COMP 47460

Assignment 2

Brian McMahon

15463152

UCD School of Computer Science
University College Dublin

November 29, 2024

Question 1.1

Evaluate the performance of three basic classifiers on your dataset:

- Decision Tree
- Neural Network
- kNN (k=1)

Carefully consider the evaluation measure(s) that you use for this exercise and justify why you selected the particular evaluation measure(s). Use the Weka Vote ensemble method (meta -> Vote) to combine the Decision Tree, Neural Network and 1-NN classifiers. Evaluate the performance of the Vote ensemble method with 3 different combination rules (there are 6 possibilities: Average of probabilities, Product of probabilities, Majority voting, Minimum probability, Maximum probability, Median). Provide a justification for the difference in accuracy when using different combination rules.

0.1 Basic Classifiers

0.1.1 Data Normalisation and Accuracy Metric Policy

We begin by providing a policy for cleaning the provided Spotify data. Most features in the dataset are already normalised between 0 and 1. For consistency and to avoid certain features from overpowering our classifiers, in KNN for example, we decide to normalise all of the input features. Models were trained on both base features and normalised input features to evaluate the effect and it was found that the difference was negligible for our three classifiers. Even so we continue to use normalised data for the remainder of the experiments. Results of the un-normalised data can be seen in table 12 in appendix A section 1.1.

The output classes (e.g., pop, rock, jazz) do not have a clear linear distinction or inherent ordering. For example, there is no greater significance in misclassifying "jazz" as "pop" versus "EDM." This lack of hierarchical or consequential relationships between the classes reduces the importance of metrics like recall for individual classes. Instead, overall accuracy becomes the primary metric of interest. Additionally, unlike the previous assignment in the medical domain where misclassification can have serious consequences, there are no severe implications for selecting the wrong class in this context. Therefore, our focus is on optimizing the overall accuracy of class predictions. Alongside this our goal is to create base classifiers which are in some disagreement, to leverage the effects of ensemble methods.

0.1.2 Decision Tree

We use the J48 algorithm the implementation of the C4.5 decision tree algorithm in Weka. The tree is trained on the input data using 10-folds cross validation. The accuracy of the decision tree can be seen in table 2. We experiment more with decision tree parameters in section 1.1.3

0.1.3 kNN (k=1)

We train a 1 nearest neighbour classifier on the input data. Again we use 10 folds cross validation. The results are shown in table 2. Different distancing schemes (no-diff, 1/d, and 1-d) were evaluated and it was found that an inverse distance policy $\frac{1}{D}$ was most accurate by our metrics defined in section 0.1.1, so this distancing policy was used. We also investigate the performance of 3-NN and 5-NN setups and the results are shown in table 13 in appendix A section 1.1. It was found that these models perform slightly better than 1-NN, this could imply that music in similar genres is clustered but there is high variance between individual tracks, making 1-NN more likely to misclassify the songs. However as instructed we proceed to use a 1-NN configuration.

0.1.4 Neural Network

We train several different network configurations to evaluate the best configuration. The results of this is shown in table 1. over-fitting is also a common problem in neural nets as they can arbitrarily generate features in more complex network configurations. This means that network size must be considered to avoid over-fitting. Several methods exist for evaluating if the model is over-fitting, alongside general rules of thumb for configuring a network. Three equations are chosen eq. (1) [1], eq. (2) [2], and eq. (3) [3] for estimating the optimal number of hidden neurons. Where N_h is the number of hidden neurons, N_i is the number of input features N_o is the number of output features, N_s is the number of samples of training data, N_w is the number of weights in a network, and α is some scaling factor between 2 and 10.

From eq. (1) and eq. (2) we determine a starting estimate of N_h as 8 and 72 ($\alpha = 5$) respectively. These are not deterministic values but just a good place to start guessing. We then use eq. (3) to give us a general sense of each of the models evaluated in table 1. For fig. 1 and fig. 2 we see the rate of accuracy increase is diminishing as the networks grow larger, indicating that for some of the larger networks over-fitting is present. Overall, using our metrics we decide to use a 1 hidden layer network with 17 neurons (t=17). We feel that this is a good balance between the equations which have helped guessing and the accuracy gain metric. Notice that we have also investigated several hidden configurations with two layers. This was to establish if less neurons could perhaps perform better if they were in a configuration allowing for higher order features, the effect of shape on the network was also analysed i.e. pyramid/reverse pyramid. However, the multilayer configurations seemed to perform worse compared to their single layer counterparts.

$$N_h = \sqrt{N_i \times N_o} \tag{1}$$

$$N_h = \frac{N_s}{\alpha \times (N_i + N_o)} \tag{2}$$

$$N_s \geq 30 \times N_w \tag{3}$$

Hidden Layers	N_w	Accuracy	EDM	Latin	Pop	Rap	Rock
a (=9)	153	0.547	0.598	0.328	0.393	0.679	0.734
t (=17)	289	0.583	0.675	0.369	0.410	0.703	0.751
i (=12)	204	0.572	0.656	0.351	0.397	0.700	0.748
o (=5)	85	0.526	0.571	0.324	0.361	0.659	0.715
60	1020	0.589	0.671	0.398	0.439	0.686	0.744
120	2040	0.584	0.660	0.410	0.435	0.675	0.733
2,2	38	0.482	0.543	0.167	0.218	0.729	0.747
4,4	84	0.518	0.572	0.336	0.296	0.672	0.713
10,8	240	0.568	0.642	0.340	0.389	0.685	0.779
2,15	129	0.489	0.580	0.241	0.253	0.680	0.678
5,7	130	0.532	0.586	0.299	0.356	0.691	0.722

Table 1: Performance of Multilayer Perceptron Configurations with Different Hidden Layers

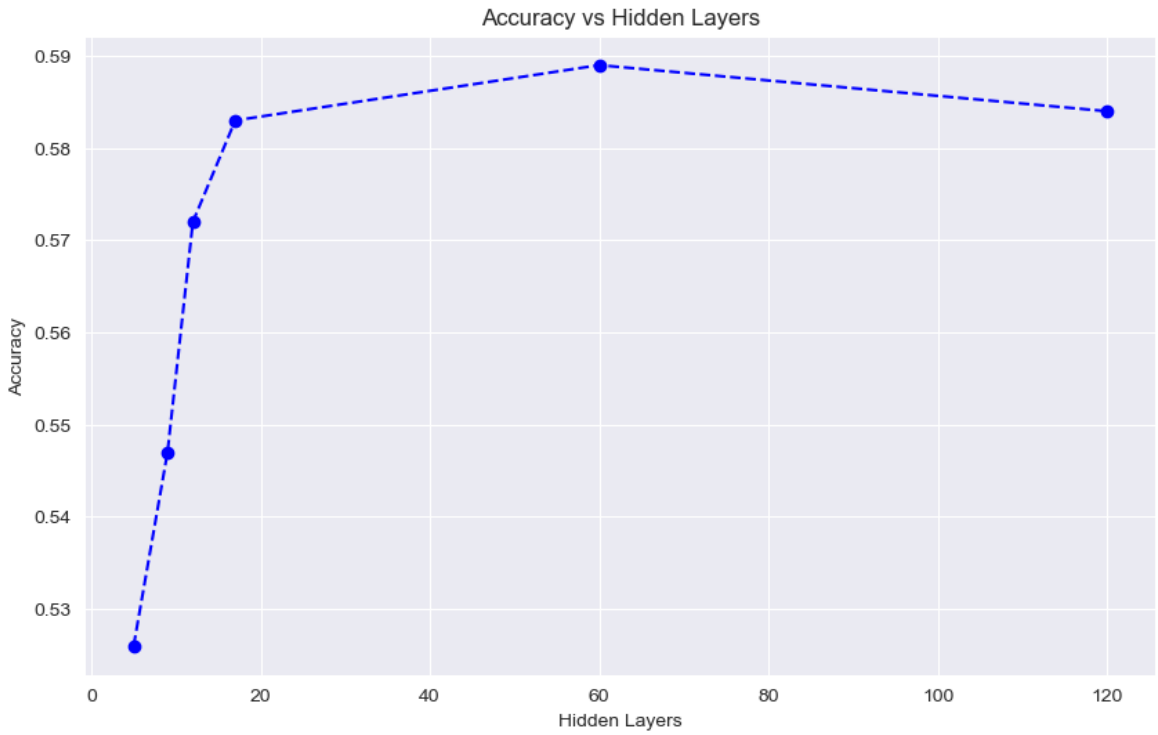

Figure 1: Accuracy gain as hidden layer neurons are increased

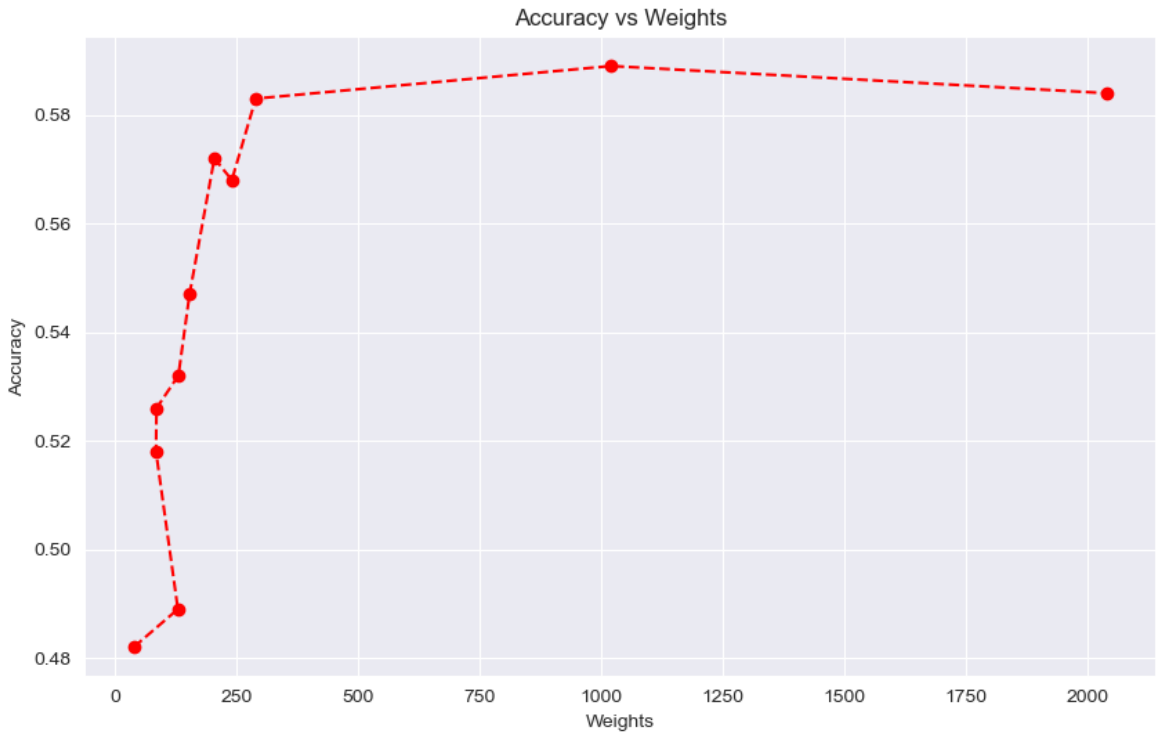

Figure 2: Accuracy gain as number of weights in the network are increased, includes multi-hidden layer networks.

0.1.5 Classifier Results and Discussion

The accuracy results of the three chosen classifiers for our ensemble are shown in table 2. We notice that classifiers seem to have their own speciality, or at least perform differently when predicting output classes of music. For example while the 1-NN classifier performs on average the worst, it actually performs best for Latin music class. This will allow us to leverage the power of vote ensembles to use the best predictors in each situation.

Classifier Type	Accuracy	EDM	Latin	Pop	Rap	Rock
J48	52.21%	0.605	0.411	0.332	0.603	0.655
1-NN	48.24%	0.570	0.421	0.326	0.501	0.591
Perceptron	58.33%	0.675	0.369	0.410	0.703	0.751
Best Performer	**Perceptron**	**Perceptron**	**1-NN**	**Perceptron**	**Perceptron**	**Perceptron**

Table 2: Classifier Performance: J48 Decision Tree, 1-Nearest Neighbour (1-NN), Perceptron, and Best Performer; with normalised input data. Only models selected for ensemble are shown.

0.2 Ensemble

In this section we evaluate each of the weighting schemes for the weka ensemble class by training 5 different ensemble classifiers. We examine the benefit and downside of each voting policy and offer a conclusion about the most optimal policy.

0.2.1 Average of Probabilities

Average of probabilities (AVG) averages all of the probabilities from different classifiers in the ensemble for each prediction class. The class with the highest average of probability is then selected as the chosen class. This method works best when all classifiers are reasonably accurate, if one classifier performs poorly it can affect the overall decision. "Specialist" classifiers, classifiers who perform well on a specific class, are less effective in this voting regime, as their predictions are diluted.

For our classifiers this scheme appears to work poorly compared to others, perhaps indicating a stronger disagreement between classifiers. For example the Perceptron is significantly better than 1-NN at predicting Rap music class, 0.703 to 0.501 respectively (20%). This strong disagreement is weakened by the averaging scheme, making the AVG ensemble poor at predicting the rap class in comparison to the maximum probability scheme.

0.2.2 Product of Probabilities

Product of probabilities (PRD) computes the product of all the probabilities generated by the classifiers. The highest probability class is then chosen. This method gives stronger weighting to classifiers which have extreme outputs, due to the feature of multiplication. eg. $(0.5 \times 0.5 \times 0.6 = 0.15)$ and $(1.0 \times 0.1 \times 0.5 = 0.05)$. Even though the total probabilities sum to the same, the extreme probabilities are more strongly weighted and impact the output class more.

Our ensemble model in this scheme performs the best compared to the other schemes. This indicates that perhaps our individual classifiers are outputting significant differences. Also none of our classifiers predict classes with a very low probability. This scheme is more effective when models have strong confidence, and do not exhibit overconfidence.

0.2.3 Majority Voting

Majority voting (MAJ) using a voting scheme, where each classifier in the ensemble has a vote. The class with the most votes then becomes the predicted class. This method is most effective when most of the classifiers are predicting the correct class and we're not relying on "specialist"

classifiers. The weakness is that this scheme gives no credit for classifiers with high confidence compared to their less confident ensemble buddies.

For our ensemble this method performs poorly, having the worst overall accuracy. Again this seems to indicate that out classifiers are somewhat "specialist" in their regions of prediction, and the predictive power of these more confident classifiers is being reduced.

0.2.4 Minimum Probability

Minimum probability (MIN) selects the minimum prediction probability for each class, from the ensemble classifiers. The class with the highest probability is then chosen. This means this scheme is highly sensitive to classifiers who are confident that the sample is not part of the class. This method works well if individual classifiers are not confident in their predictions, but some classifiers are confident in the sample being excluded (anti-prediction). However this scheme is weak to classifiers which wrongly predict low probabilities.

In our ensemble it seems that no classifier is strongly confident in "anti-prediction", so this scheme should work well. When examined in table 3 we see that indeed it does perform well compared to our other schemes in terms of overall accuracy.

0.2.5 Maximum Probability

Maximum probability (MAX) selects the maximum prediction probability for each class, from the ensemble classifiers. The class with the highest probability is then chosen. This is the same as saying the most confident classifier gets to pick. This method works well when you have "specialist" classifiers which are extremely good in one domain. This scheme is weak when classifiers incorrectly return strong probabilities.

For our ensemble this model performs well yielding a high accuracy. This again indicates our separate classifiers are somewhat specialist. However it does perform more weakly when none of the classifiers exhibit strong probabilities. For example in the pop music class, all classifiers perform similarly and poorly, table 2. This ensemble scheme performs more weakly in this pop domain than MIN or MAJ indicating that they handle these uncertain classifiers better.

0.2.6 Conclusion about Ensemble

We conclude by examining table 3 and considering our accuracy metrics. We have already established that there are less downsides with type 2 errors when associated with music prediction. Rather overall accuracy should be our main goal. i.e. There is no greater significance in misclassifying "jazz" as "pop" versus "EDM". So our chosen "best" ensemble is **Product of Probabilities** (PRD). PRD performs well across all classes and boast the highest overall accuracy. It also maintains a nice balance of certainty and uncertainty, as sometimes our classifiers are very certain, and sometimes they are somewhat inconclusive.

Combination Rule	Key	Accuracy	EDM	Latin	Pop	Rap	Rock
Average of Probabilities	AVG	91.89%	91.9%	88.0%	86.4%	94.8%	98.7%
Product of Probabilities	PRD	94.85%	93.2%	92.7%	92.6%	96.9%	99.3%
Majority Voting	MAJ	90.31%	90.6%	85.6%	83.9%	94.0%	97.9%
Minimum Probability	MIN	94.74%	92.7%	92.4%	92.9%	96.9%	99.4%
Maximum Probability	MAX	94.64%	94.0%	92.6%	91.0%	96.8%	99.2%
Best Performer		**PRD**	**MAX**	**PRD**	**PRD**	**PRD**	**MIN**

Table 3: Performance of Different Combination Rules and Voting Schemes

Question 1.2

Return to the full data set and apply ensembles with bagging using the three classifiers from Task (a). Investigate the performance of these classifiers as the ensemble size increases (e.g., in steps of 2 from 2 to 20 members). Using the best performing ensemble size, investigate how changing the number of instances in the bootstrap samples affects classification performance.

0.3 Ensemble Size

We have investigated each of the classifiers; j48, kNN, and Perceptron, in a bagging model. The results from varying the ensemble size can be seen in fig. 3 and table 4. For each of our classifiers the optimum ensemble size is 20 for the J48, 1NN, and Perceptron classifiers respectively. Note that the Perceptron gains very little accuracy for increased ensemble size, so in the next section we choose to use the ensemble n=12, as n=20 is very costly to train for our chosen network (17 hidden neurons). Perceptron with bagging and n=20 takes about 3 hours to train on my machine.

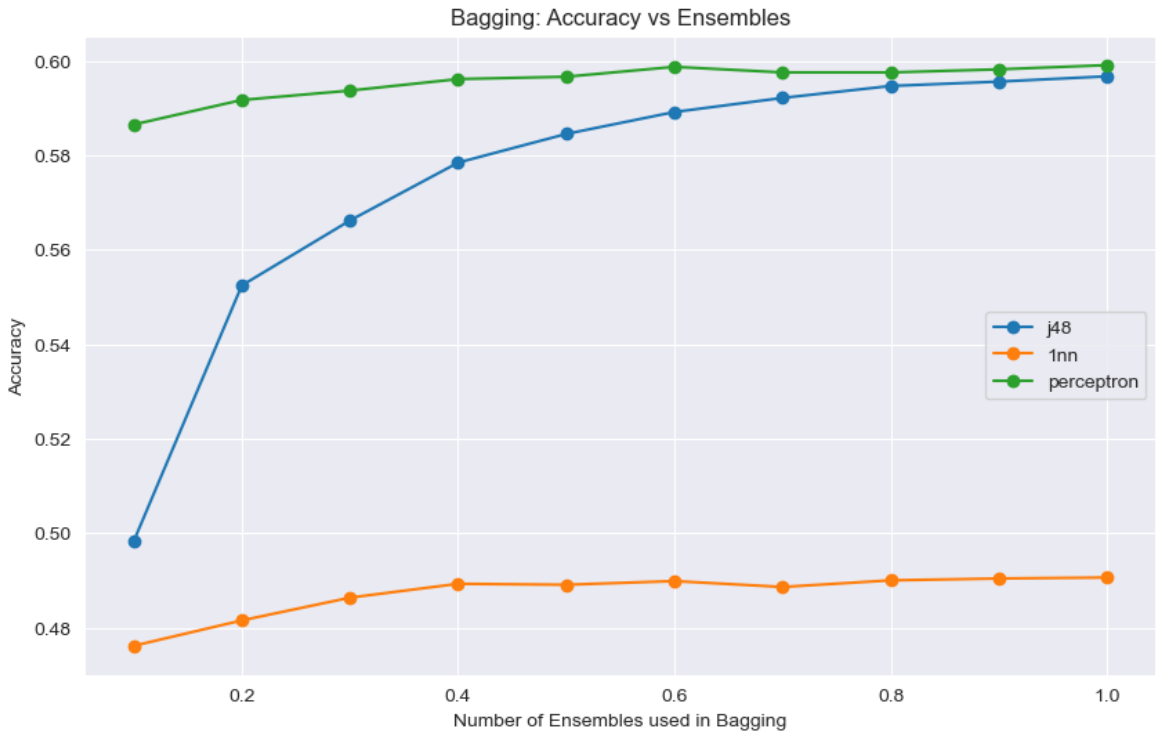

Figure 3: Performance of classifiers whilst varying the number of ensembles used in bagging.

Ensemble Size	Accuracy		
	J48	1NN	Perceptron
2	49.83%	47.62%	58.65%
4	55.24%	48.16%	59.17%
6	56.62%	48.64%	59.37%
8	57.84%	48.93%	59.61%
10	58.44%	48.91%	59.66%
12	58.91%	48.99%	59.87%
14	59.21%	48.86%	59.75%
16	59.47%	49.00%	59.75%
18	59.56%	49.04%	59.82%
20	59.67%	49.06%	59.91%

Table 4: Performance of classifiers whilst increasing the number of ensembles used in bagging.

0.4 Bag Size

In this section we show the effect that varying the bag size has on model accuracy. For this experiment we hold the number of ensembles stationary and vary the bag size parameter. We use ensemble size of 20, 20, and 12 for J48, 1NN, and Perceptron respectively. We use 12 ensembles for the Perceptron as it achieves near-optimal accuracy, while the increased training time at 20 ensembles makes its use impractical. The results of varying bag size can be seen in fig. 4 and table 5. In Question 1.4 we investigate the hyper parameters of our models to optimise our accuracy and generalisation, so the discussion of best parameters is left until then, the body of work in optimisation can be seen in Appendix-A section 1.1.3. However, we do note that the two classifiers that seem to be performing best in the bagging methodology are Decision Trees and Multi-layer Perceptron.

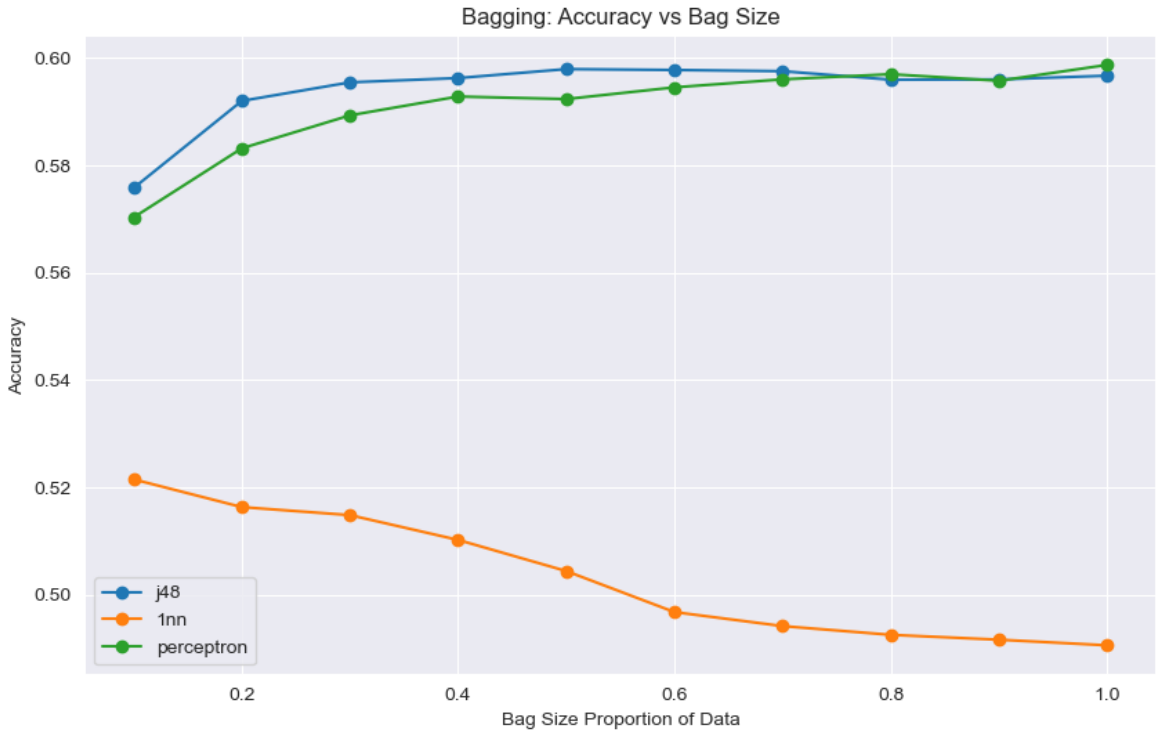

Figure 4: Performance of J48, 1NN, and Perceptron classifiers across varying bag size.

Bag Size	Accuracy		
	J48	1NN	Perceptron
0.1	57.58%	52.15%	57.03%
0.2	59.20%	51.64%	58.32%
0.3	59.55%	51.49%	58.93%
0.4	59.63%	51.03%	59.28%
0.5	59.79%	50.45%	59.24%
0.6	59.78%	49.68%	59.45%
0.7	59.75%	49.42%	59.60%
0.8	59.60%	49.26%	59.70%
0.9	59.60%	49.17%	59.58%
1.0	59.67%	49.06%	59.87%

Table 5: Performance of J48, 1NN, and Perceptron when increasing the bag size used in bagging.

Question 1.3

Apply ensembles with random subspacing using the three classifiers from Task (a). Investigate the performance of these classifiers as the ensemble size increases (e.g., in steps of 2 from 2 to 20 members). Using the best performing ensemble size, investigate how changing the number of features used when applying random subspacing affects classification performance (i.e. the "subspace size").

0.5 Ensemble Size

In this section we investigate how the number of ensembles affects the classifier accuracy, for our three classifiers, when using subspacing. We see from fig. 5 and table 6 that, for our three classifiers, an increase in the number of ensembles yields an increase in the accuracy of the model. Due to the Perceptron's long training time, we use 12 ensembles in the next section.

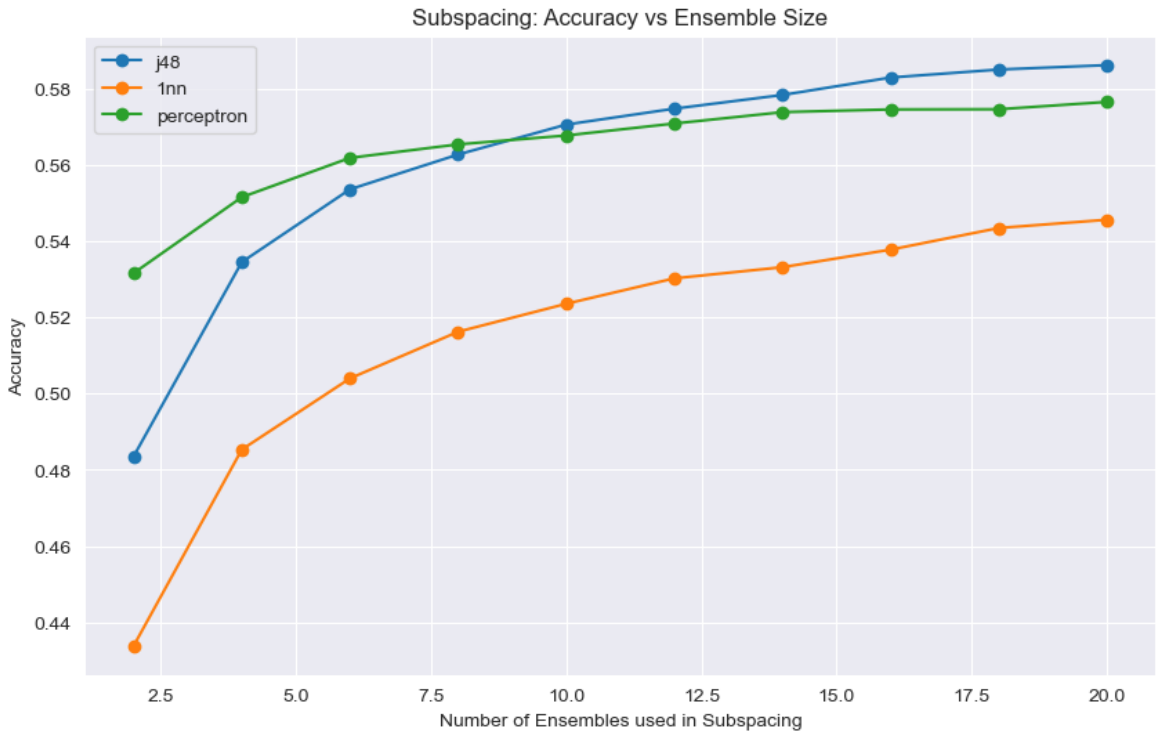

Figure 5: Performance of classifiers whilst varying number of ensembles present in subspacing.

Ensemble Size	Accuracy		
	J48	1NN	Perceptron
2	48.36%	43.38%	53.16%
4	53.45%	48.53%	55.15%
6	55.36%	50.40%	56.19%
8	56.27%	51.62%	56.54%
10	57.06%	52.36%	56.77%
12	57.48%	53.02%	57.09%
14	57.84%	53.32%	57.39%
16	58.29%	53.78%	57.46%
18	58.50%	54.35%	57.46%
20	58.62%	54.56%	57.65%

Table 6: Performance of classifiers whilst varying number of ensembles present in subspacing.

0.6 Subspace Size

In this section we show the effect that varying subspace size, the number of attributes per ensemble, has on classifier accuracy. For this experiment we hold the number of ensembles stationary and vary the subspace size parameter. We use ensemble size of 20, 20, and 12 for J48, 1NN, and Perceptron respectively. We use 12 ensembles for the Perceptron as it achieves near-optimal accuracy, while the increased training time at 20 ensembles makes its use impractical. We see that 1NN performs better relative to J48 and Perceptron under this ensemble schema. Finally we note that we see some fall off as we increase the number of features available in each ensemble, this fall off is especially noted for our 1NN classifier.

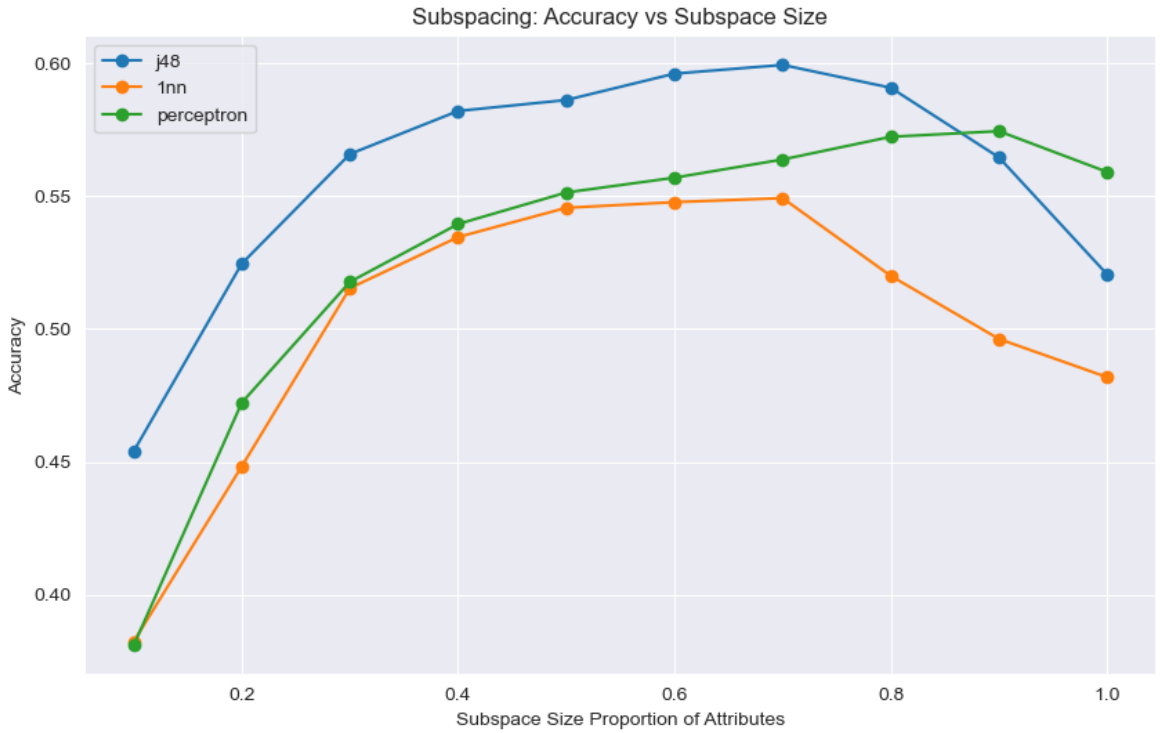

Figure 6: Performance of J48, 1NN, and Perceptron classifiers across varying subspace size when using subspacing.

Subspace Size	Accuracy		
	J48	1NN	Perceptron
0.1	45.40%	38.23%	38.09%
0.2	52.46%	44.83%	47.23%
0.3	56.59%	51.54%	51.78%
0.4	58.21%	53.46%	53.95%
0.5	58.62%	54.56%	55.14%
0.6	59.61%	54.78%	55.70%
0.7	59.94%	54.93%	56.38%
0.8	59.08%	52.00%	57.23%
0.9	56.46%	49.63%	57.45%
1.0	52.06%	48.19%	55.91%

Table 7: Performance of J48, 1NN, and Perceptron when increasing the subspace or proportion of attributes used in subspacing.

Question 1.4

Based on the lectures, which set of classifiers is expected to benefit from bagging techniques more and which set of classifiers is expected to benefit from random sub-spacing techniques more? For your dataset, determine the best ensemble strategy for each of these classifiers. Discuss if this is in line with what you expected.

0.7 Bagging

Based on the lectures and further research our expectation is that bagging will be most successful with unstable models, in our case Decision Trees and Multi-Layer Perceptron (MLP), and will be least successful with stable models, in our case 1NN. This is because in general bagging ensembles tend to remove variance from model outputs due to signal noise. Decision Trees and MLP are highly influenced by noise and thus are expected to perform well with bagging. 1NN which is already robust in the presence of noise is less benefited by bagging and in some cases bagging can diminish the accuracy. To highlight the improvement of bagging we examine the accuracy of our base classifiers from section 0.1.5 and compare with the results of bagging in Question 1.2. We see in table 8 that overall bagging has improved J48 the most and MLP the least.

Classifier	Base	Bagging
J48	52.21	59.79
1NN	48.24	52.15
MLP	58.33	59.91

Table 8: Comparison of base classifiers to ensemble methods using bagging

Decision trees are high-variance models, meaning small changes in the training data can lead to significantly different tree structures and predictions. Decision trees construct specific rules based on the training data, however noisy data can cause the tree to build complex rules leading to spurious predictions and complexity [4]. Pruning the tree during or after the training is one possibility to reduce over-fitting to the noise [5]. We investigate this heavily in appendix A section 1.1.3 by varying the parameters of the decision tree to reduce over-fitting in the training data. However another approach to reducing the affect of noise is bagging. Bagging helps mitigate over-training by averaging out the effects of noisy data points across many ensembles.

Multi-Layer Perceptrons also have the ability to create complex rules which can lead to over-fitting of the data. Complex features can be created in the MLP hidden layers to account for small perturbations in the training data. However, these features might not represent the underlying trends, leading to over-fitting. This highlights the importance of carefully designing the network setup when working with MLPs ensuring there is enough data to train all the weightings accurately. We carefully designed our network in section 0.1.4 so this doesn't appear to be a problem for us. Bagging can still improve performance by reducing the impact of variance in our model [6].

In the case of kNN classifiers, especially those with distance weighting, the output prediction is highly dependant on the closest samples in the dataset. This makes these models highly resilient to noise, as distant training samples will not affect the output by much. This resilience to noise is referred to as high stability. The reason bagging is less effective for these high stability models is that the likelihood of excluding high importance samples from the bag is the same as excluding non-impactful noise. If small distance samples are excluded it will hinder the kNN classifiers ability to make good predictions. In the case of the 1NN classifier, the probability of excluding the closest point from any one bag can be calculated based on the size of the data and the size of the bag [7]. We show this calculation in Appendix A section 1.1.4 and list the results in table 10. From this table, we see that the use of bagging decreases the probability of including the closest samples in the classifier, which directly hinders the kNN's ability to predict the output class accurately.

0.8 Subspacing

Based on the lectures and further research our expectation is that feature subspacing will be most successful with our 1NN classifier and least successful with the Decision Trees and Multi-Layer Perceptron (MLP) classifier. This is because limiting the number of features available to the decision tree and MLP limit their ability to model complex relationships between features. However kNN's classifiers don't suffer from this as they don't build relationships between features. Furthermore, we have previously established, that due to the way kNN's compute their distance metric they can be hindered by high dimensionality, or so called curse of dimensionality. When a lot of features are present the distance separation between samples tend to decrease (Hypercube example) [8]. By reducing the number of features using subspacing we can increase the kNN classifiers ability to distinguish between samples. From table 9 we see that 1NN network has performed better with subspacing than with bagging. Unexpectedly J48 also performed better with subspacing. Finally, MLP performed worse even than the base classifier trained in section 0.1.5.

Classifier	Base	Subspacing
J48	52.21	59.94
1NN	48.24	54.93
MLP	58.33	57.65

Table 9: Comparison of base classifiers to ensemble methods using feature subspacing

0.9 Optimal Strategies and Conclusions

In this section we present our optimal models and discuss whether they are in line with expectations. In some cases, as with Decision Trees, we offer extra work that we display in Appendix-A section 1.1.3. This extra work relates to how well the models are generalised and if over-fitting is occuring. In many cases these extra plots can offer insight into the results that we've seen and help us interpret any unexpected results. As many models have been trained I found it a great help to use WEKA's CLI and a bash script to run experiments automatically. I provide the bash script and python Regex in Appendix B section 1.2.

0.9.1 Decision Tree

Best Classifier Decision Tree			
Method: Subspacing	Ensembles: 20	Subspace Size: 70%	Accuracy: 59.94%

The best ensemble strategy for our J48 Decision Tree classifier was found to be using subspacing with an ensemble size of 20 and 70% subspacing size. This was not as expected. We had expected our best ensemble to be with bagging however we have yielded improvements with both bagging and subspacing. We have made a large improvement from the base classifier to the subspacing ensemble with an accuracy improvement of 52.21% $\rightarrow$ 59.94%. To try and explain this we observe the Appendix plots in section 1.1.3 where fig. 17, fig. 18, fig. 19, and fig. 20 show that our trained model does not generalise well to the test data. The large improvement in accuracy from 52.21% $\rightarrow$ 59.94% might be explained by this over-fitting of training data. This indicates that noise in the training data is being modelled with special rules in the decision tree which do not generalise well to the test data. To investigate this we investigated some of the parameters of the decision tree in a further exercise to attempt to reduce the over-fitting. See fig. 22 , fig. 21, fig. 24, fig. 23, and fig. 25 where we vary several training parameters to reduce over-fitting. However, this is left as an exercise for the future as we are already getting a bit effusive.

0.9.2 MLP

Best Classifier MLP			
Method: Bagging	Ensembles: 20	Bag Size: 100%	Accuracy: 59.91%

The best ensemble strategy for our multi-layer Perceptron classifier was found to be using bagging with an ensemble size of 20 and 100% bag size. This was as expected as our prediction was, that bagging methods work well with unstable classifiers like MLP. We have only made a small accuracy improvement on the base model 58.33% $\rightarrow$ 59.91%. The reason for this could be that the model was already fit quite well to the data. Normally bagging would remove the effects caused by noise in an attempt to generalise and avoid over-fitting. However if we observe fig. 13 and fig. 14 we can see that the model is already generalising quite well across ensemble and bag sizes. We assume this is due to our efforts in choosing good network parameters from the outset. One final note is that our classifier has actually gotten worse when using subspacing, this implies that our model was fitted well, and restricting features hinders its ability to make relationships across features.

0.9.3 kNN

Best Classifier kNN			
Method: Subspacing	Ensembles: 20	Subspace Size: 70%	Accuracy: 54.93%

The best ensemble strategy for our 1NN classifier was found to be using subspacing with an ensemble size of 20 and 70% subspacing size. This was as we expected as generally kNN classifiers work well with feature subspacing. It is interesting to note that in fig. 6 that there is a clear point at which the number of features starts to decrease the accuracy of our model. The expectation is that this decrease in accuracy is due to the curse of dimensionality as the number of features increase. Up until this point the model has been improving, but then there is a clear fall off.

Finally one other interesting point to note is that in fig. 4 we notice that as the bag size increases the accuracy of 1NN decreases. However this was not our expectation. According to table 10 we had expected that as the bag size increased the performance to increase also[1]. The expectation is that there is some complex relationship between feature size and this bag size, and that under a smaller number of features we might obtain a result more in keeping with table 10.

Samples	Bag Size (%)	Probability of Exclusion
21812	10%	0.9048
	20%	0.8187
	30%	0.7408
	40%	0.6703
	50%	0.6065
	60%	0.5488
	70%	0.4966
	80%	0.4493
	90%	0.4066
	100%	0.3679

Table 10: Effect of Bag Size on Probability of Sample Selection for a Training Size of 21812.

[1] I'd like to know what you think about this, maybe you can answer in feedback

Question 1.5

Some of the features may be correlated with others or have dependencies on other features. Build a linear regression model to predict the energy feature from this set of features: tempo, loudness, liveness. Compute the regression model using Linear Regression and SGD. Show the regression model and comment on the quality of the model and any differences you observe between SGD and Linear Regression.

0.10 OLS vs SGD

A linear regression model can be represented as eq. (4) where y is the predicted variable and x_1 to x_n are the feature variables. ϕ_1 to ϕ_n are the weights of our features in predicting the output y, and ϕ_0 is the intercept of the predicted feature. For our case we have three feature variables: tempo, loudness, and liveness. So our linear equation reduces to eq. (5) where ϕ_1, ϕ_2, and ϕ_3 represent the weights of tempo, loudness, and liveness in predicting the output respectively.

$$y = \phi_0 + \phi_1 x_1 + ... + \phi_n x_n \tag{4}$$

$$y = \phi_0 + \phi_1 x_1 + \phi_2 x_2 + \phi_3 x_3 \tag{5}$$

The difference between the linear regression model and SGD in WEKA is the way they update the weightings for these feature variables. The linear regression implementation explicitly solves for the best weightings ϕ_0 to ϕ_3 using ordinary least squares (OLS) by minimising the squared error function $(y - \hat{y})^2$ called the loss function (L). This is an analytical solution with an exact value. SGD on the other hand computes weightings by incrementally changing the weightings proportionally to the negative of the gradient of the loss function $\propto -\frac{\partial L}{\partial \phi}$. While OLS explicitly solves for the best weighting parameters it requires the inversion of the matrix of input features. This operation is $O(n^3)$ where n is the size of the matrix. Hence OLS scales poorly as the number of features grows. SGD on the other hand depends on the number of steps it takes to converge, which is given roughly as $O(kn^2)$ so when n is very large is recommended to use gradient descent instead of the closed form solution of OLS linear regression [9].

0.11 Model Results and Conclusions

The two models were trained on the Spotify data, the results of the fitting parameters are shown in eq. (6). Table 11 shows the standard linear regression error metrics. Both models agree that the correlation coefficient is 0.687, indicating that 68.7% of the variation in the energy variable can be explained by the three feature variables. Across the metrics Linear regression has performed better than SGD, with slightly lower error metrics in every category. This is as expected because linear regression has used OLS to directly compute the optimal parameters. We notice however that SGD has obtained a very similar result, with no major differences. This indicates that SGD has found the same minimum but just needs a smaller step size to approach the OLS parameters. In regards to fitting time SGD has taken 0.33 seconds to train the model and linear regression has taken 0.18 seconds. However as the number of features becomes larger the SGD will begin to outperform the closed form solution in regards to training time. This make SGD appropriate for training when many parameters are included in the model, and OLS appropriate when the number of parameters is smaller. See fig. 7 and fig. 8 for correlation plots.

$$\begin{aligned} \text{energy}_{OLS} &= 1.8483 \times \text{loudness} + 0.1185 \times \text{liveness} + 0.1175 \times \text{tempo} - 0.9159 \\ \text{energy}_{SGD} &= 1.8506 \times \text{loudness} + 0.1239 \times \text{liveness} + 0.1157 \times \text{tempo} - 0.9174 \end{aligned} \tag{6}$$

Metric	Linear Regression	SGD
Correlation Coefficient	0.687	0.687
Mean Absolute Error (MAE)	0.099	0.101
Root Mean Squared Error (RMSE)	0.124	0.125
Relative Absolute Error (RAE)	0.717	0.725
Root Relative Squared Error (RRSE)	0.727	0.730

Table 11: Evaluation Metrics for Linear Regression and SGD Models

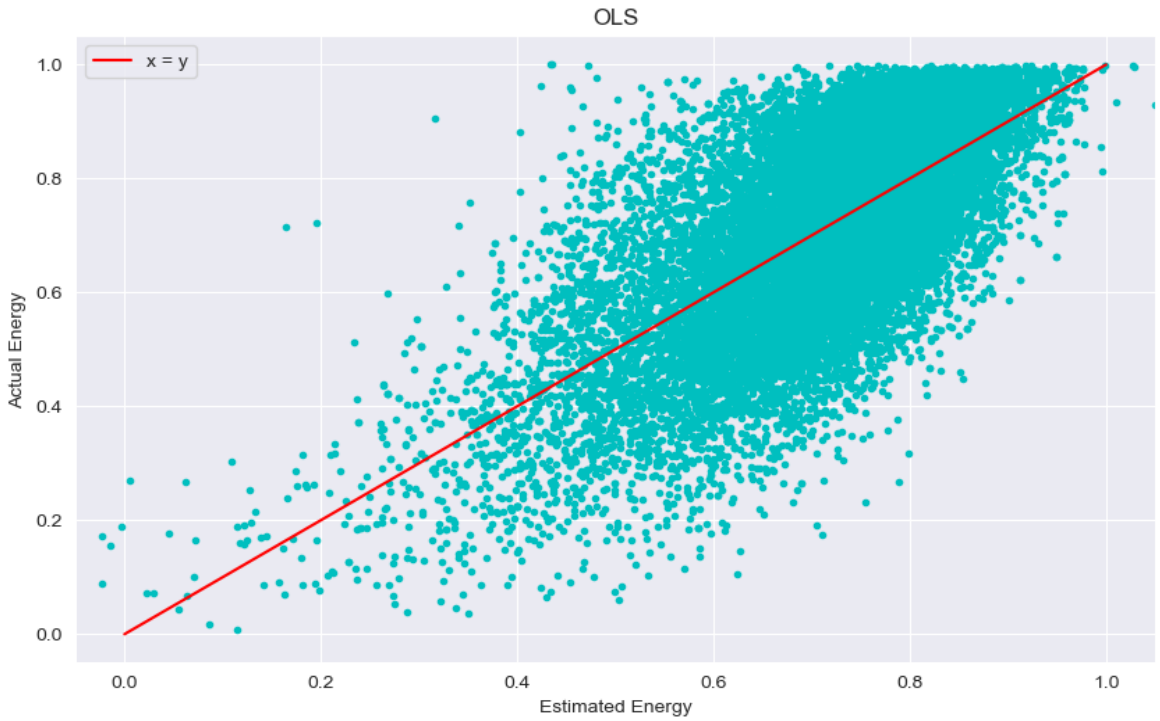

Figure 7: Plot of OLS estimation against the actual energy score. Correlation = 0.687.

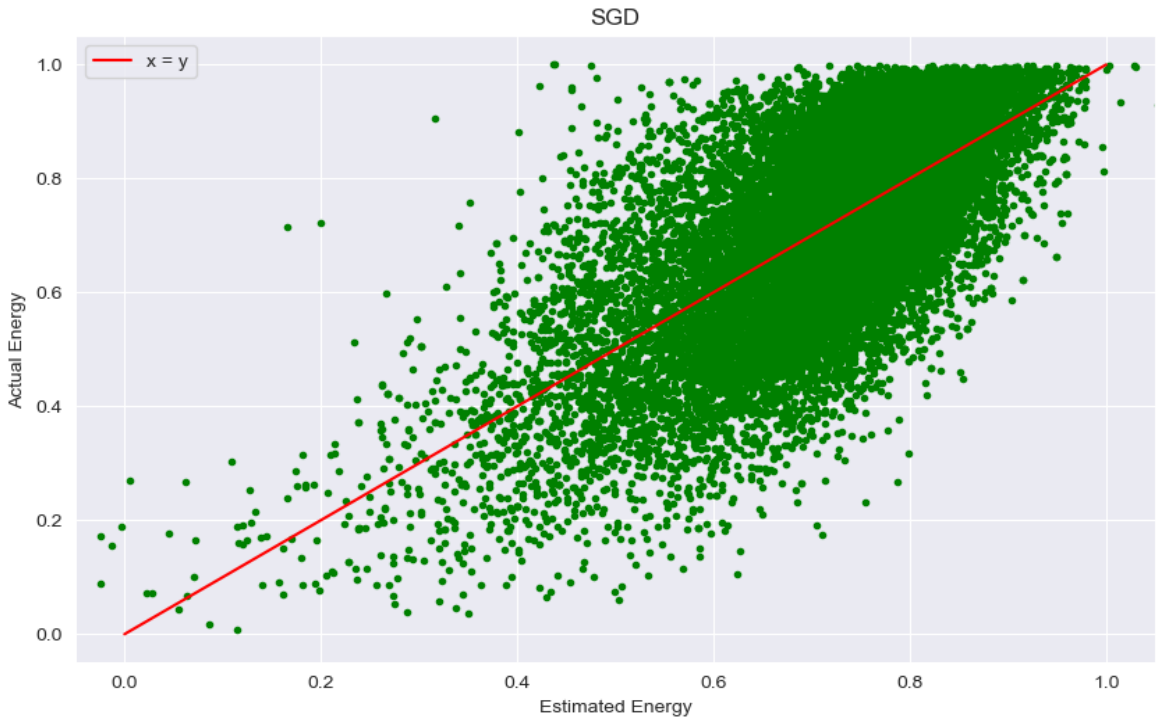

Figure 8: Plot of SGD estimation against the actual energy score. Correlation = 0.687.

References

[1] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd. Prentice Hall, 1998.

[2] S. Exchange, *How to choose the number of hidden layers and nodes in a feedforward neural network?* Accessed: 2024-11-24, 2012. [Online]. Available: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw.

[3] comp.ai.neural-nets FAQ, *Part 3 of 7: Generalization - what is overfitting and how can i avoid it?* FAQ on comp.ai.neural-nets, Accessed: 2024-11-24, 1998. [Online]. Available: http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-3.html.

[4] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, pp. 81–106, 1986.

[5] D. F. Nettleton, A. Orriols-Puig, and A. Fornells, "A study of the effect of different types of noise on the precision of supervised learning techniques," *Artificial intelligence review*, vol. 33, pp. 275–306, 2010.

[6] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation, and active learning," *Advances in neural information processing systems*, vol. 7, 1994.

[7] B. Efron, *Bootstrap methods: another look at the jackknife*. Springer, 1992, pp. 569–593.

[8] N. Kouiroukidis and G. Evangelidis, "The effects of dimensionality curse in high dimensional knn search," in *2011 15th Panhellenic Conference on Informatics*, IEEE, 2011, pp. 41–45.

[9] A. Ng, A. Bagul, and G. Ladwig, *Supervised machine learning: Regression and classification*, Coursera, Instructors: Andrew Ng, 2024. [Online]. Available: https://www.coursera.org/learn/machine-learning.

[10] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, pp. 123–140, 1996.

[11] R. R. Bouckaert, E. Frank, M. Hall, *et al.*, *Weka manual, version 3.9.3*, Accessed: 2024-11-27, Sep. 2018. [Online]. Available: https://ml.cms.waikato.ac.nz/weka/documentation.html.

1 Appendix

1.1 Appendix - A

1.1.1 Investigation of non-normalised data with classifiers

Classifier Type	Accuracy	EDM	Latin	Pop	Rap	Rock
J48	0.519	0.607	0.404	0.326	0.605	0.648
1-NN	0.482	0.570	0.421	0.326	0.501	0.591
Perceptron	0.547	0.598	0.328	0.393	0.679	0.734
Best Performer	**Perceptron**	**J48**	**1-NN**	**Perceptron**	**Perceptron**	**Perceptron**

Table 12: Classifier Performance: J48 Decision Tree, 1-Nearest Neighbour (1-NN), Perceptron, and Best Performer; without normalisation

1.1.2 Investigation of different KNN setups

Classifier Type	Accuracy	EDM	Latin	Pop	Rap	Rock
3 Inverse Distance Weighted NN	50.110	0.596	0.418	0.343	0.527	0.617
5 Inverse Distance Weighted NN	52.201	0.611	0.432	0.367	0.560	0.635
5 Nearest Neighbours	49.670	0.659	0.453	0.322	0.501	0.530

Table 13: Performance of Different kNN setups. I thought these would be better as the different music types should exhibit some similarity, but variance between songs is high. However, I guess the variance between the song you're classifying is high also.

1.1.3 Generalising and Optimising our Classifiers

In this section we include the plots generated while trying to optimise our parameters and to generalise the classifiers, avoiding over-fitting in the training samples compared with the test samples. We use the accuracy and RMSE of the train set vs the test set to determine if we have over-fit the train. (Next plage as latex is freaking out)

1NN

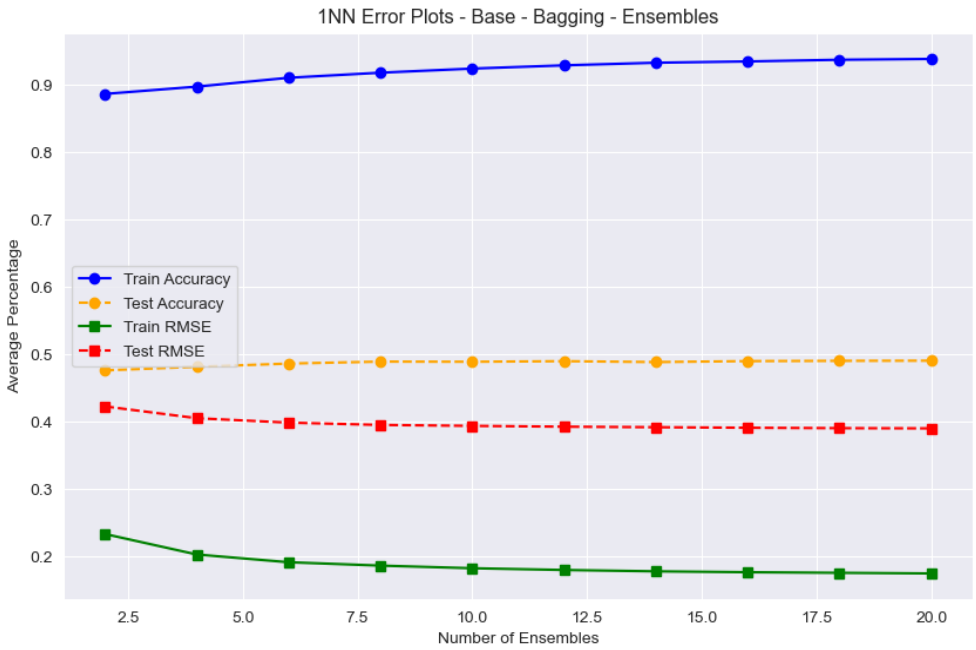

Figure 9: 1NN Error Plots - Base - Bagging - Ensembles

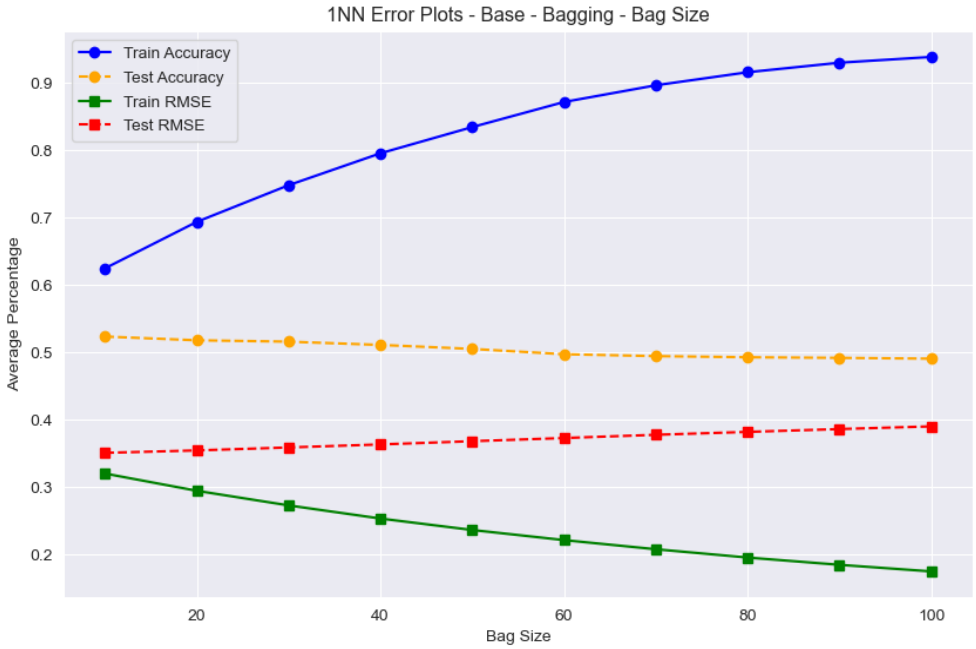

Figure 10: 1NN Error Plots - Base - Bagging - Bag Size

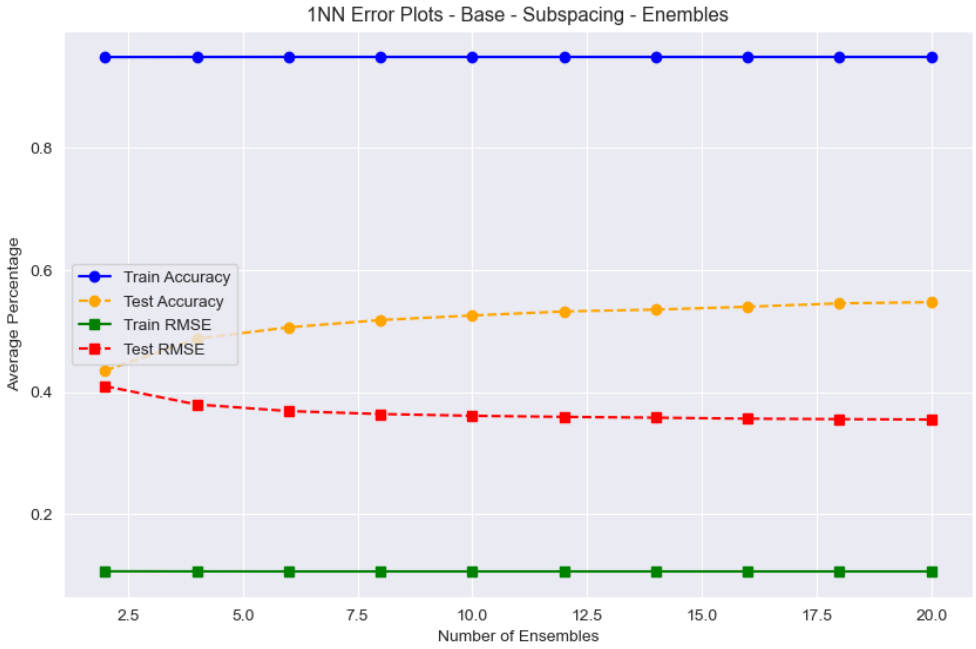

Figure 11: 1NN Error Plots - Base - Subspacing - Ensembles

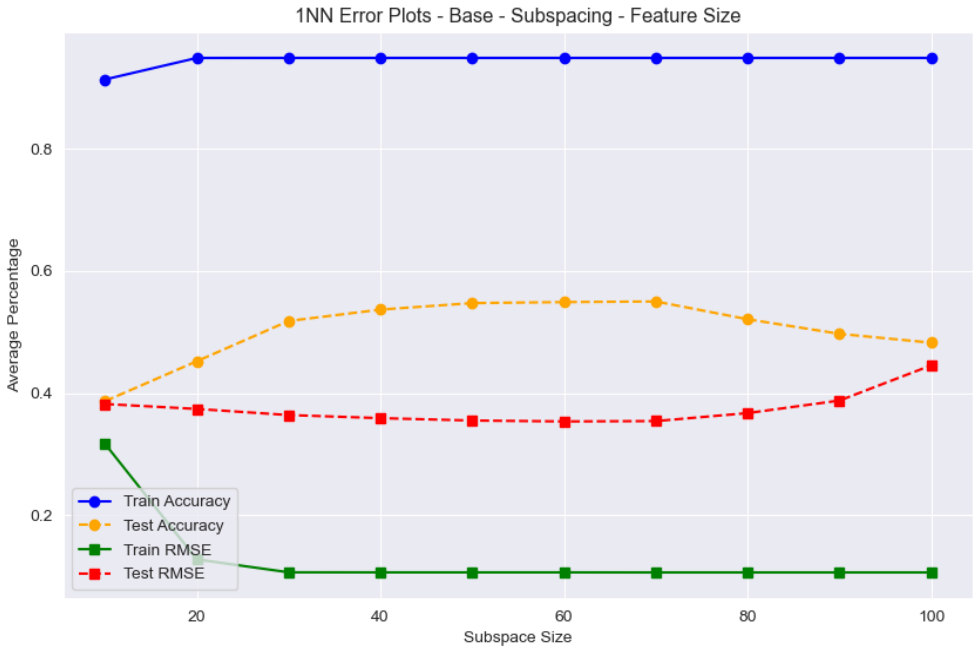

Figure 12: 1NN Error Plots - Base - Subspacing - Feature Size

Multi-layer Perceptron

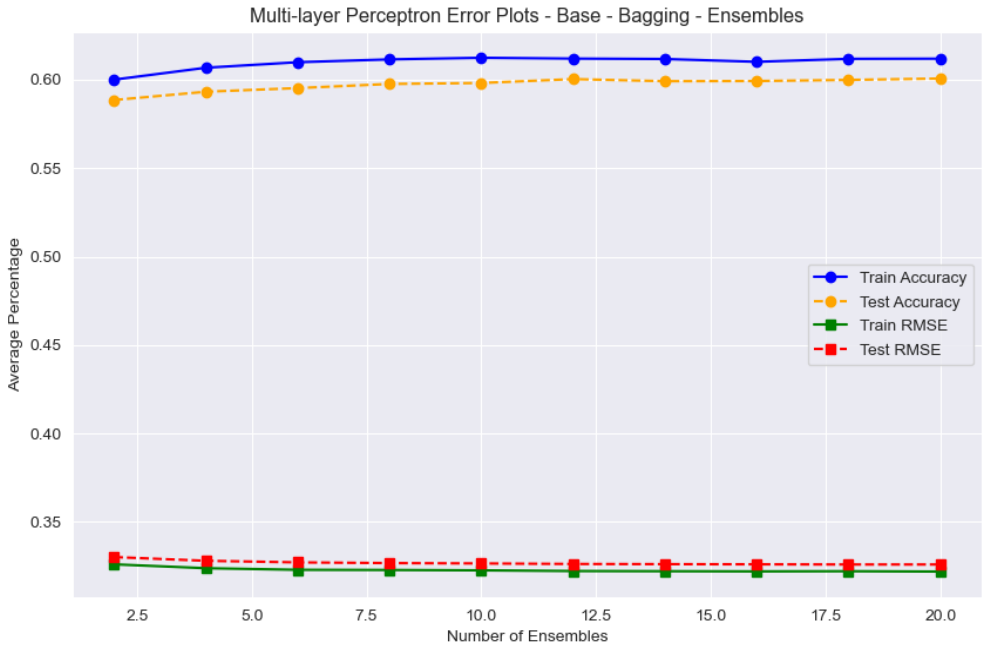

Figure 13: Multi-layer Perceptron Error Plots - Base - Bagging - Ensembles

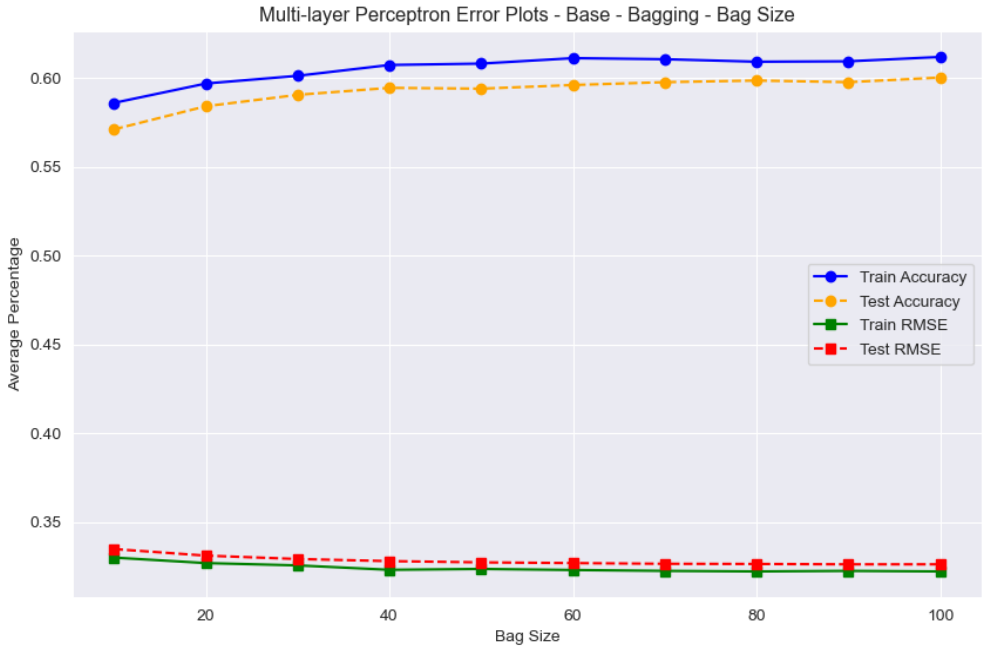

Figure 14: Multi-layer Perceptron Error Plots - Base - Bagging - Bag Size

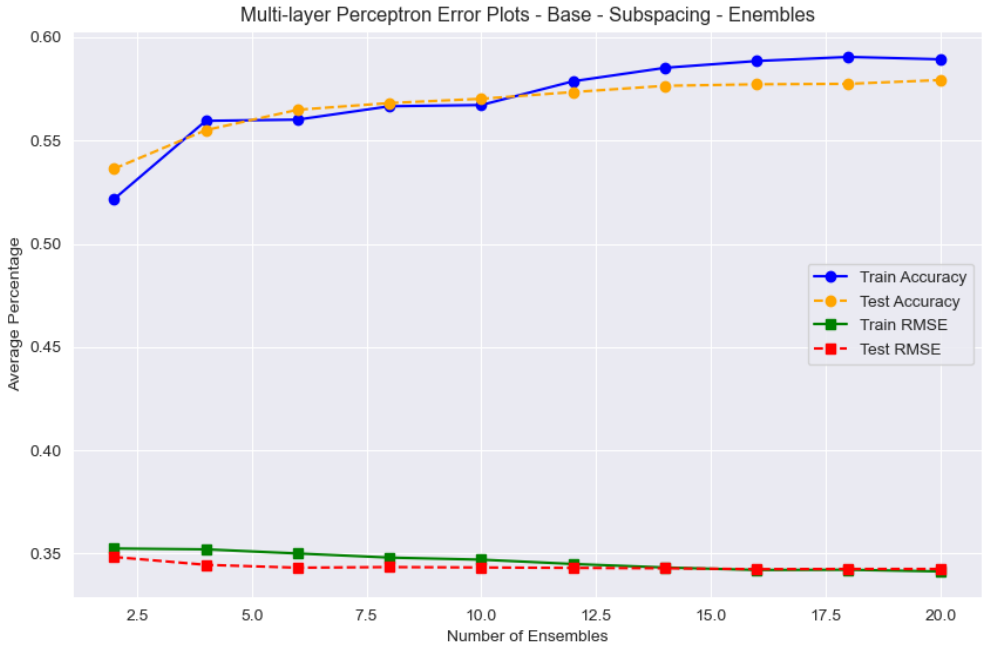

Figure 15: Multi-layer Perceptron Error Plots - Base - Subspacing - Ensembles

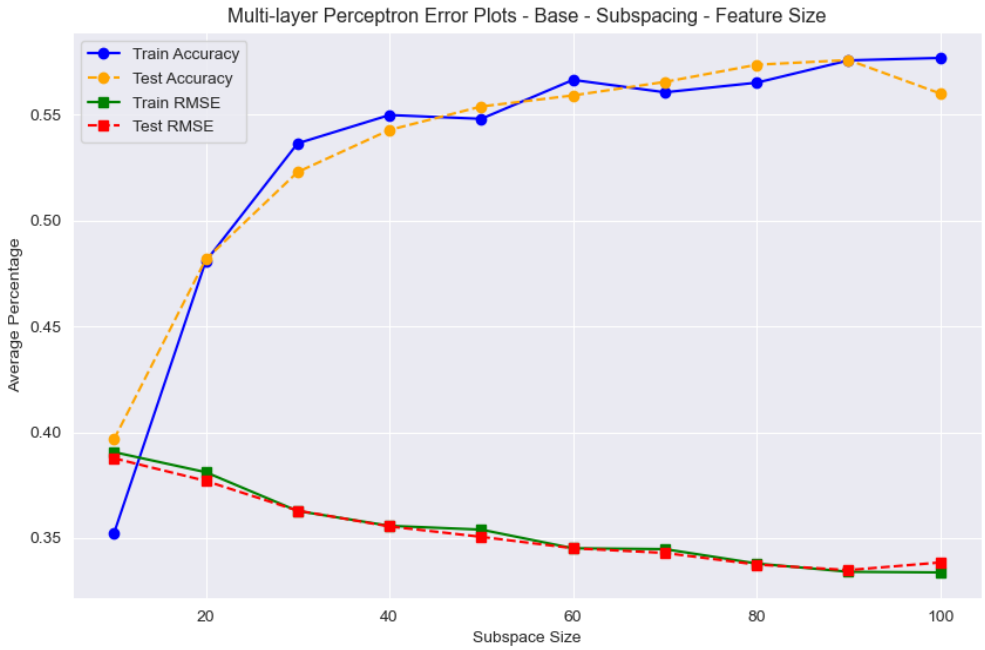

Figure 16: Multi-layer Perceptron Error Plots - Base - Subspacing - Feature Size

J48 Decision Tree

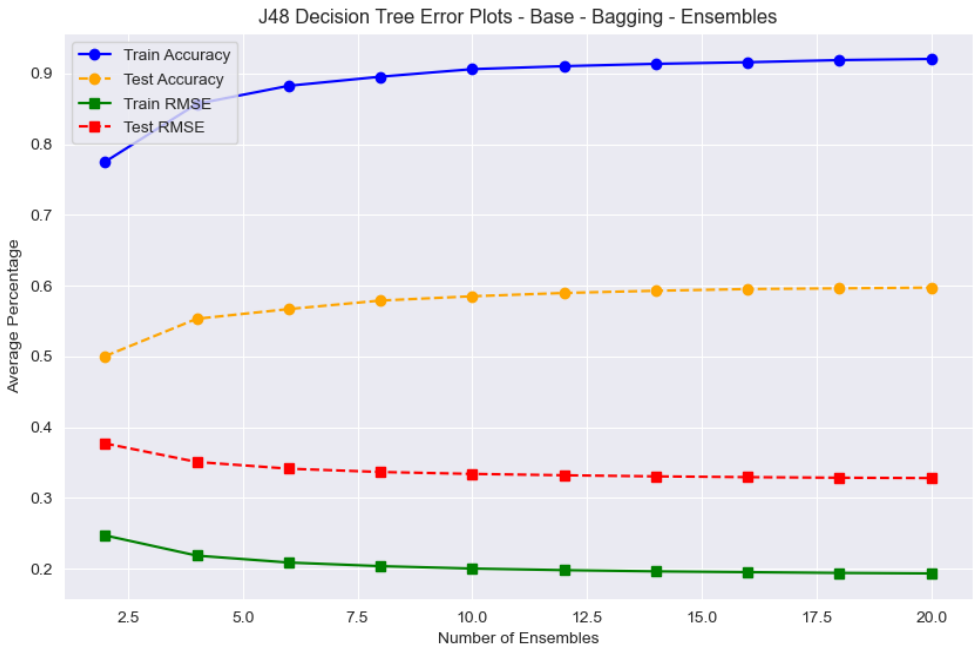

Figure 17: J48 Decision Tree Error Plots - Base - Bagging - Ensembles

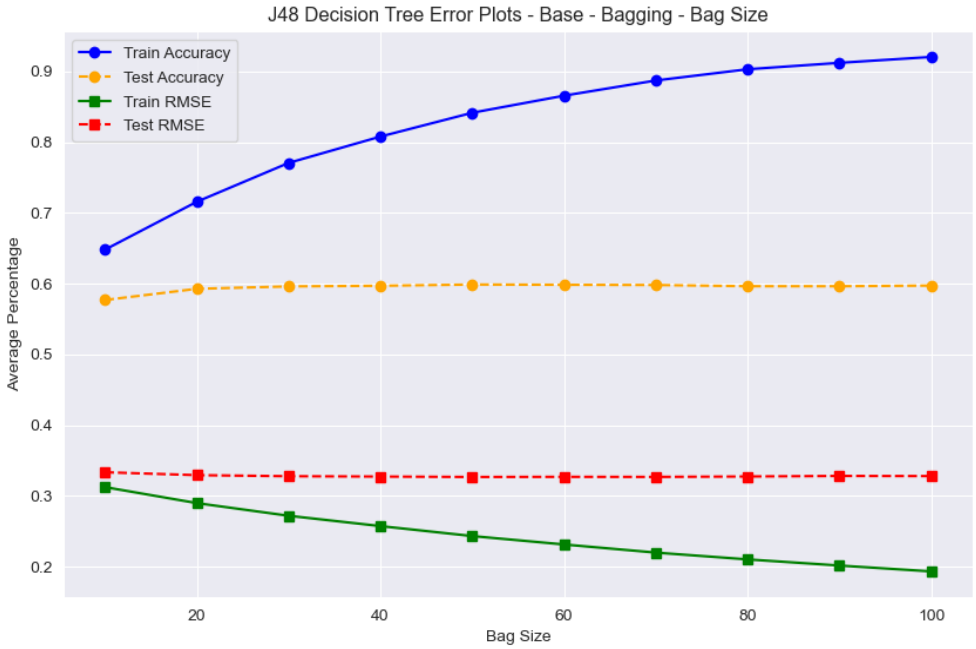

Figure 18: J48 Decision Tree Error Plots - Base - Bagging - Bag Size

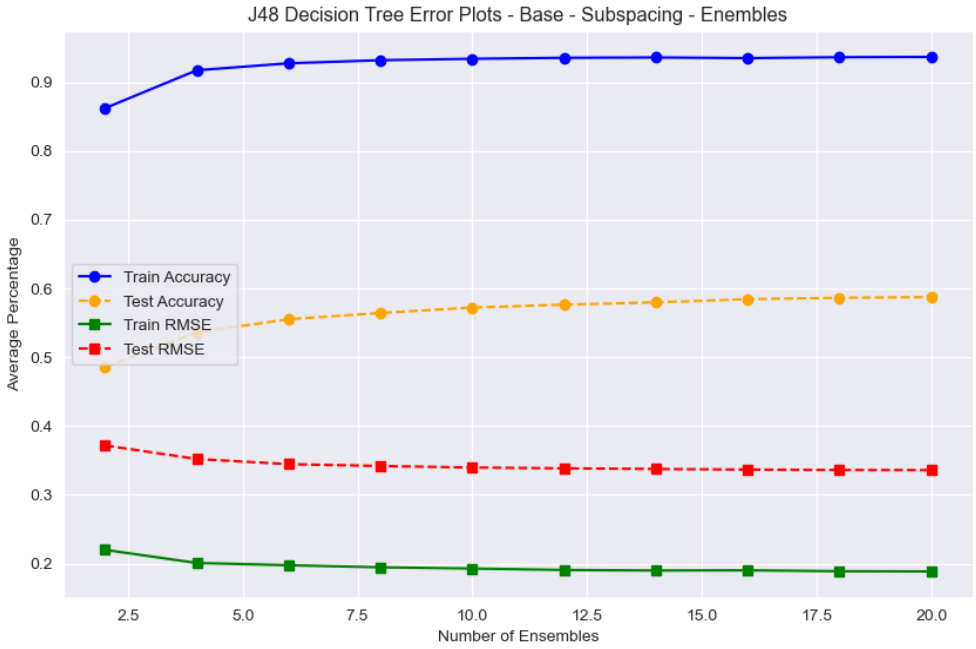

Figure 19: J48 Decision Tree Error Plots - Base - Subspacing - Ensembles

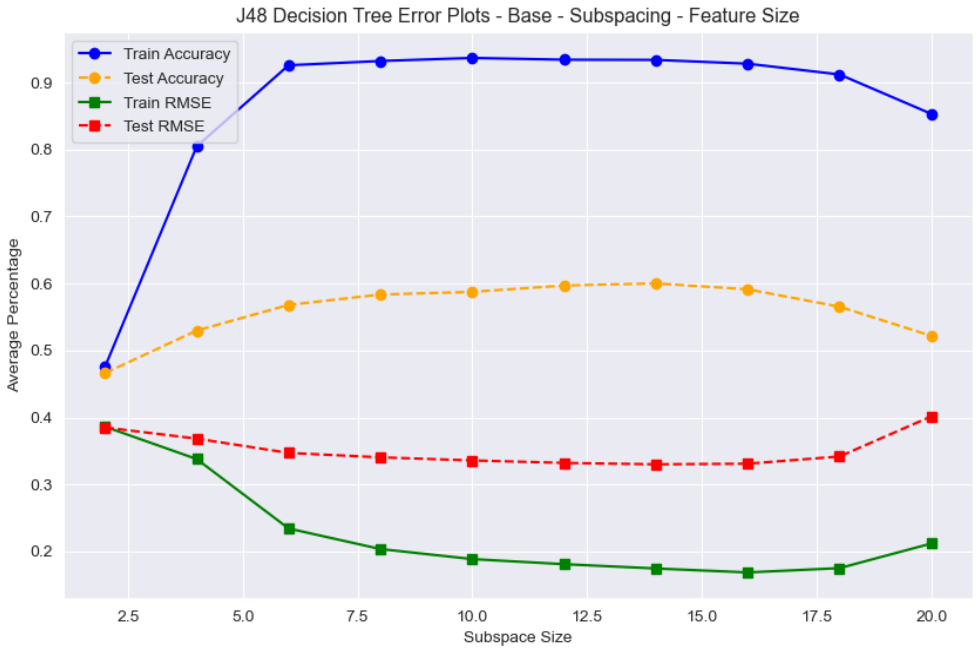

Figure 20: J48 Decision Tree Error Plots - Base - Subspacing - Feature Size

J48 Extra Generalisation

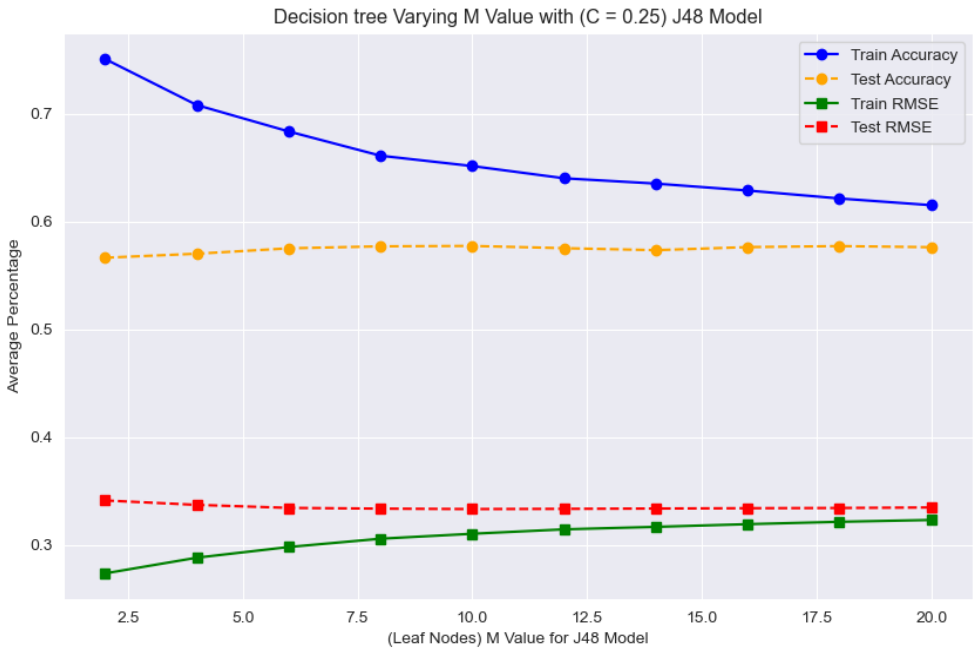

Figure 21: Decision Tree Varying M Value with (C = 0.25) J48 Model

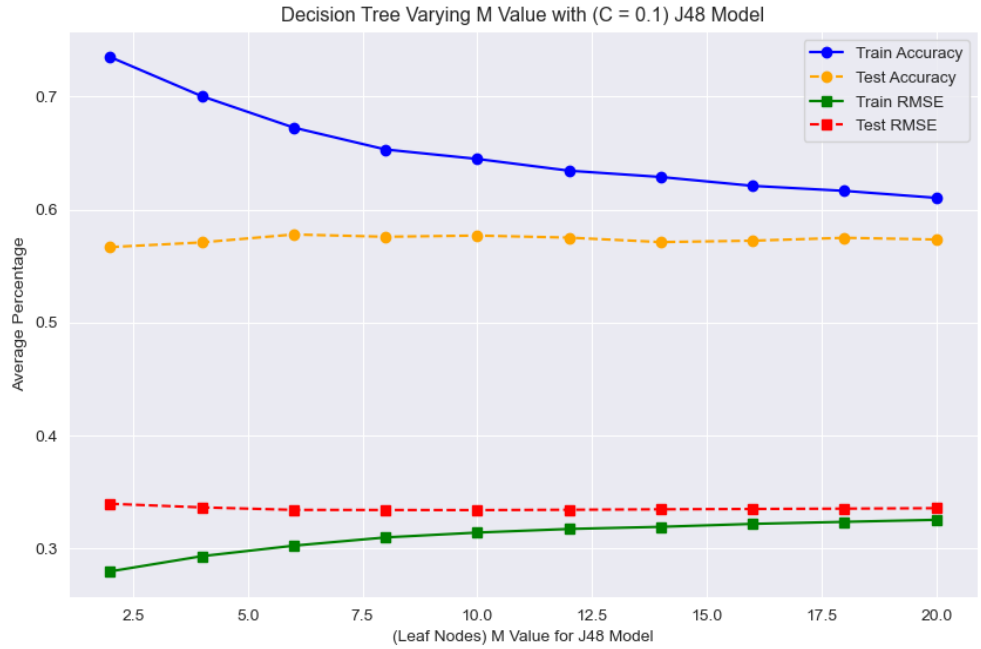

Figure 22: Decision Tree Varying M Value with (C = 0.1) J48 Model

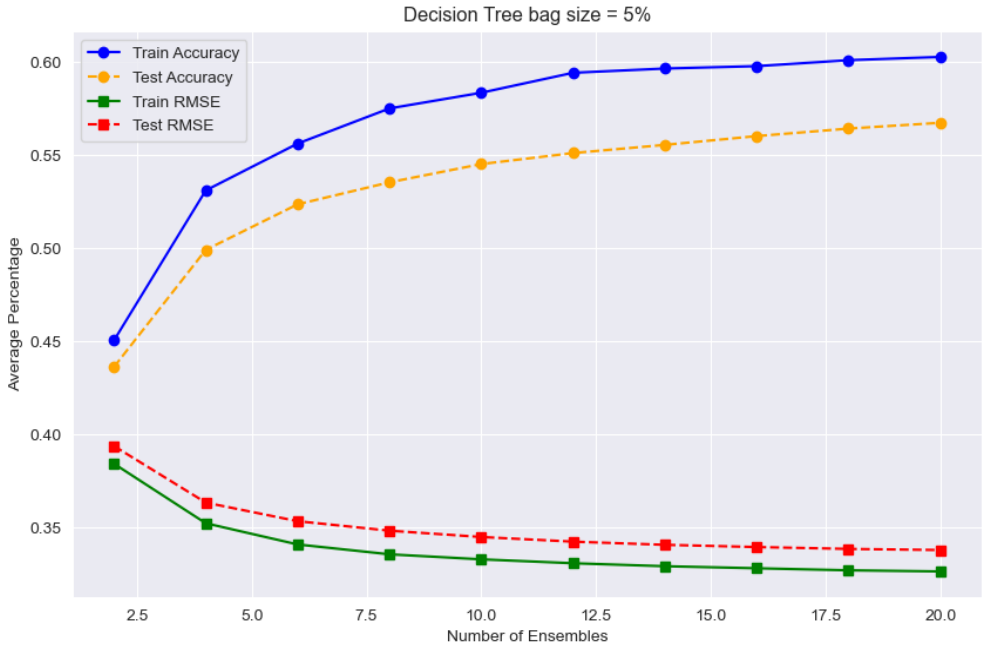

Figure 23: Decision Tree Error Plots - Bag Size = 5

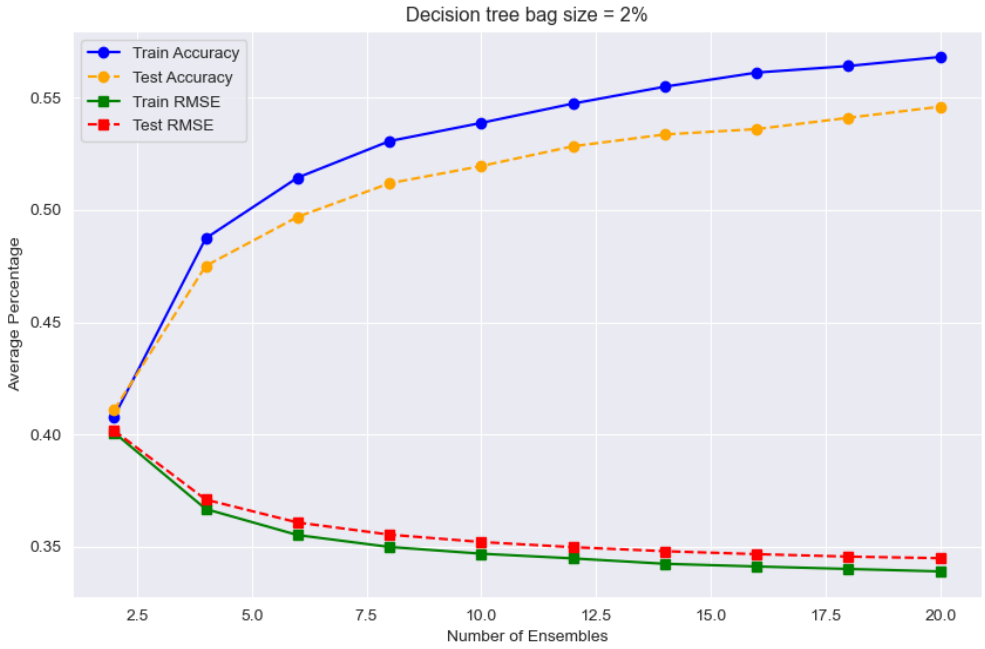

Figure 24: Decision Tree Error Plots - Bag Size = 2

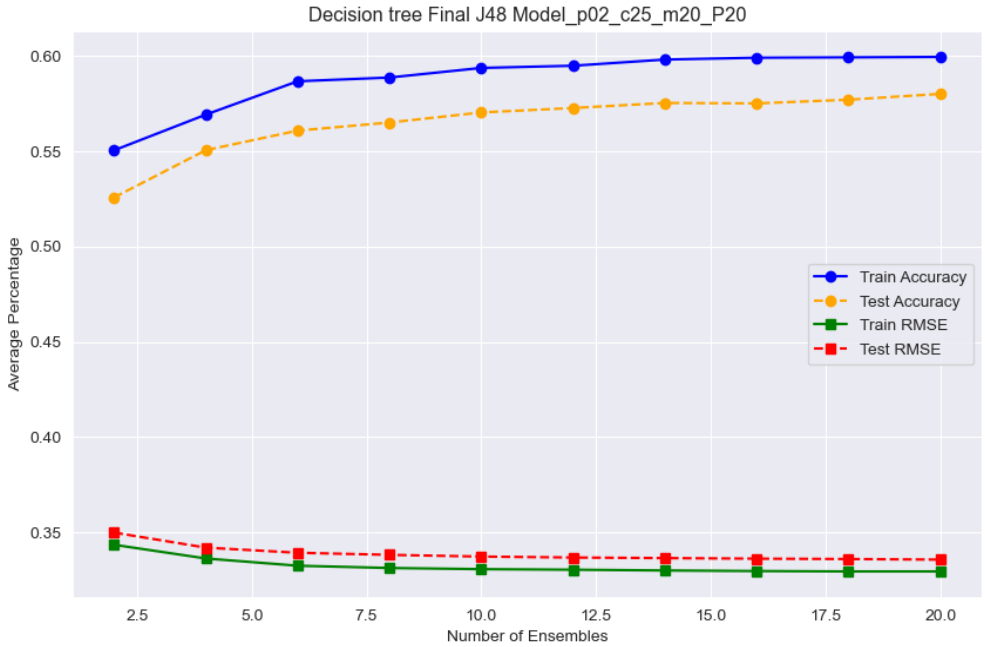

Figure 25: Decision Tree Final J48 Model (p=0.2, c=0.25, m=20, P=20)

1.1.4 Probability of including a sample in the bag for 1NN classifier

If you take a group of training samples of size N the probability of choosing the nearest neighbour (1NN) is:

$$\frac{1}{N}$$

Therefore the probability of excluding the sample is:

$$(1 - \frac{1}{N})$$

If you choose 2 items from the sample at random, with replacement, the probability becomes:

$$(1 - \frac{1}{N})^2$$

The chance of picking our closest neighbour does not increase as we replace the sample in the group N. So if we choose B samples (a bag size of B) the probability becomes

$$(1 - \frac{1}{N})^B$$

In most bagging, including our WEKA ensemble, the default bag size is $B = N$

$$(1 - \frac{1}{N})^N$$

As $N \to \infty$ the probability of excluding our most important element approaches the constant value $(1 - \frac{1}{e}) = 0.632$ a value stated in [10].

$$\lim_{N \to \infty} \left(1 - \frac{1}{N}\right)^N = \frac{1}{e}$$

However for our case we explicitly know our data size and compute the probability of our sample being included in our training data in table 10.

1.2 Appendix - B

1.2.1 BASH for WEKA CLI

I attempted to use knowledge flow and experimenter GUI's for WEKA but for larger experiments the CLI is necessary as the GUI tends to keep the models in memory and I was running out when training a lot of models. The memory can be increased but can also just use the CLI. The base classifiers are easy to train in the CLI but the syntax of training multiple classifiers is strange, as is the case in the bagging example. So I've included a simple example below, as it was extremely necessary. Classifiers can be swapped in out and result are parsed from the text files using python Syntax of multi models was found in [11].

Spotify Recomendation API

```
#!/bin/bash

CLAS="/Applications/weka-3.8.6.app/Contents/app/weka.jar" # WEKA path as a variable
OUTPUT_J48="/Users/brianmcmahon/Documents/COMP 47460 - ML/Assignment_2/WEKAoutputs/output_j48.txt"
OUTPUT_1NN="/Users/brianmcmahon/Documents/COMP 47460 - ML/Assignment_2/WEKAoutputs/output_1nn.txt"
OUTPUT_PERCEPTRON="/Users/brianmcmahon/Documents/COMP 47460 - ML/Assignment_2/WEKAoutputs/output_perceptron.txt"
# Output paths as a variables
MODEL="/Users/brianmcmahon/Documents/COMP 47460 - ML/Assignment_2/trees.model"  # Model path as a variable
DATA="/Users/brianmcmahon/Documents/COMP 47460 - ML/Assignment_2/spotify_15463152_normalized.arff" # Data path as a variable

echo "Working on J48"

for i in 2 4 6 8 10 12 14 16 18 20
do
  echo "Working on J48 i i"
  java --add-opens java.base/java.lang=ALL-UNNAMED -cp CLAS \
    weka.classifiers.meta.Bagging \
    -P 100 \
    -S 1 \
    -num-slots 1 \
    -I i \
    -t "DATA" \
    -W weka.classifiers.trees.J48 -- -C 0.25 -M 2 >> "OUTPUTJ48"
    echo "BREAKBREAKBREAK" >> "OUTPUTJ48"
done

echo "Working on 1NN"

for i in 2 4 6 8 10 12 14 16 18 20
do
    echo "Working on 1NN i i"
  java --add-opens java.base/java.lang=ALL-UNNAMED -cp CLAS \
    weka.classifiers.meta.Bagging \
    -P 100 \
    -S 1 \
```

```
    -num-slots 1 \
    -I i \
    -t "DATA" \
    -W weka.classifiers.lazy.IBk -- -K 1 -W 0 -I -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\"" >> "OUTPUT1NN"
    echo "BREAKBREAKBREAK" >> "OUTPUT1NN"
done

echo "Working on Perceptron"

for i in 2 4 6 8 10 12 14 16 18 20
do
    echo "Working on perceptron i i"
  java --add-opens java.base/java.lang=ALL-UNNAMED -cp CLAS \
    weka.classifiers.meta.Bagging \
    -P 100 \
    -S 1 \
    -num-slots 1 \
    -I i \
    -t "DATA" \
    -W weka.classifiers.functions.MultilayerPerceptron \
    -- -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H t \
    >> "OUTPUTPERCEPTRON"
    echo "BREAKBREAKBREAK" >> OUTPUTPERCEPTRON
done
```

1.2.2 Python Regex of WEKAtext output

Python Regex to Extract Data from txt files

```
#python
file1 = "WEKAoutputs/output_1nn_subspace.txt"
# Function to extract metrics from a section
def extract_metrics(section):
    metrics = {}
    patterns = {
        "Accuracy (%)": r"Correctly Classified Instances\s+\d+\s+([\d\.]+) %",  # Extract only the percentage
        "Kappa statistic": r"Kappa statistic\s+([\d\.]+)",
        "Mean absolute error": r"Mean absolute error\s+([\d\.]+)",
        "Root mean squared error": r"Root mean squared error\s+([\d\.]+)",
        "Relative absolute error (%)": r"Relative absolute error\s+([\d\.]+) %",  # Capture percentage directly
        "Root relative squared error (%)": r"Root relative squared error\s+([\d\.]+) %",  # Capture percentage directly
        "Total Number of Instances": r"Total Number of Instances\s+(\d+)"
    }
    for key, pattern in patterns.items():
        match = re.search(pattern, section)
        if match:
```

```
            metrics[key] = float(match.group(1))  # Convert matched string to float
        return metrics

# Read the file
with open(file1, "r") as file:
    data = file.read()

# Split by model runs
model_runs = data.split("BREAKBREAKBREAK")

# Lists to hold the results
train_results = []
test_results = []


# Process each model run
for run in model_runs:
    train_section = re.search(r"=== Error on training data ===\n(.*?)\n=== Detailed Accuracy By Class ===", run, re.DOTALL)
    test_section = re.search(r"=== Stratified cross-validation ===\n(.*?)\n=== Detailed Accuracy By Class ===", run, re.DOTALL)

    if train_section:
        train_metrics = extract_metrics(train_section.group(1))
        train_results.append(train_metrics)

    if test_section:
        test_metrics = extract_metrics(test_section.group(1))
        test_results.append(test_metrics)

# Convert to DataFrames
train_df = pd.DataFrame(train_results)
test_df = pd.DataFrame(test_results)
```

1.3 Appendix - C$_{hristmas}$

1.3.1 Effusive Apology

I'm very[2] sorry for being effusive, but i had a lot of fun investigating the different neural network combinations and this added a lot of length.

1.3.2 BONUS: Using Spotify Recommendation API

Sometimes my friend sends me songs on Spotify to share their music with me. However when I share music back I receive negative feedback on my music taste. I believe that there is bias involved in this criticism, and that my song recommendations are actually similar to theirs. So I've used Spotify's API recommendation feature to generate tracks and send them back. Here's a snippet of the code:

Spotify Recomendation API

```
# Set your client_id and client_secret
CLIENT_ID="xxx000xxx000xxx000xxx000xxx000"
CLIENT_SECRET="000xxx000xxx000xxx000xxx000xxx"

# Request the access token from Spotify
curl -X POST "https://accounts.spotify.com/api/token" \
     -H "Content-Type: application/x-www-form-urlencoded" \
     -d "grant_type=client_credentials&client_id=CLIENT_ID&
   client_secret=CLIENT_SECRET"

# Access token from above
ACCESS_TOKEN="BQDUV8DUrvv7mPY-ui-pzLZaMuRgcZ30IZOwR1IeQ84CXMwl-
   ygnnxGD5XsLK-
   MviNcKnpPIkjDNFDUQWypXcaccF7QXAhSoelcGVwt35itiVBG8D2g"

LIMIT=1 # Number of response tracks
MARKET="GB"
SEED_TRACKS="3n3Ppam7vgaVa1iaRUc9Lp" # an example track that was
    shared with me

# Make the API call to get recommendations
curl -X "GET" "https://api.spotify.com/v1/recommendations?limit=
   LIMIT&market=MARKET&seed_tracks=SEED_TRACKS" \
     -H "Authorization: Bearer ACCESS_TOKEN"
```

If you made it here it's time for a pint

[2] very very very very very very very very very very very very very very very very very very very very very very sorry