Text Analytics: 2nd Assignment

Tsirmpas Dimitris
Drouzas Vasilis

February 13, 2024

Athens University of Economics and Business
MSc in Data Science

Contents

1 Introduction

This report will briefly discuss the theoretical background, implementation details and decisions taken for the construction of MLP models for sentiment analysis and POS tagging tasks.

This report and its associated code, analysis and results were conducted by the two authors. Specifically, the sentiment analysis task was performed by Drouzas Vasilis, and the POS-tagging task by Tsirmpas Dimitris. This report was written by both authors.

Note that due to the relative custom code complexity, most of the code used in this section was developed and imported from python source files located in the ‘tasks‘ module. In-depth documentation and implementation details can be found in these files.

2 POS Tagging

POS tagging is a language processing task where words in a given text are assigned specific grammatical categories, such as nouns, verbs, or adjectives. The objective is to analyze sentence structure.

In this section we describe how we can leverage pre-trained word embeddings to create a context-aware MLP classifier.

2.1 Dataset

Acquiring and preprocessing our data with the goal of eventually acquiring a sufficient representation of our text is the most difficult and time-consuming task. We thus split it in distinct phases:

- Original dataset acquisition and parsing
- Qualitative analysis and preprocessing
- Transformation necessary for the NLP task

Each of these distinct steps are individually analyzed below.

2.1.1 Acquisition

We select the English EWT-UD tree, which is the largest currently supported collection for POS tagging tasks for the English language.

This corpus contains 16622 sentences, 251492 tokens and 254820 syntactic words, as well as 926 types of words that contain both letters and punctuation, such as ’s, n’t, e-mail, Mr., ’s, etc). This is markedly a much higher occurrence than its siblings, and therefore may lead to a slightly more difficult task.

The dataset is made available in `conllu` format, which we parse using the recommended `conllu` python library. We create a dataframe for every word and its corresponding POS tag and link words belonging to the same sentences by a unique sentence ID. The data are already split to training, validation and test sets, thus our own sets correspond to the respective split files.

We are interested in the UPOS (Universal Part of Speech) tags for English words.

2.1.2 Qualitative Analysis

Our training vocabulary is comprised of 16654 words. We include qualitative statistics on the sentences of our dataset in Tables 1 and 2. The splits are explicitly mentioned separately because the splitting was performed by the dataset authors and not by random sampling. We would therefore like to confirm at a glance whether their data are similar.

Set	Mean	Std	Min	25%	50%	75%	Max
Training	16.31	12.4	1	7	14	23	159
Validation	12.56	10.41	1	5	10	17	75
Test	12.08	10.6	1	4	9	17	81

Table 1: Summary and order statistics for the number of words in the sentences of each data split.

Set	Total Word Count	Total Sentence Count
Training	204614	12544
Validation	25152	2001
Test	25096	2077

Table 2: Total text volume of each data split.

2.1.3 Preprocessing

Given the nature of our task we can not implement preprocessing steps such as removing punctuation marks, stopwords or augmenting the dataset. Thus, the only meaningful preprocessing at this stage would be converting the words to lowercase. We believe that the context of each word will carry enough information to distinguish its POS tag regardless of case.

Another issue we need to address before continuing is that of words being part of (depending on) other words for their POS tag. Those would be words such as "don't", "couldn't" or "you're". In the standard UPOS schema these are defined as two or more separate words, where the first is represented by its standard POS tag, and the rest as part of that tag (UPOS tag "PART"). For instance, "don't" would be split into "do" and "n't" with "AUX" and "PART" tags respectively. In our dataset, these words are represented both in the manner described above followed by the full word ("don't") tagged with the pseudo-tag "_". We remove the latter representation from the working dataset.

For the word embeddings we originally used a Word2Vec variant implemented in the `spacy` library called `en_core_web_md`. The model seemed suitable for our needs because of the similarities in domain (pre-trained on blogs, news and comments which fits our dataset). However, it proved extremely slow and thus constrained the amount of embeddings we could reasonably procure, limiting our classifier.

Thus we use the fasttext `cc.en.300` model. This model has a total size of 7GB which may present OOM issues in some machines, but calculates embeddings extremely fast, allowing us to augment our training set from 65,000 to 150,000 windows.

The general algorithm to calculate the window embeddings on our dataset can be found in Algorithm 1. The algorithm uses a few external functions which are not described here for the sake of brevity. `get_window()` returns the context of the word inside a sentence, including padding where needed, `embedding()` returns the word embedding for a single word and `concatenate` returns a single element from a list of elements. The rest of the functions should be self-explanatory. Note that this algorithm does not represent the actual python implementation.

2.2 Baseline Classifier

We create our own classifier which classifies each token by the majority label associated with it. The classifier is defined as a subclass of sklearn's classifier superclass and thus can seamlessly use it in most sklearn-provided functions such as `classification_report()` and its implementation can be found in the `tasks.models` module.

The results of the classifier can be found in Table 3, 4 and 5. The results make intuitive sense, since most words in the English language can be classified in a single label, irrespective of context.

Algorithm 1 Window Embedding creation algorithm from raw-text sentences.

Input sentences: a list of sentences

Output tuple(windows, targets): the window embeddings and the POS tag corresponding to the median word of each window

```
windows = list()
targets = list()

for sentence in sentences do
    for word in sentence do
        window = get_window(word, sentence)
        target = get_tag(word)
        windows.add(window)
        targets.add(target)
    end for
end for

window_embeddings = list()
for window in windows do
    word_embeddings = list()
    for word in window do
        if word is PAD_TOKEN then
            word_embeddings.add(zeros(embedding_size))
        else
            word_embeddings.add(embedding(word))
        end if
    end for
    window_embedding = concatenate(word_embeddings)
    window_embeddings.add(window_embedding)
end for

return window_embeddings, one_hot(targets)
```

For example, ”is” will always be classified as ”AUX”, and all punctuation marks will be classified as ”PUNCT”.

Thus, besides quantitative statistics such as categorical accuracy and f1-score, we should pay close attention to the precision and recall statistics for the more variable POS tags such as ”NOUN” or ”VERB” in order to properly evaluate our MLP classifier.

2.3 MLP Classifier

2.3.1 Hyper-parameter tuning

We use the `keras_tuner` library to automatically perform random search over various hyper-parameters of our model.

The parameter search consists of:

- The depth of the model (the number of layers)
- The height of the model (the number of parameters by layer)
- The learning rate

The parameter search does NOT consist of:

- Dropout rate, since dropout rarely changes the final result of a neural network, but rather tunes the trade-off between training time and overfit avoidance
- Activation functions, since they rarely significantly influence the model's performance

With this scheme we hope to maximize the area and granularity of our search to the hyper-parameters that are most likely to significantly influence the final results.

We implement early stopping and set a maximum iteration limit of 70. We assume that if a model needs to go over that limit, it may be computationally inefficient, and thus less desirable compared to a slightly worse, but much more efficient model. Additionally, we use a relatively large batch size to improve training times since this operation is very computationally heavy. We don not yet aim to create the best classifier, so slightly suboptimal weights are not a problem for the purposes of the hyperparameter search. We use a relatively very large batch size to improve training times since this operation is very computationally heavy. We don't yet aim to create the best classifier, so slightly suboptimal weights are not a problem for the purposes of the hyperparameter search.

2.3.2 Training

We now re-train our model with a much smaller batch size and keep track of the training history and best weights by validation loss.

Unfortunately, the different batch size means we can not rely on the hyper parameter search to get an estimation of training epochs. Thus, we rely on early-stopping on the validation data to ensure our model does not overfit as a result of training time.

We use the categorical accuracy stopping criterion instead of loss. This may lead to situations where validation loss increases, but so does accuracy [1]. This represents a trade-off between our model being more confidently incorrect about already-misclassified instances, but better at edge cases where the classification is more ambiguous. We previously discussed how the strength of a context-aware classifier lies in these kinds of distinctions, which justifies our choice of favoring correct edge-case classifications in the expense of more confidently incorrect misclassifications.

This phenomenon is demonstrated in Figure 1.

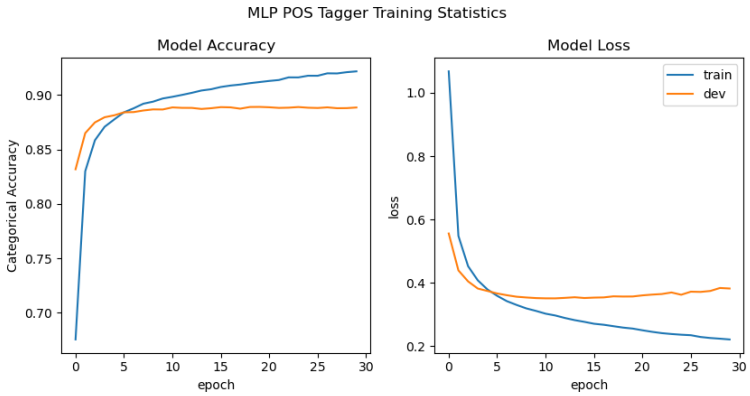


Figure 1: Loss and accuracy on the training and validation sets depending on the number of epochs.

2.3.3 Results

The results of our MLP classifier compared to the baseline models mentioned above can be found in Tables 3, 4 and 5. We include precision, recall and F1 scores for each individual tag, as well as their macro average denoted by the "MACRO" tag in the tables. We **can not use PR-AUC scores**, since they are only defined for binary classification tasks.

We note an increase in all metrics for our MLP classifier, especially in tags such as "SCONJ" (subordinating conjunction), and VERB and NOUN (which we hypothesized at the beginning of the report). The only exception is the "X" tag (other) which is attributed to unintelligible material, foreign words and word fragments. This is an acceptable drawback, since these are easily caught by preprocessing or weak classifiers.

Another notable observation is the 100% precision in all categories but "X" both by our baseline and MLP classifiers. This indicates that precision is not a significant metric in our task.

3 Sentiment Analysis

Sentiment analysis, also known as opinion mining, is the process of analyzing text to determine the sentiment or emotional tone expressed within it. The goal of sentiment analysis is to understand the attitudes, opinions, and emotions conveyed by the text.

3.1 Dataset

Here we will be working with the Cornell Movie Review dataset, which consists of 2000 movie reviews, split equally in 1000 positive and 1000 negative ones. The goal here will be to develop classifiers that will effectively understand whether a review is a positive or negative one, based on the data it has been trained on.We begin by taking a brief look into our dataset.

3.1.1 Average Document Length

The average document length in words and characters is:

- Average number of words: 746.3405
- Average number of characters: 3893.002

Table 3: Results on the training dataset.

model	tag	precision	recall	f1
Baseline	ADJ	1.000	0.880	0.936
Baseline	ADP	1.000	0.816	0.899
Baseline	ADV	1.000	0.776	0.874
Baseline	AUX	1.000	0.929	0.963
Baseline	CCONJ	1.000	0.995	0.997
Baseline	DET	1.000	0.923	0.960
Baseline	INTJ	1.000	0.773	0.872
Baseline	NOUN	1.000	0.903	0.949
Baseline	NUM	1.000	0.968	0.984
Baseline	PART	1.000	0.896	0.945
Baseline	PRON	1.000	0.983	0.991
Baseline	PROPN	1.000	0.871	0.931
Baseline	PUNCT	1.000	0.988	0.994
Baseline	SCONJ	1.000	0.476	0.645
Baseline	SYM	1.000	0.842	0.914
Baseline	VERB	1.000	0.854	0.921
Baseline	X	1.000	0.629	0.772
Baseline	MACRO	0.858	0.853	0.854
MLP	ADJ	1.000	0.934	0.966
MLP	ADP	1.000	0.898	0.946
MLP	ADV	1.000	0.866	0.928
MLP	AUX	1.000	0.964	0.982
MLP	CCONJ	1.000	0.998	0.999
MLP	DET	1.000	0.984	0.992
MLP	INTJ	1.000	0.778	0.875
MLP	NOUN	1.000	0.948	0.973
MLP	NUM	1.000	0.985	0.993
MLP	PART	1.000	0.987	0.993
MLP	PRON	1.000	0.965	0.982
MLP	PROPN	1.000	0.862	0.926
MLP	PUNCT	1.000	0.995	0.998
MLP	SCONJ	1.000	0.637	0.778
MLP	SYM	1.000	0.809	0.895
MLP	VERB	1.000	0.895	0.944
MLP	X	1.000	0.327	0.493
MLP	MACRO	0.923	0.873	0.887

Table 4: Results on the validation dataset.

model	tag	precision	recall	f1
Baseline	ADJ	1.000	0.800	0.889
Baseline	ADP	1.000	0.824	0.903
Baseline	ADV	1.000	0.739	0.850
Baseline	AUX	1.000	0.920	0.959
Baseline	CCONJ	1.000	0.992	0.996
Baseline	DET	1.000	0.917	0.957
Baseline	INTJ	1.000	0.598	0.748
Baseline	NOUN	1.000	0.891	0.943
Baseline	NUM	1.000	0.737	0.849
Baseline	PART	1.000	0.897	0.946
Baseline	PRON	1.000	0.986	0.993
Baseline	PROPN	1.000	0.461	0.631
Baseline	PUNCT	1.000	0.984	0.992
Baseline	SCONJ	1.000	0.484	0.652
Baseline	SYM	1.000	0.855	0.922
Baseline	VERB	1.000	0.763	0.865
Baseline	X	1.000	0.020	0.040
Baseline	MACRO	0.806	0.757	0.773
MLP	ADJ	1.000	0.873	0.932
MLP	ADP	1.000	0.887	0.940
MLP	ADV	1.000	0.763	0.865
MLP	AUX	1.000	0.933	0.965
MLP	CCONJ	1.000	0.981	0.990
MLP	DET	1.000	0.972	0.986
MLP	INTJ	1.000	0.747	0.855
MLP	NOUN	1.000	0.902	0.949
MLP	NUM	1.000	0.928	0.963
MLP	PART	1.000	0.975	0.987
MLP	PRON	1.000	0.937	0.967
MLP	PROPN	1.000	0.743	0.853
MLP	PUNCT	1.000	0.988	0.994
MLP	SCONJ	1.000	0.542	0.703
MLP	SYM	1.000	0.783	0.878
MLP	VERB	1.000	0.834	0.910
MLP	X	1.000	0.102	0.185
MLP	MACRO	0.847	0.817	0.824

Table 5: Results on the test dataset.

model	tag	precision	recall	f1
Baseline	ADJ	1.000	0.783	0.878
Baseline	ADP	1.000	0.820	0.901
Baseline	ADV	1.000	0.800	0.889
Baseline	AUX	1.000	0.918	0.957
Baseline	CCONJ	1.000	1.000	1.000
Baseline	DET	1.000	0.926	0.962
Baseline	INTJ	1.000	0.727	0.842
Baseline	NOUN	1.000	0.887	0.940
Baseline	NUM	1.000	0.651	0.788
Baseline	PART	1.000	0.915	0.956
Baseline	PRON	1.000	0.985	0.992
Baseline	PROPN	1.000	0.485	0.653
Baseline	PUNCT	1.000	0.983	0.991
Baseline	SCONJ	1.000	0.497	0.664
Baseline	SYM	1.000	0.870	0.930
Baseline	VERB	1.000	0.753	0.859
Baseline	X	1.000	0.030	0.059
Baseline	MACRO	0.811	0.766	0.779
MLP	ADJ	1.000	0.848	0.918
MLP	ADP	1.000	0.883	0.938
MLP	ADV	1.000	0.812	0.896
MLP	AUX	1.000	0.945	0.972
MLP	CCONJ	1.000	0.993	0.997
MLP	DET	1.000	0.972	0.986
MLP	INTJ	1.000	0.778	0.875
MLP	NOUN	1.000	0.899	0.947
MLP	NUM	1.000	0.926	0.962
MLP	PART	1.000	0.988	0.994
MLP	PRON	1.000	0.944	0.971
MLP	PROPN	1.000	0.710	0.830
MLP	PUNCT	1.000	0.983	0.991
MLP	SCONJ	1.000	0.523	0.687
MLP	SYM	1.000	0.711	0.831
MLP	VERB	1.000	0.842	0.915
MLP	X	1.000	0.030	0.059
MLP	MACRO	0.836	0.811	0.813

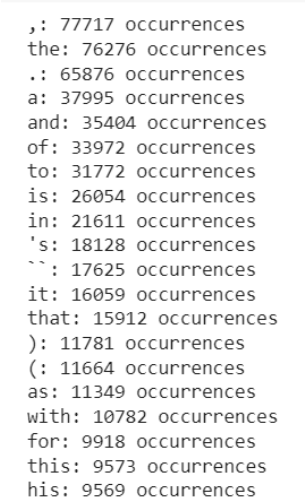


Figure 2: The 20 most common words in the text, along with their occurences.

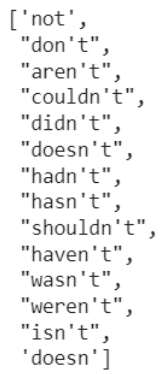


Figure 3: The 'important' words we decided to keep for this sentiment analysis problem.

3.1.2 Pre-processing

For demonstration reasons, we start by printing the 20 most frequent words in the text, in Figure 2.

Most of these words are actually stop words. As in most text classification problems, we would typically need to remove the stop words of the text.

The english stopwords is a package of 179 words that in general, would not help in a sentiment analysis problem. But, since they include terms that are negative, removing them could prove harmful for our case, since we are dealing with a sentiment analysis problem.

e.g. imagine the phrase "I didn't like the film" to end up "like film". Disastrous, right?

So, the plan is to remove all the stop words that include negative meaning before the preprocessing. The stop words that we decided to keep in the text are shown in Figure 3.

Moving on to the pre-processing task, the steps performed are the following:

- Combination to a single document.
- Convertion to lowercase.

Set	Total Word Count	Total Document Count
Training	36624	1400
Validation	16948	300
Test	16780	300

Table 6: Total text volume of each data split.

- Lemmatization and stop words extraction.
- Punctuation removal.
- Number removal.
- Single characters removal.
- Converting multiple spaces to single ones.

3.1.3 Splitting the dataset

We decided to split the (processed) dataset into the training set (70%), development set (15%) and test set (15%). The sizes of each set are shown in Table 6.

3.1.4 TF-IDF

We used the unigram and bi-gram TF-IDF features, defining the maximum number of features to 5000. The shapes of the data are as follows:

- Training data: (1400, 5000)
- Development data: (300, 5000)
- Test data: (300,5000)

3.1.5 Feature selection with SVD

We performed dimensionality reduction with the `TruncatedSVD()` method, reducing the number of features from 5000 to 500. The new shapes are:

- Training data: (1400, 500)
- Development data: (300, 500)
- Test data: (300, 500)

3.2 Classifiers

3.2.1 DummyClassifier

DummyClassifier makes predictions that ignore the input features. This classifier serves as a simple baseline to compare against other more complex classifiers.The strategy to generate predictions was set to 'most_frequent', meaning that the predict method always returns the most frequent class label. The results of this classifier are demonstrated in Figure 4.

As expected, the results are poor since the decision of the classifier depends exclusively only the majority class.

```
Classification Report on Development Set:
              precision    recall  f1-score   support

         neg       0.00      0.00      0.00       157
         pos       0.48      1.00      0.65       143

    accuracy                           0.48       300
   macro avg       0.24      0.50      0.32       300
weighted avg       0.23      0.48      0.31       300

-----------------------------------------------------
Classification Report on Training Set:
              precision    recall  f1-score   support

         neg       0.00      0.00      0.00       690
         pos       0.51      1.00      0.67       710

    accuracy                           0.51      1400
   macro avg       0.25      0.50      0.34      1400
weighted avg       0.26      0.51      0.34      1400

-----------------------------------------------------
Classification Report on Test Set:
              precision    recall  f1-score   support

         neg       0.00      0.00      0.00       153
         pos       0.49      1.00      0.66       147

    accuracy                           0.49       300
   macro avg       0.24      0.50      0.33       300
weighted avg       0.24      0.49      0.32       300
```

Figure 4: Classification results of DummyClassifier for training,test and validation sets.

3.2.2 Logistic Regression

Logistic Regression is a statistical method used for binary classification tasks, where the output variable takes only two possible outcomes. Before applying Logistic Regression, we will perform a grid search to find the optimal parameters to run the classifier. The parameters we tried are the following:

- Solver: We tested 'liblinear' and 'saga' solvers
- Penalty: We tested 'l1', 'l2', 'elasticnet' reguralization penalties
- C: We tested values of 0.001, 0.01, 0.1, 1 and 10 (inverse of regularization strength)

The best hyperparameters were the following: C= 10, penalty= 'l1', solver = 'liblinear'.

Now, it is time to fit the Logistic Regression using these parameters. The results we got are shown in Figure 5 .

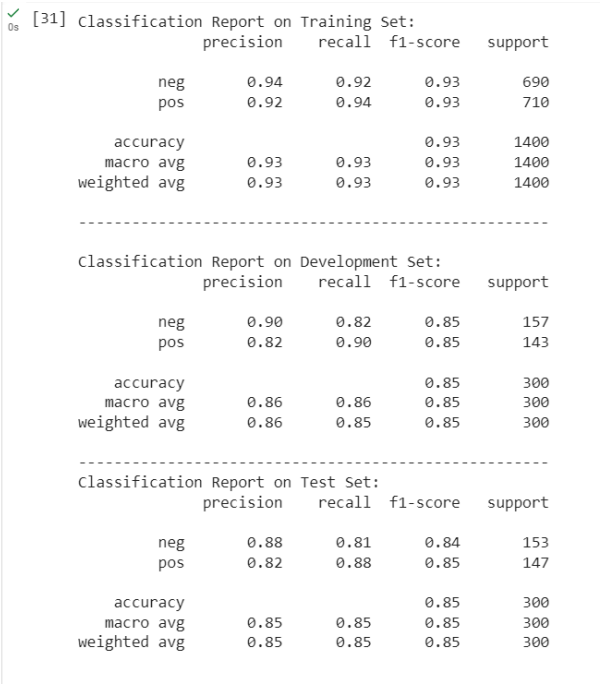
3.2.3 Our custom MLP classifier

First of all, we define the y_train_1_hot and y_dev_1_hot vectors using the LabelBinarizer and applying fit_transform() and transform() to the training and development 1-hot vectors respectively.

Now, it's time to define our MLP model. We used the SGD algorithm since for this case it provided better results than Adam. The number of epochs was set to 15. We experimented with a variety of different hyperparameter combinations (Table 7).

The process to decide the hyperparameters is simple: We defined a list of the possible hyperparameter combinations and for each one we ran the model. After that, we evaluated on the development set and we kept the model with the best development accuracy.

The optimal model consisted of the following hyperparameters:



```
[31] Classification Report on Training Set:
                precision    recall  f1-score   support

           neg       0.94      0.92      0.93       690
           pos       0.92      0.94      0.93       710

      accuracy                           0.93      1400
     macro avg       0.93      0.93      0.93      1400
  weighted avg       0.93      0.93      0.93      1400

  ----------------------------------------------------

  Classification Report on Development Set:
                precision    recall  f1-score   support

           neg       0.90      0.82      0.85       157
           pos       0.82      0.90      0.85       143

      accuracy                           0.85       300
     macro avg       0.86      0.86      0.85       300
  weighted avg       0.86      0.85      0.85       300

  ----------------------------------------------------
  Classification Report on Test Set:
                precision    recall  f1-score   support

           neg       0.88      0.81      0.84       153
           pos       0.82      0.88      0.85       147

      accuracy                           0.85       300
     macro avg       0.85      0.85      0.85       300
  weighted avg       0.85      0.85      0.85       300
```

Figure 5: Metrics of the Logistic Regression on the training, test and development sets.

Learning rate	#Hidden layers	Hidden layers size	Dropout probability	Batch size
0.001	1	64	0.4	1
0.01	2	128	0.5	64
0.1				128

Table 7: Hyperpameters tested in the development set.

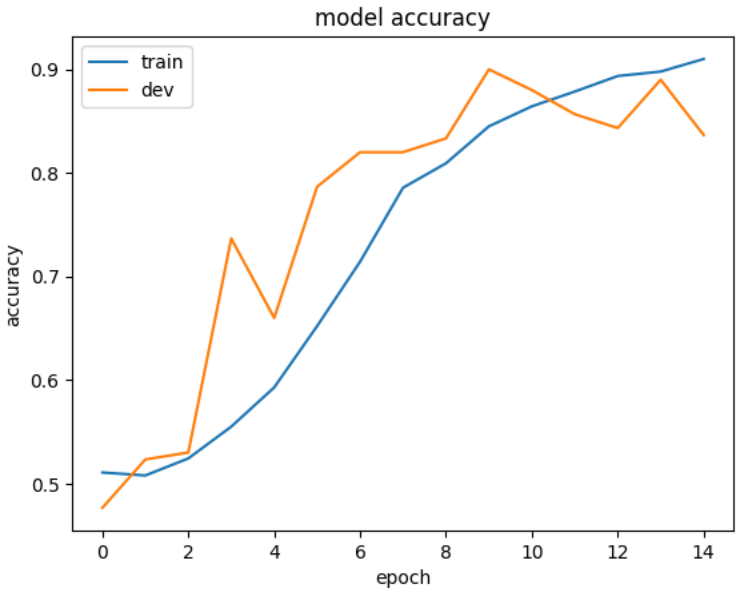


Figure 6: MLP accuracy as a function of epochs.

- Learning rate: 0.01
- Number of hidden layers: 2
- Hidden layers' size: 64
- Dropout probability: 0.4
- Batch size: 1

The results we gain are shown in Figures 6, 7.

Next, we provide the metrics (Precision,Recall, F1 score and the AUC scores) for training, development and test subsets in Figure 8.

Finally, the Macro-averaged metrics (averaging the corresponding scores of the previous bullet over the classes) for the training, development and test subsets, are shown in Figure 9.

References

[1] Soltius (https://stats.stackexchange.com/users/201218/soltius). *How is it possible that validation loss is increasing while validation accuracy is increasing as well*. Cross Validated. URL:https://stats.stackexchange.com/q/3 (version: 2023-03-28). eprint: https://stats.stackexchange.com/q/341054. URL: https://stats.stackexchange.com/q/341054.

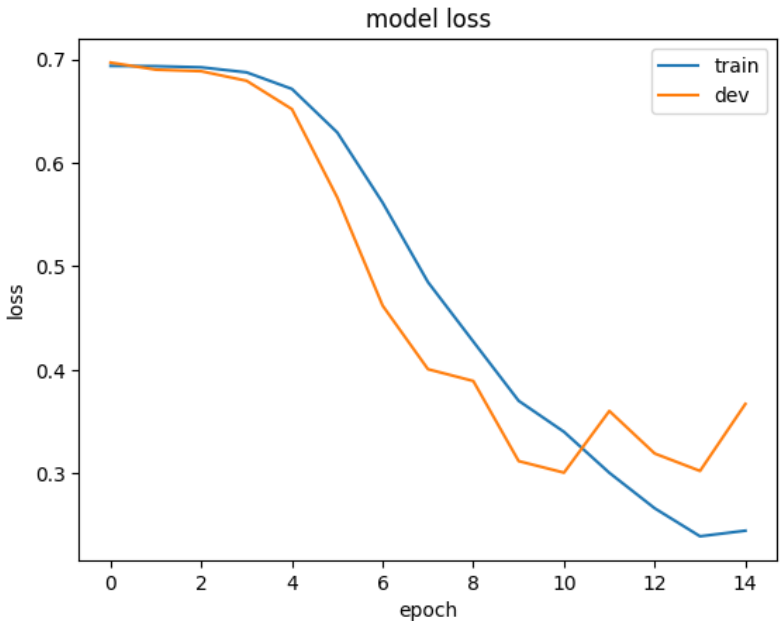


Figure 7: MLP loss as a function of epochs.

```
Class  neg :   (Training)     (Development)      (Test)
Precision    |  0.992308   |    0.915385    |   0.917293
Recall       |  0.934783   |    0.757962    |   0.797386
F1-score     |  0.962687   |    0.829268    |   0.853147
PR AUC       |  0.995746   |    0.944092    |   0.918444
------------------------------------------------------------
Class  pos :   (Training)     (Development)      (Test)
Precision    |  0.940000   |    0.776471    |   0.814371
Recall       |  0.992958   |    0.923077    |   0.925170
F1-score     |  0.965753   |    0.843450    |   0.866242
PR AUC       |  0.995746   |    0.944092    |   0.918444
------------------------------------------------------------
```

Figure 8: Metrics for the MLP classifier for both classes for the training, development and test sets.

```
Macro-averaged Scores for Training Subset:
==========================================
Macro-averaged Precision: 0.966154
Macro-averaged Recall: 0.963870
Macro-averaged F1-score: 0.964220
Macro-averaged PR AUC: 0.995746

Macro-averaged Scores for Development Subset:
==============================================
Macro-averaged Precision: 0.845928
Macro-averaged Recall: 0.840519
Macro-averaged F1-score: 0.836359
Macro-averaged PR AUC: 0.944092

Macro-averaged Scores for Test Subset:
======================================
Macro-averaged Precision: 0.865832
Macro-averaged Recall: 0.861278
Macro-averaged F1-score: 0.859694
Macro-averaged PR AUC: 0.918444
```

Figure 9: Macro-Metrics for the MLP classifier for both classes for the training, development and test sets.