BT5153 Applied Machine Learning for Business Analytics

In-class Kaggle Competition: Binary Classification Problem

Report submitted by Jinil Kim (A0206472J)

Problem Statement

A binary classification task is given for a dataset containing comments and labels. By applying machine learning techniques, predictive models that can provide "Outcome" label from "Comment" data of test dataset are to be created, and based on accuracy, the best models are to be chosen.

Data Exploration

The train and test datasets contain 44,183 and 28,200 observations, respectively, and contain no null values. The train data contains *Id*, *Comments*, and *Outcome* values while the test data only contains *Id* and *Comments*. *Comments* are text data without uniform sequence length while *Outcome* is the binary label for each unique *Id*. The goal is to predict *Outcome* values for each *Id* in the test data as accurately as possible.

Data Pre-processing

Differently pre-processed text-data was used for modeling, and each time one or more of the following text data pre-processing methods were used.

- Toggling text data to all lowercase
- Removal of numbers
- Removal of stopwords
- Removal of punctuations
- Removal of whitespace
- Removal of long words(word length > 30)
- Lemmatization

With different combinations of the methods above, results were compared for models with relatively high accuracies.

For tokenization and vectorization of sentences, *Scikit-learn*'s *CountVectorizer* and *TfidfVectorizer* were used along with *keras*'s *Tokenizer* for word embedding. For vectorizers, ngram of n=2 and n=3 were considered, and words with frequency below 0.00005 and above 0.35 were dropped in order to prevent vectorizer from creating millions of features, which would require tremendous amount of computing power to perform the task. For analyzer parameter of *TfidfVectorizer*, "word" or "char" were used. Similar to pre-processing, the results were compared once modeling was performed.

Models Explored

5 simple machine learning classifiers were used for modelling, along with two neural network models, one simple model and the other an RNN model with word embedding implemented using *genism*'s *word2vec* method. Modelling methods were carefully chosen to see different results with diverse approaches while minimizing required computing power.

For fine-tuning of for the 5 machine learning models, *GridSearchCV* was used, but for neural network models, parameters were manually tuned.

Models and hyperparameters considered for tuning are described in Table 1.

Aside from individual models, simple stacking of best models was used to improve test accuracy. This involved finding mean probabilities for the two *Outcome* values for models used in stacking and transforming them to binary values.

Evaluation Method

For each text pre-processing method used and model explored, models were evaluated based on k-fold cross validation scores with k=5. Training accuracy and more importantly cross validation score were used as criteria for model selection: submissions were made only for models with training accuracy above 80% and relatively high validation accuracies. For neural network models, *validation_split* of 0.1 was used to check for over-fitting, as significant drops in validation accuracy before training over all epochs would not improve test accuracy.

For stacking models, models were chosen based on cross validation scores, and no further validation was performed for stacked models.

Results

Text pre-processing methods were finalized after a few iterations, as from some of the observations it became clear which method fits the dataset better than others.

- Accuracy is improved when numbers and punctuations are not removed.
- Accuracy increased when analyzer is set to "word" and punctuations are kept. (Only for XGBoost, analyzer="char" gave a better result)
- Lemmatization do not improve accuracy (low-frequency words already removed)
- It seems accuracy can be further improved by higher degree of ngram. However, this makes the computation more expensive.
- Removal of stopwords and whitespace do not improve accuracy, but it allows models to become computationally cheaper.

The validation scores after hyperparameters are tuned for optimal results can be found in Table 2.

The best validation accuracy was given by logistic regression, followed by XGBoost. Unlike complex models that use linear models and are not specifically designed for binary classification, logistic regression model seems to perform best in this task due to its design using Bernoulli distribution. XGBoost alone worked nicely due to its framework with in-built optimized regularization.

Stacking models gave improved accuracies compared to individual models. Stacking models gave best accuracy when (i) varying modeling methods were involved and (ii) more weights were given to models with higher cross validation scores. Best test accuracy was obtained from a combination of two XGBoost models and two logistic regression models, each with different vectorizer and model parameters.

Appendix

Table 1: Models explored and parameters used for tuning

Model	Parameters for Tuning
Naïve-Bayes	alpha, fit_prior
Logistic Regression	C, max_iter, penalty
Decision Tree	criterion, max_depth
Random Forest	bootstrap, n_estimators, max_depth, max_features, min_samples_leaf, min_samples_split
XGBoost	n_estimators, gamma, subsample, colsample_bytree, max_depth
Neural Network	num_neurons, batch_size, epochs
LSTM Neural Network	sequence_length, batch_size, dropouts,
Stacking 1	2 Logistic Regression + 2 XGBoost
Stacking 2	2 Logistic Regression + 2 XGBoost with uneven weight

Table 2: Best validation accuracies for each model after hyperparameter tuning

Model	CV Accuracy
Naïve-Bayes	0.639478
Logistic Regression	**0.694205**
Decision Tree	0.604012
Random Forest	0.659463
XGBoost	**0.691625**

Model	Val Accuracy
Neural Network	0.5620
LSTM Neural Network	0.5757