DETECTING LOCAL EVENTS IN THE TWITTER STREAM

CHRIS POOL

Figure 1: GUI

CONTENTS

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

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 as a source of information about events that are taking place at a certain time and location. This real time information and with the addition about how people feel about these events can be very useful, for example to provide news items with more information about user sentiment.

This thesis describes a method of finding local events based on tweets with geo information and categorize them into 5 categories (No-event, Sport, Enterainment, Meeting, Incident) and to find relevant information within those tweets about the location and people involved. Local events are events that take place in a certain radius (Z) within a certain time interval (X). For this research $z = 0.5km, x = 5hours$.

My research question is: Is it possible to detect local events from 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.

You can read in the following chapters how the research was conducted

2 RELATED LITERATURE

Walter and Kaisser[1] describe in their paper Geo-spatial event detection in the Twitter stream a way to recognize geo-spatial events. They focus on small events that take place in a small area like fires and traffic jams. It differs from my research because I want to focus on locations where events of different sizes can take place but the features they use can be useful for my research. Especially how they calculate the clusterscore (cluster of tweets from a region) can be useful. Unfortunately they do not describe how they make clusters of locations.

Amineh Amini et al.: Density-Based Data Streams Clustering Survey[2] is a paper about different clustering techniques and is interesting to see which ones I can use to make clusters of the locations and the events. DBSCAN is one of the density-based clustering algorithms I maybe can use. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is developed for clustering large spatial databases with noise, based on connected regions with high density. The density of each point is defined based on the number of points close to that particular point called the point's neighborhood. The dense neighborhood is defined based on two user-specified parameters: the radius (of the neighborhood, and the number of the objects in the neighborhood

3 METHOD

In this chapter the used methods for the experiment are described.

3.1 Clustering

The first task is to cluster the tweets based on their location and time. There are several approaches in location clustering that can be used. Because the aim of the research was to make something that could work in real-time it was necessary to look for an approach that was quick. GeoHash is is a system invented by Gustavo Niemeyer that converts latitude and longitude coordinates into a hash. The more digits used means a higher accuracy. If two locations share the same prefix indicates (not always) that the locations are nearby. This algorithm is quick and easy to use and ideal for my experiment.

Figure 2 illustrates that 1 digit can be used to represent a quarter of the world, two digits 1/16 and three digits 1/64 etc.

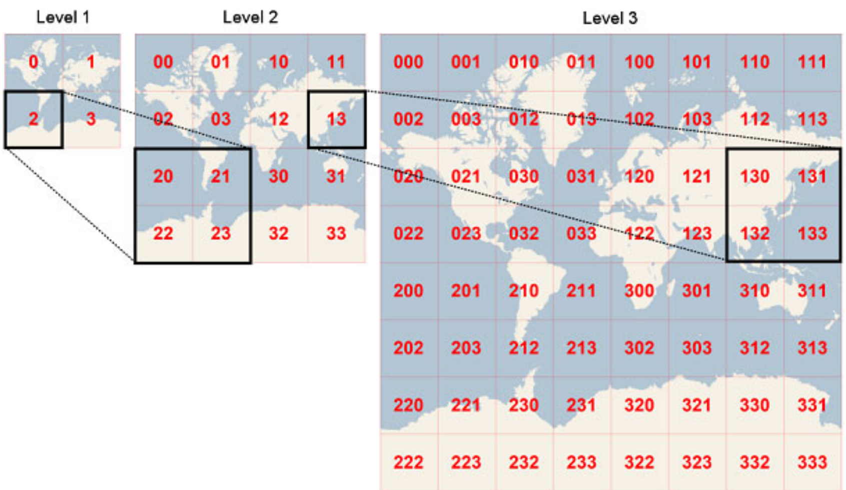

Figure 2: Geohash

There are several implementations of this algorithm in Python. I used the GeoHash library written in 2009 by Hiroaki Kawai 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. The library also provides functions to calculate the neighboring GeoHashes, this is useful because in a grid clustering approach border cases can occur and with this function neighboring GeoHashes can be explored to see if it contains tweets with the same topic and can merge them if necessary.

3.2 NLTK

Python has a lot of functions to use while doing NLP tasks but for more advanced tasks it is not powerful enough. Therefore we have chosen the Natural Language Tookit (NLTK) NLTK has a lot of modules that can be used for a wide range of NLP tasks. It is also really easy to use and well documented.

3.3 Classification

After forming the potential events (Event Candidates) 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 provides several classifier types:

- ConditionalExponentialClassifier
- DecisionTreeClassifier
- MaxentClassifier
- NaiveBayesClassifier
- WekaClassifier

In the next chapter I describe the results using the different classifiers. Some classifiers where not used because they are used together with other software.

3.4 Named Entity Reconinigzion

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

> Klaar voor Ajax (organization) - NAC(organization) @Amsterdam(location) @ArenAPoort(location) in Amsterdam(location) Zuidoost, Noord-Holland(location)

Current NER tools were designed to process large texts and perform badly with tweets due to the noisy and short style. One of the most used NER tools is Stanford NER. This Java implementation is also known as CRFClassifier. The software comes with 3 trained classifiers that perform particularly well for the three classes persons, organizations and locations. Some experiments are done with using labeled tweets as traindata but mostly in English.

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. 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 afterwards try to cluster based on proximity, time and topic. These clusters of tweets are potential events that are system should classify as event/no event. The final goal was to plot these events on a map.

4.1 System architecture

Python was uded for this system because of the excellent tools available to build a machine learning system. NLTK and Scikit learn are used for classifying the event candidates.

We designed several classes for building our system but there are 3 main modules:

EVENTCANDIDATES The module that processes the tweets by giving each tweet a GeoHash, tokenizes the tweet and passes it through to the clusterCreator module that clusters the tweets using the geoHash and the timestamp resulting in event candidates. Finally the module clusterMerger checks for each event candidate if there are any neighboring event candidates that belong to each other and merges them. Finally the event candidates are saved as a datafile.

ANNOTATER This module makes it possible for multiple judges to annotate the data created with the EventCandidates module. After finishing the annotation by both judges it calculates the Kappa score and other statistics and saves the annotation in a datafile.

EVENTDETECTIVE This module uses classification to detect events in the given event candidates and generates a Google map of events with corresponding category icons and wikified text.

4.2 Data collection

For the experiment tweets are used that have coordinates. Using a simple Grep command it was possible to retrieve only tweets with this information from the Karora machine. We collected two datasets. One set for testing and training and one for the final test (devset). The first dataset consisted of Dutch tweets from March 2015 that we have downloaded from the Karora machine. In total there where 566549 tweets with geo information. That is about 3procent of the total number of tweets published in that month. The Devset consists of 165848 tweets from the second half of April 2015.

4.3 Annotation

The data is annotated by two judges. The kappa score was 0.79 for the trainset and 0.8 for the devset. We defined 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

For the testset we annotated 1350 event candidates with these categories. In 87% of the cases the judges agreed.

	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 1: Confusion matrix testset annotation

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

	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>

Table 2: Confusion matrix devset annotation

4.4 Creating Event Candidates

We first processed all tweets by giving each tweet a GeoHash (see chapter x) and tokenize the tweet. That resulted in a dictionary of tweets that we used in the second step, clustering the tweets. The clusterCreator merged all GeoHashes and added the tweets to the correct time period using the function below :

```

#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]
```

The result is a dataset with potential events that we call Event Candidates.

With the use of GeoHash it is possible to have cases where the event is on a border resulting in two events which is actually is one event. We used the module clusterMerger to merge those clusters. This module iterates through all Event Candidates and calculates its neighbors GeoHashes. It then iterates over all neighbors if there are events in the same period. It also checks if the overlap in words is high enough, if so the clusters are merged.

4.5 Event detection

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.

4.5.1 *Feature selection*

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

4.6 Named Entity Recognizion

Because the available classifiers in the Stanford NER are not suitable for Tweets i trained my own classifier. First i collected a list of Dutch words of each category (Location, Persons, Organizations and Miscellaneous) I used the wordlist of Alpino, software that is developed by Professor Gert-jan van Noord. Then i collected one million tweets that have at least one of the words. The second step was to automatically annotate the tweets. I wrote a Python script that processes each tweet and assigns the correct class based on the wordlists to each token and creates the training file. The training file was split to make a testfile (4/5 for the training and 1/5 for testing)

4.7 Results

4.7.1 *Classifier*

	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

Table 3: Results using all features

	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 4: Confusion Matrix Naive Bayes

	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 5: Confusion Matrix Maximum Entropy

4.7.2 *NER*

I compare my results with the standard Stanford 4 class classifier. This classifier is trained on xxx and uses the same classes as my 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>

Table 6: SVM

Entity	P	R	F1	TP	FP	FN
LOCATION	0,9623	0,9363	0,9491	4718	185	321
MISC	0,9744	0,8964	0,9338	1903	50	220
ORGANIZATION	0,9711	0,9315	0,9509	4241	126	312
PERSON	0,9031	0,8773	0,8900	4583	492	641
Totals	0,9477	0,9118	0,9294	15445	853	1494

Table 7: Classifier trained with Tweets

Entity	P	R	F1	TP	FP	FN
LOCATION	0,5779	0,2239	0,3227	1128	824	3911
MISC	0,0280	0,0118	0,0166	25	869	2098
ORGANIZATION	0,2485	0,2554	0,2519	1163	3517	3390
PERSON	0,2627	0,3084	0,2837	1611	4521	3613
Totals	0,2875	0,2318	0,2567	3927	9731	13012

Table 8: Stanford 4-class classifier

	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