Experiment 3: Ensemble Learning

Tian Tian

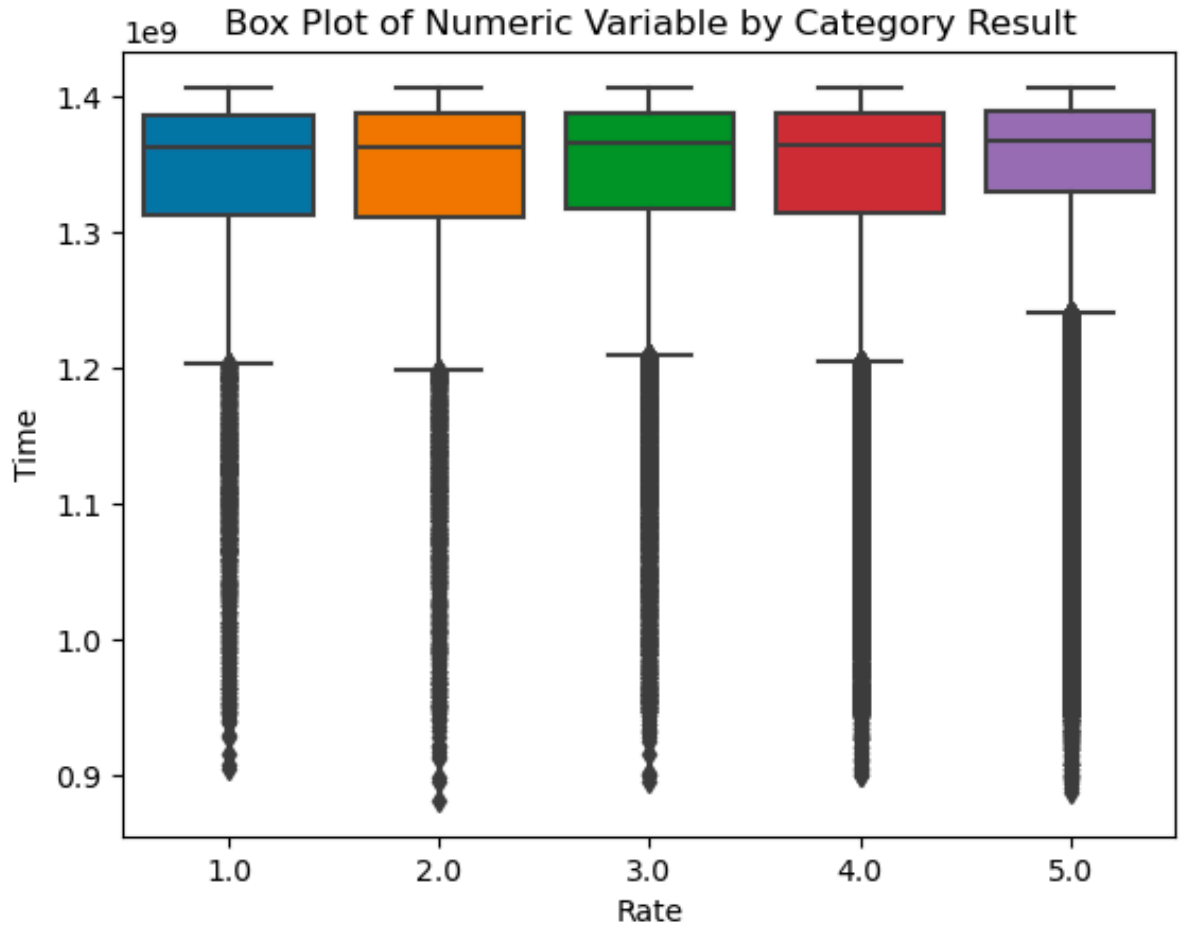
Student ID: 2021011048

I. Data Preprocessing and Analysis

In this section, we preprocess the provided data, transforming it from its raw format into numerical types that the model can handle. We need to process the reviewerID, asin, unixReviewTime, summary, and reviewText.

By simple statistics, there are a total of 219,997 reviews without missing values. Among these, there are 125,391 unique user IDs, 111,697 unique item IDs, and 5,499 unique review timestamps. All three dimensions have a considerable amount of repetition. These three dimensions are essentially categorical variables; we only need to know if they are the same or different, with no inherent order or size relationship (timestamps might have an order relationship). Therefore, we can handle them by encoding them as integers. To avoid any ordinal relationships from the encoding (which could cause issues in SVM), we can convert the encodings into high-dimensional binary vectors.

For an intuitive explanation of these three variables, we believe that the scores from the same person/the scores for the same item should follow some regularity. Therefore, reviewerID and asin can be used to determine whether it is the same person/item. On the other hand, the review timestamp (unixReviewTime) seems unrelated. However, after calculating the correlation coefficient and performing a chi-square test between the timestamp and the review score, we found that their correlation coefficient is 0.0377, and the chi-square test p-value is 2.31*10^-41, which is insufficient to assume that the two distributions are completely independent. A box plot of the timestamp and review scores also shows observable differences in the timestamp distribution for different scores. Therefore, the timestamp is also included as part of the feature vector and is processed using min-max normalization.



For the summary and reviewText, after understanding the common processing methods and researching online suggestions, we decided to first perform standard operations such as removing punctuation, tokenization, removing stop words, stemming, and lemmatization. Then, we used the Term Frequency-Inverse Document Frequency (TF-IDF) method to process the data comprehensively, considering the frequency and commonality of each word. This method assigns higher weights to words that frequently appear in one type of document but rarely appear in others, thereby achieving better classification results. Finally, we used Latent Dirichlet Allocation (LDA) to assign a corresponding topic to each summary and reviewText, aiding in the prediction of the scores.

In practice, based on previous experience (selecting too many feature words is not conducive to the model's performance), we selected 100 feature words and 20 topics for the summary, and 300 feature words and 20 topics for the reviewText. After combining the feature vectors extracted from the previous three variables, the final input feature vector has 475 dimensions (440 dimensions from the feature words and topics of summary and reviewText, two 17-dimensional binary vectors from asin and reviewerID, and one dimension from unixReviewTime).

We then randomly split the data into a training set and a test set in a 9:1 ratio, preparing for training. In this part, we observed that the distribution of overall ratings is uneven, skewing towards the higher end with a bias towards 5-star ratings, which needs to be considered when designing the model.

II. Implementation of Ensemble Learning Algorithms

In this section, I explain my understanding of the two ensemble learning algorithms to be implemented and provide a detailed explanation of the code logic I used for the implementation.

1. Bagging

Bagging is a parallel ensemble learning method. It creates multiple subsets from the original dataset through bootstrap sampling (sampling with replacement). Each subset is obtained by randomly sampling from the original data, ensuring some differences among the subsets. Then, the same learning algorithm is used to train on each subset. Finally, the predictions of all learners are averaged or voted on to obtain the final prediction.

In my implementation of the bagging algorithm, I reserved three hyperparameters: the total number of learners (n_estimators), the type of base classifier (base_classifier), and the sampling ratio for each subset (sample_size). Each time, we randomly sample a portion of the data (0.8 in practice) to train a base classifier and then determine the final prediction using majority voting.

2. AdaBoost

AdaBoost is a sequential ensemble learning method. It builds a strong learner by iteratively training a series of weak learners. In each iteration, it adjusts the weights of the samples based on the results of the previous iteration, placing more emphasis on the samples that were misclassified in the previous round. Each weak learner is trained based on the adjusted sample weights and is given a weight according to its classification accuracy, so learners with lower misclassification rates receive higher weights. Finally, the results of all weak learners are combined through weighted voting to obtain the final prediction.

In this section, we initially overlooked that the AdaBoost classifiers mentioned in the course materials and those implemented in the Python library are designed for binary classification, while we needed to handle a five-class classification problem, which initially led to all prediction results being zeros.

In the final implementation, we first implemented a binary classification SimpleAdaBoost classifier and then transformed the five-class classification problem into binary classifications for all pairwise combinations (a total of 10 pairs). We used 10 SimpleAdaBoost classifiers combined to handle this five-class classification problem. The model has two hyperparameters: the total number of iterations (n_estimators) and the type of base learners (base_estimators).

III. Performance Analysis of Combinations

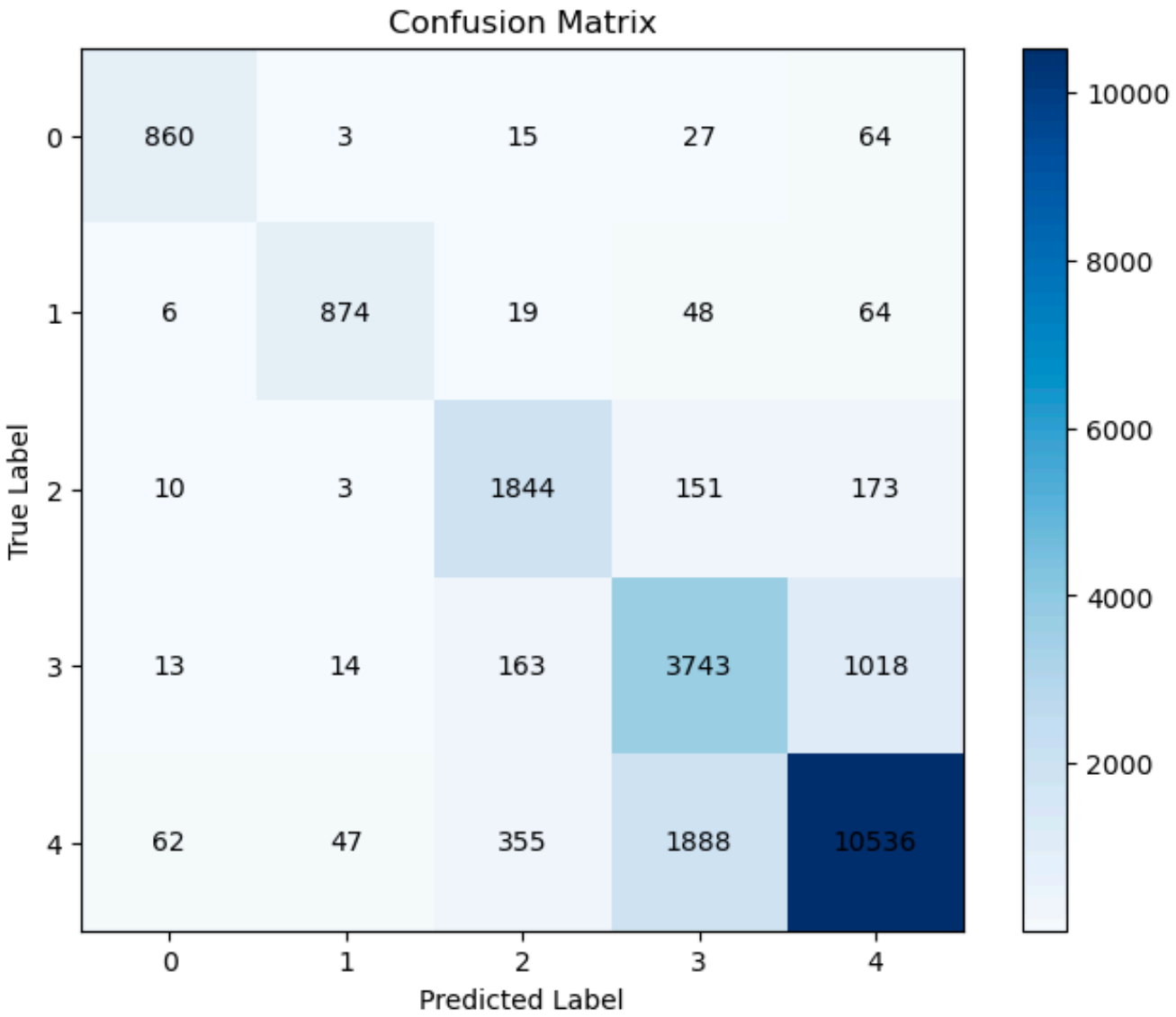
In this section, we report the specific hyperparameters used for each combination, observe the specific performance of each combination, and discuss the considerations during debugging.

To measure model performance and effectiveness, we use accuracy, precision, and recall as quantitative metrics of classification capability. We use a confusion matrix for an intuitive judgment of the classification situation and issues and use MAE and RMSE to quantitatively measure the overall error.

1. Combination of Bagging and SVM

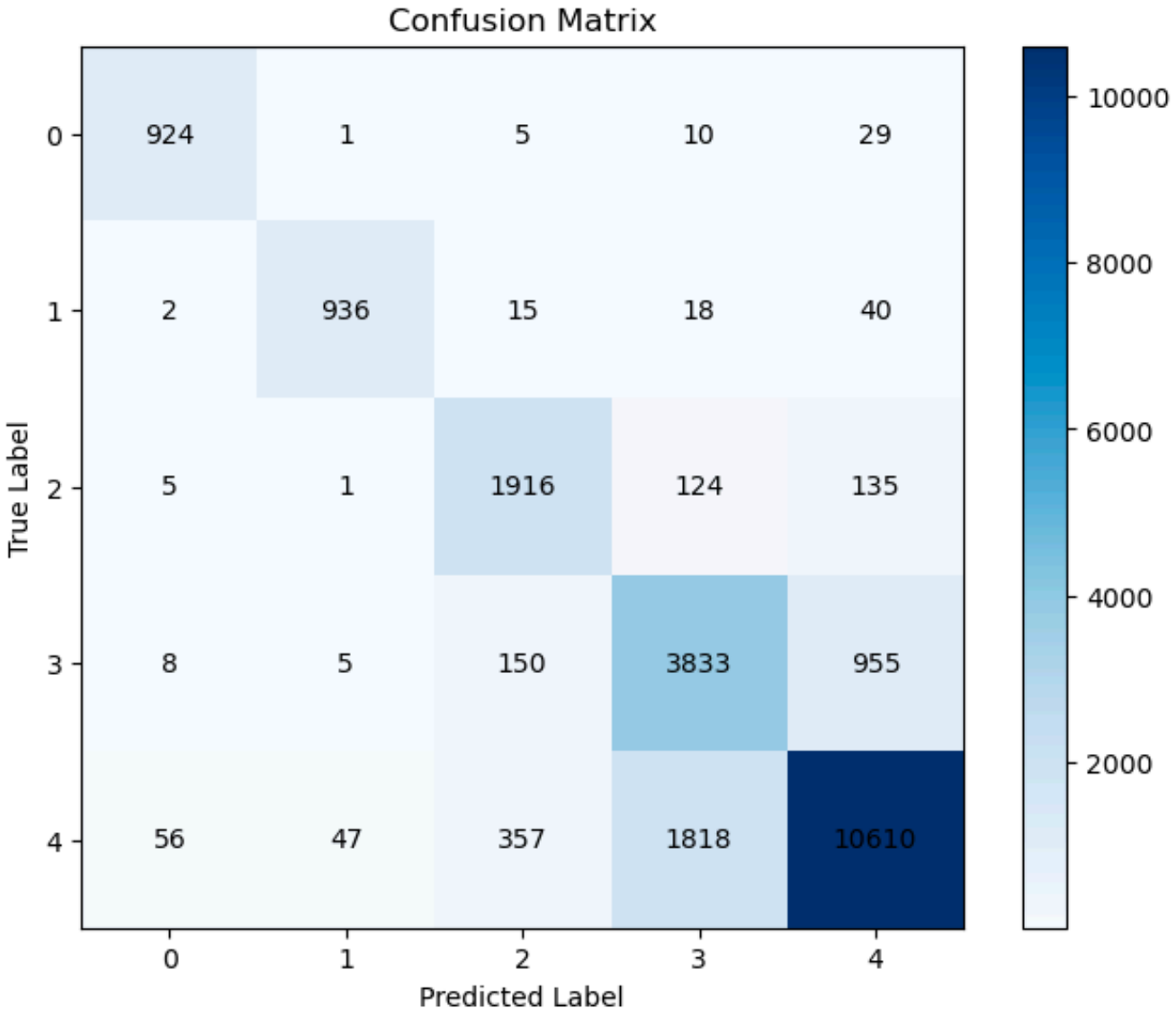
Initially, we did not set the parameters of the base classifier. We set the bagging algorithm to train a total of 15 base classifiers. The initial results were 64.46% accuracy and 58.20% precision, with the SVM using its default parameters.

Noticing that the overall ratings are more concentrated in the higher range, with fewer lower ratings, we needed to correct for the imbalanced data. We adjusted the SVM parameters, using a Gaussian kernel to ensure better classification performance, set the gamma parameter to 0.1, and used the "balanced" method to handle class distribution. The regularization parameter was set to 1 to avoid overfitting. Training with 15 base classifiers again resulted in an accuracy of 81.17%, precision of 82.61%, and recall of 83.42%. The confusion matrix is as follows:



The model shows good classification performance without significant misclassification, mainly encountering issues distinguishing between 4 and 5 ratings. We believe that to improve the model's performance in this area, more effort should be put into data preprocessing, adjusting the number of feature words and topics, as increasing the model's complexity might not have much impact.

To validate our assumption, we kept the parameters the same and increased the total number of base classifiers to 30. The results showed performance improvement, but the overall improvement was not significant (as computation time tripled, but all indicators increased by less than 5%). The accuracy was 82.81%, precision was 84.41%, and recall was 87.11%. The confusion matrix is as follows:

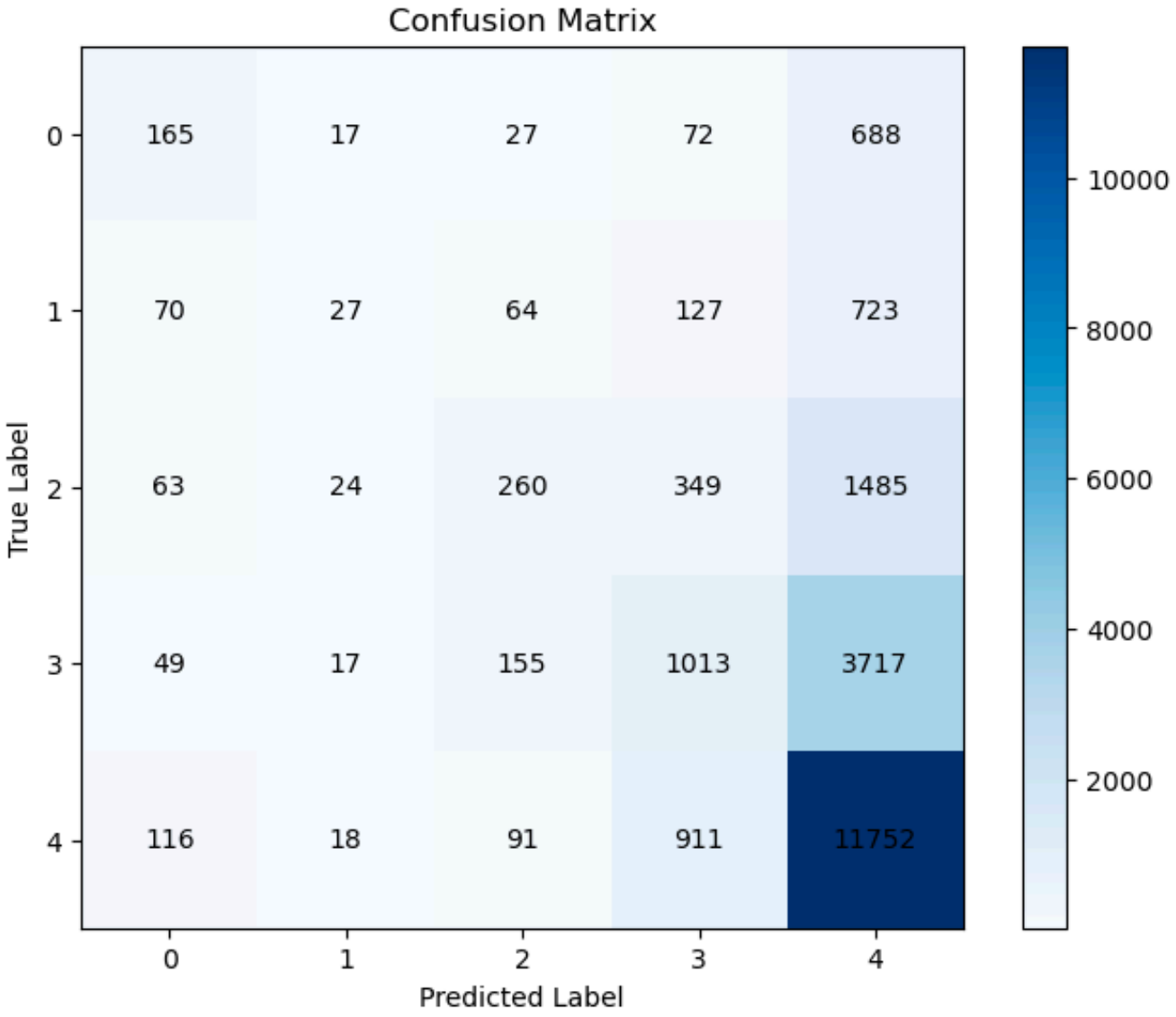


All aspects showed progress compared to the combination using 15 base classifiers. Due to the long computation time, further attempts to modify the number of base classifiers and specific SVM parameters to find the optimal combination were not feasible.

2. Combination of Bagging and Decision Tree

In this part, we trained 30 base classifiers, setting the sample weight of DecisionTree to "balanced" to balance the different rating proportions and using information entropy as the criterion for node classification. We also sampled 80% of the data each time to train the base classifiers.

The final accuracy was 60.08%, precision was 42.07%, and recall was 28.65%. The confusion matrix is as follows:



From this, we can see that the model's accuracy is relatively good, but both precision and recall are low. The confusion matrix reveals that the model predicts too many 5s, converting many 1-4 ratings to 5s. Although the overall accuracy is decent due to the high proportion of 5s in the actual results, the model's classification capability is not ideal.

3. Combination of AdaBoost and SVM

In this part, we used 10 base classifiers and a linear kernel SVM classifier, with weights set using the "balanced" method. Due to the slow computation speed of the SVM, we used the first 20,000 samples from the randomly sampled training set for training and the first 2,000 samples from the test set for testing.

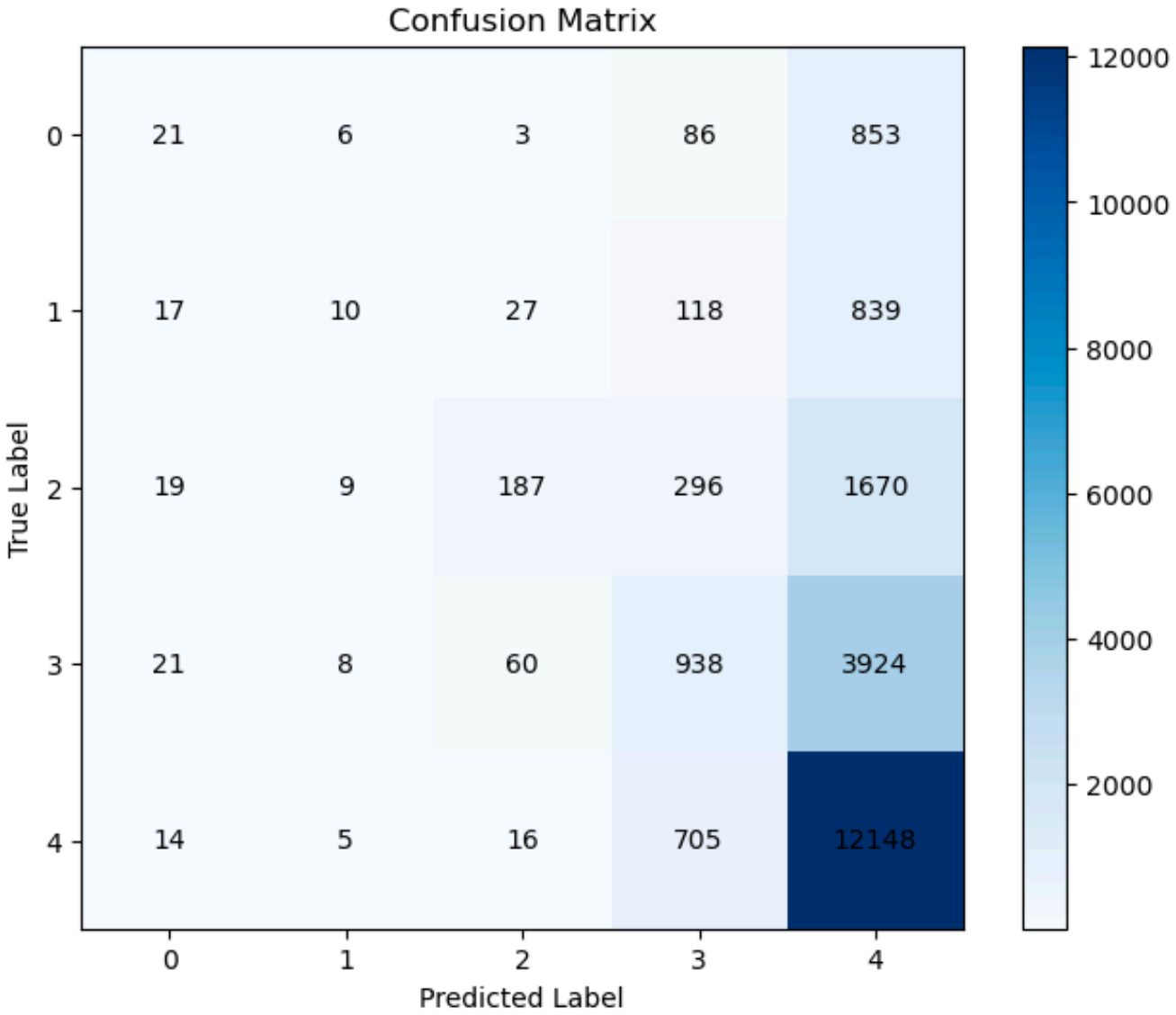
For some reason, regardless of training or kernel type, high training iterations always resulted in predicting all values as 5. Modifying parameters led to predictions worse than predicting all values as 5.

We speculate that the large number of 5-rated samples caused the imbalance, leading to problems with the sensitive AdaBoost and SVM combination. Alternatively, extracting too many high-dimensional feature vectors during data preprocessing may have caused this deviation.

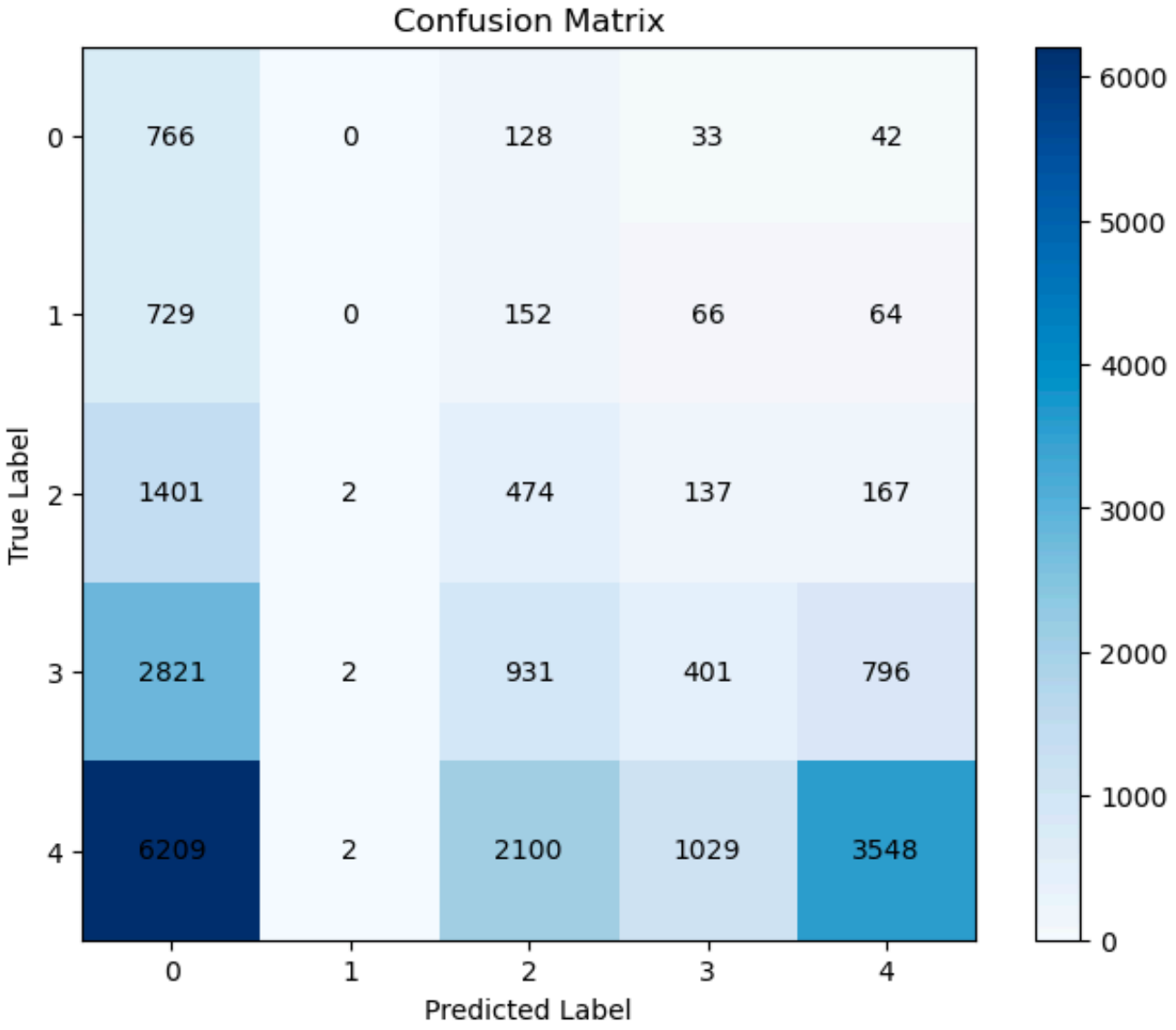
4. Combination of AdaBoost and Decision Tree

In this part, we tried two different combinations of maximum decision tree depth and the number of overall base classifiers.

Initially, we set the maximum depth of the decision tree to 3 and used 10 base classifiers. The final results were 60.47% accuracy, 43.85% precision, and 24.98% recall, similar to previous analysis results, with most predictions being 5. The confusion matrix confirmed this:



To improve the model's performance, we increased the decision tree depth and the number of base classifiers, using decision trees with a maximum depth of 5 as base classifiers and training 30 base classifiers. However, the final results were poor, with all indicators close to 20%, and the overall predictions were mostly 0 (i.e., predicting all as 1). The confusion matrix is shown below. This is a common problem encountered with the AdaBoost classifier: as the base classifiers become more complex and the number of training iterations increases, the final prediction tends to be 0. We believe this is related to the initial value implementation before training or weight adjustment after each iteration but could not solve this issue.



5. Comparative Analysis

In this section, we compare the best performances of the four models. To have a baseline, we introduce a fifth model that blindly predicts all results as 5. The metrics for the prediction results of the five models are as follows:

	Accuracy	Precision	Recall	F1	MAE	RMSE
BaggingSVM	82.81%	84.41%	87.11%	85.57%	0.2168	0.5827
BaggingDecisionTree	60.07%	42.07%	28.65%	29.82%	0.6695	1.228
AdaBoostSVM	58.1%	11.62%	20%	14.70%	0.7645	1.352
AdaBoostDecisionTree	60.47%	43.85%	24.99%	24.52%	0.6833	1.258
All predictions 5	58.58%	11.71%	20%	14.78%	0.7374	1.319

Overall, it can be observed that the combination of the Bagging ensemble learning model and SVM base learner achieved the best results, reaching the highest accuracy, precision, recall, F1 score, and the lowest MAE and RMSE. The other two models performed similarly, except for AdaBoostSVM. The results of the three models, excluding AdaBoostSVM, also surpassed the "All predictions 5" model, indicating that the trained models, except for the AdaBoost and SVM combination, have basic classification capabilities.

We also observed that the indicators measuring classification ability (e.g., accuracy) and those measuring fitting degree (e.g., MAE) are generally negatively correlated, consistent with our basic understanding and enhancing the credibility of the results.

During training, we noticed that the AdaBoost classifier is very sensitive to initialization results and tends to deviate towards predicting all results as 0 (1 rating), consistent with the known sensitivity of AdaBoost to noise and outliers learned in class.

When fixing the ensemble learning algorithm and comparing the performance of the two base classifiers, we found that the SVM classifier's performance was significantly better than that of the DecisionTree when using the Bagging ensemble learning algorithm. We speculate this is because the generated feature vector dimensions are high, making it more suitable for SVM. DecisionTree often performs well on the training set but poorly on the test set. AdaBoost's integrated DecisionTree frequently overfits, achieving 100% accuracy on the training set but lower than the "All predictions 5" model on the test set. Therefore, if using DecisionTree in the future, it is important to control the model's size and input data dimensions to prevent overfitting.

IV. Summary and Reflection

Overall, I believe I performed well in the data preprocessing part of this assignment, qualitatively and quantitatively analyzing the impact of each variable in advance, avoiding issues discovered during model training due to data preprocessing problems.

However, I did not accurately estimate the computational load of training, often waiting four to five hours only to find that training was unsuccessful and had to restart. In similar future tasks, I should first complete a full process with a smaller dataset to estimate the computation time required for the entire dataset. If the time is too long, I should appropriately reduce the data volume (since the additional benefit from increasing data volume has diminishing marginal returns), thereby shortening training time and gaining more opportunities to debug and try different parameter combinations.