Data Mining - HW 3

Deep Learning and Natural Language Processing

Simone Marretta 1911358 - Luca Scofano 1762509

1 Task 1 - Use LAMA to get predictions for masked entities

Once we've preprocessed the initial dataset we perform the substasks, the idea behind each algorithm is almost the same.

Since the module *evaluation metrics* provided to us by LAMA return a sort of ranking (ordered by log-probability) we can use that to asses whether the prediction is on point or not. We'll briefly describe each algorithm used in each subtask. **Quick remark:** for each algorithm we've used the *lower()* function to convert all strings (predictions and original masked entities) in lower case ones.

1.1 As the professor said, this is the basic heuristic, and once we have the *ranking*, we compare the **first prediction** to the original masked entity. If the claim's label is equal to ***Support*** and the **first prediction** is equal to original masked entity, then we return **1**, if it's not then we return **0**. Vice versa for claims that have ***Refutes***, if the prediction is not equal to the original entity then we return **1**, a **0** otherwise.

1.2 We applied the same concept expressed in the previous subtask, but first we created a list with **top 10 predictions** and compare this list with the original masked entity.

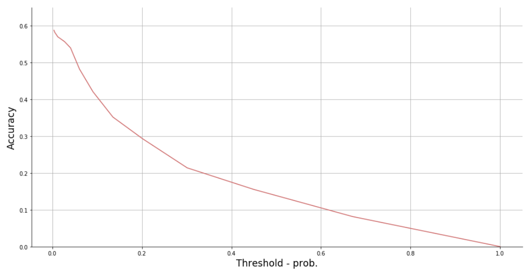
1.3 The process on how to compare the predictions and the original masked entity is the same, but how do we get the predictions? First of all, for each item in the JSON file we range through values that go from -5.6 to 0 (in the code we've used positive values, and absolute values for the log-probability, but the concept remains the same). For each value (that represents a *threshold*) we create a list where **threshold > abs(log-prob)** containing the predictions that we should use to compare with the original masked entity(like we did in subtask 1.2)

Since at the end we have lists of **0's and 1's**, to compute the *Accuracy* we just need to sum up all the 1's and divide it by the length of the list(number of items in the JSON file).

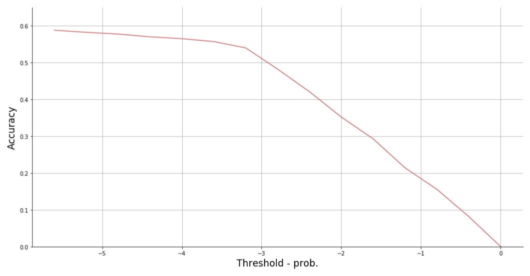
Results:

Task 1.1	Task 1.2
0.56971	0.588732

Threshold	Task 1.3
5.6	0.58914



(a) Probability



(b) Log-probability

Figure 1: Results for subtask 1.3

As we can imagine the bigger the log-probability is (the closer it is to 0) and the less chance we have to get the right prediction. The **highest** accuracy we get is by using **5.6** as the threshold, but the higher that goes and the higher the accuracy is.

2 Task 2 - Train a classifier on MASK and gold representations

For the Task 2 we decided to run the classifier on top of the representation of the CLS token of the last layer.

We tried also to run the classifier on top of the concatenation of the two vectors for each sentence(with unmasked/masked entities). At the end we performed the tuning using the CLS tokens because we observed a slightly better performance in terms of accuracy of the machine learning algorithms. What did we do?

We created numpy arrays for the x and y of the **train** dataset, the **dev** dataset and also the x of the official **test** dataset. For the binary classification we decided to use the Support Vector Machine for Classification implemented in the Scikit learn library, we picked this algorithm after running several classification algorithms and evaluating the accuracy and the Log-Loss for each of them.

We then ran a Grid Search algorithm on the train dataset with the purpose of tuning the parameters. We used the GridSearchCV algorithm implemented in the Scikit learn library using a Stratified 5-Fold cross validation. We preferred to tune our parameters on the **train** set rather than the **dev** set due to the number of samples of the train set. After doing that,we fitted our algorithm on the train dataset and then we ran it on the dev set. We performed the same thing on the **test** dataset and produced the predictions that we saved on a JSON file. The end **results** we got were:

Algorithm	**Task 2**
SVM	0.698745