NLP with Recurrent Neural Networks (RNNs)

Alviona Mancho f3322405 alv.mantso@aueb.gr / alviona.mantso@gmail.com
Anna Chatzipapadopoulou f3322411 ann.chatzipapadopoul@aueb.gr / annachatzipap@gmail.com

Natural Language Processing, 2024-2025[1]

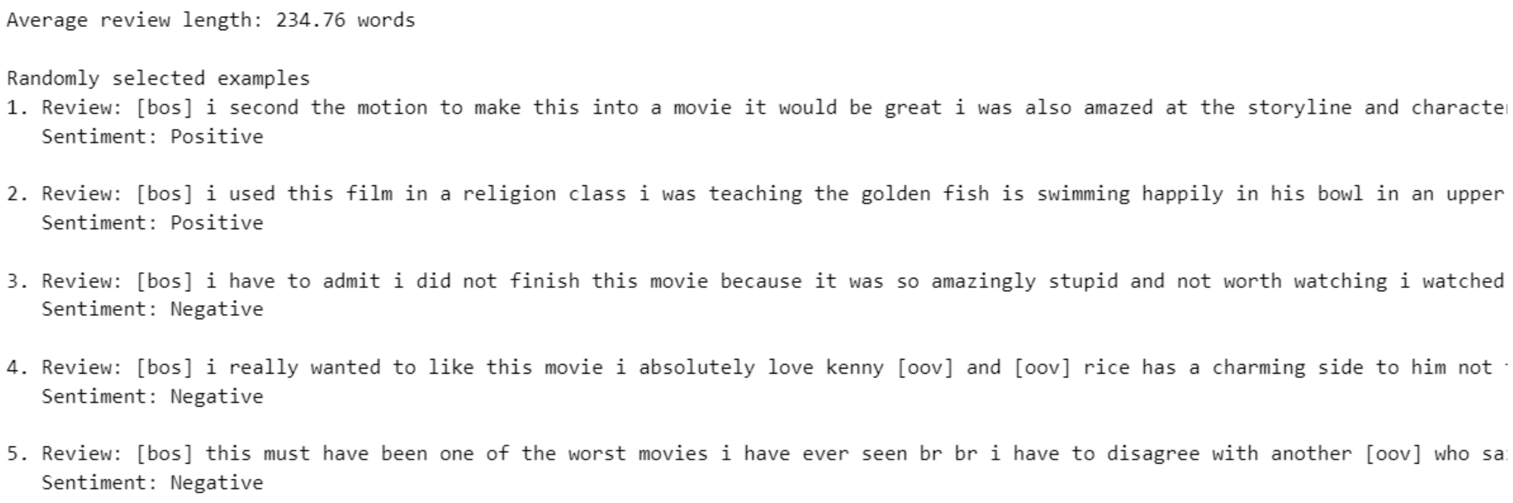
Notebook link (ex. 1) Notebook link (ex. 2)

1. Exercise 1 (Sentiment Analysis using an BiLSTM RNN with "deep" self-attention)

For this exercise we used the **IMDb Movie Reviews** dataset. The implementation includes an MLP classifier in PyTorch.

Dataset Composition

- **Content**: The dataset contains movie reviews from IMDb, which are labeled as either positive or negative.
- **Labels**: It's a binary classification dataset, with labels 0 (negative) and 1 (positive).
- **Vocabulary size**: Is set to 10,000.
- **Average document (review) length**: 234.76 words
- **Data Split**: The dataset has 25,000 reviews for training and 25,000 for testing, ensuring a balanced split of positive and negative reviews in each set. We further divide the testing data into a development (or validation) set and a final test set, each containing 12,500 reviews. For the sake of efficiency, we utilized a subset of this dataset, with 5000 samples for training and 2000 samples each for validation and testing.



```
Average review length: 234.76 words

Randomly selected examples
1. Review: [bos] i second the motion to make this into a movie it would be great i was also amazed at the storyline and characte
   Sentiment: Positive

2. Review: [bos] i used this film in a religion class i was teaching the golden fish is swimming happily in his bowl in an upper
   Sentiment: Positive

3. Review: [bos] i have to admit i did not finish this movie because it was so amazingly stupid and not worth watching i watched
   Sentiment: Negative

4. Review: [bos] i really wanted to like this movie i absolutely love kenny [oov] and [oov] rice has a charming side to him not
   Sentiment: Negative

5. Review: [bos] this must have been one of the worst movies i have ever seen br br i have to disagree with another [oov] who sa
   Sentiment: Negative
```

[1] *Both team members engaged equally in all tasks, collaborating synchronously throughout the assignment.*

Data preprocessing and Feature Extraction:

The class `DataPreprocessor` handles the preprocessing of the data. It features the following methods:

- `fetch_data(num_words):` Retrieves the IMDb dataset, limits the vocabulary size to the specified `num_words,` and splits the data into training, testing, and development sets. It also computes the average review length across all sets. Additionally, it converts each review from a sequence of word indices into readable text using a word index dictionary, where certain indices are mapped to special tokens (e.g., `[pad], [bos], [oov]`). Finally, it returns the processed training, testing, and development data, along with the average review length.
- `preprocess_data(X):` Preprocesses the text data by cleaning, tokenizing, and lemmatizing the words based on their parts of speech to improve the quality of the input data.
- `build_vocab(X_tokenized, max_words=100000):` Creates a vocabulary of the most common tokens from tokenized sentences, assigning unique indices to each token. It includes special tokens <pad> (for padding) and <unk> (for unknown tokens).
- `encode_and_pad(X_tokenized, vocab, max_length=250):` Encodes and pads tokenized sentences using the vocabulary.
- `prepare_embedding_matrix(vocab,word2vec_model,embedding_dim=300, max_words=100000):` Prepares the embedding matrix for the vocabulary using a pre-trained Word2Vec model.
- `prepare_data_loaders(train_data, val_data, test_data, y_train, y_val, y_test, batch_size=64):` Prepares PyTorch DataLoaders for training, validation, and testing.
- `generate_embeddings(X, model, embedding_dim = 300):` Generates word embeddings for the given documents using a pre-trained word embedding model. Each document is represented as the average of its word vectors.

Initially, the data is fetched using `fetch_data(num_words), where num_words` is set to 10,000, namely the vocabulary size. As described above, this method converts each review from a sequence of word indices into readable text using a word index dictionary, where certain indices are mapped to special tokens (e.g., `[pad], [bos], [oov]`). Finally, it returns the processed training, testing, and development data, containing 25,000, 12,500 and 12,500 reviews respectively. For the sake of efficiency, we utilized a subset of this dataset, with 5000 samples for training and 2000 samples each for validation and testing. Subsequently, the training, testing and development sets are passed to `preprocess_data,` which cleans and standardizes text data by removing special characters and single characters, normalizing whitespace, and converting text to lowercase. It then tokenizes the text, tags parts of speech for improved lemmatization, and lemmatizes words. Next, the vocabulary is built from the training data (`build_vocab`) and all datasets are encoded and padded using the vocabulary (`encode_and_pad).` Then the embedding matrix for the vocabulary is prepared using `word2vec-google-news-300` model from Gensim's API to obtain 300-dimensional word embeddings.

BiLSTM_Attention Class:

This class implements a Bi-directional LSTM model with an attention mechanism for sequence classification tasks. It processes input sequences through an embedding layer, a bi-directional LSTM, an attention MLP (to compute attention scores), and a dense classification layer. Dropout layers are used throughout to prevent overfitting

- `__init__` (Constructor): Initializes the model with specified parameters and constructs its layers.
  - **input_dim**: Dimension of the input embeddings.
  - **n_classes**: Number of output classes.

    dropout_prob_emb, **dropout_prob_att**, **dropout_prob_out**: Dropout probabilities for embedding, attention, and output layers, respectively.
  - **hidden_dim**: Size of the hidden layer in the dense block.
  - **lstm_hidden_dim**: Number of hidden units in the LSTM layer.
  - **lstm_stacks**: Number of stacked LSTM layers.
  - **attention_hidden_sizes**: Sizes of hidden layers in the attention MLP.
  - **max_words**: Maximum vocabulary size.
  - **matrix_embeddings**: Pre-trained embedding matrix.
- `_build_attention_mlp:` Constructs a multi-layer perceptron (MLP) to compute attention scores.
  - **input_size:** Input size to the MLP (twice the LSTM hidden dimension due to bidirectionality).
  - **hidden_sizes:** List of hidden layer sizes for the MLP.
- `forward` (Forward Pass):

  Implements the forward pass:
  - Converts input tokens into dense vectors.
  - Applies dropout to embeddings.
  - Processes embeddings to capture bidirectional sequence features.
  - Computes attention scores using the MLP.
  - Applies a softmax to get attention weights.
  - Computes a weighted sum of LSTM outputs using attention weights.
  - Applies a dense transformation with ReLU activation and dropout.
  - Finally, outputs logits using the final dense layer.

Classifier_Wrapper Class:

This class provides a training and evaluation framework for both the MLP model and the BiLSTM RNN model. It integrates the training process, prediction of class labels, and prediction of class probabilities.

- `fit`(Training):

  Trains the model over the specified number of epochs using the provided DataLoader:
  - Performs forward passes, calculates the loss using the criterion, and updates model weights using the optimizer.
  - Tracks and prints the average loss for each epoch.

    - Returns a **history** dictionary containing the loss values for each epoch.
- `predict` (Class Prediction): Predicts class labels for a given input dataset X.
- `predict_proba` (Probability Prediction): Predicts class probabilities for a given input dataset X, using softmax.

Utils Class:

This utility class provides static methods to train, evaluate, and validate an MLP model. It supports detailed metric tracking, model checkpointing, and reporting of accuracy and F1 scores.

- `train_and_evaluate:` This method trains an MLPModel instance on a given training dataset and evaluates its performance on a test dataset.
    - The training phase includes iterative optimization of model parameters using the provided optimizer and loss function over a specified number of epochs. During each epoch, the average training loss is computed and displayed to monitor progress.
    - In the evaluation phase, the model's predictions on the test dataset are compared with the true labels to compute accuracy and F1 score. The method collects all predictions and true labels during testing to calculate these metrics comprehensively. The final test accuracy and F1 score are printed.

- `train_and_validate:` This method provides a more detailed training and validation process, tracking metrics for both datasets across epochs and saving the best-performing model. During each epoch, the model's performance is evaluated on the training dataset, with metrics such as loss, accuracy, and F1 score being computed. Simultaneously, the validation set is used to assess how well the model generalizes, using the same metrics.

  The method also implements a checkpoint mechanism, where the model with the lowest validation loss is saved to a file. If a new best model is found, the previous checkpoint is deleted to conserve resources. This ensures that the best model is always available for deployment or further evaluation. Additionally, all training and validation metrics are logged in a history dictionary. At the end of training, the history and path to the best model are returned.

Experiments:

The primary objective of our experiments is to assess and compare the performance of our BiLSTM RNN classifier against three baseline models: the MLP classifier, as implemented in the previous assignment, a majority classifier and the top-performing classifier identified in a previous assignment (i.e. the Logistic Regression with SGD). First, we configure the BiLSTM classifier with initial, arbitrarily chosen hyperparameters. This initial testing allows us to observe baseline performance and helps us identify potential adjustments for further tuning. Subsequent steps involve tuning these hyperparameters for optimal performance.

Hyperparameter tuning our BiLSTM RNN Classifier:

The hyperparameter tuning is performed using a grid search approach.

Hyperparameters Tuned

- **Embedding Dropout Probability (`dropout_prob_emb`)**: Dropout for the embedding layer.
- **Attention Dropout Probability (`dropout_prob_att`)**: Dropout applied during the attention mechanism.
- **Output Dropout Probability (`dropout_prob_out`)**: Dropout applied during the final output layer.
- **Dense Hidden Layer Size (`hidden_dim`)**: Determines the dimension of the fully connected dense layer after the LSTM.
- **LSTM Hidden Dimension (`lstm_hidden_dim`)**: Specifies the dimension of the hidden LSTM layer.
- **LSTM Stacks (`lstm_stacks`)**: Defines the number of LSTM layers.
- **Attention Hidden Sizes (`attention_hidden_sizes`)**: Architecture of the attention MLP, specifying the number of layers and their sizes.
- **Learning Rate (`learning_rate`)**: Step size for the optimizer during weight updates.

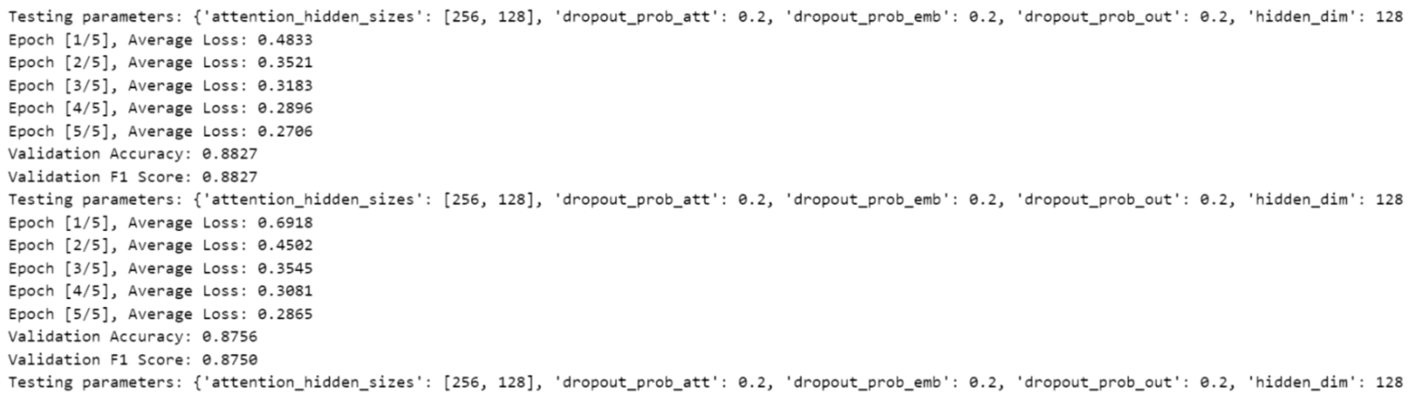
Procedure:

1. **Data Preparation**: Input data is converted into tensors and padded/truncated to ensure uniform sequence lengths. Then, a data loader is used to batch the data and feed it into the model during training.
2. **Model Initialization**: The `BiLSTM_Attention` model is instantiated with the current hyperparameter configuration.

Training: The model is trained for a fixed number of epochs (`epochs = 5`) using Adam optimizer and cross-entropy loss.

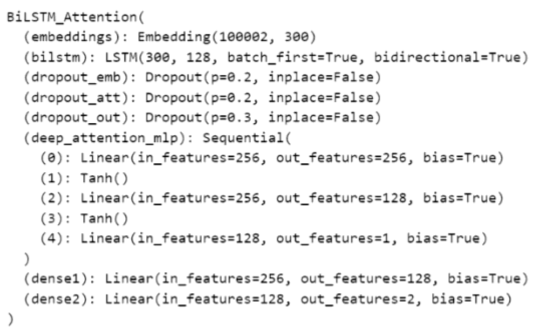
Evaluation: After each training configuration, the model is evaluated on the validation set, where the macro F1-score is used as the evaluation metric.

The best-performing set of hyperparameters is identified by iterating through all possible combinations and selecting the one that maximizes the development set's macro F1 score. The final results include the best hyperparameters and their associated performance.



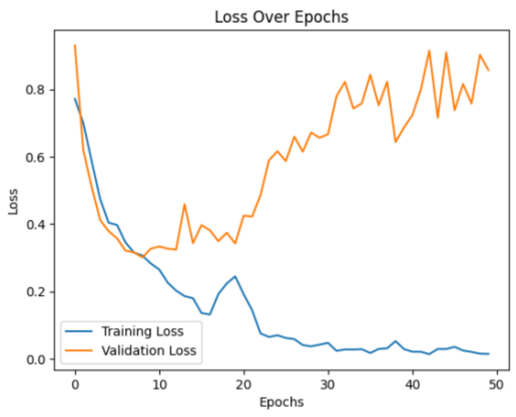
```
Testing parameters: {'attention_hidden_sizes': [256, 128], 'dropout_prob_att': 0.2, 'dropout_prob_emb': 0.2, 'dropout_prob_out': 0.2, 'hidden_dim': 128
Epoch [1/5], Average Loss: 0.4833
Epoch [2/5], Average Loss: 0.3521
Epoch [3/5], Average Loss: 0.3183
Epoch [4/5], Average Loss: 0.2896
Epoch [5/5], Average Loss: 0.2706
Validation Accuracy: 0.8827
Validation F1 Score: 0.8827
Testing parameters: {'attention_hidden_sizes': [256, 128], 'dropout_prob_att': 0.2, 'dropout_prob_emb': 0.2, 'dropout_prob_out': 0.2, 'hidden_dim': 128
Epoch [1/5], Average Loss: 0.6918
Epoch [2/5], Average Loss: 0.4502
Epoch [3/5], Average Loss: 0.3545
Epoch [4/5], Average Loss: 0.3081
Epoch [5/5], Average Loss: 0.2865
Validation Accuracy: 0.8756
Validation F1 Score: 0.8750
Testing parameters: {'attention_hidden_sizes': [256, 128], 'dropout_prob_att': 0.2, 'dropout_prob_emb': 0.2, 'dropout_prob_out': 0.2, 'hidden_dim': 128
```

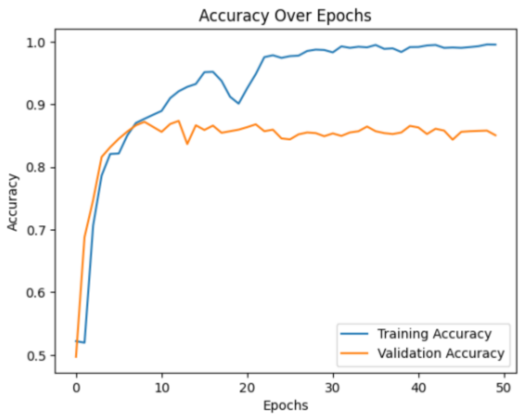
Best Model

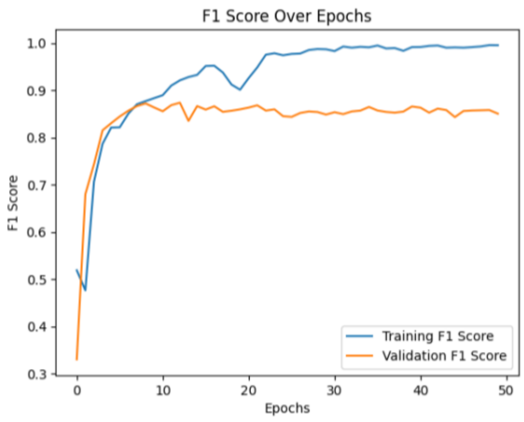


```
BiLSTM_Attention(
  (embeddings): Embedding(100002, 300)
  (bilstm): LSTM(300, 128, batch_first=True, bidirectional=True)
  (dropout_emb): Dropout(p=0.2, inplace=False)
  (dropout_att): Dropout(p=0.2, inplace=False)
  (dropout_out): Dropout(p=0.3, inplace=False)
  (deep_attention_mlp): Sequential(
    (0): Linear(in_features=256, out_features=256, bias=True)
    (1): Tanh()
    (2): Linear(in_features=256, out_features=128, bias=True)
    (3): Tanh()
    (4): Linear(in_features=128, out_features=1, bias=True)
  )
  (dense1): Linear(in_features=256, out_features=128, bias=True)
  (dense2): Linear(in_features=128, out_features=2, bias=True)
)
```

Then the classifier is trained using the best hyperparameters identified during the tuning process. The model is trained over 50 epochs using the `Utils.train_and_validate` method, which monitors performance on both the training and the validation set after each epoch. The training process is visualized in three performance plots:







Finally, we evaluate the performance of our **BiLSTM RNN Classifier** (using the best model and the optimal number of epochs according to validation loss, as shown in the plot above) against three baseline models:

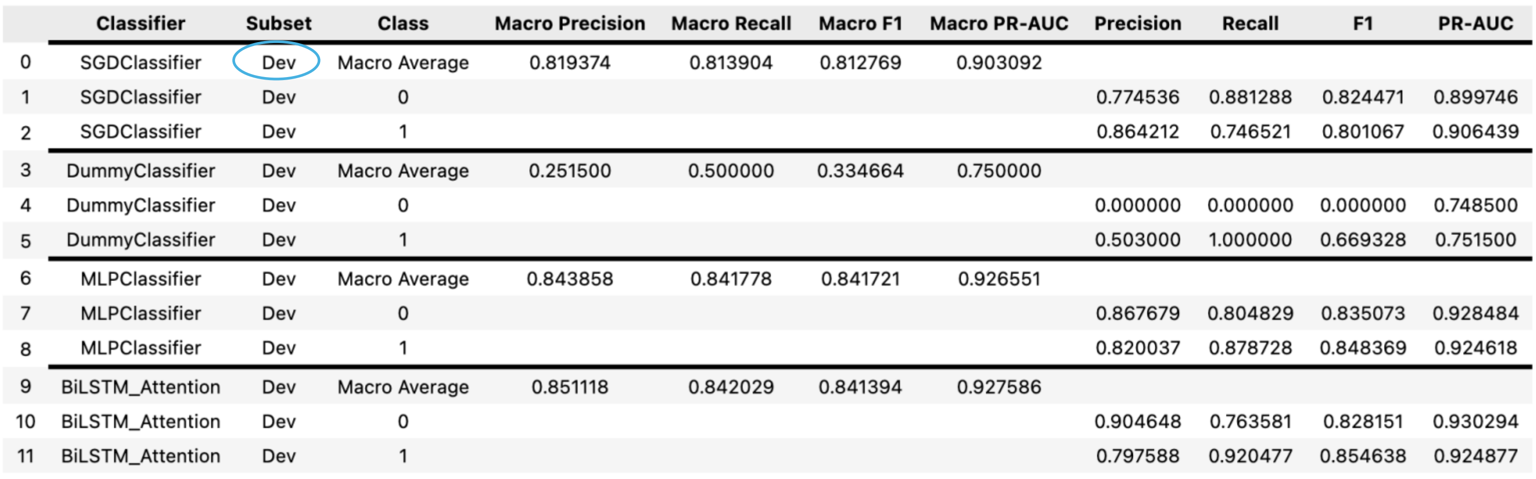
1. **MLP Classifier**

2. **Majority Classifier**: A dummy classifier from scikit-learn that always predicts the majority class observed in the training data. This serves as a simple baseline for comparison.

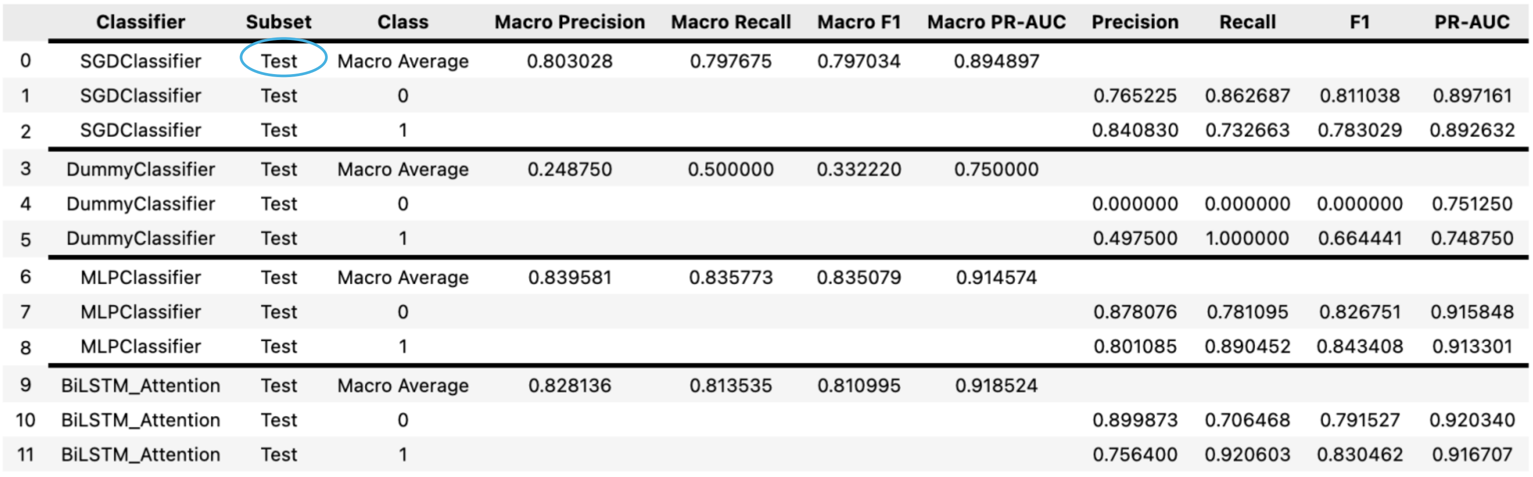
3. **Logistic Regression with SGD**: A logistic regression classifier trained using stochastic gradient descent (SGD). The hyperparameters for this classifier were tuned in a previous assignment and are directly reused here without modification.

Best Params: `{'sgd__alpha': 0.00041127525951368984, 'sgd__eta0': 0.08198575356848872, 'sgd__max_iter': 500, 'sgd__n_iter_no_change': 5, 'sgd__tol': 1e-05`

Results



	Classifier	Subset	Class	Macro Precision	Macro Recall	Macro F1	Macro PR-AUC	Precision	Recall	F1	PR-AUC
0	SGDClassifier	Dev	Macro Average	0.819374	0.813904	0.812769	0.903092				
1	SGDClassifier	Dev	0					0.774536	0.881288	0.824471	0.899746
2	SGDClassifier	Dev	1					0.864212	0.746521	0.801067	0.906439
3	DummyClassifier	Dev	Macro Average	0.251500	0.500000	0.334664	0.750000				
4	DummyClassifier	Dev	0					0.000000	0.000000	0.000000	0.748500
5	DummyClassifier	Dev	1					0.503000	1.000000	0.669328	0.751500
6	MLPClassifier	Dev	Macro Average	0.843858	0.841778	0.841721	0.926551				
7	MLPClassifier	Dev	0					0.867679	0.804829	0.835073	0.928484
8	MLPClassifier	Dev	1					0.820037	0.878728	0.848369	0.924618
9	BiLSTM_Attention	Dev	Macro Average	0.851118	0.842029	0.841394	0.927586				
10	BiLSTM_Attention	Dev	0					0.904648	0.763581	0.828151	0.930294
11	BiLSTM_Attention	Dev	1					0.797588	0.920477	0.854638	0.924877

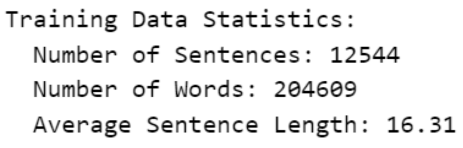


	Classifier	Subset	Class	Macro Precision	Macro Recall	Macro F1	Macro PR-AUC	Precision	Recall	F1	PR-AUC
0	SGDClassifier	Test	Macro Average	0.803028	0.797675	0.797034	0.894897				
1	SGDClassifier	Test	0					0.765225	0.862687	0.811038	0.897161
2	SGDClassifier	Test	1					0.840830	0.732663	0.783029	0.892632
3	DummyClassifier	Test	Macro Average	0.248750	0.500000	0.332220	0.750000				
4	DummyClassifier	Test	0					0.000000	0.000000	0.000000	0.751250
5	DummyClassifier	Test	1					0.497500	1.000000	0.664441	0.748750
6	MLPClassifier	Test	Macro Average	0.839581	0.835773	0.835079	0.914574				
7	MLPClassifier	Test	0					0.878076	0.781095	0.826751	0.915848
8	MLPClassifier	Test	1					0.801085	0.890452	0.843408	0.913301
9	BiLSTM_Attention	Test	Macro Average	0.828136	0.813535	0.810995	0.918524				
10	BiLSTM_Attention	Test	0					0.899873	0.706468	0.791527	0.920340
11	BiLSTM_Attention	Test	1					0.756400	0.920603	0.830462	0.916707

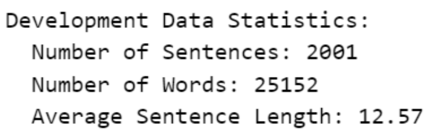
2. Exercise 2 (POS Tagging using an BiLSTM RNN with "deep" self-attention)

In this exercise, we developed a part-of-speech (POS) tagger for the **English EWT** dataset from the Universal Dependencies treebanks. The objective was to create a system that accurately assigns POS tags to words in sentences, using only the words, sentences, and POS tags available in the dataset. Additional annotations, such as syntactic dependencies, were not included in this task.

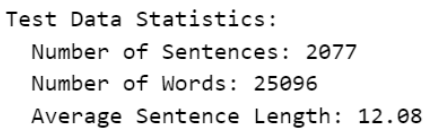
Below there are some statistics for the dataset:



```
Training Data Statistics:
  Number of Sentences: 12544
  Number of Words: 204609
  Average Sentence Length: 16.31
```



```
Development Data Statistics:
  Number of Sentences: 2001
  Number of Words: 25152
  Average Sentence Length: 12.57
```



```
Test Data Statistics:
  Number of Sentences: 2077
  Number of Words: 25096
  Average Sentence Length: 12.08
```

UtilsPOSTagging Class:

The UtilsPOSTagging class provides utility methods for preprocessing and feature engineering in part-of-speech (POS) tagging tasks.

- `parse_conllu(path)`:

Parses .conllu files to extract sentences as lists of (word, POS tag) tuples. Each word is converted to lowercase, and sentences without valid word-POS pairs are excluded.

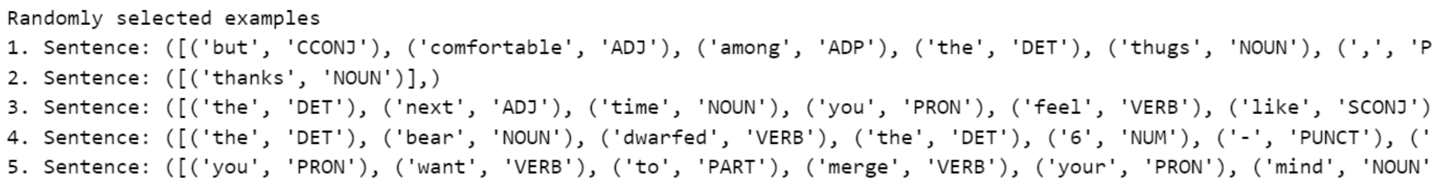
- `calculate_statistics(tagged_sentences, title)`:
  Computes and displays dataset statistics, including the number of sentences, total words, and average sentence length. This method provides a quick overview of the dataset's characteristics.
- `sliding_window(sentences_as_words, sentences_as_tags, window_size=3, embedding_dim=300)`:
  This method constructs feature vectors and labels for training a POS tagger using a Multi-Layer Perceptron (MLP). It uses a sliding window approach to incorporate contextual information around each word in a sentence. Each window consists of a target word and its neighboring words, ensuring that the model can learn patterns based on context.
  To handle words near the beginning or end of a sentence, the method pads the input with zero vectors for the embeddings and placeholders for the POS tags. This padding ensures that each word, including those at the boundaries, has a complete window of neighbors. The size of the padding depends on the sliding window size, with padding added symmetrically to both ends of the sentence.
  The method then iterates through each word in the padded sentence, extracting a window of embeddings centered on the current word. The embeddings from the window are concatenated into a single flattened vector, which serves as a feature for the model. The corresponding POS tag of the center word is extracted as the label. By iterating through all the words in the dataset, the method generates a set of features and their associated labels, ready for training the MLP.
- `get_word_embedding(word, model, embedding_dim=300):`
  Retrieves the embedding vector for a given word from a pre-trained model. If the word is not found in the model's vocabulary, it returns a zero vector of the specified embedding dimension.
- `tags(sentences):`
  Extracts POS tags from a list of sentences. Each sentence is represented as a list of tags.
- `words(sentences):`
  Extracts words from a list of sentences. Each sentence is represented as a list of words.
- `map_tags_to_numbers(tags, tag_to_int):`
  Converts a list of tags into their corresponding numeric representations based on a provided mapping (tag_to_int). This is essential for preparing labels for machine learning models.

The BiLSTM_Attention is identical to that implemented in Exercise 1. The UtilsBILSTM class functions similarly to UtilsMLP, appropriately configured for a token classification problem. Lastly, the BiLSTMAdapter class provides a training and evaluation framework the BiLSTM RNN model. It integrates the training process, prediction of class labels, and prediction of class probabilities.

MostFrequentTagBaseline Class: It is a baseline model for POS tagging that predicts tags based on their most frequent occurrence per feature or overall. It calculates tag probabilities during training and predicts either the most frequent tag for a feature or the overall most frequent tag when the feature is unseen.

- `fit`: Learns the most frequent tag per feature and overall tag probabilities.
- `predict`: Predicts tags using the learned frequencies.
- `predict_proba`: Outputs tag probabilities.

To begin with, the .conllu files are loaded and parsed using the `pyconll` library to extract useful linguistic annotations (e.g., word forms and POS tags). More specifically, the parsing function (`UtilsPOSTagging.parse_conllu`) is called for each dataset (training, development, and test).



```
Randomly selected examples
1. Sentence: ([('but', 'CCONJ'), ('comfortable', 'ADJ'), ('among', 'ADP'), ('the', 'DET'), ('thugs', 'NOUN'), (',', 'P
2. Sentence: ([('thanks', 'NOUN')],)
3. Sentence: ([('the', 'DET'), ('next', 'ADJ'), ('time', 'NOUN'), ('you', 'PRON'), ('feel', 'VERB'), ('like', 'SCONJ')
4. Sentence: ([('the', 'DET'), ('bear', 'NOUN'), ('dwarfed', 'VERB'), ('the', 'DET'), ('6', 'NUM'), ('-', 'PUNCT'), ('
5. Sentence: ([('you', 'PRON'), ('want', 'VERB'), ('to', 'PART'), ('merge', 'VERB'), ('your', 'PRON'), ('mind', 'NOUN'
```

Preparing Features and Labels (Tags)

We prepare data for training a Bidirectional Long Short-Term Memory (BILSTM) model to predict Part-of-Speech (POS) tags using sliding window word embeddings.

We use the pre-trained `word2vec-google-news-300` model from Gensim's API to obtain 300-dimensional word embeddings for the words in our dataset.

1. **Vocabulary Creation**

Word Vocabulary: Extract all unique words from the training dataset to create a `word2idx` mapping. Special tokens include: UNK (unknown words) and PAD (padding).

Tag Vocabulary: Extract all unique POS tags from the training tag sequences to create a `tag2idx` mapping.

Reverse Tag Mapping: A reverse mapping (`idx2tag`) is created to decode tag indices back into their corresponding tags.

2. **Encoding Sequences**

In this step, word and tag sequences are converted into numerical indices using the `word2id` and `tag2idx` mappings. For words not present in the vocabulary, the UNK token index is used. This step ensures that the text data is compatible with the numerical computations of the model.

3. **Padding Sequences**

Since sentences in the dataset vary in length, padding is applied to make all sequences equal in length. The PAD token is used for both words and tags. A fixed maximum length (`max_len`) is defined to standardize the sequence lengths. Sequences longer than `max_len` are truncated, while shorter ones are padded.

4. **One-Hot Encoding of Tags**

The padded tag sequences are converted to a categorical format (one-hot encoded) to match the output requirements of the model. Each tag is represented as a vector with a length equal to the total number of unique tags.

Experiments:

The primary objective of our experiments is to assess and compare the performance of our BiLSTM RNN POS Tagger against two baselines:

1. Best MLP Model: After conducting hyperparameter tuning on the MLP model, we use the configuration that achieved the best performance on the validation set. This model utilizes sliding window word embeddings.

2. Baseline Classifier: As a reference point, we implement a baseline classifier that operates as follows:

- For each word in the test set, it assigns the **most frequent tag** that the word had in the training data.
- For words not encountered during training (unseen words), the classifier assigns the **most frequent tag overall** (across all words) from the training data.

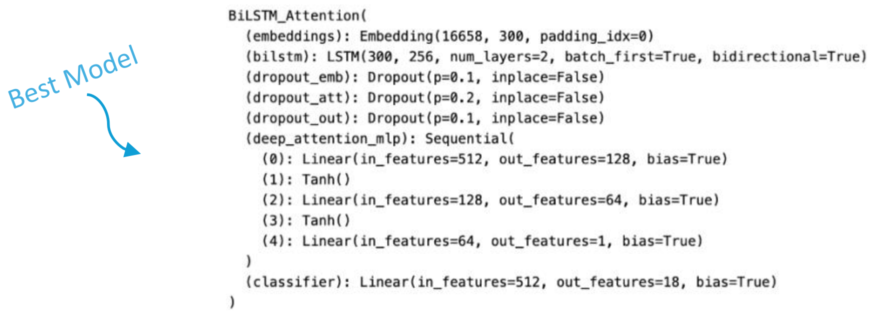
First, we configure the model with initial, arbitrarily chosen hyperparameters. This initial testing allows us to observe baseline performance and helps us identify potential adjustments for further tuning. Subsequent steps involve tuning these hyperparameters for optimal performance.

Hyperparameter Tuning for BiLSTM RNN POS Tagger:

Hyperparameters tuned

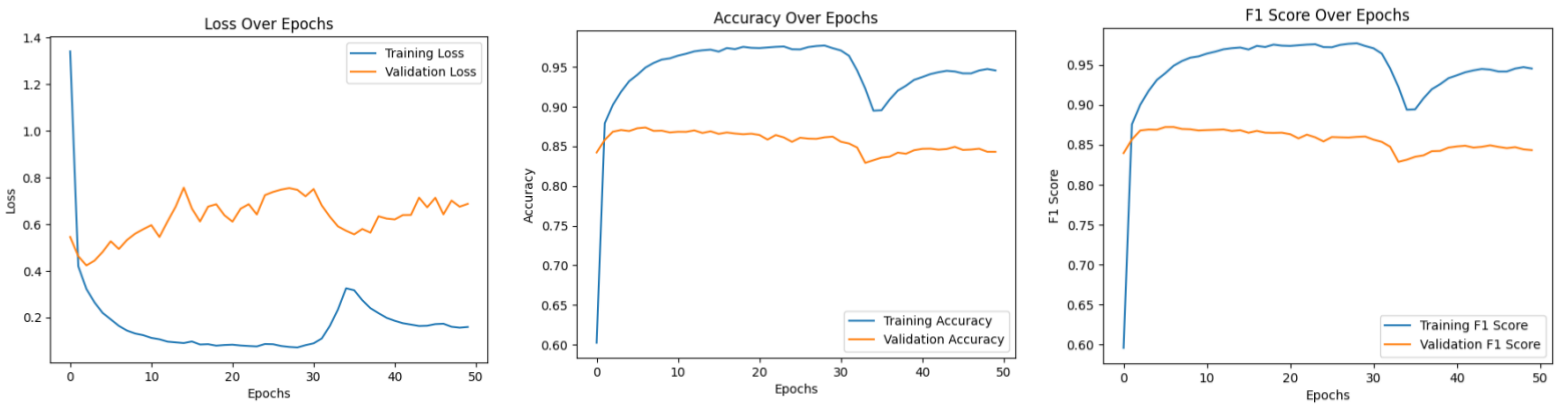
- **LSTM Hidden Layer Size (**`lstm_hidden_dim`**)**: Represents the number of hidden units in the LSTM layers.
- **Dropout Probabilities**
  - **Embeddings (**`dropout_prob_emb`**)**: Dropout applied after the embedding layer.
  - **Attention Layer (**`dropout_prob_att`**)**: Applied in the attention mechanism.
  - **Output Layer (**`dropout_prob_out`**)**: Applied before the final prediction.
- **Attention MLP Hidden Sizes (**`attention_hidden_sizes`**)**: Defines the architecture of the MLP used in the attention mechanism.
- **Learning Rate (**`learning_rate`**):** Controls the step size for weight updates during optimization.
- **Number of LSTM Layers (**`lstm_stacks`**):** Specifies the number of stacked LSTM layers in the model.

Tuning Procedure: A grid of all possible combinations of the hyperparameters is generated. Each combination is tested to evaluate its performance on the development dataset. The macro F1 score on the development set is used as the primary metric to assess model performance. The combination of hyperparameters that yields the highest macro F1 score on the development dataset is selected as the best configuration.

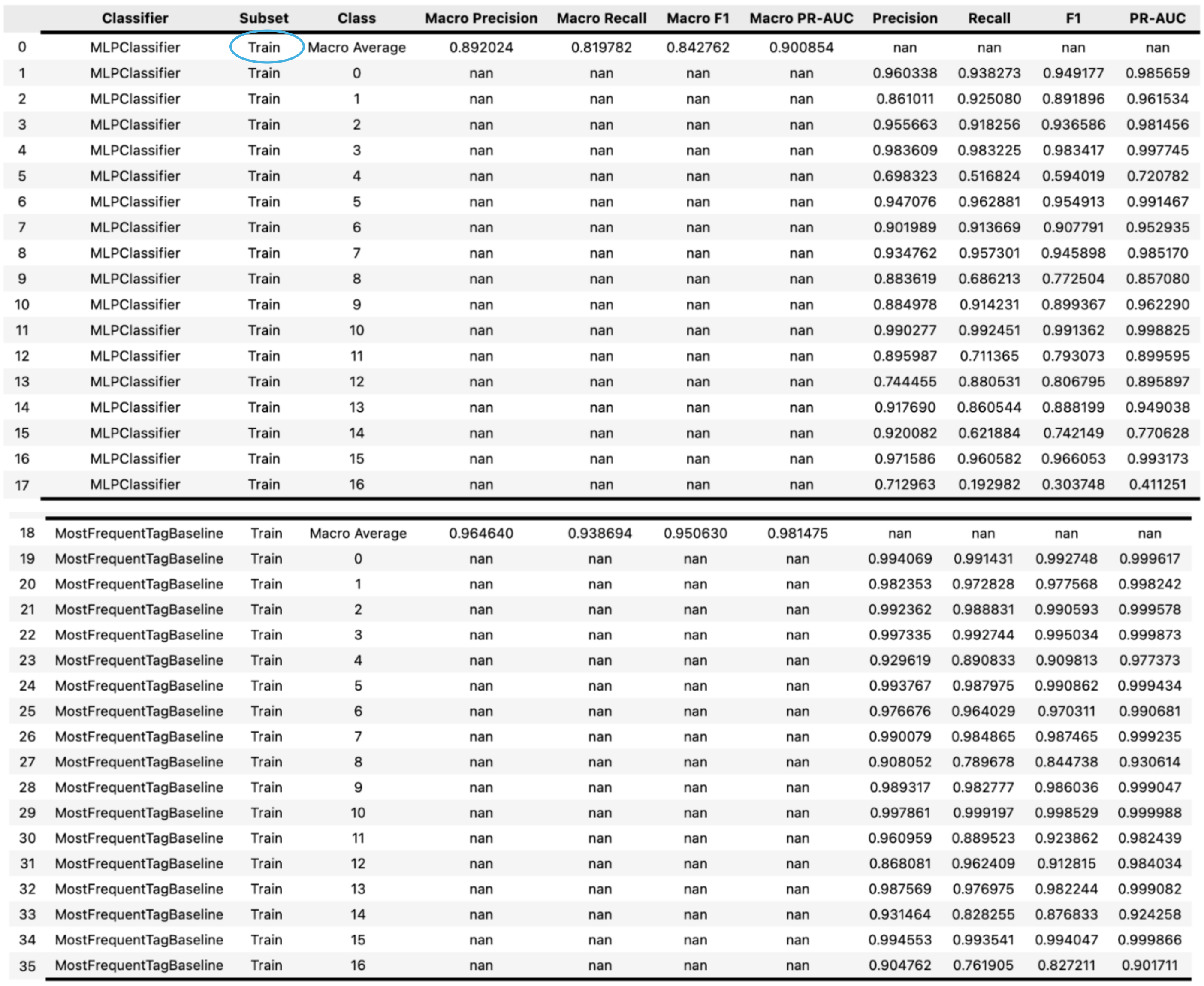


```
BiLSTM_Attention(
  (embeddings): Embedding(16658, 300, padding_idx=0)
  (bilstm): LSTM(300, 256, num_layers=2, batch_first=True, bidirectional=True)
  (dropout_emb): Dropout(p=0.1, inplace=False)
  (dropout_att): Dropout(p=0.2, inplace=False)
  (dropout_out): Dropout(p=0.1, inplace=False)
  (deep_attention_mlp): Sequential(
    (0): Linear(in_features=512, out_features=128, bias=True)
    (1): Tanh()
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): Tanh()
    (4): Linear(in_features=64, out_features=1, bias=True)
  )
  (classifier): Linear(in_features=512, out_features=18, bias=True)
)
```

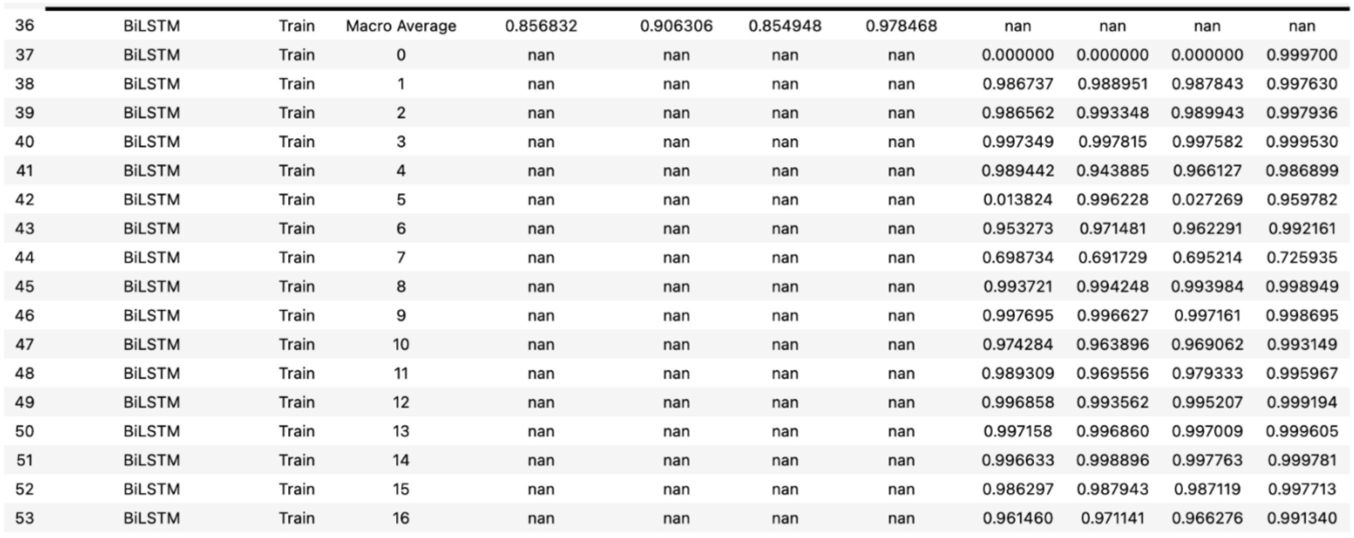
Then the MLP classifier is trained using the best hyperparameters identified during the tuning process. The model is trained over 50 epochs. The training process is visualized in three performance plots:



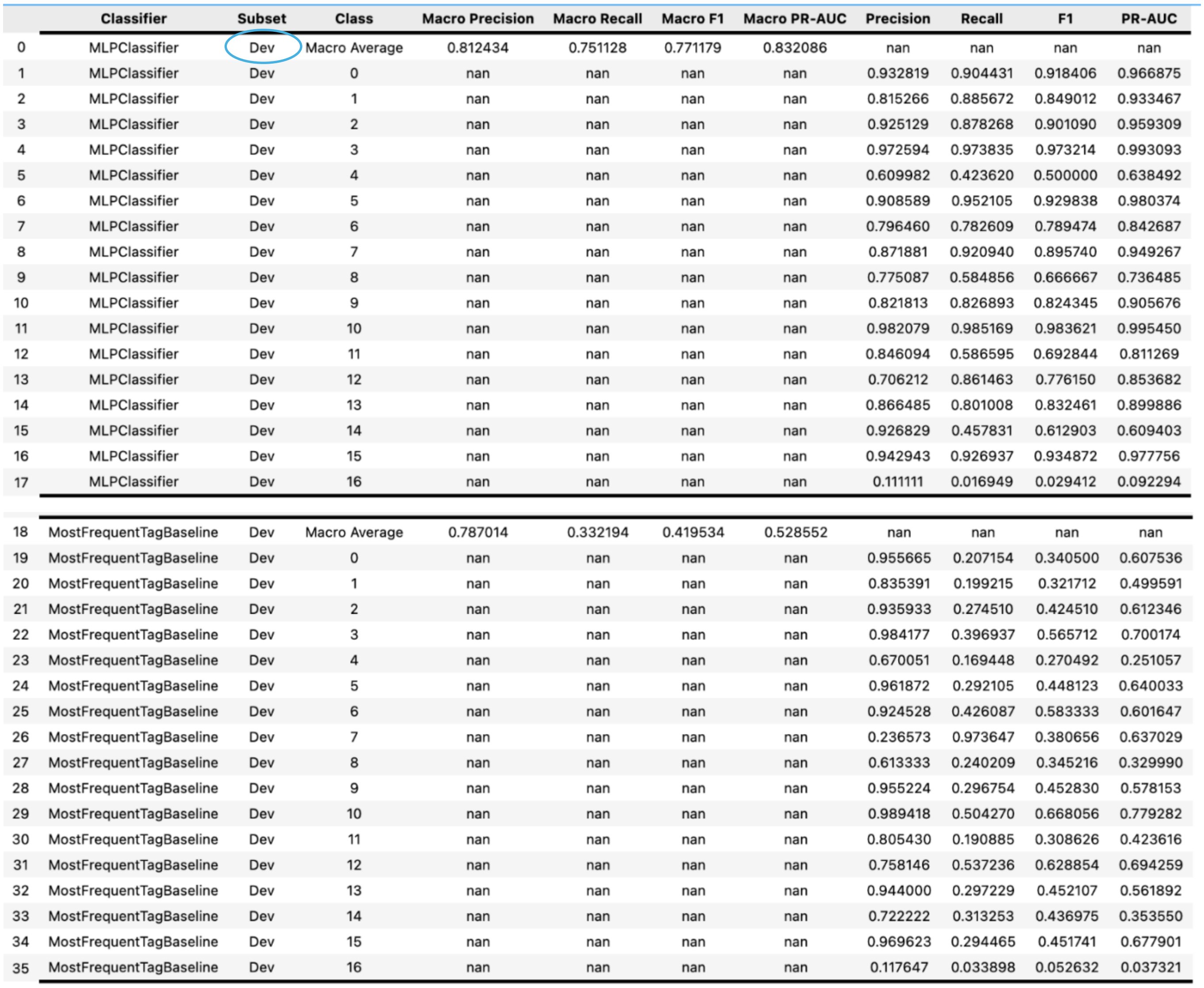
Lastly, the performance of our **BiLSTM RNN POS Tagger** (using the best model and the optimal number of epochs according to validation loss, as shown in the plot above) is evaluated against the baselines.



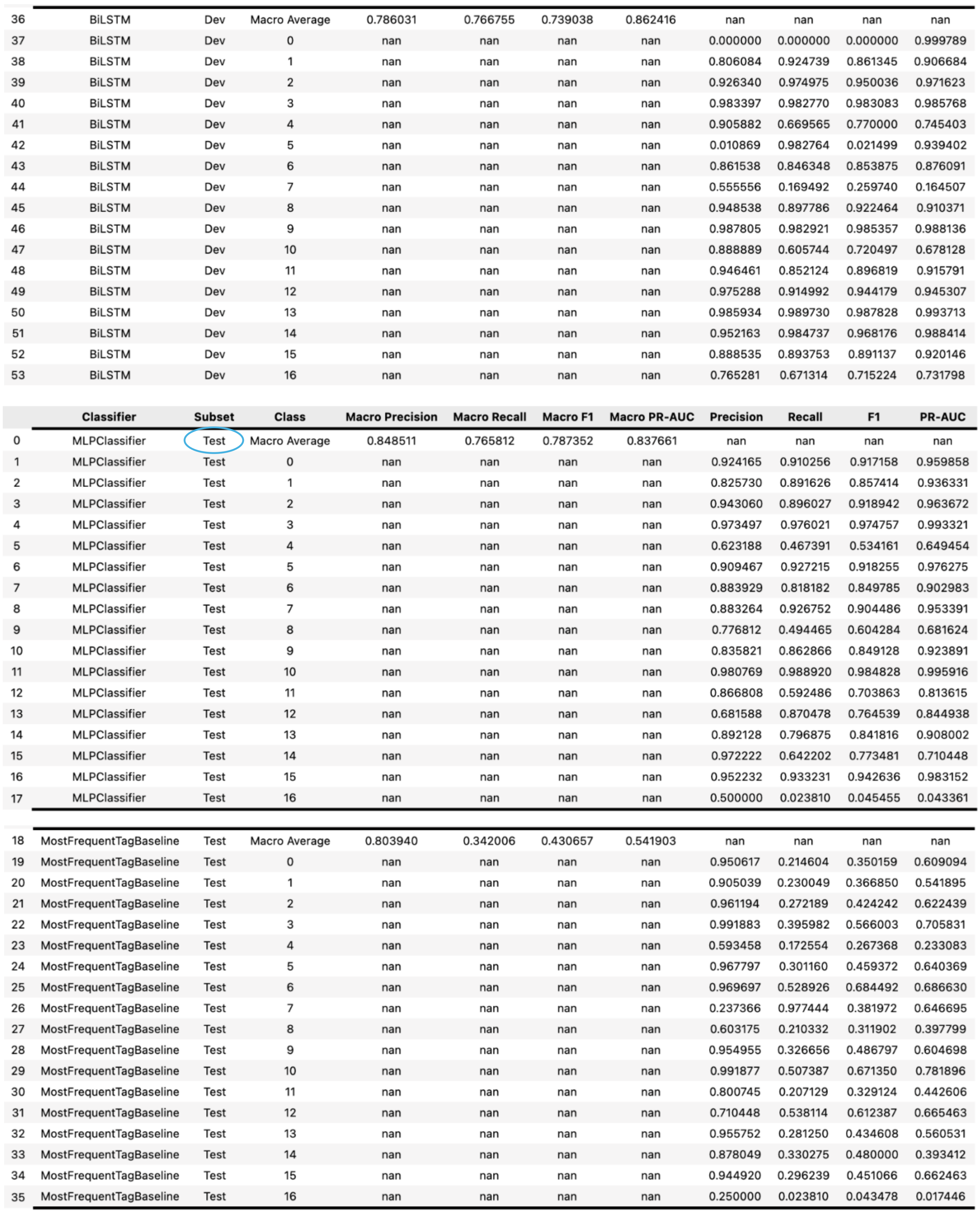
	Classifier	Subset	Class	Macro Precision	Macro Recall	Macro F1	Macro PR-AUC	Precision	Recall	F1	PR-AUC
0	MLPClassifier	Train	Macro Average	0.892024	0.819782	0.842762	0.900854	nan	nan	nan	nan
1	MLPClassifier	Train	0	nan	nan	nan	nan	0.960338	0.938273	0.949177	0.985659
2	MLPClassifier	Train	1	nan	nan	nan	nan	0.861011	0.925080	0.891896	0.961534
3	MLPClassifier	Train	2	nan	nan	nan	nan	0.955663	0.918256	0.936586	0.981456
4	MLPClassifier	Train	3	nan	nan	nan	nan	0.983609	0.983225	0.983417	0.997745
5	MLPClassifier	Train	4	nan	nan	nan	nan	0.698323	0.516824	0.594019	0.720782
6	MLPClassifier	Train	5	nan	nan	nan	nan	0.947076	0.962881	0.954913	0.991467
7	MLPClassifier	Train	6	nan	nan	nan	nan	0.901989	0.913669	0.907791	0.952935
8	MLPClassifier	Train	7	nan	nan	nan	nan	0.934762	0.957301	0.945898	0.985170
9	MLPClassifier	Train	8	nan	nan	nan	nan	0.883619	0.686213	0.772504	0.857080
10	MLPClassifier	Train	9	nan	nan	nan	nan	0.884978	0.914231	0.899367	0.962290
11	MLPClassifier	Train	10	nan	nan	nan	nan	0.990277	0.992451	0.991362	0.998825
12	MLPClassifier	Train	11	nan	nan	nan	nan	0.895987	0.711365	0.793073	0.899595
13	MLPClassifier	Train	12	nan	nan	nan	nan	0.744455	0.880531	0.806795	0.895897
14	MLPClassifier	Train	13	nan	nan	nan	nan	0.917690	0.860544	0.888199	0.949038
15	MLPClassifier	Train	14	nan	nan	nan	nan	0.920082	0.621884	0.742149	0.770628
16	MLPClassifier	Train	15	nan	nan	nan	nan	0.971586	0.960582	0.966053	0.993173
17	MLPClassifier	Train	16	nan	nan	nan	nan	0.712963	0.192982	0.303748	0.411251
18	MostFrequentTagBaseline	Train	Macro Average	0.964640	0.938694	0.950630	0.981475	nan	nan	nan	nan
19	MostFrequentTagBaseline	Train	0	nan	nan	nan	nan	0.994069	0.991431	0.992748	0.999617
20	MostFrequentTagBaseline	Train	1	nan	nan	nan	nan	0.982353	0.972828	0.977568	0.998242
21	MostFrequentTagBaseline	Train	2	nan	nan	nan	nan	0.992362	0.988831	0.990593	0.999578
22	MostFrequentTagBaseline	Train	3	nan	nan	nan	nan	0.997335	0.992744	0.995034	0.999873
23	MostFrequentTagBaseline	Train	4	nan	nan	nan	nan	0.929619	0.890833	0.909813	0.977373
24	MostFrequentTagBaseline	Train	5	nan	nan	nan	nan	0.993767	0.987975	0.990862	0.999434
25	MostFrequentTagBaseline	Train	6	nan	nan	nan	nan	0.976676	0.964029	0.970311	0.990681
26	MostFrequentTagBaseline	Train	7	nan	nan	nan	nan	0.990079	0.984865	0.987465	0.999235
27	MostFrequentTagBaseline	Train	8	nan	nan	nan	nan	0.908052	0.789678	0.844738	0.930614
28	MostFrequentTagBaseline	Train	9	nan	nan	nan	nan	0.989317	0.982777	0.986036	0.999047
29	MostFrequentTagBaseline	Train	10	nan	nan	nan	nan	0.997861	0.999197	0.998529	0.999988
30	MostFrequentTagBaseline	Train	11	nan	nan	nan	nan	0.960959	0.889523	0.923862	0.982439
31	MostFrequentTagBaseline	Train	12	nan	nan	nan	nan	0.868081	0.962409	0.912815	0.984034
32	MostFrequentTagBaseline	Train	13	nan	nan	nan	nan	0.987569	0.976975	0.982244	0.999082
33	MostFrequentTagBaseline	Train	14	nan	nan	nan	nan	0.931464	0.828255	0.876833	0.924258
34	MostFrequentTagBaseline	Train	15	nan	nan	nan	nan	0.994553	0.993541	0.994047	0.999866
35	MostFrequentTagBaseline	Train	16	nan	nan	nan	nan	0.904762	0.761905	0.827211	0.901711



36	BiLSTM	Train	Macro Average	0.856832	0.906306	0.854948	0.978468	nan	nan	nan	nan
37	BiLSTM	Train	0	nan	nan	nan	nan	0.000000	0.000000	0.000000	0.999700
38	BiLSTM	Train	1	nan	nan	nan	nan	0.986737	0.988951	0.987843	0.997630
39	BiLSTM	Train	2	nan	nan	nan	nan	0.986562	0.993348	0.989943	0.997936
40	BiLSTM	Train	3	nan	nan	nan	nan	0.997349	0.997815	0.997582	0.999530
41	BiLSTM	Train	4	nan	nan	nan	nan	0.989442	0.943885	0.966127	0.986899
42	BiLSTM	Train	5	nan	nan	nan	nan	0.013824	0.996228	0.027269	0.959782
43	BiLSTM	Train	6	nan	nan	nan	nan	0.953273	0.971481	0.962291	0.992161
44	BiLSTM	Train	7	nan	nan	nan	nan	0.698734	0.691729	0.695214	0.725935
45	BiLSTM	Train	8	nan	nan	nan	nan	0.993721	0.994248	0.993984	0.998949
46	BiLSTM	Train	9	nan	nan	nan	nan	0.997695	0.996627	0.997161	0.998695
47	BiLSTM	Train	10	nan	nan	nan	nan	0.974284	0.963896	0.969062	0.993149
48	BiLSTM	Train	11	nan	nan	nan	nan	0.989309	0.969556	0.979333	0.995967
49	BiLSTM	Train	12	nan	nan	nan	nan	0.996858	0.993562	0.995207	0.999194
50	BiLSTM	Train	13	nan	nan	nan	nan	0.997158	0.996860	0.997009	0.999605
51	BiLSTM	Train	14	nan	nan	nan	nan	0.996633	0.998896	0.997763	0.999781
52	BiLSTM	Train	15	nan	nan	nan	nan	0.986297	0.987943	0.987119	0.997713
53	BiLSTM	Train	16	nan	nan	nan	nan	0.961460	0.971141	0.966276	0.991340



	Classifier	Subset	Class	Macro Precision	Macro Recall	Macro F1	Macro PR-AUC	Precision	Recall	F1	PR-AUC
0	MLPClassifier	Dev	Macro Average	0.812434	0.751128	0.771179	0.832086	nan	nan	nan	nan
1	MLPClassifier	Dev	0	nan	nan	nan	nan	0.932819	0.904431	0.918406	0.966875
2	MLPClassifier	Dev	1	nan	nan	nan	nan	0.815266	0.885672	0.849012	0.933467
3	MLPClassifier	Dev	2	nan	nan	nan	nan	0.925129	0.878268	0.901090	0.959309
4	MLPClassifier	Dev	3	nan	nan	nan	nan	0.972594	0.973835	0.973214	0.993093
5	MLPClassifier	Dev	4	nan	nan	nan	nan	0.609982	0.423620	0.500000	0.638492
6	MLPClassifier	Dev	5	nan	nan	nan	nan	0.908589	0.952105	0.929838	0.980374
7	MLPClassifier	Dev	6	nan	nan	nan	nan	0.796460	0.782609	0.789474	0.842687
8	MLPClassifier	Dev	7	nan	nan	nan	nan	0.871881	0.920940	0.895740	0.949267
9	MLPClassifier	Dev	8	nan	nan	nan	nan	0.775087	0.584856	0.666667	0.736485
10	MLPClassifier	Dev	9	nan	nan	nan	nan	0.821813	0.826893	0.824345	0.905676
11	MLPClassifier	Dev	10	nan	nan	nan	nan	0.982079	0.985169	0.983621	0.995450
12	MLPClassifier	Dev	11	nan	nan	nan	nan	0.846094	0.586595	0.692844	0.811269
13	MLPClassifier	Dev	12	nan	nan	nan	nan	0.706212	0.861463	0.776150	0.853682
14	MLPClassifier	Dev	13	nan	nan	nan	nan	0.866485	0.801008	0.832461	0.899886
15	MLPClassifier	Dev	14	nan	nan	nan	nan	0.926829	0.457831	0.612903	0.609403
16	MLPClassifier	Dev	15	nan	nan	nan	nan	0.942943	0.926937	0.934872	0.977756
17	MLPClassifier	Dev	16	nan	nan	nan	nan	0.111111	0.016949	0.029412	0.092294
18	MostFrequentTagBaseline	Dev	Macro Average	0.787014	0.332194	0.419534	0.528552	nan	nan	nan	nan
19	MostFrequentTagBaseline	Dev	0	nan	nan	nan	nan	0.955665	0.207154	0.340500	0.607536
20	MostFrequentTagBaseline	Dev	1	nan	nan	nan	nan	0.835391	0.199215	0.321712	0.499591
21	MostFrequentTagBaseline	Dev	2	nan	nan	nan	nan	0.935933	0.274510	0.424510	0.612346
22	MostFrequentTagBaseline	Dev	3	nan	nan	nan	nan	0.984177	0.396937	0.565712	0.700174
23	MostFrequentTagBaseline	Dev	4	nan	nan	nan	nan	0.670051	0.169448	0.270492	0.251057
24	MostFrequentTagBaseline	Dev	5	nan	nan	nan	nan	0.961872	0.292105	0.448123	0.640033
25	MostFrequentTagBaseline	Dev	6	nan	nan	nan	nan	0.924528	0.426087	0.583333	0.601647
26	MostFrequentTagBaseline	Dev	7	nan	nan	nan	nan	0.236573	0.973647	0.380656	0.637029
27	MostFrequentTagBaseline	Dev	8	nan	nan	nan	nan	0.613333	0.240209	0.345216	0.329990
28	MostFrequentTagBaseline	Dev	9	nan	nan	nan	nan	0.955224	0.296754	0.452830	0.578153
29	MostFrequentTagBaseline	Dev	10	nan	nan	nan	nan	0.989418	0.504270	0.668056	0.779282
30	MostFrequentTagBaseline	Dev	11	nan	nan	nan	nan	0.805430	0.190885	0.308626	0.423616
31	MostFrequentTagBaseline	Dev	12	nan	nan	nan	nan	0.758146	0.537236	0.628854	0.694259
32	MostFrequentTagBaseline	Dev	13	nan	nan	nan	nan	0.944000	0.297229	0.452107	0.561892
33	MostFrequentTagBaseline	Dev	14	nan	nan	nan	nan	0.722222	0.313253	0.436975	0.353550
34	MostFrequentTagBaseline	Dev	15	nan	nan	nan	nan	0.969623	0.294465	0.451741	0.677901
35	MostFrequentTagBaseline	Dev	16	nan	nan	nan	nan	0.117647	0.033898	0.052632	0.037321



36	BiLSTM	Dev	Macro Average	0.786031	0.766755	0.739038	0.862416	nan	nan	nan	nan
37	BiLSTM	Dev	0	nan	nan	nan	nan	0.000000	0.000000	0.000000	0.999789
38	BiLSTM	Dev	1	nan	nan	nan	nan	0.806084	0.924739	0.861345	0.906684
39	BiLSTM	Dev	2	nan	nan	nan	nan	0.926340	0.974975	0.950036	0.971623
40	BiLSTM	Dev	3	nan	nan	nan	nan	0.983397	0.982770	0.983083	0.985768
41	BiLSTM	Dev	4	nan	nan	nan	nan	0.905882	0.669565	0.770000	0.745403
42	BiLSTM	Dev	5	nan	nan	nan	nan	0.010869	0.982764	0.021499	0.939402
43	BiLSTM	Dev	6	nan	nan	nan	nan	0.861538	0.846348	0.853875	0.876091
44	BiLSTM	Dev	7	nan	nan	nan	nan	0.555556	0.169492	0.259740	0.164507
45	BiLSTM	Dev	8	nan	nan	nan	nan	0.948538	0.897786	0.922464	0.910371
46	BiLSTM	Dev	9	nan	nan	nan	nan	0.987805	0.982921	0.985357	0.988136
47	BiLSTM	Dev	10	nan	nan	nan	nan	0.888889	0.605744	0.720497	0.678128
48	BiLSTM	Dev	11	nan	nan	nan	nan	0.946461	0.852124	0.896819	0.915791
49	BiLSTM	Dev	12	nan	nan	nan	nan	0.975288	0.914992	0.944179	0.945307
50	BiLSTM	Dev	13	nan	nan	nan	nan	0.985934	0.989730	0.987828	0.993713
51	BiLSTM	Dev	14	nan	nan	nan	nan	0.952163	0.984737	0.968176	0.988414
52	BiLSTM	Dev	15	nan	nan	nan	nan	0.888535	0.893753	0.891137	0.920146
53	BiLSTM	Dev	16	nan	nan	nan	nan	0.765281	0.671314	0.715224	0.731798

	Classifier	Subset	Class	Macro Precision	Macro Recall	Macro F1	Macro PR-AUC	Precision	Recall	F1	PR-AUC
0	MLPClassifier	Test	Macro Average	0.848511	0.765812	0.787352	0.837661	nan	nan	nan	nan
1	MLPClassifier	Test	0	nan	nan	nan	nan	0.924165	0.910256	0.917158	0.959858
2	MLPClassifier	Test	1	nan	nan	nan	nan	0.825730	0.891626	0.857414	0.936331
3	MLPClassifier	Test	2	nan	nan	nan	nan	0.943060	0.896027	0.918942	0.963672
4	MLPClassifier	Test	3	nan	nan	nan	nan	0.973497	0.976021	0.974757	0.993321
5	MLPClassifier	Test	4	nan	nan	nan	nan	0.623188	0.467391	0.534161	0.649454
6	MLPClassifier	Test	5	nan	nan	nan	nan	0.909467	0.927215	0.918255	0.976275
7	MLPClassifier	Test	6	nan	nan	nan	nan	0.883929	0.818182	0.849785	0.902983
8	MLPClassifier	Test	7	nan	nan	nan	nan	0.883264	0.926752	0.904486	0.953391
9	MLPClassifier	Test	8	nan	nan	nan	nan	0.776812	0.494465	0.604284	0.681624
10	MLPClassifier	Test	9	nan	nan	nan	nan	0.835821	0.862866	0.849128	0.923891
11	MLPClassifier	Test	10	nan	nan	nan	nan	0.980769	0.988920	0.984828	0.995916
12	MLPClassifier	Test	11	nan	nan	nan	nan	0.866808	0.592486	0.703863	0.813615
13	MLPClassifier	Test	12	nan	nan	nan	nan	0.681588	0.870478	0.764539	0.844938
14	MLPClassifier	Test	13	nan	nan	nan	nan	0.892128	0.796875	0.841816	0.908002
15	MLPClassifier	Test	14	nan	nan	nan	nan	0.972222	0.642202	0.773481	0.710448
16	MLPClassifier	Test	15	nan	nan	nan	nan	0.952232	0.933231	0.942636	0.983152
17	MLPClassifier	Test	16	nan	nan	nan	nan	0.500000	0.023810	0.045455	0.043361
18	MostFrequentTagBaseline	Test	Macro Average	0.803940	0.342006	0.430657	0.541903	nan	nan	nan	nan
19	MostFrequentTagBaseline	Test	0	nan	nan	nan	nan	0.950617	0.214604	0.350159	0.609094
20	MostFrequentTagBaseline	Test	1	nan	nan	nan	nan	0.905039	0.230049	0.366850	0.541895
21	MostFrequentTagBaseline	Test	2	nan	nan	nan	nan	0.961194	0.272189	0.424242	0.622439
22	MostFrequentTagBaseline	Test	3	nan	nan	nan	nan	0.991883	0.395982	0.566003	0.705831
23	MostFrequentTagBaseline	Test	4	nan	nan	nan	nan	0.593458	0.172554	0.267368	0.233083
24	MostFrequentTagBaseline	Test	5	nan	nan	nan	nan	0.967797	0.301160	0.459372	0.640369
25	MostFrequentTagBaseline	Test	6	nan	nan	nan	nan	0.969697	0.528926	0.684492	0.686630
26	MostFrequentTagBaseline	Test	7	nan	nan	nan	nan	0.237366	0.977444	0.381972	0.646695
27	MostFrequentTagBaseline	Test	8	nan	nan	nan	nan	0.603175	0.210332	0.311902	0.397799
28	MostFrequentTagBaseline	Test	9	nan	nan	nan	nan	0.954955	0.326656	0.486797	0.604698
29	MostFrequentTagBaseline	Test	10	nan	nan	nan	nan	0.991877	0.507387	0.671350	0.781896
30	MostFrequentTagBaseline	Test	11	nan	nan	nan	nan	0.800745	0.207129	0.329124	0.442606
31	MostFrequentTagBaseline	Test	12	nan	nan	nan	nan	0.710448	0.538114	0.612387	0.665463
32	MostFrequentTagBaseline	Test	13	nan	nan	nan	nan	0.955752	0.281250	0.434608	0.560531
33	MostFrequentTagBaseline	Test	14	nan	nan	nan	nan	0.878049	0.330275	0.480000	0.393412
34	MostFrequentTagBaseline	Test	15	nan	nan	nan	nan	0.944920	0.296239	0.451066	0.662463
35	MostFrequentTagBaseline	Test	16	nan	nan	nan	nan	0.250000	0.023810	0.043478	0.017446

36	BiLSTM	Test	Macro Average	0.756986	0.772419	0.734218	0.856256	nan	nan	nan	nan
37	BiLSTM	Test	0	nan	nan	nan	nan	0.000000	0.000000	0.000000	0.999767
38	BiLSTM	Test	1	nan	nan	nan	nan	0.792340	0.928208	0.854909	0.909961
39	BiLSTM	Test	2	nan	nan	nan	nan	0.946860	0.965517	0.956098	0.973605
40	BiLSTM	Test	3	nan	nan	nan	nan	0.985046	0.981854	0.983447	0.984585
41	BiLSTM	Test	4	nan	nan	nan	nan	0.969697	0.793388	0.872727	0.845547
42	BiLSTM	Test	5	nan	nan	nan	nan	0.010480	0.979974	0.020738	0.933735
43	BiLSTM	Test	6	nan	nan	nan	nan	0.884718	0.859375	0.871863	0.905067
44	BiLSTM	Test	7	nan	nan	nan	nan	0.000000	0.000000	0.000000	0.000769
45	BiLSTM	Test	8	nan	nan	nan	nan	0.956346	0.907905	0.931496	0.918364
46	BiLSTM	Test	9	nan	nan	nan	nan	0.983380	0.983380	0.983380	0.988621
47	BiLSTM	Test	10	nan	nan	nan	nan	0.852459	0.479705	0.613932	0.577122
48	BiLSTM	Test	11	nan	nan	nan	nan	0.930335	0.891801	0.910660	0.937003
49	BiLSTM	Test	12	nan	nan	nan	nan	0.976266	0.950693	0.963310	0.961714
50	BiLSTM	Test	13	nan	nan	nan	nan	0.987805	0.990489	0.989145	0.995577
51	BiLSTM	Test	14	nan	nan	nan	nan	0.957121	0.988924	0.972763	0.991895
52	BiLSTM	Test	15	nan	nan	nan	nan	0.897982	0.892977	0.895472	0.919696
53	BiLSTM	Test	16	nan	nan	nan	nan	0.758073	0.667148	0.709710	0.713327