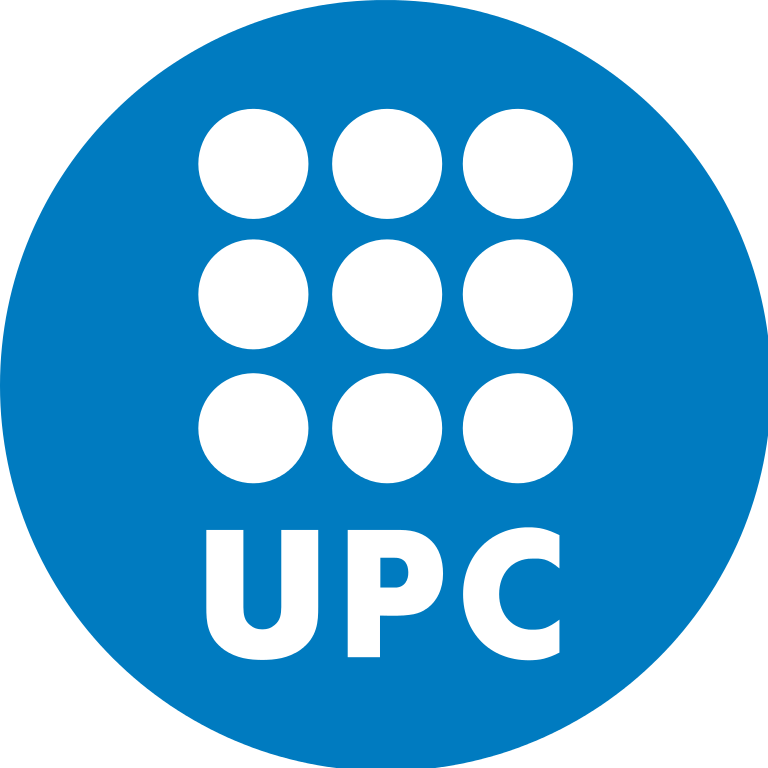
**UNIVERSITAT POLITÈCNICA DE CATALUNYA**

**BARCELONA SCHOOL OF INFORMATICS**

**Official Master on Data Science**

****

Lab 2: ML-NERC Task

**Mining Unstructured Data**

**Albert Martín Garcia**

**Ximena Moure Alassio**

**Maria Paraskeva**

Barcelona, April 2023

## 

[**Introduction 1**](#_86pkh6egmn2l)

[**Machine Learning NERC 1**](#_ja7hp9ljlbsc)

[Initial Version 1](#_aim7nzcleoez)

[First Model\* 2](#_uxis7gol0fib)

[Results and Assumptions of Model 1 3](#_5g13fal5jv2v)

[Second Model 5](#_ckk2f1em88os)

[Results and Assumptions of Model 2 6](#_gwh2dvb4qdla)

[Code 7](#_4ckwmv4kbzib)

[**Conclusions 13**](#_5kiuvichmlqo)

[**Annex 13**](#_c43apl6477k6)

[First Model 13](#_3gevh9ubxpxk)

[Second Model 17](#_xm7hbfvbh9io)

## 

## Introduction

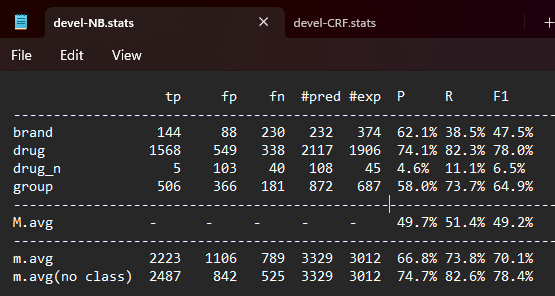
The following report concerns the act of recognizing and classifying drug names found in XML files given as argument. The objective is to try different methods and create a feature extractor that gives the best results possible. After that, with the help of a learner we will teach the model and, finally, using a classifier (in our case, two different ones) we will check our results.

## Machine Learning NERC

### Initial Version

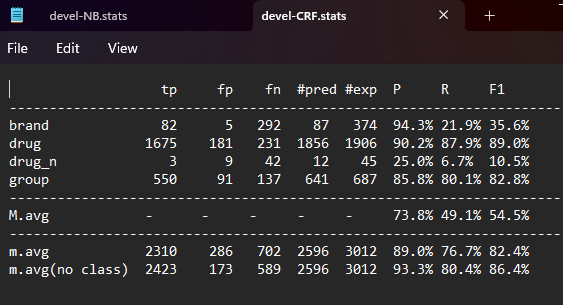
**Naive bayes**

The results of the initial version of the dataset show a 49.2% macro average F1 score for Naive Bayes. The micro average is higher at 70.1%, which means that the most frequent classes are performing more effectively in general than the average of each class separately, which is calculated by the macro average. It is apparent that the classes are imbalanced.



**CRF**

Regarding the Conditional random field model, it only achieves a 54.5% macro average F1 score and a 82.4% micro score. Similarly to the results of the Naive Bayesian method, the classes are imbalanced and the performance of the classes with the most examples are all better.



### [[1]](#footnote-0)First Model

Feature Extraction

Considering that we are dealing with drugs we decided to analyze a bit how drug names are formed and to look at the text provided and based on this we observed the following:

* Special characters: many drug names have hyphens in them.
* Suffixes: they are added to the end of the generic name of a drug and they are used to provide additional information about the drug. For example, endings in ‘-caine’, ‘-ol’, ‘-ine’ are quite common.
* Prefixes: they are also used to provide additional information about the drug. For example, ‘anti-’, ‘di-’, ‘peni-’ are very common.
* Word length: the name of the drugs tends to be quite long as well as the groups that can also have more than one word. Oppositely, brands are prone to have shorter names.
* Capitalization; Uppercase/Lowercase: in the data provided we observed that most of the times drug names are lowercase while drug brands are in uppercase. So, it would be good to check for lowercase and uppercase during feature extraction.
* Plural: many groups of drugs tend to finish with the letter ‘-s’, like ‘opiods’, ‘tranquillizers’, ‘analgesics’, etc.
* Lookup tables: we are provided with two lookup tables. So, we decided to add this to our feature extractor. We look in both lookup tables and based on the label it has on them we add 4 new features which tell if it is a drug, drug\_n, brand or group.

1. External sources Lookup

Using the Drug Bank text file, we tried to check whether a token existed in it and return a Boolean value. Scanning the text file was extremely time consuming at the point where it needed more than half an hour to terminate. In order to overcome this, a dictionary was created. The results were clearly better.

Similarly, we had the same approach using the HS database file as an additional external lookup. Finally, we achieved a 56.1% macro average F1 score in Naive Bayes and 66.5% in CRF.

1. Capitalization - Camelcase - Plural

Continuing, we checked whether a token is capitalized, includes a capital letter in the middle, or ends in -s, which would indicate the plural form of the noun. There was a 7.6% increase in the F1 score of Naive Bayes, but only a 2.4% increase in CRF.

1. Suffixes and Prefixes

Another metric we implemented was looking at the suffixes and prefixes. Making a list of the most commonly used suffixes with three, four, five, and six characters as well as a list of the most frequently used prefixes, we were able to achieve a small increase in the macro average F1 score Naive Bayes (0.4%) while the CRF score was decreased by 0.1%.

1. Numbers and Dash

Last but not least, we checked for the existence of numbers and dashes in the tokens. As can be seen below, the F1 score of the Naive method dropped back to 62.2% while that of CRF went up to 74.3%.

### Results and Assumptions of Model 1

**Naive Bayes Result Table**

|  | External Sources | Capitalization  Camelcase  Plural | Suffixes  Prefixes | Numbers  Dash |
| --- | --- | --- | --- | --- |
| brand | 69.4% | 77% | 77% | 77.1% |
| drug | 83.9% | 88.3% | 88.3% | 88.1% |
| drug\_n | 7.8% | 8.9% | 10.7% | 8.9% |
| group | 63.3% | 74.5% | 74.4% | 74.7% |
| Macro Average | **56.1%** | **62.2%** | **62.6%** | **62.2%** |
| Micro Average | 75.2% | 81.6% | 81.7% | 81.6% |

**CRF Result Table**

|  | External Sources | Capitalization  Camelcase  Plural | Suffixes  Prefixes | Numbers  Dash |
| --- | --- | --- | --- | --- |
| brand | 88.1% | 90.5% | 90.5% | 90.7% |
| drug | 92.8% | 92.9% | 92.7% | 92.9% |
| drug\_n | 4% | 26.2% | 26.2% | 29.5% |
| group | 81% | 83.9% | 83.8% | 84.3% |
| Macro Average | **66.5%** | **73.4%** | **73.3%** | **74.3%** |
| Micro Average | 88.9% | 89.9% | 89.7% | 90% |

We managed to achieve a higher F1 score in the tables above by changing the order of which extractor functions we were adding each time.

After some trials, we observed that a 62.6% macro average F1 score can be reached in the Naive Bayes method if numbers and dashes are not examined in the tokens. The F1 score sees a 0.04% decrease when checking the tokens for numbers or dashes resulting in the final 62.2% score.

Regarding the Conditional Random Field method, we reached a 74.4% F1 score by leaving the examination of suffixes and prefixes out. When added to the feature extractor, we notice a small decrease resulting in a 74.3% F1 score. It is a curious case, as one would think that classifying each token based on their suffix/prefix would improve the efficiency rather than making it worse.

The drugs that were first developed by a pharmaceutical company (brand) show a bigger increase in accuracy when using the Naive Bayes method than the CRF.

The same applies to the generic drugs (drug) showing a 4% increase to the F1 score with Naive Bayes while staying almost stable with CRF.

The biggest increase is by far when using the CRF method in drugs that haven’t been approved for human consumption (drug\_n), going from 4% all the way up to 29.5%.

Lastly, when dealing with groups of drugs (group) we can see that the Naive Bayes method allows a 11.4% increase in the macro average F1 score, while the CRF method only a 3.3%.

Regarding the difference in percentages between macro average and micro average F1 score, it is observed that the latter is always higher than the first one. This shows that the most frequent classes are performing well on the models (both Naive and CRF), so, as the micro average accentuates them, a higher score is expected. Comparing our classes we can clearly see that there is an imbalance, as the drugs that have not been approved to be used on humans for medical reasons have a much lower percentage of instances in our data set. This fact makes the micro average not a very accurate metric in order to evaluate our models’ effectiveness.

It may have come to your attention that we have not been discussing the micro average without a class that is included in the last row of each of our .stats tables. The reason is that micro average without class is not a meaningful concept in NLP.

Micro average aggregates the contributions of each class and calculates the metric based on the total number of true positives, false positives, and false negatives across all classes.

It always requires the classes to be defined and compared against each other. It emphasizes the performance of the model on the most frequent classes and can be used to evaluate the overall effectiveness of the model in terms of correctly predicting positive and negative examples for each class. Therefore, it is not possible to calculate a micro average without class, so we only dealt with the micro average that has classes.

### Second Model

We continue with the feature extraction of the second model:

1. Adding suffix of length 3 to 2 words prior and latter

First thing we did was to add the suffix of length 3 to the 2 previous and subsequent words. This improved the F1 score of the CRF model ( to 57.3%) but it diminished the F1 score of the NB model by 2% (to 47.2%). After testing this feature with the other ones and seeing the results, we decided to not use this feature anymore. So, the following results are without using this feature.

1. Adding the length of the word and the previous word

We added as a feature the length of the word and the length of the previous word. This increased the baseline F1 score of the CRF model to 63.5% and the F1 score of the NB model to 51.5%.

1. Capitalization and Camelcase

Then three more features were added: one that tells if the word is in uppercase, another that tells if it is in lowercase and the last one that tells if it is in camelcase.

These features improved the F1 score of the CRF model to 73.2%, which is an increase of 9.7 %, and the F1 score of the NB model to 62% which is an increase of 10.5%.

1. Special characters

Two more features were added to check for special characters: one that tells if the word has dashes and another one that indicates if it has numbers in it.

With these two features we didn't see a massive improvement like with the previous one, but they did improve both models' F1 score. CRF F1 score improved to 75.6% and NB F1 score improved to 62.1%.

1. Plurals

We added a feature to tell if a word ends in plural. This did not improve neither of the F1 scores. CRF score remained the same and the NB score had a very small decrease of 0.1%.

1. Prefixes and suffixes

We made a list of the most commonly used suffixes with three, four, five, and six characters as well as a list of the most frequently used prefixes. We were able to achieve a small increase in the macro average F1 score Naive Bayes (0.2%) while the CRF score decreased by 1.2%.

We tried using only suffixes with three or four characters to see if the length of the suffix had an effect but the F1 scores remained almost the same as when using suffixes with more characters.

1. Lookup tables

Finally, we added four new features which tell if it is a drug, drug\_n, brand or group based on both lookup tables. We first tried scanning the whole text file without using any auxiliary structure but it was extremely time consuming to the point where it needed more than half an hour to terminate. In order to overcome this, a dictionary was created. The results were clearly better.

We tried adding these additional features taking into account the prefixes and suffixes and without taking them into account but the result was better when we did consider them.

As a result, we ended up with a F1 score of 64% for the NB model and a F1 score of 77.1% for the CRF model.

### Results and Assumptions of Model 2

**Naive Bayes Result Table**

|  | Length | Capitalization  Camelcase | Special Characters | Plurals | Suffixes  Prefixes | External Sources |
| --- | --- | --- | --- | --- | --- | --- |
| brand | 56.1% | 77% | 75.7% | 75.5% | 76.1% | 81% |
| drug | 79.2% | 75% | 88.3% | 87.9% | 88.4% | 89.7% |
| drug\_n | 6.9% | 88.3% | 10.5% | 9.5% | 9.8% | 11.5% |
| group | 63.9% | 10.4% | 73.9% | 75% | 74.7% | 73.8% |
| Macro Average | **51.5%** | **62.0%** | **62.1%** | **62%** | **62.3%** | **64%** |
| Micro Average | 71.1% | 81.2% | 80.8% | 80.8% | 81.1% | 82.4% |

**CRF Result Table**

|  | Length | Capitalization  Camelcase | Special Characters | Plurals | Suffixes  Prefixes | External Sources |
| --- | --- | --- | --- | --- | --- | --- |
| brand | 59.4% | 88.7% | 89.1% | 89.1% | 89.2% | 91.1% |
| drug | 90.6% | 91.3% | 91.5% | 91.5% | 91.6% | 93.3% |
| drug\_n | 19.7%% | 27.6% | 36.1% | 36.1% | 31.6% | 38.1% |
| group | 84.2% | 85.1% | 85.6% | 85.5% | 85.1% | 85.8% |
| Macro Average | **63.5%** | **73.2%** | **75.6%** | **75.6%** | **74.4%** | **77.1%** |
| Micro Average | 85.3% | 88.9% | 89.3%% | 89.3% | 89.2% | 90.7% |

### Code[[2]](#footnote-1)

#! /usr/bin/python3

import sys

import re

from os import listdir

from xml.dom.minidom import parse

from nltk.tokenize import word\_tokenize

number\_patter = r'\d+'

suf\_list3 = ["mab", "pam", "vir", "cet", "ase", "cog", "vir", "min"]

suf\_list4 = ["olol", "pril", "tide", "pram", "zine", "adol", "afil", "anib", "grel", "imex", "imus"]

suf\_list5 = ["artan", "caine", "mycin", "zepam", "terol", "zosin", "bufen", "butan", "ectin", "pamil", "parin", "pidem", "zumab", "fenac"]

suf\_list6 = ["dipine", "statin", "cillin", "nazole", "tidine", "sartan", "axine", "azenil", "azepam", "azosin", "bactam", "bamate", "bersat", "curium", "giline", "gillin", "irudin", "leukin", "lukast", "orphan", "oxacin", "pamide", "poetin", "zolast"]

prefixes = ['acet', 'al', 'am', 'anti', 'ator', 'cyclo', 'di', 'deoxi','fluo', 'hydroxy', 'iso', 'ibupo', 'met', 'fluoro', 'omeo', 'pantop', 'peni', 'pro', 'quin', 'rani', 'tetra', 'simv', 'trans', 'tri', 'thio', 'phenyl', 'ortho', 'neo', 'lipo']

externalLookupDrugs = {}

with open("DrugBank.txt", encoding="utf-8") as f:

for line in f.readlines():

l = line.strip().lower().split("|")

externalLookupDrugs[l[0]] = l[1]

with open("HSDB.txt", encoding="utf-8") as f:

for line in f.readlines():

l = line.strip().lower()

externalLookupDrugs[l] = "drug"

## --------- tokenize sentence -----------

## -- Tokenize sentence, returning tokens and span offsets

def is\_camelcase(text):

return bool(re.match(r'^[A-Za-z]+(?:[A-Z][a-z]\*)\*$', text))

def tokenize(txt):

offset = 0

tks = []

## word\_tokenize splits words, taking into account punctuations, numbers, etc.

for t in word\_tokenize(txt):

## keep track of the position where each token should appear, and

## store that information with the token

offset = txt.find(t, offset)

tks.append((t, offset, offset+len(t)-1))

offset += len(t)

## tks is a list of triples (word,start,end)

return tks

## --------- get tag -----------

## Find out whether given token is marked as part of an entity in the XML

def get\_tag(token, spans) :

(form,start,end) = token

for (spanS,spanE,spanT) in spans :

if start==spanS and end<=spanE : return "B-"+spanT

elif start>=spanS and end<=spanE : return "I-"+spanT

return "O"

## --------- Feature extractor -----------

## -- Extract features for each token in given sentence

def extract\_features(tokens) :

# for each token, generate list of features and add it to the result

result = []

for k in range(0,len(tokens)):

tokenFeatures = [];

t = tokens[k][0]

tokenFeatures.append("form=" + t)

tokenFeatures.append("suf3=" + t[-3:])

### added

tokenFeatures.append(f"length={len(t)}")

tokenFeatures.append("isUpper=True" if t.isupper() else "isUpper=False")

tokenFeatures.append("isLower=True" if t.islower() else "isLower=False")

tokenFeatures.append("lower="+t.lower())

tokenFeatures.append("containsDash=True" if re.search("-", t) else "containsDash=False")

tokenFeatures.append("containsNumber=True" if re.search(number\_patter, t) else "containsNumber=False")

# because group names tend to finish with 's' like opioids, tranquilizers, analgesics, etc

tokenFeatures.append("isPlural=True" if t.endswith('s') else "isPlural=False" )

tokenFeatures.append("hasPrefix=True" if any(t.startswith(prefix) for prefix in prefixes) else "hasPrefix=False")

tokenFeatures.append("isCamelcase=" + str(is\_camelcase(t)))

# check if it is in the file DrugBank.txt or in HSDB.txt

if t.lower() in externalLookupDrugs.keys():

is\_drug = "True" if externalLookupDrugs[t.lower()] == "drug" else "False"

is\_drug\_n = "True" if externalLookupDrugs[t.lower()] == "drug\_n" else "False"

is\_brand = "True" if externalLookupDrugs[t.lower()] == "brand" else "False"

is\_group = "True" if externalLookupDrugs[t.lower()] == "group" else "False"

else:

is\_drug = "False"

is\_drug\_n = "False"

is\_brand = "False"

is\_group = "False"

tokenFeatures.append("isDrug="+ is\_drug)

tokenFeatures.append("isDrugN="+ is\_drug\_n)

tokenFeatures.append("isBrand=" + is\_brand)

tokenFeatures.append("isGroup="+ is\_group)

# probar ir sacando largos y ver cual sirve mas

if t[-3:] in suf\_list3 or t[-4:] in suf\_list4 or t[-5:] in suf\_list5 or t[-6:] in suf\_list6:

tokenFeatures.append("containsCommonSuffix=True")

else:

tokenFeatures.append("containsCommonSuffix=False")

if k>0 :

tPrev = tokens[k-1][0]

# length de la previous word

tokenFeatures.append("lengthPrev=" + str(len(tPrev)))

if k > 1:

two\_t\_prev = tokens[k-2][0]

## add suffix of 2 previous words

#tokenFeatures.append("formPrev2=" + two\_t\_prev)

#tokenFeatures.append("suf3Prev2=" + two\_t\_prev[-3:])

tokenFeatures.append("formPrev=" + tPrev)

tokenFeatures.append("suf3Prev=" + tPrev[-3:])

else :

tokenFeatures.append("BoS")

if k<len(tokens)-1 :

tNext = tokens[k+1][0]

if k < len(tokens) - 2:

two\_t\_next = tokens[k+2][0]

## add suffix two next word

#tokenFeatures.append("formPrev2=" + two\_t\_next)

#tokenFeatures.append("suf3Prev2=" + two\_t\_next[-3:])

tokenFeatures.append("formNext="+tNext)

tokenFeatures.append("suf3Next="+tNext[-3:])

else:

tokenFeatures.append("EoS")

result.append(tokenFeatures)

return result

## --------- MAIN PROGRAM -----------

## -- Usage: baseline-NER.py target-dir

## -- Extracts Drug NE from all XML files in target-dir, and writes

## -- them in the output format requested by the evalution programs.

# directory with files to process

datadir = sys.argv[1]

# process each file in directory

for f in listdir(datadir) :

# parse XML file, obtaining a DOM tree

tree = parse(datadir+"/"+f)

# process each sentence in the file

sentences = tree.getElementsByTagName("sentence")

for s in sentences :

sid = s.attributes["id"].value # get sentence id

spans = []

stext = s.attributes["text"].value # get sentence text

entities = s.getElementsByTagName("entity")

for e in entities :

# for discontinuous entities, we only get the first span

# (will not work, but there are few of them)

(start,end) = e.attributes["charOffset"].value.split(";")[0].split("-")

typ = e.attributes["type"].value

spans.append((int(start),int(end),typ))

# convert the sentence to a list of tokens

tokens = tokenize(stext)

# extract sentence features

features = extract\_features(tokens)

# print features in format expected by crfsuite trainer

for i in range (0,len(tokens)) :

# see if the token is part of an entity

tag = get\_tag(tokens[i], spans)

print (sid, tokens[i][0], tokens[i][1], tokens[i][2], tag, "\t".join(features[i]), sep='\t')

# blank line to separate sentences

print()

## 

## Conclusions

The main objective of this project was to extract different features per word that will aid in the identification of Named Entities. In order to solve it, we created feature functions that facilitated the creation of distinctive features with a binary output (True or False).

Just by adding features that looked for special characters and for capitalization and camelcase the models were significantly enhanced.

Moreover, incorporating features that reference external resources improves the distinction between drugs and brands.

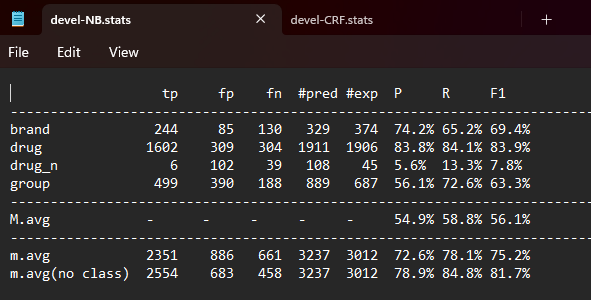
In summary, during the assignment we were able to try different features and see if they had a positive or negative impact and, based on this, try new ones or improve them as well as discard other ones.

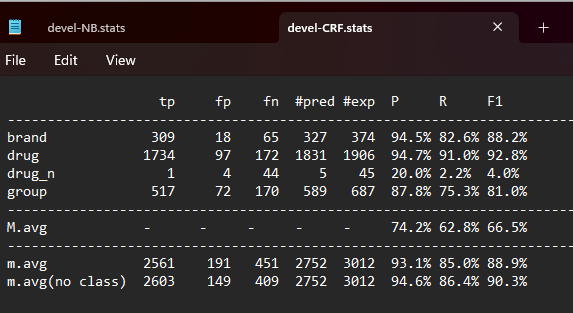
As future work, the tagging can be improved during preprocessing as the difference between drug and drug\_n is not very clear. We observed that in the XML files, entities tagged as drug\_n are also drug names, so we believe that this can have a negative impact in the model.

## Annex

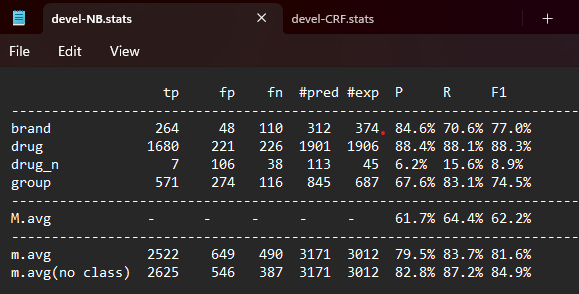
### First Model

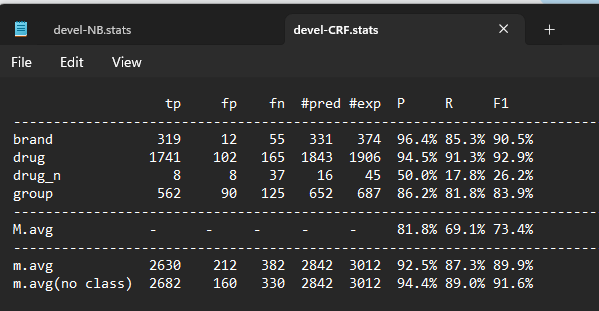
External sources Lookup



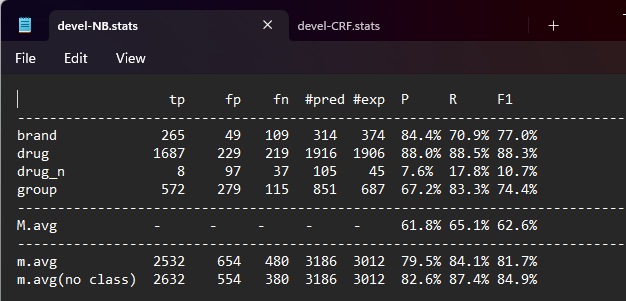


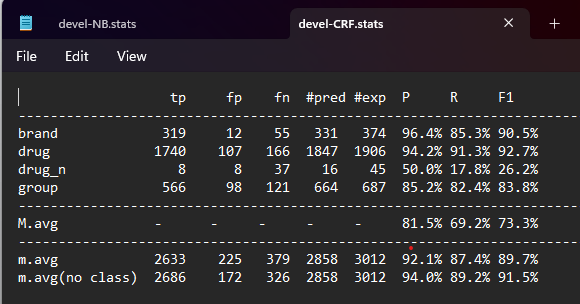
Capitalization - Camelcase - Plural



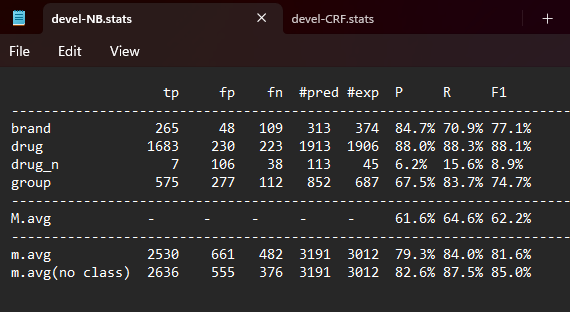


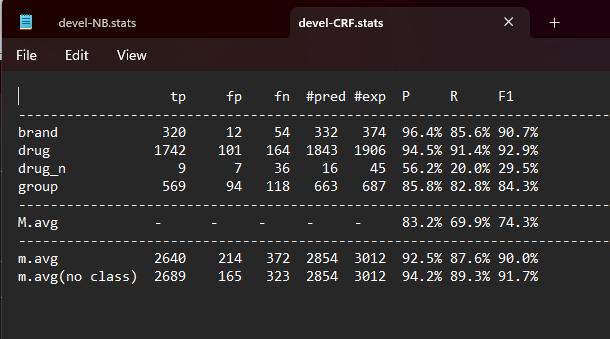
Suffixes and Prefixes





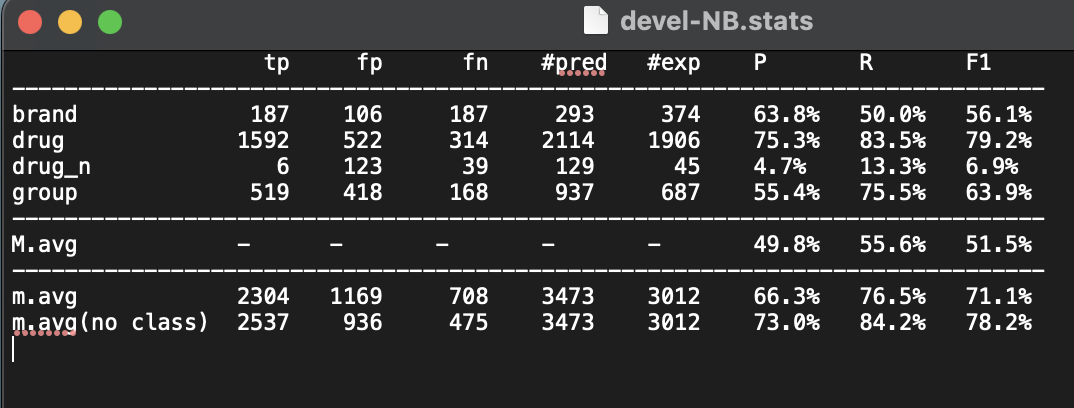
Numbers and Dash

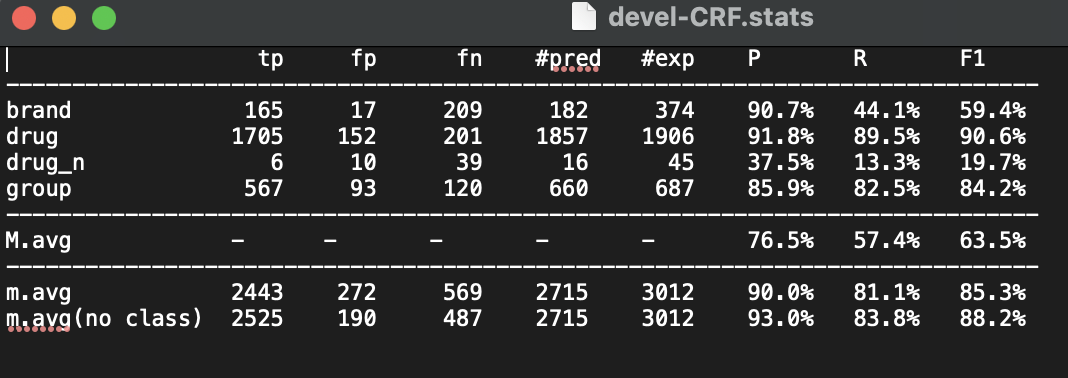




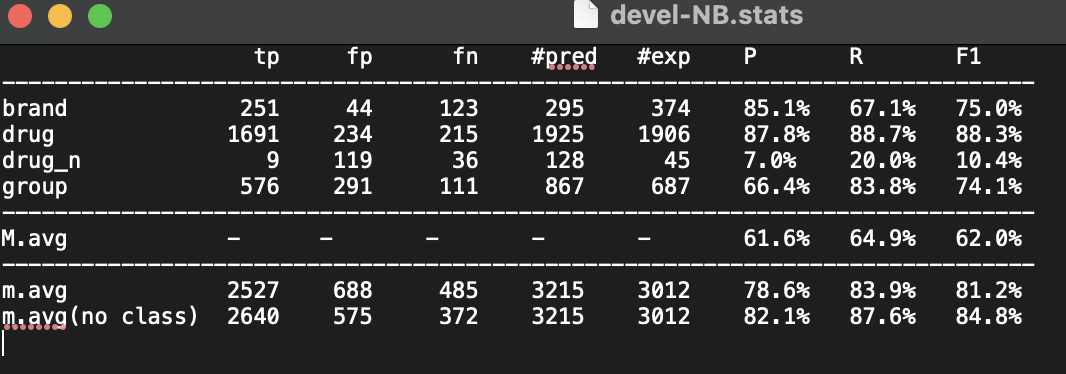
### Second Model

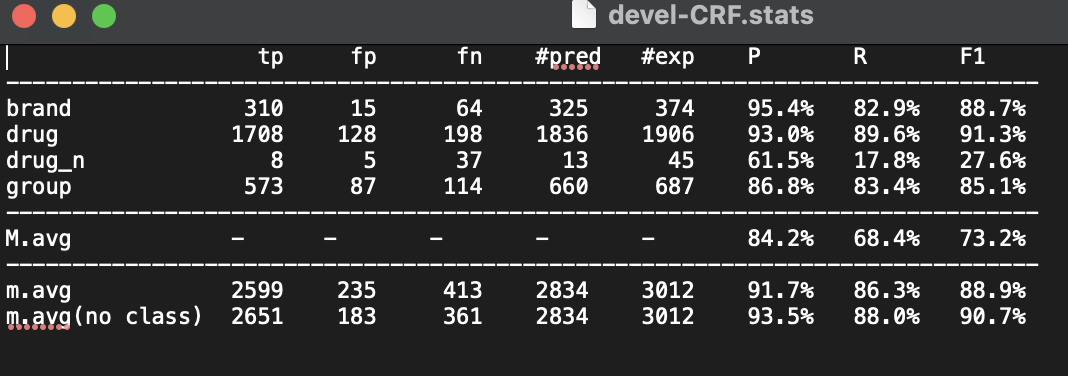
Adding the length of the word and the previous word



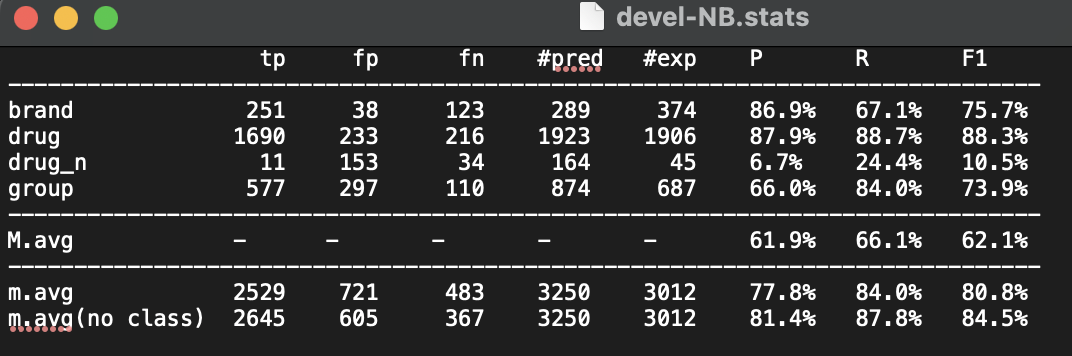


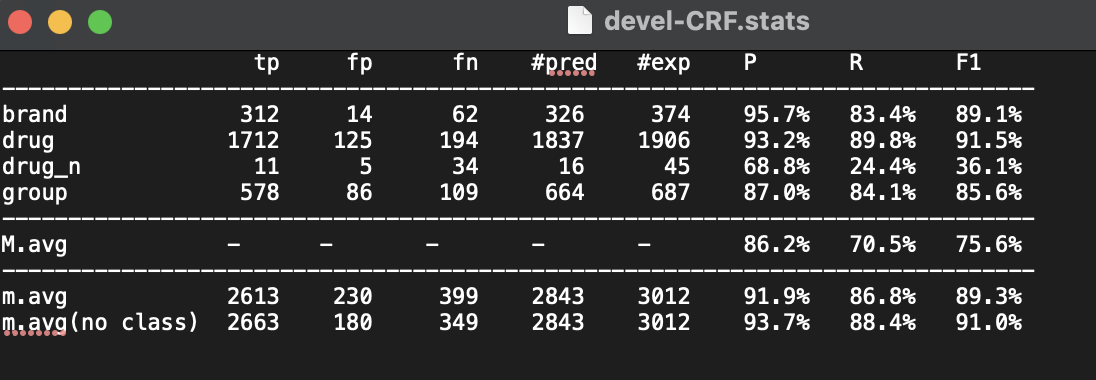
Uppercase, lowercase and camelcase



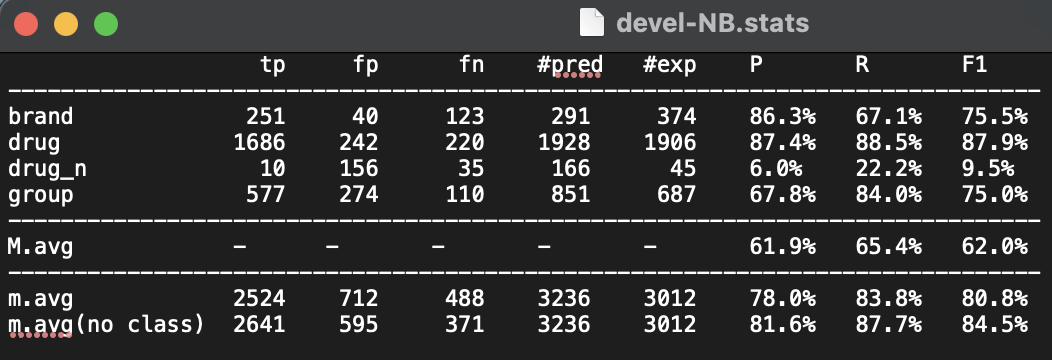


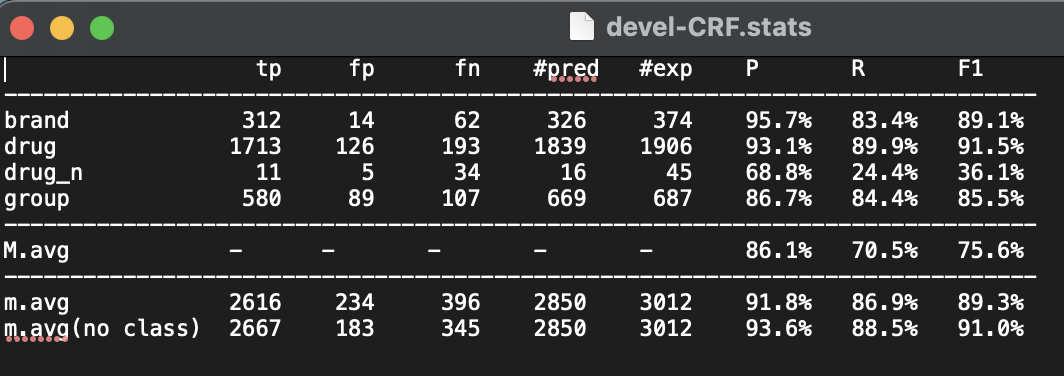
Special characters



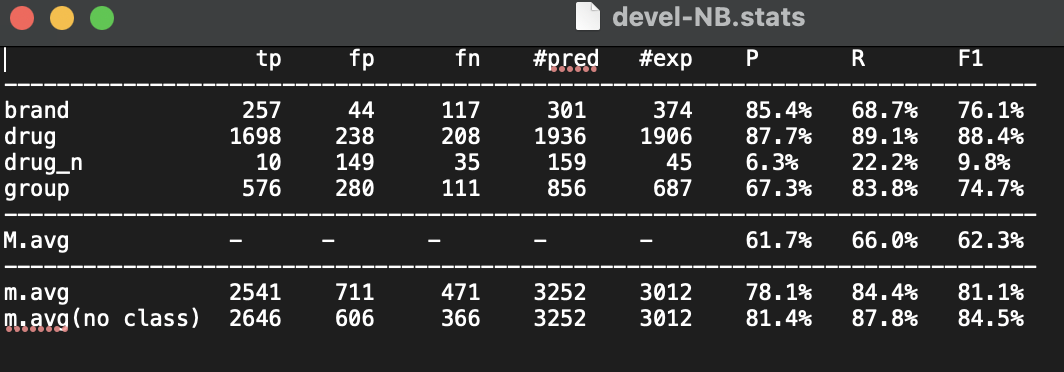


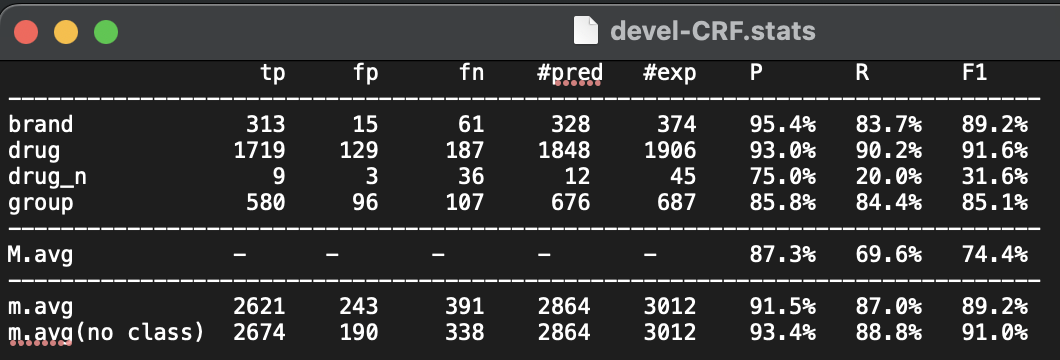
Plurals



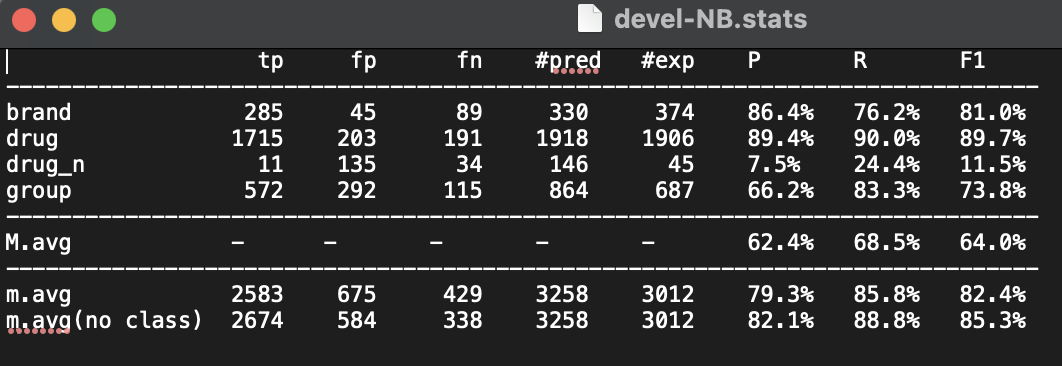


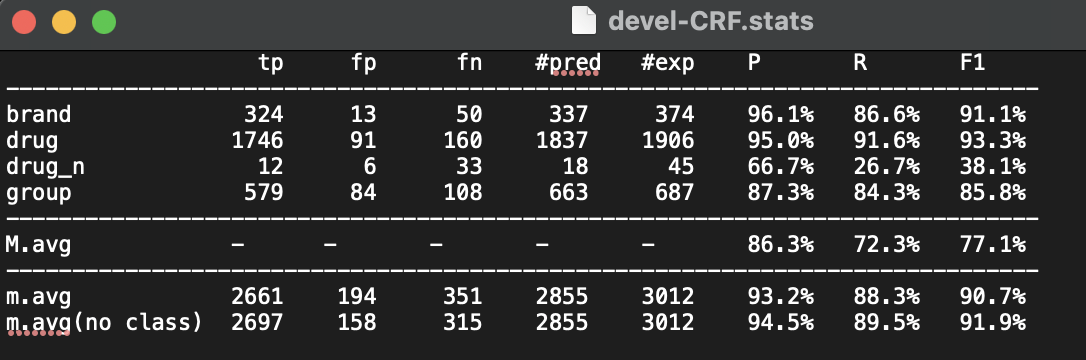
Prefixes and suffixes





Lookup











1. The results of the F1 score will be cumulatively improving as we are adding each feature in this order. Later, we will go through different orders that we tried and the results they have given us. [↑](#footnote-ref-0)
2. Parts of the code seem to be commented even though they are not. This is due to the code block’s syntax which interprets single quotes as comments in Python. [↑](#footnote-ref-1)