Diabetes Prediction Data Analysis Project

1. Introduction

Diabetes is one of the most common and hazardous diseases on the planet. It requires a lot of care and proper medication to keep the disease in control. In this data mining project, we will use the provided diabetes to train and evaluate our model with various classification approaches. At the end of the experiment, we will conclude which classifier is best suited for diabetes predication.

2. Data Preparation

Diabetes dataset has 8 features: (Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, DiabetesPedigreeFunction and Age) and 2 classes: 1 being Diabetes and 0 being No Diabetes. There are a total of 768 observations and we used a 70/30 split to train and validate our models. Therefore, our training dataset has 537 observations with 183 Yes Diabetes and 354 No Diabetes.

```
from sklearn.utils import resample
trainData = pd.concat([X_train, y_train], axis=1) # combine the X_train and y_train back
trainData.shape # total size of train data
```

```
(537, 9)
```

```
yes_diabetes = trainData[trainData.Outcome==1] # how many is yes
no_diabetes = trainData[trainData.Outcome==0] # how many is no
print("Yes Diabetes: ",yes_diabetes.shape)
print("No Diabetes: ", no_diabetes.shape)
```

```
Yes Diabetes:  (183, 9)
No Diabetes:  (354, 9)
```

Original 70% diabetes dataset for Training

Total	537
Yes Diabetes	183
No Diabetes	354

There are more negative diabetes than positive diabetes. Therefore, training data is unbalanced and skewed to one side. When we train with unbalanced data, our model would overfit to negative diabetes records. This is undesirable as our model would produce more False Negative cases and we do not want such case in predicting whether a patient has diabetes. It is better to have more False Positives than False Negative which is a patient having diabetes but we predicted no diabetes. In any cases, it would be better to re-sample the training data.

Oversampling

We decided to experiment with both oversampling and under-sampling training data.

For oversampling, we increased the positive diabetes to be the same as negative diabetes. We used sklearn sample module to replicates the positive diabetes dataset. In total, we have 708 observations for training.

Diabetes Prediction Data Analysis Project

Oversampling

```
from sklearn.utils import resample
# Supply the "Yes Diabetes" samples to match the number of "No Diabetes"
yes_diabetes_up = resample(yes_diabetes, replace=True, n_samples=len(no_diabetes), random_state=1) # reproducible results
print("Yes Diabetes: ",yes_diabetes_up.shape)
print("No Diabetes: ", no_diabetes.shape)
```

```
Yes Diabetes:  (354, 9)
No Diabetes:  (354, 9)
```

```
trainDataUp = pd.concat([yes_diabetes_up, no_diabetes])
trainDataUp.shape # total size of train up data
```

```
(708, 9)
```

```
X_train_up, y_train_up = trainDataUp.drop("Outcome",axis=1), trainDataUp["Outcome"]
print(X_train_up.shape)
print(y_train_up.shape)
```

```
(708, 8)
(708,)
```

Over-sampling Training dataset

Total	537
Yes Diabetes	183
No Diabetes	354

Undersampling

For undersampling, we cut the negatives diabetes to be the same as the number of positive diabetes. In total, we have 366 observations for training.

Undersampling

```
from sklearn.utils import resample
# Cut the "No Diabetes" samples to match the number of "Yes Diabetes"
no_diabetes_down = resample(no_diabetes, n_samples=len(yes_diabetes), random_state=1) # reproducible results
print("Yes Diabetes: ",yes_diabetes.shape)
print("No Diabetes: ", no_diabetes_down.shape)
```

```
Yes Diabetes:  (183, 9)
No Diabetes:  (183, 9)
```

```
trainDataDown = pd.concat([yes_diabetes, no_diabetes_down])
trainDataDown.shape # total size of train down data
```

```
(366, 9)
```

```
X_train_down, y_train_down = trainDataDown.drop("Outcome",axis=1), trainDataDown["Outcome"]
print(X_train_down.shape)
print(y_train_down.shape)
```

```
(366, 8)
(366,)
```

Under-sampling Training dataset

Total	366
Yes Diabetes	183
No Diabetes	183

Model Evaluation Criteria

For the effectiveness of the model, we will not solely focus on the accuracy score but also the recall (the ability of a classifier to correctly find all positive instances) as we feel that it is of most important to be able to predict the True Positive Cases (Patients with diabetes) than True Negative Cases (Patients without diabetes).

Diabetes Prediction Data Analysis Project

3. Classifications

3.1 Decision Trees

We chose Decision Tress Classifier because it is easy to implement and interpret the decision easily. Given the small size of diabetes dataset, we can also visualize the tree-like plot from Decision Tree and see which factors are influencing for a patient to be diagnosed with diabetes.

However, decision tree is prone to overfitting the data and can be biased if the target variable classes are unbalanced. Therefore, we decided to experiment not only with the original diabetes training dataset but also with oversampling and under-sampling training dataset.

3.1.1 Binary Tree Parameters and model evaluations of auto-classifications

Default values for parameters of binary tree are as follows:

- **criterion**: default is Gini. The function to measure the quality of a split. Supported criteria are “Gini” for the node impurity and “Log Loss” and “entropy” for the information gain. In Gini, the impurity is measured between 0 to 0.5 and in Entropy the impurity is lies between 0 and 1 but the only difference is entropy might be a little slower to compute because it requires to compute a logarithmic function.

- **min_samples_leaf**: default is 2. It represents the minimum number of samples required to be in the leaf node. The more we increase the number, the higher the possibility of overfitting.

- **max_depth**: Default is None. The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

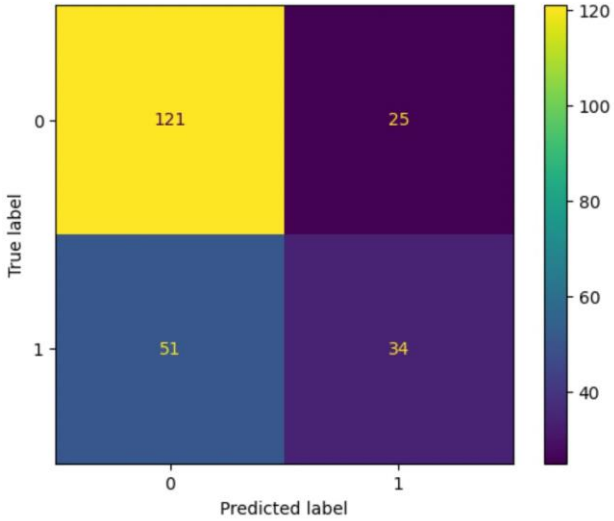
With that we ran binary tree classification to training dataset.

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(max_depth=None, min_samples_leaf=2) #default parameters: criterion="gini"
dtree.fit(X_train, y_train)
```

3.1.2 Model Evaluation

For Model Evaluation, we used classification report and confusion matrix.

	precision	recall	f1-score	support
0	0.70	0.83	0.76	146
1	0.58	0.40	0.47	85
accuracy			0.67	231
macro avg	0.64	0.61	0.62	231
weighted avg	0.66	0.67	0.65	231



Accuracy score of model is 0.67 and recall score for 0 is 0.83 and 1 is 0.40. The result is not bad but we believed we can improve the scores by tuning the value of Binary Tree parameters.

3.1.3 Parameters Tuning and any other approaches to improve the score

By visualising the default parameters tree, we can see it is a bit of a mess. Some nodes have just 1 sample in them. Since we do not have much data we will need to control the number of samples in each node and also the max depth as this can lead to overfitting and poor performance on unseen data. Hence, we will optimise the decision tree by tuning various parameters such as criterion , max_depth and min_samples_split.

- **criterion**: “Gini”, “Entroy”, “Log Loss”

- **min_samples_leaf**: represents the minimum number of samples required to be in the leaf node. The more you increase the number, the more is the possibility of overfitting. We will experiment with the range of 2 to 10.

- **max_depth**: The length of the binary tree. In general, the deeper you allow your tree to grow, the more complex the model will become because we will have more splits and it captures more information about the data and this is one of the root causes of overfitting in decision trees because the model will fit perfectly for the training data and will not be able to generalize well on test set. So, if the model is overfitting, reducing the number for max_depth is one way to combat overfitting. It is also bad to have a very low depth because the model will underfit so to find the best value, experimented with overfitting and underfitting. For max_depth, we will experiment with the range from 2 to 15.

Training with original 70% diabetes dataset

Diabetes Prediction Data Analysis Project

Total	537
Yes Diabetes	183
No Diabetes	354

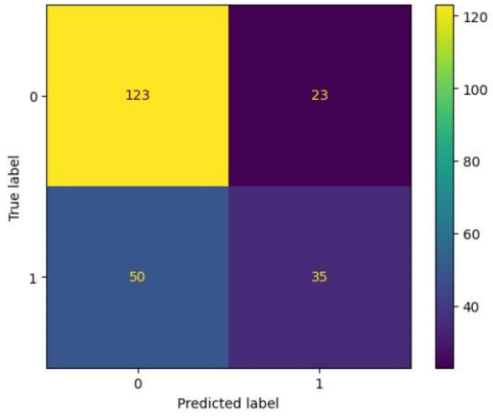
With training data

```
grid_dTree=gridSearch(DecisionTreeClassifier(),hyper_params,cv,"recall",X_train,y_train)
printModelEvaluation(y_test, grid_dTree.predict(X_test))
```

```
GCV best params:  {'criterion': 'gini', 'max_depth': 13, 'min_samples_leaf': 2}
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x14ffe6200>
              precision    recall  f1-score   support

           0       0.71      0.84      0.77       146
           1       0.60      0.41      0.49        85

    accuracy                           0.68       231
   macro avg       0.66      0.63      0.63       231
weighted avg       0.67      0.68      0.67       231
```



Training with over-sampling 70% diabetes dataset

Total	708
Yes Diabetes	354
No Diabetes	354

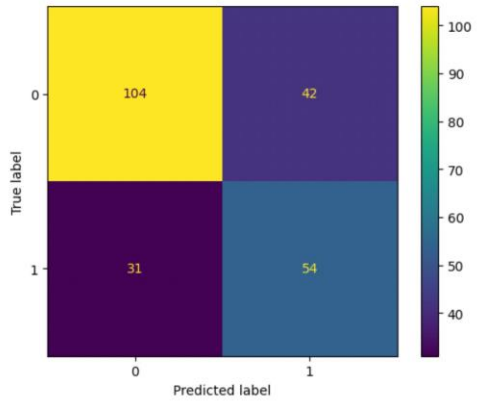
With Oversample training data

```
grid_dTree_Up=gridSearch(DecisionTreeClassifier(),hyper_params,cv,"recall",X_train_up,y_train_up)
printModelEvaluation(y_test, grid_dTree_Up.predict(X_test))
```

```
GCV best params:  {'criterion': 'log_loss', 'max_depth': 9, 'min_samples_leaf': 3}
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x13e0c39d0>
              precision    recall  f1-score   support

           0       0.77      0.71      0.74       146
           1       0.56      0.64      0.60        85

    accuracy                           0.68       231
   macro avg       0.67      0.67      0.67       231
weighted avg       0.69      0.68      0.69       231
```



Training with under-sampling 70% diabetes dataset

Total	366
Yes Diabetes	183
No Diabetes	183

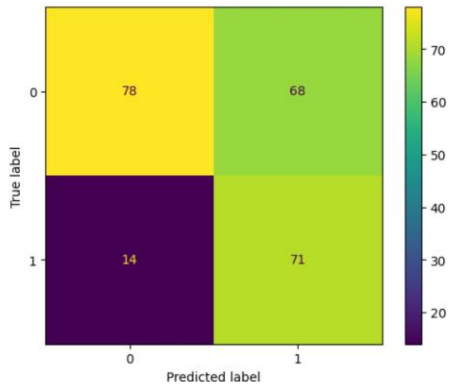
With Undersample training data

```
grid_dTree_Down=gridSearch(DecisionTreeClassifier(),hyper_params,cv,"recall",X_train_down,y_train_down)
printModelEvaluation(y_test, grid_dTree_Down.predict(X_test))
```

```
GCV best params:  {'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 5}
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x13e156350>
              precision    recall  f1-score   support

           0       0.85      0.53      0.66       146
           1       0.51      0.84      0.63        85

    accuracy                           0.65       231
   macro avg       0.68      0.68      0.64       231
weighted avg       0.72      0.65      0.65       231
```



3.1.4 Final Model of Binary Tree

In comparing the models (original, oversample, undersample), we observed resampling data gave a better accuracy for recall. The model trained with Oversampling training data gave the best scores for recall even though overall accuracy is the lowest of three. Even though model with original training data has the best accuracy score, its recall score for Yes Diabetes is low when compared to model trained with under sample data.

Our final model of Binary Tree for diabetes dataset is the model trained with under-sampling training data with criterion as entropy, max_depth as 5 and min_samples_leaf as 5 since it has the best score for Yes Diabetes recall (0.84) indicating it can identify patient with diabetes better than original model and oversample model.

When comparing with other classifier, we will use the binary tree model trained with under-sampling.

Original	GCV best params: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 7} <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x14ffb8 precision recall f1-score support 0 0.81 0.86 0.84 146 1 0.74 0.66 0.70 85 accuracy 0.79 231 macro avg 0.77 0.76 0.77 231 weighted avg 0.78 0.79 0.79 231

Diabetes Prediction Data Analysis Project

Oversample	GCV best params: {'criterion': 'entropy', 'max_depth': 9, 'min_samples_leaf': 3} <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x14fd21b70> precision recall f1-score support 0 0.76 0.74 0.75 146 1 0.57 0.60 0.59 85 accuracy 0.69 231 macro avg 0.67 0.67 0.67 231 weighted avg 0.69 0.69 0.69 231
Under-sample	GCV best params: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 5} <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x13e2b25c0> precision recall f1-score support 0 0.85 0.53 0.66 146 1 0.51 0.84 0.63 85 accuracy 0.65 231 macro avg 0.68 0.68 0.64 231 weighted avg 0.72 0.65 0.65 231

3.2 Support Vector Machine

Support Vector Machine (SVM) is one of the most popular supervised learning classifier. The objective of SVM is to find the hyperplane with the largest margin, which would distinctly classified the data in an N-dimensional space. We chose this classifier for diabetes dataset as we have not had a practice of SVM implementation with sklearn before and it would be interesting to observe how SVM would differ from other classifiers.

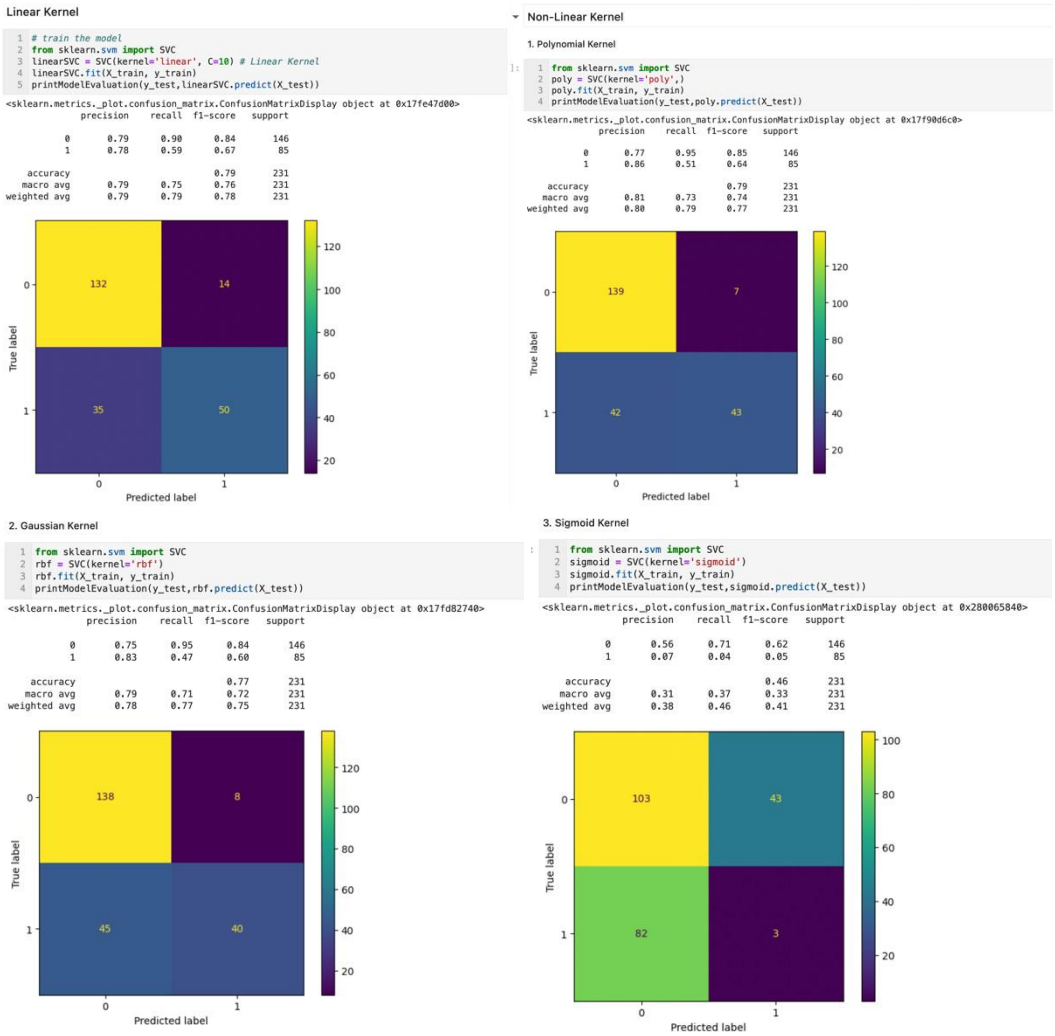
3.2.1 SVM Parameters and model evaluations of auto-classifications

3.2.1.1 SVM Parameters

For non-linear and inseparable planes, SVM uses a kernel trick to transform the input space to a higher dimensional space. We ran both linear and non-linear auto-classification of SVMs and observe the result.

For auto classification of linear SVM, Support Vector Classification C is default to 1.0.

For auto classification of non-linear SVM Classification, the degree of polynomial kernel is by default 3 and the gamma value is default to scale.



Diabetes Prediction Data Analysis Project

3.2.1.2 Model Evaluation

Confusion Matrix

	Linear SVM		Polynomial SVM		Gaussian SVM		Sigmoid SVM	
True	**Predict**							
	0	**1**	**0**	**1**	**0**	**1**	**0**	**1**
0	132	14	139	7	138	8	103	43
1	35	50	42	43	45	40	82	3

Confusion Matrix tells us that Sigmoid SVM is not suitable for diabetes dataset as it produces bad accuracy score for diabetes prediction. Linear SVM is the best of all with high accuracy rate for True Positives (50) and False Positives (35).

Classification Report

Linear SVM:

	precision	recall	f1-score	support
0	0.79	0.90	0.84	146
1	0.78	0.59	0.67	85
accuracy			0.79	231
macro avg	0.79	0.75	0.76	231
weighted avg	0.79	0.79	0.78	231

Polynomial SVM:

	precision	recall	f1-score	support
0	0.77	0.95	0.85	146
1	0.86	0.51	0.64	85
accuracy			0.79	231
macro avg	0.81	0.73	0.74	231
weighted avg	0.80	0.79	0.77	231

Gaussian SVM:

	precision	recall	f1-score	support
0	0.75	0.95	0.84	146
1	0.83	0.47	0.60	85
accuracy			0.77	231
macro avg	0.79	0.71	0.72	231
weighted avg	0.78	0.77	0.75	231

Sigmoid SVM:

	precision	recall	f1-score	support
0	0.56	0.71	0.62	146
1	0.07	0.04	0.05	85
accuracy			0.46	231
macro avg	0.31	0.37	0.33	231
weighted avg	0.38	0.46	0.41	231

Classification Report again illustrates linear SVM is best suited for Diabetes dataset. Having seen how SVM linear and non-linear classifications performs with original diabetes dataset, we can deduce the diabetes data is linearly separable. Therefore, for subsequent hyperparameter tuning, we will focus on the linear SVM.

Diabetes Prediction Data Analysis Project

3.2.3 Parameters Tuning and any other approaches to improve the score

The goals of SVM are

(1) Increase the distance of decision boundary to classes (or support vectors)

(2) Maximize the number of points that are correctly classified in the training set

There is a trade-off between (1) and (2) and it is controlled by the parameter 'c'.

C is a regularization parameter which adds a penalty for each misclassified data point.

If c is small, the penalty for misclassified points is low so a decision boundary with a large margin is chosen at the expense of a greater number of misclassifications. If c is large, SVM tries to minimize the number of misclassified examples due to high penalty which results in a decision boundary with a smaller margin. The strength of the regularization is inversely proportional to C.

For the hyper parameter tuning, we will only focus on the C Value. For the C value, we decided to experiment with the range from 0.1 to 1.6 with 10 cross validations. This should cover both overfitting and generalization issues.

Training with original 70% diabetes dataset

Total	537
Yes Diabetes	183
No Diabetes	354

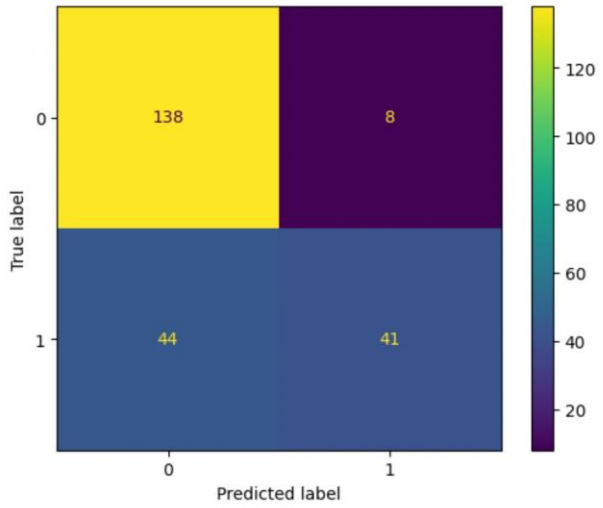
With training data

```
1 grid_SVC = gridSearch(SVC(),hyper_params,cv,"recall",X_train,y_train)
2 printModelEvaluation(y_test, grid_SVC.predict(X_test))
```

```
GCV best params:  {'C': 1.5000000000000002}
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x1386a27d0>
              precision    recall  f1-score   support

           0       0.76      0.95      0.84       146
           1       0.84      0.48      0.61        85

    accuracy                           0.77       231
   macro avg       0.80      0.71      0.73       231
weighted avg       0.79      0.77      0.76       231
```



Diabetes Prediction Data Analysis Project

Training with over-sampling 70% diabetes dataset

Total	708
Yes Diabetes	354
No Diabetes	354

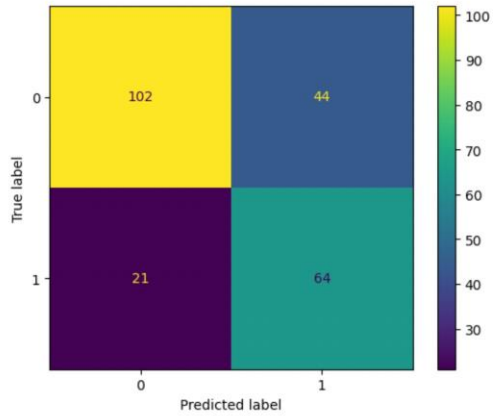
With Oversample training data

```
grid_SVC_Up = gridSearch(SVC(),hyper_params,cv,"recall",X_train_up,y_train_up)
printModelEvaluation(y_test, grid_SVC_Up.predict(X_test))
```

```
GCV best params:  {'C': 1.0}
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x138f5ffa0>
              precision    recall  f1-score   support

           0       0.83      0.70      0.76       146
           1       0.59      0.75      0.66        85

    accuracy                           0.72       231
   macro avg       0.71      0.73      0.71       231
weighted avg       0.74      0.72      0.72       231
```



Training with under-sampling 70% diabetes dataset

Total	366
Yes Diabetes	183
No Diabetes	183

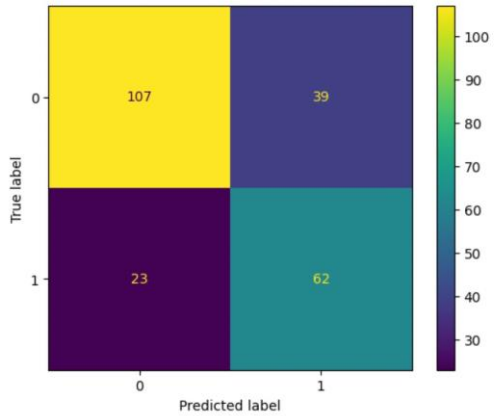
With Undersample training data

```
grid_SVC_Down = gridSearch(SVC(),hyper_params,cv,"recall",X_train_down,y_train_down)
printModelEvaluation(y_test, grid_SVC_Down.predict(X_test))
```

```
GCV best params:  {'C': 0.4}
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x138fb1d80>
              precision    recall  f1-score   support

           0       0.82      0.73      0.78       146
           1       0.61      0.73      0.67        85

    accuracy                           0.73       231
   macro avg       0.72      0.73      0.72       231
weighted avg       0.75      0.73      0.74       231
```



3.2.4 Final Model of SVM

In comparing the models (original, oversample, under-sample), we observed resampling data gave a better accuracy for recall. Especially, the model trained with Oversampling training data gave the best scores for recall even though overall accuracy is lowest of three.

For the classifier comparison, we will be using SVM model trained with over-sample data which has C value of 1.0.

Original	GCV best params: {'C': 1.5000000000000002} <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDi precision recall f1-score support 0 0.76 0.95 0.84 146 1 0.84 0.48 0.61 85 accuracy 0.77 231 macro avg 0.80 0.71 0.73 231 weighted avg 0.79 0.77 0.76 231
Over-sample	GCV best params: {'C': 1.0} <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDi precision recall f1-score support 0 0.83 0.70 0.76 146 1 0.59 0.75 0.66 85 accuracy 0.72 231 macro avg 0.71 0.73 0.71 231 weighted avg 0.74 0.72 0.72 231
Under-sample	GCV best params: {'C': 0.4} <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDis precision recall f1-score support 0 0.82 0.73 0.78 146 1 0.61 0.73 0.67 85 accuracy 0.73 231 macro avg 0.72 0.73 0.72 231 weighted avg 0.75 0.73 0.74 231

3.3 K-Nearest Neighbours

K-Nearest is a distance based algorithm which means it does take distance in consideration when learning a data set, K-Nearest tries to classify which data point belongs to which class, let's say we have a finite number of data points on a graph from these finite number data points we have five data points near to each other which implies they have a-lot in common so hypothetically it's safe to consider them a class and this what K-nearest tries to achieve to classify points to a class by clustering points similar to each other as a class.

We use K-nearest Neighbours classifier to classify whether a person has diabetes or not based on some attributes like Insulin, Blood Pressure, Skin-Thickness, BMI, Age, Glucose, Pregnancies, Diabetes Pedigree Function.

3.3.1 K-Nearest Neighbours Parameter

K-nearest Neighbours's default parameters:

- **n_neighbors**: Number of neighbours to use by default for kneighbours queries. Default is 5.
- **weights**: Weight function used in prediction. Default is 'uniform'.
- **algorith**: Algorithm used to compute the nearest neighbours. Default is 'auto'
- **leaf_size**: Leaf size passed to BallTree or KDTree. Default is 30. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.
- **p:** Power parameter for the Minkowski metric. Default is 2. When p = 1, this is equivalent to using manhattan_distance, and euclidean_distance for p = 2. For arbitrary p, minkowski_distance (l_p) is used.
- **metric**: Metric to use for distance computation. Default is "minkowski", which results in the standard Euclidean distance when p = 2.
- **metric_params:** Additional keyword arguments for the metric function. Default is None.
- **n_jobs:** The number of parallel jobs to run for neighbors search. Default is None. None means 1. -1 means using all processors.

With that we ran k-nearest neighbours classification to training dataset.

```
1 from sklearn.neighbors import KNeighborsClassifier
2 neigh = KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski',
  metric_params=None, n_jobs=None) #default parameters
3 neigh.fit(X_train, y_train)
```

3.3.2 Model Evaluation

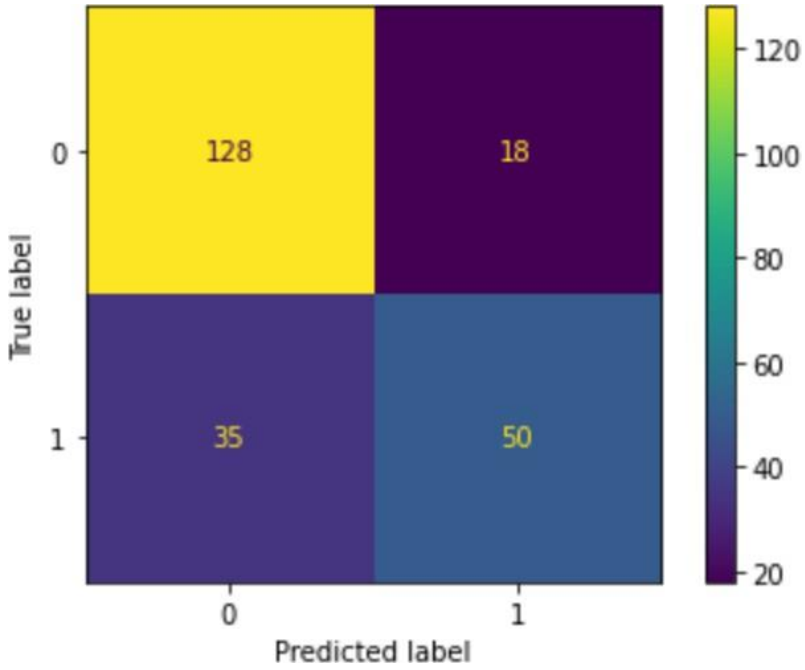
Diabetes Prediction Data Analysis Project

The default K-nearest Neighbours achieves accuracy approximately equals to **77%**.

Classification reports

	precision	recall	f1-score	support
0	0.79	0.88	0.83	146
1	0.74	0.59	0.65	85
accuracy			0.77	231
macro avg	0.76	0.73	0.74	231
weighted avg	0.77	0.77	0.76	231

Confusion Matrix



Recall score for 0 is 0.88 and 1 is 0.59. The result is not bad but we believed we can improve the scores by tuning the value of K-Nearest Neighbour parameters.

3.3.3 Parameters Tuning

We are going to tune the following parameters **n_neighbors, leaf_size, p, weights, metric** using Grid Search CV.

Diabetes Prediction Data Analysis Project

```
[38]  1 from sklearn.model_selection import GridSearchCV
      2
      3 hyper_params = {
      4     'n_neighbors': list(range(1, 31)),
      5     'weights': ('uniform', 'distance'),
      6     'leaf_size': (10, 20, 30, 40),
      7     'p': (1,2),
      8     'metric': ('braycurtis', 'minkowski', 'chebyshev')
      9 }
     10
     11 cv = 10 # cross validation 10
```

Training with original 70% diabetes dataset

Total	537
Yes Diabetes	183
No Diabetes	354

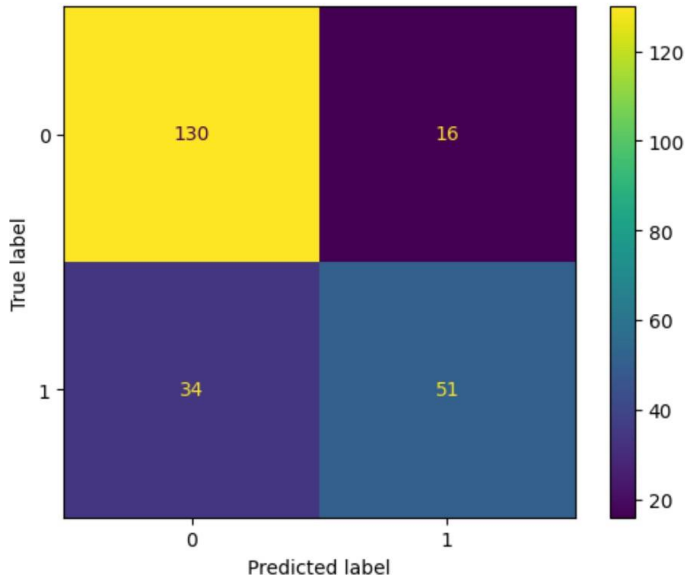
With training data

```
1 from sklearn.neighbors import KNeighborsClassifier
2 grid_neigh=gridSearch(KNeighborsClassifier(),hyper_params,cv,"recall",X_train,y_train)
3 printModelEvaluation(y_test, grid_neigh.predict(X_test))
```

```
GCV best params:  {'leaf_size': 10, 'metric': 'braycurtis', 'n_neighbors': 7, 'p': 1, 'weights': 'distance'}
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x174ca6140>
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       146
           1       0.76      0.60      0.67        85

    accuracy                           0.78       231
   macro avg       0.78      0.75      0.75       231
weighted avg       0.78      0.78      0.78       231
```



Training with over-sampling 70% diabetes dataset

Total	708
Yes Diabetes	354

Diabetes Prediction Data Analysis Project

No Diabetes	354

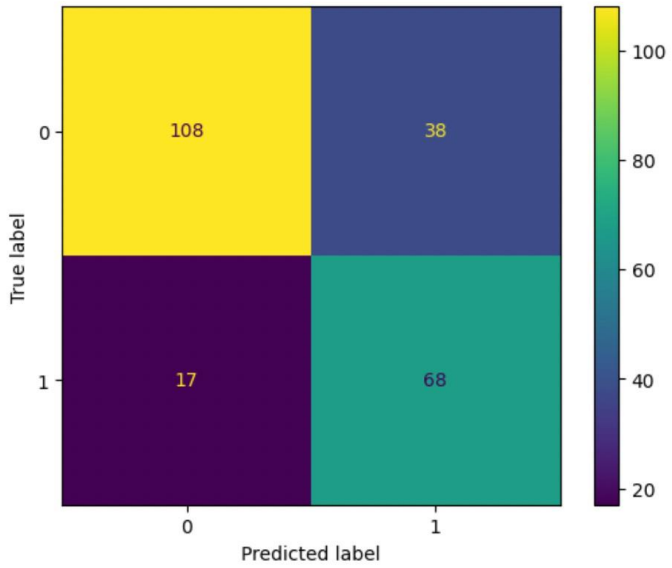
With Oversample training data

```
1 grid_neigh_Up=gridSearch(KNeighborsClassifier(),hyper_params,cv,"recall",X_train_up,y_train_up)
2 printModelEvaluation(y_test, grid_neigh_Up.predict(X_test))
```

```
GCV best params:  {'leaf_size': 10, 'metric': 'minkowski', 'n_neighbors': 18, 'p': 2, 'weights': 'distance'}
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x178e62980>
              precision    recall  f1-score   support

           0       0.86      0.74      0.80       146
           1       0.64      0.80      0.71        85

    accuracy                           0.76       231
   macro avg       0.75      0.77      0.75       231
weighted avg       0.78      0.76      0.77       231
```



Training with under-sampling 70% diabetes dataset

Total	366
Yes Diabetes	183
No Diabetes	183

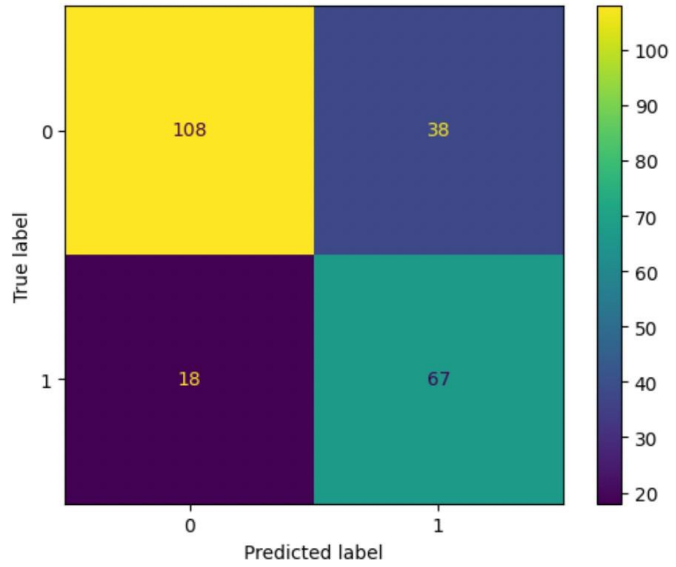
With Undersample training data

```
grid_neigh_Down=gridSearch(KNeighborsClassifier(),hyper_params,cv,"recall",X_train_down,y_train_down)
printModelEvaluation(y_test, grid_neigh_Down.predict(X_test))
```

```
GCV best params:  {'leaf_size': 10, 'metric': 'braycurtis', 'n_neighbors': 28, 'p': 1, 'weights': 'distance'}
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x13e2b1510>
              precision    recall  f1-score   support

           0       0.86      0.74      0.79       146
           1       0.64      0.79      0.71        85

    accuracy                           0.76       231
   macro avg       0.75      0.76      0.75       231
weighted avg       0.78      0.76      0.76       231
```



3.3.4 Final Model

In comparing the models (original, oversample, undersample), we observed resampling data gave a better accuracy for recall. Especially, the model trained with Oversampling training data gave the best scores for recall. For the overall accuracy score, there is not much difference in value between all three models.

For the classifier comparison, we will be using K-Nearest neighbour model trained with over-sample data which has the parameter values as leaf_size = 10, metric = minkowski, n_neighbour = 18, p = 2 and weights = distance.

In all there cases, GCV returns weighting as distance for parameter value which means it is better to weight based on the distance than uniformly in defining the neighbours.

Original	GCV best params: {'leaf_size': 10, 'metric': 'braycurtis', 'n_neighbors': 7, 'p': 1, <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x174ca6140> precision recall f1-score support 0 0.79 0.89 0.84 146 1 0.76 0.60 0.67 85 accuracy 0.78 231 macro avg 0.78 0.75 0.75 231 weighted avg 0.78 0.78 0.78 231

Diabetes Prediction Data Analysis Project

Over-sample	GCV best params: {'leaf_size': 10, 'metric': 'minkowski', 'n_neighbors' <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at precision recall f1-score support 0 0.86 0.74 0.80 146 1 0.64 0.80 0.71 85 accuracy 0.76 231 macro avg 0.75 0.77 0.75 231 weighted avg 0.78 0.76 0.77 231
Under-sample	GCV best params: {'leaf_size': 10, 'metric': 'braycurtis', 'n_neighbors': 28, 'p': 1, 'weig <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x13e2b1510> precision recall f1-score support 0 0.86 0.74 0.79 146 1 0.64 0.79 0.71 85 accuracy 0.76 231 macro avg 0.75 0.76 0.75 231 weighted avg 0.78 0.76 0.76 231

Diabetes Prediction Data Analysis Project

4. Conclusion

4.1 Best classifier for diabetes

Below is the results table of the three classifiers from predicting testing data (30% of diabetes dataset with 146 No Diabetes and 85 Yes Diabetes).

	Decision Tree	**SVM**	**K-Nearest**
Trained with	**Under-sample**	**Over-sample**	**Over-sample**
Accuracy Score	0.65	0.72	0.76
Recall-0	0.53	0.70	0.74
Recall-1	0.84	0.75	0.80
True-0	78	102	108
True-1	71	64	68
False-0	14	21	17
False-1	68	44	38

Table: Comparison table for three classifiers

1. In terms of **overall accuracy score**, K-Nearest gave the best score at 0.76.
   Best: K-Nearest, followed by SVM.

2. When comparing the **recall scores** for positive cases (Yes Diabetes), Decision Tree produces a better result at 0.84, followed by K-Nearest at 0.80. Binary Tree model trained with under-sample seems to have a bias towards positive cases as the difference between recall-0 score (0.53) and recall-1 score (0.84) is quite vast whereas the other models have more or less of balanced scores for recall-0 and recall-1.
   Best: K-Nearest

3. In addition to successfully predicting whether a patient has diabetes, we also want our model to **avoid False-0 cases** which is predicting no diabetes when in fact the patient has the diabetes. For that, we look at the metrics of False-0 in the table. Binary Tree has 14 of such cases and K-Nearest is at 17 with difference of 3.
   Best: Binary Tree, closely followed by K-Nearest.

4. **False-1 cases** are where patients do not have diabetes but the model predicts to be Yes Diabetes. It is not as important as False-0 since prevention is better than cure. Binary Tree has 68 of such cases and K-Nearest is at 38 with difference of 30.
   Best: K-Nearest

After assessing the metrics, we believe K-Nearest with oversampling training data is the best model for Diabetes prediction. This K-Nearest model has the parameter values as leaf_size = 10, metric = minkowski, n_neighbour = 18, p = 2 and weights = distance. If we wish our model to be overly pessimistic, Decision Tree Model trained with under-sampling training data would be the choice.

4.2 Lessons Learnt

- In all experiments, we observed that resampling training data produces a better score than original training data set. Therefore, we learnt to pre-process the data before training the model.
- Decision Tree tends to overfit to the training dataset and in assessing the model efficiency, we should not only look at accuracy scores but also other factors such as recall, predictions, etc.
- There is not a hard and fast rule when deciding which classifier to use for model training. Experiments are the key coupled with proper and clean training dataset.

4.3 Improvement considerations

- We assume our diabetes dataset has no outliners but in reality, it may not be true. We could have taken in consideration of anomalies before training the models.

5. Neural Networks

Additionally, we decided to do an experiment with Neural Networks out of curiosity and also because we have 4 members in the team. We noticed that changing some important parameters such as learning rate, epoch, batch size and number of hidden neurons and layers would improve the model score. We deviated from the sklearn and built the hidden layers with keras library.

The reason to choose Neural Networks are given below:

- Neural Networks can learn and model non-linear and complex relationships, which is important because in real-life, many of the relationships between inputs and outputs are non-linear as well as complex.
- After learning from the initial inputs and their relationships, it can infer unseen relationships on unseen data as well, thus making the model generalize and predict on unseen data.
- Unlike many other prediction techniques, Neural Networks does not impose any restrictions on the input variables (like how they should be distributed). Additionally, many studies have shown that Neural Networks can better model heteroskedasticity i.e., data with high volatility and non-constant variance, given its ability to learn hidden relationships in the data without imposing any fixed relationships in the data.

5.1 Build the model using Default Parameters

Batch Size: The batch size in iterative gradient descent is the number of patterns shown to the network before the weights are updated. It is also an optimization in the training of the network, defining how many patterns to read at a time and keep in memory.

Number of Epoch: The number of epochs is the number of times the entire training dataset is shown to the network during training.

Learning Rate : The learning rate is a high parameter that controls how much the model can be adjusted when the weight is improved as a response to the expected error.

Neurons: An artificial neuron is a connection point in an artificial neural network. Artificial neural networks, like the human body's biological neural network, have a layered architecture and each network node (connection point) has the capability to process input and forward output to other nodes in the network.

We have implemented the model using default parameters and got the training score as 67.59% and test score as 65.58%

```
[65] from keras.models import Sequential
     from keras.layers import Dense
     model = Sequential()
     # First hidden layer
     model.add(Dense(64, input_shape=(8,), activation='relu'))
     # Output layer
     model.add(Dense(1,activation='sigmoid'))
     # Compile the model
     model.compile(optimizer='adam', loss='binary_crossentropy',metrics=['accuracy'])
     # Training
     model.fit(X_train,y_train)
```

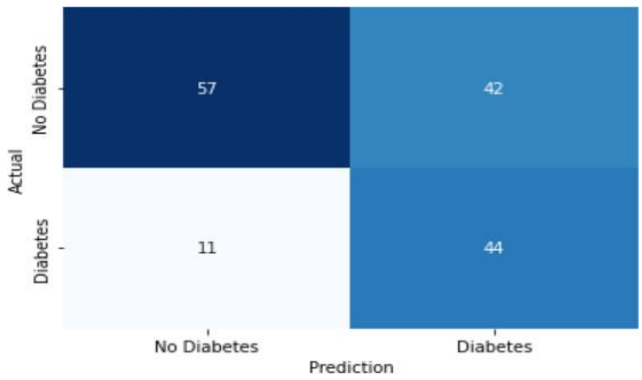
```
20/20 [==============================] - 1s 2ms/step - loss: 0.6711 - accuracy: 0.6173
<keras.callbacks.History at 0x7fdb0efad250>
```

```
20/20 [==============================] - 0s 1ms/step - loss: 0.6171 - accuracy: 0.6759
Training Accuracy: 67.59%

5/5 [==============================] - 0s 2ms/step - loss: 0.6154 - accuracy: 0.6558
Testing Accuracy: 65.58%
```

5.2 Model Evaluation

Confusion Matrix:



Classification Report

	precision	recall	f1-score	support
0	0.84	0.58	0.68	99
1	0.51	0.80	0.62	55
accuracy			0.66	154
macro avg	0.67	0.69	0.65	154
weighted avg	0.72	0.66	0.66	154

5.3 Parameters Tuning and any other approaches to improve the score

How to Tune Batch Size, Number of Epochs, Learning Rate and Hidden layer?

Batch Size:
The size of a batch must be more than or equal to one and less than or equal to the number of samples in the training dataset.

Number of Epoch:
The number of epochs can be set to an integer value between one and infinity. There are no particular rule for how to configure these parameters. We must try different values and see what works best for your problem. The best parameters and best score for Batch Size and Number of Epochs using GridSearchCV are given below:

Learning Rate:
The learning rate will interact with many other aspects of the optimization process, and the interactions may be nonlinear. Nevertheless, in general, smaller learning rates will require more training epochs. Conversely, larger learning rates will require fewer training epochs. Further, smaller batch sizes are better suited to smaller learning rates given the noisy estimate of the error gradient.

A traditional default value for the learning rate is 0.1 or 0.01, and this may represent a good starting point on your problem.

Number of Hidden Neuron:
The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer. The number of hidden neurons should be less than twice the size of the input layer

The best score for Learning Rate, Number of Epoch, Batch Size and Number of Hidden Neurons in the Hidden Layer using GridSearchCV are given below:

```
grid_result = grid.fit(X_train,y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.778471 using {'batch_size': 50, 'epochs': 30, 'model__neurons': 64, 'optimizer__learning_rate': 0.1}
```

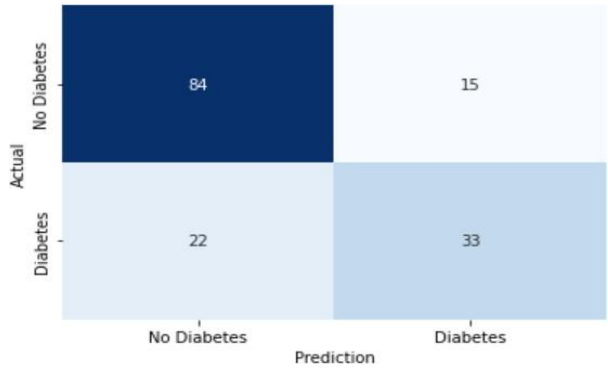
5.4 Final Model

Now, we have implemented the model using the best parameters found by GridSearchCV and got below score:

```
20/20 [==============================] - 0s 2ms/step - loss: 0.3923 - accuracy: 0.8160
Training Accuracy: 81.60%

5/5 [==============================] - 0s 3ms/step - loss: 0.5111 - accuracy: 0.7597
Testing Accuracy: 75.97%
```

Confusion Matrix:



Classification Report:

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.85      0.82        99
           1       0.69      0.60      0.64        55

    accuracy                           0.76       154
   macro avg       0.74      0.72      0.73       154
weighted avg       0.75      0.76      0.76       154
```

5.5 Reason not to choose

Though neural networks are working good with the dataset, there are some disadvantages that motivates us not to consider neural networks within three selected classifiers:

- **Take time to be developed:** We all know that a neural network will take much longer to train rather than a traditional ML algorithm, and we don't always have the computational power and even if we do, sometimes, we just don't have the time to wait so long and to decide whether the end result will be worth the wait or not. Since diabates dataset is a more complex dataset, to get good accuracy will be more costly in terms of computational power and time.

- **Require too much data:** Neural Networks require much more data than traditional Machine Learning algorithms to do their job well. Most of the times you will find yourself struggling with a Neural Network if you have too little data or you don't have much data, which you won't have sometimes. For Example, a Naive Bayes or a Decision tree would deal really well when we are having less data.

- **Hard to interpret most of the times:** Most of the times a neural network will give us good results if we are using it for the right problem, but if it doesn't perform well, we will have a lot of trouble finding why it didn't go as expected, especially a Deep Neural network (which will be the case most of the times). For example, if we are trying to predict diabetes and we expected the output to be diabetes, but instead we got non-diabates, it would be quite hard to figure out why the neural network gave such an input, instead a traditional ML algorithm such as a Decision Tree, will be much more easy to interpret.

Diabetes Prediction Data Analysis Project

End of document