Name Entity Recognition and Classification (NERC)

Master in Data Science, FIB
Mining Unstructured Data

April 18th, 2023

Pau Comas Herrera - Odysseas Kyparissis

Table of Contents

Introduction

The goal of this report is to present a detailed approach to marking pharmacological substances or expressions, in sentences that have been extracted from biomedical texts. Firstly, an explanation about selected Machine Learning (ML) algorithms takes place, followed by a deeper dive into the extract-features process of the solution, and ends by presenting different experimenting results and the most important conclusions.

In more detail, inside the mentioned sentences that exist in the corpora, there are four types of entities that need to be recognized and subsequently classified, namely ***drug***, ***brand***, ***group***, and ***drug_n***. The drug type is employed to indicate human medicines known by a generic name, whereas medicines described by a trade or brand name are labeled as brand entities. Also, since it is quite frequent to find descriptions of drug-drug interactions that involve groups of drugs in texts, the group type was used to label such groups. The last entity type, drug_n, denotes active substances that have not been authorized for human use, including but not limited to toxins or pesticides.

Machine Learning NERC

Selected Algorithms

In order to recognize and classify the pharmacological entities of the available data that were extracted from biomedical texts, we experimented with two different classifiers: Naive Bayes[1] (NB) and Conditional Random Fields[2] (CRF).

The NB classifier is a probabilistic classifier that works on the principle of Bayes' theorem. It assumes that all features are independent of each other and uses this assumption to calculate the probability of a given feature belonging to a particular class. The Multinomial variant of NB, which was used as one of the two classifiers, is specifically designed for working with discrete features, such as word counts or frequency of occurrence. We chose this classifier because it is simple to implement and has a low computational cost. In that way, it was possible to compare the performance of more complex models at the training/execution-time level compared to NB and derive conclusions about the tradeoff between predictability power and training /execution-time.

On the other hand, CRF is a type of probabilistic model that is used for sequence labeling tasks such as named-entity recognition (NER), part-of-speech (PoS) tagging, and speech recognition. It is a discriminative model that models the conditional probability distribution over the output labels given the input sequence. One of the key advantages of CRF is that it can capture the

[1] https://scikit-learn.org/stable/modules/naive_bayes.html
[2] https://sklearn-crfsuite.readthedocs.io/en/latest/

dependencies between adjacent output labels, which makes it particularly useful for tasks where the order of the input sequence matters. Additionally, CRF models are trained using maximum likelihood estimation or maximum a posteriori estimation, and the model parameters are learned using gradient-based optimization techniques such as stochastic gradient descent or conjugate gradient descent. One of the main benefits of CRF is that it can handle more complex features than NB because it can take into account contextual information and dependencies between adjacent words/tokens, which makes them more accurate than other methods such as rule-based systems or simple classifiers like NB. Finally, it has been shown that CRF models outperform other methods for sequential data such as text, especially for named entity recognition tasks.

To conclude, in order to compare the performance of the two classifiers, we conducted several experiments on a computational and classification-power level that are presented in the Experiments and Results chapter. The results showed that CRF outperformed Naive Bayes in terms of the overall F-score. This was expected, given that CRF is a more sophisticated model that takes into account the sequential nature of text.

Feature Extraction

In natural language processing (NLP) tasks such as NER, feature extraction plays a crucial role in achieving high accuracy and performance. In this chapter, we will discuss the different features that were tried, discarded, and used in the development of a feature extraction function.

To begin with, instead of sequentially adding features and checking how they affect the performance of the classifier models in the entities classification task, a different approach was followed. More specifically, we calculated a big set of potential features that could improve the classification results, and then by analyzing the weights assigned by the model to each feature (for each distinct label), we discarded the ones with low predictability power.

Before continuing to present the whole set of the potential features that were generated, it is necessary to refer to two external sources which are available for extracting extra information which can lead to better classification results. The two external sources are respectively named Hazardous Substances Data Bank[3] (HSDB) and DrugBank[4] database. HSDB is a factual, non-bibliographic data bank focusing upon the toxicology of potentially hazardous chemicals and in the available version that was used during the development of the current solution, 6,054 hazardous items were included. Additionally, the DrugBank database is a comprehensive, freely accessible, online database containing information on drugs and drug targets. The content of this source includes 113,827 items in total, which represent 23,952 drugs, 85,828 brands and 4,047 groups. Consequently, by diving into those sources, we could generate features that provide better classification of the target variables.

The following is a description of the whole set of features initially generated:

[3] https://pubchem.ncbi.nlm.nih.gov/source/11933
[4] https://go.drugbank.com/

- Form of the token (***form=***)
- Lowercased form of the token (***lower=***)
- Length of the token (***length=***)
- A set of flags indicating whether the token matches certain words from four different lists (***DB_drug_list***, ***DB_brand_list***, ***DB_group_list***, and ***hsdb_list***)
- Whether the token is at the beginning or end of a sentence (**BoS**, **EoS**)
- Suffixes of length 3, 4, 5, and 6 of the token (e.g. ***suf3=***, ***suf4=***, etc.)
- A set of flags indicating whether the token has special characters (***POSC***), such as punctuation or digits, or occurrences of one capital letter (***OC***) or more (***MTOC***).
- Prefixes of length 3, 4, 5, and 6 of the token (e.g. ***pre3=***, ***pre4=***, etc.)
- POS tag of the token, as determined by the NLTK[5] library (***POS=***).
- Lemma of the token, as determined by the WordNet lemmatizer[6] (***lemma=***).
- The same features as above for the previous one or two tokens, depending on the position of the current token in the sentence (e.g. ***form_prev_1=***, ***form_prev_2***, ***form_next_1***, ***form_next_2***, *etc.*). In this case, checks were performed in order to validate if a token is at the start or at the end of a sentence in order to avoid exceeding the sentence length or trying to select tokens with indexes like -1. All different cases are taken into account.

One important thing that needs to be clarified here is that the result of the ***extract-features.py*** script that is generating the features used for training the classifiers, is making use of the B-I-O schema tagging. B-I-O schema tagging, also known as the IOB (Inside-Outside-Beginning) tagging scheme, is a commonly used method for NER. In the B-I-O schema, each word in a text is tagged with a label that indicates whether it is part of an entity or not, and if so, which type of entity it belongs to. The labels are assigned based on the position of the word within an entity. There are three types of labels used in the B-I-O schema: B (beginning), I (inside), and O (outside). The B label is used to tag the first word in an entity, while the I label is used for all subsequent words that are part of the same entity. The O label is used for all words that are not part of an entity. Consequently, the B-I-O tags are being used by the classification algorithms to detect if a word in the sentence is an entity, and then to classify those entities into one of the following target labels: ***B-drug, B-brand, B-group, B-drug_n, I-drug, I-brand, I-group, I-drug_n*** and ***O***.

To continue with, in our approach, once the above-mentioned features were extracted, we calculated the maximum absolute values of scores for the state features of the CRF model for each distinct target label (***B-drug, B-brand, B-group, B-drug_n, I-drug, I-brand, I-group, I-drug_n*** and ***O***), which were then sorted and saved into separate CSV files. The score of the feature indicates the weight or importance assigned to a specific feature from the model during the training process. We selected to apply this technique on the CRF model instead of the NB, due to the fact that we already knew from the initial try of the default ***extract_features*** function, that the CRF model performs better in this task. Then we created visualizations of the max scores for each distinct type of feature and we selected to keep only the top 10 for each label.

[5] https://www.nltk.org/
[6] https://www.nltk.org/_modules/nltk/stem/wordnet.html

For providing you a sample of the visualizations, we include below one of the produced graphs (specifically for label ***B-drug***). The rest of the plots can be found here.

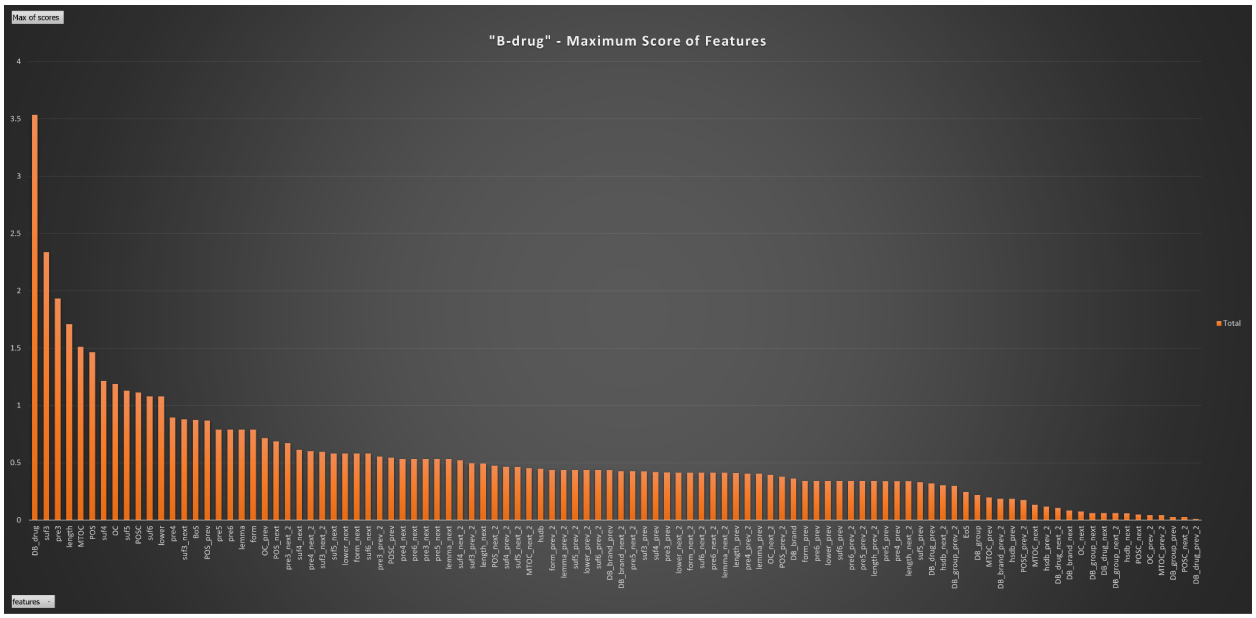


Figure 1. Maximum Score of Features for Label ***B-drug***

In the figure presented above, for example, it can be easily seen which are the 10 most important features for identifying label ***B-drug***. The following table includes the same information for all the labels, in a sorted by the max-value manner.

Table 1. Top 10 Features per Target Label

B-drug	B-brand	B-drug_n	B-group	I-brand	I-drug	I-drug_n	I-group	O
DB_ drug	*MTOC*	*POSC*	*POS*	*MTOC_ prev*	*suf3*	*MTOC_ next_2*	*POS*	*BoS*
suf3	*DB_ brand*	*suf3*	*length*	*length*	*OC_ prev*	*POSC_ prev*	*POS_ prev*	*length*
pre3	*BD_drug*	*suf5*	*BoS*	*POS_ prev*	*POSC_ prev*	*OC*	*MTOC*	*MTOC*
length	*OC*	*suf6*	*suf3*	*MTOC*	*DB_ drug_ prev*	*pre3*	*POSC*	*EoS*

MTOC	*length*	*pre4*	*POS_ prev*	*length_ prev*	*lower*	*length*	*pre3*	*suf4*
POS	*POSC*	*pre3*	*pre5*	*POS_ next*	*MTOC*	*suf3*	*POSC_ prev*	*POSC*
suf4	*BoS*	*lower*	*pre4*	*suf3_ prev*	*length*	*length_ prev*	*suf3*	*suf3*
OC	*POS*	*suf4*	*pre3*	*length_ next_2*	*suf6*	*pre3*	*pre4*	*DB_ brand*
suf5	*suf5*	*DB_drug*	*DB_group*	*POS_ next_2*	*suf5*	*POSC_ prev*	*EoS*	*POS*
POSC	*suf6*	*length*	*pre6*	*POS*	*suf4_ prev*	*suf3_ prev*	*lemma*	*pre3*

As it can be observed, In the lists of the top 10 features for each label there were several overlaps, as it was expected. By following this approach and by keeping in the solution only the most important features for each label, we concluded in the final code for the ***extract_features*** function which is included in the following section. The goal of this approach was to use a small number of features compared to the full feature-space generated, making in such a way the calculation of the feature vectors as well as the training of the models faster and more efficient without losing predictability power.

Code

In this section, the final version of the ***extract_features*** function, as well as two additional methods that were used in the extracting features process are presented.

Python

```
nltk.download('averaged_perceptron_tagger', quiet=True)
nltk.download('wordnet', quiet=True)
def extract_features_final(tokens, DB_drug_list, DB_brand_list, DB_group_list, hsdb_list):
    result = []
    # Initialize the WordNet lemmatizer
```

```
    lemmatizer = WordNetLemmatizer()

    for k in range(len(tokens)):
        tokenFeatures = []
        t = tokens[k][0]

        tokenFeatures.append("form=" + t)
        tokenFeatures.append("lower=" + t.lower())
        tokenFeatures.append("length=" + str(len(t)))


        if t in DB_drug_list:
            tokenFeatures.append("DB_drug")
        if t in DB_brand_list:
            tokenFeatures.append("DB_brand")
        if t in DB_group_list:
            tokenFeatures.append("DB_group")
        if t in hsdb_list:
            tokenFeatures.append("hsdb")

        if k == 0:
            tokenFeatures.append("BoS")
        elif k == len(tokens) - 1:
            tokenFeatures.append("EoS")

        tokenFeatures.append("suf3=" + t[-3:])
        tokenFeatures.append("suf4=" + t[-4:])
        tokenFeatures.append("suf5=" + t[-5:])
        tokenFeatures.append("suf6=" + t[-6:])

        # add the string "OC" if there is one capital letter in the word
        # or add the string "MTOC" if there are more than one capital letters in the word
        if sum(1 for c in t if c.isupper()) == 1:
            tokenFeatures.append("OC")
        elif sum(1 for c in t if c.isupper()) > 1:
            tokenFeatures.append("MTOC")

        # 5. add the string "POSC" if there is presence of special characters in the word,
        # like numbers, dashes, pantuaction marks, etc.
        if any(c in string.punctuation or c.isdigit() for c in t):
            tokenFeatures.append("POSC")

        tokenFeatures.append("pre3=" + t[:3])
        tokenFeatures.append("pre4=" + t[:4])
        tokenFeatures.append("pre5=" + t[:5])
        tokenFeatures.append("pre6=" + t[:6])

        # part of speach (POS) tag of the word
        pos_tag = nltk.pos_tag([t])[0][1]
        tokenFeatures.append("POS=" + pos_tag)

        pos = pos_tag[0].lower() if pos_tag and pos_tag[0].lower() in {'n', 'v', 'a', 'r', 's'} else
'n'
        lemma = lemmatizer.lemmatize(t, pos)
        tokenFeatures.append("lemma=" + lemma)

        if k > 0:
```

```
            t_prev = tokens[k - 1][0]

            tokenFeatures.append("length_prev=" + str(len(t_prev)))

            if t_prev in DB_drug_list:
                tokenFeatures.append(f"DB_drug_prev")

            tokenFeatures.append("suf4_prev=" + t_prev[-4:])

            if sum(1 for c in t_prev if c.isupper()) == 1:
                tokenFeatures.append("OC_prev")
            elif sum(1 for c in t_prev if c.isupper()) > 1:
                tokenFeatures.append("MTOC_prev")

            if any(c in string.punctuation or c.isdigit() for c in t_prev):
                tokenFeatures.append("POSC_prev")

            pos_tag = nltk.pos_tag([t_prev])[0][1]
            tokenFeatures.append("POS_prev=" + pos_tag)

        if k < len(tokens) - 2:  # We are before the second last word of the sentence,
            # take the same features of the next two words
            t_next = tokens[k + 2][0]

            if sum(1 for c in t_next if c.isupper()) == 1:
                tokenFeatures.append(f"OC_next_2")
            elif sum(1 for c in t_next if c.isupper()) > 1:
                tokenFeatures.append(f"MTOC_next_2")

        result.append(tokenFeatures)
    return result
```

As it can be observed, the ***extract_features_final*** function has 4 additional parameters from the default one, being ***DB_drug_list, DB_brand_list, DB_group_list,*** and ***hsdb_list*** respectively. Those parameters are referring to the lists generated by using the external sources (DrugBank and HSDB) for extracting important information that lead to better classification results. The functions that create those lists are presented in the following block:

Python

```
def ext_drug_bank(drug_bank_dir):

    with open(drug_bank_dir, 'r', encoding="utf8") as f:
        data = f.read()

        entries = data.split('\n')
        DB_drug_list = []
        DB_brand_list = []
        DB_group_list = []

        # Loop through each entry
        for n, entry in enumerate(entries):
            if n + 1 == len(entries):
                continue
```

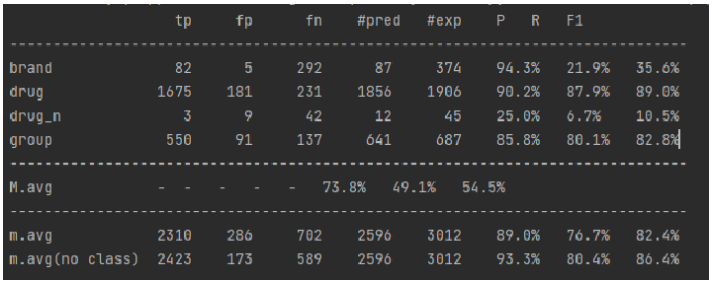
```
            item_type = entry.split('|')[1]
            item = entry.split('|')[0]
            if item_type == 'drug':
                DB_drug_list.append(item)
            elif item_type == 'brand':
                DB_brand_list.append(item)
            elif item_type == 'group':
                DB_group_list.append(item)

    return DB_drug_list, DB_brand_list, DB_group_list

def ext_HSDB(hsdb_dir):

    with open(hsdb_dir, 'r', encoding="utf8") as f:
        data = f.read()
        hsdb_list = data.split('\n')

    return hsdb_list
```

Consequently, in the main part of the ***extract-features.py*** file, first there is a call to the functions creating the lists, followed by the call of extracting features function presented above. Finally, here the respective version of the ***extract_features_final*** function implemented for NB can be found (we did not include it in the report for space-saving reasons).

Experiments and Results

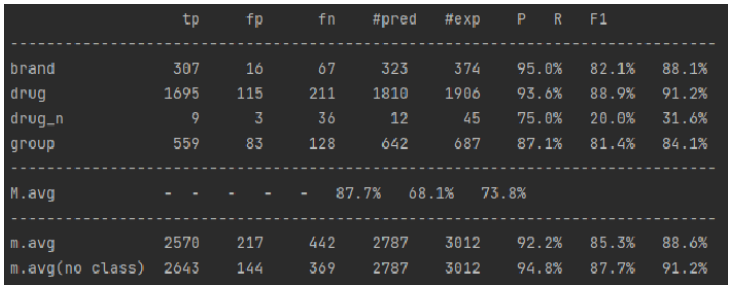
In this section, results obtained on the devel and test datasets, for different algorithms, feature combinations, and parameterizations of the classifiers are presented.

To begin with, the analysis of the feature selection process is presented. As mentioned previously, the testing on features importance and selection was performed following the approach of feature scores and top 10 most important features per label only on CRF with default parametrization.The initial version of the features extraction process achieved a **55%** macro average F1 on ***devel*** test with a total of **8** features, and **56%** on **test** set. Next, the inclusion of the whole feature-set, which consists of **105** features, achieved approximately a **74%** macro average F1 on **devel** and **66.5%** on the test set. The final version of the ***extract_features*** function, which ended up using only **31** features, led to an F1 macro-score of **72%** for the **devel** set and **66.4%** on the test set. Additionally, the execution time of the ***extract-features.py*** script, reduced from **709.28 seconds** (for the whole set of features) to **202.64 seconds** (for the final version of features), a reduction of 71.41%, or approximately **3.5 times faster**, while the predictability power of the model remain approximately the same. In the following figures, the final prediction statistics are depicted for the above-mentioned tries. NB was not included in this experiment, since it only increased its predictability power from **49%** with initial features to **52.5%** with the whole feature space on the ***devel*** set.



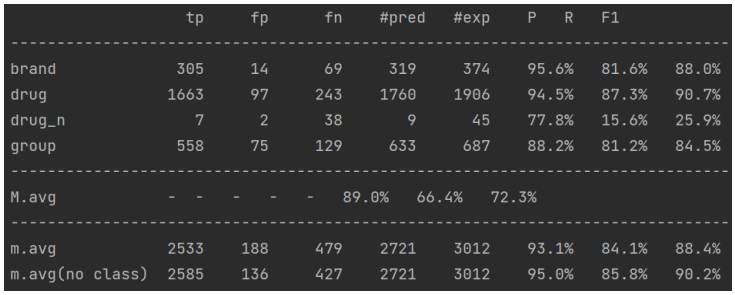
	tp	fp	fn	#pred	#exp	P	R	F1
brand	82	5	292	87	374	94.3%	21.9%	35.6%
drug	1675	181	231	1856	1906	90.2%	87.9%	89.0%
drug_n	3	9	42	12	45	25.0%	6.7%	10.5%
group	550	91	137	641	687	85.8%	80.1%	82.8%
M.avg	-	-	-	-	-	73.8%	49.1%	54.5%
m.avg	2310	286	702	2596	3012	89.0%	76.7%	82.4%
m.avg(no class)	2423	173	589	2596	3012	93.3%	80.4%	86.4%

Figure 2. Statistics with Initial Version of Features - Devel Set



	tp	fp	fn	#pred	#exp	P	R	F1
brand	307	16	67	323	374	95.0%	82.1%	88.1%
drug	1695	115	211	1810	1906	93.6%	88.9%	91.2%
drug_n	9	3	36	12	45	75.0%	20.0%	31.6%
group	559	83	128	642	687	87.1%	81.4%	84.1%
M.avg	-	-	-	-	-	87.7%	68.1%	73.8%
m.avg	2570	217	442	2787	3012	92.2%	85.3%	88.6%
m.avg(no class)	2643	144	369	2787	3012	94.8%	87.7%	91.2%

Figure 3. Statistics with the Whole Feature Set - Devel Set



	tp	fp	fn	#pred	#exp	P	R	F1
brand	305	14	69	319	374	95.6%	81.6%	88.0%
drug	1663	97	243	1760	1906	94.5%	87.3%	90.7%
drug_n	7	2	38	9	45	77.8%	15.6%	25.9%
group	558	75	129	633	687	88.2%	81.2%	84.5%
M.avg	-	-	-	-	-	89.0%	66.4%	72.3%
m.avg	2533	188	479	2721	3012	93.1%	84.1%	88.4%
m.avg(no class)	2585	136	427	2721	3012	95.0%	85.8%	90.2%

Figure 4. Statistics with the Final Feature Set - Devel Set

Now, concerning the hyper-parametrization of the models, we tried several configurations for both NB and CRF. For NB, ***alpha*** is a hyperparameter that is used to prevent zero probability estimates. It is a smoothing parameter that helps to account for the possibility of unseen or rare events in the training data. After applying a grid hyper parametrization test on the alpha parameter with values (0.1, 0.5, 1.0, 1.5, 2.0). We found out that the default value (1.0) was performing best, while the reduction of the value increased the variance of the model, leading to overfitting on the training data and reduced generalization on the test data, while the execution time of the model training did not have a big difference. On the other hand, when alpha was set to higher values, it led to over-smoothing of the data, which resulted in underfitting of the model.

For CRF, the parameters that were modified are the following: ***c1***, ***c2***, ***max_iterations*** and ***min_freq***. The first two mentioned parameters (c1 and c2) control the L1 and L2 regularization terms, respectively, used to prevent overfitting of the model. Parameter *c1* controls the sparsity of the model, while *c2* controls the strength of the regularization. Higher values of *c1* and *c2* result in a more sparse model with stronger regularization. On the other hand, the parameter *max_iterations* specifies the maximum number of iterations to run during training. It can be used

to prevent overfitting or to limit the computational resources required for training. Finally, *min_freq* specifies the minimum frequency threshold for features in the data. Features that occur less frequently than this threshold are ignored during training. This can help to reduce the dimensionality of the feature space and prevent overfitting. In this case, we applied a grid search parametrization, applying the following values for the mentioned parameters: *c1*: [0.01, 0.1, 1.0], *c2*: [0.01, 0.1, 0.25, 1.0], *max_iterations*: [50, 100, 200] and *min_freq*: [1, 2, 3].

The best results were obtained with values *c1* = 1, *c2* = 0.25, *max_iterations* = 100 and *min_freq* = *1.* With this set up the training phase took approx. 10 seconds and achieved the results depicted below. When min_freq was set to higher values then important features were discarded which resulted in reduction of the predictability power. The *max_iterations* parameter did not affect the final results at all. When parameter c1 was receiving values lower than 1 the results worsened significantly while, the value 0.25 for *c2* provided the best result.

	tp	fp	fn	#pred	#exp	P	R	F1
brand	312	15	62	327	374	95.4%	83.4%	89.0%
drug	1693	116	213	1809	1906	93.6%	88.8%	91.1%
drug_n	7	3	38	10	45	70.0%	15.6%	25.5%
group	581	91	106	672	687	86.5%	84.6%	85.5%
M.avg	-	-	-	-	-	86.4%	68.1%	72.8%
m.avg	2593	225	419	2818	3012	92.0%	86.1%	89.0%
m.avg(no class)	2647	171	365	2818	3012	93.9%	87.9%	90.8%

Figure 5. Statistics with the Final Feature Set and Hyper-parametrization - Devel Set

Conclusions

In conclusion, the presented experiments demonstrate the effectiveness of the feature selection process for Name Entity Recognition and Classification (NERC) of drug entities from biomedical documents. Additionally, from Table.1, we can derive some conclusions about the most important features for each distinct target label and try to find a logical explainability. For example, for ***B-drug*** we can see that ***DB_drug***, ***suf3***, ***pre3***, ***length***, ***MTOC***, and ***suf4*** are among the top 10 features. This clearly indicates that external sources provide predictability power and the rest of the features capture different aspects of drug names, such as common prefixes or suffixes, words of big length, and occurrences of multiple capital letters. The same logic can be applied to the rest of the labels.

Additionally, the analysis showed that using a subset of the total feature space led to a reduction in execution time without significant reduction in the model's predictability power. The final version of the feature extraction function, using only 31 features, achieved a macro-average F1 score of 72% on the devel set and 66.4% on the test set, which is comparable to the results obtained using the whole set of features. The experiments also revealed that the CRF algorithm outperformed other algorithms in this task, with an F1 score of 66.4% on the test set. Finally by

applying the optimal hyper parameters on CRF, it increased its predictive power only for a minimum amount of **0.5%**.

Overall, these findings highlight the importance of feature selection and model hyper-parameterization in NERC tasks and the potential for improving both execution time and model performance.