rijksuniversiteit groningen

faculteit der letteren

P2000 locatiedata als classifier voor tweets

Gericht zoeken naar verbanden en randzaken rondom een event.

Olivier Louwaars

S2814714

24 juni 2015

P2000 locatiedata als classifier voor tweets

Gericht zoeken naar verbanden en randzaken rondom een event.

Groningen, juni 2015

Auteur: Olivier Louwaars
Studentnummer: 2814714

Afstudeerscriptie in het kader van: Informatiekunde
Faculteit der Letteren
Rijksuniversiteit Groningen

Begeleiders onderwijsinstelling: Prof. dr. J. Bos
Faculteit der Letteren
Rijksuniversiteit Groningen

M. Nissim, PhD
Faculteit der Letteren
Rijksuniversiteit Groningen

Voorwoord

Hoewel het als Premasterstudent een aardige omslag is van het HBO naar het WO, komt de opgedane kennis van beide niveaus samen bij het maken van een scriptie aan het eind van de opleiding. Dankzij een hoge collegedichtheid, is het gelukt om binnen een jaar op te klimmen naar het niveau van een Bachelor student Informatiekunde.
Hiervoor ben ik mijn beide begeleiders, prof. dr. J. Bos, M. Nissim, PhD en alle andere docenten Informatiekunde dankbaar. Ook mijn studiegenoten en iedereen die een kritische blik op deze scriptie heeft geworpen wil ik hierbij bedanken.

Voor de programmacode en scripts die voor dit onderzoek gebruikt zijn verwijs ik u door naar https://github.com/obipls/scriptie

Olivier Louwaars
S2814714

Inhoud

1. Inleiding 1
2. Methode 4
   2.1. Data verzamelen 4
      2.1.1. P2000 data 4
      2.1.2. Twitterdata 5
   2.2. Data koppelen 5
   2.3. Data annoteren 7
   2.4. Data classificeren 8
3. Resultaten 9
   3.1. Precision, recall en F-score 9
   3.2. Confusion matrix 9
   3.3. Features 10
4. Discussie 11
   4.1. Conclusie 11
   4.2. Aanbevelingen 11
   4.3. Implementatie 11

Bibliografie 12

Bijlagen 13
   Bijlage I. Stroomdiagram dataverzameling 13
   Bijlage II. Scrapy 14
      P2000_spider.py 14
      Items.py 14
   Bijlage III. Alerts 15
      jsoncleaner.py 15
      adres.py 16
      getgeo.py 17
   Bijlage IV. Tweets 18
      downloader.py 18

gettweets.py 18
tweetsgeo.py 18
Bijlage V. Matches 19
hasher.py 19
matcher.py 20
Bijlage VI. Classify 22
classify.py 22

Samenvatting

De aanleiding voor het schrijven van deze scriptie was het framework *Twitcident* (Abel et al. 2012). Met Twitcident kan een gebruiker zien hoe er op sociale media gereageerd wordt op een gebeurtenis waarbij inzet van hulpdiensten is vereist. Door middel van bepaalde kernwoorden en zoektermen kunnen de tweets en statusupdates bij elkaar worden gezocht om zo een ruimer beeld te scheppen dan slechts een noodmelding. Opvallend hierbij is echter dat Abel *et al.* geen gebruik maakten van bekende geografische data van beide informatiestromen. In dit onderzoek is daarom ingegaan op precies die informatie. Hierdoor kunnen berichten die van dezelfde locatie komen sneller aan elkaar worden gekoppeld. Als data voor het onderzoek zijn 500.000 tweets en 100.000 *P2000* meldingen gebruikt. *P2000* is het nationale waarschuwingssysteem van Nederlandse hulpdiensten waarmee deze onderling communiceren nadat een melding bij de meldkamer is binnengekomen. Van de helft van deze meldingen was een locatie te bepalen, waarna de juiste tweets bij elke melding konden worden gezocht op basis van GPS coördinaten. Door deze coördinaten volgens het principe van *Geohash* (Beatty 2005) om te rekenen naar een code voor een bepaalde plaats, kunnen eenvoudig de omringende codes gevonden worden. De lengte van de GeoHash bepaalt de straal van de cirkel waarin gezocht wordt. Door een hash van zeven tekens te berekenen, ontstaat er een straal van ~100 meter rond het punt waar de noodoproep betrekking op heeft. Alle tweets in deze cirkel werden vervolgens bij de oproep geplaatst. Op deze manier ontstond er een lijst van 39.000 meldingen waarbij een of meerdere tweets in de buurt werden gepubliceerd. Van deze 39.000 waren er rond de 3.800 tweets op dezelfde dag als de oproep geplaatst. Door vervolgens alleen de tweets op te nemen die in de twee uur voor of de drie uur na een gebeurtenis zijn geplaatst (~1500), had ongeveer een kwart van de meldingen een of meerdere bijbehorende, relevante tweets. Dit betekent dat een *baseline* algoritme dat altijd zegt dat een tweet géén betrekking heeft op een melding, in 75% van de gevallen goed zit. Deze 75% geldt als basis waar de prestaties van een systeem mee vergeleken kunnen worden. Dat systeem bestond in dit geval uit een *classifier* die op basis van woordfrequenties leert en toepast of een woord vaak in een wel of niet relevantie tweet staat. De te bouwen classifier moet dus in ieder geval significant beter presteren dan deze basis om een toevoeging te zijn. Door te leren van 600 geannoteerde tweets, presteerde een *Naive Bayes classifier* inderdaad significant beter, met een *accuracy* van 92% (p=0,000). Ook de *F-score* (resp. 0,86 en 0,93) en de *precision* (0,75 en 0,92) stegen significant terwijl de *recall* daalde (1,0 en 0,93). Het is dus zeker aan te raden om niet alleen op geografische informatie, maar ook op inhoud van tweets te selecteren bij het koppelen van tweets aan meldingen.

1. Inleiding

Elke dag worden er vanuit Nederland meer dan 5 miljoen tweets verstuurd over de meest uiteenlopende onderwerpen (Twittermania 2012). Soms alleen interessant voor de schrijver, soms voor een klein groepje lezers, maar soms ook voor heel Nederland, zoals bij een ramp. Als een tweet voor een groep mensen interessant kan zijn, is het van belang dat hij door hen wordt gezien. Zo kan het bij een ramp interessant zijn om te zien wat twitteraars in de omgeving hierover te zeggen hebben. In deze scriptie zal worden gekeken naar het effect van een geografische voorselectie (dus tweets uit een bepaald gebied) rond de locatie van een noodmelding voor hulpdiensten. Deze selectie bestaat dan uit een klein en hopelijk relevant deel van de dagelijkse ongeordende en onoverzichtelijke hoeveelheid tweets.

In 2012 is door Abel *et al.* gekeken naar de inhoud en betekenis van tweets die betrekking hebben op een incident. Zij hebben in hun studie naar de combinatie van noodmeldingen en tweets gekeken naar de inhoud en betekenis van tweets die betrekking hebben op een incident. Dit leidde tot de ontwikkeling van het framework Twitcident[1]. Uit hun onderzoek kan vooral veel informatie gehaald worden met betrekking tot de uiteindelijke implementatie van dit project, zoals het gebruik van de API (Application Programma Interface) van Twitter en hoe deze gebruikt kan worden om real-time Nederlandse tweets te herkennen en te downloaden. Aangezien het eenvoudiger is om te werken met bestaande data, is er voor het huidige onderzoek gekozen om geen live data te gebruiken. Dit is echter wel het uiteindelijke doel, en daarvoor zijn de genoemde technieken om plaatjes en video's uit tweets te halen in het onderzoek van Abel *et al.* dan ook nuttig. Als de resultaten naar wens zijn, kan het systeem online gezet worden, waar het te maken krijgt met binnenstromende tweets en noodmeldingen in plaats van vaste datasets. Abel *et al.* beschrijven de manieren waarop zij de juiste berichten afvangen (Twitter API en P2000 RSS feed) en door welke services (TwitPic, TwitVid) en API's (REST) ze vervolgens de juiste tweets verkrijgen. De onderzoekers hebben de geo-locatie van tweets niet meegenomen als criterium, waarschijnlijk omdat slechts 0,77% van de tweets toen een locatie meekreeg van de gebruiker (Semiocast 2012). Doordat het percentage tweets met geo-locatie aardig gestegen is, nu rond de 3% (Pool 2015), is het interessant om hier nu wel naar te kijken.

Ook Li *et al*. richtten zich op het detecteren en groeperen van events, maar maakten wel veel gebruik van de ingebouwde geo-locaties van tweets (indien aanwezig). Li *et al*. zien twitteraars als journalisten die overal verspreid zijn en voortdurend nieuwsberichten van 140 tekens maken. Hun bedoeling is om de eerste die over een gebeurtenis (CDE , *Crime and Disaster related Event*) schrijft op te sporen, en van hieruit gerelateerde tweets te vinden om zo geografische verbanden te kunnen leggen en deze inzichtelijk te maken aan gebruikers. Het systeem is bedoeld om achteraf verbanden te kunnen zien, bijvoorbeeld een serie branden rond een bepaald

[1] http://wis.ewi.tudelft.nl/twitcident

punt, en dus niet om incidenten live te volgen. Naast de geo-locatie die Twitter automatisch aan tweets geeft, geven de onderzoekers ook tips voor als deze informatie ontbreekt. Zo kan er gekeken worden naar een patroon in tweets van een gebruiker en als er bij vergelijkbare tweets wel een locatie staat, kan deze dan gebruikt worden. Ook het netwerk van een twitteraar is van belang. Iemand die hij 'volgt' en vaak in zijn tweets noemt zal waarschijnlijk in de buurt wonen, waardoor de locatie van de twitteraar 'voorspeld' zou kunnen worden. Deze informatie is wederom pas voor de implementatie interessant, aangezien er in de statische twitterdata niet naar volgers en hun locatie gezocht kan worden.

Hoewel deze twee onderzoeken op het eerste gezicht dus vooral relevant lijken voor een later stadium, zijn beide juist de aanleiding geweest om dit onderzoek te starten. Ze stellen vast dat het inderdaad zin heeft om een koppeling tussen tweets en noodmeldingen te maken en wat een dergelijke koppeling toe zou voegen. Daarnaast missen Abel *et al.* de toekenning op basis van geo-locaties van tweets, die juist in dit onderzoek centraal staan.

Als bron van de meldingen voor hulpdiensten zullen berichten worden gebruikt die zijn verstuurd via het *P2000* netwerk. Dit is het communicatiesysteem dat door de verschillende (Nederlandse) hulpdiensten wordt gebruikt om melding te maken van incidenten en hulpverleners aan te sturen. De communicatie via het P2000 systeem is niet versleuteld en kan door iedereen worden opgevangen en gepubliceerd op bijvoorbeeld een website. Door op basis van deze berichten een voorselectie van tweets te maken, kunnen tweets die er tekstueel wellicht niets mee te maken hebben, maar wel van dezelfde plaats afkomstig zijn, toch als relevant bestempeld worden. Ook kan het interessant zijn om te zien of er voorafgaand of juist na afloop van een melding tweets opduiken die context kunnen schetsen. Bovendien is de structuur van de data interessant. Voor de onderlinge communicatie is een standaard opgesteld waardoor elke melding een vast stramien heeft met constante parameters zoals locatievermelding en, indien van toepassing, een omschrijving van de situatie. Dit laatste ontbreekt echter vaak, en dan ziet een gebruiker alleen een melding met de strekking: "Brandweer met grote spoed naar adres X". Het combineren met tweets zal dit kunnen verbeteren, doordat buurtbewoners vertellen wat ze zien. Hoewel hulpdiensten zelf wel precies weten wat er aan de hand is kunnen ook zij hun voordeel doen met informatie uit tweets uit de buurt, zoals de beschrijving van een verdacht persoon.

Het uiteindelijke doel van dit onderzoek is het koppelen van tweets aan een melding voor hulpdiensten. Door vervolgens alleen naar deze tweets en de centraal staande noodmelding te kijken en met elkaar te vergelijken, kan een computer vrij eenvoudig leren om relevante en niet relevante tweets te herkennen, en dit vervolgens zelf toepassen op nieuwe meldingen en tweets. Het proces van leren en relevantie toekennen is eenvoudiger en sneller op een kleine hoeveelheid data met daarin zowel een aanzienlijk deel relevante als irrelevante voorbeelden om beide te kunnen herkennen. Als er bijvoorbeeld 50 tweets relevant zijn bij een incident, is

het veel lastiger om deze 50 uit de verzameling van 5 miljoen te halen dan uit een verzameling van 1000 tweets. Natuurlijk zullen er ook irrelevante tweets op dezelfde locatie verschijnen, maar door alleen in de buurt te selecteren zal het allergrootste deel waarschijnlijk afvallen.

In deze scriptie zal het volgende onderzocht worden: “Is het gebruik van een geografische voorselectie voldoende voor het vinden van relevante tweets bij een noodmelding in een klein geografisch gebied binnen Nederland? En zo niet, wat is hiernaast dan nog meer vereist om relevante tweets bij een noodmelding te vinden? ”

Voor de te realiseren applicatie is het van belang dat een goede inventarisatie van de bestaande software wordt gemaakt, om dubbel werk te voorkomen en goed werkende koppelingen tussen de verschillende onderdelen te kunnen maken. In hoofdstuk 2, Methode, zal aan de orde komen op welke manier de software precies toegepast zal worden. De resultaten die geboekt worden zullen vervolgens gerapporteerd worden in hoofdstuk 3, waarna er geconcludeerd wordt in hoeverre deze resultaten daadwerkelijk beter of slechter zijn dan van toeval verwacht mag worden in hoofdstuk 4.

2. Methode

De gehele methode voor dit onderzoek is ingericht op het gebruik van de *Python* voor de programmacode. Zowel door de bestaande kennis en uitvoerige documentatie van deze programmeertaal, als het gebruiksgemak voor taal gerelateerde opdrachten is *Python* ideaal.

2.1. Data verzamelen

Voor dit onderzoek zijn twee bronnen gebruikt: P2000 data met alle noodmeldingen van Nederlandse hulpdiensten, en de Twitter database van de Rijksuniversiteit Groningen met daarin alle tweets in het Nederlands van de afgelopen vijf jaar. Na een pilotstudy op ongeveer 20.000 P2000 meldingen bleek begin mei dat er aan de ene kant erg veel data uit beide bronnen nodig waren om een fatsoenlijk aantal ´matches' tussen tweets en meldingen te kunnen maken, terwijl aan de andere kant het verzamelen en verwerken van data steeds trager ging. Om die reden is er besloten tot een middenweg, namelijk de data van één maand. Deze arbitraire keuze bleek genoeg data op te leveren, terwijl de hoeveelheid verwerkbaar bleef. In bijlage I is te zien hoe de dataverzameling verliep.

2.1.1. P2000 data

Hoewel P2000 data ongecodeerd wordt verstuurd, is er nog wel speciale apparatuur nodig om mee te kunnen luisteren. Gelukkig wordt dit al gedaan door radioamateurs en worden de berichten vervolgens in een database opgeslagen. Helaas is de toegang tot deze database niet gratis, en kan er via de sites niet direct in de database gezocht worden. Live data is eenvoudiger te krijgen, door een abonnement op een RSS feed die automatisch updates doorgeeft, maar voor oude berichten was een script nodig dat de meldingen automatisch kon downloaden.
Deze eerste stap van het onderzoek was direct een grote, namelijk het verzamelen van veranderende data op dezelfde URL van de gekozen P2000 website[2]. Deze site geeft de meldingen per 15 weer, met zo min mogelijk metadata er omheen. Aangezien de website asynchroon loopt blijft de URL ongeacht de inhoud hetzelfde, waardoor een *webcrawler* niet door kan naar eerdere pagina's door de URL aan te passen. Het drukken op de 'vorige' knop door de gebruiker moest dus gesimuleerd worden door een script. Uiteindelijk bleek de knop door te verwijzen naar een PHP-script dat wel het paginanummer in de URL heeft, zodat daar de spider uit de *Scrapy*[3] module heen gestuurd kon worden. *Scrapy* heeft als voordeel dat het gebruikers van tevoren laat definiëren welke informatie of HTML-tags hij wel en niet wil downloaden, zodat het strippen van webpagina's veel sneller kan verlopen dan wanneer de hele pagina gedownload moet worden.

---

[2] http://p2000-online.net
[3] http://scrapy.org

Zo heeft *Scrapy* op 10.250 pagina's alleen de meldingen voor de volledige maand april gedownload waar passende tweets bij gezocht moesten worden.
De enige overeenkomst tussen de tweets en de meldingen zou de locatie zijn, en dus was de volgende stap dat iedere melding een GPS-coördinaat kreeg toegewezen in plaats van de plaats en straatnaam die er nu (soms) in stond. Een coördinaat is universeel en uniek, wat het vergelijken en samenvoegen van meerdere locaties mogelijk maakt. Door voor iedere melding te kijken of er een straatnaam met bijbehorende stad in stond die ook voor kwam in een database met alle plaats- en straatnamen van Nederland, kon er voor ruwweg 50.000 meldingen een adres met eventueel huisnummer gevonden worden. Al deze adressen werden met behulp van de module *Geopy*[4]naar een lengte- en breedtegraad vertaald. Geopy helpt gebruikers om adressen en coördinaten bij elkaar te zoeken, en kan via *Open Street Maps* bij bijna alle adressen op aarde. Helaas mag er via de gratis service slechts een beperkt aantal adressen per etmaal opgezocht worden (35.000), waardoor het een aantal dagen duurde om alle coördinaten te verkrijgen.

2.1.2. Twitterdata

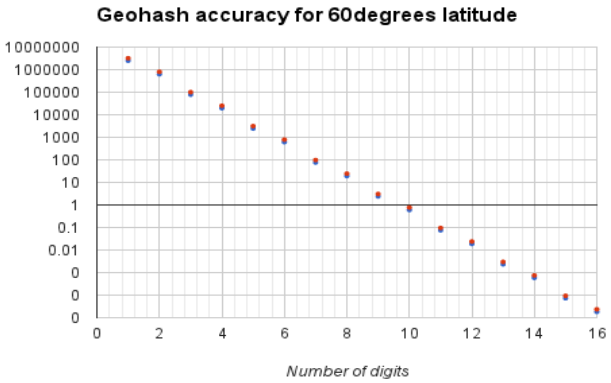
Voor het downloaden van de tweets was een eenvoudig script afdoende, er hoefde voor iedere tweet uit april alleen gekeken te worden of deze een GPS-coördinaat bevatte, waarna hij aan de dataset toegevoegd kon worden. De faculteit Letteren van de Rijksuniversiteit Groningen verzamelt al jaren Nederlandse tweets, waarna deze beschikbaar worden gesteld aan studenten en onderzoekers om mee te werken. Naast de data zelf worden er ook een aantal instrumenten aangeboden waarmee de data geselecteerd of bewerkt kan worden. Specifieke gegevens van iedere tweet worden opgeslagen in een groot aantal kolommen, die niet altijd allemaal nodig zijn. Met het script *Tweet2Tab* kunnen de gewenste kolommen geselecteerd en opgeslagen worden. Voor dit onderzoek waren de tijd, datum, locatie, gebruikersnaam en tekst van belang.

2.2. Data koppelen

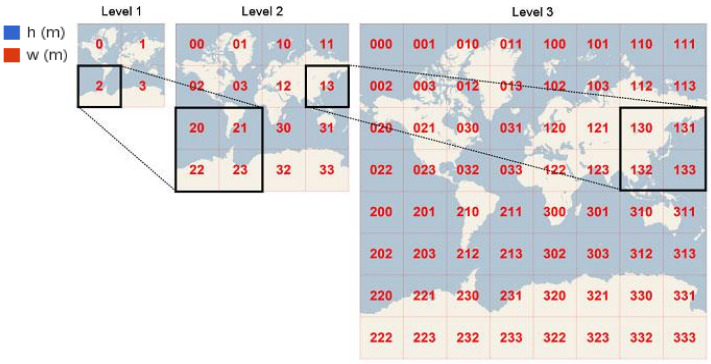
Aangezien de coördinaten voor elke tweet en melding erg nauwkeurig zijn, is de kans op overlap tussen de coördinaten van een tweet en een melding verwaarloosbaar. Om die reden zijn alle coördinaten versleuteld volgens het G*eoHash* (Beatty 2005) principe. Een G*eoHash* is een reeks letters die een coördinaat representeert, waarbij de lengte van de hash gelijk is aan de precisie van het coördinaat. Zoals in Figuur 1 te zien is, heeft een hash van lengte 7 een precisie van ~100 meter. Dit is voldoende voor dit onderzoek, aangezien iemand op deze afstand iets mee krijgt van een incident. Een ander voordeel van een *geohash*, is dat van elk punt altijd alle 'buren' bekend zijn. De eerste zes letters zijn gelijk, en de zevende geeft de positie weer ten opzichte van het eerste punt. Als een punt tegen een grensvlak aan ligt, en dus een buurman heeft die met andere letters begint (Figuur 2), houdt de functie die alle buren berekent hier rekening mee en zal dit punt ondanks een andere code toch meenemen.

[4] https://geopy.readthedocs.org

Op deze manier worden er acht vierkanten verkregen rondom een punt (dus negen in totaal), waarna er wordt gezocht naar een tweet met dezelfde *hash* als één van deze negen vierkanten (Figuur 3). In de praktijk betekent dit dat er voor elk van de 500.000 tweets moet worden gekeken of deze voorkomt in een van de negen mogelijkheden van de 50.000 meldingen, waardoor het runnen van dit script het meest tijdrovend was van het onderzoek. Voor 39.000 meldingen bleek een of meerdere tweets van dezelfde locatie te zijn verzonden.



Figuur 1: De precisie van een GeoHash met x aantal tekens. Bron: http://goo.gl/W9dVMI



Figuur 2: Versimpelde uitleg van het opbouwen van een GeoHash Bron: https://goo.gl/yxRNBT

```
1 def lochasher(lines):
2         hashDict = {}
3         for line in lines:
4             if type(line[1]) != tuple:
5               hashed = Geohash.encode(round(float(line[0][0]),6)
6               ,round(float(line[0][1]),6),7)
7               hashDict.setdefault(hashed,[]).append(line)
8             else:
9               hashed = Geohash.encode(round(float(line[0][1]),6)
10              ,round(float(line[0][0]),6),7)
11              hashedneighbors = geohash.neighbors(hashed)
12              hashedneighbors.append(hashed)
13              hashDict.setdefault((' '.join(hashedneighbors)
14              ,line[1]),[]).append(line)
15        return hashDict
16
17 def locmatcher(tweets,alerts):
18        matchDict={}
19        pbar=ProgressBar()
20        x=0
21        for hashedtw in pbar(tweets):
22            for hashedal in alerts:
23                if  hashedtw in hashedal[0].split():
24                    matchDict.setdefault(hashedal[0]
25                    ,[]).append(hashedtw)
26                    x+=1
27        print(x)
28        return matchDict
```

Figuur 3: Programmacode voor de koppeling van meldingen en tweets.

2.3. Data annoteren

Nadat de tweets aan de bijbehorende locatie gekoppeld waren, kon er een eerste analyse van de paren gemaakt worden. Iedere melding had gemiddeld drie tweets, met als hoogste negen. Aangezien deze tweets van de hele maand april konden zijn is er als eerst gefilterd op tweets van dezelfde dag als de melding. De ongeveer 4.000 overgebleven paren hadden gemiddeld minder dan twee tweets, waarop is besloten om voor 700 meldingen (1.400 tweets) te annoteren of een tweet hier wel of niet relevant voor was. Na annotatie bleek echter dat slechts een achtste van de tweets relevant was, waardoor een systeem dat iedere tweet als niet relevant zou classificeren (*most frequent class baseline*) het in 87,5% goed zou hebben. Door de relevante tweets te analyseren werd duidelijk dat het overgrote deel daarvan binnen twee uur vóór en drie uur ná een incident werden gepost. Door alleen deze tweets (~1800) te selecteren werd een kwart van de tweets relevant, waardoor het percentage door de *baseline* goed voorspelde antwoorden daalde. De *baseline classifier* bleef echter voorspellen dat geen enkele tweet relevant is, terwijl het doel juist was om relevante tweets te kunnen herkennen. Zodoende moest een systeem dat relevante tweets wél weet te herkennen getraind worden.

Van de relevante tweets bleek een groot deel gebaseerd op P2000 berichten, en verwezen daar soms ook naar (Tabel 1). Vaak gaf de tweet echter nog wel een aanvulling op de melding, waardoor aan een schijnbaar nietszeggende melding toch wat context kan worden toegevoegd. Ook kranten, omroepen en de hulpdiensten zelf zijn actief op twitter en melden na afloop of ter plaatse wat er aan de hand is. ‘Normale’ twitteraars die toevallig ergens passeerden waren zeldzaam in de data, en kwamen vaker voor naarmate het incident groter werd.

Tabel 1: Een aantal gematchte meldingen en tweets

A1 AMBU OG902 Botersloot 3011HE Rotterdam rt bon 38075
#ROTTERDAM #RRM | Bedrijf / instelling 1 Botersloot rt daadwerkelijke overval . politie doet onderzoek | http://t.co/GcjLac8A5r #p2000
PRIO 1 Ongeval Letsel Buiten : : Oude Raadhuisstraat : Didam (OVDB Alarm , rv) (GRIP 1)
@rachidfinge attractie is nu leeg !! Brandweer heeft de inzittende gered
PRIO 1 Buitenbrand : : Rolklaver : Kampen 041092 042095 042086 (BR: middel) (AGS Alarm) (GRIP 1)
Brandweer Kampen : geen asbest in woonwijk gevonden na brand volkstuinencomplex http://t.co/2uwutbUByA #112overijssel #rtvoost
Prio 2 assistentie kleine ontsmetting (inmelden: BNH-INCI-30) (-) (OPS: zeer grote brand) (GRIP: 2) Noordergeestkerk Noordergeeststraat 9 Heiloo Brand Bijeenkomst 4333
Grote brand vannacht bij leegstaande kerk in #Heiloo . Alleen de toren staat nog overeind . @RTVNH http://t.co/K6DiS3McoP
A1 5011LN X : HAG Tilburg Beethovenlaan X Beethovenlaan X HAG Tilburg Beethovenlaan Tilburg 32848
Tilburg - Verwarde Tilburger slaat voorruit auto kapot : De politie heeft dinsdag 28 april 2015 omstreeks 16.35... http://t.co/bYz1twXaZ2

2.4. Data classificeren

Per melding bleek na alle bewerkingen gemiddeld iets meer dan één tweet binnen het gestelde tijdskader geplaatst, waarop het nieuwe aantal te annoteren tweets op 600 is gesteld zodat opnieuw een derde van alle tweets geannoteerd werd. Voor deze tweets is handmatig aangegeven of ze relevant (1) of niet relevant (0) waren voor de betreffende noodoproep, waarna er met behulp van het *Naive Bayes* (Manning, Raghavan en Schütze 2008) algoritme door het systeem kon worden geleerd waar een (ir)relevante tweet aan te herkennen is. *Naive Bayes* gaat uit van *probabilities*, dit betekent dat het kansen berekent voor elk woord dat er in wordt gestopt: de kans dat het woord in een relevante tweet staat, de kans dat het in een irrelevante tweet staat, de kans dat een tweet relevant is en de kans dat het woord überhaupt voorkomt. Door deze kansen vervolgens weer door elkaar te delen of met elkaar te vermenigvuldigen, komt het algoritme tot een conclusie: wel of niet relevant. In Vergelijking 1 is in woorden te zien welke kansen het algoritme gebruikt. *Posterior* is de kans dat het woord in een relevante tweet staat. *Prior* is het aantal keer dat het woord eerder al voorkwam in een relevante tweet gedeeld door het totaal aantal tweets. *Likelihood* is de kans dat het woord voorkomt in alle tweets en *evidence* is de kans dat een tweet positief is.

$$posterior = \frac{prior \times likelihood}{evidence}$$

Vergelijking 1: Naive Bayes in woorden

Het doel is om het systeem te leren documenten te classificeren. Dit is een vorm van *supervised machine learning*, een methode om cmputers zichzelf dingen aan te leren. Hierbij zijn de mogelijke klassen op voorhand gegeven en is van de documenten in de trainingsdata door de handmatige annotatie bekend tot welke klasse zij behoren. Het categoriseren wordt gedaan door het voorspellen van de meest waarschijnlijke klasse op basis van onder andere woordfrequentie. Het systeem leert met behulp van de trainingsdata (meestal 80% van de totale data) welke woorden bij welke klasse het meeste voorkomen. In een resterende test-set (20%) van documenten kan vervolgens worden getest hoe goed de *Naive Bayes classifier* werkt. In *Python* kan middels de NLTK[5] (Natural Language Toolkit) module op deze manier geclassificeerd worden.

[5] http://nltk.org

3. Resultaten

In dit hoofdstuk worden de resultaten van de in hoofdstuk 2 beschreven werkwijze getoond. Deze resultaten worden in perspectief gezet door ze te vergelijken met de prestaties van een *baseline classifier*, die een beslissing neemt alleen op basis van de vaakst voorkomende klasse (in dit geval irrelevant) in een dataverzameling. De resultaten worden geobjectiveerd met behulp van *precision, recall* en *F-score*. *Precision* is de hoeveelheid juiste beslissingen die het systeem neemt ten opzichte van het totaal aan beslissingen, terwijl *recall* het aantal juist genomen beslissingen ten opzichte van het totaal aantal juist te nemen beslissingen is. *F-score* is het harmonisch gemiddelde tussen deze twee maten.

3.1. Precision, recall en F-score

Door van 80% van de data te leren, en het geleerde op de overige 20% data toe te passen, was het model in staat om 92% van de tweets aan de juiste categorie toe te wijzen. Een op het oog significant verschil dat ook wordt ondersteund door een gepaarde t-test (p=0,000).
Aangezien het basisalgoritme iedere tweet kwalificeerde als niet-relevant (most frequent class principe), heeft het alle niet-relevante tweets juist (*recall* is 100%). Dat het vervolgens een groter aantal tweets in deze categorie heeft dan er in zouden moeten zitten heeft alleen invloed op de *precision* (75%). Het harmonisch gemiddelde van deze twee levert een *F-score* op van 0,86, terwijl het nieuwe model 0,92 voor de relevante en zelfs 0,94 voor de irrelevante tweets scoort (Tabel 2). Hoewel dit verschil een stuk kleiner is, betekent dit dus wel een forse toename in *precision* (van 0,75 naar respectievelijk 0,95 en 0,90). Daarentegen daalt de *recall* (van 1,0 naar 0,93), maar met een kleiner percentage.

Tabel 2: Scores van het nieuwe model vergeleken met de Baseline (most common)

	PRECISION	RECALL	F-SCORE
BASELINE	0,75	1,0	0,86
RELEVANT	0.958333	0.884615	0.92
IRRELEVANT	0.909091	0.967742	0.9375

3.2. Confusion matrix

Naast deze totaalscores is het ook interessant om te zien hoe vaak de *classifier* de juiste beslissing nam, en hoe vaak de verkeerde. Tabel 2 laat zien welke keuzes gemaakt zijn door het getrainde algoritme, waarbij de r voor referentie staan (wat het zou moeten zijn) en de v voor voorspeld (wat het systeem denkt dat het is). Opvallend hier aan is het relatief grote percentage dat eigenlijk relevant is, maar niet zo beoordeeld wordt. Hieruit blijkt dat er bepaalde signaalwoorden die een mens wel aan een gebeurtenis zou koppelen, niet als zodanig door de machine herkend worden. Dit kan verholpen worden door extra nadruk op bepaalde woorden te leggen, die de doorslag geven naar positief of negatief als ze aangetroffen worden. Aangezien het aantal tweets waarop uiteindelijk het getrainde algoritme losgelaten wordt slechts beperkt is (~300,

dus 75 relevant) is het gevaarlijk om bepaalde woorden aan te wijzen die nu relatief vaak voorkomen, omdat dit misschien toeval is en ze over het geheel gezien juist niet belangrijk zijn. Voor dit principe, *overfitting* genaamd, wordt gewaarschuwd in het artikel *Domain Adaptation: Overfitting and Small Sample Statistics* (Kakade, Foster en Salakhutdinov 2011). Om deze reden is er ook gekozen om zeer spaarzaam voorkomende woorden te negeren, en alleen de 3000 meest voorkomende woorden als trainingsdata te gebruiken. Als een woord immers slechts eenmaal voorkomt koppelt de classifier dit direct als belangrijk woord aan een bepaalde keuze.

Tabel 3: Confusion matrix van de verdeling van de scores

	IRRELEVANT (V)	RELEVANT(V)
IRRELEVANT (R)	52.6%	1.8%
RELEVANT (R)	5.3%	40.4%

3.3. Features

Ook is het mogelijk en aan te raden om (te experimenteren met) het aantal woorden, ofwel features, waarmee getraind wordt te beperken. Woorden die in zowel relevante als irrelevante tweets veel voor komen, zoals lidwoorden en voornaamwoorden, kunnen eruit gefilterd worden door ze op te nemen in een *stoplist*. Deze woorden worden vervolgens uitgesloten van het automatisch leren. In dit experiment voegde een stoplijst echter niets toe en gaat wederom het argument van *overfitting* op: als de *classifier* te specifiek getraind wordt, gaat hij de mist in bij nieuwe data die andere woorden bevat. De meest informatieve woorden voor de testset staan in Tabel 3, waarbij de verhouding van de kans dat het woord in de ene klasse voorkomt ten opzichte van de kans dat het in de andere klasse voorkomt wordt genoemd. Een ‘@’ komt bijvoorbeeld 7,8 keer vaker voor bij een niet relevante tweet dan bij een relevante, terwijl ‘Brandweer’ 6,9 keer vaker relevant dan niet relevant is. Door de kleine hoeveelheid (en daardoor wellicht weinig variabele) testdata, kunnen ook woorden die minder logisch lijken en een mens niet zouden helpen juist heel hoog scoren voor de machine.

Tabel 4: Informatieve woorden en de kans van voorkomen in een klasse

WOORD	WAARDEN	VERHOUDING
GAAT	rel : irrel	13.8 : 1
1	rel : irrel	11.3 : 1
WEER	irrel : rel	8.2 : 1
@	irrel : rel	7.8: 1
!	irrel : rel	7.5 : 1
NIET	irrel : rel	7.0 : 1
BRANDWEER	rel : irrel	6.9 : 1
/	rel : irrel	6.9 : 1
MAAR	irrel : rel	6.4 : 1
(	rel : irrel	6.0 : 1

4. Discussie

De gevonden resultaten en gebruikte methodes zullen in dit hoofdstuk worden geëvalueerd en beoordeeld, verder wordt er ook vooruitgekeken op eventueel vervolgonderzoek en de real-time implementatie van de resultaten.

4.1. Conclusie

Op basis van de gevonden resultaten, kan er geconcludeerd worden dat de filtering niet afdoende is als een tweet uit hetzelfde tijdsframe en van dezelfde locatie komt als een noodmelding via P2000. Hoewel een aanzienlijk deel van de tweets die hieraan voldoen relevant zijn voor de melding, zijn er teveel andere tweets die hier niet aan voldoen en is de balans te scheef om er direct mee te kunnen werken. Zo kan er niet zonder meer vanuit gegaan worden dat een tweet die in deze selectie wordt aangetroffen relevant is. Wel is aangetoond dat het op tijd en locatie voorselecteren van tweets een bijzonder goede basis is voor het verdere classificeren. Het percentage van 25% relevante tweets bij een tijdsvenster van vijf uur rond een gebeurtenis halveerde immers al bij een hele dag mét een overeenkomstige locatie, laat staan als er gekeken zou worden naar tweets op dezelfde dag door een hele stad of zelfs heel Nederland. Het percentage relevante tweets zou dan verwaarloosbaar worden.

Hoewel er is begonnen met een ruime hoeveelheid tweets en P2000 meldingen, bleef daar door steeds strengere selecties niet veel van over. Belangrijke verfijningen zoals *feature selection* (het benadrukken van bepaalde belangrijke woorden) en het gebruik van een *stoplist* waren niet mogelijk door een te gering aantal tweets in de uiteindelijke trainings- en testdata.

4.2. Aanbevelingen

De aanbeveling van een grotere hoeveelheid data ligt erg voor de hand, maar hierbij moet wel rekening worden gehouden met de extra verwerkingstijd die hiermee gemoeid is. Voor de tweets is dit probleem minder groot omdat hier weinig *preprocessing* voor nodig is, maar voor de P2000 meldingen zijn veel stappen nodig om alle ingesloten informatie om te zetten naar een nuttig formaat.

Ook kan er gekeken worden naar het kleine aantal tweets dat nu per melding gevonden wordt, en of er een manier is om dit te vergroten. Aangezien alle tweets met geo-locatie in dit onderzoek gekoppeld zijn aan een melding, zou er in het vervolg ook naar tweets zonder locatie, maar met bijvoorbeeld overeenkomende straatnamen of gebouwen gekeken kunnen worden.

4.3. Implementatie

De scripts in dit onderzoek kunnen zomaar niet worden meegenomen naar een live versie, aangezien deze bedoeld zijn voor het werken met bestaande dataverzamelingen. De benodigde aanpassingen zijn echter klein, en met behulp van diverse Twitter API's (Abel, et al. 2012) en de P2000 feed kan het programma real-time tweets bij een melding zoeken op basis van locatie.

Bibliografie

Abel, Fabian, Claudia Hauff, Geert-Jan Houben, Ke Tao, en Richard Stronkman. 2012. „Twitcident: Fighting Fire with Information from Social Web Streams." In *WWW 2012, Proceedings of the 21st World Wide Web Conference 2012*, 305-308. Lyon, France: Companion Volume. Geopend Maart 9, 2015.

Beatty, Bryan Kendall (Sammamish, WA, US). 2005. Compact text encoding of latitude/longitude coordinates. United States Patent 20050023524. Geopend April 15, 2015.

Kakade, Sham, Dean Foster, en Ruslan Salakhutdinov. 2011. „Domain Adaptation: Overfitting and Small Sample Statistics." *Cornell University Library.* Geopend Mei 16, 2015. http://arxiv.org/abs/1105.0857v1.

Li, Rui, Kin Hou Lei, R Khadiwala, en K.C.-C. Chang. 2012. „TEDAS: A Twitter-based Event Detection and Analysis System." In *2012 IEEE 28th International Conference on Data Engineering (ICDE)*, 1273 - 1276. Washington, DC: IEEE. Geopend Maart 9, 2015.

Manning, Chrisopher D., Prabhakar Raghavan, en Hinrich Schütze. 2008. „13 Text Classification and Naive Bayes." In *Introduction to Information Retrieval*, door Chrisopher D. Manning, Prabhakar Raghavan en Hinrich Schütze, 206,207, 234-242. Cambridge: Cambridge University Press.

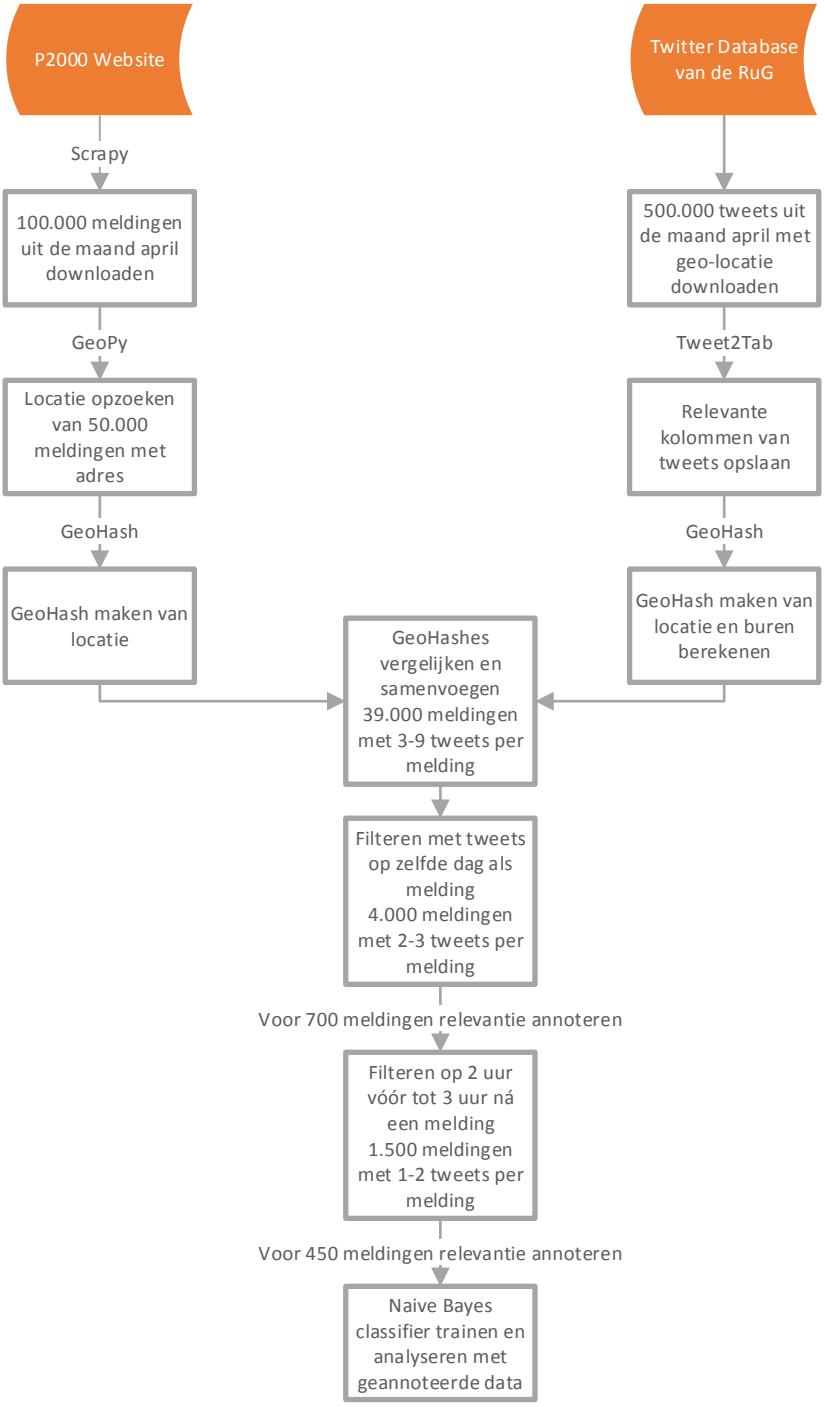
Pool, Chris. 2015. „Detecting local events in the twitter stream." juni. Geopend juni 12, 2015.

Semiocast. 2012. „Twitter reaches half a billion accounts." *http://semiocast.com.* 30 juli. Geopend juni 10, 2015. http://semiocast.com/en/publications/2012_07_30_Twitter_reaches_half_a_billion_accounts_140m_in_the_US.

Twittermania. 2012. „5 miljoen Nederlandstalige tweets per dag." *Twittermania.* 1 maart. Geopend juni 9, 2015. http://twittermania.nl/2012/03/5-miljoen-nederlandstalige-tweets-dag/.

Bijlagen

Bijlage I. Stroomdiagram dataverzameling



Bijlage II. Scrapy

P2000_spider.py

```
#Functioncall = scrapy crawl p2000 -o <datafile>.json

import scrapy
from p2000.items import *

class P2000Spider(scrapy.Spider):
        name = "p2000"
        allowed_domains = ["p2000-online.net"]
        start_urls = ["http://www.p2000-online.net/p2000.php?Pagina=
        %d&NoViewCaps=1&AutoRefresh=uit&Pagina=%d" %(n,n) for n in
        range(0, 10250)]

        def parse(self, response):
                for sel in response.xpath('//tr'):
                    item = P2000Item()
                    item['date'] = sel.xpath('td[1]/text()').extract()
                    item['time'] = sel.xpath('td[2]/text()').extract()
                    item['cause'] = sel.xpath('td[5]/text()').extract()
                    yield item
```

Items.py

```
        import scrapy
        class P2000Item(scrapy.Item):
            time = scrapy.Field()
            date = scrapy.Field()
            cause = scrapy.Field()
            pass
```

Bijlage III. Alerts

jsoncleaner.py

```
import json, re, nltk, pickle
from nltk.tag.stanford import NERTagger
from collections import Counter

def RepresentsInt(s):
    try:
        int(s)
        return True
    except ValueError:
        return False

def cleanup():

    adressfile='adresses.pickle'
    adresses=pickle.load(open(adressfile, 'rb'))

    with open('aprilp2000.json') as data_file:
         alerts = json.load(data_file)
    print("Linecount = {}".format(len(alerts)))

    # Remove all empty causes from the list of alerts
    alerts = [d for d in alerts if d.get('cause') != []]
    print("Empty cases removed, linecount = {}"
    .format(len(alerts)))

    # Remove all alerts or other lines without a time
    alerts = [d for d in alerts if ':' in ''
    .join(d.get('time'))]
    print("Empty timestamps removed, linecount = {}"
    .format(len(alerts)))

    # Remove all alerts without a priority index (Capital letter
    #followed by a space and one number)
    alerts = [d for d in alerts if re.search('[A-P]:?\s?[0-9]',
    ''.join(d.get('cause')))]
    print("No priority entries removed, linecount = {}"
    .format(len(alerts)))

    print(len(alerts))

    # Gather all causes in 1 list for geosnatcher
    causes= [d.split() for d in [''.join(d) for d in
    [d.get('date')+[" "]+d.get('time')+[" "]+d.get('cause')
```

```
    for d in alerts]]]
    print("List created")

    causeDict = {}
    for cause in causes:
         for i,token in enumerate(cause):
             if str(token) in adresses.keys():
                city=token
                for j in range(len(cause)-1):
                    if cause[j] in adresses.get(token):
                        street=cause[j]
                        if RepresentsInt(cause[j+1]) == True:
                            if int(cause[j+1]) < 100:
                               adress=[str(cause[j+1])
                               ,str(street),str(city)]
                               causeDict[(cause[0],cause[1])]
                               =[cause[2:],adress]
                        else:
                            adress=[str(street),str(city)]
                            causeDict[(cause[0],cause[1])]
                            =[cause[2:],adress]
    with open('causesaddr.pickle','wb') as f:
         pickle.dump(causeDict,f)

cleanup()
```

adres.py

```
import csv
from collections import defaultdict
import pickle
with open('adressen.csv') as csvfile:
      adressen = csv.reader(csvfile, delimiter=';')
      adresDict=defaultdict(list)
      for adres in adressen:
              if adres[2] != "Postbus":
                              adresDict[adres[1]].append(adres[2])
      for item in adresDict.values():
               item=set(item)
               item=list(item)
with open('adresses.pickle','wb') as f:
              pickle.dump(adresDict,f)
```

getgeo.py

```
from geopy.geocoders import Nominatim
import pickle

def locationsnatcher(causes):
        locDict={}
        geoList=[]
        geolocator = Nominatim()
        x=0
        for cause in causes:
            fullAddress= ' '.join(causes.get(cause)[-1])
            geoList.append([cause,fullAddress])
        print(len(geoList))

        for loc in geoList:
            location = geolocator.geocode(loc[-1], timeout=15)
            print(x)
            x+=1
            if location != None:
                 locgeo=(location.latitude,location.longitude)
                 locDict[(locgeo,loc[0])]=[location.address,
               causes.get(loc[0])[0]]
        return locDict

def main():
    addressfile='causesaddr.pickle'
    addresses=pickle.load(open(addressfile, 'rb'))
    coordinates=locationsnatcher(addresses)
    print (len(coordinates))
    with open('geolocs.pickle','wb') as f:
        pickle.dump(coordinates,f, protocol=0)

if __name__ == '__main__':
        main()
```

Bijlage IV. Tweets

downloader.py

```
#Call: python downloader.py | gettweets.py > output.txt
import os
import sys
def main():
	for i in range (4,6):
		gettweets= "zcat /net/corpora/twitter2/Tweets/2015/0"
		+str(i)+"/2015???????.out.gz | /net/corpora/
		twitter2/tools/tweet2tab
		-i date coordinates place user words"
		os.system(gettweets)
main()
```

gettweets.py

```
import sys
def main():
	tweets = []
	for line in sys.stdin:
		elements = line.strip().split('\t')
		if elements[1] != '' :
			tweets.append(elements)
			print('\t'.join(elements))
main()
```

tweetsgeo.py

```
import pickle
def gettweetlocs():
    tweetDict = {}
    with open("tweets.txt") as f:
       for line in f:
          tweet = line.strip().split('\t')
          tweetgeo = tweet[1].split()
          tweetgeo = (tweetgeo[0],tweetgeo[1])
          tweetDict[tweetgeo,tweet[-2]]=
          (tweet[0],tweet[1])
       return tweetDict

def main():
	coordinates=gettweetlocs()
	with open('tweettd.pickle','wb') as f:
		pickle.dump(coordinates,f)
if __name__ == '__main__':
	main()
```

Bijlage V. Matches

hasher.py

```
import pickle, Geohash, geohash
from progressbar import ProgressBar

def lochasher(lines):
	hashDict = {}
	for line in lines:
		if type(line[1]) != tuple:
			hashed = Geohash.encode(round(float(line[0][0]),6)
				,round(float(line[0][1]),6),7)
			hashDict.setdefault(hashed,[]).append(line)
		else:
			hashed = Geohash.encode(round(float(line[0][1]),6)
				,round(float(line[0][0]),6),7)
			hashedneighbors = geohash.neighbors(hashed)
			hashedneighbors.append(hashed)
			hashDict.setdefault((' '.join(hashedneighbors)
				,line[1]),[]).append(line)
	return hashDict

def locmatcher(tweets,alerts):
	matchDict={}
	pbar=ProgressBar()
	x=0
	for hashedtw in pbar(tweets):
		for hashedal in alerts:
			if  hashedtw in hashedal[0].split():
				matchDict.setdefault(hashedal[0],[])
				.append(hashedtw)
	return matchDict

def getvalues(tweets,alerts,matches):
	pbar=ProgressBar()
	taDict={}
	for alert in pbar(alerts):
		for match in matches:
			if alert[0] == match:
				alert=alerts.get(alert)[0]
				for tweethash in matches.get(match):
					tweet=tweets.get(tweethash)[0]
					taDict.setdefault(alert,[])
					.append(tweet)
	return(taDict)

def main():
```

```
45      tweets = pickle.load(open('tweetlocs.pickle','rb'))
46      alerts = pickle.load(open('geolocsall.pickle','rb'))
47      hashedtweets = lochasher(tweets)
48      with open('hashedtweets.pickle','wb') as f:
49              pickle.dump(hashedtweets,f)
50      hashedalerts = lochasher(alerts)
51      with open('hashedalerts.pickle','wb') as f:
52              pickle.dump(hashedalerts,f)
53      matched=locmatcher(hashedtweets,hashedalerts)
54      with open('matchedvalues.pickle','wb') as f:
55              pickle.dump(matched,f)
56      hashedtweets = pickle.load(
57              open('hashedtweets.pickle','rb'))
58      hashedalerts = pickle.load(
59              open('hashedalerts.pickle','rb'))
60      matched = pickle.load(open('matchedvalues.pickle','rb'))
61      values=getvalues(hashedtweets,hashedalerts,matched)
62      with open('allmatches.pickle','wb') as f:
63              pickle.dump(values,f)
64
65 if __name__ == '__main__':
66      main()
```

matcher.py

```
1  import pickle, random
2  from collections import Counter
3  from progressbar import ProgressBar
4  from datetime import *
5
6  def matcher():
7       pbar=ProgressBar()
8       textDict={}
9       matches = pickle.load(open('allmatches.pickle','rb'))
10      alerts = pickle.load(open('geolocsall.pickle','rb'))
11      tweets = pickle.load(open('tweetlocs.pickle','rb'))
12      tweetmeta = pickle.load(open('tweettd.pickle', 'rb'))
13      for match in pbar(matches):
14          key=(' '.join(alerts.get(match)[1]),match[1])
15          tweetkeys=set(matches.get(match))
16          for tkey in tweetkeys:
17              value=tweets.get(tkey),tkey[1],tweetmeta.get(tkey)[0]
18              textDict.setdefault(key,[]).append(value)
19      return textDict
20
21 def comparedays():
22      texts = pickle.load(open('alertstweets.pickle','rb'))
23      samedaytexts={}
```

```
24      for alert, tweets in texts.items():
25              for tweet in tweets:
26                      tweetdate=tweet[-1][8]+tweet[-1][9]+
27                      tweet[-1][7]+tweet[-1][5]+tweet[-1][6]+
28                      tweet[-1][4]+tweet[-1][2]+tweet[-1][3]
29                      if str(alert[1][0]) == tweetdate:
30                        samedaytexts.setdefault(alert,[]).append(tweet)
31      return samedaytexts
32
33 def comparetimes():
34      texts = pickle.load(open('alertstweets.pickle','rb'))
35      sametimetexts = {}
36      for alert in texts:
37              adate=alert[-1][0].split('-')
38              atime=str(alert[-1][1]).split(':')
39              atimestamp=datetime(int("20"+adate[2]),int(adate[1]),
40                      int(adate[0]),int(atime[0]),int(atime[1]))
41              atimestampmin=atimestamp-timedelta(hours=2)
42              atimestampmax=atimestamp+timedelta(hours=3)
43              for tweet in texts.get(alert):
44                  ttime=tweet[-1].split()[1].split(':')
45                  tdate=tweet[-1].split()[0].split('-')
46                  ttimestamp=datetime(int(tdate[0]),int(tdate[1]),
47                     int(tdate[2]),int(ttime[0]),int(ttime[1]))
48                  if ttimestamp > atimestampmin
49                  and ttimestamp<atimestampmax:
50                      sametimetexts.setdefault(alert,[]).append(tweet)
51      return sametimetexts
52
53 def getannodata():
54      texts = pickle.load(open('baselinetimes.pickle','rb'))
55      for x,match in enumerate(random.sample(texts,20)):
56              print (str(x+1)+'\t'+match[0])
57              for y,tweet in enumerate(texts.get(match)):
58                      print (str(y+1)+'\t'+tweet[0]+'\t'+tweet[1]+'\t0')
59
60 if __name__ == '__main__':
61      texts=matcher()
62      with open('alertstweets.pickle','wb') as f:
63              pickle.dump(texts,f)
64      samedays=comparedays()
65      with open('baseline.pickle','wb') as f:
66              pickle.dump(samedays,f)
67      sametimes=comparetimes()
68      with open('baselinetimes.pickle','wb') as f:
69              pickle.dump(sametimes,f)
70      getannodata()
```

Bijlage VI. Classify

classify.py

```
import nltk.metrics, sys, pickle, random
from itertools import chain
from tabulate import tabulate
from collections import Counter,defaultdict
from nltk import word_tokenize as wordToks
from nltk.classify import NaiveBayesClassifier as NBC, accuracy
reload(sys)
sys.setdefaultencoding("utf-8")

def getfeatures(tweet,testingSet):
      featureDict ={}
      tweetTokens = set(wordToks(tweet))
      for word in testingSet:
         if word in tweetTokens:
                featureDict[word] = True
         else:
                featureDict[word] = False
      return featureDict

def classify():
      tweetSet = set()
      negtweetSet=set()
      for annotated in open("dagannotatie1.txt"):
         tweet = annotated.strip().split("\t")
         if len(tweet) == 4:
            if tweet[3].strip() is '0':
               negtweetSet.add((tweet[1], tweet[3]))
            else:
               tweetSet.add((tweet[1], tweet[3]))
      tweetSet.update(random.sample(negtweetSet,150))
      wordSet = chain(*[nltk.word_tokenize(i[0]) for i in tweetSet])
      wordAmount = Counter(wordSet)
      testingSet = set([word for word,
               n in wordAmount.most_common(250)])
      featureList = []
      for tweet, tag in tweetSet:
         featureDict = getfeatures(tweet,testingSet)
         featureList.append((featureDict,tag))
      divider = int(0.8*len(featureList))
      trainTweets = featureList[:divider]
      testTweets = featureList[divider:]
      classifier = NBC.train(trainTweets)
      print(accuracy(classifier,testTweets))
```

```
44      ref = []
45      tagged = []
46      for f,e in testTweets:
47          ref.append(e)
48          tagged.append(classifier.classify(f))
49      cm = nltk.ConfusionMatrix(ref,tagged)
50      print(cm.pretty_format(sort_by_count=True,
51              show_percents=True, truncate=9))
52      refsets = defaultdict(set)
53      taggedsets = defaultdict(set)
54      for i, (feats, label) in enumerate(testTweets):
55          refsets[label.strip()].add(i)
56          observed = classifier.classify(feats)
57          taggedsets[observed.strip()].add(i)
58      table = [["Precision", "Recall", "F-score"], ["Positive",
59      nltk.metrics.precision(refsets['1'], taggedsets['1']),
60      nltk.metrics.recall(refsets['1'], taggedsets['1']),
61      nltk.metrics.f_measure(refsets['1'], taggedsets['1'])],
62      ["Negative", nltk.metrics.precision(refsets['0'],
63              taggedsets['0']), nltk.metrics.recall(refsets['0'],
64              taggedsets['0']), nltk.metrics.f_measure(refsets['0'],
65              taggedsets['0'])]]
66      print(tabulate(table,headers="firstrow",tablefmt="plain"))
67      print 'pos precision:', nltk.metrics.precision(refsets['1'],
68              taggedsets['1'])
69      print 'pos recall:', nltk.metrics.recall(refsets['1'],
70              taggedsets['1'])
71      print 'pos F-measure:', nltk.metrics.f_measure(refsets['1'],
72              taggedsets['1'])
73      print 'neg precision:', nltk.metrics.precision(refsets['0'],
74              taggedsets['0'])
75      print 'neg recall:', nltk.metrics.recall(refsets['0'],
76              taggedsets['0'])
77      print 'neg F-measure:', nltk.metrics.f_measure(refsets['0'],
78              taggedsets['0'])
79      print("\n")
80      print("informative features are: {}"
81              .format(classifier.show_most_informative_features()))
82
83  if __name__ == '__main__':
84      classify()
```