Åbo Akademi- Mini Project 2
Tweet Sentiment Analysis
NLP

Markku Pulli

14.2.2021

Abstract

Exercise includes two different machine learning models for sentiment analysis on the given dataset. Data is from twitter tweets. Goal is to find best fitting algorithms to generate most accurate outcome and predictions. RNN and RandomForest models were utilized during this exercise. Detail performance evaluation is also included in this report. There was no target goal set but this was more like study to find out how well different algorithms perform and compare. RNN outperforms RF qualifier.

Introduction

The data used in this experiment is Sentiment140 dataset, a publicly available data set created by three graduate students at Stanford University: Alec Go, Richa Bhayani, and Lei Huang. The data comprises approximately 160,000 automatically annotated tweets.

Activities included input data analysis and processing, learning algorithm selection and optimization and finally analyzing the results to select optimal model.

Data Observations and Data Processing

The data includes tweets from Twitter. There are two columns of data. 'sentiment_label' and 'tweet_text'. Data shape is (160000, 2).

Input data includes.

- total count of records: 160,000
- Total count of features in each record: 1
- No data was found missing -> no data imputation need.

Data is provided in '.tsv' format. Attributes included are text string and numerical, [object, and integer].

Text sentence column is pre-processed. Task and activities include:

- Special character removal
- Tokenizing
- Padding
- Embedding using GloVe
- Removing STOPWORDS

Text string, all the tags and special characters were removed as non-informative data. Only the words with prediction power are included in the analysis.

Tokenizing, sentences transformed to tokens. [1]

Padding, words are converted to numeric integer values as tokens.

Word embedding, mapping words or phrases from the vocabulary to vectors. The words with similar meaning are similar representation. GloVe pre-trained word vectors were used this case. File name ‘glove.6B.50d.txt. Other vectorization options are recommended to try Word2vec or TextVectorization.

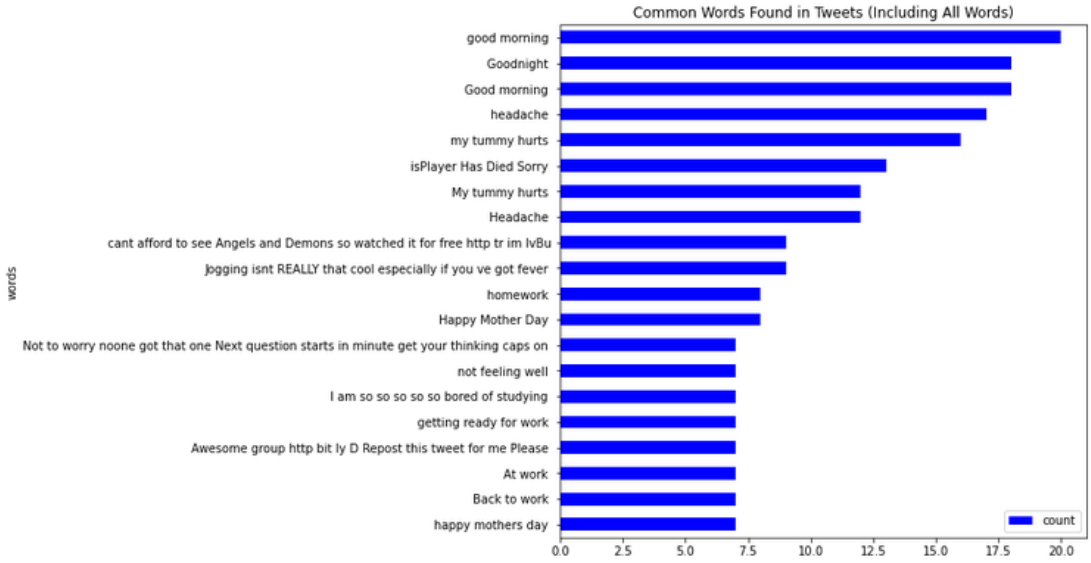

Figure1: Common words and sentences found in tweets.

After all data analysis was completed input data was separated to training set (70%) and test set (30%) before fitting the model. Also 80/20 used during the training.

Methodology/Modeling

Keras/Tensorflow RNN and scikit RandomForest Classifier were used during the exercise. Below are the network layouts for both.

RandomForest Classifier

Number of estimators and max depth parameters were optimized.

Initial parameter set:

rf = RandomForestClassifier()

'n_estimators': [50, 100, 200, 250],

‘max_depth': [20, None]

```
BEST PARAMS: {'max_depth': 20, 'n_estimators': 250}

0.579 (+/-0.005) for {'max_depth': 20, 'n_estimators': 50}
0.585 (+/-0.005) for {'max_depth': 20, 'n_estimators': 100}
0.589 (+/-0.008) for {'max_depth': 20, 'n_estimators': 200}
0.593 (+/-0.007) for {'max_depth': 20, 'n_estimators': 250}
0.571 (+/-0.006) for {'max_depth': None, 'n_estimators': 50}
0.58 (+/-0.008) for {'max_depth': None, 'n_estimators': 100}
0.585 (+/-0.006) for {'max_depth': None, 'n_estimators': 200}
0.587 (+/-0.006) for {'max_depth': None, 'n_estimators': 250}
```

Figure 2: RandomForest Classifier results.

Below three parameter sets were selected as these had best training performance.

rf1 = RandomForestClassifier(n_estimators=200, max_depth=20)

rf2 = RandomForestClassifier(n_estimators=200, max_depth=None)

rf3 = RandomForestClassifier(n_estimators=250, max_depth=None)

Next all three test predictions were completed. From above three models the best test accuracy is with rf3 settings. No limitations on tree depth. Delta to other two models is less than 0.2%. For model selection simplicity should be considered as performance delta is so low. Note each time RF classifier was run results slightly differ.

RNN

Final network layout and parameter settings:

```
keras.Sequential([
    layers.Embedding(vocab_size, 50, weights=[embedding_matrix], input_length=maxlen),
    layers.Dropout(0.2),
    layers.Conv1D(128, 3, activation='relu'),
    layers.Dropout(0.2),
    layers.MaxPool1D(3),
    layers.Dropout(0.2),
    layers.Bidirectional(layers.LSTM(128, return_sequences=True)),
    layers.Dense(1, activation='sigmoid)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

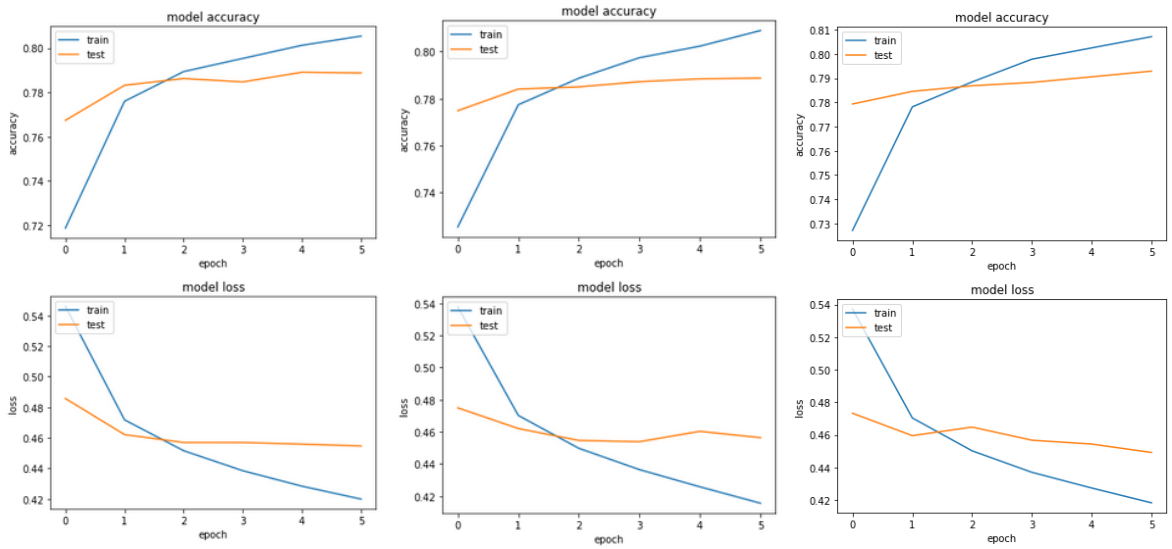

Figure 3: RNN model accuracy and loss trending. Three different models are presented.

Modeling included multiple rounds of parameter and network layout optimizations. Surprisingly most changes had minor or no impact on model performance. Test data loss and accuracy in general did not show any improvement after 2nd or 3rd epoch. This will bring the question if data input analysis and data wrangling has room for improvement. Data preparation techniques what were not tried do to time limitations are stemming[2] and lemmatization[3].

Model Name	Training Loss	Training Accuracy	Test Loss	Test Accuracy
RNN	0.416	0.810	0.459	0.793
RandomForest		0.590		0.600

Table 1: Model performance comparison.

Network Layouts

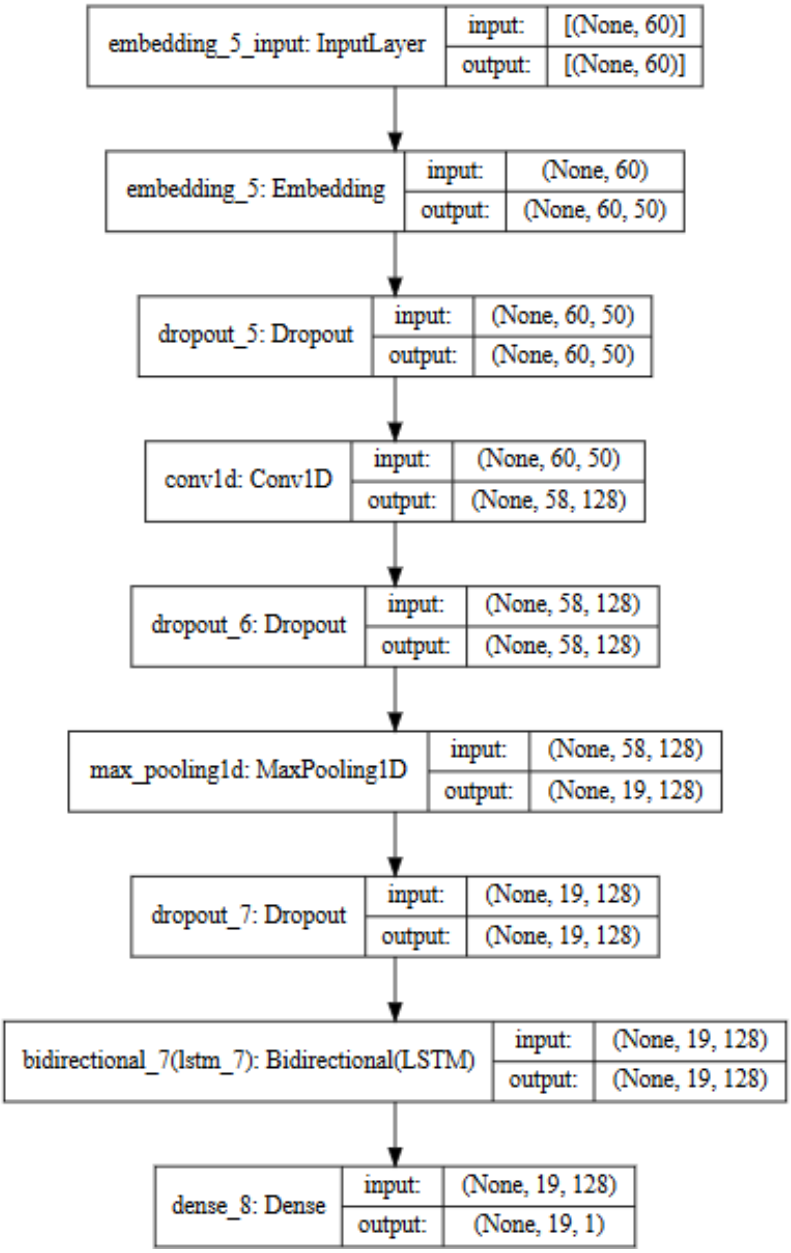

Figure 4: RNN network layout.

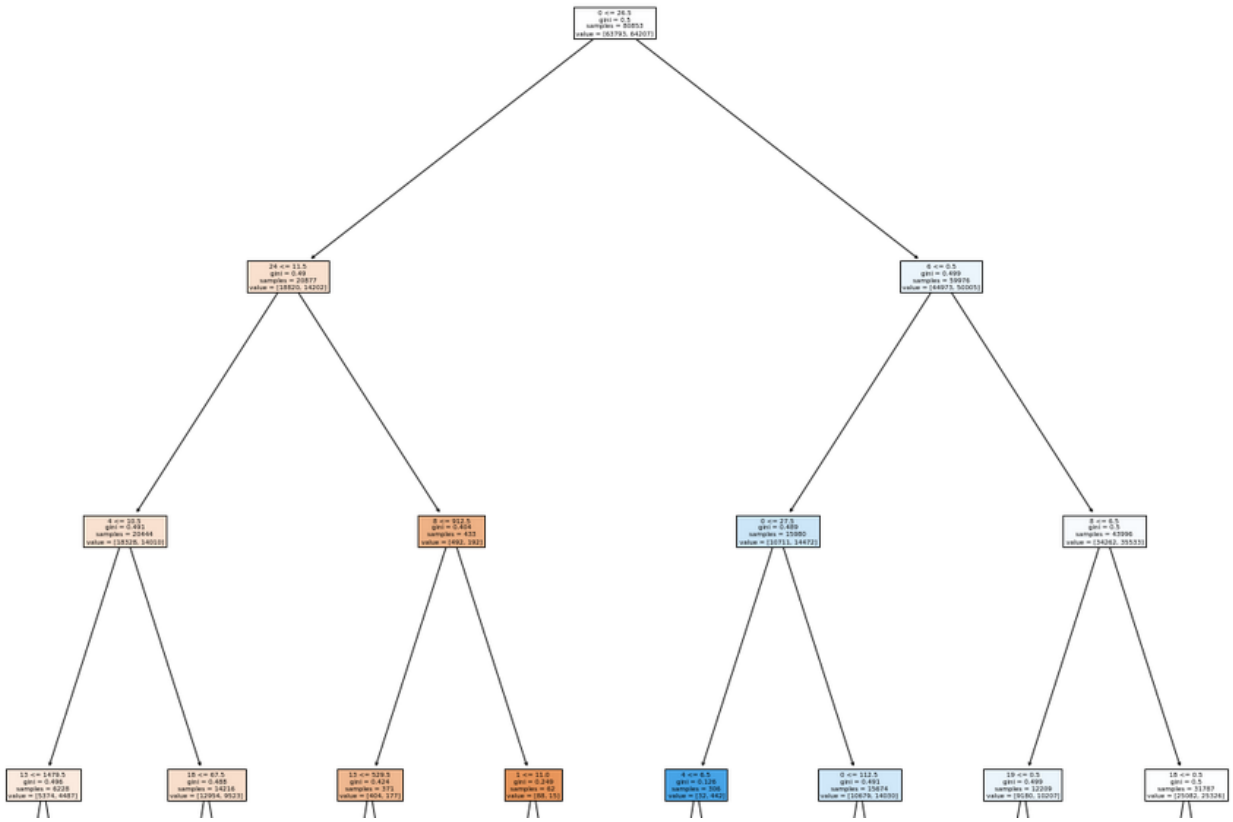

Figure 5: RandomForest Classifier network layout. Only first three layers are shown.

Results & Conclusions

Model tuning in RNN case is extremely slow. Adding a LSTM layer in network will have major impact on model fit time. As an observation RNN modeling is extremely time consuming to train. One parameter change and running the model fit again can take close to 30 minutes. It was extremely challenging to improve model accuracy above 79%. Network layout or hyperparameter tuning only had marginal impact on model test performance. With more time and investigation on finding best NLP algorithm results can be improved. Also, different approaches how to prepare the data is recommended to try. But with limited time and resources this will be another exercise.

As there is no preset target accuracy model optimization was finalized at 79% accuracy on RNN. RandomForest classifier after optimization is best at 60%. This is very low. Embedding was not done with RandonForest as there was not enough time to develop fully functioning code. Result show neural network in this application can outperform RandomForest.

References

[1]: Tokenization: Separate a chunk of continuous text into separate words. For a language like English, this is fairly trivial, since words are usually separated by spaces. However, some written languages like Chinese, Japanese and Thai do not mark word boundaries in such a fashion, and in those languages text segmentation is a significant task requiring knowledge of the vocabulary and morphology of words in the language. Sometimes this process is also used in cases like bag of words (BOW) creation in data mining. [Wikipedia]

[2]: Stemming. The process of reducing inflected (or sometimes derived) words to a base form (*e.g.*, "close" will be the root for "closed", "closing", "close", "closer" etc.). Stemming yields similar results as lemmatization, but does so on grounds of rules, not a dictionary. [Wikipedia]

[3]: Lemmetization. The task of removing inflectional endings only and to return the base dictionary form of a word which is also known as a lemma. Lemmatization is another technique for reducing words to it's normalized form. But in this case, the transformation actually uses a dictionary to map words to their actual form. [Wikipedia]