Data Science Lab Homework 4

Group 13

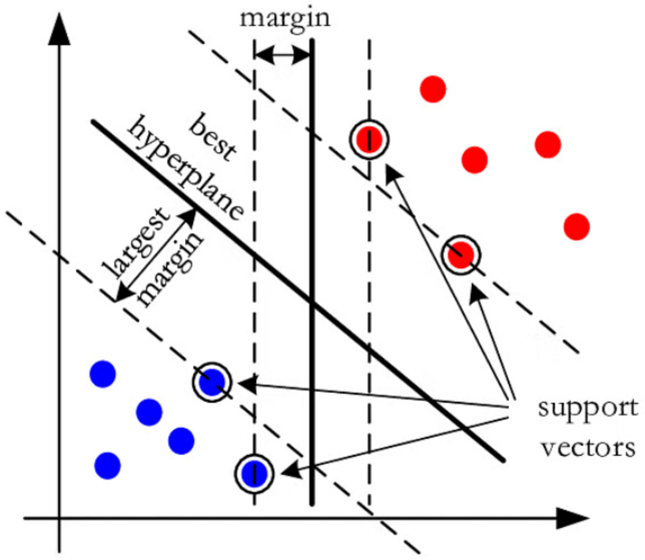
1 Question

Download Heart Disease data from https://archive.ics.uci.edu/datasets and apply depth based classifier, SVM based classifier, K-NN based classifier (choose K with proper justification), Kernel density function based classifier on the aforesaid two data sets and compute empirical miss-classification probability for each classifier.

2 Introduction

2.1 SVM-based Classifier

A Support Vector Machine (SVM) algorithm is a non-probabilistic classifier aiming to generate hyperplanes that divide the data points of two classes in the vector space. It is a supervised machine learning algorithm used for classification and regression tasks. It's particularly effective in high-dimensional spaces and in cases where the number of dimensions is greater than the number of samples. The image below shows how "support" vectors are calculated such that the margin (or distance) between the vectors of two classes is the most. SVM optimizes this margin metric to find the best hyperplane for all the categories that's why SVMs are popular for disease prediction since they can effectively categorize tabular data into different categories.



2.2 Depth-based Classifier

Depth-based classifiers, such as decision trees and random forests, are powerful machine learning algorithms commonly used for both classification and regression tasks. Decision trees recursively partition the feature space based on individual feature values, providing an interpretable sequence of if-else conditions. However, they can over-fit when the tree grows too deep. In contrast, random forests are ensembles of decision trees trained on random subsets of the data and features, mitigating overfitting by aggregating predictions. Both algorithms find applications in classification and regression tasks, with decision trees being interpretable and random forests offering enhanced generalization capabilities.

2.3 Kernel Density Function-based Classifier

Kernel density-based classifiers are non-parametric techniques used for density estimation and classification tasks. In classification, kernel density estimation is used to model the conditional probability distributions of different classes, and the class of a new data point is determined based on which class has the highest density at that point. Kernel density-based classifiers are particularly useful when dealing with data distributions that are not easily modeled by parametric methods and can handle complex, multi-modal data distributions.

2.4 KNN-based Classifier

The k-Nearest Neighbors (k-NN) algorithm is a versatile machine learning technique used for classification and regression tasks. It determines the class or predicts the value of a new data point by examining the labels or values of its k nearest neighbors in the feature space, based on a distance metric such as Euclidean distance. While simple to implement and understand, k-NN's computational cost can be high for large datasets, and the choice of the parameter k is crucial for its performance.

3 Comparing Different Machine Learning Models

Comparing different machine learning models involves evaluating their performance using various metrics and then selecting the model that performs the best according to your criteria. We'll be making confusion matrix for all the models and calculate it's accuracy, precision, recall and F1 score.

- **Accuracy**: Accuracy measures the proportion of correct predictions among the total number of predictions. It gives an overall measure of correctness but may not be suitable for imbalanced datasets. It's calculated as:

$$Accuracy = \frac{True\ Positives + True\ Negatives}{True\ Positives + True\ Negatives + False\ Positives + False\ Negatives}$$

- **Precision**: Precision measures the proportion of true positive predictions among the total number of positive predictions. It provide insights into how well the model identifies positive instances and avoids false positives. It's calculated as:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

- **Recall(Sensitivity)** Recall measures the proportion of true positive predictions among the total number of actual positive instances. It provide insights into how well the model identifies positive instances and avoids false negatives. It's calculated as:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

- **F1 Score**: The F1 score is the harmonic mean of precision and recall. It provides a single score that balances both precision and recall. It balances precision and recall and can be useful when you want to consider both false positives and false negatives. It's calculated as:

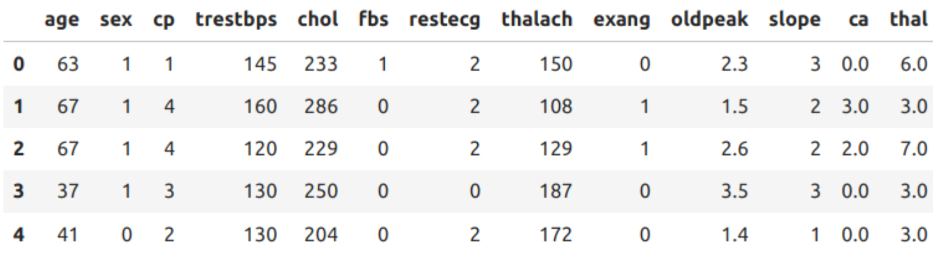
$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

As in our case we want to minimize chances of a false positives as they can lead to unnecessary anxiety, additional medical tests, treatments, and healthcare costs for patients who do not have the disease. So we are gonna prioritize on increasing precision along with accuracy.

4 Different Machine Learning Approaches on Heart Disease Dataset

To test out the how different models perform on this task, we will be using the **UCI Heart Disease** having 14 features and 303 samples.

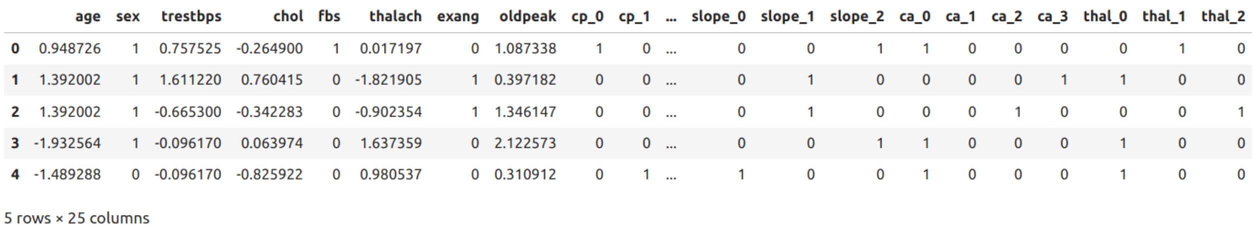
We load our dataset and first of all we check for any missing values, as there are missing values present in this dataset we'll replace all the missing values of both features(having missing values) by their respective mode values.
The first few samples of dataset are as follows:



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0
3	37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0

We see that some of them are categorical variables. The basics of feature engineering and data science tell us that such columns need to be encoded to avoid unintentional bias. For example, as shown below, columns like chest pain (cp), restecg, slope, ca and thal need to be one-hot encoded in addition to others. Moreover, the columns of age, cholesterol (chol), Rest BP (trestbps), thalach, and oldpeak need to be normalized.

After applying one hot-encoding and normalization the dataset looks like the below:



	age	sex	trestbps	chol	fbs	thalach	exang	oldpeak	cp_0	cp_1	...	slope_0	slope_1	slope_2	ca_0	ca_1	ca_2	ca_3	thal_0	thal_1	thal_2
0	0.948726	1	0.757525	-0.264900	1	0.017197	0	1.087338	1	0	...	0	0	1	1	0	0	0	0	1	0
1	1.392002	1	1.611220	0.760415	0	-1.821905	1	0.397182	0	0	...	0	1	0	0	0	0	1	1	0	0
2	1.392002	1	-0.665300	-0.342283	0	-0.902354	1	1.346147	0	0	...	0	1	0	0	0	1	0	0	0	1
3	-1.932564	1	-0.096170	0.063974	0	1.637359	0	2.122573	0	0	...	0	0	1	1	0	0	0	1	0	0
4	-1.489288	0	-0.096170	-0.825922	0	0.980537	0	0.310912	0	1	...	1	0	0	1	0	0	0	1	0	0

5 rows × 25 columns

Then we trained our various machine learning models on this normalized dataset with a train-test split of 80% to 20% of original samples respectively.

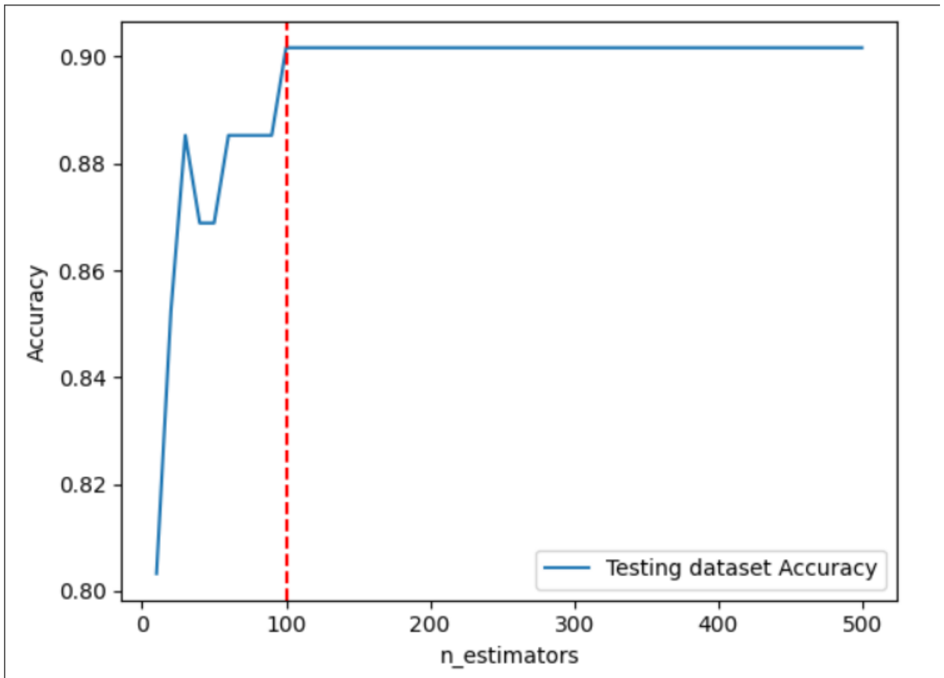
4.1 SVM-based Classifier

The svm.SVC function in sklearn has several hyperparameters. We explored the primary ones i.e. choice of kernel(linear, poly, rbf and sigmoid), regularization parameter, and degree of the polynomial kernel and we found that the best kernel choice is 'RBF', with regularization parameter being 2.0. This choice of hyperparameters gives 88.5% accuracy and 90.3% precision on the test set.

```
svm_classifier = svm.SVC(kernel='rbf', C=2)
svm_classifier.fit(X_train, y_train)

out = svm_classifier.predict(X_test)

test_accuracy = accuracy_score(y_test, out)
print('test_accuracy = ', test_accuracy)
# 0.8852459016393442

test_precision = precision_score(y_test, out)
print(test_precision)
# 0.9032258064516129
```

4.2 Depth-based Classifier

For Depth based classifer we used Random Forest Classifier, and calculated the accuracy on test set for various values of the n estimators parameter on tree level.



```
Best choice of n_estimators: 100
```

With the choice of 100 estimators, our random forest classifier gives around 90.1% accuracy and 93.3% precision on the test set.

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
print(rf_classifier.score(X_test, y_test))
# 0.9016393442622951

pred = rf_classifier.predict(X_test)
print(precision_score(y_test, pred))
# 0.9333333333333333
```

4.3 Kernel Density based Classifier

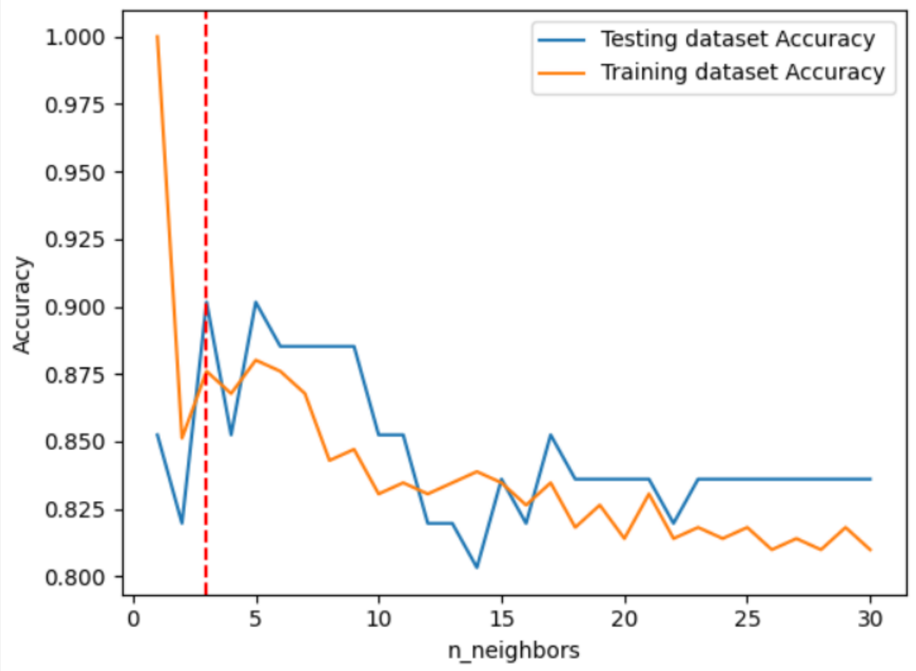
We first seperated our training dataset on the bases of target class and applied Kernel density model on it. When a new data point comes it is assigned the class on the basis of the density of that point in both of the classes.

```
kdc = KernelDensityClassifier()
kdc.train(X_train_0, X_train_1)

pred = kdc.predict(X_test)
print(accuracy_score(y_test, pred))
# 0.8852459016393442

print(precision_score(y_test, pred))
# 0.9032258064516129
```

Kernel Density Classifier gives us around 88.5% accuracy and 90.3% precision on test set.

4.4 KNN based Classifier

Using the KNeighborsClassifier module from sklearn, we build a KNN model and iteratively tune the hyper-parameter n-neighbors (number of neighbours to be checked for every data point).



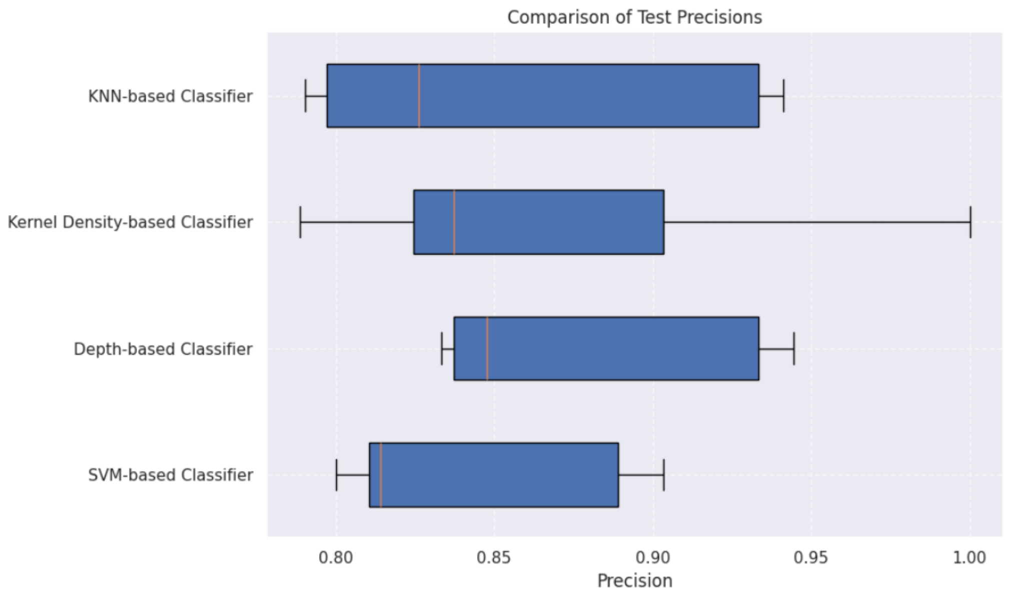
```
Best choice of neighbors is  3
```

We see that the best choice of K is 3, and the accuracy and precision of this model is around 90.2% and 93.3% respectively.

```python
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

print(knn.score(X_test, y_test))
# 0.9016393442622951

pred = knn.predict(X_test)
print(precision_score(y_test, pred))
# 0.9333333333333333
```

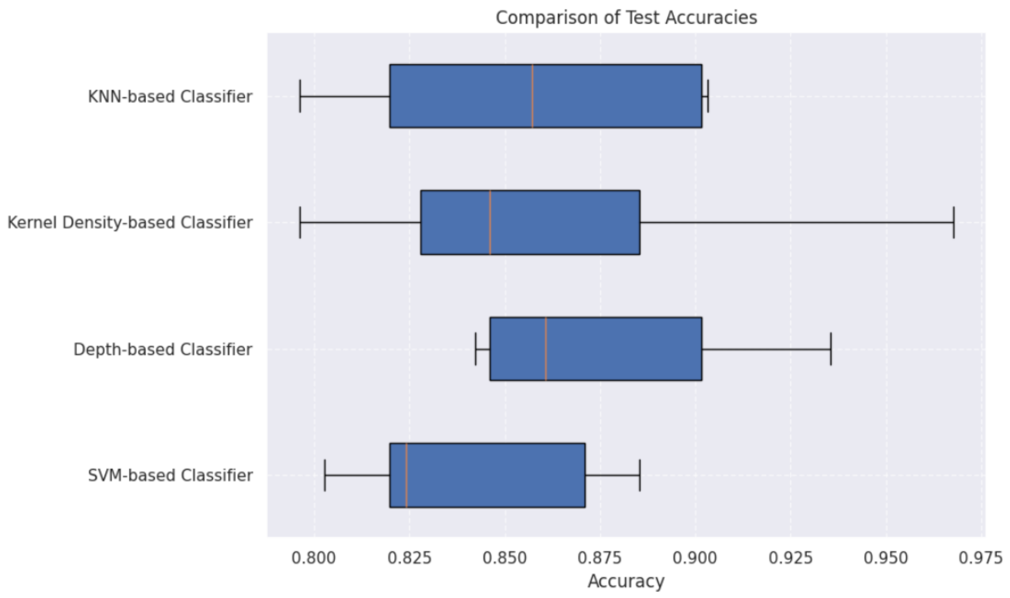
4.5 Box plots on Precision of various model

Now we check the Precision of our final models at different splits of our dataset like 90/10, 80/20, 70/30, 60/40 and 50/50. The we make a box plot out of them.



4.6 Box plots on Accuracy of various model

Now we check the accuracy of our final models at different splits of our dataset like 90/10, 80/20, 70/30, 60/40 and 50/50. The we make a box plot out of them.



After trying four different machine learning techniques to predict heart disease, it is clear that **Depth Based(Random Forest) Classifier** perform the best on average on our UCI dataset. However, feature engineering and hyperparameter tuning in other models can also yield comparable results.