CSE 498: Text Mining

Fall 2016

Extraction of Drug-Drug Interactions from BioMedical Texts

Sachin Joshi, Xinyang Zhang, Zhiyu Chen

Project Description

Drug drug interaction can be described as a change in the effects of one drug by the presence of another drug. The task of extracting drug drug interaction deals with,

- **Task 1:** The recognition and classification of pharmacological substances (drugs) and
- **Task 2:** Extraction of drug drug interaction from Biomedical texts.

These interactions can become very dangerous and increase the health care costs. The goal of this project is extraction of drug drug interactions from the DDI corpus.

The data set used for this project can be downloaded from https://www.cs.york.ac.uk/semeval-2013/task9/index.php%3Fid=data.html. The drug drug interaction (DDI) task relies on this DDI corpus. The DDI corpus is a semantically annotated corpus of documents describing:

- The drug drug interaction from the DrugBank database and,
- MEDLINE abstracts on the subject of drug drug interactions.

This corpus is intended for training Information Extractions (IE) systems that helps us to identify and classify pharmacological substances as well as extract drug drug interaction from the biomedical texts. The DDI corpus is provided in XML and contains two subdirectories:

- **DrugBank:** Contains 572 documents describing drug drug interactions from the DrugBank database.
- **MedLine:** Contains 142 abstracts on the subject of drug drug interactions.

The corpus has been manually annotated with pharmacological substances (drugs) and the interactions between them. An example of the annotated document is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<document id="DDI-DrugBank.d117">
  <sentence id="DDI-DrugBank.d117.s0" text="Concomitant use with iron supplements may result in the reduced absorption of iron.">
    <entity id="DDI-DrugBank.d117.s0.e0" charOffset="21-35"
      type="group" text="iron supplement"/>
    <entity id="DDI-DrugBank.d117.s0.e1" charOffset="78-81"
      type="drug" text="iron"/></sentence></document>
```

The following table summarizes the main features of the DDI corpus:

	DUGBANK Training			MedLine Training		
	Total	**Avg. Doc**	**Avg. Sentences**	**Total**	**Avg. Doc**	**Avg. Sentences**
Documents	572			142		
Sentences	5675			1301		
Drugs	8197			1228		
Brand	1423			14		
Group	3206			193		
No Human	103			401		
Total Drugs:	**12929**	**22.6**	**2.3**	**1836**	**12.9**	**1.4**
DDIs						
ddi	178			10		
advice	819			8		
effect	1548			152		
mechanism	1260			8162		
Total DDIs:	**3805**	**6.6**	**0.7**	**232**	**1.6**	**0.2**

Table 1

Drug interactions have been frequently reported in journals, making medical literature the most effective source for their detection. Therefore, Information Extraction (IE) proves to be of great benefit in the pharmaceutical industry, allowing identification and extraction of relevant information on DDIs and providing an interesting way of reducing the time spent by health care professionals on reviewing the literature.

The detection of drug-drug interaction from the biomedical texts has been conceived with a dual objective:

- Advancing the state-of-art of text mining techniques applied to the pharmacological domain and,
- Providing a common framework for evaluation of the participating systems and other researchers interested in the task.

Task 1:

Recognition and classification of pharmacological substances

The task of recognition and classification of drug names concerns the named entity extraction of mentions of pharmacological substances in text. This named entity task is a crucial first step for information extraction (IE) of drug-drug interactions. In this task, four types of pharmacological substances are defined:

- **drug:** Any chemical agent used in the treatment, cure, prevention or diagnosis of diseases which has been approved for human use. This type only represents generic drugs.
- **brand:** Any drug that was first developed by a pharmaceutical company.
- **group:** Any term in text designating a chemical or pharmacological relationship among a group of drugs.
- **drug_n:** Any chemical agent that affects living organisms. It's an active substance but it has not been approved to be used in humans with a medical purpose.

To accomplish the objectives of this task, I further divided my task into smaller subtasks. A detailed description of these subtasks is given below. I have used Java programming language to achieve the goals of my subtasks as well as the objective of task 1 as a whole.

Subtask 1

The first step is to extract all the text sentences from each of the 572 documents corresponding to the DrugBank data set. To accomplish this, I first combined all the 572 XML documents corresponding to the DrugBank data set into a single text file named **joined.txt** using the **Join Multiple XML Files Into One** software.

After combining all the XML documents into a single text file, the next step was to design a Java code that could extract all the text sentences from the text file **joined.txt**. The Java file named **extractSentences.java** is responsible for extracting all the text sentences. The program reads the text file **joined.txt**, extracts all the sentences and writes these extracted sentences in a new text file named **sentence.txt**. The program can be compiled in the command prompt using the following command:

javac extractSentences.java

After our program is successfully compiled, we can run our program and obtain our results using the following command:

java extractSentences

Note that both the files, i.e., **joined.txt** and **extractSentences.java** must be contained within the same directory. The final result is stored in the text file **sentence.txt**.

Subtask 2

After extracting all the sentences, the next step was to tag each of these sentences. These sentences are tagged using the Part-Of-Speech (POS) tagger. The tagged sentences are stored in a new text file named **Tagged.txt**.

Subtask 3

Since our primary task includes extracting the drug names from the text corpus, we have used CRF++ to accomplish our task. CRF++ is a simple, customizable and open source implementation of Conditional Random Fields (CRF) for segmenting/labeling sequential data. CRF++ is designed for generic purpose and is applied to a variety of NLP tasks, such as Named Entity Recognition, Information Extraction and Text Chunking.

Both the training file and the test file need to be in a particular format for **CRF++** to work properly. Generally speaking, training and test file must consist of multiple **tokens**. In addition, a **token** consists of multiple (but fixed-number) columns. Each token must be represented in one line, with the columns separated by white space (spaces or tabular characters). A sequence of token becomes a **sentence**. To identify the boundary between sentences, an empty line is put in. There are 3 columns for each token:

- The word itself.
- The part-of-speech associated with each word.
- The label corresponding to the entity type of each word.

Therefore, this subtask deals with the process of converting the tagged sentences contained in the file **Tagged.txt** into the CRF++ format. To convert the tagged sentences into a list of token, POS tag and entity type to train the model, I have designed a Java program that extracts each word and its corresponding POS tag from each of the tagged sentences contained in the file **Tagged.txt**. The file **extractWordsTags.java** contains the program for extracting the words and their corresponding POS tags. This program reads the text file **Tagged.txt** and writes the extracted words and their POS tags into two separate text files named **Words.txt** and **Tags.txt** respectively. Each word and its corresponding POS tag occupy a single line in their respective text files. The program can be compiled in the command prompt using the following command:

javac extractWordsTags.java

To run the program and obtain the required results we use the following command:

java extractWordsTags

Note, again the text file **Tagged.txt** and the Java file **extractWordsTags.java** must be contained within the same directory.

Subtask 4

The final step of converting the sentences into the CRF++ format is to assign a label to each word that corresponds to its entity type (drug, brand, group, drug_n). To accomplish this goal, I first extract all the pharmacological substances and their corresponding entity types (i.e., drug, brand, group, drug_n) present in each sentence. The Java file **type.java** contains the code that extracts the pharmacological substances and their corresponding entity type present in each sentence. The program reads the text file **joined.txt** that contains all the 572 XML documents corresponding to the DrugBank dataset and writes the pharmacological substances and their corresponding entity type (i.e., drug, brand, group, drug_n) present in each sentence in 2 separate text files, **subst.txt** and **entity.txt** respectively.

After extracting all the pharmacological substances and their corresponding entity type present in each sentence, the last step is to assign a label to each word present in the text file **Words.txt**. This task is achieved by the Java program **labels.java**. This program reads three text files, namely **Words.txt**, **subst.txt**, **entity.txt** and stores the contents of each of these text files in three different string type array. Then it does a string comparison where it matches each substance with each of the words and assigns it an appropriate label corresponding to its entity type. The following table describes the label of each word based on its entity type:

Entity Type of Word	Label
drug	D
brand	B
group	G
drug_n	D_n
Others	O

Table 2

The labels of each word are stored in a text file named **Labels.txt**. Finally the converted sentences in the CRF++ format are stored in the file **DB_CRF++.csv** and **DB_CRF++.txt**.

Subtask 5

As a general purpose tool, we also have to specify the feature templates in advance. Here we follow up the template format from the official website which is the same as ColNLL 2000. Specifically, unigram features within a window of size 5 and bigram features (current token and previous token) are used in task 1. The file **templates** is the template we use.

Subtask 6

The next task is to split the extracted sentences into training and testing data set. Both the training and testing data set is required to be in CRF++ format. I have used R programming language to partition the extracted sentences into two subsets namely training and test data sets. The R file named **partition.R** contains the code for partitioning the extracted sentences. We need to install the R programming environment so as to execute this file. However the file can be opened in any of the text editors such as Notepad++ in order to review the code. After running the program, the desired results have been categorized into 4 different files:

1. **Training.txt/Training.csv:** This file contains random partitioned sentences that contribute to approximately 80% of the entire extracted sentences and belongs to the training data set.
2. **Testing.txt/Testing.csv:** This file contains random partitioned sentences that contribute to approximately 20% of the entire extracted sentences and belongs to the training data set.

The next step is to convert each of the training and the testing data set into CRF++ format. This can be achieved by following the above mentioned steps, i.e., from **Subtask 1** to **Subtask 4**. The final result is stored in the following mentioned files:

- **Train.csv/Train.txt:** This file contains the training data in CRF++ format.
- **Test.csv/Test.txt:** This file contains the testing data in CRF++ format.

Results and Discussion

Training

After installing CRF++ toolkit and running the command **crf_learn -f 3 -c 1.5 templates Train.txt model** in the directory **task1**, it will begin to train the model and finally generate the trained model which is named as **model**. Here we could specify a lot of parameters and we specify the cutoff threshold (-f) as 3 and hyper-parameter C (-c) as 1.5.

Evaluation

As described in section subtask 6, we use 20% of the data as testing set. First, we run the command **crf_test -m model Test.txt > rs.txt** to predict the labels from testing set and write the results into **rs.txt**. Then after running **python task1_evaluate.py**, the evaluation results will be stored in file **task1_report.txt**.

We use precision, recall and F1 score as our evaluation metrics for both task 1 and task2. The results are summarized in Table 3. For drug and group, we have a good performance, while for Drug_n, we have the worst performance. The different size of training set and testing set for different name entities could be the reason why the performance of different classes varies. For

the Drug, which has the largest support have the best performance, while for Drug_n, the performance is extremely bad and its support is only 10.

	precision	recall	f1-score	support
B	0.69	0.53	0.59	257
D	0.9	0.83	0.86	1965
D_n	0	0	0	10
G	0.76	0.86	0.81	478
O	0.98	0.99	0.99	21612
avg / total	0.97	0.97	0.97	24322

Table 3

Task 2:

Extraction of Drug-Drug interaction

Drug-drug interaction (DDI) is an important topic in healthcare, which is defined as the change in the effects of one drug by the presence of another drug. In recent years, a lot of researchers begin to use machine learning techniques to solve related problems [1][2].

Actually, DDI belongs to the task of relation extraction which is one of the most important topics in NLP. In task2, there are four types of drug-drug interactions:

- **advise**: This category is assigned to those drug-drug interactions in which a recommendation or advice regarding the concomitant use of two drugs involved in them is described.
- **effect**: Describe the effect of a drug-drug interaction
- **mechanism**: Describe how a pharmacokinetic interaction occurs
- **int:** State that interactions occur and does not provide any information about the interaction, so none of the other types can be assigned.

Traditional methods of relation extraction often depend on the quality of designed features. Recently, deep learning has achieved the state of the art results in computer vision and speech recognition tasks. It also has become a hot topic in the NLP community. One of the reasons why deep learning is popular right now is because it rescues us from feature engineering to some extent.

Here, we explore the usage of convolutional neural network (CNN) in relation extraction tasks. Though we were originally planning to follow the paper [3], due to the limited time, we only use the simplest structure of CNN for this NLP task [4].

Preprocessing

For this task, we consider the following preprocessing. For each sentence in the corpuses, we first replace characters that represent interesting entities with special tokens that have a form of **__xxx__yyy__zzz__**. This helps to deal with entities that contains a space. Then we do the regular tokenization, decapitalize tokens, lemmatize tokens with WordNet Lemmatizer.

Now we get the sentences as a list of sentences which we fill do further process.

Word Embedding

Word embedding is a technique to get distributed representation of words. Such representation can capture more information than counting (e.g. bag of word), make it good for various tasks. The most popular algorithm for word embedding is Google's Word2Vec. It is based on Skip Gram method.

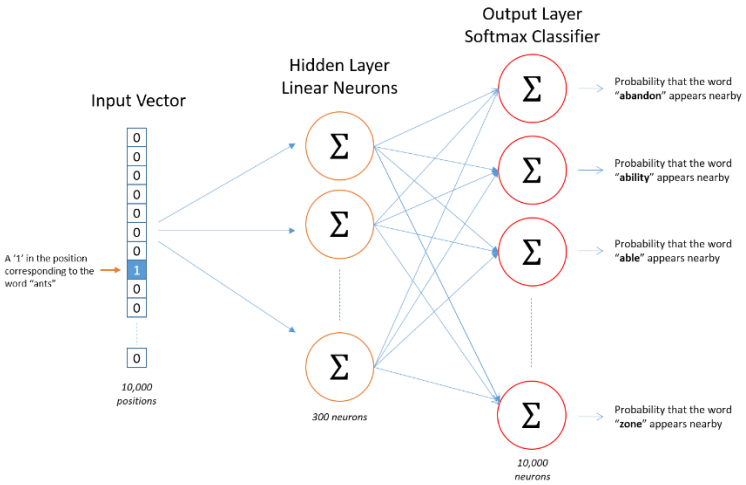

Figure 1

The above figure shows the neural network structure of skip gram method. The method can generate embeddings that capture the context of each token; while contexts are of great importance in relation extraction. For this task, we use the implementation of genism and consider the following parameters for word embedding:

Parameter name	Values
Dimension	50
Min word count	1
Maximum iterations	100

Table 4

All code for preprocessing and word embedding is in **vectorization.py**.

To run the code, in the command line, we have the following command:

```
$ python vectorization.py
```

Please make sure you put the **Train** under the current working directory.

CNN for relation extraction

We adopted the codes from Yoon Kim's github repository for our multiclass-classification problem. The CNN structure is described in Figure 2. The difference is (1) the input layer: we only use one channel which is our trained word embedding rather than multi-channnels described in the figure. (2) The output layer: we have five outputs (four ddi types and one "false'" type).

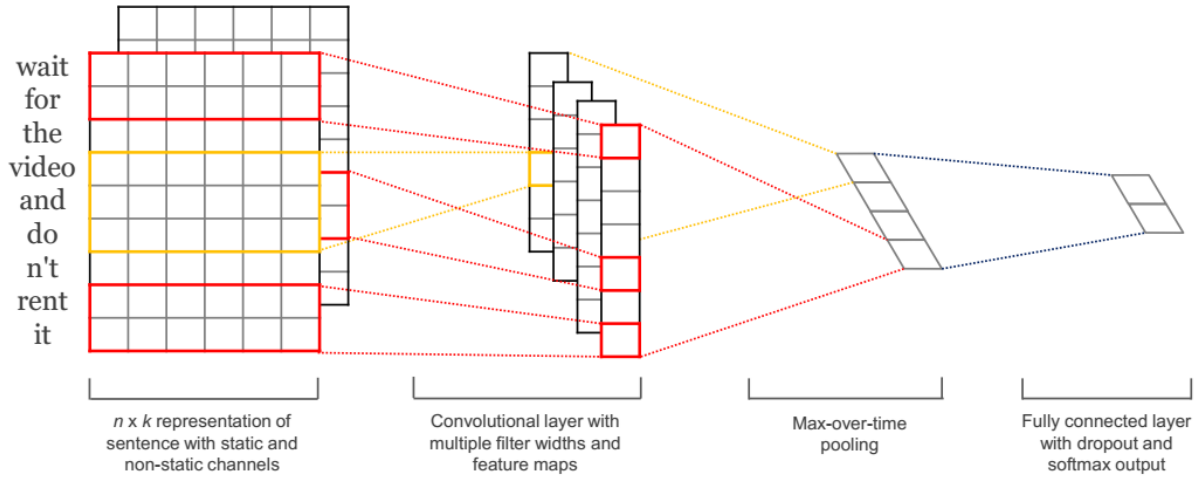

Figure 2: CNN for sentence classification from [4]

To run the code, in the command line, we have the following command:

```
$THEANO_FLAGS=mode=FAST_RUN,device=cpu,floatX=float32pythonconv_net_sentence.py
```

The evaluation results will be written into **log.txt**.

Results and Discussion

In this task, we use 2/3 data as training set and 1/3 data as testing set. We use the same parameters described in [4] and run 10 times to get the average performance as shown in Table 4. The training process is **extremely time consuming** with CPUs ! We present our best run in Table 5.

relations	precision	recall	f1-score	support
FALSE	0.85	0.99	0.92	6978
effect	0.54	0.14	0.22	494
mechanism	0.44	0.03	0.06	453
advise	0.58	0.12	0.19	285
int	0	0	0	63
avg / total	0.8	0.85	0.79	8273

Table 5

The same with task1, the performance of different relations varies. For **int**, obviously this simple CNN structure cannot handle with this case. For other types, though the results are far from satisficatiory, it is better than nothing. One reason for the poor performance could be losing information too quickly during the convolutional layer and the max-pooling layer. That's why there are some recent works trying to modify this structure [3][5]. We still believe that by encoding more information, like position embedding and other traditional features, neural network based methods will be useful.

References

[1] Yates, Andrew, and Nazli Goharian. "ADRTrace: detecting expected and unexpected adverse drug reactions from user reviews on social media sites." European Conference on Information Retrieval. Springer Berlin Heidelberg, 2013.

[2] Feldman, Ronen, et al. "Utilizing text mining on online medical forums to predict label change due to adverse drug reactions." Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015.

[3] Zeng, Daojian, et al. "Distant supervision for relation extraction via piecewise convolutional neural networks." Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP), Lisbon, Portugal. 2015.

[4] Kim, Yoon. "Convolutional Neural Networks for Sentence Classification."

[5] Nguyen, Thien Huu, and Ralph Grishman. "Relation extraction: Perspective from convolutional neural networks." Proceedings of NAACL-HLT. 2015.