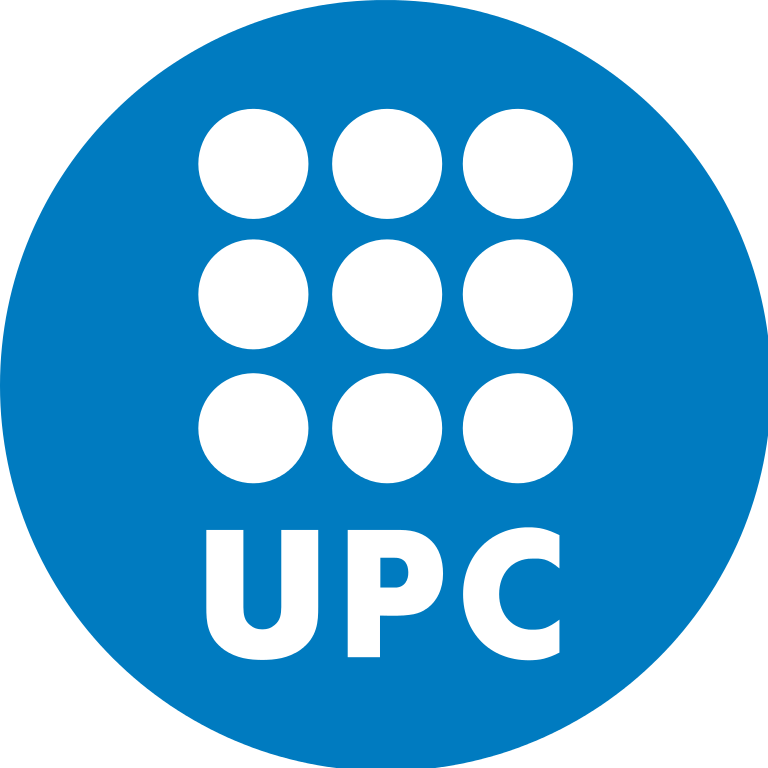
**UNIVERSITAT POLITÈCNICA DE CATALUNYA**

**BARCELONA SCHOOL OF INFORMATICS**

**Official Master on Data Science**

****

Lab 3: ML-based DDI

**Mining Unstructured Data**

**Albert Martín Garcia**

**Ximena Moure Alassio**

**Maria Paraskeva**

Barcelona, April 2023

[**Introduction 1**](#_g8uvynyzl3dg)

[**Models 1**](#_nsvok8keok6s)

[**Feature extraction 2**](#_v5w9jr49oo19)

[**Results 3**](#_a2f694zczfjr)

[Original 3](#_xr2ehc87ni0w)

[Words in vicinity 3](#_4i86yzialom3)

[Relative position 4](#_4yxakh2w70pf)

[Simpler path relationships 4](#_akuhlltq50ak)

[Extended syntax path features 5](#_arixo35qgs43)

[Testing the final models 6](#_lytntlnst2kp)

[**Code 6**](#_18w3pws4hfnj)

[**Conclusions 12**](#_50s7p6yzj13c)

# Introduction

This is the report for the third lab task, where the objective was to develop a Machine Learning model able to predict the relationship between two given entities in a sentence. The main task to perform was to implement the code to extract the most relevant features that would allow the model to perform such predictions, while the secondary task was to also try multiple models and compare their performances on the chosen features.

The report is split into three sections, the first summarizing the models used to perform the classification, detailing the reasoning behind trying it, the experiments performed, and providing the rationale behind our final decision of either using or disregarding its use.

# Models

Initially, we listed all models we thought would deserve some consideration for the task, and debated whether or not they would fit our needs before performing any kind of practical analysis. They are listed next, highlighted in bold font if we used them experimentally, followed by the reasoning behind our decision:

* **Naive Bayes Classifier:** While the precondition of non-correlation of features would surely not apply for our features, as common sequences of words would generate correlated variables, we believe its computational efficiency and its good behavior on high-dimensional spaces would make it perform well for our problem.
* **Support Vector Machine:** SVM is computationally expensive for non-linear Kernels, and we are not confident in the linear separability of the features we generate, as there may be different kinds of *effect* relationships, leading to separate clusters of the same class. Still, the library has a special implementation for the linear kernel optimized for sparse inputs, so it should be feasible to use if we end up generating large amounts of features.
* Decision Tree/ Random Forest: Given the amount of features we expect to generate, we believe this model would not be useful, as the decision splits on single variables would not make much sense unless the tree is deep enough to make multiple relevant decisions like “verb=consider”, “lem=not”, but this model is extremely prone to overfitting, making us think that most of the decision will be simply consequence of the particular word choices of the train dataset, instead of actual meaningful decisions about the relationship.
* KNN: The amount of features simply makes using this model unfeasible.
* K-Means: As explained before, we do not expect classes to belong to well-separated clusters, and while using several clusters per class may allow us to detect a higher proportion of them, we believe the high amount of features would not let this value even come close to the truth, and probably would not even converge, meaning SVM would always be better.

In the end, Naive Bayes ended up performing the best.

# Feature extraction

As the model works on only binary features, we have to ensure the presence or absence of a variable adds some useful information for the model, which means numerical variables will be split in buckets of similar meaning. Detailed below are all of the inferred features, with a small description of the reason we decided to include them when relevant.

The first kind of features we extracted were about the individual words present in the sentence, paying special attention to the given entities. In general, each word in the vicinity[[1]](#footnote-0) of the entities adds its lemma, word and lemma+rel, as we believe some of them may serve as qualifiers regardless of their position.

To encode the position as a variable, we have determined 6 classes, them being words before the first entity, words in between closest to the fist, in between closest to the second and words after the second, in addition to the two entities.

Ideally we would have the type of the two entities, but as the framework does not pass it into the function, we assumed the information was not made available to us on purpose. Thus, we added for each of the entities all the features that we used in the previous lab, as they should allow the model to discern the type of each entity, helping the decision. However this lead to a drastic decrease in the model’s accuracy, which lead us to believe that the word alone is enough for the model to learn the required information to know the class[[2]](#footnote-1), making the rest of the features redundant and even detrimental.

Next, we decided to add features extracted from the paths from the entities to the lowest common ancestor of them. For each path, we insert as a feature the concatenation of the node relationships, in addition to each lemma+POS and relationship found on the way as an individual feature, as we believe some of them, like “not” may be relevant to the decision-making. The length of the path is also added, encoded in buckets like all numerical features.

Sometimes the only important part of the path is the syntactic function of the block containing each entity, or perhaps the one of the second element in the path from root to the element. Thus, we generated features for subpaths from elements to the root (or lcs) of length 2, beginning at one of the elements or the root. Long paths would generate infrequent features, so we expect those that have constant size will help increase the performance for entities far apart.

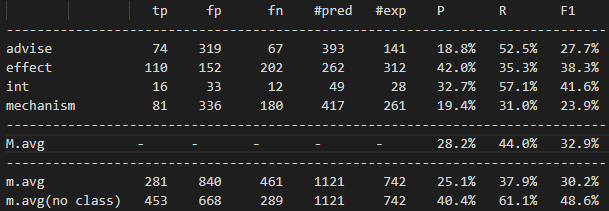
# Results

This section summarizes the results obtained after adding different subsets of the features to our vectors. It follows an incremental approach that allows us to see the effect of adding each of the features relative to having only the previous features, easily allowing us to determine the improvement. While ideally we would also show the effect of each subset relative to the original execution, this would simply be too lengthy and would not guarantee that there is no overlapping between the cases whose predictability is being improved at each step.

## Original

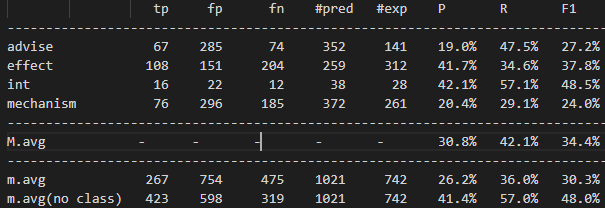
Following are shown the results of our initial execution for reference, having the word, lemma and tag of the word next to the first entity, with an additional feature indicating whether an entity is present in the middle and the path features showcasing the lemmas and relationships of the nodes between both entities and the lowest common ancestor.

They allow the model to obtain a macro and micro average of 32.9% and 48.5% respectively:



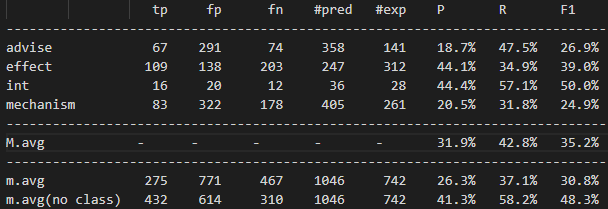
## Words in vicinity

Next we added the features for the words in the vicinity of the elements, by simply uncommenting the commented loop in the given file and changing the iteration limits from the element positions to the bounds of the lowest common ancestor of the elements.



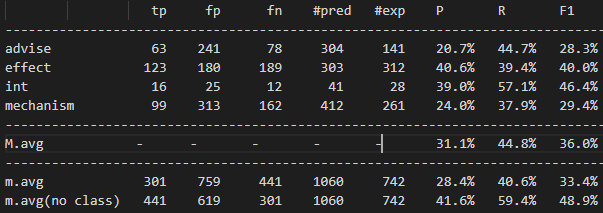
## Relative position

Next we added a feature that encodes the position of the word relative to the entities, which can have 6 values: l,1,m1,m2,2,r.



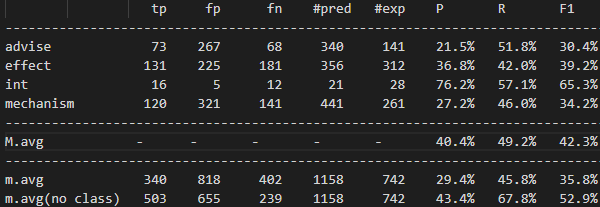
## Simpler path relationships

Next, we simply modified the preexisting path features to be much simpler, only encoding information about the relationship of the node and omitting the one belonging to the root, as it does not matter the role of the segment on the overall sentence, just the relationship between entities. This makes these features much more common, allowing the model to better learn their meanings.

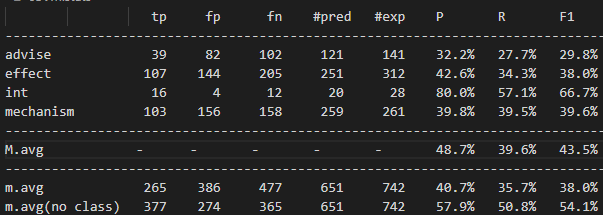


## Extended syntax path features

Finally, we added the rest of the features from the paths to the root or lowest common ancestor, them being the total length, the individual relationships and lemmas in the path, and the short subpaths from and to the entities of lengths 1 and 2.

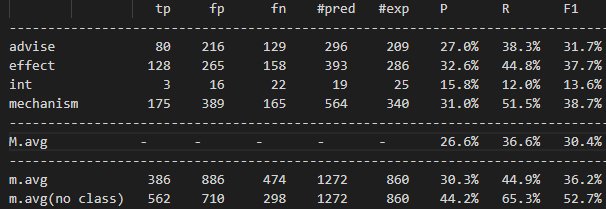


While we trained each individual subset of features with both chosen models, the results were quite close in all cases, so we decided to only show the intermediate results of the Naive Bayes Classifier. Still, the final results had a small but significant difference in performance, so we show below the statistics of the SVM classifier:



## Testing the final models

Unfortunately, the results obtained on the test dataset are much worse than those obtained on the devel. In this case, Naive Bayes performs slightly better, obtaining an average F1 slightly above 30%:



# Code

The code of the function to extract the features has been embedded below, properly indented and formatted, ready to be copied and pasted and executed if need be. The order of the generated features in the function is the same as in the description provided in the corresponding section, so intermediate results may be retrieved by commenting subsections of the code intuitively. The only exception are the features generated by the original version, which have been mostly removed or modified.

def extract\_features(tree, entities, e1, e2) :

feats = set()

# get head token for each gold entity

tkE1 = tree.get\_fragment\_head(entities[e1]['start'],entities[e1]['end'])

tkE2 = tree.get\_fragment\_head(entities[e2]['start'],entities[e2]['end'])

if tkE1 is not None and tkE2 is not None:

lcs = tree.get\_LCS(tkE1,tkE2)

vicBeg,vicEng=tree.get\_offset\_span(lcs)

# features for tokens in vicinity of E1 and E2

for tk in range(vicBeg, vicEng) :

tk=tkE1+1

try:

while (tree.is\_stopword(tk)):

tk += 1

except:

return set()

word = tree.get\_word(tk)

lemma = tree.get\_lemma(tk).lower()

tag = tree.get\_tag(tk)

rel=tree.get\_rel(tk)

feats.add("lib=" + lemma)

feats.add("wib=" + word)

feats.add("rel=" + rel)

feats.add("lpib=" + lemma + "\_" + tag)

#Position embedding

posStr=""

if tk<tkE1:

posStr="l"

elif tk==tkE1:

posStr="1"

elif 2\*tk<tkE2+tkE1:

posStr="m1"

elif tk<tkE2:

posStr="m2"

elif tk==tkE2:

posStr="2"

else:

posStr="r"

feats.add("pos="+posStr)

#Features for the two entities

for i,tk in enumerate([tkE1,tkE2]):

word = tree.get\_word(tk)

lemma = tree.get\_lemma(tk).lower()

tag = tree.get\_tag(tk)

rel=tree.get\_rel(tk)

feats.add("Elib=" + lemma)

feats.add("Ewib=" + word)

feats.add("Erel=" + rel)

feats.add("Elpib=" + lemma + "\_" + tag)

#Original entity presence feature

eib = False

for tk in range(tkE1+1, tkE2) :

if tree.is\_entity(tk, entities):

eib = True

feats.add('eib='+ str(eib))

#Path beween entities

path1 = tree.get\_up\_path(tkE1,lcs)

path1Str = "<".join([tree.get\_rel(x) for x in path1])

feats.add("path1="+path1Str)

path2 = tree.get\_down\_path(lcs,tkE2)

path2Str = ">".join([tree.get\_rel(x) for x in path2])

feats.add("path2="+path2Str)

path = path1Str+"<"+">"+path2Str

feats.add("path="+path)

#Extended path features

pathLength=len(path1)+len(path2)+1

pathLengthBins=[1,2,3,4,5,7,9,11]

feats.add("pathLength="+str(

[i for i,v in enumerate(pathLengthBins) if v<=pathLength][-1]

))

for i in path1+[lcs]+path2:

word = tree.get\_word(i)

lemma = tree.get\_lemma(i).lower()

tag = tree.get\_tag(i)

rel=tree.get\_rel(i)

feats.add("pathlib=" + lemma)

feats.add("pathrel=" + rel)

feats.add("pathlpib=" + lemma + "\_" + tag)

for i in range(min(2,len(path1))):

feats.add("subPathUp1"+str(i)+"="+"<".join(

tree.get\_rel(x) for x in path1[:i+1]

))

for i in range(min(2,len(path1))):

feats.add("subPathDown1"+str(i)+"="+"<".join(

tree.get\_rel(x) for x in path1[-i-1:]

))

for i in range(min(2,len(path2))):

feats.add("subPathDown2"+str(i)+"="+">".join(

tree.get\_rel(x) for x in path2[:i+1]

))

for i in range(min(2,len(path2))):

feats.add("subPathUp2="+str(i)+""+">".join(

tree.get\_rel(x) for x in path2[-i-1:]

))

return feats



# Conclusions

While improvement of our generated features was not as high as we would have hoped for, being slightly above a relative increment of 30% on the devel dataset, we are satisfied with the results we have obtained. We believe that this project has helped us understand how to exploit the given textual data to the maximum and has given us intuition about what kind of methods work best when it comes to extracting relevant information that allows us to infer the relationship between words.

Furthermore, while that is not the intended takeaway from the project, this lab has given us a new perspective to text analysis, particularly for relationship recognition. Before, if we had been asked to develop a program to detect this kind of relationships, we would have probably come up with some overly complicated hard-coded algorithm to analyze the text, but now we understand that Machine Learning techniques are much more versatile than we knew. In the future, not only in NLP tasks, we will also consider developing Machine Learning solutions in problems where finding an algorithmic solution is harder than reformulating the task as a classification problem and implementing feature extraction and modeling to achieve our goal.

1. We consider words to be in the vicinity of the entities if they belong in the subtree rooted in the lowest common ancestor of them. [↑](#footnote-ref-0)
2. As there is no need to detect them anymore, the number of words that will generate that feature is much less, so the model is perhaps directly associating the word with the class. [↑](#footnote-ref-1)