CS 506 Midterm Report

Jae Hong Lee
Boston University
jhonglee@bu.edu

Abstract

For the CS 506 Midterm competition I'm using Principal-Component Analysis (PCA) and XGBoost (Extreme Gradient Boosting) to predict the star rating associated with user reviews from Amazon Movie Reviews using the available features dataset provided by Kaggle class midterm. This is my GitHub repository address.

1 Introduction

1.1 Dataset

The movie rating dataset used for this midterm has the following attributes:

- Id: a unique identifier associated with a review
- Product Id: unique identifier for the product
- User Id: unique identifier for the user
- Helpfulness Numerator: number of users who found the review helpful
- Helpfulness Denominator: number of users who indicated whether they found the review helpful
- Time: timestamp for the review
- Summary: brief summary of the review
- Text: text of the review
- Score: rating between 1 and 5

The dataset for movie ratings, which will be used in this midterm, consists of 1,697,533 rows. I need to split it based on the score column. Out of the total data, 1,485,341 rows have a score, forming the Training Set, while 212,192 rows have missing scores, which make up the Testing Set. After splitting, the Training Set has 54 missing text columns and 28 missing summary columns, while the Testing Set has 8 missing text columns and 4 missing summary columns.

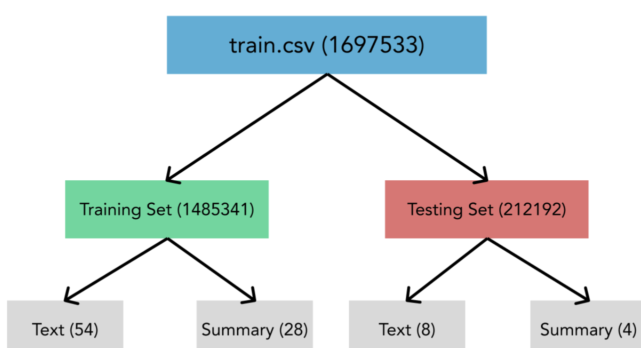

Figure 1: Data Split of train.csv

Due to missing values in the dataset, I believe XGBoost can be effectively applied, as it is capable of handling missing attributes in the data. The method can still make use of the available data, even when some features are missing. Certain attributes in the

Training Set, such as Id, Product Id, and User Id, are simply unique identifiers and do not provide valuable information for the model. These fields are more administrative and don't predict outcomes. Therefore, excluding them from the analysis could streamline the process. With these attributes removed, a decision tree might be a more appropriate method, as it can focus on the remaining meaningful features to make better predictions.

1.2 Model Overview

Decision tree models offer several advantages. They can handle both numerical and categorical data and perform automatic feature selection, making them well-suited to complex problems where interactions between variables are critical. Decision trees also capture non-linear relationships, enabling them flexible in various scenarios. Once trained, they make predictions quickly and provide clear, interpretable decision paths. However, they can be prone to overfitting, particularly on small datasets, as they may create overly complex trees that capture noise in the data. Additionally, decision trees tend to struggle with small variations or noisy data, though techniques like pruning can reduce this risk.

On the other hand, K-Nearest Neighbors (KNN) is a simple and intuitive algorithm that doesn't require a training phase, making it computationally light at the start. It is non-parametric, meaning it makes no assumptions about the underlying data distribution, which can be useful when the true relationship between features is unknown. KNN has its disadvantages. It can be computationally expensive during the prediction phase, as it requires storing and searching through the entire training set for each new prediction. KNN also struggles when irrelevant features are present or when the data is not scaled properly, as distance calculations become less meaningful. Additionally, KNN's performance declines when dealing with high-dimensional data (many attributes), as the algorithm becomes less effective at measuring meaningful distances.

2 Preprocessing Features

2.1 Helpfulness Feature Creation

From the starter code, I used this feature. To assess user feedback reliability, an additional feature called Helpfulness was created. This feature is the ratio of ‘HelpfulnessNumerator‘ to ‘HelpfulnessDenominator‘, providing a proportionate measure of helpfulness. Missing values were replaced with 0 to avoid errors in the model. After creating this feature, the original ‘HelpfulnessNumerator‘ and ‘HelpfulnessDenominator‘ columns were dropped.

2.2 Text Vectorization with GloVe and Custom Embeddings

To capture semantic information from the Text column, GloVe embeddings and a custom Vector model were utilized. GloVe embeddings provide pre-trained, context-independent word vectors, while the custom Vector model was trained specifically on the project's dataset to capture its own patterns and nuances. The word vectors from GloVe and Vectors were combined using different methods (‘concat‘, ‘average‘, or ‘weighted‘), allowing flexibility in representing each word based on its contextual meaning. These word vectors were then averaged to create a single vector representation for each sentence, effectively transforming text data into meaningful vectorized form.

2.3 Feature Extraction: Text & Summary Length and Sentiment Scores

For both the Text and Summary columns, additional attributes were extracted, including text length (character count) and sentiment scores. VADER sentiment analysis was applied to determine the sentiment polarity of both ‘Text‘ and ‘Summary‘, capturing positive, negative, or neutral sentiments. This added feature helps the model learn the emotional tone of the reviews. Both text length and summary length were also computed to assess verbosity and detail in reviews, providing additional context for model learning.

2.4 TF-IDF and Count Vectorization in Summary

For the Summary column, TF-IDF and CountVectorizer with N-grams were applied to analyze word and phrase patterns. TF-IDF captures term importance by weighting words that are frequent in the document but rare in the overall dataset, helping the model focus on meaningful keywords. CountVectorizer with N-grams captures common word pairs, such as "not good," providing the model with more detailed phrase-level context. This combination of TF-IDF and CountVectorizer features allows the model to learn both the importance of individual words and the structure of phrases in summaries.

2.5 Final Data Preparation and Combination

After transforming all features, the text vectors, TF-IDF, CountVectorizer, and other numeric features (like sentiment scores and length features) were combined into a single sparse matrix for efficient storage and processing. The final combined dataset includes both the original numerical features and the vectorized text attributes, ready for model training. This preprocessing pipeline effectively transforms raw text and numerical data into a structured format that captures both text semantic information and the quantitative insights from numerical attributes. The combined features are expected to enhance the model's ability to accurately interpret and predict based on the data.

3 Experiments

3.1 XGBoost Model

The experiment dataset underwent Preprocessing, and the eight columns after the features processing, the highest variance were selected. These datasets were then used to split the data into a training set and a test set in a 3:1 ratio. This ensured that I could accurately train our data, but wouldn't overfit so testing would still be accurate. The models were trained on this split data, and accuracy was evaluated. I conducted experiments using xgboost models: learning rate, max_depth, min_child_weight, and etc parameters were used before calculating final accuracy

The initial parameter I used was the followings:

Initial Parameter Results	
Parameters	Values
N_estimator	100
Learning Rate	0.1
Max_depth	5
Min_child_weight	5
Gamma	0
subsample	0.8
colsample_bytree	0.8

The Accuracy that I got from this was **0.602045**

3.1.1 Hyper Parameter Tuning

XGBoost models, contain several parameters, but I will only mention the parameters I touched.

- N_estimator : [100, 200, 300, 400, 500] Number of gradient boosted trees
- Learning Rate : [0.01, 0.05, 0.1, 0.2, 0.3] Boosting learning rate.
- Max_depth : [3, 4, 5, 6, 7, 8, 9, 10] Maximum tree depth for base learners.
- Min_child_weight : [1, 3, 5, 7] Minimum sum of instance weight(hessian) needed in a child.
- Gamma : [0, 0.1, 0.2, 0.3, 0.4] Minimum loss reduction required to make a further partition on a leaf node of the tree.
- subsample : [0.6, 0.7, 0.8, 0.9, 1.0] Subsample ratio of the training instance.
- colsample_bytree : [0.6, 0.7, 0.8, 0.9, 1.0] Subsample ratio of columns when constructing each tree.

3.2 Result

I applied GridSearchCV through the parameters of XGBoost. Based on the results of the experiments, the fianl accuracy scores are as follows:

Initial Parameter Results	
Parameters	Values
N_estimator	500
Learning Rate	0.1
Max_depth	4
Min_child_weight	3
Gamma	0.1
subsample	0.8
colsample_bytree	1.0

XGBoost Fine Tuning score was: 0.6409585

As I ran the grid-search to find the best parameter, this was my best result. I used this model to train the X_submission, and the result of the public leader board was 0.64315

References

[1] xgboost developers. *Frequently Asked Questions*. 2022. URL: https://xgboost.readthedocs.io/en/stable/faq.html.

[2] Jeffrey Pennington. *GloVe: Global Vectors for Word Representation*. 2014. URL: https://nlp.stanford.edu/projects/glove/.

[3] *SciKitLearn Documentation*. Accessed 25 Oct. 2024. URL: https://scikit-learn.org/stable/index.html.

[4] *SciKitLearn TfidfVectorizer Documentation*. Accessed 25 Oct. 2024. URL: https://scikit-learn.org/1.5/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.