Advanced Human Language Technology

Task 1 & 2

Valeriu Vicol, Guillermo Creus

Barcelona, March 27, 2022

Contents

1 Introduction

This work is based in the Named Entity Recognition and Classification (NERC) task proposed in SemEval 2013, the 7th International Workshop on Semantic Evaluations. In particular, the main goal of this report is to extract relevant Named Entities (NE) from the provided biomedical texts and classifying them into four main groups: "drug", "drug_n", "group" and "brand".

On the other hand, the scope of this lab was to explore the data provided in order to understand it better and be able to build two types of classifiers. The first one is a rule-based classifier, which given a tokenized sentence it will classify each token into 5 groups: "drug", "drug_n", "group", "brand" and "NONE".

For example, the following sentence: "Maximal exercise testing, a maneuver often applied to cardiac patients, does not significantly alter the serum digoxin level." contains the drug digoxin. Consequently, rules will have to be developed so that (ideally) every token gets a prediction of "NONE" except digoxin.

Once the performance of this classifier has been assessed, a set of handcrafted features will be designed in order to feed them to a Machine Learning based system. Finally, the 2 developed systems will be compared to determine the one that has a better performance, according to a selected metric. In the chapters below, the work performed to solve this task will be explained, as well as the results obtained.

2 Rule-based baseline

2.1 Exploratory data analysis

In order to build the rule-based classifier, it is important to get insights on the data provided. The whole dataset is divided into three datasets: train, devel and test. The first one will be used to train the models, the second one to validate them and the last one to test them.

Table 1: Basic information on the train dataset

	Size	Unique words	Mean Length (μ)	Std Length (σ)	Length interval $\mu \pm \sigma$
Drug_n	527	23	16.43	9.37	[7.07, 25.80]
Drug	7256	540	11.02	3.29	[7.73, 14.30]
Brand	1158	123	7.98	2.75	[5.23, 10.74]
Group	2686	350	19.04	9.28	[9.76, 28.32]

As it can be seen in Table 1, the Drug class is the most populous one at 7256 instances, with 540 different ones. Also, Drug_n is the minority class and it is the most diverse class. That is, the proportion of different words is the highest among all of the classes. That means that it will be harder to predict since the case history will be more diverse.

Below a more detailed analysis of the dataset is shown; every field is a key, value dictionary that indicates a feature on a class. For example, if the frequency of number of digits and/or capital letters in tokens is equal to 2 it means that in the token there are two characters in the digits and/or capital letters set.

Drug_n:

Size of the dataset: 527
Frequency of digits in the token: {0: 404, 2: 30, 3: 10, 1: 51, 4: 6, 5: 12, 6: 14}
Frequency of number of capital letters in the token: {0: 275, 1: 50, 2: 50, 3: 96, 4: 49, 5: 7}

Frequency of number of characters ∈ {−; _; ,} and/or capital letters in tokens: {0: 202, 1: 49, 2: 62, 3: 118, 4: 40, 5: 17, 6: 17, 7: 8, 8: 14}
Frequency of 3 characters prefixes: [('pcp', 22), ('18-', 17), ('ibo', 17), ('1,3', 16), ('(-)', 15)]
Frequency of starting letter: [('n', 474), ('r', 368), ('t', 361), ('l', 329), ('-', 301)]
Frequency of number of dashes in tokens: {0: 377, 1: 89, 3: 16, 2: 23, 6: 8, 5: 14}

Drug:

Size of the dataset: 7256
Frequency of digits in the token: {0: 7186, 1: 35, 2: 14, 3: 5, 5: 15, 4: 1}
Frequency of number of capital letters in the token: {0: 5929, 4: 19, 1: 1185, 2: 60, 3: 57, 7: 2, 14: 2, 9: 1, 5: 1}
Frequency of number of characters ∈ {−; _; ,} and/or capital letters in tokens: {0: 5892, 4: 33, 1: 1199, 3: 46, 2: 77, 7: 2, 14: 2, 5: 3, 9: 1, 6: 1}
Frequency of 3 characters prefixes: [('phe', 247), ('met', 213), ('war', 195), ('dig', 171), ('pro', 164)]
Frequency of starting letter: [('n', 7063), ('r', 4697), ('l', 4558), ('t', 4356), ('c', 3208)]
Frequency of number of dashes in tokens: {0: 7179, 1: 75, 2: 2}

Brand:

Size of the dataset: 1158
Frequency of digits in the token: {0: 1150, 2: 7, 1: 1}
Frequency of number of capital letters in the token: {5: 93, 1: 382, 11: 4, 9: 84, 7: 248, 8: 125, 2: 25, 10: 10, 6: 98, 0: 58, 3: 20, 4: 9, 16: 2}
Frequency of number of characters ∈ {−; _; ,} and/or capital letters in tokens: {5: 95, 1: 383, 11: 4, 9: 84, 7: 248, 8: 125, 3: 27, 10: 10, 6: 98, 0: 57, 4: 8, 2: 17, 16: 2}
Frequency of 3 characters prefixes: [('asp', 65), ('ind', 30), ('tax', 25), ('peg', 24), ('tri', 24)]
Frequency of starting letter: [('n', 696), ('r', 693), ('l', 526), ('t', 492), ('s', 411)]
Frequency of number of dashes in tokens: {0: 1146, 1: 12}

Group:

Size of the dataset: 2686
Frequency of digits in the token: {0: 2640, 1: 36, 2: 8, 3: 2}
Frequency of number of capital letters in the token: {0: 1929, 2: 66, 1: 314, 4: 82, 3: 201, 5: 87, 22: 1, 14: 1, 6: 5}
Frequency of number of characters ∈ {−; _; ,} and/or capital letters in tokens: {0: 1766, 1: 438, 2: 95, 4: 92, 3: 192, 5: 77, 22: 1, 14: 1, 6: 20, 7: 3, 9: 1}
Frequency of 3 characters prefixes: [('ant', 491), ('tri', 98), ('bet', 78), ('cor', 73), ('con', 72)]
Frequency of starting letter: [('t', 3945), ('s', 3837), ('n', 3638), ('c', 2493), ('r', 2416)]
Frequency of number of dashes in tokens: {0: 2440, 1: 229, 2: 15, 3: 2}

In addition, the 5 character suffixes were obtained for the 4 classes. It must be added that from the exploratory data analysis not many useful insights were obtained since the frequency of the rules is not sufficient to be precise. However, it must be noted that an external source of knowledge will be used to complement the analysis done.

2.2 Rule-set construction

As it was stated in the previous section, an external source of knowledge will be used. This consists of two txt files, where one is a drugbank and the other has information on drugs, brands and groups. Unfortunately, the class drug_n is not present in these files, so in the first approach, there will be a clear focus on this class. In addition,

rules will be evaluated with the average of the F1 score of the 4 class classification.

The construction process was the following:

1. If the token is in the external source of knowledge, then predict its class (100% accuracy).
   F1 Macro average (devel) = 44.0%. This result is surprising, since it is not even predicting drug_n.

2. It can be seen that drug_n is the only class with multiple tokens with ≥ 3 dashes. Therefore, if the token has more than 3 dashes, it will be predicted as drug_n.
   F1 Macro average (devel) = 45.0%.

3. From the past subsection, it was observed that if a token has digits, commas, dashes and/or underscores it was more likely to come from the class drug_n.
   F1 Macro average (devel) = 45.7%.

4. It was time to analyze whether the suffixes increased the macro average. The rules added said that the prediction of the class of a token if its 5 character suffix belonged to the set of the 10 most common suffixes of that class. The best result (considering all possible permutations, which affect the result) was:
   F1 Macro average (devel) = 45.0%. Since the result decreased the score, the rules were discarded.

5. As it can be seen in Table 1, the length of the class group is the one with the interval with the largest length. Therefore, a rule was created so that a token would be predicted as a group if the length was greater than 25 characters. Unfortunately, this rule was discarded since the best permutation yielded a macro average of 45.7%.

6. Lastly, as the number of uppercase characters is abnormally high in the class brand, a rule was created but it dropped the macro average to 45.0 %.

In the end, only 3 rules were enough to boost the score to 45.7% in the validation set (devel). No matter how the rules were created, they dropped the score, so these three will be the only ones used in the rule-based classifier.

2.3 Code

```
def classify_token(txt):
    # If token is in the external source of knowledge, give its prediction
    if txt.lower() in external:
        return external[txt.lower()]
    elif txt.count('-') >= 3:
        return 'drug_n'
    elif sum(i.isupper() or i in set(['-', '_', ',']) for i in txt) >= 3:
        return "drug_n"
    else:
        return "NONE"
```

2.4 Experiments and results

As one can see in Figures 1, 2, 3 the metrics of the rule-based classifier are very similar for all 3 data-sets, at around 45%. This is a good sign because it means that the rules are consistent across the three datasets, implying a uniform data distribution.

One can see from Figures 1, 2, 3 that the rule-based classifier designed is able to detect most of instances from class `brand` and `drug`, but it struggles to detect classes `drug_n` and `group`.

Clearly, the way to go would be to define more rules oriented to these two classes. Several strategies were tried, but they only decreased the score. A clear option would be to create a data bank for drug_n instances, which would probably boost the score. However, since the goal of this exercise was to create rules, this strategy was not pursued.

Additionally, during the execution of this part it was concluded that the rules have to very specific for each class in order to be effective. In most of the cases when new rules were added, they were decreasing the scores. Overall, having 3 rules and achieving close to 45% as F1 macro average in the test results is satisfactory at this stage, despite having plenty of room for improvement, where a ML based would be more convenient.

	tp	fp	fn	#pred	#exp	P	R	F1
brand	1005	1121	153	2126	1158	47.3%	86.8%	61.2%
drug	5957	1171	1299	7128	7256	83.6%	82.1%	82.8%
drug_n	133	1906	394	2039	527	6.5%	25.2%	10.4%
group	597	1014	2089	1611	2686	37.1%	22.2%	27.8%
M.avg	-	-	-	-	-	43.6%	54.1%	45.5%
m.avg	7692	5212	3935	12904	11627	59.6%	66.2%	62.7%
m.avg(no class)	8522	4382	3105	12904	11627	66.0%	73.3%	69.5%

Figure 1: Metrics on train

	tp	fp	fn	#pred	#exp	P	R	F1
brand	319	252	55	571	374	55.9%	85.3%	67.5%
drug	1588	305	318	1893	1906	83.9%	83.3%	83.6%
drug_n	21	528	24	549	45	3.8%	46.7%	7.1%
group	137	282	550	419	687	32.7%	19.9%	24.8%
M.avg	-	-	-	-	-	44.1%	58.8%	45.7%
m.avg	2065	1367	947	3432	3012	60.2%	68.6%	64.1%
m.avg(no class)	2276	1156	736	3432	3012	66.3%	75.6%	70.6%

Figure 2: Metrics on devel

	tp	fp	fn	#pred	#exp	P	R	F1
brand	251	284	23	535	274	46.9%	91.6%	62.1%
drug	1631	295	496	1926	2127	84.7%	76.7%	80.5%
drug_n	12	426	60	438	72	2.7%	16.7%	4.7%
group	182	294	511	476	693	38.2%	26.3%	31.1%
M.avg	-	-	-	-	-	43.1%	52.8%	44.6%
m.avg	2076	1299	1090	3375	3166	61.5%	65.6%	63.5%
m.avg(no class)	2306	1069	860	3375	3166	68.3%	72.8%	70.5%

Figure 3: Metrics on test

3 Machine learning NERC

3.1 Feature extraction

First of all, it is relevant to say that in this task, since the amount of features can be so huge, they will be stored in a sparse way, and all of them will be binary. That is, the name of the feature will be stored for an instance, if and only if it is activated in that instance. Additionally, it is important to comment on the nature of the features, which can be of two types: word and context type. That is, they can include information of the form being predicted, or its context.

Having said that, the word feature types are the following:

- Form and 3 character suffix
- External source of knowledge
- Number of dashes
- Whether the token is all uppercase
- Number of digits in the word
- Whether the token had a suffix from the 5 most common ones from one of the 4 classes.
- If the token belongs to the string punctuation
- If the token has slashes
- Last letter of the token

The context ones are the features from the neighbor tokens. For example, given a depth of 1, it will evaluate the features from the previous list on the left and right tokens (at distance one). Depth equal to 2 would extract features from the left and right tokens at distance 1 and 2.

These features, although seeming random, have been carefully selected and they were added in a sequential manner. That is, first, only the features of word type were used. After having selected the best and achieving an F1 macro average of 64.3% (in devel), it was time to add the features of context type, which helped boost the score to 74.2%.

It must be said, that this model was much more forgiving than the rule-based one because there was almost no need to remove features, since almost all of the features helped increase the metrics. There were only a few features that did not improve (so they were removed):

- Number of uppercase characters
- Lowercase form of the token
- 4 and 5 character suffix

Lastly, a parameter that was tuned was the depth. A hypothesis was that increasing the depth to 2 or 3 tokens to the right and left would increase the score. It turned out to be false, decreasing the score slightly.

3.2 Code

```
def extract_features(tokens):

    most_common_suffixes = {
        # 3 character suffixes
        3: {
            'drug': ['ide', 'cin', 'ole', 'one'],
            'brand': ['rin', 'CIN', 'XOL', 'SYS', 'RON'],
            'group': ['nts', 'ics', 'nes', 'ors', 'ids'],
            'drug_n': ['ate', 'PCP', 'ANM', '-MC']
        },
        # 4 character suffixes
        4: {
            'drug': ['pine', 'zole', 'mine', 'arin'],
            'brand': ['irin', 'OCIN', 'AXOL', 'ASYS', 'IOXX'],
            'group': ['tics', 'ants', 'tors', 'ines', 'ents'],
            'drug_n': ['NANM', 'aine', '8-MC']
        },
        # 5 character suffixes
        5: {
            'drug': ['azole', 'amine', 'farin', 'mycin'],
            'brand': ['pirin', 'DOCIN', 'TAXOL', 'GASYS', 'VIOXX'],
            'group': ['gents', 'itors', 'sants', 'etics', 'otics'],
            'drug_n': ['gaine', '18-MC', '-NANM']
        }
    }

    result = []
    for k in range(0, len(tokens)):
        tokenFeatures = []
        # Extract the token
        t = tokens[k][0]

        # ------ WORD FEATURES ------

        # Append the base form
        tokenFeatures.append("form=" + t)
        # Append the 3 char suffix
        tokenFeatures.append("suf3=" + t[-3:])

        # External source of knowledge
        if t.lower() in external:
            tokenFeatures.append("externalDictSays=" + external[t.lower()])

        # Number of dashes
        n_dashes = len(re.findall('-', t))
        tokenFeatures.append("numberOfDashes=" + str(n_dashes))

        # Indicates if the token is all uppercase
        n_upper = sum(i.isupper() for i in t)
        if n_upper == len(t):
            tokenFeatures.append("allUppercase")

        # Number of digits that saturates at 3
        n_digits = len(re.findall('\d', t))
        n_digits = '=' + str(n_digits) if n_digits < 3 else '>=3'
        tokenFeatures.append("numberOfDigits" + n_digits)

        # Includes the most common suffixes (3, 4 and 5 char suffixes)
        for n_suffix in most_common_suffixes.keys():
            for d_class in ['drug', 'brand', 'group', 'drug_n']:
```

```
            if t[-n_suffix:] in most_common_suffixes[n_suffix][d_class]:
                tokenFeatures.append(
                    "suffixBelongsToClass_" + d_class + "_WithNoChars=" + str(n_suffix
                        ))

    # If it's punctuation
    if t in punctuation:
        tokenFeatures.append("tokenIsPunctuation")

    # Slashes
    tokenFeatures.append("/inToken=" + str('/' in t))
    # Last letter
    tokenFeatures.append("lastLetter=" + t[-1])

    # ------------------------------------------------

    # ---------------- CONTEXT FEATURES ---------------

    # Depth --> Looks left and right (same features)
    for d in range(1, depth + 1):
        if k >= 0 + d:
            tPrev = tokens[k - d][0]
            tokenFeatures.append("depth-" + str(d) + "_" + "formPrev=" + tPrev)
            tokenFeatures.append("depth-" + str(d) + "_" + "suf3Prev=" + tPrev[-3:])

            if tPrev in punctuation:
                tokenFeatures.append("depth-" + str(d) + "_" + "prevTokenIsPunctuation
                    ")

            if tPrev.lower() in external:
                tokenFeatures.append("depth-" + str(d) + "_" + "externalDictSaysPrev="
                    + external[tPrev.lower()])

            n_dashes = len(re.findall('-', tPrev))
            tokenFeatures.append("depth-" + str(d) + "_" + "numberOfDashesPrev=" + str
                (n_dashes))

            n_upper = sum(i.isupper() for i in tPrev)
            if n_upper == len(tPrev):
                tokenFeatures.append("depth-" + str(d) + "_" + "allUppercasePrev")


            n_digits = len(re.findall('\d', tPrev))
            n_digits = '=' + str(n_digits) if n_digits < 3 else '>=3'
            tokenFeatures.append("depth-" + str(d) + "_" + "numberOfDigitsPrev" +
                n_digits)

            for n_suffix in most_common_suffixes.keys():
                for d_class in ['drug', 'brand', 'group', 'drug_n']:
                    if tPrev[-n_suffix:] in most_common_suffixes[n_suffix][d_class]:
                        tokenFeatures.append("depth-" + str(d) + "_" + "
                            prevSuffixBelongsToClass_" + d_class + "_WithNoChars=" +
                            str(n_suffix))

            tokenFeatures.append("depth-" + str(d) + "_" + "/inToken=" + str('/' in
                tPrev))
            tokenFeatures.append("depth-" + str(d) + "_" + "lastLetter=" + tPrev[-1])

        elif k == 0:
            tokenFeatures.append("BoS")

        if k <= len(tokens) - 1 - d:
```

```
                tNext = tokens[k + d][0]
                tokenFeatures.append("depth+" + str(d) + "_" + "formNext=" + tNext)
                tokenFeatures.append("depth+" + str(d) + "_" + "suf3Next=" + tNext[-3:])


                if tNext in punctuation:
                    tokenFeatures.append("depth+" + str(d) + "_" + "nextTokenIsPunctuation
                        ")

                if tNext.lower() in external:
                    tokenFeatures.append("depth+" + str(d) + "_" + "externalDictSaysNext="
                        + external[tNext.lower()])

                n_dashes = len(re.findall('-', tNext))
                tokenFeatures.append("depth+" + str(d)
               + "_" + "numberOfDashesNext=" + str(n_dashes))

                n_upper = sum(i.isupper() for i in tNext)
                if n_upper == len(tNext):
                    tokenFeatures.append("depth+" + str(d) + "_" + "allUppercaseNext")

                n_digits = len(re.findall('\d', tNext))
                n_digits = '=' + str(n_digits) if n_digits < 3 else '>=3'
                tokenFeatures.append("depth+" + str(d) + "_" + "numberOfDigitsNext" +
                    n_digits)

                for n_suffix in most_common_suffixes.keys():
                    for d_class in ['drug', 'brand', 'group', 'drug_n']:
                        if tNext[-n_suffix:] in most_common_suffixes[n_suffix][d_class]:
                            tokenFeatures.append("depth+" + str(d) + "_" + \\ "
                                nextSuffixBelongsToClass_" + d_class + "_WithNoChars=" +
                                str(n_suffix))

                tokenFeatures.append("depth+" + str(d) + "_" + "/inToken=" + str('/' in
                    tNext))
                tokenFeatures.append("depth+" + str(d) + "_" + "lastLetter=" + tNext[-1])

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

            # ------------------------------------------------------------

        result.append(tokenFeatures)

    return result
```

3.3 Selected algorithm

The base algorithm used in the ML based classifier is a `crf` with parameters `feature.minfreq=1` and `c2=0.1`, both selected after a thorough hyper-parameter selection. For further information, one can go to Table 2, where different F1 macro averages are reported for different hyper-parameter combinations. With default parameters the F1 macro average is 74.2%, which is the highest.

Table 2: F1 macro average in Score (devel) for different hyper-parameters

`feature.minfreq` /`c2`	0.1	0.5	1
1	74.2%	70.4%	67.2%
3	66.4%	66.9%	66.9%
5	68%	66.5%	66.8%

Also, as part of this lab, a maximum entropy model was set up to compare results. Unfortunately, due to system incompatibility it was impossible to run it in the authors' computers. Even running everything on docker was not enough since some errors still popped up, so it was decided that `crf` would be the model for the ML based classifier.

3.4 Experiments and results

	tp	fp	fn	#pred	#exp	P	R	F1
brand	1132	14	26	1146	1158	98.8%	97.8%	98.3%
drug	6955	122	301	7077	7256	98.3%	95.9%	97.0%
drug_n	397	54	130	451	527	88.0%	75.3%	81.2%
group	2492	105	194	2597	2686	96.0%	92.8%	94.3%
M.avg	-	-	-	-	-	95.3%	90.4%	92.7%
m.avg	10976	295	651	11271	11627	97.4%	94.4%	95.9%
m.avg(no class)	11042	229	585	11271	11627	98.0%	95.0%	96.4%

Figure 4: Model results on train

	tp	fp	fn	pred	exp	P	R	F1
brand	307	14	67	321	374	95.6%	82.1%	88.3%
drug	1745	121	161	1866	1906	93.5%	91.6%	92.5%
drug_n	9	2	36	11	45	81.8%	20.0%	32.1%
group	547	73	140	620	687	88.2%	79.6%	83.7%
M.avg	-	-	-	-	-	89.8%	68.3%	74.2%
m.avg	2608	210	404	2818	3012	92.5%	86.6%	89.5%
m.avg(no class)	2648	170	364	2818	3012	94.0%	87.9%	90.8%

Figure 5: Model results on devel

	tp	fp	fn	#pred	#exp	P	R	F1
brand	240	26	34	266	274	90.2%	87.6%	88.9%
drug	1861	99	266	1960	2127	94.9%	87.5%	91.1%
drug_n	4	4	68	8	72	50.0%	5.6%	10.0%
group	544	114	149	658	693	82.7%	78.5%	80.5%
M.avg	-	-	-	-	-	79.5%	64.8%	67.6%
m.avg	2649	243	517	2892	3166	91.6%	83.7%	87.5%
m.avg(no class)	2736	156	430	2892	3166	94.6%	86.4%	90.3%

Figure 6: Model results on test

In Figure 4 we can observe the results of the crf for train dataset, as expected we have good results for training data-set, model has good scores for all classes, lowest f1 score we have for class `drug_n` 81.2%, which means that model clearly can identify entities of this class comparing to ruled based classifier where we have only 10% f1 score for the same class.

Additionaly, in Figure 5 one can observe the results of devel dataset overall we have good f1 score of 74%, and also we can observe that results for class `drug_n` decreased significantly with 49%, which is reasonable as the data-set is unbalanced and we don't have to many examples of this class, so model is not able to learn well particular properties of this class.

Lastly, in Figure 6 one can see the results of the model on the test dataset. The overall f1 score is not bad at 67%, but as in devel dataset we have the same problem with `drug_n` class.

For future work, in order to solve this problem, we should create some data augmentation scripts for increasing the presence of this class in dataset. With this improvement, it would easier to increase overall score of classifier.

4 Conclusion

In conclusion, two different classifiers were designed in order to solve the NERC problem. Both methods were successful at this task, showing that NLP techniques are very useful in day to day problems. They showed the importance of analyzing data with the purpose of creating features and rules.

The rule based classifier achieved a 44.6 % F1 macro average, compared to a 67.6% of the ML based. The difference in performance was expected since the rule based was very limited to the rules created, their order and their precision. In the ML based one was able to develop as much features as one wanted in the design process, and tune them later. That way, a lot more features were fed to the model and its chances of achieving a higher F1 macro average were higher.

For future work, it would be interesting to try more advanced techniques, were the model has the freedom to explore different feature combinations. That is, the model would also be responsible for designing the features fed to the algorithm. That way, a human would not be able to manually tune the features, which is an inefficient process since one can not know when it is a good time to stop adding or removing features.