rijksuniversiteit groningen
faculteit der letteren

Event Detective

Detectie en verrijking van gebeurtenissen op Twitter

Bachelorscriptie Informatiekunde

David de Kleer

| 5 juni 2015 |

Inhoudsopgave

Voorwoord

TODO

Samenvatting

In deze scriptie presenteer ik een state-of the art benadering voor event detection op Twitter. Hiervoor hebben we (Chris Pool en ik) gebruik gemaakt van een methode voor spatio-temporal clustering, waarbij we tweets clusteren binnen een bepaalde GeoHash en tijdsinterval.

1 Inleiding

TODO

2 Theoretisch kader

Het onderwerp van deze scriptie valt binnen de wetenschappelijke literatuur onder het thema *event detection* (in sociale media), waarbij dit thema op zijn beurt weer is in te passen binnen *Information Retrieval*. Volgens Manning et al. is Information Retrieval *"het vinden van ongestructureerd materiaal in grote informatiecollecties, waarbij dit materiaal tegemoet komt aan een informatiebehoefte"*. In event detection willen we binnen een grote collectie ongestructureerde data (in ons systeem gaat het om tweets) gebeurtenissen of *events* detecteren om mensen van events in de wereld op de hoogte te kunnen houden (de informatiebehoefte).

De oorsprong van event detection is het door DARPA[1] gesponsorde TDT-programma, oftewel *Topic Detection and Tracking* (Atefeh & Khreich 2013). Het idee achter TDT was om de grote hoeveelheid informatie afkomstig uit verschillende nieuwsbronnen (in tekstvorm) op te delen in samenhangende nieuwsberichten, ontwikkelingen in reeds bestaande *news events* te detecteren en nieuwe news events te ontdekken (Allan 2002). Het TDT-programma bestond dus oorspronkelijk uit respectivelijk drie taken: de *segmentatie*, *detectie* en *opsporing* van events (Atefeh & Khreich 2013). Het doel hiervan was om gebruikers inzicht te bieden in (de ontwikkeling van) nieuwe en interessante gebeurtenissen die op de wereld plaatsvinden (Allan 2002).

Een systeem dat in staat is om event detection uit te voeren moet dus volgens TDT nieuwe of oudere, niet eerder geïdentificeerde events kunnen ontdekken. Event detection kan voor TDT dus uit respectievelijk twee taken bestaan: *on-line* (dus *real-time*) *detection* en *retrospective detection* (Yang et al. 1998). Het systeem dat wordt beschreven in deze scriptie valt onder retrospective detection, omdat we events in tweets uit het verleden willen ontdekken. Na enige aanpassing zou het mogelijk zijn om ons systeem ook on-line detection te laten uitvoeren, door te zorgen dat er constant input van de Twitter stream kan worden geanalyseerd. Hieruit blijkt dus dat een systeem voor event detection zowel een on-line als een retrospective component zou kunnen hebben.

Atefeh & Khreich geven aan dat het wat betreft de types events die kunnen worden gedetecteerd kan gaan om *specified* of *unspecified* events. Wanneer een systeem specified events detecteert, is het type event dat wordt gedetecteerd door het systeem bekend, of gaat het om een gepland event (Atefeh & Khreich 2013). Voorbeelden uit de wetenschappelijke literatuur zijn systemen voor de detectie van aardbevingen (Sakaki et al. 2010), concerten (Benson et al. 2011) en festivals (Lee & Sumiya 2010). Wanneer het gaat om de detectie van unspecified events is het onbekend wat de events die we willen detecteren precies inhouden, of in gaan houden. Dit is het geval in bijvoorbeeld het laatste nieuws, denk aan een ongeval of een onderwerp dat onverwachts veel aandacht krijgt. Voorbeelden uit de wetenschappelijke literatuur zijn systemen voor de detectie van news events (Sankaranarayanan et al. 2009), incidenten (Abel et al. 2012), of algemene (ook kleinschalige) events die in de wereld plaatsvinden (Walther & Kaisser 2013 en Becker et al. 2011). Omdat we in ons systeem de laatstgenoemde algemene events willen detecteren, richten we ons op de detectie van unspecified events.

Volgens Atefeh & Khreich bestaat event detection uit drie hoofdprocessen: *"data preprocessing, data representation and data organization or clustering"*. *Data preprocessing* heeft alles te maken met de voorbereiding van "rauwe" data voordat deze door een systeem kan worden gebruikt (denk aan het filteren van stopwoorden, tokenisatie van tekst en het kiezen van goede datastructuren). *Data representation* gaat om het selecteren van voor event detection nuttige eigenschappen binnen de voorbereide data, bijvoorbeeld dat twee berichten binnen een bepaald tijdsinterval zijn gepubliceerd en qua inhoud op elkaar lijken. *Data organization* heeft betrekking op het daadwer-

[1] Defense Advanced Research Projects Agency, het instituut voor onderzoek naar geavanceerde technologie van het Amerikaanse Ministerie van Defensie.

kelijk indelen of organiseren van berichten in events, waarbij een systeem een model (gebaseerd op regels of *machine learning*[2]) gebruikt om de gekozen datarepresentatie te organiseren. Deze eigenschappen kunnen soms vaker voorkomen: in ons systeem komt data organization bijvoorbeeld drie keer voor.

Onze aanpak van event detection vertoont overeenkomsten met de aanpak van Walther & Kaisser. We gebruiken hetzelfde achterliggende idee: wanneer tweets binnen een bepaald tijdsinterval en op dezelfde locatie qua inhoud gerelateerd zijn, zou het goed kunnen dat ze een event bespreken. Walther & Kaisser gebruiken daarnaast een aantal nuttige *features* (eigenschappen binnen de datarepresentatie) in hun event detection waarop onze features zijn geïnspireerd. Onze classificiatie is daarentegen niet *binair* (wel/geen event), maar *categorisch*, omdat we algemene events in categorieën (zoals *sport* of *bijeenkomst*) willen indelen. Daarnaast gebruiken we een andere representatie voor de locatie (geen radius rondom tweets, maar *GeoHash*[3], en probeer ik events te verrijken met tweets zonder geo-informatie en Chris Pool met named *entity recognition*.

3 Methode: van data tot detective

Nu volgt een beschrijving van de werking van ons systeem, van het verzamelen van de data tot de uiteindelijke detectie, verrijking en visualisatie van events. Het grootste gedeelte van deze processen wordt afgehandeld door een serie programma's in de programmeertaal Python, die in dit hoofdstuk behandeld zullen gaan worden. Wanneer er wordt verwezen naar een bestand (binnen een repository), is dit bestand te vinden in de git-repository `https://github.com/chrispool/Thesis/`. Als er geen uitleg over het gebruik van een programma wordt gegeven, is dit gewoon uit te voeren als `./programmaNaam.py`. Ik zal ten eerste beschrijven hoe we aan een van de meest fundamentele delen van ons systeem zijn gekomen: de data.

3.1 Verzamelen van data

We willen ons systeem trainen op retrospective data, dus op tweets uit het verleden. Onze tweet datasets zijn afkomstig van de Linuxserver *Karora* (`karora.let.rug.nl`) van de Rijksuniversiteit Groningen. Deze server filtert constant (grotendeels) Nederlandstalige tweets uit de Twitter stream. De gefilterde tweets worden samen met hun metadata per uur in gecomprimeerde (`.out.gz`) bestanden in de map `/net/corpora/twitter2/Tweets` (op Karora) geplaatst. Met het volgende commando is het nu mogelijk om bijvoorbeeld de tweets (inclusief alle metadata) van 27 maart 2015, om 3 uur 's middags, te verkrijgen.

```
$ zcat /net/corpora/twitter2/Tweets/2015/03/20150327:15.out.gz}
```

`zcat` is identiek aan `gunzip -c`, dit betekent dat het bestand met tweets gedecomprimeerd wordt en de gedecomprimeerde data wordt weggeschreven naar standard output.

Het programma `tweet2tab` (zie ook `scripts/tweet2tab` in de repository) is in staat om een aantal velden uit de zojuist verkregen gedecomprimeerde data te filteren en te scheiden met tabs, zodat deze velden eenvoudig door programma's kunnen worden ingelezen. Voor ons systeem hebben we tweets nodig die beschikken over de velden:

- `text`: de tekst van de tweet
- `coordinates`: breedte- en lengtegraad van de gebruiker toen de tweet werd gepost
- `user`: de naam van de gebruiker
- `date`: datum en tijd

Om deze velden te verkrijgen kan het volgende commando worden gebruikt op de output van het zojuist genoemde `zcat`-commando (op Karora):

[2]Binnen machine learning kunnen algoritmes worden gebruikt om patronen in een datarepresentatie te bepalen en aan de hand daarvan data te classificeren

[3]zie figuur 1 en de uitleg daarboven

```
/net/corpora/twitter2/tools/tweet2tab -i text coordinates user date
```

Dit is in feite alle tweet data die we nodig hebben voor verdere verwerking. Er is alleen nog een probleem: alle tweets worden meegenomen in het vorige commando, niet alleen de tweets die voorzien zijn van een locatie! Het formaat van een tweet zonder locatie is (`<tab>` is het scheidingsteken):

```
Kan niet slapen.<tab><tab>HOIIKBENMERLE<tab>2015-05-05 01:03:20 CEST Tue
```

Er is dus nog een derde stap nodig om alleen de tweets met locatie te verkrijgen: alle tweets die een leeg locatieveld hebben moeten worden overgeslagen. Dit kan zowel met een `grep`-commando als een klein script geschreven in Python (zie `scripts/get_geotweets` in de repository):

- `grep`: Gebruik de Perl reguliere expressie (optie -P) `^[^\t]+\t[^\t]+`. Deze reguliere expressie is waar wanneer er aan het begin van een regel 1 of meerdere keren een karakter staat dat geen tab is, vervolgens 1 keer een karakter dat wel een tab is en daarna weer 1 of meerdere keren een karakter dat geen tab is.
- `get_geotweets.py`: splitst alle regels in standard input op tabs en kijkt of het tweede element niet leeg is. De tweet wordt geprint als dit het geval is.

Nu volgen twee voorbeeldcommando's die alle commando's combineren en laten zien hoe alle tweets van 27 maart op Karora kunnen worden verzameld in het bestand `march27.txt`.

Met `get_geotweets.py`:

```
$ zcat /net/corpora/twitter2/Tweets/2015/03/20150327???.out.gz |
/net/corpora/twitter2/tools/tweet2tab -i text coordinates user date |
python3 get_geotweets.py > march27.txt
```

Met `grep`:

```
$ zcat /net/corpora/twitter2/Tweets/2015/03/20150327???.out.gz |
/net/corpora/twitter2/tools/tweet2tab -k text coordinates user date |
grep -P "^[^\t]+\t[^\t]+" > march27.txt
```

Het formaat van de verzamelde tweets is nu (`<tab>` is het scheidingsteken):

```
Ik moet slapen <tab>4.584649 51.854845<tab>tundraful<tab>2015-03-27 00:01:17 CET Fri
```

3.2 `EventCandidates` - Generatie van event candidates

Net als Walther, et al. willen we in ons systeem *event candidates* verzamelen: groepen of clusters van tweets die een gebeurtenis zouden kunnen zijn. Hiervoor groeperen we tweets die binnen een bepaalde tijd en plaats gepost zijn. We maken dus eigenlijk gebruik van *spatiotemporal clustering*: *"a process of grouping objects based on their spatial and temporal similarity"* (Kisilevich, et al.).

Om event candidates te kunnen verzamelen is het van belang om eerst de verzamelde tweets en hun metadata om te zetten in een geschikte datastructuur in Python, de taak van de `TweetPreprocessor` (3.2.1). Spatiotemporal clustering vindt in ons systeem plaats in twee stappen: de `ClusterCreator` (3.2.2) maakt clusters van tweets binnen een bepaalde locatie en een bepaald tijdsinterval, de `ClusterMerger` (3.2.3) voegt sommige gevonden clusters samen en selecteert event candidates. Het idee achter de `ClusterCreator` en de `ClusterMerger` lijkt op het idee achter de `ClusterCreator` en de `ClusterUpdater` van Walther, et al.

Het programma `EventCandidates` accepteert als argumenten een bestand met tweets die gegenereerd zijn door een van de laatste twee commando's in sectie 3.1, en de naam van de dataset met event candidates die het programma moet generen.

```
use: ./eventCandidates.py tweetfile datasetname
```

3.2.1 `TweetPreprocessor` – Tweets representeren in Python

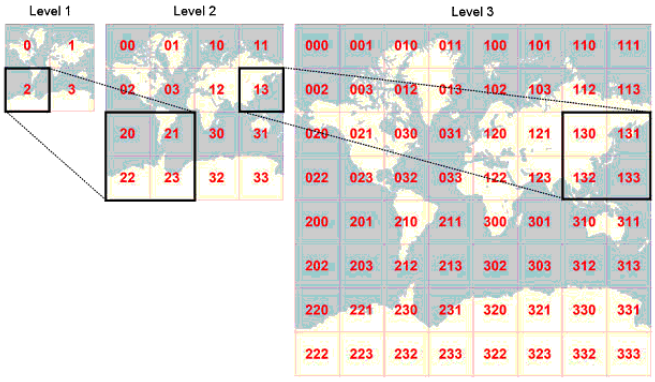
We hebben ervoor gekozen om tweets op te slaan als Python `dictionaries` (in een `list`), waarbij de `keys` strings zijn en de `values` de verzamelde tweet (meta)data in de vorm van strings. Een *tweet dictionary* ziet er nu dus zo uit:

```
tweetDict = {"text" : "tekst van tweet", "lon" : "lengtegraad", "lat" : "breedtegraad",
"user" : "gebruikersnaam", "localTime" : "tijd in UTC"}
```

Het is belangrijk om in deze stap alvast na te denken over de volgende stap: hoe gaan we tweets clusteren op locatie en tijd, en wat is daarbij een handige en efficiënte representatie van deze clusters? Hoe locatie- en tijdsdata in tweet dictionaries worden opgeslagen beïnvloedt immers hoe we deze data kunnen clusteren!

De **tijd** van tweets wordt standaard weergegeven als `jaar-maand-dagnummer uur:minuut:seconde tijdszone dag`. Wanneer uit deze string `jaar-maand-dagnummer uur:minuut:seconde` wordt gefilterd is het mogelijk tijd en datum met behulp van de time- en datetime-module in ***Unix Time*** om te zetten, oftewel het aantal seconden dat is verstreken sinds 1 januari 1970. Hierdoor is het mogelijk om te rekenen met seconden wanneer tijden van tweets met elkaar vergeleken moeten worden, wat eenvoudiger is dan rekenen met een datum of tijd. Er kunnen direct wiskundige operatoren worden toegepast op tijden in Unix Time (omdat het integers zijn) en event candidates kunnen onder één enkel getal worden opgeslagen. Bezwaar tegen deze aanpak zou betrekking kunnen hebben op het geval wanneer tweets van twee verschillende events op dezelfde plaats op exact dezelfde seconde gepost worden. Wanneer dit het geval is, overschrijft één van beide events het andere event. Omdat dit waarschijnlijk niet vaak voor zal komen, nemen we dit risico.

De **locatie** van tweets wordt standaard weergegeven als een coördinaat (lengte- en breedtegraad). Een representatie van locatie-informatie waarin coordinaten binnen hetzelfde gebied onder dezelfde identifier kunnen worden geplaatst is ***GeoHash***. Dit systeem deelt de wereld als het ware op in "hokjes" in een raster, waarbij een locatie dus niet meer bestaat uit een specifieke coordinaat, maar uit een specifiek hokje (dat dus meerdere coordinaten omvat). Ieder hokje heeft een alfanumerieke code, waarbij de precisie van het hokje hoger wordt (en daarmee het betreffende gebied op aarde kleiner) wanneer de code langer is. Figuur 1 geeft weer hoe dit ongeveer werkt.



Figuur 1: GeoHash, de wereld binnen een raster

We hebben gekozen om een GeoHash van precisie 7 (7 alfanumerieke karakters) te gebruiken. Hierbij is de grootte van de hokjes in het raster kleiner of gelijk aan 173x173 meter[4]. We maken voor de omzetting van lengte- en breedtegraden naar GeoHashes gebruik van de Python-module

[4] `http://www.movable-type.co.uk/scripts/geohash.html`

`python-geohash`[5].

Nu het probleem van locatie en tijd is opgelost, is het handig om te kijken naar de **tokenisatie** van de tekst van een tweet. Wanneer we bijvoorbeeld de tekst van tweets met elkaar willen vergelijken komt het van pas om beschikking te hebben over een lijst met tokens die in deze tweets voorkomen. Om tokens te kunnen extraheren is het nodig om woordgrenzen te vinden, waarvoor we een simpele tokenizer hebben geschreven die gebaseerd is op de volgende reguliere expressie: `[^a-zA-Z0-9#@]+`. Alle tekens in een string waarvoor niet geldt dat ze 1 of meerdere keren `a-z`, `A-9`, `0-9`, `#`, of `@` bevatten, worden vervangen door een spatie (met behulp van `pattern.sub` in de re-module van Python). We hebben besloten om vóór het toepassen van de reguliere expressie links uit tweets te verwijderen (we willen puur te kijken naar de tekst). Wanneer er vervolgens een lijst van woorden zonder links overblijft, filteren we stopwoorden uit tweets met behulp van een lijst van stopwoorden die is samengesteld door de Nederlandse stopwoorden van de `nltk`-module[6] van Python te combineren met een lijst van stopwoorden op internet (zie `corpus/stopwords.txt` in de repository).

Als we nu locatie, tijd en tokenisatie toevoegen aan het bestaande tweet dictionary ziet deze er als volgt uit:

```
tweetDict = {"text" : "tekst van tweet", "tokens" : "getokeniseerde tekst",
"lon" : "lengtegraad", "lat" : "breedtegraad", "user" : "gebruikersnaam",
"localTime" : "datum en tijd", "unixTime" : "Unix Time in seconden",
"geoHash" : "GeoHash behorend bij lengte- en breedtegraad van tweet"}
```

3.2.2 `ClusterCreator` – Tweets clusteren op GeoHash en tijd

We moeten nu op basis van de gegenereerde tweet dictionaries tijd- en locatieclusters in een datastructuur zetten die we in het verdere systeem zullen gaan gebruiken. Omdat locaties met tijden gebruikt kunnen worden als (redelijk) unieke identifiers, zijn ze in combinatie geschikt als keys voor een dictionary. Er kunnen op verschillende tijden gebeurtenissen plaatsvinden binnen dezelfde locatie, dus is een logische en efficiënte representatie van clusters een dictionary met als keys de locaties (GeoHash) en als values dictionaries met als keys tijden (Unix Time) en als value een lijst van tweet dictionaries.

Merk hierbij op dat lijsten van tweet dictionaries binnen onze gekozen representatie clusters of *candidate clusters* zijn. Candidate clusters zijn clusters van tweets die binnen een bepaalde GeoHash en een bepaald tijdsinterval gepost zijn. Een candidate cluster kan blijven groeien in (tweet) omvang zolang dit cluster "in leven is", wat betekent dat er een nieuwe tweet wordt gepost binnen de tijd van de laatste tweet op de locatie van het `cluster + 60 minuten`.

De structuur van een candidate cluster dictionary is dus als volgt:

```
clusters = { "GeoHash 1" : { "Unix Time laatste_tweet" : [ tweetDict1, ...], ...}, ...}
```

3.2.3 `ClusterMerger` – Clusters samenvoegen en event candidates selecteren

De clusters die we hebben verzameld vallen binnen een vrij klein gebied (kleiner of gelijk aan 173x173 meter), wat niet voldoende is voor events waarvan de deelnemers over grotere gebieden zijn verspreid. Daarom hebben we drie stappen aan het clusterproces toegevoegd die zorgen dat clusters worden samengevoegd met andere clusters in de buurt wanneer ze qua locatie (**stap 1**), tijd (**stap 2**) en inhoud (**stap 3**) overlappen. Dit werkt als volgt:

- **Stap 1 - Overlap op locatie**: Omdat locaties bestaan uit GeoHashes of "hokjes", is het mogelijk om voor deze hokjes te bepalen wat de "buurhokjes" zijn. Dit zijn de 8 hokjes die ieder hokje kunnen omringen (zie figuur 2). Buren van een GeoHash zijn

[5] `https://code.google.com/p/python-geohash/` (zie ook `modules/geohash.py` in de repository)
[6] `http://www.nltk.org/`

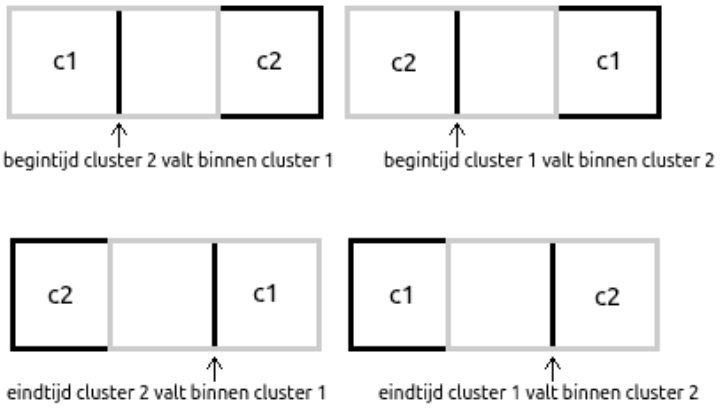
eenvoudig te vinden door de `neigbors`-methode van de `geohash`-module te gebruiken: `geohash.neigbors(GeoHash)`.

Figuur 2: Een GeoHash van precisie 5 met zijn 8 buren

Een goede vraag om nu te stellen is: wanneer stop je met het kijken naar buren? Het is immers mogelijk om voor de buren van een GeoHash weer te kijken naar hun buren, voor die buren weer, etcetera... We hebben ervoor gekozen om maar een keer de buren van een GeoHash op te zoeken, omdat een gebied van zo'n 400x400 meter ons een goed formaat leek voor een gemiddeld event.

We kunnen nu de clusters in alle buren (wanneer aanwezig) van een GeoHash bijlangs gaan om te kijken of ze qua tijd en inhoud overlappen met een cluster in de oorspronkelijke GeoHash.

- **Stap 2 - Overlap in tijd**: We kijken of clusters in tijd overlappen door de begin- en eindtijd van een cluster in de oorspronkelijke GeoHash met de begin- en eindtijden van clusters in een naburige GeoHash te vergelijken. Figuur 3 laat zien hoe in slechts vier vergelijkingen te bepalen is of twee clusters in tijd overlappen.



Figuur 3: Tijdsoverlap van twee clusters in vier vergelijkingen

- **Stap 3 – Overlap op inhoud**: Om te kunnen zien of twee clusters op inhoud overlappen bepalen we per cluster de 10 woorden met de hoogste *tf-idf score*. De tf-idf score wordt berekend door eerst te tellen hoe vaak een woord in een cluster voorkomt. Deze uitkomst wordt vermenigvuldigd met het logaritme van de totale hoeveelheid clusters gedeeld door

gedeeld door in hoeveel clusters een woord voorkomt (*idf*, zie figuur 4). idf is een compensatie voor woorden die vaak in veel documenten voorkomen, denk hierbij aan stopwoorden.

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Figuur 4: Berekening van idf voor een term t in documenten D

Omdat na deze stappen de clusters zijn samengevoegd kunnen we nu event candidates gaan selecteren. Een samengevoegde cluster is in ons systeem een event candidate wanneer er minstens **twee tweets** in staan van minstens **twee gebruikers**. Deze event candidates worden door `EventCandidates` met behulp van de `json`-module[7] opgeslagen als `data/datasetnaam/EventCandidates.json`.

Het verschil tussen candidate clusters en event candidates is dus dat:

- *candidate clusters* (`ClusterCreator`) alleen aan de voorwaarde moeten voldoen dat ze tweets binnen dezelfde GeoHash en hetzelfde tijdsinterval bevatten
- *event candidates* (`ClusterMerger`) kunnen **bestaan** uit (samengevoegde) candidate clusters afkomstig uit een GeoHash **en** zijn buren, waarbij een voorwaarde is dat ze minstens twee tweets en twee gebruikers bevatten

3.3 Annotatie van event candidates

We willen events uit event candidates filteren met behulp van *supervised learning*. Binnen supervised learning is een leeralgoritme in staat om patronen vinden in gelabelde data, en op deze manier zelf in staat om labels aan nieuwe data toe te kennen. Om een leeralgoritme voor supervised learning toe te kunnen passen moeten we dus als eerste de door `EventCandidates` (3.2) gevonden event candidates voorzien van een label. Dit gebeurt met behulp van de `Annotator` (3.3.1). We gebruiken de `Annotator` om allebei dezelfde dataset te annoteren, zodat we objectiever labels toe kunnen kennen aan event candidates. De evaluatie van onze annotatie en het opschonen van event candidates op basis van de evaluatie vindt plaats in `AnnotationEvaluation` (3.3.2).

3.3.1 `Annotator`: Interactieve annotatie van event candidates

De annotatie van event candidates is een interactief proces. Eerst moet er een door EventCandidates gegenereerd dataset met event candidates worden ingelezen. Daarna worden alle event candidates één voor één weergegeven en kan een annotator voor iedere event candidate een label toekennen. We hebben besloten om event candidates niet binair te labelen (event/geen event), maar om categorieën toe te kennen aan events. Het gaat hierbij om de volgende 5 categorieën:

1. `geen_event`: geen event
2. `sport`: events die betrekking hebben op sport
3. `entertainment`: concerten, festivals, theater, tv-shows...
4. `bijeenkomst`: bijeenkomsten van mensen rondom een bepaald (gespreks)motief
5. `incident`: ongelukken, branden, inbraken, vermissingen...
6. `anders`: event past niet in een andere categorie

[7] Met `json` kunnen datastructuren in Python eenvoudig in leesbaar formaat worden opgeslagen en later worden gebruikt door andere programma's, zie ook `https://docs.python.org/3/library/json.html`

De datastructuur waarin annotaties worden opgeslagen lijkt erg veel op de datastructuur waarin event candidates zijn opgeslagen. In feite zijn de lijsten met tweet dictionaries vervangen door nummers van event types:

```
clusters = {geoHash : { unix_time_laatste_tweet : categorie_nummer, ... }, ... }
```

De `Annotator` accepteert als argument de naam van een annotator, waardoor voor een `json`-bestand met annotaties (dit bestand staat voor iedere annotator in `data/datasetnaam/annotation_Name-judge.json`) kan worden geïdentificeerd wie de annotator was, en annotaties op deze manier een unieke naam kunnen krijgen.

```
use: Annotator.py Name-judge
```

3.3.2 `AnnotationEvaluation`: Annotatie evalueren en event candidates opschonen

Omdat we event candidates door twee annotatoren laten annoteren, kunnen we voor annotaties de *interannotator agreement* bepalen – de mate waarin twee annotatoren het eens zijn – met behulp van de *Kappa-score*, zie figuur 5. Hierin staat $Pr(a)$ voor de hoeveelheid gevallen waarin annotatoren het met elkaar eens zijn en $Pr(e)$ voor de kans dat ze het eens zijn.

$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)}$$

Figuur 5: Berekening van de kappa-score

`AnnotationEvaluation` vraagt als input een dataset met annotaties van twee annotatoren. Daarna bereiden we voor beide annotatoren twee lijsten voor, waarvan in lijst 1 de labels 0 t/m 5 (kappa-score voor categorieën) staan en in lijst 2 de labels 0 en 1 (kappa-score voor event of geen event). Deze lijsten staan voor beide annotatoren in dezelfde volgorde, waarna door een eigen implementatie van formule 2 de Kappa-score voor categorieën en event/geen event wordt berekend, met daarbij een *confusion matrix* voor categorieën en de *accuracy*. De laatste twee statistieken kunnen we enigszins vergelijken met de prestaties van ons systeem (als mensen het al in een bepaald percentage van de gevallen niet eens zijn, is het moeilijk om van een kunstmatig systeem te verwachten dat het deze gevallen goed identificeert).

Wanneer we het in onze annotatie oneens waren over de categorie van een event candidate wordt deze event candidate samen met de bijbehorende annotatie weggegooid. De overblijvende event candidates en annotaties worden in `data/datasetnaam/sanitizedEventCandidates.json` en `data/datasetnaam/sanitizedAnnotation.json` opgeslagen. Dit is onze *ground truth*, de *gold standard data* waaraan een leeralgoritme zich moet gaan aanpassen (Kobielus 2014).

3.4 Trainen van categorie en event classifiers

Omdat we met behulp van de annotatieprogramma's van sectie 3.3 train- en testdata kunnen annoteren, kunnen we na het annotatieproces *classifiers* gaan trainen. Een *classifier* is in supervised learning een functie die met behulp van een leeralgoritme automatisch klassen/labels toekent aan documenten, op basis van training op geannoteerde data (Manning et al. 2008). Om classifiers te kunnen genereren moeten we eerste een verzameling eigenschappen van event candidates identificeren. Daarin moeten onze classifiers patronen kunnen vinden die kunnnen helpen bij de identificatie van labels van (nieuwe) event candidates. Om deze reden moet de verzameling eigenschappen waarop we de classifiers trainen dus worden geëxtraheerd uit alle event candidates waarop we onze classifiers willen toepassen. De extractie van deze eigenschappen of features is de taak van de `FeatureSelector` (3.4.1). Het daadwerkelijke trainen, testen en evalueren van classifiers gebaseerd op verschillende leeralgoritmen vindt plaats in de `ClassifierCreator` (3.4.2).

3.4.1 `FeatureSelector`: Extractie van features uit event candidates

Als we willen dat onze classifiers goed presteren, is het van belang om een nuttige verzameling features van event candidates te bepalen. We hebben voor onze experimenten 9 features geïdentificeerd:

- `wordOverlapUser`: Berekent de overlap van woorden tussen gebruikers binnen een event candidate, waarbij de score hoger wordt wanneer overlappende woorden een hoge df-waarde hebben. In hoe meer documenten per gebruiker een woord staat, des te hoger de score wordt, waarbij hashtags een bonus krijgen omdat ze een gemeenschappelijk thema of onderwerp aan kunnen geven. De score wordt vermenigvuldigd met de hoeveelheid gebruikers, gedeeld door de hoeveelheid tweets binnen de event candidate, en afgerond op 0.5 zodat de kans op overlap met andere event candidates groter is.
- `wordOverlapSimple`: Simpele score voor woordoverlap: de hoeveelheid woorden die overlappen voor elke tweet binnen een event candidate (hashtags krijgen wederom een bonus) wordt gedeeld door de hoeveelheid tweets binnen deze event candidate en afgerond op 0.5.
- `wordOverlap`: Lijkt vrij veel op `wordOverlapSimple`, het verschil is dat de overlaptellingen van woorden worden vermenigvuldigd met de idf-scores van deze woorden binnen een event candidate.
- `location`: De gemiddelde locatie van de gebruikers in een event candidate wordt bepaald door het (*Cartesisiaans*[8]) gemiddelde van de lengte- en breedtegraden van de tweets binnen de event candidate te berekenen. Dit coördinaat wordt omgezet naar een GeoHash van precisie 6 met behulp van de `geohash`-module: `geohash.encode(breedtegraad, lengtegraad, precisie)`.
- `uniqueUsers`: De hoeveelheid unieke gebruikers binnen een event candidate.
- `nTweets`: De hoeveelheid tweets binnen een event candidate.
- `wordFeatures`: Binaire features die aangeven of de top n woorden met de hoogste tf-idf score wel of niet in een event candidate voorkomen. Deze word features gebruiken we om een classifier voor event candidate categorieën (*category classifier*) te trainen.
- `category`: In welke categorie een getrainde category classifier een event candidate indeelt.

Wanneer een programma de features uit een event candidate wil extraheren, is het mogelijk om `getFeatures(candidate, features)` te gebruiken. `candidate` is hierin een event candidate, `features` is een lijst van de zojuist genoemde features in de vorm van strings. Deze methode levert een dictionary op, met als keys de namen van features en als values de waarden van deze features voor een event candidate.

3.4.2 `ClassifierCreator`: Training en evaluatie van classifiers

Om classifiers te kunnen trainen moeten eerst de geannoteerde data en bijbehorende event candidates worden ingelezen in een geschikte representatie. Welke data precies wordt ingelezen hangt af van de *modus* waarin de ClassifierCreator wordt gebruikt. Er zijn twee modi, namelijk **DEVTEST** en **TEST**:

- **DEVTEST modus**: De input bestaat uit één enkel geannoteerd dataset. Deze dataset wordt opgesplitst door een willekeurige *80/20 split* (80% traindata en 20% testdata) toe te passen. De DEVTEST modus doorloopt tien iteraties waarin classifiers worden getraind en geëvalueerd op de willekeurig opgesplitste testdata: *cross-validation*.

[8] Officieel moet er bij berekening van een gemiddelde coördinaat rekening worden gehouden met de kromming van het aardoppervlak, anders ontstaat er een afwijking in het uiteindelijke coördinaat. Omdat het bij een event candidate gaat om een klein vlak op aarde zal de afwijking van dit coördinaat waarschijnlijk niet zo groot zijn, dus berekenen we gewoon het "standaard" gemiddelde van de coördinaten (net alsof ze punten in een Cartesiaans coördinatenstelsel zijn: een platte aarde!)

- **TEST modus**: De input bestaat uit twee geannoteerde datasets, namelijk een train- en een testdataset. Deze modus doorloopt één iteratie, waarbij classifiers getraind kunnen worden op de gehele dataset van de DEVTEST modus en getest op een nieuwe dataset die ons programma nooit eerder heeft "gezien". Features kunnen voor een DEVTEST namelijk helemaal worden geoptimaliseerd, waarbij de keuze van features misschien helemaal niet geschikt is voor toepassing op een nieuwe dataset: dit komt naar voren in de "echte" test. Dit geeft een eerlijkere indicatie van de prestaties van het systeem.

De train- en testdata worden in het programma opgeslagen als een lijst van tuples, waarbij de tuples bestaan uit een event candidate (herinnering: dit was een lijst van tweet dictionaries) en het bijbehorende label afkomstig van de annotaties. De structuur is dus als volgt:

```
dataset = [ ([{"text" : "tweet text", ...}, ...], "bijbehorend label"), ...]
```

Nu de train- en testdata in een geschikte representatie zijn opgeslagen, kunnen we classifiers trainen. We hebben ervoor gekozen om gebruik te maken van twee classifiers in ons systeem:

- `categoryClassifier`: De eerste classifier die het systeem traint probeert op basis van woorden in een event candidate de categorie (zie 3.3.1 voor de labels) van een event candidate te achterhalen. Voor de training extraheren we eerst de `wordFeatures` (3.4.1) uit de train- en testdata. Features hebben de volgende structuur:

  ```
  features = [( {"wordFeatures": ["word1" : False, ...]}, "bijbehorend label"), ...]
  ```

  De `nltk`-module kan classifiers trainen met dergelijke lijsten van `tuples` met `feature dictionaries` en labels. De classifier die we voor categorieen gebruiken is een *Naive Bayes* classifier van de python-module `scikit-learn`[9], die kan worden getraind met behulp van een eenvoudige wrapper van de `nltk`-module: `SklearnClassifier`. Deze wrapper kan worden gebruikt om `scikit-learn` classifiers te trainen die data in hetzelfde formaat als `nltk`-classifiers accepteren. De reden dat we niet de standaard Naive Bayes implementatie van `nltk` gebruiken is dat de implementatie van `scikit-learn` sneller blijkt te werken en vrijwel identieke resultaten oplevert.

- `eventClassifier`: Voor de tweede classifier extraheren we wederom features (3.4.1) uit event candidates, die op dezelfde manier worden opgeslagen als voor de eerste classifier. De `eventClassifier` gebruikt de door de `categoryClassifier` toegekende labels met daarbij andere features (zoals `wordOverlapUser` of `location`) om nogmaals te bepalen onder welke categorie een event thuishoort. We hebben besloten om voor deze classifier te kiezen voor de Naive Bayes classifier van `nltk`.

De `categoryClassifier` is te zien als een soort basisclassifier die alleen *word features* gebruikt om event candidates in categorieën in te delen. De `eventClassifier` past *finetuning* toe op de categorielabels van de `categoryClassifier`, met behulp van features die meer betrekking hebben op de *metadata* van event candidates. We willen hiermee een conceptuele scheiding aangeven.

Wanneer beide classifiers zijn getraind berekent het programma een aantal statistieken voor de evaluatie. Deze statistieken omvatten de *precision*, *recall*, *f-score*, *accuracy* en *confusion matrix* voor iedere training van de classifiers (iteratie). De statistieken worden met behulp van de `metrics` van `nltk` bepaald en een aantal daarvan met behulp van de `tabulate`-module (`zie modules/tabulate.py`) in een tabel gezet.

Als het programma in de TEST modus wordt uitgevoerd worden de (laatst) getrainde classifiers opgeslagen als binaire bestanden in `data/datasetnaam/eventClassifier.bin` en `data/datasetnaam/categoryClassifier.bin`, met behulp van de pickle-module[10]. `pickle` is vergelijkbaar

[9] `http://scikit-learn.org`
[10] `https://docs.python.org/3/library/pickle.html`

met tttjson, alleen worden bestanden in binair in plaats van leesbaar formaat opgeslagen. Het doel van beide modules is gelijkwaardig, namelijk *serialisatie*[11].

3.5 Detectie, verrijking en visualisatie van events

De in 3.4 getrainde classifiers kunnen nu worden toegepast om events te detecteren op basis van nieuwe event candidates. Dit is het proces dat het aan ons systeem gelijknamige (hoofd)programma `EventDetective` (3.5.1) uitvoert. De verrijking van events door middel van tweets zonder geo-informatie vindt plaats in `EventDetectiveChart` (3.5.2). De bestanden in de map `/vis/map` (3.5.3) handelen de visualisatie van events af. `EventDetective` en `EventDetectiveChart` hebben een visualisatiecomponent (`generateMarkers`, zie 3.5.3) omdat ze de input voor de visualisatie moeten geven.

3.5.1 `EventDetective` – detectie van events

De naam `EventDetective` is afkomstig van event detection, het centrale thema van het onderzoek onderliggend aan het onderwerp van deze scriptie, en een soort *"detective-metafoor"*. Een detective (*classifier*) krijgt vaak een grote hoeveelheid bewijsmateriaal (*event candidates met geëxtraheerde features*) waaruit hij moet bepalen welke bewijzen leiden naar de juiste conclusie, de oorzaak van het misdrijf (*of een event candidate wel of geen event is*).

Om events te kunnen detecteren vereist de `EventDetective` als input eerst een dataset met classifiers en vervolgens een dataset met nieuwe event candidates, waarin events moeten worden gedetecteerd. Vervolgens wordt de FeatureSelector (3.4.1) toegepast om features voor de `category-` en de `eventClassifier` uit de nieuwe event candidates te halen. Beide classifiers bepalen daarna een categorielabel voor de event candidates. Wanneer ze onder een categorie vallen die niet gelijk is aan `geen_event` worden ze aan een lijst met events toegevoegd.

3.5.2 `EventDetectiveChart` – verrijking van events

`EventDetectiveChart` erft over van `EventDetective` en bevat qua detectie van events dezelfde funcionaliteit. De extra stap die in dit programma plaatsvindt is de toevoeging van tweets zonder geo-informatie. Omdat er niet meer zoveel tijd was voor een uitgebreide extra componenent heb ik voor deze stap een *Proof of Concept* of versimpeld prototype ontwikkeld.

Het is van belang dat tweets zonder geo-informatie vóór volledige processing met behulp van *heuristieken* kunnen worden gescand op overeenkomst met een event, omdat we hiervoor *alle* Nederlandse tweets in een periode moeten scannen. Op **de dag** 5 mei 2015 zijn er bijvoorbeeld 760000 tweets gepost (100 MB), waarbij de hoeveelheid tweets over **de hele maand** maart ter vergelijking ongeveer 570000 tweets (79 MB) bedraagt. Wanneer een maand aan tweets zonder geo-informatie moet worden gescand, gaat het al gauw om ongeveer 10 GB(!) aan tweets.

Een schaalbare benadering van dit probleem is om eerst de belangrijkste woorden uit een event te verzamelen. Onder belangrijke woorden versta ik simpelweg de n woorden met de hoogste *df-waarden* (in hoeveel tweets een woord voorkomt) binnen een event, waarbij termen met hashtags (belangrijk) dubbel worden geteld. Vervolgens kunnen steeds alle tweets binnen het tijdsinterval van een event worden ingelezen (bijvoorbeeld de tweets van die dag of van de uren waarin het event nog "in leven was"). In een echte toepassing is het van belang dat ingelezen tweets na verwerking steeds worden uitgelezen om te voorkomen dat het geheugen helemaal volloopt.

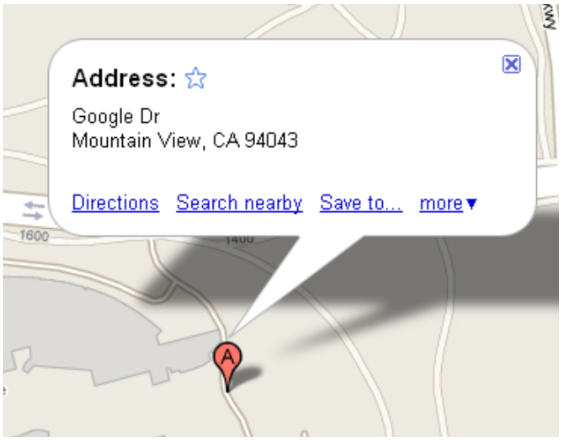
Een eenvoudige heuristiek die aangeeft dat tweets verder moeten worden verwerkt is wanneer ze twee van de top drie woorden uit de lijst van woorden met de hoogste df-waarden van een event bevatten. Een woord uit die top drie mag in een tweet voorkomen als `" woord"` (let op de spatie!), `"@woord"` of `"#woord"`. Op deze manier is het zelfs niet eens nodig om de velden van de tweet te splitsen om ze apart in te kunnen lezen. De velden van tweets die genoeg woorden bevatten om verder te worden verwerkt worden wel gesplitst en in een tweet dictionary

[11] datastructuren of objecten opslaan zodat ze later weer kunnen worden opgebouwd

gezet. Hiervoor was een aanpassing aan de `TweetPreprocessor` (3.2.1) nodig, omdat de tweet data zonder geo-informatie het veld voor locatie mist. Eventuele stappen die hierna nog plaats zouden kunnen vinden is het berekenen van een *similarity score* met de tweets in een event, maar hier ben ik verder niet meer aan toegekomen.

3.5.3 `generateMarkers` en `vis/map/` – visualisatie van events

Om events te visualiseren en de resultaten van ons systeem te kunnen presenteren hebben we gebruik gemaakt van *markers* met een `InfoWindow` op een *Google Map*, zie figuur 6.



Figuur 6: Voorbeeld van een Google Map Marker met een InfoWindow

De inhoud van het `InfoWindow` bestaat uit *de tweets die horen bij een event, samen met hun metadata (tijd, gebruiker, locatie)*, in chronologische volgorde. Om de coordinaten van een marker te bepalen maken we net als voor de `location`-feature (3.4.1) gebruik van het *Cartesiaans gemiddelde van de lengte- en breedtegraden* van de tweets binnen een cluster. We hebben de *categorie* van een event nodig omdat we voor elke categorie een uniek icoon willen gebruiken, zodat gelijk aan een marker te identificeren is om welk type event het gaat (zie figuur 7). Al deze data wordt door de `EventDetective` (3.5.1) in `generateMarkers` opgeslagen als een variabele in een Javascriptbestand (`vis/map/js/markers.js`) met het formaat:

```
locations = [['tweet tekst en metadata', gemiddeldeBreedtegraad, gemiddeldeLengtegraad,
    'categorielabel'], ...]
```

Elke lijst binnen deze `locations`-lijst is een event dat met behulp van `vis/map/map.html` kan worden gevisualiseerd. De werking van `vis/map/map.html` is vrij vanzelfsprekend, er wordt een Google Map gemaakt met behulp van de Google Maps API en markers (met iconen per categorie) worden gegenereerd aan de hand van de lijsten in locations.

Figuur 7: Marker iconen: entertainment, incident, bijeenkomst, anders, sport

Naast de standaard visualisatie heb ik nog een extra visualisatie-optie aan het systeem toegevoegd: een Google map met staafdiagram, met behulp van het *jQuery framework* van `HighCharts`[12]. Het idee hierachter is als volgt: wanneer events worden verrijkt met tweets zonder

[12] `http://www.highcharts.com/products/highcharts`

geo-informatie, en er wordt bijvoorbeeld een belangrijke voetbalwedstrijd gevonden is het makkelijk om overspoeld te raken door de hoeveelheid tweets bij een dergelijk event. Daarom leek het me handig om tweets te groeperen per twee minuten[13] en in een staafdiagram te laten zien, waarbij maximaal 10 tweets uit die periode worden weergegeven wanneer de muiscursor op een staaf wordt gezet. Op deze manier is het eenvoudig om trends in grote hoeveelheden tweets te ontdekken.

`EventDetectiveChart` genereert met behulp van zijn (*overridden*) `generateMarkers` dezelfde data als de `EventDetective`, met daarbij als toevoeging de *x- en y-coordinaten* (x is de tijd, y de hoeveelheid tweets binnen twee minuten) van het staafdiagram, de *titel van het event* in de vorm van de drie woorden met de hoogste df-waarde en de *ondertitel* in de vorm van de dag waarop het event plaatsvond. De in 3.5.2 gevonden tweets zonder locatie-informatie worden **dikgedrukt** weergegeven in de `InfoWindow`s bij markers op de kaart. De structuur van de eerder genoemde `locations`-lijst is afgezien van de toevoeging van deze data precies hetzelfde.

De visualisatie van events met hun bijbehorende staafdiagrammen vindt plaats in `vis/map/map_chart.html`. Het verschil met `vis/map/chart.html` is dat de pagina nu in twee delen wordt verdeeld, waarbij aan de linkerkant de Google Map met markers wordt weergegeven en aan de rechterkant de `HighCharts`-grafiek die hoort bij een aangeklikte marker.

4 Experimenten, resultaten en evaluatie

Ik zal nu de met onze `EventDetective`-software uitgevoerde experimenten – inclusief verkregen resultaten en evaluatie – beschrijven.

4.1 Data verzamelen event candidates genereren

Voor onze gezamenlijke event detection experimenten hebben we gebruik gemaakt van twee geannoteerde datasets (om te trainen en te testen, zie de TEST-modus in 3.4.2). De dataset waarop we hebben getraind bestaat uit alle 566549 tweets met geo-informatie van maart 2015 (`/corpus/march.txt`). Onze testset bestaat uit de 165848 tweets met geo-informatie van de tweede helft van april 2015 (`/corpus/april.txt`). Beide datasets zijn verzameld zoals is beschreven in 3.1. We hebben `EventCandidates` (3.2) gebruikt om de event candidates uit beide datasets te extraheren (zie `data/dataset/EventCandidates.json` voor de trainset en `data/testset/EventCandidates.json` voor de testset).

4.2 Het annotatieproces

We hebben voor onze train- en testset allebei met behulp van de `Annotator` (3.3.1) respectievelijk 1350 en 500 event candidates geannoteerd. Vervolgens heeft `AnnotationEvaluation` (3.3.2) de kappa-score en de confusion matrices van onze annotaties bepaald, zie tabel 1 en 2. Bij elke confusion matrix worden de kappa-score voor categorieën (*categoryKappa*), voor event/geen event (*eventKappa*) en de *accuracy* (hoe vaak we het eens waren) gegeven.

	anders	**bijeenk**	**entertain**	**geen_event**	**incident**	**sport**
anders	<1>	.	.	2	1	1
bijeenk	21	<207>	7	25	.	2
entertain	3	.	<20>	5	.	.
geen_event	14	27	18	<619>	9	9
incident	1	2	.	12	<178>	.
sport	1	.	.	5	.	<60>

Tabel 1: Trainset - categoryKappa: 0.79, eventKappa: 0.8, accuracy: 0.868

[13] Dit is mogelijk door de Unix Time van tweets te delen door 120, af te ronden en weer vermenigvuldigen met 120. Deze afgeronde Unix Times kunnen als keys in een dictionary met als values lijsten van tweet dictionaries worden gebruikt om tweets op twee minuten te groeperen.

	anders	bijeenk	entertain	geen_event	incident	sport
anders	<.>	.	.	.	1	.
bijeenk	4	<110>	13	9	.	3
entertain	.	.	<8>	2	.	.
geen_event	3	19	4	<199>	4	2
incident	1	.	.	.	<78>	.
sport	.	1	2	2	.	<60>

Tabel 2: Testset - categoryKappa: 0.82, eventKappa: 0.8, accuracy: 0.86

We hebben dus voor zowel train- als testset kappa-scores van 0.8 behaald. Volgens Manning et al. is een agreement boven de 0.8 "goed", dus we kunnen zeggen dat er hier sprake is van een behoorlijk goede annotator agreement.

Naast de generatie van statistieken met betrekking tot interannotator agreement, heeft `AnnotationEvaluation` de event candidates met bijbehorende annotatie waarvan we het over de categorieën oneens waren gefilterd, en onze annotatie samengevoegd. De opgeschoonde event candidates en annotatie zijn voor de traindata te vinden onder `data/dataset/sanitizedAnnotation.json` en `data/dataset/sanitizedEventCandidates.json` (1085 candidates), voor de testdata onder `data/testdata/sanitizedAnnotation.json` en `data/testdata/sanitizedEventCandidates.json` (430 candidates).

Uit de confusion matrices blijkt dat we in onze annotatie niet altijd overeenstemming hebben bereikt over of een event candidate wel of geen event is. Er waren bijvoorbeeld twijfelachtige gevallen als "is een opening een bijeenkomst?" of "is de beschrijving van een debat op televisie een event?". De categorie `anders` zorgt ook voor wat onduidelijkheid en zou in de toekomst eventueel weggelaten kunnen worden. Ook in de categorie `entertainment` zijn we het niet altijd eens, maar dit wordt wellicht veroorzaakt omdat we erg brede categorieën gebruiken. Ondanks deze bezwaren waren we het toch in 86% van de gevallen met elkaar eens.

4.3 Detectie van events: Trainen en testen van classifiers

We hebben de `ClassifierCreator` (3.4.2) gebruikt om classifiers te trainen en de prestaties van ons systeem te beoordelen. We gebruiken in ons systeem twee classifiers voor event detection:

- `categoryClassifier`: Een classifier met alleen word features, die op basis van woorden event candidates in de in 3.3.1 genoemde categorieën indeelt. We hebben empirisch bepaald dat deze classifier het beste werkt wanneer we voor de word features de 800 woorden met de hoogste tf-idf score selecteren, en we als classifier Naive Bayes gebruiken.
- `eventClassifier`: Een classifier die de categorielabels van de eerste classifier verrijkt met features gebaseerd op tweet metadata (zie 3.4.1 voor alle features). We hebben empirisch bepaald dat deze classifier het beste werkt wanneer we `category` van de `categoryClassifier` combineren met `wordOverlapUser`, `wordOverlapSimple` en `location`, en we als classifier Naive Bayes gebruiken.

We hebben in de DEVTEST een gemiddelde accuracy van 89% behaald, waarvoor we 10 keer cross-validation hebben toegepast. We hebben gekozen om *accuracy* als eindcijfer van ons systeem te gebruiken, zodat we een eenduidige score voor de classificatie konden geven. De accuracy is te berekenen door de hoeveelheid goed geclassificeerde event candidates te delen door de totale hoeveelheid geclassificeerde event candidates. Voor de resultaten van de DEVTEST met de beste features (inclusief *precision*, *recall* en *f1-score* per categorie), zie tabel 6 in de bijlage (6).

Bij de TEST (`./ClassifierCreator -test`) hebben we met 3 verschillende classifiers getest: *Naive Bayes*, *Maximum Entropy* en *SVM*[14], zie tabel 3. De classifiers presteerden allemaal net iets boven de 80%, maar Naive Bayes presteerde het beste, met een accuracy van 84%. In tabel 4 staat de confusion matrix van de Naive Bayes classifier.

[14] *Support Vector Machine*

	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

Tabel 3: Resultaten van de TEST met de vier beste 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>

Tabel 4: Confusion matrix voor de Naive Bayes classifier

Omdat we individuele bijdrage van de verschillende features in de TEST wilden beoordelen, hebben we afzonderlijk met deze features getest, zie tabel 5 (en tabel 7 in de bijlage (6) voor de bijbehorende precision, recall en f1-score per categorie). In tabel 5 staan ook de *baseline performance* (iedere event candidate is geen event) en de *upper bound* (menselijke (onze) prestaties op een event dection taak) gebaseerd op de accuracy van onze annotatie.

	Accuracy
All features	0.84
Category	0.81
Location	0.51
WordOverlapSimple	0.63
WordOverlapUser	0.6
WordOverlapUser, Category	0.82
Baseline Performance	0.46
Upper Bound	0.86

Tabel 5: Afzonderlijke features, baseline performance en upper bound

Uit tabel 5 blijkt dat word features verreweg de meeste invloed hadden op de prestatie van de classifier, omdat deze dan al tot een accuracy van 81% komt. De `wordOverlap`-features dragen veel minder bij: de accuracy ligt daarmee nog zo rond de 60%. De `location`-feature maakt nog minder verschil, want de accuracy is met deze feature maar 5% hoger dan de baseline. Over deze feature is ook discussie mogelijk: op dezelfde plaats kunnen immers verschillende categorieën events plaatsvinden.

In vergelijking met de baseline en de upper bound presteert ons systeem goed. De accuracy voor alle features is met 84% namelijk 38% beter dan de baseline en slechts 2% slechter dan de upper bound. Omdat Walther & Kaisser een vergelijkbaar systeem hebben gebouwd ligt het voor de hand om ons systeem met hun systeem te vergelijken. Het lijkt erop dat zij alleen cijfers presenteren voor events die `real-world events` zijn, oftewel de categorie `wel_event`: alle categorieën behalve *geen_ event* van ons systeem samengenomen. Voor deze categorie krijgen we binnen ons systeem een f1-score van 88%, terwijl Walther & Kaisser een f1-score van 84% behaalden (zie

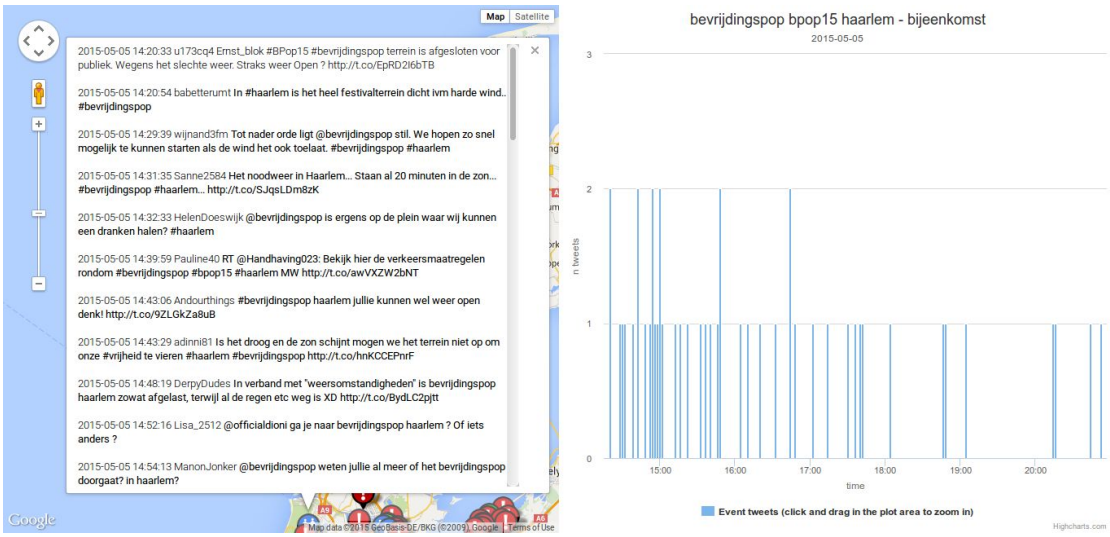
de code in comments bij "`uncomment voor wel_event`" in de `ClassifierCreator`). Hierbij is wel belangrijk om op te merken dat wij het wat betreft events in de categorie `incident` vrij makkelijk hebben, omdat meldingen van incidenten vaak behoorlijk op elkaar lijken. Daarom zijn ze makkelijk te identificeren met behulp van word features.

4.4 Visualisatie en verrijking van events

Een beschrijving van de resultaten van ons systeem zou natuurlijk niet compleet zijn zonder de visualisaties van `vis/map/` toe te voegen. Nadat de `EventDetective` (3.5.1) de door de `ClassifierCreator` gegenereerde classifiers heeft toegepast op een dataset met event candidates, worden de gedetecteerde events in een lijst in Javascript-formaat opgeslagen (`vis/map/js/markers.js`). Wanneer nu `vis/map/map.html` wordt geopend, is figuur 8 het resultaat: een Google Map met markers voor events.

Figuur 8: Visualisatie van events in april 2015 door vis/map/map.html

`EventDetectiveChart` (3.5.2) kan events verrijken met tweets zonder geo-informatie en voegt (in samenwerking met `vis/map/map_chart.html`) een staafdiagram toe aan de visualisatie. Figuur 9 geeft een voorbeeld van een verrijkt event met staafdiagram. Tweets zonder geo-informatie zijn dikgedrukt.



Figuur 9: Voorbeeld van een verrijkt event met bijbehorend staafdiagram

5 Conclusie

TODO

#	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
1	0.89	0.90	0.93	0.91	0.80	0.75	0.77	0.00	0.00	0.00	0.84	0.81	0.83	0.97	1.00	0.98	0.00	0.00	0.00
2	0.87	0.93	0.89	0.91	0.80	0.80	0.80	0.00	0.00	0.00	0.72	0.85	0.78	0.90	0.95	0.92	0.00	0.00	0.00
3	0.91	0.90	0.96	0.92	0.67	0.60	0.63	0.00	0.00	0.00	0.93	0.84	0.88	0.97	1.00	0.99	0.00	0.00	0.00
4	0.86	0.86	0.92	0.89	0.93	0.72	0.81	1.00	0.17	0.29	0.72	0.78	0.75	0.97	0.92	0.94	0.00	0.00	0.00
5	0.91	0.93	0.95	0.94	0.62	0.71	0.67	0.00	0.00	0.00	0.88	0.81	0.85	0.97	0.97	0.97	0.00	0.00	0.00
6	0.9	0.95	0.90	0.92	0.80	0.89	0.84	0.00	0.00	0.00	0.80	0.88	0.83	0.89	0.97	0.93	0.00	0.00	0.00
7	0.93	0.95	0.92	0.94	0.68	0.94	0.79	0.00	0.00	0.00	0.92	0.94	0.93	1.00	0.95	0.97	0.00	0.00	0.00
8	0.9	0.91	0.92	0.91	0.71	0.91	0.80	0.67	0.50	0.57	0.86	0.84	0.85	1.00	0.95	0.97	0.00	0.00	0.00
9	0.87	0.91	0.90	0.91	0.78	0.70	0.74	0.33	0.25	0.29	0.71	0.79	0.75	0.97	0.94	0.96	0.00	0.00	0.00
10	0.88	0.90	0.90	0.90	0.57	0.57	0.57	0.50	0.50	0.50	0.80	0.83	0.81	1.00	0.93	0.96	0.00	0.00	0.00
Avg.	0.89	0.91	0.92	0.92	0.74	0.76	0.74	0.25	0.14	0.16	0.82	0.84	0.83	0.96	0.96	0.96	0.00	0.00	0.00

Tabel 6: Resultaten van de DEVTEST met de features category, wordOverlapUser, wordOverlapSimple en location

	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

Tabel 7: Resultaten van de TEST voor de beste features afzonderlijk

Referenties

Abel, F., Hauff, C., Houben, G.-J., Stronkman, R. & Tao, K. (2012). Twitcident: fighting fire with information from social web streams. In *Proceedings of the 21st international conference companion on world wide web* (pp. 305–308).

Allan, J. (2002). Introduction to topic detection and tracking. In *Topic detection and tracking* (pp. 1–2). Springer.

Atefeh, F. & Khreich, W. (2013). A survey of techniques for event detection in twitter. *Computational Intelligence*.

Becker, H., Naaman, M. & Gravano, L. (2011). Beyond trending topics: Real-world event identification on twitter. *ICWSM*, *11*, 438–441.

Benson, E., Haghighi, A. & Barzilay, R. (2011). Event discovery in social media feeds. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1* (pp. 389–398).

Kobielus, J. (2014). *The ground truth in agile machine learning.* Verkregen van `http://ibmdatamag.com/2014/06/the-ground-truth-in-agile-machine-learning/`

Lee, R. & Sumiya, K. (2010). Measuring geographical regularities of crowd behaviors for twitter-based geo-social event detection. In *Proceedings of the 2nd acm sigspatial international workshop on location based social networks* (pp. 1–10).

Manning, C. D., Raghavan, P. & Schütze, H. (2008). *Introduction to information retrieval* (Dl. 1). Cambridge university press Cambridge.

Sakaki, T., Okazaki, M. & Matsuo, Y. (2010). Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on world wide web* (pp. 851–860).

Sankaranarayanan, J., Samet, H., Teitler, B. E., Lieberman, M. D. & Sperling, J. (2009). Twitterstand: news in tweets. In *Proceedings of the 17th acm sigspatial international conference on advances in geographic information systems* (pp. 42–51).

Walther, M. & Kaisser, M. (2013). Geo-spatial event detection in the twitter stream. In *Advances in information retrieval* (pp. 356–367). Springer.

Yang, Y., Pierce, T. & Carbonell, J. (1998). A study of retrospective and on-line event detection. In *Proceedings of the 21st annual international acm sigir conference on research and development in information retrieval* (pp. 28–36).