Paper for COMP30027 Report

Anonymous

Word Count: 2482(not included reference and captions)

1 Introduction

The main objective of this project is utilizing a machine learning pipeline to predict the rating of a book. This learning is based on various information from a summary of the book. Modelling part consists of two primary stages, pre-processing and model learning, which will be discussed in detail later.

2 Data Overview

The dataset used for training is sourced from a book review platform, containing 23,063 instances, 9 features, and a target variable called "rating_label."

Initially, we conducted a thorough analysis of the dataset to identify potential challenges that might hinder the efficacy of the model training process. By scrutinizing the data, we were able to pinpoint areas of concern, such as missing values, inconsistencies, and imbalanced class distribution, which could negatively impact the model's performance. Subsequently, we employed suitable preprocessing techniques to address and resolve these identified issues.

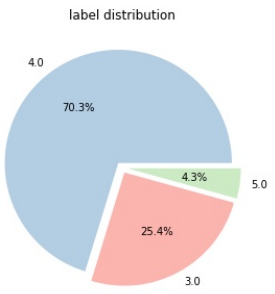

Figure 1: Label distribution

The presented figure illustrates a highly imbalanced distribution of labels within the dataset, where class "4.0" constitutes 70% of the total instances. This imbalance may result in the model exhibiting a bias towards predicting class "4.0".

To address this issue, the application of class weighting strategies can be an effective solution. Indeed, we have incorporated this approach in the subsequent stages of model training to ensure a more balanced representation of classes and to mitigate the model's inclination to predominantly predict class "4.0".

3 Methodology

[1]

3.1 Pre-processing

This section delves into the intricacies of Preprocessing methodologies and their significance in enhancing the quality of input data for machine learning applications.

Feature Name	Feature Types
Name	object
Authors	object
PublishYear	int64
PublishMonth	int64
PublishDay	int64
Publisher	object
Language	object
pagesNumber	int64
Description	object
rating_label	float64

Figure 2: Feature types

Due to the presence of object-type features in this dataset, which cannot be directly learned by the model, feature engineering becomes essential.

To address the issue of missing data, employing the 'fill NaN' method proves to be a viable solution. In this approach, all instances

[1] referred sklearn

Feature Name	if missing value
Name	False
Authors	False
PublishYear	False
PublishMonth	False
PublishDay	False
Publisher	True
Language	True
pagesNumber	Unsure
Description	False
rating_label	False

Figure 3: Missing Values

of missing data are replaced with an 'unknown' label. While it may be feasible to predict the language based on the description, the results obtained through such predictions are not entirely satisfactory, possibly due to translation-related factors. Similarly, accurately predicting the publisher can be challenging. Consequently, we replaced missing values with 'unknown' labels, to effectively handle the missing data in our dataset.

Beyond missing values, our dataset also contains instances of apparent noise. For instance, entries with a 'pagenumber' recorded as zero are likely erroneous. Given that there are only 196 rows featuring a zero 'pagenumber', we elected to remove these rows from our dataset. This decision is predicated on the notion that such a small proportion of data is unlikely to significantly impact the overall integrity or interpretability of the dataset.

Concurrently, the dataset harbors instances of exact duplicate entries. To maintain the authenticity and quality of our data, we have chosen to address this issue by removing these duplicate instances.

3.2 Feature Engineering

The dataset can be divided into three main parts: the document part, which includes features like the book description; the category part, which includes features such as publisher and language; and the directly usable features.

To handle the category part, we employ an OneHotEncoder after addressing any missing values. This transformation enables us to convert nominal values into numeric ones, facilitating the combination of features.

Using the template method, we develop two types of transformations that can be applied to models.

The count vectorizer faces the issue of high dimensionality after transformation, whereas the document vectorizer offers a flexible dimensionality but lacks interpretability.

On the other hand, the doc vectorizer offers greater flexibility in dimensional while ensuring a manageable number of dimensions. It considers the context and order of the text, effectively capturing the similarity between texts.

After combining the transformations of the document features and other features, they may have different scales, making it difficult to compare them directly. To address this issue, we employ a normalizer to ensure consistency in scale.

3.3 Feature Selection

When applying the VarianceThreshold filter to identify more correlated features, it is surprising to observe that none of the features have a variance higher than 0.08 after normalization. This suggests that there is no strong evidence to distinguish labels from these features.

After that, we utilize both chi-square (chi2) and mutual information (MI) selection methods. Chi2 is better suited for nominal features, while all our features are numeric. On the other hand, MI selection is more time-consuming.

In the end of the pre-processing stage, we reduce dimensional further by PCA. This technique not only helps in dimension reduction but also ensures the orthogonality of the features.

3.4 Model selection

After performing pre-processing on the dataset, we have selected five models to work with.

Considering the high dimensionality of the dataset, we decided to start with Random Forest. Random Forest is well-suited for handling high-dimensional data as it can process features in parallel, increasing stability. Additionally, since the initial data is text-based, we opted for MLP due to its strong performance in classifying textual information. Following that, KNN was chosen as our third option because it approaches data in a geometric manner without relying on a predefined function, which sets it apart from the other models.

After implementing these three algorithms, we observed mediocre accuracy, which was similar to the baseline performance (0R). We decided to explore more powerful ensemble models such as XGBoosting. XGBoosting is known for its ability to handle sparse matrices and effectively address challenging points within the

dataset.

Finally, we naturally selected the stacker model, which is considered the strongest ensemble model in our approach.

4 Discussion and Critical Analysis

4.1 Random Forest

Random Forest is an ensemble learning model that contains multiple decision trees, each tree is aimed to trained independently by sampling from the training set with replacement.

The Random Forest algorithm's inherent design negates the necessity for explicit feature selection. Comparative evaluations of models employing various feature selection methodologies, each with varying numbers of features, have demonstrated that these do not yield a model performance superior to that of a Random Forest model without explicit feature selection.

Attributed to its random selection of samples. This stochastic approach not only enhances its stability but also effectively curbs the likelihood of overfitting, a common pitfall in machine learning. Thus, the Random Forest algorithm provides a robust and reliable model, demonstrating superior performance in comparison to many alternative approaches.

Metric(Weighted)	Value
Accuracy	71.31%
Recall	71.31%
Precision	72.92%
F1Score	62.00%

Figure 4: Random Forest Performance

Pred \ True	3	4	5
3	97	1086	0
4	32	3155	0
5	1	190	12

Figure 5: Random Forest Confusion Matrix

As evident from the table, the Random Forest model exhibits acceptable accuracy. However, its F1 score is notably low. As highlighted in our data analysis, the underlying dataset is heavily imbalanced, suggesting that the Random Forest model may not have sufficiently learned the classes '3.0' and '5.0'.

4.2 Multi-Layer Perceptron

[2]

Metric(W)	W/HN	W/LN	UW/HN	UW/LN
Accuracy	67.79%	69.53%	68.47%	70.07%
Recall	67.79%	69.53%	68.47%	70.07%
Precision	66.51%	66.96%	66.57%	67.28%
F1Score	66.59%	67.07%	66.66%	67.30%

Figure 6: MLP Performance
*W/UW: Weighted/Unweighted
HN/LN: High/Low Neurons

The Multi-Layer Perceptron (MLP) is a neuron network model known for its efficacy in tackling complex scenarios, particularly those involving sparse matrices.

However, in the context of this specific dataset, the MLP model did not demonstrate optimal performance. This outcome underscores the fact that model suitability is highly dependent on the characteristics of the dataset.

4.3 Error Analysis and Improvement of MLP

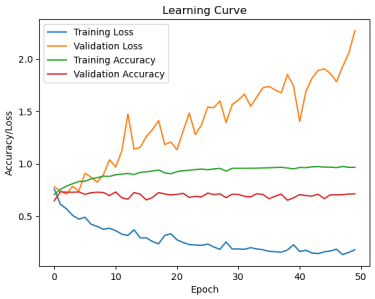

Figure 7: UW/LN MLP learning curve

Initially, The MLP model was enhanced with two hidden layers, comprising 64 and 32 neurons respectively. However, the resulting performance metrics were mediocre, with notably low values for accuracy and F1 score. Furthermore, an increasing trend in the validation loss was observed, suggesting a tendency towards overfitting.

In an effort to address the significant imbalance in the label distribution within this dataset, a strategy involving class weighting was employed. This approach was designed to counter the skewed label distribution, thereby

[2] referred Keras

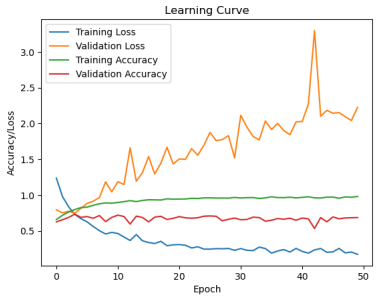

Figure 8: W/LN MLP learning curve

creating a more balanced learning environment for the model.

However, the implementation of class weighting unfortunately led to a decline in performance. The model exhibited difficulties in effectively predicting outcomes within the training set, indicative of a severe underfitting issue. This inadequate performance may be ascribed to a potential deficiency in the complexity of the model's architecture.

Specifically, the model may lack the requisite number of neurons and layers needed to adequately capture and learn from the complexity inherent in the given dataset. This situation emphasizes the critical role that the complexity of a model plays when grappling with intricate, high-dimensional data scenarios.

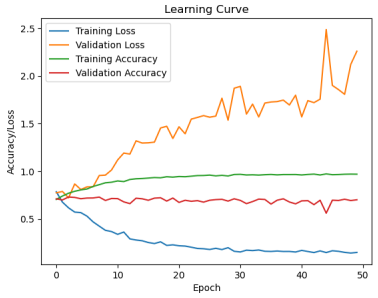

Figure 9: UW/HN MLP learning curve

Given the complexity of the features, an attempt was made to discern deeper relationships through the expansion of the MLP model with additional hidden layers. This decision was based on the premise that a more complex model might better capture intricate patterns within the data. Indeed, a notable improvement in accuracy of train set was observed.

However, as demonstrated in the figure, this increased complexity led to substantial overfitting, while the model appears to have adeptly learned the relationship between features and labels within the training set, it affects the ability of generalization. This disparity underscores the overfitting issue, revealing that the model has likely over-learned the training set's specific characteristics rather than the general structure of the data.

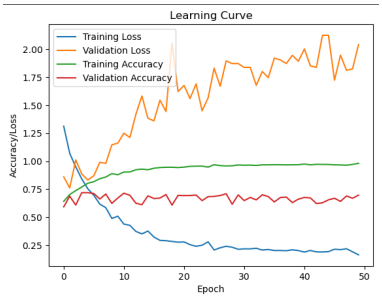

Figure 10: W/HN MLP learning curve

Subsequently, class weighting was applied to the more complex Multi-Layer Perceptron (MLP) model, yet the results remained unsatisfactory. While the model demonstrated an improved ability to predict within the training set, its performance on the test set was markedly poor.

It is postulated that this discrepancy may largely be attributed to the quality of the features since the correlation between the features and label is not strong enough.

4.4 K-Nearest Neighbor

K-Nearest Neighbors is an instance-based model that predicts labels in a geometric manner, which sets it apart from tree-based algorithms. It offers interpretability and simplicity due to its straightforward structure.

One characteristic of KNN is its lazy training approach, which means it does not produce a decision function. However, this approach results in increased computation during the prediction phase.

Furthermore, KNN's decision algorithm provides flexibility when dealing with complex data and non-linear relationships. So, KNN outperforms other geometric methods such as Support Vector Machines in this task.

Metric(Weighted)	Value
Accuracy	71.15%
Recall	71.15&
Precision	65.82%
F1Score	60.30%

Figure 11: KNN Performance

Pred \ True	3.0	4.0	5.0
3.0	18	1118	0
4.0	15	3251	15
5.0	1	182	13

Figure 12: Confusion Matrix of KNN

3

4.5 Error Analysis for KNN

However, KNN is sensitive to noise and unbalanced label distributions. As mentioned in the data analysis, the dataset exhibits significant class imbalance. This directly affects the performance of KNN, particularly reflected in the recall score. KNN tends to discriminate major labels while overlapping significantly with the others. Since in that region, the instance of "4.0" occurs too frequently and confuse the prediction.

4.6 XGBoost

[4] XGBoost combines gradient boosting with decision trees. Like other boosting models, XGBoost focuses on correcting the errors made by the model in the previous iterations.

In our task, XGBoost proves to be one of the most effective models for handling the dataset, particularly for cases that present high classification difficulty. Boosting algorithms, such as XGBoost, excel in addressing such challenges. Additionally, XGBoost has a superior approach for handling the sparse matrix created during pre-processing.

Metric(Weighted)	Value
Accuracy	73.12%
Recall	73.12%
Precision	71.98%
F1Score	65.40%

Figure 13: XGB Performance

As a result, XGBoost achieves an accuracy exceeding 73% on the validation set and 72.492% on the test set. XGB is good at distinguishing the label "4.0" instance, while there is significantly an issue with the label "3.0". There might be a under fit that XGB doesn't learn about the label "3.0" instances enough.

4.7 Error Analysis of XGB

XGBoost, like other tree-based algorithms, since the trees are not parallel, is susceptible to noise and the risk of overfitting. To mitigate this issue, we incorporate a Bagging method to reduce the model's variance and improve generalization.

[4]referred XGBoost

Pred \ True	3	4	5
3	129	1012	0
4	62	3196	3
5	5	147	19

Figure 14: Confusion Matrix of XGB

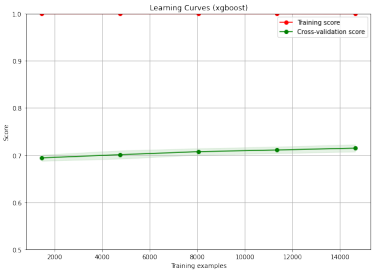

Figure 15: the learning curve of XGBoosting, showing it overfitting now, but improving. More instances for training may perform better

In order to enhance the performance of XGBoost, it would be beneficial to increase the number of instances in the dataset. The learning curve analysis indicates that the model is currently overfitting, but improving while the size of data increase, implying that it can be improved by expanding the dataset size.

4.8 Stacking Model

Stacker, as a combined model consisting of multiple base classifiers and a meta classifier, offers a vast number of potential combinations that could yield good performance. To simplify the process, we select certain models that have shown promise in previous research, including Logistic Regression, K-Nearest Neighbors, Random Forest, XGBoost, and MLP. Furthermore, we conducted experiments with these models as the meta classifier and discovered that XGBoost and Logistic Regression yield good performance for this task.

Metric(W)	Value(XGB)	Value(LGR)
Accuracy	71.94%	71.73%
Recall	71.94%	71.73%
Precision	72.36%	70.01%
F1Score	61.35%	60.67%

Figure 16: Performance of Stackers

In our practical implementation, we start by using tuned base classifiers. However, we ob-

served that even adding the same model with different hyperparameters can enhance the performance of the stackers. Additionally, incorporating a bagging ensemble technique for the meta classifier effectively reduces the variance. Consequently, the stacker demonstrates performance on par with the best-performing base model in the task and successfully mitigates variance.

Pred \ True	3	4	5
3	40	1101	0
4	12	3247	2
5	1	167	3

Figure 17: Stacker_XGB Confusion Matrix

4.9 Error Analysis of Starkers

Theoretically, stackers should avoid some mistakes made by the base classifiers and provide an optimal answer at the meta layer. However, in this case, the performance of the stackers does not exhibit significant improvement. We speculate that this could be due to similar mistakes being made in the base layer, indicating areas of confusion where correct predictions are rare.

Stackers have a high computational complexity, particularly when performing cross-validation. While utilizing the 'StackingCVClassifier' and conducting double-layer cross-validation for more stable results, we opt to evaluate the performance using a holdout set.

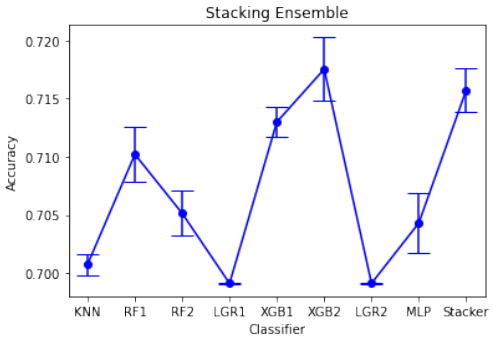

Figure 18: Error bars for base classifiers and stacker

The stacker predicts "4.0" even more than base classifiers. As the model becomes more familiar with label "4.0", it achieves a high precision in distinguishing it, but does not show improvement in recognizing other labels.

5 Conclusions

During the result process, multiple models were employed to predict the book ratings, with the highest achieved accuracy being 73.21% (while get 72.49% on test set). By employing grid search with cross-validation, the model achieves greater stability and reduces bias, resulting in more reliable performance.

It's worth noting that in the training set, there is significantly unbalanced. This implies that 0r baseline would achieve an accuracy of 70.3%, thereby making the incremental improvement rendered by our models less significant.

This result can likely be ascribed to two predominant factors:

The highly skewed distribution of labels, which inherently biases the model to predict "4.0" more frequently. To mitigate such imbalance, more data from label "3.0" and "5.0" should be collect.

Also, the deployment of more sophisticated models capable of handling such situations could be considered.

The features may not be providing sufficient informative content for the model to discern meaningful patterns. While using text processing techniques, this may have introduced redundancy due to the inclusion of non-informative words. Therefore, implementing more refined text processing methodologies could be beneficial.

Although increasing the class weight of minor label can aid in capturing the features specific, there exists significant overlap between instances labeled as "4.0" and others. Therefore, incorporating additional features becomes necessary to achieve a clearer separation between these two labels.

Furthermore, the introduction of additional, potentially informative features (such as book genre, price, etc.) could enhance the utility of the feature set.

In conclusion, achieving better prediction quality relies on balancing the label distribution and identifying more correlated factors. Given the promising nature of this topic, it is worthwhile to further develop both the dataset and the model in the future.

6 Reference

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

Keras: F. Chollet et al., "Keras," https://keras.io, 2015.

XGBoost: Chen, T., Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785-794). ACM. https://doi.org/10.1145/2939672.2939785