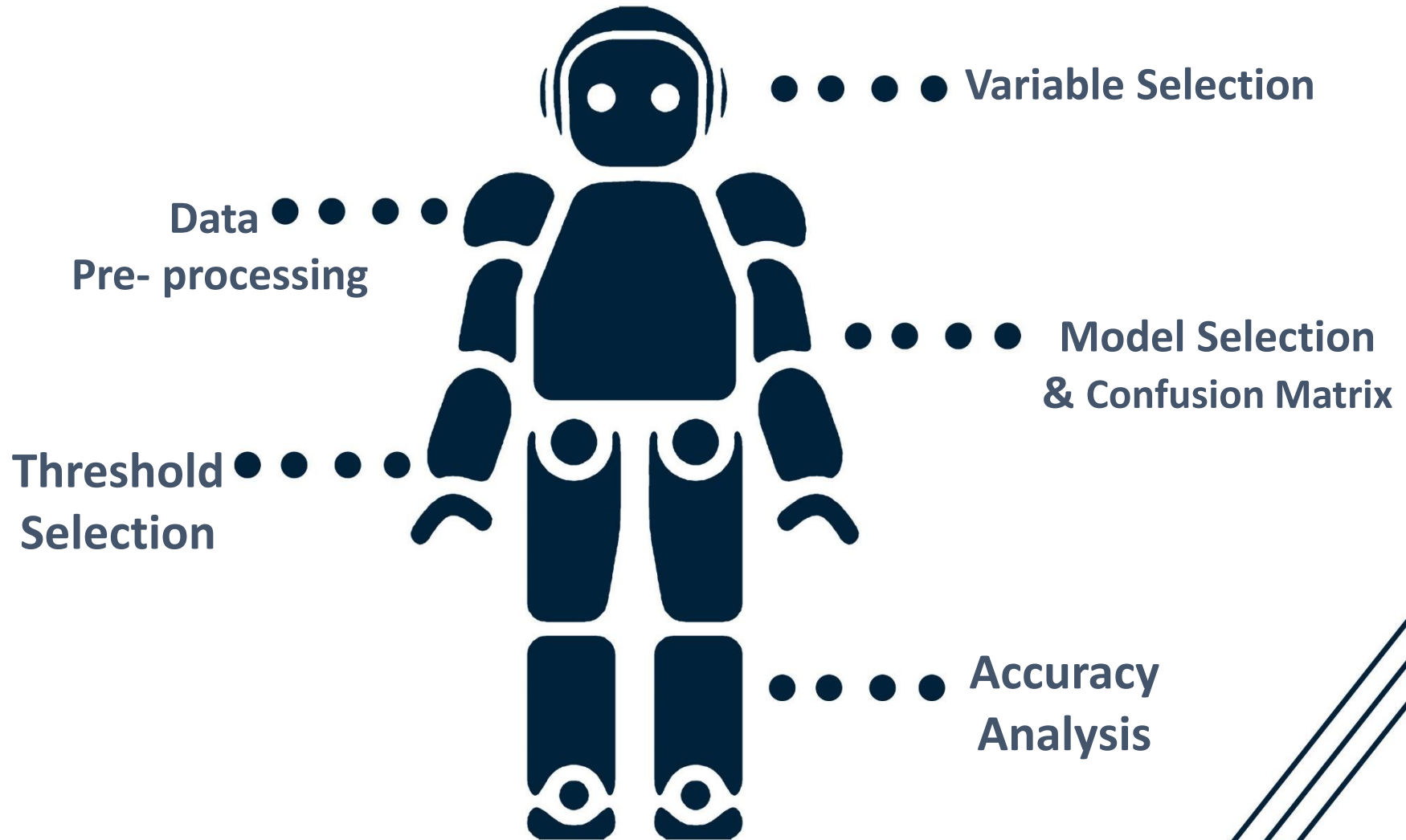


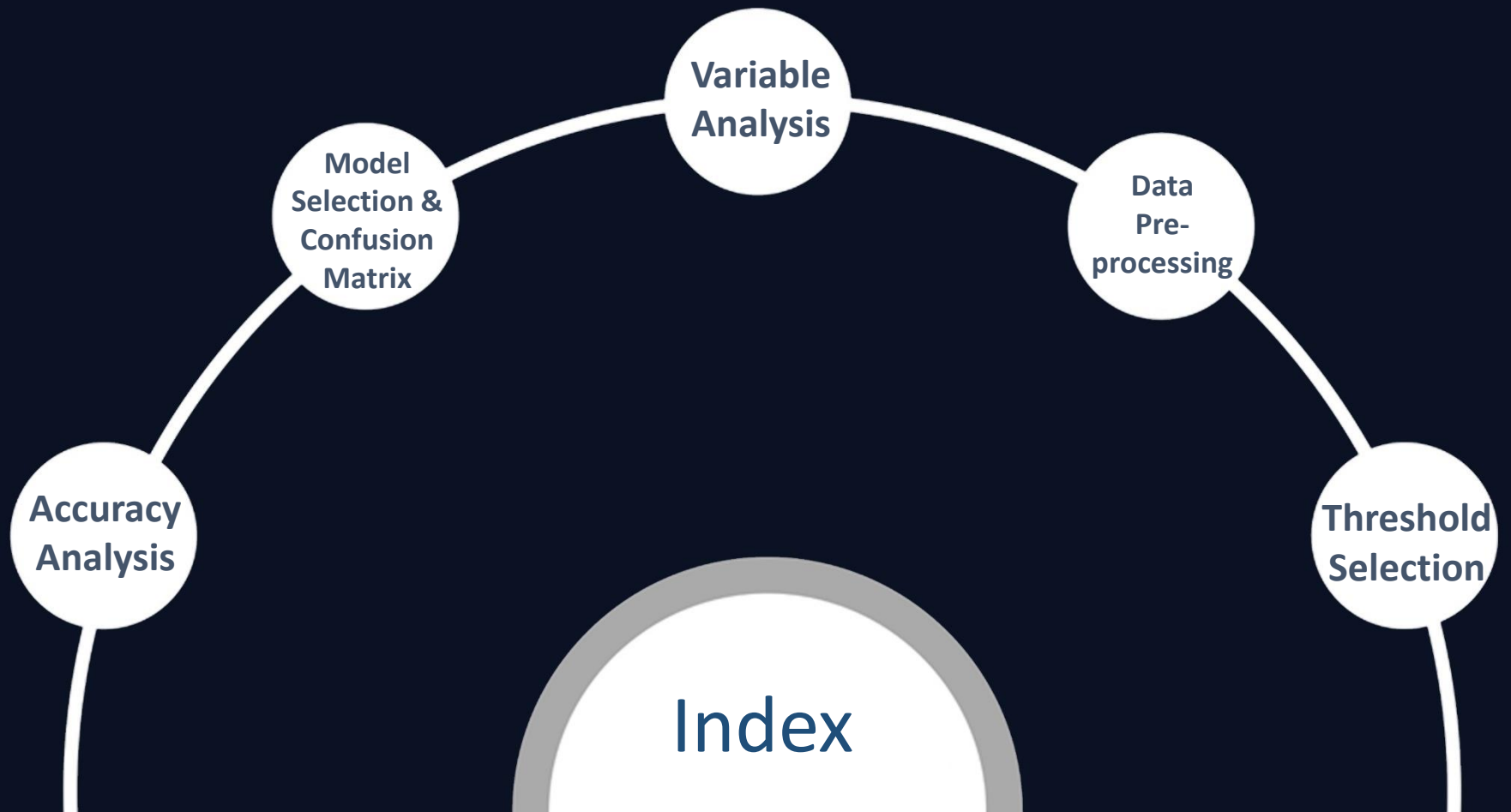
TEAMdotAI

Zishan Sami
Harshit Anand

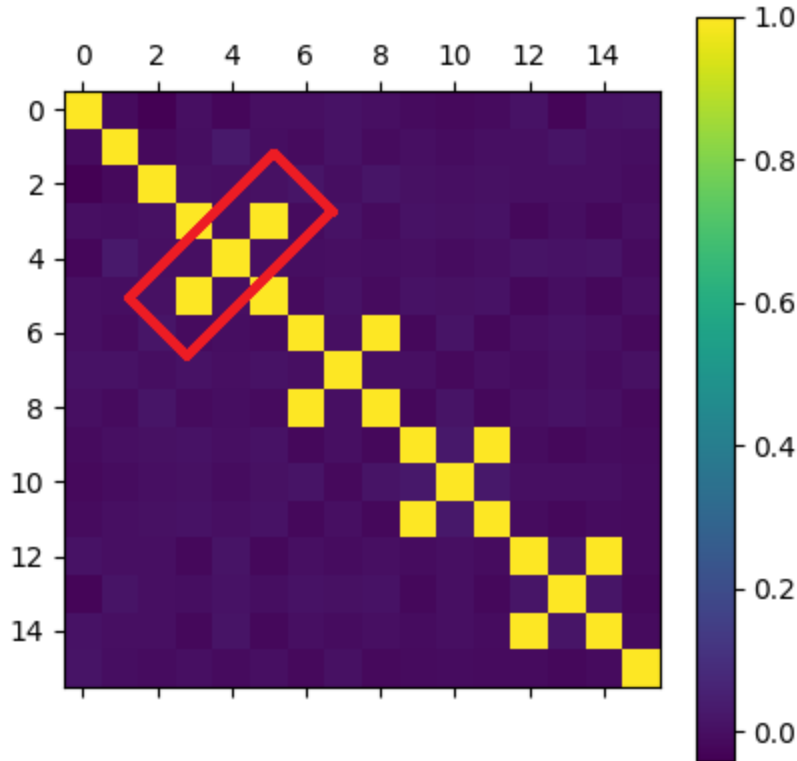
In order to approach the problem, we used **Python** and the module used was **Scikit Learn**.

Index





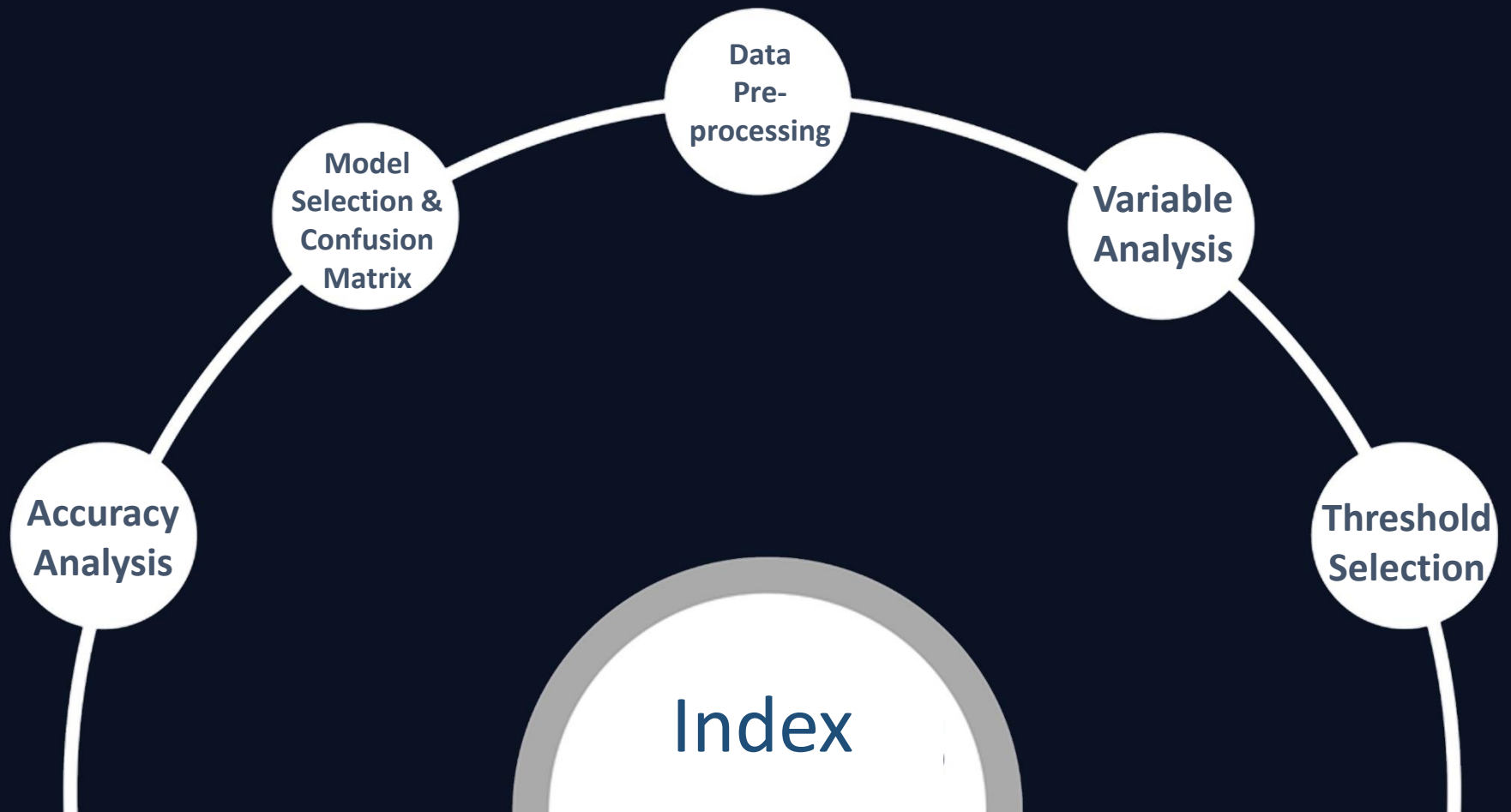
Variable Analysis



- 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)



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

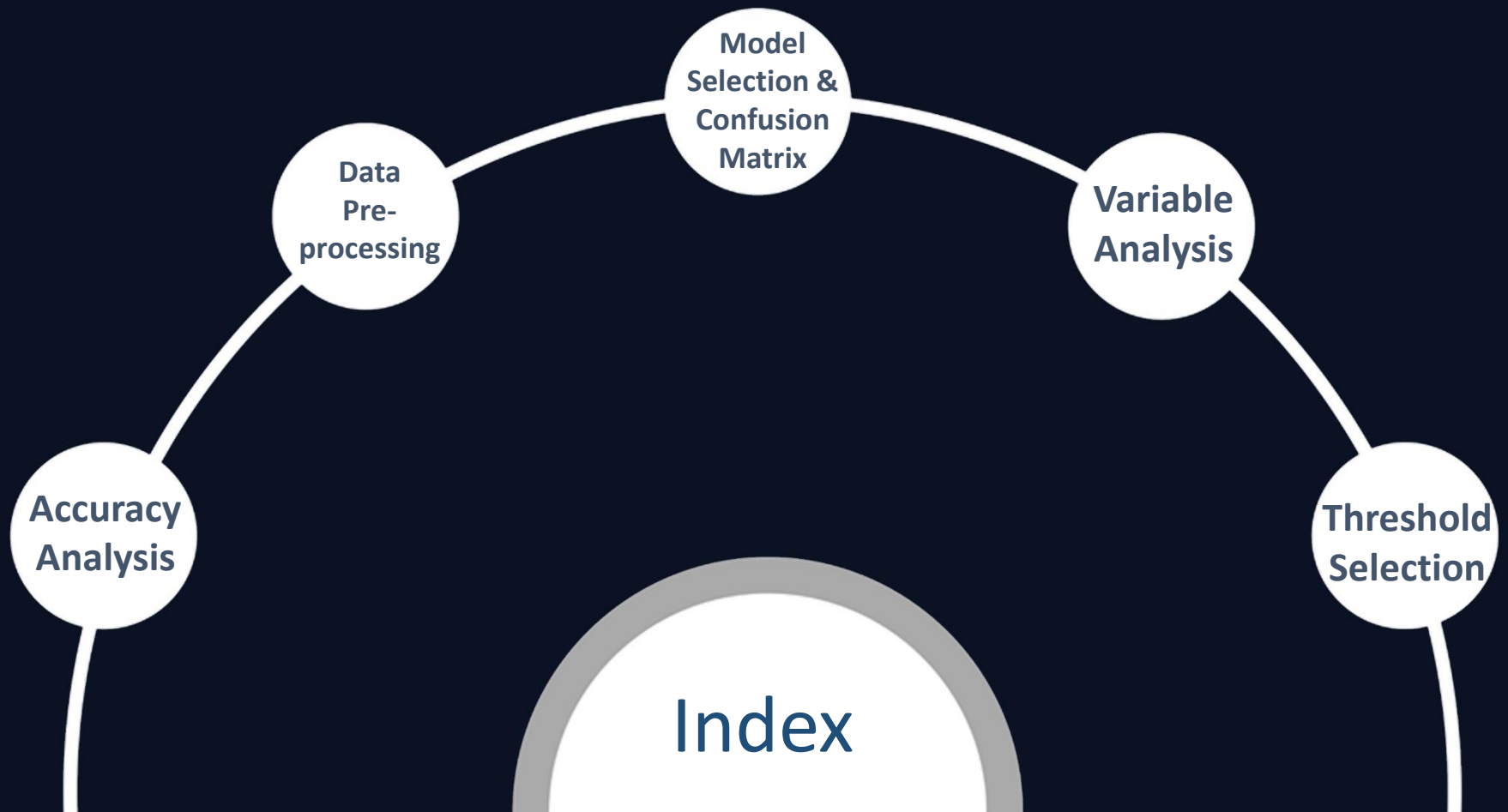
The Dataset was split into

- **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.

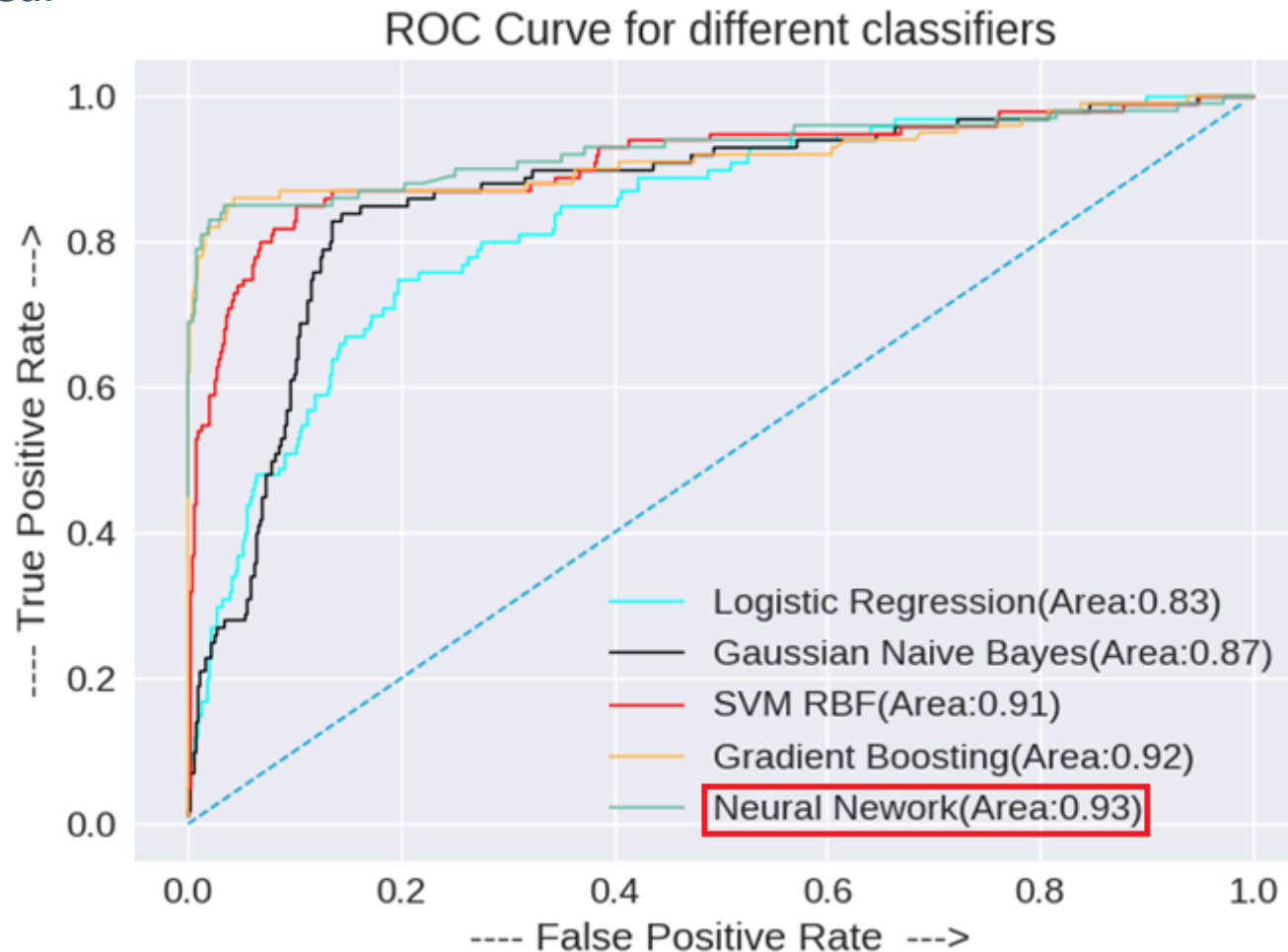
Churn percentage in full data:	14.14%
Churn percentage in training set:	14.51%
Churn percentage in validation set:	13.33%
Churn percentage in test set:	13.20%

Difference between full data and test data is 0.94%

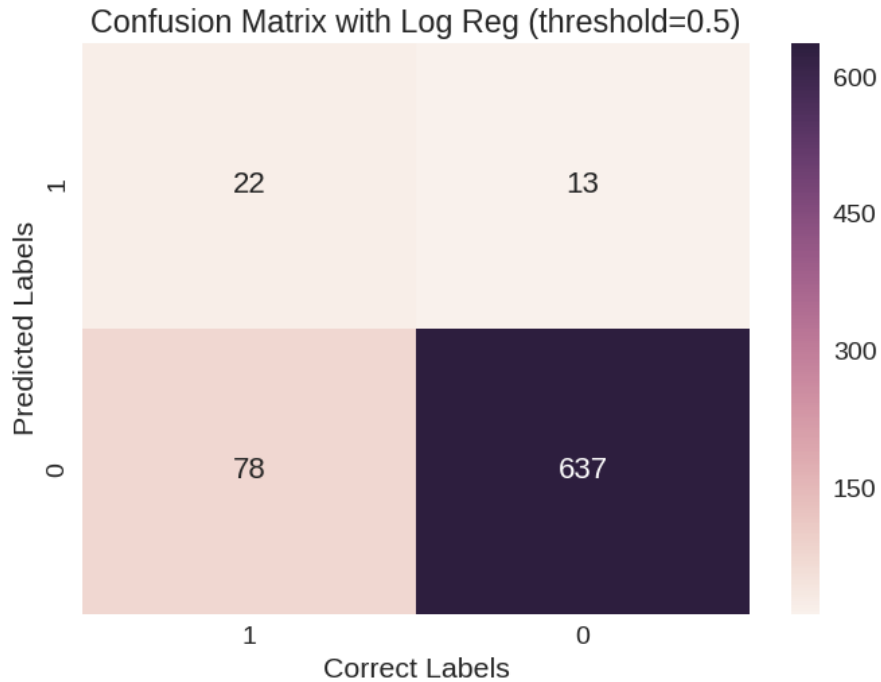


Model Selection

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



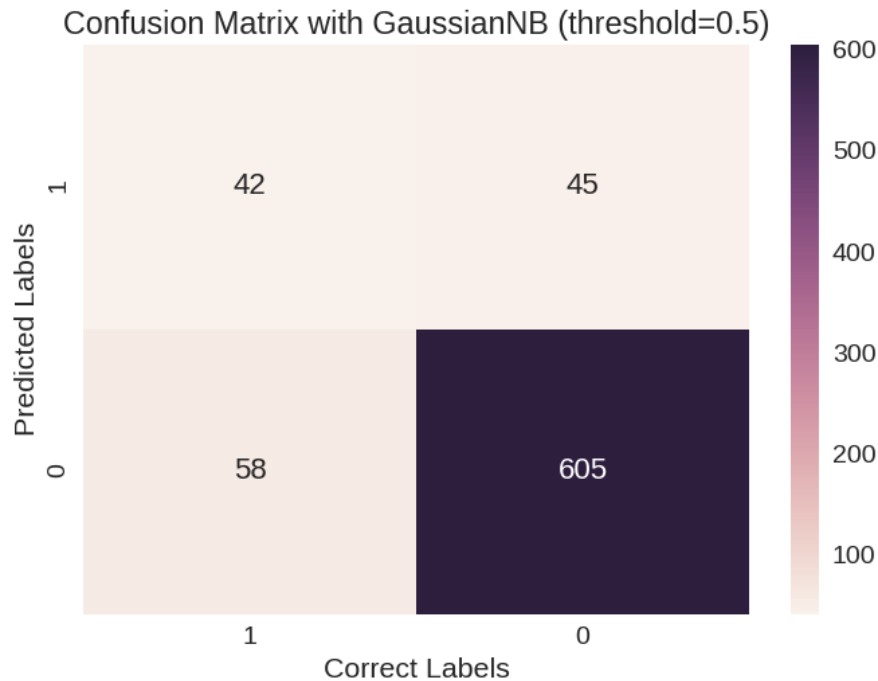
AUC: 0.833

Specificity: 0.98

Sensitivity/Recall: 0.22

Precision: 0.63

Naive Bayes'



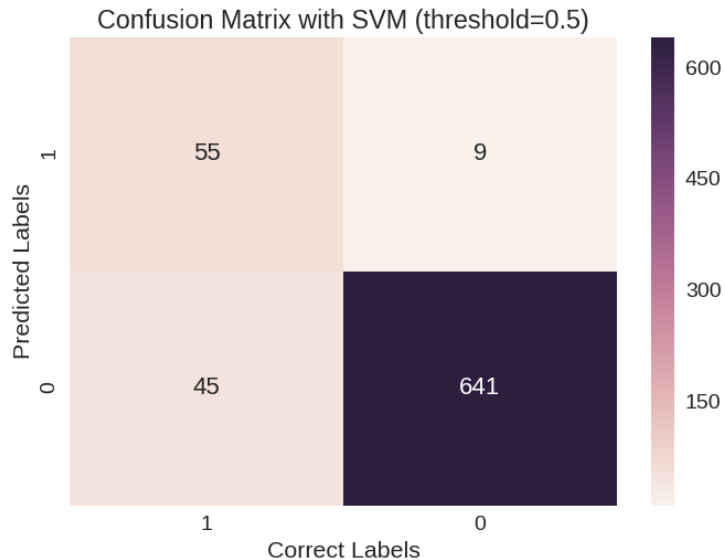
AUC: 0.865

Specificity: 0.93

Sensitivity/Recall: 0.42

Precision: 0.48

SVM Classifier



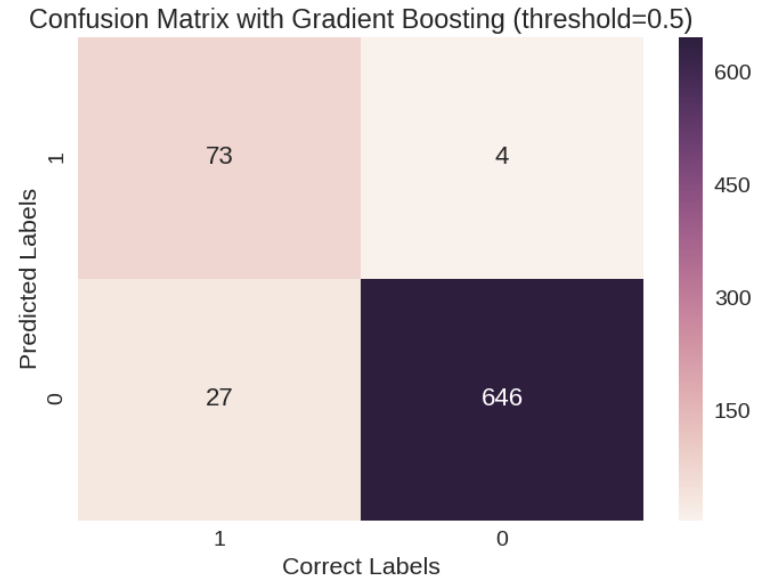
AUC: 0.911

Specificity: 0.99

Sensitivity/Recall: 0.55

Precision: 0.86

Gradient Boosting



AUC: 0.917

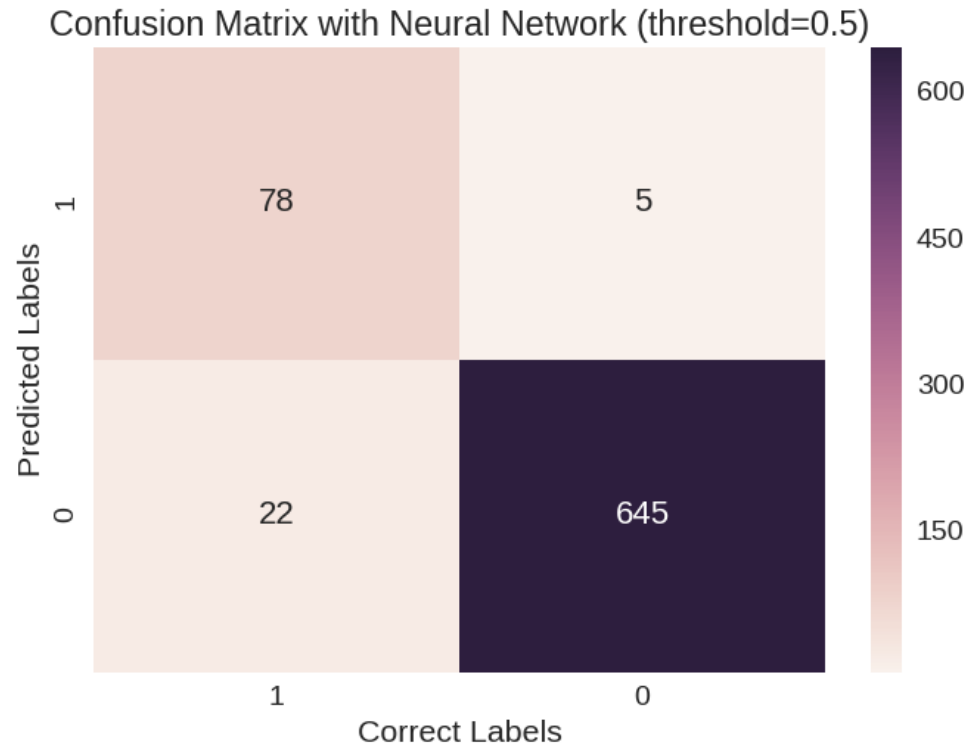
Specificity: 0.99

Sensitivity/Recall: 0.73

Precision: 0.95

Neural Network

Selected Model

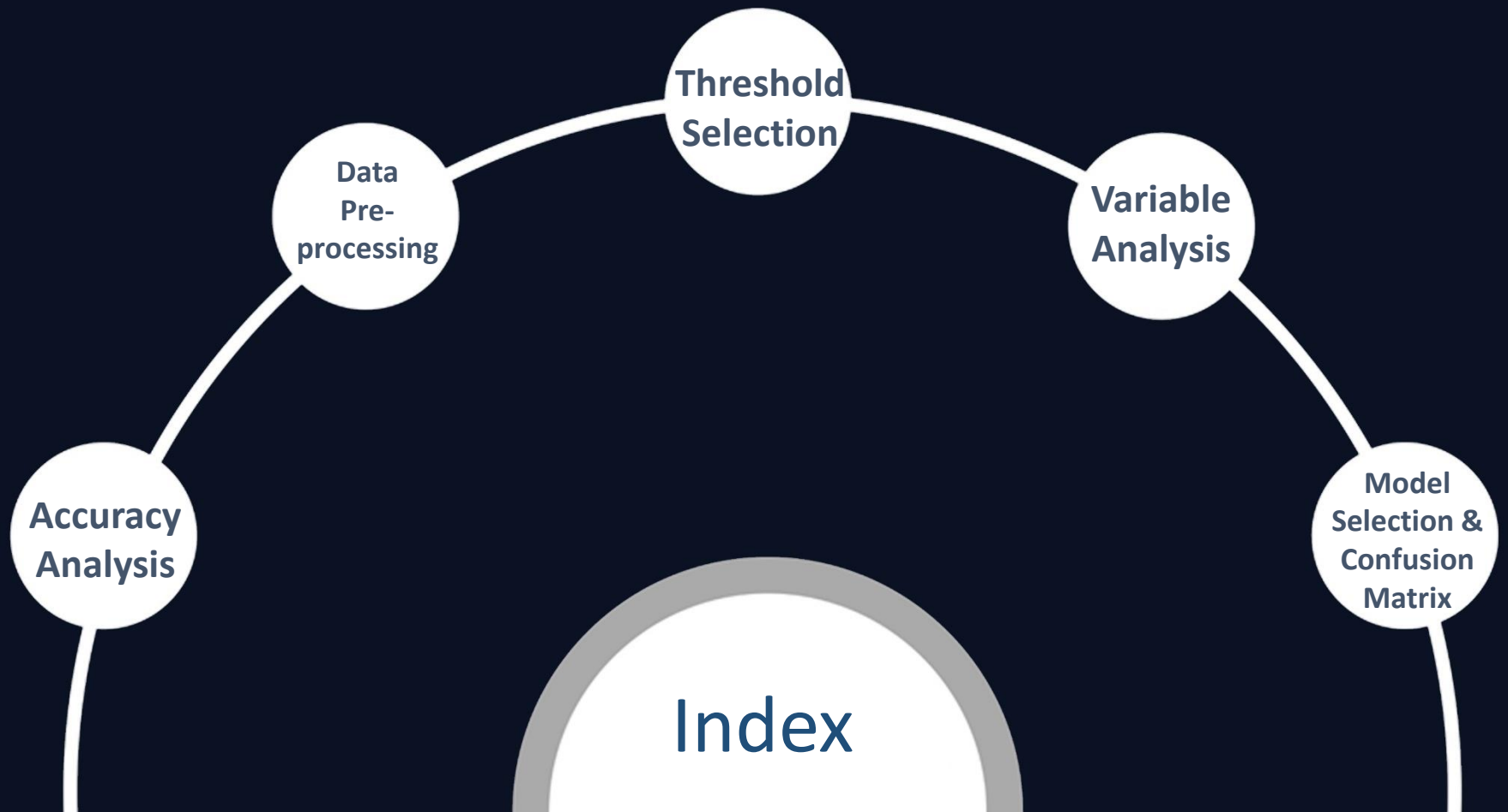


AUC: 0.927

Specificity: 0.99

Sensitivity/Recall: 0.78

Precision: 0.94



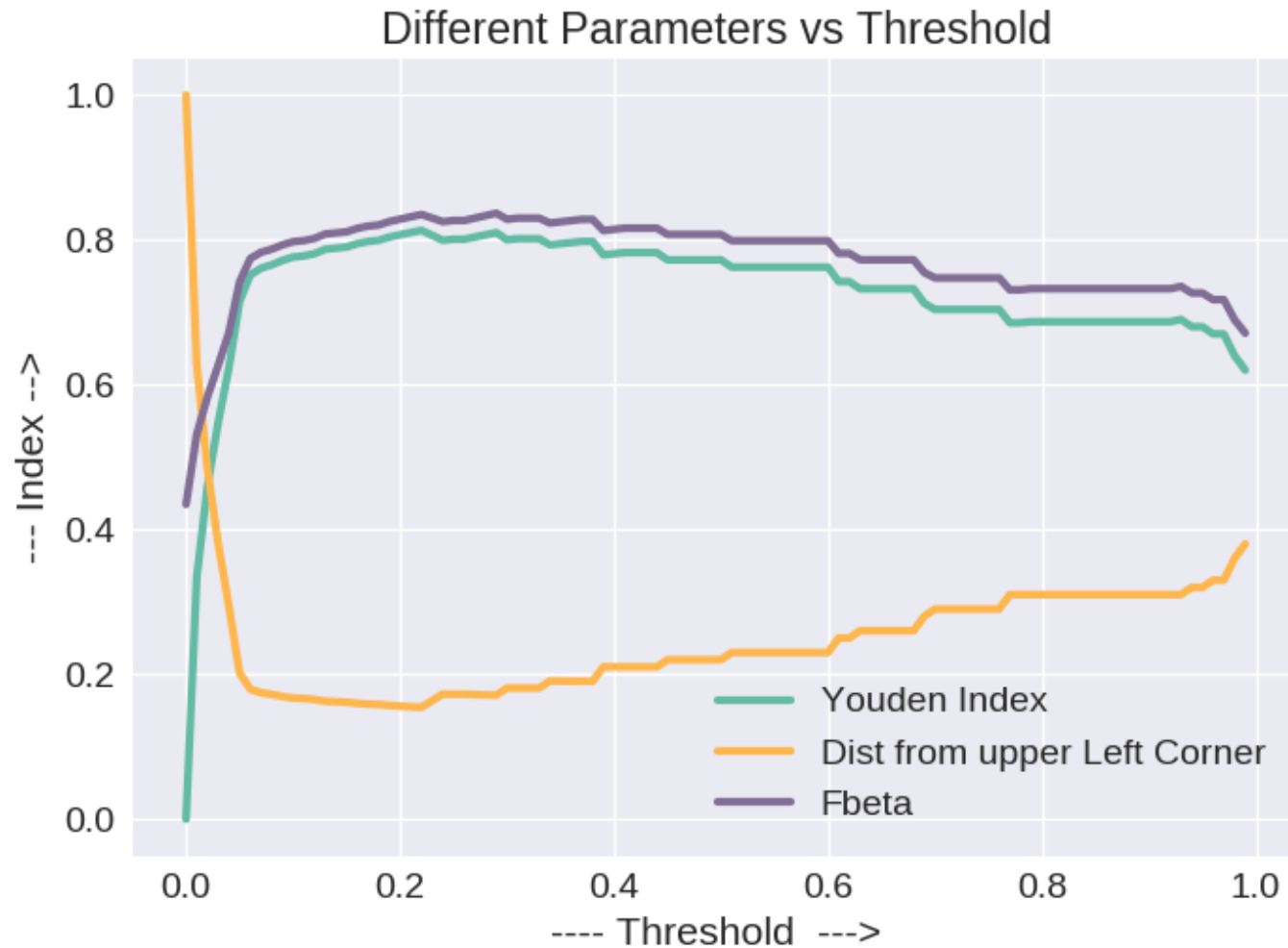
Threshold Selection

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 :

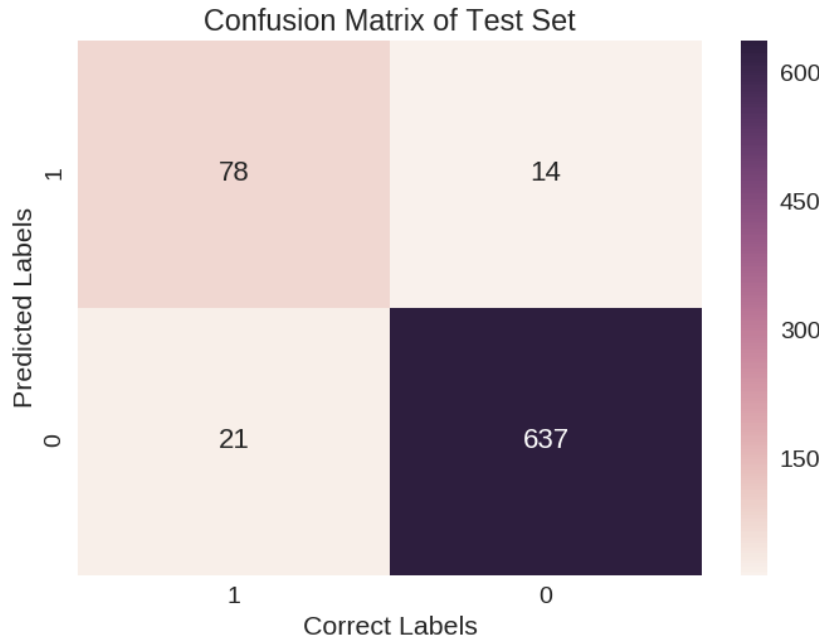
- **Youden Index:** $(\text{Sensitivity} + \text{Specificity} - 1)$
- **Distance of the point from the upper left corner on the ROC curve**
- **F-Beta value:** $2/(F_{\beta}) = (1/(\beta * \text{Recall})) + (1/(\text{precision}))$

against varying thresholds



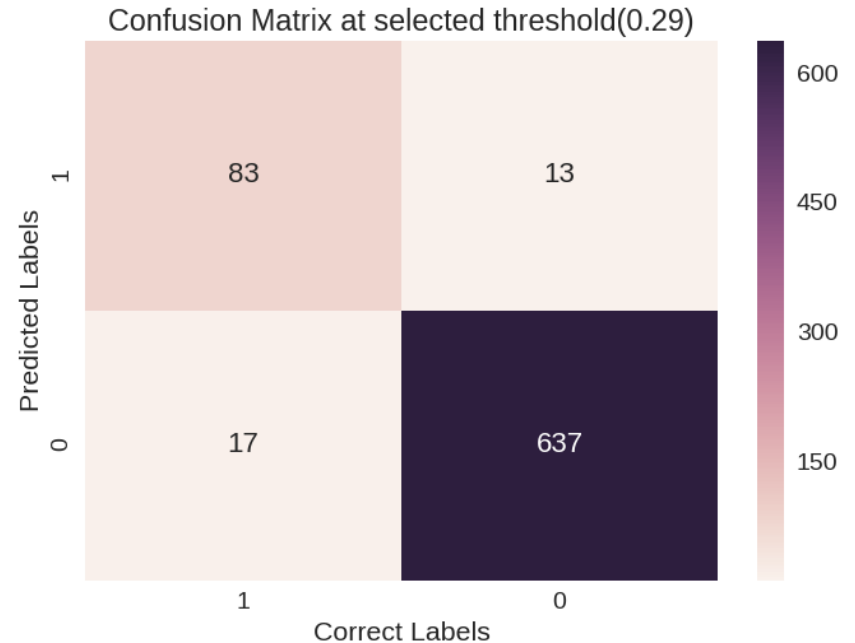
Selected Threshold: 0.29

Confusion Matrix



Test Data

Specificity: 0.98
Sensitivity/Recall: 0.81
Precision: 0.85
Test 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

```
graph TD; TS((Threshold Selection)) --- DP((Data Pre-processing)); DP --- AA((Accuracy Analysis)); AA --- VA((Variable Analysis)); VA --- MSCM((Model Selection & Confusion Matrix)); MSCM --- TS; AA --- Index((Index));
```

**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

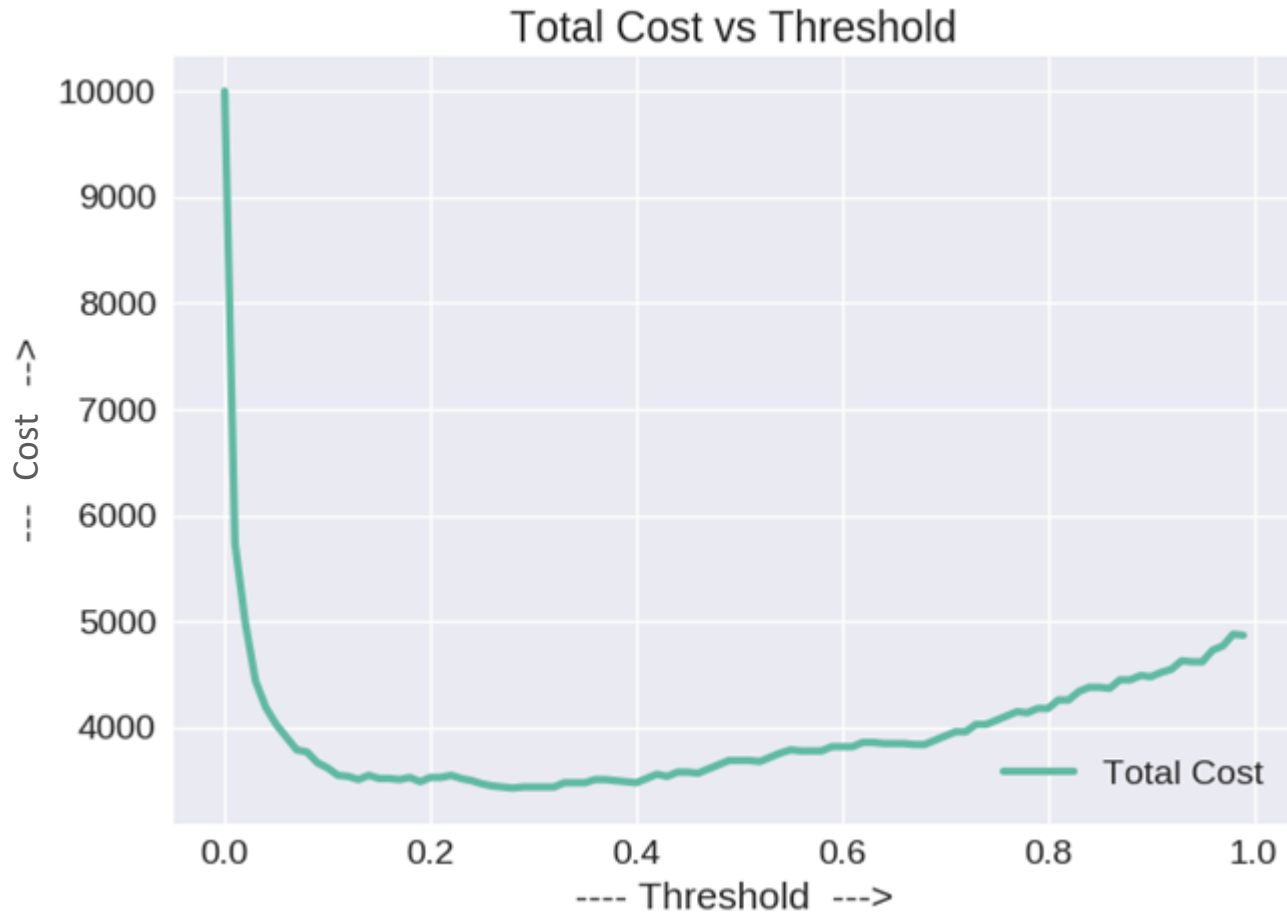
1	TP	FP
0	FN	TN
	1	0

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.

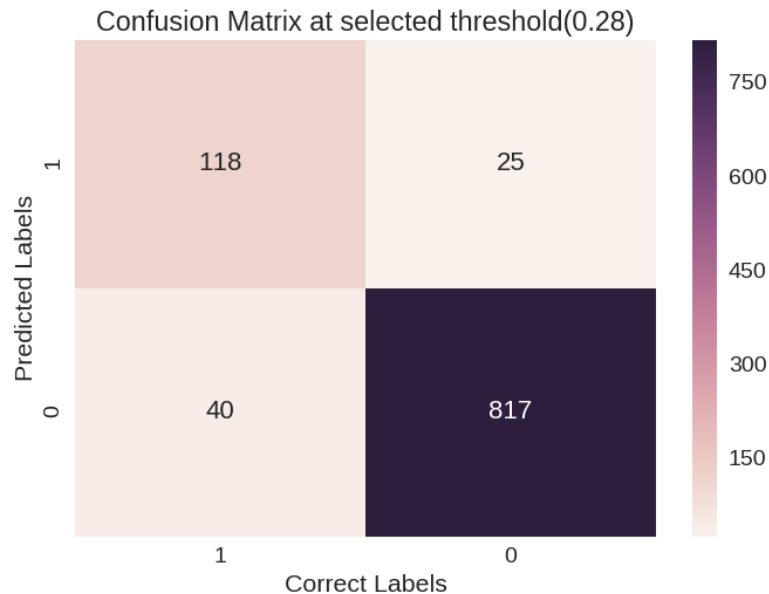
While we would be spending Cost of Acquisition on FN number of customers.

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

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



Selected Threshold: 0.28



Validation Data

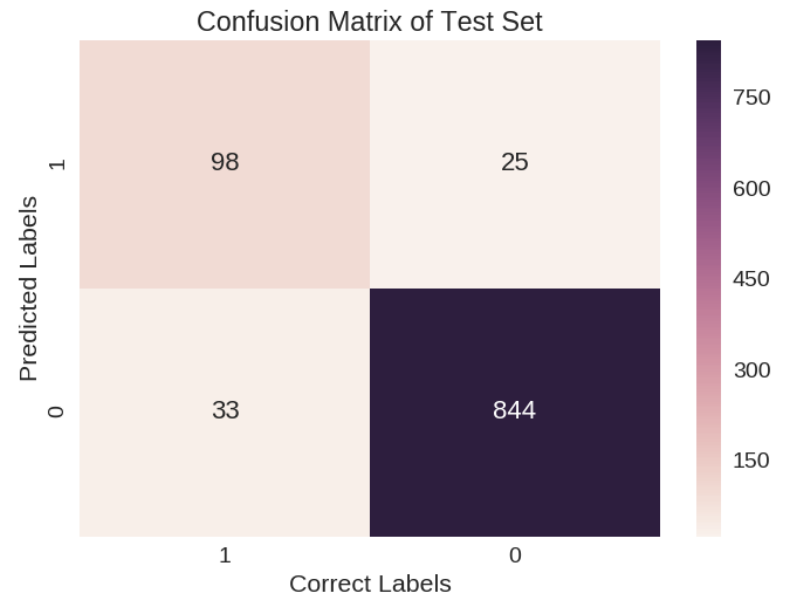
Specificity: 0.97

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

```

1 import numpy as np
2 from sklearn import preprocessing, cross_validation, svm
3 from sklearn.linear_model import LogisticRegression
4 from sklearn.ensemble import GradientBoostingClassifier
5 from sklearn.naive_bayes import GaussianNB
6 from sklearn.neural_network import MLPClassifier
7 import pandas as pd
8 from sklearn import metrics
9 import matplotlib.pyplot as plt
10 import seaborn as sns
11
12 plt.style.use('seaborn-darkgrid')
13 plt.rcParams['axes.facecolor'] = '#f7f1eb'
14
15 df = pd.read_csv('newccd.csv')
16 df.drop(['Id'], 1, inplace=True)
17
18 # df = df[['', '', '', '']]
19 X = df.drop(['churn'], 1)
20 y = list(df.churn)
21
22
23
24 ## Model Deciding through highest AUC (Area Under ROC curve)
25 print "Churn percentage in full data:", np.sum(1*(np.array(y)==1))/5000.0
26
27 ## Train Test Split
28 X_train, X_test, y_train, y_test = cross_validation.train_test_split(X, y, test_size=0.3)
29 X_val, X_test, y_val, y_test = cross_validation.train_test_split(X_test, y_test, test_size=0.5)
30 print "Churn percentage in training data:", np.sum(1*(np.array(y_train)==1))/float(len(y_train))
31 print "Churn percentage in val data:", np.sum(1*(np.array(y_val)==1))/float(len(y_val))
32 print "Churn percentage in test data:", np.sum(1*(np.array(y_test)==1))/float(len(y_test))
33
34
35
36 ## Model Selection
37 print "\nLogistic Regression Analysis:"
38 clf = LogisticRegression(solver='sag', max_iter=500)
39 ## Training
40 clf.fit(X_train, y_train)
41 prob = clf.predict_proba(X_val)[:, 1]
42 fpr1, tpr1, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1)
43 AUC1 = metrics.auc(fpr1, tpr1)
44 print "AUC: ", AUC1
45 ## Confusion Matrix at initial Threshold=0.5

```

```

43 AUC1 = metrics.auc(fpr1, tpr1)
44 print "AUC: ", AUC1
45 ## Confusion Matrix at initial Threshold=0.5
46 pred = (prob>=0.5)*1
47 tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
48 conf_matrix = np.array([[tp,fp],[fn,tn]])
49
50 sensitivity = tp/float(fn+tp)
51 specificity = tn/float(fp+tn)
52 precision = tp/float(tp+fp)
53 df_cm = pd.DataFrame(conf_matrix, index=["1","0"],columns=["1","0"])
54 print "Confusion Matrix at default Threshold=0.5"
55 print df_cm
56 print "Specificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2),"\\n"
57 sn.set(font_scale=1.4)
58 sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
59 plt.title("Confusion Matrix with Log Reg (threshold="+str(0.5)+")")
60 plt.xlabel("Correct Labels")
61 plt.ylabel("Predicted Labels")
62 plt.show()
63
64
65 ## Model Selection
66 print "\\nGaussian Naive Bayes Analysis:"
67 clf = GaussianNB()
68 ## Training
69 clf.fit(X_train, y_train)
70 prob = clf.predict_proba(X_val)[: ,1]
71 fpr2, tpr2, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1)
72 AUC2 = metrics.auc(fpr2, tpr2)
73 print "AUC : ", AUC2
74 ## Confusion Matrix at initial Threshold=0.5
75 pred = (prob>=0.5)*1
76 tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
77 conf_matrix = np.array([[tp,fp],[fn,tn]])
78
79 sensitivity = tp/float(fn+tp)
80 specificity = tn/float(fp+tn)
81 precision = tp/float(tp+fp)
82 df_cm = pd.DataFrame(conf_matrix, index=["1","0"],columns=["1","0"])
83 print "Confusion Matrix at default Threshold=0.5"
84 print df_cm
85 print "Specificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2),"\\n"
86 sn.set(font_scale=1.4)
87 sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='a')

```

```

85 print "Specificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2),"\\n"
86 sn.set(font_scale=1.4)
87 sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
88 plt.title("Confusion Matrix with GaussianNB (threshold="+str(0.5)+")")
89 plt.xlabel("Correct Labels")
90 plt.ylabel("Predicted Labels")
91 plt.show()
92
93 ## Model Selection
94 print "\\nSVM Analysis:"
95 clf = svm.SVC(probability=True,kernel='rbf')
96 ## Training
97 clf.fit(X_train, y_train)
98 prob = clf.predict_proba(X_val)[:,-1]
99 fpr3, tpr3, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1)
100 AUC3 = metrics.auc(fpr3, tpr3)
101 print "AUC : ", AUC3
102 ## Confusion Matrix at initial Threshold=0.5
103 pred = (prob>=0.5)*1
104 tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
105 conf_matrix = np.array([[tp,fp],[fn,tn]])
106
107 sensitivity = tp/float(fn+tp)
108 specificity = tn/float(fp+tn)
109 precision = tp/float(tp+fp)
110 df_cm = pd.DataFrame(conf_matrix, index=["1","0"],columns=["1","0"])
111 print "Confusion Matrix at default Threshold=0.5"
112 print df_cm
113 print "Specificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2),"\\n"
114 sn.set(font_scale=1.4)
115 sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
116 plt.title("Confusion Matrix with SVM (threshold="+str(0.5)+")")
117 plt.xlabel("Correct Labels")
118 plt.ylabel("Predicted Labels")
119 plt.show()
120
121
122 ## Model Selection
123 print "\\nGradient Boosting Analysis:"
124 clf = GradientBoostingClassifier()
125 ## Training
126 clf.fit(X_train, y_train)
127 prob = clf.predict_proba(X_val)[:,-1]
128 fpr4, tpr4, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1)
129 AUC4 = metrics.auc(fpr4, tpr4)

```

```

121
122 ## Model Selection
123 print "\nGradient Boosting Analysis:"
124 clf = GradientBoostingClassifier()
125 ## Training
126 clf.fit(X_train, y_train)
127 prob = clf.predict_proba(X_val)[: ,1]
128 fpr4, tpr4, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1)
129 AUC4 = metrics.auc(fpr4, tpr4)
130 print "AUC : ", AUC4
131 ## Confusion Matrix at initial Threshold=0.5
132 pred = (prob>=0.5)*1
133 tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
134 conf_matrix = np.array([[tp,fp],[fn,tn]])
135
136 sensitivity = tp/float(fn+tp)
137 specificity = tn/float(fp+tn)
138 precision = tp/float(tp+fp)
139 df_cm = pd.DataFrame(conf_matrix, index=["1","0"],columns=["1","0"])
140 print "Confusion Matrix at default Threshold=0.5"
141 print df_cm
142 print "Specificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2),"|
143 sn.set(font_scale=1.4)
144 sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
145 plt.title("Confusion Matrix with Gradient Boosting (threshold="+str(0.5)+")")
146 plt.xlabel("Correct Labels")
147 plt.ylabel("Predicted Labels")
148 plt.show()
149
150
151 ## Model Selection
152 print "\nNeural Network Analysis:"
153 clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(10,), random_state=1,activation='relu',max_iter=300)
154 ## Training
155 clf.fit(X_train, y_train)
156 prob = clf.predict_proba(X_val)[: ,1]
157 fpr5, tpr5, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1,drop_intermediate=False)
158 AUC5 = metrics.auc(fpr5, tpr5)
159 print "AUC : ", AUC5,"\n"
160 ## Confusion Matrix at initial Threshold=0.5
161 pred = (prob>=0.5)*1
162 tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
163 conf_matrix = np.array([[tp,fp],[fn,tn]])
164
165 sensitivitv = to/float(fn+to)

```



```

151 ## Model Selection
152 print "\nNeural Network Analysis:"
153 clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(10,), random_state=1,activation='relu',max_iter=300)
154 ## Training
155 clf.fit(X_train, y_train)
156 prob = clf.predict_proba(X_val)[:,:1]
157 fpr5, tpr5, thresholds = metrics.roc_curve(np.array(y_val), prob, pos_label=1,drop_intermediate=False)
158 AUC5 = metrics.auc(fpr5, tpr5)
159 print "AUC : ", AUC5,"\n"
160 ## Confusion Matrix at initial Threshold=0.5
161 pred = (prob>=0.5)*1
162 tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
163 conf_matrix = np.array([[tp,fp],[fn,tn]])
164
165 sensitivity = tp/float(fn+tp)
166 specificity = tn/float(fp+tn)
167 precision = tp/float(tp+fp)
168 df_cm = pd.DataFrame(conf_matrix, index=["1","0"],columns=["1","0"])
169 print "Confusion Matrix at default Threshold=0.5"
170 print df_cm
171 print "Specificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2),"|
172 sn.set(font_scale=1.4)
173 sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
174 plt.title("Confusion Matrix with Neural Network (threshold="+str(0.5)+")")
175 plt.xlabel("Correct Labels")
176 plt.ylabel("Predicted Labels")
177 plt.show()
178
179
180
181 ## plotting ROC curve for all the models
182 x = np.arange(0.0,1.0,0.01)
183 plt.plot(x,x,'--',color="#00a2d9",linewidth=1)
184 plt.plot(fpr1,tpr1,label="Logistic Regression(Area:"+str(round(AUC1,2))+")", color="cyan",linewidth=1)
185 plt.plot(fpr2,tpr2,label="Gaussian Naive Bayes(Area:"+str(round(AUC2,2))+")", color="black",linewidth=1)
186 plt.plot(fpr3,tpr3,label="SVM RBF(Area:"+str(round(AUC3,2))+")", color="red",linewidth=1)
187 plt.plot(fpr4,tpr4,label="Gradient Boosting(Area:"+str(round(AUC4,2))+")", color="#ffb74d",linewidth=1)
188 plt.plot(fpr5,tpr5,label="Neural Network(Area:"+str(round(AUC5,2))+")", color="#62bba4",linewidth=1)
189
190 plt.legend(loc='lower right')
191 plt.title("ROC Curve for different classifiers")
192 plt.xlabel("---- False Positive Rate --->")
193 plt.ylabel("---- True Positive Rate --->")
194 plt.show()
195

```



```

196
197 ## Choosing the suitable Threshold Value at point with lowest distance from the upper left corner on the ROC curve
198 x = np.arange(0.0,1.0,0.01)
199 print "Choosing threshold at point closer to upper left corner in ROC curve:"
200 YIndexList = []
201 dList = []
202 fb = []
203 Highest_YIndex=0 # Initializing it to zero
204 Lowest_d = 1
205 highest_Fbeta=0
206 for threshold in x:
207     pred = (prob>=threshold)*1
208     tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
209     sensitivity = tp/float(fn+tp)
210     specificity = tn/float(fp+tn)
211     d = np.sqrt(np.power(1-sensitivity,2)+np.power(1-specificity,2))
212     YIndex=sensitivity+specificity-1
213     fbeta = metrics.fbeta_score(y_val, pred,beta=2)
214     if fbeta>=highest_Fbeta:
215         highest_Fbeta=fbeta
216         selected_threshold=threshold
217     YIndexList.append(YIndex)
218     dList.append(d)
219     fb.append(fbeta)
220
221 plt.plot(x,YIndexList,color='#62bba4',label="Youden Index",linewidth=3)
222 plt.plot(x,dList,color='#ffb74d',label="Dist from upper Left Corner",linewidth=3)
223 plt.plot(x, fb,color='#816E94',label="Fbeta",linewidth=3)
224 plt.legend(loc='lower right')
225 plt.title("Different Parameters vs Threshold")
226 plt.xlabel("---- Threshold --->")
227 plt.ylabel("---- Index -->")
228 plt.show()
229 ## Confusion Matrix at optimum threshold value
230 pred = (prob>=selected_threshold)*1
231 tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
232 conf_matrix = np.array([[tp,fp],[fn,tn]])
233 df_cm = pd.DataFrame(conf_matrix, index=["1","0"],columns=["1","0"])
234 print "Confusion Matrix at selected threshold=", selected_threshold
235 print df_cm
236
237 sn.set(font_scale=1.4)
238 sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
239 plt.xlabel("Correct Labels")
240 plt.ylabel("Predicted Labels")

```

```

235 print df_cm
236
237 sn.set(font_scale=1.4)
238 sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
239 plt.xlabel("Correct Labels")
240 plt.ylabel("Predicted Labels")
241 plt.title("Confusion Matrix at selected threshold("+str(selected_threshold)+")")
242 plt.show()
243
244
245 sensitivity = tp/float(fn+tp)
246 specificity = tn/float(fp+tn)
247 precision = tp/float(tp+fp)
248 acc = (tp+tn)/float(tp+tn+fn+fp)
249
250 prob = clf.predict_proba(X_train)[: ,1]
251 fpr, tpr, thresholds = metrics.roc_curve(np.array(y_train), prob, pos_label=1)
252 Training_AUC = metrics.auc(fpr, tpr)
253
254 print "\nSpecificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:",np.round(precision,2)
255 print "val AUC:",AUC5,"| Training AUC:",Training_AUC,"| val. Accuracy:",acc
256
257
258 print "\nTest set Analysis:"
259 prob = clf.predict_proba(X_test)[: ,1]
260 fpr, tpr, thresholds = metrics.roc_curve(np.array(y_test), prob, pos_label=1)
261 Val_AUC = metrics.auc(fpr, tpr)
262
263 pred = (prob>=selected_threshold)*1
264 tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_test), pred).ravel()
265 conf_matrix = np.array([[tp,fp],[fn,tn]])
266 df_cm = pd.DataFrame(conf_matrix, index=["1","0"],columns=["1","0"])
267 print "Confusion Matrix of Test Set:"
268 print df_cm
269
270 sn.set(font_scale=1.4)
271 sn.heatmap(df_cm, annot=True,annot_kws={"size": 16},fmt='g')
272 plt.xlabel("Correct Labels")
273 plt.ylabel("Predicted Labels")
274 plt.title("Confusion Matrix of Test Set")
275 plt.show()
276
277 sensitivity = tp/float(fn+tp)
278 specificity = tn/float(fp+tn)
279 precision = tp/float(tp+fp)

```

```

274 plt.title("Confusion Matrix of Test Set")
275 plt.show()
276
277 sensitivity = tp/float(fn+tp)
278 specificity = tn/float(fp+tn)
279 precision = tp/float(tp+fp)
280 acc = (tp+tn)/float(tp+tn+fn+fp)
281 print "\nSpecificity:", np.round(specificity,2),"| Sensitivity/Recall:", np.round(sensitivity,2),"| Precision:", np.round(precision,2)
282 print "Test AUC:", Val_AUC,"| Test Accuracy:", acc
283
284 ## Choosing the suitable Threshold Value using
285 x = np.arange(0.0,1.0,0.01)
286 print "Threshold selection:"
287 YIndexList = []
288 dList = []
289 fb = []
290 cost_list = []
291 Highest_YIndex=0 # Initializing it to zero
292 Lowest_d = 1
293 highest_Fbeta=0
294 tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), (prob>=0.01)*1).ravel()
295 ratio = 5
296 lowest_cost = 10*(tp+fp)+ratio*10*fn
297 for threshold in x:
298     pred = (prob>=threshold)*1
299     tn, fp, fn, tp = metrics.confusion_matrix(np.array(y_val), pred).ravel()
300     sensitivity = tp/float(fn+tp)
301     specificity = tn/float(fp+tn)
302     d = np.sqrt(np.power(1-sensitivity,2)+np.power(1-specificity,2))
303     YIndex=specificity+sensitivity-1
304     fbeta = metrics.fbeta_score(y_val, pred,beta=2)
305     cost = 10*(tp+fp)+ratio*10*fn
306     if cost<=lowest_cost:
307         lowest_cost=cost
308         selected_threshold=threshold
309     YIndexList.append(YIndex)
310     dList.append(d)
311     fb.append(fbeta)
312     cost_list.append(cost)
313 cost_list=np.array(cost_list)
314 # cost_list = cost_list/float(np.max(cost_list))

```