Text Analytics: 3rd Assignment

Tsirmpas Dimitris
Drouzas Vasilis

February 18, 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, while also allowing partial modeling of Out Of Vocabulary (OOV) words. The model is used to calculate the embedding matrix which is later attached to the RNN model.

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 Tables 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. 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

The model we use is the pre-trained optimal model used in the previous assignment. We follow the same preprocessing and caching steps as in the previous assignment. Since the model is not trained again, we use only a subset of the original training data (25,000 windows) in order to save on scare main-memory resources. We consider this a representative sample for comparison with other classifiers due to the sample size (law of large numbers).

Results can be found in Tables 3, 4 and 5.

2.4 RNN Classifier

2.4.1 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 number of bidirectional layers

- Whether to use Layer Normalization or dropout

- Whether to use variational (recurrent) dropout

- The number of self-attention layers

- The number of neurons in each self-attention 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

Layer Normalization and dropout are kept mutually exclusive because of research indicating that the presence of both generally degrades performance during inference [2]. The article specifically mentions Batch Normalization, so we assume the same effect will most likely present itself using layer normalization on the grounds that both operate on the same principles.

RNNs present a much more challenging task with the resources available on the local machine. We thus implement early stopping and set a maximum iteration limit of 30, assuming 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 and set a relatively small number of iterations available to the tuner.

2.4.2 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.
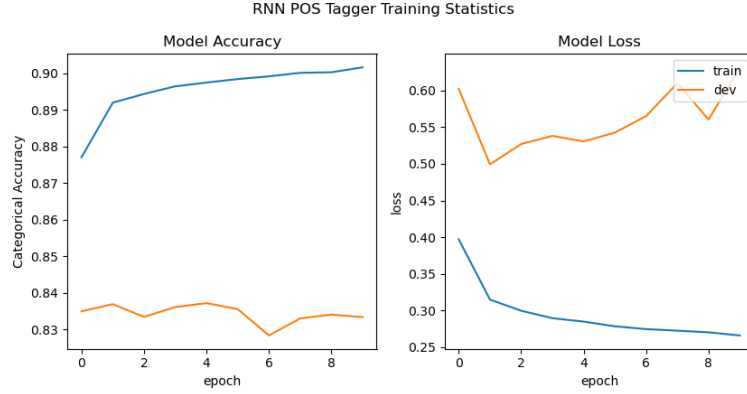
Figure 1: Loss and accuracy on the training and validation sets depending on the number of epochs.

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.

2.4.3 Results

The results of our RNN classifier compared to the previous MLP and baseline models mentioned above can be found in Tables **??**, **??** and **??**. 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.

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

Table 3: Results on the training dataset.

precision			recall			f1		
Baseline	MLP	RNN	Baseline	MLP	RNN	Baseline	MLP	RNN
1.000	1.000	1.000	0.893	0.927	0.049	0.943	0.962	0.094
1.000	1.000	0.000	0.665	0.910	0.000	0.799	0.953	0.000
1.000	1.000	1.000	0.830	0.856	0.297	0.907	0.923	0.459
1.000	1.000	1.000	0.785	0.969	0.289	0.880	0.984	0.448
1.000	1.000	0.000	0.993	0.994	0.000	0.996	0.997	0.000
1.000	1.000	1.000	0.951	0.974	0.000	0.975	0.987	0.000
1.000	1.000	0.000	0.866	0.855	0.000	0.928	0.922	0.000
0.850	0.906	0.060	0.835	0.880	0.088	0.837	0.890	0.043
1.000	1.000	1.000	0.896	0.940	0.024	0.945	0.969	0.047
1.000	1.000	1.000	0.890	0.965	0.045	0.942	0.982	0.087
1.000	1.000	1.000	0.886	0.954	0.568	0.939	0.976	0.725
1.000	1.000	1.000	0.950	0.967	0.032	0.974	0.983	0.063
1.000	1.000	1.000	0.839	0.831	0.039	0.913	0.907	0.074
1.000	1.000	1.000	0.988	0.996	0.002	0.994	0.998	0.003
1.000	1.000	1.000	0.415	0.699	0.001	0.587	0.823	0.002
1.000	1.000	1.000	0.827	0.851	0.001	0.905	0.919	0.003
1.000	1.000	1.000	0.887	0.893	0.002	0.940	0.944	0.004
1.000	1.000	1.000	0.640	0.379	0.140	0.780	0.550	0.246

Table 4: Results on the validation dataset.

precision			recall			f1		
Baseline	MLP	RNN	Baseline	MLP	RNN	Baseline	MLP	RNN
1.000	1.000	1.000	0.825	0.885	0.113	0.904	0.939	0.204
1.000	1.000	0.000	0.669	0.887	0.000	0.802	0.940	0.000
1.000	1.000	1.000	0.796	0.768	0.319	0.886	0.869	0.483
1.000	1.000	1.000	0.792	0.929	0.266	0.884	0.963	0.420
1.000	1.000	0.000	0.990	0.983	0.000	0.995	0.992	0.000
1.000	1.000	0.000	0.942	0.965	0.000	0.970	0.982	0.000
1.000	1.000	0.000	0.635	0.770	0.000	0.777	0.870	0.000
0.787	0.849	0.057	0.730	0.822	0.080	0.744	0.831	0.040
1.000	1.000	1.000	0.885	0.898	0.012	0.939	0.946	0.024
1.000	1.000	1.000	0.608	0.925	0.016	0.756	0.961	0.031
1.000	1.000	1.000	0.869	0.901	0.549	0.930	0.948	0.709
1.000	1.000	1.000	0.947	0.943	0.029	0.973	0.971	0.057
1.000	1.000	1.000	0.462	0.737	0.026	0.632	0.849	0.051
1.000	1.000	1.000	0.983	0.988	0.000	0.991	0.994	0.001
1.000	1.000	0.000	0.442	0.569	0.000	0.613	0.726	0.000
1.000	1.000	0.000	0.747	0.797	0.000	0.855	0.887	0.000
1.000	1.000	1.000	0.801	0.845	0.002	0.890	0.916	0.004
1.000	1.000	1.000	0.022	0.184	0.030	0.044	0.310	0.058

Table 5: Results on the test dataset.

| precision | | | recall | | | f1 | | |
Baseline	MLP	RNN	Baseline	MLP	RNN	Baseline	MLP	RNN
1.000	1.000	1.000	0.827	0.891	0.115	0.905	0.942	0.206
1.000	1.000	0.000	0.667	0.900	0.000	0.801	0.948	0.000
1.000	1.000	1.000	0.827	0.796	0.313	0.905	0.886	0.477
1.000	1.000	1.000	0.784	0.945	0.258	0.879	0.972	0.410
1.000	1.000	0.000	0.985	0.988	0.000	0.992	0.994	0.000
1.000	1.000	0.000	0.953	0.963	0.000	0.976	0.981	0.000
1.000	1.000	0.000	0.717	0.533	0.000	0.835	0.696	0.000
0.778	0.847	0.052	0.741	0.812	0.080	0.748	0.824	0.041
1.000	1.000	1.000	0.891	0.893	0.017	0.943	0.944	0.034
1.000	1.000	1.000	0.539	0.877	0.013	0.700	0.934	0.026
1.000	1.000	1.000	0.897	0.896	0.573	0.946	0.945	0.729
1.000	1.000	1.000	0.950	0.943	0.034	0.974	0.971	0.065
1.000	1.000	1.000	0.481	0.777	0.026	0.649	0.875	0.051
1.000	1.000	1.000	0.979	0.988	0.001	0.989	0.994	0.001
1.000	1.000	0.000	0.440	0.592	0.000	0.611	0.744	0.000
1.000	1.000	0.000	0.835	0.760	0.000	0.910	0.864	0.000
1.000	1.000	1.000	0.826	0.857	0.001	0.905	0.923	0.002
0.000	1.000	1.000	0.000	0.211	0.015	0.000	0.348	0.029

- Average number of characters: 3893.002

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.

- Conversion to lowercase.

- Lemmatization and stop words extraction.

- Punctuation removal.

- Number removal.

- Single characters removal.

- Converting multiple spaces to single ones.

7

```
,: 77717 occurrences
the: 76276 occurrences
.: 65876 occurrences
a: 37995 occurrences
and: 35404 occurrences
of: 33972 occurrences
to: 31772 occurrences
is: 26054 occurrences
in: 21611 occurrences
's: 18128 occurrences
``: 17625 occurrences
it: 16059 occurrences
that: 15912 occurrences
): 11781 occurrences
(: 11664 occurrences
as: 11349 occurrences
with: 10782 occurrences
for: 9918 occurrences
this: 9573 occurrences
his: 9569 occurrences
```

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

```
['not',
 "don't",
 "aren't",
 "couldn't",
 "didn't",
 "doesn't",
 "hadn't",
 "hasn't",
 "shouldn't",
 "haven't",
 "wasn't",
 "weren't",
 "isn't",
 'doesn']
```

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

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.

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.

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

```
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.

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.

- Penalty: We tested 'l1', 'l2' 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= 1, penalty= 'l2', 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 50 and early stopping was used. 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: 0.1

- Number of hidden layers: 1

- Hidden layers' size: 64

- Dropout probability: 0.4

- Batch size: 64

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.

[2] Xiang Li et al. *Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift*. 2018. arXiv: 1801.05134 [cs.LG].
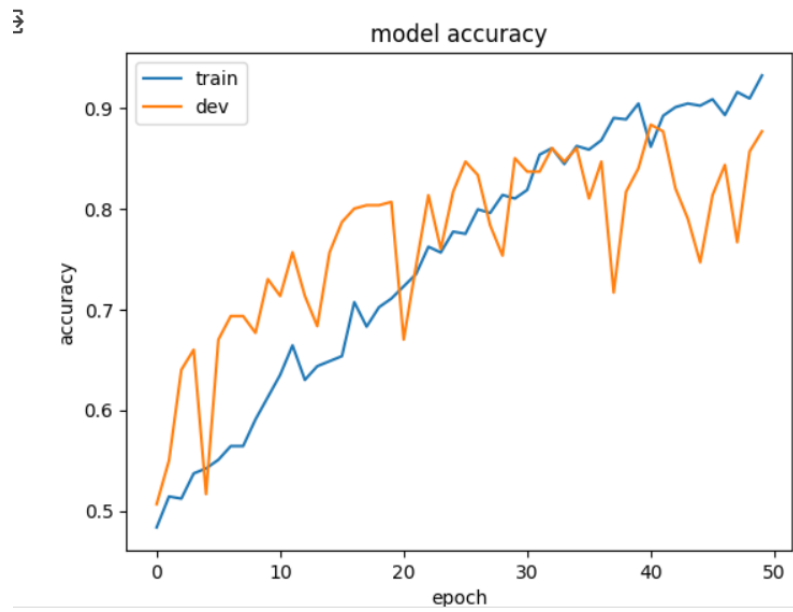
Figure 6: MLP accuracy as a function of epochs.
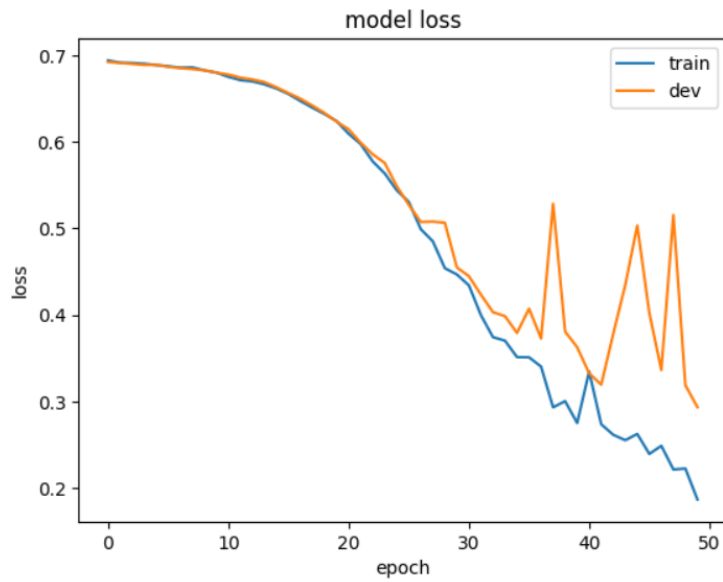

Figure 7: MLP loss as a function of epochs.

```
Class  neg :     (Training)       (Development)       (Test)
Precision    |   0.973837    |     0.900000     |    0.852349
Recall       |   0.971014    |     0.859873     |    0.830065
F1-score     |   0.972424    |     0.879479     |    0.841060
PR AUC       |   0.996832    |     0.945601     |    0.921112
--------------------------------------------------------------
Class  pos :     (Training)       (Development)       (Test)
Precision    |   0.971910    |     0.853333     |    0.827815
Recall       |   0.974648    |     0.895105     |    0.850340
F1-score     |   0.973277    |     0.873720     |    0.838926
PR AUC       |   0.996832    |     0.945601     |    0.921112
--------------------------------------------------------------
```

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: 0.972874
Macro-averaged Recall: 0.972831
Macro-averaged F1-score: 0.972850
Macro-averaged PR AUC: 0.996832

Macro-averaged Scores for Development Subset:
=============================================
Macro-averaged Precision: 0.876667
Macro-averaged Recall: 0.877489
Macro-averaged F1-score: 0.876599
Macro-averaged PR AUC: 0.945601

Macro-averaged Scores for Test Subset:
======================================
Macro-averaged Precision: 0.840082
Macro-averaged Recall: 0.840203
Macro-averaged F1-score: 0.839993
Macro-averaged PR AUC: 0.921112
```

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