Report for BT5153 In-class Kaggle Competition

1. Background & Summary

This is a binary classification problem related to text mining. The original training data contains 44183 rows with three columns which are ‘Comment’, ‘Outcome’ and ‘Id’, respectively. Essentially, we would need use the texts from column ‘Comment’ to predict the result in column ‘Outcome’ that is either 1 or 0 where 1 stands for large voting numbers and 0 means small voting numbers in stackoverflow. To tackle this problem, an ensemble model composed of three base models which are xgboost, lightgbm and neural networks respectively is constructed, which manages to notch a final test accuracy of around 75%.

2. Data Pre-processing

A standard procedure about text pre-processing is leveraged on the dataset so as to clean the texts as the starting point. Specifically, some key steps include but not limited to lowercasing texts, noise removal such as punctuations and urls, stopwords removal, contraction expansion (e.g., I’m -> I am, we’re -> we are), text normalization such as stemming and lemmatization. Although the purpose of both stemming and lemmatization is to reduce morphological variation, they both suffer from some drawbacks, e.g., semantics loss. In general, they are not used unless there is some improvement of model performance. After comparing the performances from a couple of tests between these two techniques, it is decided to use lemmatization due to its relatively better performance.

3. Feature Engineering

As feature selection plays a significant role in model performance, it have been approached with considerable efforts. Firstly, CountVectorizer and TfidfVectorizer are employed to encode the texts with word counts or word tfidf values. After a couple of trials with either CountVectorizer or TfidfVectorizer and with different sets of parameters such as ‘token_pattern’, ‘max_df’, ‘min_df’ and ‘ngram_range’, CountVectorizer is finally selected based on the cross-validation scores, which may make sense in this particular task since TfidfVectorizer may penalize some frequent words that are indeed important for classifying high (or low) quality answers. After calibrating the parameters for CountVectorizer, there are more than 10000 words as word features in the vocabulary. Nevertheless, there could be some words which have little feature importance and may therefore be potentially prone to overfitting especially when our training data size is not huge actually. In the next section, it would be elaborated how feature selection has been conducted on top of these word features. In addition, there are 117 handcrafted features created from scratch based on three levels of text structure, i.e., sentence-level, word-level and character-level. A full list of these features could be found in Appendix A. All the features are normalized using standard scaler in scikit-learn.

4. Modeling

Pertaining to the modelling stage, it starts from a train/test split from which 20% of the training data are utilized for validation and preventing overfitting. Subsequently, a toolset of both machine learning and deep learning models is handpicked, built and evaluated with cross validation. In terms of cross validation, stratified 5-fold random sampling is leveraged to ensure that each fold would be a good representative of the whole data.

Naïve bayes, as a simple probabilistic classifier, is chosen to be the baseline model since it is fast, suitable for high-dimensional datasets and popular for text categorization problem. Indeed, the performance of the baseline model as shown from Table 1 in Appendix B also indicates its potential in this task. Apart from Naïve Bayes, some other commonly used classifiers that are also in the toolset are logistic regression, support vector machine, xgboost and lightgbm. Several neural network structures with a combination of convolutional neural networks (CNN) and recurrent neural networks (RNN) as depicted in the Appendix C are also implemented and evaluated, since neural network structures like LSTM and GRU are widely used for sequence data like text because of their capability of learning long-term dependencies. Nevertheless, probably due to the limitation of dataset size, the performances of neural networks are not quite superior to xgboost or lightgbm with a validation accuracy hovering around 72% - 73%.

In terms of choices of final models, xgboost, lightgbm and neural networks are selected for hyperparameter tuning since their base models could already achieve more than 71% validation accuracy. As mentioned in the previous section, it may be prone to overfitting when using all the words counts (or tfidf values), therefore, only the top 1000 tokens in terms of feature importance are extracted as the refined vocabulary for the CountVectorizer. Including the 117 handcrafted features, there are 1117 features for the boosted tree models to learn. On the other hands, word embeddings and a subset of handcrafted features are the main inputs for the neural network models. Due to concern about the out-of-vocabulary problem, character embedding using 1dCNN has been considered as well, though it is not deployed into the final model since it seems no conspicuous strengths of using it in this task and it's easily trapped in overfitting. Regarding hyperparameter tuning, methods like grid search, random search and Bayesian optimization are adopted, which help boost the performance by about 1% - 2%.

In order to further increase the prediction accuracy, an ensemble model with stacking is built on top of the three base models above. A multi-layer perceptron model works as a meta-classifier and learns from predictions of the three models, which increases the validation accuracy to around 74% after hyperparameter tuning, higher than the individual base models. Besides, it shows little signs of overfitting.

5. Discussion

From the confusion matrix and the classification report, it could be realized that the final model still has some comparatively weakness in recognizing low quality texts. In order to enhance the model capability, if unable to have more data, one potential direction would be to revert back to feature engineering. It is tricky yet important, e.g., 77 out of the top 100 features of the xgboost model are actually handcrafted.

Appendices

A. Table of Handcrafted Features

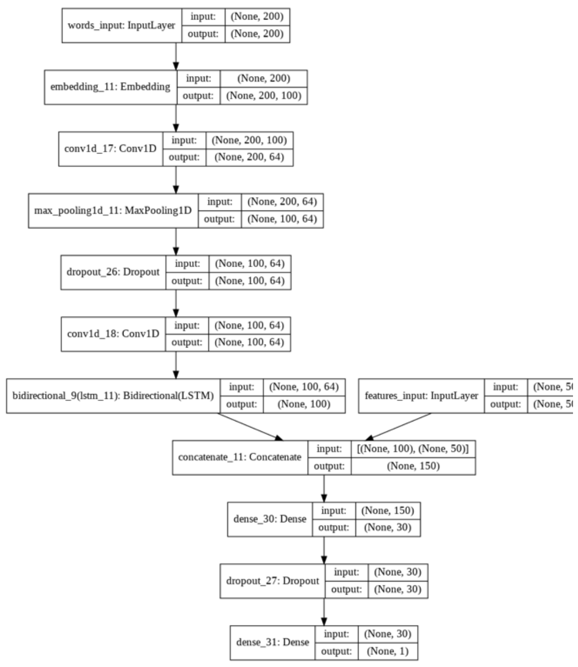
Features (number of columns created)	**Explanations**
text_len (1)	Number of tokens (words) in a text.
sent__num (1)	Number of sentences in a text
avgWrd__len (1)	average length of words in a text
url__cnt (1)	number of websites (links) in a text
digit__cnt (1)	number of tokens having digits in a text
bracket__cnt (1)	number of brackets or parentheses in a text
equal__cnt (1)	number of ‘=’ or ‘<-’ in a text
verb__cnt (1)	number of verbs in a text
noun__cnt (1)	number of nouns in a text
adv__cnt (1)	number of adverbs in a text
adj__cnt (1)	number of adjectives in a text
at__user (1)	number of ‘@’ in a text
comment__cnt (1)	number of ‘#’ in a text
libKeywords__cnt (1)	number of ‘library’ or ‘package’ in a text
codeMethod__cnt (1)	number of text snippets having pattern like *.* or *(*
keywrds_weight (1)	sum of keywords’ weights in a text
nonStop__cnt (1)	number of non-stopwords in a text
continuousChar__cnt (1)	number of text snippets having pattern with more than 2 (>=3) successive occurrences of a letter
continuousDigit__cnt (1)	number of text snippets having pattern with more than 2 (>=3) successive occurrences of a digit
continuousPunct__cnt (1)	number of text snippets having pattern with more than 2 (>=3) successive occurrences of a non-word character
`Char`__cnt (`Char` stands for any standard char, e.g., a-z, 0-9, etc.) (67)	number of different standard chars in a text
`Punct`__continuous__cnt (`Punct` stands for any punctuation in ".,-_()[]{}!?:;'\"/\\%$`&=*+^~|<>") (30)	number of text snippers having pattern with more than 2 (>=3) successive occurrences of a punctuation for each single punctuation

B. Model Performances of Some Key Models

Model	Training Accuracy	5-Fold Cross-Validation Accuracy	Test Accuracy
Naïve Bayes	0.7229	0.6636	0.6641
Logistic Regression	0.8611	0.6313	0.6482
Support Vector Machine	0.8057	0.6231	0.6381
Xgboost (after hyper-parameter tuning)	0.8679	0.7274	0.7245
Lightgbm (after hyper-parameter tuning)	0.8538	0.7192	0.7229
Ensemble Model	~0.73	~0.73	~0.74

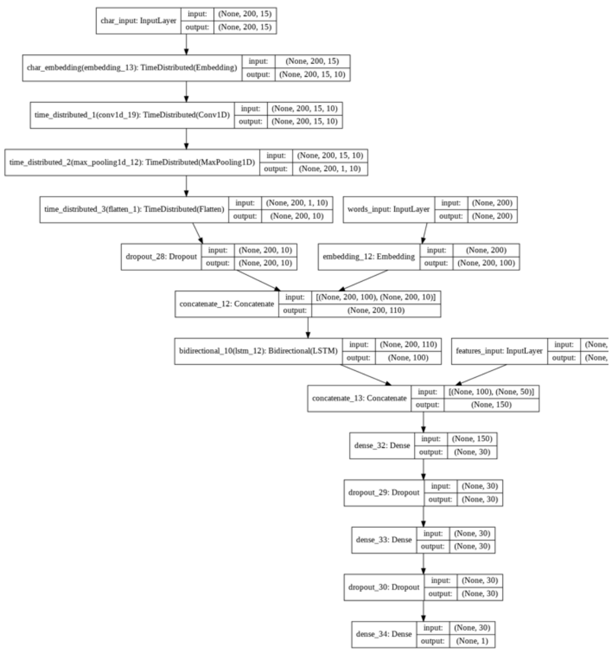
Table 1: Model Performances

C. Typical Neural Networks Structures under Experiments



words_input: InputLayer input: (None, 200) output: (None, 200)
embedding_11: Embedding input: (None, 200) output: (None, 200, 100)
conv1d_17: Conv1D input: (None, 200, 100) output: (None, 200, 64)
max_pooling1d_11: MaxPooling1D input: (None, 200, 64) output: (None, 100, 64)
dropout_26: Dropout input: (None, 100, 64) output: (None, 100, 64)
conv1d_18: Conv1D input: (None, 100, 64) output: (None, 100, 64)
bidirectional_9(lstm_11): Bidirectional(LSTM) input: (None, 100, 64) output: (None, 100)
features_input: InputLayer input: (None, 5 output: (None, 5
concatenate_11: Concatenate input: [(None, 100), (None, 50)] output: (None, 150)
dense_30: Dense input: (None, 150) output: (None, 30)
dropout_27: Dropout input: (None, 30) output: (None, 30)
dense_31: Dense input: (None, 30) output: (None, 1)

Convnets + BILSTM (with hand-crafted features)



char_input: InputLayer input: (None, 200, 15) output: (None, 200, 15)
char_embedding(embedding_13): TimeDistributed(Embedding) input: (None, 200, 15) output: (None, 200, 15, 10)
time_distributed_1(conv1d_19): TimeDistributed(Conv1D) input: (None, 200, 15, 10) output: (None, 200, 15, 10)
time_distributed_2(max_pooling1d_12): TimeDistributed(MaxPooling1D) input: (None, 200, 15, 10) output: (None, 200, 1, 10)
time_distributed_3(flatten_1): TimeDistributed(Flatten) input: (None, 200, 1, 10) output: (None, 200, 10)
words_input: InputLayer input: (None, 200) output: (None, 200)
dropout_28: Dropout input: (None, 200, 10) output: (None, 200, 10)
embedding_12: Embedding input: (None, 200) output: (None, 200, 100)
concatenate_12: Concatenate input: [(None, 200, 100), (None, 200, 10)] output: (None, 200, 110)
bidirectional_10(lstm_12): Bidirectional(LSTM) input: (None, 200, 110) output: (None, 100)
features_input: InputLayer input: (None, output: (None,
concatenate_13: Concatenate input: [(None, 100), (None, 50)] output: (None, 150)
dense_32: Dense input: (None, 150) output: (None, 30)
dropout_29: Dropout input: (None, 30) output: (None, 30)
dense_33: Dense input: (None, 30) output: (None, 30)
dropout_30: Dropout input: (None, 30) output: (None, 30)
dense_34: Dense input: (None, 30) output: (None, 1)

1D_Char_Embedding + Convnets + BILSTM (with hand-crafted features)