Assignment 2 - Exercise 9 - Text Classification with Multi-Layer Perceptrons

Contributor: Dimitris Vougioukos - f3352411

The exercise 9 focuses on building an MLP classifier for sentiment analysis on an existing dataset and comparing its performance with baseline classifiers.

The following chapters will outline the steps taken and the design decisions made to achieve the desired results.

Exercise 9

At the beginning of the exercise, we chose to download and use the `movie_reviews` dataset from the `NLTK` library, which contains various movie reviews. We examined its structure and identified two file types, `neg` (negative review) and `pos` (positive review), representing the distinct classes under which the reviews are categorized. Specifically, we observed that the dataset contains 1.000 reviews categorized under the `neg` class and another 1.000 reviews under the `pos` class. This defines a binary classification problem.

After that, we extracted the raw text of each review along with its corresponding class/label and stored them in two separate lists for further processing. During the preprocessing phase, we first applied the `sent_tokenize` function from the `NLTK` library to segment the reviews into sentences. Then, we used the `word_tokenize` function from the same library to tokenize the sentences. We filtered out stopwords, lemmatized the remaining tokens to their base forms, and combined them to create the final processed reviews. We observed that the average length of the reviews was 342 tokens.

Next, we used the `train_test_spli`t function from the `sklearn` library to divide the dataset into three subsets: training, validation, and test sets, with the following sizes.

Subset	Size - Number of Reviews
Training	1120
Validation	480

Subset	Size - Number of Reviews
Test	400

The next step involved generating different numerical representations of the reviews using TF-IDF and Word2Vec embeddings.

1. TF-IDF

To represent the review texts using TF-IDF, we initialized a TF-IDF vectorizer and applied it to the dataset.

2. Word2Vec

To represent the review texts using Word2Vec embeddings, we first downloaded the pre-trained `word2vec-google-news-300` model, which was trained on a vast Google News dataset. We then defined a function to generate the embeddings, as described below.

Function- find_word2vec_embeddings

This function converts a set of tokenized reviews into numerical representations using Word2Vec embeddings.

The function follows these steps:

- Iterates through each review.
- For each word, checks if it exists in the provided Word2Vec embeddings and appending its vector representation.
- Computes a single vector representation for the review
  - If no words in the review have embeddings, assigns a zero vector.
  - If the words in the review have embeddings, calculates the mean (centroid) of all word embeddings in the review.
- Returns all review embeddings.

To build the MLP classifier, we implemented classes and functions for each step of training, evaluation, and testing, as described below.

Class - MLPClassifier

This class defines a neural network (MLP) for binary classification and consists of two functions.

Function - Constructor - init

This is the constructor of the class which builds the network dynamically as it follows.

- Iterates through hidden_layers and adds:
  - Fully connected layers.
  - ReLU activation function.
  - Optional Dropout, BatchNorm1d, or LayerNorm if specified.
- Adds an output layer including 1 neuron with a Sigmoid activation fuction.

Function - forward

This function passes the input data through the sequential neural network (MLP) and returns the output of the Sigmoid activation function in the final neuron, which represents the probability of the positive class.

Function - train_mlp_classifier

This function trains an MLP classifier using a given dataset, optimizes its weights, and implements early stopping to prevent overfitting.

The function follows these steps:

- Initialize training settings
  - Uses `Binary Cross-Entropy Loss` for binary classification.
  - Optimizes using `Adam` optimizer with a specified learning rate.
  - Implements early stopping with a patience of 3 epochs.
- Training Loop
  - Iterates over the mini-batches od the training dataset
    - Resets the optimizer gradients.
    - Performs forward pass to get predictions.
    - Computes loss.
    - Applies backpropagation to compute gradients.
    - Updates model weights.
  - Stores the average training loss for the epoch.
- Validation Loop
  - Iterates over the mini-batches of the validation dataset.
  - Computes validation loss.

- Stores the average validation loss for the epoch.
- Perform early stopping:
    - If the new validation loss is lower, saves the model and deletes the previous best model checkpoint.
    - If no improvement for 3 consecutive epochs, early stopping is triggered.
- Returns
    - A list of training losses per epoch.
    - A list of validation losses per epoch.
    - The file path of the best model checkpoint.

Function - compute_classification_scores

This function is used to compute the classification metrics of a classifier, including precision, recall, f1-score, and precision-recall AUC.

The function follows these steps:

- Computes precision, recall, and f1-score for `neg` (negative) and `pos` (positive) classes and the macro-averaged values of them.
- Computes precision-recall AUC for `neg` (negative) and `pos` (positive) classes.
- Computes macro-averaged precision-recall AUC
- Returns the classification scores

Function - evaluate_mlp_classifier

This function evaluates a trained MLP classifier on a given dataset by computing classification metrics.

The function follows these steps:

- Sets the classifier to evaluation mode
- Iterates over the mini-batches of the provided dataset
    - Gets predicted probabilities from the classifier.
    - Converts probabilities to binary predictions using a 0.5 threshold.
- Compute classification scores using the `compute_classification_scores` function.
- Returns the classification scores.

After defining all necessary functions, we trained and evaluated multiple MLP classifiers using different text representations (TF-IDF, Word2Vec) and classifier configurations (hidden layers, dropout, normalization).

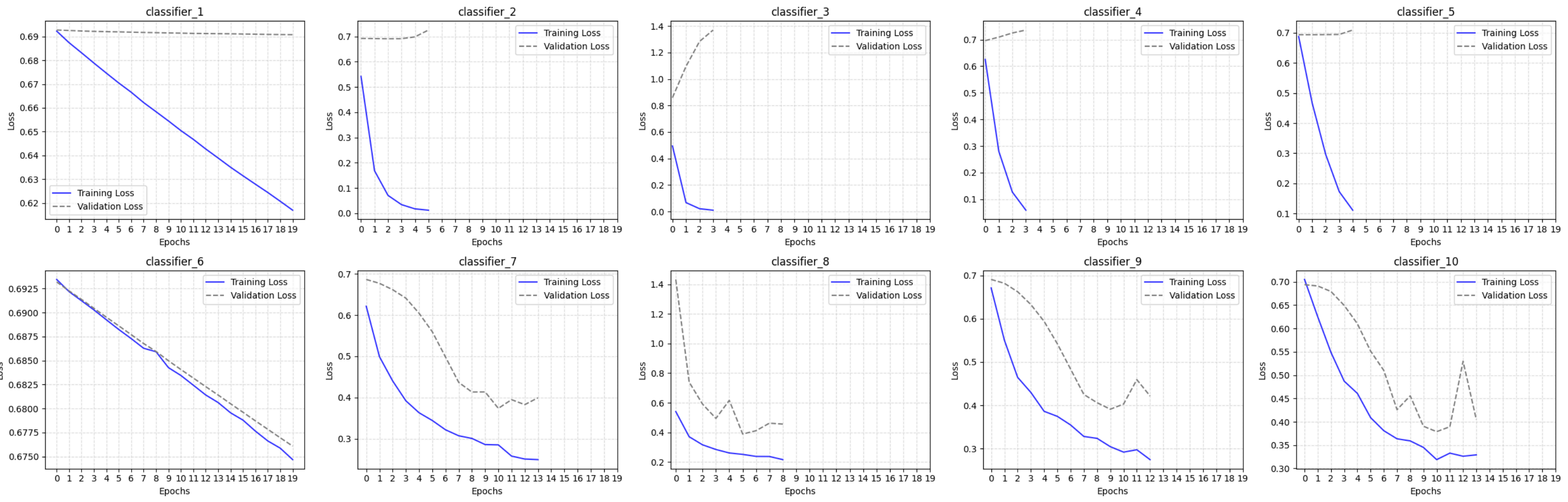
First, we converted the training, validation, and test data into `PyTorch` tensors and loaded them into `DataLoaders` for batch processing. For each classifier configuration, we initialized an MLP model, trained it using the `train_mlp_classifier function`, and evaluated it on the training, validation, and test sets using the `evaluate_mlp_classifier` function, utilizing the best performing checkpoint for each classifier.

Below are the configurations tested for all classifiers.

Classifier	Text Representation	Hidden Layers	Dropout Probability	Batch Normalization	Layer Noralization
Classifier_1	TF-IDF	None	0	False	False
Classifier_2	TF-IDF	One hidden layer with 256 neurons	0.3	True	False
Classifier_3	TF-IDF	One hidden layer with 256 neurons	0	True	True
Classifier_4	TF-IDF	Two hidden layers with 256 and 128 neurons respectively	0.3	True	False
Classifier_5	TF-IDF	Three hidden layers with 256, 128, 64 neurons respectively	0.3	True	False
Classifier_6	Word2Vec	None	0	False	False
Classifier_7	Word2Vec	One hidden layer with 256 neurons	0.3	True	False
Classifier_8	Word2Vec	One hidden layer with 256 neurons	0	True	True
Classifier_9	Word2Vec	Two hidden layers with 256 and 128 neurons respectively	0.3	True	False
Classifier_10	Word2Vec	Three hidden layers with 256, 128, 64 neurons respectively	0.3	True	False

For all MLP classifiers, we tracked the training and validation losses across all epochs during training process and recorded the classification scores for the training, validation, and test datasets.

After collecting all the loss values, we visualized the corresponding curves, as shown below.



As seen in the plots above, all MLP classifiers using TF-IDF text representation perform poorly. The validation loss increases rapidly from the early epochs, while the training loss decreases, indicating that the model memorizes the training data but fails to generalize to unseen data (overfitting). A possible reason for this is that TF-IDF treats words as independent entities, failing to capture their meaning and context.

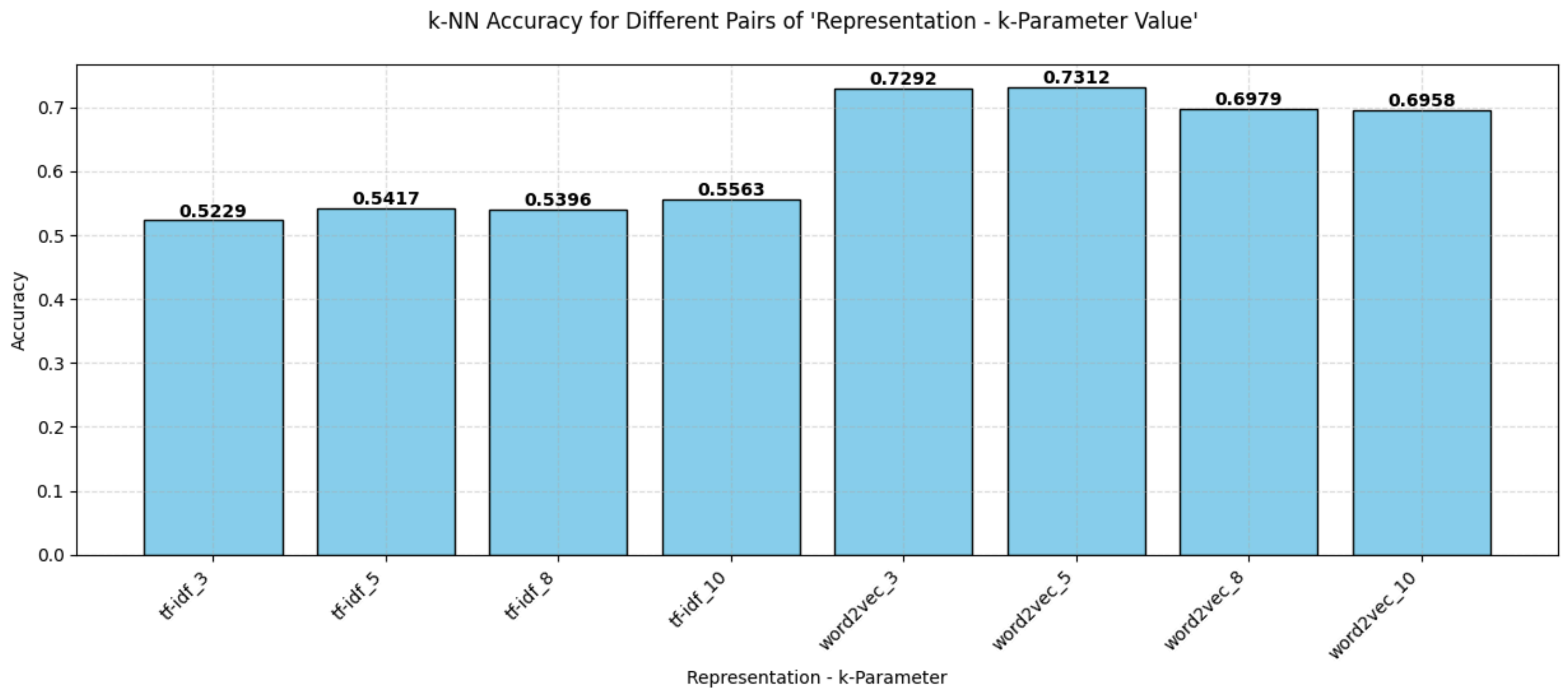
In contrast, classifiers using Word2Vec representations perform significantly better. Their loss curves decrease throughout training until a point where the validation loss starts to rise, triggering early stopping.

Among all the classifiers, the one that achieved the lowest validation loss was classifier_7, with a value of approximately 0.3737. Therefore, we selected it for comparison with the baseline classifiers, as detailed below.

For the baseline classifiers, i build a k-Nearest Neighbors (k-NN) classifier and a majority classifier.

k-Nearest Neighbors (k-NN) Classifier

To select the best version of the classifier, i tuned it across different values of the k-parameter ([3, 5, 8, 10]) and experimented with different text representations (TF-IDF and Word2Vec), keeping the one that achieved the highest accuracy score.



From the bar plot above, we observed that the best performing version of the k-NN classifier is the one using 5 nearest neighbors (k-parameter) and Word2Vec embeddings, achieving an accuracy score of 0.7312. Using this configuration, we then evaluated its performance on the training, development, and test datasets using the `compute_classification_scores` function.

Majority Classifier

To build this classifier, we identified the most frequently occurring class in the training dataset. We then generated predictions for the training, validation, and test datasets by assigning this majority class to all review instances. Finally, we evaluated its performance using the `compute_classification_scores` function.

Classification Scores across all Classifiers

Training Dataset

Class	Metric	MLP Classifier	k-NN Classifier	Majority Classifier
neg	Precision	0.9251	0.8237	0.0
	Recall	0.8821	0.8429	0.0
	F1-Score	0.9031	0.8332	0.0
	Precision-Recall AUC Score	0.9658	0.9091	0.75
pos	Precision	0.8874	0.8391	0.5
	Recall	0.9286	0.8196	1.0
	F1-Score	0.9075	0.8293	0.6667
	Precision-Recall AUC Score	0.9698	0.9108	0.75
macro avg	Precision	0.9062	0.8314	0.25
	Recall	0.9054	0.8313	0.5
	F1-Score	0.9053	0.8312	0.3333
	Precision-Recall AUC Score	0.9678	0.91	0.75

Validation Dataset

Class	Metric	MLP Classifier	k-NN Classifier	Majority Classifier
neg	Precision	0.8435	0.7094	0.0
	Recall	0.8083	0.7833	0.0
	F1-Score	0.8255	0.7446	0.0
	Precision-Recall AUC Score	0.9209	0.8141	0.75
pos	Precision	0.816	0.7581	0.5
	Recall	0.85	0.6792	1.0
	F1-Score	0.8327	0.7165	0.6667

Class	Metric	MLP Classifier	k-NN Classifier	Majority Classifier
	Precision-Recall AUC Score	0.9041	0.7891	0.75
macro avg	Precision	0.8297	0.7338	0.25
	Recall	0.8292	0.7312	0.5
	F1-Score	0.8291	0.7305	0.3333
	Precision-Recall AUC Score	0.9125	0.8016	0.75

Test Dataset

Class	Metric	MLP Classifier	k-NN Classifier	Majority Classifier
neg	Precision	0.8514	0.7062	0.0
	Recall	0.745	0.745	0.0
	F1-Score	0.7947	0.7251	0.0
	Precision-Recall AUC Score	0.9011	0.7804	0.75
pos	Precision	0.7733	0.7302	0.5
	Recall	0.87	0.69	1.0
	F1-Score	0.8188	0.7095	0.6667
	Precision-Recall AUC Score	0.8849	0.7638	0.75
macro avg	Precision	0.8124	0.7182	0.25
	Recall	0.8075	0.7175	0.5
	F1-Score	0.8067	0.7173	0.3333
	Precision-Recall AUC Score	0.893	0.7721	0.75

From the classification scores across all datasets, we reached the following conclusions:

- The MLP Classifier is the best-performing model, achieving high Precision, Recall, F1-Score, and AUC across all datasets.

- The k-NN Classifier struggles with generalization and performs significantly worse than the MLP.
- The Majority Classifier is ineffective, as expected, since it does not differentiate between classes.