1em0pt

Exploratory Wrangling and Annotation of Tweets

Berkay Dinçer

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of the requirements for the degree of
Master of Science

Sabancı University

June, 2015

Exploratory Wrangling and Annotation of Tweets

Approved by:

Assoc. Prof. Dr. Yücel Saygın .....
(Dissertation Supervisor)

Assoc. Prof. Dr. Hüsnü Yenigün .....

Assoc. Prof. Dr. Ali Inan .....

Date of Approval: .....

Exploratory Wrangling and Annotation of Tweets

Berkay Dinçer

Computer Science and Engineering, Master's Thesis, 2015

Thesis Supervisor: Yücel Saygın

Abstract

Twitter is an ever growing social platform that is full of ideas and opinions. Huge amount of data is produced daily that is usually too cumbersome to process and mine for the opinions of individuals. As of 2010, 55 million tweets are sent daily and the number is doubled by now. Also twitter data is not structured as a text based information source, considering the lack of structure of the data along with its huge volume, it is nearly impossible to have a healthy summarization of all the ideas and opinions at real time. Therefore in this work we propose a set of algorithms to cluster relevant tweets and similar tweets talking about the same concept on twitter domain.

We demonstrate and explain how this information can be used on tweets. As a side benefit we also use these algorithms to detect bots or spammer accounts on twitter since we place such tweets to the same clusters. We show that by transforming twitter data into a clustered structure we are able to overcome problems such as detecting bots and providing a neat summary of the data. these are solvable by transforming the unstructured data environment of twitter to a structured data environment by forming clusters and buckets over the data feed. Another interesting observation we made is that the clusters we form follow the Pareto principle therefore by inspecting only 20% of the clusters we can cover 80% of the whole data.

to my beloved family...

Acknowledgements

I would like to show my gratitude to those people who supported and assisted me to pass each obstacle I faced throughout my academic career and in my life. To begin with I would like to thank to my thesis advisor, Assoc. Prof. Dr. Yücel Saygın for his endless support and for sharing his wisdom and guidance on every tough step I had to take towards my graduation. Also I would like to thank to my former instructors who shared their knowledge and experience on every course I took. It goes without saying that every instructor of Computer Science Department contributed a lot to me during my graduate and undergraduate years.

Many thanks to the members of thesis defense committee Assoc. Prof. Dr. Hüsnü Yenigün and Assoc. Prof. Dr. Ali Inan for their presence and valuable feedbacks on this thesis.

I also would like to thank to all of my friends for all the good laughs and moments and for their endless support throughout these years. Special thanks to Inanç Arın for all the insights on the hard algorithms we have to come up for this thesis but most of all for his precious friendship. Also I would like to thank to my childhood friends that are no different than brothers for me who are Berkay Eren Gülcan, Berke Gözneli, Sarp Çolakoğlu and Tan Özdoğan for everything. I truly feel blessed and lucky to have friends like you.

Last but not least, I want to express my infinite gratitude to my beloved family who encouraged me on my every step. This thesis and many other things would not have been possible without them.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Social media has been an active research topic with the rise of platforms such as Facebook, Twitter and Foursquare. As the volume of the data increases it is harder to analyze but since the data contains a lot of valuable information analysis is also very rewarding. In addition to that since the data is quite large and growing rapidly the need for algorithms/tools that give an insight about the content of the data is needed. According to the twitter conference [21] twitter has 106 million users as of 2010 and it is growing by 300.000 users per day. It attracts 180 million unique users everyday and 55 million tweets are produced daily. Twitter is also the data source for many research areas such as sentiment analysis, data mining, machine learning and network analysis topics. There are a lot of papers and projects experimenting by using twitter data. Most of the work is focused on sentiment analysis such as [2] even though there is interesting work such as [1] that is conducting network analysis to predict Flu epidemics. Twitter offers a real time view of opinions on different subjects including daily life, politics, trends and news. For researchers working on data analysis / data mining subjects twitter contains a huge amount of data, mainly consisting of opinions, hence making the platform incredibly valuable.

1.1 Motivation

The motivation behind this work is about providing an overview of a tweet set in terms of context and opinions to the end user. Having huge amount of data on various subjects from daily life to politics is a great value, however the size of the data can also be quite confusing as it is harder to make sense and distinguish between the topics and ideas that people are

debating/tweeting about. In this work we wanted to cluster the twitter data by proposing a set of clustering algorithms which we can use to group the relevant topics and ideas into clusters. Even though there are some clustering algorithms designed for clustering user networks [16], we focused on clustering text data adapted and tuned specifically for twitter. This allows us to see the data in a more structured way compared to the native, unstructured environment of twitter.

There are a lot of benefits we obtain by creating clusters based on topics and ideas. For example, labeling of data is a very cumbersome job to carry out. With the recent discoveries in social media along with machine learning, need for the labeled data to train models is quite high. In addition to that there are some problems that are still troublesome to solve. For example, conducting sentiment analysis to determine whether a given tweet is pro-government or not is currently a very hard problem to solve if not impossible. There are some work that focus on the problem of sentiment analysis based on perspective such as [20]. We believe the approach mentioned in [20] is only suitable for black and white issues such as the well-known Israel-Palestine conflict where there are only 2 sides of the conflict with certain different interests conflicting each other. However for socially and politically more complicated conflicts where there are more than two sides involved this still remains as a hard problem to solve. Such problems are quite common in today's world and we frequently face with such problems that require automation of very complex tasks. We believe, in order to carry out these tasks a fully automated system is not possible with today's technology, but the best we can do is to aid the people carrying out this task as much as possible. One example is Amazon's Mechanical Turk, a platform offered by amazon to solve this problem with the idea of crowd sourcing [15]. A platform such as this offers quality data with relatively inexpensive fees. There are people working for this platform to achieve tasks that are provided by some people such as labeling data, or marking spots on maps etc.. These jobs can be expressed as jobs that require human decision making. These tasks require a lot of time and human attention and can be improved in terms of speed.

1.2 Contribution

Contribution of this thesis is a set of clustering algorithms which can be used to aid with human labeling. We can also employ semi automated approaches such as active learning where the user helps the classifier during the labeling and vice versa. With the help of the algorithms which we will further explain in this work, we are able to reduce the labeling time up to 80% while adding none or minimal errors to the data set. In this work the user does not label individual tweets, but rather labels whole clusters or buckets. Our algorithm positions tweets such that inside a given cluster or a bucket tweets share the same label and therefore clusters formed by our algorithm can be used to speed up the labeling process.

Interestingly another observation we made by investigating the results produced by the algorithm is the observation of Pareto Principle. Regardless of the data set we observe the top 20% of the clusters/buckets contain the 80% of all tweets. Pareto principle is used in many areas such as business management, medical science and software engineering. For example [18] states that by fixing the top 20% of the most reported bugs one can fix 80% of related errors and crashes. With the help of the algorithms we propose here we were able to reduce the data size to 20% of the original data size. Which means we can give a good summary of the whole data by just looking at the 20% by assigning the relevant tweets to same clusters. An observer looking at the data with the purpose of skimming would be spending effort on the redundant part of the data where the real value is the non-redundant part. The algorithm we describe here also allows the user to see the non-redundant parts only.

Another benefit is a domain specific benefit that is applicable to twitter domain. By forming clusters using our clustering algorithms we are able to see the impact of a given tweet. Most of the users in twitter re-tweet a certain tweet by adding some comments to the original tweet. Our algorithm performs almost perfectly for such situations, allowing us to see the cluster size for a hot topic and allowing us to extract only the comments that are done on this subject rather than the re-tweet itself. One other thing is that we were also able to cluster the accounts that are "'spammers"' or "'bots"'. These accounts aim to increase their follower numbers by using popular hash tags even though they do not tweet about the content.

The experiments were done by using 3 different data sets with 2 different languages. Here we also demonstrate that our algorithm is domain independent and language independent. We

gathered data on Charlie Hebdo events in France that consist of English tweets, tweets that are related with Kobani events in Turkey-Syria and tweets that are tagged with the hash tag #sendeAnlat, following the murder of Ozgecan Aslan which attracted a lot of public attention in conventional and social media. Then from these data sets we formed clusters in terms of similarity and assigned each tweet to its respective cluster along with the other tweets that are similar to it. We evaluate the performance of our algorithm with the help of a similarity measure where we are going to explain in the upcoming chapters of this thesis. Also we test the performance of the algorithm by comparing the clustering results with human labeled data.

2 Preliminaries and Background

In this section, we formally introduce some notations and required background knowledge which we use to solve the problem of clustering similar tweets based on their content. Firstly, we introduce *Longest Common Sub Sequence algorithm* which is a similarity metric between two strings, we specifically tailored this algorithm to fit with our needs in Twitter domain as we are going to be using the output of this algorithm as a basis for our clustering algorithm. Implementation of *LCS* algorithm will also be provided. Then, *Active Learning* and *Naive Bayes Classification* techniques will be mentioned to further explain the relationship between our clustering algorithm and Active Learning. Lastly we will be giving details about *Play framework* as we used this framework to convert our algorithm into a web application for end users.

2.1 LCS algorithm

Longest Common Sub sequence problem is a well known problem in Computer Science. It has various uses from spelling error correction to DNA or protein sequence comparisons [4]. Longest Common Sub sequence, by definition is the number of identical symbols in two strings that is also ordered. The formal definition of the LCS algorithm can be explained as follows. We are comparing two strings which are denoted by s_1 and s_2 a sub sequence s is formed by deleting $s_1.length - s.length$ characters from s. However a common sub sequence between two strings s_1 and s_2 is the common maximal length sub sequence that exist in both strings. As stated in [4] LCS problem is a special case of edit distance problem. The distance between two strings s_1 and s_2 is the number of minimum operations needed to transform s_1 to s_2 or s_2 to s_1. Therefore they are closely related in terms of time complexity.

There were many studies in order to optimize this algorithm in terms of time and space complexity. The algorithm we use in this work achieves $O(n * m)$ time complexity where n is the length of the first string and m is the length of the second string. Although there are better implementations of this algorithm in terms of time complexity by omitting some special cases as explained in [4] we decided to use a generic version of the algorithm that is not tuned for any special circumstances.

Pseudo code for LCS problem can be found below:

Algorithm 1 Longest Common Sub sequence Algorithm (*char X[1..m], char Y[1..n]*)

```
1:  declare array C[0..m, 0..n]
2:  for i = 1 → m do
3:      C[i, 0] ← 0
4:  end for
5:  for j = 1 → n do
6:      C[0, j] ← 0
7:  end for
8:  for i = 1 → m do
9:      for j = 1 → n do
10:         if X[i] = Y[j] then
11:             C[i, j] ← C[i − 1, j − 1] + 1
12:         else
13:             C[i, j] ← maximum(C[i, j − 1], C[i − 1, j])
14:         end if
15:     end for
16: end for
17: return C[m, n]
```

An example for LCS algorithm:

		M	**Z**	**J**	**A**	**W**	**X**	**U**
	0	0	0	0	0	0	0	0
X	0	0	0	0	0	0	1	1
M	0	1	1	1	1	1	1	1
J	0	1	1	2	2	2	2	2
Y	0	1	1	2	2	2	2	2
A	0	1	1	2	3	3	3	3
U	0	1	1	2	3	3	3	4
Z	0	1	2	2	3	3	3	4

Table 1: An example to find the longest common sub sequence of two words

The example in table 1 finds the length and the longest common sub sequence itself. The underlined elements form the longest common sub sequence while the maximum of all the numbers in the table gives the length of the longest common sub sequence. Here in this example first string is: WZJAWXU and the second string is: XMJYAUZ. This example is solved by dynamic programming and is identical with the algorithm we use in this thesis. Algorithm starts iterating by comparing the first characters of both strings which are M and X. Since we do not find a match, we move on to compare second character of the second string and first character of the first string which corresponds to M and M. Here we have a match and we raise the length of the longest common sub sequence by 1. The underline in the table states that this is the point of change for the length of the longest common sub sequence. We keep iterating over the strings until we fill out the table, one can follow the state of the algorithm and the indices where the length of the resulting string change by looking at the underlined numbers in the table 1.

After we get the longest common sub sequence between two strings we use this outcome to determine a similarity score. This score is indeed the score we use to measure the similarity of two tweets. Since this thesis is focused on finding similar tweets and since tweets have a limited length of 140 characters as of 2015, we show that this method works for the given domain quite well. We will be elaborating the way we calculate this score in chapter 3.1.

2.2 Active Learning

In machine learning, sentiment analysis existing methods need a large amount of training data for supervised learning methods. Creating a corpus with correct labels is usually the bottleneck of creating a model [22]. Nearly all the time data annotation is done by humans [9].

Here we aim to reduce this time by using Active Learning as part of the tool we developed. Active Learning is an aspect of Machine Learning and mostly focus on the sampling techniques of machine learning algorithms. The aim of active learning is to pick which data to label based on some metrics. There exists various sampling techniques to solve this problem. Some sampling techniques are Uncertainty sampling, Query by committee, Expected model change, Expected error reduction and Variance reduction [19]. The sampling technique we

used to train our Naive Bayes classifier is uncertainty sampling. Uncertainty sampling takes one variable into account while deciding on which data to label. That is, the confidence of the classifier while labeling the test data. In order to achieve this we used a probabilistic classifier to learn about the confidence of our classifier given a test data to label. This approach decrease the need for the amount of data needed to train our classifier greatly [13].

2.3 Clustering

Clustering is the problem of grouping similar objects to the same or similar groups and it has various application from Medicine to Business Intelligence. Usually clustering is an unsupervised learning method. Unsupervised learning does not require any training data unlike supervised models. Some clustering algorithms are DBSCAN, k-nn, k-means and etc.. [5]. We used a clustering approach in this work for various reasons. First we work on a variety of data that contains everything from daily life communication to political debates. The training corpus needed in order to train a classifier to "'tag"' the data with correct labels is not feasible considering the variety of possible topics on different domains. In addition to this training corpus is not the only problem, creating a domain specific corpus is already a hard task to carry out, while training a classifier that is an expert in every domain is near impossible. Therefore we decided to use a clustering technique to group the similar data together. As mentioned before, we used *LCS algorithm* as a basis for our clustering method and came up with an algorithm that clusters tweets.

2.4 Play Framework

Play framework is a framework that is based on a Model-View-Controller architecture. Play framework uses Java on the back end and Scala-html on the front end side. We used Play Framework to build an application for the end user so they can upload their data and use our clustering algorithm. Play framework is used through the industry, LinkedIn and Coursera are some websites that use Play framework.

We choose to use Play framework because we already had a specialized Naive Bayes classifier for sentiment analysis written with WEKA. WEKA has a java library that implements a

variety of machine learning and data mining algorithms. Since the classifier and other Natural Language Processing algorithms were written in java we were comfortable with integrating the NLP-Machine learning part with the clustering part with no problems. Clustering algorithm that we will describe in the upcoming chapters runs on the back end. We were able to display the results to the end user with the help of scala templates that allows the use of arguments while rendering a web page and adjusts the page dynamically.

2.5 Elastic Search

We also used Elastic Search as the database running behind our tool. Elastic Search is basically a search server based on Lucene, an open source information retrieval software library. Since twitter data is a fast growing data we wanted to minimize the query costs and wanted to use a reliable distributed storage technique. Elastic Search is being used by many companies/organizations including Quora, CERN, Soundcloud, Github, Mozilla.

Elastic search the items that are stored are called documents. If we make an analogy to the traditional relational databases a document is a row of data in a table. Elastic search helps us in a couple of things. Firstly, it allows a very flexible (hence the name) database environment where a user can store unstructured data since every document can have its own format. Second benefit is that it allows a very fast query infrastructure with the use of shards and replicas. A shard in Elastic Search is a sepaWrate Apache Lucene index and can be placed in a different server/computer in order to handle computationally heavy tasks. When a user wants to run a query on a system that is built with many shards ElastWic Search server sends the query to the relevant shards and shards are tasked to execute the query and send the results to the Elastic Search server. The architecture that we describe here uses the computational power efficiently in order to answer some more complex queries that involve a big sized data. Since we track Tweets using the live stream API of twitter we usually end up writing queries that will return a good amount of data. This distributed architecture allows us to handle queries in a very fast way.

In addition to this the distributed nature of Elastic Search also allows high availability [12]. In elastic search we can use replicas of shards to achieve better availability and throughput. A replica of a shard is basically the exact copy of a shard. In the event of a loss of a shard

we can use the other replicas and keep the system stable and available. We believe Elastic Search is a good solution that meets the needs of this work.

2.6 Naive Bayes Classification Model

Naive Bayes classifier is a probabilistic classifier which is based on Bayes' theorem. The brief logic behind this model is well known Bayesian formula which takes into consideration frequency and combinations of values in historical data. Naive Bayes aims to calculate the means and variances of the parameters by using training data in a supervised learning way to make a classification model. For this process, Maximum Likelihood method is used as one of the well known methods for parameter estimation [3].

In this work we used a Naive Bayes classifier for the sentiment analysis part. There are 2 important reasons behind this decision. In sentiment analysis a Naive Bayes classifier is a widely used classifier due to the nature of the algorithm. Since Naive Bayes assigns equal weights to all features it performs well for the specialized task of predicting Sentiment of a given text or a tweet. The features we extract from a tweet have no superiority over one on other. Therefore usage of Naive Bayes for a sentiment task is a widely used and common concept.

Second reason is because it is a probabilistic classifier. Since we adopted uncertainty sampling for the active learning session of the tool we need a confidence score for each data / tweet we automatically classify. Due to the nature of the Naive Bayes algorithm we are able to get a confidence level based on conditional probability of features. [**?**, **?**]

$$P(c_i|\vec{d_j}) = \frac{P(c_i)P(\vec{d_j}|c_i)}{P(\vec{d_j})}$$

where $P(\vec{d_j})$ is the probability that a randomly chosen document has weights of vector representation as $\vec{d_j}$, $P(c_i)$ means probability that a randomly chosen document belongs to category c_i. To explain $P(\vec{d_j}|c_i)$, we need to make an assumption such that the words in the document are neither dependent on their position nor the length of the document.

$$P(\vec{d_j}|c_i) = \prod_{k=1}^{|T|} P(w_{kj}|c_i)$$

Classification methods contain some main steps for the process. These steps are defined in [**?**] as the figure 1.

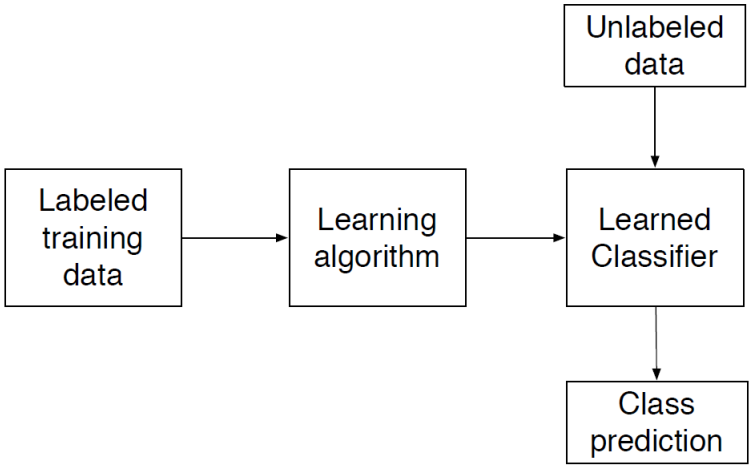


Figure 1: Steps of a classification process

By definition, we need a *labeled training data* for learning algorithm. Then, this algorithm calculates the mean and variance values for each attribute in the data set; this information becomes as a *learned classifier* (it can be also called as *learned model*). Lastly, when a *unlearned data* comes into question for classification, process ends with a *class prediction* according to the *learned classifier* which we prepared before.

The reason we used Naive Bayes Classifier in this work is because we needed a confidence value in order to implement an Active learning method on top of our tool. We used a thresholding method with Maximum Likelihood to boost the performance of our classifier.

2.7 w-Shingling

W-shingling is a technique first proposed by [14] to measure document similarity in an efficient way. w-shingling is also called n-grams as known as contiguous sub sequences of

tokens in a document. W denotes the size of a single shingle generated from the document, in other words w is the size of the sliding window. Here is an example to further clarify the concept of shingling.

Say we have a string where we denote it as $String_1$ = "'We are all born mad. Some remain so."'. The shingle set we create from this string is denoted as S_1 where $w = 10$. Under this conditions S_1 = {We are all,e are all , are all b,are all bo,re all bor,e all born,.....,remain so.}. Different w values will yield different sets. W value is important in order to determine the similarity between two documents, if w is too short than we might be missing the actual content of the document especially if the window size is shorter than the average word size. For bigger w values we might be having a hard time to find a match between two documents as we are comparing two items that are relatively bigger. As the w increases the chances of having two similar items in two sets decrease.

In order to measure the similarity of two documents we first create w-shingles out of these two documents and create two sets, A and B. Resemblance or similarity score can be calculated as:

$$r(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$

This definition of the resemblance score is identical to Jaccard Coefficient. This value will give us a score between 0 and 1 where 1 means the two documents are identical and zero means two documents are completely unrelated.

We use this technique in this thesis as a computationally efficient way of clustering. We offer two algorithms for clustering of tweets where one algorithm is more accurate but also computationally heavy and the other one is computationally more light-weight however less accurate. Creating shingles from N tweets with tweets of average length n takes O(N*n) time.

2.8 Multidimensional Scaling

We used Multidimensional scaling in order to visualize our data based on a distance matrix that we form. Multidimensional scaling is a statistical approach that is used in many areas such as psychology, marketing and mathematics. For example [17] uses multidimensional scaling for workplace behavior research. [6] uses it and makes a comparison with the al-

gorithm he describes in the field of social network analysis. The aim of multidimensional scaling is to find and position the points in a space of n-dimensions with given a distance matrix. The input data to be analyzed are a set of items where there exists distance metric among the items. The distance metric is then used to calculate and form a distance matrix, based on the mathematical properties of the distance measure, this matrix can also be a symmetric matrix.

Formally, we have a set of items $N = n_1, n_2, n_3, ..., n_n$ we calculate a distance matrix with a defined distance metric. Here for the sake of simplicity we define a distance metric between two pairs as $d(n_i, n_j)$. The distance matrix we form from this distance measure can be expressed as,

$$DM_{m,n} = \begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,n} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m,1} & d_{m,2} & \cdots & d_{m,n} \end{pmatrix}$$

From now on we need to come up with N vectors such that $v_1, v_2, v_3, ... \in \mathbb{R}^N$ such that $|d_i - d_j| = d_{i,j} \forall i, j \in N$

Classical MDS uses euclidean distance where as in our case the distance in related with the longest common sub sequence score of two strings [11]. Multidimensional scaling produces such points where the distances between pairs are preserved.

3 Clustering Algorithms For Twitter

Labeling operation when done on a big amount of data is a very time consuming job, especially considering the requirements of training a good classifier. A good amount of training data is usually very cumbersome to label. For modern problems such as sentiment analysis or topic categorization, models being used are trained extensively with more than 10.000 training sets. Some companies such as Amazon assembled teams offshore to solve this problem. Mechanical Turk is such an example, basically they gathered a certain amount of people and requested a certain amount of data to be labeled. In this work we aim to solve this problem. However, Sentiment Analysis turns into a very hard problem when you start labeling the data as positive with respect to an element or negative with respect to an element.

As an example lets say we have a tweet t that belongs to the political domain in a country. The tweet t has a positive sentiment for candidate c_1 here a candidate is a political figure to be elected to an office, but it implies negative sentiment towards candidate c_2. In order to reveal this sentiment one needs to use some special techniques in NLP which is not the context of this work but one also have to label the data accordingly so that the model in training can learn better and eventually will be able to distinguish between positive and negative sentiments toward a certain candidate. We believe this task cannot be fully automated but can be shortened in terms of time with the clustering algorithm we are going to explain, which is also one of the main contributions of this work.

In this section, we are going to give a formal definition of the problem by introducing the methods that we use to support our claim and solve the problem of time spent on the labeling process as well as giving an overview on the data set. However we discovered that the overview we provide can also be used as the solutions of many other problems as explained

in 1.

Given a set of twitter data consisting tweets, we try to place those tweets such that similar tweets or the tweets with similar meanings fall into the same cluster while the tweets that are not similar will fall to different clusters.

3.1 Clustering of Tweets

3.1.1 Clustering with representatives

Say we have a set of tweets $T = \{t_1, t_2, t_3....., t_n\}$.

We would like to create clusters $C = \{c_1, c_2, c_3....., c_n\}$ such that $t_i, t_j \in c_k$ and all elements of c_k are similar or related and $c_0 \cup c_1 \cup c_2..... \cup c_n = T$ and $\forall c_i, c_j \in C, c_i \cap c_j = \emptyset$ We define similar tweets as follows. For a given distance function s between two tweets t_i and t_j if $d(t_i, t_j) < t$ where t is a pre-determined threshold then we conclude that t_1 and t_2 are similar. Initially we have a list of clusters $c_1, c_2....., c_n$ where each cluster contains a single tweet. We start from the first cluster c_1 and for each remaining cluster we check $d(rep(c_i), rep(c_j))$, where the representative of a cluster is shown as $rep(c_i)$. If $d(rep(c_i), (repc_j)) < v$ then we merge c_j into c_1 and remove c_j from the list of clusters.

$$v_{normalized} = \frac{length(r_i)+length(s_j)-2*length(LCS(r_i,s_j))}{length(r_i)+length(s_j)} \tag{1}$$

Where r_i and s_j are the strings we compare and $LCS(r_i, s_j)$ are the longest common sub sequence string between those strings. The pseudo code for the clustering algorithm can be found below.

Algorithm 2 Clustering

```
for each c_i ∈ C do
    for each c_j ∈ C − c_i do
        if d(rep(c_i, c_j)) ≤ v then
            c_i ← c_i ∪ c_j
            C ← C − c_j
        end if
    end for
end for
```

The algorithm 2 has a trade off between time and accuracy. Since we remove a cluster c_j

when its elements are added to another cluster c_i we save $N/2$ comparisons on average for each c_j that is removed. This allows the algorithm to run faster, but with less precision. Since we assume all tweets are individual clusters at the beginning of this algorithm, it is safe to say that each cluster representative is the tweet itself for the initialization phase. However when we start forming clusters as Algorithm 2 executes, we start merging some clusters and adding tweets into existing clusters. The cluster representatives do not change. Therefore we actually indirectly allow some tweets t to be in a cluster c_i where the similarity between $t_i \in c_i$ and $t_j \in c_i$ is less than the given threshold. However for all elements in c_i the similarity between the cluster representative $rep(c_i)$ and an item in the cluster t is always greater than the threshold. By adding a margin for error we save $N/2$ comparisons on average for each t in T.

We also further investigated and formalized the error we introduce to the algorithm by using triangular inequality. We explain the amount of error with the help of visualization of the data. However detailed evaluations for the clustering are provided in section 5. We define a distance metric based on LCS as $d = 1 - similarity$ where d is the distance between two tweets t_1, t_2. Let t_3 be a cluster representative of c_3 and $t_1, t_2 \in c_3$. We know that $d(t_1, t_3), d(t_2, t_3) \leq threshold$ however $d(t_1, t_2)$ is not guaranteed to be less than the threshold. According to the distance we defined, if two strings, t_1, t_3 and t_2, t_3 are above the threshold in terms of similarity then they are at least $1 - threshold$ near to each other. Since we form a line between t_1, t_2 this line can be at most $(2 - 2 * threshold)$ based on the triangular inequality.

Furthermore we did some tweaks to improve the running time of the algorithm. Since calculating the LCS similarity is a computationally expensive task to do, we first conduct a test before we proceed to calculate a LCS score between two pair of tweets. In order for a pair of tweets to have a good LCS score they should be sharing common characters. If a character say "'x"' exists in $tweet_1$ but does not exist in $tweet_2$ we can comfortably say, without investigating further, this pair will have a similarity score less than 1. Therefore we first conduct a character search for both tweets and see if they share common characters. After this point we calculate a score based on the number of characters they share. This score is calculated by,

$$PreLCSScore = \frac{Numberofcommoncharacters}{Lengthofshortesttweet}$$

If PreLCSScore is above 0.9 (Note that this score is always between 0-1). We then proceed to compute the LCS score between the pair. Therefore we avoid an unnecessary O(n*m), where n is the length of the first pair and m is the length of the second pair, calculation by executing an O(n) algorithm where n is the length of the shortest tweet.

Another performance improvement is to remove the exact matches by building a hashmap with a single hashmap over the whole dataset. We simply iterate over the whole dataset and hash each tweet with the key value as itself. If we come across the same exact tweet we place the tweet to the bucket with the other exact matches of that tweet. This, in general takes O(N) and done once at the start of the clustering algorithm.

3.1.2 W-Shingling with Bucketization

Here we propose another algorithm that helps with the clustering of the tweet and as a substitute for the algorithm we proposed in the previous section. This algorithm is relatively light-weight in terms of computational complexity, however it is less accurate. In this algorithm we start creating w-shingles from our tweet set denoted as T. We do a single pass over the tweet set T and end up with a shingle set s_n belonging to each tweet t_n. We then proceed to inspect the shingle set generated by the shingling algorithm over the set T. We take the first shingle set belonging to the first tweet, denoted as t_1 and the set s_1. Then we create a new bucket b_1 with an identifier s_1. Here we define an object called a bucket. Buckets have tweets and a bucket identifier consisting of a set of shingles.

We always keep a bucket list with every new bucket created. After we inspect the first shingle we proceed to compare it with the bucket identifiers in the bucket list. Lets call this list BL. When we find a match between the produced shingle and the bucket's identifier with above a certain threshold according to the resemblance score we mentioned in 2.7 we place the tweet to that bucket. If we fail to find a match after scanning all the bucket objects in BL we proceed to create a new bucket and set the identifier of that bucket to the shingle we produced. While trying to place a tweet to a bucket we perform a linear search over the buckets

in the *BL*. Overall this algorithm works faster than the algorithm we propose in the previous sub section. We create shingles with 1 pass over the whole data set which corresponds to a complexity of O(N*n) where N is the number of tweets in the data set and n is the average length of a tweet. For each tweet we create a shingle set and compare this shingle set with all of the identifiers in the buckets. Hence we perform a linear search over the existing buckets and compare each set with the bucket identifier to calculate Jaccard coefficient between 2 sets, which computationally corresponds to O(M*n) where M is the total number of buckets. Therefore the overall complexity corresponds to O($M * N * n^2$). The main advantage of this method is that almost always M < N. Therefore we end up with a better computational complexity.

3.2 Labeling

The labeling on top of this algorithm works as following. After the execution of first stage and second stage of the algorithm described in 3 and 2 we now have clusters $c \in C$. For each of the clusters c we have a cluster representative of c_i denoted as Cr_i. The user starts labeling by evaluating the cluster representative of the biggest cluster. The user at this point decides and picks one of the labels for the cluster representative depending on the domain (Positive-Objective-Negative for sentiment analysis). When the user assigns a label to the cluster representative Cr_i. We basically assign the label of the cluster representative to all of the elements of the cluster that the user choose to label. Benefits of this approach is obvious. The user labels the whole cluster by only evaluating the corresponding cluster header.

3.2.1 Active Learning

We also developed an active learning approach with a Naive Bayes classifier in this work in order to aid with the labeling of the clusters. We used uncertainty sampling as the sampling mechanism. In this section we will describe the algorithm and the process of active learning integrated into the clustering algorithm that we developed. As we explained in Section-2.6 Naive Bayes classifier is a probabilistic classifier. For each class in the training data the classifier calculates.

$$P(c_i|\vec{d_j}) = \frac{P(c_i)P(\vec{d_j}|c_i)}{P(\vec{d_j})}$$

For every c_i in the training data. $P(c_i|\vec{d_j})$ is called the confidence value. For a given feature vector, the value $P(c_i|\vec{d_j})$ returns the probability of the feature vector belonging to the class c_i. Usually when working on sentiment analysis, we have 3 classes, "'positive"' , "'objective"' and "'negative"'. The classifier we use in this work is a classifier adapted for sentiment analysis in Turkish. Therefore we use some Turkish specific NLP techniques in order to generate a feature vector from a given tweet, nevertheless the application of this technique to any other language is trivial.

The feature vector consists of 19 features. We explain every feature we use in this chapter. The first feature we use is the average polarity value for all words in a tweet. We calculate this value with the basic average formula:

$$\text{averagePolarity} = \text{Pol}(\text{w}_1) + Pol(w_2) + ... + Pol(w_n)\frac{}{n} \tag{2}$$

For each word we get the polarity of that keyword from SentiTurkNet a Turkish polarity lexicon for sentiment analysis. A word in SentiTurkNet consists of 3 polarity values towards 3 sentiments, that are positive, objective and negative. Therefore the $Pol(w_i)$ value of a word is determined as the average of the 3 values provided by SentiTurkNet. Another feature we use in this work called dominantPolarity is calculated again by using the polarity values from SentiTurkNet. However rather than averaging 3 values this time we take the maximum of 3 values and calculate the average of these values for a given tweet. Therefore in order to calculate dominantPolarity we first calculate:

$$\text{domPol} = \max(\text{w}_{positive}, w_{objective}, w_{negative}) \tag{3}$$

Where $w_{positive}, w_{objective}, w_{negative}$ are the polarity values towards the 3 sentiments in SentiTurkNet. After this step we calculate average of these values for each word in a given tweet and come up with a single value for a given tweet. The other features are described in table 2

Our Naive Bayes classifier proceeds to label a given cluster header if asked by the user. In

Feature Name	Explanation
avgDeltaTfIdf	average polarity of Δtf*idf
posEmot	1 if positive emoticon exists, 0 otherwise
negEmot	1 if negative emoticon exists, 0 otherwise
numOfNegEmot	# of negative emoticons
numOfPosEmot	# of positive emoticons
numOfAdj	# of adjectives
numOfNoun	# of nouns
numOfVerb	# of verbs
numOfExcl	# of exclamation marks
numOfQuest	# of question marks
numOfPosSeed	# of positive seed words
numOfNegSeed	# of negative seed words
numOfSwears	# of swears
numOfUppercase	# of words in uppercase (e.g WHAT)
numOfrepeatedChars	# of repeated characters in a word (e.g loooove)
specialTwitterToken	1 if it contains hashtags, usertags or other twitter specific elements
avgScoreNegPMI	average score of negative point wise mutual information values of words
avgScorePosPMI	average score of positive point wise mutual information values of words

Table 2: Feature vector for Naive Bayes classifier

this case we assume the user trusts the classifier and it is confident that the classifier will label the cluster header as precise as the user. We believe this assumption is not far from realistic scenarios since such classifiers exist and is implemented in several systems such as *radian6* also sentiment analysis is not the main contribution of this thesis. The labeling algorithm with active learning approach proceeds as follows:

We pick the biggest cluster c from the cluster set C, which is the result of the function $max(c_1, c_2, c_3....c_i)$. We then proceed to get the cluster representative from the cluster c, $rep(c)$. We generate a feature vector from $rep(c)$ and we calculate all of the features mentioned in this part. After this step we feed the feature vector to our Naive Bayes classifier. Naive bayes classifier returns us a confidence vector with 3 values. $P_{confidence_1}, P_{confidence_2}, P_{confidence_3}$. According to maximum likelihood approach we get $max(P_{confidence_1}, P_{confidence_2}, P_{confidence_3})$ and pick the most probable class given a feature vector. At this point we pick a threshold say, v. If $max(P_{confidence_1}, P_{confidence_2}, P_{confidence_3})$ is not bigger than v we return this cluster header to the user and user manually labels it. If that is not the case classifier labels the cluster

header, and also the whole cluster and proceeds to fetch another cluster from the cluster set S. In case if the maximum confidence value for a given class is below the threshold the manual labeling of the user is then fed into the classifier to adjust the classifier accordingly. After the adjustment, classifier continues labeling by fetching another cluster s from the cluster set S. This approach improves the existing classifier model most by providing a human guidance to adjust the conditional probabilities on the data points where the classifier is most unsure of.

4 Prototype System Design & Implementation

WATT ,Wrangling and Annotation Tool for Twitter, is a modular software that supports extendability of new features as it is being developed without touching the existing code. In this section we will be explaining the functionality and the implementation of the system design and give an insight about the overall architectural design of the application.

As you can see from 2 There are 3 Application layers, Web application layer, logic layer and data access layer.

4.1 Web Application Layer

In this layer we have several module that aid the user on various tasks. WATT currently have 3 modules, filtering clustering and real time. The code for each module is separated and isolated, therefore changes on a given module, such as adding/removing features have zero effect on other modules of the software. On top of the Web application layer we have a request router that directs the HTTP requests coming from the user or from an external application to the relevant module.

This layer uses the Model-View-Controller (MVC) paradigm. MVC pattern consists of models that represent the data. In other words models are classes that help us encapsulate the relevant data within the model. Views are for the visualization of the data, since this is a web application views are HTML pages that are informative to the users. This can be interpreted as a Graphical User Interface (GUI). Controllers are code segments that are responsible from building models and views, code in the controller runs on the server (back end of the application) and output is transferred to a view (front end) before it is displayed to a user. We have build this application with modularity in mind so that scalability and maintenance of the software wouldn't be causing problems while we are developing the new modules. We also use REpresentational State Transfer (REST) architecture which allows other applications to use WATT with the Application Programming Interface (API) we developed through the requests they send over the network. It is possible for an external application to learn the state of the

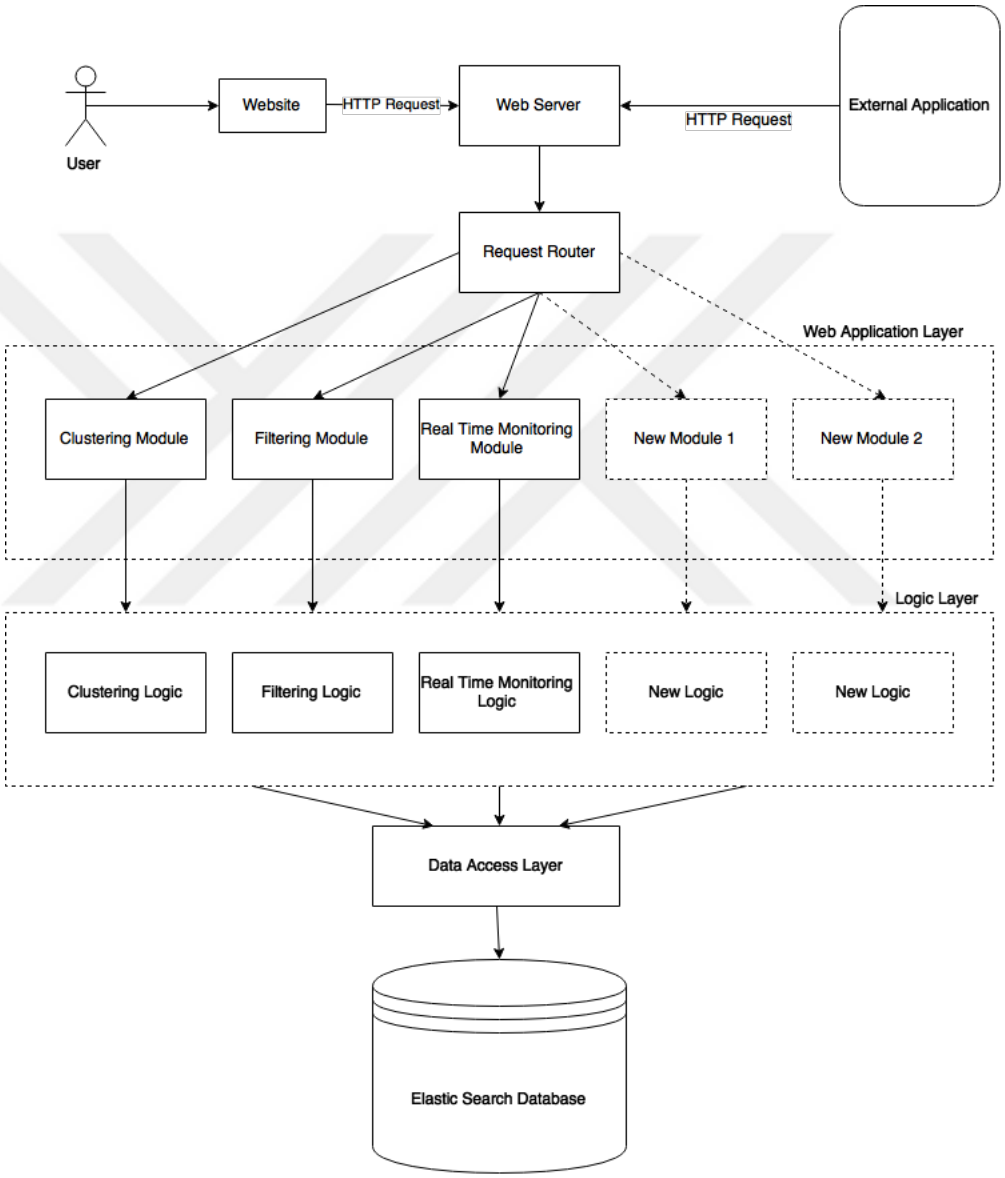


Figure 2: Overview of WATT

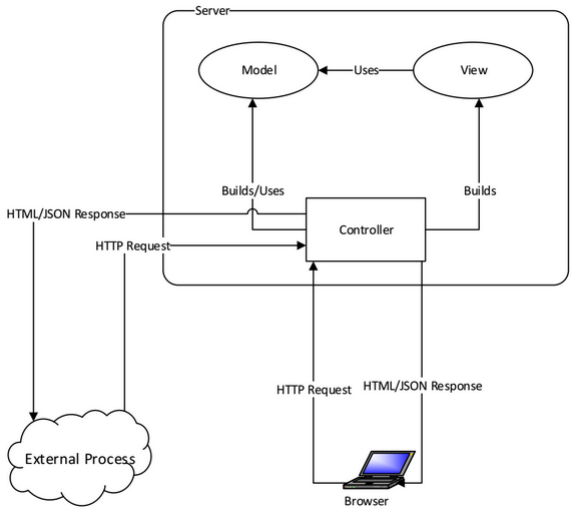


Figure 3: MVC Architecture

clustering process or query other things over the network by sending a REST request to the request router.

4.2 Logic Layer

This layer is responsible from all of the manipulation on the data objects. The set of algorithms we define and explain in chapter 3.1 is in this layer. Therefore it can be said that this layer is the core layer or the essential layer of the software. The main functionality resides in this layer. A simplified summary of this layer can be seen from 4. Preprocessing, Clustering / Bucketization, Labeling, Exploratory Analysis and Active Learning are the modules inside this layer that give the user a hindsight about their twitter data. 5 is a screenshot that belongs to the labeling module of the logic layer. One can see the current labeled and unlabeled data amount from the pie chart at the top of the page. At the bottom of the page we provide labels and corresponding tweets to the user where each label is a button itself. One can use the buttons to label the data and informed about the new labeled and unlabeled volume of the data. Figure 6 belongs to the exploratory analysis part. All clusters and their elements sorted according to the size in descending order can be seen from this module. One can evaluate clusters and decide if they are healthy or not.

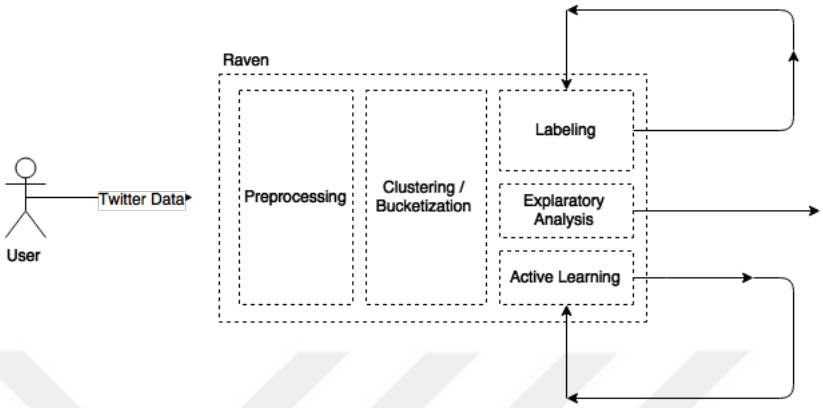
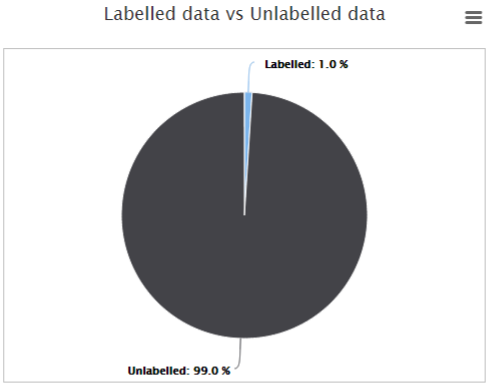


Figure 4: Feature View of WATT



Tweets

Index	Sentiment	Sentiment	Sentiment	Reason	User	Tweet	Label
0	Positive	Objective	Negative		owner	RT @gtatlipinar: İHANET DEĞİLSE NEDİR BU KOBANİ EYLEMRİNDE ARABA YAKMAYA ÇALIŞAN PARALEL POLİS GÖREVDEN ALINDI	-2
1	Positive	Objective	Negative		owner	RT @YilmazGedik: Kobani Halk Meclisi Başkanı Ayşe Efendi: Türkiye IŞİD?in Yenilmesini Engelleyen Güçtür...	-2

Figure 5: Labeling UI

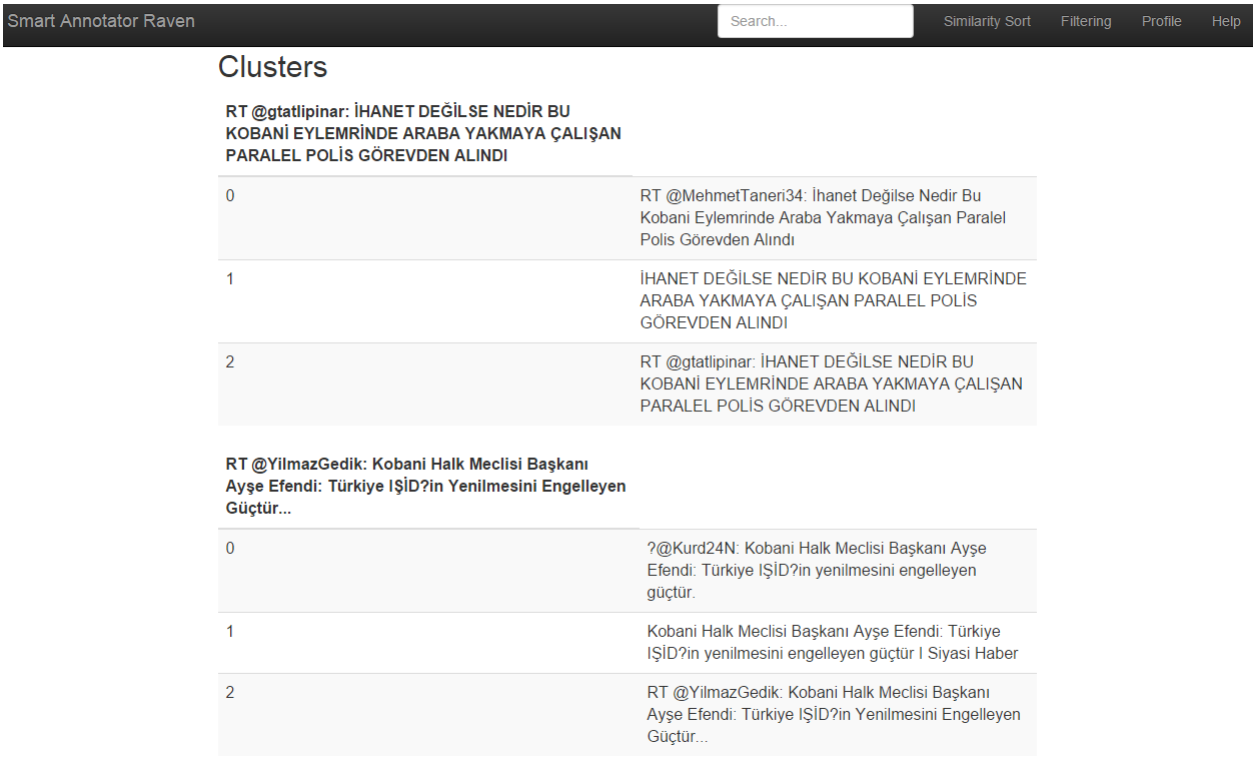


Figure 6: Cluster View UI

4.3 Data Access Layer

Data Access Layer contains the code segment to reach the Elastic Search (ES) database. This layer is responsible of the communication between the Elastic Search database and the logic layer. When logic layer calls the data access layer with a certain request, data access layer proceeds to connect the ES database and query over all the available nodes. Results are stored in a JSON object format, represented as a Hash Map of type <String,Object>. Suppose we want to retrieve the text of a tweet with id 293243. Logic layer calls the getDocument function from the Data Access Layer and data access layer returns an object of type Map <String,Object>. If the developer wants to reach the text of this tweet, then one have to run Map.get("'text"'); function on the existing Hash Map data structure. We wanted to use the flexibility provided by the ES as much as possible by loading the data into a Hash Map of type object. Note that this also allows us to create unstructured data entries inside the ES database. For example a document can have a field called time stamp while the other documents in the same table does not need to have a field called time stamp. If such a field

exists, a developer can always reach that field of the object by loading the JSON object into a Hash Map structure and making a query with the key value. We believe this provides an important flexibility.

5 Experimental Evaluation

In this work, we run several experiments to validate our results. Firstly we used 3 data sets to work with, (1) Ozgecan Aslan with the hash tag #sendeAnlat, (2)tweets following the attack on Charlie Hebdo and (3) the tweets following the political conflict in Kobani, a region in Syria near the Turkish border. All data sets had more than 10.000 tweets which we believe is more than enough to generalize our results. Tweets are collected based on keywords and hash tags with using live stream API of twitter. Live stream API of twitter allows the user to collect specific tweets that a user is interested in up to a certain volume. For the Charlie Hebdo case we collected tweets with hashtag #CharlieHebdo. For Kobani case we collected tweets containing the keywords Kobani, Hatay, Halep. The name of 3 cities that is geographically close to the conflict. Lastly for the tweets about the murder of Ozgecan Aslan we collected the hash tag #sendeAnlat. A hash tag that is encouraging people to share their personal stories about sexual harassment. Out of all these sets we were able to reduce the data size to approximately 20% of the original data size in worst case scenario.

5.1 Evaluating the Cluster Quality

First experiment we conducted was to measure the cluster impurity by doing a pairwise similarity calculation within clusters. This method first takes a cluster c from the cluster set C. Then we calculate $LCSSimilarity(t_i, t_j) \forall t \in c$ and $c \in C$. Then we calculate the average value for these scores and we normalize them according to the size of the cluster. This is needed because we wanted to reward the algorithm more for creating a pure and a big cluster rather than forming a cluster with less size but same purity. Needless to say in the ideal case, we

want our algorithm to form the biggest cluster possible with the maximum purity. The score we use to calculate this similarity is lcs similarity again. This time however, we calculate this score for all elements of a cluster. Needless to say the complexity for this experiment is quite high.

Algorithm 3 InterClusterPurity (*Cluster set: C*)

```
1:  for i = 1 → C do
2:      cluster = S[x]
3:      for j = 1 → c_i do
4:          pair1 ← cluster[j]
5:          for k = j + 1 → c_i do
6:              pair2 ← cluster[k]
7:              score ← LCSScore(pair1, pair2)
8:              score = score ∗ |c|
9:              clusterScore[k] ← score
10:         end for
11:     end for
12:     averageScore = calculateAverage(clusterScore)
13: end for
```

After this calculation we end up with a similarity value, calculated for each cluster and weighted according to the cluster size. Formally we have a score list for each $c \in C$ where a score is denoted as $score_{c1}, score_{c2}$... etc. Then for all the scores we calculate, $TotalSize = c_1 + c_2 + c_3 + ... + c_i$ and $TotalScore = score_{c1} + score_{c2} + score_{c3} + ... + score_{ci}$. The final value is calculated as: $FinalScore = TotalScore/TotalSize$. We believe this value is a good purity metric for two reasons, it is deterministic unlike the algorithm we propose here. Which means it does not involve a trade-off between the speed-accuracy. Therefore the value we will have here is definite. The other reason is that we believe this is a pretty good measure rewarding the right things while also punishing the algorithm. The metric we describe rewards the algorithm for forming the biggest pure cluster possible. Lets say we have 2 cluster c_1 and s_2 and lets say the purity of these clusters are the same, but in terms of size $c_1 > c_2$. Our metric we provide assigns a better score to c_1 than c_2. It makes more sense since forming a bigger cluster with the best purity possible would be the ideal case in our work.

We also calculated extra cluster similarity score and evaluated the results. Extra cluster similarity is calculated by picking a cluster first from our cluster set C. Say we picked a cluster c_1

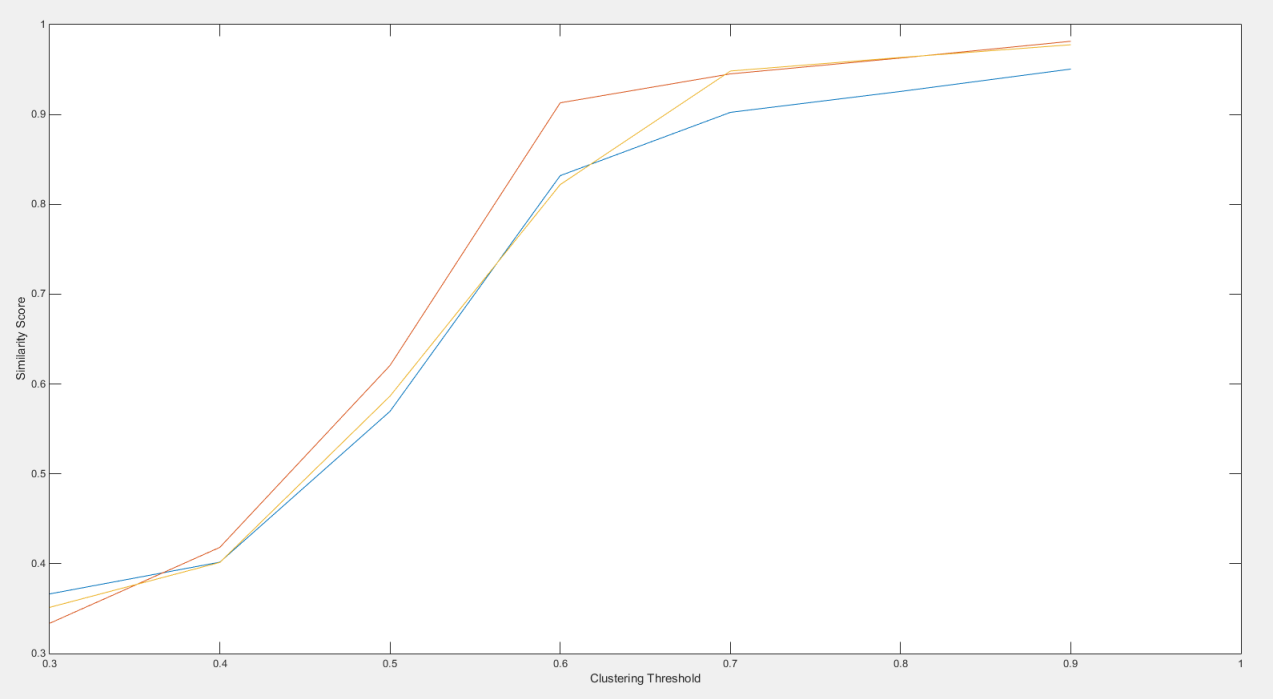


Figure 7: Cluster Purity vs. Threshold

we then proceed to merge the rest of the clusters to a single cluster which can be expressed as $c_{N1} = c_2 \cup c_3 \cup c_4 ..... \cup c_n$. At this step of the algorithm we have two clusters, c and c_{N1}. Then we proceed to measure the pairwise similarity between these clusters by picking a tweet t from the first cluster c and comparing t with every other element in c_{N1} with its counterpart cluster. The average similarity score of all data set is 0.38. Therefore we expect the extra cluster purity score to be lower than 0.38, this is a good indication that we are clustering the "'right items"' i.e items that are similar to each other to minimize the pairwise difference between the whole data set. If the resulting extra cluster score after the execution of our clustering algorithm is higher than the all pairwise similarity score, then we can state that our clustering algorithm fails to create clusters that are similar with each other.

The average similarity score between every pair of the data set is 0.38.
We experimented with 7 different thresholds and calculated the similarities within clusters with the method we explained above. Which means we created some clusters by executing the algorithm with 7 different thresholds, respectively 0.9, 0.8, 0.7, 0.6, 0.5, 0.4 and 0.3. From the resulting clusters we calculated the similarity measure inside those clusters.

From figure 7 we can clearly see that there is a hard threshold in between 0.5-0.6. This is

clearly an indication for the optimal threshold to run the algorithm. Note that as we explained in section 3.1 higher threshold values result in much more running time compared to the lower thresholds, also while higher threshold values take longer time they also fail to capture the paraphrases and comments. They create more than 1 cluster for the same topics and same comments on the same events/objects. This is not the desired behavior. On the other hand executing the algorithm with lower thresholds such as 0.30 result in very short time, however as you can clearly see from the graph purity of the clusters are not in the desirable levels. If we were to make a comparison between the higher thresholds, that is 0.6, 0.7, 0.8, 0.9. We clearly see the difference cluster purity between these thresholds are quite low. Considering the execution time of the algorithm we believe the most optimal choice for threshold is around 0.6.

Also we have calculated the average lcs similarity between all pairs in the data set. The algorithm to test the all pairs longest common sub sequence value is pretty straightforward.

Since our similarity score is symmetric there is no need to compare $pair_i$ and $pair_j$ again. Therefore the second loop of this algorithm starts iterating from $i + 1$. The average pairwise similarity score gives us a hindsight about which thresholds can be used to distinguish between clusters.

In this part of the experiments we came up with a distance metric based on the lcs score of two strings. That is if $string_1$ and $string_2$ are similar with a similarity score s, we calculate the distance between these strings as $d_{12} = 1 - s$. The maximum value of d is 1 and minimum value is 0. Needless to say, if the strings are exactly the same the distance between those pairs are 0 and if the strings do not share a common sub sequence the distance is 1.

5.2 Multi Dimensional Scaling

In order to experiment on our clustering algorithm and visualize the data we use a method called Multidimensional scaling. In order to achieve this we first create a distance matrix DM by calculating the similarity score between every pair.

$$\mathrm{DM}_{m,n} = \begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,n} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m,1} & d_{m,2} & \cdots & d_{m,n} \end{pmatrix}$$

This distance matrix is a symmetric matrix by definition due to the symmetry property of our distance measure. We have stated before that our distance measure is calculated as $d_{12} = 1 - s$ where s is the longest common sub sequence score between two strings. Therefore $DM = DM^T$. This property actually reduces the computation time of the distance matrix since for the half of the matrix one does not need to calculate the distances again. However the order of the complexity does not change.

This is a very expensive operation since we calculate this matrix in N^2 operations where N is the number of elements in the data set. For each of these operations we also calculate the longest common sub sequence of these strings. The complexity of lcs was $O(n * m)$ as we described earlier. Combined the algorithm to calculate a distance matrix from N^2 pairs are $O(N^2(m * n))$ where m is the length of the first string, n is the length of the second string. However we only use these expensive methods to check if our approach yields a logical clustering scheme.

After this step the goal of the multidimensional scaling is to find N points in space so that for each point we preserve the distances between those pairs. Formally, we want to find the coordinates of N points where $d_{11}, d_{12}, ...d_{1n}.....d_{21}, d_{22}.....d_{nn}$ is preserved.

With this method we can map the distance matrix into a space with a dimension choice of ours. Intuitively the points that are closer should fall into the same cluster whereas the points that are far away should belong to other clusters. This is a common data mining concept that is relevant with Hierarchical clustering. In order to further understand the performance of our algorithm we colored the strings with different colors so that it is possible to distinguish between clusters. Figure 8 represents a data clustered with our algorithm with threshold 0.6. Different thresholds result with different clustering results as expected however even though the distance metric is the metric we use to do the clustering we also mentioned in part 3.1 that we also introduce an error margin to the algorithm in order to reduce time complexity. Therefore it is possible to see two pairs such that their distances is larger that the 1 - threshold

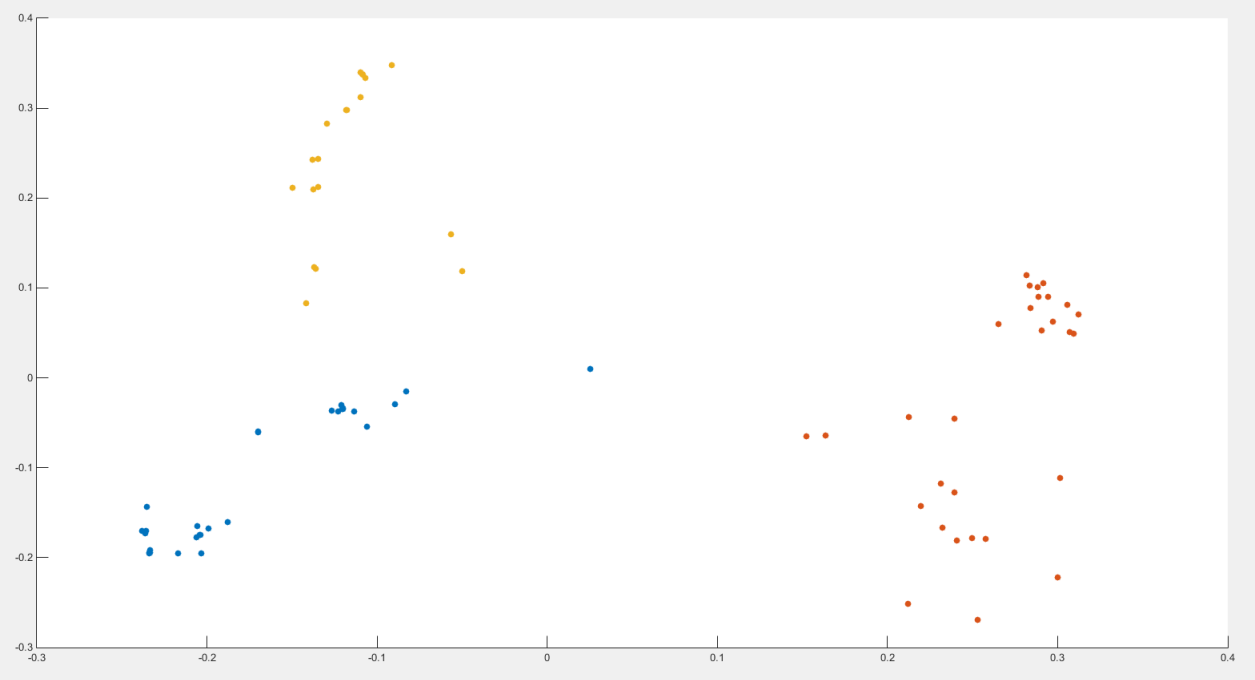


Figure 8: Multidimensional Scaling of 3 clusters

value. In this case since we clustered this data with a threshold of 0.6, it is possible to see two points that are more than 0.4 units away but still belong to the same cluster.

Interestingly the clusters we form with the optimal threshold 0.6 satisfy the Pareto principle. There are some research involving the Pareto principle in computer science such as [18]. We have noticed that the 20% of the clusters we got as the result of the execution of our algorithm contains the 80% of the data set. Which means by covering the top 20% of the clusters we nearly cover all of the data. This indicates that by looking at the top 20% of the data we can see nearly all of the topics being discussed in the data set. This is a useful way to understand and summarize the whole data set.

From 9 and 10 you can see the cluster size distribution with respect to the threshold value being used. Low thresholds result in less clusters and big cluster sizes while high thresholds result in many clusters with small cluster sizes. Both situations are not desired considering the functionality of this clustering algorithm. The distribution in terms of the percentages were all same for all the data sets we have experimented on. Satisfying Pareto principle allows us to carry out labeling tasks much more efficiently and allows us to boost the performance of our classifier with Active Learning principles, especially by using uncertainty sampling.

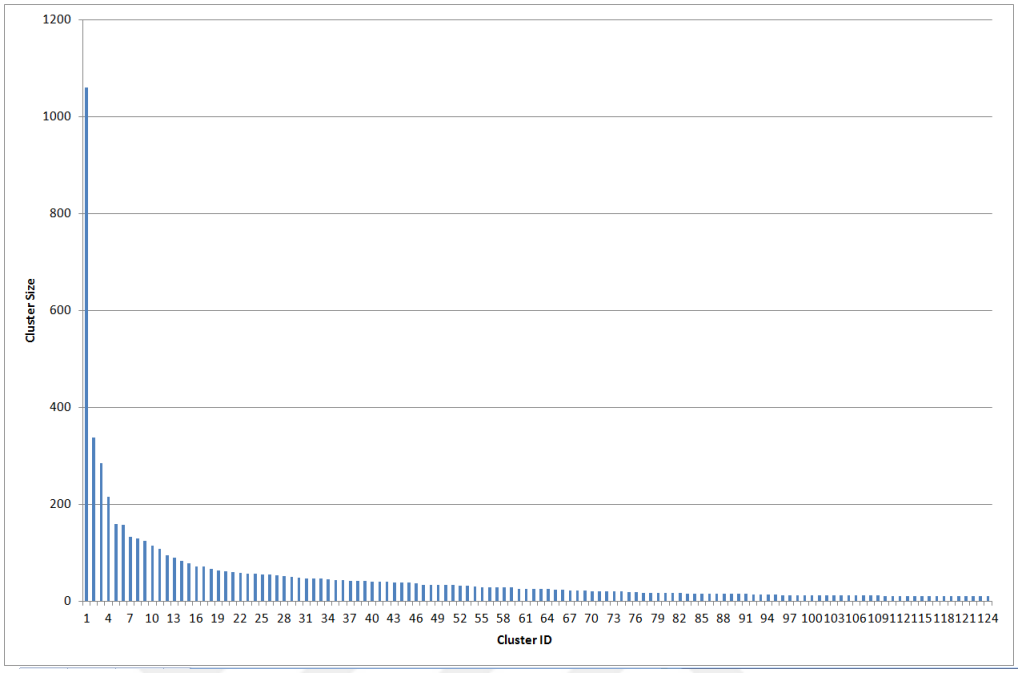


Figure 9: Cluster Distribution with 0.6 threshold

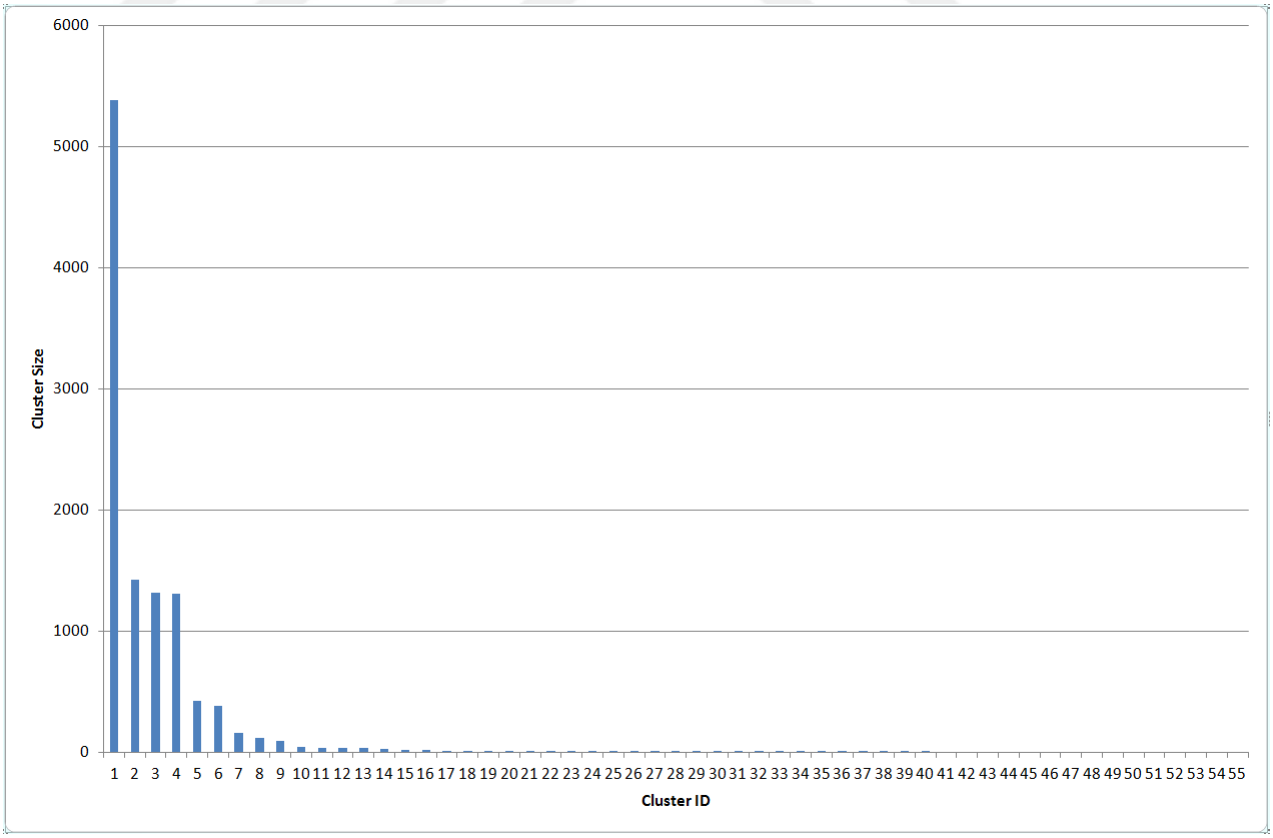


Figure 10: Cluster Distribution with 0.4 threshold

5.3 Human Labeling

In this experiment we compared the results from the algorithm with human clustered results. We asked the users to inspect the top 20 clusters produced by the algorithm and mark the tweets which they think does not belong to the assigned cluster. In top 20 clusters we had 2326 tweets, it was decided that 2304 tweets were placed to the right clusters while 22 of them were misplaced. Interestingly 22 of the misplaced tweets were in the same cluster. For example our algorithm misplaced the tweet: ”RT @imctelevizyonu: Kobani Kantonu Başbakan Muslim: Kobani’nin yuzde 70’i YPG denetiminde” into a cluster with a cluster representative: ”RT @imctelevizyonu: Kobani Kantonu Başbakan Muslim: Kobani butun dunya için sembol haline geldi” .This result indicate that our algorithm agree with human judgement up to 98%.

6 Related Work

In this thesis, we are providing an algorithm to determine the similar tweets. There are extensive research on measuring the similarity between documents or long texts and some literature on short texts [8]. However for the special domain of twitter there are no or very little research. [8] uses machine learning techniques and synonyms in order to calculate a similarity measure from short paragraphs, however their work is not suitable for the twitter domain as the length of the strings are limited to 140 characters in twitter domain. Also their definition of similarity is different from the similarity definition we provide in this thesis. Authors define the two texts similar if they share the same focus on a common concept. However our similarity definition is a bit more specialized, we observe that with the method we use similar texts not only share the focus object but they also share same ideas on same concepts.

[7] is another work that focuses on document similarity measures by using map reduce techniques. They build a similarity matrix by using map reduce framework from all the pairs. Since the time complexity of building a pairwise similarity matrix is exponential the main contribution of their work is to make this calculation efficient. *Similarity measure* defined in [7] is described as the multiplication of two vectors. A document is represented as a vector denoted as W_d of term weights $w_{t,d}$ that indicates the importance of each term t in the document. Note that they ignore the ordering of these terms. Similarity measure is defined as:

(4)

V is the vocabulary set and d_i, d_j are two documents in comparison. A term will contribute to the similarity measure only if it has non-zero weights in both. Their algorithm uses this property to build the similarity matrix. The inverted index structure they use also takes the advantage of this property.

[10] also uses LCS in order to measure the similarity between two strings. They propose an algorithm that is modified in terms of normalization. They call their similarity measure Normalized longest common sub sequence (NLCS) which is,

$$v_1 = NLCS(r_i, s_j) = \frac{length(LCS(r_i, s_j))^2}{length(r_i) * length(s_j)} \tag{5}$$

However they also use a semantic dictionary in order to measure the distances between words. This is a common practice and used extensively through Information retrieval and NLP topics. Researchers usually use semantic networks or knowledge-based measures to calculate how similar given two words are. While these methods have certain advantages such as detecting synonyms that otherwise would yield 0 or low similarity scores. They bound the method being used to a single language since every language has its own semantic network. The method we describe in this thesis works independently of the language being provided except for the active learning part where we train a classifier to determine the sentiment. However on several sections we already stated that sentiment analysis is not the main contribution of this work.

7 Conclusion and Future Work

In this thesis, we propose two new clustering algorithms to group the relevant/similar tweets into clusters. One is computationally lightweight compared to the other algorithm while being less accurate. There are several reasons behind the need for clustering, we want to summarize the data set without losing the important topics/subjects and use the summarized version of the whole data set for different purposes.

We show that twitter data is mostly redundant and can be grouped into clusters based on relevancy. There are also some concerns that we wanted to address such as bot detection, labeling and active learning. We are able to distinguish tweets that are produced by bots or automated accounts since we cluster all of the automated tweets into a spam cluster. By observation most of the spam/bot tweets try to hijack popular hash tag in order to be visible. However they mostly tweet about the same thing which is contextually unrelated to the hash tags they are hijacking. Both of the algorithms mentioned here can distinguish such tweets and place them in a separate cluster or to a separate bucket.

As we explained in chapter 5 the output of the algorithm usually follows the Pareto principle where the top 20% of the clusters contain the 80% of the data. Therefore we shorten the labeling time at about one fifth of the original time. In addition to shortening the labeling time, in order to generate training data, we also aid the user to label the data that are otherwise cannot be labeled through automatic means. One example is the political situation around Kobani and Hatay. There are too many sides of the political debate that it is nearly impossible to understand if a given tweet is pro or anti government. In order to be able to answer such questions when given a tweet, we aid the user to label the whole data by selectively offering the best cluster to label.

One other thing that we applied this is active learning. From the clusters we form, we provide the cluster that the classifier is most unsure of to the user so he/she can label it and allow the model to improve. We choose to adapt uncertainty sampling technique to go with our clustering algorithm. This part of the thesis is integrated into the clustering algorithms as the classifier determining the sentiment of the sentences does not determine the sentiment of a single tweet but rather determines the sentiment of a given cluster. Since a cluster contains more than one tweet we boost the accuracy of our classifier by providing the classifier with the required training data to adjust itself. The cluster we pick in order to adjust the classifier is picked because of the low confidence value that the classifier provides. Therefore we aim to boost the performance of our cluster by providing it more than one training data on the data set that it is having the most trouble. However we believe the other methods in active learning can also be used and integrated into the clustering algorithm we describe. The main idea here is that the classifier does not label/train on single tweets.

We evaluated the results based on a string similarity metric called LCS. We also used human perception to evaluate the cluster health. We visualized and further explained the algorithm we describe in 3.1 part. We also explained that the algorithm runs with a threshold and we conducted experiments to determine the optimal thresholds. While we observed that there are no single optimal threshold value for all the data sets, it is usually a value between 0.5-0.6. Theoretically it is not right to say that two strings are similar in terms of meaning if the distance between two strings in terms of the score we propose in 3.1 is less than 0.4. However by inspecting the data we couldn't be able to come up with such examples and cases where two strings have completely different meanings with less than 0.4 distance. The figure 7 explains the experiments and their results in order to find this optimal value.

Considering our experimental results we state that the algorithm we propose forms clusters that are relevant with each other according to human perception. There are also various usage areas for practical applications of this algorithm as we mentioned such as data summarization, bot detection and active learning.

References

[1] Gandhe A. C. Lazarus R. Ssu-Hsin Yu Achrekar, H. Predicting flu trends using twitter data. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference*, pages 702 – 707. IEEEXPLORE, 2011.

[2] Ilia Vovsha Owen Rambow-Rebecca Passonneau Apoorv Agarwal, Boyi Xie. Sentiment analysis of twitter data. pages 30–38, 2011.

[3] İnanç Arın. Identification of anonymous users in twitter. 2012.

[4] H. Raita T. Bergroth, Hakonen. A survey of longest common subsequence algorithms. pages 39 – 48.

[5] Derya Birant and Alp Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.

[6] Ronald L Breiger, Scott A Boorman, and Phipps Arabie. An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling. *Journal of mathematical psychology*, 12(3):328–383, 1975.

[7] Tamer Elsayed, Jimmy Lin, and Douglas W Oard. Pairwise document similarity in large collections with mapreduce. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 265–268. Association for Computational Linguistics, 2008.

[8] Vasileios Hatzivassiloglou, Judith L Klavans, and Eleazar Eskin. Detecting text similarity over short passages: Exploring linguistic feature combinations via machine learning. In *Proceedings of the 1999 joint sigdat conference on empirical methods in natural language processing and very large corpora*, pages 203–212. Citeseer, 1999.

[9] Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. Ontonotes: the 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*, pages 57–60. Association for Computational Linguistics, 2006.

[10] Aminul Islam and Diana Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(2):10, 2008.

[11] Joseph B Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2):115–129, 1964.

[12] Rafal Kuc and Marek Rogozinski. *ElasticSearch server*. Packt Publishing Ltd, 2013.

[13] David D Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the eleventh international conference on machine learning*, pages 148–156, 1994.

[14] Udi Manber et al. Finding similar files in a large file system. In *Usenix Winter*, volume 94, pages 1–10, 1994.

[15] Samuel D. Gosling Michael Buhrmester, Tracy Kwang. Amazon's mechanical turk a new source of inexpensive, yet high-quality, data? pages 3–5, 2011.

[16] Isabelle Stanton Robert E. Tarjan Nina Mishra, Robert Schreiber, 2014.

[17] Sandra L Robinson and Rebecca J Bennett. A typology of deviant workplace behaviors: A multidimensional scaling study. *Academy of management journal*, 38(2):555–572, 1995.

[18] Paula Rooney. Microsoft's ceo: 80-20 rule applies to bugs, not just features. News 1648, CRN.com, 2002.

[19] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294. ACM, 1992.

[20] Janyce Wiebe Alexander Hauptmann Wei-Hao Lin, Theresa Wilson. Which side are you on? identifying perspectives at the document and sentence levels. 2006. [Online; accessed 22-July-2012].

[21] J. Yarow. Business Insider Article. businessinsider.com/twitter-stats, 2010.

[22] Jingbo Zhu, Huizhen Wang, Tianshun Yao, and Benjamin K Tsou. Active learning with sampling by uncertainty and density for word sense disambiguation and text classification. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 1137–1144. Association for Computational Linguistics, 2008.