Introduction

Problem Formulation

The dataset we are working with is UCI's letter recognition dataset containing 20,000 samples and 16 features. The features are listed in the link above. Each feature in the dataset is processed from the original images to represent important information. The labels are the associated capital letter in the image. I have a few questions I want to answer based on the problem formulation…

1. Which model performs the best in terms of test accuracy, runtime, interpretability?
2. Which set of letters are the easiest to distinguish? Which are harder to distinguish?
3. Does dimension reduction improve the models?

Model Assessment

The first goal of this analysis is to determine which of the five models—k-nearest neighbors, a decision tree, a random forest, SVM, or a neural network—best fits the dataset. For each model, the training will be using five-fold validation for 5 different hyperparameter values to determine the optimal value based on minimum validation error. In order to determine which classifier is the best, I will both consider quantitative measures like the runtime and error as well as qualitative measures like model interpretability. For runtime, I will measure the total time taken to find the optimal hyperparameter value, train the model, and test the model on unseen data. I chose this method, as this process will likely represent a data science workflow when using any given model. For error, I will use the test error, as that is the most unbiased method of assessing the accuracy of the model given the dataset at hand.

Binary Classification

In order to make this a binary classification problem, we segmented the dataset into three tasks: distinguishing between H and K, between M and Y, and between D and O. I expect that D and O will be the most difficult to distinguish, as they both have a circular motion between them, where the only difference is the straight line on the left side of the D. On the contrary, I think that H and K and M and Y will be easier to classify, as the lines of the letters seem more different. For instance, M has two vertical lines on the left and right of the letter while Y has only one vertical line in the middle of the letter. As a result, I would expect higher values for M rather than Y for features measuring vertical on pixels or edge on pixels.

Dimension Reduction

In cases where the dataset has large dimensionality, dimension reduction is important to ensure that the model is able to quickly and efficiently train on the dataset. Although the features are already compressed from a many-pixel image to 16 features, it might still be important to reduce the dimensionality down even further. To determine if this is a useful strategy, I will use three

methods of dimensionality reduction on all models. First, I will drop the feature with the lowest variance for all classification datasets. Then, I will drop one of the two features that have high correlations across the three classification problems. Finally, I will run PCA on the remaining 14 features and use the 4 principal components for each classification problem as my reduced datasets. To determine whether the dimension reduction was useful, I will use a similar method in choosing the best model by considering runtime and error between the two types of datasets for the same model.

Results

Model Descriptions

Before displaying the results, here is a description and general advantages/disadvantages for each model.

1: k-Nearest Neighbors

k-Nearest Neighbors (kNN) identifies unlabeled points by aggregating the labels of the k nearest data points. In classification, often the majority class label is used as an aggregation strategy and ties are broken through random selection. The advantages of using this model is that it is relatively easy to interpret, requires no training time, and can model non-linear clusters. These advantages are primarily because kNN makes predictions purely based on the dataset itself and so is a non-parametric model. One disadvantage is that the runtime is linear to the size of the dataset, so the runtime will be especially long for large datasets. Another disadvantage is the curse of dimensionality: as the number dimensions increase, the distance between points increases, making it computationally more difficult to calculate distances between points. Given this

2: Decision Tree

Classification decision trees use features of the data to split samples into different subsets representing similar label classes. Classifying a given sample is as simple as starting at the root condition and recursing downwards until a leaf node is met, which will assign a given label to the sample. The main advantage of this model is that it is very easy to interpret, as decisions can be clearly displayed and labeled, making it great for data scientists needing transparency in their model. One key disadvantage is that decision trees are very sensitive to small changes in the dataset and hence tend to overfit to training data. This is because decision trees—especially high-depth trees—can make large classification decisions based on a small number of samples. Some solutions to this include restricting the number of leaf nodes or the minimum number of samples in a leaf node.

3: Random Forest

Random forests is a form of ensemble learning in which the aggregation of multiple decision trees' predictions are used to make predictions on class labels. This is done using bootstrap aggregating and feature randomness. Bootstrap aggregating (bagging) is the process of doing uniform sampling of the training data to form multiple subsets of data. Then, each dataset is used to learn a single decision tree. This enables feature randomness, where each decision tree learns specific patterns from the dataset. This model decreases the high variability of decision trees by averaging across all predictions to make its final prediction. One advantage of random forest is that it makes use of decision trees without overfitting. Furthermore, like decision trees, it is able

to use data that has not been scaled or normalized, so pre-processing time can be lower. One key disadvantage is its runtime, as tens or hundreds of decision trees must be trained and evaluated during the process, which is computationally expensive.

4: SVM

Support Vector Machines (SVMs) attempt to maximize the margin between binary class labels by maximizing the distance between the closest points to the classification boundary, which are called support vectors. Support vector machines can be thought of as an extension to linear classification, as it takes a linear classifier and applies a nonlinear transformation to the samples that the classification boundary can be nonlinear. The SVM used here is a soft-margin SVM, where a certain number of points can be misclassified based on a tunable hyperparameter C. Like many of the other models above, one advantage of SVM is that it can handle nonlinearity in the dataset. One key feature of SVM is that the C value is a built-in method of regularization in the model, enabling the model to generalize well. One key issue with SVMs is that the runtime for training is especially long.

5: Artificial Neural Network

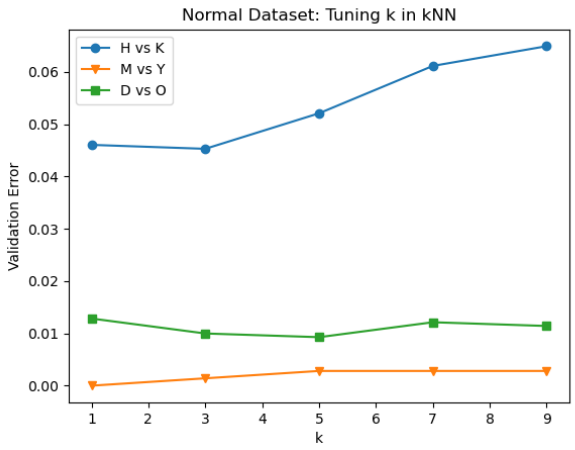
Artificial neural networks are models built from layers of connected nodes. This can be decomposed into the input layer of features inputs, the hidden layer(s) which do more of the process, and the output layers which output a class label. Other than the input layer, all layers get a linear combination of the previous layer's values and apply an activation function to it to restrict the range of the output. The main advantages of artificial neural networks is that they are very flexible and able to approximate nearly any function. This is in part because neural networks can be flexibly designed to have a specific structure to fit a specific use case. Because of this same flexibility, neural networks are also difficult as certain architectures may provide good results while others will not. Furthermore, one key disadvantage is that neural networks require a large amount of data and a large computation time to train, which makes it ineffective as a quick solution like kNN.

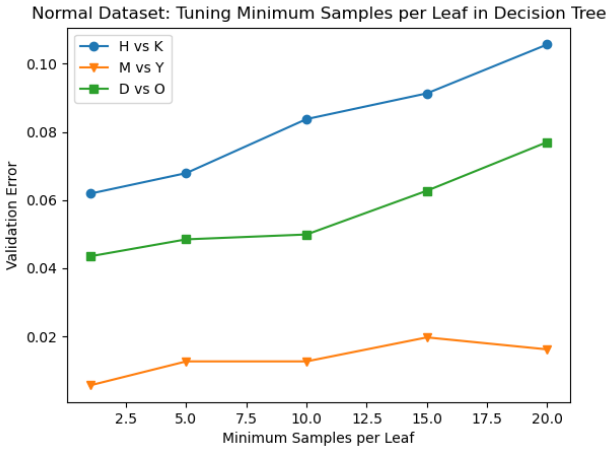
Dimension Reduction Techniques

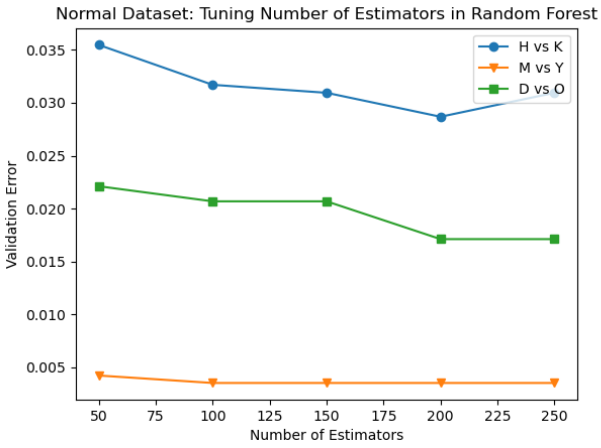
As prefaced in the intro, the three dimension techniques I will be using is removing a low-variance feature, removing a highly-correlated feature, and four-component PCA. Firstly, removing a low variance feature is a simple quality filtering method that requires calculating the variance of all features in the training data (so to make sure that the test set is unbiased). Similarly, removing a highly-correlated feature is an unsupervised filtering method that requires calculating all pairwise correlations between features and removing a feature with high correlation. Finally, PCA creates a covariance matrix from normalized data to determine along which components the great amount of spread can be captured. This results in orthogonal components that capture a certain percentage of the variance of the samples in the dataset.

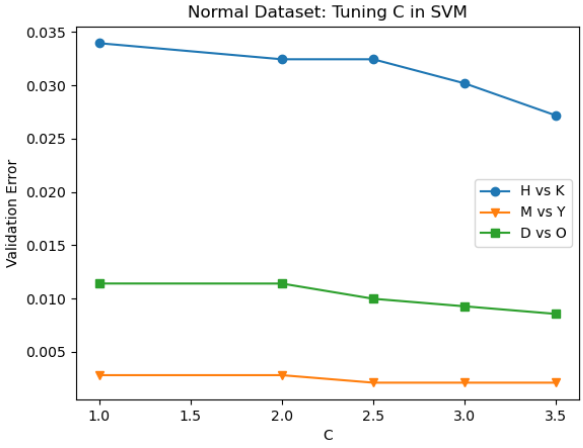
Original Dataset Results

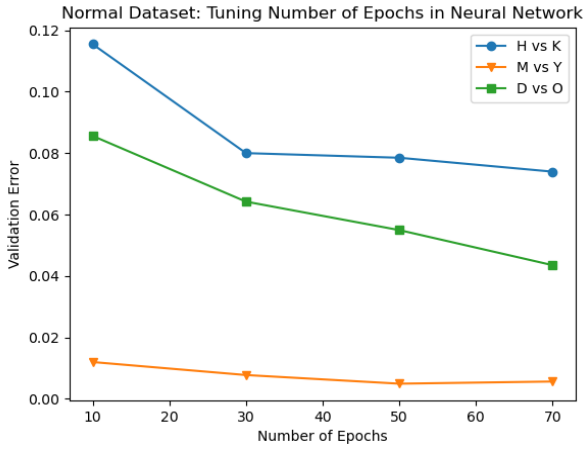
Below is the test error, runtime, and validation error plots for each of the five models tested. Each data point represents the average of the 5-fold cross validation error for each hyperparameter value.





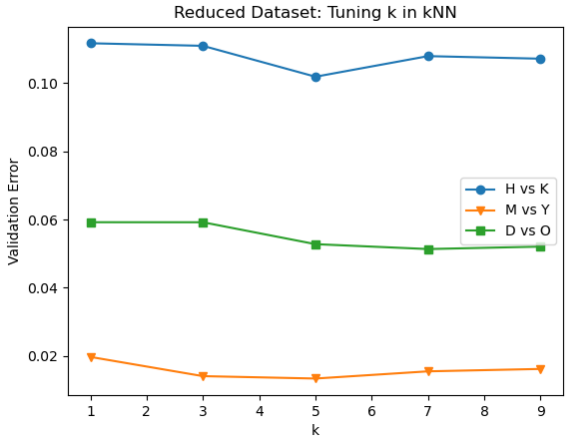


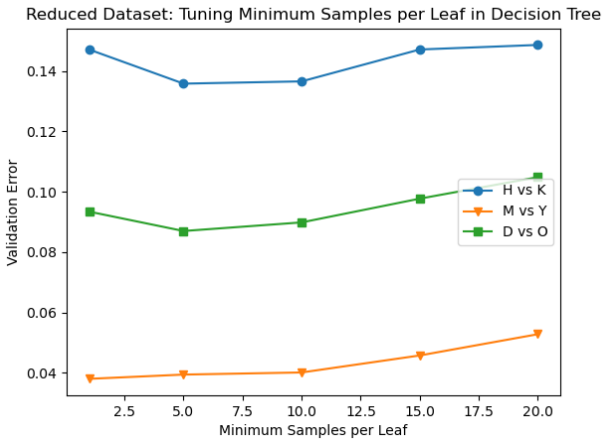


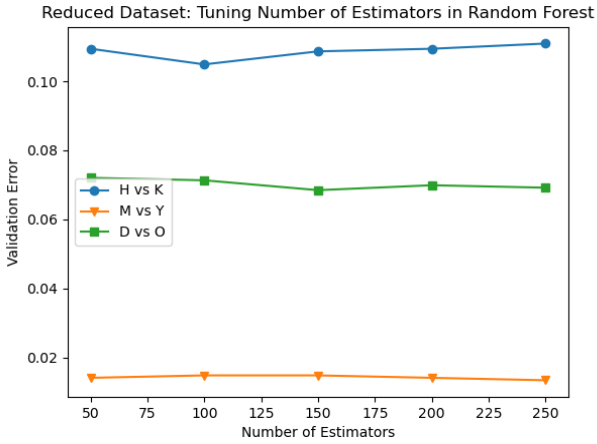


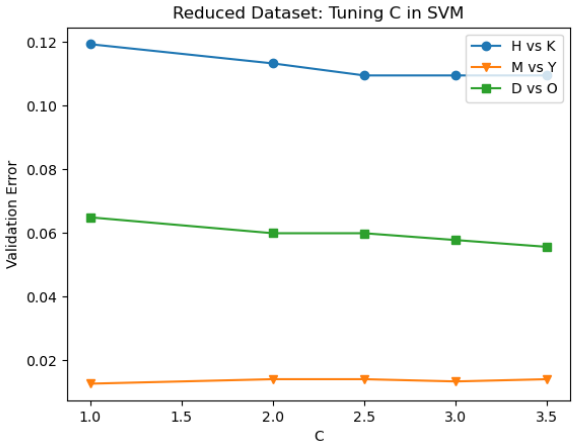
Reduced Dataset Results

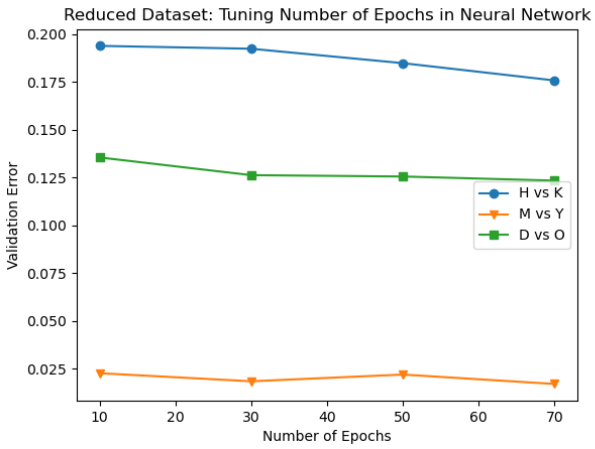
Below is the test error, runtime, and validation error plots for each of the five models tested.











Discussion

Original Dataset Performance

Below are the results from each dataset, where the best quantities (by row) are in bold.

	Dataset	**kNN** (k)	**Decision Tree** (minimum samples per leaf)	**Random Forest** (number of estimators)	**SVM** (C, regulariz-ation constant)	**ANN** (number of epochs)
Hyper-parameter	**H and K**	3	1	200	3.5	70
	M and Y	1	1	100	2.5	50
	D and O	5	1	200	3.5	70
Test Error	**H and K**	0.0473	0.0676	0.0338	**0.0135**	0.0717
	M and Y	**0.0000**	0.0063	**0.0000**	**0.0000**	0.0049
	D and O	**0.0000**	0.0192	0.0064	0.0064	0.0478
Runtime (seconds)	**H and K**	0.6424	**0.2290**	8.6336	0.5736	71.8001
	M and Y	0.5432	**0.2271**	7.1981	0.2751	75.4447
	D and O	0.5116	**0.2408**	7.5554	0.4593	74.3160

Firsting consider the test error for each model, for H and K, SVM performed the best. For M and Y, SVM, random forests, and kNN tied as the best performing model, mostly because this classification seemed relatively easy across all models. D and O also seemed relatively easy to classify for all models, where kNN had the best performance. Because of the small amount of training data, it makes sense that the ANN and decision tree did not perform as well, as ANNs require a large amount of training data and decisions trees tend to overfit to small amounts of data. Despite this, I was surprised that the results were not as bad as expected, all still having over 90% test accuracy for each classification dataset. As a result, if I had to choose a classifier for all classification problems, I would either choose SVM or kNN, as they both consistently had low test errors for all three datasets.

Now considering the runtime, it seems like a decision tree had the fastest runtimes for all models, followed closely by kNN, SVM. Random forests and ANNs were significantly slower, random forests being one order of magnitude larger and ANNs being two orders of magnitude larger than the fastest three models. This makes sense, as random forests are formed from multiple decision

trees, so you would expect that its runtime would also be significantly affected. Similarly, ANN also should be relatively slow, as training a model of this size requires optimization of a large number of parameters. Overall, I would choose an SVM for the H and K problem, kNN for the D and O problem, and either of the two for the M and Y both because of its efficiency in runtime and low test error.

Reduced Dataset Performance

	Dataset	**kNN** (k)	**Decision Tree** (minimum samples per leaf)	**Random Forest** (number of estimators)	**SVM** (C, regulariz-ation constant)	**ANN** (number of epochs)
Hyper-parameter	**H and K**	5	5	100	2.5	70
	M and Y	5	1	250	1.0	70
	D and O	7	5	150	3.5	70
Test Error	**H and K**	0.0946	0.1689	0.0811	**0.0676**	0.1758
	M and Y	**0.0127**	0.0380	**0.0127**	0.0190	0.0169
	D and O	0.0513	0.0769	0.0513	**0.0449**	0.1234
Runtime (seconds)	**H and K**	0.3327	**0.1312**	11.4763	0.5056	87.8982
	M and Y	0.3129	**0.1175**	9.4626	0.1584	95.8763
	D and O	0.2918	**0.1372**	9.9797	0.3442	97.5133

First considering the test error after test reduction, it seems like nearly all of the values increased with dimension reduction compared to the original dataset with up to 6% (0.06) increase in error. Some trends seemed to stay the same, where the ANN seemed to perform the worst and SVM seemed to perform the best. Other trends differed, where the random forest seemed to comparatively perform better than other models for the reduced dataset. This can be better explained by looking more into the dimension reduction techniques used.

In general, filtering methods like removing low variance and highly-correlated values are useful because they are relatively quick and do not require you to train a model. This is both a benefit and detriment, as the small difference in a low variance feature may serve to be very useful in a classification problem. Similarly, two highly correlated features may still be useful if their uncorrelated values pick up something unique between the samples of different labels. This might be the case here, causing the test error for models to increase. PCA falls under the same

boat as the filtering methods, as it does not use the labels and hence does not factor in the effect of the features on the class labels. Furthermore, while running the 4-component PCA, I noticed it only captured around 70-80% of the variance in the samples, which meant that 20-30% of the variance was unable to be used in the reduced dataset. While PCAs do attempt to parse away the useful variance from noise, some of that chunk of missing variance may have been crucial to class label prediction.

Now considering runtime, all models but the random forest and the neural network seemed to be faster. This makes sense, as a lot of the models like kNN are reliant on the size of the dataset, so a smaller dataset should allow the runtime to decrease. However, I was surprised in particular that the random forests took a longer time, especially since the decision trees were faster and bootstrap aggregation should also be faster with a lower-dimensional dataset. Since the different runtime was a relatively small change, this might be due to other factors out of my control like the applications run during the reduced dimension training. Given that it was a smaller network and a smaller dataset, the increase in time for the neural network was also surprising. Other than differing computational power between running the two datasets, this could also be due to using the same hyperparameters without tuning or the different network architecture between the two models.

Outcome

Returning to the initial analysis questions, firstly, the model I would choose to classify each problem for this data would be kNN. The reason why I chose kNN over SVM despite SVM's slightly better test performance and runtime is because kNN has model interpretability. Unlike an SVM that transforms the original dimensions into a latent nonlinear space, the kNN can be well interpreted, as you can describe which of the k images were used to label the data points in question. As a result, this important qualitative factor trumps the small gains in performance and efficiency.

Secondly, I realized that my predictions for the easiest and hardest classification problems were slightly off. As described well by both the validation plots and the test error tables, H and K was consistently the most difficult letter pair to distinguish, followed by D and O then M and Y. While I was right in my predictions for M and Y, D and O seemed to be more distinguishable than expected, possibly due to the straight vertical line in the letter D or it possibly encompassing less space than the letter O. Judging by the result, H and K were likely more difficult to distinguish than expected because of the way that a K can be written similarly to an H. Overall, most of the models' error values had 90% or more accuracy, so overall the classifiers were successful.

Lastly, given that dimension reduction increased the testing error and decreased the runtime, I do not think that dimension reduction is useful for this case. Since each dataset is only a 16-feature

dataset with around 1500 samples, most computers should be able to handle the training of a model for such a dataset. On the contrary, if this dataset were much larger or had a greater number of features, dimension reduction would be a great way to reduce the training time required to build a model. Using extremely low-variance features or features with correlations near 1 or even PCA, removing potentially less useful parts of the dataset could make a much greater difference at a smaller difference in error measures. To better parse apart the effects of each method, it might also be important to remove features using one method at a time to determine which techniques hurt the test error measures the most.

• *(20 pts total) Your code (in a language you choose) including: o (10 pts) The code itself o (10 pts) Brief instructions or documentation on where to find the specific lines that implement each of the classifiers, and each dimension reduction method.*