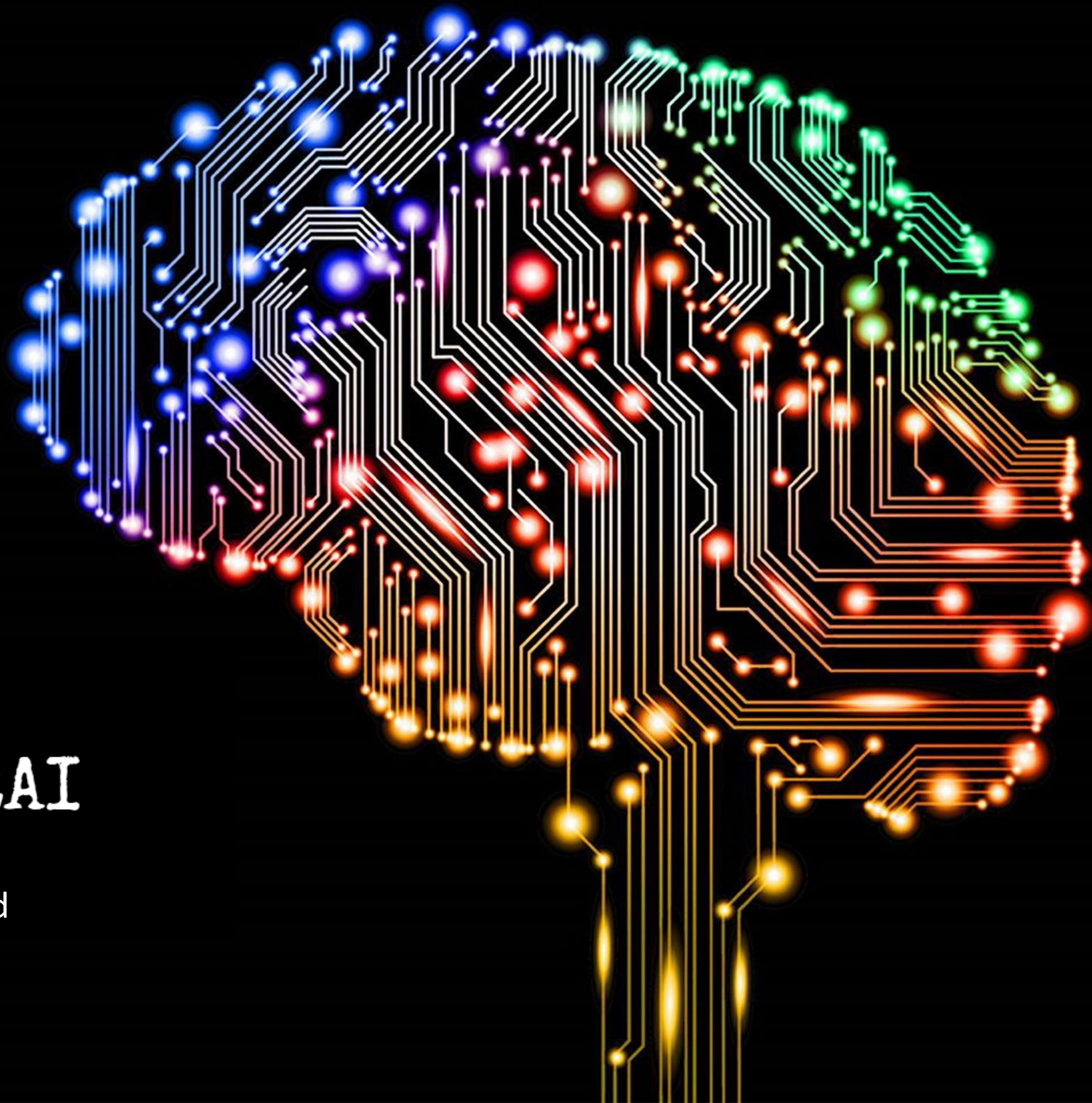


TEAMdotAI

Zishan Sami
Harshit Anand

In order to approach the problem, we used **Python** and the module used was **Scikit Learn**.

Index

Variable Selection

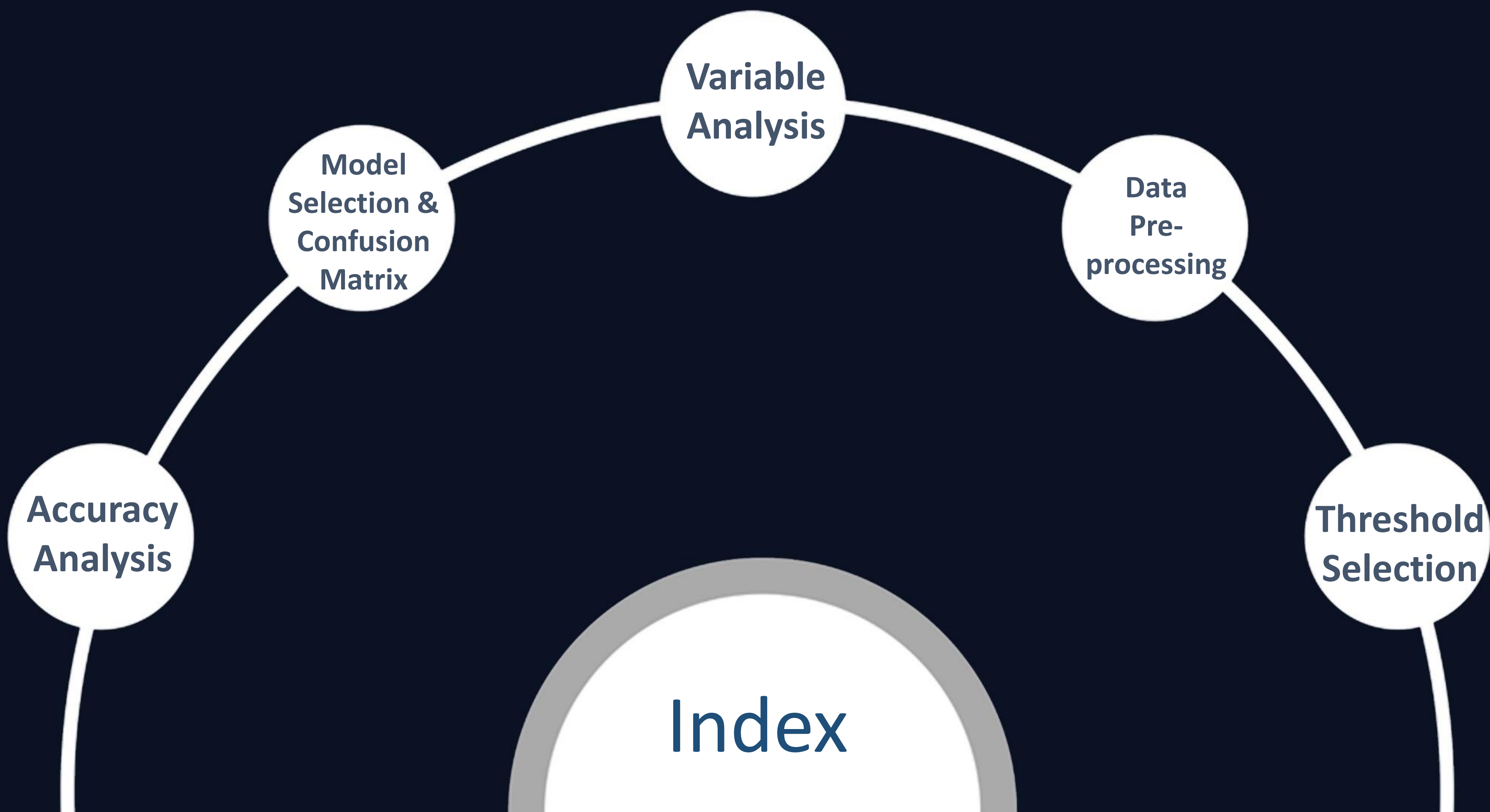
Data Pre- processing

Model Selection & Confusion Matrix

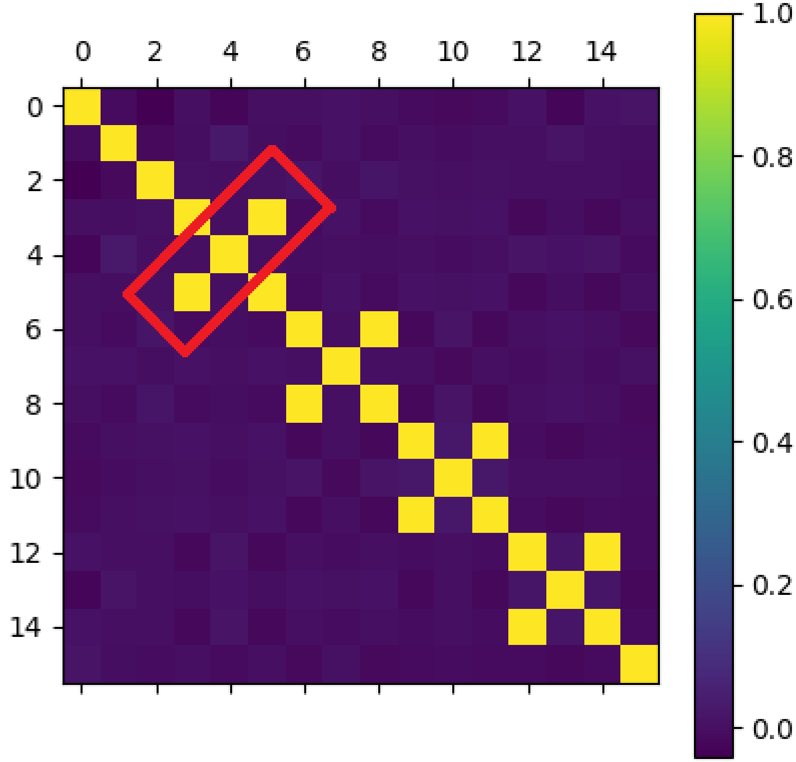
Threshold Selection

Accuracy Analysis

Index

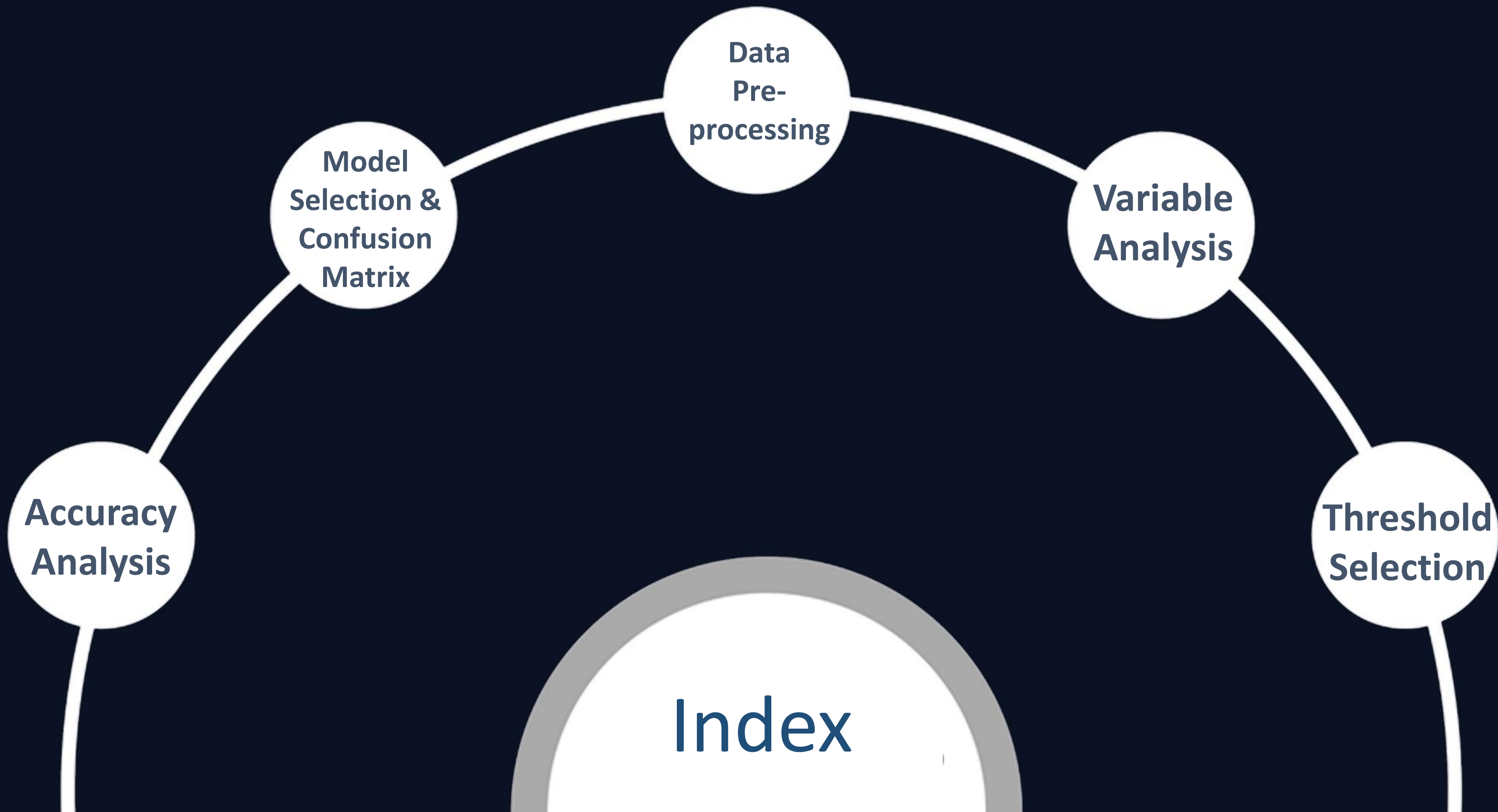

Variable Analysis
Data Pre-processing
Threshold Selection
Model Selection & Confusion Matrix
Accuracy Analysis

Variable Analysis



- High Correlation between certain equally spaced variables was observed
- These variables were identified to be total_*_minutes and Total_*_charges

We derived one feature out of these two using Principal Component Analysis(PCA)


Accuracy Analysis
Model Selection & Confusion Matrix
Data Pre-processing
Variable Analysis
Threshold Selection
Index

Data Pre- processing

- After the variable selection and normalization of some features was done to match the scale.

- We converted some features which were in string format('yes/no') into Boolean(1/0).

- Removing of variables like area_code, phone_number, state & id.

S	L
total_intl_calls	total_eve_minutes
3	197.4
3	195.5
5	121.2
7	61.9
3	148.3

Large Difference in values

Dataset Split

The Dataset was split into

•Train Data: Used to train the different models

•Validation Data: Used to test the different models and find respective Area Under ROC Curve, sensitivity, specificity and precision. Threshold selection was done using this.

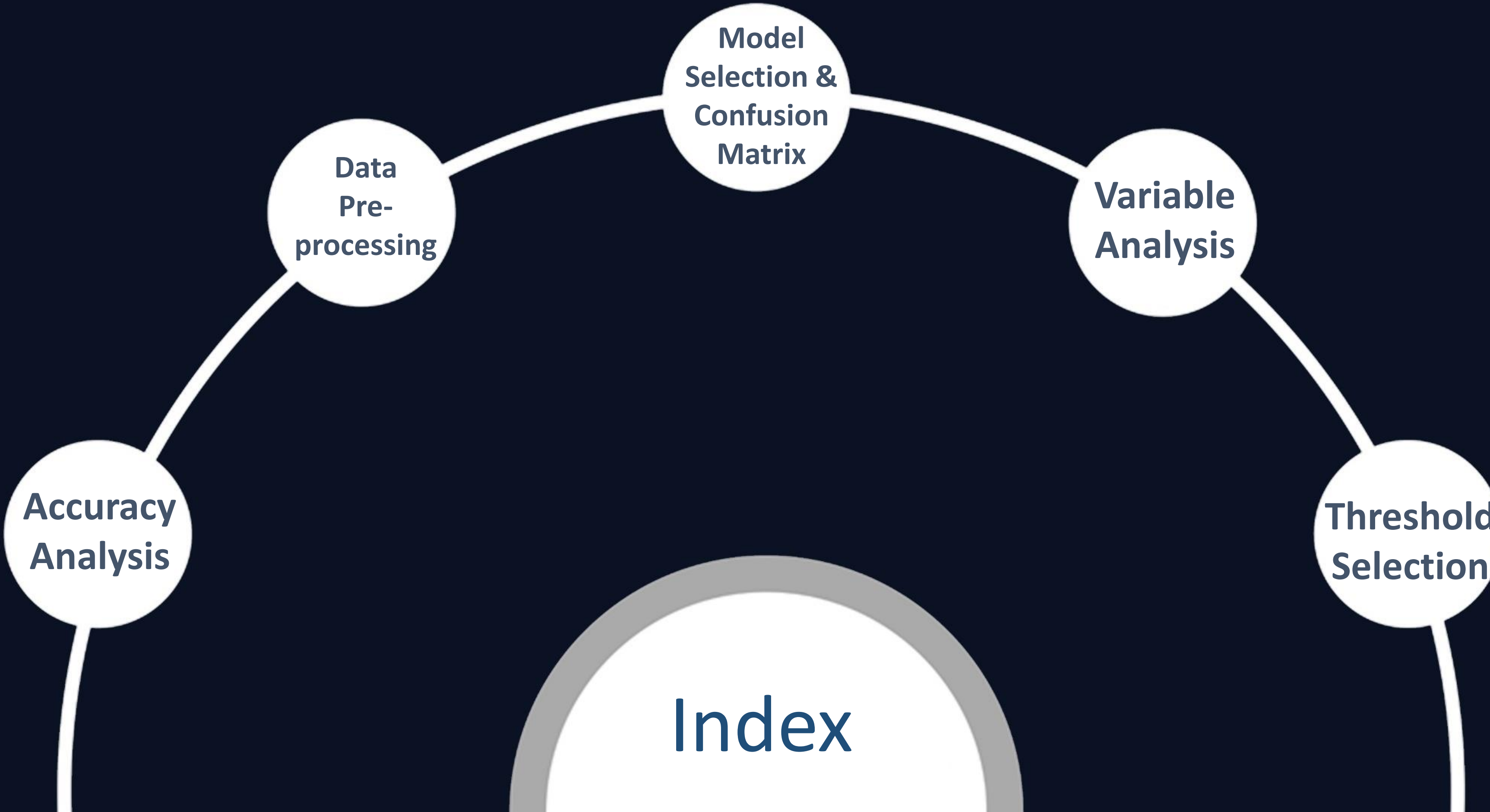
•Test Data: Separate Dataset strata on which the validated model was tested upon to find the various parameters. This dataset can be thought of as an unknown Dataset on which the model can be applied.

The total Dataset of 5000 was split into 3500:750:750 for Train, Validation and Test data respectively.

Churn percentage in full data:	14.14%
Churn percentage in training set:	14.51%
Churn percentage in validation set:	13.33%
Churn percentage in test set:	13.20%

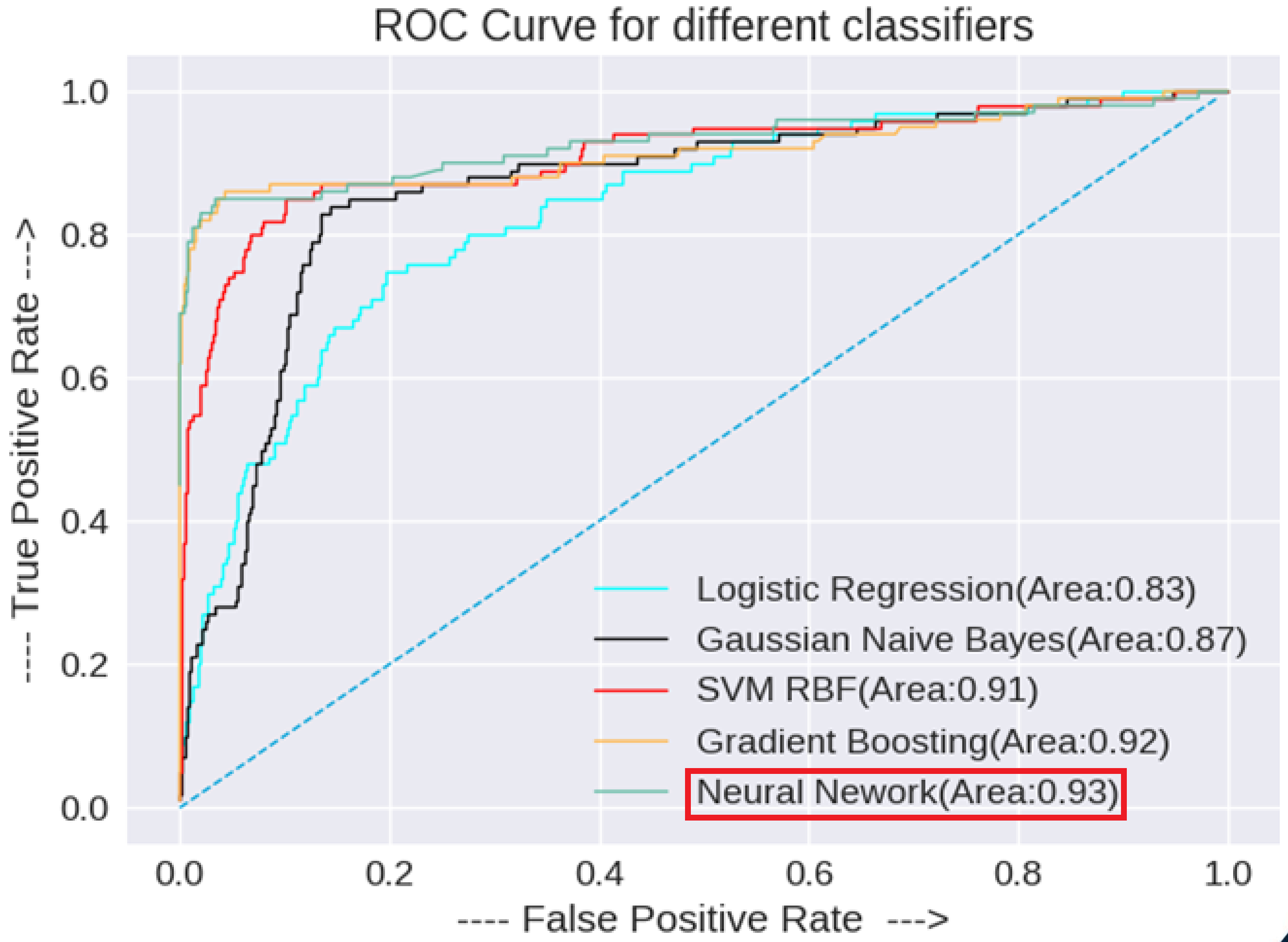
Difference between full data and test data is 0.94%

Index

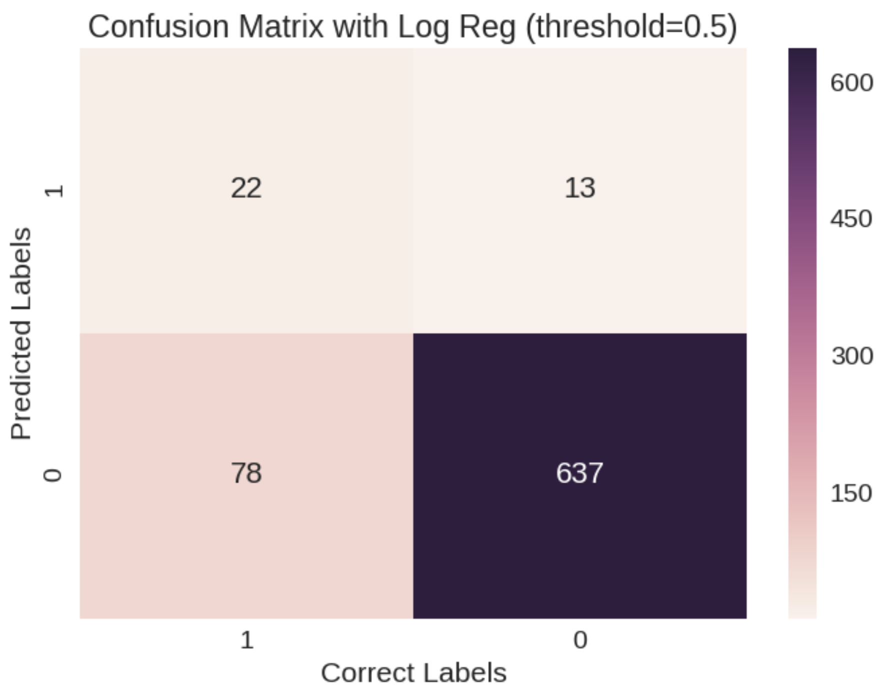

Accuracy Analysis
Data Pre-processing
Model Selection & Confusion Matrix
Variable Analysis
Threshold Selection

Model Selection

We decided to go with Area under ROC curve parameter. Classifiers were tested and the classifier with the highest Area Under ROC Curve (AUC-ROC) was selected.



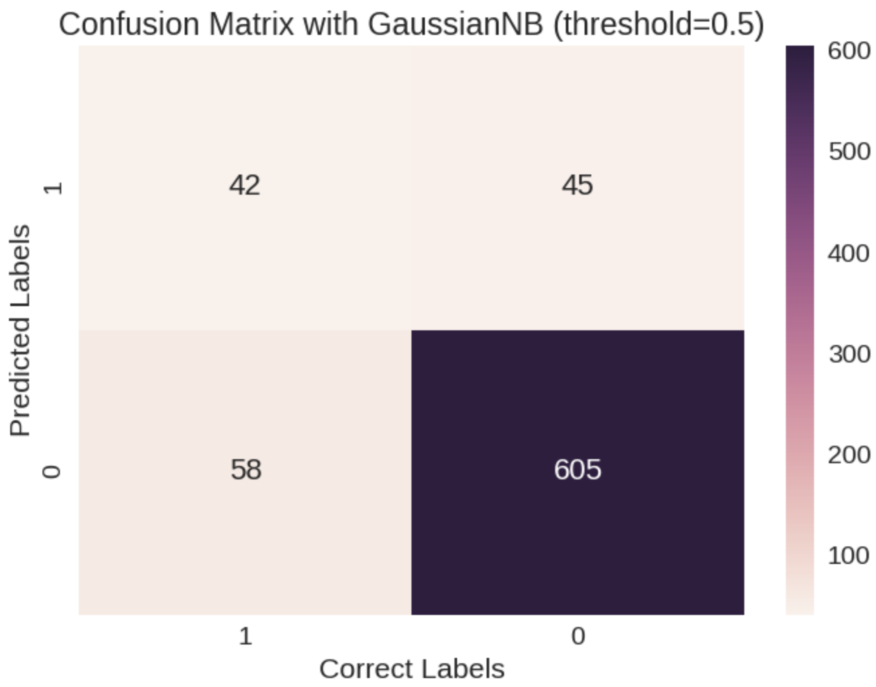
Logistic Regression



AUC: 0.833

Specificity: 0.98
Sensitivity/Recall: 0.22
Precision: 0.63

Naive Bayes'



AUC: 0.865

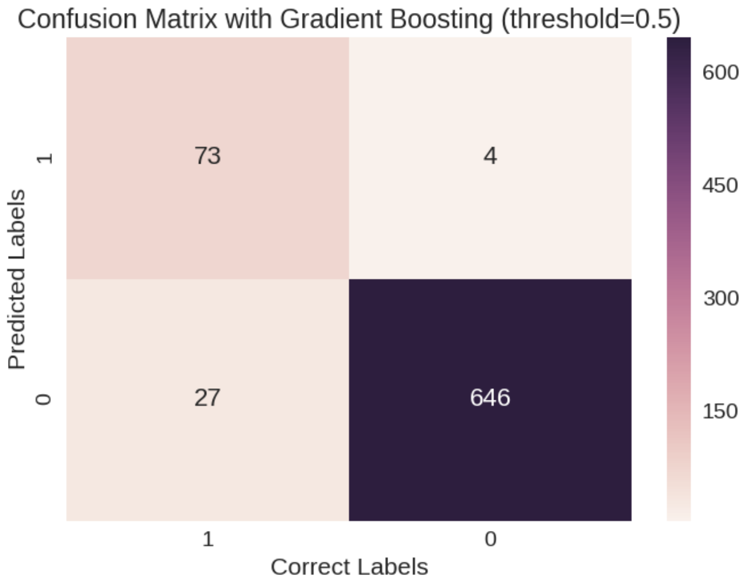
Specificity: 0.93
Sensitivity/Recall: 0.42
Precision: 0.48

SVM Classifier

AUC: 0.911

Specificity: 0.99
Sensitivity/Recall: 0.55
Precision: 0.86

Gradient Boosting

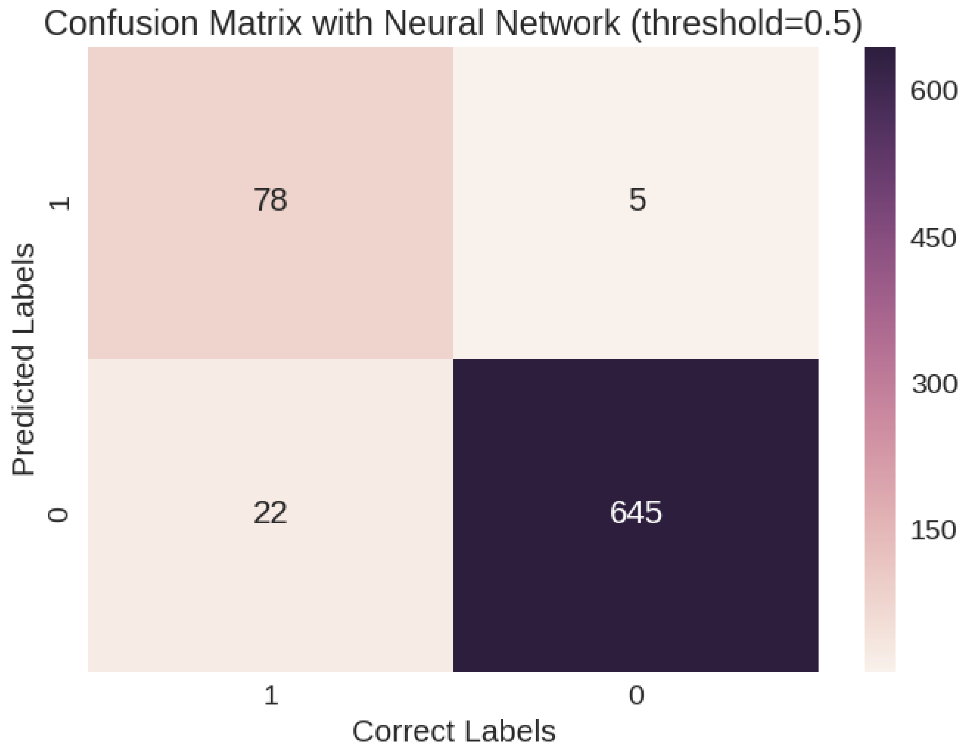


AUC: 0.917

Specificity: 0.99
Sensitivity/Recall: 0.73
Precision: 0.95

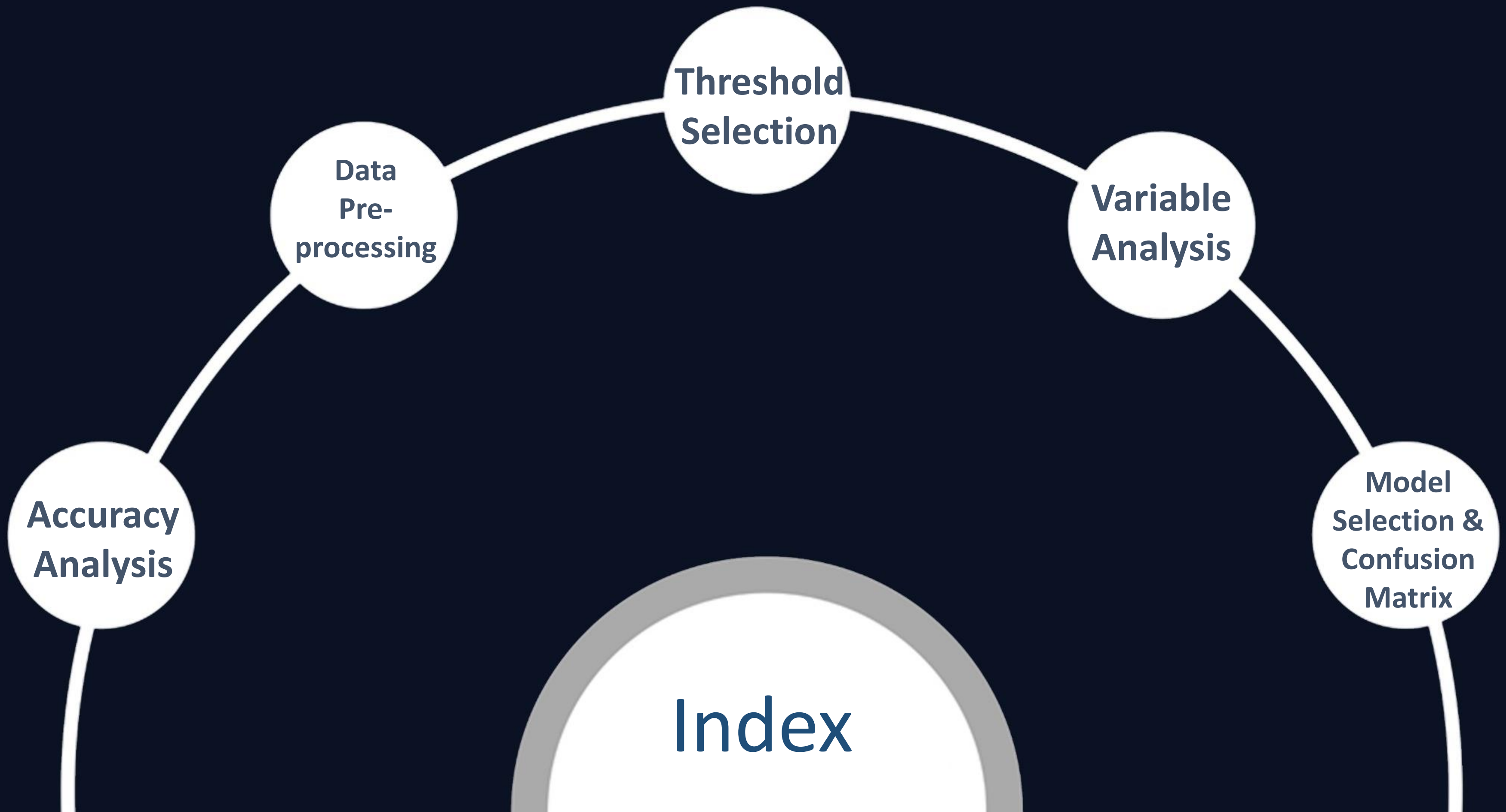
Neural Network

Selected Model



AUC: 0.927

Specificity: 0.99
Sensitivity/Recall: 0.78
Precision: 0.94


Threshold Selection
Data Pre-processing
Variable Analysis
Accuracy Analysis
Model Selection & Confusion Matrix
Index

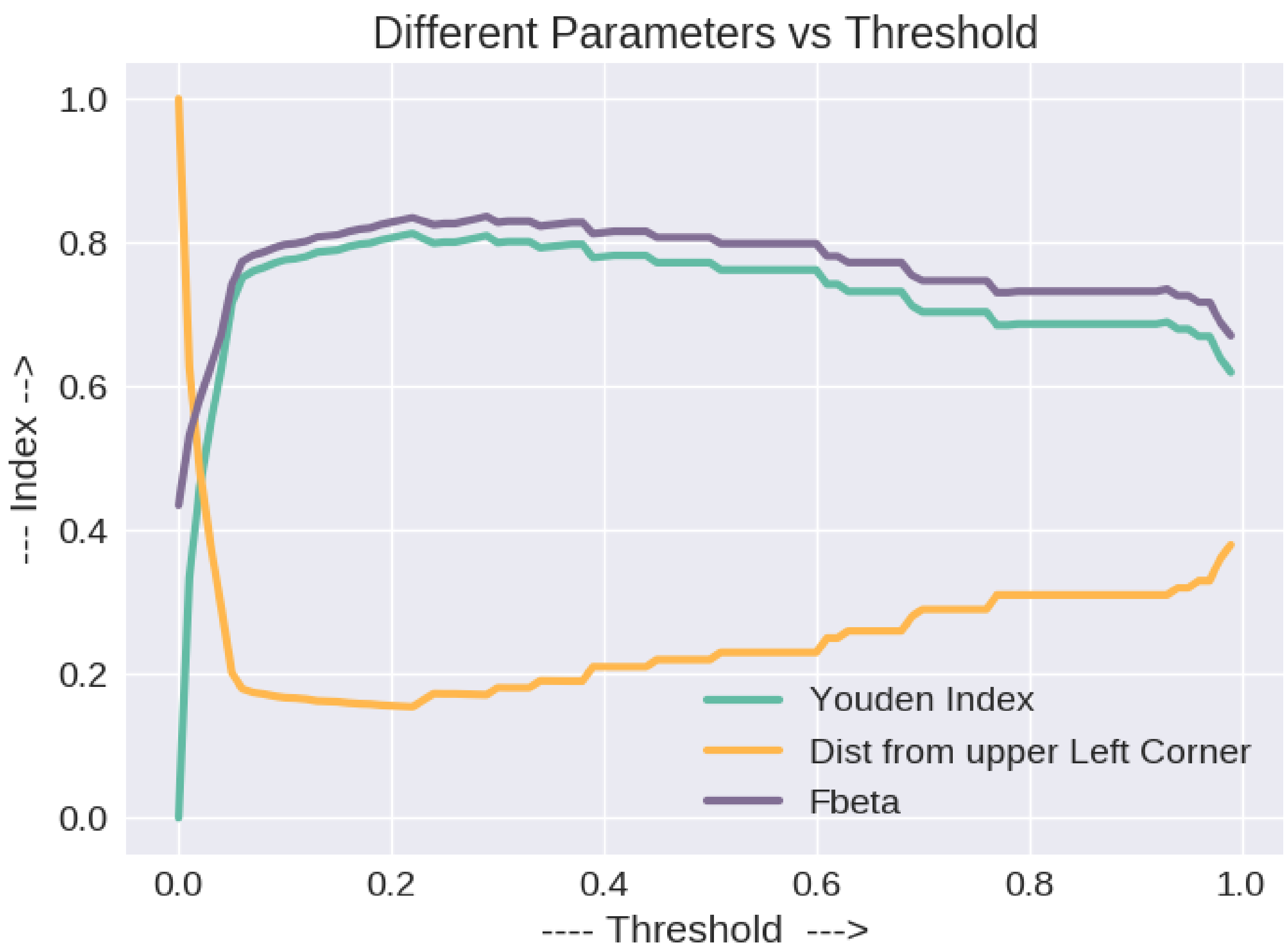
Threshold Selection

As we saw at the threshold value of 0.5, we have high precision, however , Recall value is low.

To choose the right threshold value for classification we plotted different parameters like :

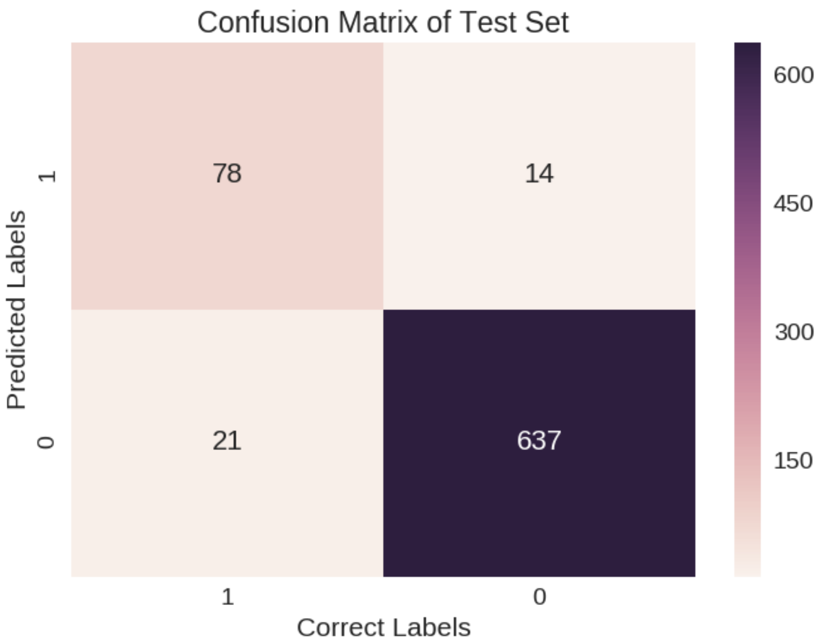
- **Youden Index:** (Sensitivity + Specificity - 1)
- **Distance of the point from the upper left corner on the ROC curve**
- **F-Beta value:** : 2/(Fbeta)= (1/(beta*Recall)) + (1/(precision))

against varying thresholds



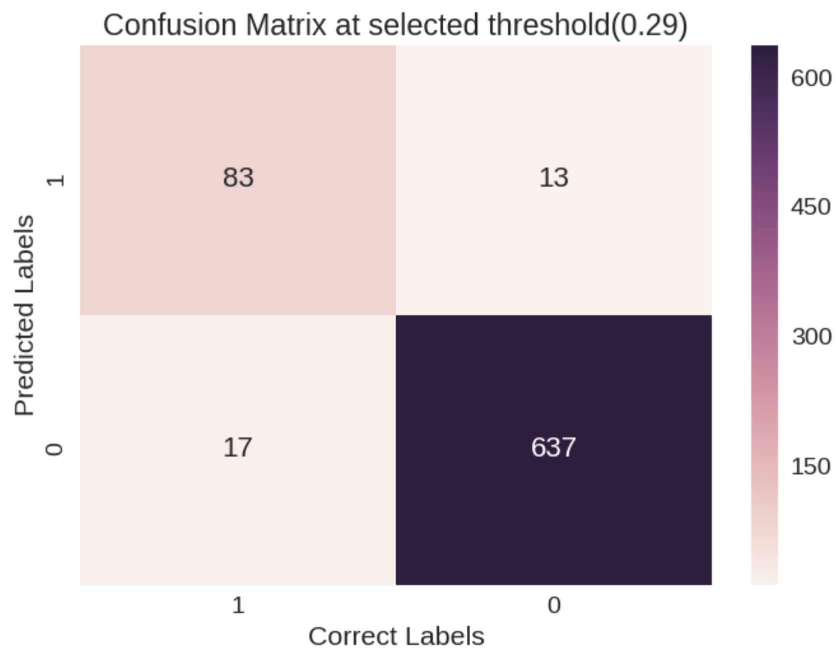
Selected Threshold: 0.29

Confusion Matrix



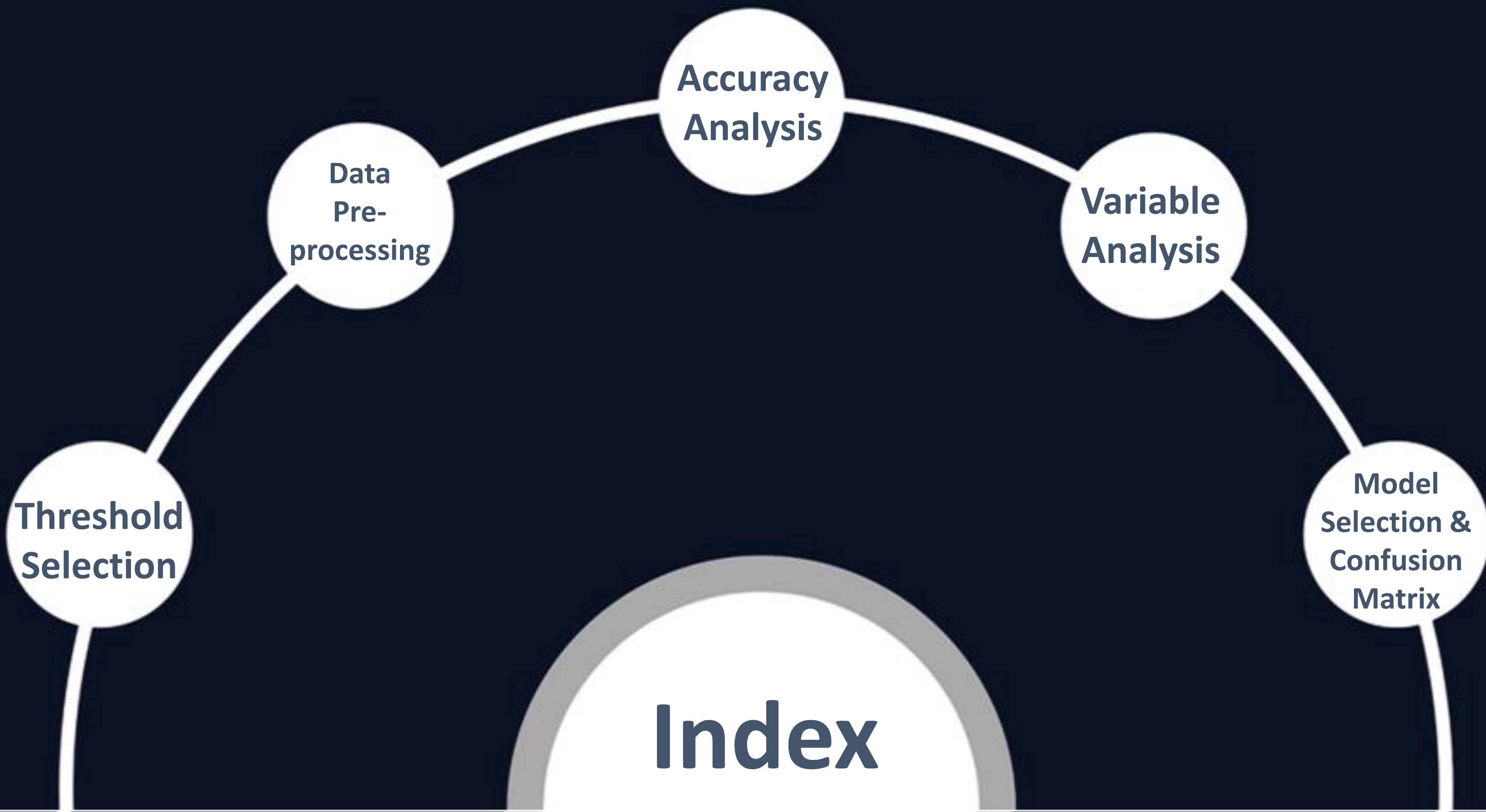
Test Data

Specificity: 0.98
Sensitivity/Recall: 0.81
Precision: 0.85Test
AUC: 0.903233564524
Test Accuracy: 0.953333333333



Validation Data

Specificity: 0.98
Sensitivity/Recall: 0.83
Precision: 0.86
val AUC: 0.927284615385
val. Accuracy: 0.96


Accuracy Analysis
Data Pre-processing
Variable Analysis
Threshold Selection
Model Selection & Confusion Matrix
Index

We analyzed the impact of each variable by omitting them and then observing the Area Under ROC Curve.

We reached to the conclusion that following had the most impact:

- total_day_charge
- number_customer_service_calls
- international_plan

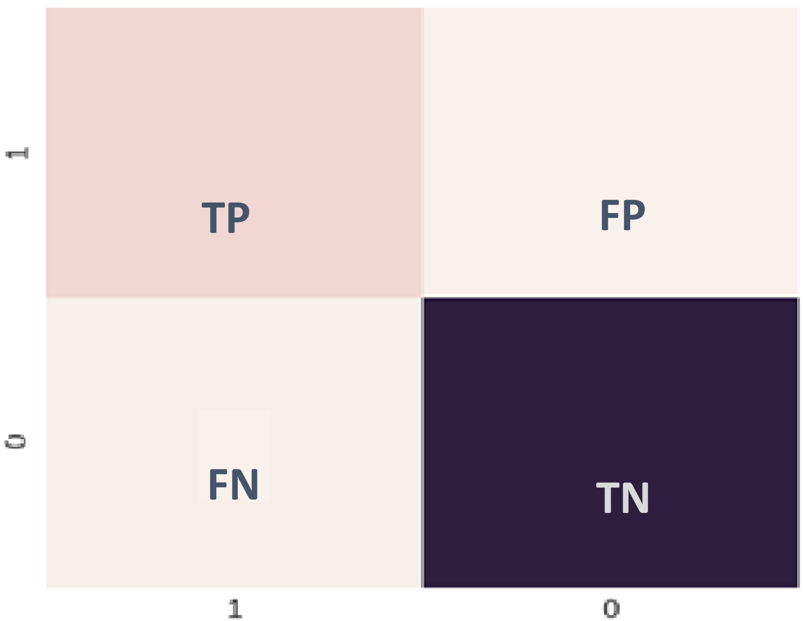
Variables like account_lenght has minimal effect. Finally, we observed that we had the highest Area Under ROC Curve when all the features(prepared dataset after variable analysis) were taken into consideration.

Alternative Approach

Cost Analysis

For a Company, in order to balance to the budget along with the Retention and Acquisition costs, their balance of Recall and Precision can be dependent on cost of acquisition and cost of retention.

Approach Taken

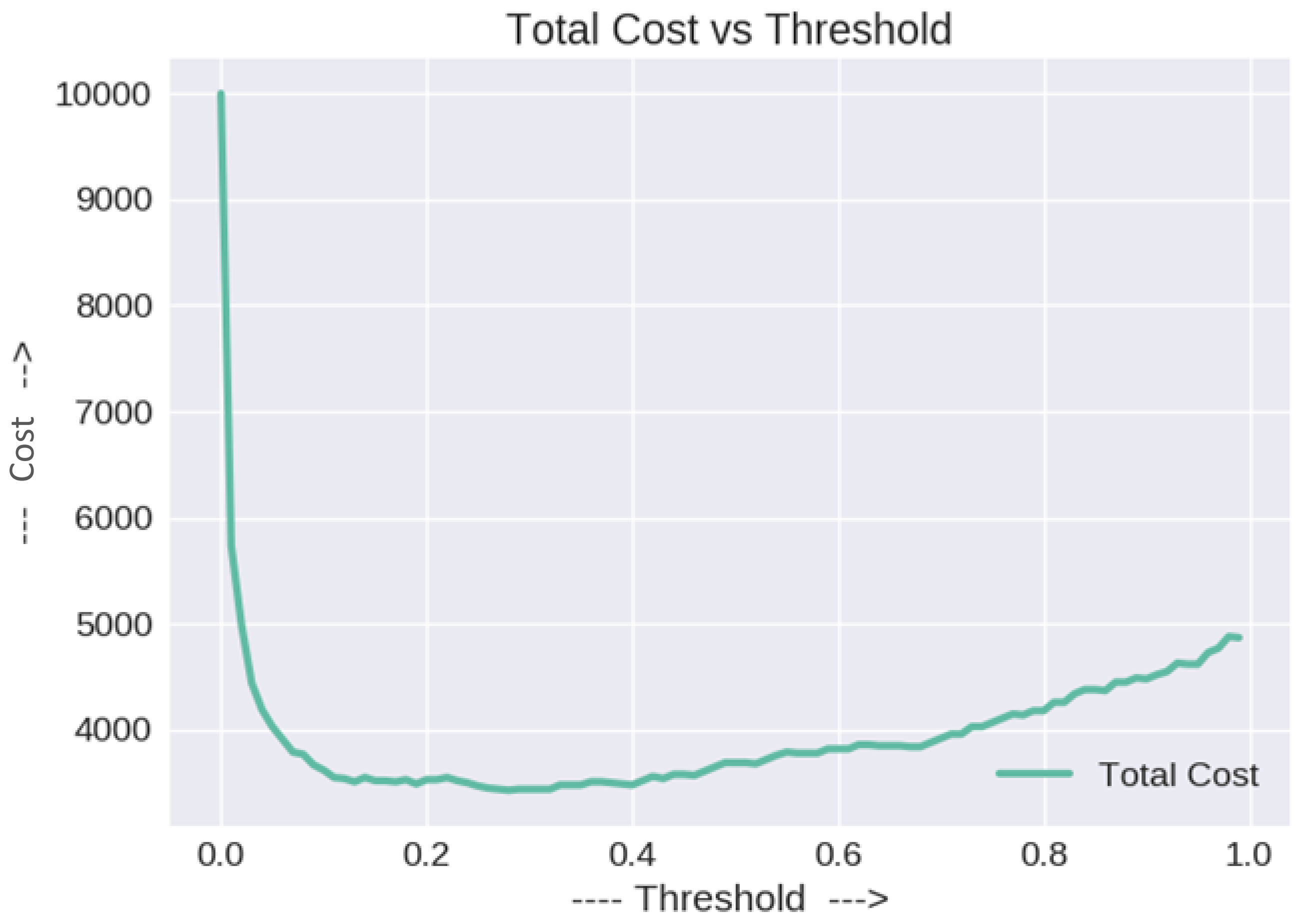


Using our model, we would get the total number of customers leaving as TP+ FP so, we would be spending Cost of Retention on them.

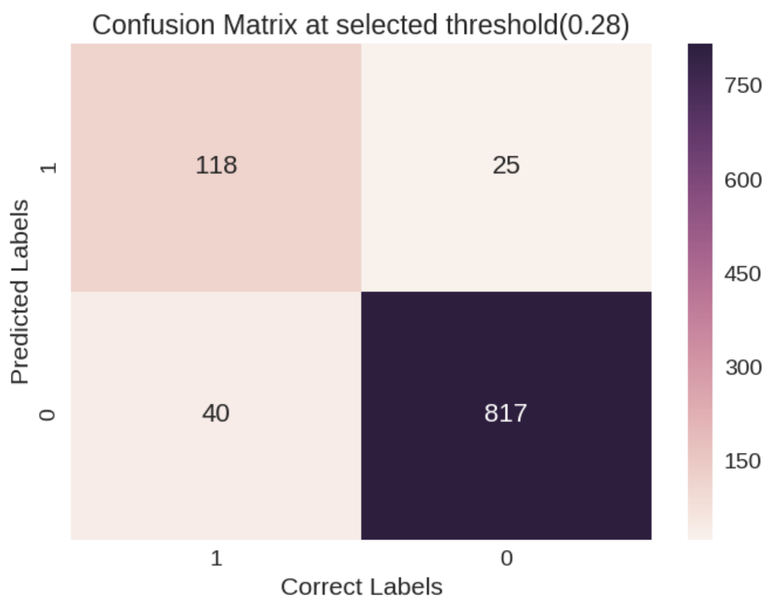
While we would be spending Cost of Acquisition on FN number of customers.

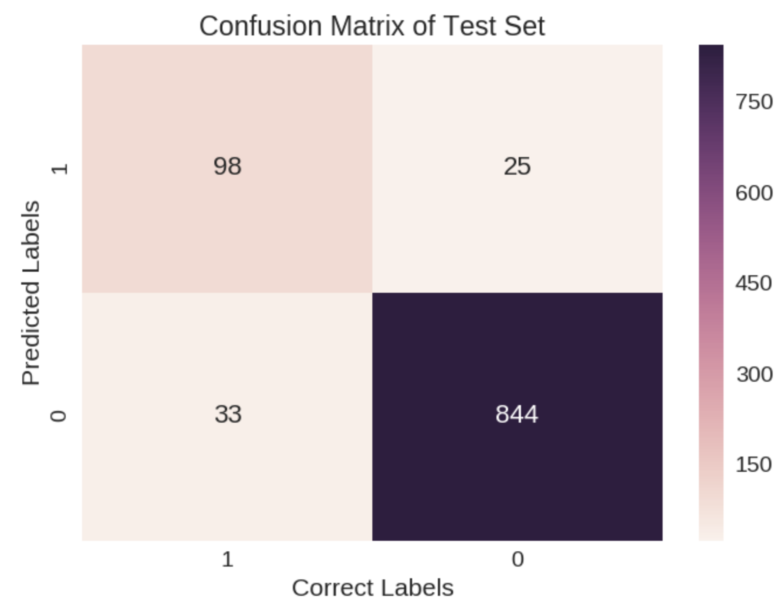
Cost of Retention: Rs. 10/ person
Cost of Acquisition: Rs. 50/ person

$$\boxed{\text{Cost} = 10(TP + FP) + 50(FN)}$$



Selected Threshold: 0.28





Validation Data

Specificity: 0.97
Sensitivity/Recall: 0.75
Precision: 0.83
val AUC: 0.882809916113 | Training AUC: 0.966490499186
val. Accuracy: 0.935

Test Data

Specificity: 0.97
Sensitivity/Recall: 0.75
Precision: 0.8
Test AUC: 0.88829399415 | Test Accuracy: 0.942

Appendix

```
import numpy as np
from sklearn import preprocessing,cross_validation, svm
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
import pandas as pd
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sn

plt.style.use('seaborn-darkgrid')
plt.rcParams['axes.facecolor'] = '#f7f1eb'

df = pd.read_csv('newccd.csv')
df.drop(['Id'], 1, inplace=True)

# df = df[['', '', '', '']]
X = df.drop(['churn'],1)
y = list(df.churn)



## Model Deciding through highest AUC(Area Under ROC curve)
print "Churn percentage in full data:",np.sum(1*(np.array(y)==1))/5000.0

## Train Test Split
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X,y,test_size=0.3)
X_val, X_test, y_val, y_test = cross_validation.train_test_split(X_test,y_test,test_size=0.5)
print "Churn percentage in training data:",np.sum(1*(np.array(y_train)==1))/float(len(y_train))
print "Churn percentage in val data:",np.sum(1*(np.array(y_val)==1))/float(len(y_val))
print "Churn percentage in test data:",np.sum(1*(np.array(y_test)==1))/float(len(y_test))



## Model Selection
print "\nLogistic Regression Analysis:"
clf = LogisticRegression(solver='sag',max_iter=500)
## Training
clf.fit(X_train, y_train)
prob = clf.predict_proba(X_val)[:,1]
fpr1, tpr1, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1)
AUC1 = metrics.auc(fpr1, tpr1)
print "AUC: ", AUC1
## Confusion Matrix at initial Threshold=0.5
```

```
AUC1 = metrics.auc(fpr1, tpr1)
print "AUC: ", AUC1
## Confusion Matrix at initial Threshold=0.5
pred = (prob>=0.5)*1
tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
conf_matrix = np.array([[tp,fp],[fn,tn]])

sensitivity = tp/float(fn+tp)
specificity = tn/float(fp+tn)
precision = tp/float(tp+fp)
df_cm = pd.DataFrame(conf_matrix,  index=["1","0"],columns=["1","0"])
print "Confusion Matrix at default Threshold=0.5"
print df_cm
print "Specificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2),"\n"
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
plt.title("Confusion Matrix with Log Reg (threshold="+str(0.5)+")")
plt.xlabel("Correct Labels")
plt.ylabel("Predicted Labels")
plt.show()


## Model Selection
print "\nGaussian Naive Bayes Analysis:"
clf = GaussianNB()
## Training
clf.fit(X_train, y_train)
prob = clf.predict_proba(X_val)[:,1]
fpr2, tpr2, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1)
AUC2 = metrics.auc(fpr2, tpr2)
print "AUC : ", AUC2
## Confusion Matrix at initial Threshold=0.5
pred = (prob>=0.5)*1
tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
conf_matrix = np.array([[tp,fp],[fn,tn]])

sensitivity = tp/float(fn+tp)
specificity = tn/float(fp+tn)
precision = tp/float(tp+fp)
df_cm = pd.DataFrame(conf_matrix,  index=["1","0"],columns=["1","0"])
print "Confusion Matrix at default Threshold=0.5"
print df_cm
print "Specificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2),"\n"
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
```

```python
print "Specificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2),"\n"
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
plt.title("Confusion Matrix with GaussianNB (threshold="+str(0.5)+")")
plt.xlabel("Correct Labels")
plt.ylabel("Predicted Labels")
plt.show()

## Model Selection
print "\nSVM Analysis:"
clf = svm.SVC(probability=True,kernel='rbf')
## Training
clf.fit(X_train, y_train)
prob = clf.predict_proba(X_val)[:,1]
fpr3, tpr3, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1)
AUC3 = metrics.auc(fpr3, tpr3)
print "AUC : ", AUC3
## Confusion Matrix at initial Threshold=0.5
pred = (prob>=0.5)*1
tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
conf_matrix = np.array([[tp,fp],[fn,tn]])

sensitivity = tp/float(fn+tp)
specificity = tn/float(fp+tn)
precision = tp/float(tp+fp)
df_cm = pd.DataFrame(conf_matrix,  index=["1","0"],columns=["1","0"])
print "Confusion Matrix at default Threshold=0.5"
print df_cm
print "Specificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2),"\n"
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
plt.title("Confusion Matrix with SVM (threshold="+str(0.5)+")")
plt.xlabel("Correct Labels")
plt.ylabel("Predicted Labels")
plt.show()


## Model Selection
print "\nGradient Boosting Analysis:"
clf = GradientBoostingClassifier()
## Training
clf.fit(X_train, y_train)
prob = clf.predict_proba(X_val)[:,1]
fpr4, tpr4, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1)
AUC4 = metrics.auc(fpr4, tpr4)
```

```

## Model Selection
print "\nGradient Boosting Analysis:"
clf = GradientBoostingClassifier()
## Training
clf.fit(X_train, y_train)
prob = clf.predict_proba(X_val)[:,1]
fpr4, tpr4, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1)
AUC4 = metrics.auc(fpr4, tpr4)
print "AUC : ", AUC4
## Confusion Matrix at initial Threshold=0.5
pred = (prob>=0.5)*1
tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
conf_matrix = np.array([[tp,fp],[fn,tn]])

sensitivity = tp/float(fn+tp)
specificity = tn/float(fp+tn)
precision = tp/float(tp+fp)
df_cm = pd.DataFrame(conf_matrix,  index=["1","0"],columns=["1","0"])
print "Confusion Matrix at default Threshold=0.5"
print df_cm
print "Specificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2),"\n"
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
plt.title("Confusion Matrix with Gradient Boosting (threshold="+str(0.5)+")")
plt.xlabel("Correct Labels")
plt.ylabel("Predicted Labels")
plt.show()


## Model Selection
print "\nNeural Network Analysis:"
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(10,), random_state=1,activation='relu',max_iter=300)
## Training
clf.fit(X_train, y_train)
prob = clf.predict_proba(X_val)[:,1]
fpr5, tpr5, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1,drop_intermediate=False)
AUC5 = metrics.auc(fpr5, tpr5)
print "AUC : ", AUC5,"\n"
## Confusion Matrix at initial Threshold=0.5
pred = (prob>=0.5)*1
tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
conf_matrix = np.array([[tp,fp],[fn,tn]])

sensitivity = tp/float(fn+tp)
```

```
## Model Selection
print "\nNeural Network Analysis:"
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(10,), random_state=1,activation='relu',max_iter=300)
## Training
clf.fit(X_train, y_train)
prob = clf.predict_proba(X_val)[:,1]
fpr5, tpr5, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1,drop_intermediate=False)
AUC5 = metrics.auc(fpr5, tpr5)
print "AUC : ", AUC5,"\n"
## Confusion Matrix at initial Threshold=0.5
pred = (prob>=0.5)*1
tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
conf_matrix = np.array([[tp,fp],[fn,tn]])

sensitivity = tp/float(fn+tp)
specificity = tn/float(fp+tn)
precision = tp/float(tp+fp)
df_cm = pd.DataFrame(conf_matrix,  index=["1","0"],columns=["1","0"])
print "Confusion Matrix at default Threshold=0.5"
print df_cm
print "Specificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2),"\n
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
plt.title("Confusion Matrix with Neural Network (threshold="+str(0.5)+")")
plt.xlabel("Correct Labels")
plt.ylabel("Predicted Labels")
plt.show()



## plotting ROC curve for all the models
x = np.arange(0.0,1.0,0.01)
plt.plot(x,x,'--',color="#00a2d9",linewidth=1)
plt.plot(fpr1,tpr1,label="Logistic Regression(Area:"+str(round(AUC1,2))+")", color="cyan",linewidth=1)
plt.plot(fpr2,tpr2,label="Gaussian Naive Bayes(Area:"+str(round(AUC2,2))+")", color="black",linewidth=1)
plt.plot(fpr3,tpr3,label="SVM RBF(Area:"+str(round(AUC3,2))+")", color="red",linewidth=1)
plt.plot(fpr4,tpr4,label="Gradient Boosting(Area:"+str(round(AUC4,2))+")", color="#ffb74d",linewidth=1)
plt.plot(fpr5,tpr5,label="Neural Nework(Area:"+str(round(AUC5,2))+")", color="#62bba4",linewidth=1)

plt.legend(loc='lower right')
plt.title("ROC Curve for different classifiers")
plt.xlabel("---- False Positive Rate  --->")
plt.ylabel("---- True Positive Rate --->")
plt.show()
```

```python
## Choosing the suitable Threshold Value at point with lowest distance from the upper left corner on the ROC curve
x = np.arange(0.0,1.0,0.01)
print "Choosing threshold at point closer to upper left corner in ROC curve:"
YIndexList = []
dList = []
fb = []
Highest_YIndex=0 # Initializing it to zero
Lowest_d = 1
highest_Fbeta=0
for threshold in x:
    pred = (prob>=threshold)*1
    tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
    sensitivity = tp/float(fn+tp)
    specificity = tn/float(fp+tn)
    d = np.sqrt(np.power(1-sensitivity,2)+np.power(1-specificity,2))
    YIndex=specificity+sensitivity-1
    fbeta = metrics.fbeta_score(y_val, pred,beta=2)
    if fbeta>=highest_Fbeta:
        highest_Fbeta=fbeta
        selected_threshold=threshold
    YIndexList.append(YIndex)
    dList.append(d)
    fb.append(fbeta)

plt.plot(x,YIndexList,color='#62bba4',label="Youden Index",linewidth=3)
plt.plot(x,dList,color='#ffb74d',label="Dist from upper Left Corner",linewidth=3)
plt.plot(x, fb,color='#816E94',label="Fbeta",linewidth=3)
plt.legend(loc='lower right')
plt.title("Different Parameters vs Threshold")
plt.xlabel("---- Threshold  --->")
plt.ylabel("--- Index -->")
plt.show()
##  Confusion Matrix at optimum threshold value
pred = (prob>=selected_threshold)*1
tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
conf_matrix = np.array([[tp,fp],[fn,tn]])
df_cm = pd.DataFrame(conf_matrix, index=["1","0"],columns=["1","0"])
print "Confusion Matrix at selected threshold=", selected_threshold
print df_cm

sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
plt.xlabel("Correct Labels")
plt.ylabel("Predicted Labels")
```

```python
print df_cm

sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
plt.xlabel("Correct Labels")
plt.ylabel("Predicted Labels")
plt.title("Confusion Matrix at selected threshold("+str(selected_threshold)+")")
plt.show()


sensitivity = tp/float(fn+tp)
specificity = tn/float(fp+tn)
precision = tp/float(tp+fp)
acc = (tp+tn)/float(tp+tn+fn+fp)

prob = clf.predict_proba(X_train)[:,1]
fpr, tpr, thresholds = metrics.roc_curve(np.array(y_train), prob, pos_label=1)
Training_AUC = metrics.auc(fpr, tpr)

print "\nSpecificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2)
print "val AUC:",AUC5,"| Training AUC:",Training_AUC,"| val. Accuracy:",acc


print "\nTest set Analysis:"
prob = clf.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = metrics.roc_curve(np.array(y_test), prob, pos_label=1)
Val_AUC = metrics.auc(fpr, tpr)

pred = (prob>=selected_threshold)*1
tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_test), pred).ravel()
conf_matrix = np.array([[tp,fp],[fn,tn]])
df_cm = pd.DataFrame(conf_matrix, index=["1","0"],columns=["1","0"])
print "Confusion Matrix of Test Set:"
print df_cm

sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
plt.xlabel("Correct Labels")
plt.ylabel("Predicted Labels")
plt.title("Confusion Matrix of Test Set")
plt.show()

sensitivity = tp/float(fn+tp)
specificity = tn/float(fp+tn)
precision = tp/float(tp+fp)
```

```
plt.title("Confusion Matrix of Test Set")
plt.show()

sensitivity = tp/float(fn+tp)
specificity = tn/float(fp+tn)
precision = tp/float(tp+fp)
acc = (tp+tn)/float(tp+tn+fn+fp)
print "\nSpecificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2)
print "Test AUC:",Val_AUC,"| Test Accuracy:",acc

## Choosing the suitable Threshold Value using
x = np.arange(0.0,1.0,0.01)
print "Threshold selection:"
YIndexList = []
dList = []
fb = []
cost_list = []
Highest_YIndex=0 # Initializing it to zero
Lowest_d = 1
highest_Fbeta=0
tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), (prob>=0.01)*1).ravel()
ratio = 5
lowest_cost = 10*(tp+fp)+ratio*10*fn
for threshold in x:
    pred = (prob>=threshold)*1
    tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
    sensitivity = tp/float(fn+tp)
    specificity = tn/float(fp+tn)
    d = np.sqrt(np.power(1-sensitivity,2)+np.power(1-specificity,2))
    YIndex=specificity+sensitivity-1
    fbeta = metrics.fbeta_score(y_val, pred,beta=2)
    cost = 10*(tp+fp)+ratio*10*fn
    if cost<=lowest_cost:
        lowest_cost=cost
        selected_threshold=threshold
    YIndexList.append(YIndex)
    dList.append(d)
    fb.append(fbeta)
    cost_list.append(cost)
cost_list=np.array(cost_list)
# cost_list = cost_list/float(np.max(cost_list))
```