Final Project

Tina Giagnoni

April 19, 2021

1 Introduction

Famous artists frequently have a distinct style which a casual viewer can easily identify the artist that created a painting. However, occasionally experts even debate over who the artist was or if a painting is a fake. We will use machine learning techniques to determine if we can find an algorithm that can identify artists given painting features as well as the casual viewer can.

We have a data set of 64 features for 2220 paintings painted by one of 8 artists or an "other" artist. The artist labels are as follows: 0: Pierre-Auguste Renoir, 1: Raphael, 2: Leonardo da Vinci, 3: Sandro Botticelli, 4: Francisco Goya, 5: Vincent van Gogh, 6: Pablo Picasso, 7: Albrecht Durer, 8: Others. Figure 1 shows a sample from each of the 9 classes.

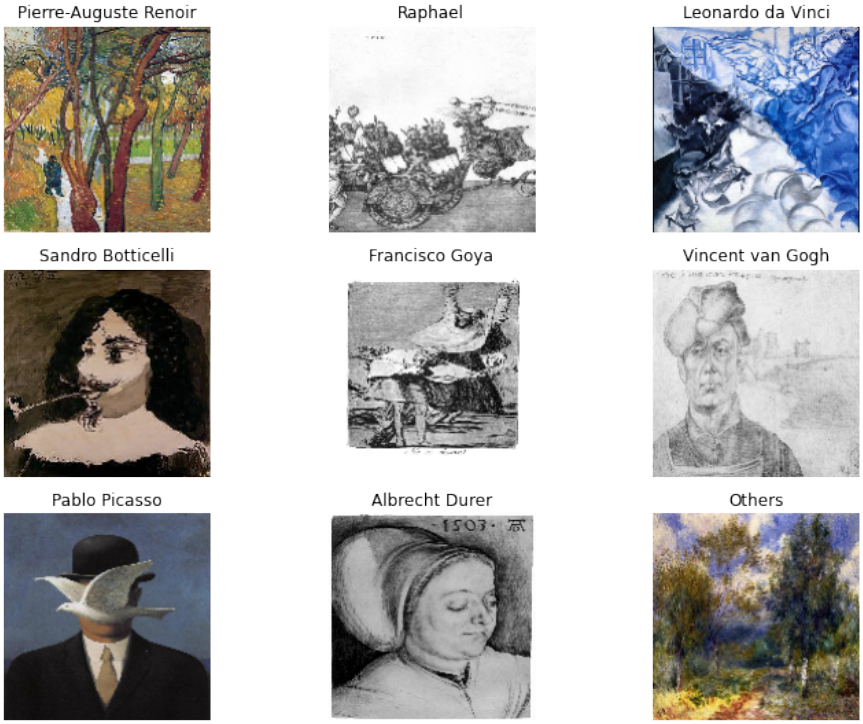

Figure 1: Sample from each artist (class)

Before preprocessing the data we checked the class distribution of the data which is shown in figure 2. We can see that classes 1, 2, and 3 are underrepresented compared with the other classes. This will effect how we interpret the scores from our experiments.

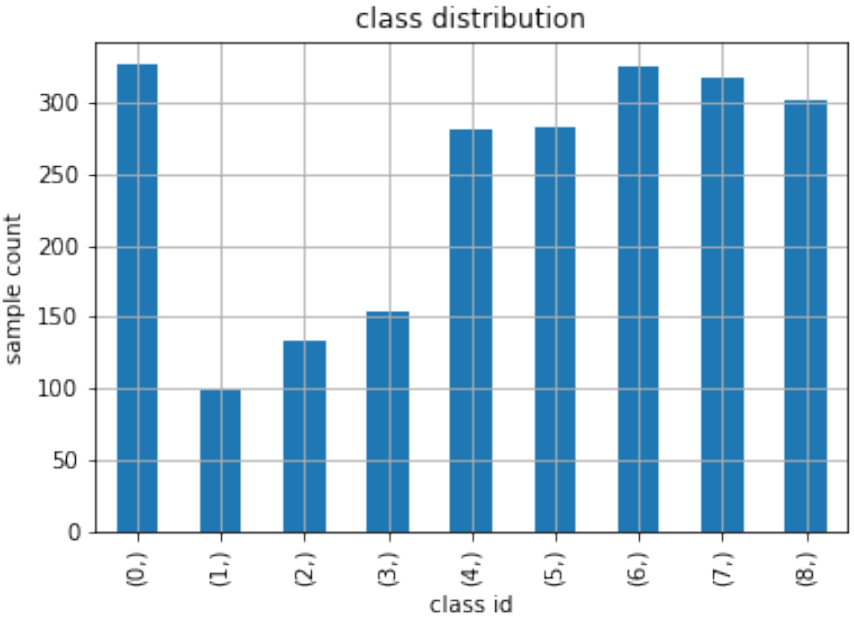

Figure 2: Class distribution

We will first determine a preprocessing procedure by comparing different scaling methods, principal component analysis (PCA), and feature selection. The four base classifiers we will be using are k-nearest neighbors (KNN), kernel support vector machines (SVC), multilayer perceptron (MLP), and random forest (RF). Once we have determined the best preprocessing procedure and hyperparameters for the individual classifiers we will attempt to improve upon some base classifiers with two ensemble methods: bagging and adaboost. We will compare all of our classifiers by three different measures: accuracy, F1 measure, and ROC AUC.

2 Preprocessing

Preprocessing is an essential step in model building. While this data set is clean (e.g. no missing entries) but after looking at the feature statistics the data does not appear to have been scaled, and has 64 features. So we can explore the benefits scaling and dimensionality reduction. Before beginning any experiments we split our data into training and testing using a 90-10 split. So the training set size is 1998 and the test set size is 222. The test set will be reserved for final comparisons so the algorithms do not learn on it during the training process. For testing within the training process we will use 4-fold cross validation on the designated training set.

To use k-fold cross validation we designate, k, the number of times we will "fold" or split the data set. Meaning, if we have a data set of size n and we are applying k-fold cross validation, we would randomly split the data set into k validation sets with n/k samples in

each set (give or take a sample if n is not divisible by k). Then we run the algorithm k times, training the data on the $n - n/k$ samples not in the validation set, and then test the generated model with the reserved samples. Thus, in our experiments where we use 4-fold cross validation the training set size is 1498 or 1499 and the validation set size is 499 or 500.

We will be comparing three different preprocessing techniques: scaling, principal component analysis (PCA), and feature selection. We will explain what each technique does, our results, then decide on what combination of techniques applied to the data yields the best results which will be used to train our models.

2.1 Scaling

Scaling can be a very important step in preprocessing. Many algorithms are sensitive to data scaling so that when features are on different scales (magnitudes) some algorithms tend to deem features on larger scales to be more important even when they may not be. We compared three scaling processes: no scaling, standard scaler and min-max scaler. With no scaling, we just use the data as it is. Standardizing data simply takes data and centers it around 0 and adjusts the standard deviation to 1 using

$$x_{std} = \frac{x_i - \overline{x}}{s}$$

where $\overline{x}$ is the mean and s is the standard deviation of the training data. The min-max scalar shifts all values to be between 0 and 1 by

$$x_{std} = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}.$$

We used a gridsearch with 4-fold cross validation to get the scores presented in table 1, and used three different scoring measures: accuracy, F1 measure and ROC. We want to maximize all of these measures.

Accuracy tells us how well our model correctly classifies the data, with the formula

$$\text{Accuracy} = \frac{\text{number of correctly classified samples}}{\text{total number of samples}}$$

The F1 measure is

$$F1 = 2 \times \frac{\text{precision } \times \text{ recall}}{\text{precision } + \text{ recall}}$$

Precision tells us how good the model is at labeling the points correctly classify true positives, and recall tells us how good the model is at catching all positive cases. The F1 score tells gives us a balanced representation of precision and recall. In our case, since we have a multi-class problem, this is the average of the F1 score of each class. Typically, for unbalanced data sets F1 can be a more informative measure than accuracy.

The receiver operating characteristic curve (ROC) and area under the curve (AUC) are related as AUC is the area percentage under the curve. The ROC curve plots sensitivity against 1-specificity and assesses the classification model's performance at different threshold

values, and we are able to analyze both characteristics at the same time. Similar to F1, the ROC AUC can be a more informative measure than accuracy with unbalanced data sets.

Table 1 shows the mean accuracy, F1 and ROC validation scores from the 4 fold cross-validation for each of the four algorithms. Note that we will describe in detail each algorithm in section 3.

	KNN			SVM			MLP			RF		
	None	Std	M-M	None	Std	M-M	None	Std	M-M	None	Std	M-M
Acc	0.840	0.767	0.819	0.909	0.894	0.917	0.926	0.927	0.931	0.870	0.870	0.870
F1	0.837	0.755	0.811	0.912	0.895	0.918	0.926	0.926	0.931	0.868	0.868	0.868
ROC	0.970	0.942	0.958	0.995	0.992	0.995	0.995	0.993	0.994	0.983	0.983	0.983

Table 1: Mean validation scores for each classifier and scaling procedure

For KNN, no scaling performed the best, and then min-max scaling performed similarly, and standardizing the data did not do well at all with an accuracy of 0.767 F1 of 0.755. The ROC scores were similar despite scaling method.

When we ran the gridsearch on kernel SVM, we also tried different kernels (using default parameters): linear, polynomial,and rbf. It was clear that the rbf kernel preformed the best since it scored consistently higher than the other two kernels on every measure. From this point on, we will only use the rbf kernel with our SVM. The min-max scaler performed the best with accuracy of 0.917, F1 of 0.918 and ROC of 0.995. No scaling did the second best and standard scaling saw a large drop in the scores.

The MLP algorithm performed well overall with the min-max scaler performing the best with the accuracy and F1 measures, but doing marginally better with no scaling and the ROC measure. We note that SVM and MLP algorithms are sensitive to scaling, so it is interesting that for both algorithms no scaling performs competitively with the other two scaling methods. This indicates that the data may already be on similar scales or the difference in scales of the features is meaningful.

The random forest algorithm is resistant to the scaling of the data, so all the scores were the same.

Moving forward, we will use no scaling with KNN and RF, and min-max scaling with SVC and MLP algorithms. Figure 3 shows the accuracy, F1 and ROC training and testing scores for each algorithm (red/pink for KNN, purple/lilac for SVC, blue/cyan for MLP and green/mint for RF). We can see that KNN and RF consistently overfit by the larger gaps between the training and testing scores across all measures for every scaling procedure.

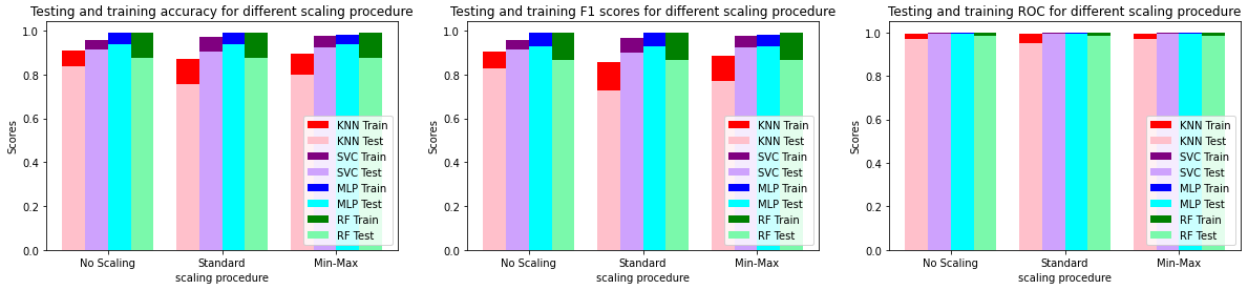

Figure 3: Different scaling procedures accuracy, F1 and ROC scores

2.2 Principal Component Analysis (PCA)

We also applied principal component analysis (PCA) to our data. PCA allows us to reduce the numbers of features we use by projecting the data onto a smaller subspace while still explaining the majority of the variance in the data using singular value decomposition. Typically to use PCA we begin by standardizing the data since the magnitude of the values of the features influence the size of the eigenvectors which determines which principal components are the largest. However, as we saw in section 2.1 all of the methods did not perform as well with standardization. We will work under the impression that the larger values correlate to more important features.

We decided to perform PCA in two different ways. For KNN and RF we will use PCA with no scaling. The scree plot for PCA with no scaling is shown in figure 4. For SVC and MLP we used min-max scaling with PCA, and the scree plot is shown in figure 5. We can see that the scaling makes a difference in the amount of variance explained in the first few principal components. Thus, for each scaling method we used different numbers of principal components to try. With no scaling 28 principal components explain 95% of the variance and 39 principal components explain 99% of the variance. With min max scaling 41 principal components explain 95% of the variance and 49 principal components explain 99% of the variance. In both cases, this is a drop in dimensionality from 64 features.

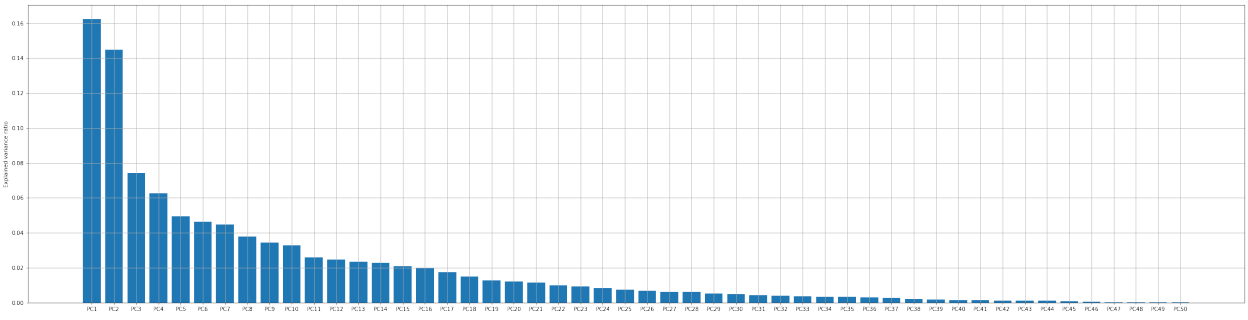

Figure 4: Scree plot for PCA without scaling

For the KNN and RF algorithms we ran a gridsearch and 4-fold cross-validation with the two different number of principal components for no scaling: 28 and 39. For the SVC and MLP algorithms we ran a gridsearch with the two different number of principal components for min max scaling: 41 and 49. We summarized the results in table 2, and included the

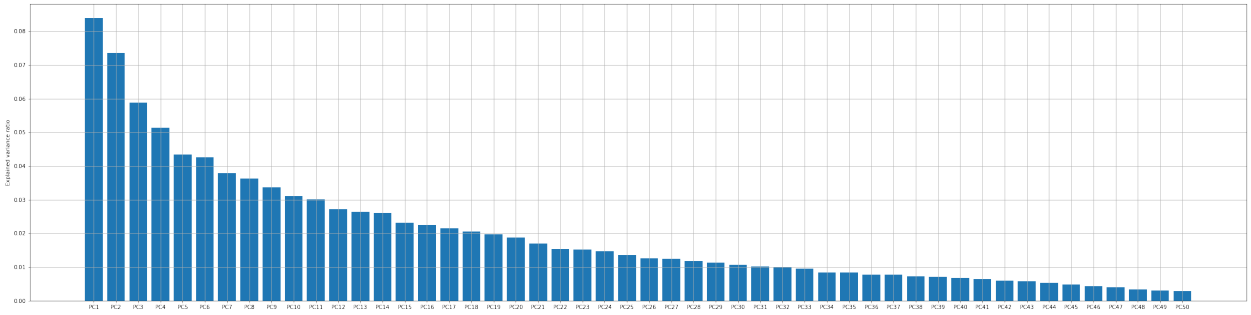

Figure 5: Scree plot for PCA with min-max scaling

scores without using PCA but with each classifier's optimal scaling procedure. Both KNN and RF selected 39 to be the best number of principal components to use and SVC and MLP selected 49 to be the best number of principal components. We can see that KNN has a slight drop in all scores; SVC scores stay consistent; MLP does slightly better with PCA; and RF scores drop significantly.

	Accuracy		F1		ROC		
	Train	Test	Train	Test	Train	Test	PCs
KNN	0.911	0.838	0.905	0.826	0.996	0.970	N/A
SVC	0.974	0.923	0.975	0.922	0.999	0.995	N/A
MLP	0.982	0.937	0.980	0.928	1.00	0.996	N/A
RF	0.992	0.874	0.992	0.864	1.00	0.982	N/A
KNN	0.909	0.833	0.902	0.822	0.996	0.964	39
SVC	0.972	0.923	0.973	0.922	1.00	0.994	49
MLP	0.982	0.941	0.980	0.937	1.00	0.997	49
RF	0.992	0.829	0.992	0.814	1.00	0.981	39

Table 2: Training and testing scores for the best number of principal components

2.3 Feature Selection

Feature selection is another technique for dimensionality reduction. There are many ways to choose features to keep and discard, but we used ANOVA which compares the variance between the different classes to the variance within each class, and measures the linear dependence between the features and the class labels. To implement this in Python we used SelectKBest with f_classif (for classification problems). We ran a grid search and 4-fold cross-validation with 40, 45, and 50 best features, and a summary of the results is in table 3. All algorithms chose 50 to be the best number of features. Reducing the features to 50 had very little effect on the scores for all of the classifiers

	Accuracy		F1		ROC		
	Train	Test	Train	Test	Train	Test	# features
KNN	0.911	0.838	0.905	0.826	0.996	0.970	all
SVC	0.974	0.923	0.975	0.922	0.999	0.995	all
MLP	0.982	0.937	0.980	0.928	1.00	0.996	all
RF	0.992	0.874	0.992	0.864	1.00	0.982	all
KNN	0.911	0.839	0.905	0.826	0.996	0.970	50
SVC	0.974	0.923	0.975	0.922	1.00	0.995	50
MLP	0.982	0.937	0.980	0.931	1.00	0.996	50
RF	0.992	0.874	0.992	0.861	1.00	0.979	50

Table 3: Training and testing scores for selecting the best features

2.4 Combinations and comparisons

Now that we have some idea on how PCA and feature selection effect the scores for each algorithm, we combined the two dimensionality reduction techniques in a grid search with 4-fold cross-validation. The results are summarized in table 4. For KNN the combination of feature selection and PCA dropped the accuracy and ROC AUC scores slightly, and raised the F1 score. SVC and MLP saw no change in accuracy, and no change in F1 for SVC and as slight raise in F1 for MLP, and the ROC AUC scores were negligibly different for SVC but the same for MLP. RF performed worse on the accuracy and F1 measures and negligibly better by the ROC AUC measure.

	Accuracy		F1		ROC			
	Train	Test	Train	Test	Train	Test	# features	PCs
KNN	0.911	0.838	0.905	0.826	0.996	0.970	all	N/A
SVC	0.974	0.923	0.975	0.922	0.999	0.995	all	N/A
MLP	0.982	0.937	0.980	0.928	1.00	0.996	all	N/A
RF	0.992	0.874	0.992	0.864	1.00	0.982	all	N/A
KNN	0.909	0.833	0.902	0.831	0.996	0.964	50	39
SVC	0.974	0.923	0.975	0.922	1.00	0.994	50	49
MLP	0.984	0.937	0.983	0.931	1.00	0.996	50	49
RF	0.992	0.865	0.992	0.858	1.00	0.983	50	39

Table 4: Training and testing scores for selecting the best number of features and principal components

We used the results shown on tables 2, 3 and 4 to determine the optimal preprocessing procedures for each classifier which is shown in Table 5. KNN had the best overall results with selecting 50 features. However, if we look back at figure 3, KNN is prone to overfitting this data set. Therefore, in an attempt to mitigate that we will also include PCA with feature selection for KNN. The SVC classifier showed no improvements over any feature selection or PCA, so we will use both to reduce complexity. MLP was unique in that it was overall improved with PCA, but not with feature selection. Thus, we will only use PCA with the MLP classifier. Finally, RF did poorly with PCA or a combination of feature selection and

PCA. So we will only use feature selection with RF, even though it tends to overfit like KNN the score drops from PCA were too large to justify using it.

	Scaling	# features	PCs
KNN	None	50	39
SVC	Min-Max	50	49
MLP	Min-Max	all	49
RF	None	50	N/A

Table 5: Preprocessing procedure for each algorithm

3 Hyperparameter Tuning

We have already mentioned the four classifiers we are using in this project: k-nearest neighbors(KNN), Gaussian kernel support vector machine (SVC-RBF), Multilayer perceptron (MLP), and Random Forest (RF). Each algorithm has different hyperparameters that need adjusting to optimize the performance of the classifier. We will pick the most important parameters to tune for each algorithm and run a gridsearch with 4-fold cross validation to find the best hyperparameters for each model.

3.1 k-nearest neighbors(KNN)

The KNN algorithm memorizes the training data, and for predictions the algorithm finds the nearest uses the k nearest neighbors to give a label to a new data point, by a majority vote. The algorithm has one parameter: k = the number of neighbors the algorithm uses to average to make a prediction on a new data point. Additionally, the user must also determine a distance metric for the algorithm to use. In our case, we used the default distance metric Minkowski.

Before running the gridsearchcv for different k we selected the best 50 features and used 39 principal components to train and test the KNN algorithm. Figure 6 shows the testing and training accuracy (top) and F1 scores (bottom) for different values of k. We used a grid search to help choose the best k, but a gridsearch only outputs the parameter that gives the highest test score and does not consider overfitting. Thus, figure 6 gives us more insight into the best number of neighbors. For $k < 10$ we have an over complicated model which is overfit to the training data. For $k = 10$ we can see the testing and training accuracy and F1 scores are the close without sacrificing scores, so we would consider this the "sweet spot" which minimizes overfitting while still producing adequate results. Figure 7 gives the ROC curve for our model with $k = 10$. Overall, the ROC curve looks good, but our model falsely classified some paintings as 3 (Botticelli) and 8 (others). Our final scores are in table 6. The scores are not perfect, but we have manged to mitigate some of the overfitting issues we saw before hyperparameter tuning.

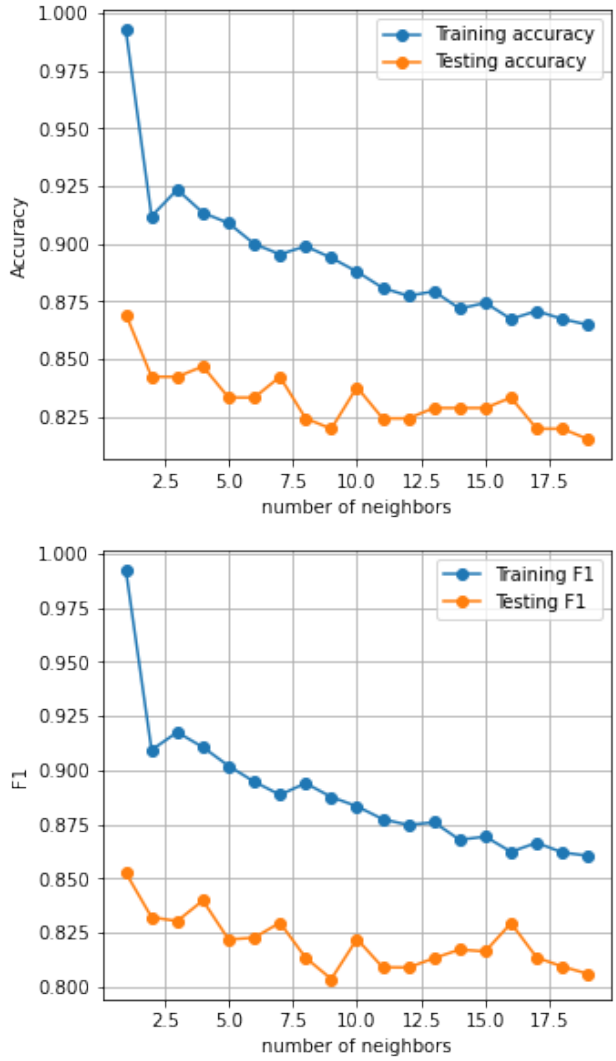

Figure 6: Top row: train/test accuracy for different KNN parameters, Bottom row: train/test F1 score for different KNN parameters

	Training	Testing
Accuracy	0.888	0.838
F1	0.883	0.822
ROC AUC	0.994	0.976

Table 6: KNN scores with 10 neighbors

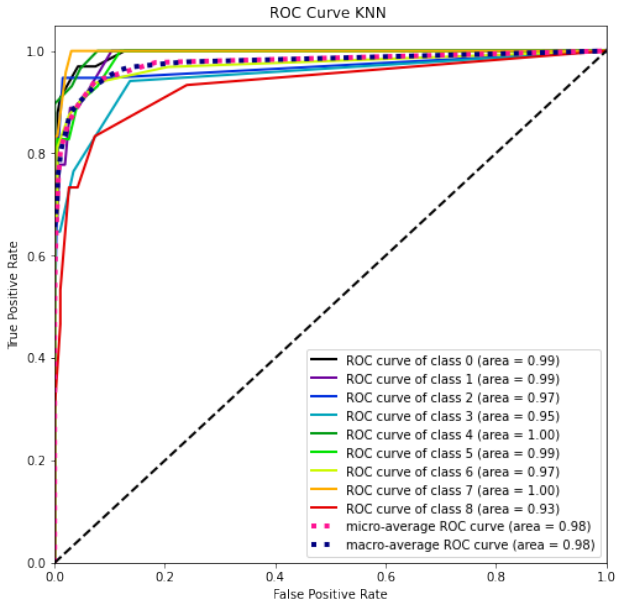

Figure 7: ROC curve for KNN with best parameters

3.2 Gaussian kernel support vector machine (SVC-RBF)

While support vector machines are a linear method, kernel SVM works around the limitations of SVM by projecting the data into higher dimensions. The algorithm creates new features through a function ϕ, which maps existing features to a new feature. With enough additional features, the originally non-linearly-separable data can now be linearly separable. Once the data is linearly separable we use the kernel function to project the model back into the original feature space and now have a classifier that is not linear. Clearly mapping back and forth is time consuming, so to get around the computational expense all we need is a kernel function

$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)}) \cdot \phi(x^{(j)}).$$

There are many kernel options, and in section 2 we determined that the rbf kernel would be best for this data set.

The Gaussian kernel (rbf) is given by

$$K(x^{(i)}x^{(j)}) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) = \exp\left(-\gamma\|x_i - x_j\|^2\right)$$

where the γ parameter which controls the radius of influence of the selected support vectors. If γ is too large, the model can overfit, and if it is too small, the model may underfit. The second parameter to be adjusted is C. Smaller values of C allows for a larger margin between classes and allows for more misclassified samples, thus creating a less complex model.

We started with using the min-max scaler on the data and selecting the the 50 best features and extracting the first 49 principal components. Then we ran a grid search with 4-fold cross-validation to select the best C and γ. Figure 8 illustrates how the testing and training accuracy (top row) and F1 scores (bottom row) change with different values of C (with fixed $\gamma = 1.0$) and differnt values of γ (with fixed $C = 0.6$).

We can see that the distance between the testing and training scores are minimized at $C = 0.6$ and $\gamma = 1.0$, and in table 7 we can see that the training and testing scores with the best parameters are relatively close indicating a model that is not over or underfit. Furthermore, the ROC curve (figure 9) our model misclassified some samples as classes 2 (Leonardo da Vinxi), 6 (Pablo Picasso), and 8 (others).

	Training	Testing
Accuracy	0.953	0.932
F1	0.953	0.934
ROC AUC	0.999	0.996

Table 7: SVM scores with $C = 0.6$ and $\gamma = 1.0$

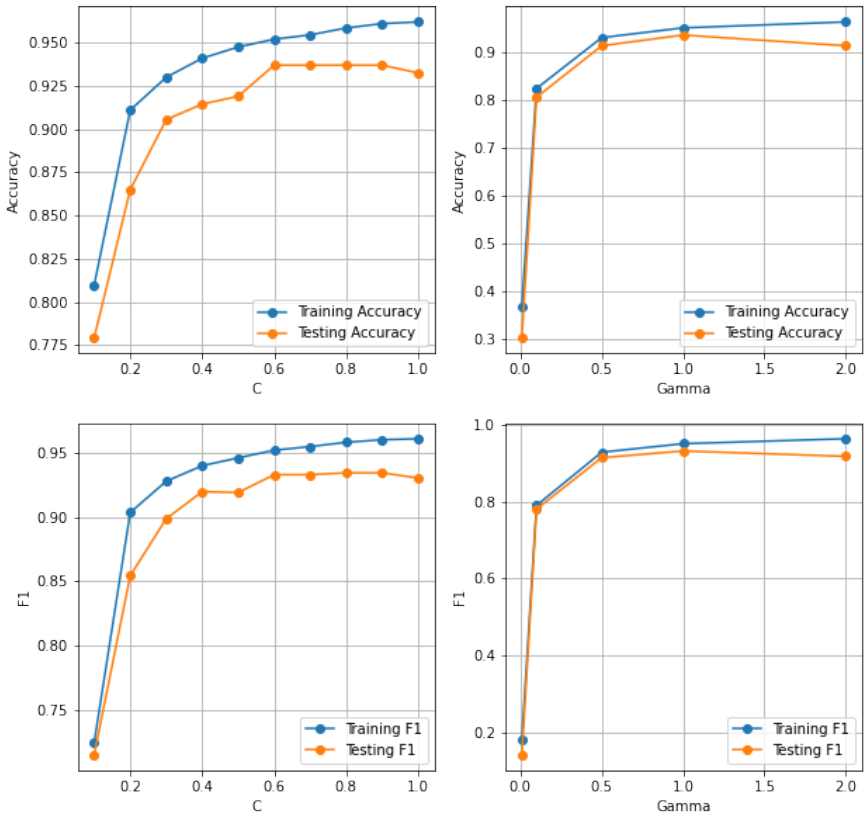

Figure 8: Top row: train/test accuracy for different SVC parameters, Bottom row: train/test F1 score for different SVC parameters

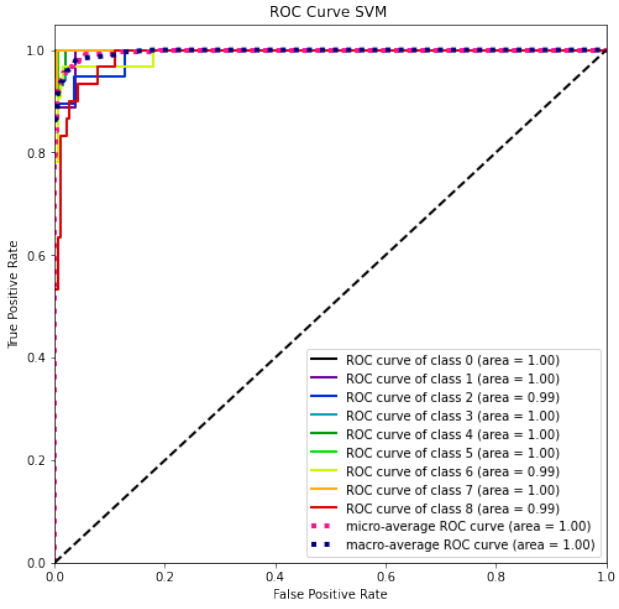

Figure 9: ROC curve for SVC with best parameters

3.3 Multilayer perceptron (MLP)

The multilayer perceptron (MLP) is a generalization of linear models that use many steps of processing (using a neural network) to make a model. It has at least three layers: an input layer, hidden layer, and output layer. The MLP can have many hidden layers, which can create a non-linear decision boundaries when coupled with non-linear activation functions. The activation function is applied to the result after computing the weighted sum for each hidden unit.

We started by using the min-max scaling on the data then extracting 49 principal components before proceeding onto hyperparameter tuning. The two parameters to be adjusted were α. which is a regularization term which penalizes how large the weights can be; and hidden_layer_sizes which controls how many layers and how many hidden units per layer there are in the model. Figure 10 shows the training and testing accuracy (top) and F1 scores(bottom) for various α (with hidden_layer_sizes=(94,)) and hidden_layer_sizes (with $\alpha = 0.01$). It is clear that there are no "sweet spots" where the gap between the training and testing scores is minimized. Thus, we seek to maximize the testing score in this case. We expect the accuracy to improve with more hidden layers, but it is unnecessary to make a model that complex when our scores are good as shown in table 8. Additionally, looking at the ROC curve in figure 11, we can see the model does well, with the most commonly false positives of classes 6 (Pablo Picasso) and 8 (others).

	Training	Testing
Accuracy	0.982	0.950
F1	0.980	0.945
ROC AUC	1.0	0.997

Table 8: MLP scores with $\alpha = 0.01$ and hidden_layer_sizes=(94,)

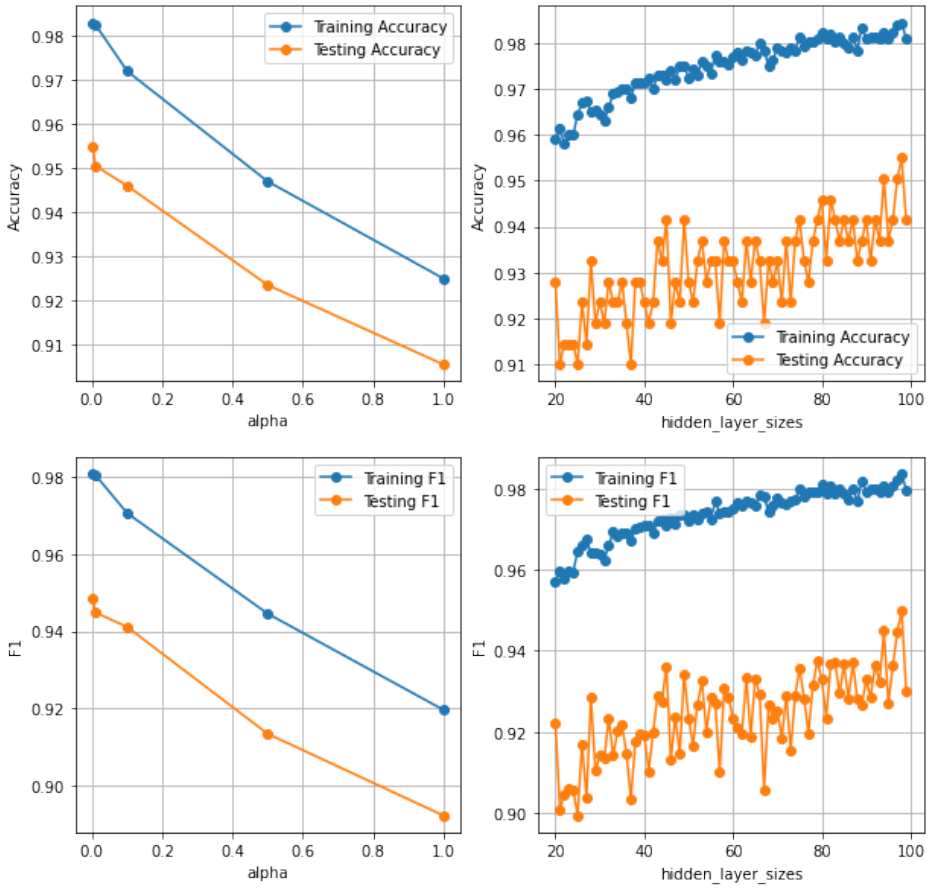

Figure 10: Top row: train/test accuracy for different MLP parameters, Bottom row: train/test F1 score for different MLP parameters

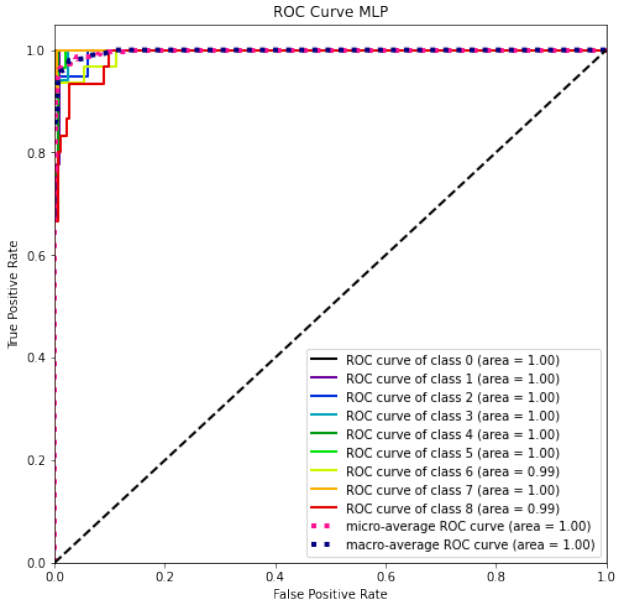

Figure 11: ROC curve for MLP with best parameters

3.4 Random Forest (RF)

Random forests are an ensemble method which constructs multiple trees based on subsets of the given data that each "vote" on the classification output. Each tree may overfit on part of the data, but all together they balance one another. The trees are built randomly through a bootstrapping process that randomly selects observations with replacement to build the trees. Using 4-fold cross validation with the grid search we chose the number of trees to build (n_estimators) and how big the tree could be (max_depth), and the minimum number of samples in each leaf (min_samples_leaf) as the three hyperparameters to fine tune.

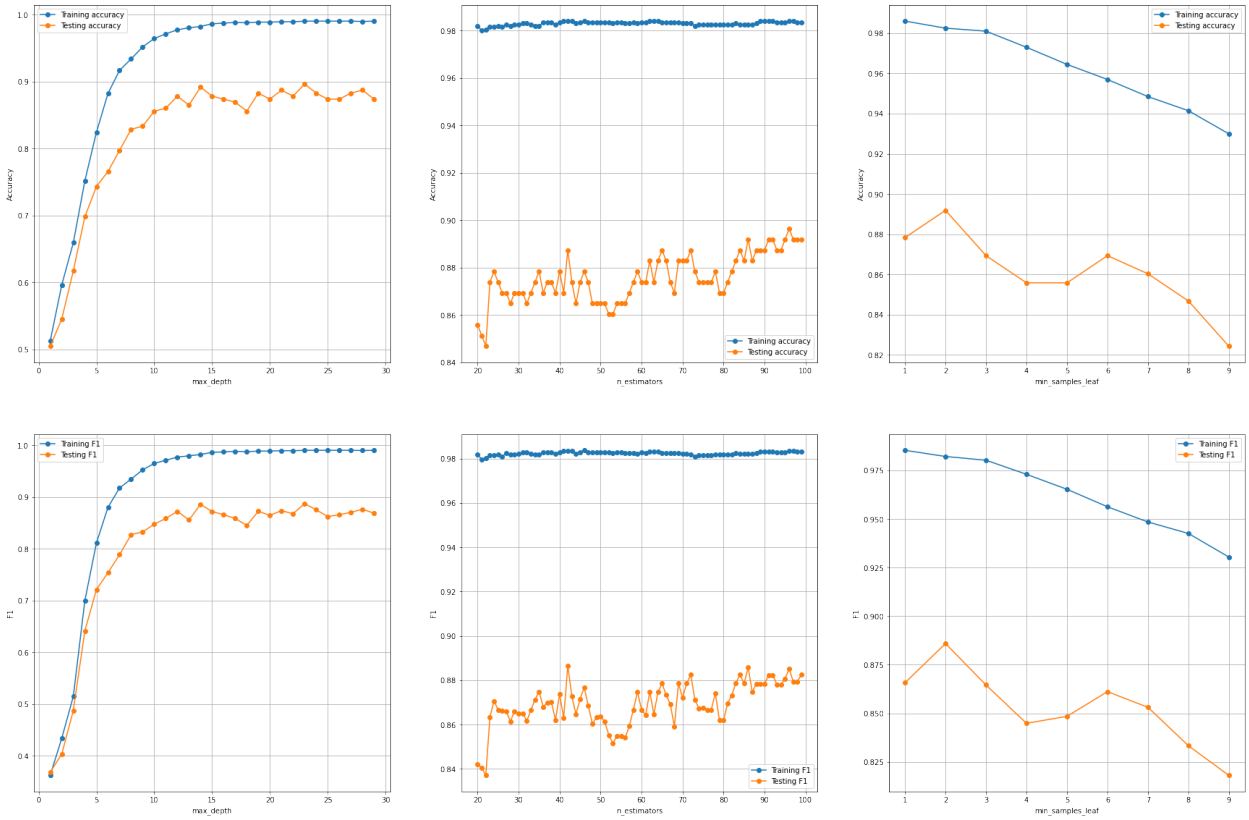

Figure 12: Top row: train/test accuracy for different RF parameters, Bottom row: train/test F1 score for different RF parameters

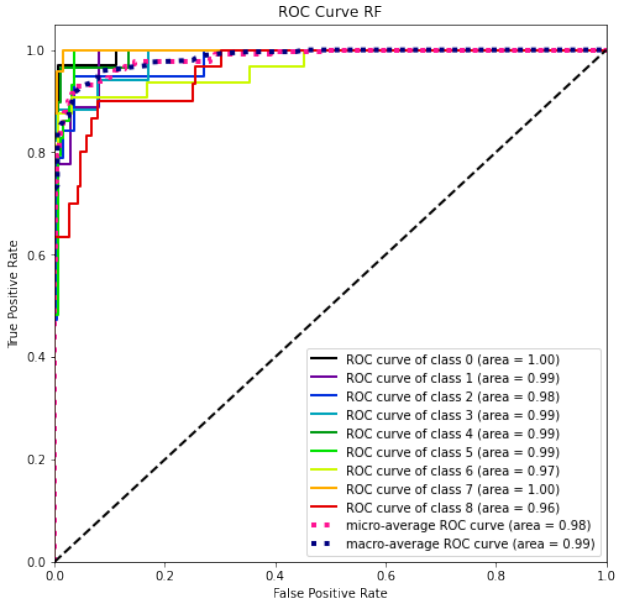

Figure 13: ROC curve for RF with best parameters

The random forest classifier is resistant to scaling, so we did not scale the data for this algorithm, but we did select the best 50 features to use before hyperparameter tuning. In section 2 we noted that random forest tended to overfit the data, so we hoped to mitigate that by reducing the complexity of the model by reducing the number of trees from the default 100, making the individual trees smaller (default max_depth is none), and raising the number of samples in each leaf up from 1. Figure 12 compares the training and testing accuracy (top) and training and testing F1 scores (bottom) for varying values in the three parameters to tune. A quick glance shows us that closing the gap between training and testing scores while still doing a good job classifying would be difficult. The easiest hyperparameter to fix was min_samples_leaf = 2 where there is a clear peak in the testing scores. Though, not as drastic, we can see a peak at max_depth=14. When finding the best number of trees to grow, we decided to prioritize accuracy over the F1 score, since the F1 score stayed around 0.88 regardless of the number of trees. Thus, the best n_estimators=(86,). The ROC curve (figure 13) shows us that the RF classifier most frequently incorrectly classified paintings as class 8 (others). Table 9 summarizes all of the scores for our best RF model.

	Training	Testing
Accuracy	0.982	0.892
F1	0.982	0.886
ROC AUC	1.0	0.985

Table 9: RF scores with max_depth= 14, min_samples_leaf=2, n_estimators=86

3.5 Comparisons

To summarize our four classifiers, we can compare their confusion matrices. Figure 14 shows the confusion matrices for KNN, SVC, MLP, and RF with no scaling, feature selection or hyperparameter tuning. Figure 15 shows the confusion matrices for the four classifiers each with their best scaling and feature selection procedure and best hyperparameter settings. We can see that the improvements with scaling, dimensionality reduction, and hyperparameter tuning were slight since all classifiers did relatively well with their default settings and given all of the data. In particular, KNN had the same accuracy before and after tuning, but the paintings they classify correctly change. SVC improves over all classes except class 2, MLP improves over all classes, and RF improves over all classes except class 3.

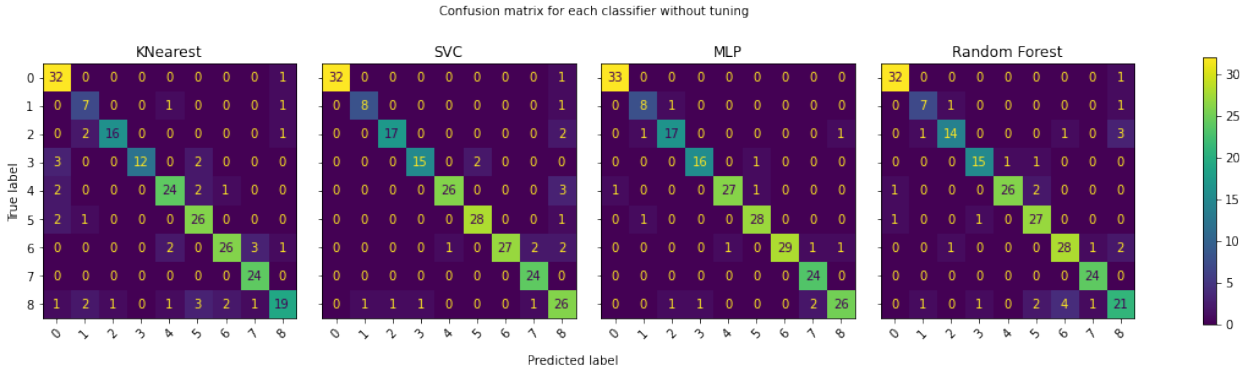

Figure 14: Confusion matrix for KNN, SVC, MLP, and RF with no tuning

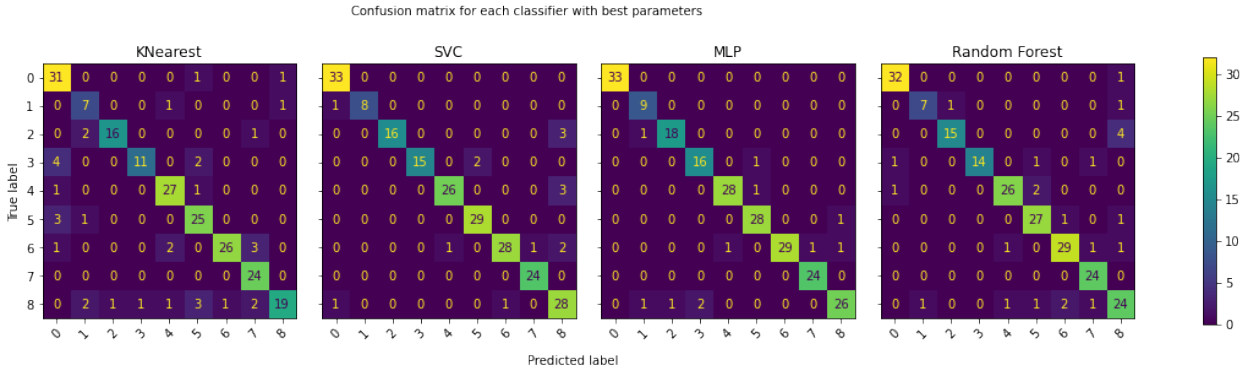

Figure 15: Confusion matrix for KNN, SVC, MLP, and RF with best parameters

We have also and summarized all classifier scores in table 10. We can see that KNN performs the worst, and MLP performs the best at every measure.

	KNN		SVC		MLP		RF	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing
Accuracy	0.888	0.838	0.953	0.932	0.982	0.950	0.982	0.892
F1	0.883	0.822	0.953	0.934	0.980	0.945	0.982	0.886
ROC AUC	0.994	0.976	0.999	0.996	1.0	0.997	1.0	0.985

Table 10: All classifier scores after tuning

Since our data is features of paintings we are able to actually view the misclassified samples. Figures 16, 17, 18, and 19 give the first 5 misclassified samples by KNN, SVC, MLP, and RF, respectively. We can see that there were some paintings that were difficult for all four classifiers to correctly classify. Figure 20 shows all 6 samples that were misclassified by every classifier. Half of the paintings in figure 20 are by Pablo Picasso, so we might deduce that he had a drastic change in style at some point in his career, and our classifiers would not be able to determine this. If we wanted to fix this we may want to do some research, and then separate his paintings based on the time at which his style changed and categorize Picasso as "early" and "late" Picasso.

Figure 16: First 5 misclassified samples by KNN

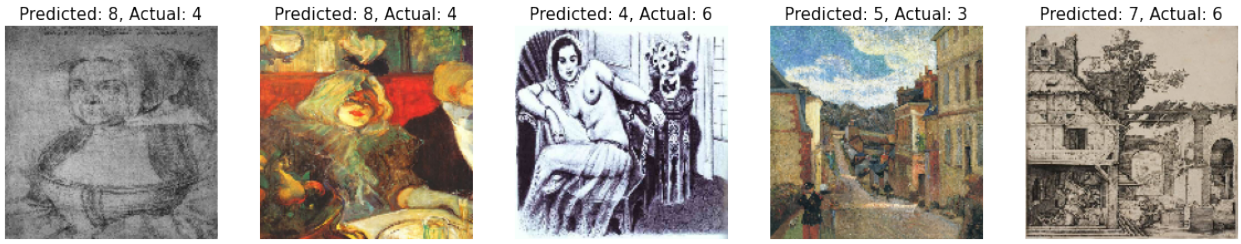

Figure 17: First 5 misclassified samples by SVC

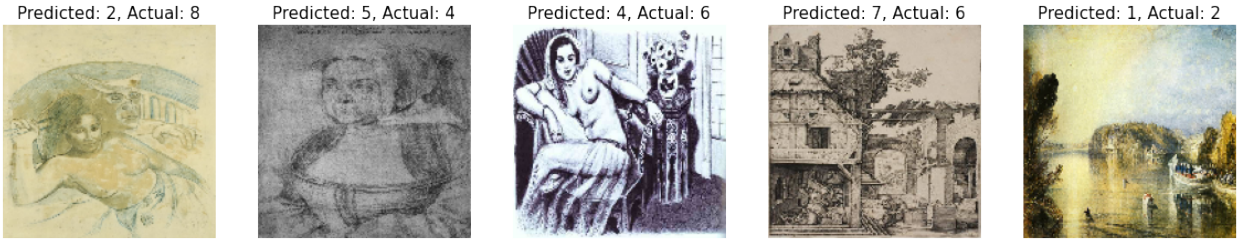

Figure 18: First 5 misclassified samples by MLP

Figure 19: First 5 misclassified samples by RF

Figure 20: All misclassified samples by all classifiers

4 Ensemble Methods

Ensemble methods for classification problems combine multiple classifiers to make a meta-classifier. Combining multiple classifiers could make a better classifier since we would hope that the majority of classifiers would correctly classify an observation. There are multiple ways to create an ensemble, and we will be looking at two different methods: bagging and adaboost.

4.1 Bagging

A simple ensemble method uses majority voting, where all of the classifiers in the ensemble classify an observation and the label with the most votes is used. Bagging is related to the majority vote method, but instead of training every classifier on the training data, each classifier is trained on a bootstrap sample (random sample with replacement from the training set) so that each classifier is slightly different. This is important if our ensemble is an ensemble of the same type of classifier. Otherwise we would simply be training the same classifier repeatedly with no variety. Then the ensemble of different base learners use majority voting to classify a sample. Random forest is an ensemble method of this type with decision trees as the base classifier, which we already explored in section 3.4. We decided not to use it as a base learner in bagging since we could simply grow more trees in our random forest.

The bagging classifier has a few parameters that we adjusted. The first being the base classifier, the second was the maximum number of samples to draw (with replacement), and the number of estimators to train. We compared our best KNN, SVC, and MLP classifiers to a bagged classifier using one of the best classifiers as the base classifier. We found that bagging did not overall improve any of our best classifiers and tended to overfit it, despite tuning the hyperparameters. Figures 21, 22, and 18 show the confusion matricies for the tuned, untuned and best bagged classifiers for KNN, SVM, and MLP, respectively. We can see that for the SVM and MLP classifiers bagging did not make any improvements over the tuned base classifiers.

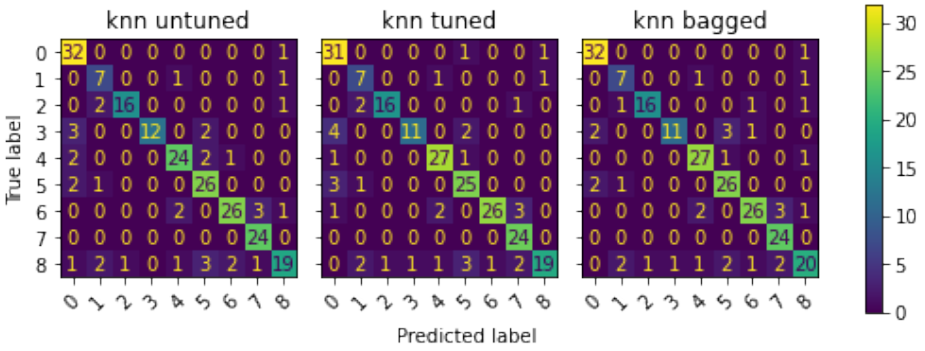

Figure 21: Confusion matrix for KNN untuned, tuned and bagged

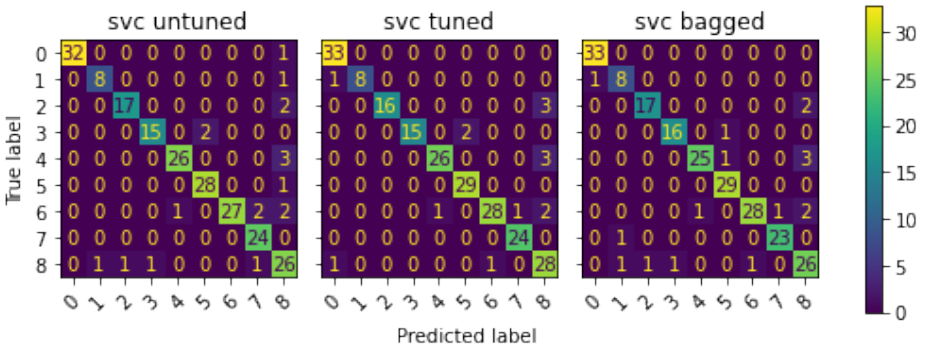

Figure 22: Confusion matrix for SVC untuned, tuned and bagged

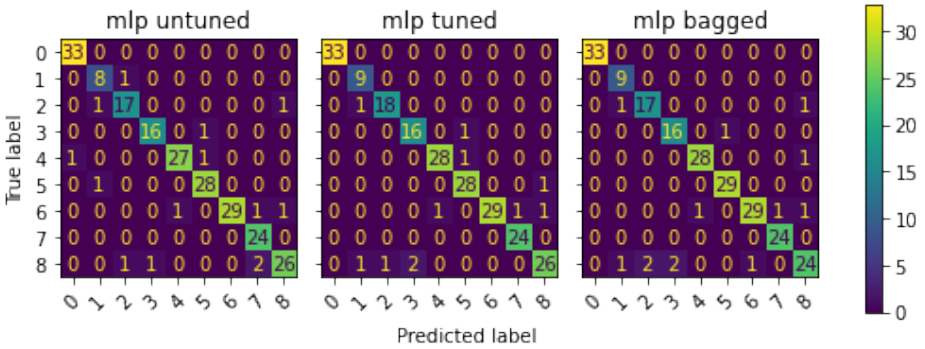

Figure 23: Confusion matrix for MLP untuned, tuned and bagged

Aside from our KNN classifier, the SVM and MLP classifiers were both strong learners. Since KNN was our weakest learner, and our tuned model performed similarly without tuning, we reran the bagging algorithm with KNN and its default paramaters, and found it was the best version of KNN we had so far based on testing scores. Table 12 shows the comparison between the untuned, tuned, and bagged KNN algorithms. However, bagging still tended to overfit to the training data. Figure 24 shows the first five misclassified samples by the bagging classifier with KNN as the base learner, and we can see that they are the same as in figure 16: the first five misclassified samples by our best version of KNN. Furthermore, both the KNN and bagged KNN classifiers made the same predictions on those samples. Overall, the results from bagging were not worth the computational complexity, and we generally had better results with our best base classifiers.

	KNN Training	Untuned Testing	KNN Training	Tuned Testing	KNN Training	Bagged Testing
Accuracy	0.888	0.838	0.911	0.838	0.927	0.851
F1	0.883	0.822	0.905	0.826	0.924	0.835
ROC AUC	0.994	0.976	0.996	0.970	0.998	0.981

Table 11: Comparison of KNN untuned, tuned and bagged

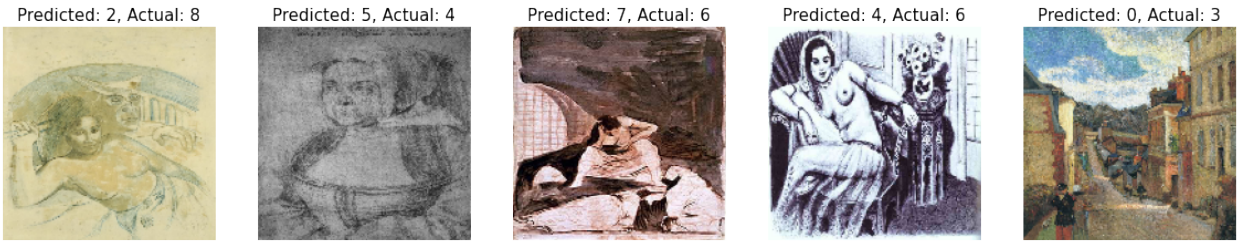

Figure 24: Bagging misclassifications with KNN as base classifier

4.2 Adaboost

Unlike bagging, adaboost uses the entire training set to train weak learners through an iterative procedure. The algorithm trains the model, then uses weights to weight the examples it misclassified, then retrains the classifier. We used adaboost with three different base classifiers: decision tree, SVM, and random forest.

Since decision tree was not one of our base learners that we tuned in sections 2 and 3, we did that here before using the tree with adaboost. The best scaling procedure was none, with 28 principal components, and min_samples_leaf=3 gave the best scores given in table 12. We used this tree as our base classifier in the adaboost algorithm. We tuned two parameters we tuned in adaboost: n_estimators and learning_rate. The first parameter sets maximum number of estimators at which boosting is terminated, and the second shrinks the contribution of each classifier. Through a gridsearch with 4-fold cross-validation, we found the best n_estimators=30 and learning_rate=1.0. We can compare the scores of tree and adaboost in table 12 and there is clearly a signifigant improvement over the base learner. The confusion matricies for the tree and adaboosted tree in figure 25 show a dramatic improvement, particularly in classes 5, 6, and 8 where class 8 more than doubles in it's correctly classified samples.

	Tree		Adaboost	
	Training	Testing	Training	Testing
Accuracy	0.894	0.586	0.992	0.887
F1	0.887	0.575	0.992	0.881
ROC AUC	0.996	0.812	1.0	0.980

Table 12: Comparison of Decision Tree and Adaboost with Tree

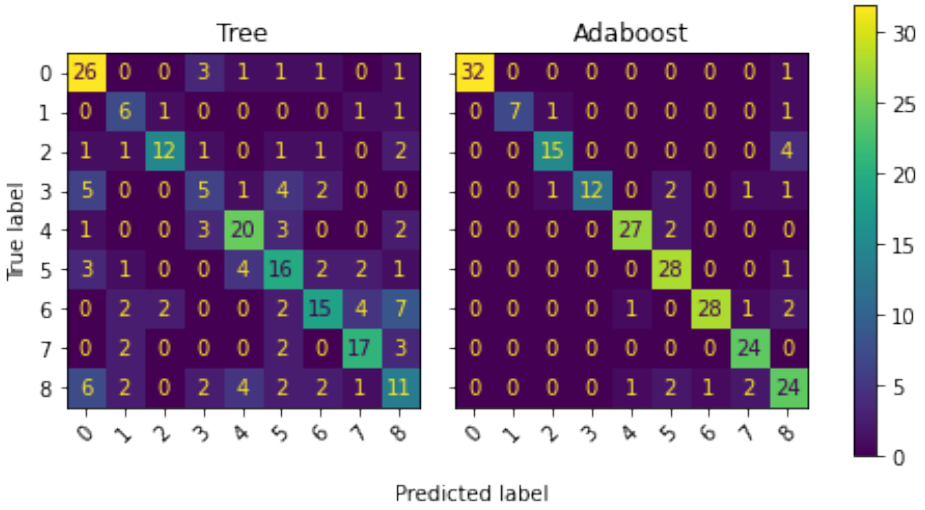

Figure 25: Decision tree and adaboost with tree confusion matricies

Like bagging, adaboost did not improve SVM, even after tuning the adaboost hyperparameters. In a gridsearach with 4-fold cross-validation we found that n_estimators=5 and learning_rate=1.0 gave the best results, but as we can see in table 13, the adaboosted model was underfit and had signifigantly lower scores than our best SVM model. In figure 26 we can compare the confusion matricies and see that adaboost does not classifiy better in any category.

	SVC		Adaboost	
	Training	Testing	Training	Testing
Accuracy	0.952	0.937	0.843	0.842
F1	0.952	0.933	0.848	0.845
ROC AUC	0.999	0.996	0.986	0.977

Table 13: Comparison of SVC and Adaboost with SVC

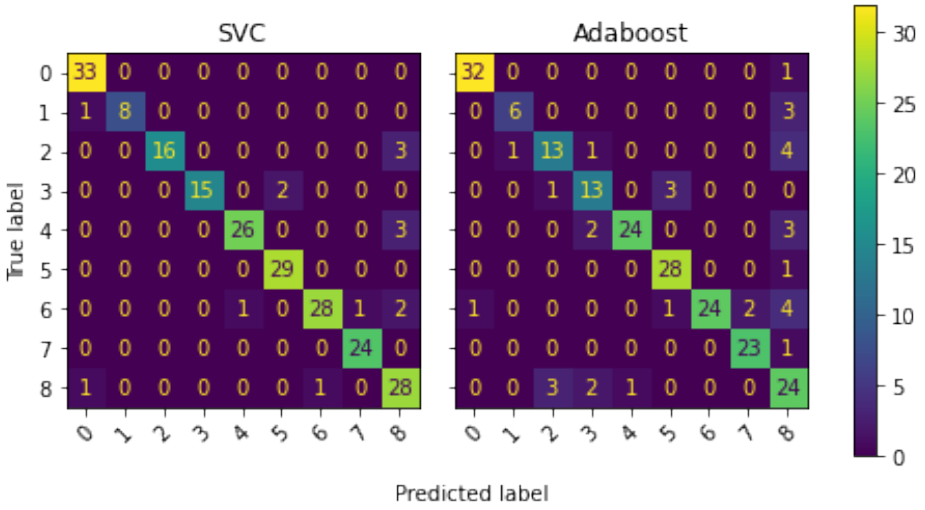

Figure 26: SVC and Adaboost with SVC confusion matricies

Our last adaboost model was with random forest as the base classifier. Recall that RF is already an ensemble of bagged trees, so in this case we are using two ensemble methods with tree as the actual base learner. The best parameters were learning_rate= 0.5 and n_estimators=4. This produced the best adaboost results with the scores shown in table 14. The accuracy was slightly improved but the F1 and ROC AUC scores were lowered. The confusion matricies in figure 27 show that the adaboosted method only adds one correctly classified sample to class 2 and 4, but misses one sample in class 6 that RF correctly classified. So in this case, we would not choose the more complicated adaboost model over the simpler RF for a slight improvement in accuracy. Figure 28 shows the first 5 misclassified samples by this adaboost model. When we compare this with figure 19 (the first five misclassified samples by RF) we can see that they made some different mistakes, and two of the three samples they both misclassified the adaboost and RF models made different predictions.

	RF		Adaboost	
	Training	Testing	Training	Testing
Accuracy	0.982	0.892	0.990	0.896
F1	0.982	0.886	0.990	0.885
ROC AUC	1.0	0.985	1.0	0.983

Table 14: Comparison of RF and Adaboost with RF

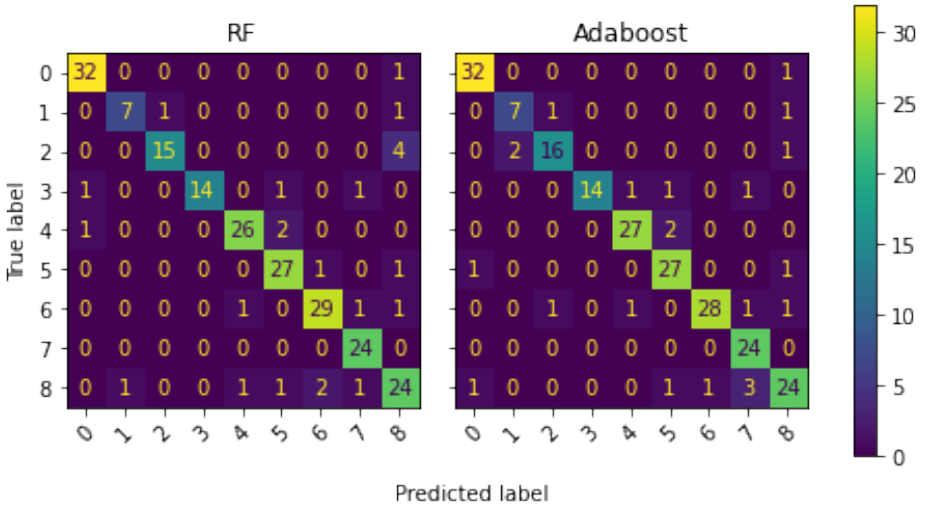

Figure 27: SVC and Adaboost with SVC confusion matricies

Figure 28: Adaboost misclassifications

4.3 Comparisons

We saw that with very weak classifiers like decision trees the ensemble methods are extremely effective in raising scores. However, with strong learners both bagging and adaboost tended to drop scores and overfit to the training data. In the one case where adaboost slightly raised the accuracy for the random forest, using the ensemble was not worth the extra computational complexity. Similarly, in the one case where KNN was improved by bagging, it was not enough to justify the extra computation time. Overall, the ensemble methods we used were only really effective in the case of very weak learners like the decision tree. In section 3.1 we had the decision tree which was a poor classifier for this data set, but we saw that bagging (random forest in the intermediate project) drastically improved its performance, as did adaboost for the decision tree.

5 Conclusions

In our intermediate report we ran multiple experiments trying to find the best KNN, SVM, MLP, and RF classifiers for this data set. We found that MLP performed the best by every measure. Then using two ensemble methods: bagging and adaboost, no matter what base classifier we used we could not outperform our best MLP classifier. Table 15 summarizes our

best bagged, adaboosted, and mlp classifiers. The bagged MLP classifier was competitive with the tuned MLP classifier, but was clearly more overfit. The adaboost classifier with random forest performed the worst and was very overfit compared to bagging or MLP as we can see by the large gap in the testing and training scores. It is clear that the tuned MLP classifier was our best classifier across all of our experiments by all three measures we used.

	Bagging (MLP)		Adaboost (RF)		MLP	
	Training	Testing	Training	Testing	Training	Testing
Accuracy	0.981	0.941	0.982	0.892	0.982	0.950
F1	0.979	0.935	0.982	0.886	0.980	0.945
ROC AUC	1.0	0.997	1.0	0.985	1.0	0.997

Table 15: Comparison best bagged, adaboost, and MLP classifiers