BT5151 Kaggle Competition

Submitted by: Liu Tong / Matriculation Number: A0206538A

Section1: Data Preprocessing & Feature engineering

The first step for normal NLP data preprocessing is to convert all characters to lower case. As the original entries in the datasets are already in lower case, so this step is not needed. Furthermore, the tf-idf vectorizer will change all characters to lower case by default. Other data cleaning techniques, such as stop words filtering, number filtering, punctuation filtering and lemmatization are tried out onto the dataset. However, in the end, these techniques are not applied as it turns out some stop words, and special symbols are indeed important and meaningful for this case.

TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. It is a common method for scoring words. It can be easily applied through TfidfVectorizer through sklearn feature extraction package. However, during transformation, the default parameters for token pattern and ngram range are changed as listed below.

Parameter	Default	Final Setting
analyzer	*word*	*char*
ngram_range	*(1, 1)*	*(1,6)*

As the original dataset contains comments, focusing on programming. Hence, it is good to detects symbols such as {, }, [,], :, \, <, - etc which may represent codes. This is why the analyzer is changed from default, which is word to character. Furthermore, search on ngrams are performed and (1,6) give the best validation result. Different settings for min_df and max_df, which ignore terms that have a document frequency strictly lower or higher than the given threshold, were also explored. It turns out the default values result in highest validation accuracy. The total number of features after transformation is 5,509,079.

Lastly, dataset balance was checked. The counts for label 1s and 0s are 24,992 and 19,191 respectively, which are not too screwed. Hence, dataset is considered pretty balanced and no sampling technique is applied.

Section2: Model Setup

In this section, we will consider how we can develop and evaluate predictive models for the provided dataset. The final outcome is either 0 or 1, hence the problem is a binary classification. Binary error is used as the evaluation metric. Original dataset after tfidf vectorization is split based on a 4 to 1 ratio to train and validation dataset. X is the feature dataset, which is the sparse metrics. Each row, it contains the tf-idf scores for different grams extracted from this particular comment. The number of rows is the same as number of comments. Y is the outcome, which is the prediction outcome for each comment, either 0 or 1. Model is trained based on the train set and evaluated based on validation set.

Section 3: Model Exploration & Hyperparameter Tuning

Different models explored are listed in table1. Some pros and cons are also listed based

on what I observed during the process. The winning model is lightGBM after hyperparameter tuning.

Model	Accuracy Score	Comments
KNN	0.690	Accuracy is low as expected, as KNN is vulnerable to the curse of dimensionality.
SVM	0.721	Accuracy is moderate, but really slow in training, not to mention hyperparameter tuning using grid search.
Logistic Regression	0.745	Relatively good accuracy achieved within a short period of time. Easy to tune hyperparameters.
LightGBM	0.747	Compared to XGboost, lightGBM can achieve good accuracy much faster. Easy to tune hyperparameters.
Neural Network	0.739	Feed forward neural network, with 6 layers and 10 neurons for each layer. (sallow and wide)

Table1: Accuracy Table for Different Models

You can either setup lightGBM using sklearn API or just "native training" API, which is a wrapper around the backend C++ library. Despite the easily integration of sklearn API with full sklearn toolkit, such as pipelines with data preprocessing, hyperparameter optimization, model evaluation, etc.; sklearn API does not include the option for early stopping, which I personally think is an important parameter for speeding up training process and preventing model overfitting. Hence, the native training API is used. For hyper-parameter tuning, a loop providing different parameters and recoding averaged performance was constructed to choose the best parameter set. Among the many hyperparameters for lightGBM, 'learning_rate' and 'num_leaves' are probably the most important ones for my model. Learning rate can extensively affect the accuracy, and number of leaves affects the training speed and model overfitting. After parameter tuning, the model validation accuracy is improved from 0.723, which is using all default setting to 0.747 with hyperparameter tuning.

Section 4: Results & Conclusion

Appendix shows the top30 most importance features extracted from the best lightGBM model. As the dataset is focus on comments from probably programmers, the special symbols such as ": ,()>%?=" and alphabet characters like "py, val, try, import, int, if" are important features.

The final predictions, result output from three models are ensembled; the final decision depends on what the majority of the votes (models) think. This help prevent overfitting from a single model. Furthermore, by combining different learners, the final prediction accuracy is higher than each single model.

Even though the winner model is lightGBM, I still believe a fine-tuned neutral network model can reach better accuracy. However, there are many types of neural networks, and each type can possess its own structure. Furthermore, different types of model may work better for certain features extraction, for example, word embeddings are probably better features for CNN instead of tf-idf. Thus, to tune a neural network model that performs well is very complex and time consuming.

Appendix

LightGBM Model Top 30 Important Features

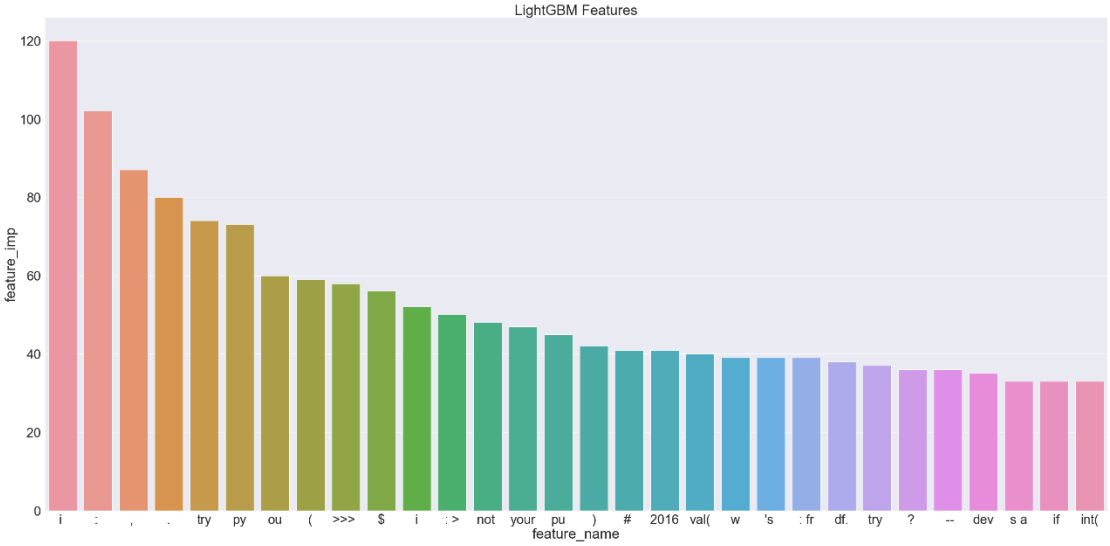

Figure1: LightGBM Model Top 30 Important Features Based On ***"Split"***

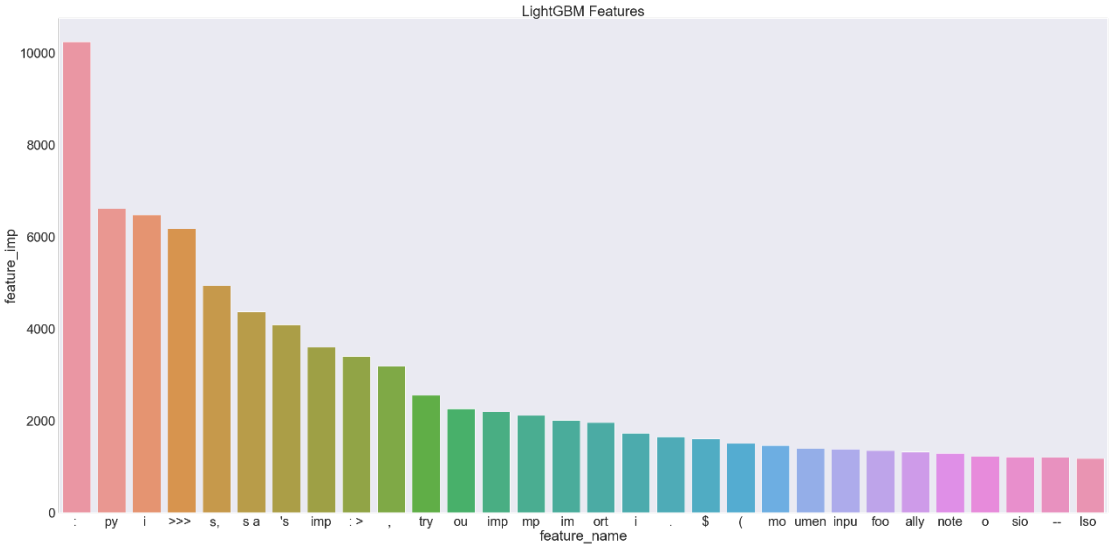

Figure2: LightGBM Model Top 30 Important Features Based On ***"Gain"***