Junyi Zhu

Kaggle Name: Junyi Astalos

Work Flow

Start with TF-IDF + SVM

At the first attempt, we combine summary with text into a new column "combined".

Then we preprocessed the "combined" by text normalization such as lower and remove symbols. Further with a Lemmatizer we get 'cleaned_combined'.

Here comes Term Frequency-Inverse Document Frequency (TF-IDF) technique, which calculates the importance of each word in the document based on how frequently it appears in the document and across the entire corpus. With TF-IDF, we can use these feature matrices to train and evaluate various machine learning models, such as Naive Bayes(NB) and Support Vector Machines(SVM).

With the pipeline TF-IDF+SVM, we get a 60% accuracy on local validation set at the startup.

Try to improve the pipeline

There are so many choices to improve the pipeline. My attempts include:

(❌ does not directly improve the performance ✔ actually improve and change my build)

1. **Replace Lemmatizer with Glove.**❌

   However, it turns out Lemmatizer works better, so there is no need to change. Furthermore, I experiment the case without lemmatizer, neither change anything.

2. **Compare SVM with NB**❌

   NB is quite a vulnerable model to the imbalanced dataset, it turns to predict towards 5 and cannot learn much about 2-4 samples. Adaboost is also tried to create tens of NB to save the performance, however, it does not beat SVM and only has at most 58% acc. As SVM takes too much time, it is impossible to base Adaboost on SVM here.

3. **From classification to regression**✔

   At the first, I didn't realized float values can be accepted. However, after I realized the ordinal relationship of the ratings [1,2,3,4,5] makes it possible to see the problem as a regression one. Since then, the methods are no more limited to classification.

4. **Extract more features**✔

   Note that we should prevent information leakage here. In other words, we should not use val/test set data for training or feature extraction.

   1. Sentiment Analysis: TextBlob or VADER to extract the sentiment polarity (positive, negative, or neutral) and subjectivity scores for each review.
   2. Review Length: The length of a review might be related to the movie rating.
   3. Readability Scores: Readability scores like Flesch-Kincaid, Gunning Fog, or SMOG index can help measure the complexity of the text. These scores can be useful features for predicting movie ratings.

1. Flesch Reading Ease

   This metric calculates the readability of a text based on the average sentence length (i.e., the number of words divided by the number of sentences) and the average number of syllables per word. Higher scores indicate easier readability, while lower scores indicate more complex text.

2. SMOG Index (Simple Measure of Gobbledygook)

   This readability metric estimates the years of education a person needs to understand a text. It is based on the number of polysyllabic (3 or more syllables) words in the text and the square root of this number.

3. Gunning Fog Index

   This metric estimates the number of years of formal education required to understand a text on first reading. It takes into account the average sentence length and the percentage of complex words (words with three or more syllables) in the text.

4. Part-of-Speech (POS) Tagging

   To count the number of nouns, verbs, adjectives, and adverbs in each review.

5. HelpfulnessScore = HelpfulnessNumerator / HelpfulnessDenominator

6. Product mean score.

   Based on training set, we calculate the mean score for each product, and then we use the scores as features in training and testing set. A product never seen in training set but appear in test set will be marked mean of mean scores.

7. User mean score.

   Similarly designed as Product score.

8. Product ID and User ID one-hot encoding.

   Last feature added but works

5. **Comapre SVM with LightGBM and CatBoost** ✓

   1. LightGBM (Light Gradient Boosting Machine): LightGBM is a gradient boosting framework developed by Microsoft. It is designed to be efficient and scalable, Text Preprocess Word Embedding Feature Regression Model MSE(local) Lemmatizer TF-IDF * LightGBM 0.61 making it particularly well-suited for large-scale datasets or datasets with a large number of features. LightGBM uses a unique technique called Gradient-based One-Side Sampling (GOSS) to filter out data instances for finding a split value and Exclusive Feature Bundling (EFB) to reduce the number of features. These techniques enable LightGBM to handle large-scale data and high-dimensional data more efficiently than other GBDT algorithms.

      Using LGBM improves my score on kaggle from 1.0+ to around 0.8 (meanwhile change my definition of the problem from classification to regression as I use a regressor now, previous is SVM).

   2. CatBoost: CatBoost is a gradient boosting library developed by Yandex. It is designed to work well with categorical features without the need for manual preprocessing or encoding. CatBoost uses a combination of ordered boosting and a novel algorithm for handling categorical features called "ordered target statistics." This allows CatBoost to effectively process categorical data while reducing overfitting. CatBoost is known for its high performance and accuracy, making it a popular choice for a wide range of machine

learning problems, especially those with many categorical features. The result is LightGBM works best, which gets a 0.61 MSE score on validate set.

6. **Try combine TF-IDF with doc2vec**❌

   Does not work, possibly out of non-context embedding, so result in noise.

7. **PCA/SVD** ❌

   Does not wok, mainly because of out of ram problem.

8. **Hyperparams Finetuning**✔

   It is almost always better to tune somehow...

Final Build

Word Embedding	Feature	Regression	MSE(local)	Kaggle
TF-IDF	*	LightGBM	0.59	0.76

*: TFIDF + Sentiment(TextBlob + VADER) + Review Length + Readability Scores(Flesch-Kincaid, Gunning Fog, or SMOG) + POS tagging + Helpfulness Features + Product mean score + User mean score + one-hot encoding Product ID and User ID

My exploration for the better pipeline during the competition follows one-pass optimization, that is, for example in tuning hyperparameters, we don't have enough time for GridSearch, so we optimize every param only once except it works.