COMP30027 A2 Written Report

1. Introduction

In this report I will be talking about my findings on the book rating dataset and how the two models I trained behaved and performed. The first model I will talk about is a decision tree model and the second model is an SVM. They both performed similarly in terms of accuracy but with slight, but I think important, differences which I will talk about in detail. The runtimes were a major difference however as the Decision Tree models were much quicker to be trained but in return performed worse than the Support Vector models.

2. Assumptions and Data Used

Upon first inspection of the dataset, it was clear that there were many redundant features measured such as publish dates, publishers, and languages. In my opinion the most important features would be either the book's name, the author's name or the description of the book since they all relate to logical assumptions; similar book names should be rated similar, same authors will produce a similar level of quality work, and similar contents in books will be similarly rated. Before training my hypothesis was that out of these assumptions the strongest would be the author assumption since it made more sense to me logically.

In both models trained I decided to use the same dataset(s) in order to be able to accurately compare the performances of both models. The datasets used were the sparsity vectors train_authors_vec, train_name_vec and train_desc_vec. I decided these would be useful files to use for the training as a) they contained the most important information about each book and b) they were sparsity vectors meaning training times would be minimal.

3. Model 1: Decision Tree

The first model created was a decision tree. The initial model was trained with the default hyper parameters to get a baseline for future comparison and also get a feel of what each hyperparameter did. While training the model, I started with the train_name_vec dataset. I initially was only using a simple train/test split but quickly decided against it and switched over to using 5-fold cross validation to measure the

model's performance with better consistency. With default hyperparameters the DT model only performed at 63% accuracy.

I then started manually tuning the model, from my trials I concluded that changing the criterion, splitter, min_samples_leaf, and min_samples_split did not produce noticeable changes in accuracy so for the rest of the training and tuning process I left these hyperparameters out. What did have a major impact on the model was max_depth. I quickly found out that this parameter in particular both impacted training time and accuracy substantially and got me to around 0.7 accuracy if I used values within a reasonable range (2-7). I also found that max_leaf_nodes also played a role in improving the decision tree, although not as much as max_depth. A hyperparameter that decreased performance slightly (-0.003) but improved training times substantially (by around 6 seconds consistently) was max_features which made sense since this limits the tree's scope of interest, potentially missing important relationships.

After finding the hyperparameters that improved the accuracy of the model I decided to write a function to iteratively work through a list of potential values for each parameter (max_depth and max_leaf_node) and trained 3 separate models for each dataset, train_name_vec, train_auth_vec and train_desc_vec and got the ideal values for each of them, here are my findings;

For the name dataset the function returned that the best values for max_depth and max_leaf_node were 3 and 7 respectively. Similarly for the author set they were 5 and None, and for description they were 3 and 5. To my surprise they all returned very similar accuracies when I put these values in for each model; 0.7058 for name, 0.7061 for author and 0.7044 for description, which meant they can all predict the rating to a similar degree.

3.1 Majority Voting?

Since all three Dt models had similar accuracies, I thought it would be a good idea to implement a simple majority voting system and so I wrote a function that counted the occurrences of each value from the 3 prediction lists and put the majority vote into a separate folder. I then measured the prediction accuracy with this list hoping I could get a boost in prediction accuracy, however to my surprise the accuracy was usually either as good as the worst model or slightly better, but never better than the best model, and so I decided to not implement this.

```
voted accuracy =  0.7066984608714503
name acc =  0.7090830262302189
auth acc = 0.7095165835681769
desc acc =  0.7064816822024713
```

Figure 1: accuracy comparison

3.2 Further tuning and more findings

As I continued to work with the dataset I decided to look deeper into the predictions, I noticed that each separate model was predicting **a lot** of 4.0 ratings and very little 3.0 or 5.0 ratings. I found this to be quite strange, until I ran a counter function on the training dataset and saw that 70% of the data was rated 4.0. This was quite disappointing as it meant that my models were performing only as good as a 0R model – in terms of accuracy at least. With this new founding I decided to generate a classification report for both the "tuned" model and a default model for the author vector dataset, below are the results.

	precision	recall	f1-score	support
3.0	0.42	0.32	0.36	1164
4.0	0.75	0.75	0.75	3242
5.0	0.14	0.34	0.20	207
accuracy			0.62	4613
macro avg	0.44	0.47	0.44	4613
weighted avg	0.64	0.62	0.63	4613

Figure 2: classification report for default hyperparameters

	precision	recall	f1-score	support
3.0	0.92	0.01	0.02	1164
4.0	0.71	1.00	0.83	3242
5.0	0.71	0.05	0.09	207
accuracy			0.71	4613
macro avg	0.78	0.35	0.31	4613
weighted avg	0.76	0.71	0.59	4613

Figure 3: classification report for tuned hyperparameters

As it can be seen above, since I tuned my models using an accuracy-based approach, I unknowingly lowered the f1-scores significantly, especially for 3.0 and 5.0 ratings. I then reran my tuning algorithm to get better hyperparameters and this time, it returned that the best hyperparameters for max_depth and max_leaf_nodes were None for every model which made sense as these would be limiting the model's decision making substantially.

As I learned that the dataset was unbalanced in terms of class distributions I decided to use a Support Vector Classifier model for my next model as it should be better in these situations.

4. Support Vector Class

Knowing that the SVM model should handle unbalanced class proportions better than the DT model. I wrote a tuning function that ran through given values for the parameters C, kernel, degree, and weight as I believed these would have the biggest impact on the performance of the model. Knowing that SVM will take longer to train each model I decided to only run the training function on the author vector dataset since a) it gave me the highest accuracy in previous but more importantly b) it would result in the lowest run time out of all the datasets since it contained less data.

4.1 Tuning based on accuracy

I did some basic training on the author vector dataset to see if the model would perform better than the DT counterpart, and for the most part it did, not just in terms of accuracy but also in f1 scores. Below is a classification report I got from a test run.

	precision	recall	f1-score	support
3.0	0.56	0.16	0.24	1127
4.0	0.74	0.95	0.83	3285
5.0	0.63	0.15	0.25	201
accuracy			0.72	4613
macro avg	0.64	0.42	0.44	4613
weighted avg	0.69	0.72	0.66	4613

Figure 4: SVM tuned for high accuracy

As it can be seen from figure 4, this model got a slightly higher accuracy when compared to the DT model from the same dataset, but it also had a better average f1 score.

4.2 Tuning based on f1 score

I wanted to compare a model tuned for maximising the f1 scores for each given hyperparameter against the model trained on the accuracy evaluation and below is the classification report from this model.

	precision	recall	f1-score	support
3.0	0.47	0.20	0.28	1127
4.0	0.74	0.91	0.82	3285
5.0	0.44	0.23	0.30	201
accuracy			0.70	4613
macro avg	0.55	0.45	0.47	4613
weighted avg	0.66	0.70	0.66	4613

Figure 5: SVM tuned for f1 score

This graph shows even when tuned for a higher f1 score, my model is still predicting with around the same accuracy score, and I couldn't get the f1 scores quite high enough as I would have liked. Nevertheless, I feel much more confident in this SVM model than all the other models.

5. Conclusions

I found that the SVM model worked much better when compared to the DT model but took **much** longer to train since the algorithm had to constantly keep running until convergence, and depending on the parameters this sometimes took multiple minutes. Because of this, my tuning function took hours to complete. However, although it took a long time to train, I think it was worth the time spent as the decision tree model was behaving *almost* like a 0R model and I couldn't really say if it really learned much from the dataset. I also found out that the reason why I had a 7% jump in accuracy while my initial tunings for the Decision Tree model was that by limiting the depth of the model, I was getting it closer to a 0R model -since the accuracy got to roughly 70% but the f1 scores plummeted down.

I believe that since the dataset was very imbalanced, neither of the models performed as well as they could have but I still managed to train a model that predicted the classes to some success and perhaps in the future I will need to find ways of forcing the dataset to be more balanced by potentially keeping some entries of the majority class out of the training set.