rijksuniversiteit groningen
faculteit der letteren

Event Detective

Detectie en verrijking van gebeurtenissen op Twitter

Bachelorscriptie Informatiekunde

David de Kleer

1 juni 2015

Inhoudsopgave

Voorwoord

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ut lobortis justo, id rhoncus libero. Sed tincidunt odio eget nulla faucibus facilisis. Vivamus egestas mauris sit amet vehicula laoreet. Aenean commodo pharetra turpis ac euismod. Integer tristique eu tortor at dignissim. Vestibulum libero purus, tincidunt sed mollis eu, dapibus eu tellus. Quisque dolor eros, cursus sed diam vel, fringilla lobortis mauris. Phasellus vel arcu ac nisl vestibulum venenatis pellentesque eget lacus. In mauris ex, convallis vitae neque venenatis, pretium mattis mi. Nulla laoreet egestas enim, porttitor fringilla enim vehicula vitae. Nunc rhoncus ligula risus, quis gravida quam dapibus eu. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

Curabitur sollicitudin fringilla elementum. Vestibulum tincidunt volutpat ligula, a lobortis massa eleifend ac. Duis sit amet libero purus. Donec sed dictum nunc. Sed vel erat eu urna ullamcorper aliquam. Morbi vitae volutpat tellus. Quisque justo dolor, semper eu consectetur ut, mattis at nunc. Aliquam id tellus rhoncus, lobortis velit sed, tempus erat.

Nunc a quam sed nunc blandit dignissim. Aliquam pharetra orci ex, eu dictum nibh aliquam ut. Maecenas quis accumsan felis, a fermentum purus. Vivamus dignissim odio eu est malesuada, a suscipit enim accumsan. Proin sit amet hendrerit velit. Praesent sem elit, finibus in ligula ut, eleifend vulputate orci. Praesent varius cursus purus. Ut enim lectus, blandit ut eros rutrum, aliquam varius ipsum. Vestibulum vel orci venenatis, ornare ligula vitae, egestas tortor. Vestibulum ut tincidunt nulla. Phasellus nec ligula id mi pulvinar tincidunt. Pellentesque non lobortis velit.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Fusce ut massa ac metus bibendum ullamcorper sed eu risus. Vestibulum iaculis nunc velit, ut aliquam turpis consequat non. Sed finibus at eros congue facilisis. Phasellus eget nunc fermentum, aliquet mauris nec, tristique dui. Fusce interdum imperdiet sem, eu tempor mauris tincidunt ut. Praesent ut tellus molestie, scelerisque quam ullamcorper, malesuada dui. Fusce accumsan justo non lacinia rhoncus.

Praesent tristique molestie nibh, eget dapibus ipsum consequat vitae. Duis at magna a felis mollis mollis a viverra libero. Nam lobortis gravida consequat. Praesent faucibus dui nulla, a vehicula elit convallis vel. Morbi nec nisl sed ante consectetur tincidunt sollicitudin sit amet elit. Morbi at sollicitudin ex. Nulla ullamcorper, lorem faucibus malesuada tristique, purus turpis sagittis augue, sit amet condimentum est quam quis quam.

Samenvatting

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ut lobortis justo, id rhoncus libero. Sed tincidunt odio eget nulla faucibus facilisis. Vivamus egestas mauris sit amet vehicula laoreet. Aenean commodo pharetra turpis ac euismod. Integer tristique eu tortor at dignissim. Vestibulum libero purus, tincidunt sed mollis eu, dapibus eu tellus. Quisque dolor eros, cursus sed diam vel, fringilla lobortis mauris. Phasellus vel arcu ac nisl vestibulum venenatis pellentesque eget lacus. In mauris ex, convallis vitae neque venenatis, pretium mattis mi. Nulla laoreet egestas enim, porttitor fringilla enim vehicula vitae. Nunc rhoncus ligula risus, quis gravida quam dapibus eu. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

Curabitur sollicitudin fringilla elementum. Vestibulum tincidunt volutpat ligula, a lobortis massa eleifend ac. Duis sit amet libero purus. Donec sed dictum nunc. Sed vel erat eu urna ullamcorper aliquam. Morbi vitae volutpat tellus. Quisque justo dolor, semper eu consectetur ut, mattis at nunc. Aliquam id tellus rhoncus, lobortis velit sed, tempus erat.

Nunc a quam sed nunc blandit dignissim. Aliquam pharetra orci ex, eu dictum nibh aliquam ut. Maecenas quis accumsan felis, a fermentum purus. Vivamus dignissim odio eu est malesuada, a suscipit enim accumsan. Proin sit amet hendrerit velit. Praesent sem elit, finibus in ligula ut, eleifend vulputate orci. Praesent varius cursus purus. Ut enim lectus, blandit ut eros rutrum, aliquam varius ipsum. Vestibulum vel orci venenatis, ornare ligula vitae, egestas tortor. Vestibulum ut tincidunt nulla. Phasellus nec ligula id mi pulvinar tincidunt. Pellentesque non lobortis velit.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Fusce ut massa ac metus bibendum ullamcorper sed eu risus. Vestibulum iaculis nunc velit, ut aliquam turpis consequat non. Sed finibus at eros congue facilisis. Phasellus eget nunc fermentum, aliquet mauris nec, tristique dui. Fusce interdum imperdiet sem, eu tempor mauris tincidunt ut. Praesent ut tellus molestie, scelerisque quam ullamcorper, malesuada dui. Fusce accumsan justo non lacinia rhoncus.

Praesent tristique molestie nibh, eget dapibus ipsum consequat vitae. Duis at magna a felis mollis mollis a viverra libero. Nam lobortis gravida consequat. Praesent faucibus dui nulla, a vehicula elit convallis vel. Morbi nec nisl sed ante consectetur tincidunt sollicitudin sit amet elit. Morbi at sollicitudin ex. Nulla ullamcorper, lorem faucibus malesuada tristique, purus turpis sagittis augue, sit amet condimentum est quam quis quam.

1 Inleiding

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Fusce ut massa ac metus bibendum ullamcorper sed eu risus. Vestibulum iaculis nunc velit, ut aliquam turpis consequat non. Sed finibus at eros congue facilisis. Phasellus eget nunc fermentum, aliquet mauris nec, tristique dui. Fusce interdum imperdiet sem, eu tempor mauris tincidunt ut. Praesent ut tellus molestie, scelerisque quam ullamcorper, malesuada dui. Fusce accumsan justo non lacinia rhoncus.

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 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) 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.

Onze aanpak van event detection vertoont overeenkomsten met de aanpak van Walther & Kaisser. We gebruiken hetzelfde achterliggende idee: wanneer tweets binnen een bepaald tijdsinter-

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

val 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*[2] 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):

```
/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):

[2] Eigenschappen van in ons geval tweets, waarin een algoritme patronen moet ontdekken om een model te kunnen maken van wat een event is.
[3] zie figuur 1 en de uitleg daarboven

```
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

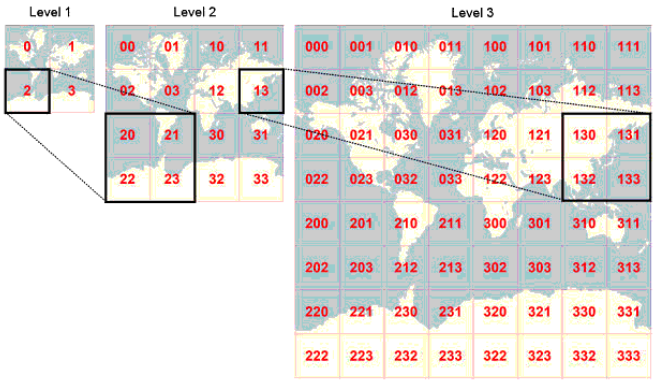
We hebben ervoor gekozen om tweets op te slaan als Python `dictionaries` (in een `list`), waarbij de `keys` strings zijn en de `values` de verzamelde tweet (meta)data in de vorm van strings. Een *tweet dictionary* ziet er nu dus zo uit:

```
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 `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

[4] `http://www.movable-type.co.uk/scripts/geohash.html`
[5] `https://code.google.com/p/python-geohash/` (zie ook `modules/geohash.py` in de repository)

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

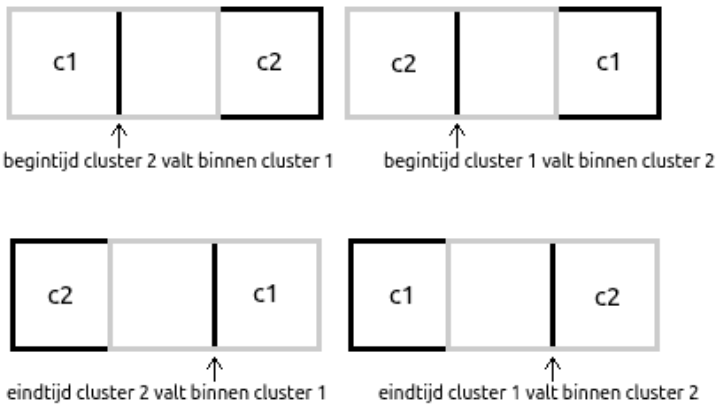
[6] `http://www.nltk.org/`

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 hoe vaak een woord in alle clusters 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`
2. `sport`
3. `entertainment`
4. `bijeenkomst`
5. `incident`
6. `anders`

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, ... }, ... }
```

[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 `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).

4 Resultaten en discussie

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Fusce ut massa ac metus bibendum ullamcorper sed eu risus. Vestibulum iaculis nunc velit, ut aliquam turpis consequat non. Sed finibus at eros congue facilisis. Phasellus eget nunc fermentum, aliquet mauris nec, tristique dui. Fusce interdum imperdiet sem, eu tempor mauris tincidunt ut.

Praesent ut tellus molestie, scelerisque quam ullamcorper, malesuada dui. Fusce accumsan justo non lacinia rhoncus.

5 Conclusie

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Fusce ut massa ac metus bibendum ullamcorper sed eu risus. Vestibulum iaculis nunc velit, ut aliquam turpis consequat non. Sed finibus at eros congue facilisis. Phasellus eget nunc fermentum, aliquet mauris nec, tristique dui. Fusce interdum imperdiet sem, eu tempor mauris tincidunt ut. Praesent ut tellus molestie, scelerisque quam ullamcorper, malesuada dui. Fusce accumsan justo non lacinia rhoncus.

Referenties

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).