Model 1: K-Nearest Neighbors Classifier

Model choice

K-nearest neighbors classification works on the principle that it finds the k-nearest data point(s) of the training set to make a prediction. It assigns a label to a class that wins in voting. The weights can be uniform or distance. If it is uniform, each nearest neighbor has the same value, whereas with distance weights, closer neighbors have a larger influence than the ones further away.

There are several reasons why I chose the k-nearest neighbors classifier as the first model. The most important reason is that I knew it works on music genre classification because it is a supervised learning algorithm and performs well on classification tasks. The dataset consists only of 5645 records and 12 variables. Therefore, although the computational complexity of k-nn is quite high, running the code should not be too slow.

Input to classifier

After splitting the data into training and test data (train size = 0.25), a scatter plot is made to show an interaction between variables (figure 1). On the plot, it is visible that there are quite many outliers, some distributions are skewed, and the classes are not separated extensively in some variables. The shape of X_train is 4233 rows and 12 columns and the shape of X_test is 1412 rows and 12 columns. Summary statistics, consisting of mean, sd, etc., of each variable is also inspected.

As a part of inspecting the data, the presence of missing values in the dataset is checked and several of them are detected. For this reason, a simple imputer with a median strategy is used, as the median strategy is good for skewed distributions. This means that missing values are replaced with the median value of non-missing values. Since the k-nn is based on distance, it is necessary to have variables of the same magnitude because otherwise, the algorithm would put more importance on variables with higher magnitude. To solve this problem, a standard scaler is used to make a 0 mean of each variable and a standard deviation of 1.

Hyperparameter tuning

2 hyperparameters for tuning are selected: k (number of neighbors) and metric. To get an idea of how many k should be tested, a graph of the first 50 k-neighbors with uniform weights on the x-axis and accuracy on the y-axis (figure 2) is plotted. If the k-neighbors number is too small, the model has very high training accuracy but the model is overfitted, so the test accuracy is low. With a too large number of neighbors, both test and training accuracy are falling. Therefore, it can be seen that the optimal k-neighbors is not higher than 30, so the k-range between 1 and 30 is used for tuning the model.

To tune the model, a grid search is used. It tries all combinations of k-neighbors from 1 to 30 and Minkowski, Euclidean, Manhattan, and Hamming metrics. Metric is used to calculate the distance between points. Manhattan matric calculates it in a grid-like path, Euclidean is a straight line, Minkowski combines Manhattan and Euclidean matrics, and Hamming calculates the distance between 2 vectors. Manhattan metric is tested because it is preferable in high dimensions, Euclidean gives the shortest distance, Minkowski is the most used, and Hamming is used for comparing binary data strings. Sorting it by the highest accuracy value revealed that the combination of 17-neighbors and Manhattan matric has the highest mean test score. If the chosen hyperparameters were different, for example, k is 10 and metric is Euclidean, the training accuracy would be higher, while

test accuracy higher and distance measures would be shorter because it would measure them by straight lines.

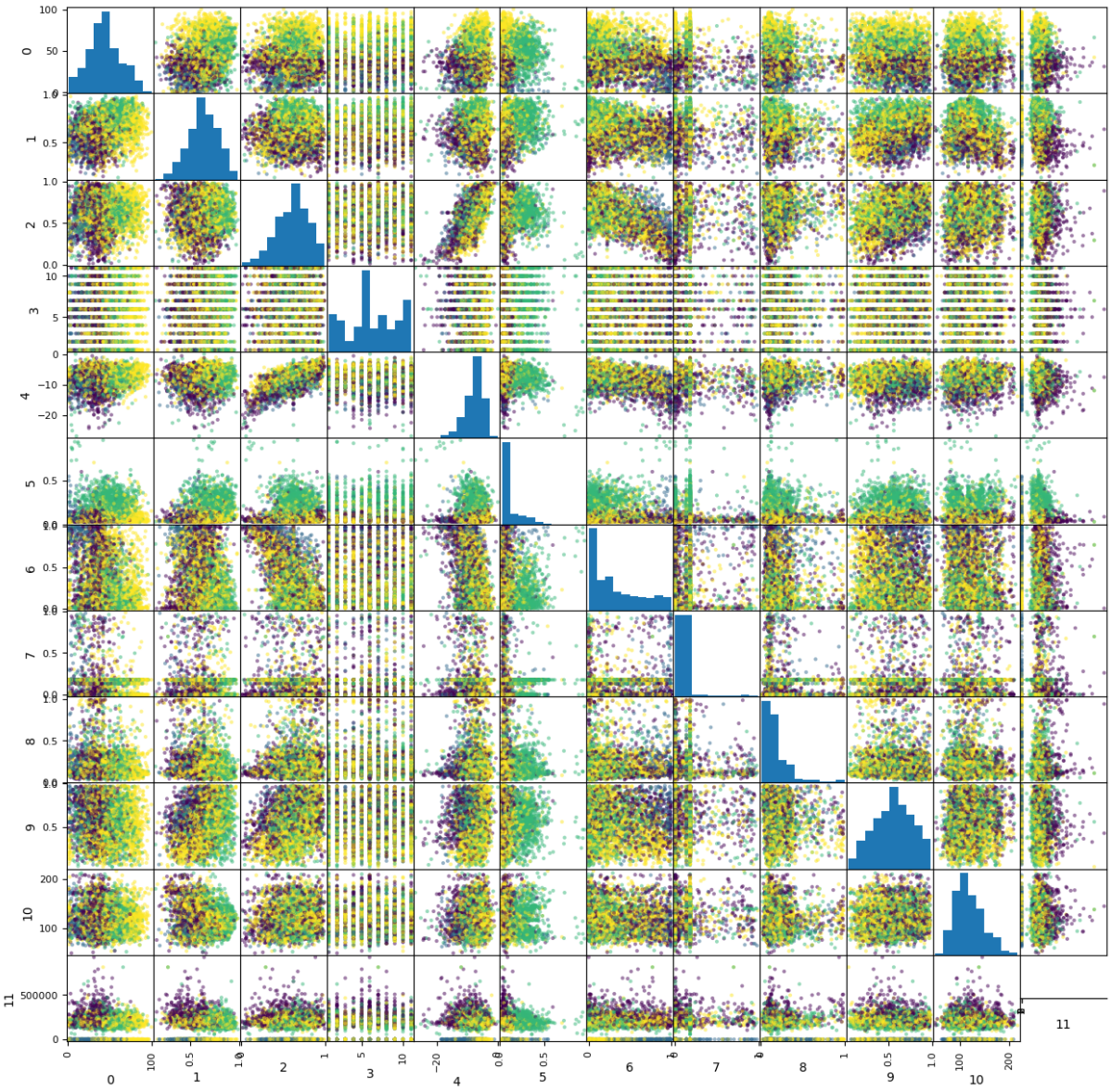

Figure 1: scatter matrix representing relationships between variables

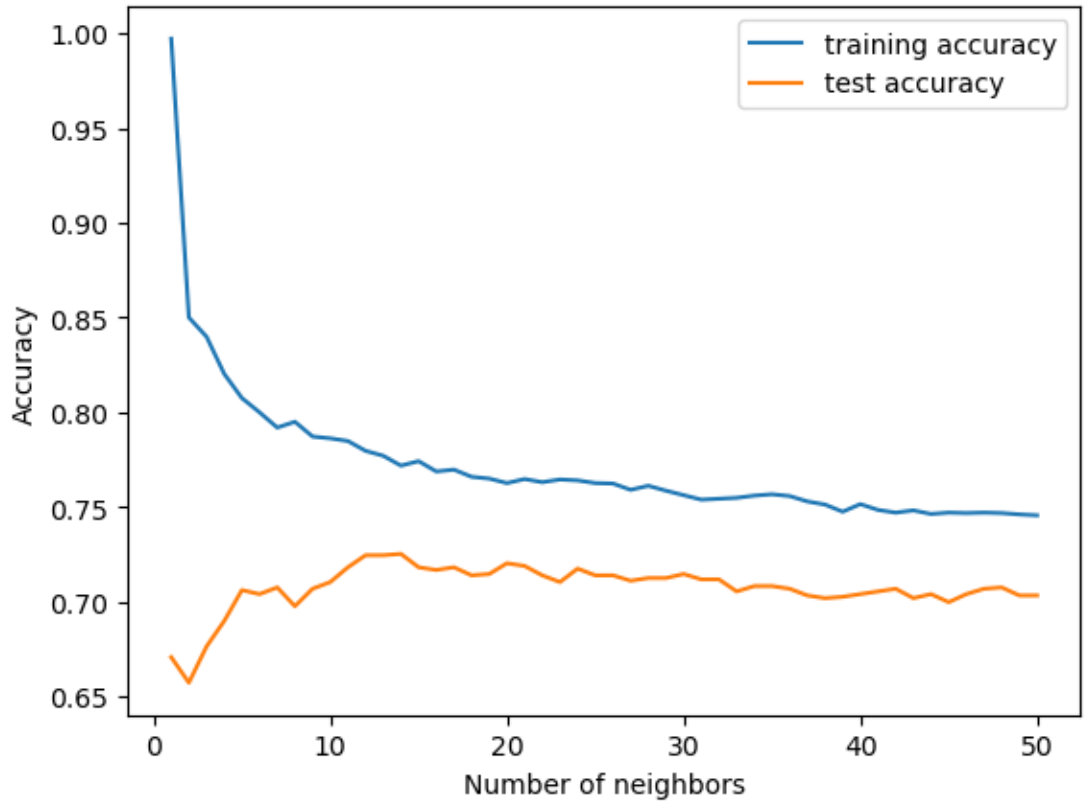

Figure 2: Accuracy vs. Number of neighbors

Model training

To find out how well the model generalizes on a dataset, 10-fold cross-validation is done. It splits the data into 10 blocks and always uses 9/10 of the data for training the method and 1/10 of the data for testing (validation set). It runs until each block of data has been used for testing. For this model, the average cross-validation score is 0.74 and scores range from 0.72 to 0.77. The performance of the training set is also 0.74, therefore it shows that the model is not overfitting because if it would, the cross-validation score would have to perform poorly, while it would perform well on the training set. All in all, the best setting for the model is a combination of 17-neighbors and Manhattan matric hyperparameters because they are not overfitting and optimize mean test scores.

To be able to visualise the decision regions, the number of dimensions are reduced to 2 with t-distributed Stochastic Neighbor Embedding (t-SNE). Then it fits the classifier to the t-SNE results and y_train and plots the decision regions (figure 3).

Results

The overall accuracy of this model is 0.71, the weighted f1-score is 0.71, and the cross- validation score is 0.74, which shows relatively good performance and generalization of the model. The confusion matrix (figure 4) shows rows that correspond to the true classes and columns that correspond to predicted classes. Numbers in green squares represent the number of test values that are predicted correctly. Numbers that are not in green rectangles are not predicted correctly. For example, 194 samples of class 0 (blues) are classified in the prediction as blues, 9 samples of blues as hip-hop, and 116 as pop.

The classification report (figure 5) shows the precision, recall, and f1-score of each class. Class blues has relatively low both precision and recall, so the f1-score is also relatively low as it considers both of them. hip-hop has also low recall and f1-score but precision is higher. The reason is that fewer classes are mistakenly classified as hip-hop (precision) but there are more false negatives for hip-hop (recall). The highest f1-score has class 1 (Bollywood) with both recall and precision of 0.90. Therefore, while blues and hip-hop are the most difficult classes to classify, Bollywood is the easiest.

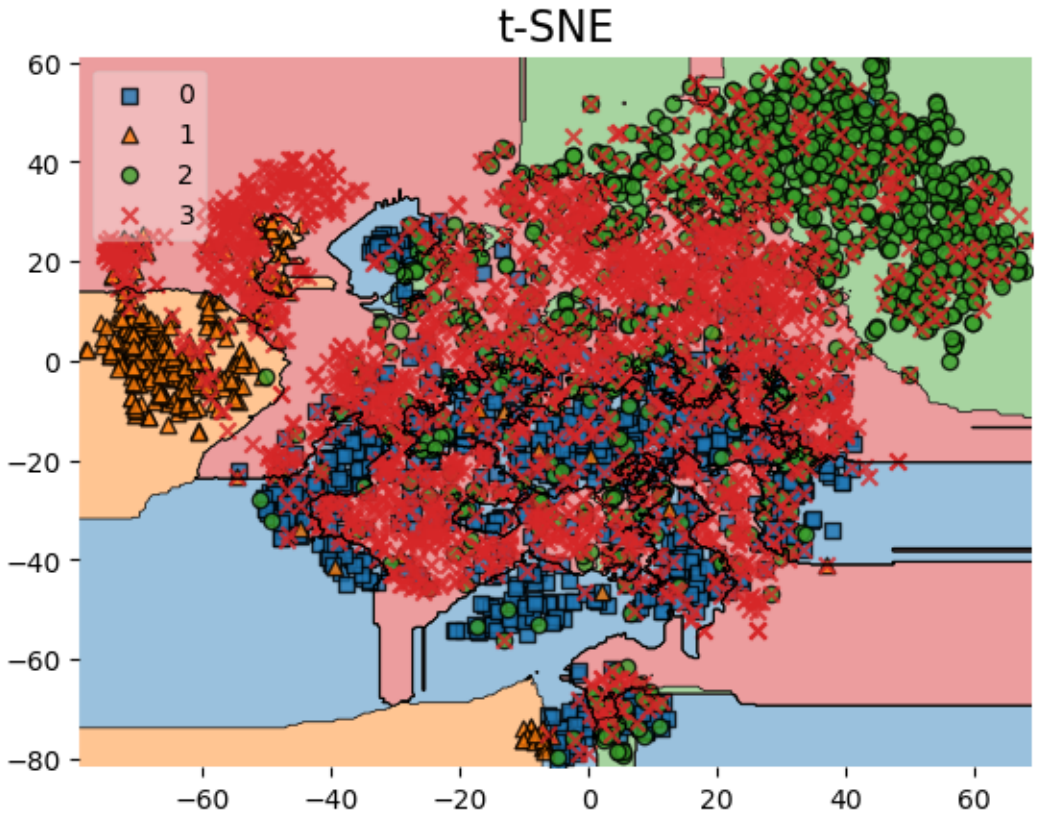

Figure 3: 2-dimensional decision region boundary

Model 2: Decision Tree Classifier

Model choice

A decision tree classifier splits samples into regions based on if/else questions by decision nodes. Leaf nodes represent the class. Training of this supervised learning algorithm is done by finding the best split at a feature. The predictions are done by moving from the root node and answering questions at decision nodes until the value arrives at the leaf node.

I have chosen the decision tree because it performs well in classification. One of the main advantages is that it does not require many assumptions. The only hyperparameters I use are chosen for pruning but even without them, it would be possible to make a decision tree model, although there would be a high risk of overfitting. Another advantage of the decision tree algorithm is that it produces an easily readable visualization of the final tree.

Input to classifier

Simple imputation with a median strategy is used to replace missing values. Although it is not necessary, a scaler is used to increase model accuracy. These two processes are applied the same way as in the k-nn model.

Hyperparameter tuning

3 hyperparameters are tuned using grid search. These are the maximum depth of a tree (max_depth), the minimum number of samples required to split an internal node (min_sample_split), and the minimum number of samples required to be at a leaf node (min_samples_leaf). Tuning hyperparameters can take a long time, so plotting the accuracies of each of them is meant to reduce

candidate values. The maximum depth graph (figure 6) shows that after max_depth=20, both training and test accuracy are stable. However, test accuracy has its peak before max_depth = 10 and then it decreases. Therefore, max_depth values are tested until 10. The minimum samples split (figure 7) grows in test accuracy and declines in training accuracy as the minimum samples split parameter increases. After testing on a larger set of values, the optimal tuning range from 10 to 25 is determined. With increasing minimum samples leaf parameter value (figure 8), training accuracy declines, while test accuracy stays relatively at the same level but with high fluctuations. Therefore, values between 5 and 10 are tuned. Grid search determines that the optimal hyperparameters combination (highest mean test score) is with a maximum depth of 9, minimum samples split of 25, and minimum samples leaf of 5. If other hyperparameters were chosen, for example, maximum depth=3 (instead of 9), both test and training accuracy would be lower. If the minimum samples split is 14 (instead of 25), the training accuracy is higher and the test is lower. If the minimum samples leaf is 25 (instead of 5), the training accuracy is lower, while test accuracy is higher.

```
Confusion matrix:
[[194   0   9 116]
 [  3  87   0   7]
 [ 23   1 216 125]
 [ 55   4  62 510]]
```

Figure 4: confusion matrix of k-nn model

	precision	recall	f1-score	support
0	0.71	0.61	0.65	319
1	0.95	0.90	0.92	97
2	0.75	0.59	0.66	365
3	0.67	0.81	0.73	631
accuracy			0.71	1412
macro avg	0.77	0.73	0.74	1412
weighted avg	0.72	0.71	0.71	1412

Figure 5: classification report of k-nn model

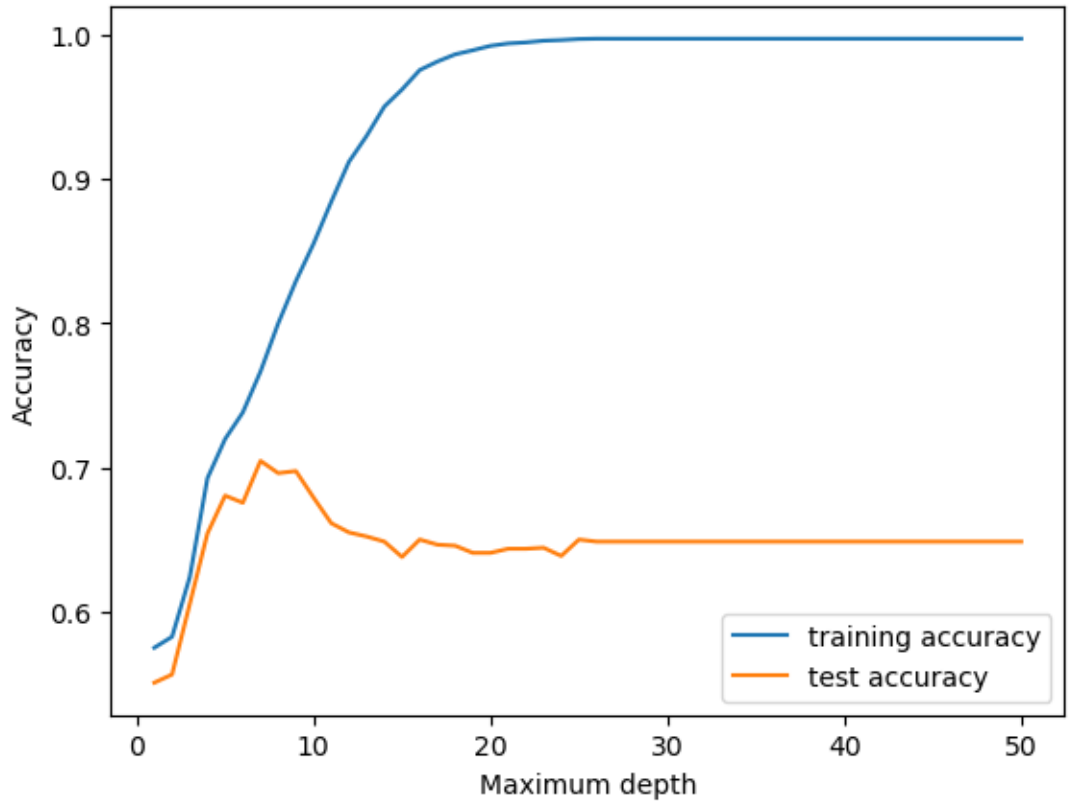

Figure 6: Accuracy vs. Maximum depth

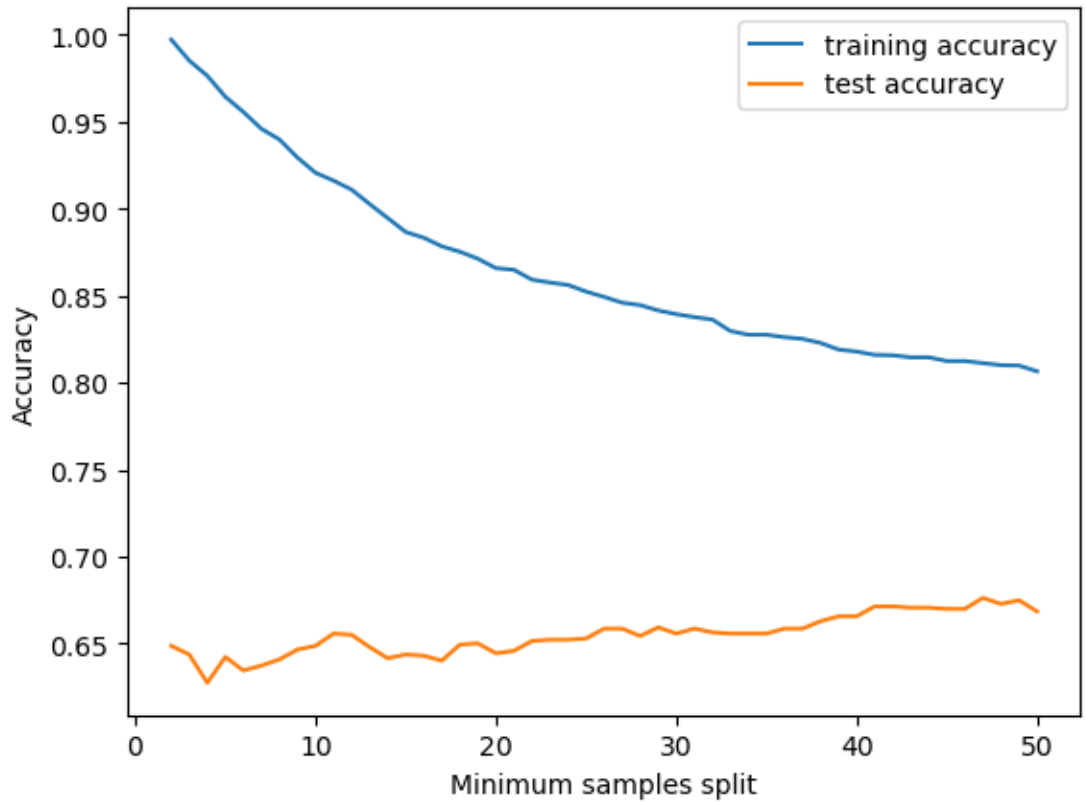

Figure 7: Accuracy vs. Minimum samples split

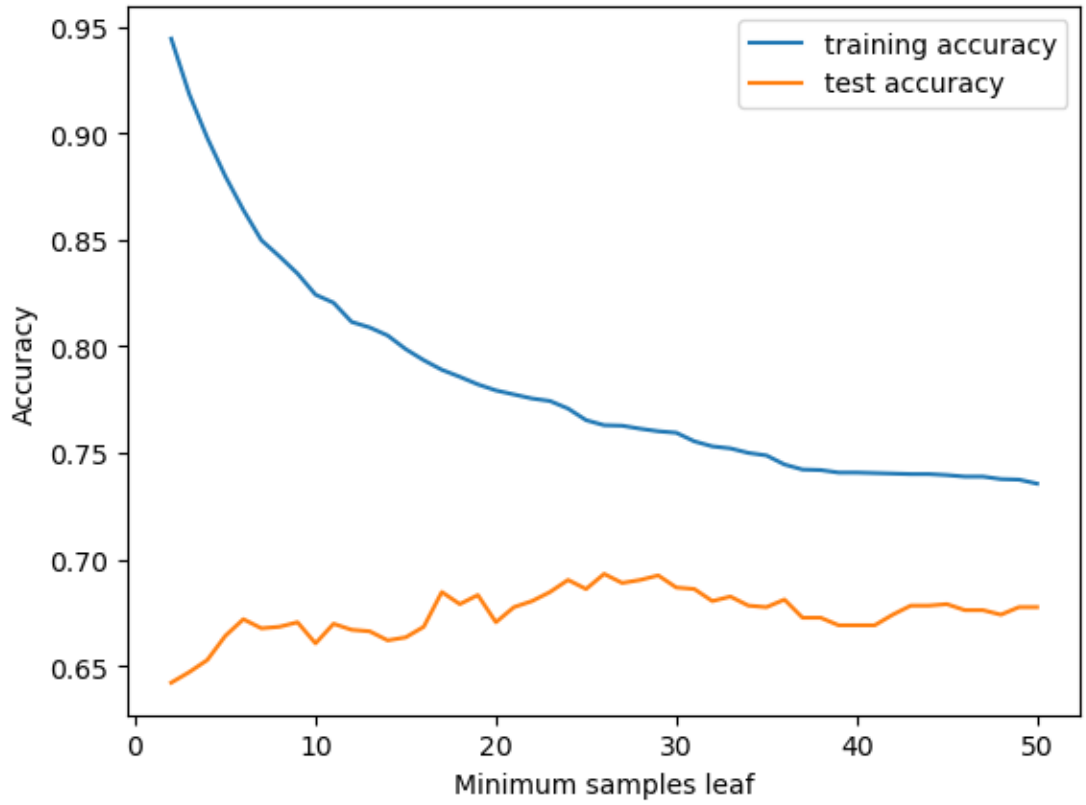

Figure 8: Accuracy vs. Minimum samples leaf

Model training

Similarly to the first model, cross-validation is done to prove the final accuracy is not just a result of pure luck and that the model is not overfitting. Cross-validation scores for this model range from 0.68 to 0.73, so the range is not too wide. The mean cross-validation score is 0.71, the same as the training accuracy, so the model is not overfitting. Therefore, the optimal combination of hyperparameters is a maximum depth of 9, minimum sample split of 25, and minimum sample leaf of 5.

This decision tree (figure 9) is built using Gini Impurity. When selecting which feature should be used for splitting, the feature with the lowest impurity is determined as the best; it divides the highest number of features into the correct class. The importance of each feature for this model is shown on figure 10.

Results

The overall accuracy of the model is 0.70 and the cross-validation score is 0.71. Since the dataset is quite unbalanced, I use the weighted average f1-score, which is 0.70. Overall, these measures show that the model performs relatively well.

The confusion matrix (figure 11) and classification report (figure 12) show that class 2 (hip-hop) is the hardest to predict. 225 hip-hop classes are predicted correctly as hip-hop, while 117 of them are predicted as pop and 23 as blues. Reversely, 19 blues and 79 pop samples are wrongly classified as hip- hop. Class 0 (blues genre) is only a bit easier to classify with an f1-score of 0.67. On the other

hand, class 1 (Bollywood) is the easiest to classify with an f1-score of 0.87, however, it has the lowest support as there are the least samples in the Bollywood class.

Figure 9: decision tree

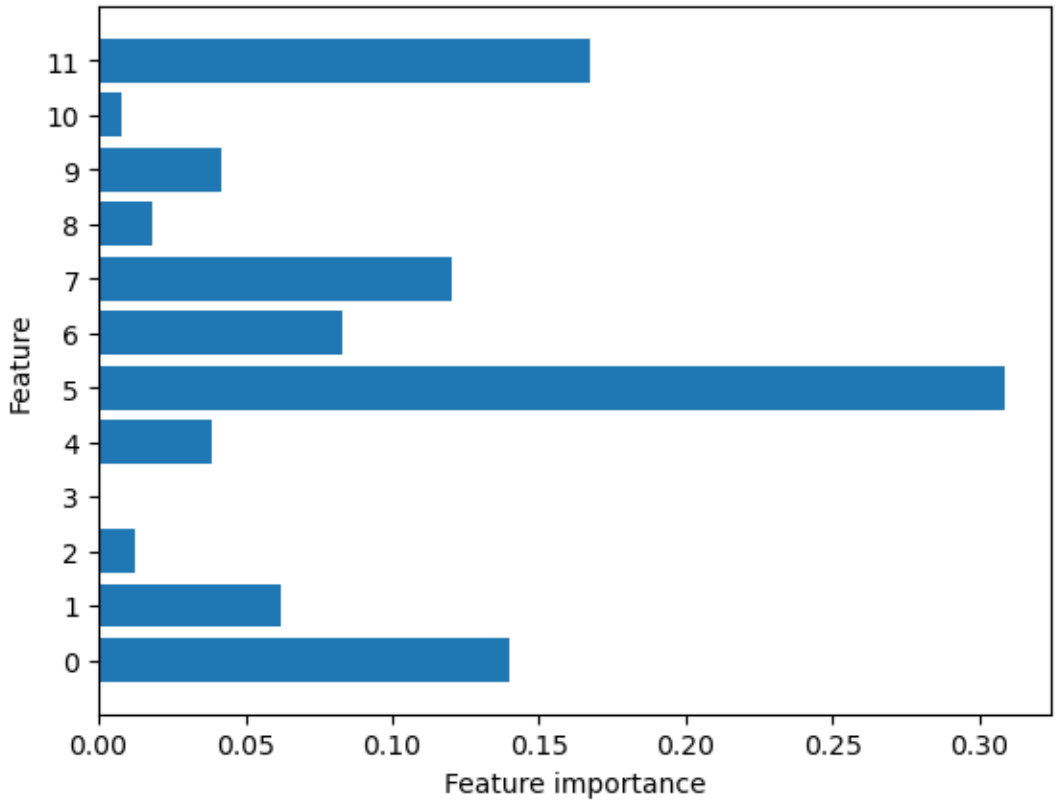

Figure 10: feature importance

```
Confusion matrix:
[[208   0  19  92]
 [  3  83   0  11]
 [ 23   0 225 117]
 [ 65  10  79 477]]
```

Figure 11: confusion matrix of decision tree model

	precision	recall	f1-score	support
0	0.70	0.65	0.67	319
1	0.89	0.86	0.87	97
2	0.70	0.62	0.65	365
3	0.68	0.76	0.72	631
accuracy			0.70	1412
macro avg	0.74	0.72	0.73	1412
weighted avg	0.70	0.70	0.70	1412

Figure 12: classification report of decision tree model

Comparison of both models

Overall, the k-nn classification model performs better. Its cross-validation score is 0.74, while it is only 0.71 for the decision tree classification model. Furthermore, test accuracy and average weighted f1-score are 0.71 for the k-nn, while it is 0.70 for the decision tree.

The speed of both models is measured in wall time and it measures grid search, fitting, and cross-validation time. The decision tree model performs faster at 40.95 seconds, while it takes 1 minute and 10.90 seconds to perform mentioned tasks in the k-nn model. The biggest difference in training speed is in the grid search part, where the decision tree is faster the k-nn by 30.5 seconds. However, the decision tree is also faster in fitting and cross-validation.

In conclusion, although the k-nn model has higher accuracy than the decision tree, the decision tree might still be a better option in cases where model speed needs to be optimized.

References

Agarwal, V. (2019, September 26). *plot_decision_regions with error "Filler values must be provided when X has more than 2 training features."*. StackOverflow. Retrieved December 10, 2022, from https://stackoverflow.com/questions/52952310/plot-decision- regions-with-error-filler-values-must-be-provided-when-x-has-more

ARUNMOHAN_003. (2021). *Pruning decision trees – tutorial*. Kaggle. Retrieved December 10, 2022, from https://www.kaggle.com/code/arunmohan003/pruning-decision-trees- tutorial

Baeldung. (2022, November 15). *F-1 score for multi-class classification*. Baeldung on Computer Science. Retrieved December 10, 2022, from https://www.baeldung.com/cs/multi-class-f1-score

Boorman, G. *Supervised Learning with scikit-learn*. DataCamp. Retrieved December 10, 2022, from https://app.datacamp.com/learn/courses/supervised-learning-with-scikit- learn

Chandradas, A. (2021, February 23). *5 Methods to Check for NaN values in in Python*. Towards Data Science. Retrieved December 10, 2022, from https://towardsdatascience.com/5-methods-to-check-for-nan-values-in-in-python-3f21ddd17eed

Godalle, E. (2022, November 28). *DecisionTree hyper parameter optimization using Grid Search.* ProjectPro. Retrieved December 10, 2022, from https://www.projectpro.io/recipes/optimize-hyper-parameters-of-decisiontree-model- using-grid-search-in-python

Gohrani, K. (2019, November 10). *Different Types of Distance Metrics used in Machine Learning*. Medium. Retrieved December 10, 2022, from https://medium.com/@kunal_gohrani/different-types-of-distance-metrics-used-in- machine-learning-e9928c5e26c7

Gulati, H. (2022, February 11). *Hyperparameter Tuning in Decision Trees and Random Forests*. Section. Retrieved December 10, 2022, from https://www.section.io/engineering-education/hyperparmeter-tuning/

Jaadi, Z. (2022, September 26). *A Step-by-Step Explanation of Principal Component Analysis (PCA)*. Builtin. Retrieved December 10, 2022, from https://builtin.com/data- science/step-step-explanation-principal-component-analysis

Kumar, V. (2021, August 19). *KNN Classifier in Sklearn using GridSearchCV with Example*. Machine Learning Knowledge. Retrieved December 10, 2022, from https://machinelearningknowledge.ai/knn-classifier-in-sklearn-using-gridsearchcv-with-example/

Leung, K. (2022, January 4). *Micro, Macro & Weighted Averages of F1 Score, Clearly Explained*. Towards Data Science. Retrieved December 10, 2022, from

https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f

M., J., (2020, November 13). *Construct a Decision Tree and How to Deal with Overfitting*. Towards Data Science. Retrieved December 10, 2022, from https://towardsdatascience.com/construct-a-decision-tree-and-how-to-deal-with- overfitting-f907efc1492d

Shaneb. (2020, July 30). Is there a numpy builtin to reject outliers from a list. StackOverflow. Retrieved December 10, 2022, from https://stackoverflow.com/questions/11686720/is- there-a-numpy-builtin-to-reject-outliers-from-a-list

Sharma, P. (2019, August 25). *Why is scaling required in KNN and K-Means?* Medium. Retrieved December 10, 2022, from https://medium.com/analytics-vidhya/why-is- scaling-required-in-knn-and-k-means-8129e4d88ed7