Extraction of Goals from Premier League Match Reports using Natural Language Processing Techniques

Student Project

presented by
Jochen Hülß & Matthias Rabus
Matriculation Number 1376749 & 1207834

submitted to the
Chair of Information Systems V
Prof. Dr. Christian Bizer
University Mannheim

Mai 2013

Contents

List of Figures

Chapter 1

Project Summary

1.1 Application Domain and Goals

In recent years our society underwent a remarkable change in terms of the ubiquitous availability of electronically-written text. The reader might think of the rise of e-ink interfaces and the subsequent shift from print media to electronic books, papers, magazines and so forth. Apart from editorial content, a massive development of informal information sources has been taken place.
On the one hand, the blogosphere puts the right of speech on an entirely new level. On the other hand, numerous possibilities have been emerging to customers reviewing, recommending and, thus, influencing the perception of products and services in a written way. Automatically retrieving structured information from these various sources of written text is yet a challenging task (Cellier et al., 2010, p.1). However, robust models in this domain can even help gaining a more transparent overview in emergency situations such as disaster relief by analyzing Twitter feeds.

Due these plentiful reasons, the domains of text mining and Information Extraction (IE) have drawn their attention to us. Both domains tied together enable the automatic identification of selected types of entities, relations, or events in free text (Grishman, 2005, p.545). Our research interest and also project goal is two-fold. Firstly, we want to find and apply patterns extracting a particular event from an unknown textual source. Secondly, the robustness of the patterns found should be assessed by classifying an unlabeled source. According to Weiss et al. (2005) the first task can be referred to as Information Retrieval because we provide "clues" that determine the matching documents. They also argue that the second task is considered as a Document Classification one.

More specifically, we want to exemplify written football match reports from England's Premier League to investigate whether the event of a goal can be mined for automatically. With the help of manually-found seeds, our hypothesis is that a model classifying a document according to the occurrence of a goal with an accuracy of at least 75% is achievable based on automatically-labeled training data. Moreover, the model found should also work on a sample unknown and unlabeled match report.

The research report is comprised of a description of the obtained data set [1.2], our various preprocessing steps [1.3], the applied web data mining methods [1.4] as well as an evaluation of our results [1.5].

1.2 Structure and size of the data set

Since many NLP tools work best with English language, we would like to increase our chance of success by using football reports of the English Premier League. The BBC sports department offers match reports for every game of the currently ongoing season 2012/2013. The amount of reports is sufficient for the purpose of this project. A Premier League season consists of 38 match days with 10 games each. BBC provides one report per game. The BBC sports department is a good data source because the reports are neutrally written and have no preference for any team. Furthermore, the used language has got a higher level and should be, compared to spoken word, easier to analyze.

The first step is gathering the data from the BBC homepage. Crawling the data will be described more detailed in the following section 1.3 about preprocessing. These data was the input for the following classification process. Thus we wanted to classify each sentence whether it contained a goal or not, we had to generate a sentence-wise input first. The result of our crawling process was one HTML file per report. Like every HTML file, our results had a nested tree structure including some JavaScript and CSS, as well. A problem occurred was that the text fragments themselves contained HTML tags, for example references to other reports. For extracting the plain text from the rest of the HTML documents, we employed XPATH. Then we had to divide each football text into the sentences themselves. This on itself is a challenging task, because a punctuation mark could be a non-sufficient separator, for example if we consider the name of the stadium "St. James Park". The extracted sentences will be a sufficient input for the further research. As we started the research several match days before the end of the Premier League season, our data set comprises a total of 342 football match reports.

1.3 Preprocessing

1.3.1 Defining Manual Patterns

As outlined in the previous section, the project started with gathering the data. We used RapidMiner to crawl the data for us. RapidMiner is an open source data mining tool developed by rapid-i.com. RapidMiner allows the user to define a so-called process, which is a combination of certain operators. The operators are connected by defined in- and outputs and can be parameterized to perform specific actions.
RapidMiner provides an operator to crawl the World Wide Web. BBC's Premier League result page is structured as follows: It has a main site from which every game report is liked. The reports are grouped by the day of the match. The first task of the crawler is to find the links to the game reports within the results page and then download each linked page into a seperate HTML file. Then the text has to be extracted from the HTML files. Therefor we use RapidMiner as well. The RapidMiner process takes all HTML files and extracts the plain text. This is achieved with the Cut Document-operator and an XPath-query. The extracted text is then written to one file for all reports because the Write as Text-operator can not write into multiple files.

Afterwards we had to do two steps in parallel. First we had to seperate the text file that contained the whole text from all reports and second we had to manually generate patterns that described goals. For both tasks some helpful methods were implemented in Java. To use the text for our classifier, it had to be splitted into sentences and each sentence had to be written into an seperate file. Accomplishing this task we used a build-in Java class: BreakIterator. The class provides a method that can cut a given text into sentences given a specific language, English in this case. The previous preprocessing step created some additional lines of text, for example for every new processed file, that also have been removed within this step. As mentioned before, the results of the text splitting were saved and written into seperate text files for further use, but also processed to find some sentences containing goals and some counter examples.

If we want to train a classifier to distinguish the sentences for us, we will first have to create examples. Thus, there are rarely two similar sentences describing a goal, you can find certain patterns if you analyze sentences using Natural Language Processing (NLP) tools. "Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications." (Liddy, 2001,

p.1) To construct sufficient patterns, we needed two elements of Natural Language Processing: Named Entity recognition (NER) and Part-of-Speech tagging (POS). POS assigns to each word its part of speech tag. It determines if it is a verb, noun, adjective, etc (Voutilainen, 2005, p.219). Whereas, NER determines if a word is an entity, for example a name, a place or a city. The Stanford NLP Group provides a useful Java library with many build-in language processing tools, including NER and POS tagging. Using this library, we were able to print out a sentence including named entities and part-of-speech tags. We did this for four match reports to manually detect patterns. In total, we were able to extract 14 generic patterns indicating the scoring of a goal. One of the inferred patterns is displayed in Figure 1.1.

[PERSON POS ∗ NN IN ∗ c o r n e r]

Figure 1.1: Manual Pattern for "Rooney's volley from Mata's corner."

1.3.2 Possibilities of Automated Pattern Matching

As the next preprocessing step, the manual patterns had to be applied on our entire corpus of football match reports in order to find more positive training data for the subsequent classification in an automated way. For this task a search engine capable of querying for n-grams consisting of POS tags, recognized named entities, free tokens, and an arbitrary number of wildcards is needed. A n-gram is a sequence containing n literals.

A comprehensive search engine for n-grams has been developed by Sekine and Dalwani (2010). Their published work depicts a search engine capable of querying up to 7-grams with POS tags, named entities, and free tokens. The user interface is conveniently web-based. However, their search engine is not publicly accessible and restricted to Wikipedia articles. Moreover, their solution lacks a support of an arbitrary number of wildcards.

A thorough analysis of alternatives, also taking an own implementation into consideration, lead us the way to GATE, the "General Architecture for Text Extraction". Cunningham et al. (2002, p.1) states about their tool that "[t]he GATE architecture has enabled us not only to develop a number of successful applications for various language processing tasks (such as Information Extraction), but also to build and annotate corpora and carry out evaluations on the applications generated." A more detailed look on GATE unveiled its powerful and extensive baseline function set. By default, GATE provides a pipeline of text processing resources for

common NLP tasks. For instance, a tokeniser, a sentence splitter, a POS tagger, a named entity gazetteer, and an orthomatcher discovering relations between entities and assigning new annotations to tokens (Cunningham et al., 2002, p.5). Each processing module can be separately added to a processing pipeline and then applied on language resources such as plain text, html, pdf, etc. employing GATE's GUI.

Apart from standard components, GATE exposes interfaces to extend its processing resources to user's requirements. One of these is the Java Annotation Patterns Engine (JAPE). JAPE was also introduced by Cunningham et al. (1999). According to them, "[a] JAPE grammar consists of a set of phases, each of which consists of a set of pattern/action rules, and which run sequentially". Sequential pattern mining (Agrawal and Srikant, 1995) incorporates the concept that sequences of elements can repeatedly occur in a data set and thus form patterns. This idea is essential for IE and makes JAPE grammar rules a viable solution to automatically label unknown language resources derived from manually-inferred patterns. "Patterns can be specied by describing a specic text string, or annotations previously created by modules such as the tokeniser, [POS tagger], gazetteer, or document format analysis" (Cunningham et al., 2002, p.5). A sample JAPE grammar rule depicting the manual pattern of Figure 1.1 is shown in Figure 1.2.

```
Rule:  Goal3
(
  ({Person.rule  ==  PersonFinal}|{Token.category  ==  NNP}
   |{Lookup.majorType  ==  country_adj})
  {Token.category  ==  POS}
  ({Token.length  >  1})*
  ({Token.category  ==  NN}|{Token.category  ==  NNS})
  {Token.category  ==  IN}
  ({Token.length  >  1})*
  {Token.string  =~  corner}
)
:goalSequence  —>
  :goalSequence.Goal  =  {kind=Goal,  rule=Goal3}
```

Figure 1.2: Jape Grammer Rule for "Rooney's volley from Mata's corner."

The final preprocessing step became necessary due to GATE's restricted export capabilities. GATE outputs its annotations only as XML tags. These indicate the starting and end character position of a positively-labeled sentence within the entire

text corpus. I.e., we were bound to parse this XML tree with respect to the goal entity. This purpose served the Java API for XML Processing and some routines to write each positive and a corresponding number of negative sentences into a file. After all, we faced an evenly-distributed set of automatically-labeled sentences.

1.4 Actual Web Mining

After generating the input for our model we had to find a good model for our pre-processed data. Therefor we experimented with three different approaches: Naïve Bayes, Support Vector Machine (SVM) and k-Nearest Neighbor (k-NN) classifier. The detailed results will be discussed in the following section. This section provides an overview of the structure of the processes. Each process starts with importing the data into the process. This operator includes three sub-operators, which split the text into words (Tokenize), then filter stop words and certain words like season, half and minutes. The last operator is configured manually. The next operator balances the number of inputs because there are more sentences containing no goal, then ones including goals. Following the balancing we used a so-called Validation operator in which the model was learned, applied to the data and the performance of our results were measured.

- Naïve Bayes
  The first approach used a Naïve Bayes classifier within the validation. "Bayesian classifiers assign the most likely class to a given example described by its feature vector. Learning such classifiers can be greatly simplified by assuming that features are independent given class[. ...] The success of naive Bayes in the presence of feature dependencies can be explained as follows: optimality in terms of zero-one loss (classification error) is not necessarily related to the quality of the fit to a probability distribution [... .] Rather an optimal classifier is obtained as long as both the actual and estimated distributions agree on the most probable class." (Rish, 2001, p.1)

- Support Vector Machine (SVM)
  The process remained the same but we changed the model to be a SVM. "Support vector machines are based on the Structural Risk Minimization principle from computational learning theory. The idea of structural risk minimization is to find a hypothesis h for which we can guarantee the lowest true error. The true error of h is the probability that h will make an error on an unseen and randomly selected test example. [...] SVMs are very universal learners. In their basic form, SVMs learn linear threshold function. [...] One

remarkable property of SVMs is that their ability to learn can be independent of the dimensionality of the feature space. SVMs measure the complexity of hypotheses based on the margin with which they separate the data, not the number of features." (Joachims, 1998, p.2)

- k-Nearest Neighbor (k-NN)
  The most successful attempt used the k-NN technique. "One of the oldest and simplest methods for pattern classification is the k-nearest neighbors (kNN) rule. The kNN rule classifies each unlabeled example by the majority label of its k-nearest neighbors in the training set. Despite its simplicity, the kNN rule often yields competitive results and in certain domains, when cleverly combined with prior knowledge, it has significantly advanced the state-of-the-art. By the very nature of its decision rule, the performance of kNN classification depends crucially on the way that distances are computed between different examples. When no prior knowledge is available, most implementations of kNN compute simple Euclidean distances (assuming the examples are represented as vector inputs). Unfortunately, Euclidean distances ignore any statistical regularities that might be estimated from a large training set of labeled examples." (Weinberger et al., 2006, p.1) We achieved the best results with a k of 50.

Adding the operator "Weight by Information Gain" and "Select by Weights" improved the results as well. The first operator assigns each element of our word vector a normalized value and the second operator selects elements with a weight that is higher then a determined threshold. In our case we achieved the best results using a threshold of 0.05. These two operators replaced the "Sample" operator and improved the results and the runtime of the process, due the size of the data has been reduced. After the improvement was done, we wanted to use the trained model on unseen data. Therefor we added a "Process Documents"- and an "Apply Model"- operator after the Validation operator. The disadvantage of this was that the results were the sentences itself and the assigned class, so we had to evaluate the results by hand.

1.5   Results and Evaluation

The results shown in this section were calculated on the data set introduced in Section 1.2. After the preprocessing steps the football match reports were split into 9,211 sentences, each of which is separately store as text file. Out of this corpus, the 14 JAPE grammar rules (see Subsection 1.3.2) were able to match and thus extract 758 sentences containing a goal in an unsupervised manner. Given an even

number of negative and positive seeds for classification, the following performance measurements base on a total of 1,516 input sentences. The negative seeds were taken randomly from the entire corpus excluding those sentences marked as positive.

At first, we attempted to perform the validation of classification models with the complete sentences as attributes. Thus, the classifiers employed 1,498 attributes. However, it turned out to be not a viable solution as none of the classifiers were able to predict a single positive sentence correctly. From these findings we derive that entire sentences are not similar enough to one another to serve as attributes for classification. Hence, the measurements in Subsections 1.5.1 to 1.5.3 were conducted with an additional sentence tokeniser and stop-word filter returning now 3,624 word-level attributes for classification.

1.5.1 First Iteration: Naïve Bayes

The measurements of the first iteration employing a Naïve Bayes classifier are exposed to the reader in Figure 1.3. Peculiarly-high is the overall number of positive prediction. Naturally, this yields in a good positive class recall but takes a loss in the negative class recall. Also the positive class precision is not as strong as expected taking into account that the patterns for positive seeds are automatically-labeled though explicitly and precisely defined. The Naïve Bayes classifier results in an accuracy of 64.58%.

	true neg	true pos	class precision
pred. neg	411	190	68.39%
pred. pos	347	568	62.08%
class recall	54.22%	74.93%	

Figure 1.3: Performance of Naïve Bayes classifier

1.5.2 Second Iteration: SVM

The performance of SVM's classification validation is shown in Figure 1.4. This result chart has considerably different characteristics from the previous one. As initially expected, the positive class precision is very strong. But by taking into account that the occurrence of positively-predicted sentences is by far fewer than in the Naïve Bayesian case, it is obvious that the positive class recall suffers from the good class precision. Thus, the overall accuracy of 70.59% is higher than in the first iteration, however, is not yet meeting the demand of our hypothesis.

	true neg	true pos	class precision
pred. neg	654	342	65.66%
pred. pos	104	416	80.00%
class recall	86.28%	54.88%	

Figure 1.4: Performance of Support Vector Machine classifier

1.5.3 Third Iteration: Nearest-Neighbor

Figure 1.5 depicts the performance sheet of the 50-Nearest-Neighbor method. Strikingly-high is the class recall of positive sentences. As in the first iteration, the 50-NN method buys this figure with trading off its class precision which, in turn, is comparatively low. This high number of positive predictions can be explained through the vast amount of attributes. Those are equally-important for Nearest-Neighbor's distance measurement and, thus, even player names and words such as "bench", "inspiration", or "tense" indicate for the algorithm goals because a positive seed contained them at least once. Nevertheless, the basic application of 50-NN is the first model that holds the hypothesis of an accuracy of at least 75.00%.

	true neg	true pos	class precision
pred. neg	469	90	83.90%
pred. pos	289	668	69.80%
class recall	61.87%	88.13%	

Figure 1.5: Performance of Nearest-Neighbor method

1.5.4 Fourth Iteration: Weighted Nearest Neighbor

As argued above, the high number of attributes led to a too big positively-classified sentence set. This is because we introduced the Rapidminer operators "Weight by Information Gain" and "Select by Weight" (see Section 1.4). We determined that a selection of words with a weight greater than 0.05 yields the best performance. Whereby the top-5 word indicating a goal are "scored", "net", "twice", "corner", and "volley". This restriction only supplies a relatively small number of 235 attributes to the classifier but yet resulting in our best classification model validation shown in Figure 1.6. The table unveils a more homogeneously-spread result of precision and recall figures where all are fluctuating around 80%. This totals in our highest model accuracy of 80.88%.

	true neg	true pos	class precision
pred. neg	594	126	82.50%
pred. pos	164	632	79.40%
class recall	78.36%	83.38%	

Figure 1.6: Performance of Weighted Nearest-Neighbor method

1.5.5   Application on sample Match Report

The findings of the performance measurement in the previous subsections were for training and model validation purposes. To achieve the project goal, the models found are applied on an randomly-chosen, unknown, and unlabeled text, namely the match report of Chelsea vs. Sunderland on April, 7th 2013. We manually counted 24 sentences in the report where eight ones indicate a goal even though the game was only a 2:1 victory for Chelsea. Figure 1.7 draws the resulting measures for all applied classifiers. As found in the validation steps, the Nearest-Neighbor methods yield again the highest accuracies.

Classifier	Positive		Negative		Accuracy
	Precision	Recall	Precision	Recall	
Naïve Bayes	0.33	0.75	0.67	0.25	50.0%
SVM	0.33	0.75	0.67	0.25	50.0%
50-NN Basic	0.64	0.92	0.88	0.75	81.3%
50-NN Weighted	0.71	0.63	0.82	0.88	75.0%

Figure 1.7: Application of different investigated classification models on a sample report with 24 sentences (8 positive, 12 negative)

Figure 1.8 depicts a plot of this practical case. The best possible score is in the upper-right corner with a precision of 1 and also a recall of 1. Each classifier has got two data points - a positive and negative one. The best overall plot is scored by 50-NN Weighted negative. However, the best combined plot and, thus, the best accuracy for this specific match report scores the 50-NN Basic model. Yet this proves that our hypothesis holds also on unknown and unlabeled data.

1.6   Conclusion

The project paper outlined a way to detect goals in sentences of football match reports. It was possible to train different classifiers to achieve this goal. Based on the given data the k-NN method provided the highest success rates. An interesting
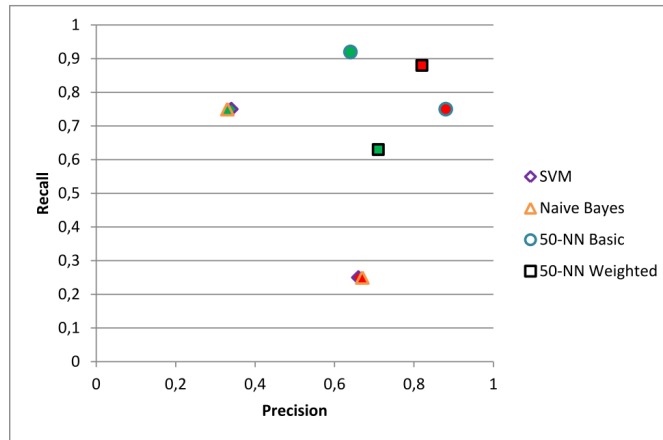
Figure 1.8: Plot of classification models applied on a sample report (green = positive, red = negative)

additional feature is the semi-automated generation of example sentences using GATE and JAPE grammar rules. They provided the opportunity to generate many examples from some manually-created rules.

We experimented with three different classifiers - Naïve Bayes, Support Vector Machine and k-Nearest Neighbor and different parameter settings. In the end we gained a precision of 76.5% and a recall of 75.5% for our k-NN process with $k = 50$ using the operators "Weight by Information gain" and "Select by Weights".

Possible challenges for further research could be the determination if a goal is described in multiple sentences or multiple times within a report to conclude the exact result from a given report. Furthermore, you could think about adopting the discussed techniques to other languages, for example German or Spanish.

Bibliography

Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 3–14, Washington, DC, USA. IEEE Computer Society.

Cellier, P., Charnois, T., Plantevit, M., and Crmilleux, B. (2010). Recursive sequence mining to discover named entity relations. In Cohen, P., Adams, N., and Berthold, M., editors, *Advances in Intelligent Data Analysis IX*, volume 6065 of *Lecture Notes in Computer Science*, pages 30–41. Springer Berlin Heidelberg.

Cunningham, H., Cunningham, H., Maynard, D., Maynard, D., Tablan, V., and Tablan, V. (1999). Jape: a java annotation patterns engine.

Cunningham, H., Maynard, D., Bontcheva, K., and Tablan, V. (2002). Gate: A framework and graphical development environment for robust nlp tools and applications. In *Proceedings of the 40th Meeting of the ACL.*

Grishman, R. (2005). Information extraction. In *The Oxford Handbook of Computational Linguistics*, pages 545–559. Oxford University Press.

Joachims, T., editor (1998). *Machine Learning: ECML-98: Text categorization with Support Vector Machines: Learning with many relevant features*, volume 1398.

Liddy, E. D. (2001). *Natural Language Processing.* Encyclopedia of Library and Information Science. 2 edition.

Rish, I. (2001). An empirical study of the naive bayes classifier. page 5.

Sekine, S. and Dalwani, K. (2010). Ngram search engine with patterns combining token, pos, chunk and ne information. In *LREC'10*, pages –1–1.

Voutilainen, A. (2005). Part-of-speech tagging. In *The Oxford Handbook of Computational Linguistics*, pages 219–232. Oxford University Press.

Weinberger, K. Q., Blitzer, J., and Saul, L. K. (2006). Distance metric learning for large margin nearest neighbor classification. In *In NIPS*. MIT Press.

Weiss, S., Indurkhya, N., and Zhang, T. (2005). *Text Mining: Predictive Methods for Analyzing Unstructured Information*. Springer-Verlag New York.