College Data

The college data set contains statistics for a large number of US colleges. There are 777 observations and 18 variables. The data is used to compare colleges both private and public based on a variety of attributes to help a similarly varied student population. There are 17 quantitative variables and 1 qualitative categorical variable labeled 'Private'. The categorical variable has 'Yes' and 'No' labels. This is a typical binary classification problem. The goal is to predict whether a college is private or public based on a combination of data attributes provided for each college.

Data source: the data set was taken from Github in csv file format and is maintained on the following repository: [https://github.com/selva86/datasets/blob/master/college.csv (https://github.com/selva86/datasets/blob/master/college.csv)](https://github.com/selva86/datasets/blob/master/college.csv)

In [3]:
```
# Import pyspark package
import pyspark
```

In [4]:
```
# Import Pyspark and SparkSession library
from pyspark.sql import SparkSession
```

In [5]:
```
spark = SparkSession.builder.appName('classifier').getOrCreate()
```

In [6]:
```
# Load the data
df = spark.read.csv('College.csv',inferSchema=True,header=True)
```

In [7]:
```
df.head(2)
```

Out[7]:
```
[Row(School='Abilene Christian University', Private='Yes', Apps=1660, Accept=12
32, Enroll=721, Top10perc=23, Top25perc=52, F_Undergrad=2885, P_Undergrad=537,
Outstate=7440, Room_Board=3300, Books=450, Personal=2200, PhD=70, Terminal=78,
S_F_Ratio=18.1, perc_alumni=12, Expend=7041, Grad_Rate=60),
 Row(School='Adelphi University', Private='Yes', Apps=2186, Accept=1924, Enroll
=512, Top10perc=16, Top25perc=29, F_Undergrad=2683, P_Undergrad=1227, Outstate=
12280, Room_Board=6450, Books=750, Personal=1500, PhD=29, Terminal=30, S_F_Rati
o=12.2, perc_alumni=16, Expend=10527, Grad_Rate=56)]
```

In [8]:
```
df.printSchema()
```

```
root
 |-- School: string (nullable = true)
 |-- Private: string (nullable = true)
 |-- Apps: integer (nullable = true)
 |-- Accept: integer (nullable = true)
 |-- Enroll: integer (nullable = true)
 |-- Top10perc: integer (nullable = true)
 |-- Top25perc: integer (nullable = true)
 |-- F_Undergrad: integer (nullable = true)
 |-- P_Undergrad: integer (nullable = true)
 |-- Outstate: integer (nullable = true)
 |-- Room_Board: integer (nullable = true)
 |-- Books: integer (nullable = true)
 |-- Personal: integer (nullable = true)
 |-- PhD: integer (nullable = true)
 |-- Terminal: integer (nullable = true)
 |-- S_F_Ratio: double (nullable = true)
 |-- perc_alumni: integer (nullable = true)
 |-- Expend: integer (nullable = true)
 |-- Grad_Rate: integer (nullable = true)
```

In [9]:
```
# Import VectorAssembler and Vectors
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

In [10]:
```
df.columns
```

Out[10]:
```
['School',
 'Private',
 'Apps',
 'Accept',
 'Enroll',
 'Top10perc',
 'Top25perc',
 'F_Undergrad',
 'P_Undergrad',
 'Outstate',
 'Room_Board',
 'Books',
 'Personal',
 'PhD',
 'Terminal',
 'S_F_Ratio',
 'perc_alumni',
 'Expend',
 'Grad_Rate']
```

In [11]:
```
# Dropping irrelevant columns (for the target variable)
assembler = VectorAssembler(
  inputCols=['Apps',
             'Accept',
             'Enroll',
             'Top10perc',
             'Top25perc',
             'F_Undergrad',
             'P_Undergrad',
             'Outstate',
             'Room_Board',
             'Books',
             'Personal',
             'PhD',
             'Terminal',
             'S_F_Ratio',
             'perc_alumni',
             'Expend',
             'Grad_Rate'],
              outputCol="features")
```

In [12]:
```
import pandas as pd
import numpy as np
```

In [13]:
```
pd.DataFrame(df.take(5), columns=df.columns).transpose()
```

Out[13]:

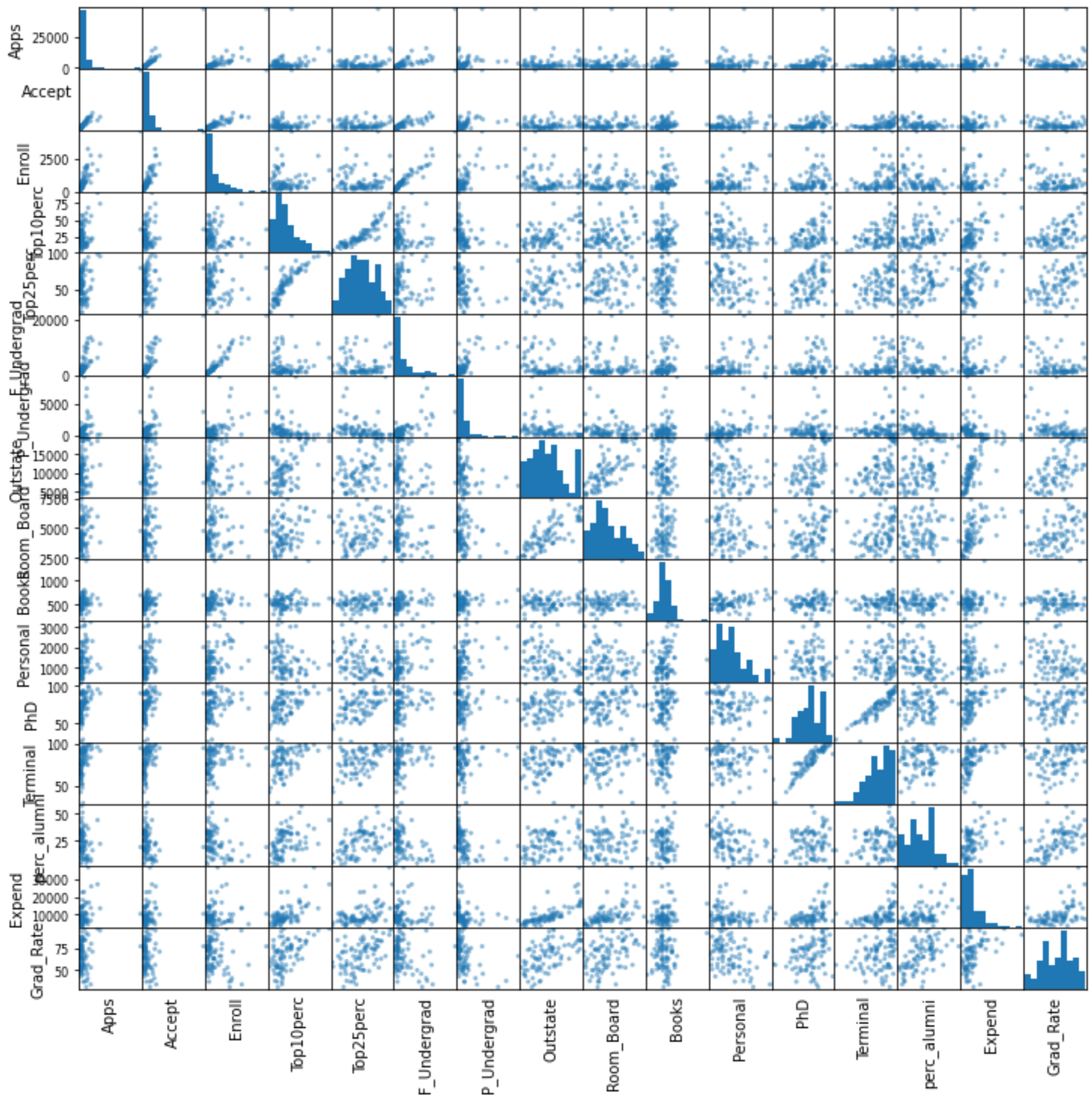
	0	1	2	3	4
School	Abilene Christian University	Adelphi University	Adrian College	Agnes Scott College	Alaska Pacific University
Private	Yes	Yes	Yes	Yes	Yes
Apps	1660	2186	1428	417	193
Accept	1232	1924	1097	349	146
Enroll	721	512	336	137	55
Top10perc	23	16	22	60	16
Top25perc	52	29	50	89	44
F_Undergrad	2885	2683	1036	510	249
P_Undergrad	537	1227	99	63	869
Outstate	7440	12280	11250	12960	7560
Room_Board	3300	6450	3750	5450	4120
Books	450	750	400	450	800
Personal	2200	1500	1165	875	1500
PhD	70	29	53	92	76
Terminal	78	30	66	97	72
S_F_Ratio	18.1	12.2	12.9	7.7	11.9
perc_alumni	12	16	30	37	2
Expend	7041	10527	8735	19016	10922
Grad_Rate	60	56	54	59	15

In [14]:
```
# Summary Statistics
num_features=[t[0] for t in df.dtypes if t[1]=='int']
df.select(num_features).describe().toPandas().transpose()
```

Out[14]:

	0	1	2	3	4
summary	count	mean	stddev	min	max
Apps	777	3001.6383526383524	3870.2014844352884	81	48094
Accept	777	2018.8043758043757	2451.11397099263	72	26330
Enroll	777	779.972972972973	929.17619013287	35	6392
Top10perc	777	27.55855855855856	17.640364385452134	1	96
Top25perc	777	55.7966537966538	19.804777595131373	9	100
F_Undergrad	777	3699.907335907336	4850.420530887386	139	31643
P_Undergrad	777	855.2985842985843	1522.431887295513	1	21836
Outstate	777	10440.66924066924	4023.0164841119727	2340	21700
Room_Board	777	4357.526383526383	1096.6964155935289	1780	8124
Books	777	549.3809523809524	165.10536013709253	96	2340
Personal	777	1340.6422136422136	677.071453590578	250	6800
PhD	777	72.66023166023166	16.328154687939314	8	103
Terminal	777	79.70270270270271	14.722358527903374	24	100
perc_alumni	777	22.743886743886744	12.39180148937615	0	64
Expend	777	9660.17117117117	5221.76843985609	3186	56233
Grad_Rate	777	65.46332046332046	17.177709897155403	10	118

In [15]:
```python
# Plotting the data to visualize correlations
from pandas.plotting._misc import scatter_matrix
import matplotlib as plt
%matplotlib inline
num_data=df.select(num_features).sample(False,0.10).toPandas()

axs=scatter_matrix(num_data, figsize=(12,12));

n=len(num_data.columns)
for i in range(n):
    v=axs[1,0]
    v.yaxis.label.set_rotation(0)
    v.yaxis.label.set_ha('right')
    v.set_yticks(())
    h=axs[n-1,i]
    h.xaxis.label.set_rotation(90)
    h.set_xticks(())
```



Feature Creation

In [16]:
```
# Transforming the data
output = assembler.transform(df)
```

In [17]:
```
# importing StringIndexer
from pyspark.ml.feature import StringIndexer
```

In [18]:
```
# Fit the data to include all labels in index and identify categorical features
to be indexed
indexer = StringIndexer(inputCol="Private", outputCol="Private_Index")
output_fixed = indexer.fit(output).transform(output)
```

In [19]:
```
final_data = output_fixed.select("features",'Private_Index')
```

In [20]:
```
labelCol='Private_Index'
featuresCol='features'
```

Building Logistic Regression Model

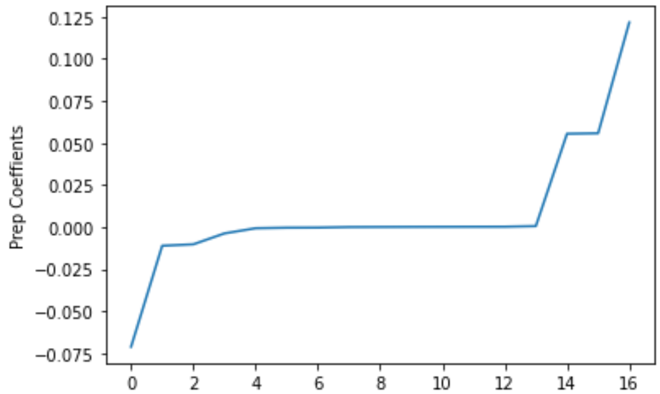
In [21]:
```
# Split the data into training and test sets etting test with 20% held out for
testing
train_data,test_data = final_data.randomSplit([0.8,0.2])
```

In [22]:
```
# Importing libraries for LR and Ensemble models
from pyspark.ml.classification import (DecisionTreeClassifier,GBTClassifier,
                                       RandomForestClassifier, LogisticRegressi
on)
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
```

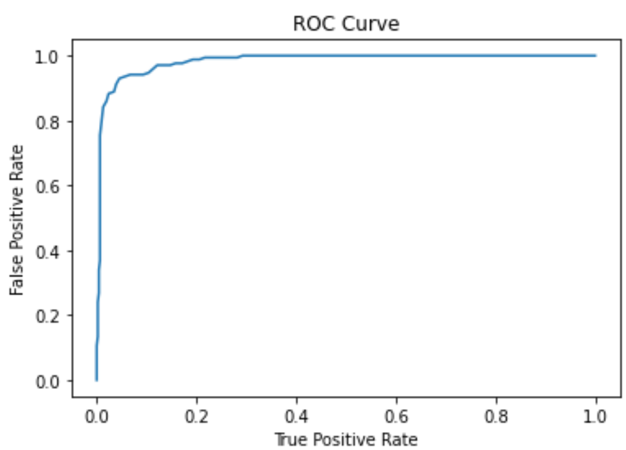
In [23]:
```
# Train a LR model
lr=LogisticRegression(labelCol='Private_Index', featuresCol='features', maxIter
=10)
```

In [24]:
```
# Fit the LR training data
lr_model = lr.fit(train_data)
```

In [25]:
```
# Plot the model coefficients
import matplotlib.pyplot as plt
prep=np.sort(lr_model.coefficients)

plt.plot(prep)
plt.ylabel('Prep Coeffients')
plt.show()
```

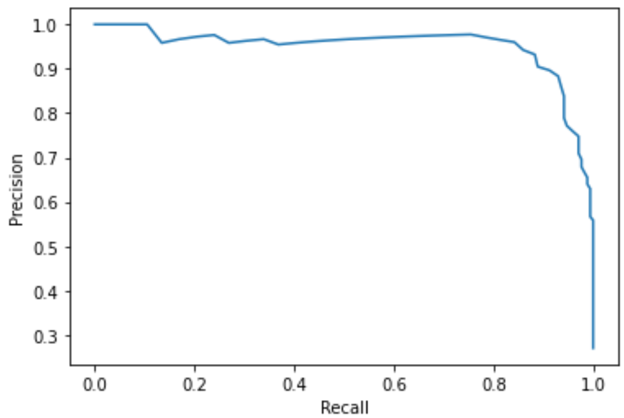


In [26]:
```
# Plot the training set area Under the Curve (ROC)

trainingSummary = lr_model.summary

roc=trainingSummary.roc.toPandas()
plt.plot(roc['FPR'],roc['TPR'])
plt.ylabel('False Positive Rate')
plt.xlabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()

print('Training set areaUnderROC: '+
     str(trainingSummary.areaUnderROC))
```




```
Training set areaUnderROC: 0.983426570765308
```

In [27]:
```
# Plot the precision and Recall

pr=trainingSummary.pr.toPandas()
plt.plot(pr['recall'],pr['precision'])
plt.ylabel('Precision')
plt.xlabel('Recall')
plt.show()
```



In [28]:
```
# Output the prediction
predictions = lr_model.transform(test_data)
```

In [29]:
```
predictions.select('Private_Index','rawPrediction','prediction','probability').
show(15)
```

```
+-------------+--------------------+----------+--------------------+
|Private_Index|       rawPrediction|prediction|         probability|
+-------------+--------------------+----------+--------------------+
|          0.0|[5.72734501315134...|       0.0|[0.99675485561250...|
|          0.0|[7.50228433022565...|       0.0|[0.99944848195525...|
|          0.0|[8.39166993487219...|       0.0|[0.99977330309602...|
|          1.0|[-0.3919311458612...|       1.0|[0.40325250321875...|
|          0.0|[3.24366935276524...|       0.0|[0.96244496227335...|
|          0.0|[6.23598644305756...|       0.0|[0.99804612798478...|
|          0.0|[6.90356083187637...|       0.0|[0.99899680415642...|
|          0.0|[5.15968295478223...|       0.0|[0.99428927928795...|
|          0.0|[6.88408642893246...|       0.0|[0.99897709622317...|
|          0.0|[5.94505648685292...|       0.0|[0.99738808554353...|
|          0.0|[7.90173104767709...|       0.0|[0.99963003460418...|
|          0.0|[-2.7329261186684...|       1.0|[0.06105819255378...|
|          0.0|[1.99764254521529...|       0.0|[0.88054933806327...|
|          0.0|[5.46411907271369...|       0.0|[0.99578179753101...|
|          0.0|[5.05534513005442...|       0.0|[0.99366521915903...|
+-------------+--------------------+----------+--------------------+
only showing top 15 rows
```

In [30]:
```
# Import BCE to evaluate Logistic Regression Model
from pyspark.ml.evaluation import BinaryClassificationEvaluator
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib as plt
from matplotlib import pyplot
%matplotlib inline
```

In [31]:
```
# Evaluate prediction and compute test error
evaluator=BinaryClassificationEvaluator(labelCol="Private_Index")
lr_acc=evaluator.evaluate(predictions)
print('Test Area Under ROC ',lr_acc)
```

```
Test Area Under ROC  0.9584896810506569
```

Building Classification Models

In [32]:
```
# Using an ensemble model of three classifiers to compare to LR results

dtc = DecisionTreeClassifier(labelCol='Private_Index',featuresCol='features')
rfc = RandomForestClassifier(labelCol='Private_Index',featuresCol='features')
gbt = GBTClassifier(labelCol='Private_Index',featuresCol='features')
```

In [33]:
```
# Training the models
dtc_model = dtc.fit(train_data)
rfc_model = rfc.fit(train_data)
gbt_model = gbt.fit(train_data)
```

In [34]:
```
# Model prediction comparison
dtc_predictions = dtc_model.transform(test_data)
rfc_predictions = rfc_model.transform(test_data)
gbt_predictions = gbt_model.transform(test_data)
```

In [35]:
```
# Computing test errors
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

acc_evaluator = MulticlassClassificationEvaluator(labelCol="Private_Index", pre
dictionCol="prediction", metricName="accuracy")
```

In [36]:
```
# Evaluating models
dtc_acc = acc_evaluator.evaluate(dtc_predictions)
rfc_acc = acc_evaluator.evaluate(rfc_predictions)
gbt_acc = acc_evaluator.evaluate(gbt_predictions)
```

In [37]:
```
print('-'*80)
print('Decision Tree accuracy: {0:2.2f}%'.format(dtc_acc*100))
print('-'*80)
print('Random Forest accuracy: {0:2.2f}%'.format(rfc_acc*100))
print('-'*80)
print('Gradient Boosting accuracy: {0:2.2f}%'.format(gbt_acc*100))
```

```
-------------------------------------------------------------------------------
-
Decision Tree accuracy: 88.28%
-------------------------------------------------------------------------------
-
Random Forest accuracy: 91.72%
-------------------------------------------------------------------------------
-
Gradient Boosting accuracy: 88.97%
```

Evaluating & Tuning the GBT Classifier

To see if we could improve the prediction we'll tune the GBT Classifier.

In [38]:
```
gbt = GBTClassifier(labelCol='Private_Index',featuresCol='features',maxIter=10)
gbtModel=gbt.fit(train_data)
predictions=gbtModel.transform(test_data)
predictions.select('Private_Index','rawPrediction','prediction','probability').
show(15)
```

```
+-------------+--------------------+----------+--------------------+
|Private_Index|       rawPrediction|prediction|         probability|
+-------------+--------------------+----------+--------------------+
|          0.0|[1.33783013041163...|       0.0|[0.93557504242290...|
|          0.0|[1.32573928510657...|       0.0|[0.93410206292642...|
|          0.0|[1.32573928510657...|       0.0|[0.93410206292642...|
|          1.0|[1.23389058829213...|       0.0|[0.92185206852881...|
|          0.0|[1.15971812693003...|       0.0|[0.91047399973305...|
|          0.0|[1.32573928510657...|       0.0|[0.93410206292642...|
|          0.0|[1.32573928510657...|       0.0|[0.93410206292642...|
|          0.0|[1.32600689658882...|       0.0|[0.93413500113583...|
|          0.0|[1.32573928510657...|       0.0|[0.93410206292642...|
|          0.0|[1.32573928510657...|       0.0|[0.93410206292642...|
|          0.0|[1.32573928510657...|       0.0|[0.93410206292642...|
|          0.0|[1.11507054117874...|       0.0|[0.90292372593374...|
|          0.0|[1.23567946527255...|       0.0|[0.92210942420479...|
|          0.0|[1.32573928510657...|       0.0|[0.93410206292642...|
|          0.0|[1.32520005409080...|       0.0|[0.93403564667863...|
+-------------+--------------------+----------+--------------------+
only showing top 15 rows
```

In [39]:
```
# Evaluate the GBTClassifier
evaluator = BinaryClassificationEvaluator(labelCol='Private_Index')
print("Test Area Under ROC: " + str(evaluator.evaluate(predictions, {evaluator.
metricName:"areaUnderROC"})))
```

```
Test Area Under ROC: 0.9404315196998123
```

In [40]:
```
print(gbt.explainParams())
```

```
cacheNodeIds: If false, the algorithm will pass trees to executors to match ins
tances with nodes. If true, the algorithm will cache node IDs for each instanc
e. Caching can speed up training of deeper trees. Users can set how often shoul
d the cache be checkpointed or disable it by setting checkpointInterval. (defau
lt: False)
checkpointInterval: set checkpoint interval (>= 1) or disable checkpoint (-1).
E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: t
his setting will be ignored if the checkpoint directory is not set in the Spark
Context. (default: 10)
featureSubsetStrategy: The number of features to consider for splits at each tr
ee node. Supported options: 'auto' (choose automatically for task: If numTrees
== 1, set to 'all'. If numTrees > 1 (forest), set to 'sqrt' for classification
and to 'onethird' for regression), 'all' (use all features), 'onethird' (use 1/
3 of the features), 'sqrt' (use sqrt(number of features)), 'log2' (use log2(num
ber of features)), 'n' (when n is in the range (0, 1.0], use n * number of feat
ures. When n is in the range (1, number of features), use n features). default
= 'auto' (default: all)
featuresCol: features column name. (default: features, current: features)
impurity: Criterion used for information gain calculation (case-insensitive). S
upported options: variance (default: variance)
labelCol: label column name. (default: label, current: Private_Index)
leafCol: Leaf indices column name. Predicted leaf index of each instance in eac
h tree by preorder. (default: )
lossType: Loss function which GBT tries to minimize (case-insensitive). Support
ed options: logistic (default: logistic)
maxBins: Max number of bins for discretizing continuous features.  Must be >=2
and >= number of categories for any categorical feature. (default: 32)
maxDepth: Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; de
pth 1 means 1 internal node + 2 leaf nodes. (default: 5)
maxIter: max number of iterations (>= 0). (default: 20, current: 10)
maxMemoryInMB: Maximum memory in MB allocated to histogram aggregation. If too
small, then 1 node will be split per iteration, and its aggregates may exceed t
his size. (default: 256)
minInfoGain: Minimum information gain for a split to be considered at a tree no
de. (default: 0.0)
minInstancesPerNode: Minimum number of instances each child must have after spl
it. If a split causes the left or right child to have fewer than minInstancesPe
rNode, the split will be discarded as invalid. Should be >= 1. (default: 1)
minWeightFractionPerNode: Minimum fraction of the weighted sample count that ea
ch child must have after split. If a split causes the fraction of the total wei
ght in the left or right child to be less than minWeightFractionPerNode, the sp
lit will be discarded as invalid. Should be in interval [0.0, 0.5). (default:
0.0)
predictionCol: prediction column name. (default: prediction)
probabilityCol: Column name for predicted class conditional probabilities. Not
e: Not all models output well-calibrated probability estimates! These probabili
ties should be treated as confidences, not precise probabilities. (default: pro
bability)
rawPredictionCol: raw prediction (a.k.a. confidence) column name. (default: raw
Prediction)
seed: random seed. (default: -5888732474670061308)
stepSize: Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the
contribution of each estimator. (default: 0.1)
subsamplingRate: Fraction of the training data used for learning each decision
tree, in range (0, 1]. (default: 1.0)
thresholds: Thresholds in multi-class classification to adjust the probability
of predicting each class. Array must have length equal to the number of classe
s, with values > 0, excepting that at most one value may be 0. The class with l
argest value p/t is predicted, where p is the original probability of that clas
s and t is the class's threshold. (undefined)
validationIndicatorCol: name of the column that indicates whether each row is f
or training or for validation. False indicates training; true indicates validat
```

```
ion. (undefined)
validationTol: Threshold for stopping early when fit with validation is used. I
f the error rate on the validation input changes by less than the validationTo
l, then learning will stop early (before `maxIter`). This parameter is ignored
when fit without validation is used. (default: 0.01)
weightCol: weight column name. If this is not set or empty, we treat all instan
```

In [41]:
```
# Parametric tunning and cross-validation of the number of trees with Estimator
paramGrid = (ParamGridBuilder()
           .addGrid(gbt.maxDepth, [2,4,6])
           .addGrid(gbt.maxBins, [10,30])
           .addGrid(gbt.maxIter,[5,10])
           .build())
cv=CrossValidator(estimator=gbt,
                  estimatorParamMaps=paramGrid,
                  evaluator=evaluator, numFolds=5)
```

In [42]:
```
# Run cross validation
cvModel = cv.fit(train_data)
predictions = cvModel.transform(test_data)
evaluator.evaluate(predictions)
```

Out[42]:
```
0.9674015009380864
```

Summary

The Logistic Regression model for our binary classification had a Training set area Under ROC of 0.98 and a Test area under ROC of 0.96, which is pretty close, which provides information that our model is performing very well.

Compared to the other Classifiers it looks like Logistic Regression performed better. Random Forest comes slightly closer behind. On the other hand after using Cross Validation to evaluate and tune our GBT Classifier the outcome was very close to what we have for Logistic Regression at around 0.97. GBT performance improved quite well compared to the base line outcome we observed at around 0.89 vs after tuning the model we got 0.97. This is also better than the Random Forest performance.

In [ ]:
```
```