UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

MINING UNSTRUCTURED DATA
Document Structure and Language Detection

MUD Course Project

April 25, 2022

Daniel Arias
Mohana Fathollahi

Contents

1 Introduction

In the first part of assignment, we will focus on detecting name of drugs and classification of them based on four classes that have been defined. We applied different combination of features to train a CRF model and test this model on devel data set, and best combination based on accuracy selected. Moreover we used extra resources to covere more cases and increase accuracy. At the end, these features used for test set to find performance of this model on test set.
In the second part, we should consider interaction between two drugs and classify each pair of drug in four classes that have been defined, there are some cases that there is not interaction between drugs. To achieve this goal, we consider syntactic information in a sentence and clue verbs between two drugs. Therefore different features have been extracted and we consider some of them that improve accuracy on devel data set. Finally, we applied Megam model that has trained with our features to find accuracy for test set.

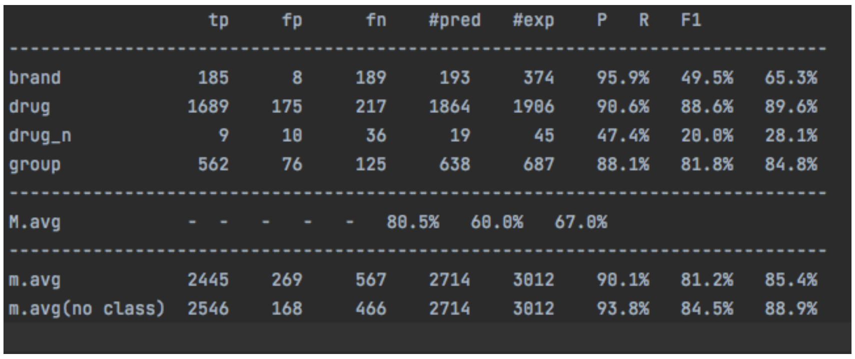
2 Named Entity Recognition and Classification - NERC

2.1 Feature Extraction

To improve accuracy of model we tried different features and different combination of them to find which one of them has more effect on increasing accuracy.
Current features are main word, previous word, next word and last three letter of each of these words, suffixes. In this situation accuracy is 53%.
In below different scenarios for features have been explained and accuracy after applying each of them has been provided. When adding a feature did not improve accuracy, we did not consider it and just keep features that improve accuracy.

- Using the lower case form of a word instead of the exact word increases accuracy to 65%. Additionally, instead of using suffix for all three words, it is just used for the main word and we added suffix 2, considering the last two letters of a word.

```
        tokenFeatures.append("lower="+t.lower())
        tokenFeatures.append("suf3="+t[-3:])
        tokenFeatures.append("suf2="+t[-2:])
        tokenFeatures.append("lowerPrev="+tPrev.lower())
        tokenFeatures.append("lowerNext="+tNext.lower())
```



```
                  tp      fp      fn   #pred    #exp      P      R      F1
------------------------------------------------------------------------------
brand            185       8     189     193     374   95.9%  49.5%  65.3%
drug            1689     175     217    1864    1906   90.6%  88.6%  89.6%
drug_n             9      10      36      19      45   47.4%  20.0%  28.1%
group            562      76     125     638     687   88.1%  81.8%  84.8%
------------------------------------------------------------------------------
M.avg              -   -    -    -    -   80.5%   60.0%   67.0%
------------------------------------------------------------------------------
m.avg           2445     269     567    2714    3012   90.1%  81.2%  85.4%
m.avg(no class) 2546     168     466    2714    3012   93.8%  84.5%  88.9%
```

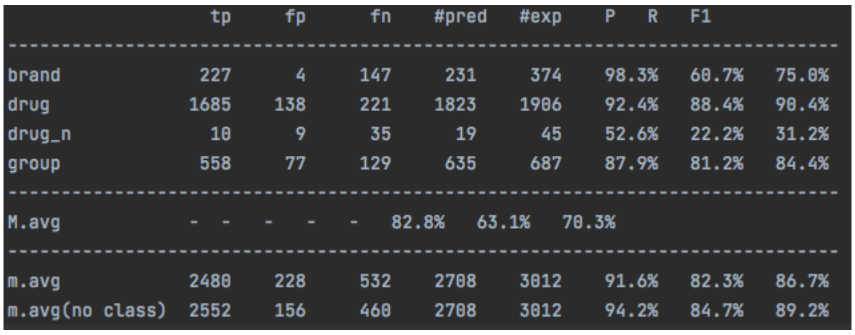
Figure 1: Using lower case and suffixes

- Next, Part of speech (PoS) tag for three words has been considered that increases accuracy to 70%.

```
        tokenFeatures.append("form.pos="+t_pos)
        tokenFeatures.append("formPrev.pos=" + tPrev_pos)
        tokenFeatures.append("formNext.pos=" + tNext_pos)
```

Additionally, we add features for improving the recognition of the specific Part of Speech for **Nouns** and **Proper Nouns** in singular and plural form.

```
        tokenFeatures.append("isNN=" + str(t_pos == 'NN'or t_pos == 'NNS'))
        tokenFeatures.append("isNNP=" + str(t_pos == 'NNP' or t_pos == 'NNPS'))
```



```
                  tp      fp      fn   #pred    #exp      P      R      F1
------------------------------------------------------------------------------
brand            227       4     147     231     374   98.3%  60.7%  75.0%
drug            1685     138     221    1823    1906   92.4%  88.4%  90.4%
drug_n            10       9      35      19      45   52.6%  22.2%  31.2%
group            558      77     129     635     687   87.9%  81.2%  84.4%
------------------------------------------------------------------------------
M.avg              -   -    -    -    -   82.8%   63.1%   70.3%
------------------------------------------------------------------------------
m.avg           2480     228     532    2708    3012   91.6%  82.3%  86.7%
m.avg(no class) 2552     156     460    2708    3012   94.2%  84.7%  89.2%
```

Figure 2: Using POS

- Using these three methods that return Boolean.
  - isupper: This method returns True if all characters in the string are uppercase, otherwise, returns “False”.

type=”brand” text=”ELSPAR”

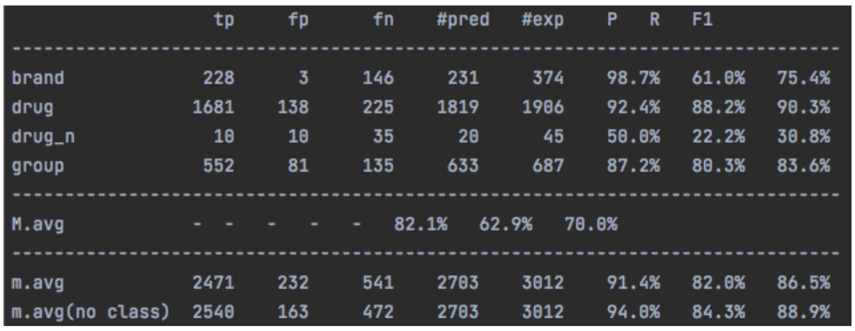
– istitle: This method returns True if all words in a text start with a upper case letter, AND the rest of the word are lower case letters, otherwise False.

type=”drug” text=”Panobinostat”

– isdigit: This method returns True if all the characters are digits, otherwise False.

type=”drug_n” text=”angiotensin 1”

When these functions were applied, accuracy decreased to 0.3%.



```
                   tp     fp      fn    #pred    #exp     P    R    F1
------------------------------------------------------------------------------
brand             228      3     146     231     374    98.7%   61.0%   75.4%
drug             1681    138     225    1819    1906    92.4%   88.2%   90.3%
drug_n             10     10      35      20      45    50.0%   22.2%   30.8%
group             552     81     135     633     687    87.2%   80.3%   83.6%
------------------------------------------------------------------------------
M.avg              -    -    -    -    -    82.1%   62.9%   70.0%
------------------------------------------------------------------------------
m.avg            2471    232     541    2703    3012    91.4%   82.0%   86.5%
m.avg(no class)  2540    163     472    2703    3012    94.0%   84.3%   88.9%
```

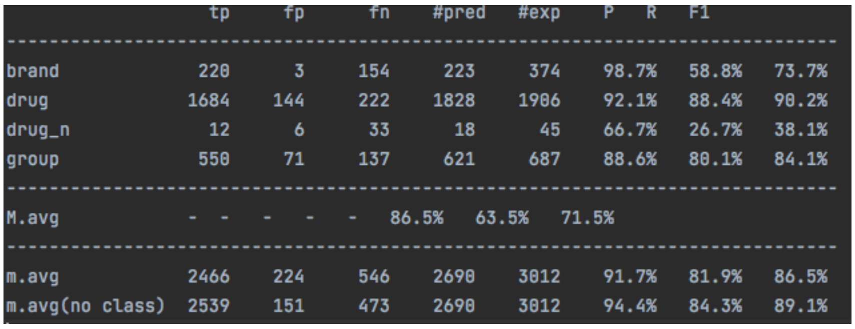
Figure 3: Using title, upper and digit

- The name of drugs do not follow irregular format, therefore stemming will be useful in finding root of the drug name. While, previous and next words can follow irregular format and lemmatization should be used in finding root of these words. The accuracy after using lemmatization and stemming did not change and remain 70.3%.

```
        tokenFeatures.append("stemm=" + SnowballStemmer('english').stem(t))
        tokenFeatures.append("lemmaPrev=" + lemmatizer.lemmatize(tPrev))
        tokenFeatures.append("lemmaNext=" + lemmatizer.lemmatize(tNext))
```

- Contain hyphen could increase accuracy to 71.5%, because in some of the drug names there is a hyphen between two parts, such as:
type=”drug” text=”L-arginine”

```
        tokenFeatures.append("containshyphen=" + str("-" in t))
```



```
                  tp     fp      fn    #pred    #exp     P    R    F1
---------------------------------------------------------------------------
brand            220      3     154     223      374    98.7%   58.8%   73.7%
drug            1684    144     222    1828     1906    92.1%   88.4%   90.2%
drug_n            12      6      33      18       45    66.7%   26.7%   38.1%
group            550     71     137     621      687    88.6%   80.1%   84.1%
---------------------------------------------------------------------------
M.avg             -  -    -    -    -    86.5%   63.5%   71.5%
---------------------------------------------------------------------------
m.avg           2466    224     546    2690     3012    91.7%   81.9%   86.5%
m.avg(no class) 2539    151     473    2690     3012    94.4%   84.3%   89.1%
```

Figure 4: Using hyphen

- In this part, we consider existing numbers and roman numbers in previous and next words, because in some drugs we have these numbers, such as:

  $\boxed{\text{type="drug\_n" text="Buforin II"}}$

  But it could not improve accuracy, maybe there are other words in sentence that has number after them and model get confused to classify them correctly, therefore we can say that this feature is not specific feature for just drugs.

- Using n-gram
  In order to have more clear distinction for the Drugs, we collect the most frequent sequences of characters (n_grams) with n between two and three from the given devel and train file, the result has been shown in table 1. These features take a value of one if the candidate has the corresponding character sequence and zero if it does not, for example in table 2 existing of top grams in **Ketamine** has been analyzed.

  $\boxed{\text{type="drug" text="ketamine"}}$

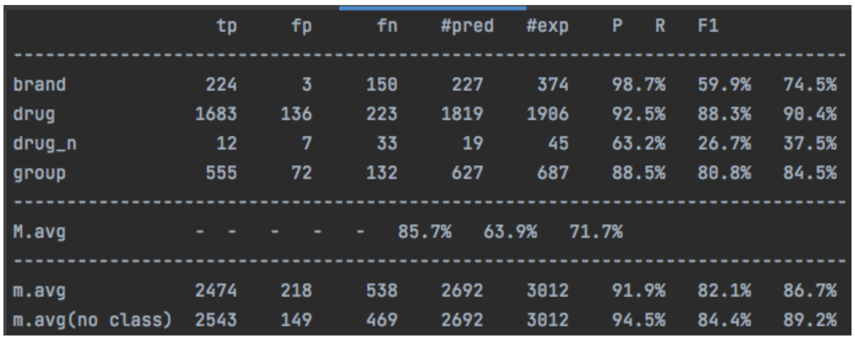
n_gram	occurrence
in	980
ne	663
ine	465
ro	428
id	347
ol	327
et	299
am	296
ri	295
en	289
ide	176
ami	173
ate	156
min	134

Table 1: n_gram Ocurrences

n_gram	occurrence
in	1
ne	1
ine	1
ro	0
id	0
ol	0
et	1
am	1
ri	0
en	0
ide	0
ami	1
ate	0
min	1

Table 2: ketamine

In this case, **ketamine**, summation of occurrence of top n_grams is seven that is larger than our threshold, five, therefore the function that we consider for that, count_n_gram, returns True. The question can be raised is that, why we consider five as a threshold? Because until five, accuracy improves but after this accuracy did not change and based on figure 5 accuracy remained at 71.7%.



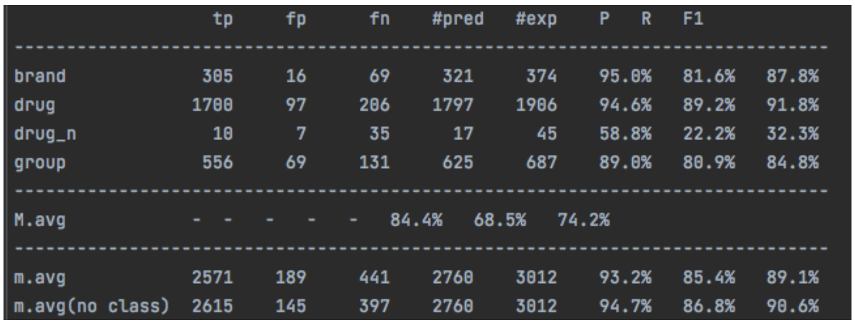
```
                     tp      fp       fn    #pred    #exp     P    R    F1
------------------------------------------------------------------------------
brand               224       3      150      227     374   98.7%   59.9%   74.5%
drug               1683     136      223     1819    1906   92.5%   88.3%   90.4%
drug_n               12       7       33       19      45   63.2%   26.7%   37.5%
group               555      72      132      627     687   88.5%   80.8%   84.5%
------------------------------------------------------------------------------
M.avg                 -   -     -    -     -   85.7%   63.9%   71.7%
------------------------------------------------------------------------------
m.avg              2474     218      538     2692    3012   91.9%   82.1%   86.7%
m.avg(no class)    2543     149      469     2692    3012   94.5%   84.4%   89.2%
```

Figure 5: Using n_gram

2.2 External Resource

Finally, we added features that look up in external resources provided in the lab resources files (DrugBank.txt and HSDB). Drugbank.csv has information for three groups; drug, group and brand. First, this xml file parsed based on these three groups and then we used a function that check if the word is belong to this group or not. The best accuracy that we could get in devel dataset based on table 6 is 74.2%.

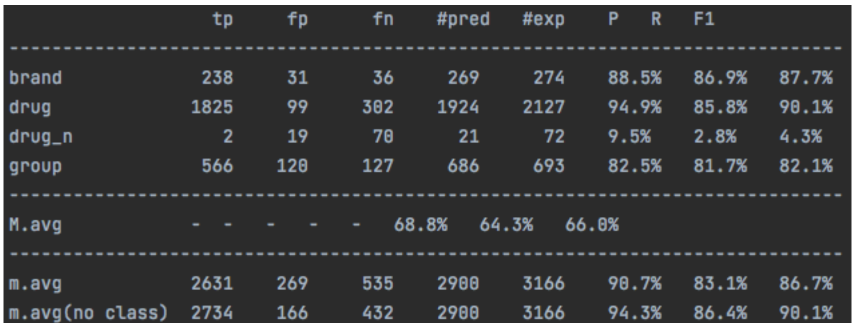
```python
        tokenFeatures.append("inExtSource_Drug=" + str(is_in_external_source_drug(t)))
        tokenFeatures.append("inExtSource_Brand=" + str(is_in_external_source_brand(t)))
        tokenFeatures.append("inExtSource_Group=" + str(is_in_external_source_group(t)))
```



```
                    tp      fp       fn     #pred    #exp     P    R    F1
------------------------------------------------------------------------------
brand              305      16       69      321      374     95.0%   81.6%   87.8%
drug              1700      97      206     1797     1906     94.6%   89.2%   91.8%
drug_n              10       7       35       17       45     58.8%   22.2%   32.3%
group              556      69      131      625      687     89.0%   80.9%   84.8%
------------------------------------------------------------------------------
M.avg               -   -    -    -    -   84.4%   68.5%   74.2%
------------------------------------------------------------------------------
m.avg             2571     189      441     2760     3012     93.2%   85.4%   89.1%
m.avg(no class)   2615     145      397     2760     3012     94.7%   86.8%   90.6%
```

Figure 6: Applying extra resources

Based on features that we extracted, our model is trained and in the figure 7 result of applying this model on test set has been shown, based on this we could reach to accuracy equal to 66%, in the drug_n we did not have extra resources and because of that we could not reach to high accuracy for that clss.



```
                    tp      fp       fn     #pred    #exp     P    R    F1
------------------------------------------------------------------------------
brand              238      31       36      269      274     88.5%   86.9%   87.7%
drug              1825      99      302     1924     2127     94.9%   85.8%   90.1%
drug_n               2      19       70       21       72     9.5%    2.8%    4.3%
group              566     120      127      686      693     82.5%   81.7%   82.1%
------------------------------------------------------------------------------
M.avg               -   -    -    -    -   68.8%   64.3%   66.0%
------------------------------------------------------------------------------
m.avg             2631     269      535     2900     3166     90.7%   83.1%   86.7%
m.avg(no class)   2734     166      432     2900     3166     94.3%   86.4%   90.1%
```

Figure 7: Result of test dataset

2.3 CRF Model

To consider impact of different training algorithms in CRF on accuracy of model: two training algorithm have been compared together; l2sgd and lbfgs. l2sgd has l2 regularization and lbfgs has l1/l2 regularization.
Both of these algorithms try to maximize the logarithm of the likelihood of the training data. Except regularization these algorithms has another difference; l2sgd approach to the optimal feature weights rapidly but has slow convergence at the end. While, lbfgs has different behaviour and first improve slowly and at the end converge to optimal very fast. In our model, we could get better result when we used lbfgs with 200 iteration.

2.4 Conclusion

For this task, we need to extract multiple features per word that will help recognize a Named Entity. We developed feature functions that will assist in generating unique features whose output has to be True or False. For these features we tried to add as much context around each token in order to assist better in the classification, that is why we performed some of the features not only in the main token but in the previous and in the next token from the phrase as well.

As we could observe in the results, adding features that only look for in the form or in the content of the token, doesn't improve significantly the accuracy due to the classifier tends to label the entities as **drug**, these could be because there was not much clear difference between the entities form of drug and drug_n. Of all the features, the Part of Speech (**PoS**) was the main feature that gave us more improvement in the accuracy in the F1 and helped more to recognize better between the brands, groups and drugs.

Additionally, having features that look up in external resources improves in the differentiation between drugs and brands as the *DrugBank.txt* has labels for brand, drugs and groups).

To conclude, as it was an iterative process of running the model and having feedback from the results we obtained, we could decide to enhance the features that give more improvement or remove the ones that may mislead the classifier.

2.5 Future Work

After narrowing down the best features that gave us the best accuracy, we know that some of them could be enhanced but for the scope of the lab and in order to be compliant with the deadline of the project we could not dig deeper on them. However for future work we acknowledge the following ones:

- Better parsing for HSDB.txt:

  As we used this the external resources, in here the classification is **brand** but we would need to do some prepossessing and a better function to look for into the sentence in order to get the name of the brand out of it.

  ```
  0.15% Potassium Chloride In 3.3% Dextrose and 0.30% Sodium Chloride Injection|brand
  ```

- Better parsing for DrugBank full_database.xml Same as before we would need to have a better parsing for external resources full database DrugBank due to the parserXML.go developed only takes the drug names from the file, we only consider getting the names of the drugs due to the file was very large and will took as much time to extract all the properties such as the synonyms for the name of the drug, the product names and the groups from it.

```go
type Drugs struct {
        XMLName xml.Name `xml:"drugbank"`
        Drugs   []Drug   `xml:"drug"`
}
type Drug struct {
        XMLName xml.Name `xml:"drug"`
        Name    string   `xml:"name"`
}
```

- Adding the relation between tokens: As the PoS tag was one of the main features that gave us much improvement, we thought that it would be an interesting feature to analyze the dependency tree and add the relation between the tokens as a feature.

3 Drug-Drug Interaction - DDI

In this part, we should classify interaction between the drugs according to the sentence. Therefore, we should consider relation between the different words in the sentence and features for each word.
One way to find relation between words in a sentence is using dependency tree, that has been shown in the figure 8. In dependency tree we should find a path between words that we want. The question can be raised is that which path should be considered. Based on some studies, the shortest path between two nodes, in our case between pair of drug, is likely to carry the most valuable information about their relationship [1].
The features that have been considered for each word is based on the code in below:

```python
word = tree.get_word(tk)
lemma = tree.get_lemma(tk).lower()
tag = tree.get_tag(tk)
feats.add("lib=" + lemma)
feats.add("wib=" + word)
feats.add("lpib=" + lemma + "_" + tag)

if tree.is_entity(tk, entities):
    feats.add("eib")
```

[1]Extracting drug–drug interactions from literature using a rich feature-based linear kernel approach by Sun Kim, Haibin Liu, Lana Yeganova, W. John Wilbur

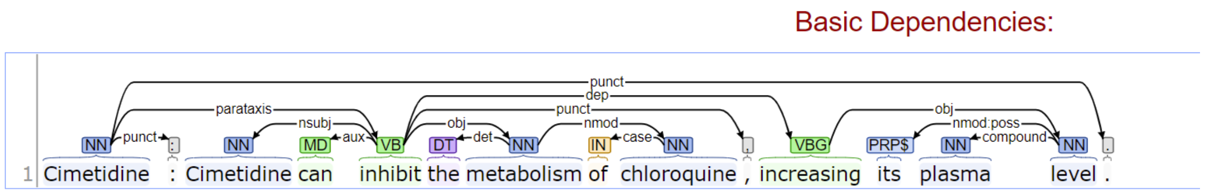


Figure 8: Dependency Tree

Lowest Common Subsumer of two nodes considered as LCS and path from tkE1 to LCS and tkE2 to LCS considered as two part of shortest path.

First feature that considered is dependency path that encoded with respect to below example:REF
cause considered as LCS that connect METH and sch.
$dep\_path\_1 = nsubj(cause, METH)$
$dep\_path\_2 = obl(cause, sch)$

```
        dep_path_1 = tree.get_rel(tkE1) + '(' + tree.get_lemma(lcs) + ',' +
        ↪  tree.get_lemma(tkE1) + ')'
        dep_path_2 = tree.get_rel(tkE2) + '(' + tree.get_lemma(lcs) + ',' +
        ↪  tree.get_lemma(tkE2) + ')'
```

Another feature that may give us valuable information is considering part of speech tagging for words that are exist in shortest path. We used code in below to extract these features.

```
        rel_path1 = "<".join([tree.get_rel(x) for x in up_path])
        rel_path2 = ">".join([tree.get_rel(x) for x in down_path])
        feats.add("rel_path1=" + rel_path1)
        feats.add("rel_path2=" + rel_path2)
        rel_path = rel_path1 + "_" + tree.get_rel(lcs) + "_" + rel_path2
        feats.add("rel_path=" + rel_path)
```

Moreover, relation between words in dependency tree can give us high chance to classify interaction between drugs with better accuracy.

```
    rel_path1 = "<".join([tree.get_rel(x) for x in up_path])
        rel_path2 = ">".join([tree.get_rel(x) for x in down_path])
        feats.add("rel_path1=" + rel_path1)
        feats.add("rel_path2=" + rel_path2)
        rel_path = rel_path1 + "_" + tree.get_rel(lcs) + "_" + rel_path2
        feats.add("rel_path=" + rel_path)
```

In the below code combination of relation of words in dependency tree and part of speech has been considered.

```
        pos_rel_path1 = "<".join([tree.get_tag(x) + "_" + tree.get_rel(x) for x in up_path])
        pos_rel_path2 = ">".join([tree.get_tag(x) + "_" + tree.get_rel(x) for x in down_path])
        feats.add("pos_rel_path1=" + pos_rel_path1)
        feats.add("pos_rel_path2=" + pos_rel_path2)
        pos_rel_path = pos_rel_path1 + "_" + tree.get_tag(lcs) + "_" + tree.get_rel(lcs) + "_"
        ↪  + pos_rel_path2
        feats.add("pos_rel_path=" + pos_rel_path)
```

We considered two approach for extracting clue verb:

- First approach: considering list of synonyms related to each class, effective, advice, interaction and mechanism. As we are not expert in pharmacy and medical terminology these synonyms may not be complete or cover all situation.
- Second approach; extracting clue verbs based on training data set. In this approach, we considered verbs that exist between two drugs and extract a list of verbs that have higher frequency in each class, but in interaction class we could not find a lot of verbs. maybe because of having imbalanced data set. In table below we can see amount of members in each class. In the interaction class we have very few cases compare to other classes that has two consequences, first; list of clue verb for interaction will be so small and we may not consider most verbs that relate to interaction, next there is a risk of overfiting. The model that is training with small number of data cannot learn important features related to the class and instead of learning, it will memorize features that can cause bad performance in test dataset.

advice	effect	mechanism	interaction
697	1444	1028	231

3.1 First approach

For this part we wanted to add the presence of clue verbs and the effect in the relation in the pair of the entities. In order to achieve this we select the verb and its different tag types. Given that we wanted to identify if the verb has a relation of **Cause** or **Effect** in order to improve the accuracy in those classifiers.

```
    if tag in ["VB","VBP","VBZ","VBD","VBN","VBG"]:
        if is_cause(lemma):
            feats.add("isCauseBy=" + str(True))
            feats.add("csib=" + "DRUG1" + "_" + lemma + "_effect" + "_" + "DRUG2")
```

```python
        if is_effect(lemma):
            feats.add("isEffectOf=" + str(True))
            feats.add("effib=" + "DRUG1" + "_" + lemma + "_effect" + "_" + "DRUG2")
```

Since the mechanism relation between entities could be different types, we focus on recognize the main ones for us like **increase**, **decrease**, **inhibit** and **absorb**.

”Digoxin: Digoxin concentrations are **increased** by about 15% when digoxin and carvedilol are administered concomitantly.”

”Aminosalicylic acid may also **decrease** the absorption of vitamin B12, which can lead to a deficiency.”

”Aminosalicylic acid may decrease
the amount of digoxin (Lanoxin, Lanoxicaps) that gets **absorbed** into your body.”

”Cimetidine: Cimetidine, a known **inhibitor** of
renal tubular secretion of organic bases via the cationic transport system,
caused a 50% increase in pramipexole AUC and a 40% increase in half-life (N= 12).”

For this reason, we got the synonyms for these main verbs and added them in the *is_mechanims* function.

```python
        if is_mechanism(lemma):
            feats.add("isMechanism=" + str(True))
            feats.add("mechib=" + "DRUG1" + "_" + lemma + "_mechanism" + "_" + "DRUG2")
```

In the same way, we recognize the interaction between the entities by finding the interact verb alongside their synonyms in the *is_interaction* function.

”Clinical studies with celecoxib have identified
potentially significant **interactions** with fluconazole and lithium.”

```python
        if is_interaction(lemma):
            feats.add("isInteraction=" + str(True))
            feats.add("intib=" + "DRUG1" + "_" + lemma + "_interact" + "_" + "DRUG2")
```

Besides we find the Modal (**MD**) tag in the sequence between the entities adding the word and the *isAdvice* to the features.

”If the two drugs are coadministered,
the beta blocker **should** be withdrawn several days before the gradual withdrawal of clonidine.”

```
    if tag == 'MD' and word.lower() in ["without", "must", "should", "have", "would"]:
        feats.add("isAdvise=" + str(True))
        feats.add("aib=" + word)
        feats.add("ar=" + "DRUG1" + "_" + lemma + "_advice" + "_" + "DRUG2")
```

We not only wanted to add label features we add extra information to the classifier encoding syntactic information, for this reason, we added the relationship between the drugs, the lemma of the verb and the classification label. As the goal of the model is to detect the interaction between the drugs, the name of the entities is not relevant in these scenario. Therefore, we anonymize the name of them replacing it with the words *DRUG1* and *DRUG2*.

3.2 Second approach

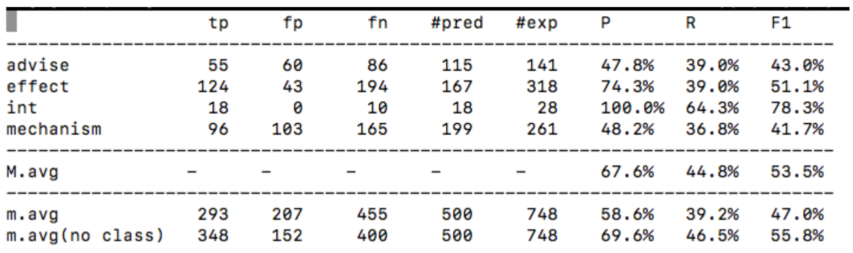
In this approach, we used different synonyms compere to first approach. Next, add True to features that we define for each class if verb belong to list of synonyms for specific class.

```
    advice_synonymes =["avoid","replace","recommend","require","consider",
    ↪   "coadminister","monitor","metabolize","consider","treat","retard","discontinue",
        ...]
    effect_synonymes =
    ↪   ["prolong","potentiate","combine","block","depress","enhance","attenuate","prevent",..]
    mechanism_synonymes
    ↪   =["contain","expect","lead","inhibit","displace","impair","accelerate","delay",..]
    interaction_synonymes= ["interact","classify",'suggest',"relate","identify",..]

    if is_advice(lemma):
    feats.add("isAdviceOf=" + str(True))
    if is_effect(lemma):
        feats.add("isEffectOf=" + str(True))
    if is_interaction(lemma):
        feats.add("isInteraction=" + str(True))
    if is_mechanism(lemma):
        feats.add("isMechanism=" + str(True))
```

3.2.1 Result

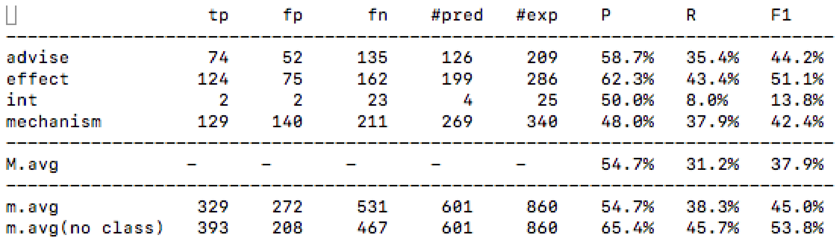
After creating a model, we could get 53.5% accuracy for devel dataset that has been shown in figure 9.



```
                         tp      fp      fn   #pred    #exp    P        R        F1
------------------------------------------------------------------------------------
advise                   55      60      86     115     141    47.8%    39.0%    43.0%
effect                  124      43     194     167     318    74.3%    39.0%    51.1%
int                      18       0      10      18      28    100.0%   64.3%    78.3%
mechanism                96     103     165     199     261    48.2%    36.8%    41.7%
------------------------------------------------------------------------------------
M.avg                  -       -       -       -       -       67.6%    44.8%    53.5%
------------------------------------------------------------------------------------
m.avg                   293     207     455     500     748    58.6%    39.2%    47.0%
m.avg(no class)         348     152     400     500     748    69.6%    46.5%    55.8%
```

Figure 9: Result of second approach on devel dataset

In the Figure 9 accuracy for test set has been provided, and as we can see test accuracy is much lower than devel accuracy in interaction class because of reasons that explained before.



```
                         tp      fp      fn   #pred    #exp    P        R        F1
------------------------------------------------------------------------------------
advise                   74      52     135     126     209    58.7%    35.4%    44.2%
effect                  124      75     162     199     286    62.3%    43.4%    51.1%
int                       2       2      23       4      25    50.0%    8.0%     13.8%
mechanism               129     140     211     269     340    48.0%    37.9%    42.4%
------------------------------------------------------------------------------------
M.avg                  -       -       -       -       -       54.7%    31.2%    37.9%
------------------------------------------------------------------------------------
m.avg                   329     272     531     601     860    54.7%    38.3%    45.0%
m.avg(no class)         393     208     467     601     860    65.4%    45.7%    53.8%
```

Figure 10: Result of second approach on test dataset

3.3 Megam Model

Language models offer a way to assign a probability to a sentence or other sequence of words, and to predict a word from preceding words. n-grams are models estimate words from a fixed window of previous words. n-gram probabilities can be estimated by counting in a corpus and normalizing (the maximum likelihood estimate).

3.4 Conclusion

In this part, most important features that improve accuracy were related to structure of sentence, relation between words and POS of each word in sentence. While, other features like lemmatization for tokens that exist between two drugs and combination of this feature with other features could not help us to improve accuracy.

3.5 Future Work

After establishing the best features for the Drug-Drug Interaction, we know that some of them could be improved. Within the case of checking the main verb, we only added one-word synonyms to the list. Alternatively, we could add compose synonyms or phrasal verbs that have the same meaning. Also, we narrow the verb analysis between the entities, however, the clue verb could also be before the first entity or at the end of the second entity, checking all of these cases could give more information to the classifier for an improved prediction. Additionally, applying some techniques to deal with imbalanced data set can be a possible way to improve accury.