Recognition and Classification of drugs names

Daniel Navarrete Jiménez
Master in Advanced Telecommunications Technologies
ETSETB - Polytechnic University of Catalonia
Barcelona, Spain
daniel.navarrete.jimenez@estudiantat.upc.edu

Àlex Castells i Arnau
Master in Advanced Telecommunications Technologies
ETSETB - Polytechnic University of Catalonia
Barcelona, Spain
alex.castells.i.arnau@estudiantat.upc.edu

***Abstract*—In this work we present two different approaches to do drug recognition and drug classification at DDI corpus. First of all, we present a rule based model, where we model the classification via rules we define while exploring the data. Afterwards, we develop machine learning algorithm based on sequence prediction to do the task.**

***Index Terms*—rule based model, machine learning, drug recognition, drug classification**

I. Introduction

In this assignment we were challenged to recognize and classify drug names. The data used to do that was the DDI corpus which is a semantically annotated corpus of documents describing drug-drug interactions from the DrugBank database and MDELINE abstract on the subject of drug-drug interaction. This corpus has been annotated with pharmacological substances and interactions between them. Classification and recognition are important fields in the state of the art of the Natural Language Processing field. This tasks can be approached in different ways, for instance we have carry out with two different approaches, the first one is a rule based model and the second one is a Conditional Random Field machine learning algorithm. To develop the classifier to be used in this domain we had to write a python program that parses all XML files in the folder, given as argument and recognizes and classifies drug names.

II. Part 1: Rule Based Classifier

In this section we are trying to do our classification task using a Rule Based algorithm. The program had to use simple heuristic rules to carry out the task. This is an assumption that will carry us some problems because we will not be able to obtain a really good performance because we are not using any learneable algorithm, we are just putting "if else" rules to classify the class of the word.

A. Dataset

First of all, we have to take a quick look at the dataset, the DDI corpus. This dataset is given to us divided in three folders. The development, train and test folder. This differentiation is not really useful here, because we are not training an algorithm with a large train set to be lately tested with different datasets and extract metrics.[2] In every XML file we find the same structure:

1) <document>: The root element of the XML, it has the following attributes
   a) id: An unique id which is composed by the name of the corpus and an identifier which is a number preceded by a letter D.
2) <sentence>: Each sentence of the document is contained with a sentence element.
   a) id: A unique id which is composed by the name of the corpus, the id of the document and the id beginning with a letter S and followed by the index of the sentence.
   b) text: contains the text of the sentence.
   c) entity: correspond to all annotated pharmacological substances.
3) <entity>: Annotated pharmacological substances, part of the XML file which will give us the labels to predict.
   a) id: A unique id which is composed by the name of the corpus, the id of the document, the id of the sentence and the id of the entity beginning with the letter E.
   b) text: contains the text of the entity.
   c) charOffsets: contains the start and end positions separated by a dash of the mention in the sentence.
   d) type: stores the type of the pharmacological substance: drug, drug_n, brand and group.

As we have mentioned just above, we want to obtain the classes of the pharmacological substances that appear in the XML sentences, so we have to extract all sentences in the field <sentence>: text and obtain the labels in the <sentence>: <entity>: type. Our goal will be obtain the words from the sentence and classify them in one of the four classes: drug, drug_n, brand and group.

B. Tokenizer

The tokenizer algorithm we tokenize all sentences in <sentence>: text field. We also obtain the initial position of the word in the sentence and the final position of the word in the sentence. We had to take special attention at the white spaces, because in the word tokenizer that we chose to use, the nltk.tokenize.word_tokenizer() the white spaces did not count as a word but they count as a position. So we had to take a look and update +1 the position if between words we had a white space.

C. Rule Based

Once we have the tokenized sentence, we are going to classify word by word. We decided to use the following rules:

1) isUpper: If any of the letters of the word is upper, we classify the word as brand
2) sufix: if the word has the following sufix: azole, idne, amine, mycin, we classify it as drug.
3) number: if the word has a number and it is shorter than 5 letters, we classify it as brand.
4) list_drugs: if the word is in the list of drugs provided (HSDB.txt), we classify it as a drug.

This has been implemented by an ”if else” long function where all the words of every sentence are put into. This returns us a list of dictionaries with the following structure: {text: word, offset: start_word_position end_word_position, type: predicted class}. Reference at V Annex.

D. Results

The following tables show the obtained results from executing the Rule Based System both with the Devel set and with the Test set.

TABLE I
OUTPUT LOG TEST IN RULE BASED SYSTEM

Class	Precision	Recall	F1-score
brand	17.9%	48.3%	26.1%
drug	62.7%	8.6%	15.2%
drug_n	0.0%	0.0%	0.0%
group	0.0%	0.0%	0.0%
M. avg	20.1%	14.2%	10.3%
m. avg	1.1%	10.1%	1.9%
m. avg (no classes)	8.2%	78.9%	14.9%

TABLE II
OUTPUT LOG DEVEL IN RULE BASED SYSTEM

Class	Precision	Recall	F1-score
brand	20.5%	54.4%	29.8%
drug	67.2%	10.5%	18.2%
drug_n	0.0%	0.0%	0.0%
group	0.0%	0.0%	0.0%
M. avg	21.9%	16.2%	12.0%
m. avg	1.4%	13.2%	2.5%
m. avg (no classes)	8.5%	80.5%	15.3%

III. PART 2: MACHINE LEARNING ALGORITHM

In this section we are trying to do our classification task using a Machine Learning algorithm. This is changing our paradigm because we will need to extract features from our dataset, choose and train a model using these features, and then make predictions and obtain metrics of the classification task. As we are using a Machine Learning algorithm, one thing we want to take as an advantage from using a Rule Based model is the possibility of obtaining information from the sequence. We will be classifying each word separately but taking a look at the surrounding words of each token. As well as in the rule based model, we have three datasets, the train dataset, which we used to train our machine learning model, the development dataset, which we used to develop the program and to finetune the hyperparameters, and the test dataset, which we used to obtain the classification metrics.

A. Feature Extractor

This part creates the features that will be used by the ML classifiers to train and perform the classification task. We use the directory with all the files to extract the features from as an input and we storage these features in a coma separated values file. This has been decided because we are managing all our data with Pandas library. We are extracting the features related to the next and previous word of the sentence, the last four letters, if the word has capital letters, if has numbers, if has dashes and also the word itself. To extract the features from the xmls files, we tokenize sentence by sentence and then we are extracting all this information and storing to the csv file. In our csv file we have several columns:

1) sentence: sentence id
2) start_word: position of the sentence where the word starts
3) start_word: position of the sentence where the word ends
4) tag: tag of the word
5) form: word
6) suf4: last four letters of the word
7) prev: previous word of the sentence (if the word is the first one a token ”<START>” is added)
8) next: next word of the sentence (if the word is the first one a token <ENDadded)
9) has_dash: True if the word contains a dash, False if not
10) has_number: True if the word contains a number, False if not
11) has_capital: True if the word contains a capital letter, False if not

In further experiments we add two more features to try to improve the performance:

1) punct: True if the word contains a punctuation sign, False if not.
2) suf3: Last three letters of the word

To do that we have deloped two functions:

1) extract_features(): function that extract all features from every word taking their surrounding words into account. Reference at V Annex
2) get_tag(): function that extracts all tags of every entity in order to assign labels at every word. Reference at V Annex

B. Learner

Once we have our features extracted, we need to choose and train a machine learning algorithm. As our Learner algorithm we decided to use a CRF (Conditional Random Field). This algorithm is often applied in pattern recognition and used for structured prediction, taking this into account we decided to use this. Firstly we implement a function called ”prepare_features(csv_file)” to read the coma separated values and prepare the features to be fitted in the model. This method convert the features from the csv file to an array of features per sentence and the tags in the csv a list of tags per sentence. [1] We used the Conditional Random Field from the library ”pycrfsuite” with C1 parameter at 1 and C2 parameter at 0.001.

C. Classifier

Once we have trained the model, we have to obtain the classification metrics for devel test dataset and development dataset. Here we are making the inference with the trained model to the test and development features. We have implemented the inference and we are taking the predictions and the group truth printing a classification report (function given by sklearn.metrics library). We also had implemented a function called output_entities. Reference at V Annex

D. Results

In this section we present the results tables with which we are going to compare the Rule Based Classifier with the ML Algorithm and its variants.

Comparing the Rule Based System (RBS) Results showed into the Tables I and II with the other ones we can notice the better performance of the Machine Learning algorithm system. Also there is not much difference between testing RBS with the Devel set and the Test set.
The F1 Score can not even achieve the 20% of score in any of both cases. So we conclude the high level of difficult to develop a Rule Based System in a Natural Language Processing task.

TABLE III
OUTPUT LOG TEST WITH 4 FEATURES

Class	Precision	Recall	F1-score
brand	93.6%	25.3%	39.9%
drug	86.7%	75.7%	80.8%
drug_n	0.0%	0.0%	0.0%
group	46.0	43.5%	44.7%
M. avg	56.6%	36.6%	41.4%
m. avg	76.4%	62.4%	68.7%
m. avg (no classes)	78.8%	64.4%	70.8%

TABLE IV
OUTPUT LOG DEVEL WITH 4 FEATURES

Class	Precision	Recall	F1-score
brand	96.6%	7.8%	14.4%
drug	86.1%	77.8%	81.7%
drug_n	80.0%	8.9%	16.0%
group	44.0%	39.1%	41.4%
M. avg	76.7%	33.4%	38.4%
m. avg	75.5%	59.6%	66.6%
m. avg (no classes)	78.2%	61.7%	69.0%

Following the natural evolution of the process we have studied the idea of increase the number of features learned by the system. The following tables shows the obtained results.

Finally due to the observed drug_n class results we thought about adding and modifying some features, but also to modify the number of iterations of the learner.

TABLE V
OUTPUT LOG TEST WITH 7 FEATURES

Class	Precision	Recall	F1-score
brand	85.8%	54.5%	66.7%
drug	84.6%	76.9%	80.6%
drug_n	0.0%	0.0%	0.0%
group	45.7	43.8%	44.7%
M. avg	54.0%	43.0%	48.0%
m. avg	75.2%	65.8%	70.2%
m. avg (no classes)	78.1%	68.4%	72.9%

TABLE VI
OUTPUT LOG DEVEL WITH 7 FEATURES

Class	Precision	Recall	F1-score
brand	93.8%	24.7%	39.5%
drug	86.2%	77.6%	81.7%
drug_n	60.0%	13.3%	21.8%
group	43.9	39.1%	41.3%
M. avg	72.0%	38.7%	46.1%
m. avg	76.0%	61.6%	68.0%
m. avg (no classes)	78.8%	63.8%	70.5%

The fact that we have decreased the number of iterations is that we have tried to train the system with Devel and Train both sets. We have done it due to the difficult of increase the F1-score in the drug_n class. So it may overfits.
The following results show how the drug_n class score increases but the global F1 score decreases.

TABLE VII
OUTPUT LOG TEST WITH 9 FEATURES (30 ITERATIONS)

Class	Precision	Recall	F1-score
brand	82.4%	52.1%	63.8%
drug	84.7%	73.6%	78.8%
drug_n	7.1%	1.4%	2.3%
group	46.6%	38.9%	41.1%
M. avg	54.5%	41.5%	46.5%
m. avg	74.5%	62.4%	67.9%
m. avg (no classes)	77.4%	64.9%	70.6%

TABLE VIII
OUTPUT LOG DEVEL WITH 9 FEATURES (30 ITERATIONS)

Class	Precision	Recall	F1-score
brand	91.9%	40.8%	56.5%
drug	85.4%	77.0%	81.0%
drug_n	11.1%	4.4%	6.3%
group	41.6	35.1%	38.1%
M. avg	57.5%	39.3%	45.5%
m. avg	75.1%	62.1%	68.0%
m. avg (no classes)	78.4%	64.8%	70.9%

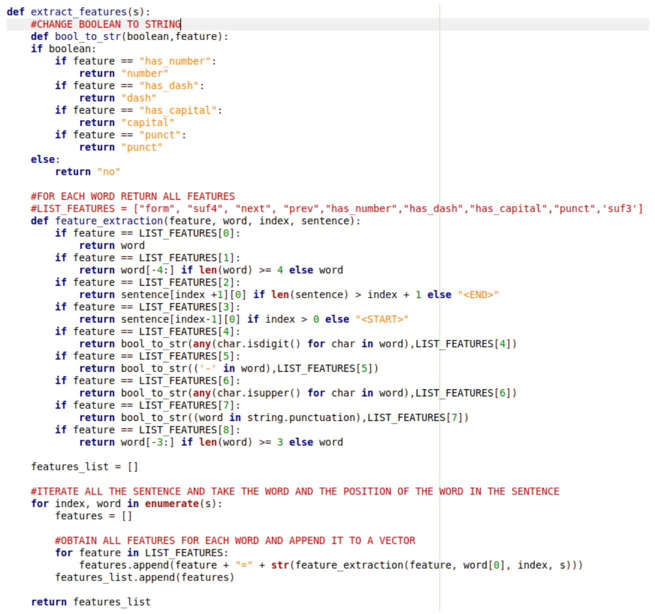
IV. CONCLUSIONS

To sum up we can talk that the Machine Learning algorithm achieves a greater performance than the Rule Bases System. Regarding the Machine Learning algorithm we can conclude that we have a good performance. The global F1-Score is injured due to the drug_n class, i.e the fact that it appears very few times in the test set. The Scores obtained from the other class are good enough to consider that the extracted features are appropriate for the task of Classification and Recognition of drugs names.

We tried to optimize the hyper-parameters using the ones that appear in the documentation [3] but it did not increase considerably the F1-score. We also modified the number of iterations but it seems very limited to reach a much grater performance.

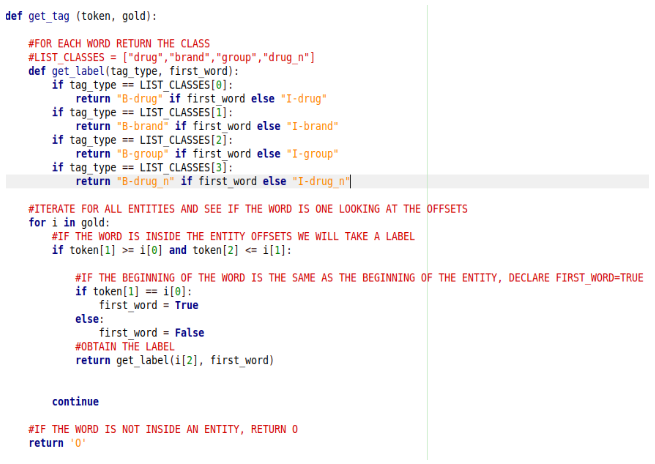
V. Annex

A. Code for extract features



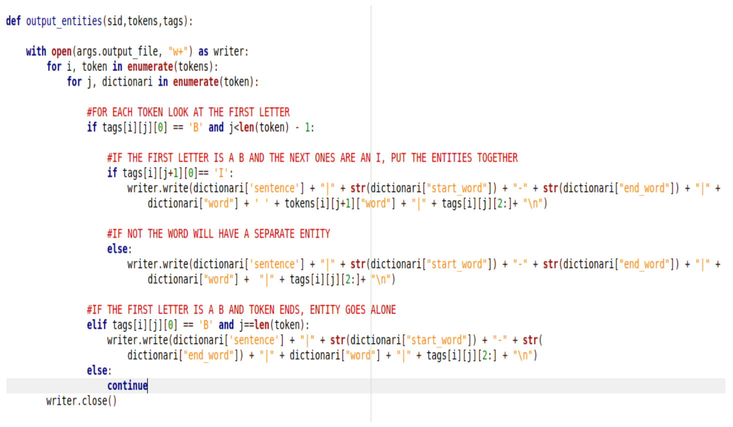
```
def extract_features(s):
    #CHANGE BOOLEAN TO STRING
    def bool_to_str(boolean,feature):
    if boolean:
        if feature == "has_number":
            return "number"
        if feature == "has_dash":
            return "dash"
        if feature == "has_capital":
            return "capital"
        if feature == "punct":
            return "punct"
    else:
        return "no"

    #FOR EACH WORD RETURN ALL FEATURES
    #LIST_FEATURES = ["form", "suf4", "next", "prev","has_number","has_dash","has_capital","punct",'suf3']
    def feature_extraction(feature, word, index, sentence):
        if feature == LIST_FEATURES[0]:
            return word
        if feature == LIST_FEATURES[1]:
            return word[-4:] if len(word) >= 4 else word
        if feature == LIST_FEATURES[2]:
            return sentence[index +1][0] if len(sentence) > index + 1 else "<END>"
        if feature == LIST_FEATURES[3]:
            return sentence[index-1][0] if index > 0 else "<START>"
        if feature == LIST_FEATURES[4]:
            return bool_to_str(any(char.isdigit() for char in word),LIST_FEATURES[4])
        if feature == LIST_FEATURES[5]:
            return bool_to_str(('-' in word),LIST_FEATURES[5])
        if feature == LIST_FEATURES[6]:
            return bool_to_str(any(char.isupper() for char in word),LIST_FEATURES[6])
        if feature == LIST_FEATURES[7]:
            return bool_to_str((word in string.punctuation),LIST_FEATURES[7])
        if feature == LIST_FEATURES[8]:
            return word[-3:] if len(word) >= 3 else word

    features_list = []

    #ITERATE ALL THE SENTENCE AND TAKE THE WORD AND THE POSITION OF THE WORD IN THE SENTENCE
    for index, word in enumerate(s):
        features = []

        #OBTAIN ALL FEATURES FOR EACH WORD AND APPEND IT TO A VECTOR
        for feature in LIST_FEATURES:
            features.append(feature + "=" + str(feature_extraction(feature, word[0], index, s)))
        features_list.append(features)

    return features_list
```

B. Code for get tag



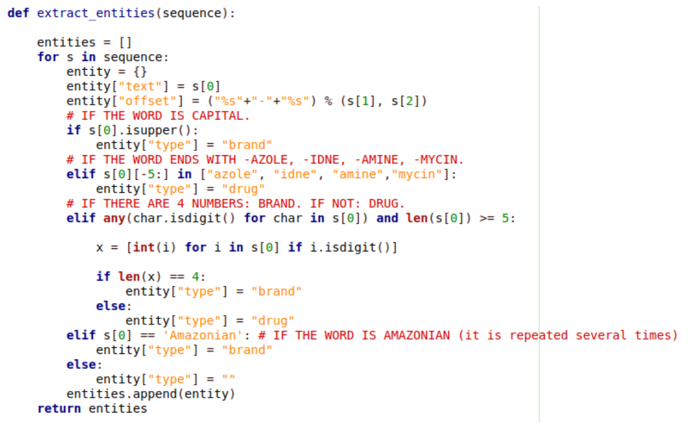
```
def get_tag (token, gold):

    #FOR EACH WORD RETURN THE CLASS
    #LIST_CLASSES = ["drug","brand","group","drug_n"]
    def get_label(tag_type, first_word):
        if tag_type == LIST_CLASSES[0]:
            return "B-drug" if first_word else "I-drug"
        if tag_type == LIST_CLASSES[1]:
            return "B-brand" if first_word else "I-brand"
        if tag_type == LIST_CLASSES[2]:
            return "B-group" if first_word else "I-group"
        if tag_type == LIST_CLASSES[3]:
            return "B-drug_n" if first_word else "I-drug_n"

    #ITERATE FOR ALL ENTITIES AND SEE IF THE WORD IS ONE LOOKING AT THE OFFSETS
    for i in gold:
        #IF THE WORD IS INSIDE THE ENTITY OFFSETS WE WILL TAKE A LABEL
        if token[1] >= i[0] and token[2] <= i[1]:

            #IF THE BEGINNING OF THE WORD IS THE SAME AS THE BEGINNING OF THE ENTITY, DECLARE FIRST_WORD=TRUE
            if token[1] == i[0]:
                first_word = True
            else:
                first_word = False
            #OBTAIN THE LABEL
            return get_label(i[2], first_word)


        continue

    #IF THE WORD IS NOT INSIDE AN ENTITY, RETURN O
    return 'O'
```

C. Code for output entities



```
def output_entities(sid,tokens,tags):

    with open(args.output_file, "w+") as writer:
        for i, token in enumerate(tokens):
            for j, dictionari in enumerate(token):

                #FOR EACH TOKEN LOOK AT THE FIRST LETTER
                if tags[i][j][0] == 'B' and j<len(token) - 1:

                    #IF THE FIRST LETTER IS A B AND THE NEXT ONES ARE AN I, PUT THE ENTITIES TOGETHER
                    if tags[i][j+1][0]== 'I':
                        writer.write(dictionari['sentence'] + "|" + str(dictionari["start_word"]) + "-" + str(dictionari["end_word"]) + "|" +
                            dictionari["word"] + ' ' + tokens[i][j+1]["word"] + "|" + tags[i][j][2:]+ "\n")

                    #IF NOT THE WORD WILL HAVE A SEPARATE ENTITY
                    else:
                        writer.write(dictionari['sentence'] + "|" + str(dictionari["start_word"]) + "-" + str(dictionari["end_word"]) + "|" +
                            dictionari["word"] +  "|" + tags[i][j][2:]+ "\n")

                #IF THE FIRST LETTER IS A B AND TOKEN ENDS, ENTITY GOES ALONE
                elif tags[i][j][0] == 'B' and j==len(token):
                    writer.write(dictionari['sentence'] + "|" + str(dictionari["start_word"]) + "-" + str(
                        dictionari["end_word"]) + "|" + dictionari["word"] + "|" + tags[i][j][2:] + "\n")
                else:
                    continue
        writer.close()
```

D. Code for extract entities



```
def extract_entities(sequence):

    entities = []
    for s in sequence:
        entity = {}
        entity["text"] = s[0]
        entity["offset"] = ("%s"+"-"+"%s") % (s[1], s[2])
        # IF THE WORD IS CAPITAL.
        if s[0].isupper():
            entity["type"] = "brand"
        # IF THE WORD ENDS WITH -AZOLE, -IDNE, -AMINE, -MYCIN.
        elif s[0][-5:] in ["azole", "idne", "amine","mycin"]:
            entity["type"] = "drug"
        # IF THERE ARE 4 NUMBERS: BRAND. IF NOT: DRUG.
        elif any(char.isdigit() for char in s[0]) and len(s[0]) >= 5:

            x = [int(i) for i in s[0] if i.isdigit()]

            if len(x) == 4:
                entity["type"] = "brand"
            else:
                entity["type"] = "drug"
        elif s[0] == 'Amazonian': # IF THE WORD IS AMAZONIAN (it is repeated several times)
            entity["type"] = "brand"
        else:
            entity["type"] = ""
        entities.append(entity)
    return entities
```

References

[1] Sam Galen. *CoNLL 2002.ipynb*. https://github.com/scrapinghub/python-crfsuite/blob/master/examples/CoNLL202002.ipynb. [Online; accessed 19-March-2021]. 2016.

[2] César de Pablo Sánchez Isabel Segura-Bedmar Paloma Martínez. *Using a Shallow Linguistic Kernel for Drug-Drug Interaction Extraction.*

[3] Mikhail Korobov. *Sklearn-crfsuite*. https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html#hyperparameter-optimization. [Online; accessed 21-March-2021]. 2015.