Nikki Xie
CSE 514 Data Mining
Programming assignment 2 report
2022/4/17

Introduction

1) Description of the problem and motivations:

The problem is about predicting 26 alphabet letters with the feature values. The problem set gives us 20000 instances. Each instance contains 16 primitive numeric attributes as the input X, and with those 16 feature values, we will predict response variable letters. We will build five classifiers, KNN, SVM, Decision Tree, Random Forest and ANN. Every classifier has its advantages and disadvantages. For example, KNN is the easiest to implement and we can easily explain its logic to others. While ANN is good at handling complex problems since it can be tweaked in many ways, its logic is usually hidden and it is difficult to explain it to others. The "best" classifier is not deterministic and is problem-based. Above all, there are several factors to consider when determining the best classifier.

First, we need to consider the computational complexity. Some models will take longer to train and require lots of computation power; while others are easy to train and we can compute the results in a short timeframe.

Second, we also need to consider the interpretability of the model. Though some models behave well in giving the correct predictions, their logic is usually complex and is difficult to interpret step by step. With different motivations of building the model, we need to pick the right one.

Thirdly, we also need to consider the use case of the model. If the model will likely be used on future data, we need to consider the extensibility of the model - whether we can easily tweak the model to fit future data or we can partly save the results for further learning(transfer learning). For example, this can be done in ANN but can be difficult for other models.

Lastly and most importantly, the base requirement is that it gives consistent good performance on training and testing data sets. If its performance shrinks significantly on validation data, we can say that the model is likely overfitted on the training data, and thus we can not regard it as a good classifier.

2) An introduction to the binary classification problems. Which pair of letters did you choose for the third problem? Which pair do you predict will be the easiest or hardest to classify?

The binary classification problem is that we are predicting the correct text label given input feature space X, which consists of 16 features. *I chose 'B' and 'I' as the additional pair*. I think the difficulty of predicting a pair is correlated with the input data set it has. If the pair's input vector space has features that are highly correlational. It will be harder to distinguish the classes and may require more data sets to train a model that can do such. Also, I think the number of samples we have is also very important. According to the problem description, each letter has a different number of data inputs. For example, 'U' has the most input of 813. 'Z' and 'H' on the other hand only have 734 input values.

3) An explanation for why dimension reduction is or is not useful for this problem. Include some discussion on which methods are “better,” and what factors should be considered in determining a dimension reduction method as “good” or “bad.”

Dimension reduction *can be implemented for this problem but not necessarily*. We use dimension reduction when we cannot reach a good performance or in practical sense, to store all data with all feature values are too expensive. If we already can have a good performance without dimension reduction, there’s no need to introduce additional models because they take additional time to train.

For this problem, we can see all models perform well in predicting the class label already without dimension reduction. And with dimension reduction, some models even perform worse than on the original dataset. It means that forcing dimension reduction makes us lose some valuable information about the dataset. For example, after we do dimension reduction, KNN’s average accuracy drops from 99% to 96%. It can be explained by that reducing the feature space from 16 to 4 makes us lose some important features. For RF and DT, it even takes longer for them to be trained because I used Ranfom Forest to do forward selection and RF takes long to train.

For the second part in terms of practicality, the dataset only contains 20,000 samples. It doesn’t require that much space to store all of them. But if the dataset is way larger than this, we may need to do dimension reduction. Also, when the dimension is 16, it is difficult to collect this much samples.

Results

1) **KNN**
   a) Brief description of the classifier and its general advantages and disadvantages

      KNN classifies the class label for new data by using the labels of its k nearest neighbors. It is usually used as a classifier.

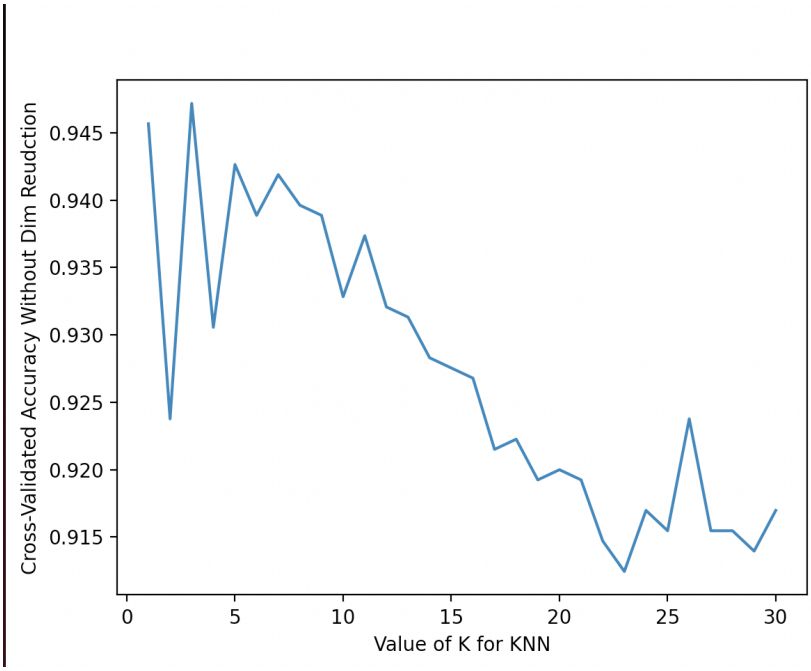
      KNN is easy to implement and usually computes quickly. Also, it's easily interpretable. We can explain the logic to people who are not even in this field. We can also potentially visualize the data and classification if the dimension is low, but not for this problem.

      However, it requires a lot of time and storage space. We need to store all the samples. Also, since we have to calculate the distance for all sample pairs, it can be very inefficient.

   b) Graph:

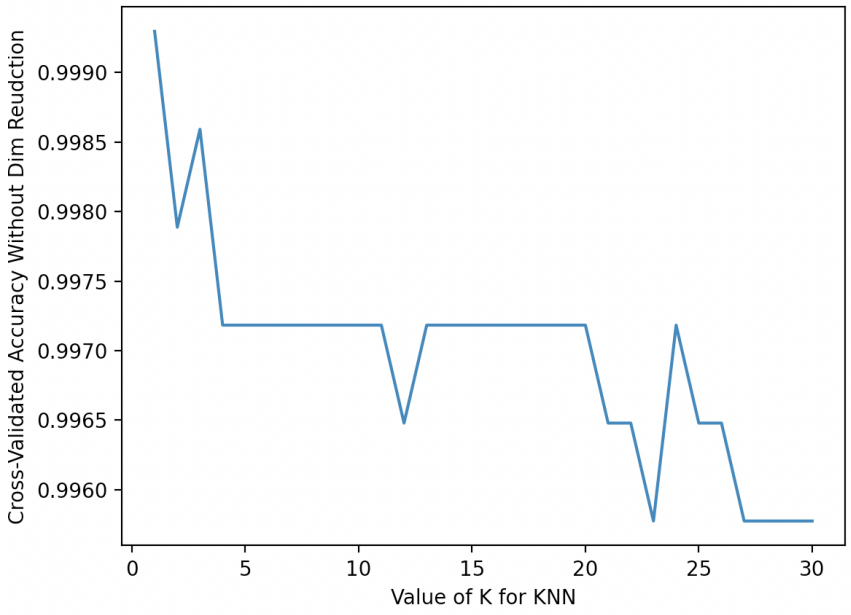
      I tweaked the number of K for KNN models, on a range from 1 to 30.

      For set 1, it shows that the highest performance(accuracy) is reached when k = 3. The accuracy at that K is 0.96.



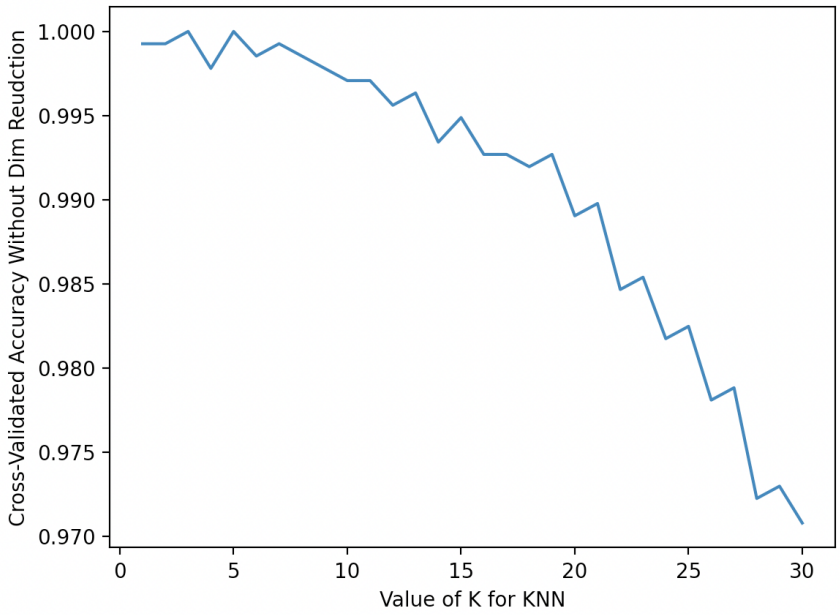
Cross-validation for KNN without dim reduction on set 1

For set 2, the best performance is reached when k =1. We get an accuracy of 1.0.



Cross-validation for KNN without dim reduction on set 2

For set 3, we get the best performance of 1.0 when k is 3.



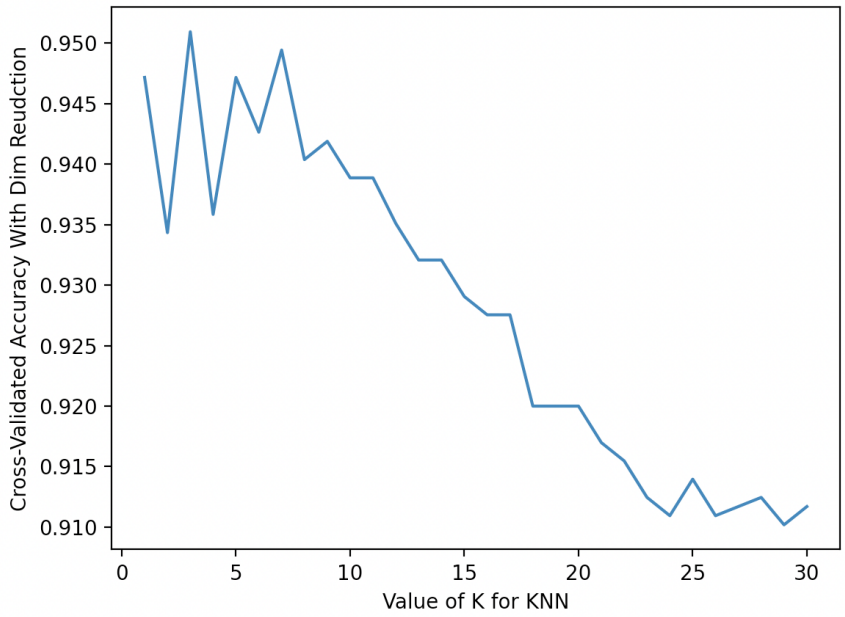
Cross-validation for KNN without dim reduction on set 3

c) Dimension reduction method:

For KNN, I used ***variance*** to filter out features that have low variance. Low variance means that the feature values for different samples varies little, which means we are likely to get little information from the feature values. I kept the top four features with the highest variance.

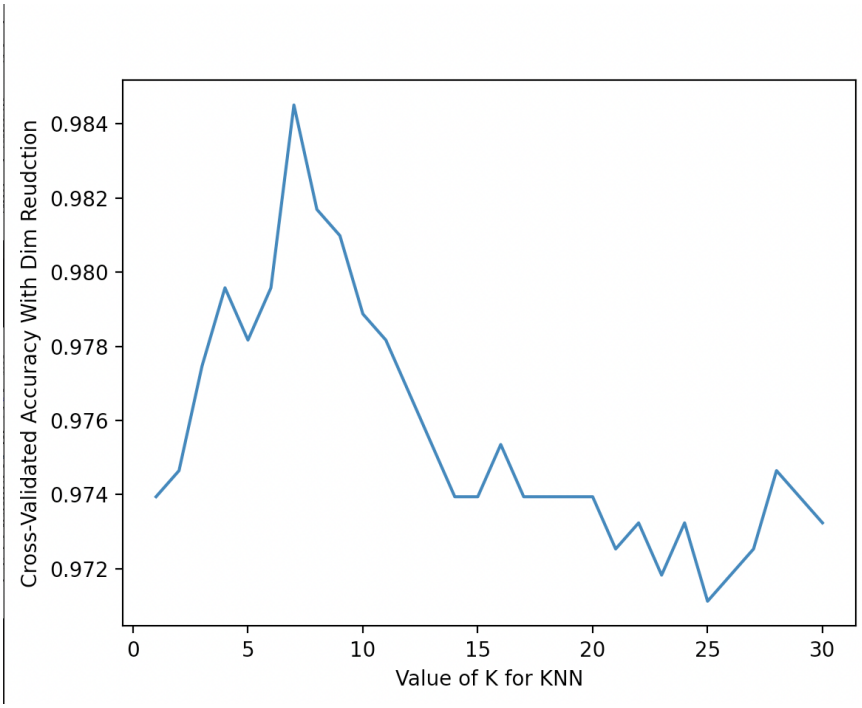
d) Graph for dim reduced sample set

For set 1, we get our best performance of 0.93 when k = 3.



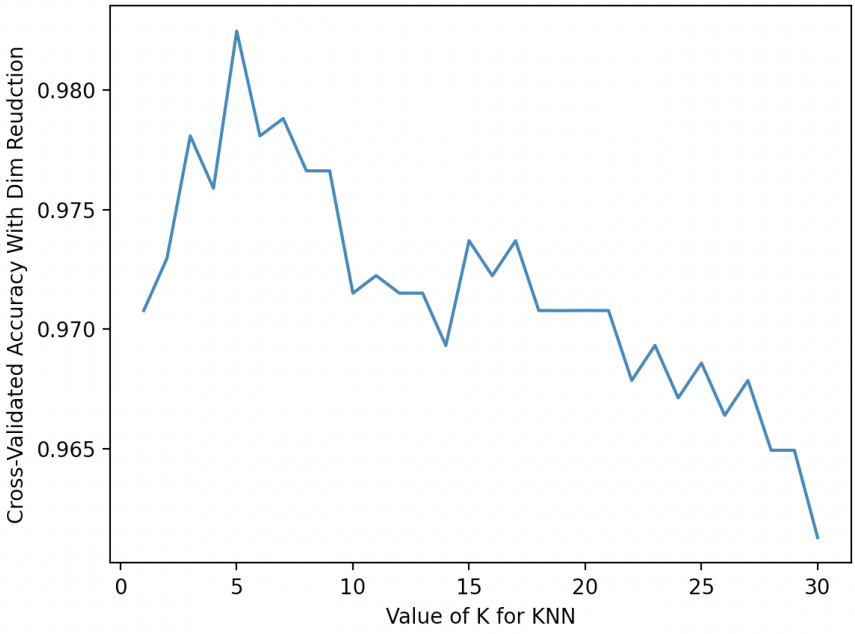
Cross-validation for KNN with dim reduction on set 1

For set 2, we get best performance of 0.97 when k = 7.



Cross-validation for KNN with dim reduction on set 2

For set 3, we get the best performance of 0.98 when k = 5.



Cross-validation for KNN with dim reduction on set 3

2) Decision Tree

a) Brief description of the classifier and its general advantages and disadvantages

Decision tree is a supervised learning method that can both be used as a classifier and a regressor. It builds a classification model based on decision rules at each split. And it determines which feature to split on at a level basedon the gini impurity of the nodes.

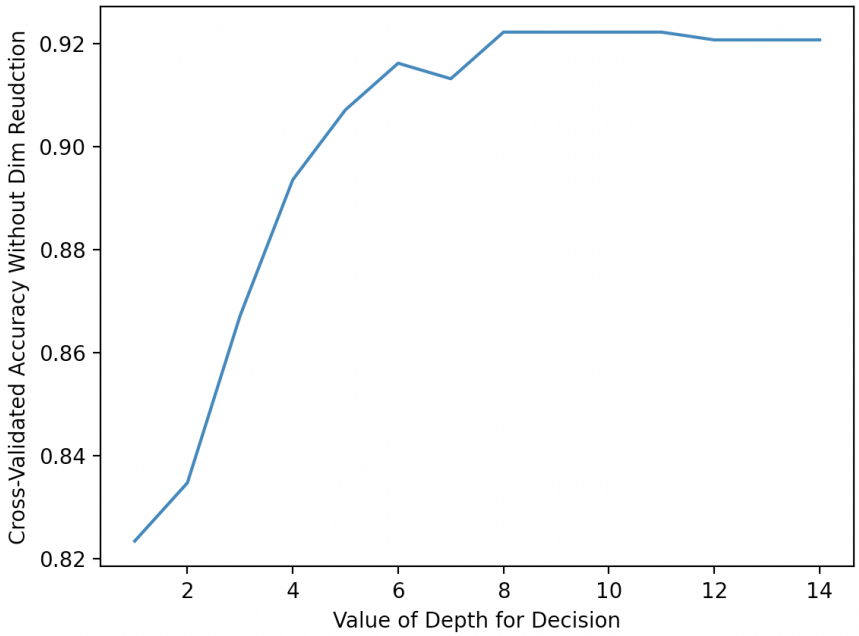
Decision tree is easy to apply. And humans can visualize and check the results at each split. It can be easily interpreted.

However, we can overfit our model to the training set by splitting too much. In that way, it's likely that we will not predict accurately for future data.

b) Graph

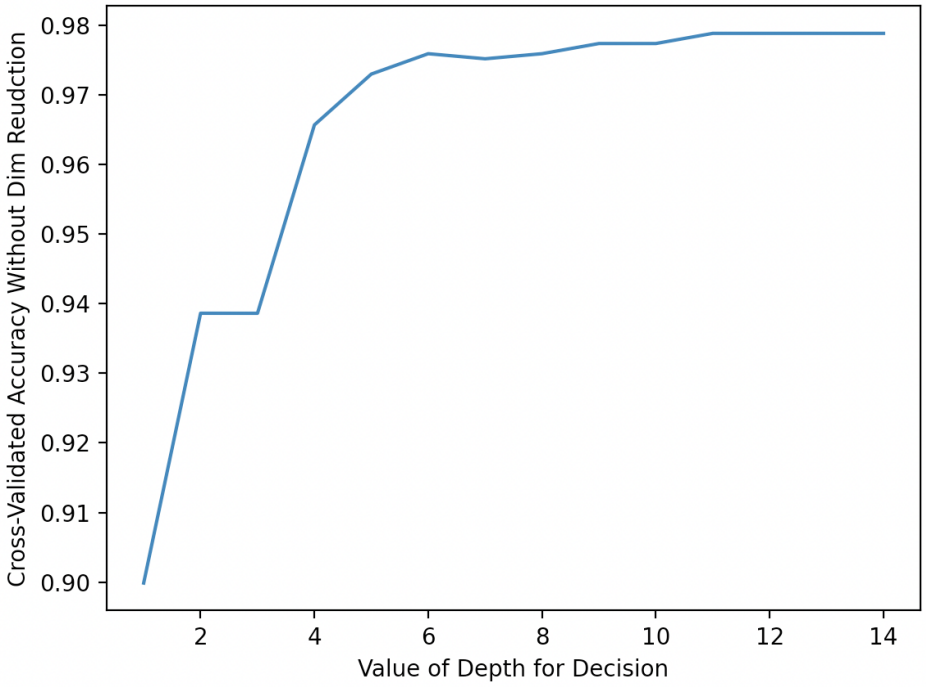
I tweaked the depth of the tree on a range from 1 to 15.

For set 1, it shows that the highest performance(accuracy) is reached when depth = 8. The accuracy at that depth is 0.93.



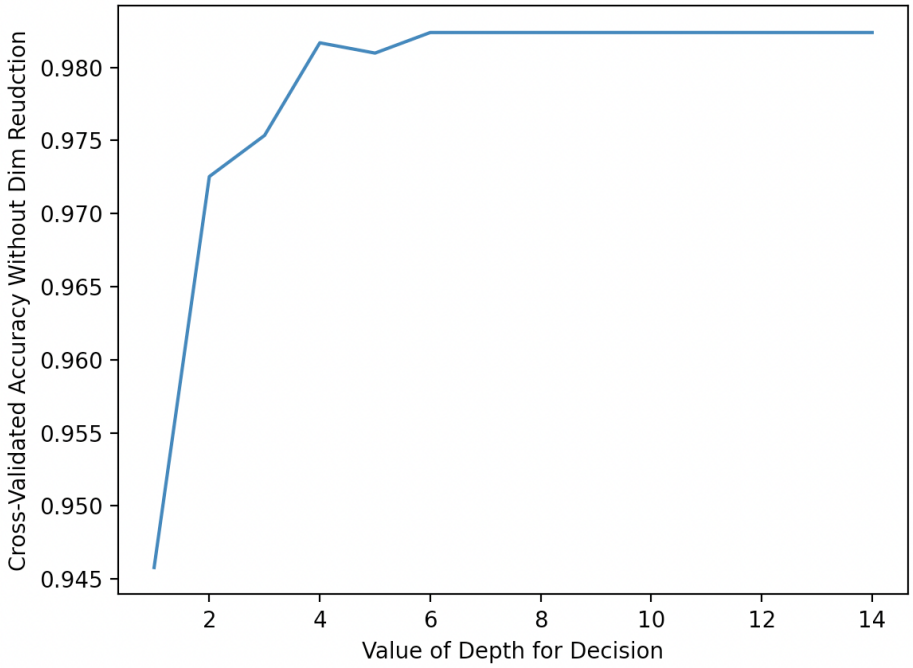
Cross-validation for Decision Tree without dim reduction on set 1

For set 2, we get the best performance of 0.97 when depth is 14.



Cross-validation for KNN without dim reduction on set 2

For set 3, the best performance is reached when depth = 7. We get an accuracy of 1.0.



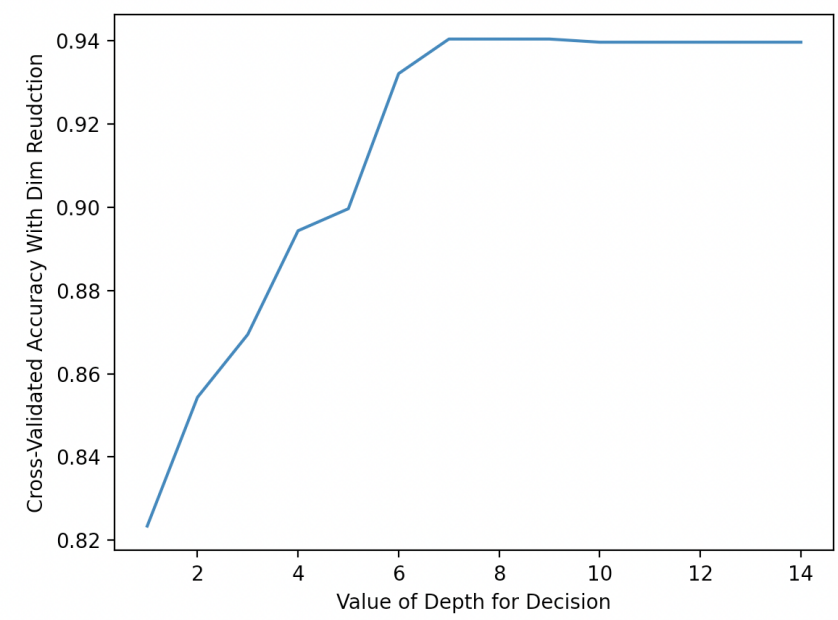
Cross-validation for Decision Tree without dim reduction on set 3

c) Dimension reduction method:

For the decision tree, I used ***greedy forward feature selection*** with a random forest classifier. This iteratively selects out the top four samples that are most important.

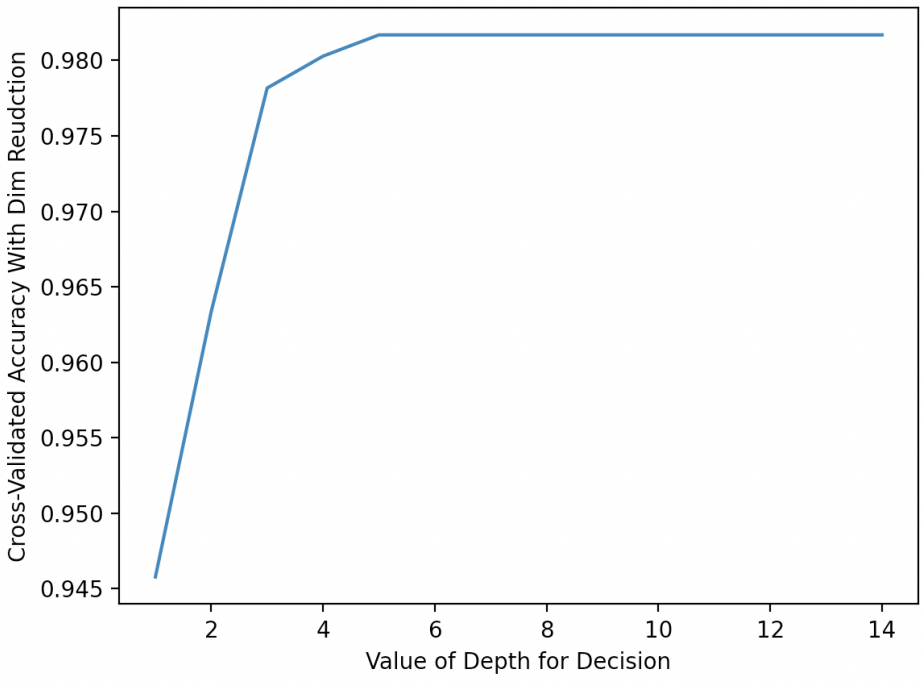
d) Graph for dim reduced sample set

For set 1, we get our best performance of 0.94 when depth = 7.



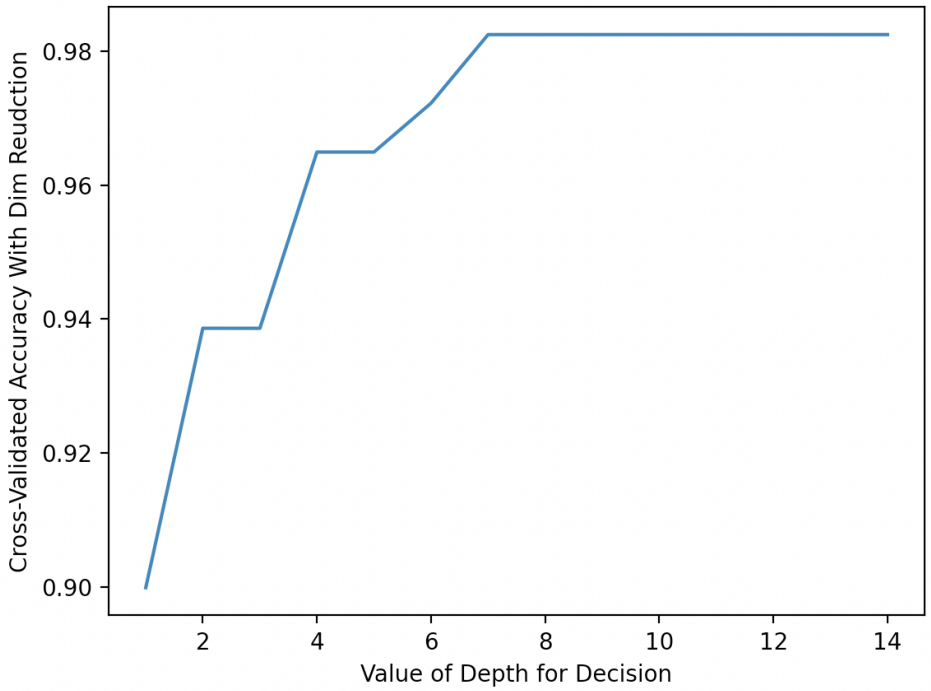
Cross-validation for KNN with dim reduction on set 1

For set 2, we get a best performance of 0.99 when depth = 5.



Cross-validation for KNN with dim reduction on set 2

For set 3, we get the best performance of 0.99 when depth = 7.



Cross-validation for KNN with dim reduction on set 3

3) Random Forest

a) Brief description of the classifier and its general advantages and disadvantages

Random forest is an ensemble learning method that can be used both as a classifier and a regressor. It consists of multiple decision trees. When it is as a classifier, it will output the class label given by most individual decision trees. When it is as a regressor, it will output the mean value given by individual decision trees.

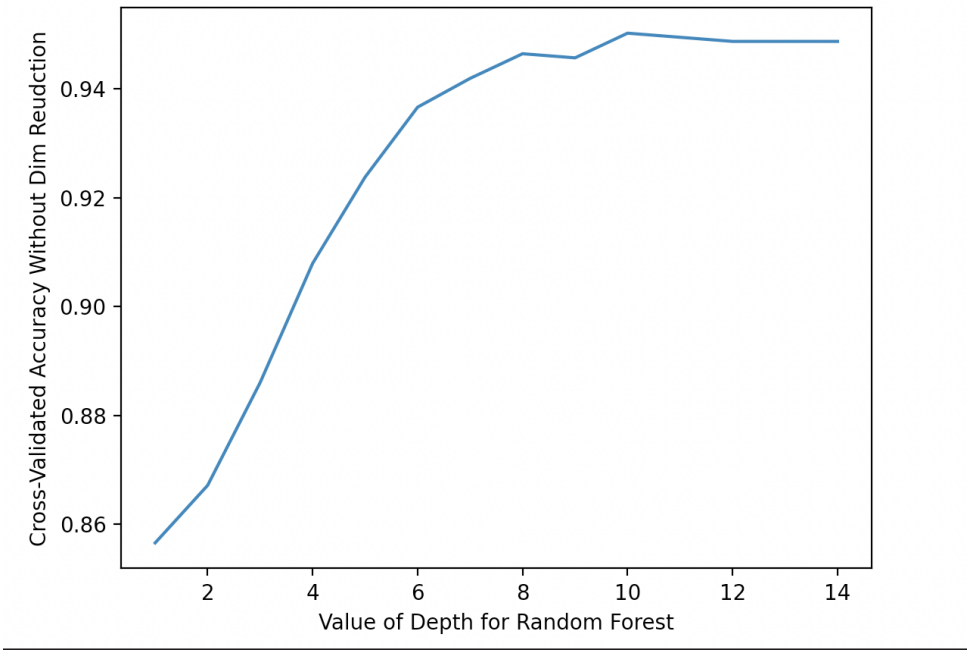
Random Forest can give us high accuracy while avoiding overfitting as single decision tree does.

However, it takes longer to train.

b) Graph

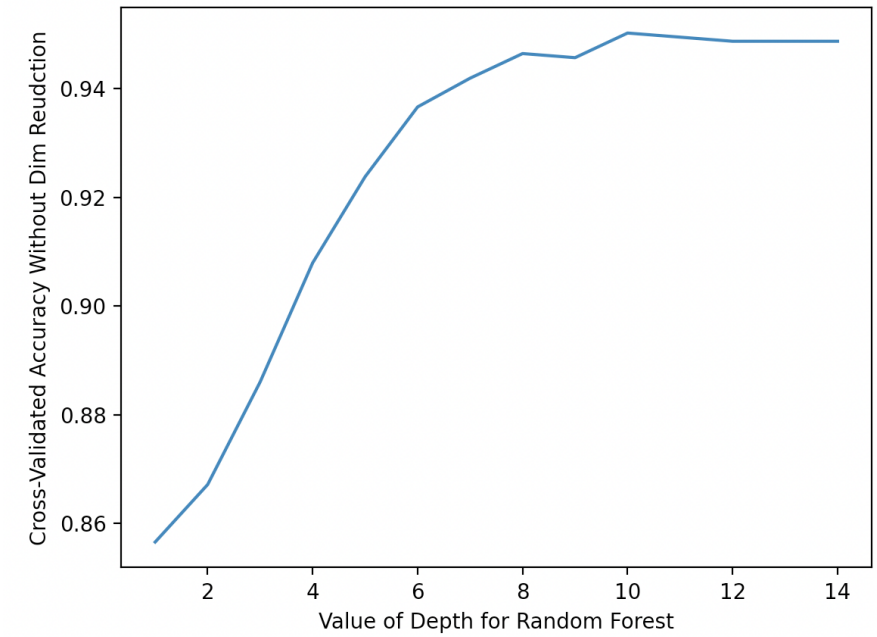
I tweaked the depth of the tree on a range from 1 to 15.

For set 1, it shows that the highest performance(accuracy) is reached when depth = 10. The accuracy at that depth is 0.97.



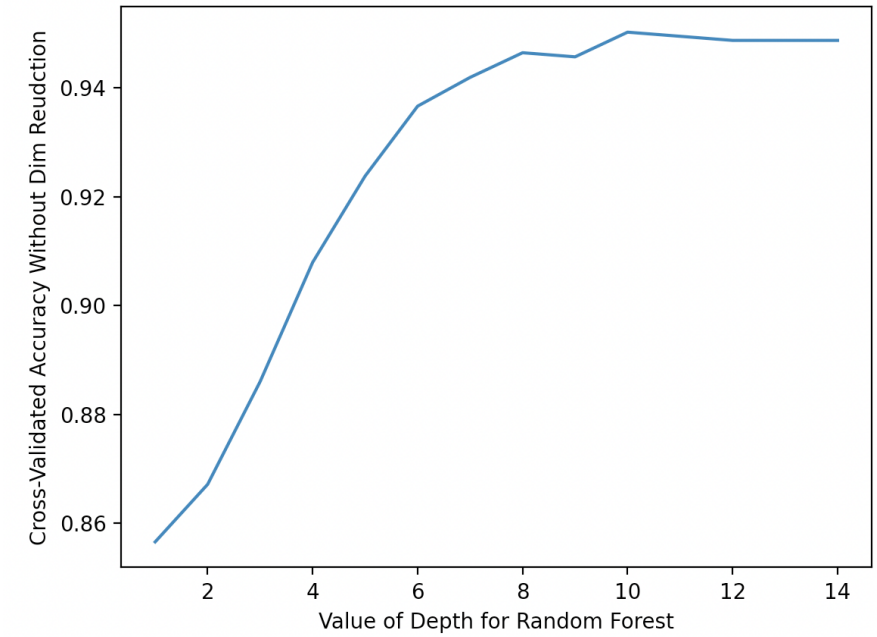
Cross-validation for Random Forest without dim reduction on set 1

For set 2, it shows that the highest performance(accuracy) is reached when depth = 11. The accuracy at that depth is 0.99.



Cross-validation for Random Forest without dim reduction on set 2

For set 3, it shows that the highest performance(accuracy) is reached when depth = 10. The accuracy at that depth is 0.99.



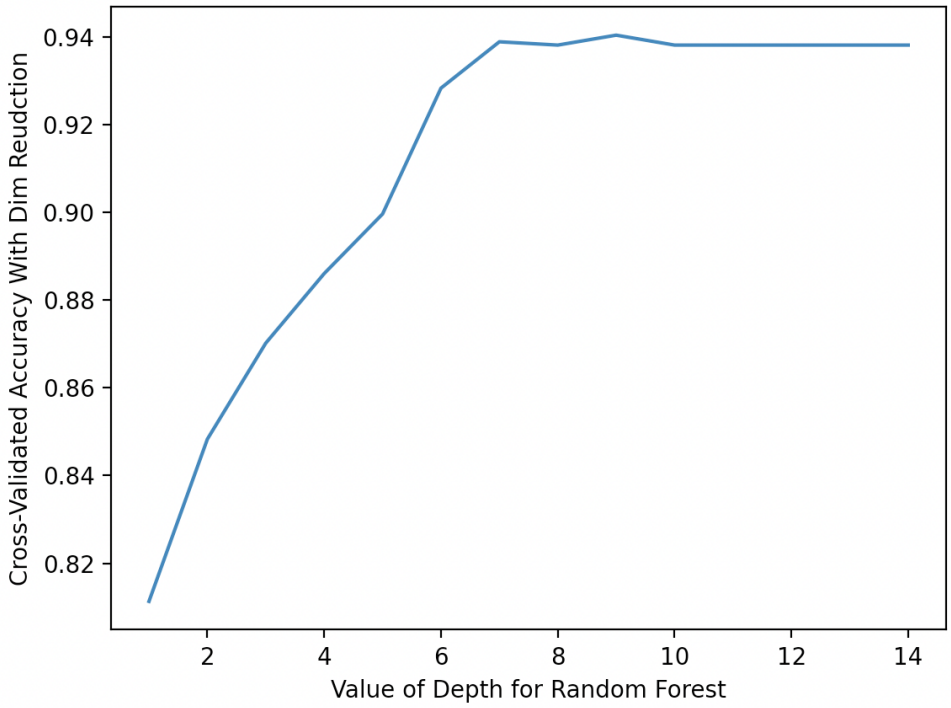
Cross-validation for Random Forest without dim reduction on set 3

c) Dimension reduction method:

For the random forest, I used ***greedy forward feature selection*** with a random forest classifier. This iteratively selects out the top four samples that are most important.

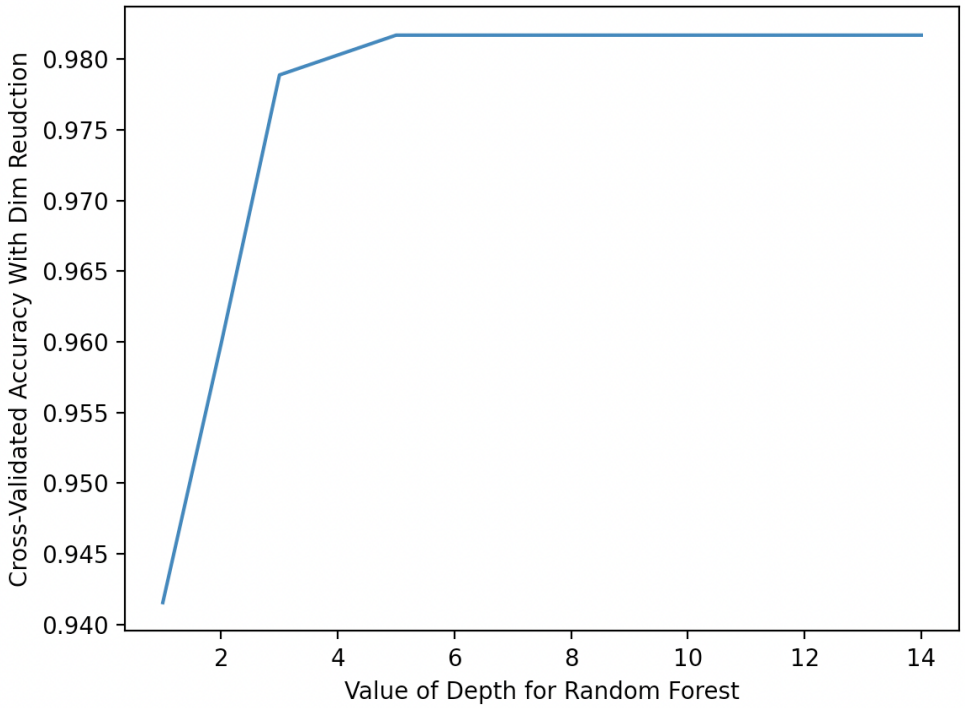
d) Graph for dim reduced sample set

For set 1, we get our best performance of 0.95 when depth = 7.



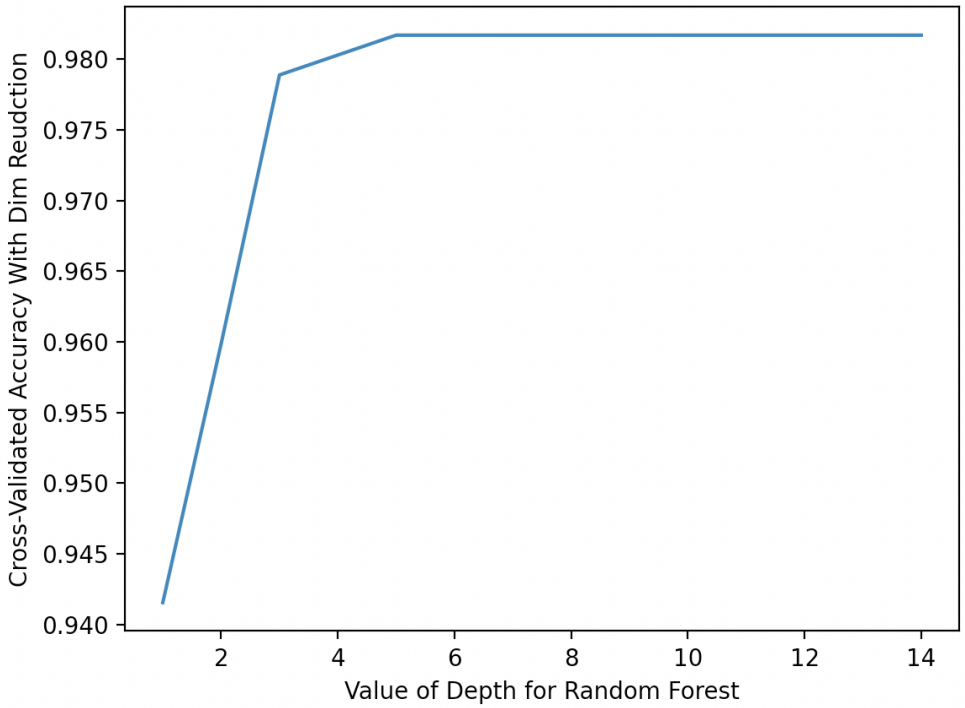
Cross-validation for Random Forest with dim reduction on set 1

For set 2, we get our best performance of 1.0 when depth = 5.



Cross-validation for Random Forest with dim reduction on set 2

For set 3, we get our best performance of 0.97 when depth = 5.



Cross-validation for Random Forest with dim reduction on set 3

4) SVM

a) Brief description of the classifier and its general advantages and disadvantages

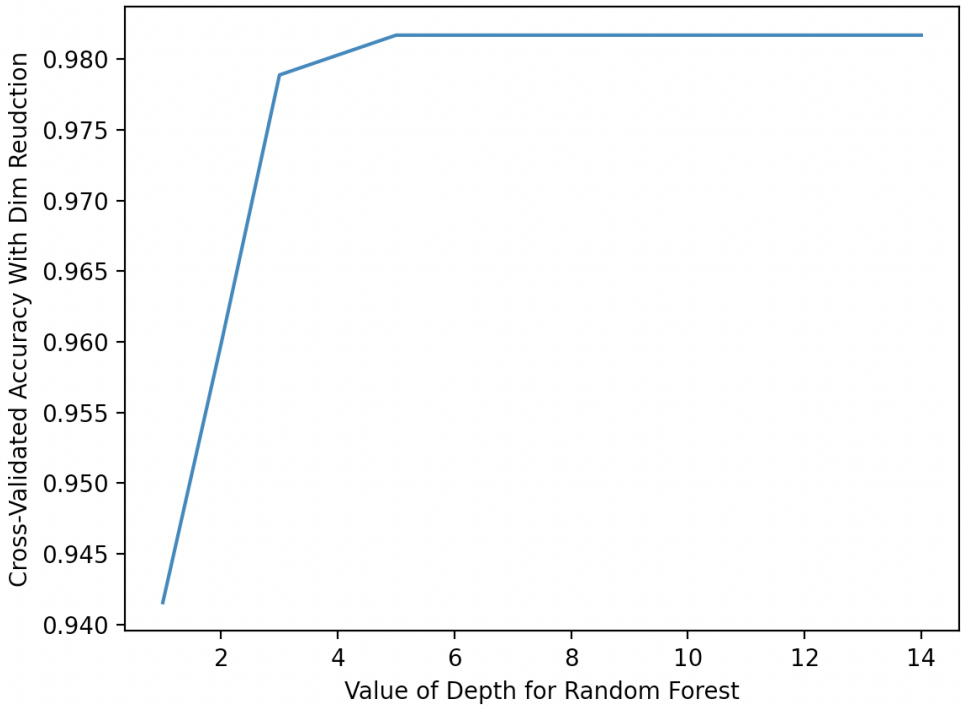
SVM is a supervised linear learning method that can act as a regressor and classifier. It works by first finding a hyperplane. The best hyperplane can 1) maximize accuracy and 2) maximize the margin between classes. When data is not linearly separable, we need to use kernels to project data to higher dimensions when they can be linearly separated by SVM.

However, it takes longer to train.

b) Graph

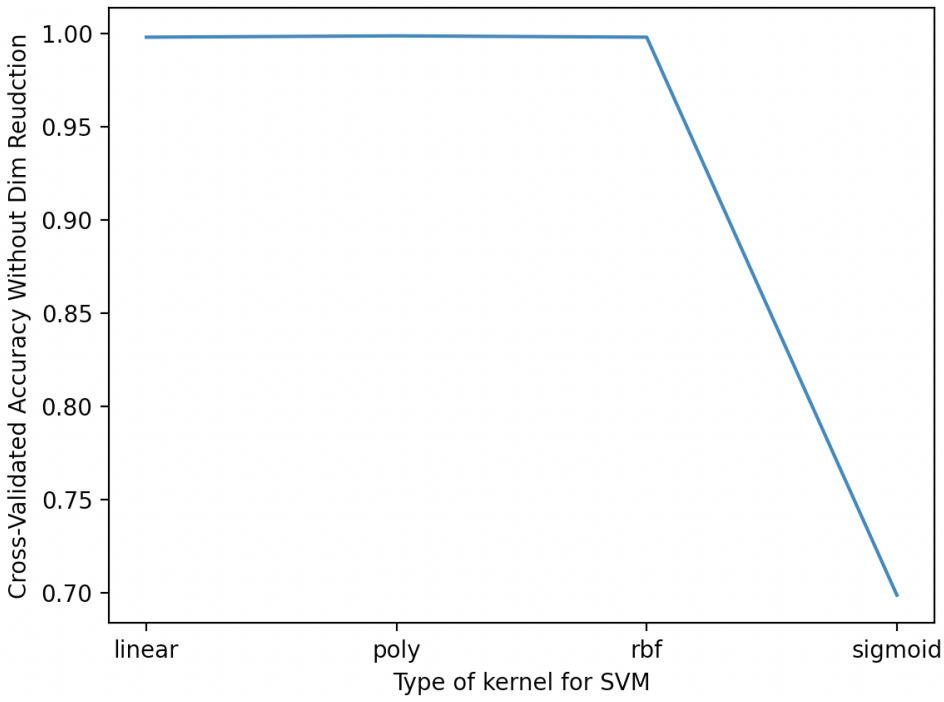
I tweaked a categorical hyperparameter, the kernel used including linear, rbf, poly, and sigmoid.

For set 1, we get our best performance of 0.99 when we use poly as the kernel.



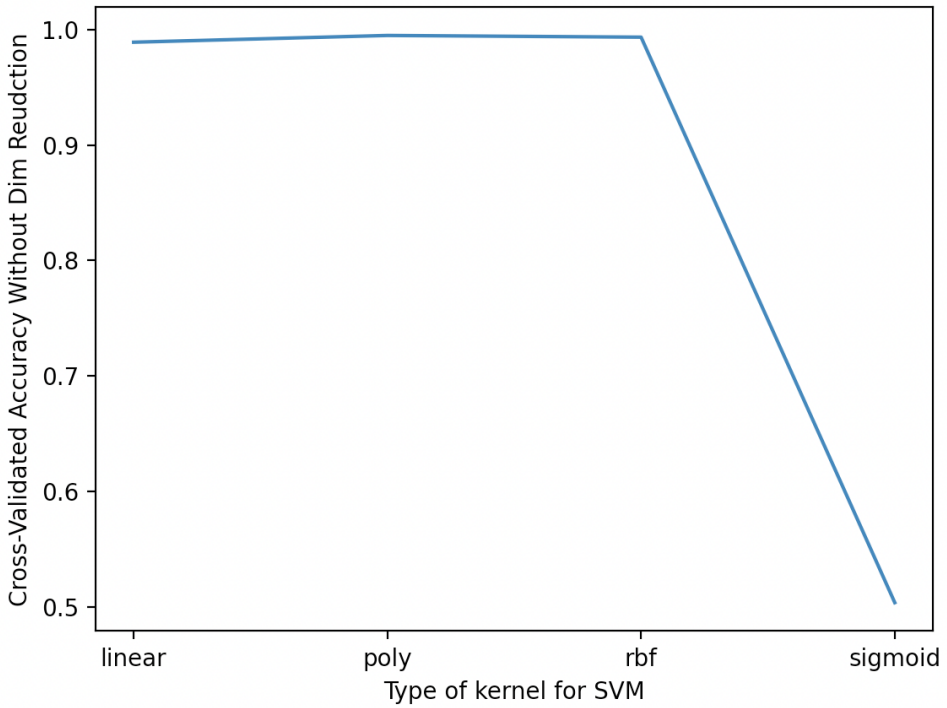
Cross-validation for SVM without dim reduction on set 1

For set 2, we get our best performance of 1.0 when we use poly as the kernel.



Cross-validation for SVM without dim reduction on set 2

For set 3, we get our best performance of 0.99 when we use poly as the kernel.



Cross-validation for SVM without dim reduction on set 3

c) Dimension reduction method:

For the SVM, I first wanted to use ***correlation filters*** to eliminate some features that have highly-correlated features in the data set. So I checked the correlation matrix and got the result below:

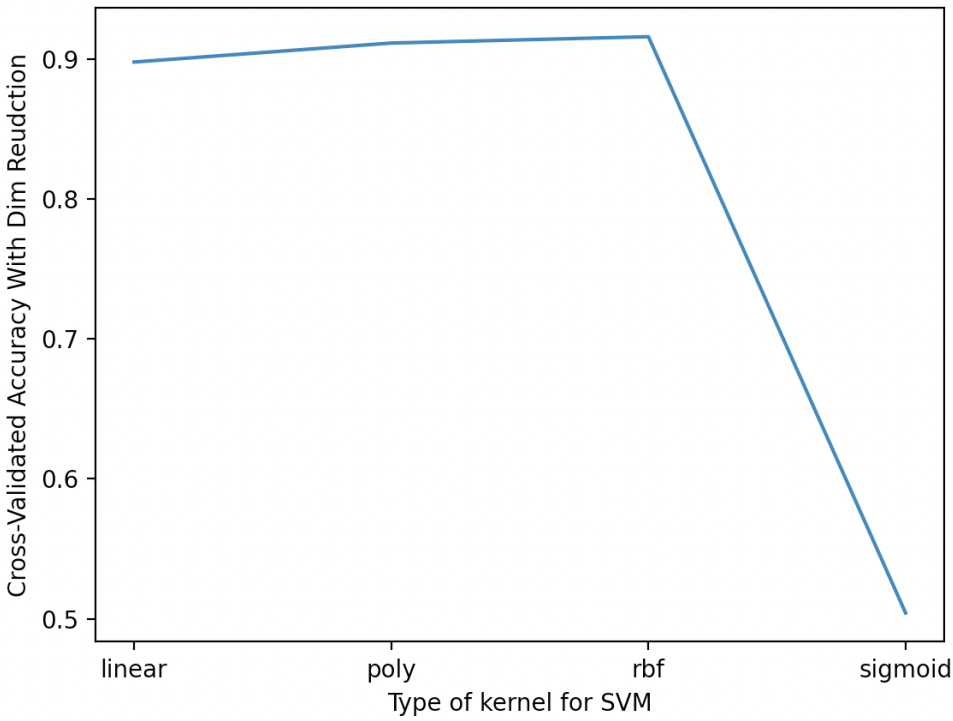
	1	2	3	...	14	15	16
1	1.000000	0.825619	0.875030	...	0.041999	0.275621	-0.150844
2	0.825619	1.000000	0.724912	...	-0.001970	0.267101	-0.067352
3	0.875030	0.724912	1.000000	...	-0.081478	0.414513	-0.226571
4	0.720759	0.849769	0.702218	...	-0.049132	0.341307	0.008167
5	0.605022	0.629549	0.778510	...	-0.204261	0.645038	-0.123221
6	0.173815	0.097215	0.212105	...	-0.022354	0.155475	-0.007441
7	0.019811	-0.015202	-0.009472	...	0.224623	-0.055689	-0.360085

8	-0.237701	-0.107918	-0.440695	...	0.079862	-0.591079	0.129998
9	0.291163	0.152212	0.464334	...	-0.032296	0.329444	-0.273539
10	0.380501	0.192408	0.510370	...	0.059463	-0.050012	-0.368193
11	0.015034	0.001555	0.036075	...	0.360252	-0.112790	-0.529061
12	-0.154970	-0.118982	-0.160609	...	0.008223	-0.118153	0.154331
13	0.484168	0.463902	0.426044	...	-0.099411	0.574503	0.025210
14	0.041999	-0.001970	-0.081478	...	1.000000	-0.228747	-0.251629
15	0.275621	0.267101	0.414513	...	-0.228747	1.000000	0.117912
16	-0.150844	-0.067352	-0.226571	...	-0.251629	0.117912	1.000000

We can see that there are only 4 visibly highly correlated features, but our goal is to reduce our feature dimension size to 4. So I think using correlation filters will likely eliminate many features that could be valuable to train our model. Then I used ***embedded filters***. Random forest or decision trees innately prioritize the splitting on features that minimize gini impurity. The earlier features are splitted in the tree means that they are more important. So I used a random forest classifier to give me feature importance and kept the top four features.

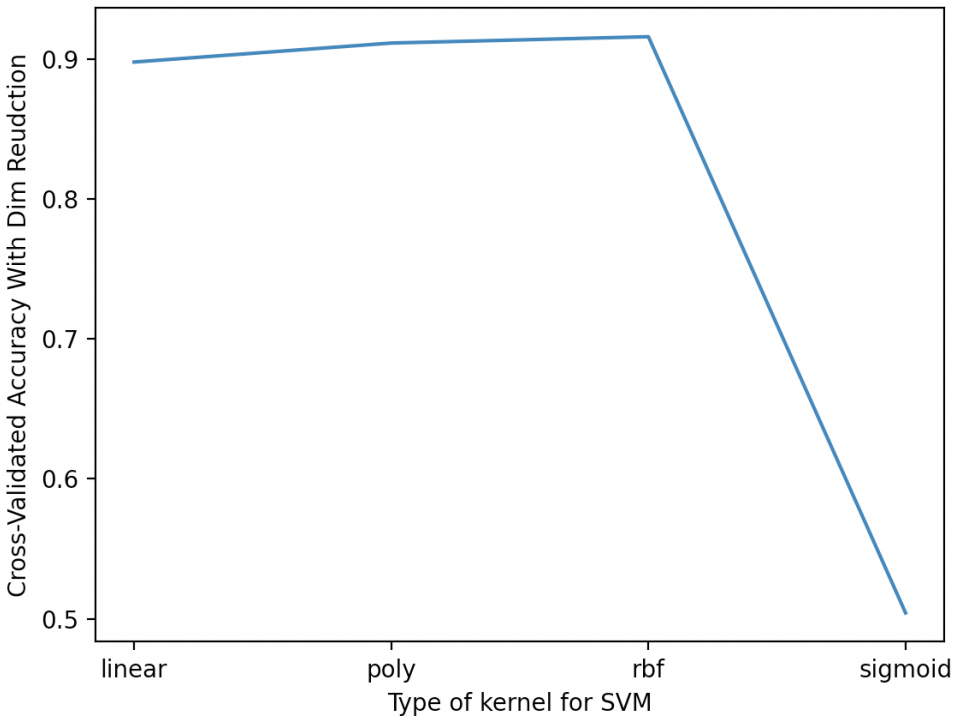
d) Graph for dim reduced sample set

For set 1, we get our best performance of 0.94 when using poly as our kernel



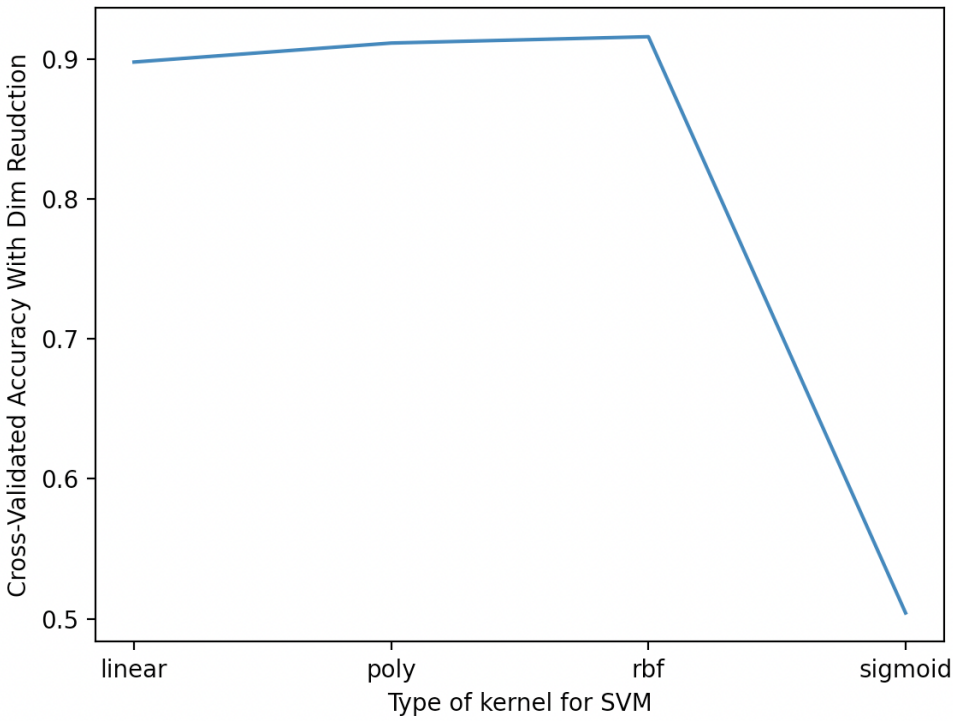
Cross-validation for SVM with dim reduction on set 1

For set 2, we get our best performance of 0.97 when using rbf as our kernel



Cross-validation for SVM with dim reduction on set 2

For set 3, we get our best performance of 0.95 when using rbf as our kernel



Cross-validation for SVM with dim reduction on set 3

5) ANN

a) Brief description of the classifier and its general advantages and disadvantages

ANN mimics human neuron networks. It includes input, hidden and output layers.

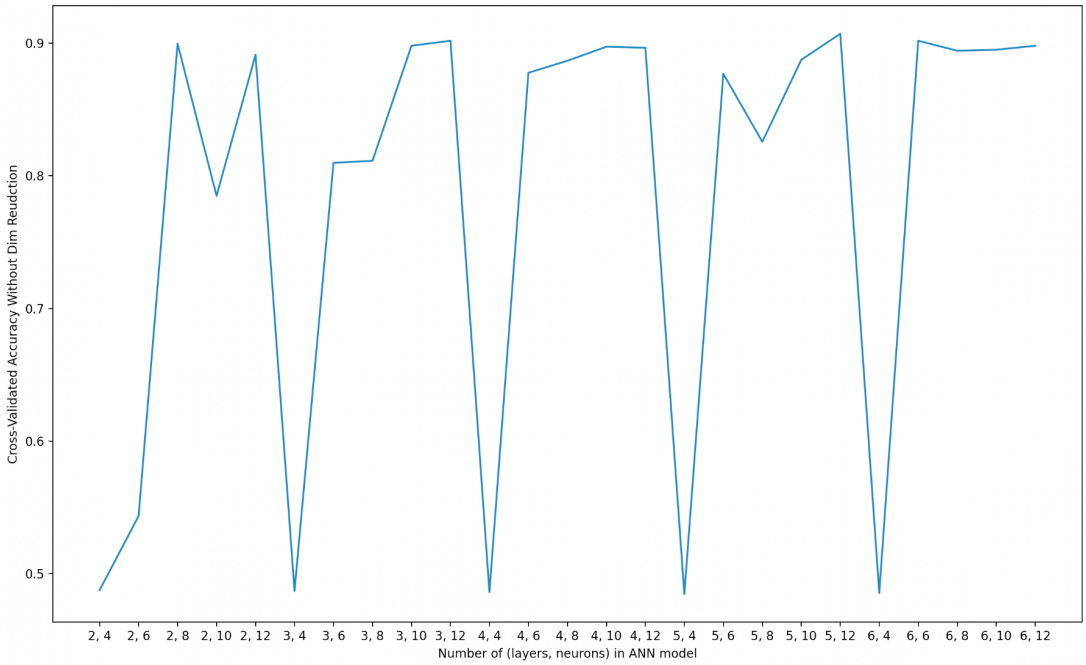
It is good at identifying patterns and is able to pass important information to the next layer. Unlike SVM, which is a linear learning model, ANN can learn and model non-linear and complex relationships. It can also infer unseen information and unseen data.

However, it takes longer to train and requires processors with parallel processing power. Also, although it can give reliable predictions, the model itself can not be explained and interpreted easily. We cannot tell how it gets to the final result.

b) Graph

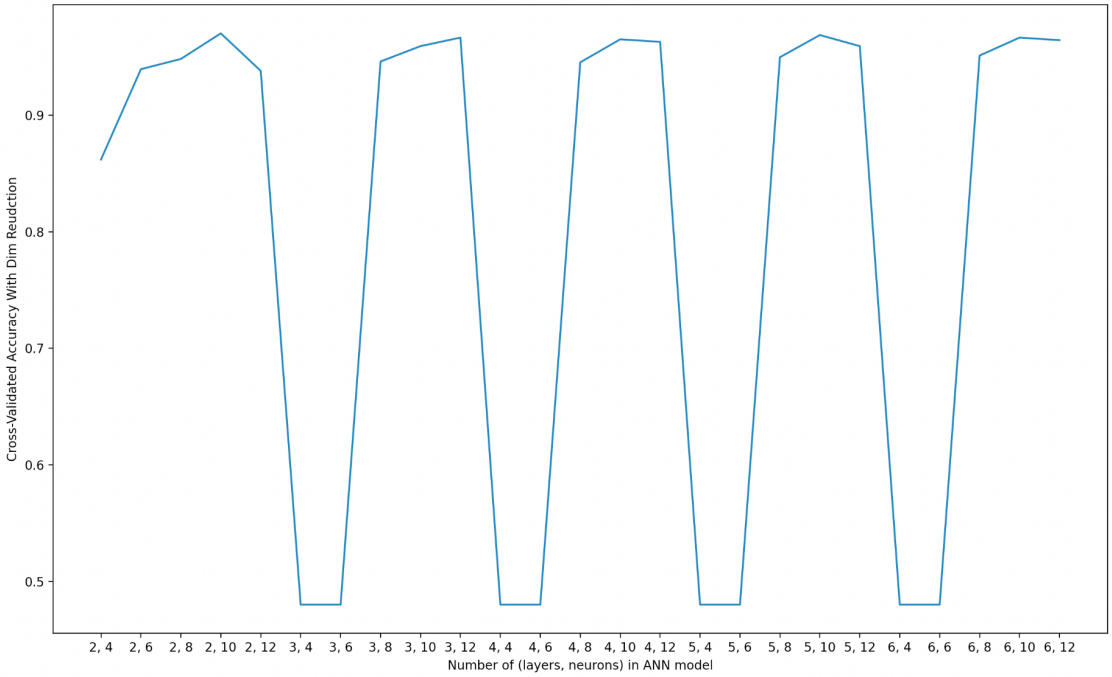
I tweaked the ***layer and the neuron size*** of the network. Layer ranges from [2,3,4,5,6] and neuron size ranges from [4,6,8,10,12].

For the original data set 1, we get our best performance of 0.94 when we use 5 layers and 12 neurons.



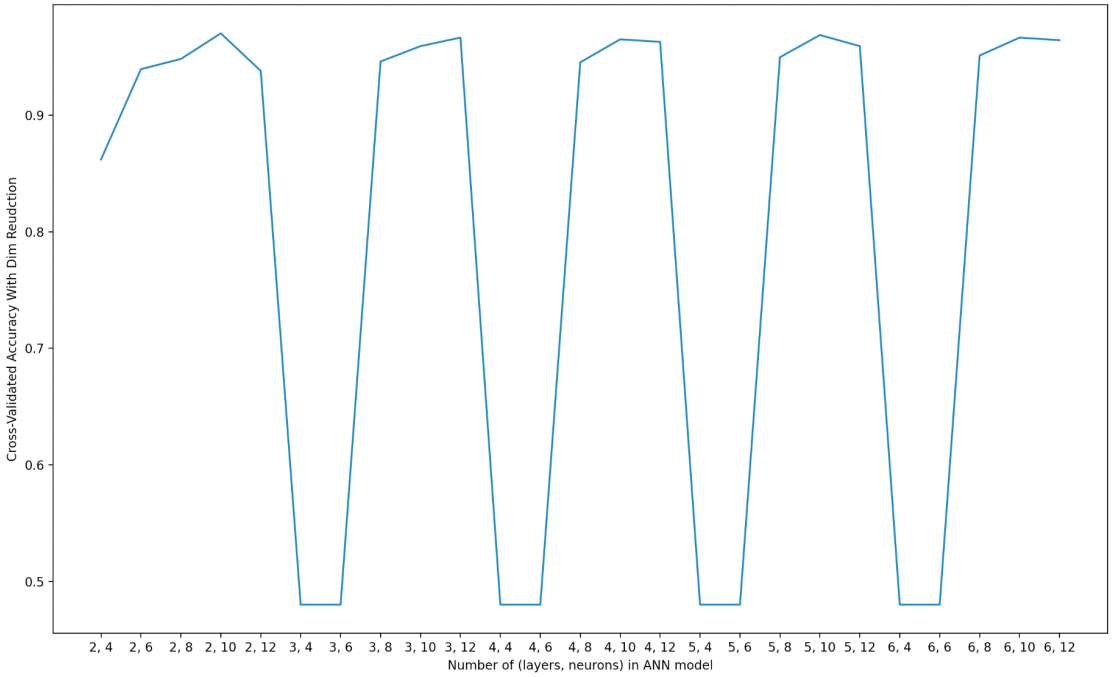
Cross-validation for ANN without dim reduction on set 1

For the data set 2, we get our best performance of .97 when we use 5 layers and 12 neurons on the test data set. ANN performs well on this data set, but still not as good as it does on filtered data.



Cross-validation for ANN without dim reduction on set 2

For the data set 3, we get our best performance of .79 when we use 2 layers and 10 neurons on the test data set. Although ANN has pretty good performance on training data set with 10 epochs and 20 batch size, it doesn't maintain the performance on the test set with the parameters tested on the training set.



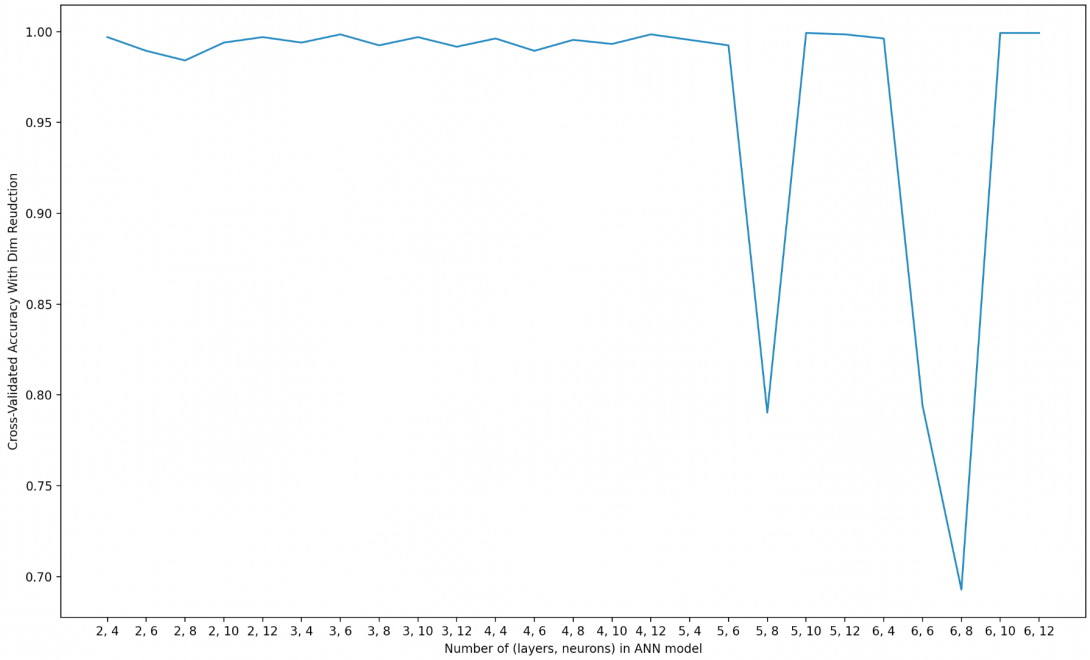
Cross-validation for ANN without dim reduction on set 3

c) Dimension reduction method:

For ANN, I used **autoencoders** to reduce the dimensions. Autoencoder is an additional model trained for us to select out valuable features from original feature set. It does so by enforcing a bottleneck(in our case, I used a bottleneck of 4) to force the feature dimensions to reduce. We then use the trained encoder to preprocess our original X data set.

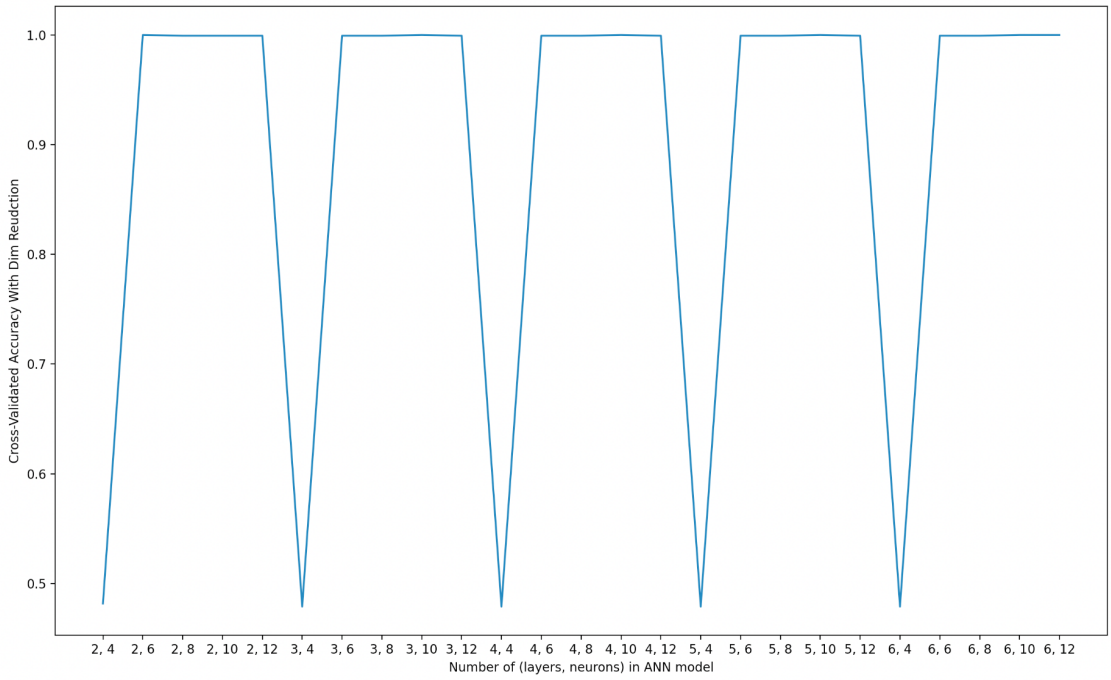
d) Graph for dim reduced sample set

For the dim-reduced data set 1, we get our best performance of 1.0 when we use 5 layers and 10 neurons.



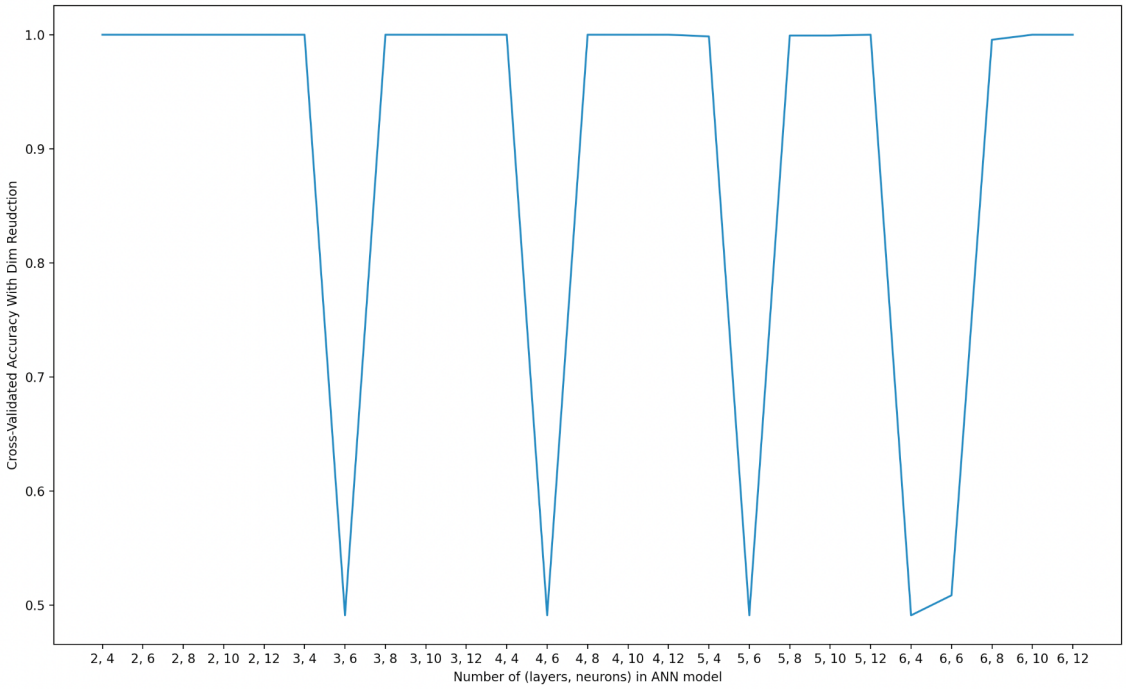
Cross-validation for ANN with dim reduction on set 1

For the data set 2 with dimension reduction, we get our best performance of 1.0 when we use 2 layers and 6 neurons. But we can notice that ANN performs consistently pretty well on this data set. The parameter layer doesn't affect the performance a lot, but we can see the number of neurons still have some affect on the performance. When the neuron size is four, we get pretty bad performance around 0.5.



Cross-validation for ANN with dim reduction on set 2

For the data set 3 with dimension reduction, we get our best performance of 1.0 when we use 2 layers and 4 neurons. But we also can notice that ANN performs consistently pretty well on this data set. The parameter layer doesn't affect the performance a lot, but we can see the number of neurons still have some affect on the performance. When the neuron size is six, we get pretty bad performance around 0.5.



Cross-validation for ANN with dim reduction on set 3

Results

1. **Performance and exe time for validation sets before dim reduction**

original	KNN	DT	RF	SVM	ANN
exe_time	2.4s	0.3s	7.2s	0.57s	825s
accuracy	99%	97%	98%	99%	92%

2. **Performance and exe time for validation sets after dim reduction**

dim_reduced	KNN	DT	RF	SVM	ANN
exe_time	0.94s	21.5s	21s	0.95s	731s
accuracy	96%	97%	97%	95%	100%

*exe time for ANN excludes the time to train autoencoders

*the exe time for DT is much longer after dim reduction because I used greedy forward selection with random forest to select out features.

3. Lessons Learned

For this problem set, I would choose **SVM** classifier because compared to other classifiers, it has benefits below. 1) It has an overall one of the best performance on the training dataset . Even though after we do dimension reduction, we lose some accuracy. But that can be explained by that we force the feature set to only contrain four features. 2) It takes the shortest amount of time to train. 3) comprared to DT and RF, its results can be more easily interpeted. We can look at the hyperplane that seperates out the two classes and know what conditions are. On the other hand, it's hard for us to track through the split rules for DT or RF.

Dimension reduction helps reduce the overall ***execution time*** for KNN and ANN classifiers, but increases the execution time for other classifiers. This is reasonable because with dim reduction, the computation needed for classification is significatnly lessoned for these two models. For KNN, we know that the most computation-heavy part is calculating the distance between data_to_be_classified and its k nearest neighbors. With the original dataset, we need to calculate the distances between all 16 feature values from the feature values of the data_to_be_classified. So reducing the feature space can largely decrease the amount of computation we need to do. For ANN, similarly, decreasing the number of features largely decrease the computation needed due to the optimization logic it has. On the other hand, with dimension reduction the ***accuracy*** is lower for KNN. It can be explained by the fact that reducing the feature space makes samples cluster together. For this problem set, samples may be clustered after dimension reduction. It's likely that the labels given by k nearest neighbors produce a tie, or a majority that is incorrect, and this causes the prediction to be wrong. For KNN, the sparser the samples are, the more likely we will get an accurate prediction. It is also notable that the performance for ANN decreases significantly after feature reduction. This can be explained by that autoencoder did a good job to force a bottleneck on the input features and successfully extract valuable information from them. Because I set the epoch to 10, ANN without dimension reduction may end up with a higher performance if I set the epoch to 100 or higher. But that means we have to take much longer time for the model to be trained. The time to train autoencoder is definitely much shorter on these 20,000 samples. the overall execution time increases but the accuracy doesn't increase by a lot. This is probably because I used forward selection with a random forest to select out features and random forest classifiers generally takes long to train. So the benefit of using dimension reduction for this problem set doesn't outweigh the cost of training additional models to do dim reduction. It costs addtional

time and forcing the dimension reduction makes us lose some valuable information about the original data.

However, even if I choose SVM as the classifier for this problem, if we are given a different problem dataset, we may choose the other classifiers. We can notice there are several characteristics about the dataset we are given. 1) the *overall sample size is not too large*. There are only 20,000 samples, but the feature dimension is comparably large, with 16 different features. Thus, we can predict that the data is likely sparsely distributed. 2)*the problem is straightforward* - we are given features about the text and with the given feature values, we will predict the character. Using ANN in our case is wasteful because it is a simple-logic problem with well-defined featuers. Although it achieves the best performance, the execution time is 1000 times more than other classifiers. Depending on the scenario, I may have different choices of classifier.

- If there are much more data than 20,000 with the same or more dimensions(16):
    - I may still choose SVM because SVM will project data onto higher dimensions if the data is not linearly separable. Compared to Decision trees or Random forest classifiers, its hyperflane can give a clear global optimal values for all features, whereas DT or RF will be much harder to track the full optimal path up to the leaf node.
    - I may also choose KNN in this scenario because with more dimensions, it means the data is more sparsely distributed. KNN can be a good candidate for a classifier when the data is sparse in dimensions.
- If there are much more data with less dimensions:
    - I may choose RF or DT. I will first try DT because it doesn't take as long as RF to be trained. But if its performance on the test dataset isn't well, I may use RF because RF can prevent the problem DT usually face, overfitting.
    - I may also choose ANN because in this case, we may find SVM cannot perform as well because it cannot linearly separate out data as good with such amount of dimensions. But ANN is good at extracting information. As we can see from the performance of ANN after dim reduction, autoencoders largely increase the performance of ANN with little time cost to train the autoencoder model and decerases the overall execution time. Although for this problem set ANN's execution time is the latest of all, when the problem dataset is huge, ANN can be a good candidate that gives a good balance between execution time and accuracy compared to other models.