Programming Assignment 2 Report
Joshua Gabella
CSE 514A – Data Mining

Introduction

Classification of handwritten letters has obvious importance in a world full of written and typed records. Write-to-Text functionality and document scanning to produce editable text documents could result in productivity boosts through modes like PDF editing. Handwriting considered illegible by some can simply be read by a computer and the writing converted to easy-to-read text. There are two important factors to consider in choosing a model. The obvious factor is performance – how well does the model perform on validation sets/test data. We want the model to be as accurate as possible, although in the case of editable documents, a few letters here and there won't hurt as long the words are still interpretable despite the typos. The second factor I would like to see from such a machine learning model is "portability". Datasets can be large, and non-parametric models like K-Nearest Neighbors will require the dataset to be expressly available on the device performing the operation. Given that this functionality could be useful on a small mobile device, like a smartphone, parametric models will serve better. You only need to store the parameters needed to compute the classification, and the data can be stored offsite. Any need to change the parameters can be made with a simple software update.

In the following report, I have tested models and data reduction techniques across three different binary classification problems: classifying "H" vs. "K", "M" vs. "Y", and "D" vs. "O". The first two were handed out as default classification problems, but the third pick was decided upon because anecdotally, it can be difficult to read a person's D vs. their O in an uppercase format.

I implemented several forms of dimension reduction including Principal Components' Analysis (PCA) and Variance Threshold feature selection (VT). Dimension reduction can be useful to store less data (not to be mistaken with fewer points), speed up computation, and to find more useful relationships in the data for models to base decisions off. I encountered few of these benefits in the following report. The dataset contained only 20,000 datapoints of 16 features which is extremely manageable with modern hardware, so neither storage nor model computation is a problem.

Models Used and Descriptions

K-Nearest Neighbors:

A non-parametric model that compares a test data point to all points in the training set. The K "closest" training points (determined by one of several distance metrics) vote on the label of the test point based off of a weighted or uniform average of the labels of the K training points.

Benefits: No training required, only the dataset itself.

Costs: Testing is computationally expensive and requires access to the dataset.

Decision Tree:

A parametric model in the shape of a tree where the leaf nodes are labels. Splits from the root are based off of information metrics in the feature space (ex: entropy).

Benefit: Parametric model. Easy to compute and understand.

Costs: Prone to overfitting with high bias and low variance at greater tree depths.

Random Forest:

A parametric model that aggregates the decisions of separately built decision trees often built with bootstrapped datasets.

Benefits: Similarly easy to understand. Combats overfitting of a regular Decision Tree with low-depth Decision Trees that are low bias and high variance but aggregates them to reduce expected variance.

Cons: Difficult to understand the building process. Expensive to train and tune hyperparameters with large number of estimators.

Support Vector Machine:

A parametric model that seeks to maximize margin the margin of a linear separator from the data. It can take advantage of kernels to transform the feature space so that the data is linearly separable, and can assume a hard margin or soft margin with punishment.

Benefits: Not too difficult to comprehend and is highly adaptable.

Costs: The theory becomes mathematically rigorous to comprehend.

Neural Network:

A parametric model comprised of many linear perceptrons.

Benefits: Not difficult to understand on a surface level. Extremely powerful.

Cons: Requires massive datasets. Training is very expensive and requires several initializations as error optimization is non-convex.

Results

Hyperparameter tuning began with 5-fold cross validation of each model on each problem without any dimensionality reduction. For K-Nearest Neighbors, the hyperparameters I chose to tune were the algorithm used to compute the nearest neighbor (brute force, KDTree, BallTree), the number of neighbors voting on the new label, and the voting scheme (uniform, weighted). Something to notice is that the K-NN model performed well with the M/Y problem and the D/O problem but was noticeably worse with the H/K problem. The interpolated line between plotted hyperparameter points behaves similarly within problems.

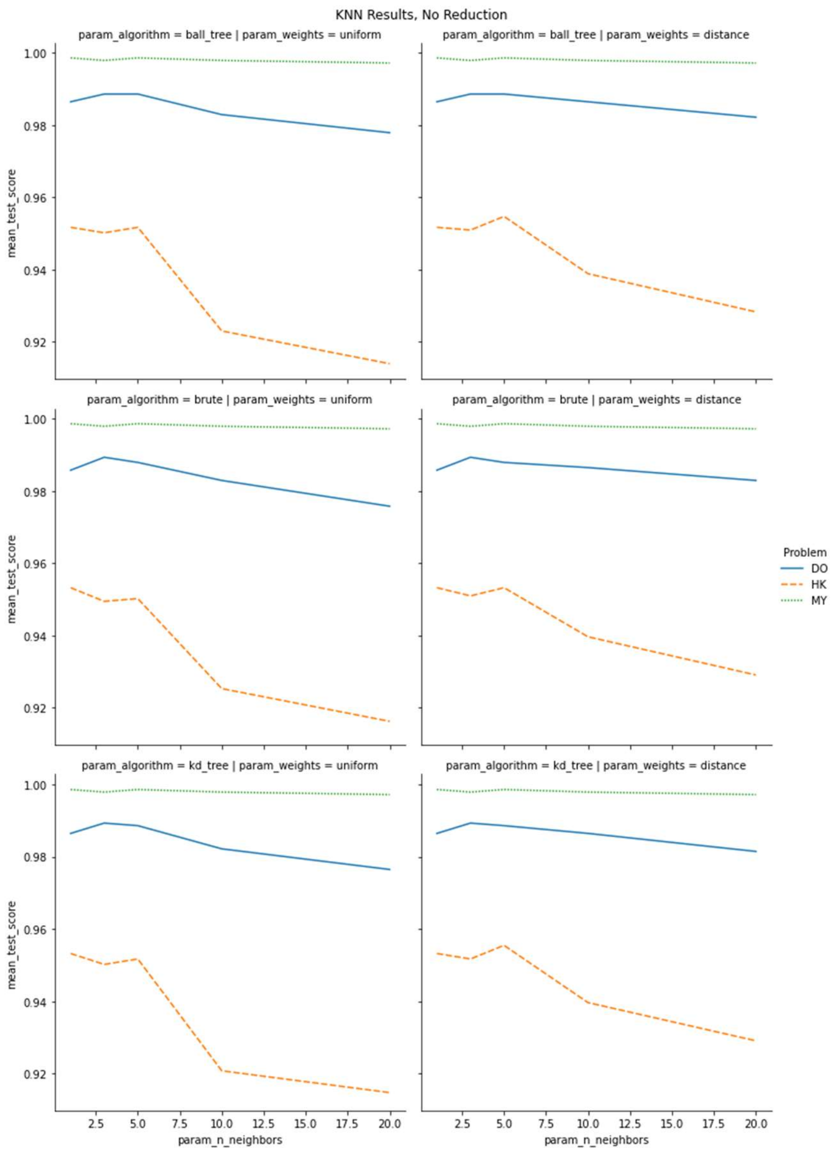


Figure 1. Performance of K-NN with different hyperparameters on different problems

The next graph shows decision tree performance with different maximum depths. This is where we see our first taste of feature selection. Here I show the difference in performance of tree models built with different criterion for splitting on the features, embedded into the models themselves. Notice the structure of the graphs remain very similar despite different embedded feature selection methods. The entropy criterion has a slightly better average performance across problems however. In a real-world environment, I would likely choose a maximum depth close to the plotted joints since continuing to increase max depth just allows room for overfitting, and all trees seem to hit stable performance with fewer than 10 levels in the tree. Again, we see better performance classifying M/Y, then D/O, and H/K has the lowest validation score at all hyperparameter values.

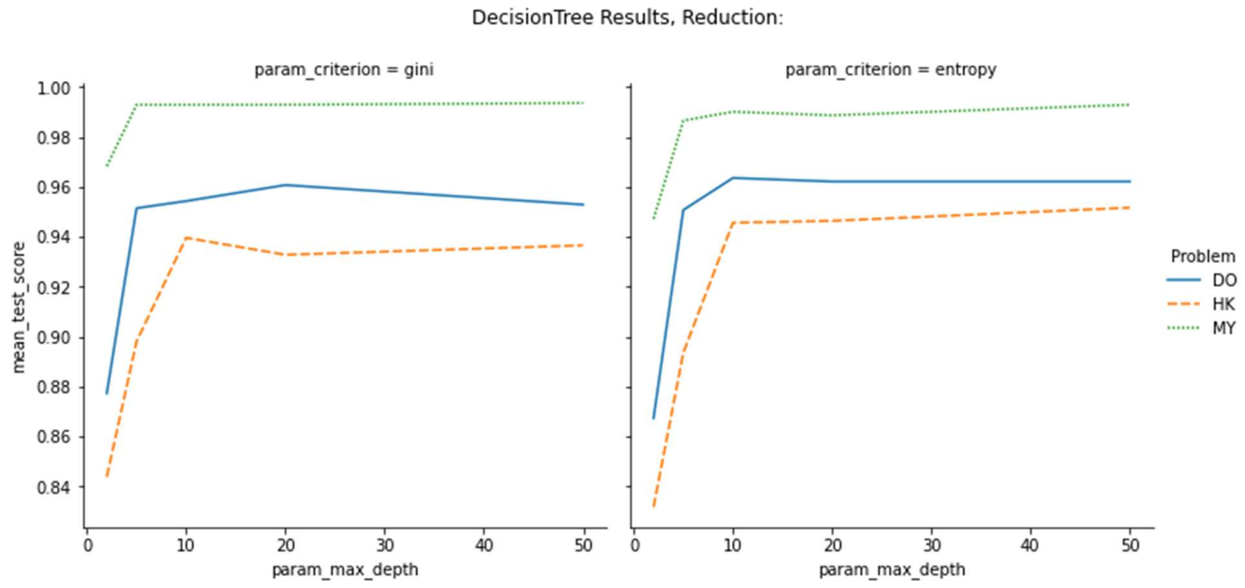


Figure 2. DecisionTree performance with differing maximum depths.

Next we see the Random Forest cross validation scores which show the same problem difficulty hierarchy as K-NN and Decision Tree. The hyperparameters being tuned were the number of estimators (decision trees) that comprise the forest and the split criterion (entropy or gini). Our validation performance is marginally better than the Decision Tree performance, and performance on the H/K problem is improving drastically since the K-NN model cross validation.

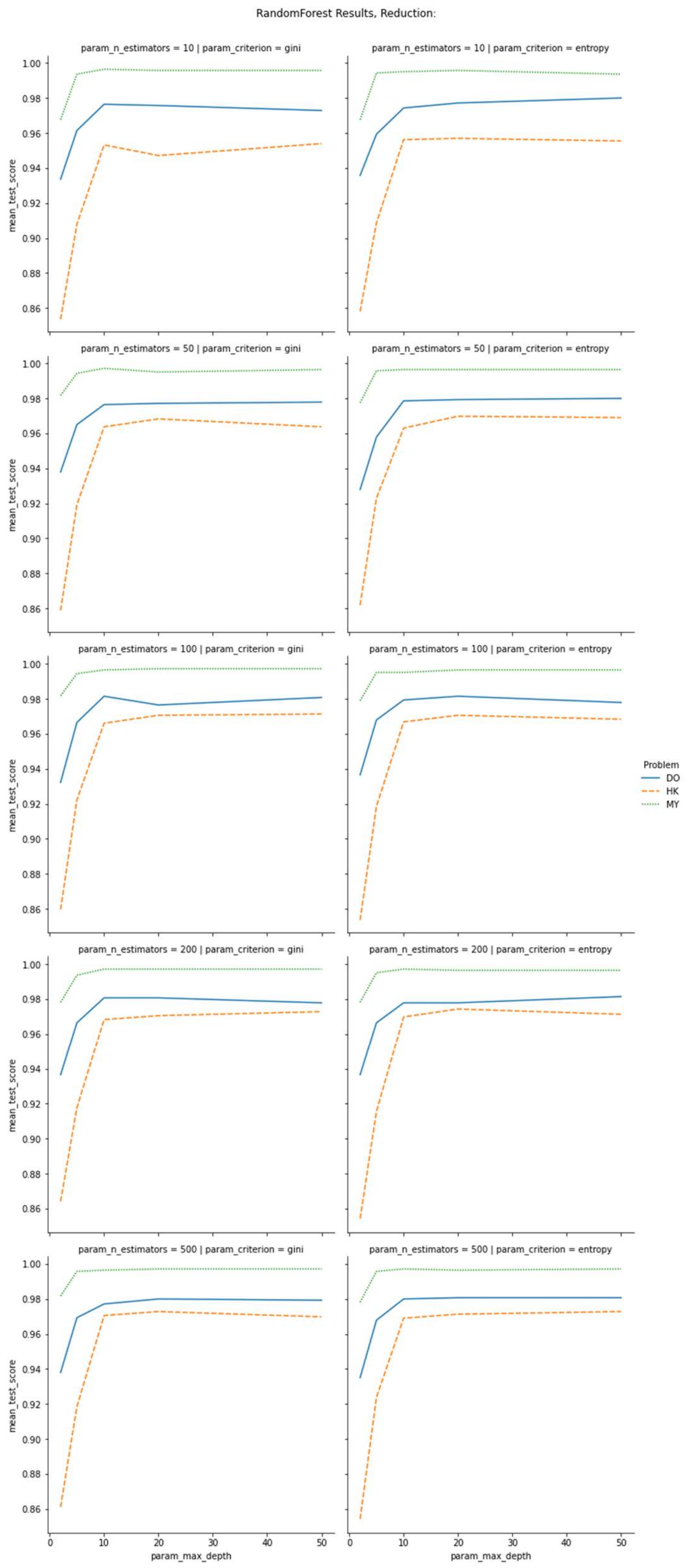


Figure 3. RandomForest hyperparameter tuning via 5-Fold Cross Validation

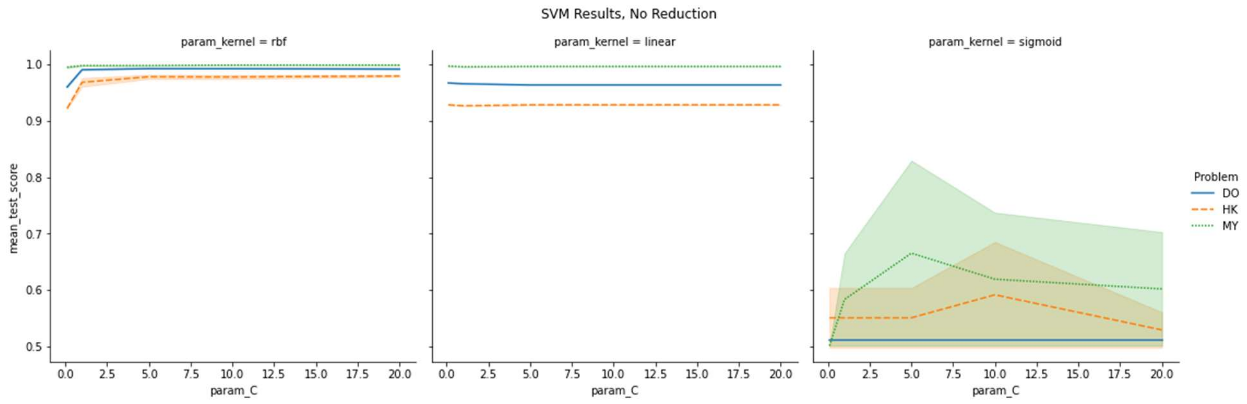


Figure 4. Support Vector Machine Hyperparameter Tuning via 5-Fold Cross Validation

Above we see excellent performance from SVMs built with the rbf kernel at most values of C. Remember that the smaller C value enforces a more strict margin for the linear separator. We see such poor performance with the sigmoid kernel constructed SVM that the confidence intervals around the interpolated data points become very large and visible. However an RBF support vector machine shows the best and most consistent performance so far in our validation sets.

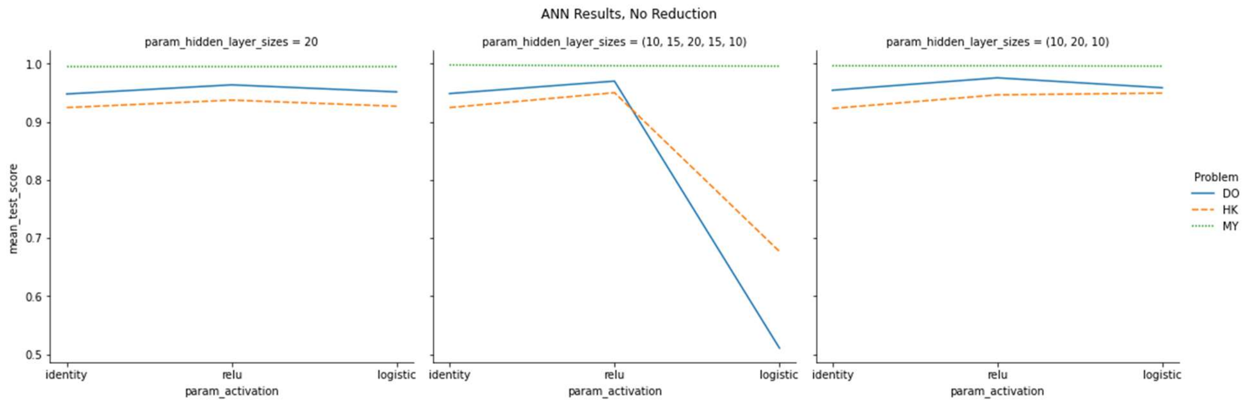


Figure 5. Artificial Neural Network Hyperparameter Tuning via 5-Fold Cross Validation

The artificial neural network was tested with three different structures and three different activation functions. The ANN with five hidden layers saw drastically worse performance with a logistic activation function but remained on par with other models in all other scenarios.

Models on the Data with Reduced Dimensions

The following section shows cross validation performance of models on the same data but with reduced dimensions from one of three techniques.

Principal Components Analysis:

> Projecting data points to a different but real feature space of principal components. One can select up to as many principal components as desired; here I chose four. PCA was used on datasets to train SVM, ANN, DecisionTree, and RandomForest models.

Variance Threshold:

> Excise features whose variance is less than a given threshold. I only kept the four highest variance features. This feature selection method was used on data that trained the SVM and ANN models.

DecisionTree Splitting Method (Gini Impurity vs. Entropy):

> This feature selection is embedded into the tree split structure of both DecisionTree and the aggregate version, RandomForest. The features chosen to split on either maximize the decrease in impurity, or minimize entropy.

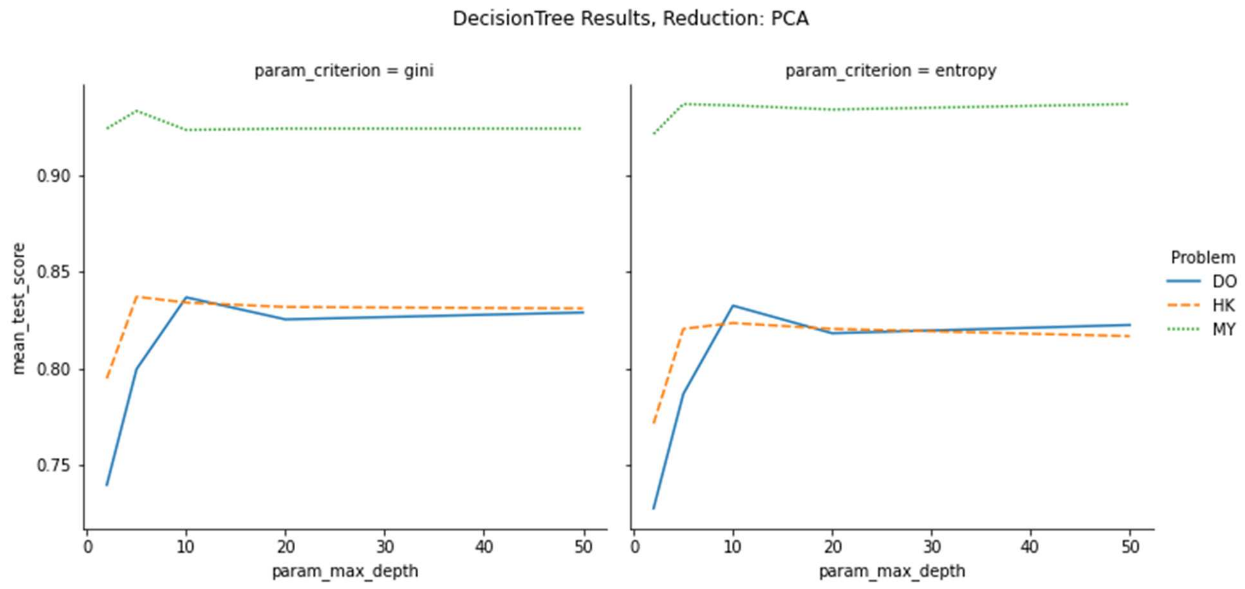
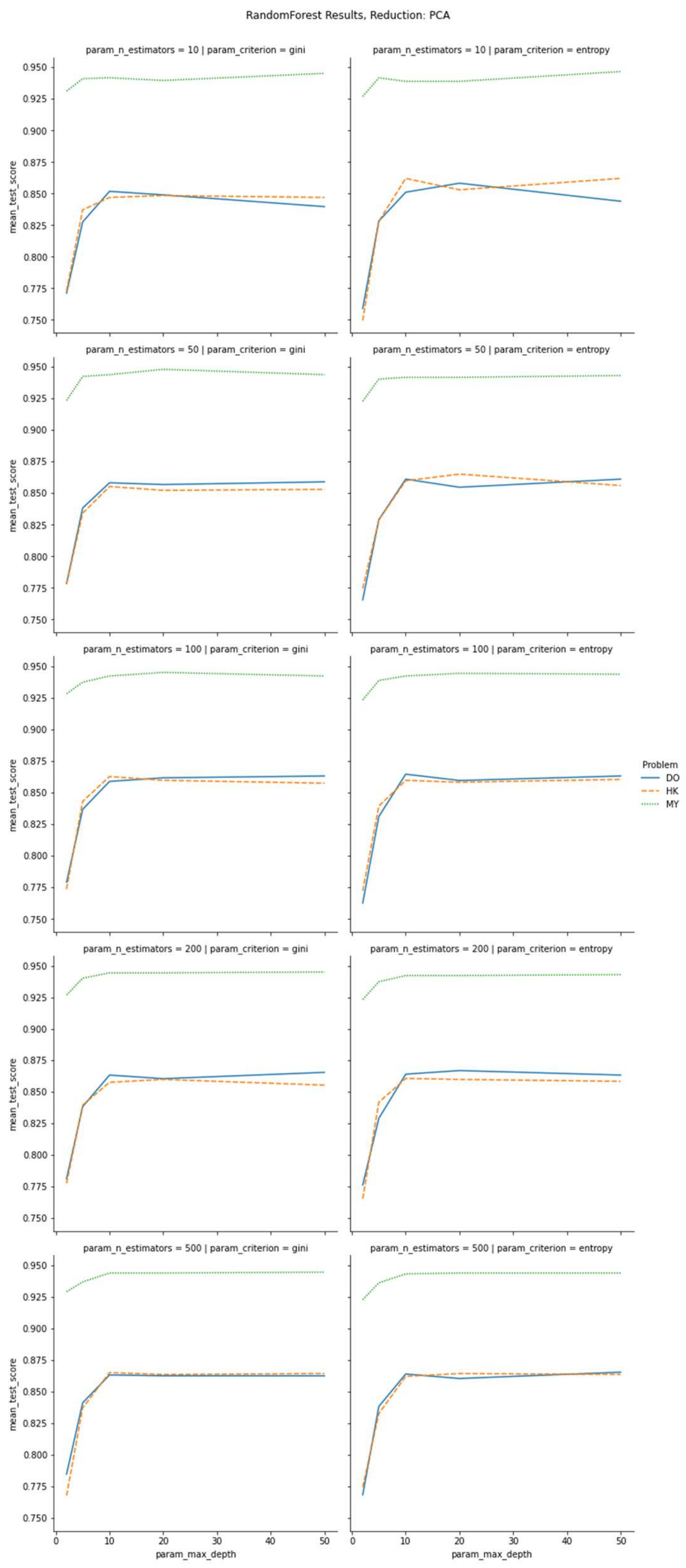


Figure 6. DecisionTree performance with PCA and Gini vs. Entropy criteria as a function of max depth

While DecisionTrees have built in feature selection, I thought it would be interesting to see the effects of reduced dimensionality and feature space transformation provided by PCA on the DT model as well. The model performed significantly worse on the H/K and D/O problems with reduced dimensionality.



RandomForest tells a similar story which is to be expected since the units of this model are the DecisionTree model.

Given a model making decision splits based off of feature information and purity, it seems intuitive that drastically reducing the number of dimensions would increase the risk of a performance hit of the model.

Figure 7. Decision Tree Performance by hyperparameters with PCA

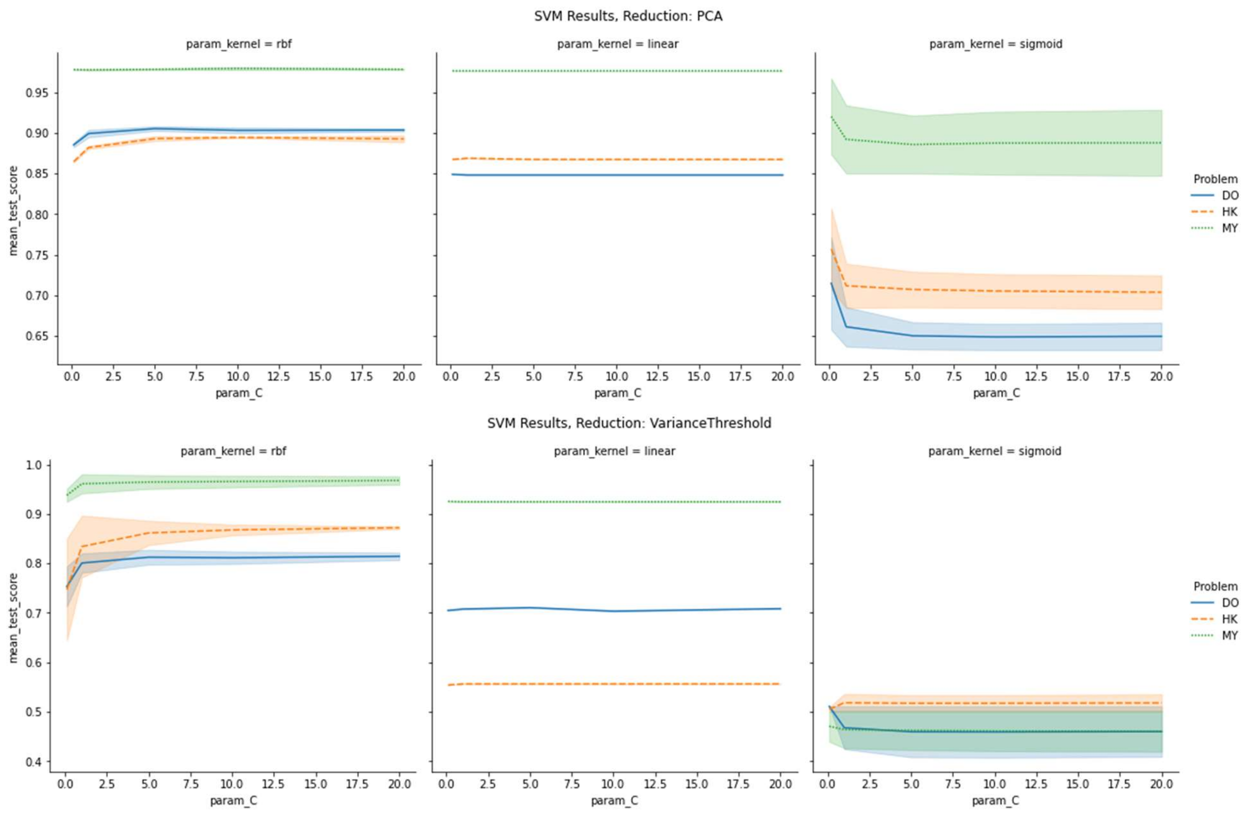


Figure 8. SVM Performance by hyperparameter setting with PCA and VT

I opted to try both PCA and VT and their affects on SVM and ANN largely out of absent-minded curiosity. SVM performs much better after PCA than when the data is reduced with a variance threshold. The overall shapes are similar but there is a much larger spread in the SVM results with VT.

Looking at Figure 9 shows an almost identical pattern when comparing PCA and VT on a neural network's performance. Graph shape is consistent but the spread is drastically increased with VT over PCA.

Lastly, I used variance threshold feature selection when looking at the effects of dimensionality reduction on K-Nearest Neighbors. I wanted to see what simply reducing the feature space instead of transforming it as well would look like on K-Nearest Neighbors. Looking at the difference between Figure 1 and Figure 10, it becomes clear that those last 12 features were important in distinguishing D/O, as not only does general performance drop, but the problem difficulty hierarchy is upset. The model now performs better on the H/K problem than the D/O problem. This is consistent with most models we've seen post-dimensionality reduction where they have performed similarly on the D/O and H/K problems.

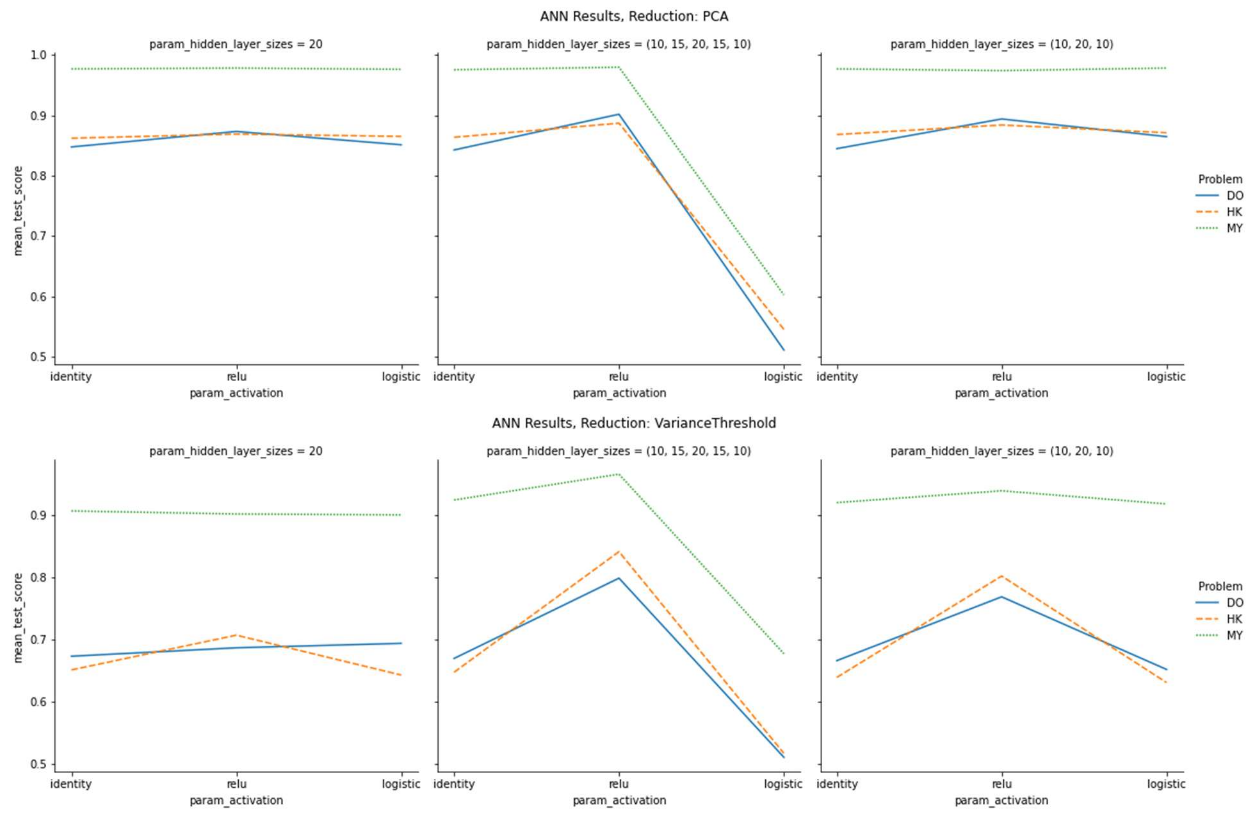


Figure 9. ANN Performance by hyperparameter setting with PCA and VT

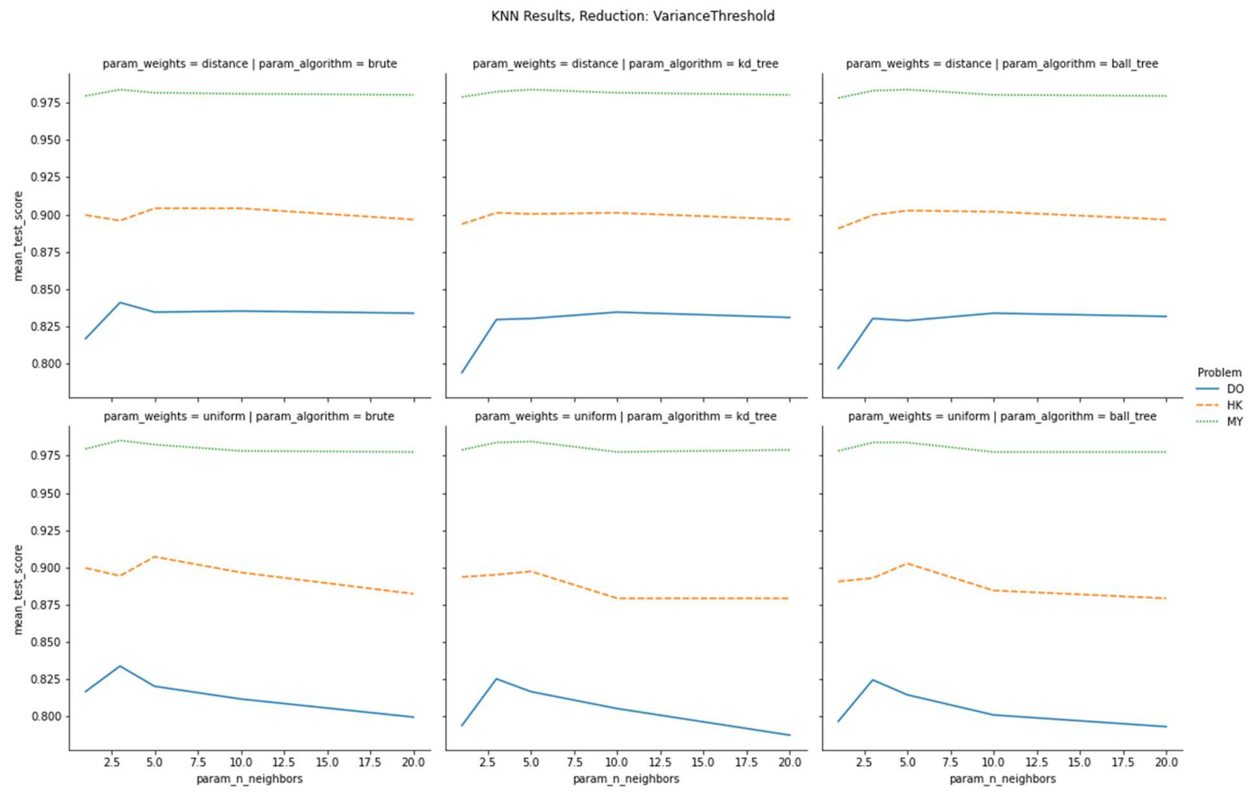


Figure 10. K-NN Performance by hyperparameter setting with VT

Discussion

Data Unreduced	H/K Problem	M/Y Problem	D/O Problem
K-NN	Time: 0.004s Mean Accuracy: 0.946	Time: 0.005s Mean Accuracy: 1.0	Time: 0.005s Mean Accuracy: 0.987
Decision Tree (Entropy)	Time: 0.001s Mean Accuracy: 0.824	Time: 0.0 (?) Mean Accuracy: 0.930	Time: 0.000995s Mean Accuracy: 0.840
Decision Tree (Gini)	Time: 0.001s Mean Accuracy: 0.831	Time: 0.0 (?) Mean Accuracy: 0.975	Time: 0.000995s Mean Accuracy: 0.840
Random Forest (Entropy)	Time: 0.001s Mean Accuracy: 0.837	Time: 0.001 Mean Accuracy: 0.981	Time: 0.001 Mean Accuracy: 0.878
Random Forest (Gini)	Time: 0.001s Mean Accuracy: 0.872	Time: 0.001 Mean Accuracy: 0.987	Time: 0.001 Mean Accuracy: 0.974
SVM	Time: 0.007 Mean Accuracy:0.953	Time: 0.006 Mean Accuracy: 1.0	Time: 0.006 Mean Accuracy: 0.981
Artificial Neural Network	Time: 0.0 Mean Accuracy: 0.953	Time: 0.001 Mean Accuracy: 1.0	Time: 0.0 Mean Accuracy: 0.962

Table 1. Model performance on reserved test set with no dimensionality reduction and ideal parameters picked after 5-fold cross validation.

Reduced Data	H/K Problem	M/Y Problem	D/O Problem
K-NN (VT)	Time: 0.003 Mean Accuracy: 0.91	Time: 0.004 Mean Accuracy: 1.0	Time: 0.003 Mean Accuracy: 0.833
Decision Tree (PCA)	Time: 0.0 Mean Accuracy: 0.851	Time: 0.0 Mean Accuracy: 0.854	Time: 0.0 Mean Accuracy: 0.814
Random Forest (PCA)	Time: 0.001 Mean Accuracy: 0.851	Time: 0.001 Mean Accuracy: 0.841	Time: 0.001 Mean Accuracy: 0.814
SVM (VT)	Time: 0.007 Mean Accuracy: 0.878	Time: 0.005 Mean Accuracy: 0.981	Time: 0.007 Mean Accuracy: 0.788
Artificial Neural Network (VT)	Time: 0.0005 Mean Accuracy: 0.702	Time: 0.0 Mean Accuracy: 0.968	Time: 0.0 Mean Accuracy: 0.635
SVM (PCA)	Time: 0.003 Mean Accuracy: 0.845	Time: 0.003 Mean Accuracy: 0.994	Time: 0.004 Mean Accuracy: 0.885
Artificial Neural Network (PCA)	Time: 0.001 Mean Accuracy: 0.858	Time: 0.0 Mean Accuracy: 0.987	Time: 0.001 Mean Accuracy: 0.845

Table 2. Model performance on reserved test set with dimensionality reduction (listed in parentheses next to the model) and ideal parameters picked after 5-fold cross validation.

Tables 1 and 2 show the performance of trained models with the best validated hyperparameters on a reserved test set. The most standout relation between the two tables is the consistently reduced performance of the models trained on reduced data. This is not a problem that benefits from dimensionality reduction of 16 features to 4. Problems that may benefit from such techniques will likely have datasets with much larger feature sparces and increased sparsity within the data. Image classification, reconstruction, and media-centric problems will often benefit from more heavy-handed feature selection techniques.

Within Table 1, the SVM model is the most attractive. The only other models that perform on par with it are K-Nearest Neighbor and the Neural Network. Based on the problem's use cases described in the introduction, I would prefer a parameterized model over non-parameterized which precludes me from favoring K-NN, and the neural network is needlessly complicated. It should be noted that with the full dataset, splitting on gini impurity performs consistently better than splitting on entropy for both the DecisionTree and RandomForest models.

Glancing at Table 2, it becomes immediately clear that the D/O problem performance suffers greatly with reduced dimensionality for all models. Within this table, the top three models from Table 1 still reign: K-NN (VT), SVM (PCA), and ANN (PCA). It's a more difficult decision to choose between SVM and K-NN here because of the 7% increased performance of K-NN on the H/K problem. They perform otherwise similarly. Without knowledge of the hardware that may be executing the classification task – as in what sorts of devices may be running this code – the responsible selection would still be the SVM (PCA). Non-parameterized models suffer greatly with reduced processing speed, lack of parallel processing capabilities, and limited storage size. While these problems are going away even on small, portable devices in the modern era, they still may exist.

I didn't consider runtime because every model, including K-NN, performed the prediction tasks within thousandths of a second which is not significant enough to take into account. What I did learn from this practice was that dimensionality reduction and feature selection have their use cases, and there are problems where these techniques don't belong. This binary classification problem suffered quite a bit when the data was manipulated with a variance threshold or transformation and reduction to principal components.

Something else I've learned is that with some planning, I could make my code more modular. As it is, the code exists in a Jupyter Notebook which works wonders for formatting and legibility, but it is not written in a way to take advantage of the modularity provided by individual executable cells.