

Text Analytics: 3rd Assignment

Tsirmpas Dimitris
Drouzas Vasilis

February 27, 2024

Athens University of Economics and Business
MSc in Data Science

Contents

1	Introduction	2
2	POS Tagging	2
2.1	Dataset	2
2.1.1	Acquisition	2
2.1.2	Qualitative Analysis	2
2.1.3	Preprocessing	3
2.2	Baseline Classifier	3
2.3	MLP Classifier	4
2.4	RNN Classifier	4
2.4.1	Hyper-parameter tuning	4
2.4.2	Training	5
2.4.3	Results	5
3	Sentiment Analysis	11
3.1	Dataset	11
3.1.1	Average Document Length	11
3.1.2	Pre-processing	11
3.1.3	Splitting the dataset	13
3.1.4	SpaCy	13
3.1.5	Padding the sequences	13
3.1.6	Embedding matrix	13
3.2	Classifiers	14
3.2.1	DummyClassifier	14
3.2.2	Logistic Regression	14
3.2.3	Our custom MLP classifier	15
3.2.4	Our custom RNN classifier	16

1 Introduction

This report will briefly discuss the theoretical background, implementation details and decisions taken for the construction of RNN 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.

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 RNN 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	18.96	11.78	5	10	16	24	159
Validation	15.66	10.05	5	8	13	20	75
Test	12518	10.33	5	8	13	20	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	15967	10539
Validation	24005	1538
Test	23811	1535

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.

As we can see from Table 1, there is a sizable portion of our sentences that feature very few words. In order to make the RNN training more efficient, we choose to discard sentences with very few words. We also set a window size equal to the 90% percentile of sentence word count, meaning that 90% of our windows will fully fit the training sentences. The rest will be automatically split into more sentences, and as such don't need to be excluded from the dataset.

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. We note a high accuracy for most tags, which 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 scarce 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. We utilize a layered, bidirectional RNN with GRU cells and a Time-Distributed dense layer.

The parameter search consists of:

- The size of the GRU hidden state
- The number of bidirectional layers
- Whether to use Layer Normalization or dropout
- Whether to use variational (recurrent) dropout
- 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 computationally 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.

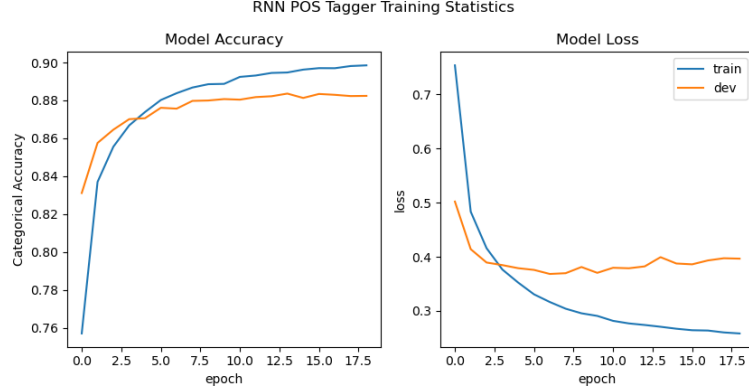


Figure 1: Loss and accuracy on the training and validation sets depending on the number of epochs.

2.4.2 Training

Because of the large computational costs of our optimal model, we keep the very large batch size (256) used in tuning. We do however allow our model to train for more iterations and with more leniency, by increasing the epochs with no improvement before Early Stopping interrupts the training.

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. Training loss and accuracy curves can be found 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 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.

Focusing on the test results we make the following observations:

- The RNN model overall does NOT outperform our MLP classifier, nor the Baseline by overall MACRO-F1 score. This most likely means that a small, lean model, which only considers the immediate vicinity of the target word is sufficient for the POS tagging task in this dataset.
- The RNN model almost completely ignores symbols ("SYM" tag), defined as "a word-like entity that differs from ordinary words by form, function, or both", which "can be substituted by normal words". Symbols have the second lowest support after unknown words (721 instances in the training dataset). It does however beat the MLP classifier and almost matches the scores for "unknown" words ("X" tag), meaning that this is probably not an issue with sparsely seen categories. The RNN also completely ignores punctuation marks.
- In most other categories, the RNN closely matches the MLP baseline.

Table 3: Results on the training dataset.

model	tag	precision	recall	f1
Baseline	ADJ	1.000	0.892	0.943
	ADP	1.000	0.665	0.799
	ADV	1.000	0.829	0.906
	AUX	1.000	0.785	0.879
	CCONJ	1.000	0.993	0.996
	DET	1.000	0.951	0.975
	INTJ	1.000	0.860	0.925
	NOUN	1.000	0.896	0.945
	NUM	1.000	0.880	0.936
	PART	1.000	0.886	0.940
	PRON	1.000	0.950	0.974
	PROPN	1.000	0.834	0.909
	PUNCT	1.000	0.988	0.994
	SCONJ	1.000	0.415	0.587
	SYM	1.000	0.834	0.909
	VERB	1.000	0.888	0.941
	X	1.000	0.578	0.732
	MACRO	0.846	0.831	0.832
MLP	ADJ	1.000	0.923	0.960
	ADP	1.000	0.918	0.957
	ADV	1.000	0.848	0.918
	AUX	1.000	0.966	0.983
	CCONJ	1.000	0.996	0.998
	DET	1.000	0.975	0.987
	INTJ	1.000	0.813	0.897
	NOUN	1.000	0.942	0.970
	NUM	1.000	0.970	0.985
	PART	1.000	0.965	0.982
	PRON	1.000	0.968	0.984
	PROPN	1.000	0.826	0.905
	PUNCT	1.000	0.997	0.999
	SCONJ	1.000	0.663	0.797
	SYM	1.000	0.896	0.945
	VERB	1.000	0.905	0.950
	X	1.000	0.540	0.701
	MACRO	0.929	0.889	0.904
RNN	ADJ	1.000	0.933	0.965
	ADP	1.000	0.882	0.937
	ADV	1.000	0.777	0.875
	AUX	1.000	0.955	0.977
	CCONJ	1.000	0.980	0.990
	DET	1.000	0.977	0.988

Continued on next page

Table 3: Results on the training dataset.

model	tag	precision	recall	f1
	INTJ	1.000	0.778	0.875
	NOUN	1.000	0.922	0.959
	NUM	1.000	0.994	0.997
	PART	1.000	0.980	0.990
	PRON	1.000	0.923	0.960
	PROPN	1.000	0.824	0.903
	PUNCT	1.000	0.015	0.029
	SCONJ	1.000	0.590	0.742
	SYM	1.000	0.009	0.018
	VERB	1.000	0.854	0.921
	X	1.000	0.578	0.732
	MACRO	0.862	0.763	0.751
CNN	ADJ	1.000	0.911	0.954
	ADP	1.000	0.856	0.922
	ADV	1.000	0.790	0.882
	AUX	1.000	0.902	0.949
	CCONJ	1.000	0.980	0.990
	DET	1.000	0.978	0.989
	INTJ	1.000	0.778	0.875
	NOUN	1.000	0.922	0.959
	NUM	1.000	0.988	0.994
	PART	1.000	0.993	0.996
	PRON	1.000	0.959	0.979
	PROPN	1.000	0.828	0.906
	PUNCT	1.000	0.015	0.029
	SCONJ	1.000	0.450	0.621
	SYM	1.000	0.009	0.018
	VERB	1.000	0.898	0.946
	X	1.000	0.553	0.712
	MACRO	0.830	0.753	0.741

Table 4: Results on the validation dataset.

model	tag	precision	recall	f1
Baseline	ADJ	1.000	0.820	0.901
	ADP	1.000	0.669	0.802
	ADV	1.000	0.793	0.885
	AUX	1.000	0.791	0.883
	CCONJ	1.000	0.990	0.995
	DET	1.000	0.942	0.970

Continued on next page

Table 4: Results on the validation dataset.

model	tag	precision	recall	f1
	INTJ	1.000	0.644	0.783
	NOUN	1.000	0.884	0.938
	NUM	1.000	0.633	0.775
	PART	1.000	0.870	0.931
	PRON	1.000	0.948	0.973
	PROPN	1.000	0.466	0.635
	PUNCT	1.000	0.984	0.992
	SCONJ	1.000	0.441	0.612
	SYM	1.000	0.855	0.922
	VERB	1.000	0.803	0.891
	X	1.000	0.020	0.040
	MACRO	0.785	0.738	0.748
MLP	ADJ	1.000	0.885	0.939
	ADP	1.000	0.887	0.940
	ADV	1.000	0.768	0.869
	AUX	1.000	0.929	0.963
	CCONJ	1.000	0.983	0.992
	DET	1.000	0.965	0.982
	INTJ	1.000	0.770	0.870
	NOUN	1.000	0.898	0.946
	NUM	1.000	0.925	0.961
	PART	1.000	0.901	0.948
	PRON	1.000	0.943	0.971
	PROPN	1.000	0.737	0.849
	PUNCT	1.000	0.988	0.994
	SCONJ	1.000	0.569	0.726
	SYM	1.000	0.797	0.887
	VERB	1.000	0.845	0.916
	X	1.000	0.184	0.310
	MACRO	0.849	0.822	0.831
RNN	ADJ	1.000	0.864	0.927
	ADP	1.000	0.892	0.943
	ADV	1.000	0.738	0.850
	AUX	1.000	0.951	0.975
	CCONJ	1.000	0.978	0.989
	DET	1.000	0.976	0.988
	INTJ	1.000	0.598	0.748
	NOUN	1.000	0.790	0.883
	NUM	1.000	0.904	0.950
	PART	1.000	0.966	0.982
	PRON	1.000	0.932	0.965
	PROPN	1.000	0.473	0.642
	PUNCT	1.000	0.007	0.015

Continued on next page

Table 4: Results on the validation dataset.

model	tag	precision	recall	f1
CNN	SCONJ	1.000	0.599	0.750
	SYM	1.000	0.014	0.029
	VERB	1.000	0.828	0.906
	X	1.000	0.245	0.393
	MACRO	0.833	0.692	0.690
	ADJ	1.000	0.835	0.910
	ADP	1.000	0.871	0.931
	ADV	1.000	0.743	0.852
	AUX	1.000	0.888	0.941
	CCONJ	1.000	0.977	0.988
	DET	1.000	0.978	0.989
	INTJ	1.000	0.598	0.748
	NOUN	1.000	0.794	0.885
	NUM	1.000	0.901	0.948
	PART	1.000	0.995	0.998
	PRON	1.000	0.965	0.982
	PROPN	1.000	0.475	0.644
	PUNCT	1.000	0.007	0.015
	SCONJ	1.000	0.469	0.638
	SYM	0.000	0.000	0.000
	VERB	1.000	0.820	0.901
	X	1.000	0.163	0.281
	MACRO	0.773	0.675	0.671

Table 5: Results on the test dataset.

model	tag	precision	recall	f1
Baseline	ADJ	1.000	0.824	0.903
	ADP	1.000	0.667	0.800
	ADV	1.000	0.830	0.907
	AUX	1.000	0.783	0.878
	CCONJ	1.000	0.985	0.992
	DET	1.000	0.954	0.976
	INTJ	1.000	0.733	0.846
	NOUN	1.000	0.888	0.941
	NUM	1.000	0.555	0.714
	PART	1.000	0.896	0.945
	PRON	1.000	0.950	0.974
	PROPN	1.000	0.475	0.644
	PUNCT	1.000	0.983	0.992

Continued on next page

Table 5: Results on the test dataset.

model	tag	precision	recall	f1
	SCONJ	1.000	0.442	0.613
	SYM	1.000	0.861	0.926
	VERB	1.000	0.824	0.903
	X	0.000	0.000	0.000
	MACRO	0.777	0.744	0.749
MLP	ADJ	1.000	0.875	0.933
	ADP	1.000	0.889	0.941
	ADV	1.000	0.805	0.892
	AUX	1.000	0.924	0.961
	CCONJ	1.000	0.997	0.998
	DET	1.000	0.969	0.984
	INTJ	1.000	0.774	0.873
	NOUN	1.000	0.896	0.945
	NUM	1.000	0.843	0.915
	PART	1.000	0.862	0.926
	PRON	1.000	0.939	0.969
	PROPN	1.000	0.741	0.851
	PUNCT	1.000	0.984	0.992
	SCONJ	1.000	0.542	0.703
	SYM	1.000	0.783	0.878
	VERB	1.000	0.869	0.930
	X	1.000	0.114	0.205
	MACRO	0.838	0.812	0.821
RNN	ADJ	1.000	0.875	0.933
	ADP	1.000	0.883	0.938
	ADV	1.000	0.774	0.872
	AUX	1.000	0.948	0.973
	CCONJ	1.000	0.974	0.987
	DET	1.000	0.976	0.988
	INTJ	1.000	0.605	0.754
	NOUN	1.000	0.779	0.876
	NUM	1.000	0.779	0.876
	PART	1.000	0.978	0.989
	PRON	1.000	0.926	0.962
	PROPN	1.000	0.474	0.643
	PUNCT	1.000	0.009	0.017
	SCONJ	1.000	0.558	0.716
	SYM	0.000	0.000	0.000
	VERB	1.000	0.842	0.914
	X	1.000	0.350	0.519
	MACRO	0.783	0.690	0.691
CNN	ADJ	1.000	0.840	0.913
	ADP	1.000	0.863	0.926

Continued on next page

Table 5: Results on the test dataset.

model	tag	precision	recall	f1
	ADV	1.000	0.785	0.880
	AUX	1.000	0.888	0.940
	CCONJ	1.000	0.970	0.985
	DET	1.000	0.976	0.988
	INTJ	1.000	0.605	0.754
	NOUN	1.000	0.784	0.879
	NUM	1.000	0.771	0.871
	PART	1.000	0.995	0.998
	PRON	1.000	0.965	0.982
	PROPN	1.000	0.473	0.643
	PUNCT	1.000	0.009	0.017
	SCONJ	1.000	0.469	0.638
	SYM	0.000	0.000	0.000
	VERB	1.000	0.831	0.908
	X	1.000	0.287	0.447
	MACRO	0.779	0.677	0.675

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

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.

```
,: 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 occurrences.

```
['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.

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.

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

Table 6: Total text volume of each data split.

Set	Mean of sequence length	Standard deviation of sequence length
Training	312.97	134.5
Development	315.1	139.3
Test	305	131.2

Table 7: Mean and standard deviation of the sequence length in training, development and test sets.

- 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 SpaCy

As an additional step to our pre-processing function, we also used SpaCy in order to proceed to the sentence splitting and the tokenization, in the same manner as we discussed in the lab. In the training dataset, we find out that the average word length dropped from 2586.9 (before tokenization) dropped to 312.98 (after tokenization). More statistics about the mean and the standard deviation of the sequence length on the training, development and test sets can be found in Table 7.

3.1.5 Padding the sequences

After that, we used the Tokenizer module from keras preprocessing with maximum number of words to 100000 (so we kept all words actually) and we replaced all rare words with UNK values. We keep a word index (a dictionary where the keys are words (tokens) and the values are their corresponding indices in the tokenizer's vocabulary). Eventually we find out that the number of unique words in the index is 36637.

Next steps involve converting the tokenized sets to sequences and padding these sequences.

3.1.6 Embedding matrix

We downloaded the fasttext binary model that includes pretrained word embeddings. The procedure to create the embedding matrix was the following: We iterated over the word_index dictionary, and for each word we check whether the index is within the limit of MAX_WORDS. If so, we retrieve the word vector from the fasttext model and we assign it to the corresponding word row in the embedding matrix.

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 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 on 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
- Penalty: We tested 'l1', 'l2' regularization 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 .

```

0s [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      0.93      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      0.86      0.86      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      0.85      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 8: Hyperparameters tested in the development set.

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 8).

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:

- Learning rate: 0.1
- Number of hidden layers: 1
- Hidden layers' size: 64
- Dropout probability: 0.4

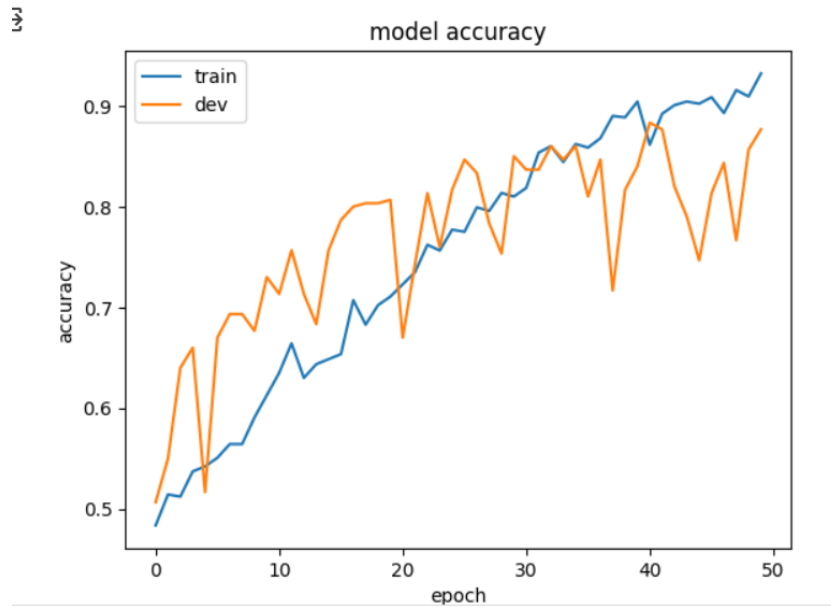


Figure 6: MLP accuracy as a function of epochs.

- 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.

3.2.4 Our custom RNN classifier

We start by creating a Self Attention class, which builds a sequential model as we discussed in the lab. We create the one-hot vectors we will need and now we are ready to construct our RNN model. The RNN model we create is a Sequential one, with:

- An embedding layer, which produces dense vector of fixed size. It utilizes the embedding matrix and sets the pre-trained word embeddings to non-trainable.
- Bidirectional GRU layers (processing the input)
- The self attention layer on the MLP.
- Dense layers, with 'relu' as the activation function.
- Dropout, output layers and the compilation part (using Adam this time).

The hyperparameters we will use here are summarized in Table 9.

We utilize Keras Tuner in order to find the optimal hyperparameters. The best ones are the following:

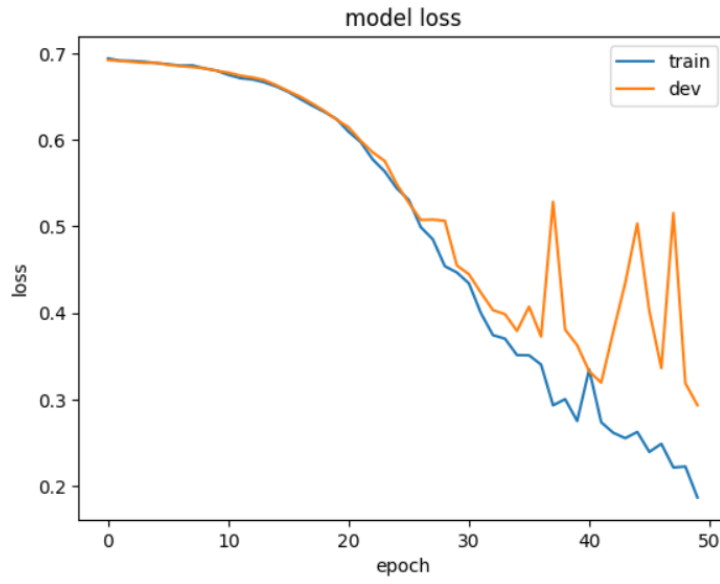


Figure 7: MLP loss as a function of epochs.

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.

Learning rate	#Hidden layers	Hidden layers size	Dropout probability	GRU size	MLP Units
0.001	1	64	0.2	100	64
0.01	2	128	0.25	150	128
0.1	3	256	0.3	200	256
			0.35	250	
			0.4	300	
			0.45	350	
			0.5	400	
				450	
				500	

Table 9: Hyperparameters tested in the development set.

```
▶ 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.

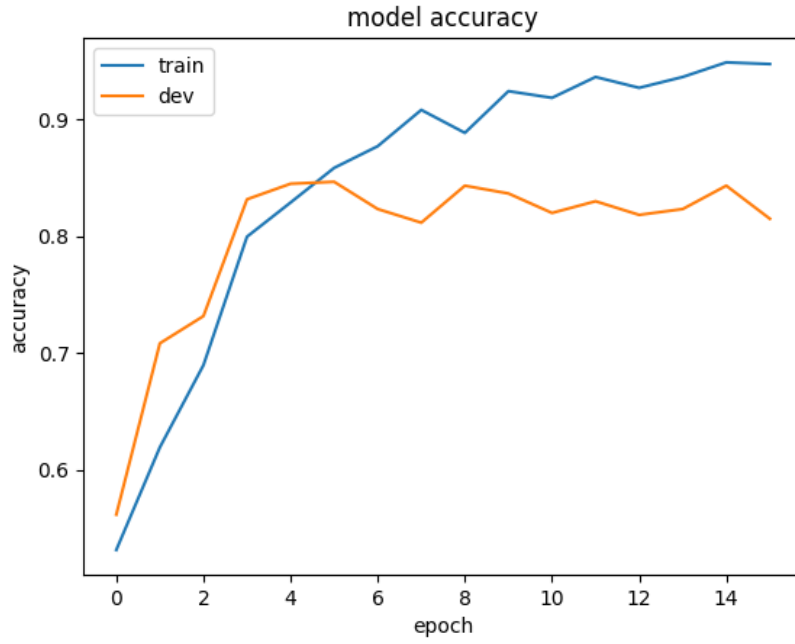


Figure 10: RNN accuracy as a function of epochs.

	Precision	Recall	f1-score	support
neg	0.87	0.95	0.91	690
pos	0.94	0.86	0.9	710
accuracy			0.9	1400
macro avg	0.91	0.9	0.9	1400
weighted avg	0.91	0.9	0.9	1400

Table 10: Classification report on the training set.

- GRU Size: 250
- Dropout rate: 0.3
- MLP layers: 1
- MLP Units: 64
- MLP hidden layer size: 256
- Learning Rate: 0.01

The results we gain are shown in Figures 10, 11.

Finally, we provide the classification report for training, development and test subsets in Tables 10 , 11, 12 and the AUC scores in table 13 .

The MACRO AUC scores were found to be the following:

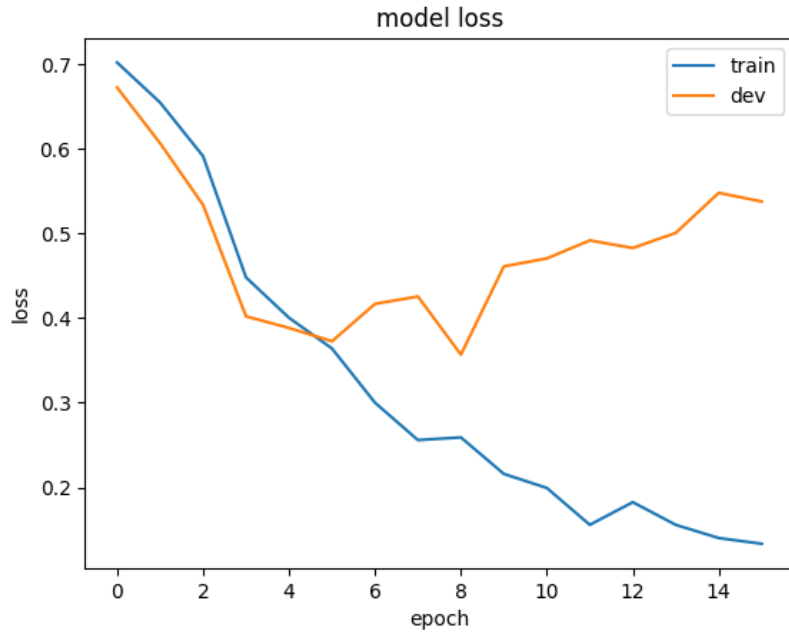


Figure 11: RNN loss as a function of epochs.

	Precision	Recall	f1-score	support
neg	0.82	0.9	0.86	157
pos	0.88	0.79	0.83	143
accuracy			0.85	300
macro avg	0.85	0.84	0.85	300
weighted avg	0.85	0.85	0.85	300

Table 11: Classification report on the development set.

	Precision	Recall	f1-score	support
neg	0.82	0.85	0.84	153
pos	0.84	0.81	0.82	147
accuracy			0.83	300
macro avg	0.83	0.83	0.83	300
weighted avg	0.83	0.83	0.83	300

Table 12: Classification report on the test set.

Class	Training	Development	Test
neg	0.96419	0.91706	0.90574
pos	0.96408	0.91737	0.90565

Table 13: AUC stats for training, development and test sets.

- Training set: 0.9641
- Development set: 0.9172
- Test set: 0.9057

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/341054>. (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](https://arxiv.org/abs/1801.05134) [cs.LG].