university of groningen / faculty of arts

Detecting local events in the Twitter stream

by

Chris Pool

Bachelor thesis
Information sciences
Chris Pool
S2816539
June 14, 2015

DETECTING LOCAL EVENTS IN THE TWITTER STREAM

CHRIS POOL

Chris Pool
S2816539
June 14, 2015

PREFACE

This thesis is written as part of my Bachelor Information sciences at the University of Groningen that is part of my one year pre-master.

I would like to thank my supervisors Malvina Nissim and Johan Bos for helping me during my research. I would also like to thank Rob van der Goot, PhD at the University of Groningen who helped me with creating a Named Entity Classifier.

Most of the programming is done together with David de Kleer, I would especially like to thank him for the pleasant collaboration.

Chris Pool

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PYTHON CODE

ABSTRACT

With the growing number of people using social media services like Twitter it becomes interesting to use this enormous amount of data as a source of information about things happening at a certain location and time.

The goal of my research is to develop a system that uses tweets with geo-information to detect events taking place at a specific location, for example concerts, football matches, fires or traffic jams and recognize the named entities in those events. Events that take place on a larger scale like elections and extreme weather conditions are ignored. This event detection part of my research is done together with David de Kleer, student at the University of Groningen. For the the final part of my research I developed a system that can detect important entities in the events returned by the system.

The first step is collecting the data. We used the tweets available on the Karora server[1]. This server stores all Dutch tweets and can be easily accessed. We downloaded all tweets with geo-information published in May 2015 to use in developing the system. Tweets from the last two weeks in April 2015 are used to test the system.

The second step is to create clusters of collected tweets, these clusters are potential events that we call *eventCandidates*. We create these *eventCandidates* by clustering tweets by location and time. To make these *eventCandidates* an algorithm is used called GeoHash, this algorithm converts latitude and longitude coordinates into a hash, this hash indicates an area where the location is in. The more digits used for this hash, the smaller the area, and thus more accurate. For clustering by time I use the UNIX timestamp, this timestamp is a system that represents time in seconds since 1 January 1970.

We both annotated the same datasets making it possible to calculate the inter-annotator agreement, the result is a kappa score of 0.79 for our training set en 0.8 for our test set, bot scores indicate a good agreement(Manning *et al.* 2008). With the annotated data we trained a Naive Bayes classifier using NLTK. Using the development dataset as training data and the test dataset as test data this resulted in an accuracy of 84%, with a upperbound of 86% and baseline of 46%[2] we see this as a good result.

With the classified *eventCandidates* using the categories: No event, Meeting, Entertainment, Incident, Sport and Other I tried to detect named entities within those events. The Stanford Named Entity Recognizer was trained using automatically annotated tweets. The results are reasonable, but it shows the difficulty in detecting entities in such short and noisy pieces of text.

The events, including the named entities, are plotted on a Google map making it easy to check out the events.

1 Linux server available for students and staff of the University of Groningen
2 Baseline: Every event candidate is regarded as no event which is the largest category

1 INTRODUCTION

With the growing number of people using social media services like Twitter makes it interesting to use this enormous amount of data as a source of information about things happening at a certain location and time.

About 3% of all tweets contains geo-information[3]. That looks like a small portion but because of the enormous amount of tweets available this gives great insights about the opinion of users at a certain location. This can be used for example to provide news items with more information about the opinion of people. The research of Walther & Kaisser was an inspiration for this research. They did a research about event detection using tweets from the New York metropolitan area.

This event detection part of my research is done together with David de Kleer, student at the University of Groningen. The final part of my research was to develop a system that can detect named entities in the events.

This thesis describes a method of finding local events in the Netherlands based on tweets with geo-information and categorizing them into five categories (No-event, Sport, Entertainment, Meeting and Incident). Also a method to find relevant information within those events about the location and people involved is described.

My research questions are:

1. Is it possible to automatically detect local events in the Twitter stream that take place in the Netherlands based on Tweets with geo-information?
2. if so, is it possible to detect what type of event it is?
3. and is it possible to detect the important entities in these events?

This thesis is divided in four chapters. The first chapter, related literature, describes the research that has already been done about this subject. The second chapter is the method section where I describe the methods we used for our experiment. In the third chapter I describe our experiment, all code I am referring to can also be found in the GitHub repository[4]. In the final chapter you can find my conclusion.

3 Based on my own data, see chapter 2 for literature research
4 http://www.github.com/chrispool/thesis

2 RELATED LITERATURE

In our research we try to detect events in a large amount of data , in our case tweets. Event detection in newspapers and other media has been addressed in the TDT research program, this program tries to find and follow events in news stories published in newspapers or shown on TV. According to Allan *et al.* (1998) events can be defined as *"real-world occurrences that unfold over space and time"* (Allan *et al.* 1998).

Since the launch of Twitter in 2006 the popularity of this micro-blogging service is increasing rapidly. With 302 million users sending about 500 million tweets per day this is a huge source of information (Statista 2015). In the Netherlands the number of active users was 2.8 million in 2015 according to the research conducted by Turpijn *et al.* (2013).

Unfortunately there are no figures available about the number of tweets with geo-information. Doing a small test on our own data has shown that about 3% of tweets has geo-information. Figure 1 is a heatmap of tweets with geo-information and shows a lot of tweets in the Netherlands. The Netherlands is one of the countries with the highest twitter accounts to population ratio (Dawson 2012).

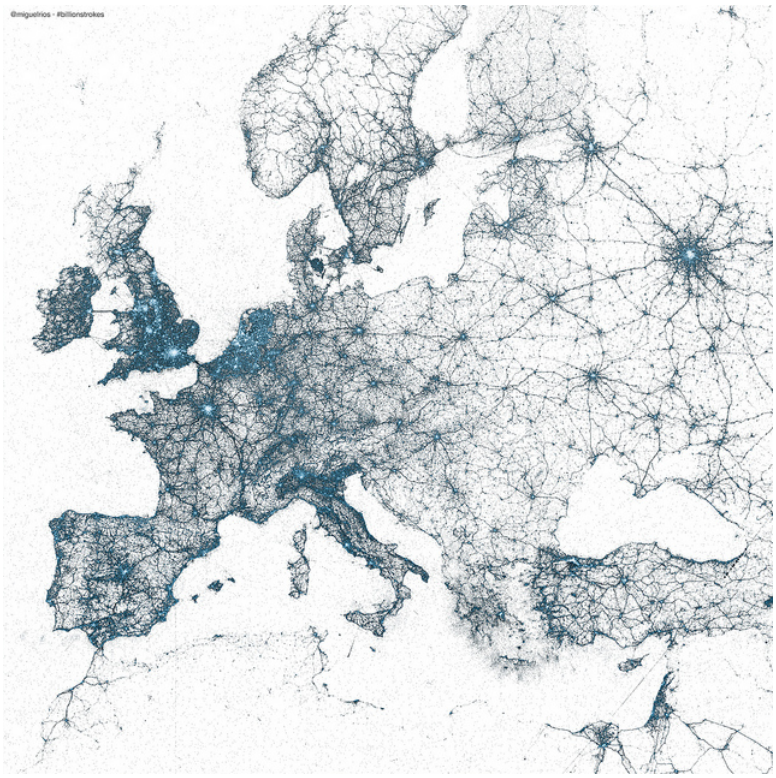

Figure 1: Tweets with geo-information (Ríos 2014)

Using geo-information from tweets has been done in several works, for example Sakaki *et al.* (2010) in their paper *Earthquake shakes Twitter users: real-time event detection by social sensors* uses geo-information of tweets to detect earthquakes by analyzing all tweets that contain certain keywords like *earthquake* or *shaking*. In the research by Walther & Kaisser (2013) they describe the process of detecting events in the Twitter stream, this research was the

inspiration for my research and builds on several techniques they discuss.

Data clustering is a field where a lot of research is done but according to Birant & Kut (2007) most studies focus on discovering clusters from ordinary data (non-spatial and non-temporal data), so they are impractical to use for clustering spatial-temporal data. They see knowledge discovery from spatial-temporal data as very promising (Birant & Kut 2007). There are five different types of clustering algorithms: Partitional, Hierarchical, Grid-based, Model-based and Density-based (Han *et al.* 2006). Because most algorithms cluster the data using a frozen dataset it is not possible to add values without clustering the data again making it impossible for real-time use.

GeoHash is is a system invented by Gustavo Niemeyer[5] that converts latitude and longitude coordinates into a *geoHash*. The hash represents an area where in the location is present. The length of the hash indicates the precision, more digits used indicates a smaller area, and thus a higher precision. This grid based algorithm can be used to make clusters of our tweets that works very fast and can also be used in real-time.

For detecting important entities in the events I want to use Named Entity Recognition. NER is the proces of labeling data with corresponding categories like person, organization or location (Manning *et al.* 2008). Current NER tools were designed to process large texts and perform poorly with tweets due to the noisy and short style of tweets, also the limit of 140 characters makes it difficult to recognize entities because the lack of context (Alan Ritter & Etzioni 2011 and Liu *et al.* 2011). A key feature for named entity recognition is the use of capital letters, unfortunately the use of capital letters in tweets if often less reliably and make it hard to recognize entities. According to Alan Ritter & Etzioni (2011) the Stanford NER, a state of the art classifier, focusses to much on capital letters making it perform poorly on tweets.

5 http://en.wikipedia.org/wiki/Geohash

3 DATA EN METHOD

The goal of my research is to develop a system that can detect events that take place at a specific location, for example concerts, football matches, fires and traffic jams and recognize the named entities in those events. Events that take place on a larger scale like elections and extreme weather conditions are ignored. This chapter describes the methods used in our system and the motivation why we chose these methods.

3.1 Data collection

For the experiment tweets are used that have geo-information. Using a simple Grep[6] command it was possible to retrieve only tweets with this information from the Linux server *Karora*[7] that is available for students and staff of the University of Groningen. For our system we retrieved two datasets, one for training and developing our system that we call *devset* and one for testing our final system that we call *testset* The *devset* consists of Dutch tweets posted in March 2015 that we have downloaded from *Karora*. In total there where 566,549 tweets with geo-information. That is about 3% of the total number of tweets published in that month. The *testset* consists of 165,848 tweets published in the second half of April 2015. The tweets are stored in a tab separated text file with the fields tweet, latitude, longitude, username and timestamp

3.2 Pipeline

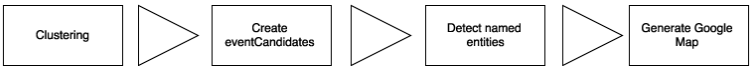

Figure 2: Pipeline

The process is to first cluster the tweets by location and time and to determine if the clusters are events or not and if so, which type of event it is. The final step is to detect the interesting entities in the events. In this section you can find a description of the used methods. The implementation can be found in the next chapter.

6 Command-line utility for searching plain-text data sets for lines matching a regular expression
7 karora.let.rug.nl

3.2.1 *Clustering*

The first task is to cluster the tweets by location and time. There are several approaches in location clustering that can be used as discussed in the previous chapter. We wanted to use a clustering algorithm that potentially could work in real-time so it was necessary to look for an approach that was quick and accurate. GeoHash is is a system invented by Gustavo Niemeyer[8] that converts latitude and longitude coordinates into a *geoHash*. The hash represents an area where in the location is present. The length of the hash indicates the precision, more digits used indicates a smaller area, and thus a higher precision. If two locations share the same prefix it indicates (not always[9]) that the locations are nearby.

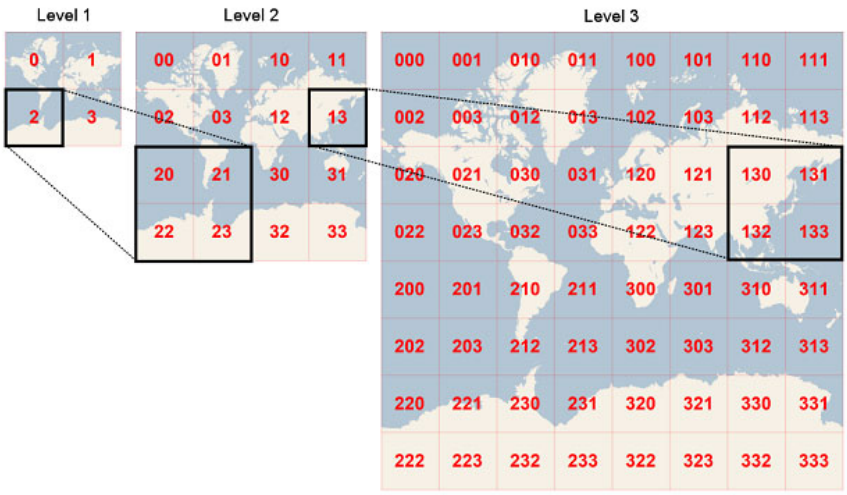

Figure 3: Geohash

Figure 1 illustrates how geoHash works. The first digit indicates which halve of the earth the area is on. The second digit divides that area in four areas and that continues resulting in table 1 that shows the different lengths of the *geoHash* and the accuracy of that length. Because the dimensions of the area are different at each latitude the values represent the worst-case scenario at the equator.

8 http://en.wikipedia.org/wiki/Geohash
9 Some neighboring geoHashes start with a different character as can be seen in figure 1

Table 1: Geohash Precision

GeoHash length	Area width x height
1	5,009.4km x 4,992.6km
2	1,252.3km x 624.1km
3	156.5km x 156km
4	39.1km x 19.5km
5	4.9km x 4.9km
6	1.2km x 609.4m
7	152.9m x 152.4m
8	38.2m x 19m
9	4.8m x 4.8m
10	1.2m x 59.5cm
11	14.9cm x 14.9cm
12	3.7cm x 1.9cm

There are several implementations of this algorithm in Python. We used the GeoHash library written in 2009 by Hiroaki Kawai[10] because of the good documentation and excellent features. The function to calculate the *geoHash* of a location requires the desired accuracy, latitude and longitude of that location and returns the *geoHash* for that location as can be seen in code example 1.

```
lon = 52.348
lat = 6.2342
accuracy = 7

hash = geohash.encode(lon,lat,accuracy)
>>> print(hash)
u1k3y1c
```

Python code 1: Example of geoHash encoding

This library also provides functions to calculate the neighboring geoHashes and this is useful because in a grid clustering approach border cases can occur. We use this functionality to merge our neighboring *eventCandidates* as described in 4.3 Creating eventCandidates.

For clustering timestamps the UNIX timestamp can be used. This is a system to represent time using the number of seconds since 1 January 1970. Python has built-in functions to deal with UNIX timestamps. The tweets are sorted on timestamp making it easy to make clusters of time series. The implementation of these functions can be found in chapter 3.3 Event Detection.

3.2.2 *Classification*

After forming the potential events (*eventCandidates*) the Event Candidates have to be classified using the categories:

- No Event
- Meeting
- Entertainment
- Incident

10 https://github.com/hkwi/python-geohash

- Sport
- Other

We decided for these categories because after looking at the data these made the most sense. The Meeting category, which is to largest in both datasets, could be divided further. For example political meetings or rallies ,congresses and book presentations are part of this category but for our goal, to plot them on a Google map we did not want to use too many categories. It is possible to train the classifier with other categories but requires a new annotation.

In our experiment we have chosen to use the MaxEntClassifier, NaiveBayes and a SVM classifier. Using NLTK it was easy to train the classifiers. Each classifier requires the same format of train data making it easy to compare the results of the different classifiers. In chapter 4 you can find a more detailed description about training the classifiers.

3.2.3 *Named Entity Recognition*

For finding the important persons, organizations and locations I want to use Named Entity Recognition(NER). NER is the proces of labeling data with corresponding categories like person, organization or location. For example:

> Na 14 minuten is het raak! 1-0 voor **Woudrichem(Organization)**.
> Doelpunt **Roy van Sonsbeek(Person)**

As discussed in chapter 2 the standard NER classifiers perform poorly with tweets, therefore I decided to train my own classifier to detect named entities. With help from Rob van der Goot, PhD student at the University of Groningen I created an automatically annotated dataset that I want to use to train a classifier. I do this by using lists of Dutch words for each category (Location, Person, Organization and Miscellaneous) and label those words in downloaded tweets creating a sliver standard.

The purpose of training a classifier and not using a word list for named entity detection is that the word lists contains well known entities like *Ajax* and *Feyenoord* but not the name of the local football club and those frequent occurring local entities I want to recognize. The detailed explanation of training the classifier can be found in chapter 4.

3.3 Evaluation

For evaluating event detection we annotated both datasets by giving all *eventCandidates* a label. We created a gold standard by removing all *eventCandidates* we did not agree on. With this gold standard we can calculate the precision, recall and F-score of our system for each category and overal accuracy (Manning *et al.* 2008). For the named entity recognition we can also calculate the precision, recall and f-score for each category. For the event detection we use the largest category as baseline which is the no-event category(40%) and for upper-bound we use the annotation accuracy (86%).

4 IMPLEMENTATION AND RESULTS

For our experiment I worked together with David de Kleer in collecting the data and building the Python modules. The goal was to build a system that can detect events that take place at a specific location, for example concerts, football matches, fires and traffic jams and recognize the named entities in those events. Events that take place on a larger scale like elections and extreme weather conditions are ignored. The input for this system are tweets that have coordinates that we cluster by location and time. These clusters of tweets are potential events (*eventCandidates*) that the system classifies using the following categories:

- **Other (OTH)** All events that are other then the following categories
- **Meeting (MEE)** All events that are meetings or conferences
- **Entertainment (ENT)** All events that have to do with concerts, movies or theater
- **No Event (NOE)** No Event
- **Incident (INC)** Events that have to do with an incident
- **Sport (SPO)** All events that have to do with sport

4.1 System architecture

Python was used as the programming language for this system because of the excellent tools available to build a machine learning system. NLTK and Scikit learn are used for classifying *eventCandidates*. GitHub[11] was used as file repository. The system consists of five different modules each containing multiple scripts. In this section the main properties of the modules are described. In the following sections the modules are explained in detail.

EVENTCANDIDATES The module that generates the *eventCandidates* by clustering the tweets by *geoHash* and timestamp and saves the result as a JSON file.

ANNOTATOR The module that makes it possible for multiple judges to annotate the data created with the EventCandidates module and calculates the inter-annotator agreement.

CLASSIFIERCREATOR The module that creates/trains the classifier(s) using the annotated *eventCandidates.*

EVENTDETECTIVE The module that uses the trained classifier to classify eventCandidates and generate markers for the Google map.

NER The module that uses the NER classifier that is explained in the Named Entity Recognition section of this chapter.

11 http://www.gitHub.com/chrispool/thesis

4.2 Creating Event Candidates

For clustering the tweets and preparing them for annotation and classification we use the EventCandidates module. This module processes all tweets from a given dataset and generates the *eventCandidates* using four scripts. TweetPreprocessor.py, ClusterCreator.py, ClusterMerger.py and EventCandidates.py

TWEETPREPROCESSOR.PY reads the text file with tweets and tokenizes the tweets and removes the frequent occurring words from the tokens, convert the time string to a Unix timestamp and converts the latitude en longitude into a geoHash. The output of this script is a list of dictionaries where each dictionary is a tweet with the structure as found in code example 2.

```
{'text': '@MrFrankLegs Daar gaan we. Een vicieuze cirkel ??',
'lat': 52.964285,
'geoHash': 'u1kjqcc',
'localTime': '2015-04-24 10:54:16',
'user': 'sandersierdsma',
'tokens': ['@mrfranklegs', 'gaan', 'vicieuze', 'cirkel'],
'lon': 5.923195,
'unixTime': 1429865656}
```

Python code 2: Tweet dictionary

CLUSTERCREATOR.PY iterates over all tweet dictionaries and adds the tweet dictionary to a list in the new two-dimensional dictionary as seen in code example 3. With as first key the geoHash and the second key the Unix timestamp of the last added tweet.

```
clusters['u15y07t']['1429604082'] = [...list of tweet dictionaries...]
```

Python code 3: Result dictionary

With this two-dimensional dictionary we can check for each tweet if we can add it to an existing *eventCandidate* or that we have to create a new *eventCandidate*.

We do this by iterating over all tweet and try to append the tweet to the dictionary described in code example 3. We first look if the geoHash of the tweet is present in the dictionary, if not a new entry is added with the geoHash as key. If the geoHash is present we look if there is a timestamp as key that is within 60 minutes of the tweet, if so we append it to this cluster, if not we create a new cluster(*eventCandidate*)

CLUSTERMERGER.PY checks if there are clusters that overlap in time and subject and are neighboring areas. For example the situation in figure 2, the tweets (red dots) are all about the same event and published in the same period but they are in two different areas.

The ClusterMerger.py script tries to merge situations as described. We do this by iterating over the *eventCandidates* and calculate all neighbors of the current *eventCandidate*. We then iterate over all neighboring areas and if an *eventCandidate* is found we calculate the word overlap (code example 4) to see if the *eventCandidates* refer to the same event. If that is the case we merge the *eventCandidates* and continue the loop until all neighboring areas are checked.

During the development of our system we discussed the number of iterations. If you iterate only once you will merge only the neighbors next to

Figure 4: Geohash border case

you. You can do this more than once to merge the neighbors of the neighbors etc. We chose to use only one iteration because we want local events that take place at a small area.

We calculate the word overlap to see if the two *eventCandidates* refer to the same event. The function *calculteWordOverlap*(code example 5) does that by looking at the 10 most common word in both clusters. For each word that overlaps a score of one time the tf-idf[12] value is awarded. If a hashtag or username overlaps the score is tf-idf * 2. If the total score exceeds the threshold the function returns true.

```
1  def _calculateWordOverlap(self,clusterA, clusterB):
2      wordsClusterA = self._getImportantWords(10, clusterA)
3      wordsClusterB = self._getImportantWords(10, clusterB)
4      result = {}
5
6      #intersect the two lists and adding the scores
7      for wordA, scoreA in wordsClusterA:
8          for wordB, scoreB in wordsClusterB:
9              if wordA == wordB:
10                 result[wordA] = scoreA + scoreB
11                 if wordA[0] == '#':
12                     result[wordA] *= 2
13                 if wordA[0] == '@':
14                     result[wordA] *= 2
15
16     if sum(result.values()) > self.THRESHOLD:
17         return True
18     else:
19         return False
```

Python code 4: Calculate word overlap

EVENTCANDIDATES.PY is the wrapper class for this module. This script generates all *eventCandidates* using the scripts described above. This function requires two parameters. The first is the name of the text file with the unprocessed tweets. The second is the desired name of the dataset. This script saves a JSON file with all *eventCandidates*.

12 term frequency - inverse document frequency, used to reflect how important a word is in a corpus

4.3 Annotation

This module consists of two scripts: Annotator.py and AnnotationEvaluation.py.

ANNOTATOR.PY is an interactive program to annotate the created *eventCandidates* using the Event candidates module. The program shows all tweets in the *eventCandidate* and prompts you to assign a category to them and saves the result in a JSON file. We used Annotator.py for two datasets: *devset*(1250 *eventCandidates*) and *testset*(500 *eventCandidates*). We annotated the *devtest* in the beginning of our research and the *testset* dataset in the final period of our research. The annotation is done with two annotators making it possible to calculate the inter-annotator agreement.

ANNOTATIONEVALUATION.PY is used to calculate the inter-annotator agreement. This is a statistic to determine the quality of annotation and also shows how difficult the task is. We use these statistics also to define our upper-bound. In table 2 the results of the annotation can be found. A kappa score of 0.79 and 0.8 is regarded as good (Manning *et al.* 2008).

Table 2: Annotation result

	Kappa score	Accuracy
devset	0.79	87%
testset	0.80	86%

The confusion matrixes shows the confusion between both judges. Some categories have a higher accuracy because it is easier to detect. For example a lot of *eventCandidates* annotated as a incident have words like *112* or *brandweer* in them where as in the Meeting category it was sometimes unclear if it was an event.

Table 3: Confusion matrix development dataset annotation

	OTH	**MEE**	**ENT**	**NOE**	**INC**	**SPO**
OTH	<1>	.	.	2	1	1
MEE	21	<207>	7	25	.	2
ENT	3	.	<20>	5	.	.
NOE	14	27	18	<619>	9	9
INC	1	2	.	12	<178>	.
SPO	1	.	.	5	.	<60>

Table 4: Confusion matrix test dataset annotation

	OTH	**MEE**	**ENT**	**NOE**	**INC**	**SPO**
OTH	<.>	.	.	.	1	.
MEE	4	<110>	13	9	.	3
ENT	.	.	<8>	2	.	.
NOE	3	19	4	<199>	4	2
INC	1	.	.	.	<78>	.
SPO	.	1	2	2	.	<8>

We used the *eventCandidates* we agreed on to create our gold standard. Resulting in 1085 *eventCandidates* in the *devset* and 403 *eventCandidates* in the *testset*.

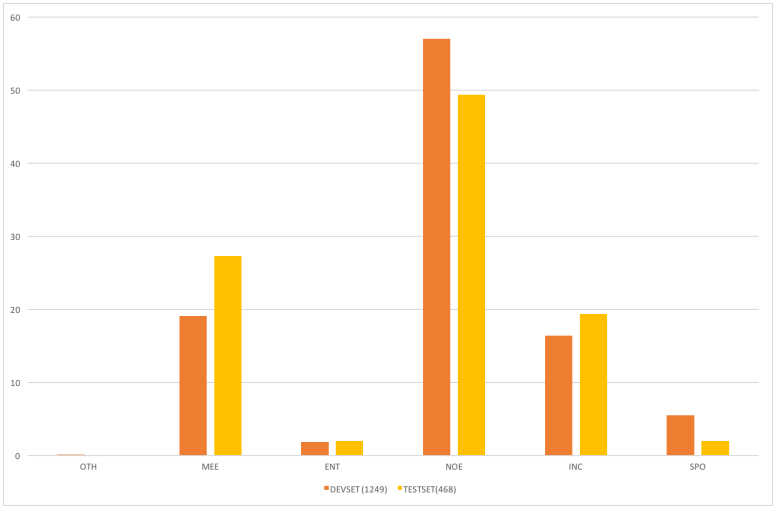

Figure 5: Division dataset

Figure 5 shows the percentage of the corresponding dataset.

4.4 Training classifiers

The ClassifierCreator module has two scripts: classifierCreator.py and featureSelector.py. With the annotation and the *eventCandidates* ready we started with training the classifiers. The module can be used in normal mode and in test mode. The first mode lets you select one dataset and the script divides the dataset in a train and test set (80%/20%). In test mode you can select two different datasets, one for training and the other for testing. The test mode was used for calculating the final statistics about our classifier.

Using the annotation and the *eventCandidates* we start by preparing the training data. We have chosen to use two classifiers, one for classifying using words as features (*categoryClassifier*). And a second classifier using metadata as features (*eventClassifier*). The output of the *categoryClassifer* is used as a feature in the eventClassifier.

We created featureSelector.py as a helper function that generates the features in the format required by NLTK. We designed this in a very modular way so we could experiment with using different features and combinations of features.

This function, as seen in code example 5, requires a list of the desired features and returns the features in a format suitable for training the classifier for a given cluster(*eventCandidate*).

The *categoryClassifier* uses the N most frequent words as features. In order to calculate this feature the system needs to know all used words in the *eventCandidates*, this is done while initializing the features module.

```
1 def getFeatures(self, candidate, features):
2         returnFeatures = {}
3         for feature in features:
```

```
            if feature in self.featureTypes:
                method = getattr(self, "_" + feature)
                if feature == 'wordFeatures':
                    wordFeatures = method(candidate)
                    #add word features to dictionary to be able to combine features
                    for key in wordFeatures:
                        returnFeatures[key] = wordFeatures[key]
                else:
                    returnFeatures[feature] = method(candidate)
            else:
                print("The feature", feature, "is not available.")

        return returnFeatures
```

Python code 5: Selecting features

With the features in the correct format we can train the classifiers using NLTK, that we also use to calculate the performance of the classifier which can be found in the result section.

```
self.classifierBFeatures =  ['category', 'location','wordOverlapSimple','wordOverlapUser']
for candidate, label in self.testData:
            featuresB = self.featureSelector.getFeatures(candidate, self.classifierBFeatures)
            self.featureKeys = featuresB.keys()
            self.testB.append((featuresB, label))

for candidate, label in self.trainData:
    featuresB = self.featureSelector.getFeatures(candidate, self.classifierBFeatures)
    self.featureKeys = featuresB.keys()
    self.trainB.append((featuresB, label))

self.classifierB = nltk.NaiveBayesClassifier.train(self.trainB)
```

Python code 6: Selecting features

4.4.1 *Feature selection*

The *categoryClassifier* uses the most frequent words as features. The *eventClassifier* uses the output of the first classifier as one of the features. This feature is by far the most valuable for the system. In table 4 the performance of the different features used individual can be found using the *testset*. In appendix 1 the complete results can be found.

Table 5: Effect of features on testset

	Accuracy
All features	0.84
Category	0.81
Location	0.51
WordOverlapSimple	0.63
WordOverlapUser	0.6
WordOverlapUser, Category	0.82

CATEGORY The result of the classifier that uses the most frequent words as features.

LOCATION The first 5 characters in the geoHash are used as a feature. It will be easier to detect events on locations were events often take place.

WORDOVERLAPSIMPLE This feature is a numeric value that represents how many tweets consist of the same words. The score is calculated by counting the occurrences of each type and dividing it by the number of tweets. Hashtags get a bonus score.

WORDOVERLAPUSER This feature calculates the overlap of types among users. The score is the highest when all users use the same words. The result is the log of the score and rounded to 0.5.

4.4.2 *Results event detection*

We are very satisfied with the results of our classifier(s). With an accuracy of 84% it is only a few procent less than our upper-bound of 87% and a lot better then our baseline of 40%. Naive Bayes was the best performing classifier but the differences where not that big. The table with the precision and recall of each category using Naive Bayes can be found in appendix 1 and the confusion matrixes for all classifiers can be found in appendix 2.

Table 6: Results using all features

	Naive Bayes			Maximum Entropy			SVM		
	Accuracy = 84%			Accuracy = 83%			Accuracy = 81%		
	P	R	F	P	R	F	P	R	F
NOE	0.85	0.90	0.88	0.83	0.93	0.88	0.85	0.83	0.84
SPO	0.77	0.49	0.60	0.77	0.49	0.60	0.77	0.49	0.60
ENT	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MEE	0.76	0.79	0.77	0.79	0.74	0.76	0.65	0.81	0.72
INC	0.97	0.97	0.97	0.94	0.94	0.94	1.00	0.97	0.99
OTH	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

To compare our research with other researches we also calculated our f-score for all categories except no-event because most researches do binary event-detection (event/ no-event). If we compare our f-score of 88% with the research of Walther & Kaisser (f-score of 86%) we can see our classifier performs slightly better. A reason could be is that we also use automatically generated tweets of emergency calls that are easy to recognize for our classifier.

4.5 Event Detective

This module uses the trained classifiers to generate an interactive map and adds the results of the Named Entity Recognition classifier described in the next section.

To create an interactive map we have chosen to use the Google map API because of the good documentation and excellent features. We generate a javascript file with all information about the markers we want to plot on the map.

4.6 Named Entity Recognition

As discussed in chapters two and three a classifier was trained for detecting interesting entities within the events. The Stanford Named Entity software was used for this.

4.6.1 *Data collection*

For creating the silver standard I used the approach described in the previous chapter by first selecting lists of Dutch words for each category I want to recognize: locations, organizations, persons and miscellaneous. I use the word lists that also are used in the Alpino software[13] For each category I have a list of Dutch words related to that category. The wordlists are used to retrieve tweets that contain one or more of those words.

I downloaded one million tweets and wrote a Python script to create the training file as can be seen in code example 11 that can be used to train your own classifier. In this example it shows that the training file is not perfect. The name *van Zutphen* is a word in the person wordlist and thus is annotated as a name but in this case *van* should not be annotated and *Zutphen* should be annotated as a Location.

```
1  De   O
2  Sprinter     O
3  van      PERSON
4  Zutphen      PERSON
5  naar     O
6  Arnhem   LOCATION
7  van      O
8  00:06    O
9  vertrekt     O
10 van      O
11 spoor    O
12 1    O
13 Spoorwijziging   O
14 7690     O
```

Python code 7: Training file NER

13 Alpino is a dependency parser for Dutch, developed in the context of the PIONIER Project Algorithms for Linguistic Processing.

4.6.2 *Training*

The training file (as can be seen in code example 7) was used to train the classifier. The documentation about the possible features was not so good and because the time it took to train the classifier (10 -14 hours) I could not experiment that much with different settings. I finally used the features also used for their standard classifier. That resulted in the results described in the next section.

4.6.3 *Results*

The results can be found in table 7. The results show the difficulty of recognizing named entities in tweets and compared to other researches about named entity recognition the performance is not so high. Ashwini & Choi (2014) conducted a research on finding what they call targetable named entities *"named entities in a targeted set (e.g, movies, books, TV shows)"*, and can be compared to my system. Their resulting f-scores are between 76% and 78%.

Table 7: NER Classifier trained with Tweets

Entity	P	R	F1	TP	FP	FN
LOCATION	0,7333	0,7333	0,7333	11	4	4
MISC	0,0909	0,500	0,1538	1	10	1
ORGANIZATION	0,7500	0,8824	0,8108	15	5	2
PERSON	0,6154	0,5517	0,5818	16	10	13
Totals	0,5972	0,6825	0,6370	43	29	20

5 CONCLUSION AND DISCUSSION

We are very pleased with the results of our classifier, compared to other researches our classifier performs very well and can be easily converted to a system that uses real-time tweets. Improvements could be to take a further look at our features and maybe think about tokenizing the tweet better. For example user names like *@chrisPool* are detected by our system as *chris Pool* but more difficult situations also occur like *ArenA* and our system does not process that well.

The named entity classifier results are a bit disappointing but it also shows how difficult the task is to recognize named entities in tweets. Because most classifiers do detection by looking at capitalized letters the classifier often tries to classify the wrong words. Improvements could also be to think about the automatic annotation of the training file. If a word occurs in more than one category list the system should use the most logical one. For example the case described in 4.7.1 data collection the location should have been the choice instead of person.

Looking back at my research questions I can say that is possible to detect and categorize local events in the Twitter stream using tweets with geo-information. Further analysis has to be done to find the optimal amount and types of categories. Our system would make a nice website where people can see events in their neighborhood and filter categories they are interested in. Detecting the important entities is difficult, tweets are too short and noisy to detect entities. Improvements could be made by thinking about the processing of the tokens and to use the annotation by the user (*Hashtag, emoticon*) more efficient.

6 APPENDIX 1

	accuracy	geen_event	sport	entertainment	bijeenkomst	incident	anders
		P R F	P R F	P R F	P R F	P R F	P R F
All features	0.84	0.85 0.90 0.88	0.77 0.49 0.60	0.00 0.00 0.00	0.76 0.79 0.77	0.97 0.97 0.97	0.00 0.00 0.00
Category	0.81	0.85 0.82 0.84	0.73 0.46 0.56	0.00 0.00 0.00	0.66 0.84 0.74	0.99 0.97 0.98	0.00 0.00 0.00
Location	0.51	0.49 0.94 0.65	0.00 0.00 0.00	0.00 0.00 0.00	0.66 0.25 0.36	0.50 0.06 0.11	0.00 0.00 0.00
WordOverlapSimple	0.63	0.60 0.85 0.71	0.00 0.00 0.00	0.00 0.00 0.00	0.57 0.33 0.42	0.77 0.83 0.80	0.00 0.00 0.00
wordOverlapUser	0.6	0.57 0.93 0.71	0.00 0.00 0.00	0.00 0.00 0.00	0.00 0.00 0.00	0.66 0.90 0.76	0.00 0.00 0.00
wordOverlapUser, category	0.82	0.86 0.83 0.85	0.77 0.49 0.60	0.00 0.00 0.00	0.66 0.85 0.74	1.00 0.97 0.99	0.00 0.00 0.00

7 APPENDIX 2

Table 8: Confusion Matrix Naive Bayes

	OTH	MEE	ENT	NOE	INC	SPO
OTH	<.>	.	.	1.	.	.
MEE	.	<82>	2	14	.	12
ENT	.	.	<.>	1	.	.
NOE	.	27	5	<179>	1	7
INC	.	.	.	2	<76>	.
SPO	.	1	1	2	1	<16>

Table 9: Confusion Matrix Maximum Entropy

	OTH	MEE	ENT	NOE	INC	SPO
OTH	<.>	1	.	.	.	.
MEE	.	<81>	2	7	2	11
ENT	.	.	<.>	.	1	.
NOE	.	26	5	<186>	1	7
INC	.	1	.	4	<73>	.
SPO	.	1	1	2	1	<16>

Table 10: Confusion Matrix SVM classifier

	OTH	MEE	ENT	NOE	INC	SPO
OTH	<.>	1	.	.	.	.
MEE	.	<89>	2	31		14
ENT	.	.	<.>	.		.
NOE	.	19	5	<166>	1	4
INC	.	.	.	.	<73>	.
SPO	.	1	1	2	1	<17>

REFERENCES

Alan Ritter, Sam Clark, Mausam, & Etzioni, Oren. 2011. Named Entity Recognition in Tweets: An Experimental Study. *Pages 1524–1534 of: Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Computer Science and Engineering, University of Washington.

Allan, James, Carbonell, Jaime G, Doddington, George, Yamron, Jonathan, & Yang, Yiming. 1998. Topic detection and tracking pilot study final report.

Ashwini, Sandeep, & Choi, Jinho D. 2014. Targetable Named Entity Recognition in Social Media. *arXiv preprint arXiv:1408.0782*.

Birant, Derya, & Kut, Alp. 2007. ST-DBSCAN: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, **60**(1), 208–221.

Dawson, R. 2012. *Which countries have the most Twitter users per capita?* [Online; accessed 3-June-2015].

Han, Jiawei, Kamber, Micheline, & Pei, Jian. 2006. *Data mining, southeast asia edition: Concepts and techniques*. Morgan kaufmann.

Liu, X., Zhang, S., Wei, F., & Zhou, M. 2011. Recognizing named entities in tweets. *Pages 359–367 of: In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Computer Science and Engineering, University of Washington.

Manning, Christopher D, Raghavan, Prabhakar, & Schütze, Hinrich. 2008. *Introduction to information retrieval*. Vol. 1. Cambridge university press Cambridge.

Ríos, Miguel. 2014. *The geography of Tweets*. [Online; accessed 7-June-2015].

Sakaki, Takeshi, Okazaki, Makoto, & Matsuo, Yutaka. 2010. Earthquake shakes Twitter users: real-time event detection by social sensors. *Pages 851–860 of: Proceedings of the 19th international conference on World wide web*. ACM.

Statista. 2015. *Number of monthly active Twitter users worldwide from 1st quarter 2010 to 1st quarter 2015 (in millions)*. [Online; accessed 3-June-2015].

Turpijn, Loes, Kneefel, Samantha, & van der Veer, Neil. 2013. *Nationale Social Media Onderzoek 2015*. Newcom Research and Consultancy B.V.

Walther, Maximilian, & Kaisser, Michael. 2013. Geo-spatial event detection in the twitter stream. *Pages 356–367 of: Advances in Information Retrieval*. Springer.