COMP4650/6490 Document Analysis Assignment 2 – ML

In this assignment, you will build several text classification models on a provided movie review dataset. The dataset consists of 50,000 review articles written movies on IMBD, each labelled with the sentiment of the review – either positive or negative. In addition there are another 50,000 reviews without labels.

In this assignment you will:

1. Become familiar with the Pytorch framework for implementing neural network-based machine learning models.
2. Develop a better understanding of how machine learning models are trained in practice, including partitioning of datasets and evaluation.
3. Study how changing the values of various model hyper-parameters impacts their performance.

Throughout this assignment you will make changes to the provided code to improve the models. In addition, you will produce an answers file with your responses to each question. Your answers file must be a .pdf file named u1234567.pdf where u1234567 is your Uni ID. You should submit a .zip file containing all of the code files and your answers pdf file, **BUT NO DATA.**

Your answers to coding questions will be marked based on the quality of your code (is it efficient, is it readable, is it extendable, is it correct).

Your answers to discussion questions will be marked based on how convincing your explanations are (are they sufficiently detailed, are they well-reasoned, are they backed by appropriate evidence, are they clear).

Question 1 – A simple linear classifier (15%)

You have been provided with various code files to create, train and evaluate models on the given dataset. To begin with, read through the code of and then run *train_vector_classifier.py.* For the vector classifier, the documents will be represented as tf-idf vectors and a simple linear classifier (LogisticRegressor) is trained to predict classes.

Your first task is to change the various settings in this file to try and improve the validation accuracy of the model. The options you can change are:

1. The pre-processor in *preprocessor.py*. You can re-use your pre-processor from assignment 1, or develop a new one.
2. The *get_vector_representation* function in *data_loader.py.* This function specifies how lists of tokens will be transformed into tf-idf vectors. You can find documentation for the CountVectorizer parameters here: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
3. The settings of the LogisticRegressor model in *train_vector_classifier.py.* You can find documentation for the available parameters here: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Pick **three** changes you made that you think caused the most increase in performance, and for each one briefly describe what effect this change has on your model and why you believe this lead to the increased performance.

Leave your code file with the best performing settings.

Question 2 – Embedding based classifier (15%)

Next you will implement an embedding based classifier in pytorch. The model is a simple variation on FastText (you can read more about it here if you are interested https://arxiv.org/pdf/1607.01759.pdf). In *models.py* there is a class FastText, your task is to implement the forward function of this class. Once you have implemented it you can then run *train_embedding_classifier.py* to run your model.

For each input sentence, the model computes its output in 3 steps:

1. Embed the input IDs to create vector representations of each word in the sentence.
2. Aggregate the vectors together to create one vector that represents the entire sentence. You can use any aggregation method you want, e.g. sum, max, average (note that average is significantly more difficult to implement due to padding), etc.
3. Run the aggregated vector through a linear layer to compute predicted logits for each class.

Question 3 – Tuning a pytorch model (15%)

Similarly to question 1, change the various settings of your FastText model to try and improve its validation accuracy. Here are some things you may want to change:

1. The aggregation method used by FastText.
2. The dimension of the embedding vectors.
3. The optimizer, including learning rate, momentum or even algorithm.
4. The number of epochs trained for.
5. The batch size.
6. The preprocessor.

In your answers report **five** different settings you tried, along with their accuracies. You may optionally include some plots of losses over time. For each setting briefly describe why you believe this led to increased/decreased performance.

Leave your code file with the best performing settings.

Question 4 – Comparison of models (10%)

Compare the performance of your best LogisticRegressor model to your best FastText model. Which one performed better? You should explain what measures you are using to judge these models and why. You should also explain why you think the LogisticRegressor/FastText performed better than the other on this dataset.

Question 5 – Implement Word2Vec (15%)

You will now train a Word2Vec model to produce embeddings for the words in the vocabulary. You will need to complete the *batch_method* function of the *TextDS* class in *data_loader.py.* This function receives as input a DataFrame, where each cell of the "ids" column is a list of the token IDs from some movie review. Each movie review should be converted into a list of context windows, with the centre word of each window used as the target and the other words used as input. The same FastText model from before will be used to do the prediction. Once you have implemented the function, you will be able to run *train_word2vec.py.*

Question 6 – Compare pre-trained embeddings (10%)

After train_word2vec.py has finished running it will produce a file of word embeddings, and you will then be able to run *train_pretrained_classifier.py. train_pretrained_classifier.py* trains a FastText classifier, except instead of the embeddings being learned from scratch, they will be read from the provided file and then frozen. Make sure that your *train_pretrained_classifier.py* script uses the same settings as *train_embedding_classifier.py* so that you can compare them.

In your answers report the accuracy of the pretrained classifier with the original word2vec settings, along with **Three** other settings. Did your FastText model perform better with or without pre-trained embeddings? Why do you think this is?

Leave your code file with the best performing settings.

Question 7 – Further Modification (20%)

Your final task is to implement some further modification to the system to improve performance. Exactly what kind of modification you implement is left up to you.

Some examples of modifications you could make:

- Replace the linear classifier in FastText with a multi-layer neural network.
- Implement a position dependent word2vec model.
- Replace the aggregation in FastText with a sequence model – such as GRU or LSTM.
- Use a Transformer to perform the classification.
- Anything else you can think of.

You should implement your changes to this question in *train_final_classifier.py* and leave the existing files as they are.

You can change the proportion of training/validation/testing data in __settings__.py if you wish to use more of the data in training, though it will make the scripts take longer to run.

In your answer to this question provide a list every change that you tried along with its corresponding accuracies.

Academic Misconduct Policy: All submitted written work and code must be your own (except for any provided starter code, of course) – submitting work other than your own will lead to both a failure on the assignment and a referral of the case to the ANU academic misconduct review procedures: ANU Academic Misconduct Procedures.