized for this project can be accessed via. It consists of a CSV file with the following columns:

- ItemID: This column serves as the primary key, assigning a unique number to each tweet.
- Sentiment: This column indicates the sentiment of each tweet. A value of 0 denotes a negative sentiment, while a value of 1 signifies a positive sentiment.
- SentimentText: The content of the tweets is contained within this column.

The dataset consists of a total of 100,000 tweets. After analyzing the data, we aimed to describe some basic properties of the dataset. Our research concluded that there are 56,462 positive tweets and 43,538 negative tweets (figure 1). The tweet with the highest word count consisted of 70 words, while the shortest tweet contained only 1 word. On average, users tend to use 15 words per tweet. This average applies to 4,167 tweets. Additionally, the most frequently used word (after applying our pipeline) is "good", and 72,796 people tweeted 20 words or less.

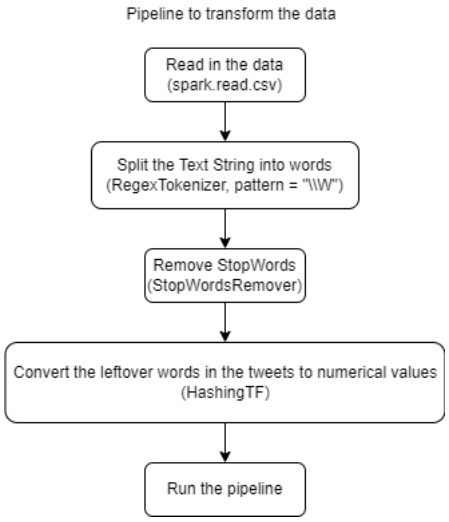
Figure 1: Basic Visulization of the dataset



2.3 Preprocessing of the data

The first step of our the pipeline is tokenization, which is the process of splitting the text into individual words or tokens. This is done using the "RegexTokenizer". The "inputCol" parameter is set to "SentimentText", which is the column in the DataFrame that contains the text to be tokenized.. The "pattern" parameter of the Tokenizer is set to "
W", which means any non-word character is considered a delimiter. The second step in the pipeline is "stop words removal". Stop words are common words that do not carry much meaningful information and are often removed in text mining tasks. The "StopWordsRemover" class is used for this purpose. In order to eliminate all necessary stop words we imported the english stopwords corpus fromm nltk.corpus and added some custom stop words, like websites or hyperlinks. The third step in the pipeline is feature extraction. The "HashingTF" class converts all the words leftover after removing the stop words into numerical feature vectors. To run these steps sequentially and have them contained in one object, we created a variable called pipeline. It includes all the steps mentioned above. This also makes it easier to rerun the pipeline if necessary. As mentioned in the introduction, our goal is to gain an overview of the extensive dataset provided.We want to create a Logistic Regression Model that can predict the sentiment of a tweet with at least 70% accuracy. Additionally, we would like to understand the relationships between words used in tweets, such as identifying

the words most likely used in the context of a tweet that also contains the word "happy."



2.4 Clustering the data with the help of Word2Vec

In order to get to a point of clustring the words in a way that makes sense, we decided to train a Word2Vec model. Word 2Vec is a standard method of generating word embeddings: This is a process of each word being mapped to an vector in a high dimensional space. By mapping words to continuous vector representations, word embeddings enable NLP models to understand the similarity, analogy, and relatedness between different words. we decided on our vector to be of the dimension 2000.

```
w2vModel = Word2Vec(sentences=sentences, vector_size=2000,
    workers=1, seed=1337)
w2vModel.save("word2vec.model")
```

After creating the model, we then we then used it to find out, which words are most often used with the most common emotions exressed in tweets:

```
listHappy=w2vModel.wv.most_similar("happy")
listsad=w2vModel.wv.most_similar("sad")
listgood=w2vModel.wv.most_similar("good")
listbad=w2vModel.wv.most_similar("bad")
```

Our goal is to have words with similar context occupy close spatial positions when we end up plotting our data.

	Happy	Sad	Good	Bad
0	birthday	amelia_grace	nice	sick
1	catirah	bad	lovely	horrible
2	mother	face	great	pain
3	camper	jealous	blondeyy	sucks
4	mothers	shit	wonderful	crap
5	belated	aswel	ashesborn	right
6	chocolatesuze	lucky	rough	hate
7	blanquis26	hate	perfect	uggh
8	bday	cry	exciting	worse
9	wonderful	sucks	bexmith	cold

2.4.1 Dimensionreduction suing Singular Vector Decomposition (SVD)

After generating the 2000-dimensional vector representation for each word, the next step is to apply Singular Value Decomposition (SVD) to these vectors. This will allow us to plot the data points (words) on a 2-dimensional coordinate system in the subsequent step. This is acomplished by importing svd from scipy.linalg.

```
from scipy.linalg import svd
from numpy import zeros, diag
```

We also just want to look at the 750 "most important" words in our dataset so we end up with a 750×2000 matrix, in which each row is a representation of one word and the columns represent one of the features of the word. After creating our $U, \sigma,$ *and* V^T matrices and reducing the dimensions to only 2 by selecting only the two largest singular values and their corresponding singular vectors from these 3 matrices and multiplying them together.

```
U, s, VT = svd(vectors750, full_matrices=False)
Sigma = zeros((vectors750.shape[0], vectors750.shape[1]))
Sigma[:vectors750.shape[0], :vectors750.shape[0]] = diag(s)
n_elements = 2
Sigma = Sigma[:, :n_elements]
VT = VT[:n_elements, :]
#reconstruct
B = U.dot(Sigma.dot(VT))
```

2.4.2 Using k-means to cluster the data and visualize

Now we can plot the data in a 2-dimensional space in which words most often used together are close to one another in the euclidean space. We use the k-means package provided by sklearn.cluster and devide the points in 20 cluster and also end up plotting the respective centroid of each cluster to visualize the data better. In the result one can see that for example, words that describe days in a week points in time belong to a cluster, but also

words that are apart of positive emotions. There is a lot of noise towards the middle of the figure, which is due to the fact that a lot of words can be used in a variety of contexts.

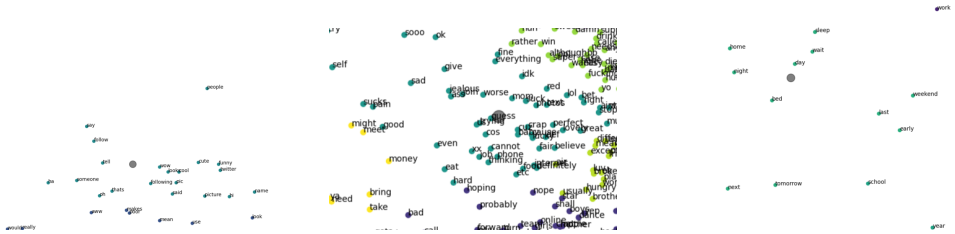


Figure 2: On the left: positive emotions(wow, cool, cute, funny)
Middle: negative emotions(jealous, sad, worse, cannot)
Right: Timestamps with used in the context of sleep (bed, night, slep, tomorrow)

2.5 A-priori to find words most often used together

Next, we used to use a simple implementation of the apriori-algorithmin order to also find words often used together. This time only the number of occurences together in a tweet is examined. We also divided our pre-processed dataset into two separate dataframes. One dataframe includes only positive tweets, while the other includes only negative tweets. The top 5 most frequently used words in positive tweets were "good," "thanks," "love," "lol," and "like." In negative tweets, the most common words were "get," "like," "know," "sorry," and "go." The most frequently occurring tuple in positive tweets was [morning, good], and in negative tweets, it was [could, wish].

2.6 Logistic Regression Model

In the final step of our project, we developed a Logistic Regression Model to classify tweets based on their sentiment. To begin, we randomly divided the data into 80% training data and 20% test data. When creating the Logistic Regression Model, we set a maximum of 1000 iterations to ensure convergence even if the gradient descent steps become "stuck". Additionally, we included a regularization parameter to prevent overfitting on the training data. As for the features used to train the model, we utilized the features from the hashingTF in our initial pipeline.

```
Split training and Testing
split_data=train.randomSplit([0.8,0.2])
trainSplit=split_data[0]
testSplit=split_data[1]
trainSplit = pipeline.transform(trainSplit).select("Sentiment",
    "filtered", "features")
testSplit = pipeline.transform(testSplit).select("Sentiment",
    "filtered", "features")
lr = LogisticRegression(labelCol = 'Sentiment',
    featuresCol='features', maxIter=1000, regParam=0.01)
lrModel = lr.fit(trainSplit)
print("Done")
```

3 Results and Conclusion

4 References