Machine Learning 2024 – Mini Project 2

Introduction

The purpose of this project was to analyze a dataset of tweets with a label for the sentiment of each tweet. The goal was to produce two machine learning models trained on the data that can predict the sentiment class.

The sentiment in the dataset is classified as either "0" or "4" so for this problem algorithms that can handle binary classification will be used.

The project is executed in a python Jupyter notebook, using libraries sklearn, NLTK, LightGBM, Tensorflow/Keras among others for the algorithms and various tools for data processing.

Data processing

Before prediction can be done on the data steps need to be taken to process the data into a usable format. First step is loading the data and exploring it to see what issues may need to be addressed before proceeding to training a model.

Data load

The data was provided in a .tsv format. For reading this data into the Jupyter notebook the python package pandas was used, with the function read_csv. To correctly load the data a delimiter was specified as the .tsv file used tabs as delimiters.

Exploration

Once data was loaded the first step of exploring the data was to use pandas functions .info() and .head() to get an overview of the data. From this can be observed that there are a number of categorical columns and a number of numeric columns. The target column for prediction is the column "sentiment_label", that is a binary category formatted as "object".

In the next step the data was checked for missing values by selecting rows from the data where the pandas function .isna().any(axis=1) is true. The function .isna() returns a pandas dataframe where any null values show as "True" Boolean value and all else as false. The addition of .any(axis=1) applies the "True" value to the entire row. The main dataframe is then sliced with this Boolean data frame to produce a slice of the original dataframe where any row containing a null value is kept. No rows are returned by this, so no null values exist in the data. (Pandas Documentation, 2024)

Next the balance of the "sentiment_label" column was analyzed. The categories were balanced with 80.000 rows each with values "0" and "4". This meant that no further actions related to balancing the categories were necessary.

Text processing – removing unwanted text

With the help of the regex function re.sub(), parts of the text that was considered as irrelevant for the predictions were removed. The first regex pattern removed strings related to web addresses, such as "http", "www", and ".com". The second removes special characters and punctuation, and finally the third pattern replaces any occurrence of more than three characters with only two of that character.

Text processing – lemmatization

Using the Natural Language Toolkit (NLTK) class WordNetLemmatizer(), the words in the text were reduced to their base form. This reduces the dimensionality of the dataset and allows the model to form stronger correlations between the sentiment and the words. (Natural Language Toolkit Documentation, 2023)

Train and test split

The data was separated into two datasets, one for training the model and one for validating the accuracy of the model. The validation dataset was selected as 20% of the total data, randomly sampled.

Text vectorization/tokenization

Two different methods were attempted for this task, one for each model. The first approach was using TfidfVectorizer, converting the text data into a TF-IDF matrix (Term Frequency-Inverse Document Frequency). This tokenizes the words and weights them according to their occurrence frequency in the data.

The second approach used the BertTokenizer, which is a method used for BERT models (Bidirectional Encoder Representations from Transformers). This tokenizer works by breaking down the text into tokens, which may be whole words or smaller parts of words. The tokenization is based on a predefined vocabulary for the tokens. This method can produce better results than the TfidfVectorizer, but is more computationally intensive.

Training the models

The selected algorithms for this were LightGBM and a neural network. LightGBM works on a principle of tree-based learning, in which it combines multiple weaker models to create strong combined model. It accomplished this by a boosting method in which the weaker models are sequentially trained with a focus on the previous models errors.

Neural networks work by passing values through nodes/neurons that apply a weight and bias to the value before it is passed to the next node. The network is then fitted to the data by tweaking the weights and biases when an incorrect prediction is made. Neural networks are very adaptable and are suitable for text analysis as it may discover correlations between words that other models may struggle with.

The learning for LightGBM was additionally done with a grid search approach where the parameters of the models were programmatically tweaked, and the results of each run were recorded to be able to select the best overall configuration of the model.

LightGBM

First the LightGBM model was trained, with the grid search being done for “num_leaves” and “learning_rate” in the ranges 40-151 increment 20, and 0.01-0.30 increment 0.05 respectively. “num_leaves” determines the complexity of the trees in the model, a higher number leading to higher complexity and potentially better defined partitions in the feature space. “learning_rate” determines the size of steps for the gradient descent, which means that a larger value may let the model approach the ideal value faster, but a higher value may also cause the model to overshoot the ideal value. (LightGBM Documentation, 2023)

The metrics tracked were accuracy and ROC AUC score, accuracy being the percentage of correct predictions. The threshold between classes in binary classification is normally set at 0.5, but ROC AUC (Receiver Operating Characteristic Area Under the Curve) evaluates the accuracy for the full range of the threshold from 0 to 1.

Early stopping was enabled to potentially reduce overfitting of the model.

The outcome of training the model showed that the best configuration for this problem is a "num_leaves" of 120 and a learning rate of 0.06. This resulted in an accuracy of ~78.9% and an ROC AUC score of ~86.8%.

Num_leaves	Learning_rate	Accuracy	ROC_AUC
120	0.06	0.789312	0.868390
140	0.11	0.788438	0.867642
80	0.11	0.788062	0.868019
140	0.06	0.788031	0.868572
100	0.06	0.787750	0.868156

Table 1, Top performing configurations of LightGBM.

```
              precision    recall  f1-score   support

           0       0.79      0.78      0.79     16002
           1       0.79      0.79      0.79     15998

    accuracy                           0.79     32000
   macro avg       0.79      0.79      0.79     32000
weighted avg       0.79      0.79      0.79     32000
```

Table 2, Classification score for the top performing LightGBM configuration.

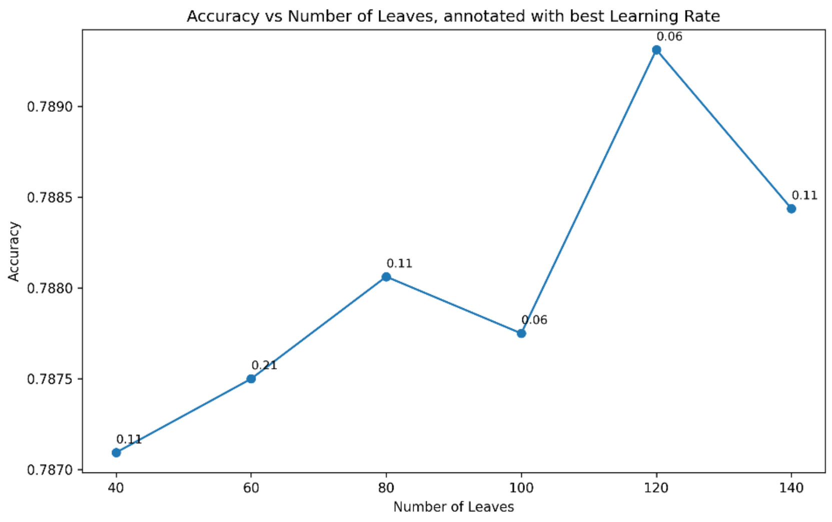

Image 1, The performance for each step of "num_leaves", with the best "num_leaves".

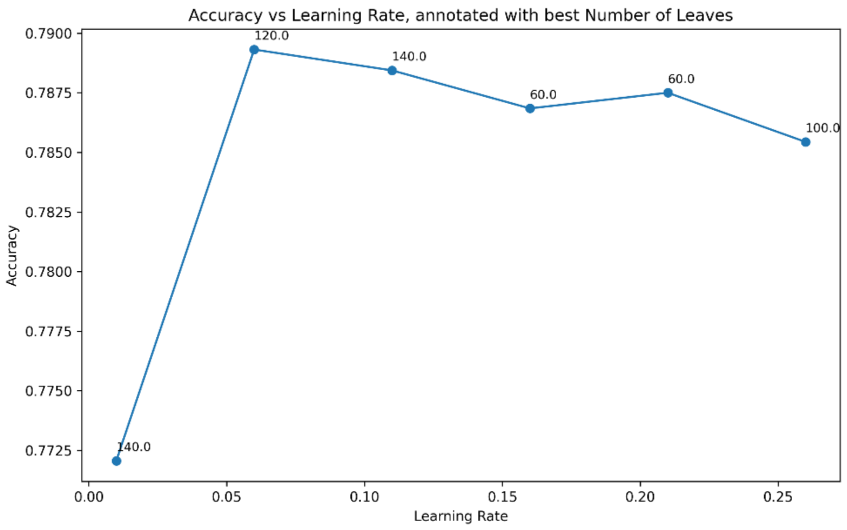

Image 2, The performance for each step of "learning_rate", with the best "num_leaves".

Neural Network

The neural network was structured as a sequential model with three layers. The input layer was an Embedding layer, which transform the data to fixed-size vectors. This is necessary because all input data needs to maintain the same fixed size. The input features count for this layers was set to 30.522, as this is the size of the BertTokenizer dictionary. The second layer was an LSTM unit (Long Short-Term Memory, which in a manner can remember contexts from earlier points in the training sequence. This is very suitable for sentiment analysis as sentiment can be very dependent on small details in a sentence, such as a "not" changing the whole meaning of the sentence. The output layer was a Dense layer with a single output with the sigmoid activation function, which returns a value between 0 and 1. (Tensorflow Documentation, 2024)

Same as for the previous model, the metrics tracked were accuracy and ROC AUC.

The best configuration with this algorithm was found at 290 estimators, which resulted in a model with ~79.5% accuracy and ~87.7% ROC AUC.

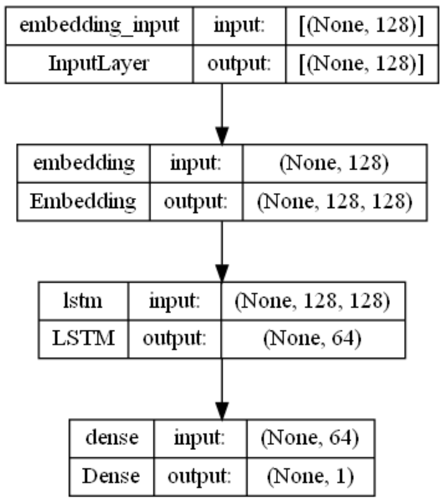

Image 3, Structure of the neural network.

```
              precision    recall  f1-score   support

           0       0.79      0.80      0.80     16002
           1       0.80      0.79      0.79     15998

    accuracy                           0.80     32000
   macro avg       0.80      0.80      0.80     32000
weighted avg       0.80      0.80      0.80     32000
```

Table 3, Classification score for the neural network prediction.

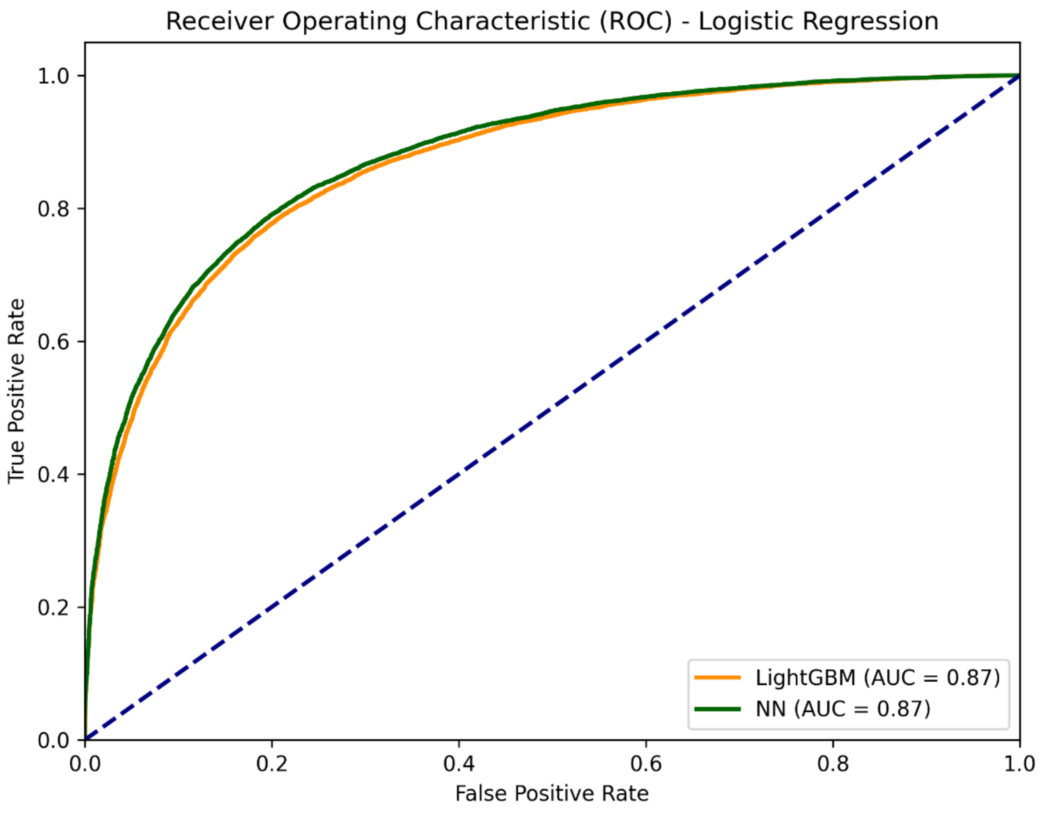

Image 4, Comparison of the ROC AUC between the two models.

Conclusions

Both models were able to predict the classes with similar accuracy, with a slight edge to the neural network. The edge is only 0.6% though and this may not be relevant for an actual use case. The ROC AUC metric also gives a slight advantage for the neural network model, here the difference is 0.9%, so still not a significant difference. Additionally a simpler model in LogisticRegression from sklearn was tested, though not presented in this report. This model showed similar numbers for the accuracy as well, so it is not clear that this task specifically gets significant benefit from the use of more advanced models.

For the data preprocessing it was quite unclear what did and didn’t have significance for the prediction. The first tries with minimal preprocessing gave a result of 75%+ accuracy, and the steps added later only gave marginal increases in the accuracy.

References

LightGBM Documentation. (2023). Retrieved from https://lightgbm.readthedocs.io/en/stable/

Natural Language Toolkit Documentation. (2023). Retrieved from https://www.nltk.org/

Pandas Documentation. (2024). Retrieved from https://pandas.pydata.org/docs/index.html

Scikit-learn Documentation. (2024). Retrieved from https://scikit-learn.org/stable/

Tensorflow Documentation. (2024). Retrieved from https://www.tensorflow.org/