Chapter 8: Tree Methods

Ex1: College

We will be using a college dataset (College.csv) to try to classify colleges as Private or Public based off these features:

```
Private: A factor with levels No and Yes indicating private or public un
iversity
Apps: Number of applications received
Accept: Number of applications accepted
Enroll: Number of new students enrolled
Top10perc: Pct. new students from top 10% of H.S. class
Top25perc: Pct. new students from top 25% of H.S. class
F.Undergrad: Number of fulltime undergraduates
P.Undergrad: Number of parttime undergraduates
Outstate: Out-of-state tuition
Room.Board: Room and board costs
Books: Estimated book costs
Personal: Estimated personal spending
PhD: Pct. of faculty with Ph.D.'s
Terminal: Pct. of faculty with terminal degree
S.F.Ratio: Student/faculty ratio
perc.alumni: Pct. alumni who donate
Expend: Instructional expenditure per student
Grad.Rate: Graduation rate
```

In [1]:
```
import findspark
findspark.init()
```

In [2]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('treecode').getOrCreate()
```

In [3]:
```
# Load training data
data = spark.read.csv('College.csv',inferSchema=True,header=True)
```

In [4]:
```
data.count()
```

Out[4]:
```
777
```

In [5]:
```
data.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 [6]:
```
data.head()
```

Out[6]:
```
Row(School='Abilene Christian University', Private='Yes', Apps=1660, Accept=123
2, Enroll=721, Top10perc=23, Top25perc=52, F_Undergrad=2885, P_Undergrad=537, O
utstate=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)
```

Spark Formatting of Data

In [7]:
```
# It needs to be in the form of two columns
# ("label","features")

# Import VectorAssembler and Vectors
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

In [8]:
```
data.columns
```

Out[8]:
```
['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 [9]:
```
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 [10]:
```
output = assembler.transform(data)
```

Deal with Private column being "yes" or "no"

In [11]:
```
from pyspark.ml.feature import StringIndexer
```

In [12]:
```
indexer = StringIndexer(inputCol="Private", outputCol="PrivateIndex")
output_fixed = indexer.fit(output).transform(output)
```

In [13]:
```
final_data = output_fixed.select("features",'PrivateIndex')
```

In [14]:
```
train_data,test_data = final_data.randomSplit([0.7,0.3])
```

The Classifiers

In [15]:
```
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import GBTClassifier,RandomForestClassifier
from pyspark.ml import Pipeline
```

Create all three models:

In [16]:
```
# Use mostly defaults to make this comparison "fair"

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

Train all three models:

In [17]:
```
# Train the models (its three models, so it might take some time)
dtc_model = dtc.fit(train_data)
rfc_model = rfc.fit(train_data)
gbt_model = gbt.fit(train_data)
```

Model Comparison

Let's compare each of these models!

In [18]:
```
dtc_predictions = dtc_model.transform(test_data)
rfc_predictions = rfc_model.transform(test_data)
gbt_predictions = gbt_model.transform(test_data)
```

Evaluation Metrics:

In [19]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

In [20]:
```
# Select (prediction, true label) and compute test error
acc_evaluator = MulticlassClassificationEvaluator(labelCol="PrivateIndex",
                                                  predictionCol="prediction",
                                                  metricName="accuracy")
```

In [21]:
```
dtc_acc = acc_evaluator.evaluate(dtc_predictions)
rfc_acc = acc_evaluator.evaluate(rfc_predictions)
gbt_acc = acc_evaluator.evaluate(gbt_predictions)
```

In [22]:
```
print("Results:")
print('-'*80)
print('A single decision tree - accuracy: {0:2.2f}%'.format(dtc_acc*100))
print('-'*80)
print('A random forest ensemble - accuracy: {0:2.2f}%'.format(rfc_acc*100))
print('-'*80)
print('A ensemble using GBT - accuracy: {0:2.2f}%'.format(gbt_acc*100))
```

```
Results:
--------------------------------------------------------------------------------
-
A single decision tree - accuracy: 92.51%
--------------------------------------------------------------------------------
-
A random forest ensemble - accuracy: 95.59%
--------------------------------------------------------------------------------
-
A ensemble using GBT - accuracy: 92.95%
```

Optional Assignment - play around with the parameters of each of these models, can you squeeze some more accuracy out of them? Or is the data the limiting factor?