Global Scientific JOURNALS

GSJ: Volume 9, Issue 8, August 2021, Online: ISSN 2320-9186
www.globalscientificjournal.com

A DEMONSTRATION OF APPLIED MACHINE LEARNING ON MULTI-SOURCE DATASETS FOR ANALYZING AND IMPROVING IN INCIDENTS MANAGEMENT AND RESPONSE TIME

Patrick Mukala [*1] and Godwill Ilunga [†2]

[1]*Faculty of Engineering, University of Johannesburg, South Africa. Email: *patrick.mukala@gmail.com*
[1]*Faculty of Science, Department of Mathematics and Computing ,Université Pédagogique Nationale, D.R. Congo*
[1,2]*Universit´e Nouveaux Horizons, Lubumbashi, D.R. Congo. Email: †ilunga.godwill@unhorizons.org*

KeyWords

Big Data, Data Mining, Clustering, Machine Learning, Text mining, Weather data, Web scrapping.

ABSTRACT

Various machine learning algorithms constitute vital solutions to big data processing. Data from various sources, in disparate formats can be integrated and seamlessly analyzed when the right tools and techniques are made use of. In this paper, we report on application called Respond! reporting tool. This tool was realised by ICT students from Fontys University of Applied Sciences as part of their graduating project from the Applied Data Science minor. The aim of the project was to develop a tool based on advanced machine learning techniques aimed at gathering information from a number of various different sources (news outlets and websites, social media and weather forecasting sources) in order to allow an automatic management of fire incidents in the Netherlands. The paper demonstrates the extensive research conducted on how to apply machine learning techniques on weather data. Making use of a combination of web scrapping tools, social media mining and a number of machine learning techniques such as support vector machines and regression modeling, the tool provides some viable results that can be used to monitor weather information.

Introduction

Big data technologies provide extensively useful means for processing large volumes of data that cannot be supported by traditional relational databases. While there are increasing interests in adopting and shifting to these new data processing paradigms, the landscape lacks clear guidelines on how to emarbk on such processes for scores of compagnies. This leads to a state of of somewhat panic and desiorientation to keep up with the hype without fully assessing what the most appropriate avenue would be for firms and organizations from different industries. Ekbia et al. in [2] report on a number of dilemmas that arise with the advent of Big Data. The purpose of their work was to shed light on the common underlying issues pertaining to the adoption of Big Data frameworks. We believe that a critical aspect of such dilemmas is lack of concrete cases and lessons learnt that could inspire other users on how the algorithms, tools and techniques fare on real cases.

In this paper, we tackle such an issue and report on a typical case that requires the wrangling of data from multiple data sources to provide some insights. It is in the context of applied data science that we hope this case will demonstrate how existing scientific knowledge can be put to use in order to solve practical problems. Similar undertakings can be mentioned from the literature. Amodei et al. Demonstrate the use of machine learning algorithms for speech recognition in both English and mandarin settings [1]. In

[4], the authors make use of process and data analytics methodologies to provide a practical example of how students' learning behaviours can be modeled and analyzed. In [8] , the author presents a case of Big Data in the advertising world.In [3],[5], [7],[6], salient scientific knowledge such as semantic search, abstract state machines, ontology engineering and process mining are applied on open source software repositories data to provide empiric evidence of learning patterns in these contexts.

In this paper, we describe a case conducted by ICT students at the Fontys University of Applied Sciences. The aim is to provide some insights into how students navigated a project requiring the use of tools and techniques popular in the Big Data realm. Making use of streaming data about fire incidents, social media comments and weather reports, this case explores how this wealth of data can be used to optimize the fire incidents response time.

The rest of the paper is structured as follows: in Section 2, we describe the case study and the problem that needed to be solved; in Section 3 we describe the research question and expected outcomes, in Section 4, we describe the approach, tools and techniques used to solve the research question, in Section 5 we discuss the tool and results obtained and in Section 6, we conclude the paper.

Case Description: Respond! Incident Reporting

The fire department in Brabant (Midden- and West- Brabant) is a public entity with a number of re- sponsibilities: preventing and fighting fires, giving assistance during accidents, educating and checking the preventive measures. In addition, the fire department spends a lot of time investigating the fires and other incidents. The fire department consists of full time workers and volunteers. Once an incident gets reported to one of the emergency centers and the fire department is required, a group of fire fighters are dispatched within a couple of minutes. While traveling to the incident, they decide how they should handle the situ- ation. After the fire the investigation team will collect information and evidence to establish the causes of the fire and learn from the decisions made.

In order to be able to improve their response time and be better prepared, they need to take advantage of the volumes of data related to incidents that they can collect. Respond! was therefore commissioned to help collect and analyze incidental data. Making use of the BRAIN (BRAndweer Informatie Netwerk) system, they can collect data from a wide array of sources but they still need to find ways to integrate these data before making sense of the contents. This constituted the trigger in the task assigned to students at Fontys.

A key part of the assignment (case) consists of collecting, cleaning and analyzing that data in a way that it is meaningful and helpful to the fire investigation team. Specifically, the goal was to collect and analyze the data that already exists and the data that is being generated every time an incident occurs. In the end, this data will be translated into a visual and friendly application that will be used by the fire department in the Netherlands.

Research Question and Expected Outcomes

In order to provide the solution to the case described above, it is critical to primarily define its scope based on a specific research question. The main question besides collecting data can be formulated as follows: "What information is relevant to a specific incident as it is reported and what patterns can be observed from this information?"

Although this question seems clear enough, the main expectations from students included providing a list of incidents, filtering abilities on incidents, specific information about incidents they are reported, some text analysis to decide weather the reported information is relevant, providing geographical data, opinion or reporting data from social media platforms, providing related weather information as well as news articles pertaining to that incident. In a nutshell, the information should incorporate:

- Observations from firefighters
- Relevant Facebook posts
- Relevant news articles from Google
- Pictures from news articles, tweets and Facebook posts
- A timeline of events regarding the incident
- Information from the dispatch central (GMK1)
- Interactive visualization of the weather information
- Etc.

The challenge in this work was to collect this information, parse it as much as possible and integrate these different versions of data relating to common incidents and give a summary of the critical information about the incidents including possible causes of fire incidents, victims, size of fire, people involved, building plan for where the fire occurred, the location where the fire started in the building, spread of the fire, the amount of water that was used on the incident, if external source of water was used, and what source was used, chemical storage information of the building etc. as shown in Figure 1

Approach, Tools and Techniques

The project was carried out following an agile methodology which included fixed time sprints. This provided some flexibility and also while reporting on the project progress, some more light could be shed with newer instructions they received in order to come up with the desired result.
A number of tools and techniques are still being developed and are widely available, but for this case, we mention a limited list of some of the tools including programming languages that were used to solve the problem and also given that students were expected to create a tool that encompasses the solution.

Tools

The project was conducted using primarily Python and Flask. Python as a programming language contains a rich repository of libraries for data analysis purposes. Specifically, Python contains libraries for creating web servers. Web servers contain quite some logic that we reused a lot, and therefore reducing the needs to recreate a new one from scratch. Flask was chosen as a web server framework because of its simplicity in allowing to add endpoints. These endpoints will become REST endpoints, emitting JSON data. These endpoints represent incidental information that will be used for visualization.

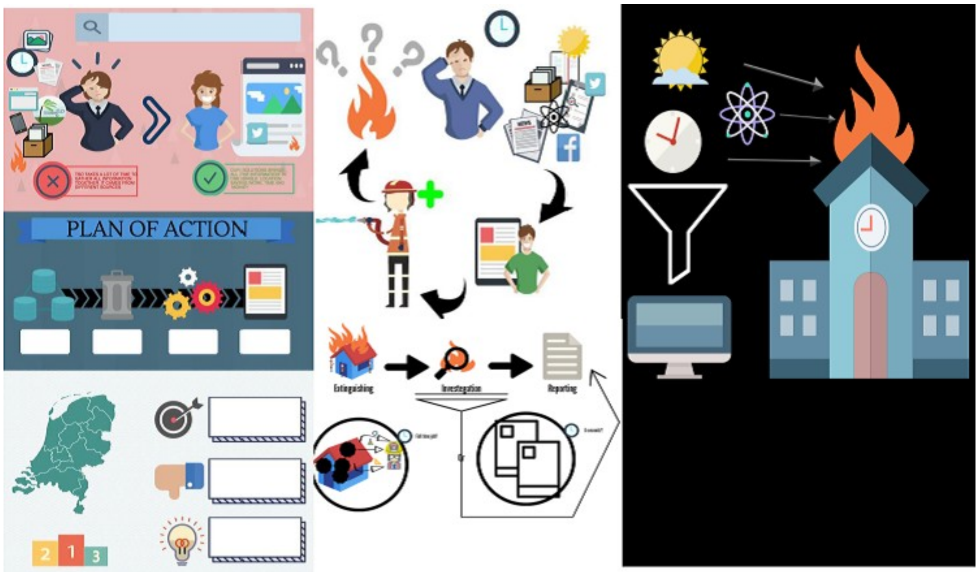


Figure 1: A graphical Representation of the Problem Domain

On the front-end, Angular 2 was used. Angular 2 is a JavaScript Open-source front-end framework developed by Google. It allows developers to build dynamic web applications. One giant advantage is that we can completely separate the development of back-end and front-end. From the start of the project we only made REST endpoints for the application, which is in charge of most logic. Angular 2 uses only these endpoints for its operations; this means that it can be deployed on a separate server, as long as it accesses these endpoints. The reason this procedure was because the application can now be used by BRAIN, the application from Respond as reported in Section 2.

As a consolidation repository, we made use of Mongo DB as the central repository for holding data. It is a document store database and suitable because it can support different data types and sources. To visualize and explore the database we used one of the products of Mongo DB, called Compass. Compass allowed us to interact with the data. The database we received from the company has all the information about the incidents, such as incident ID, date, location, etc. There are many fields and after going through all the fields, talking with Respond! and the fire department, we selected the most important information to be shown at the front end. The links of the articles that are related to an incident are also stored in the database. The same for the weather data and Twitter API.

In order to get data about weather and from social media platforms, we made use of a number of APIs. We used OpenWeatherMap for gathering the weather information. OpenWeatherMap is already an API we can use, while we have to scrape the KNMI website ourselves. However, because OpenWeatherMap is very expensive for past data, we decided that we should still use KNMI for the past data. Therefore, we made use of OpenWeatherMap.

It provides live data, something we couldn't have with the KNMI website. We discussed this issue with both the fire department and Respond!. We proposed to have a live fetching solution. This solution entails a background scheduler, a program which runs in the background of the web server, which will watch for new incidents on a preset time frame. Whenever it finds incidents that happened less than 10 hours ago, it will fetch the weather for that hour and save it into the database. This way we have an additional collection containing the weather information for some time after the incident. The weather fetching works by calling the OpenWeatherMap API, this takes coordinates to fetch the weather from, it also defaults to get current weather which is what we want. There is quite a complex system for the coordinates in place in the backend. The coordinates for incidents are saved as "rijksdriehoeksco¨ordinaten" in Dutch. This coordinate system is only used in the Netherlands, and in order to use it with OpenWeatherMap, we will have to take some steps first. OpenWeatherMap actually expects a city ID to get weather from. These city ID's are provided by OpenWeatherMap as a .csv file. The following steps are done for fetching weather on location:

- Fetching all city codes and storing it in the MongoDB.
- Converting the coordinates of a new incident to longitude coordinates, which are more common.
- With these coordinates, it looks for the closest city using MongoDB's geospatial indexes.
- The city id is used in querying the weather API.
- The results are saved in the MongoDB.

In order to get data from social media, the most popular searching terms were used in conjunction with the available APIs. One thing learned here is that searching for the name of the place is absolutely important. Filtering on things like 'fire' and 'fire department', give more meaning to the filtering. At the start of the implementation, we were calling the search API (REST endpoints) which Twitter opens for everyone. This API allows us to search for optional and mandatory terms. In theory, this method works pretty well, but in practice, Twitter forces a lot of constraints. It seemed that this API could not look further back than a week. This means that a similar solution as for the weather should be introduced. The biggest problem, however, is that the results are not complete. When searching for one specific incident we got a lot of results on the actual Twitter search page, but only 2 in the API. The same search query was used. The Twitter API page said the following: "The Twitter Search API is part of Twitter's REST API. It allows queries against the indices of recent or popular Tweets and behaves similarly to, but not exactly like the Search feature available in Twitter mobile or web clients, such as Twitter.com search. The Twitter Search API searches against a sampling of recent Tweets published in the past 7 days. Before getting involved, it's important to know that the Search API is focused on relevance and not completeness. This means that some Tweets and users may be missing from search results. If you want to match for completeness you should consider using a Streaming API instead. Looking at this it becomes pretty clear that the search API wouldn't work as we would like. The disadvantage of using the Streaming API is that it only works when the application is listening while the incident has only just happened. Luckily this is exactly as how the application will be linked to the database. Using the Streaming API, we basically subscribe to a certain search term (or multiple) and each time a new tweet is placed (matching) we get notified. Using Tweepy, a Python library, we don't have to write many line of code for this to work. A Python object will have a function, which will be called on a new tweet, and from there it will process it.

The Twitter API service is, unfortunately, very constrained as well. We have to deal with the following difficulties:

1. The API is not as comprehensive as the search API. Only mandatory include terms (we can multiple). This means we have to search very broad and filter a lot.
2. Only one stream can be opened at a time. This means that whenever a new incident happens (or an old need to be unsubscribed) we have to restart the stream with new arguments, which are all piled in a heap as well. There is a need to filter based on the incident.

The Streaming API is not ideal, but the completeness is worth it. After implementing the Streaming API, and running some tests with it, it seemed to work well. Each tweet with the correct terms is included in the Streaming API, which means that everything works as expected. The problem with multiple tweet streams is fixed by having the stream as a static object throughout the whole application. Each hour a scheduler checks if new incidents have happened. Basically, it collects incidents up to 10 hours after the incident. This scheduler changes the Twitter Streaming API's search query. The streaming object is calling an object we made with each tweet. In this tweet, we are grouping the tweet with the incident. Also, we are checking if the tweet is somehow relevant with a simple algorithm. Whenever it's relevant it's going to be saved in the database, with the tweet ID and relevancy score included. The current implementation of the filtering algorithm is quite simple as described below, but because it's built very modularly it can easily be improved.

The sorting algorithm first looks in the tweets to see which incident it belongs to. It looks at the tweet text for the place name (e.g. "Amsterdam") and will always find a match because the streamer is searching on place name. When it has found the incident, it is going to give a relevancy score. This relevancy score is either a negative number, which means that the tweet is not relevant and should be discarded. These tweets are not saved to the database, so for now mismatches are irreversible. Relevancy is based by scanning the tweet text for certain keywords defined in the 'constants.py' file. Whenever the tweet receives a score of 0, it's flagged as an automated message. It's quite common tweets are found about the incident that are simply bots posting data from the p2000 system, and these have to be labeled accordingly. For now, the way to find automated tweeters, is by looking at the tweeter name and tweet for certain terms, which are configured in the 'constants.py' file. If the tweet is considered interesting, a score is calculated based on the number of matching keywords, note that an incident usually doesn't generate that many "interesting" tweets, so this score is not that important.

Another source of information to look at was News. According to the fire department, one of the important sources of information about incidents is the news. Therefore, one of the goals was to gather this information and structure it, so the fire investigation team can use it. There were two possibilities to gather the news information. The first possibility was to use APIs of the most common used news sites. However, the application would be very dependent on a few specific sites and it was preferred to collect as much information as possible. Therefore, we thought of a second possibility, which is inspired by the research of the fire investigators itself. The idea was to use a search engine in which different sites will show up. This would make the application less dependent and give a bigger variety of results. Because this only gives the links to the articles, we needed to extract the text from the HTML (news scraping). To find relevant articles regarding an incident, the first step is getting a collection of articles. There are a bunch of possible sources where articles can be retrieved from.

Google - One of the most used sources for finding online articles is Google. Google used to have a bunch of APIs which could be used to find sites and articles. The most practical APIs being: the Google News API, which lets you exclusively look for news articles; the Google Search API, which allows any searching on the Google engine; finally, there is the Google Web Search API which is an extended version of the Google Search API. The problem with these Google APIs is that they are all deprecated. This means, we could technically use them for now but they are bound to be discontinued soon anyway.

Bing - Another big search engine is Bing. Bing is a search engine from Microsoft that can be accessed using the Bing News API which works similar to the deprecated Google News API. The Bing News API has a lot of interesting features that could be used in the project. Some of these are filtering on the Dutch language, returning the results in a JSON format and a direct link to the articles themselves.

Eventregistry - There is also a service called Eventregistry that could perhaps be used. The idea behind Eventregistry is that it already sorts the articles as events. This means that only the event of the fire has to be found and it would be possible to just ask for the articles related to the event. The problem with this is that Eventregistry is a global service that works on events across the world. The result of this is that a lot of small events in the Netherlands are not being registered.

Faroo - The final considered search engine is a free service called Faroo. Faroo is a search engine that allows filtering on News and language. The language filter, however, does not include Dutch. A big advantage of the Faroo API is that it uses a lot of different connection types and protocols like REST, JSON, and XML. The biggest problem with Faroo, however, is that it only allows one request per second which means especially in implementation context is too slow which has a big negative impact on the implementation speed.

A number of additional APIs used to scrap news articles include Scrapy, BeautifulSoup, and Newspaper.

Techniques

These techniques refer to a number of Machine Learning algorithms used and incorporated in the tool that allowed the analysis of these data from different sources. Especially when getting text data from social media and news articles. When gathering, news articles a lot of noise slips through. The idea is to get only the news articles that are relevant to a specific given incident. Therefore, we worked on an algorithm to give each news article a relevancy score. This score could be used to eliminate irrelevant links and sort the relevant links. We had several discussions on how to best define this score and iterated a lot through the algorithm.

First iteration One of the main problems we faced was that we did not have any training data. Therefore, we decided to create our own dataset. First, we had discussions with the fire investigation team and asked them how they determined whether an article was relevant. Then we used their input to create training data: we searched for some articles and guessed a relevancy score from 1 to 10. These guesses were based on some rules those more relevant articles:

- Have larger texts;

- Contain info about victims;
- Contain info about the building;
- Contain info about the cause;
- Have more pictures;
- Are more recent;
- Are from one of the preferred publishers.

The idea was to let the algorithm find a pattern in these scores. After we created the training data we cleaned and prepared the dataset first. We extended the dataset with the features we gathered from the news scraping. The features we used for our first try were:

- Text of the article;
- Number of pictures;
- Number of movies;
- Time between publish date and incident date;
- And the publisher.

We thought these might be related to the relevancy. Sometimes we could not extract the publish date correctly, so that caused missing data in our data set. However, we found out that this messiness mostly occurs for sites that are not real news articles. For this reason, we decided to discard all the rows with missing values for publish date. For the publishers, the fire investigation team already mentioned the most important ones and we already made these as a preference in the search API. The other values for publisher were very divergent, so if we would use these as input features for our algorithm this could result in an overfitting model. Therefore, we decided to not look at this feature anymore. We split up the features into two parts: text features and other features. We made this division because the text will have one feature for each unique word. Therefore, the number of text features will be enormous compared to the other features. We did not want the text features to overshadow the other features. For the text features, we created a bag of words to translate the text into numeric values. We already removed the default Dutch stopwords. These stopwords were retrieved from NLTK21. For the algorithm, we thought of a regression, because the score is continuous output. We tried to apply a Support Vector Regressor with a linear kernel, because according to the sklearn documentation22 Support Vector Machines is one of the best algorithms for text learning. This roughly resulted in an r2-score of 0.4. This was not that bad for a first try, but tuning the parameters did not help improving the score.

For the other features, it was going worse. Because we analyzed the relations of these features to the scores already, we knew there was not clear relationship. Therefore, we did not expect a good score for the algorithm, but we still wanted to try it. Again, we chose a regression, because the score is continuous output. We tried a Linear Regression and a Random Forest Regressor. The Linear Regression was just a default try, but this resulted in a negative r2-score. The Random Forest Regressor we tried because we thought we could predict the score for the other features based on yes-no questions. After all we are expecting that the fire investigation team determines the relevancy based on yes-no questions. However, this second algorithm did not improve the r2-score.

Hence, we reconsidered scoring.Since the r2-scores of the previous algorithms were that bad, we decided to reconsider the scoring system. Also, for the previous data set we did not store the actual incident, so we did not have crucial information like the date the incident happened. We created a new training set with predefined rules to determine the score. The idea was that the algorithm would find the patterns we set up beforehand. However, we did not want to hardcode these rules, because the idea was to provide a possibility for the user to drag articles around. By dragging around in the front-end the user would reconsider the relevancy of the items and therefore updating the training data. When we would train the algorithm again with the updated scores, this would result in different rules for determining the score. However, we still needed these rules for our initial training data. The scoring system we came up with has a range from -9 till 18.

We determined that the publish date should have a real big influence. We thought the time between publish date and incident date should have a range from 0 to 5. If the publish date does not fall into this range it is immediately marked as irrelevant (-9). If the publish date falls into the range it should immediately have a high score, especially in the beginning. When the article gets older it is starting to become less relevant again. This resulted in the following rules:

- For 0 to 2 days after the incident we gave a score of 9;
- For 3 days, a score of 8;
- For 4 or 5 days, a score of 7;
- And bigger than this range a score of 0;

After we determined the score based on the publish date we still want to take three other features into consideration: the total score of the text features; the number of pictures and the presence of videos. Because we wanted the combination of these features to have the same level of impact as the publish date, we thought the maximum score for these three features together should be 9 as well. That is also how we came up with the -9 score for irrelevant publish date, because we wanted to make sure that irrelevant articles scored really low. Furthermore, we wanted the three mentioned features to have an equal weight in the maximum score of 9, so they all got a score range from 0 to 3. Regarding the text features, we thought of concepts that might influence the relevancy: Victim information, information about the cause, information about the vehicles and information about the timeline of the incident.

We looked at several articles and how they contained these concepts and came up with the following scoring system: If one of these concepts was never described, we gave a score of 0; If it was mentioned once or twice, we gave a score of 1; For three or four times mentioning, we gave 2 points; And finally, for mentioning these concepts 5 times or more, we gave 3 points. To downscale texts that are not about fire at all, we decided to give these texts a score of -9. For the number of pictures, we divided the number of pictures in ranges: No pictures results in score 0; 1 to 3 pictures in a score of 1; 4 to 6 pictures in a score of 2; And more than 6 pictures in a score of 3.

This new scoring system basically introduced two decision moments for marking an article as irrelevant: if the text is not about fire; and if the publish date falls not into the defined range. Because this is a binary decision (relevant or irrelevant) and the regressions gave a lot of unwanted scores between -9 and 0, we thought of splitting the algorithm up into a classification first and then a regression. The classification is supposed to filter out irrelevant articles and the regression will determine a score for ordering the items. In this case the filtering is more important than the ordering. Finally, we came up with the following steps to determine the score:

- A classification whether the text is about fire;
- A classification whether the publish date is in range;
- A regression for the text score;
- A regression for the total score, to combine all features.

For the two classifications, we gave a total score of -9 already, if the article was considered irrelevant. Also, we added fields to store whether this irrelevancy was based on the text or the publish date.

As described above the text classification classifies whether the text is about fire. If the text is not about fire, we immediately give the article a score of -9. Again, we created a bag of words and removed the Dutch stopwords. The first algorithm we tried was Support Vector Machines, because this is one of the best algorithms for text learning according to the sklearn documentation. A cross validation of 10 times with a linear kernel, resulted in a mean accuracy score of roughly 0.75. We also tried the rbf kernel and tuned some parameters, but this did not improve the accuracy. Although the accuracy was not bad, we wanted to accomplish a higher score. The second algorithm we tried was Random Forest Tree. If the classification could be based on the existence or frequency of certain words, then we thought we could use a tree of yes-no questions to determine whether the article is about fire. However, this accuracy score was roughly the same as for Support Vector Machines. Then we tried K-Nearest Neighbors. Because our classification has texts that are about fire and texts that are not about fire, we basically want to know for a new text to which group it is closer. The accuracy for this algorithm was a little bit higher. For the parameter weights we used ‘distance’, to make sure that closer neighbors have a greater influence than neighbors further away. Because our training set was relatively small and we wanted to have an odd number for the number of neighbors, we chose to look at 3 neighbors for the predictions. These combinations resulted in an accuracy score of roughly 0.77. Because we wanted to improve the accuracy score even further, we implemented the TFIDF transformer. This transformer down scales the weight of words that occur a lot in all of the articles and up scales rare words. Because these words occur less often, they contain more information about the content of the article. This preprocessing step increased the accuracy score of K-nearest neighbors to roughly 0.9.

During our first iteration of the algorithm we noticed that the result depended for 90% on the date. This is because if a news article is too old or too recent we don’t want to show it. But this meant that a good article that might be 2 days older than a bad article would get a lower score. To resolve this issue, we decided that it might be better to make sure that the news articles with a date that’s far away from the incident are always considered as irrelevant. This can be done with classification where we can classify between “within date range” and “not within date range”. Once we have this classification we can remove all the news articles that are not within the date range. We then have a more specific dataset with only news articles within the date range and can go for a more precise prediction. For the date range we had to decide a decision boundary for our training data. We decided to consider all the articles that are written after the incident are “within date range” and all the articles written before the incident are “not within date range”. The first algorithm we tried was SVM because we expect a very linear decision boundary. This was true as we repeatedly got an accuracy score of 1.

After the classifications, we only have the texts about fire left. For these texts, we want to determine a score from 0 till 2. For this continuous output, we tried the regressor variants for the algorithms we tried in the text classification. Additionally, we tried a default Linear Regression. The K-Nearest Neighbors Regressor gave the best results again. However, the r2-score of this regressor was still very low. It seems like there are no clear patterns in the scores of these texts. Possibly because the concepts we defined (victims, cause, vehicles, timeline) usually show up in the texts in very different ways with very different words. Unfortunately, we did not have time to improve this score further. Therefore, we advise to try to improve this regression in the future. However, for now it is not a big problem that the accuracy it not that great. The score will only be used for the ordering and because the articles at this stage are already marked as relevant, the fire investigation team will probably read all the articles anyway. Also, one of the causes for these low scores could be that we did not have much data points left after discarding the articles that are not about fire. The score might already increase when a bigger training set is used.

For the regressor concerning the total score of a news article we tried a Random Forest regressor. The reason for this choice was that we created our score with a set number of questions. Since the Random Forest regressor works in a similar way we assumed that it would be the best choice for trying to pick up this trail. We did try the other regressor briefly but they were less accurate. With the Random Forest regressor we were able to get a r2-score around 0.97.

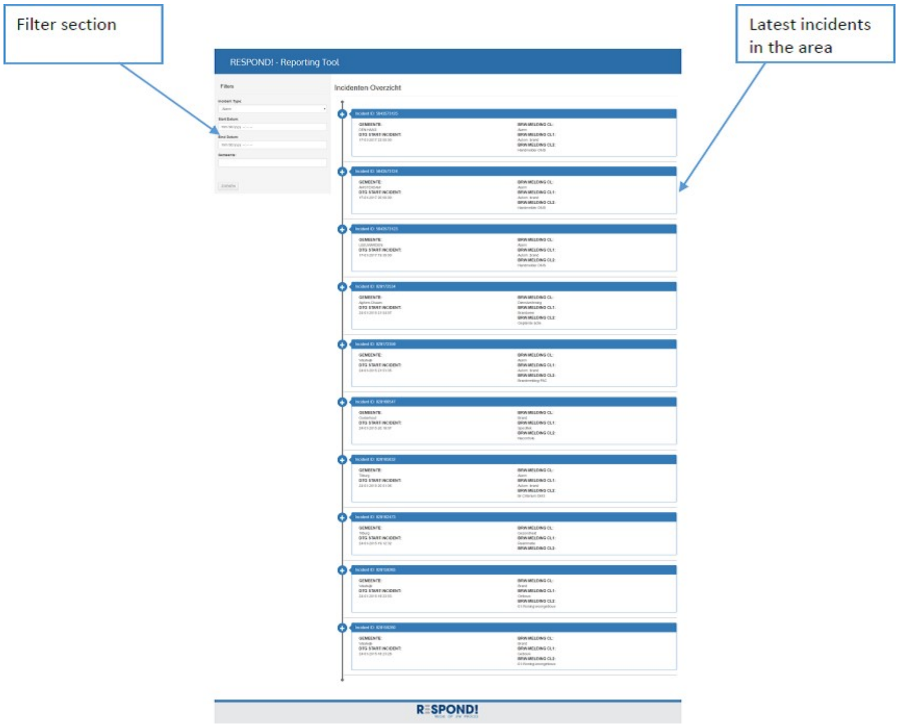
Because the idea was to take the changes of users into consideration in our prediction and because we want the algorithm to improve over time we did a little research about how to retrain our algorithms. If we refitted the algorithm, then the previous learned patterns were discarded. This means that we have to use the whole data set again for retraining the algorithm. However, we tested how much time this operation would take for a large data set of roughly 10.000 records; this took less than a few seconds. But the idea was that the patterns could change over time, so we planned to only use the last x number of records for retraining the algorithm, although this is not implemented yet. We also did not want to retrain the algorithm every time when we needed a prediction. So, we decided to train the algorithm when the application starts and retrain once every day with the updated data.

Results

The tool, exemplifying the use of different techniques, comprises a number of pages as seen in Figures 2 and 3 . The first one is the overview page that allows you to see the most recent incidents in the area. On the left side, there is a side menu that allows the user to input search criteria in the fields in order to find the required incident. On the right side, there are the incidents that the user can click on. The overview page displays the incident ID, distract name, date of the incident, and the categories of the incident. Once the user clicks on the desired incident the application takes the user to the second page.

As demonstrated in Figure 3, the tool allows the user can see the details on the left side about the incident that was selected. Information is directly fetched from the mongo DB web server. On the right the user can see the weather including the temperature and weather conditions at the time of the incident. The user can also look at the recorded data: for example, go further and see what the temperature was a few hours after the incident. Below that there is the Twitter section that contains the relevant tweets regarding the incident.

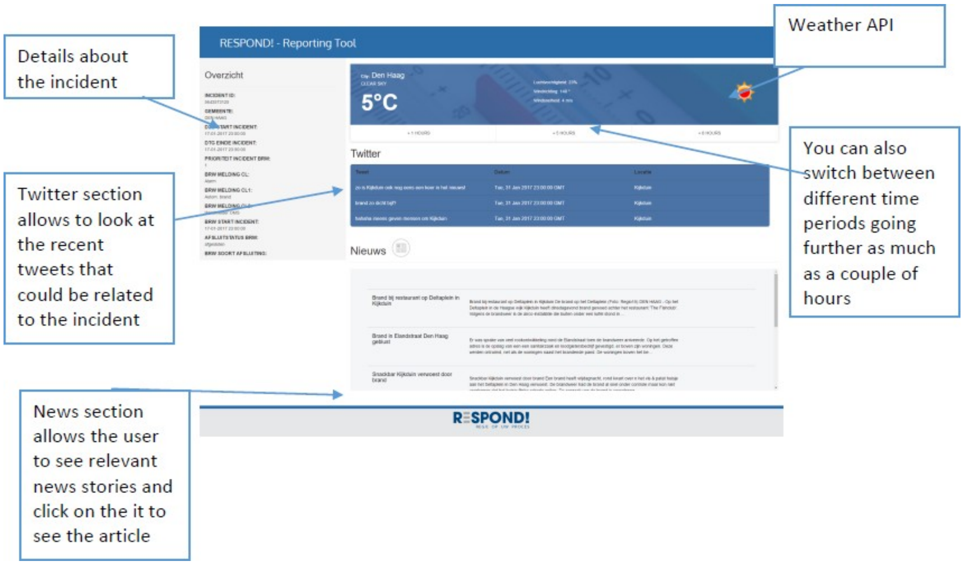
The section contains the tweet title, date and location at which it has been posted. Below that there is also the news feed that uses machine learning techniques to pick out the most relevant news articles to the particular incident. This section also has the title of the articles and the sample text so the user can read an



Filter section

Latest incidents in the area

Figure 2: A screenshot of the Reporting Tool



Details about the incident

Weather API

You can also switch between different time periods going further as much as a couple of hours

Twitter section allows to look at the recent tweets that could be related to the incident

News section allows the user to see relevant news stories and click on the it to see the article

Figure 3: A screenshot of another page of the Tool

extract of it before choosing to read more. The user can also click on an article to go to the actual website where the article is located and get additional information if needed.

Conclusion and Future Works

Although we already have a nice application for the fire investigation team to work with, there all still some improvements we suggest for the future. The regressor for the text learning is currently not working very well. There are a bunch of possible reasons this could happen. It might be because there are a lot of different words that can be used for the same concepts. For example, the reason might be in an article but there are a lot of different ways this can be said and a lot of different reasons that might have happened. Perhaps this could be solved by simply increasing the training data set.

The longer the application is running the bigger the dataset will become. This does however also mean that the application will take longer and longer to fit the regressor and classifiers. A possible solution to this is only using a part of the data. We suggest using the last entries in the dataset. This would help speed up the algorithm but it also means that the most recent incident will have the biggest influence in the algorithms, which will cause changing the patterns over time. The idea of the front-end was that the user would have all kinds of sections of information regarding an incident. Like Social Media, Weather, News etc. Originally, we were planning on making these sections draggable so the user can put them in order to his liking and let the system generate a better order for new incidents. This could be realized with Machine Learning.

In the news scraping process the "none news sites" are removed because they generally do not have a publish date. This, however, does not always work perfectly. It will be a valuable improvement to make sure the searching engine only returns news sites instead of all sites. The tweet relevancy algorithm could be updated to work similar to the algorithm that decides the relevancy of News Articles. This would make it more dynamic and allow it to change based on the feedback the fire department gives. Currently we only get information from Twitter, but there are obviously many other Social Media sources that can be used. One of the examples is Facebook. Posts on Facebook are often longer than the 140 characters' limit of tweets. Instagram is another possible Social Media channel that can be used because of all the pictures on there. For Instagram, for example, there could be thought of a Machine Learning algorithm that filters out the relevant pictures. In the current version is not possible to export an incident with all the content to Excel or another format for csv.

Acknowledgment

Although done under our supervision, this work and the technical details were executed by these students: Vania Alves Figueiredo, Alexander van Cuijk, Edwin Dijsselbloem, Thomas Kleinen- dorst and Tomek Szpak. This is, therefore an exposé of their graduation project form the Applied Data Science Minor from Fontys University of Applied Sciences.

References

[1] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In International Conference on Machine Learning, pages 173–182, 2016.

[2] Hamid Ekbia, Michael Mattioli, Inna Kouper, Gary Arave, Ali Ghazinejad, Timothy Bowman, Venkata Ratandeep Suri, Andrew Tsou, Scott Weingart, and Cassidy R Sugimoto. Big data, big- ger dilemmas: A critical review. Journal of the Association for Information Science and Technology, 66(8):1523–1545, 2015.

[3] Patrick Mukala. Process models for learning patterns in FLOSS repositories. PhD thesis, University of Pisa, 2013.

[4] Patrick Mukala, Joos CAM Buijs, Maikel Leemans, and Wil MP van der Aalst. Learning analytics on coursera event data: A process mining approach. In SIMPDA, pages 18–32, 2015.

[5] Patrick Mukala, Antonio Cerone, and Franco Turini. Ontolifloss: Ontology for learning processes in floss communities. In International Conference on Software Engineering and Formal Methods, pages 164–181. Springer, 2014.

[6] Patrick Mukala, Antonio Cerone, and Franco Turini. Process mining event logs from floss data: state of the art and perspectives. In International Conference on Software Engineering and Formal Methods, pages 182–198. Springer, 2014.

[7] Patrick Mukala, Antonio Cerone, and Franco Turini. Mining learning processes from floss mailing archives. In Conference on e-Business, e-Services and e-Society, pages 287–298. Springer, 2015.

[8] Nico Neumann. The power of big data and algorithms for advertising and customer communication. In
2016 International Workshop on Big Data and Information Security (IWBIS), pages 13–14. IEEE, 2016.