# Programming Assignment 2

Name: Amit Sarker, ID: 500969

## 1 Introduction

The main goal of this assignment is to get a hands-on experience with dimension reduction and the comparison of different classification models on a given dataset. In this assignment, we need to test seven different classification models: k-Nearest Neighbors (KNN), Decision Tree classifier, Random Forest classifier, Support Vector Machines (SVM), Artificial Neural Network (ANN), Logistic Regression, and Naive Bayes classifier. Logistic Regression, and Naive Bayes classifier are my choice of classifiers for the extra credit. There are many different types of classification tasks that we can perform. Each task often requires a different algorithm because each one is used to solve a specific problem. For each problem, we must select the right algorithm. That's why we need to determine the "best" classifier for a given problem. It's important to evaluate the trade-off between aspects like model performance, computational complexity, validation accuracy, model interpretability, and select the model that's best to solve the problem. For this reason, I have used the validation accuracy and computational complexity to find the "best" model for this specific problem.

Classification is a process that necessitates the application of machine learning algorithms to learn how to assign a class label to problem domain instances. Classifying emails as "spam" or "not spam" is an easy example to comprehend. Binary classification refers to predicting one of two classes. In this assignment, we have used the Letter Recognition dataset provided by the University of California Irvine Machine Learning Repository. For the first two classification problem, I have used the Pair H, K and Pair M, Y. For the third problem I have used the Pair A, B. By checking the dataset, i think the Pair M, Y will be the easiest to classify because it has more date instances compared to the other pairs.

Techniques for reducing the number of input variables in a dataset are known as dimension reduction. More input features often make a predictive modeling task more challenging to model, more generally referred to as the curse of dimensionality. With a high number of dimensions in the feature space, the volume of that space can be rather huge, and the points (rows of data) that we have in that space frequently reflect a tiny and non-representative sample. This can dramatically impact the performance of machine learning algorithms fit on data with many input features. Therefore, it is often desirable to reduce the number of input features. However, it is also crucial to ensure that critical data characteristics are not lost. There are many techniques that can be used for dimension reduction and there is no single best method for dimension reduction. For this assignment, I have used at least three different dimension reduction strategies and comparing their results to those without dimension reduction. The most common dimension reduction methods are: selecting the features with highest variance for KNN, SVM, and Decision Tree and PCA for ANN.

# 2 Results

In this section, for each classifier, I will first add a brief description of the classifier and I will also add the cross validation results for both with and without dimension reduction.

## 2.1 k-nearest neighbors (KNN)

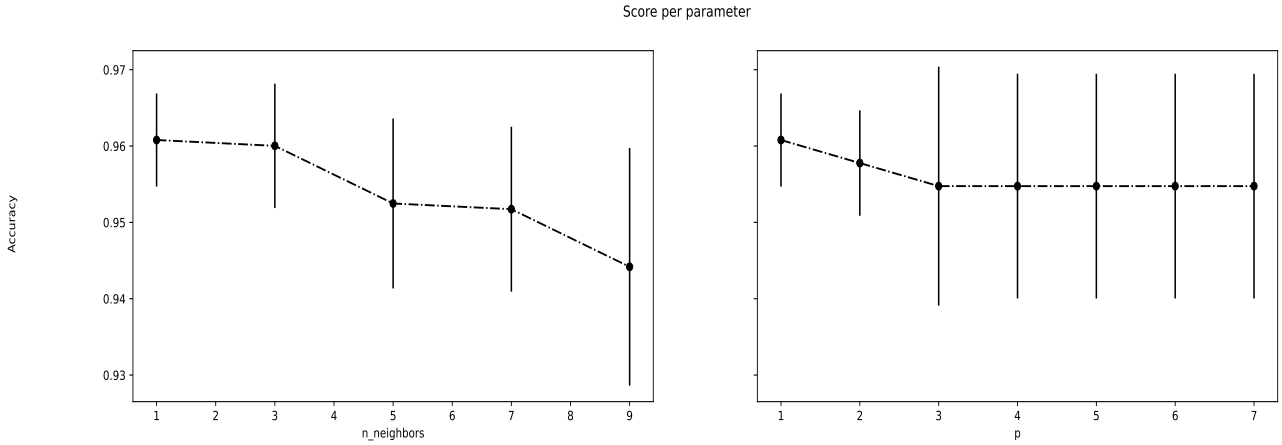
The KNN algorithm assumes that objects that are similar are near together. To put it another way, related items are close together. The algorithms calculate the distances between a given data point in the set and any other $K$ numbers of data points in the dataset that are near to the original point, then vote for the category with the highest frequency. Typically, Euclidean distance is used as a distance measurement. As a result, the final model is just labeled data in a space. The main advantages of the KNN classifier is that it is very simple to implement and intuitive to understand. However, for large datasets, the prediction complexity of KNN is very high and KNN is also very sensitive to outliers.

In this assignment, I have used $n\_neighbors$ and $p$ hyperparameters. For the distance calculation, I have used Manhattan distance for $p = 1$, Euclidean distance for $p = 2$, and Minkowski distance for $p > 2$. Figure 1 shows the hyperparameter tuning results for KNN without dimension reduction.

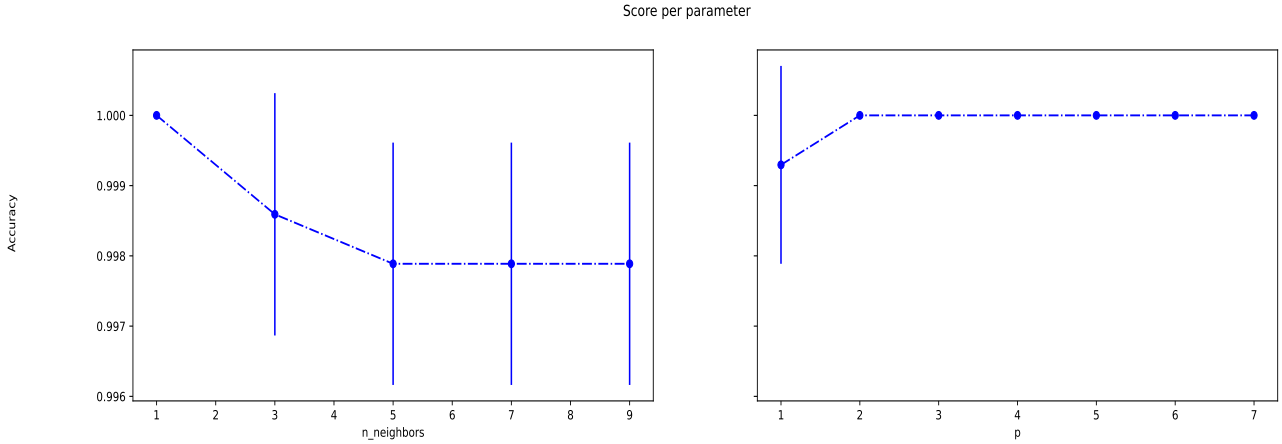
In Figure 1a, we notice that for $n\_neighbors = 1$ and $p = 1$ is the best hyperparameter values for Pair 1 (H and K) with an accuracy of 96%. In Figure 1b, $n\_neighbors = 1$ and $p = 2$ is the best hyperparameter values for Pair 2 (M and Y) with an accuracy of 100%. In Figure 1c, $n\_neighbors = 1$ and $p = 1$ is the best hyperparameter values for Pair 3 (A and B) with an accuracy of 100%.

For dimension reduction, I have used the four features with highest variance in each set. Values change very slightly across samples when features have a low variance. That is why, features with high variance are more crucial than the low variance features. Figure 2 shows the hyperparameter tuning results for KNN with dimension reduction.

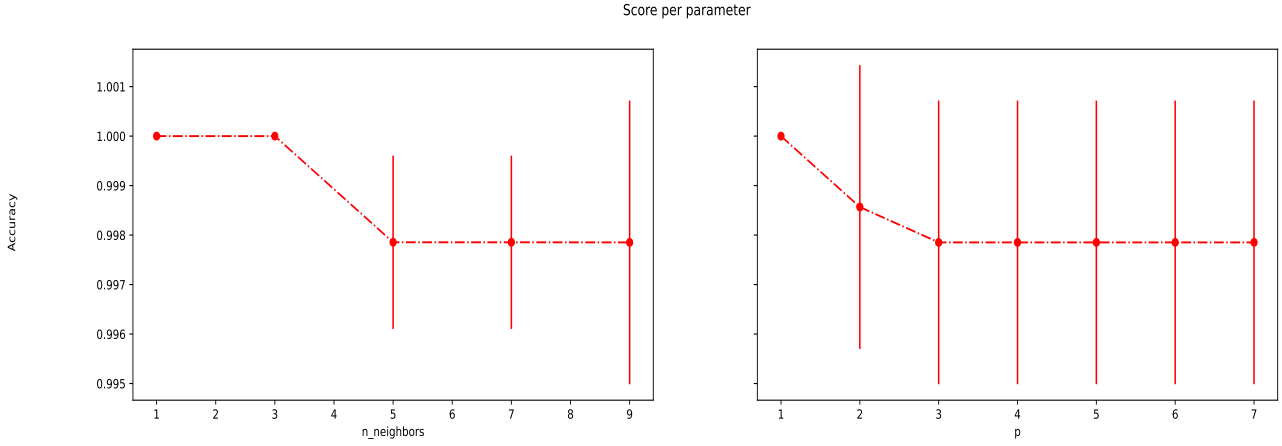
In Figure 2a, we notice that for $n\_neighbors = 1$ and $p = 2$ is the best hyperparameter values for Pair 1 (H and K) with an accuracy of 90%. In Figure 2b, $n\_neighbors = 3$ and $p = 2$ is the best hyperparameter values for Pair 2 (M and Y) with an accuracy of 99%. In figure 2c, $n\_neighbors = 5$ and $p = 1$ is the best hyperparameter values for Pair 3 (A and B) with an accuracy of 97%.




(a) Pair 1 (H and K)

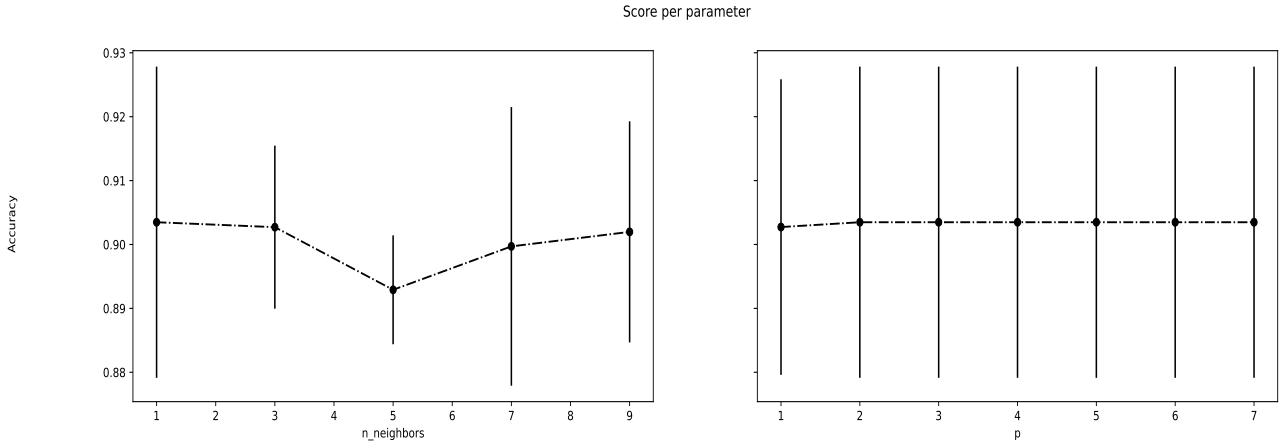



(b) Pair 2 (M and Y)

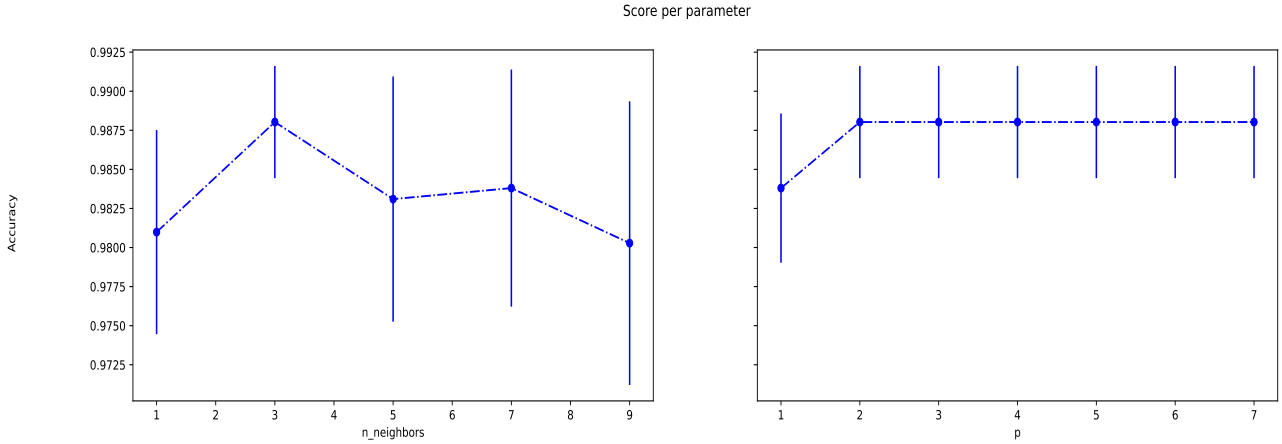



(c) Pair 3 (A and B)

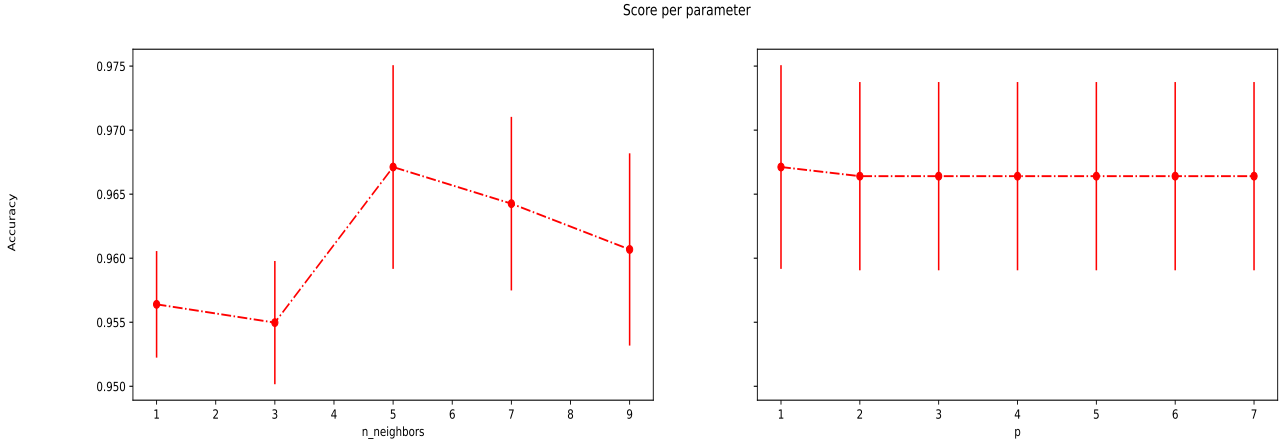
Figure 1: Cross validation results for KNN (Without dimension reduction)




(a) Pair 1 (H and K)




(b) Pair 2 (M and Y)




(c) Pair 3 (A and B)

Figure 2: Cross validation results for KNN (With dimension reduction)

## 2.2 Decision Tree

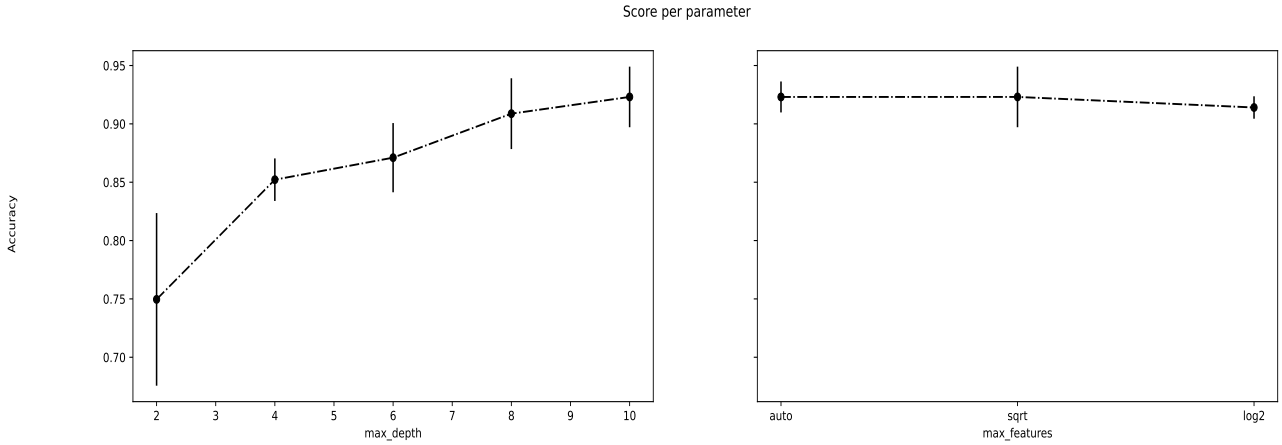
Decision Tree is a Supervised Machine Learning Algorithm that makes judgments based on a set of rules, similar to how people do. The idea behind Decision Trees is to build yes/no questions using dataset attributes and then partition the dataset until all data points belonging to each class are isolated. With this process we organize the data in a tree structure. The main advantages of Decision Tree is it requires less effort for data preparation during pre-processing compared to other classification algorithm. However, a slight change in the data might result in a significant change in the decision tree's structure, producing instability.

For Decision Tree, I have used $max\_depth$ and $max\_features$ as the hyperparameters. When looking for the split, the $max\_features$ is specified as the amount of features. For example, if $max\_features$ = sqrt, then I have calculated the $max\_features$ = sqrt(number of features). Figure 3 shows the hyperparameter tuning results for Decision Tree without dimension reduction.

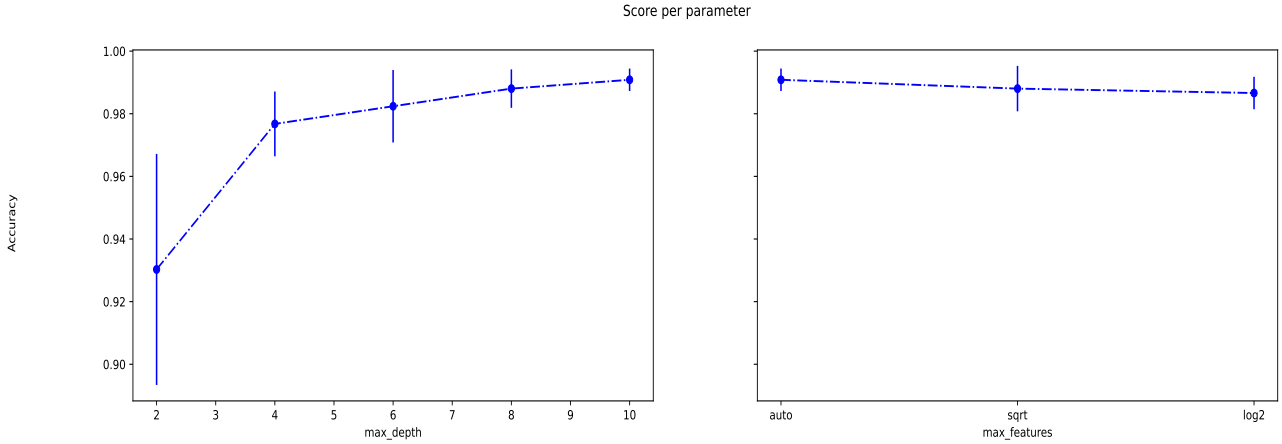
In figure 3a, we notice that for $max\_depth = 10$ and $max\_features = sqrt$ is the best hyperparameter values for Pair 1 (H and K) with an accuracy of 92%. In figure 3b, $max\_depth = 10$ and $max\_features = auto$ is the best hyperparameter values for Pair 2 (M and Y) with an accuracy of 99%. In Figure 3c, $max\_depth = 8$ and $max\_features = sqrt$ is the best hyperparameter values for Pair 3 (A and B) with an accuracy of 99%.

For dimension reduction, as like KNN, I have used the four features with highest variance in each set as values change very slightly across samples when features have a low variance. That is why, features with high variance are more crucial than the low variance features. Figure 4 shows the hyperparameter tuning results for Decision Tree with dimension reduction.

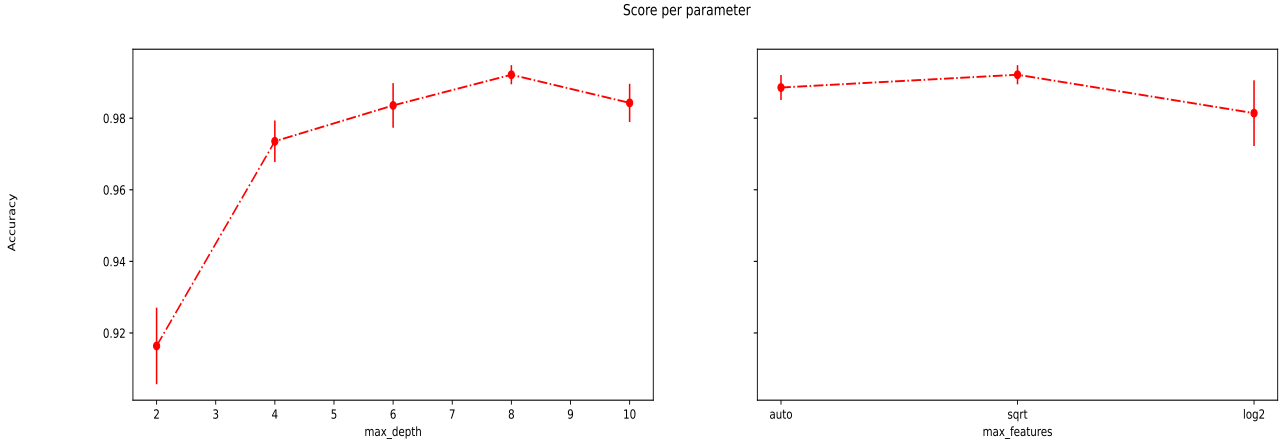
In Figure 4a, we notice that for $max\_depth = 10$ and $max\_features = auto$ is the best hyperparameter values for Pair 1 (H and K) with an accuracy of 90%. In Figure 4b, again $max\_depth = 10$ and $max\_features = auto$ is the best hyperparameter values for Pair 2 (M and Y) with an accuracy of 98%. In Figure 4c, $max\_depth = 10$ and $max\_features = log2$ is the best hyperparameter values for Pair 3 (A and B) with an accuracy of 96%.




(a) Pair 1 (H and K)

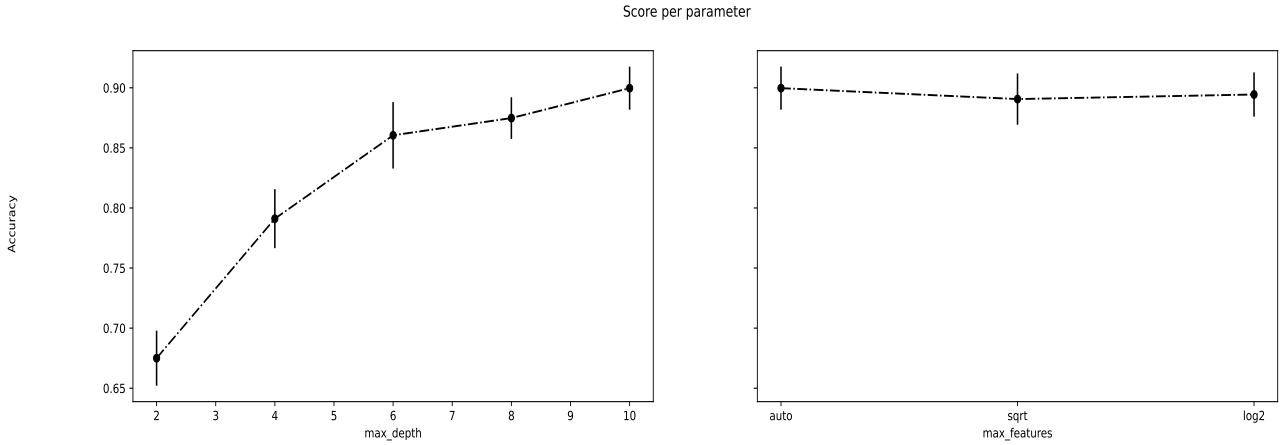



(b) Pair 2 (M and Y)

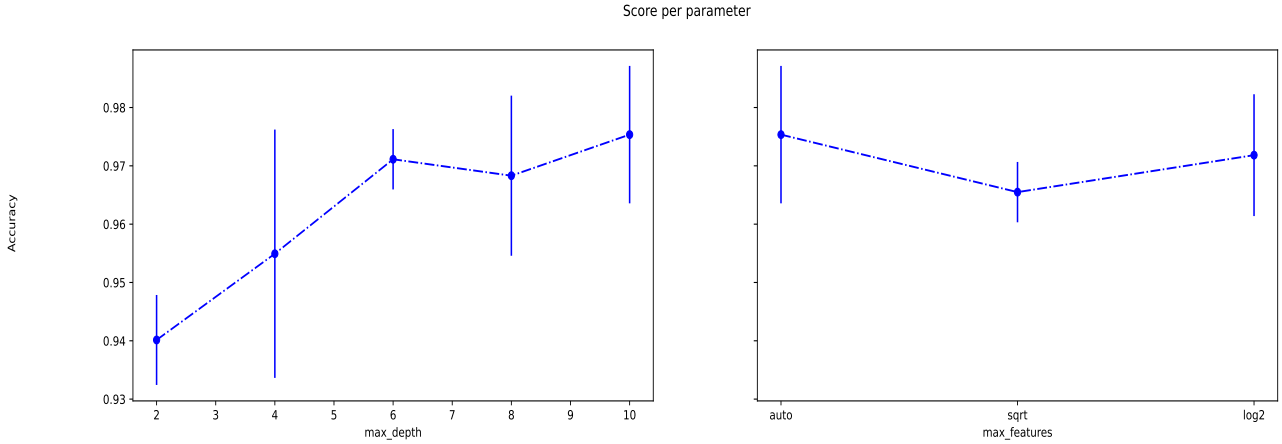



(c) Pair 3 (A and B)

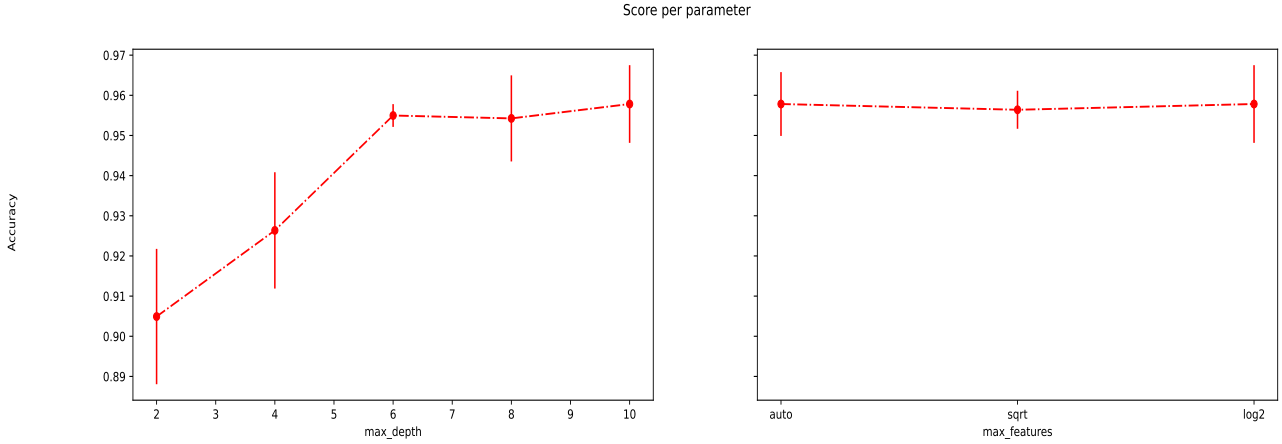
Figure 3: Cross validation results for Decision Tree (Without dimension reduction)




(a) Pair 1 (H and K)




(b) Pair 2 (M and Y)




(c) Pair 3 (A and B)

Figure 4: Cross validation results for Decision Tree (With dimension reduction)

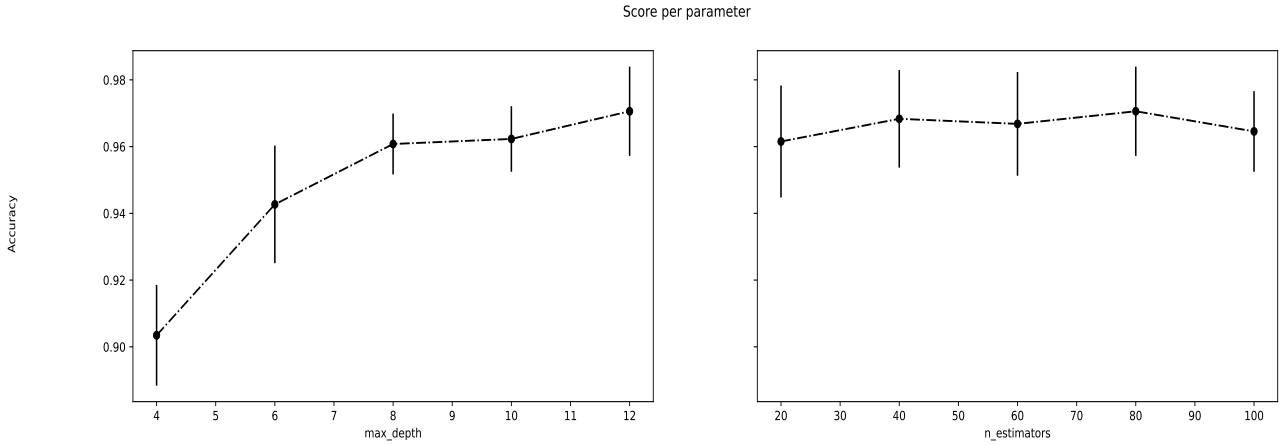
## 2.3 Random Forest

As the name indicates, a random forest is made up of a huge number of individual decision trees that work together as an ensemble. Each tree in the random forest produces a class prediction, and the class with the most votes becomes the prediction of our model. The reason for Random Forest to work so well is: a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. The main advantages of Random Forest is it can automatically handle missing values and it is usually robust to outliers and can handle them automatically. However, Random Forest creates a lot of trees and combines their outputs. To do so, this algorithm requires much more computational power and resources. For Random Forest, I have used $n\_estimators$ and $max\_depth$ as the hyperparameters. $n\_estimators$ is actually the number of trees in the forest and $max\_depth$ is the depth of a tree in the forest. Figure 5 shows the hyperparameter tuning results for Random Forest without dimension reduction.

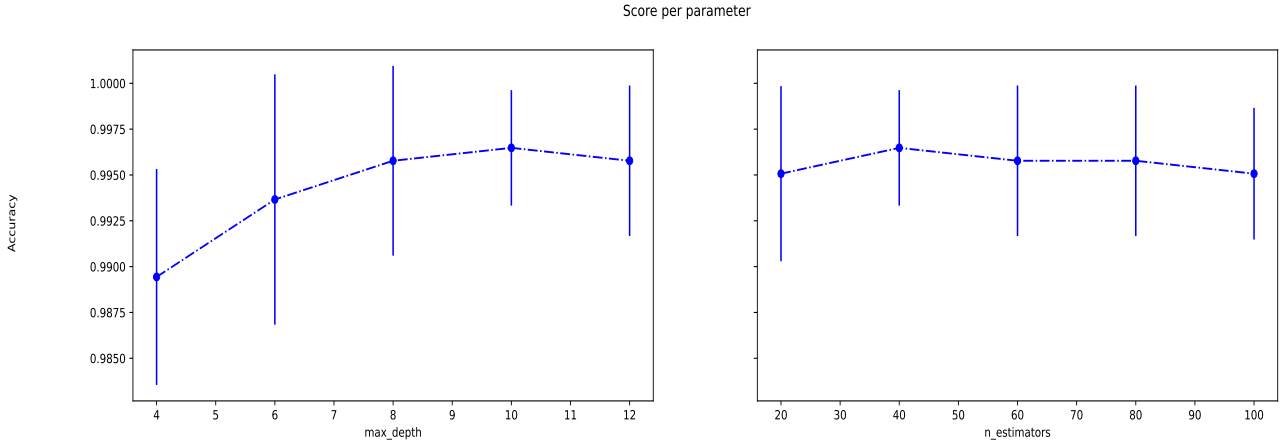
In Figure 5a, we notice that for $max\_depth = 12$ and $n\_estimators = 80$ is the best hyperparameter values for Pair 1 (H and K) with an accuracy of 97%. In Figure 5b, $max\_depth = 10$ and $n\_estimators = 40$ is the best hyperparameter values for Pair 2 (M and Y) with an accuracy of 99%. In Figure 5c, $max\_depth = 8$ and $n\_estimators = 40$ is the best hyperparameter values for Pair 3 (A and B) with an accuracy of 100%.

When training the first random forest model, I have kept the four features that are mostly utilized for splitting for dimension reduction. To find the features with most utilization for splitting, I have used the "permutation_importance" function from the sklearn.inspection library. Figure 6 shows the hyperparameter tuning results for Random Forest with dimension reduction.

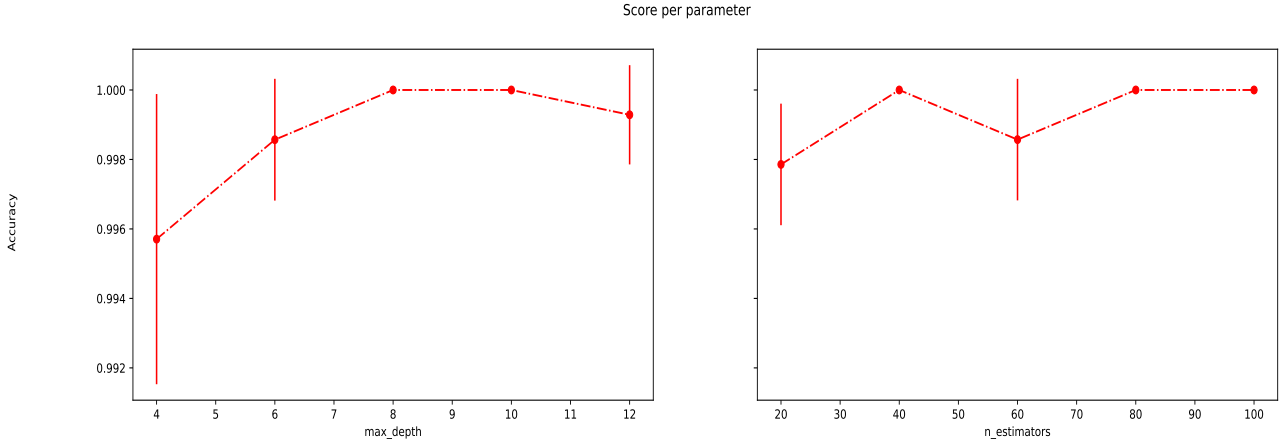
In Figure 6a, we notice that for $max\_depth = 12$ and $n\_estimators = 100$ is the best hyperparameter values for Pair 1 (H and K) with an accuracy of 92%. In Figure 6b, again $max\_depth = 4$ and $n\_estimators = 40$ is the best hyperparameter values for Pair 2 (M and Y) with an accuracy of 99%. In Figure 6c, $max\_depth = 6$ and $n\_estimators = 100$ is the best hyperparameter values for Pair 3 (A and B) with an accuracy of 99%.




(a) Pair 1 (H and K)

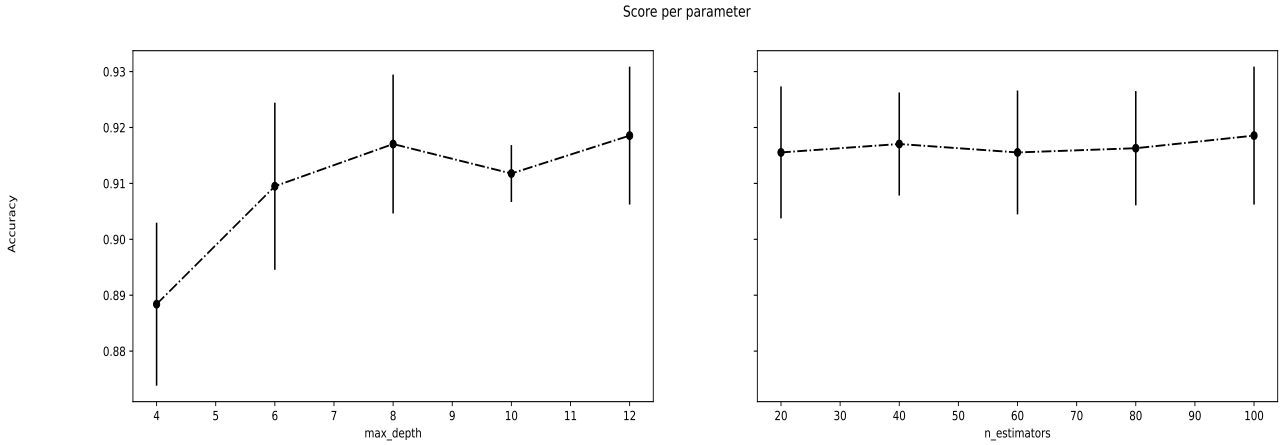



(b) Pair 2 (M and Y)

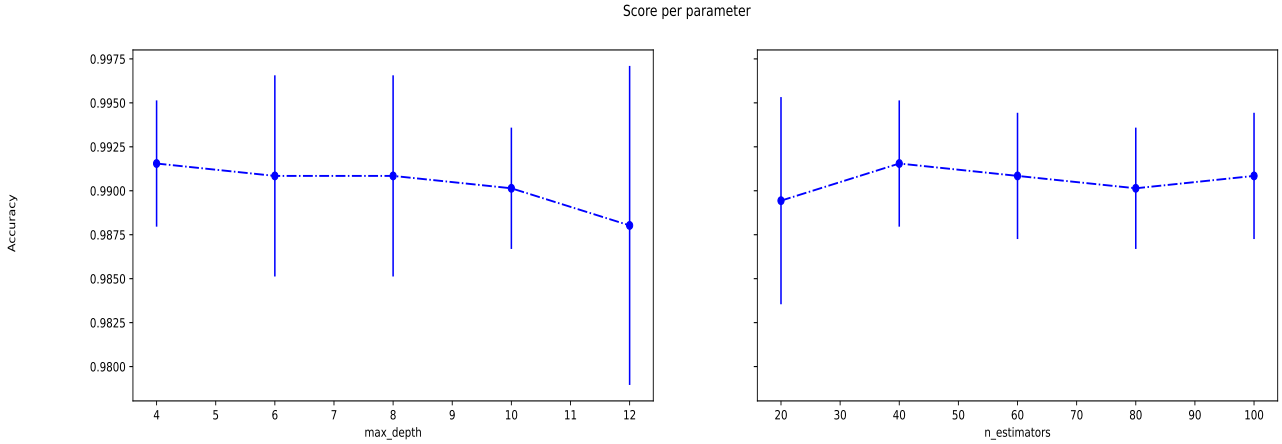



(c) Pair 3 (A and B)

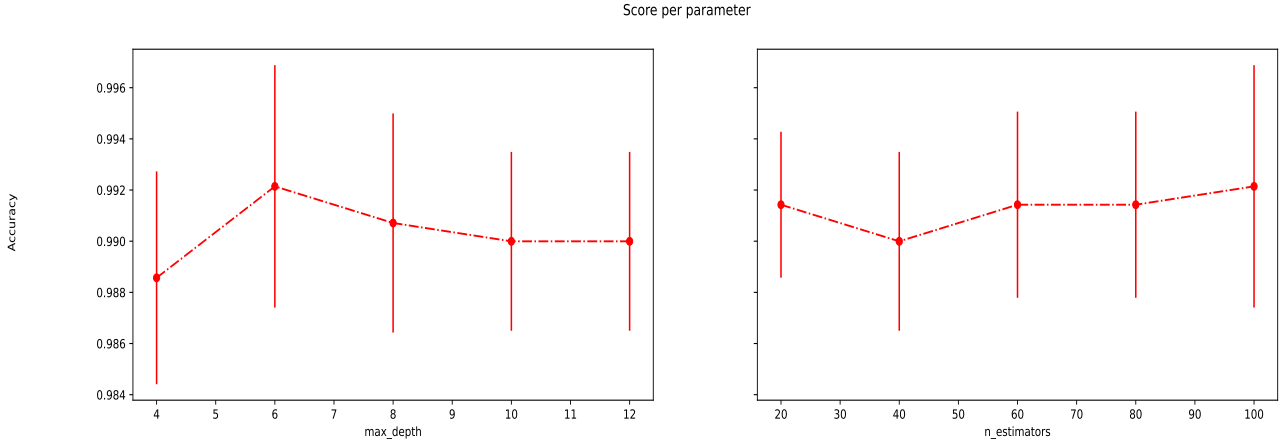
Figure 5: Cross validation results for Random Forest (Without dimension reduction)




(a) Pair 1 (H and K)




(b) Pair 2 (M and Y)




(c) Pair 3 (A and B)

Figure 6: Cross validation results for Random Forest (With dimension reduction)

## 2.4 Support Vector Machines (SVM)

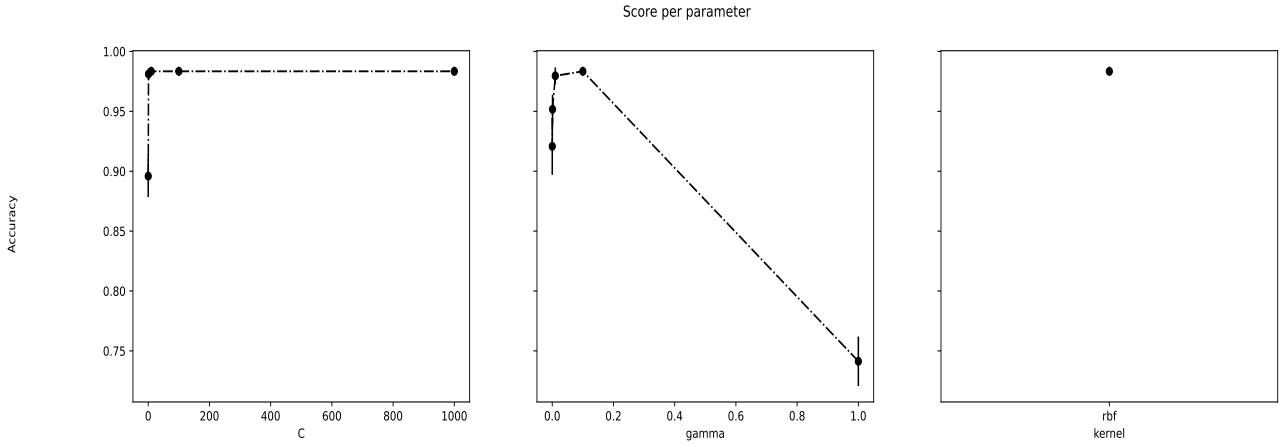
Support vector machines (SVM) is a supervised learning algorithm used for classification, regression and outliers detection. The support vector machine algorithm's goal is to identify a hyperplane in an N-dimensional space (N = the number of features) that distinguishes between data points. There are several hyperplanes from which to choose to split the two kinds of data points. In SVM, our goal is to discover a plane with the greatest margin, or the greatest distance between data points from both classes. When there is a clear margin of distinction between classes, SVM performs very effectively. SVM is also more effective in high dimensional datasets. However, SVM does not work well when the dataset is very large and the dataset has more noise.

For SVM, I have used *gamma* and $C$ as the hyperparameters. $C$ is actually called the regularization parameter and the strength of the regularization is inversely proportional to $C$. On the other hand, the *gamma* defines the level of curvature we want in the decision boundary. Figure 7 shows the hyperparameter tuning results for SVM without dimension reduction.

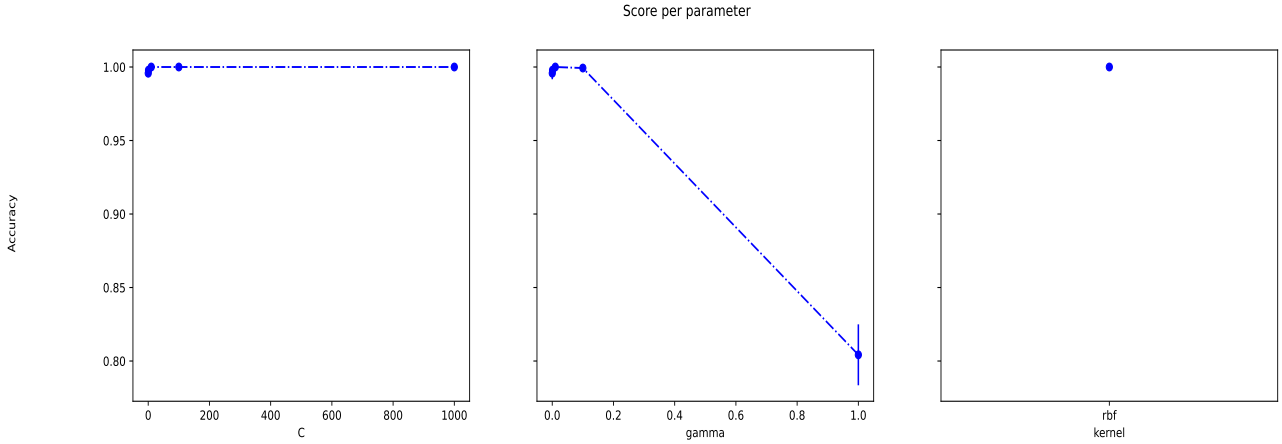
In Figure 7a, we notice that for $C = 10$ and $gamma = 0.1$ is the best hyperparameter values for Pair 1 (H and K) with an accuracy of 98%. In Figure 7b, $C = 10$ and $gamma = 0.01$ is the best hyperparameter values for Pair 2 (M and Y) with an accuracy of 100%. In Figure 7c, $C = 1$ and $gamma = 0.1$ is the best hyperparameter values for Pair 3 (A and B) with an accuracy of 99%.

For dimension reduction, as like KNN and Decision Tree, I have used the four features with highest variance in each set as values change very slightly across samples when features have a low variance. Figure 8 shows the hyperparameter tuning results for SVM with dimension reduction.

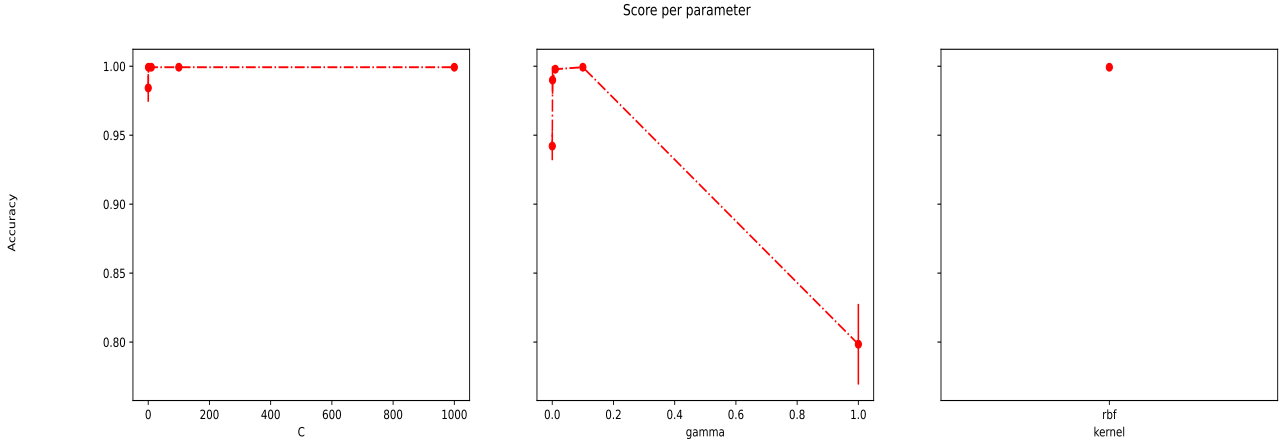
In Figure 8a, we notice that for $C = 10$ and $gamma = 1$ is the best hyperparameter values for Pair 1 (H and K) with an accuracy of 91%. In Figure 8b, again $C = 10$ and $gamma = 0.1$ is the best hyperparameter values for Pair 2 (M and Y) with an accuracy of 98%. In Figure 8c, $C = 10$ and $gamma = 0.1$ is the best hyperparameter values for Pair 3 (A and B) with an accuracy of 97%.




(a) Pair 1 (H and K)

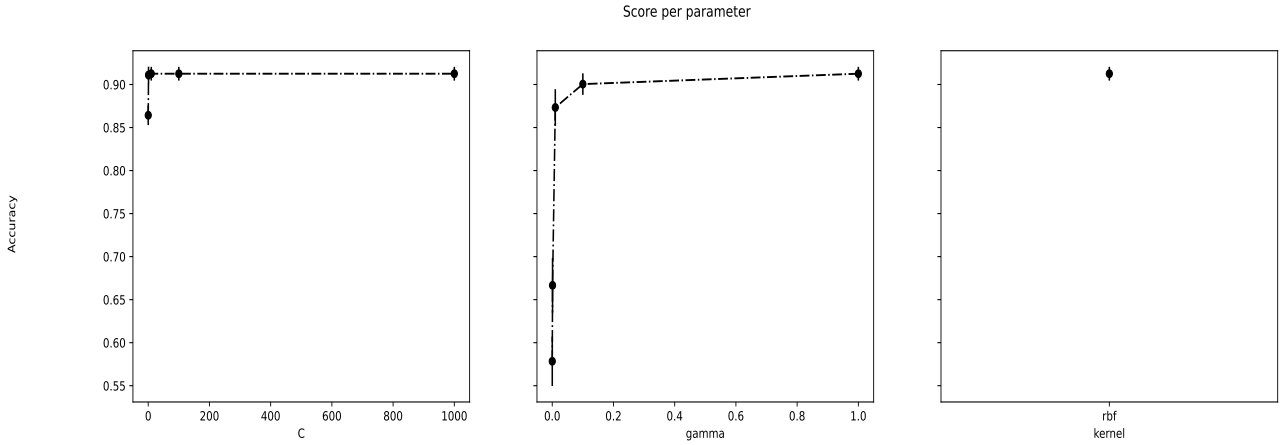



(b) Pair 2 (M and Y)

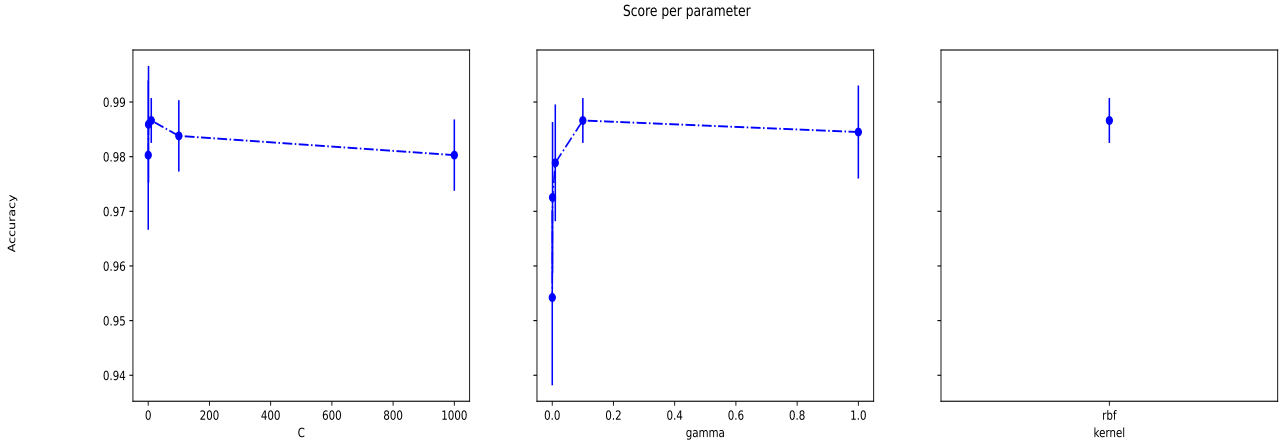



(c) Pair 3 (A and B)

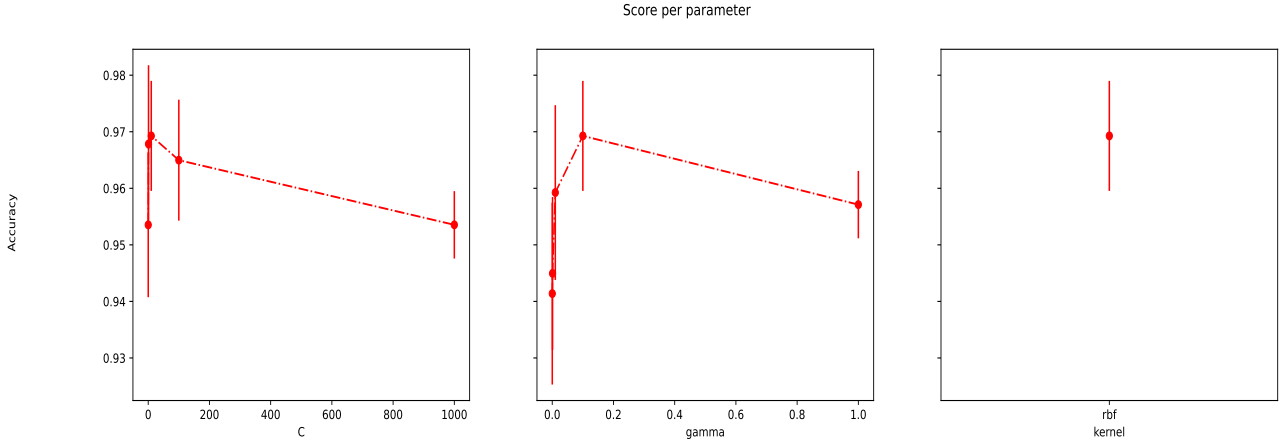
Figure 7: Cross validation results for SVM (Without dimension reduction)




(a) Pair 1 (H and K)




(b) Pair 2 (M and Y)




(c) Pair 3 (A and B)

Figure 8: Cross validation results for SVM (With dimension reduction)

## 2.5 Artificial Neural Network (ANN)

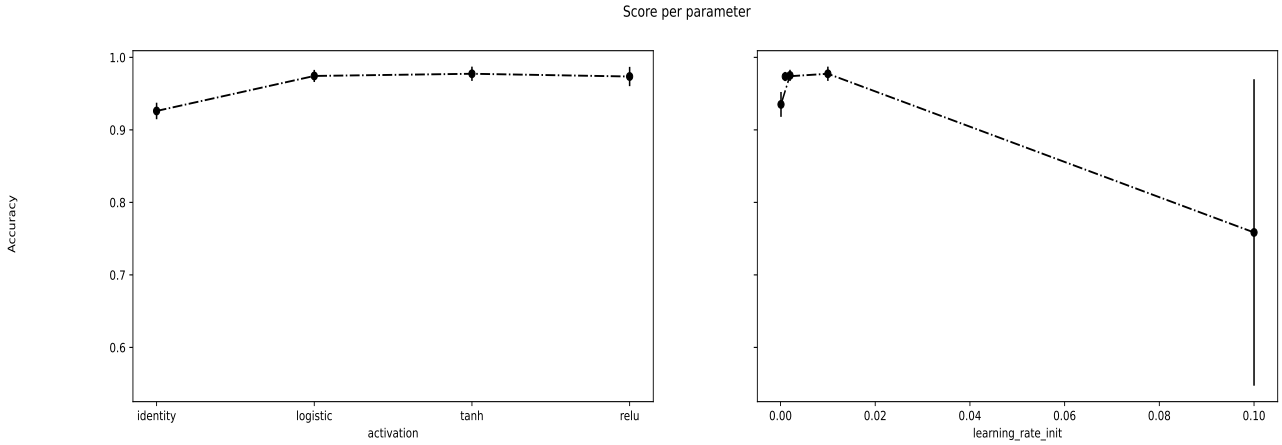
Artificial Neural Network (ANN) is a biologically inspired computational network. Artificial neurons are a set of linked units or nodes in an ANN that loosely replicate the neurons in a biological brain. The main advantages of ANN is its learning method is quite robust to noise in the training data. The training examples may contain errors, which do not affect the final output. However, ANN requires processors with parallel processing power and problems have to be translated into numerical values before being introduced to ANN.

For ANN, I have used *activation* and *learning_rate_init* as the hyperparameters. The *activation* function aids the ANN in learning complicated data patterns. On the other hand, while progressing towards the loss function's minimum, the *learning_rate_init* sets the step size at each iteration. Figure 9 shows the hyperparameter tuning results for ANN without dimension reduction.

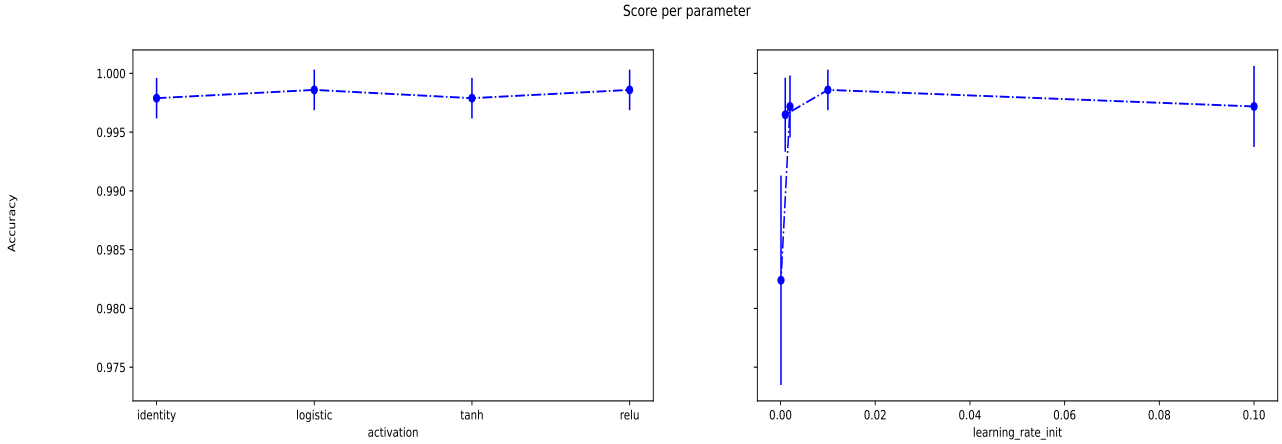
In Figure 9a, we notice that for *activation* = *tanh* and *learning_rate_init* = 0.01 is the best hyperparameter values for Pair 1 (H and K) with an accuracy of 98%. In Figure 9b, *activation* = *logistic* and *learning_rate_init* = 0.01 is the best hyperparameter values for Pair 2 (M and Y) with an accuracy of 100%. In Figure 9c, *activation* = *tanh* and *learning_rate_init* = 0.002 is the best hyperparameter values for Pair 3 (A and B) with an accuracy of 100%.

For dimension reduction, I have used the Principal Component Analysis (PCA) for ANN. PCA is a method for lowering the dimensionality of such datasets, improving interpretability while minimizing information loss. It is a projection-based method that converts data by projecting it onto orthogonal axes. It essentially turns correlated variables in a high-dimensional space into uncorrelated variables in a low-dimensional space, where the uncorrelated features are known as principle components. Figure 10 shows the hyperparameter tuning results for ANN with dimension reduction.

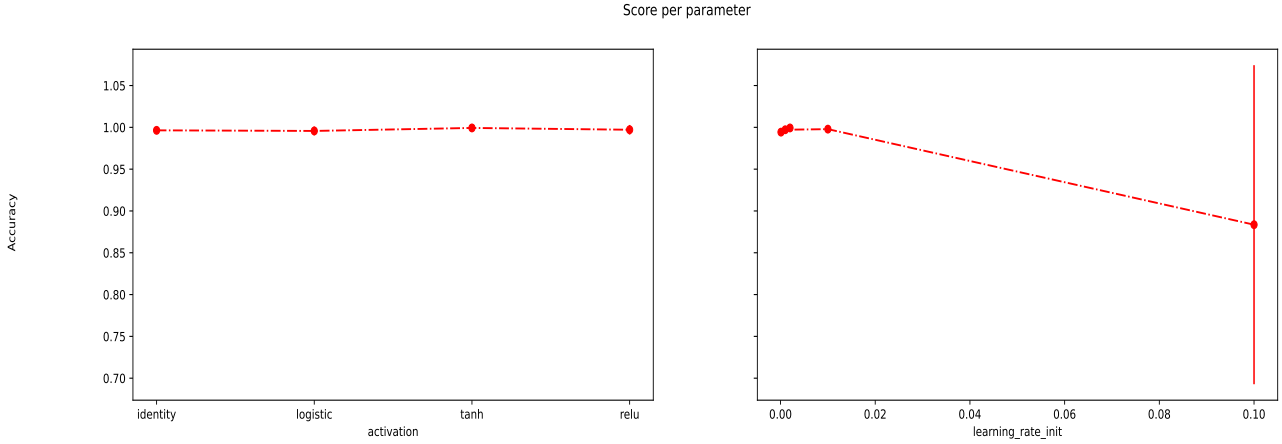
In Figure 10a, we notice that for *activation* = *logistic* and *learning_rate_init* = 0.01 is the best hyperparameter values for Pair 1 (H and K) with an accuracy of 90%. In Figure 10b, again *activation* = *tanh* and *learning_rate_init* = 0.01 is the best hyperparameter values for Pair 2 (M and Y) with an accuracy of 99%. In Figure 10c, *activation* = *relu* and *learning_rate_init* = 0.1 is the best hyperparameter values for Pair 3 (A and B) with an accuracy of 99%.




(a) Pair 1 (H and K)

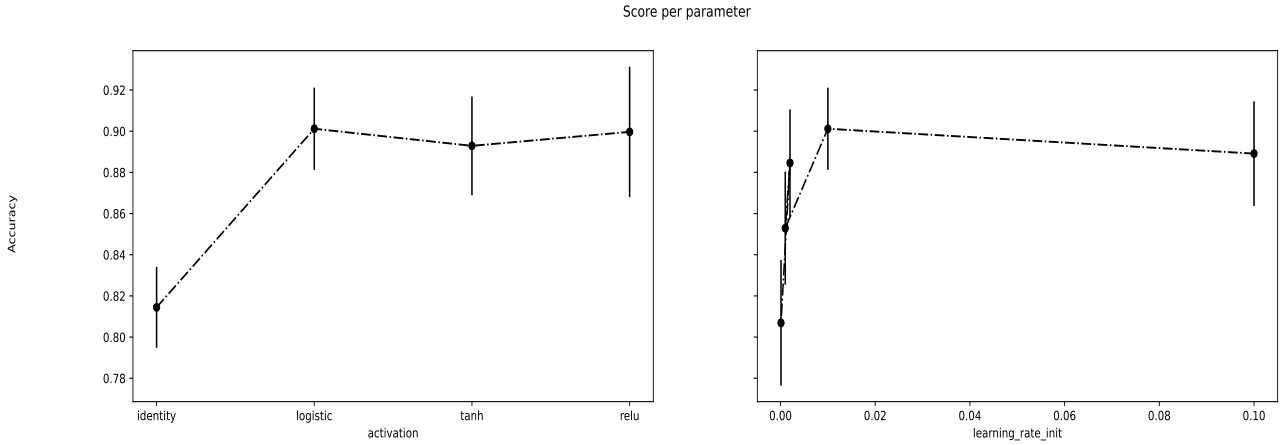



(b) Pair 2 (M and Y)

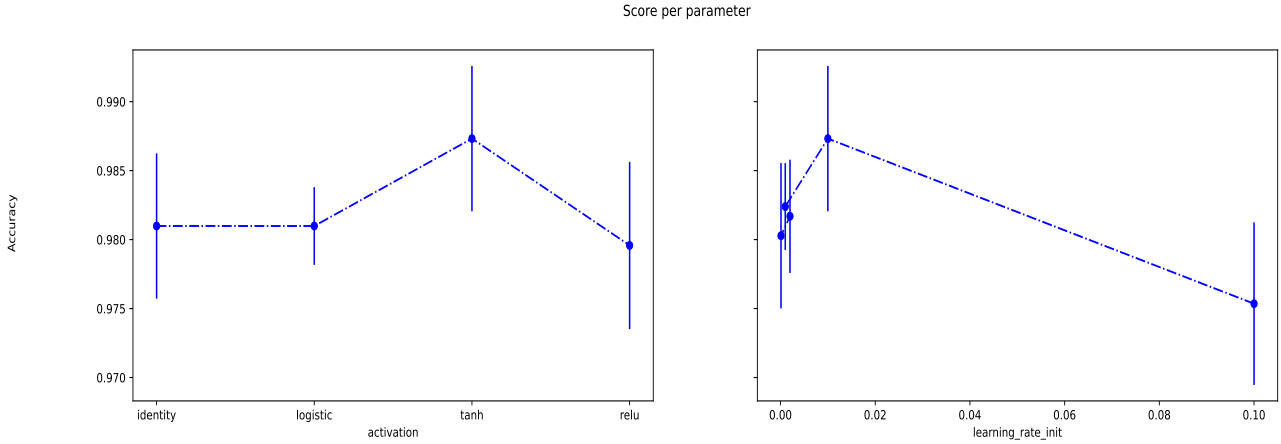



(c) Pair 3 (A and B)

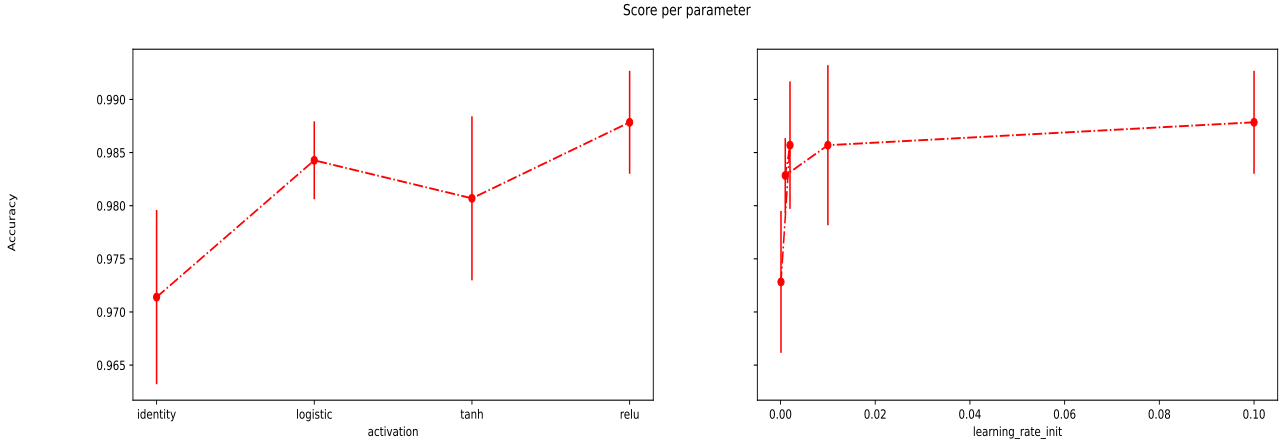
Figure 9: Cross validation results for ANN (Without dimension reduction)




(a) Pair 1 (H and K)




(b) Pair 2 (M and Y)




(c) Pair 3 (A and B)

Figure 10: Cross validation results for ANN (With dimension reduction)

## 2.6 Additional Classifier 1 - Naive Bayes

A Naive Bayes classifier is a type of probabilistic machine learning model that is used to perform classification tasks. The classifier's crux is based on the Bayes theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

In other words, a Naive Bayes classifier assumes that the existence of one feature in a class is independent to the presence of any other feature. The Naive Bayes model is simple to construct and is especially good for huge data sets. Naive Bayes is renowned to outperform even the most advanced classification methods due to its simplicity. However, Naive Bayes assumes that all features are independent, rarely happening in real life. This limits the applicability of this algorithm in real-world use cases.

For Naive Bayes, I have used *var_smoothing* as the hyperparameter. The *var_smoothing* is a stability function that smooths the curve and includes more samples that are further from the distribution mean. Figure 11 shows the hyperparameter tuning results for Naive Bayes without dimension reduction.

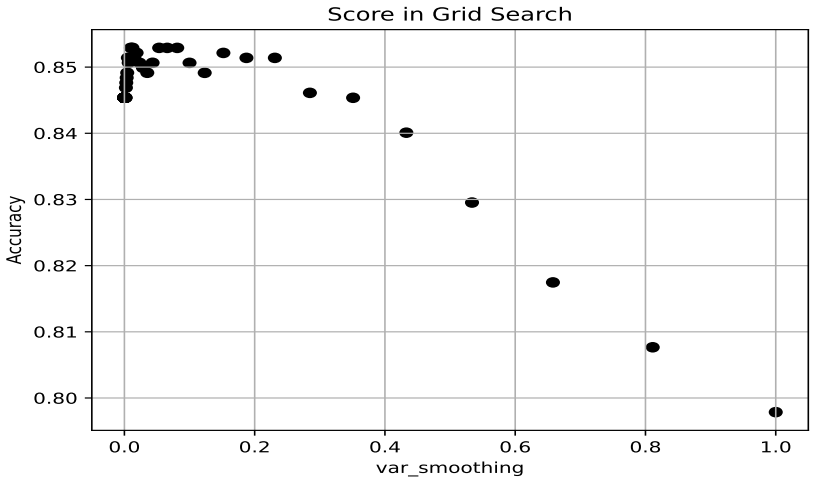
In Figure 11a, we notice that for $var\_smoothing = 0.01$ is the best hyperparameter values for Pair 1 (H and K) with an accuracy of 85%. In Figure 11b, $var\_smoothing = 0.0285$ is the best hyperparameter values for Pair 2 (M and Y) with an accuracy of 97%. In Figure 11c, $var\_smoothing = 0.0123$ is the best hyperparameter values for Pair 3 (A and B) with an accuracy of 95%.

## 2.7 Additional Classifier 2 - Logistic Regression

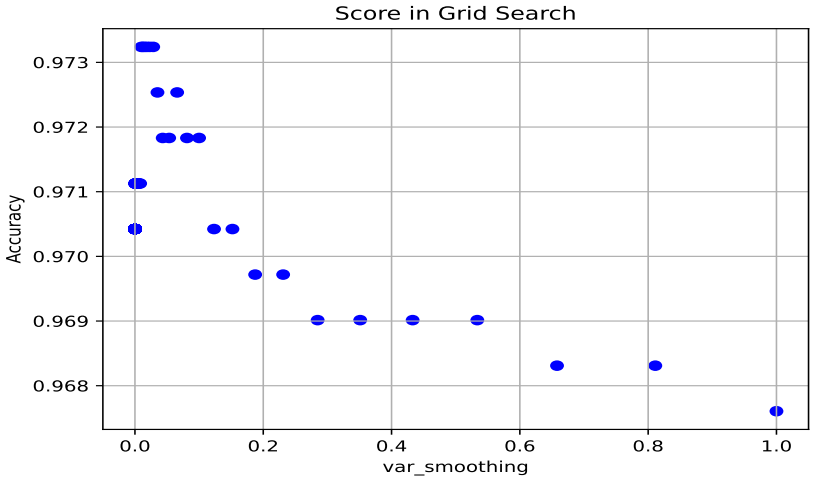
Logistic regression is a machine learning classification approach. The dependent variable is modeled using a logistic function. The dependent variable is dichotomous, which means that only two classifications are conceivable. As a result, while working with binary data, this technique is used. Logistic regression is very easier to implement, interpret, and very efficient to train. It makes no assumptions about distributions of classes in feature space. However, the assumption of linearity between the dependent and independent variables is a major limitation of Logistic Regression.Also, logistic Regression may result in overfitting if the number of observations is smaller than the number of features.

For Logistic regression, I have used *solver* and $C$ as the hyperparameters. The algorithm utilized in the optimization problem is referred to as a *solver*. "Liblinear" is commonly used for smaller datasets. "newton-cg" and "lbfgs", on the other hand, deal with multiclass problems. Figure 12 shows the hyperparameter tuning results for Logistic Regression without dimension reduction.

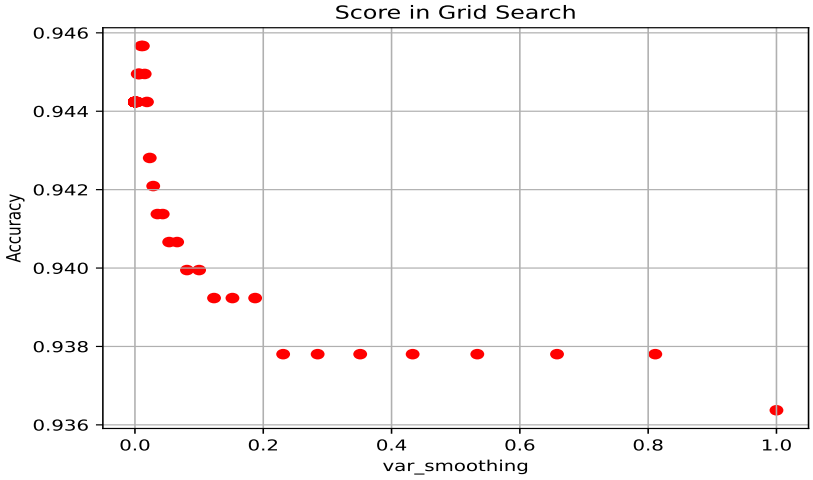
In Figure 12a, we notice that for $solver = newton - cg$ and $C = 0.1$ is the best hyperparameter values for Pair 1 (H and K) with an accuracy of 93%. In Figure 12b, $solver = liblinear$ and $C = 0.1$ is the best hyperparameter values for Pair 2 (M and Y) with an accuracy of 100%. In Figure 12c, $solver = newton - cg$ and $C = 0.1$ is the best hyperparameter values for Pair 3 (A and B) with an accuracy of 100%.




(a) Pair 1 (H and K)

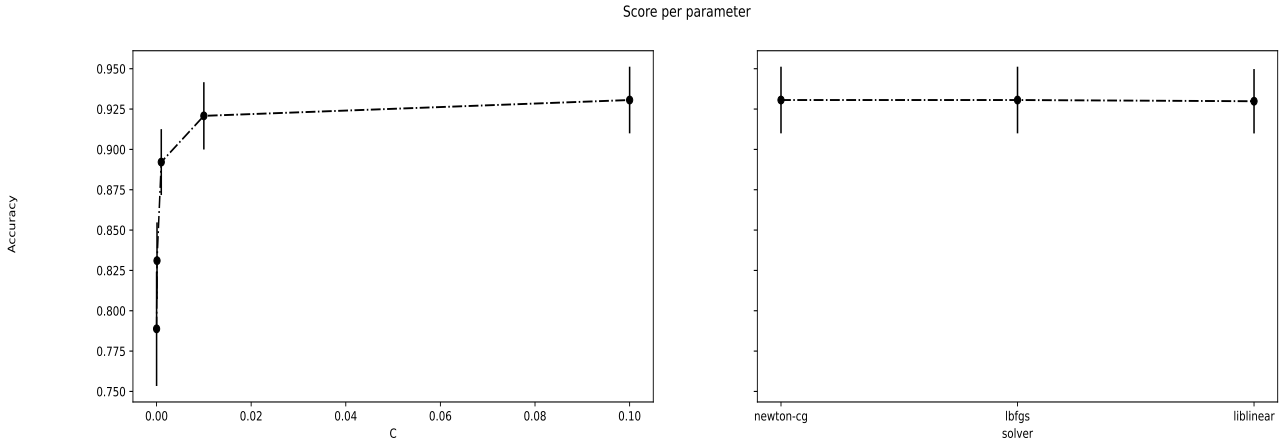



(b) Pair 2 (M and Y)

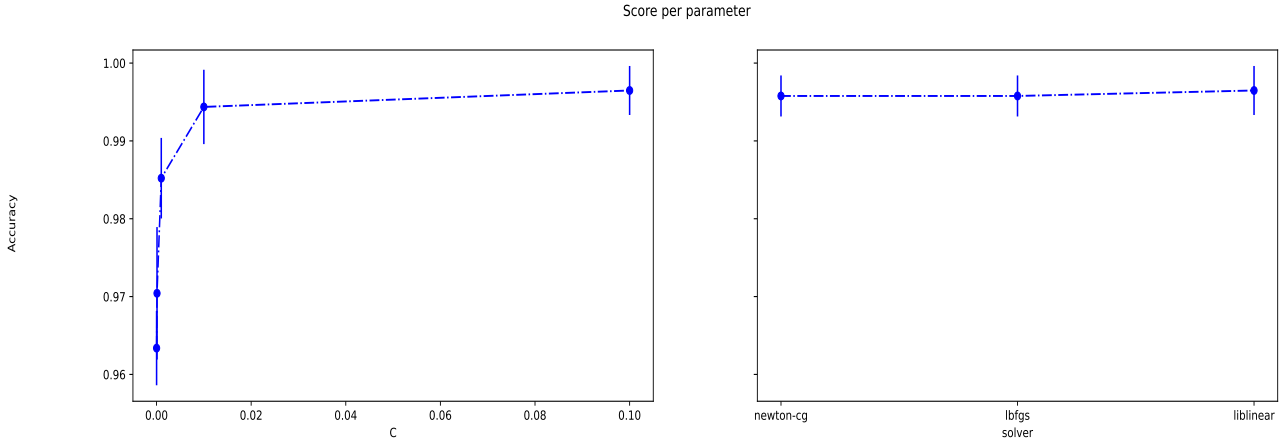



(c) Pair 3 (A and B)

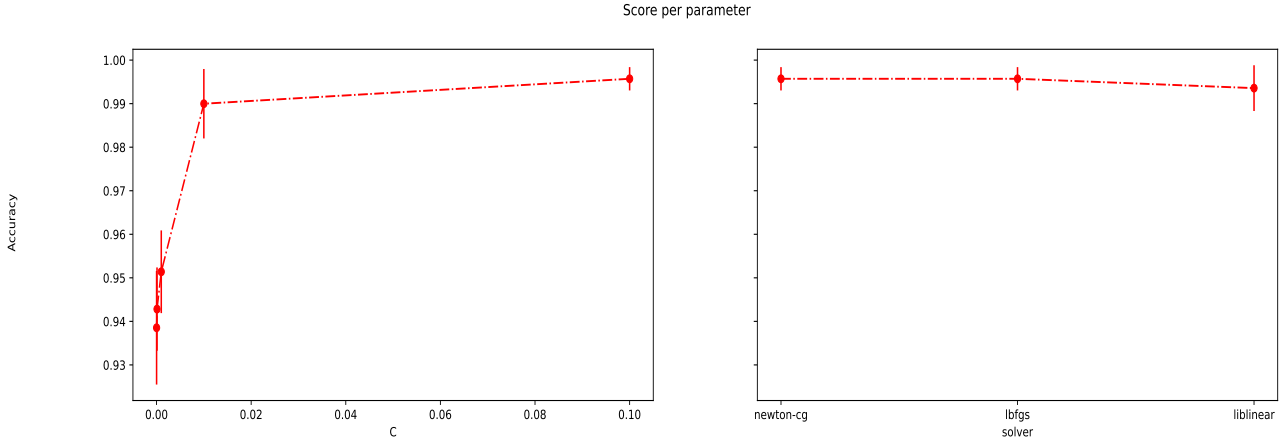
Figure 11: Cross validation results for Naive Bayes (Without dimension reduction)




(a) Pair 1 (H and K)




(b) Pair 2 (M and Y)




(c) Pair 3 (A and B)

Figure 12: Cross validation results for Logistic Regression (Without dimension reduction)

# 3 Discussion

For testing the trained models, I have made the validation dataset with 10% data from the actual dataset. Table 1 and Table 2 shows the performance of each model and the run time.

Table 1: Accuracy and Run Time of the classification models (without dimension reduction)

| Model | Pair 1 (H and K) | | Pair 2 (M and Y) | | Pair 3 (A and B) | |
|---|---|---|---|---|---|---|
| | Accuracy (%) | Run Time (ms) | Accuracy (%) | Run Time (ms) | Accuracy (%) | Run Time (ms) |
| K-Nearest Neighbors | 93.20 | 6.85 | 100 | 6.41 | 100 | 7.03 |
| Decision Tree | 92.52 | 1.29 | 98.73 | 1.06 | 97.44 | 1.00 |
| Random Forest | 97.96 | 8.92 | 99.37 | 7.79 | 100 | 6.98 |
| Support Vector Machine | 98.64 | 11.67 | 100 | 2.15 | 100 | 8.85 |
| Artificial Neural Network | 98.64 | 1.32 | 100 | 1.59 | 100 | 1.84 |
| Logistic Regression | 93.88 | 1.20 | 99.37 | 1.46 | 98.08 | 1.39 |
| Naive Bayes | 87.07 | 1.28 | 94.94 | 1.18 | 96.15 | 1.08 |

Table 2: Accuracy and Run Time of the classification models (with dimension reduction)

| Model | Pair 1 (H and K) | | Pair 2 (M and Y) | | Pair 3 (A and B) | |
|---|---|---|---|---|---|---|
| | Accuracy (%) | Run Time (ms) | Accuracy (%) | Run Time (ms) | Accuracy (%) | Run Time (ms) |
| K-Nearest Neighbors | 90.48 | 5.62 | 98.10 | 5.83 | 96.79 | 6.11 |
| Decision Tree | 91.84 | 1.02 | 96.20 | 0.90 | 96.79 | 0.91 |
| Support Vector Machine | 93.88 | 4.99 | 99.37 | 3.60 | 95.51 | 3.41 |
| Artificial Neural Network | 51.70 | 0.62 | 59.49 | 0.46 | 98.72 | 0.58 |
| Random Forest | 93.88 | 10.07 | 99.37 | 2.84 | 98.72 | 4.61 |

From Table 1, we can see that accuracy of several models for Pair 2 (M and Y) and Pair 3 (A and B) is 100%. This is logical because the size of the validation set is only 10% of the size of the actual dataset. For this reason, the models can accurately classify the data. After analyzing the performance on validation set, I believe Support Vector Machine (SVM) is the best classifier for this problem. This is because SVM has classified the dataset more accurately than the other classifiers. For both with and without dimension reduction, SVM performs the best for all three pairs. The run time is also reasonable for SVM in all the scenarios.

In Table 2, we can see that the accuracy drops for all the classifiers. This is also logical as we have reduced the dimension of the dataset from 16 to 4. Also, the run time has decreased for all the classifiers after reducing the dimensionality. The key thing to notice here is that the accuracy of ANN drops significantly when we reduce the dimensionality. For the other classifiers, accuracy does not drop significantly if we reduce the dimension.

For a new dataset, I will first check the accuracy of the models after the dimension reduction. As the reduced dataset will take less amount of run time, if we get a good accuracy then we don't need to use the actual dataset for the classification problem. Also, for a new dataset I will try to tune more hyperparameters and will try different dimension reduction approaches.