Assignment 4

Text Analytics

Submitted by:

- Panagiotis Antoniozas
- Spyros Mastrodimitris Gounaropoulos
- Panagiota Tavoularea

Introduction

For the fourth assignment, we followed the same procedures as in the previous ones up until the RNN section. Therefore, we will outline the main points once again.
Text “movie reviews” was chosen, provided by the sklearn library. We assigned our data to the variables X and Y. X is a list that holds all the reviews, while Y is an array that indicates whether each review is negative (Y=0) or positive (Y=1). We provide some useful metrics to help with the dataset's interpretation.

Number of Total Reviews	***Number of Positive Reviews***	***Number of Negative Reviews***
2000	1000	1000

Avg. Number of Words per Review	***Avg. Number of Words per Positive Review***	***Avg. Number of Words per Negative Review***
629	665	593

Avg. Number of Characters per Review	***Avg. Number of Characters per Positive Review***	***Avg. Number of Characters per Negative Review***
3893	3662	4124

Baseline Model

To establish a baseline model, we trained the algorithm using our train set. The algorithm simply assigns each data point to the class with the highest reviews. We utilised this simple model as a comparison measure for evaluating the performance of other, more complex models.

Logistic Regression

We used Logistic Regression to build a more sophisticated model for classifying the reviews. We trained the Logistic Regression algorithm on our train set and used the dev set to tune the hyperparameters. The model was evaluated using the test set. By comparing the performance of the Logistic Regression model to the baseline model, we were able to assess the extent to which the additional complexity improved the accuracy of the classification.

MLP

In addition to the baseline model and Logistic Regression, we also implemented a Multi-Layer Perceptron to classify the reviews. We trained the MLP model on the train set, optimised the hyperparameters using the dev set, and then evaluated the performance on the test set. The model was trained using backpropagation, which involves computing the error at the output and propagating it back through the layers to update the weights. By comparing the performance of the MLP model to the baseline and Logistic Regression models, we were able to assess the extent to which adding complexity in the form of hidden layers affected the accuracy of the classification.

RNN

Dataset Preprocessing

Firstly, we use the scikit-learn library to split our dataset into train and test by using the 70/30 rule and then with the same rule we divide them into train and development sets on the already-built train set. The output shows how many reviews are in each set.

Number of Reviews in Train Set : 979
Number of Reviews in Test Set : 600
Number of Reviews in Dev Set: 421

Then, we use the spacy library for the text preprocessing which provides tools for text cleaning, tokenization, and normalisation. We remove unwanted characters, stop words, and punctuation symbols from each sentence and store the remaining tokens in a list.

Afterwards, we convert the labels in ytrain, ydev, and ytest to one-hot encoded vectors in order to use them effectively as input and then, once the text has been tokenized into a sequence of words, we convert those words into a sequence of numerical indexes that can be processed by our model. Finally, we pad the sequences to ensure that they all have the same length.

Thereupon, we used Fasttext pre-trained embeddings, where each word is represented by a dense vector of floating-point values, which capture the semantic meaning of the word in the context of the language. One of the advantages of fasttext embeddings is that they can capture the meaning of subwords, in addition to whole words.

Model building

Next, we trained a Recurrent Neural Network model with Bidirectional Gated Recurrent Units (BiGRU) and Self-Attention layers for a text classification task. The model starts with an embedding layer that converts the input sequences into dense vectors of fixed size. The embeddings are initialised with pre-trained weights and are trainable during the training process. The dropout layer is added after the embedding layer to prevent overfitting.

The model then has a stack of BiGRU layers and an optional dropout layer. After the BiGRU layers, the model has a Self-Attention layer that aggregates the information from the BiGRU layers. The Self-Attention layer is followed by a fully connected layer with 256 neurons and an activation function.

The final layer is a softmax layer and the loss function is categorical cross-entropy. The model is optimised using the Adam optimizer, with a learning rate that is chosen from a set.

Hyperparameter tuning

The hyperparameters of the model that are tuned using KerasTuner are the first dropout rate, the number of BiGRU layers, the size of the BiGRU layers, the second dropout rate, the number of self-attention layers, the number of neurons in the self-attention layer, the activation function after the self-attention layer, the third dropout rate, and the learning rate. The hyperparameters are chosen using a combination of random search and Bayesian optimization.

Results - Summary

The corresponding hyperparameters include the first layer of Bidirectional GRU with 600 neurons, followed by a single self-attention layer with 600 neurons and a learning rate of 0.001. The dropout rate used in three layers is 0.5. The activation function used after the self-attention layer is relu. Finally, our model consists of two GRU layers, each containing 300 neurons. In our case, the best validation accuracy achieved is 0.8574 and the RNN model was trained for 50 epochs, with early stopping implemented using a patience of 5.

Evaluation of the Models

The tables below present the results of an evaluation conducted on a baseline classifier, logistic regression, MLP, and RNN models. The performance of these models was compared using several evaluation metrics, including precision, recall, F1, and precision-recall AUC scores and their macro-averaged equivalents. By utilising these metrics, we can compare the models' strengths and weaknesses in terms of their ability to accurately classify different datasets.

Train Set

Classifier	*Precision*		*Recall*		*F1*		*Precision - Recall AUC*		*Macro-Averages*			
	Pos	*Neg*	*Pos*	*Neg*	*Pos*	*Neg*	*Pos*	*Neg*	*Precision*	*Recall*	*F1*	*Pr -AUC*
Baseline	0.51	0	1	0	0.67	0	0.75	0.74	0.25	0.50	0.34	0.75
Logistic Regression	0.97	0.99	0.99	0.97	0.98	0.98	0.99	0.99	0.98	0.98	0.98	0.99
MLP	0.91	0.85	0.85	0.91	0.88	0.88	0.95	0.95	0.88	0.88	0.88	0.95
RNN	1	1	1	1	1	1	0.99	0.99	1	1	1	0.99

Test Set

Classifier	*Precision*		*Recall*		*F1*		*Precision - Recall AUC*		*Macro-Averages*			
	Pos	*Neg*	*Pos*	*Neg*	*Pos*	*Neg*	*Pos*	*Neg*	*Precision*	*Recall*	*F1*	*Pr -AUC*
Baseline	0.51	0	1	0	0.67	0	0.75	0.74	0.25	0.50	**0.34**	**0.75**
Logistic Regression	0.83	0.87	0.88	0.81	0.85	0.84	0.93	0.92	0.85	0.84	**0.84**	**0.93**
MLP	0.87	0.84	0.84	0.87	0.86	0.86	0.93	0.91	0.86	0.86	**0.86**	**0.92**
RNN	0.87	0.82	0.82	0.87	0.84	0.85	0.91	0.90	0.84	0.84	**0.84**	**0.91**

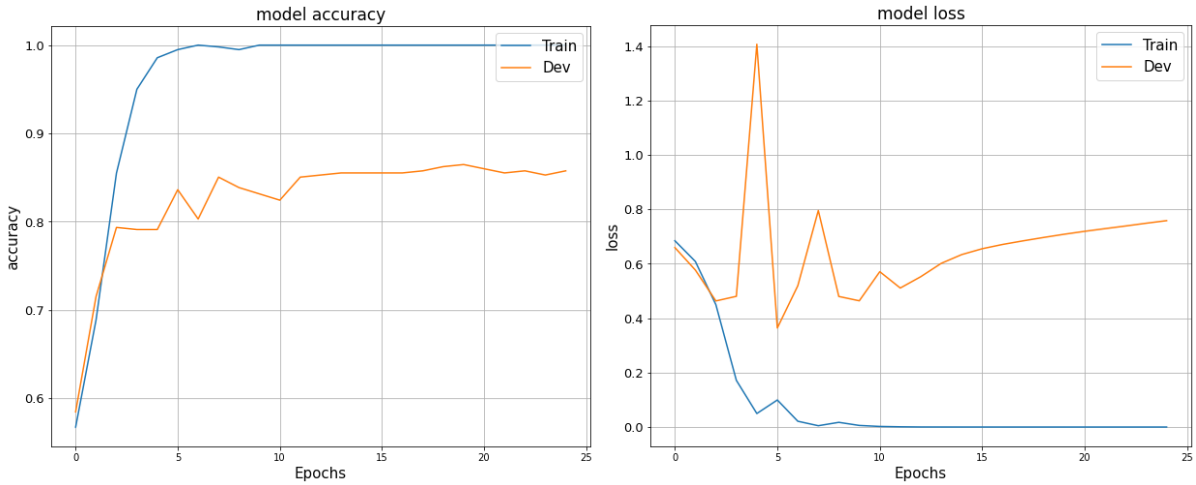
Development Set

Classifier	*Precision*		*Recall*		*F1*		*Precision - Recall AUC*		*Macro-Averages*			
	Pos	*Neg*	*Pos*	*Neg*	*Pos*	*Neg*	*Pos*	*Neg*	*Precision*	*Recall*	*F1*	*Pr -AUC*
Baseline	0.47	0	1	0	0.64	0	0.73	0.76	0.23	0.5	0.32	0.75
Logistic Regression	0.82	0.87	0.86	0.83	0.84	0.85	0.92	0.93	0.85	0.85	0.85	0.93
MLP	0.88	0.84	0.81	0.90	0.84	0.87	0.90	0.93	0.86	0.85	0.86	0.92
RNN	0.87	0.85	0.89	0.82	0.84	0.87	0.91	0.93	0.86	0.86	0.86	0.92

As we mentioned before, MLP outperformed Logistic Regression slightly, as it achieved a macro-accuracy score of 0.86 on the test set compared to Logistic Regression's score of 0.84, but Logistic Regression was preferred to MLP since MLP was seen to be quite complex and required a lot of tuning.

Based on the accuracy values obtained from the experiments, we can conclude that the MLP model is more accurate in making predictions than the RNN model. The MLP model has an accuracy of 0.86 on the test set, while the RNN model has an accuracy of 0.84. It is also observed that the RNN had the ability to learn completely the training dataset with a macro averaged F1 score of 1. A sign that the model is perhaps overqualified for the task at hand either because of the nature of the problem or because of the size of the dataset. In any case our conclusion is the preferred method is the MLP model.

Evaluation of the Learning curves



When examining the learning curve of our RNN, it's ideal to observe a positive trend in model accuracy as training advances. This indicates that the model is learning and enhancing its predictive ability. While we expect both training and validation accuracy to increase during training, they would do so at different rates. On the other hand, as the model is trained on more data, the model loss should decrease, indicating that the model is improving its fit to the data.

Observing the graphs up top, max value of model accuracy regressed at the 20th epoch and min value of model loss increased at the 5th epoch. The learning curve shows a significant gap between the training loss and the validation loss, so it is a strong indication that the model is overfitting. This is further confirmed by the fact that the training loss continues to decrease while the validation loss starts to increase.

Summary
RNN vs MLP vs Logistic Regression

After evaluating the performance of RNN, MLP, and Logistic Regression models, we found that Logistic Regression achieved the similar accuracy among them while maintaining a much lower model complexity compared to the other two. In our case, the fact that our dataset was small played a major role in the performance of our models. With a small dataset, the models may not have had enough data to learn from, leading to serious overfitting that even after significant tuning of the hyperparameters was impossible to stop. In addition, with a small dataset, the models may not have been able to capture the full complexity of the problem, leading to lower accuracy scores on the testing data. This is especially true for more complex models such as RNN and MLP, which require more data to learn the complex patterns in the data.