FMR - Football Match Reporter

Automated reporting on English Premier League matches

By Jure Štabuc

This dissertation is submitted to the Cardiff School of Journalism, Media & Cultural Studies, Cardiff University, in partial fulfilment of the requirements for the Degree of Master of Science in Computational Journalism.

September 2017

DECLARATION

Student number: 1603403

Štabuc

Mr

Jure

This work has not previously been accepted in substance for any degree and is concurrently submitted in candidature for any degree.

Signed Jure Štabuc, September 1 2017

This dissertation is being submitted in partial fulfilment of the requirements for the degree of MSc.

Signed Jure Štabuc, September 1 2017

This dissertation is the result of my own independent work/investigation, except where otherwise stated.
Other sources are acknowledged by footnotes giving explicit references, A Bibliography is appended.

Signed Jure Štabuc, September 1 2017

I confirm that the electronic copy is identical to the bound copy of the dissertation.

Signed Jure Štabuc, September 1 2017

I hereby give consent for my dissertation, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed Jure Štabuc, September 1 2017

I hereby give consent for my dissertation, if accepted, to be available for photocopying and for inter-library loans **after expiry of a bar on access approved by the Graduate Development Committee.**

Signed Jure Štabuc, September 1 2017

Table of Contents

Introduction

News media always try to go hand in hand with technological development and audience demands. The field that especially prevails in its growth is sports. Andrews (2014, 1) found out that it is the fastest-growing sector in the British media, and the same applies to most English-speaking counties. This demands that the sports stories published have to be accurate, appealing to the specific audience and introduced to it before the competition reports on same or similar topic. Especially football attracts most of the audience as it is world's most popular sport. According to Weil and others (2017), the International Federation of Association Football, FIFA estimated that at the turn of the 21st century there were approximately 250 million football players and over 1.3 billion people interested in football. Not all of them can follow football matches live so they depend on match reports. This means that the news media have to deliver them right after the final whistle to be published in next day editions or immediately on the web. Usually this means the match reports are written "in a series of takes, originally by telephone to a copy taker, but now more likely from a laptop straight into the newspaper's computer system." (Andrews 2014, 59) Andrews (2014, 61) also explains that the time constraints imposed on journalist writing running copy mean that they sometimes have to write much of it before anything of real significance has happened.

The aim of this dissertation is to discuss and develop an algorithm and to demonstrate how automated algorithm can overcome these problems and substitute a journalist in reporting on football matches. It will also deal with why automated journalism is the future of reporting on football matches, the questions that are raised around it and positive and negative aspects of using algorithms in journalism and how journalists can collaborate with machines. Carlson (2015, 417) explains that automated news content creators are an outgrowth of the intersection between journalism and the growing emphasis on data analysis known popularly as big data. Much of the discourse around the confluence has explored new data tools available to journalists, including partnerships between journalists and computer programmers - or hacks and hackers - as well as the skills journalists need for going forward.

The project will focus on building an algorithm that will produce fully automated English Premier League (EPL) match reports. It will generate text from a large amount of data with help of machine learning, natural language generation (NLG), templates and rules about which situations are regarded as good and which as bad for each of the teams in the match. The data itself will be sourced from WhoScored.com, which provides the data of every event

in a match, and from premierleague.com to get information about league standings. The dissertation will show how to report on football when good structural data is available.

Critical Review

Automated journalism is relatively new in the area of computational journalism. Graefe (2016, 14) defines it as the process of using software or algorithms to automatically generate news stories without human intervention.
As already pointed out above, because the field is quite new, not many algorithms that produce sport stories exist, let alone football match reports. All of the analysed products are developed by an agency and do not offer an open source code to see how the algorithm was developed and which programming language they used.

Bakken & Bæck

The Norwegian digital studio Bakken & Bæck developed in a collaboration with NTB news agency is Norway's first digital football reporter. Waldal (2014) explains that they wanted to develop an algorithm which would create match reports without any errors. The generated articles would not have to go through an editor but could be distributed straight to the customers.
The algorithm is fed with information from NTB's (Norwegian Press Agency) reporters who provide the data through live services. Additional data is taken from the nifs.no (Norwegian International Football Statistics) database which provides football related statistics in Norwegian language. As Waldal (2014) points out, this provided enough good data for generating articles. Graefe (2016, 14) argues that in such situations, algorithms can create content on a large scale, personalizing it to the needs of an individual reader quicker, cheaper, and potentially with fewer errors than any human journalist. The algorithm's output is a standardized structure of a match summary, which, as Waldal (2014) explains, includes a title, lead paragraph, body of text (as understood from the documentation, they use only goal related events), post-script with some additional information from the match and a conclusion. In the writing stage, the robot makes use of a large set of templates which make the language as natural as possible, Waldal (2014) mentions in the documentation. With which words or expressions the templates are filled depends on predefined rules. One of them deals with the article narrative. If the player is mentioned multiple times in a row, for example has scored a hat trick, the algorithm does not always refer to him with his full name. For a natural flow they rather use other options describing a player. After the player is mentioned by his full

name, he will be addressed only by his surname and afterwards by information referring to his age, playing position or nationality. For example: "Alexis Sanchez scored in the 7th minute. Sanchez scored again after an assistance by Özil in the 43rd minute. The Arsenal attacker settled the score in the 89th minute." It was their project that I looked upon and tried to develop a similar algorithm in my thesis.

Wordsmith - Automated Insights

Wordsmith is a NLG platform developed by Automated Insights which can create numerous data-driven narratives. As stated online (Automated Insights 2017) Wordsmith can turn data into text in any format or language. Reynolds (2015) explains that Wordsmith functions by creating branching paths. It conditionally adds words, phrases and sections that can be added, modified and removed depending on the article. Users enter data, such as quarterly figures or sports results, around which these branches are built. This story structure, once created, can be used as a template for an unlimited number of articles. All users have to do is enter new data to create a unique story. Its categories include e-commerce, financial services, weather forecasts, city guides, client updates and sports. As presented in their gallery, Automated Insights offers with Wordsmith a reporter that recaps football matches. They offer three examples of a match outcome: a home team win, an away team win and a draw.
Their algorithm produces a title which includes the name of the league, followed by a date and a title. The main part of the match report does not report on specific events such as goals. They rather mention the score the teams went into the half time and the full-time score. This is followed by a paragraph describing match statistics such as shots on and off target and corner kicks. Wordsmith concludes the article with a reference to the referee, bookings received and committed fouls by both teams. Comparing these examples to the one of Bakken & Bæck's bot, we can see that the latter is much more detailed than the Wordsmith. In comparison, both have an article structure, though Bakken & Bæck's product is much more detailed in describing events than Wordsmith.
The Wordsmith platform is used by Associated Press (AP) sports editorial. As McCormick (2016) explains, the robot is used to cover the US minor league baseball. Stacy Liberatore (2016) reported, AP started using automated software to cover 10,000 games. It is fed with data from MLB Advanced Media. As she states, human reporters were used by the AP to cover some Minor League games in 2006, but could not cover the full slate of teams and leagues. By using Wordsmith, AP can now report on games it previously could not without a staff of hundreds of dedicated journalists.

StatsMonkey

Another project that was used to report baseball games is also one of the pioneers in production of computer-generated stories. StatsMonkey was developed by the company Narrative Science in collaboration with Intelligent Information Laboratory at the Northwestern University and was able to write a basic play by play story. "In order to write these stories, StatsMonkey must determine what narratives apply to a given game, which of those narratives are most interesting or important, and which moments in the game most exemplify each narrative." (Allen et al. 2010, 2)

Allen et. all (2010, 2) argue, that StatsMonkey was not designed to list the events of the game as we have seen in the examples mentioned above. Instead it writes up the story of the game by emulating what a sports reporter would look for in a game, as they call it, the "big picture". StatsMonkey determines the importance of events, players and statistics in the game by computing a number of critical derived features above and beyond the raw data. "The system takes advantage of three important advanced baseball statistics: Win Probability, Leverage Index and Game Score." (Allen et al., 2010, 2) As they explain (Allen et al. 2010, 2-3) each of these statistics allows the system to identify key moments and players in the game. By using a decision tree to select the appropriate narrative arcs this then determines the main components of the game story and enables the system to put them together in a narrative. They already included (Intelligent Information Laboratory 2014) what other projects mentioned above did not - that the stories can be generated from the point of view of either team. This way they can produce neutral or biased stories targeted to fans or publications from the cities the teams come from.

Heliograf - Washington Post

Heliograf is a bot that was developed (WashPostPR 2016) to report key information from the 2016 Rio Olympics. It generated multi-sentence updates which appeared in The Washington Post’s blog, Twitter and other platforms. They provided readers with a daily schedule of events, results from medal events, notifying users of an event etc.

"Heliograf consists of four main parts: an observer, detector, writer, and distributor." (Johri et al. 2017, 3) The observer monitors an endpoint of an Olympic discipline and caches the most recent version. The event detection logic lies in the detector. Johri et al. (2017, 3) defined triggers that created events such as "medal results announced". The trigger then only needs to

take into account the current and previous state of data. They explain that because the event logic is simple in the Olympics, they chose to skip the detector.

For the writer part they focused on natural language generation (NLG) which consist of document planning, microplanning, and surface realisation. (Johri et al. 2017, 3) As they explain in their article (Johri et al. 2017), document planning is deciding which information will appear in the output text, microplanning is deciding which specific words should be used to express the content, and surface realisation uses the rules of grammar to convert abstract representation of sentences into actual text. For surface realisation they used a library called SimpleNLG, developed at the University of Aberdeen's Department of Computing Science. It enabled them to create a recursive hash that represented a sentence's syntactic structure and fed it to the SimpleNLG realisation engine. The final stage of the bot was distributing the generated text into their various one-way channels, such as Twitter, Slack, Wordpress etc., as well as two-way channels such as Facebook Messenger, where users ask questions about the Olympics and the bot answers them. As they found out, the Olympics bot worked well to supplement the work of journalists. "It posted medal events to the Washington Post Olympics live-blog, guiding the conversation for the live-blogging human journalist and allowing them to focus on the important analysis for the reader." (Johri et al. 2017, 4)

As we can see, there are not many products dealing with automated match report generation on the market. For the English speaking market, there are the two already mentioned, *Narrative Science* and *Automated Insights* from the United States, there is also *Arria* from the United Kingdom, but neither offers match content generation nor does it provide reports on EPL matches. Most probably Wordsmith could offer that. Although Bakken & Bæck's algorithm produces articles that can be posted online without an editor going through them, they cannot produce articles for other leagues as they write in Norwegian language and use the data of the Norwegian league.

The aim of this dissertation is to develop an algorithm which will provide reports on the EPL. In the structure of the generated article it will look upon Bakken & Bæck's one. Unlike their product which reports on the Norwegian league, the project will focus on the EPL, one of the most followed leagues in the World that attracts a large number of audience. This guarantees a larger targeting group as well as more information which would be available as data input.

Method

When developing projects that generate text from data, we are talking about Natural Language Generation. Glascott (2017) defines NLG as a subfield of artificial intelligence (AI) which produces language as output on the basis of data input. She also describes different variations of NLG. Basic NLG automatically translates data into text via Excel-like functions. In templated NLG, the user is responsible for writing templates, determining how to join ideas, and interpreting the output. Advanced NLG communicates the way humans do - infusing intelligence and intent into the process from the very beginning. It assesses the data to identify what is important and interesting to a specific audience, then automatically transforms those insights into insightful communications packed with audience-relevant information, written in conversational language.

I decided that the developed algorithm will be a templated NLG with the addition of grammatical rules. I considered this option the best to show results. I did not decide for an advanced NLG because of the lack of libraries. The best and the only natural language generation library worth mentioning, SimpleNLG, is written in Java. No library that would function in Python could be found.

The algorithm produces an article which starts with a headline referring either to the teams playing or a player's performance. This is followed by the main body of text where crucial events of the match (goals and assists) are mentioned. It finishes with a conclusion, where it explains what the result brought for the teams in the matter of table standings and how many points each of them have.

In order to produce an algorithm that automatically recaps EPL football matches I chose Python as the programming language. Python is "a simple yet powerful programming language with excellent functionality for processing linguistic data." (Bird et al. 2009) It has a good string-handling functionality (Bird et al. 2009) which is useful when working with data and generating sentences. It also permits data and methods to be encapsulated and re-used easily (Bird et al. 2009) which is useful in the development of text algorithm where some variables have to be re-used over and over again when producing templates. Python also comes with a large library, offering many frameworks and tools.

To generate articles, the data had to be fed into it. Since such data is not open source, it had to be scraped. WhoScored.com, a website specializing in analysis was chosen and data for 20 matches in the 2016/2017 season was scraped. To add linguistic variety and additional information to the generated article, the whole player information database from the same

website was also scraped. To give table position reference, I scraped every match day table from the official website of the EPL, premierleague.com.
The algorithm follows a decision tree. This means that the algorithm follows rules and conditions and when the condition is met it will be returned.
To check similarity of sentences and to prevent that the algorithm chooses sentences that are too similar, I decided to use Natural Language Processing (NLP). This, unlike NLG, turns a text into structured data (Nichols 2017) to analyse it. Numerous NLP libraries are available for Python. I have considered NLTK and spaCy. NLTK is the most famous one and has led to breakthroughs in the field. However, it is heavy and slow. (EliteDataScience 2016)
SpaCy on the other hand is a new NLP library that's focused on performance and does not flood you with options like NLTK. It is easy to use and faster. It also checks similarity between strings with only a few lines of code, whereas NLTK is much more complicated.
For the project development, I used a Minimum Viable Product approach (MVP). Before moving to the next stage, each stage of the algorithm had to be a completed entity that delivered results. With every further stage, more features were added and the project was constantly being improved.

Original research

How match reports are done nowadays

Andrews (2014, 58) makes an interesting point when he contemplates why sporting events are reported because that should determine how the job is done. If we already know the result, why should we be interested in how it was achieved? Andrews (2014) points out several reasons:

- A report offers a vicarious experience for readers who were not able to attend or watch the event themselves. Reports should therefore offer vivid descriptions of the key moments of the event.

- It also gives those who were able to attend or who watched the event on television an opportunity to relive the experience, and to compare their impressions with those of an expert observer. [...] Reports must be accurate and well-informed if they are to impress the reader who has witnessed the event.

- Some readers want to test the opinions they formed while watching the event against those of an expert. Reports can generate debate and provide a printed equivalent of the post-match discussion in the pub. People also read reports for information and entertainment. Reports should also offer analysis and criticism of tactics and performances and put the outcome of the event into context - what does it mean for a team's prospect for promotion or relegation?

Butler (1999, 66) and Andrews (2014, 63) suggest that reporters should make notes during a game in a notepad. As the match continues they should write down any important moments such as key incidents, together with the times at which they happened for each team.
They also notice that some reporters dictate a running copy straight from their notes, though most write out their takes before dictating them or filing them electronically. Usually the match report is almost finished while the game is in progress. The whole process is finished on the final whistle.

Why automation in sports journalism?

With many sport events per day they usually take a large proportion of space in print, broadcast or web media. Andrews (2014, 1) argues that sports coverage is vitally important to the health and prosperity of the print and broadcast media. He continues that the British newspaper market is the most competitive in the world and that competition takes place on the sports pages.
Media competes especially during and after events, on who will deliver the first recap or a match summary. Most of the sporting events take place at the most convenient times for the public to watch - weekends and midweek evenings. As Andrews finds out (2014, 55-59), this places difficult demands on daily and Sunday newspapers wishing to cover the events. Their first editions may go to press less than an hour after games and meetings end. To be able to report on matches in their early editions, sports desks must take in and process the copy produced by their reporters quickly and efficiently. This depends to a large degree on forward planning by the production staff on sports desks in newspaper offices and fast work by the reporter in the field.
Because of this, as Andrews (2014, 55) explains, describing sporting events is one of the key skills of the sports writer, and provides the basic content of sports pages. Reports of events occupy more space in newspapers, radio and television than any other form of sports journalism.

This growing competition affects especially journalists, who cover these events under pressure as they have to deliver reports as soon as the match ends, which can result in errors. Andrews (2014, 58-59) explains that covering a sporting event for a daily newspaper is one of the most demanding and pressurised jobs sports writers have to do. It involves writing a predetermined number of words quickly and accurately while the action on which they are reporting is still unfolding before their eyes. It demands not only a comprehensive knowledge of the sport being covered, its rules and history, and a similar knowledge of the individuals taking part, but also the ability to write accurately and entertainingly against tight deadlines and to a specific length. The production demands of newspapers may also require reporters to write two or more different versions of their reports within a very short space of time. Butler (1999, 66) defines such pieces as runners. He adds to Andrews' explanation that to be good at a runner it is simply a case of keeping cool. Although it sounds easy, it can be difficult when reporters are trying to make notes and watch a game at the same time. They could be scribbling down the details of a goal when another score happens. Or there could be a major fight with a dozen players involved. The game won't stop while reporters catch up. As Butler (1999, 66) and Andrews (2014, 55) conclude, a runner is a sure-fire way to sort the good from the bad and the ability to cover the action in a lively, informative and accurate way is an essential tool for any sports journalist.

Because of this, media is looking forward to automation of news stories. As Graefe (2016, 9) explains, if good structural data is available, once an algorithm is developed it cannot only create thousands of news stories for a particular topic, it can also do it more quickly, cheaply and potentially with fewer errors than any human journalist. As explained above, match reports are especially error prone, since they need to be produced quickly and because reporters are making brief notes of key moments during a match. Often they just shorten the names of players and cut note taking to a minimum. On one hand it allows them to focus on the game but on the other hand one gets lost quickly in unorganised notes. The development of news bots has fuelled journalists' fears that automated content production will eventually eliminate newsroom jobs, while at the same time scholars and practitioners see the technology's potential to improve news quality.

Automation raises many questions

Beyond these outcomes, Carlson (2015, 429) argues that automated journalism raises questions striking at the core of how journalism should be understood. The ability for automated journalism to pass for human writing forces a re-examination of newswriting as caught between a reliance on learned formulas and the need for individualized style. Viewed in term of its output, Carslon (2015) continues, journalism provides not only information but also a way of knowing the world that has accrued the epistemic authority to be considered legitimate. The adherence to formal patterns, while not always imaginative, rests on an argument that this style of conveying news should be respected. For journalists reacting to the automated writing, quality newswriting involves something more than what is encompassed in normative ideas of news discourse. Journalists identifying what cannot be automated touted the dramatic possibilities of news and the power of stories, which suggests a broader understanding of journalistic authority to better account for these elements.

Another concern that has been raised amongst journalists is what will happen to them. Clerwall (2014, 527) explains that when it comes to automated created content, journalists recognise that it may be a threat to some, as it may put journalists performing routine tasks out of work and it can be applied beyond sports reporting and also challenge the jobs of journalists in finance or real estate. The journalists emphasize some of the strengths of human journalists, such as creativity, flexibility and analytical skills, indicating that the more advanced journalism is not threatened by automated journalism.

Carlson (2015, 429) also demands that questions need to be asked regarding whether an increase in algorithmic judgment will lead to a decline in the authority of human judgment. This is perhaps the central question at stake with the technological drama surrounding automated journalism. Another necessary area of research on automated journalism is attention to how news audiences make sense of and interact with news produced by algorithms. After all, as he explains, the economic and authoritative underpinnings of automated journalism rest on its acceptance outside the newsroom. A final concern Carlson (2015) mentions, is the need to place automated journalism within larger discussions of automation and the future of knowledge labour. In this regard, the issues explored above expose the entrenched cultural conflict between equating technological development with progress and deep distrust of machines as dehumanizing forces. The progress of artificial intelligence extends this conflict into new terrains of knowledge and expertise that warrant much more investigation.

Positives and negatives of automated journalism

As a new field in journalism, automation has a lot of positive but also some negative aspects. Carlson (2015, 429) explains that at the positive end of the spectrum, the growth of automated journalism greatly expands the amount of available news and frees up journalists to pursue less mechanical stories. The technology also aids journalism as a smart system capable of finding patterns easily missed by human perception. Graefe (2016, 10) continues that automated journalism is most useful in generating routine news stories for repetitive topics for which clean, accurate, and structured data are available. This is especially true for football match reports where journalist have to report on 18 matches per match day. Andrews (2014, 1) explains that it is not unusual to find a hundred journalists covering a single match in English soccer's Premier League. Because there are many services who provide live, machine readable data about what happens in a match, Graefe (2016, 10) argues that algorithms are able to generate news faster, at a larger scale, and potentially with fewer errors than human journalists.

Especially in football where we have two opposing sides, sometimes from different countries, it is important that "algorithms can use the same data to tell stories in multiple languages and from different angles, thus personalizing them to an individual reader’s preferences." (Graefe 2016, 10) Graefe (2016, 10) concludes that algorithms have the potential to generate news on demand by creating stories in response to users’ questions about the data. The key drivers of automated journalism are an ever-increasing availability of structured data, as well as news organizations’ aim to both cut costs and increase the quantity of news.

The latter is especially true as Phil Andrews (2014, 6-7) states that some national newspapers will have reporters covering major sports like soccer based in specific cities or areas of the country. But with rapidly expanding sport and especially football sector more and more journalists are demanded to cover matches. Therefore Carlson (2015, 429) finds out negative predictions include increased layoffs, polarizing personalization, and the commoditization of news writing. Graefe (2016, 10) also explains that automated journalism is not useful when covering topics for which no structured data are available and is challenging when data quality is poor. He continues (2016, 10) that algorithms rely on data and assumptions, both of which are subject to biases and errors. As a result, algorithms could produce outcomes that were unexpected, unintended, and contain errors. Algorithms also cannot ask questions, explain new phenomena, or establish causality and are thus limited in their ability to observe society and to fulfil journalistic tasks, such as orientation and public opinion formation. Lastly

but most importantly, the writing quality of automated news is inferior to human writing but likely to improve, especially as natural language generation technology advances.

Executive Summary

The English Premier League (EPL) has according to Curley and Read (2016) and Dawson (2017) the highest revenue amongst all football leagues in the world. It generates 5.14 billion British pounds in TV rights. The matches are broadcast in 212 countries worldwide with an audience of over 12 million people.

As fans can usually watch only one match at a time, they want to read match reports about what happened in other stadiums. Many of the fans seek information about matches on English media websites as these are most accurate in reporting on the league. Additionally, fans who could not follow a specific match want to read a report right after the final whistle. Automated text generation is in this case important for the news organisations. With the ever-ongoing competition between them on who will publish first, the algorithm plays a crucial role.

FMR is the first algorithm that generates match reports for the EPL right after the final whistle. With FMR there is no forward planning needed before the match for reporters which can loosen time for them to prepare for other aspects of the match. Reporters who nowadays report on a match have to be quick after the end to hand in a report. By finishing something quickly there is always the risk of making errors in the text. Errors can also occur during the match when the reporter is taking notes or even misses an event on the pitch. No such problem can occur with an algorithm like FMR. There is less space for mistakes as the algorithm generates text out of available data about the match. When it has all the data needed it can write an article quicker than any reporter, which means a reporter would be replaced by a robot reporter. In the media market there is a crisis amongst media organisations who would like to cut down costs. There are 18 matches per match day which also means 18 reporters working. The algorithm can replace them and cuts down the costs. Or on the other hand the reporters can focus on football stories that need human input. In this case, the news organisation could spread the number of news about a match and the league. It could also report on background of the match, history, extensive pre-match reports and detailed match analysis and other happening after the match as well as investigative work about the EPL. This could help them to attract more users as they would prefer a media who reports on a lot more than just the match itself.

Title and Market Sector

The algorithm is named FMR which stands for Football Match Reporter. It is an algorithm written in Python and needs good structural data about the events in an EPL match. It is aimed at every news organisations that wants to report on a match quicker and more accurate than any reporter.

Competitors

The main competitors are the algorithms from Bakken & Bæeck and Wordsmith developed by Automated Insights.

Bakken & Bæck

The algorithm was developed by Norwegian digital studio Bakken & Bæck in a collaboration with NTB news. The generated articles do not have to go through an editor but can be distributed straight to the customers.
The algorithm's output is a standardized structure of a match summary. It includes a title, lead paragraph, body of text (as understood from the documentation, they use only goal related events), post-script with some additional information from the match and a conclusion.
Because their algorithm is developed around Norwegian league data and uses Norwegian language rather than English, the reach of the product is limited. Norwegian league is not popular outside of Norway so the articles can only reach people who follow Norwegian football and understand the language.

Wordsmith - Automated Insights

Wordsmith is a natural language generation platform developed by Automated Insights which can create numerous data-driven narratives that can turn data into text in any format or language. Wordsmith functions by creating branching paths. It conditionally adds words, phrases and sections that can be added, modified and removed depending on the article. Users enter data, such as quarterly figures or sports results, around which these branches are built. This story structure, once created, can be used as a template for an unlimited number of articles. All users have to do is enter new data to create a unique story.
By offering all of these their reach is only limited to the languages offered. The generated articles can reach a much bigger audience than the Norwegian product. On the other hand, this also means that the articles are rather generic as they do not mention specific events like

goals as observed from the examples on the website. Because it is not developed for a specific league such as EPL, media organisation reporting on matches from this league cannot really use it because of the high editorial standard in reporting football. The generated articles as such cannot target followers of a specific football league such as EPL audience.

Audience

Articles generated by FMR are aimed for audience who wants to read the recaps immediately after a match is finished. Because it is developed specifically for the EPL it is targeted for the large EPL audience.

Bot development

Data input

To be able to generate articles about EPL, good structural data needs to be provided to the algorithm. As mentioned in the method section, I decided for WhoScored.com as my main data provider. Because the website does not offer open source match data, nor has an API that would provide this service, the data needed to be scraped. Sample data of 20 EPL matches in the 2016/17 season was scraped with a script written in Python. When I started to collect data, the script only used BeautifulSoup, which resulted in that WhoScored.com detected the running scraper on their website and denied access. In order to overcome this I used either Incapsula cracker (Sanders, 2017) or Selenium which automates browser. The website detects them as a user who browses through it. In the who_scored_match_data scraper, the scraper loops through an array which contains links of a particular match. The scraper then locates the script tag that contains the matchCentreData variable. This variable contains all the data and events about a match. Then the data for each match is saved to a separate file as a JSON string.

To add linguistic variety and additional information to the generated article also the whole player information database from the same website was scraped. In this case Selenium was used since Incapsula cracker did not work properly due to the large database to be scraped. Selenium opens a browser window and navigates to the EPL site on WhoScored.com. There it locates the league table and finds all the team links in the table which are then stored in an array. Then the driver loops through that list and finds all the player links which are stored in another array. At the end, it loops through the players link list, opens each link, extracts information needed and stores it to a csv file.

In order to add table standings reference as a conclusion of the generated article, a table after a finished match day also had to be scraped. In this case, I used the official EPL website, premierleague.com. Selenium is also used in this case to avoid detection. The script locates table rows within the tbody tag with the class tableBodyContainer. Then it gets every second row. The reason for this is that after each standings row there is a row with expandable information which is not useful in this case. Then the scraper loops through every second row and finds the needed information: name of the club, position, how many matches they played and number of wins, defeats and draws. I noticed when comparing names of the clubs from this data with names of the clubs from the WhoScored.com data, that some are different. Premierleague.com uses official, longer names. In order to pair these data, names in this scraper had to be changed to be equal to those from the WhoScored.com data. After this is done, a JSON string for each club with the needed data is composed and saved to an array. In this way we get a list with JSON strings containing table information for each match day. After a table has been processed each array is saved in a separate folder.

Two other scrapers were developed, one to scrape the headlines of match reports and the other to scrape match reports. In this way I manually went through them and I could analyse which are the most important aspects of a match, which terminology is used, and was able to shape the templates that the algorithm will use.

The headline loops through the Guardian match report pages. It searches for every link to a match report and extracts the text of the headline from it. Then all of the headlines are written to one text file.

Second scraper takes the ESPN website and loops through the links of match reports. In each link it searches for a div tag with the class article-body. It then scrapes every paragraph from it and writes it to a text file. I chose to scrape the reports from ESPN as they focus on event description in a match rather than Guardian, BBC and other which focus more on post-match interviews.

The algorithm

First stage

In the first stage of FMR development, I focused on developing an algorithm that returns a headline and the main body of text which includes description of who and when scored, and, if someone assisted, also the assistants name. This stage is also the backbone of the algorithm as every next stage is based on it.

The algorithm opens the JSON file which stores match data. After the data is loaded, it creates a dictionary with the data that can give some general information about the match: which is the home and which the away team, how many goals each team scored and what was the final score. This information is essential for the decision tree when deciding which template to use. Some other information is also stored in the dictionary which helps with the description of the match and adds some variety to the article: the name of the stadium, names of home and away manager and how many people attended the match.

In the next step of the first stage, the algorithm loops through the events in match data and matches goal events. It extracts the player id associated with. Then it matches the id with the player in the name dictionary which is included in the match data. It then stores the names in an array.

```
#list with names of goalscores
score_names = []

for i in d["home"]["incidentEvents"]:
    #finding the goal scorers
    if i["type"]["displayName"] == "Goal":
        scorer_id = str(i["playerId"])
        score_names.append(d["playerIdNameDictionary"][scorer_id])

    else:
        pass

for i in d["away"]["incidentEvents"] :
    #finding the goal scorers
    if i["type"]["displayName"] == "Goal":
        scorer_id = str(i["playerId"])
        score_names.append(d["playerIdNameDictionary"][scorer_id])

    else:
        pass
```

Figure 1: Creating arrays

The next step was to find when the goals were scored and which team scored. I approached this again with a for loop that goes through the match events data and stores the data in two separate arrays. For the simplicity later in the decision tree, the teams are coded with 1 for home and 2 for away team.

The last step was to look if there were any footballers involved as assistants in any of the goals. The process is similar to the one where it looks for goal scorers. The only problem is if

there is not any assistant. Then the algorithm adds "no assist" to the array as the match data only provided information if there had been assistants.
Until this step, the data in all of the arrays mentioned above is stored according to the team. But in order to generate the match report, the data in the arrays needed to be ordered by the time the events occurred in the match.

```
#sorting the arrays by the time the events occurred
try:
    #a_s: sorted score_time
    #b_s: sorted score_names
    #c_s: sorted score_assist
    #d_s: sorted score_team
    a_s, b_s, c_s, d_s = map(list, zip(*sorted(zip(score_time, score_names, score_assist,  score_team), reverse=False)))
except:
    pass
```

Figure 2: Sorting arrays by the time events occured

With the arranged data there was one more aspect that needed to be developed. Usually if a player performed really well in the match, scoring multiple goals and had some assists, he is mentioned in the headline as the man of the match. This is why I created another group of arrays to find out who was the best player in a match which did not end goalless. The algorithm loops through the goal scorer list and it counts how many times the name appears in the goal scorer array. This step tells us how many goals every goal scorer scored. It appends the name to the man_of_match array and number of goals to the num_goals array. To see who was the best player, each goal is multiplied by 5 and the score stored in the points array. Also the assistants are graded similarly only that the assist is worth 2.5 points, half of the goal. If the player reaches a score of 15 or more, he and his performance will be mentioned in the headline. If it is less than that, the headline will feature sentences that mention the teams, managers or fans.

Sample of a headline mentioning player's performance:
Magical performance by Harry Kane, scoring 4 goals

Sample of a headline mentioning teams, managers or fans:
Josep Guardiola's side wins at Old Trafford with 2:1

The headline function generates a title for the article. Templates are stored in different arrays, depending on the final score: win of the home or away team, goalless draw or draw with goals and templates about player's performance. The generator follows a decision tree, depending on the final score. When the outcome of the match matches the condition, it then randomly selects a headline from the array and fills it with the information needed. Then the function returns the title.

The goals function generates the main body of the article. I defined two score counters, one for the home and one for the away team that will be useful when referencing to scores after each goal. The scores are updated after each for loop. There is also the ret variable to which the algorithm adds the strings after each for loop. In the for loop I followed similar development as in the headline function. Template sentences are stored in arrays, each array stands for a specific situation in the match of what the goal meant for each of the teams and the score: if it was the first goal in the match, first and also the last, if it meant an increase, decrease or draw of the score and who scored the last goal in the match.

Looping through the array with the scoring teams, the function follows a decision tree. When the condition is met, it randomly chooses a prewritten template, fills it with information needed and adds it to the ret string defined before the loop.

```
#advance the lead
elif (score_away - score_home >= 1) and c_s[i] != "no assist":
    ret += scored_assist_advance_away[s]
```

Figure 3: Adding random template to ret string

When the for loop finishes the iterations, the function returns the written main body of text. The content function returns the result of the goals function, or if the match ended in a goalless draw, it returns a predefined template. The match report is assembled in the article function. The function runs the headline and goals functions, creates a line space between the headline and main body and returns the match report.

After running the whole algorithm for the first time, I could notice that the minutes the match data provides are in cardinal numbers. After analysing match reports, I found out that they use ordinal numbers when describing match events. For changing integers to ordinals, the ordinal function was developed. It takes an integer and changes it into a string. It then follows a number of if…else statements to see which number was passed to the function and returns the appropriate ordinal number into the template.

```
#function that return an ordinal number
def ordinal(integer):
    #integer to string to check which number was passed in

    num_string = str(integer)

    if num_string == "1" or num_string == "-1":
        return num_string+"st"
    elif num_string == "2" or num_string == "-2":
        return num_string+"nd"
    elif num_string == "3" or num_string == "-3":
        return num_string+"rd"

    elif num_string[-1] == "1" and num_string[-2] != "1":
        return num_string+"st"
    elif num_string[-1] == "2" and num_string[-2] != "1":
        return num_string+"nd"
    elif num_string[-1] == "3" and num_string[-2] != "1":
        return num_string+"rd"

    else:
        return num_string+"th"
```

Figure 4: Function for creating ordinal numbers

Generated article after first stage

Magical performance by Harry Kane, scoring 4 goals

Home fans were not delighted when Harry Kane scored the first goal in the match. He netted the ball in the 25th minute with an assistance by Son Heung-Min. Home fans were not delighted when Dele Alli and Son Heung-Min lead Tottenham into an advantage in the 36th minute with 0:2. Ben Chilwell netted the ball for Leicester in the 59th minute, trailing 1:2 behind Tottenham. In the 63rd minute Harry Kane shot Tottenham in advance, leading 1:3. Son Heung-Min scored in 71st minute with the assist of Harry Kane for 1:4. Home fans were not delighted when Filip Lesniak and Harry Kane lead Tottenham into an advantage in the 89th minute with 1:5. Harry Kane settled the score of the match after an assist by Ben Davies in 92nd minute. Leicester 1 Tottenham 6.

Second stage

In the second stage of developing FMR I tried to improve the algorithm by adding some additional features. By analysing match reports, I found out that they all explain what the end result meant for the teams. They can gain, lose or stay in the same place on the league table. I wanted to add this information after the main text part about match events, to be some sort of conclusion of the article.

As I described above in the data input section, the tables for each match day were scraped from the official EPL web site and stored in JSON format.

The algorithm automatically matches the match day table to the match data and opens it.

The developed table function returns the conclusion about the standings. After the current match day data is opened, the function also opens the table for the previous match day if it is not the first match of the season. In this way, the article produced can give the readers an insight what the result brought to the teams, how many places did they win or lose and what is the point difference with the previous round. After the tables are opened, the function creates an empty dictionary which will store information to be inserted into templates. These include: position for home and away team after the match, position for both teams in the previous round, points after the current match and points on the table in the previous round.

It loops through every team in the match day table. When the name of the club in the table matches the name of the club currently playing, it appends the table position and points to the dictionary created before. In the same manner, the information about the previous round league standings is gathered.

The function runs similar as the headline and goals function as it follows a decision tree. First it looks what was the outcome of the match: home team win, away team win or a draw. Then for each of these outcomes the algorithm finds out what the result meant for the teams: if they advanced, dropped or stayed on the same position on the league table. Afterwards the appropriate template is returned. The templates use also the ordinal function, developed in the previous stage, to return the correct format of numbers when writing about table positions.

After testing the function a problem occurred, because some of the teams were playing the first match of the season. In this way, no referral to the previous round could be made. Also the explanation of the standings after the match does not tell anything important yet in the terms of the season and position as there were many teams with the same amount of points. This is why the function was updated to look if it was the first match day of the season and if, choose appropriate templates that do not refer to league position.

During the analysis of the produced articles in the second stage I found out that a grammatical function is needed. As the team's difference on the table can be one or multiple places or points and it would take a lot of effort to predict all I developed the plural function. It is run from the template. It takes two arguments, a number, either one referring to the position or points and a string in singular form. It then checks which number it is and returns the string in plural or singular.

```
#function that return singular or plural form to be used when describing standing in league table
def plural(num, string):

    if num == 1:
        return "{} {}".format(num, string)
    elif num == 0:
        return "{} {}".format(num, string)+"s"
    else:
        return "{} {}".format(num, string)+"s"
```

Figure 5: Plural function returns singular or plural form

Generated article after second stage

Harry Kane seals points for Tottenham against Leicester

Home fans were not delighted when Harry Kane scored the first goal in the match. He netted the ball in the 25th minute with an assistance by Son Heung-Min. Home fans were not delighted when Dele Alli and Son Heung-Min lead Tottenham into an advantage in the 36th minute with 0:2. Ben Chilwell netted the ball for Leicester in the 59th minute, trailing 1:2 behind Tottenham. Harry Kane lead the away team into a 1:3 advance in the 63rd minute. Son Heung-Min scored in 71st minute with the assist of Harry Kane for 1:4. After an assist by Filip Lesniak Harry Kane shot Tottenham in advance, leading 1:5. Harry Kane settled the score of the match after an assist by Ben Davies in 92nd minute. Leicester 1 Tottenham 6.

Tottenham kept the 2nd place from the last round with the win, having 74 points. Leicester dropped with this defeat to 15th place with 37 points.

Third stage

In the third stage developing the algorithm I tried to improve the narrative of the match report. Until now, the templates of the match events were filled with full names of the players. When a player scores multiple times in a row, the algorithm will also refer to him multiple times with the full name, which means the article will read like machine produced. As I wanted more natural language, I decided to use additional information about the players. The data was scraped, as mentioned above, from WhoScored.com and stored in JSON format. The code goes through the b_s array that holds goal scorers and in each loop it compares the goal scorer with the previous one in the list. Depending on the result of the loop it appends to a new array either the full name, surname, how old is the footballer or his playing position.

```
#if there is more or one goal scored check if a player scored in a row to use alternative description
if info["score_home"] or info["score_away"] >= 1:
    #array that holds alternative names
    alt_sen = []
    for q,w in enumerate(b_s):
        for line in alternative_names:

            if line["full_name"] == w and q == 0:
                alt_sen.append(w)
            elif w == b_s[q-1] and line["full_name"] == w:
                #if all of the alternative namings were used, use surname
                if line["position"] in alt_sen:
                    alt_sen.append(line["surname"])
                #instead of player's surname use his position
                elif line["surname"] and line["age"] in alt_sen:
                    alt_sen.append(line["position"])
                #instead of player's full name use his age
                elif line["surname"] in alt_sen:
                    alt_sen.append(line["age"])

                else:
                    #instead of player's full name use his surname
                    alt_sen.append(line["surname"])
            #if the player is already on the score sheet but did not score in a row use his surname
            elif line["full_name"] == w and w in alt_sen:
                alt_sen.append(line["surname"])
            #use the full name if the player scored for the first time
            elif line["full_name"] == w:
                alt_sen.append(line["full_name"])


else:
    pass
```

Figure 6: Creating alternative naming

When the algorithm fills in the templates, another function is run when it is filling in information about who scored. In the templates the scorer_generator function is run. It accepts either a string either a number, the name of the goal scorer or an alternative of mentioning him. The function checks if the passed information was his age, position or name and returns the correct part of the sentence.

```
def scorer_generator(string):

    if type(string) == int:
        return "the {} year old ".format(string)

    else:

        if string[0].isupper():
            return string

        else:
```

Figure 7: Function returns correct part of the sentence

When developing this function I wanted to make use of spaCy, the NLP library for string processing. With its help the code would check the passed in string information and decide whether it is a person (passed full name or surname) or a playing position and return the correct form. During testing it was mostly working fine, however it had some problems. As many footballers in the EPL are foreign and spaCy supports only some languages, primarily English, some of the footballers, were not recognised. SpaCy had most problems with Slavic, African and Latin names, on the other hand it worked well with Asian names. As wrong results were returned, the previously described method was developed. When the function is run with the data, it returns the correct string which is then used in the template.

Generated article after third stage

Harry Kane helps Tottenham eclipse Leicester

In the 25th minute Harry Kane gave Tottenham lead over Leicester with an assist by Son Heung-Min. Son Heung-Min scored in 36th minute with the assist of Dele Alli for 0:2. It was in the 59th minute when Ben Chilwell reduced the gap to 1:2. In the 63rd minute Kane shot Tottenham in advance, leading 1:3. It was in the 71st minute when Harry Kane assisted Heung-Min who extended the lead for Tottenham to 1:4. Filip Lesniak assisted Kane who scored on the horror of home fans in the 89th minute. Leicester 1 Tottenham 5. the 24 year old settled the score of the match after an assist by Ben Davies in 92nd minute. Leicester 1 Tottenham 6.

Tottenham kept the 2nd place from the last round with the win, having 74 points. Leicester dropped with this defeat to 15th place with 37 points.

Fourth stage

Until the fourth and last stage, FMR produces a headline, describes events and refers to the league table.

In this stage I wanted to improve the writing of the main body of the article. Until now sentences were randomly chosen when they met certain rules of what the result meant for the scoring team. This also meant that the algorithm could choose same sentences in different places in the article or in a row.

To create a match report with a more natural flow I used spaCy again because it makes calculating similarity between strings vectors easy, only an api call is required.

Before moving to similarity calculations, I had to redevelop the goals function which returns the event sentences. When a criteria is met, the function enters into a while loop. The reason for this is that every time a sentence will be too similar to others already generated, the loop will run again for the same score and not moving forward. Also, the function is no longer adding sentences to the existing string but it appends them to the sent array. This step was developed to check similarity of the picked sentence with others in the list.

The similarity_check function accepts the generated sentence from the goals function and SpaCy is loaded with English vocabulary. The function loops through the sent array and checks the newly generated sentence with other sentences already in that list. Afterwards it returns the similarity coefficient and appends it to the similarity_coef array.

```
#function that calculates the similarity between sentences in the article
def similarity_check(sentence):
    #array that stores similarity coefficient
    similarity_coef = []
    #loading English language
    nlp = spacy.load('en')
    #loop through the sentences already accepted
    for u in sent:
        #the sentence passed into the function
        check = nlp(sentence)
        #previous sentences
        vector = nlp(sent[sent.index(u) -1])
        #append the similarity coefficient
        similarity_coef.append(check.similarity(vector))
    return similarity_coef
```

Figure 8: Checking similarity between sentences

In the goal function is then checked if all of the coefficients in the similarity_coef array are under the value of the similarity coefficient defined. If it turns out to be true, it adds the

sentence to the sent list. If not, the loop will run again with the same score and choose another sentence.

```
advance the lead
elif score_home - score_away >= 1 and c_s[i] != "no assist":
    similarity_checker = 1

    while similarity_checker == 1:
        #randomly choosing sentences
        sentence = scored_assist_advance_home[s]
        #checking if all sentences meet the coefficient criteria
        if all(g < 0.96 for g in similarity_check(sentence)) == True:
            sent.append(sentence)
            similarity_checker = 0
            i += 1
```

Figure 9: Updated code for choosing templates

When the outer while loop comes to the end, the function joins the sentences in the sent list and returns it as a string.

After testing the whole database, I noticed a bug as the algorithm did not recognise own goals. To solve this, I created a second array where the algorithm gets names of goal scorers. Now it also checks if the key isOwnGoal equals true. If this is the case it appends "yes" to the own_goal array, if not it appends "no". Also the goals functions had to be slightly redeveloped. It now checks the items in the own_goal array first, if it equals true it increases the score for the opposite team and returns an item from the own goal array of sentences, depending on which criteria is met.

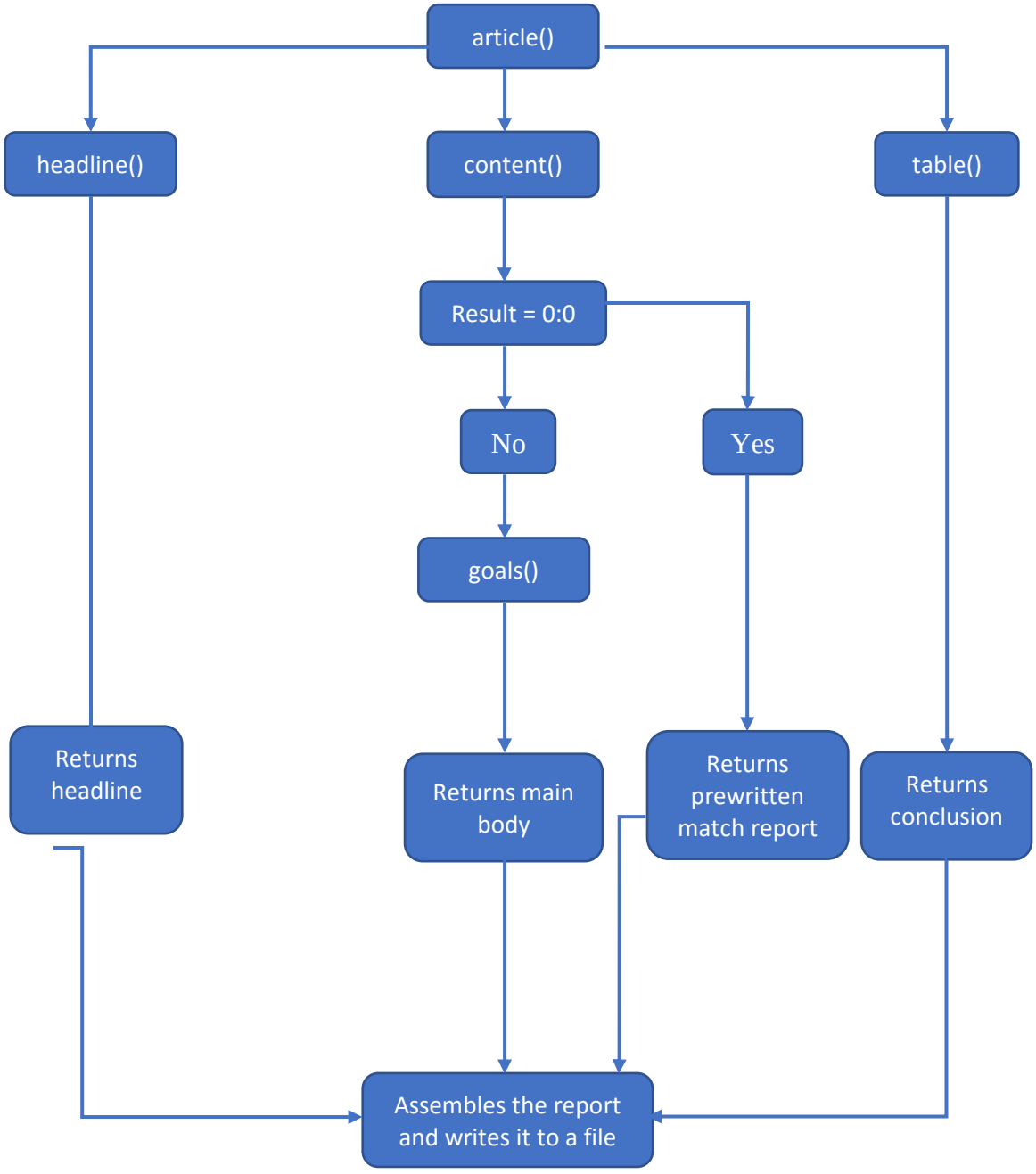
Generated article after fourth stage

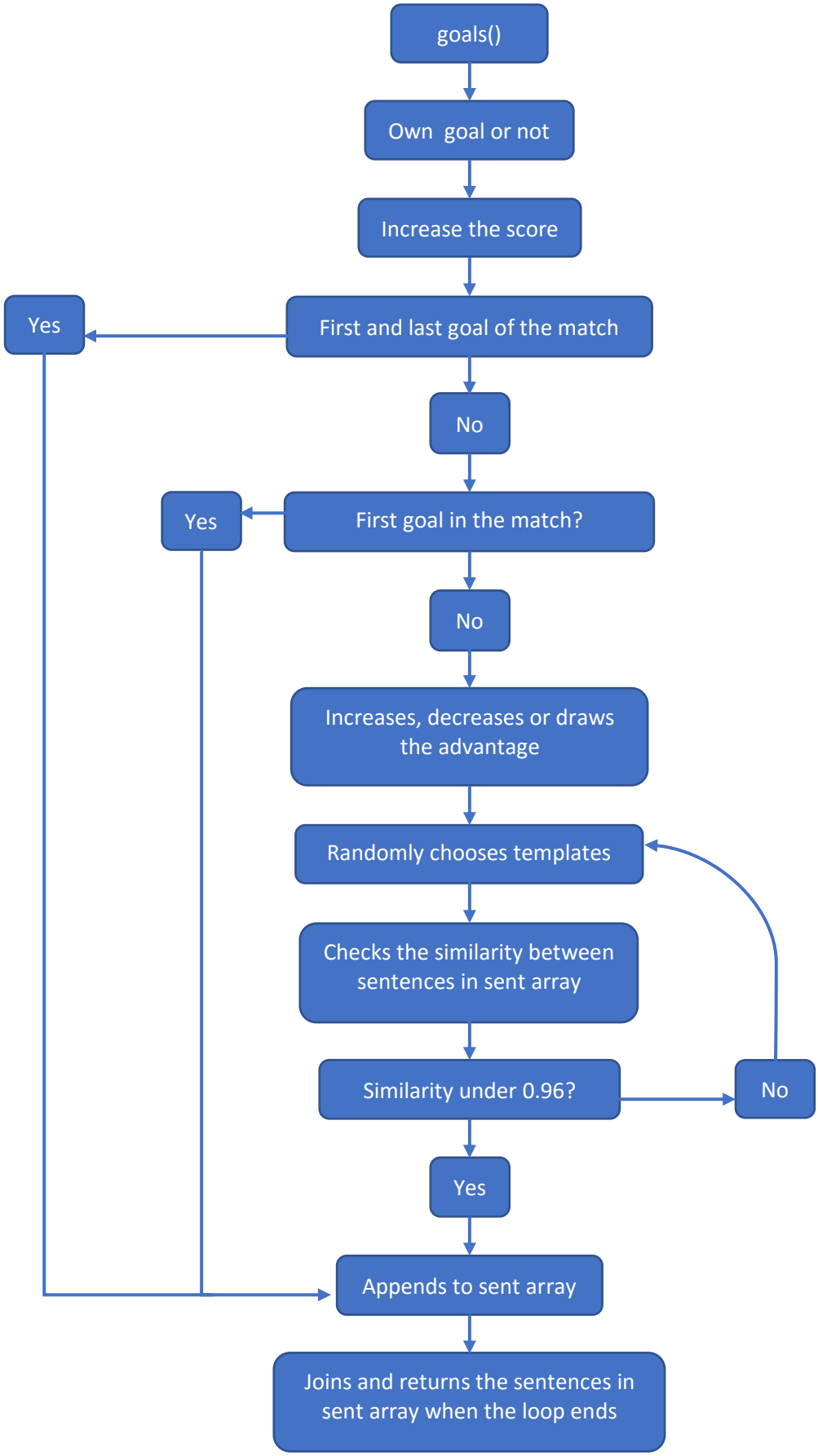
Harry Kane seals points for Tottenham against Leicester

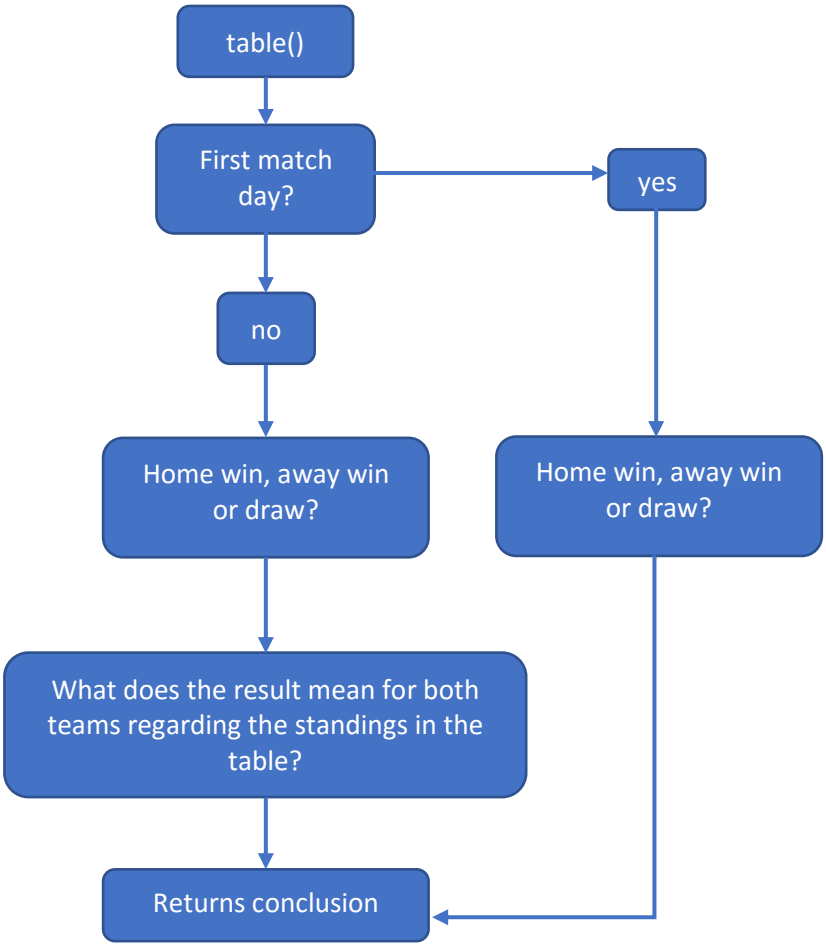
Home fans were not delighted when Harry Kane scored the first goal in the match. He netted the ball in the 25th minute with assistance from Son Heung-Min. The assist by Dele Alli was met by Son Heung-Min who shot Tottenham into 0:2 advance in the 36th minute. It was in the 59th minute when Ben Chilwell reduced the gap to 1:2. Kane scored in 63rd minute, putting Tottenham 1:3 in the lead. Harry Kane assisted Heung-Min who scored on the horror of home fans in the 71st minute. Leicester 1 Tottenham 4. Filip Lesniak prepared the opportunity for Kane who struck Tottenham in 1:5 advance in the 89th minute. The 24 Year Old scored the last goal of the match in 92nd minute who was assisted by Ben Davies. Leicester Tottenham 1:6.

Tottenham kept the 2nd place from the last round with the win, having 74 points. Leicester dropped with this defeat to 15th place with 37 points.

How FMR works - flowcharts




goals()
Own goal or not
Increase the score
First and last goal of the match
Yes
No
First goal in the match?
Yes
No
Increases, decreases or draws the advantage
Randomly chooses templates
Checks the similarity between sentences in sent array
Similarity under 0.96?
No
Yes
Appends to sent array
Joins and returns the sentences in sent array when the loop ends


table()
First match day?
yes
no
Home win, away win or draw?
Home win, away win or draw?
What does the result mean for both teams regarding the standings in the table?
Returns conclusion

Conclusion

The main part of the dissertation was to build and develop an algorithm which generates reports of EPL football matches. Most of the EPL matches are played during the weekend and at the same time. In information rich age there is no need to send reporters to cover every match as the algorithm can do the work for them.

For football reporters, it means that they will have to develop computational skills or focus more on human input stories, such as detailed analysis of matches, more quality interviews after or before matches and investigative news pieces of what is happening when English Premier League curtains are drawn after a match day. As (Graefe 2016, 10) states, journalists who cover routine topics, such as match reports, will likely be replaced by algorithms. This will also generate new jobs within the development of news automation. For the followers of the EPL who vividly read match reports, automated journalism can deliver error free content which can also raise the credibility of the media. Currently automated journalism can make content only from quality structured data which is demonstrated with FMR. As Graefe (2016) explains, when presenting facts, such as events in a football match, being quick and efficient is more important than a sophisticated narration.

With regards to this, FMR was developed, a templated based NLG algorithm that generates EPL match reports. It uses structural data provided by WhoScored.com and premierleague.com. Looking on other products already developed in this field and match reports written by journalist, gave an idea of how the articles should be generated and what structure articles should have. FMR consists of many parts which form an entity in the form of a written match report. The algorithm finds the necessary information in the data to be used in the article. After the creation of arrays which hold information about goal scorers, possible assistants, scoring teams, scoring times and goal type, a number of functions are run in order to write the match report. The headline function chooses a template from a group of headlines and fills it with information. To produce the headline templates, the actual headlines from match reports were scraped, from which the templates used were produced. The goals function is the main part of FMR as it produces the main body of the match report. Depending on the goal event, a template is randomly chosen from the groups of templates in this function and checked how similar it is to already generated content. This prevented the algorithm from choosing the same templates in the report. Finally, the table function returns a conclusion with regards to what the result meant for the opposing teams and for the standings in the league. All functions make use of grammatical functions, in order to produce an accurate report.

To conclude, automated journalism could be a benefit for news media when reporting on the EPL. Because the algorithm can write match reports in a cheaper, faster, more accurate way, and report on different matches at the same time, undoubtedly such approach will be soon used in British media.

Critical Reflection

The development of the algorithm was a major task for me as it combines development, data gathering and journalistic knowledge.

At the beginning of the course, I had some basic knowledge in coding which had to be significantly developed for producing a Master's thesis on this level. The developed algorithm gives the idea about what is possible in reporting EPL matches if structural data about football matches and events in it is available.

The development of FMR took most of the time of this dissertation as I had to learn new things in order to improve the algorithm. When I encountered a problem or a bug, I tried to solve it alone, or if I could not fix it, by searching for information on the web if somebody else had similar problems. I rarely found the needed information, as developing algorithms like FMR is new in the field. Because of this I tested different solutions I could think of until I found one that works.

After finishing the development stages, FMR can produce a match report with a headline, report on goals and the league standings of the teams involved in the match. However, the article generated after the fourth stage still has the sense of machine generated content, although far less than in the previous stages. This is especially seen in high scoring matches with 6 or more goals which could be solved by expanding the template arrays in the goals function.

To improve the algorithm, the goals function would need to be redeveloped. For now, the algorithm chooses a template from a group when it meets a rule. It is time consuming as the function has to loop through the criteria and match it with the score. This means that every possible event outcome in the match had to be defined.

The algorithm should recognise what is the outcome of the event by the data given, such as keywords, and choose the correct template and fill it with information from the data. In this way, other events in a match could be added, such as bookings and missed chances. In combination with an NLP library the templates could be altered as the library could search for

synonyms for frequently used words so that they do not repeat too often during the match report write up.

In order to further develop FMR and move from simple to advance NLG, the data should be processed through a NLP library so that FMR would produce an article with a defined bag of words and a realisation engine such as simpleNLG. With a Recurrent Neural Network which allows to model sequential data and makes a prediction at each time step (Sutskever et. all 2011, 2) the algorithm is fed with data about a match event. In this case it could make a decision of which words are needed from the corpus and produce an output with the help of the realisation engine which processes those words and returns a grammatically correct article.

In this way, a different version could also be generated, depending on which area the media organisation is from and which team is mostly supported there.

For now, the algorithm saves the article into a text file. The aim of using an algorithm instead of a journalist is also that it can write and publish articles by itself. In this case, the algorithm should be part of an application developed in Django framework. This would make it possible that the algorithm writes the story and publishes it directly on the website. In combination with a mobile app, push notifications could remind users that an article has been published, which would increase the reach for a specific media.

During the production of the algorithm I encountered some ethical issues by scraping the data from WhoScored.com as the data was not available open source. I know that producing this kind of detailed data takes a lot of effort and time and that it can only be available for payment. That is why the algorithm is for educational use only. I also used the data from the previous EPL season that was already finished at the time of writing this report, and I used a small sample of 20 matches.

Producing this project gave me an insight into what is needed to be successful in the field of computational journalism. Usually the work is divided between coders and journalists who collaborate to present a result. As I mentioned before, it was a lot of work for one person but I now definitely have an insight into what it takes to be a part of a project like this or even on a larger scale. I found out that in order to improve the project, I have to learn more about machine learning NLP and NLG, which would produce a cleaner article where readers could not distinguish between machine and human written work.

Bibliography

Allen, N. D., Templon, J.R., McNally, P. S., Birnbaum, L. and Hammond, K. J.. 2010. StatsMonkey: A Data-Driven Sports Narrative Writer. *AAAi Fall Symposium: Computational Models of Narrative.* Available at: http://www.aaai.org/ocs/index.php/FSS/FSS10/paper/download/2305/2842DRIVEN [Accessed: 30 August 2017].

Andrews, P. 2014. *Sports Journalism: a practical introduction.* London: Sage.

Automated Insights. 2017. *The Wordsmith Gallery.* Available at: https://wordsmith.automatedinsights.com/gallery [Accessed: 30 August 2017].

Bird, Steven, Klein, Ewan and Loper, Edward. 2009. *Natural Language Processing with Python.* [eBook version]. Farnham: O'Reilly Media. Available at: http://www.nltk.org/book_1ed/ [Accessed: 30 August 2017].

Butler, J. 1999. *Writing Sports Stories that Sell.* Oxford: How To Books.

Carlson, M. 2015. The Robotic Reporter. *Digital Journalism,* 3(3), pp. 416-431.

Clerwall, C. 2014. Enter the Robot Journalist. *Journalism Practice,* 8(5), 519-531.

Curley, James P. and Roeder, Oliver. 2016. English Soccer's Mysterious Worldwide Popularity. *Context* 17 March. Available at: https://contexts.org/articles/english-soccers-mysterious-worldwide-popularity/ [Accessed: 30 August 2017].

Dawson, Alan. 2017. This is how much prize money each Premier League club won in the most lucrative season ever. *Business Insider UK* 22 May. Available at: http://uk.businessinsider.com/total-prize-money-each-premier-league-club-won-in-lucrative-season-2017-5 [Accessed: 30 August 2017].

EliteDataScience. 2016. 5 Heroic Python NLP Libraries. *EliteDataScience,* Available at: https://elitedatascience.com/python-nlp-libraries [Accessed: 30 August 2017].

Glascott, Mary Grace. 2017. What is Natural Language Generation? *Narrative science.* Available at: https://narrativescience.com/Resources/Resource-Library/Article-Detail-Page/what-is-natural-language-generation [Accessed: 30 August 2017].

Graefe, A. 2016. *Guide to Automated Journalism.* Available at: https://doi.org/10.7916/D80G3XDJ [Accessed: 30 August 2017].

Intelligent Information Laboratory. 2014. *Project > Stats Monkey.* Available at: http://infolab.northwestern.edu/projects/stats-monkey.html [Accessed 30 August 2017].

Johri, A., Han, E.H.S. and Mehta, D. 2017. *Domain Specific Newsbots*. Available at: https://journalism.stanford.edu/cj2016/files/Newsbots.pdf [Accessed 30 August 2017].

Liberatore, Stacy. 2016. Your days could be numbered if you're a sports writer: The Associated Press is using AI to write Minor League Baseball articles. *Mail Online* 30 June. Available at: http://www.dailymail.co.uk/sciencetech/article-3668837/Your-days-numbered-sports-writer-Associated-Press-using-AI-write-Minor-League-Baseball-articles.html [Accessed 30 August 2017].

McCormick, R. 2016. AP's robot journalists are writing about Minor League Baseball now. *The Verge* 4 July 2016. Available at: https://www.theverge.com/2016/7/4/12092768/ap-robot-journalists-automated-insights-minor-league-baseball [Accessed: 30 August 2017].

Nichols, Nate. 2017. Natural Language Processing and Natural Language Generation: What's the Difference? *Narrative Science.* Available at: https://narrativescience.com/Resources/Resource-Library/Article-Detail-Page/natural-language-processing-and-natural-language-generation-whats-the-difference [Accessed: 30 August 2017].

Reynold, E. 2015. Wordsmith's robot journalist has been unleashed. *Wired*, 20 October 2015. Available at: http://www.wired.co.uk/article/wordsmith-robot-journalist-download [Accessed: 30 August 2017].

Sanders, Mark. 2017. *Incapsula cracker.* Available at: https://github.com/ziplokk1/incapsula-cracker-py3 [Accessed: 30 August 2017].

Sutskever Ilya, Martens, James and Hinton, Geoffrey. 2011. *Generating Text with Recurent Neural Networks*. Available at: http://www.cs.utoronto.ca/~ilya/pubs/2011/LANG-RNN.pdf [Accessed: 30 August 2017].

Van Deemter, Kess, Theune, Mariët and Krahmer, Emiel. 2003. *Real vs template-based natural language generation: a false opposition?* Available at: http://wwwhome.cs.utwente.nl/~theune/PUBS/templates-squib.pdf [Accessed 30 August 2017].

Waldal, E. 2016. Building a Robot Journalist. *B&B Stories* 18 November. Available at: https://medium.com/bakken-b%C3%A6ck/building-a-robot-journalist-171554a68fa8 [Accessed 30 August 2017].

WashPostPR. 2016. The Washington Post experiments with automated storytelling to help power 2016 Rio Olympics coverage. *The Washington Post* 5 August. Available at: https://www.washingtonpost.com/pr/wp/2016/08/05/the-washington-post-experiments-with-automated-storytelling-to-help-power-2016-rio-olympics-coverage/?utm_term=.0a4b246084f5 [Accessed 30 August 2017].

Weil, Eric, Joy, Bernard, Rollin, Jack, Giulianotti, Richard C. and Alegi, Peter Christopher. 2017. Football, Soccer. *Encyclopædia Britannica* 15 August. Available at: https://www.britannica.com/sports/football-soccer [Accessed: 30 August].

Appendices

Requirements

The dissertation project is written in Python using Jupyter Notebook. To install use **pip3 install jupyter** in terminal.

Appendix A: How to run scrapers

Requirements

When in terminal install following requirements:

Install Incapsula cracker
pip install incapsula-cracker-py3

Install Requests
pip install requests

Install Beautiful Soup
pip install beautifulsoup4

Install Selenium
pip install selenium

Match data scraper

Navigate to dissertation_code/data_scrapers in terminal and run jupyter notebook.

> cd dissertation_code/data_scrapers
> jupyter notebook

Select who_scored_match_data.ipynb

Player information scraper

Navigate to dissertation_code/data_scrapers in terminal and run jupyter notebook.

> cd dissertation_code/data_scrapers
> jupyter notebook

Select player_information.ipynb

In path_to_extension specify where ad_block folder is located. This helps selenium run with installed ad block.

Match day table data scraper

Navigate to dissertation_code/data_scrapers in terminal and run jupyter notebook.

```
> cd dissertation_code/data_scrapers
> jupyter notebook
```

Select player_information.ipynb

In path_to_extension specify where ad_block folder is located. This helps selenium run with installed ad block.

ESPN match report scraper

Navigate to dissertation_code/article_scrapers in terminal and run jupyter notebook.

```
> cd dissertation_code/article_scrapers
> jupyter notebook
```

Select espn_article_scraper.ipynb

Guardian match report headline scraper

Navigate to dissertation_code/article_scrapers in terminal and run jupyter notebook.

```
> cd dissertation_code/article_scrapers
> jupyter notebook
```

Select headline_scraper.ipynb

Appendix B: How to run the algorithm

When in terminal install spaCy using pip install spacy.

Navigate to dissertation_code/algorithm_stages in terminal and run jupyter notebook.

```
> cd dissertation_code/algorithm_stages
> jupyter notebook
```

Select algorith_final.ipynb or any of the previous development stages.

Final algorithm stage outputs are stored in articles_stage_final folder.