NTNU – Trondheim
Norwegian University of
Science and Technology

TDT4215 Web Intelligence

Semantic Web Project: Emergency

Group 11:

Eirik Fosse
Sigurd Grøneng
Michael McMillan

April 14, 2016

`http://emergency.bustbyte.no`
`https://github.com/BustByte/emergency.bustbyte.no`

Abstract

With today's web there is a growing amount of unstructured information available for anyone with an Internet connection. Combined with an advancement in computational power it is possible to create systems that reason about huge amounts of data quickly.

"The share of Americans for whom Twitter [...] serve as a source of news is continuing to rise. This rise comes primarily from more current users encountering news there rather than large increases in the user base overall, according to findings from a new survey" [1]. However, one could argue that Twitter is not ideal when analyzing a lot of data because of its API limits [8].

This report describes how we created a system to let anyone easily discover what the Norwegian police are engaged in. Through an intuitive graphical interface users can select date ranges, types of artifacts, types of events and perform free text search on all tweets ever published by the Norwegian police.

As far as we know, there are no other systems where you can monitor the actions of the Norwegian police force legally. For instance, we believe that our system can be of great interest for media companies wanting to publish police related news as soon as they occur.

Contents

1 Introduction

In the following chapter we will give an introduction to the theme of our project and a brief description of our project idea.

1.1 Theme

For our project we have chosen to focus on semantics. We have built a queryable semantic network using Protégé. Protégé is "a free, open-source ontology editor and framework for building intelligent systems" [4]. Our semantic network holds data about artifacts and events expressed in tweets.

1.2 Project idea

The Norwegian Police Service is divided into 12 police districts. These districts collectively cover Metropolitan Norway [3]. Each Police District have a central headquarter (operasjonssentral) that is responsible for logging incoming messages and dispatching patrols.

Norwegian Emergency Departments have become increasingly interested in utilizing social media platforms for communicating with the public [11]. Each police district have their own dedicated Twitter [7] profile that they use to report emergencies.

These messages convey information about crimes, accidents, fires and so on. Our idea was to gather this information in real-time and make it queryable. We also wanted to pinpoint as many incidents as possible on a map, which is updated continuously as tweets are posted. The map should also contain filtering options, with options to filter on e.g. type of incident, where it happened, and when it happened.

2 Theory

In the following chapter we explain some of the theory that we have covered in our project.

2.1 Semantic web

The web lacks ways of querying with actual semantics, and not just keywords. This is the problem the semantic web is trying to solve. For example when someone queries for "highest mountain is the world" the semantic web tries to actually understand the meaning of the words and the relation between them instead of the matching keywords. The current way of doing this is using building a queryable semantic network with tools like RDFs and OWL.

2.2 Information extraction

Information extraction is the process of retrieving data from a semi structured or unstructured source automatically. Techniques of this process includes identifying identities, coreference resolution which is finding different words that refer to the same entity and relationship extraction which is identifying what kind relationship the different entities have [13].

In our case we were analysing tweets which were written in natural language and can therefore be seen as an unstructured data source. But at the same time, the tweets often followed some kind of loose structure. For example, some tweets started with a capitalized place name followed by a colon. This indicated that capitalized words followed by a colon was a potential place name.

2.2.1 Ontology

An ontology is a description of entities and the relation between them. In computer science it is used to "tell" computers how the relation between different concepts is defined and make a computer able to make assumptions based on the axioms of the ontology [14].

One can divide an ontology into two parts, `abox` and `tbox`. The `tbox` is all the facts about class concepts, object properties and data properties. For example we can define that a "hasDegree" relationship must have a subject that is a human and the object must be a educational degree. An `abox` contains all the assertions like "Sigurd" isTypeOfHuman, and "Sigurd" hasDegree "Art Bachelor".

2.2.2 RDF and OWL

RDF is a language for describing ontologies with "triples". A triple is a statement which consists of a subject, a predicate and an object. The subject and object are classes and the predicate is the relationship between the subject and the object. OWL is a layer on top of RDF which defines what you can write in order to have a valid ontology.

RDFs most important feature is the possibility to define subclasses, sub properties and the range and domain of different properties. OWL gives more expressiveness by defining classes by which values their data properties have, and by using set operators among other features. When writing an ontology in Protégé, an owl file using concepts from RDF, RDFs and OWL will be made.

2.2.3 SPARQL

SPARQL is a query language for RDF, which can edit and retrieve triples in an ontology. Queries are built using triples and the query engine match them against the triples in the ontology. SPARQL allows the user to easily combine selected data specified in the ontology for querying.

3 Tools

In the following chapter we will give a summary of tools we have used in our project.

3.1 Protégé

Protégé is an open source editor for ontologies. It provides a graphical user interface for creating, querying and editing OWL.

3.2 SQLite

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world [6].

3.3 WebSockets

WebSockets is an advanced technology that makes it possible to open an interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply [2].

3.4 Twitter API

The Twitter micro-blogging service includes two RESTful APIs. The Twitter REST API methods allow developers to access core Twitter data. This includes update timelines, status data, and user information. The Search API methods give developers methods to interact with Twitter Search and trends data [9].

3.5 Google Maps API

The Google Maps API allow for the embedding of Google Maps onto web pages, using a simple JavaScript interface or a Flash interface. It is designed to work on both mobile devices as well as traditional desktop browser applications [12].

4 Data sources

Our main data sources were Twitter, Kartverket and Politiet. In the following chapter, we explain how we have utilized these data sources.

4.1 Twitter

All our tweets were collected from Twitter. Twitter is a service for friends, family, and coworkers to communicate and stay connected through the exchange of quick, frequent messages [10].

In the last five years, police districts around Norway have tweeted information about crimes and other things engaging the police in their districts. The oldest tweet we have is from 22nd March 2011. We want to extend our gratitude to Daniel Dale Laabak for providing us with tweets by Norwegian police districts past the 3 200 limit set by Twitter.

The tweets from the police districts usually contain information about what happend, where it happend, and when it happened. We imported over 350 000 of these tweets with the goal of extracting its semantics, and will continue to store this information as tweets are published.

4.2 Kartverket

To increase the precision of the places extracted from each tweet, we cross referenced each "potential place" with real places in a data set called "SSR SOSI UTM33". This data set is freely available by the Norwegian Mapping and Cadastre Authority (Kartverket).

4.3 Politiet

The Ministry of Justice and Public Security recently decided to merge several Police Districts. As of 1. January 2016, Norway went from having 27 districts to 12 districts. We used Politiets official web page to determine which communes each district had jurisdiction over.

4.4 Unknown source: Communes

Luckily we found a comprehensive data set containing all the 428 communes in Norway along with polygons mapping their geographical boundaries. By Googling for "norske kommuner table" (accessed at 2016.02.20) a publicly available Google Fusion file can be downloaded. We downloaded the file as a CSV file and inserted each row into the database.

5 Methodology

The following chapter explains how the system works behind the scenes.

5.0.1 Determining the location

To geographically pinpoint the tweets location we run it through a series of algorithms. This process can be divided into four steps.

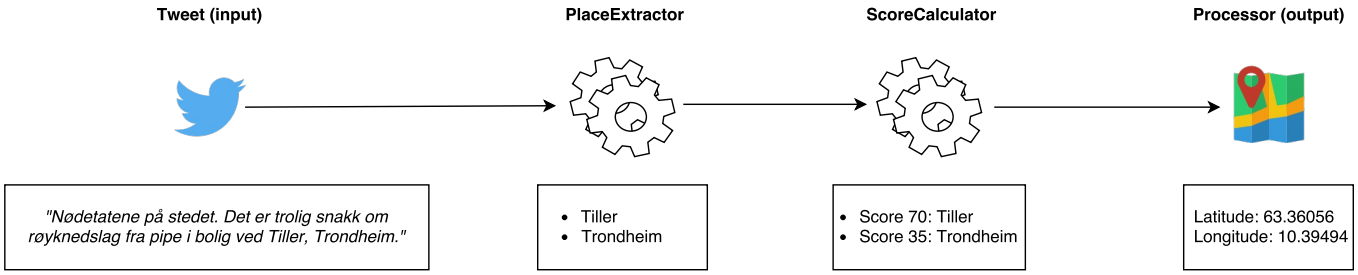

Figure 5.1: The algorithms determining the location

Natural Language Processing: First the tweet content is extracted and pushed into a `PlaceExtractor` that applies Natural Language Processing (NLP) techniques to find potential names of places in Norway. It looks for words where the first letter is capitalized, ignores words with numbers, excludes stop words and so on.

Ranking: Each potential place found in the previous step is then forwarded to a `ScoreCalculator` algorithm. The score is determined by looking at the words next to the place name. For instance, it rates places next to prepositions higher than those that are not. Places followed by a colon is also given a higher score. The score is cumulative and diminishing, meaning each score is incremented and decremented for each matched rule.

Geographic Information Retrieval: Thirdly all the potential places with their re-

spective score is fed to a `Processor`. The `Processor` iterates over each potential place ordered by their score from high to low. The processor looks for places within the district the tweet originated from to deduct which place the tweet is referring to. Figure 5.2 explains the relationships between the entities which enables us to provide a precise location for a tweet.

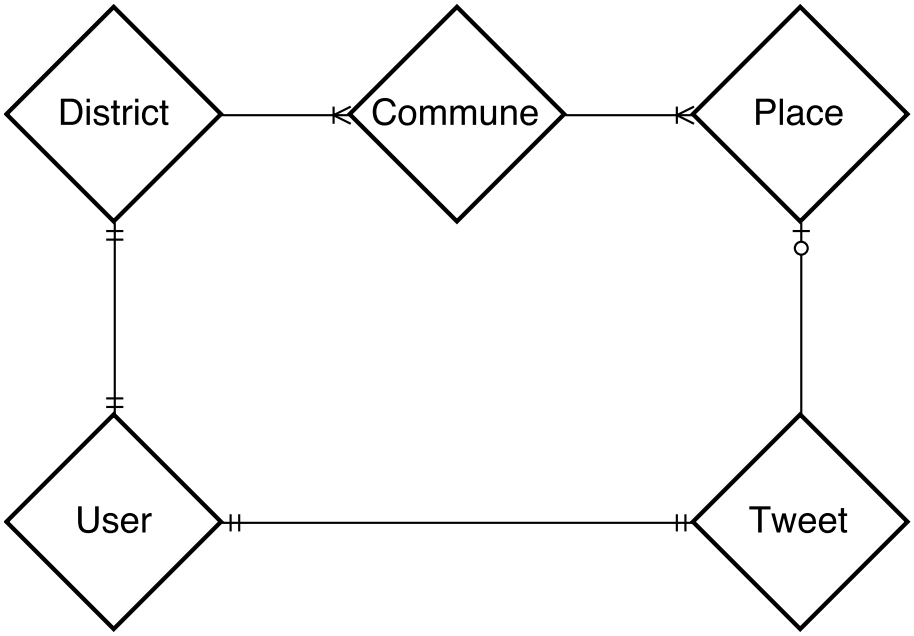

Figure 5.2: Entity Relation Diagram of all the tables

5.0.2 Determining artifacts and events

For determining artifacts and events we searched for keywords in the tweet related to the chosen topics. Sometimes there was a need to check for negations in the text to determine if it was "none-event" or an "almost-event". There were also specific rules for what could be after and before the words.

5.1 Architecture

The architecture of the system is divided into three modules: A web server that speaks WebSocket and HTTP written in Python 3, a Java (JDK 8.1) application interfacing with the ontology and a front end providing a graphical user interface.

When a new tweet is published by any of the Twitter users belonging to a police district, the web server receives the tweet on a WebSocket from Twitter. It then proceeds to process the tweet by pushing it to the ontology. A rough sketch of the architecture can be found in Figure 5.3.

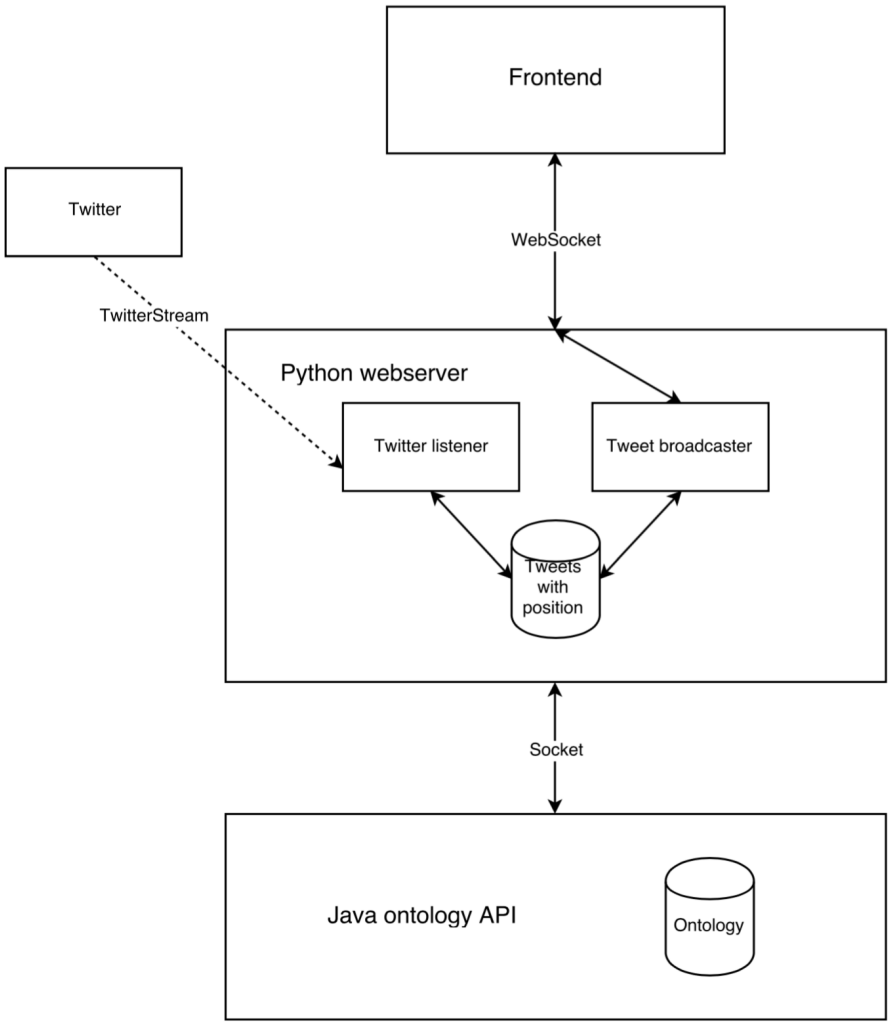

Figure 5.3: System architecture

6 Product description

Our product can be tested at `http://emergency.bustbyte.no`. Table 6.1 is showing some stats from our system.

Table 6.1: Stats

Number of unique tweets	353 609
Number of places	1 066 669
Number of tweets where a place was extracted	201 330
Drug related tweets	6 218
Fire related tweets	23 361
Tweets containing car accidents and alcohol	205
Average time between each tweet since 22 March 2011	7.69 minutes

6.1 How to use it

The frontend of the system has three different use cases. You can receive new tweets live, you can search for tweets freely, or you can combine category filters searching for tweets.

6.1.1 Live streaming from Twitter

The opening screen of the website is showing a simple map of Norway. When you enter this page you are automatically subscribed to a twitter feed running on the server. As new tweets are posted on Twitter, our server process the tweets, try to categorize them and analyze potential places mentioned in the tweet. After the tweets have been processed, they are broadcasted to all subscribed clients and placed on the map if the algorithms on the server were able to find a position. If the system were unable to find

a position, the tweet will still appear in a "list of tweets with unknown position" in the clients browser.

6.1.2 Text search

In the top right corner of the web page there is a search box, where you can provide a search query, a start date and an end date. Users can search freely in our database, but will only receive a fixed number of results. The search functionality is implemented using SQL's `LIKE` function. The search query is passed to the database, and tweets containing the search query are returned.

6.1.3 Ontology search

In the search box it is also possible to change to a category search. You can choose to search on an artifact, an event, or both combined. When our system receives a category query, we forward it directly to our ontology API. The ontology use the filters provided, and returns a list of tweets to the user.

7 Results

In this chapter we present results and discuss some of the problems we encountered.

7.1 Categorizing the tweets

With the categories we defined we could categorize around 40 percent of the tweets. The group thinks at least 90 percent can be categorized by defining more fine grained artifact and event topics, but this would make the system much more complex. It was not prioritized, and we believe a bigger ontology would cause slower response time when using the HermiT reasoner to ask for instances of a given class.

7.2 SPARQL performance issues

Initially we wanted to let the user define the relationships between the tweets, artifacts and events in the query and return results that matched. This would be available through a user interface. For example one might want to see where all the tweets about drunk driving accidents on weekends are. This can be written as shown in Listing 7.1

```
# Assuming the day of the week is represented by numbers 1-7
SELECT ?tweet WHERE {
    ?tweet hasDay ?day .
    ?tweet hasEvent ?trafficAccidentEvent .
    ?tweet hasEvent ?drunkDrivingEvent .
    ?trafficAccidentEvent rdf:type ont:trafficAccident .
    ?drunkDrivingEvent rdf:type ont:drunkDrivingEvent .
    FILTER( ?day > 4 )
}
```

Listing 7.1: SPARQL example query

When using a laptop with low hardware specifications, the queries we tested did not terminate within 5 minutes. We tried to open the ontology in Protégé for writing the queries, but the program crashed. There might be several reasons why this did not work efficiently. Maybe the ontology was made in a way that made querying very slow or the queries were written in a very inefficient way. According to an article about OWL and RDFs [5], there are some inferences that can be really slow and in the biggest ontologies they can even be uncomputable. Unfortunately we did not get to use this part in the final product.

7.3 The power of an ontology

Although we did not use the ontology for SPARQL queries, using an ontology can have some benefits over saving tweets in a relational database along with their categories. One can extract more information about the same tweets by adding new `tbox` data. For example we could define a new class `Drunk-Driving-Weekend-Accident` according to the query above, run the reasoner and then one could just ask for all the instances of this class.

7.4 Unknown true negatives

We are sure we could have categorized many more tweets by artifacts and events by mapping all the categories we could think of. Moreover, there is another problem with the mapping. We have no structured way of confirming that the tweets that actually are categorized have been so correctly. We have no test data set with correctly categorized tweets to validate against without going through all the tweets manually.

8 Further development

We found it really interesting, and learned a lot during this project. It was especially interesting to see how well our algorithms worked on the actual data set when determining locations. It turned out that they worked way better than first anticipated.

We are going to continue to develop the system, and we have some ideas for new features. An updated list of upcoming features can be found in our issue tracker at Github. A selection of things we want to implement as of today are:

- Combine free text search with category queries against the ontology.
- Expand the ontology, and maybe give returned tweets a relevance score.
- New tweets should automatically be added to the ontology, today we have to run a program manually.
- Identify the position of a tweet if it is linked to an older tweet where the position was found.
- Let journalists subscribe to certain searches so that they are notified when a new Tweet is published.
- Display some information about the live stream, text search and category search when you enter the web page.
- Show 10 latest tweets when you enter the web page.
- Refresh page/reconnect if WebSocket connection is lost.

Bibliography

[1] Pew Research Center. Social media and news survey. Technical report, 2015.

[2] MDN (Mozilla Developer Network). Websockets. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. [Accessed on April 12th 2016].

[3] Politiet. Nye politidistrikt fra 1. januar 2016. https://www.politi.no/oslo/Kampanje_226.xml. [Accessed on February 19th 2016].

[4] Protégé. About protégé. http://protege.stanford.edu/. [Accessed on February 19th 2016].

[5] Cambridge Semantics. Rdfs vs owl. http://www.cambridgesemantics.com/semantic-university/rdfs-vs-owl. [Accessed on April 12th 2016].

[6] SQLite. About sqlite. https://www.sqlite.org/about.html. [Accessed on April 12th 2016].

[7] Twitter. About twitter. https://about.twitter.com/. [Accessed on February 19th 2016].

[8] Twitter. Get statuses from user timeline. https://dev.twitter.com/rest/reference/get/statuses/user_timeline. [Accessed on April 14th 2016].

[9] Twitter. Rest apis. https://dev.twitter.com/rest/public. [Accessed on April 12th 2016].

[10] Twitter. Twitter faq. https://support.twitter.com/articles/13920. [Accessed on April 12th 2016].

[11] VG. Politiets morsomste twitter-meldinger utgis i bok. http://www.vg.no/rampelys/politiets-morsomste-twitter-meldinger-utgis-i-bok/a/10121826/. [Accessed on February 19th 2016].

[12] Programmable Web. Google maps api. http://www.programmableweb.com/api/google-maps. [Accessed on April 12th 2016].

[13] Wikipedia. Information extraction. https://en.wikipedia.org/wiki/Information_extraction. [Accessed on April 13th 2016].

[14] Wikipedia. Ontology. https://en.wikipedia.org/wiki/Ontology. [Accessed on April 14th 2016].

List of Acronyms

API Application Programming Interface.

CSV Comma-separated values.

HTTP Hypertext Transfer Protocol.

JDK Java Development Kit.

NLP Natural Language Processing.

OWL SPARQL Protocol and RDF Query Language.

RDF Resource Description Framework.

RDFs Resource Description Framework Schema.

REST REpresentational State Transfer.

SOSI Samordnet Opplegg for Stedfestet Informasjon.

SPARQL SPARQL Protocol and RDF Query Language.

SQL Structured Query Language.

SSR Sentralt stadnamnregister.

UTM33 Universal Transverse Mercator,.