Final Project : DDI Extraction

Advanced Natural Language Processing

Martin Guy and Matthias Hertel

May 18, 2017

Abstract

In this project, we perform the two tasks of the SemEval competition that was held in 2013 about natural language processing in biomedical texts. The first task is drug recognition and classification and the second task is drug-drug interaction recognition and classification. For the first task, we used a Naive Bayes classifier with syntactic features to classify words. For the second task, using an SVM with a bag-of-words feature vector was not successful. We then switched to a Naive Bayes classifier with syntactic features, parse tree features and key words. In both tasks our approaches performed better than some of the approaches from the SemEval competition.

Table of Content

Introduction

According to the task description of the task in the SemEval 2013 competition [1] a drug-drug interaction is when a drug behaves differently in presence of a second drug. The detection of these interactions is important for patients safety since these modifications of behavior of drugs can be very dangerous and increase health care costs.

1 Description of the tasks

1.1 Task 1 : Drug Name Extraction and Classification

In this task, we have to make a model that recognizes and classifies drugs in a sentence.

We can find 4 different classes of drugs:

- **brand** : if it is a branded drug name;
- **drug** : if the drug is a generic drug name;
- **drug_n** : if the drug is an active substance not approved for human use;
- **group** : if it is a drug group name.

1.2 Task 2 : Drug-Drug Interaction Classification

In this task, we are asked to extract and classify interaction between pair of drugs. In the dataset, for each pair of drugs appearing in each sentence, we have two information about this pair: if it is said in the sentence that the two drugs interact (a boolean true/false), and if so, the kind of interaction.

We can find 4 different kind of drug-drug interaction (**DDI**) annotation [3]:

- **effect** : the effect of the DDI is described;
- **mechanism** : when the process by which drugs are absorbed, distributed, metabolised and excreted are affected is described;

- **advice** : a recommendation or advice is given about using the two drugs in the pair;
- **int**: it is only said that the pair is interacting but no information about the type.

Thus, considering the class **null** stating that there is no interaction between a pair, we have **5 classes**: **null**, **effect**, **mechanism**, **advice**, **int**.

We will also consider, as a way of testing our models, the **two-classes problem {interaction / no-interaction}**.

2 Preliminaries

2.1 Performance measures

We observed, looking at statistics of the dataset, that it is a very unbalanced dataset. For instance, considering the second task, the amount of pair of drugs that do not interact represents around 85% of the total number of pairs. Using accuracy as a measure on an unbalanced dataset leads to a bias towards the most frequent class. So if a classifier predict only the null class, it will get an accuracy around 85% but will be intrinsically be bad. This is why we thought that considering other measures would help in order to evaluate in a better way our models.

The measure that we selected is the one used in the SemEval competition, that is, the micro-averaged F1-score [7].

2.2 Naive Bayes algorithm

We implemented our own version of the Naive Bayes classifier in python.

Given a feature vector f, the Naive Bayes algorithm predicts the class $\hat{c}$ that maximizes the probability $p(class = c|f)$.

To compute the probability, Bayes' rule is used:

$$p(class = c|f) = \frac{p(f|class = c) \cdot p(class = c)}{p(f)}$$

To simplify the model, all features are assumed to be independent, and the formula becomes

$$p(class = c|f) = \frac{p(class = c)}{p(f)} \cdot \prod_{f_i \in f} p(f_i = v_i|class = c).$$

The probabilities are then estimated from training data:

$$p(feature = value|class = c) = \frac{count(feature = value|class = c)}{count(feature = any\_value|class = c)}$$

$$p(class = c) = \frac{count(class = c)}{n}$$

where n is the number of training samples.

A lot of probabilities are estimated as zero, which is a problem that can be solved by smoothing. In our work we use two different smoothing techniques:

- Laplace smoothing: see [2].
- epsilon smoothing: adding a small value to all probabilities and adapting the normalizer $p(f)$ accordingly (such that the class probabilities add up to one). In our case we chose $\epsilon = 10^{-6}$.

3 Task 1 : Drug Name Extraction and Classification

3.1 Baseline algorithm

3.1.1 Approach

We implemented a baseline algorithm for the task of drug extraction and classification for later comparison of the performance of our main approach. The baseline algorithm is simply to store all drug names seen in the training data. It thus does not contain any machine learning. An algorithm that is meant to be *intelligent* should at least perform better than this simple baseline strategy.

The baseline algorithm stores the drug names and according class labels of all entities seen during training in a dictionary. If the same drug name occurs several times with different labels, the stored class label is overwritten.

For extracting and classifying drug names in a sentence, the algorithm searches for occurrences of the stored drug names in the sentence. If a stored drug name is found in the sentence, the according substring is labeled with the class label that is stored with the drug name.

class	precision	recall	F1
brand	100.00 %	13.55 %	23.88 %
drug	66.97 %	61.25 %	63.98 %
drug_n	23.07 %	7.43 %	11.25 %
group	69.42 %	70.32 %	69.87 %

Table 1: Class-wise results of the baseline approach evaluated on the test documents.

For test purposes, we trained the algorithm on 70 percent of the training documents and evaluated the performance on the remaining 30 percent.
During this analysis we realised two problems of the approach:

- A lot of false positives were introduced when the words "drug" or "drugs" appeared in a sentence. Apparently these words are labeled at least once in the documents used for training, but among the documents used for testing there are multiple documents where they are not labeled.

- Some drug names are substrings of other drug names. Therefore multiple predictions are made on the same word. Clearly, not all of these labels can be correct at the same time.

To tackle the first problem, we simply forbid the algorithm to label the words "drug" and "drugs".

To tackle the second problem, we adapted the algorithm not to label substrings of drug names found in a sentence. This was done by searching for the stored drug names in order of descending drug name length, and forbidding to label a substring that overlaps with a substring that was already labeled before. This ensures that the algorithm only labels the longest substrings of the sentence, that are stored in the drug name dictionary.

3.1.2 Evaluation

Table 1 shows the class-wise precision, recall and F1-scores for the baseline algorithm evaluated on the test dataset. The *macro-averaged F1-score* is 48.02% and the *micro-averaged F1-score* is 56.31%. When only considering the task of drug name extraction (that is, disregarding the assigned labels), the F1-score is 59.78 % (64.95 % precision, 49.70 % recall).

As expected, the algorithm shows a better precision than recall. The bad recall comes from the fact that the test documents contain drugs that were not mentioned in the training documents. The precision is far from perfect, because of inconsistencies in the dataset. The same drug names sometimes appear with different labels in the training and test data. Other drug names are labeled in some documents but not in every document where they appear. Examples for false positives created because of inconsistencies in the test data (i.e. missing labels in the test documents) are *sulfonylurea, blood thinner, alcohols, ethanol, antibiotics* (these should probably be labeled in the gold standard and therefore not be false positives). Examples for false positives created because of inconsistencies in the training data (i.e. questionable labels in the training documents) are *oxygen, antibodies, cytotoxic, antibacterial* (these should probably not be labeled and therefore not be stored in the dictionary).

3.2 Naive Bayes approach

3.2.1 Approach and features

Our main approach is to predict the class label for each word of a sentence using the Naive Bayes algorithm with syntactical features.

The Freeling ([5]) tokenizer is used to split a sentence into words. The features used for each word are described in Table 2. Word lemmas and part-of-speech (POS) tags are extracted with Freeling. Note that the number of features of a word differs with the number of n-grams (which means with the length of the word), and the position of the word in the sentence (features for the words before and after do not exist if the word is the first or last word in the sentence).

The use of n-gram features was motivated by the observation that a lot of drug names look syntactically different from regular English words. This observation is supported by Figure 1, that shows the distribution of the 100 most common 3-grams in regular words in the training dataset, compared with their frequencies in the drug names. The 3-grams in English and in drug names seem to follow different distributions, which gives evidence that they are a good feature for a probabilistic classifier.

In order to find the best hyper-parameter setting, we evaluated the algorithm with different

feature name	value type	description
n-grams	string	The n-grams of the word. Tabbing was used before the n-grams were created ([6]).
word length	int	The number of characters of the word.
first uppercase	boolean	Whether the word starts with an uppercase letter.
last 's'	boolean	Whether the word ends with an 's'.
lemma 2 before	string	The lemma of the word two words before the word to predict.
lemma 1 before	string	The lemma of the word before the word to predict.
lemma	string	The lemma of the word to predict.
lemma 1 after	string	The lemma of the word after the word to predict.
lemma 2 after	string	The lemma of the word two words after the word to predict.
POS-tag 2 before	string	The POS-tag of the word two words before the word to predict.
POS-tag 1 before	string	The POS-tag of the word before the word to predict.
POS-tag	string	The POS-tag of the word to predict.
POS-tag 1 after	string	The POS-tag of the word after the word to predict.
POS-tag 2 after	string	The POS-tag of the word two words after the word to predict.

Table 2: Features used for the first task.

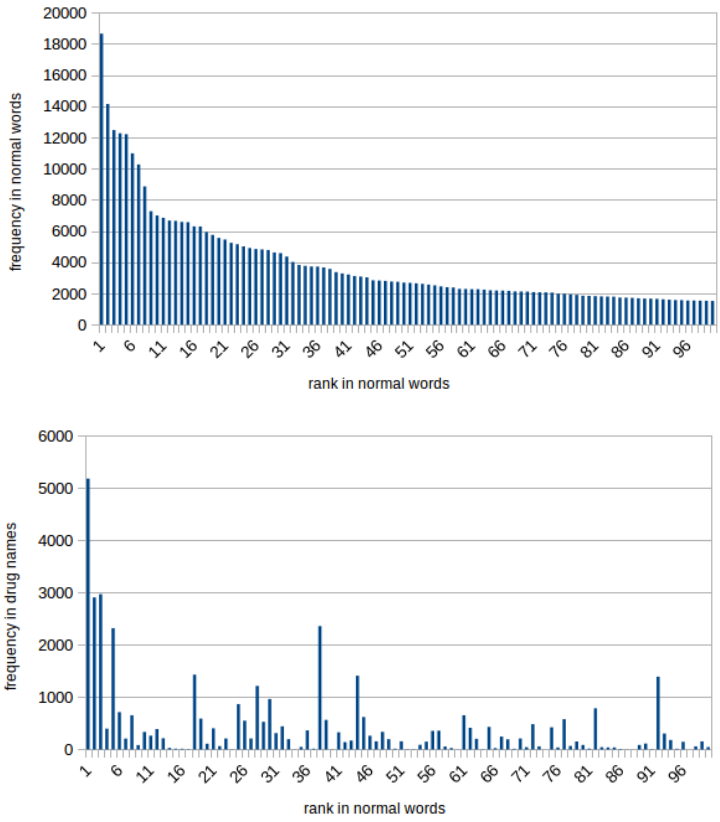

Figure 1: Distribution of the same 3-grams in English and in drug names.

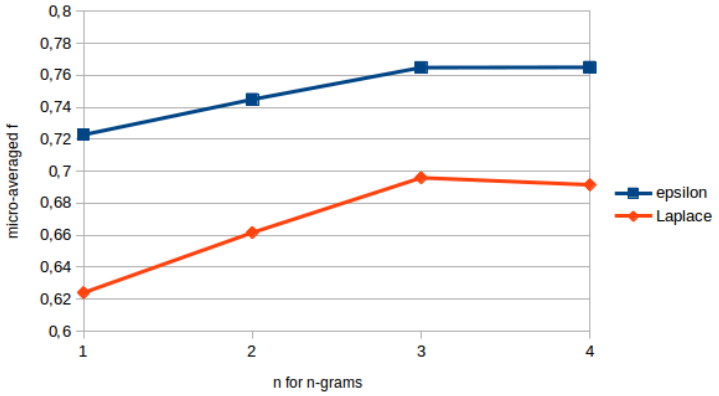

Figure 2: Micro-averaged F1-score for different values for n and epsilon- vs. Laplace-smoothing. The results are averaged over the folds of a 10-fold-cross-validation.

class	precision	recall	F1
brand	50.00 %	76.27 %	60.40 %
drug	60.63 %	81.19 %	69.42 %
drug_n	15.62 %	8.26 %	10.81 %
group	46.82 %	52.25 %	49.39 %

Table 3: Class-wise results of the Naive Bayes approach for task 1 evaluated on the test documents.

hyper-parameter settings in a 10-fold-cross-validation on the training set. Figure 2 shows the micro-averaged F1-score averaged over the folds. We observe that in general epsilon-smoothing leads to better results than Laplace-smoothing. The best scores were achieved with epsilon-smoothing and either 3-grams or 4-grams. Among these, we decided to use the simpler model, which is the 3-gram model.

3.2.2 Evaluation

Table 3 shows the class-wise precision, recall and F1-scores for the Naive Bayes approach evaluated on the test dataset. The macro-averaged F1-score is 48.23% and the micro-averaged F1-score is 56.77%. When only considering the task of drug name extraction (that is, disregarding the assigned labels), the F1-score is 64.59% (60.10% precision, 69.82% recall).

We observe that, regarding the micro-averaged and macro-averaged F1-measures and the F1-score of the extraction task (disregarding class labels), our approach beats the baseline approach, but the difference in performance between the two algorithms is small.

For the classes *brand* and *drug* and the extraction task the Naive Bayes approach is better than the baseline, whereas its precision is smaller for all classes and the extraction task.
Like the baseline algorithm, the Naive Bayes approach performs much worse for the class *drug_n* than for the other classes.

3.2.3 Improvements

Since our approach is to predict labels of single words, our system fails when a drug name is longer than one word. 124 out of the 318 drug names that were not found by our algorithm contain a blank character, which means they are longer than one word.
To deal with this problem, we propose the following possible improvements:

- concatenate consecutive words that are labeled with the same label
- train the same or another Naive Bayes classifier on two-word sequences (and three-word-sequences, and so on)
- or train another classifier, that, given that a word is labeled by our Naive Bayes classifier, predicts whether the word to the right (or to the left) has to be added

Sometimes, a drug name is only a substring of a word. For example, in the word "alcohol-addicted", only the substring "alcohol" refers to a drug. Therefore, in order to improve the algorithm's performance on these examples, it is necessary to predict labels for substrings of words.
One possible improvement is to split a word, when it contains a hyphen, and run the Naive Bayes algorithm on the substrings as well as on the whole string.

4 Task 2 : Drug-Drug Interaction Classification

Our first idea about this task was to use an SVM classifier and trying to find different feature selection strategies. After trying one not-working strategy, we tried a second one, without much better results. Then, as an SVM was too long to train, we decided to come back to the Naive Bayes, and also test it with better features based on NLP.

4.1 First approach : SVM with different feature selection strategies

4.1.1 MostFrequentBetweenStrategy

In this strategy, we consider only the text between a pair of drugs. For each sentences in each document, we count the occurrence of each word appearing between each pair of drugs.

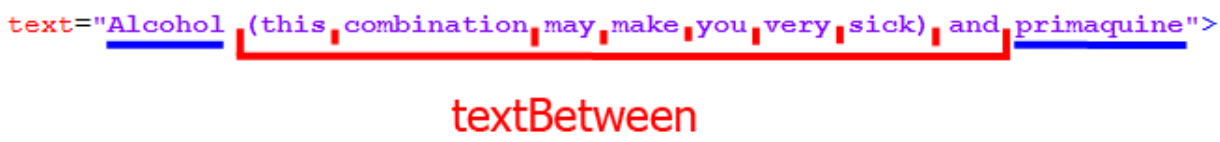

Figure 3: Text between

After getting this count, we fix a certain amount n of features that we will select. Then, we sort the words by order of occurrences, that we will denote by $(w_1, \ldots, w_k)$ where k is the total number of different words that we counted. Finally, we consider as features the most n occurring words, that is $(w_1, \ldots, w_n)$. Thus, given a pair of drug, we have a feature vector $(f_1, \ldots, f_n)$ with $f_i = 1$ if and only if w_i appears in the text between the pair.

We tried with the values of $n \in \{20; 300; 600; 1000; 10000\}$

4.1.2 EntropyStrategy

The previous strategy, no matter the number of features taken, led to really bad results. Indeed, we are considering all the words and most of the most occurring words are not relevant. They appear evenly in all the classes, and thus gather low information. Thus, our second idea was this time to gather the words that would carry the most information. That is, for each word in the text between a pair, we count its number of occurrences **given a class**. For example, in the figure 4, we know that given pair does not interact (so it is part of class null). Then we count 1 time each of the words between for the class null only.

After counting, we calculate the entropy of each word, using the formula giving the entropy of

a word w:

$$H(w) = -\sum_{c \in \mathcal{C}} p(w|c) * log(p(w|c)) \qquad \text{if } p(w|c) \neq 0 \text{ else } 0$$

where $\mathcal{C}$ is the set of classes.

We just get these probabilities by uniform counting. In order to not consider really not frequent words, we set a threshold t such that each word occurring less than this threshold is discarded. We tried with the values of $t \in \{30; 100; 500\}$

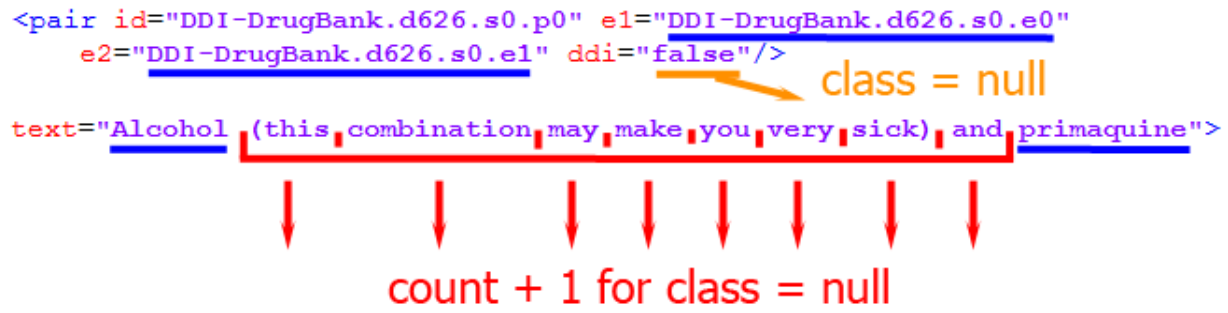

Figure 4: Example of the Entropy strategy

Like the previous strategy, we fix a certain amount n of features that we will select. Then, we sort the words by order of the lowest entropy and we consider as features the first n words with the lowest entropy. Again, given a pair of drug, we have a feature vector $(f_1, \ldots, f_n)$ with $f_i = 1$ if and only if w_i appears in the text between the pair.

4.1.3 Problem of this approach

By training an SVM with these features, we got a model that predicted only the null class. While having an accuracy around 83% (that is normal since the dataset is really unbalanced), the precision and recall was really bad. To solve this problem, we decided to put weight to classes (that impacts the C parameter of the SVM) such that it acts like all the classes are equally present. While the model did not predict only the null class, the precision and recall was still really poor. Thus, we decided that the features were not good enough and then headed toward NLP features. Meanwhile, as training an SVM was taking too much time, we also decided to get back to a Naive Bayes classifier.

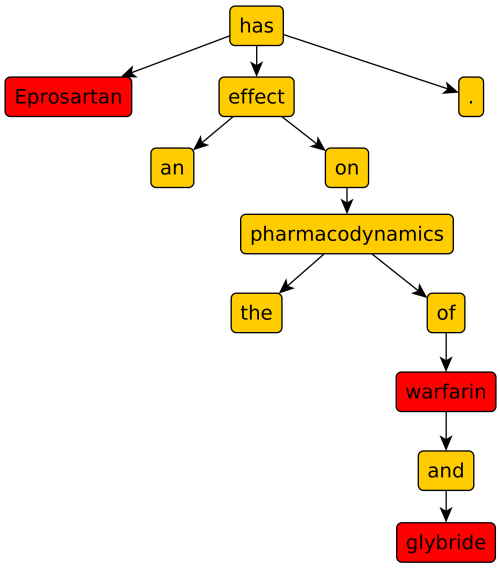

Figure 5: The dependency parse tree of a sentence containing three entities (marked in red).

4.2 Second approach : Naive Bayes with more features

4.2.1 Approach and features

Our second approach was to train a Naive Bayes classifier with more features. Like in the first task, we used epsilon-smoothing to account for feature values that were not seen in the training phase.

In order to distinguish between subject - object drug pairs and object - object drug pairs (which are unlikely to interact), grammatical features were created, using the sentences' dependency parse trees that were computed with Freeling. For each drug pair, the word and the POS-tag of the drugs' lowest common ancestor in the dependecy parse tree are used as features. Subject - object pairs are likely to have a verb as lowest common ancestor, whereas for object - object pairs conjunctions and nouns are more probable.

The lowest common ancestor is found by starting at the node of one of the entities and marking all ancestors in the tree until the root node is reached (the entity's node included). Then the tree is climbed upwards from the other entity's node, until a marked node is reached, which is returned as the lowest common ancestor.

Example: Figure 5 shows the dependency parse tree for the sentence *"Eprosartan has no effect on the pharmacodynamics of warfarin and glyburide."*, which contains the three entities *Eprosartan,*

feature name	value type	description
LCA tag	string	The POS tag of the lowest common ancestor in the dependency parse tree.
LCA word	string	The lowest common ancestor in the dependency parse tree.
words in between	string	The words between the two entities.
distance	int	The number of words between the two entities.
negation in between	boolean	Whether one of the words {*no, not, none, never*} occurs between the two entities.
sentence length	int	The number of words in the sentence.
sentence negation	boolean	Whether one of the words {*no, not, none, never*} occurs somewhere in the sentence.
key-word "increase(s)"	boolean	Whether the word "increase" or "increases" occurs in the sentence.
key-word "decrease(s)"	boolean	Whether the word "decrease" or "decreases" occurs in the sentence.
key-word "no"	boolean	Whether the word "no" occurs in the sentence.
key-word "not"	boolean	Whether the word "not" occurs in the sentence.
key-word "none"	boolean	Whether the word "none" occurs in the sentence.
key-word "never"	boolean	Whether the word "never" occurs in the sentence.
key-words "no effect"	boolean	Whether the words "no" and "effect" both occur in the sentence.
same entity	boolean	Whether the entities have the same name.

Table 4: Features used for the Naive Bayes approach for the second task.

warfarin and *glyburide*. The lowest common ancestor for the two subject - object pairs (Eprosartan, warfarin) and (Eprosartan, glyburide) is the word "has", but for the pair (warfarin, glyburide) it is the word "warfarin".

The extraction of these features took about 1 hour for the training dataset and 20 minutes for the test dataset. To speed up the following executions, the features were precomputed once and saved in a textfile for the following runs of the classifier.

In addition to the dependency parse tree features, syntactic features and key words were used, which are explained in Table 4. Note that the number of features for a drug pair varies with the number of words in between the drugs in the sentence.

Some features, that did not improve the micro-averaged F1-score (averaged during a 10-fold-cross-validation on the training documents), were excluded. These are explained in Table 5.

feature name	value type	description
drug1 type	string	The class label of the first entity.
drug2 type	string	The class label of the second entity.
drug names	string	The names of both entities.
sentence words	string	All words of the sentence.

Table 5: Features finally not used for the Naive Bayes approach for the second task.

class	precision	recall	F1
advise	36.97 %	52.03 %	43.23 %
effect	32.56 %	55.00 %	40.90 %
int	72.22 %	13.54 %	22.80 %
mechanism	37.25 %	55.62 %	44.62 %
none	*92.15 %*	*84.27 %*	*88.03 %*

Table 6: Class-wise results for the second task evaluated on the test documents.

4.2.2 Evaluation

Table 6 shows the class-wise precision, recall and F1-scores for the Naive Bayes approach evaluated on the test dataset. Without regarding the *none*-class, the macro-averaged F1-score is 44.39% and the micro-averaged F1-score is 41.74%. When only considering the task of drug interaction extraction (that is, disregarding the different types of interactions), the F1-score is 53.73% (45.82% precision, 64.96% recall).

We observe that our system performs worse on the *int*-class than on the other classes. Except for the *int*-class, we have higher recall than precision rates.

4.2.3 Improvements

Many more features could be designed in order to potentially improve the results, for example:

- considering a window of words before the first entity and after the second entity
- only using the most frequent or most discriminative words
- semantic features using WordNet and VerbNet
- semantic features using a domain-specific or general embedding technique
- using tf-idf scores as weightings of the occuring words

- using the words on the path between the entities in the dependency parse tree
- a lot more hand-designed key word features

5 Discussion

5.1 Comparison of the results

Task 1

Our approach for the first task achieved a micro-averaged F1-score of 56.77 %, which is slightly better than the baseline approach with 56.31 %.
Six groups competed on this task at the SemEval competition ([7]). Overall, sixteen approaches were evaluated (up to three approaches per group were allowed).
Our approach was better than five of the approaches from the conference. It did beat the best approaches of two groups, meaning that we would have placed fifth out of seven groups.
The best approach at the competition achieved a score of 71.50 %.

Task 2

Eight groups competed on the second task at the SemEval competition ([7]). They commited 22 approaches, where up to three approaches per group were allowed.
With a micro-averaged F1-Score of 41.74 %, our system achieved a better result than four of the approaches handed in to the competition. It did beat the best approach of one group, meaning that we would have placed eighth out of nine groups.
The best approach at the competition achieved a score of 65.10 %.

5.2 Conclusions

We encountered that the extraction and classification of drug names and drug-drug interactions are difficult tasks. Neither our approaches nor the state-of-the-art techniques from the SemEval competition achieve F1-scores better than 71.50 % and 65.10 % respectively.
Due to the unbalanced distribution of classes in the datasets and inconsistencies in the gold standard

labels, the tasks became even harder to solve. In both tasks, our algorithms behaved worst on the lowest-frequent classes (*drug_n* for task 1 and *int* for task 2).

Nonetheless, by defining features with the help of natural language processing techniques and training machine learning classifiers with them, it was possible to make many correct predictions on unseen examples.

Syntactic features like n-grams and features extracted from dependency parse trees led to good results, whereas simply using all the words from a sentence was not useful.

Our algorithms performed better than some of the approaches from the SemEval competition, but on both tasks we could not compete with the state-of-the-art techniques.

Many syntactic and semantic features could be thought of in order to further improve the results of our algorithms in the future.

The Naive Bayes classifier turned out to be more efficient and therefore more useful than Support Vector Machines in our specific case. Other classifiers could be tested and might lead to even better results.

When only considering extraction and ignoring the classification of drug names and interactions, the tasks became simpler and the results of our algorithms were better. It might therefore be useful to split the algorithms in two consecutive phases, where first the extraction is done and the extracted samples are classified afterwards in the second phase.

References

[1] SemEval Task 9 description

https://www.cs.york.ac.uk/semeval-2013/task9/

[2] ANLP, Lecture 1, Machine Learning Review

www.cs.upc.edu/~ageno/anlp/ml_review.pdf#page=33

[3] Task 9.2: Extraction of drug-drug interactions

https://www.cs.york.ac.uk/semeval-2013/task9/data/uploads/task-9.2-ddi-extraction.pdf

[4] Evaluation of the SemEval-2013 Task 9.1: Recognition and Classification of pharmacological-substances

https://www.cs.york.ac.uk/semeval-2013/task9/data/uploads/semeval_2013-task-9_1-evaluation-metrics.pdf

[5] Freeling

http://nlp.lsi.upc.edu/freeling/node/1

[6] Hannah Bast, Albert-Ludwigs-Universitt Freiburg, course "Information Retrieval", 2015, lecture "Fuzzy Search", slide 20

https://daphne.informatik.uni-freiburg.de/ws1516/InformationRetrieval/svn-public/public/slides/lecture-05.pdf#page=20

[7] SemEval-2013 Task 9 : Extraction of Drug-Drug Interactions from Biomedical Texts (DDIExtraction 2013)

https://www.cs.york.ac.uk/semeval-2013/accepted/76_Paper.pdf