Detecting local events in the Twitter stream

by

Chris Pool

Bachelor thesis
June 2, 2015

DETECTING LOCAL EVENTS IN THE TWITTER STREAM

CHRIS POOL

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 and the enormous amount of data available makes it interesting to use this data as a source of information about events that are taking place at a certain time and location. The goal of my research is to develop a system using tweets with geo-information 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 first step is collecting the data. I used the tweets available on the Karora server[1]. This server stores all Dutch tweets and can be easily accessed. I downloaded all tweets with geo-information from 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 these tweets, these clusters are potential *event candidates*. We create these clusters by location and time. To make these *event candidates* a 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.

The next step is to annotate the data, I and David de Kleer annotated the same datasets making it possible to calculate the inter-annotater agreement. With this annotated data we trained a Naive Bayes classifier. 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 *event candidates* using the categories: No event, Meeting, Entertainment, Incident, Sport and Other I try 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 detected 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 and the enormous amount of data available makes it interesting to use this data as a source of information about events that are taking place at a certain time and location. 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 and can be very informative, for example to provide news items with more information about the opinion of people.

This thesis describes a method of finding local events based on tweets with geo-information and categorizes them into 5 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. Local events are events that take place in a certain radius (Z) within a certain time interval (X). For this research $z = 0.5km, x = 2hours$.

My research question is: Is it possible to detect local events in the Twitter stream that take place in the Netherlands based on Tweets with geo-information, to see if you can detect what type of events take place and who the important people/organizations or topics are to make an interactive map of these events.

This thesis is devided in four chapters. The first chapter, related literature, describes the research that is done about this subject. The second chapter is the method section where I describe the methods I use for my experiment. In the third chapter i describe my experiment, all code I am referring to can also be found in my 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

Since the launch of Twitter in 2006 the popularity of this micro-blogging service is increasing rapidly (Statista 2015). With 302 million users sending about 500 million tweets per day this is a huge source of information. In the Netherlands the number of active users was 2.8 million in 2015 according to the research conducted by (Turpijn *et al.* 2013). The Netherlands still one of the countries with the highest twitter accounts to population ratio (Dawson 2012).

Using geo-information from tweets has been done in several researches, for example Sakaki *et al.* used geo-information of tweets to detect earthquakes by analyzing all tweets that contain certain keywords like *earthquake* or *shaking*. In literature this method of event detection is often referred to as spatio-temporal event detection. In the research by Walther & Kaisser (2013) they describe the process of detecting events in the Twitter stream, this research was the inspiration of my research and builds on several techniques they discuss.

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 the Stanford NER, a state of the art classifier, focusses to much on capital letters making it perform poorly with tweets.

3 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 my system and the motivation why I chose these methods.

3.1 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. I 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[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. If two locations share the same prefix it indicates (not always) that the locations are nearby.

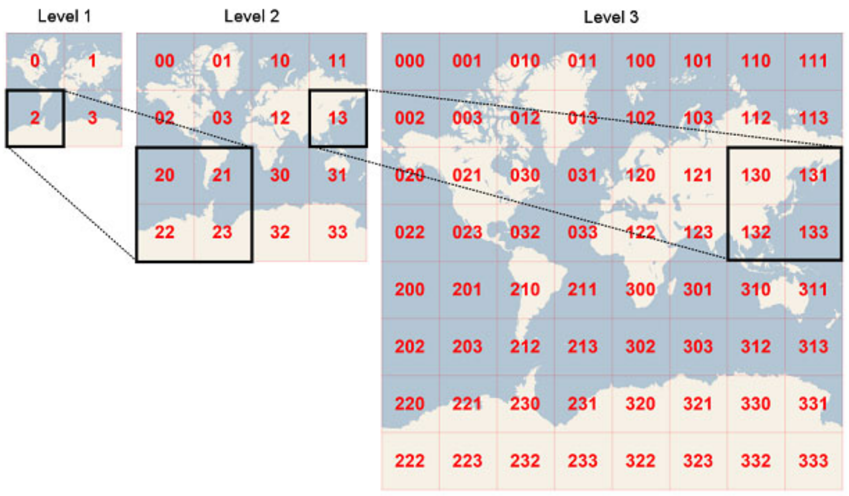


Figure 1: Geohash

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 show 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.

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

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. I used the GeoHash library written in 2009 by Hiroaki Kawai[6] 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.

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

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

Python code 1: Example of geoHash encoding

The library also provides functions to calculate the neighboring geoHashes. This is useful because in a grid clustering approach border cases can occur. This functionality can be used to merge those border cases 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 build-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.

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

3.2 Classification

After forming the potential events (*eventCandidates*) the Event Candidates have to be classified as a specific event category or not an event. NLTK provides excellent documentation and functions for classifying.
NLTK enables you to use several classifiers:

- ConditionalExponentialClassifier
- DecisionTreeClassifier
- MaxentClassifier
- NaiveBayesClassifier
- WekaClassifier
- SVMClassifier (using scikit-learn)

In my experiment I 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.

3.3 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. In the next chapter this process is explained in detail.

4 EXPERIMENT AND EVALUATION

For my 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
- **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[7] 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.

ANNOTATER 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.

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

4.2 Data collection

For the experiment tweets are used that have geo-information. Using a simple Grep[8] command it was possible to retrieve only tweets with this information from the Linux server *Karora* [9] 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

> @jokiecroky heb ze terug gebracht naar de plek waar ik ze heb opgehaald... volgende dag werd ik gebeld dat ze samen een mooi huisje hadden 5.106646 52.06455 TonyJuniorLive 2015-04-15 00:01:37 CEST Wed

4.3 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

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

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.

```python
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*. The code in code example 4 shows the clustering of the tweets.

```python

#setting used for this experiment
self.MINUTES = 60

def __createClusters(self):
        for tweet in self.tweetDicts:
            geoHash = tweet["geoHash"]
            tweetTime = tweet["unixTime"]
            foundTime = tweetTime

            if geoHash in self.clusters:
                for times in self.clusters[geoHash].keys():
                    if times <= tweetTime <= times + self.MINUTES * 60:
                        foundTime = times

            self.clusters[geoHash][foundTime].append(tweet)
            if tweetTime != foundTime:
                # use new timestamp as key to keep event active
                self.clusters[geoHash][tweetTime] = self.clusters[geoHash][foundTime]
                # Remove old key
                del self.clusters[geoHash][foundTime]
```

Python code 4: Cluster tweets

The output of this script is a dictionary with all clusters with the structure described in code example 3.

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.

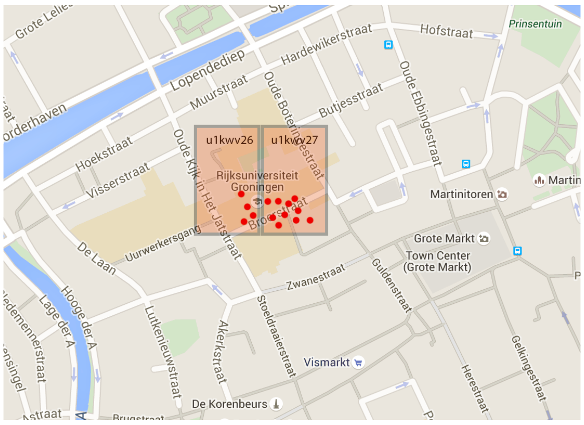


Figure 2: Geohash border case

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, if an *eventCandidate* is found we calculate the word overlap (code example 5) to see if the *eventCandidates* is the same event. If that is the case we merge the cluster and continue the loop till all neighboring areas are checked.

We decided to iterate only once over all *eventCandidates* because otherwise the area will be to big.

We calculate the word overlap to see if the two *eventCandidates* have the same topic. 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 point is awarded. If a hashtag or username overlaps two points are awarded. If a score exceeds the threshold the function returns true.

```python
def _calculateWordOverlap(self,clusterA, clusterB):
    wordsClusterA = self._getImportantWords(10, clusterA)
    wordsClusterB = self._getImportantWords(10, clusterB)
    result = {}

    #intersect the two lists and adding the scores
    for wordA, scoreA in wordsClusterA:
        for wordB, scoreB in wordsClusterB:
            if wordA == wordB:
                result[wordA] = scoreA + scoreB
                if wordA[0] == '#':
                    result[wordA] *= 2
                if wordA[0] == '@':
                    result[wordA] *= 2

    if sum(result.values()) > self.THRESHOLD:
        return True
    else:
        return False
```

Python code 5: 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*.

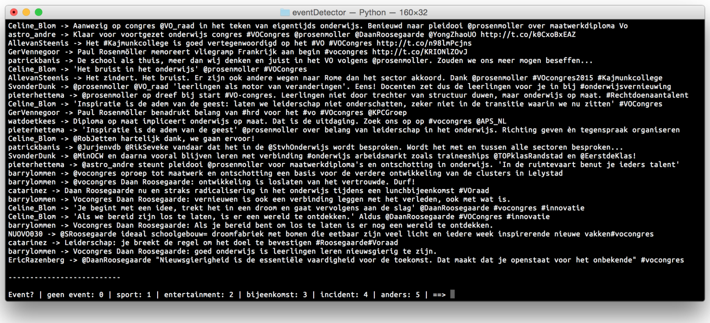
4.4 Annotation

This tool was used to annotate the two datasets: *devset* and *testset*. We annotated the *devtest* in the beginning of our research, the *testset* dataset in the final period of our research. The datasets where annotated with an interactive tool by two judges (figure 3). This tool also calculates the kappa score that can be used to determine the quality of annotation. The results of the annotation can be found in table 2.

Table 2: Annotation result

	Kappa score	Accuracy
devset	0.79	87%
testset	0.80	86%

The Kappa score indicates a good annotation. We regard the accuracy of 86% as our upperbound.



```
Celine_Blom -> Aanwezig op congres @VO_raad in het teken van eigentijds onderwijs. Benieuwd naar pleidooi @prosenmoller over maatwerkdiploma Vo
astro_andre -> Klaar voor voortgezet onderwijs congres #VOCongres @prosenmoller @DaanRoosegaarde @YongZhaoUO http://t.co/k0CxoBxEAZ
AllevanSteenis -> Het #Kajmunkcollege is goed vertegenwoordigd op het #VO #VOCongres http://t.co/n98lmPcjns
GerVennegoor -> Paul Rosenmöller memoreert vliegramp Frankrijk aan begin #vocongres http://t.co/KRIONlZOvJ
patrickbanis -> De school als thuis, meer dan wij denken en juist in het VO volgens @prosenmoller. Zouden we ons meer mogen beseffen...
Celine_Blom -> 'Het bruist in het onderwijs' @prosenmoller #VOCongres
AllevanSteenis -> Het zindert. Het bruist. Er zijn ook andere wegen naar Rome dan het sector akkoord. Dank @prosenmoller #VOcongres2015 #Kajmunkcollege
SvonderDunk -> @prosenmoller @VO_raad 'leerlingen als motor van veranderingen'. Eens! Docenten zet dus de leerlingen voor je in bij #onderwijsvernieuwing
pieterhettema -> @prosenmoller op dreef bij start #VO-congres. Leerlingen niet door trechter van structuur duwen, maar onderwijs op maat. #Rechtdoenaantalent
Celine_Blom -> 'Inspiratie is de adem van de geest: laten we leiderschap niet onderschatten, zeker niet in de transitie waarin we nu zitten' #VOCongres
GerVennegoor -> Paul Rosenmöller benadrukt belang van #hrd voor het #vo #VOcongres @KPCGroep
watdoetkees -> Diploma op maat impliceert onderwijs op maat. Dat is de uitdaging. Zoek ons op op #vocongres @APS_NL
pieterhettema -> 'Inspiratie is de adem van de geest' @prosenmoller over belang van leiderschap in het onderwijs. Richting geven èn tegenspraak organiseren
Celine_Blom -> @RobJetten hartelijk dank, we gaan ervoor!
patrickbanis -> @Jurjenvdb @RikSeveke vandaar dat het in de @StvhOnderwijs wordt besproken. Wordt het met en tussen alle sectoren besproken...
SvonderDunk -> @MinOCW en daarna vooral blijven leren met verbinding #onderwijs arbeidsmarkt zoals traineeships @TOPklasRandstad en @EerstdeKlas!
pieterhettema -> @astro_andre steunt pleidooi @prosenmoller voor maatwerkdiploma's en ontschotting in onderwijs. 'In de ruimtevaart benut je ieders talent'
barrylommen -> @vocongres oproep tot maatwerk en ontschotting een basis voor de verdere ontwikkeling van de clusters in Lelystad
barrylommen -> @vocongres Daan Roosegaarde: ontwikkeling is loslaten van het vertrouwde. Durf!
catarinez -> Daan Roosegaarde nu en straks radicalisering in het onderwijs tijdens een lunchbijeenkomst #VOraad
barrylommen -> Vocongres Daan Roosegaarde: vernieuwen is ook een verbinding leggen met het verleden, ook met wat is.
Celine_Blom -> 'Je begint met een idee, trekt het in een droom en gaat vervolgens aan de slag' @DaanRoosegaarde #vocongres #innovatie
Celine_Blom -> 'Als we bereid zijn los te laten, is er een wereld te ontdekken.' Aldus @DaanRoosegaarde #VOCongres #innovatie
barrylommen -> Vocongres Daan Roosegaarde: Als je bereid bent om los te laten is er nog een wereld te ontdekken.
NUOVO030 -> @SRoosegaarde ideaal schoolgebouw= droomfabriek met bomen die eetbaar zijn veel licht en iedere week inspirerende nieuwe vakken#vocongres
catarinez -> Leiderschap: je breekt de regel om het doel te bevestigen #Roosegaarde#Voraad
barrylommen -> Vocongres Daan Roosegaarde: goed onderwijs is leerlingen leren nieuwsgierig te zijn.
EricRazenberg -> @DaanRoosegaarde "Nieuwsgierigheid is de essentiële vaardigheid voor de toekomst. Dat maakt dat je openstaat voor het onbekende" #vocongres

--------------------------

Event? | geen event: 0 | sport: 1 | entertainment: 2 | bijeenkomst: 3 | incident: 4 | anders: 5 | ==> 
```

Figure 3: Annotation tool

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>

For the *devset* we annotated 500 event candidates with these categories. In 86% of the cases the judges agreed.

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	.	<60>

4.5 Training classifiers

With the annotation and the *eventCandidates* ready we started with training the classifiers using the script ClassifierCreator.py. This 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 80/20 train / test set. The test mode lets you 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 module that generates the features in the format required by NLTK. We designed this very modular so we could experiment with using different features and combinations of features.

This function, as seen in code example 6, 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.

```python
def getFeatures(self, candidate, features):
        returnFeatures = {}
        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 6: Selecting features

With the features in the correct format we can train the classifiers. The code in code example 7 shows that adjusting the features is easy and allows us to experiment with different settings. The first for loop iterates over all test data and makes a tuple with two values from each *eventCandidate* with at index 0 the features and at index 1 the label(category). It does the same for the train data. Training the classifier using NLTK can be done by using the train method that is available in the NLTK library. The classifierCreator module finally creates the statistics of the classifier that are used in the result section.

```python
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 7: Selecting features

4.5.1 *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 can be found. In appendix x the complete results can be found.

Table 5: Effect of features

	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.

```python
def _wordOverlapSimple(self, candidate):
        types = Counter()
        for tweet in candidate:
            types.update(set(tweet['tokens']))
        score = 0
        for t in types:
            if types[t] > 1:
                if t[0] == '#':
                    score += (types[t] * 2)
                else:
                    score += types[t]

        return round((score / len(candidate)) * 2) / 2
```

Python code 8: Word overlap feature

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.

```python
def _wordOverlapUser(self, candidate):
        '''Calculate the overlap of features among users'''
        userTypes = defaultdict(list)
        types = Counter()

        for row in candidate:
            userTypes[row['user']].extend(row['tokens'])

        for user in userTypes:
            types.update(set(userTypes[user]))
        score = 0
        for t in types:
            if types[t] > 1: #ignore if only in one tweet
                if t[0] == '#':
                    score += (types[t] * 2)
                else:
                    score += types[t]

        if score > 1:
            s = log(float(score) * float(len(userTypes.keys()) ))
            #return round((score * 2) / len(candidate))
            return round((s / len(candidate) )* 2 ) /2
        else:
            return 0.0
```

Python code 9: User word overlap feature

4.5.2 *Results event detection*

The event classifier performs good. It is a few procent less then the Upper bound of 87%.

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

The different classifiers perform almost the same. This is mainly because the category feature is important for the results. If we don't use this feature the differences between the classifiers is bigger.

Table 7: 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 8: 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 9: 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>

4.6 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. It first asks you to select a classifier you want to use. The second step is to select a dataset you want to classify.

To create an interactive map we have chosen to use Google map 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.

```python
def generateMarkers(self):
        print("Creating Google Maps markers & add WIKI links...")

        js = open('vis/map/js/markers.js','w')
        js.write('var locations = [')


        for tweets,label in self.events:
            writableCluster = ''
            gh = []
            i = 0
            avgLon = 0
            avgLat = 0
            #tweets = sorted(tweets, key=itemgetter('unixTime'));

            for tweet in tweets:
                i = i + 1
                gh.append(tweet['geoHash'])
                avgLon += float(tweet["lon"])
                avgLat += float(tweet["lat"])
                # backslashes voor multiline strings in Javascript
                writableCluster += "{} {} {} {}<br/><br/>".format(tweet['localTime'], tweet['geoHas
            # Bepaal het Cartesiaans (normale) gemiddelde van de coordinaten, de afwijking (door vo
            # van de aarde) zal waarschijnlijk niet groot zijn omdat het gaat om een klein vlak op
            # Oftewel, we doen even alsof de aarde plat is ;-)
            avgLon /= i
            avgLat /= i
            #writableCluster += "</br>" + str(ngrams).replace("'", "\\'")
            js.write("['{}', {}, {}, '{}'],".format(writableCluster,avgLat,avgLon,label))
        js.write('];')
        js.close()
```

Python code 10: Generate Javascript file

4.7 Named Entity Recognition

The default NER tagger in NLTK is the Stanford NER classifier. This is according to many researches the best available. Unfortunately there are no classifiers trained using Dutch tweets and the performance of the available classifiers was so poor I decided to train my own classifier. Because of limited time I decided to automatically annotate the training data creating a silver standard.

4.7.1 *Data collection*

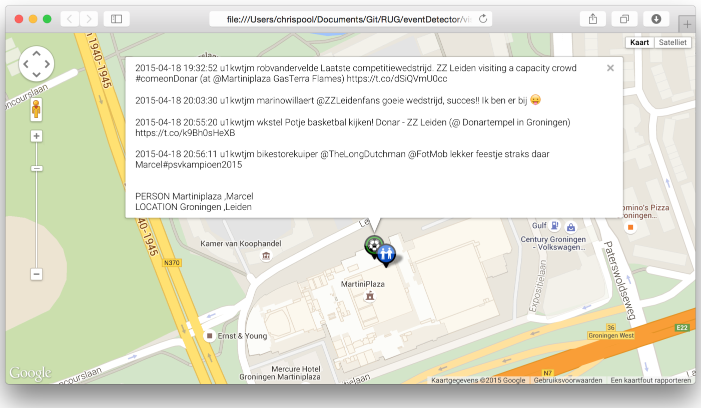
For creating the silver standard I used the approach described in xxx. by first selecting lists of Dutch words for each category I want to recognize (LOCATIONS, PERSONS, ORGANIZATIONS and MISC). I used the word lists that also are used in the Alpino software[10] The wordlists are used to retrieve tweets that contain one or more of those words from the Karora machine. I downloaded 1 million tweets and wrote a Python script to create the training file that can be used to train your own classifier.

4.7.2 *Training*

The training file was used to train the classifier and by experimenting with the features. I also annotated about 100 tweets manually to calculate the precision and recall. The results can be found in table 12.

4.7.3 *Results*

There results are not great, compared to other researches .... the classifier misses a lot of entities.


file:///Users/chrispool/Documents/Git/RUG/eventDetector/vi
Kaart
Satelliet
Kamer van Koophandel
MartiniPlaza
Ernst & Young
Mercure Hotel Groningen Martiniplaza
Gulf
Century Groningen - Volkswagen...
Patersswoldseweg
Expositielaan
N370
Groningen West
Google

2015-04-18 19:32:52 u1kwtjm robvandervelde Laatste competitiewedstrijd. ZZ Leiden visiting a capacity crowd #comeonDonar (at @Martiniplaza GasTerra Flames) https://t.co/dSiQVmU0cc

2015-04-18 20:03:30 u1kwtjm marinowillaert @ZZLeidenfans goeie wedstrijd, succes!! Ik ben er bij 😛

2015-04-18 20:55:20 u1kwtjm wkstel Potje basketbal kijken! Donar - ZZ Leiden (@ Donartempel in Groningen) https://t.co/k9Bh0sHeXB

2015-04-18 20:56:11 u1kwtjm bikestorekuiper @TheLongDutchman @FotMob lekker feestje straks daar Marcel#psvkampioen2015

PERSON Martiniplaza ,Marcel
LOCATION Groningen ,Leiden

Figure 4: NER example

Table 10: 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

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

5 CONCLUSION AND DISCUSSION

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

6 APPENDIX

	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

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.

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

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.

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.