CSE 587: Data Intensive Computing

Project Phase #2

Heart Stroke Prediction

Team Members details

Name: Anurag Lingabathina
UB Name: anuragli
UB Number: 50442601
UB Mail: anuragli@buffalo.edu

Name: Chandra Kiran Alla
UB Name: calla2
UB Number: 50468307
UB Mail: calla2@buffalo.edu

List of Algorithms and Models

1.K-Nearest Neighbor
2.Naive Bayes
3.Support Vector Machines
4.Random Forest
5.Cat Boost
6.Logistic Regression
7.Decision Tree

1. K-Nearest Neighbor

KNN is a supervised algorithm used to classify or label objects/data points. It depends on labeled input data by learning a function that produces an appropriate output when new unlabeled data is given.

Intuition: For a given unlabeled element, look at just the k most similar elements in the labeled dataset based on various attributes, and choose the label that most of those elements have

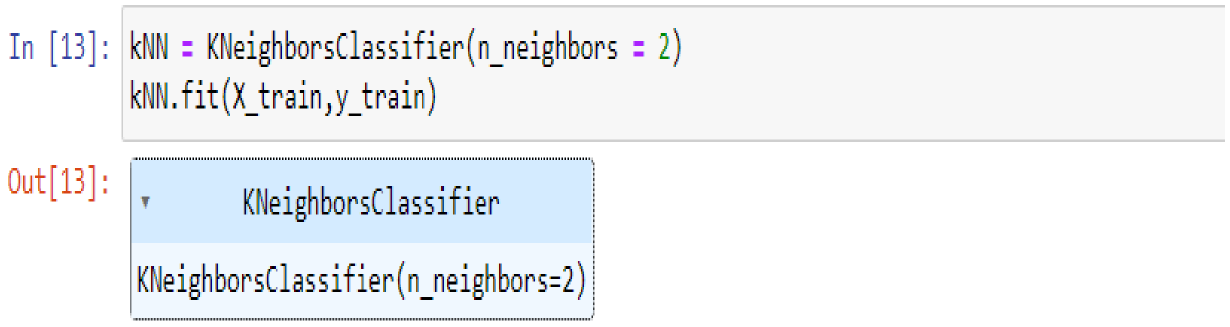
In this project/problem we look at k most similar patients data, if there are more than k/2 items labeled as stroke, then for new unlabeled data we label as stroke(1), but if not as no stroke(0)
1 - indicates Stroke
0 - indicates no Stroke

Reason to choose to KNN

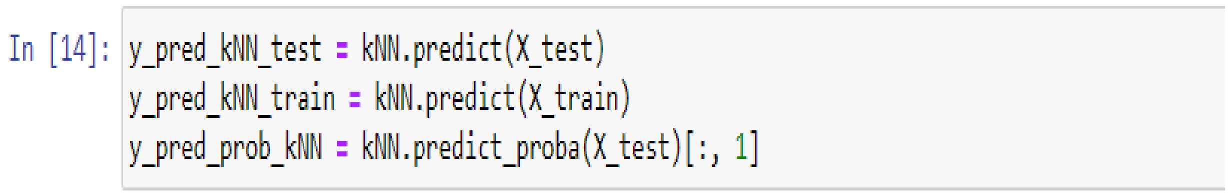
1. As we have labeled, it is easy to predict stroke or not, simply we can label unlabeled as 0 or 1. KNN outputs 1 or 0 which is required to predict stroke and it will never give outputs as NAN or any other values other than 1 or 0 for this data.
2. From above we can say the output is always noise-free
3.KNN also works well for smaller Data sets.
4. Works well for low-dimensionality spaces

KNN Algorithm

Step - 1: we clean data as required and load data
Step - 2: Now we choose K, to determine the no of neighbors we consider labeling the unlabelled data.
Step - 3: After choosing K, we calculate the distance between unlabeled data and labeled data
Step - 4: Sort the labeled data distance from low to high
Step - 5: Pick the first K entries from the sorted collection which are near to unlabeled data
Step - 6: Get the labels of the selected K entries
Step - 7: Select the label for unlabeled data based on step 6 output i.e if more than K/2 entries predicted stroke (example based on this project), then the unlabeled data is stroke positive. Else no stroke



```
In [13]: kNN = KNeighborsClassifier(n_neighbors = 2)
         kNN.fit(X_train,y_train)

Out[13]: ▾ KNeighborsClassifier
         KNeighborsClassifier(n_neighbors=2)
```



```
In [14]: y_pred_kNN_test = kNN.predict(X_test)
         y_pred_kNN_train = kNN.predict(X_train)
         y_pred_prob_kNN = kNN.predict_proba(X_test)[:, 1]
```

Effectiveness of the algorithm

We have Divided the data into training(80%) and testing (20%) of data. We trained the model using 80% data, later we predicted the stroke label using test data.

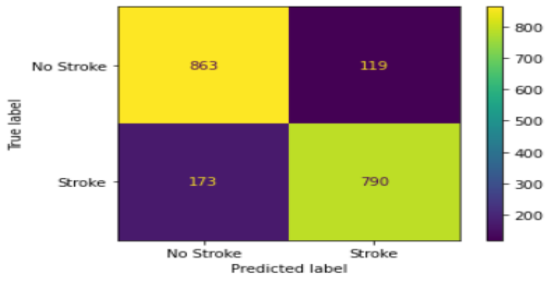
Below Table Explains Accuracy percentages for different values of K

K Value	Testing Accuracy	Training Accuracy	ROC AUC Score
2	0.8498714652956298	0.9331361707599332	0.9003908356650234
4	0.8750642673521851	0.9187347306159187	0.9155563380728502
5	0.8652956298200514	0.9008615147229009	0.9180149228163009
10	0.8524421593830335	0.8802880288028803	0.9198644130168578

For K =4 we have the best Accuracy on testing data i.e the unlabeled data and also ROC AUC score is 0.915 which is considered excellent(0.9-1)

When k =2

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22ee22839a0>
```



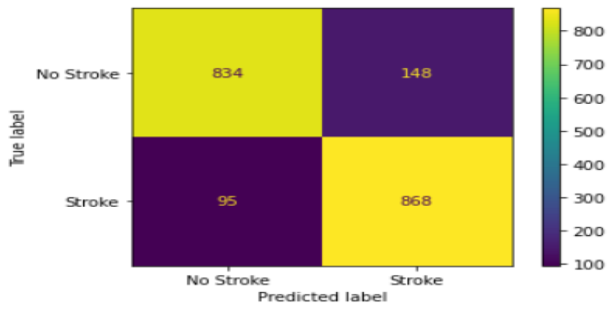
```
#Accuracy
print('Testing Accuracy:', accuracy_score(y_test, y_pred_kNN_test))
print('Training Accuracy:', accuracy_score(y_train, y_pred_kNN_train))
#ROC-AUC Score#
print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_kNN))
```

```
Testing Accuracy: 0.8498714652956298
Training Accuracy: 0.9331361707599332
ROC AUC Score: 0.9003908356650234
```

When k = 4

```
kNN = KNeighborsClassifier(n_neighbors = 4)
kNN.fit(X_train,y_train)

y_pred_kNN_test = kNN.predict(X_test)
y_pred_kNN_train = kNN.predict(X_train)
y_pred_prob_kNN = kNN.predict_proba(X_test)[:, 1]

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

confma = confusion_matrix(y_test, y_pred_kNN_test)
cmd = ConfusionMatrixDisplay(confma, display_labels=['No Stroke','Stroke'])
cmd.plot()

#Accuracy
print('Testing Accuracy:', accuracy_score(y_test, y_pred_kNN_test))
print('Training Accuracy:', accuracy_score(y_train, y_pred_kNN_train))
#ROC-AUC Score#
print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_kNN))
```

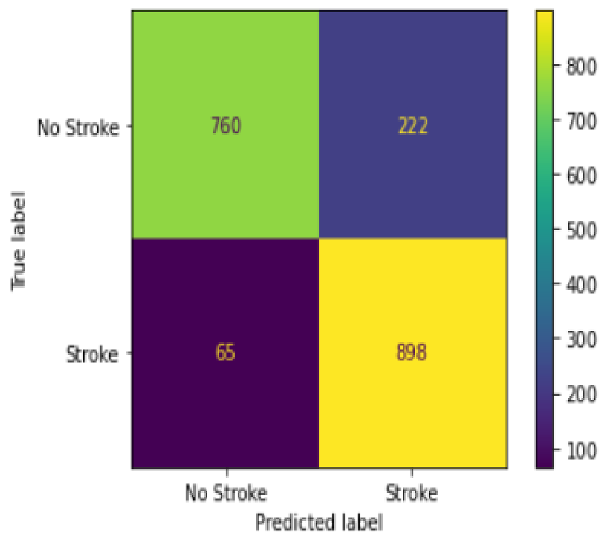
```
Testing Accuracy: 0.8750642673521851
Training Accuracy: 0.9187347306159187
ROC AUC Score: 0.9155563380728502
```



When k =10

```
kNN = KNeighborsClassifier(n_neighbors = 10)
kNN.fit(X_train,y_train)

y_pred_kNN_test = kNN.predict(X_test)
y_pred_kNN_train = kNN.predict(X_train)
y_pred_prob_kNN = kNN.predict_proba(X_test)[:, 1]

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

confma = confusion_matrix(y_test, y_pred_kNN_test)
cmd = ConfusionMatrixDisplay(confma, display_labels=['No Stroke','Stroke'])
cmd.plot()

#Accuracy
print('Testing Accuracy:', accuracy_score(y_test, y_pred_kNN_test))
print('Training Accuracy:', accuracy_score(y_train, y_pred_kNN_train))
#ROC-AUC Score#
print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_kNN))
```

```
Testing Accuracy: 0.8524421593830335
Training Accuracy: 0.8802880288028803
ROC AUC Score: 0.9198644130168578
```



Now as the model is trained and tested using data, we can say the model is able to gain intelligence so that it can predict whether a patient gets a stroke or not with an accuracy of 87.5%.

At K=4, we have an accuracy of 87.50%

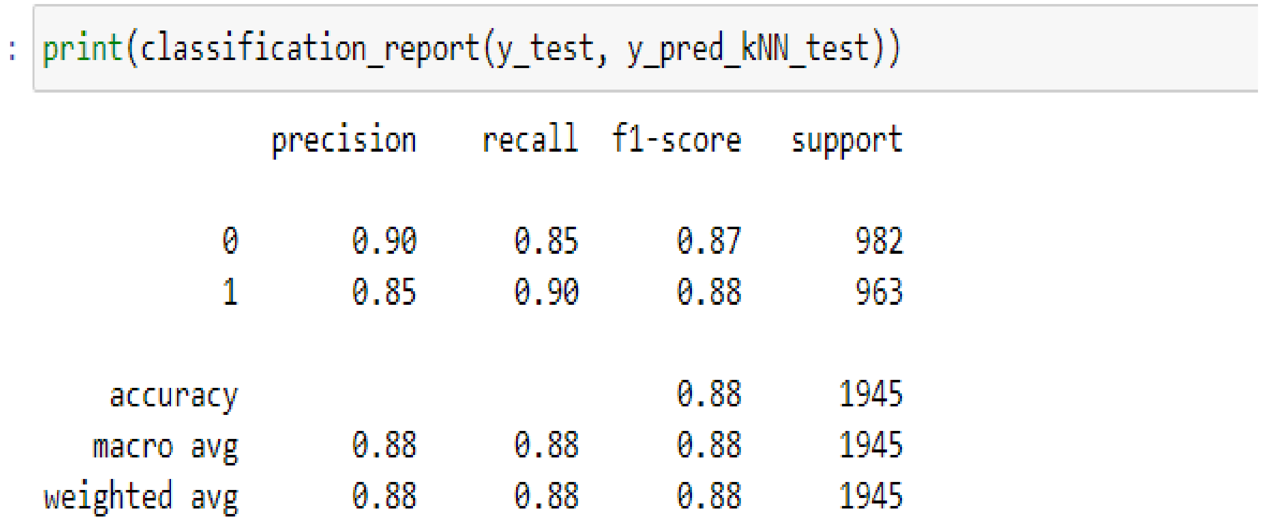
We have tested with 20% of data i.e 1945 rows. the model predicted 1702 correctly (834 Patients have no chance of stroke and 868 patients are likely to get stroke) and it wrongly predicted 95 samples as having no stroke but are likely to get stoke, at the same time it predicted 148 people might be a stroke but truly they will not get stroke

As False negative is less than False Positive, we can say we are predicting more corrections than wrongs.

From the Confusion matrix, we can get more details like precision, recall and f1-score

Precision = TP/(TP+FP)
Recall = TP/(TP+FN)

Below Diagram gives more about the precision and recall



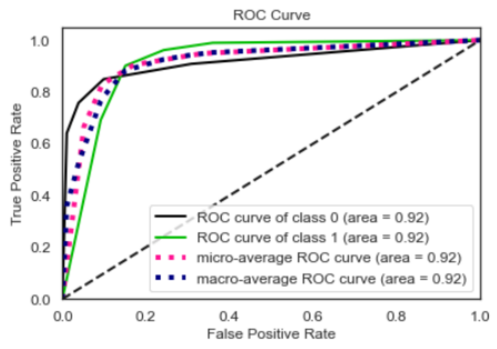
```
: print(classification_report(y_test, y_pred_kNN_test))
```

```
              precision    recall  f1-score   support

           0       0.90      0.85      0.87       982
           1       0.85      0.90      0.88       963

    accuracy                           0.88      1945
   macro avg       0.88      0.88      0.88      1945
weighted avg       0.88      0.88      0.88      1945
```

ROC CURVE

Below graph displaying model's performance over all thresholds. Two parameters are plotted on this curve: % True Positive. Rate of False Positives
In General if the ROC Curve is more than 0.9 it is considered outstanding.

```
skplt.metrics.plot_roc(y_test, kNN.predict_proba(X_test))
plt.title('ROC Curve')
plt.show()
```



2.Naive Bayes

Naive Bayes Algorithm is based on Bayes Theorem and we assume that predictors are independent of each other. In simple words, we can say Navies Bayes assumes that the features are independent of each other and that the presence of one feature doesn't affect the other.

Intuition: Make a probabilistic model – have many simple rules, and aggregate those rules together to provide a probability.

Basic principle: P(H | E) = P(E | H) * P(H) / P(E)
Posterior probability is proportional to likelihood times prior

- H – hypothesis E – evidence
- Prior = probability of the E given H; P(E | H)
- Likelihood = P(H) / P(E)
- Posterior = Probability of H given E; P(H | E)

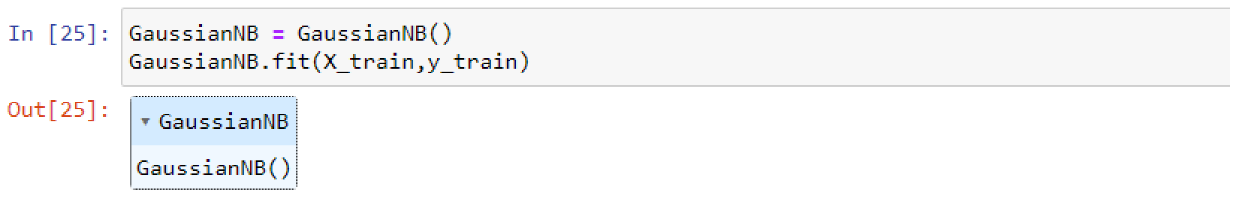
Working of Naive Bayes

We have a patient training data set with a stroke goal variable (suggesting possibilities of getting Stroke). Based on the patient's health conditions, we must now evaluate whether the patient will experience a stroke or not.

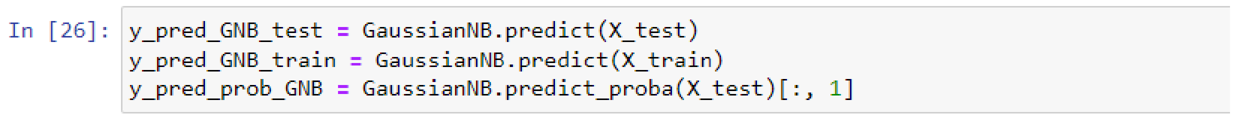
We first find the frequency of each variable, we can say we are building a frequency table.

Now we find the probability of each variable that is we are building a likelihood table

Finally Naive Bayes equation to find the Posterior probability for each class. The output or prediction is nothing but a higher probability class.



```
In [25]: GaussianNB = GaussianNB()
         GaussianNB.fit(X_train,y_train)

Out[25]: ▾ GaussianNB
         GaussianNB()
```



```
In [26]: y_pred_GNB_test = GaussianNB.predict(X_test)
         y_pred_GNB_train = GaussianNB.predict(X_train)
         y_pred_prob_GNB = GaussianNB.predict_proba(X_test)[:, 1]
```

Reason to choose Naive Bayes

1. Unlike other Models or Algorithms, Naive Bayes requires fewer data and performs better due to the independent ness of variables.
2. Naive Bayes Performs well even with multi-class predictions and it is fast as well as easy to train and test the model.
3. In the data set, we even had few categorical data as well as numerical data, so Naive Bayes performs well with both numerical and categorical

Effectiveness of the algorithm

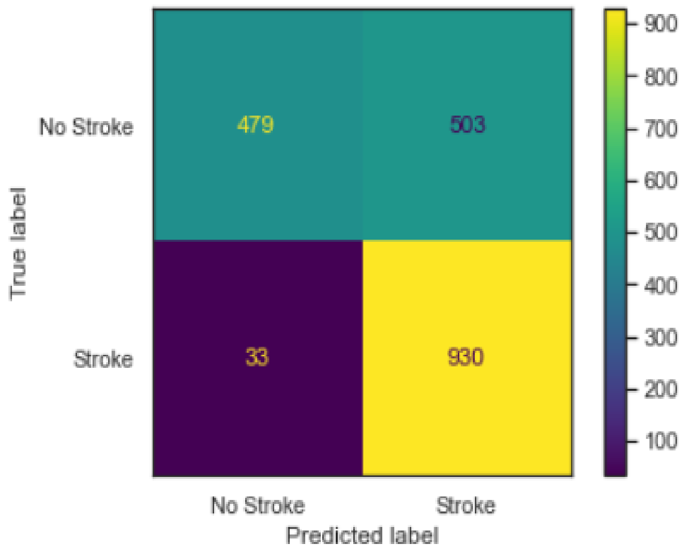
We have Divided the data into training(80%) and test(20%) of data. We trained the model using 80% data, later we are finding the probability of getting a stroke using test data.

Below is the Confusion matrix of Naive Bayes. Here we can understand that the False Positives are more i.e the model predicted that the chances of getting a stroke are more for 503 patients but in reality, these patients might not get the stroke. But the False Negative predictions are very less, only 33 means the model predicted that 33 patients never get strokes but in reality, these patients might be more likely affected by stroke.

Here we are more interested in False Negative than False Position as even if the model predicted the patient gets a stroke but if he is not affected it is good if models predicted patients are never likely to get affected by stroke but get it is not good as the patient will not take medicine to avoid stroke.

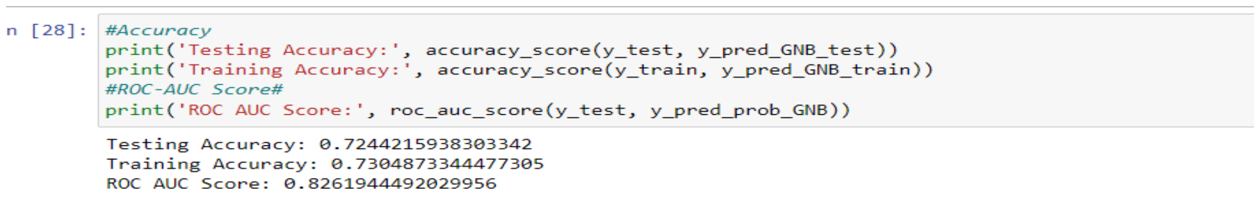
```
: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

  confgn = confusion_matrix(y_test, y_pred_GNB_test)
  cmd = ConfusionMatrixDisplay(confgn, display_labels=['No Stroke','Stroke'])
  cmd.plot()
```

```
: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22ee2d5d340>
```

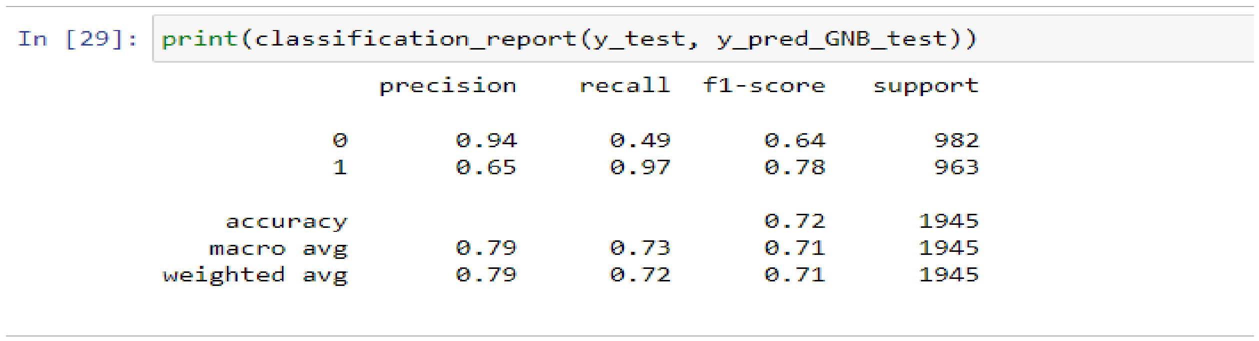


Below are details about accuracy using Naive Bayes, the accuracy is less when we compare KNN due to more False Positive but False Negative is good here compared to KNN.

Testing Accuracy: 0.7244215938303342
Training Accuracy: 0.7304873344477305
ROC AUC Score: 0.8261944492029956

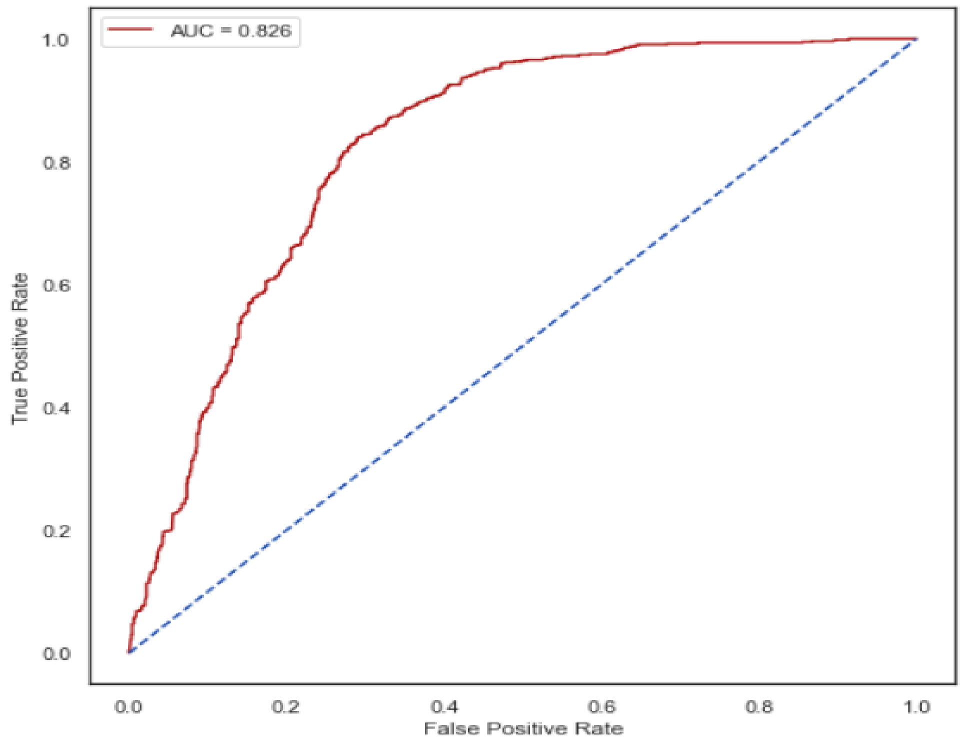


```
n [28]: #Accuracy
        print('Testing Accuracy:', accuracy_score(y_test, y_pred_GNB_test))
        print('Training Accuracy:', accuracy_score(y_train, y_pred_GNB_train))
        #ROC-AUC Score#
        print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_GNB))

        Testing Accuracy: 0.7244215938303342
        Training Accuracy: 0.7304873344477305
        ROC AUC Score: 0.8261944492029956
```



```
In [29]: print(classification_report(y_test, y_pred_GNB_test))

                      precision    recall  f1-score   support

                   0       0.94      0.49      0.64       982
                   1       0.65      0.97      0.78       963

            accuracy                           0.72      1945
           macro avg       0.79      0.73      0.71      1945
        weighted avg       0.79      0.72      0.71      1945
```

Below graph displaying a classification model's performance overall thresholds. Two parameters are plotted on this curve: % True Positive. Rate of False Positives
In General if the ROC Curve is more than 089 it is considered good.



3. Support Vector Machines [SVM]

Support Vector Machines [SVM] is a supervised learning algorithm used for classification problems.

The goal of the support vector machine [SVM] algorithm is finding a hyperplane in N-dimensional(N-D) Space (where N= number of features) that categorizes the Data points more clearly

There are many different hyperplanes used to divide between classes of data points.
The goal is to find a plane with max margin, that is maximum separation between data points from both classes. Maximum the margin distance gives more support and increases the confidence with which future data points are categorized or predicted.

Intuition

In SVM, the output of the linear function is taken into consideration. If the output is more than 1, it is associated with one class, and if it is less than 1, it is associated with a different class. We get this reinforcement range of values ([-1,1]) that serves as a margin because the threshold values in SVM are altered to 1 and -1.

Working of SVM

Each data is plotted in an N-Dimensional plane where the coordinate can be defined as the value of each feature. Now we find the right hyperplane that separates two classes to look different or non-identical and perform classification on data

In SVM the vectors are nothing but the values or coordinates of each data point. In SVM we divide the two classes by a line.

```
In [31]: SVM = SVC(probability = True)
         SVM.fit(X_train,y_train)

Out[31]:  ▾         SVC
         SVC(probability=True)
```

```
In [32]: y_pred_SVM_test = SVM.predict(X_test)
         y_pred_SVM_train = SVM.predict(X_train)
         y_pred_prob_SVM = SVM.predict_proba(X_test)[:, 1]
```

Reason to choose SVM

1. SVM really worlds well to separate the decisions
2. We can save the memory as SVM internally uses the few training points used during the decision function phase.
3. SVM is more efficient when the number of samples are less than the number of dimensions present in the data set.
4. When we have N-Dimensions, SVM is more efficient than other models

Effectiveness of the algorithm

We have Divided the data into training(80%) and testing (20%) of data. We trained the model using 80% data, later we are finding the probability of getting a stroke using test data.

By using SVM, we achieved 76% of accuracy during the training phase and 76% of accuracy when we tested the model using unseen data.

```
In [35]: #Accuracy
         print('Testing Accuracy:', accuracy_score(y_test, y_pred_SVM_test))
         print('Training Accuracy:', accuracy_score(y_train, y_pred_SVM_train))
         #ROC-AUC Score#
         print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_SVM))

         Testing Accuracy: 0.7593830334190231
         Training Accuracy: 0.7635334962067636
         ROC AUC Score: 0.8425818417919223
```

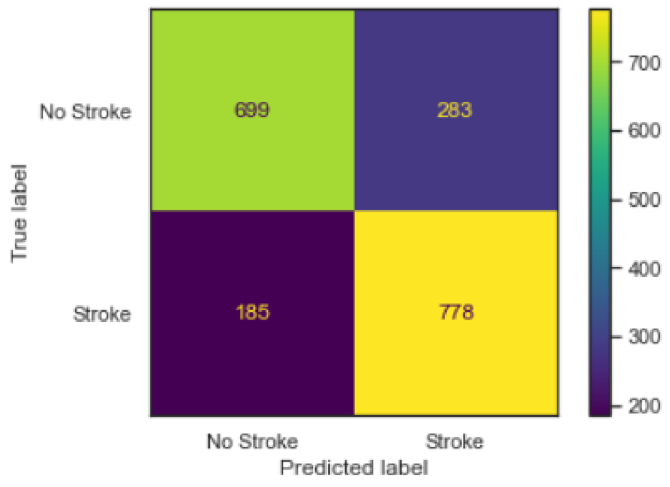
The model accuracy is decreased compared to KNN because of more False Negative and False Positive predictions by the Algorithm.

False Negative and False Positive accounted for 25% of overall test Data and also False Negatives are more. It is not good to predict that a patient is not likely to get a stroke but in reality, he gets a stroke due to the patient not giving attention to his health and will not take any treatment for this, so will face serious health conditions in the future if we predict using this model.

Below is the **confusion matrix** of SVM

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

confsv = confusion_matrix(y_test, y_pred_SVM_test)
cmd = ConfusionMatrixDisplay(confsv, display_labels=['No Stroke','Stroke'])
cmd.plot()
```

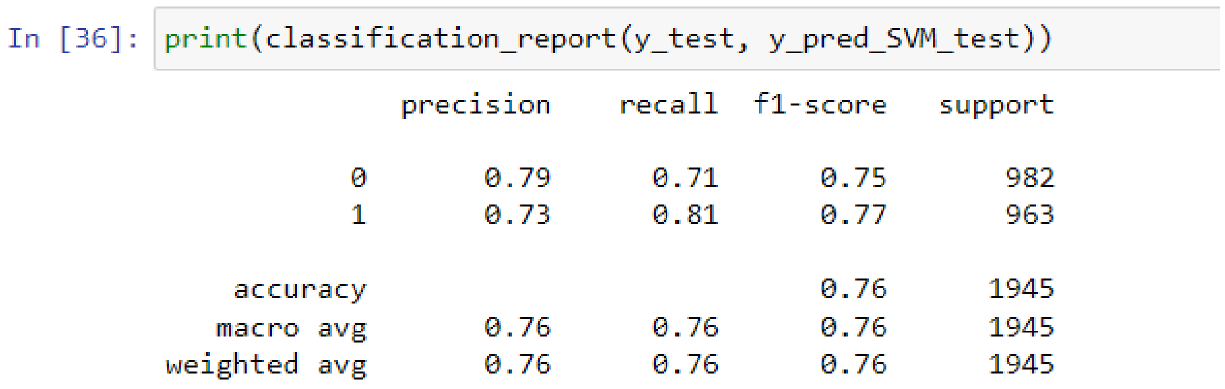
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22ee2b8cfd0>
```



From Confusion we can get more details like precision, recall, and f1-score

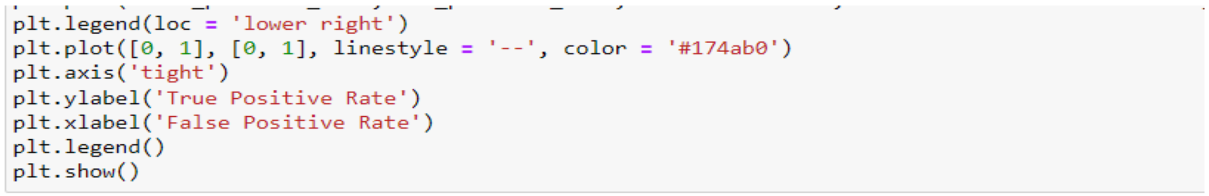
Precision = TP/(TP+FP)
Recall = TP/(TP+FN)

From recall and precision we can tell that there are more False Positive and False Negatives
Below Diagram gives more about precision and recall

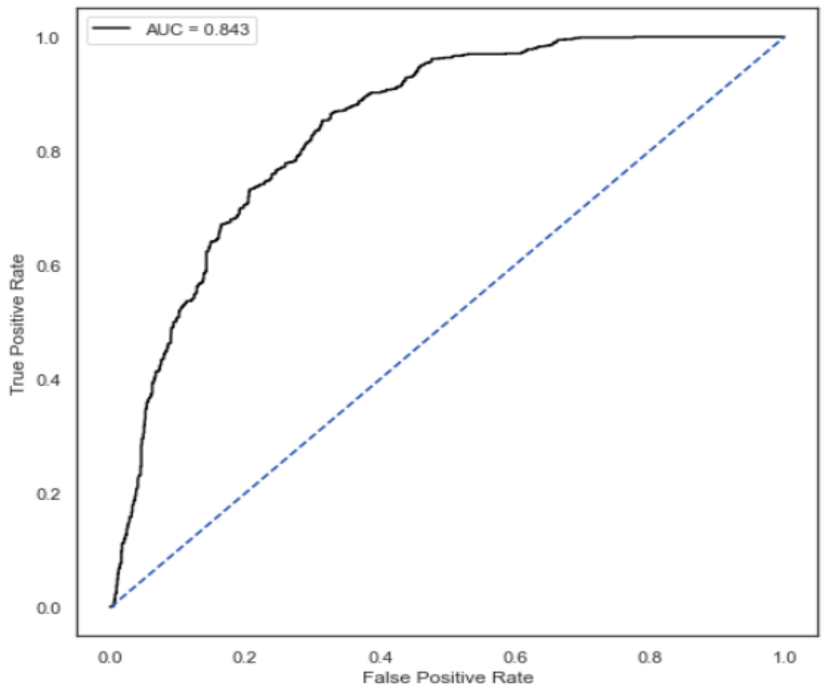


```
In [36]: print(classification_report(y_test, y_pred_SVM_test))

              precision    recall  f1-score   support

           0       0.79      0.71      0.75       982
           1       0.73      0.81      0.77       963

    accuracy                           0.76      1945
   macro avg       0.76      0.76      0.76      1945
weighted avg       0.76      0.76      0.76      1945
```

ROC Curve - Below graph displaying model's performance overall thresholds. Two parameters are plotted on this curve: % True Positive. Rate of False Positives
In General if the ROC Curve is more than 0.8 it is considered good.



```
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



Now as the model is trained and tested using data, we can say the model is able to gain intelligence so that it can predict whether a patient gets a stroke or not with an accuracy of 76% and also more likely there can be more False negatives predictions which do not hold good.

4. Random Forest

An effective supervised learning method is Random Forest, a well-known machine learning algorithm. Both Classification and Regression issues in ML can be solved with it. It is based on the idea of ensemble learning, which is the act of integrating different classifiers to address a complicated issue and enhance the performance of the model.

Intuition

The bias is unchanged because we are using several decision trees, identical to that of a single decision tree. However, when the variance declines, we reduce the likelihood of overfitting. In The Curse of Bias and Variance, I provide an intuitive explanation of bias and variance.

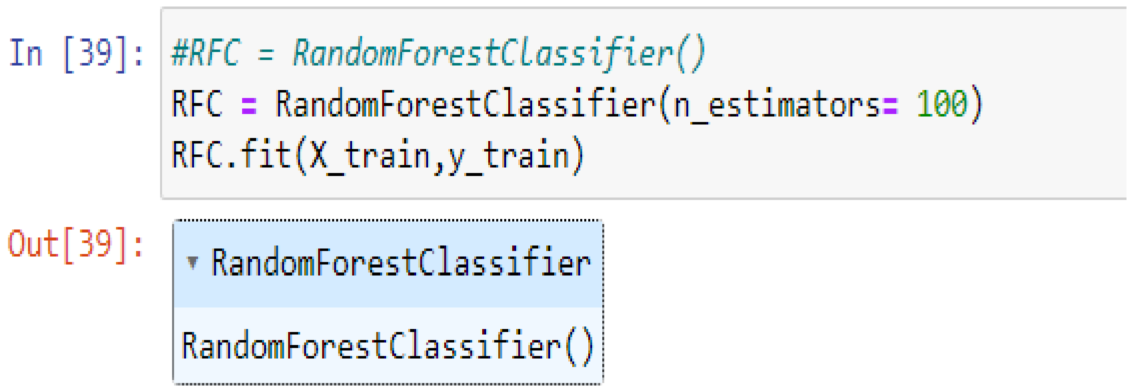
The random forest comes to the rescue when all that matters are the predictions and you want a quick and dirty way out. Regarding the model's assumptions or the dataset's linearity, you don't need to worry too much.

The random forest makes forecasts based on the majority votes of projections rather than depending on a single decision tree to determine the outcome.An increase in accuracy and a solution to the overfitting issue are provided by the larger number of trees in the forest.

Working of Random Forest

The random forest algorithm's steps are as follows:

Step 1: From a data set with 'P' records, 'K' random records are selected at random and used in the Random Forest algorithm.
Step 2: For each sample, a unique decision tree is built.
Step 3: An output will be produced by each decision tree.
Step 4: For Classification and Regression, the Final Output is Based on Majority Voting or Averaging, accordingly.



```
In [39]: #RFC = RandomForestClassifier()
         RFC = RandomForestClassifier(n_estimators= 100)
         RFC.fit(X_train,y_train)

Out[39]: ▾ RandomForestClassifier
         RandomForestClassifier()
```

Reason to choose Random Forest

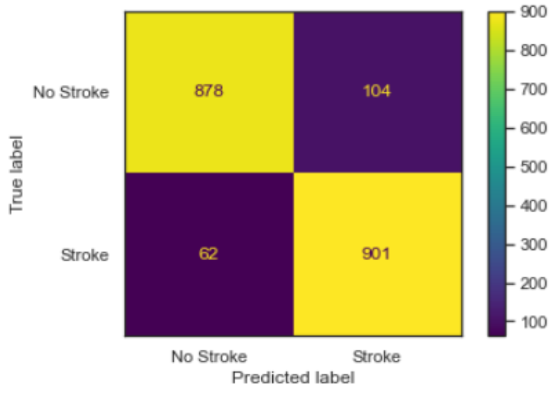
1. The random forest has typically very high accuracy.
2. Large Data sets are particularly notable for their efficiency.
3. Estimation of key classification-relevant variables
4. Forests produced can be maintained and used again.
5. Unlike competing models, It is not overfitting with extra features.

Effectiveness of Random Forest

We Divided the data into training(80%) and test(20%) of data. We trained the model using 80% data, later we are finding the probability of getting a stroke using test data.

Below is the Confusion matrix which gives more details about the efficiency of the model. Here we can clearly see that the model has more True Positive and True Negative than False Positive and False Negative. This says that the model had good efficiency that is 98% during the training phase and 91% during testing phase.

```
In [41]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

         confsv = confusion_matrix(y_test, y_pred_RFC_test)
         cmd = ConfusionMatrixDisplay(confsv, display_labels=['No Stroke','Stroke'])
         cmd.plot()

Out[41]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22ee2b87b20>
```



From Confusion matrix, we can get more details like precision, recall, and f1-score

Precision = TP/(TP+FP)
Recall = TP/(TP+FN)

From recall and precision we can tell that there are less False Positive and False Negatives
Below Diagram gives more about precision and recall

```
#Accuracy
print('Testing Accuracy:', accuracy_score(y_test, y_pred_RFC_test))
print('Training Accuracy:', accuracy_score(y_train, y_pred_RFC_train))
#ROC-AUC Score#
print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_RFC))
```

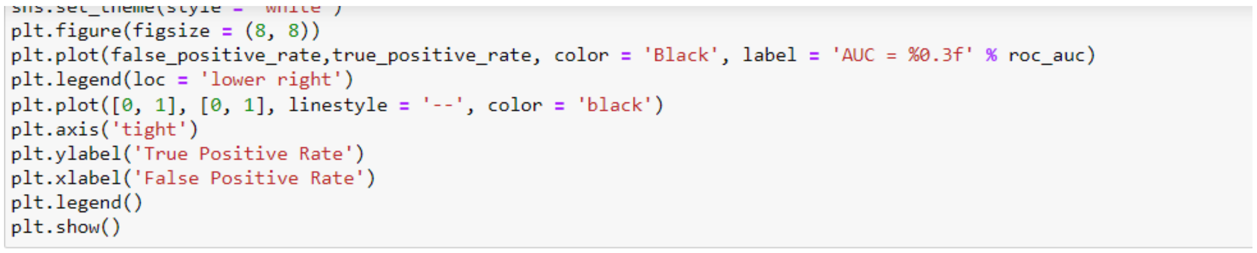
```
Testing Accuracy: 0.9146529562982005
Training Accuracy: 0.9854699755689855
ROC AUC Score: 0.9643732565197437
```

```
print(classification_report(y_test, y_pred_RFC_test))
```

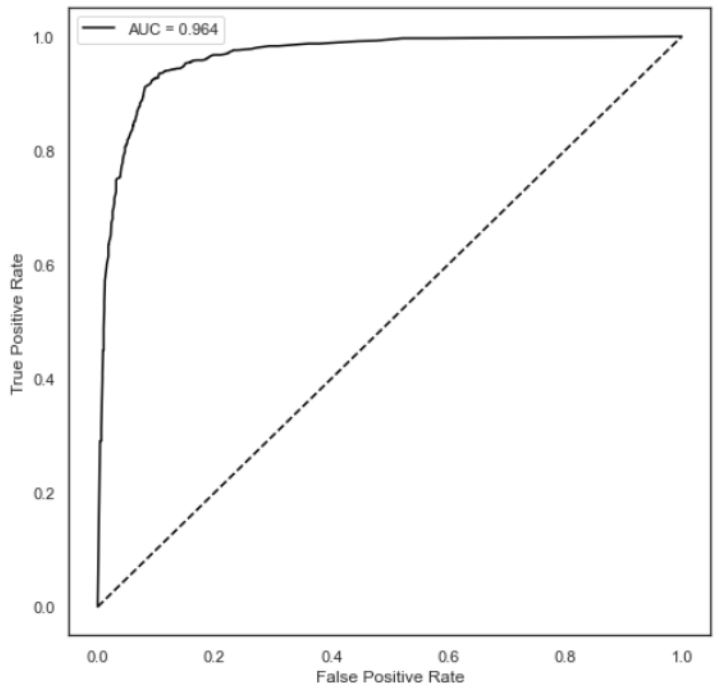
```
              precision    recall  f1-score   support

           0       0.93      0.89      0.91       982
           1       0.90      0.94      0.92       963

    accuracy                           0.91      1945
   macro avg       0.92      0.91      0.91      1945
weighted avg       0.92      0.91      0.91      1945
```

ROC CURVE

Below graph displaying model's performance over all thresholds. Two parameters are plotted on this curve: % True Positive. Rate of False Positives
we have the best Accuracy on testing data i.e the unlabeled data and also ROC AUC score is 0.964 which is considered excellent(0.9-1)



```
sns.set_theme(style = 'white')
plt.figure(figsize = (8, 8))
plt.plot(false_positive_rate,true_positive_rate, color = 'Black', label = 'AUC = %0.3f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], linestyle = '--', color = 'black')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



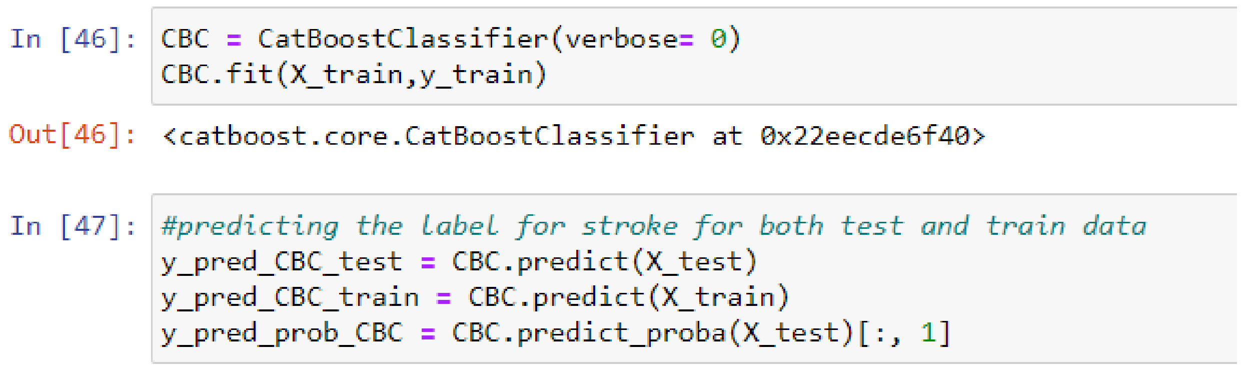
5.Cat Boost

For gradient boosting on decision trees, there is a method called CatBoost. Engineers and researchers from Yandex created it.

In order to solve classification and regression issues, gradient boosting, an ensemble machine learning approach, can be utilized regularly. Even with relatively minimal data, it functions well with heterogeneous data and is simple to use. By combining numerous poor learners, it effectively produces a powerful learner.

Intuition

CatBoost is mostly used because it is user-friendly, effective, and particularly effective with categorical variables.
Without considerable hyper-parameter adjustment, CatBoost generates good results. To achieve better results, CatBoost can be tweaked for a few key factors.
Interfaces are user-friendly using CatBoost.



```
In [46]: CBC = CatBoostClassifier(verbose= 0)
         CBC.fit(X_train,y_train)

Out[46]: <catboost.core.CatBoostClassifier at 0x22eecde6f40>

In [47]: #predicting the label for stroke for both test and train data
         y_pred_CBC_test = CBC.predict(X_test)
         y_pred_CBC_train = CBC.predict(X_train)
         y_pred_prob_CBC = CBC.predict_proba(X_test)[:, 1]
```

Reason to choose Cat Boost

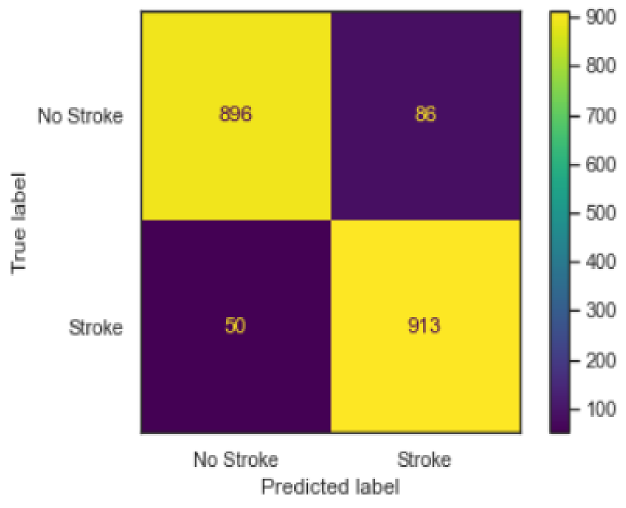
1.Cat Boost is useful in generating good results with parameters that are default.
2.Using CatBoost's approach, we can train models rapidly and effectively, even for latency tasks.
3.It helps in improving accuracy, as it reduces overfitting.
Compared to many other machine learning methods, CatBoost is exceptionally quick.
4.On GPU and CPU, the splitting, tree building, and training processes have been sped up. Training on the GPU is 20 times quicker than XGBoost.
5.Instead of pre-processing your data or spending time and effort putting it into numbers, we can improve the training outcomes with CatBoost, which enables us to utilize non-numeric factors.

Effectiveness of Cat Boost

We have Divided the data into training(80%) and testing (20%) of data. We trained the model using 80% data, later we are finding the probability of getting a stroke using test data.

Below is the Confusion matrix which gives more details about the efficiency of the model. Here we can clearly see that the model has more True Positive and True Negative than False Positive and False Negative. This says that the model had good efficiency that is 94% during the training phase and 93% during testing phase.

```
In [48]: #displaying confusion matrix
         from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

         confsv = confusion_matrix(y_test, y_pred_CBC_test)
         cmd = ConfusionMatrixDisplay(confsv, display_labels=['No Stroke','Stroke'])
         cmd.plot()

Out[48]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22eecddf1f0>
```



From Confusion matrix, we can get more details like precision, recall, and f1-score

Precision = TP/(TP+FP)
Recall = TP/(TP+FN)

From recall and precision we can tell that there are less False Positive and False Negatives

Below Diagram gives more about precision and recall

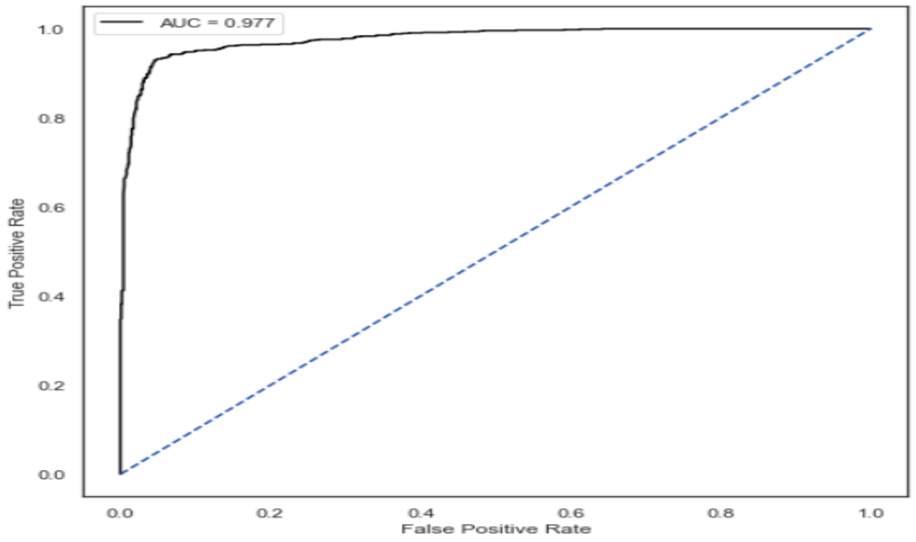
```
In [49]: #Accuracy
         print('Testing Accuracy:', accuracy_score(y_test, y_pred_CBC_test))
         print('Training Accuracy:', accuracy_score(y_train, y_pred_CBC_train))
         #ROC-AUC Score#
         print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_CBC))

         Testing Accuracy: 0.9300771208226221
         Training Accuracy: 0.9474090266169474
         ROC AUC Score: 0.9773789054486467
```

```
In [50]: #getting values for precision , recall and f1-score
         print(classification_report(y_test, y_pred_CBC_test))

                       precision    recall  f1-score   support

                    0       0.95      0.91      0.93       982
                    1       0.91      0.95      0.93       963

             accuracy                           0.93      1945
            macro avg       0.93      0.93      0.93      1945
         weighted avg       0.93      0.93      0.93      1945
```

ROC CURVE

Below graph displaying model's performance over all thresholds. Two parameters are plotted on this curve: % True Positive. Rate of False Positives
we have the best Accuracy on testing data i.e the unlabeled data and also ROC AUC score is 0.977 which is considered excellent(0.9-1)

```
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



6.Logistic Regression

Logistic Regression is a Statistical model used for calculating the odds of an event happening vs other possibilities.

The math behind Logistic Regression

Odds Ratio: $p \ / 1 - p$

Here the odds are getting stroke and no stroke

The logit function is the basic building block of Logistic Regression

$$logit(p) = log(p/1 - p) = log(p) - log(1 - p)$$

It takes an x value in the range [0,1] (ie a probability) and transforms it to y values ranging across all real numbers

The inverse of the logit function, therefore, takes a real number, and maps it to a result in the range [0,1]

$$logit^{-1}(t) = e^{t}/(1 + e^{t})$$

Finally we can use this to get the probability

$$P(C|x) = [logit^{-1}(\alpha + \beta^{T}x)]^{C} \ * \ [1 - logit^{-1}(\alpha + \beta^{T}x)]^{1-C}$$

c is the class (or label) and x is the vector of features

Finally we are converting it to fit the model and find $\alpha \ and \ \beta$ values

$$logit(P(c = 1|x)) = \alpha + \beta^{T}x$$

We have implemented logistic regression using SciKit Learn in python. Here we are calling LogisticRegression and giving training and testing data to the function to fit the model, and later predict the stroke variable.

```
In [52]: LOR = LogisticRegression()
         LOR.fit(X_train,y_train)

Out[52]: ▾ LogisticRegression
         LogisticRegression()
```

```
In [53]: y_pred_LOR_test = LOR.predict(X_test)
         y_pred_LOR_train = LOR.predict(X_train)
         y_pred_prob_LOR = LOR.predict_proba(X_test)[:, 1]
```

Reason to choose Logistic Regression

1. It is easy to extend the model if new parameters or features are added in later stages of application.
2.As the data taken has many unknown values , it is easy to for Logistic regression to classify the unknown records
3.For less dimensional data (in this data set we have less than 10 dimensions) the overfitting issue occurrence is very less
4.we can easily train logistic regression in less time and it is easy to interpret.

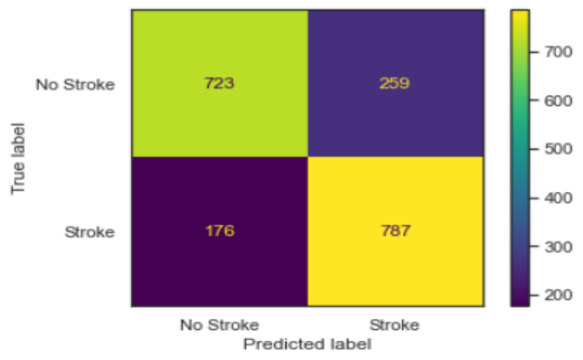
Effectiveness of the algorithm

We have Divided the data into training(80%) and test(20%) of data . We trained the model using 80% data , later we are finding the probability of getting a stroke using test data.
By using logistic regression , we have achieved 79% of accuracy during the training phase and 77% of accuracy when we test the model using unseen data.

```
In [55]: #Accuracy
         print('Testing Accuracy:', accuracy_score(y_test, y_pred_LOR_test))
         print('Training Accuracy:', accuracy_score(y_train, y_pred_LOR_train))
         #ROC-AUC Score#
         print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_LOR))

         Testing Accuracy: 0.7763496143958869
         Training Accuracy: 0.7915648707727916
         ROC AUC Score: 0.8557186152404761
```

The Accuracy is affected the model gave more False Negative and False Positives which accounted for 22% of predictions.

```
In [54]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

         confsv = confusion_matrix(y_test, y_pred_LOR_test)
         cmd = ConfusionMatrixDisplay(confsv, display_labels=['No Stroke','Stroke'])
         cmd.plot()

Out[54]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22eed0e0b50>
```



From the Confusion matrix, we can get more details like precision, recall and f1-score

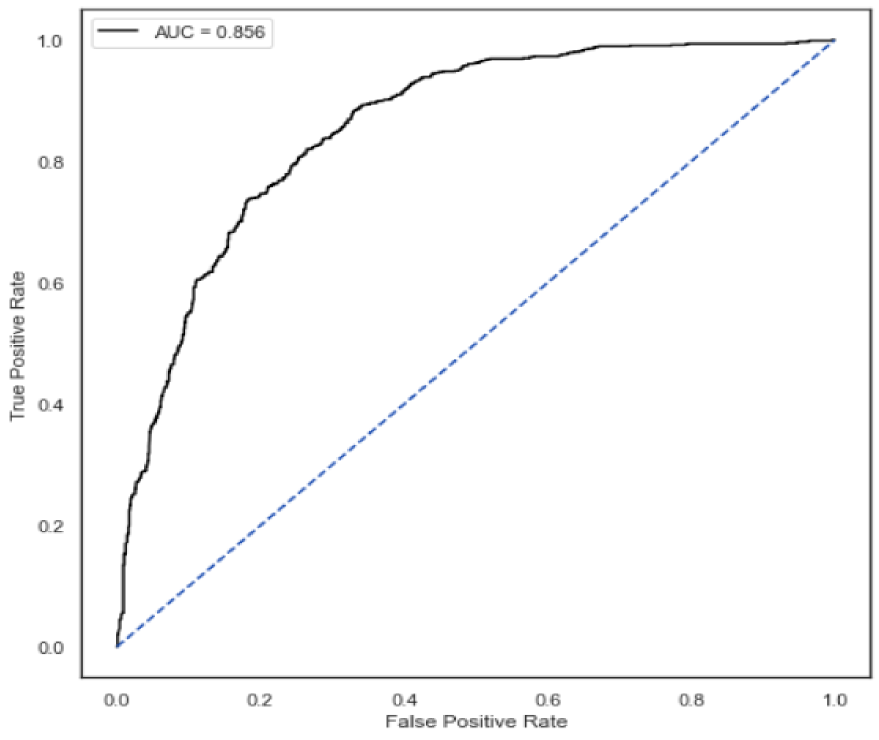
Precision = TP/(TP+FP)
Recall = TP/(TP+FN)

Below Diagram gives more about the precision and recall

```
In [56]: print(classification_report(y_test, y_pred_LOR_test))

              precision    recall  f1-score   support

           0       0.80      0.74      0.77       982
           1       0.75      0.82      0.78       963

    accuracy                           0.78      1945
   macro avg       0.78      0.78      0.78      1945
weighted avg       0.78      0.78      0.78      1945
```

ROC CURVE

Below graph displaying the model's performance overall thresholds. Two parameters are plotted on this curve: % True Positive. Rate of False Positives
In General if the ROC Curve is more than 0.8 it is considered good



.

7. Decision Tree

Decision tree is a supervised machine learning model that trains the models, using labeled input and output datasets. The objective is to learn decision rules derived from the data features to build a model that predicts the value of a target variable. Decision trees are among the quickest models for both classification and regression. It can be learned with less data than other machine learning models.

Intuition:

A Decision tree works with input and output variables that are categorical or continuous (regression). According to the most important differentiator or splitter in the input variables, we divide the population or sample into two or more homogeneous groups (or sub-populations) with this method.

Depending on the kind of target variable we have, several decision trees can be used. It comes in two varieties:

Decision Tree with a Categorical Target Variable: When a decision tree has a categorical target variable, it is referred to as a Categorical Variable Decision Tree.
Decision Tree with Continuous Target Variable: A decision tree with a continuous target variable is referred to as a Continuous Variable Decision Tree.

Decision trees, one should use caution, though, as overfitted answers can quickly be produced using decision trees. This refers to a solution that only generalizes well to fresh data samples used in its training. It is interpretable, Fast, Simple, and Logical.

Working of Decision Tree

Decision trees is a type of predictive modeling that can be used to map several options or solutions to a specific result. Different nodes make up decision trees. The decision tree's root node, which in machine learning typically represents the entire dataset, is where it all begins. The leaf node is the branch's termination point or the result of all previous decisions. The decision tree won't branch out from a leaf node. With decision trees in machine learning, the core nodes represent the data's features, and the leaf nodes represent the results.

The highest node in the decision tree, or the root node, represents the complete message or choice.

A node in a decision tree known as a decision (or internal) node is one where the preceding node branches into two or more variables.

Leaf (or terminal) node: The leaf node, also known as the external node or terminal node, is the final node in the decision tree and is located the farthest out from the root node. It has no children.

The action of splitting a node into two or more nodes It is the section where the decision splits into variables. Pruning is the process of going through and narrowing the tree down to only the most significant nodes or outcomes. It is the reverse of splitting.

```
In [58]: DTC = DecisionTreeClassifier()
         DTC.fit(X_train,y_train)

Out[58]: ▾ DecisionTreeClassifier
         DecisionTreeClassifier()
```

```
In [59]: y_pred_DTC_test = DTC.predict(X_test)
         y_pred_DTC_train = DTC.predict(X_train)
         y_pred_prob_DTC = DTC.predict_proba(X_test)[:, 1]
```

Reason to choose Decision Tree

1. Decision trees take less work to prepare the data during pre-processing than other methods do.
2. Data normalization is not necessary for a decision tree.
3. Scaling of data is not necessary when using a decision tree.
4. Additionally, we have missing data in the data set, the construction of a decision tree is not significantly impacted by missing values in the data.

Effectiveness of the algorithm

We have Divided the data into training(80%) and testing (20%) of data. We trained the model using 80% data, later we are finding the probability of getting a stroke using test data.

The model achieved 98% accuracy during the training Phase and 91% accuracy during the testing phase. We have tested the Model with 20% of data i.e 1945 rows. the model predicted 1772 correctly (882 Patients have no chance of stroke and 890 patients are likely to get stroke) and it wrongly predicted 73 samples as having no stroke but are likely to get stoke, at the same time it predicted 100 people might be a stroke but truly they will not get stroke

```
In [61]: #Accuracy
         print('Testing Accuracy:', accuracy_score(y_test, y_pred_DTC_test))
         print('Training Accuracy:', accuracy_score(y_train, y_pred_DTC_train))
         #ROC-AUC Score#
         print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_DTC))

         Testing Accuracy: 0.9110539845758355
         Training Accuracy: 0.9854699755689855
         ROC AUC Score: 0.9172493248144693
```

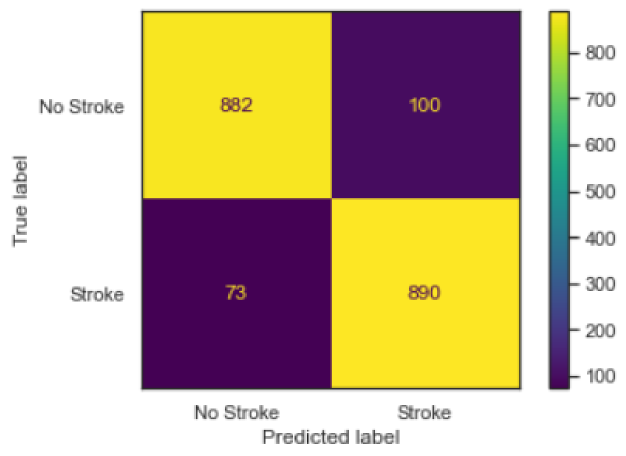
As False negative is less than False Positive, we can say we are predicting more corrections than wrongs.

Confusion Matrix

Below is the Confusion matrix of the Decision Tree. Here we can understand that the False Positives and False Negative predictions are very well balanced. Also, True Positives and True negatives are more which is good for predicting unseen data by the algorithm.

Here we are more interested in False Negative than False Position as even if the model predicted the patient gets a stroke but if he is not affected it is good if models predicted patients are never likely to get affected by stroke but get it is not good as the patient will not take medicine to avoid stroke.

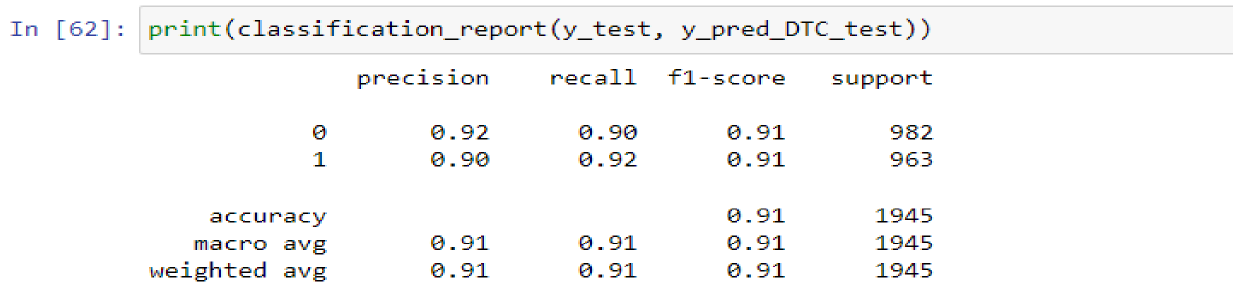
```
In [60]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

         confsv = confusion_matrix(y_test, y_pred_DTC_test)
         cmd = ConfusionMatrixDisplay(confsv, display_labels=['No Stroke','Stroke'])
         cmd.plot()

Out[60]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22eed025580>
```



From Confusion we can get more details like precision , recall and f1-score

Precision = TP/(TP+FP)
Recall = TP/(TP+FN)

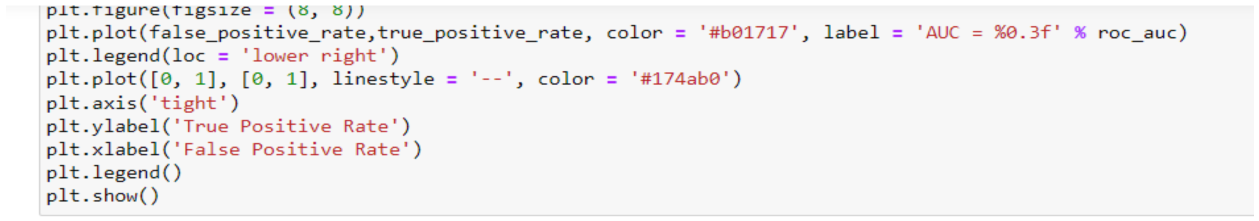
Below Diagram gives more about precision and recall



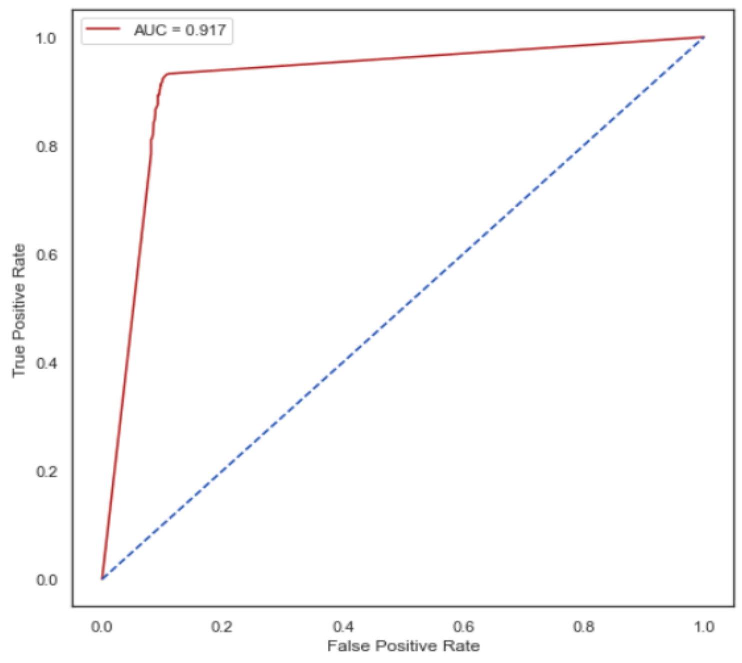
```
In [62]: print(classification_report(y_test, y_pred_DTC_test))
              precision    recall  f1-score   support

           0       0.92      0.90      0.91       982
           1       0.90      0.92      0.91       963

    accuracy                           0.91      1945
   macro avg       0.91      0.91      0.91      1945
weighted avg       0.91      0.91      0.91      1945
```

ROC CURVE

Below graph displaying the model's performance overall thresholds. Two parameters are plotted on this curve: % True Positive. Rate of False Positives
In General if the ROC Curve is more than 0.9 it is considered outstanding.



```
plt.figure(figsize = (8, 8))
plt.plot(false_positive_rate,true_positive_rate, color = '#b01717', label = 'AUC = %0.3f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



Now as the model is trained and tested using data, we can say the model is able to gain intelligence so that it can predict whether a patient gets a stroke or not with an accuracy of 91%. Not only the accuracy is high but also the ROC AUC Score is more than 0.9, as per standard if the AUC score is more than 0.9 it is considered outstanding. then the algorithm or model is really predicted more accurately on unseen data.

Comparison Between Algorithms / Models

Algorithm/Model	**Testing Accuracy**	**Training Accuracy**	**ROC AUC Score**
KNN	0.875	0.918	0.915
Naive Bayes	0.724	0.730	0.826
Support Vector Machines	0.759	0.763	0.842
Random Forest	0.913	0.985	0.966
Cat Boost	**0.930**	0.947	**0.977**
Logistic Regression	0.776	0.791	0.855
Decision Tree	0.907	**0.985**	0.913

From the above table we can say that during the training phase Decision Tree is more efficient and during testing Cat Boost is most efficient with 93% accuracy.

Here we are able to gain intelligence to predict the stroke label , this can be used in phase - 3 to build a web based application **.**

References:

https://www.kaggle.com/code/mohamedibrahim206/stroke-prediction-eda-modeling/data
https://app.mode.com/modeanalytics/reports/f2a442bd3b9a/details/notebook
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3825015
https://scikit-learn.org/stable/
https://cse.buffalo.edu/~epmikida/teaching/fa22/cse487/index.html - Lecture slides and demo codes
C. O’Neill and R. Schutt. Doing Data Science., O’Reilly. 2013.
J. VanderPlas, Python Data Science Handbook., O'Reilly. 2016.