Marcro Setting

In [ ]:
```python
from pyspark.sql import SparkSession
from pyspark.sql.types import IntegerType
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

In [ ]:
```python
spark = SparkSession.builder.appName("Adult Data Binary Classifier").getOrCreate()
```

1. Data loading

In [ ]:
```python
#Read csv file to dataframe
data = spark.read.csv("adult.csv")
data.show(5)
```

```
+---+-----------------+-------+----------+---+-------------------+------------------+--------------+------+-------+-----+----+----+--------------+-
-----+
|_c0|              _c1|    _c2|       _c3|_c4|                _c5|               _c6|           _c7|   _c8|    _c9| _c10|_c11|_c12|          _c13|
_c14|
+---+-----------------+-------+----------+---+-------------------+------------------+--------------+------+-------+-----+----+----+--------------+-
-----+
| 39|        State-gov|  77516| Bachelors| 13|      Never-married|      Adm-clerical| Not-in-family| White|   Male| 2174|   0|  40| United-States|
<=50K|
| 50| Self-emp-not-inc|  83311| Bachelors| 13| Married-civ-spouse|   Exec-managerial|       Husband| White|   Male|    0|   0|  13| United-States|
<=50K|
| 38|          Private| 215646|   HS-grad|  9|           Divorced| Handlers-cleaners| Not-in-family| White|   Male|    0|   0|  40| United-States|
<=50K|
| 53|          Private| 234721|      11th|  7| Married-civ-spouse| Handlers-cleaners|       Husband| Black|   Male|    0|   0|  40| United-States|
<=50K|
| 28|          Private| 338409| Bachelors| 13| Married-civ-spouse|    Prof-specialty|          Wife| Black| Female|    0|   0|  40|          Cuba|
<=50K|
+---+-----------------+-------+----------+---+-------------------+------------------+--------------+------+-------+-----+----+----+--------------+-
-----+
only showing top 5 rows
```

In [ ]:
```python
#change the column names of dataframe
df = data.withColumnRenamed('_c0','age').withColumnRenamed('_c1','workclass').withColumnRenamed('_c2','fnlwgt')\
.withColumnRenamed('_c3', 'education').withColumnRenamed('_c4', 'education_num')\
.withColumnRenamed('_c5','marital_status').withColumnRenamed('_c6', 'occupation').withColumnRenamed('_c7', 'relationship')\
.withColumnRenamed('_c8', 'race').withColumnRenamed('_c9', 'sex').withColumnRenamed('_c10', 'capital_gain')\
.withColumnRenamed('_c11', 'capital_loss').withColumnRenamed('_c12','hours_per_week')\
.withColumnRenamed('_c13', 'native_country').withColumnRenamed('_c14', 'income')

df.printSchema()
df.show(5)

dataset = df
```

```
root
 |-- age: string (nullable = true)
 |-- workclass: string (nullable = true)
 |-- fnlwgt: string (nullable = true)
 |-- education: string (nullable = true)
 |-- education_num: string (nullable = true)
 |-- marital_status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- capital_gain: string (nullable = true)
 |-- capital_loss: string (nullable = true)
 |-- hours_per_week: string (nullable = true)
 |-- native_country: string (nullable = true)
 |-- income: string (nullable = true)

+---+-----------------+-------+----------+-------------+-------------------+------------------+--------------+------+-------+------------+---------
---+--------------+--------------+------+
|age|        workclass| fnlwgt| education|education_num|     marital_status|        occupation|  relationship|  race|    sex|capital_gain|capital_l
oss|hours_per_week|native_country|income|
+---+-----------------+-------+----------+-------------+-------------------+------------------+--------------+------+-------+------------+---------
---+--------------+--------------+------+
| 39|        State-gov|  77516| Bachelors|           13|      Never-married|      Adm-clerical| Not-in-family| White|   Male|        2174|
0|            40| United-States| <=50K|
| 50| Self-emp-not-inc|  83311| Bachelors|           13| Married-civ-spouse|   Exec-managerial|       Husband| White|   Male|           0|
0|            13| United-States| <=50K|
| 38|          Private| 215646|   HS-grad|            9|           Divorced| Handlers-cleaners| Not-in-family| White|   Male|           0|
0|            40| United-States| <=50K|
| 53|          Private| 234721|      11th|            7| Married-civ-spouse| Handlers-cleaners|       Husband| Black|   Male|           0|
0|            40| United-States| <=50K|
| 28|          Private| 338409| Bachelors|           13| Married-civ-spouse|    Prof-specialty|          Wife| Black| Female|           0|
0|            40|          Cuba| <=50K|
+---+-----------------+-------+----------+-------------+-------------------+------------------+--------------+------+-------+------------+---------
---+--------------+--------------+------+
only showing top 5 rows
```

2. Data preprocessing

In [ ]:
```python
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
```

In [ ]:
```python
#stages in our Pipeline
stages = []
categoricalColumns = ["workclass", "education", "marital_status", "occupation", "relationship", "race", "sex", "native_country"]
```

In [ ]:
```python
for categoricalCol in categoricalColumns:
    # Category Indexing with StringIndexer
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol + "Index")
    # Use OneHotEncoder to convert categorical variables into binary SparseVectors
```

```
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
    # Add stages. These are not run here, but will run all at once later on.
    stages += [stringIndexer, encoder]
```

In [ ]:

```
# Convert label into label indices using the StringIndexer
label_stringIdx = StringIndexer(inputCol="income", outputCol="label")
stages += [label_stringIdx]
```

In [ ]:

```
# Convert values of numeric columns from string to integer
numericCols = ["age", "fnlwgt", "education_num", "capital_gain", "capital_loss", "hours_per_week"]
for nc in numericCols:
    dataset = dataset.withColumn(nc, df[nc].cast(IntegerType()))

# Transform all features into a vector using VectorAssembler
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
```

In [ ]:

```
pipeline = Pipeline(stages=stages)
pipelineModel = pipeline.fit(dataset)
preppedDataDF = pipelineModel.transform(dataset)
```

In [ ]:

```
# Keep relevant columns
cols = dataset.columns
selectedcols = ["label", "features"] + cols
dataset = preppedDataDF.select(selectedcols)
display(dataset)
```

```
DataFrame[label: double, features: vector, age: int, workclass: string, fnlwgt: int, education: string, education_num: int, marital_status: string,
occupation: string, relationship: string, race: string, sex: string, capital_gain: int, capital_loss: int, hours_per_week: int, native_country: str
ing, income: string]
```

In [ ]:

```
# Randomly split data into training and test sets. set seed for reproducibility
trainingData, testData = dataset.randomSplit([0.7, 0.3], seed=10)
trainingData.cache()

print(trainingData.count())
print(testData.count())
```

```
22685
9876
```

3. Modeling

In [ ]:

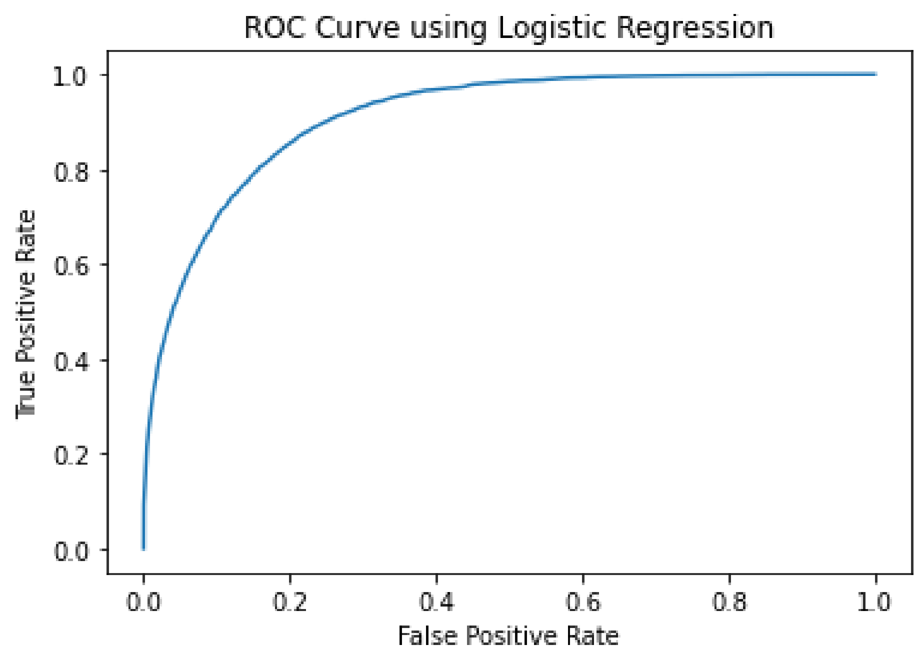
```
accuracy_dict = {}
```

In [ ]:

```
# LogisticRegression model, maxIter=10
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

lrModel = LogisticRegression(featuresCol="features", labelCol="label", maxIter=10).fit(trainingData)

# select example rows to display.
predictions = lrModel.transform(testData)
predictions.select(["label", "prediction"]).show(5)

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
accuracy_dict['Logistic'] = round(accuracy, 4)

# draw the ROC curve
trainingSummary = lrModel.summary
roc = trainingSummary.roc.toPandas()
plt.plot(roc['FPR'], roc['TPR'])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve using Logistic Regression')
plt.show()
print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))
```

```
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       1.0|
|  0.0|       1.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
+-----+----------+
only showing top 5 rows
```

```
Test set accuracy = 0.8466990684487646
```



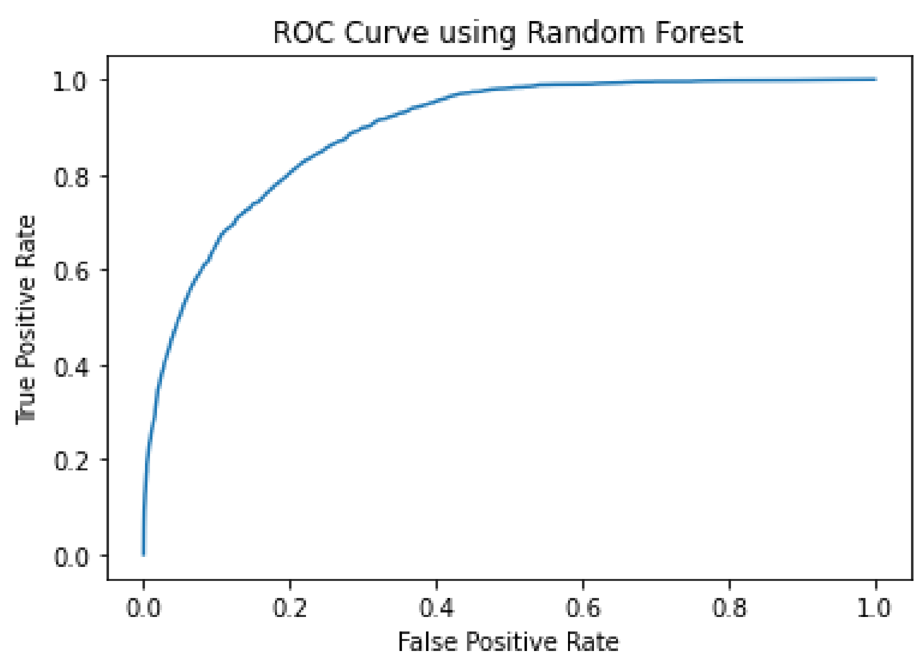

```
Training set areaUnderROC: 0.9107986021451718
```

In [ ]:

```python
# Random Forest
from pyspark.ml.classification import RandomForestClassifier

rfModel = RandomForestClassifier(featuresCol="features", labelCol="label").fit(trainingData)
predictions = rfModel.transform(testData)
predictions.select(["label", "prediction"]).show(5)

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
accuracy_dict['Random Forest'] = round(accuracy, 4)

# draw the ROC curve
trainingSummary = rfModel.summary
roc = trainingSummary.roc.toPandas()
plt.plot(roc['FPR'], roc['TPR'])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve using Random Forest')
plt.show()
print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))
```

```
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
+-----+----------+
only showing top 5 rows
```

```
Test set accuracy = 0.8146010530579182
```




```
Training set areaUnderROC: 0.893372713336966
```

In [ ]:

```python
# NaiveBayes
from pyspark.ml.classification import NaiveBayes

nbModel = NaiveBayes(featuresCol="features", labelCol="label").fit(trainingData)

# select example rows to display.
predictions = nbModel.transform(testData)
predictions.select(["label", "prediction"]).show(5)

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
accuracy_dict['NaiveBayes'] = round(accuracy, 4)
```

```
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       1.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
+-----+----------+
only showing top 5 rows

Test set accuracy = 0.7805791818550021
```

In [ ]:

```python
# Decision Tree
from pyspark.ml.classification import DecisionTreeClassifier
```

```python
dtree = DecisionTreeClassifier(featuresCol="features", labelCol="label").fit(trainingData)

# select example rows to display.
predictions = dtree.transform(testData)
predictions.select(["label", "prediction"]).show(5)

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
accuracy_dict['DecisionTree'] = round(accuracy, 4)
```

```
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
+-----+----------+
only showing top 5 rows

[Stage 4081:>                                                       (0 + 1) / 1]
Test set accuracy = 0.83424463345484
```

In [ ]:

```python
# Gradient Boosting Trees
from pyspark.ml.classification import GBTClassifier

gbt = GBTClassifier(featuresCol="features", labelCol="label").fit(trainingData)

# select example rows to display.
predictions = gbt.transform(testData)
predictions.select(["label", "prediction"]).show(5)

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
accuracy_dict['GBT'] = round(accuracy, 4)
```

```
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       1.0|
|  0.0|       1.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
+-----+----------+
only showing top 5 rows

Test set accuracy = 0.8522681247468611
```

In [ ]:

```python
# Multi-layer Perceptron
from pyspark.ml.classification import MultilayerPerceptronClassifier

layer_structure = [100, 40, 10, 2]
mlp = MultilayerPerceptronClassifier(featuresCol="features",
                                     labelCol="label",
                                     layers=layer_structure,
                                     seed=10).fit(trainingData)

# select example rows to display.
predictions = mlp.transform(testData)
predictions.show(5)

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
accuracy_dict['MLP'] = round(accuracy, 4)
```

```
+-----+--------------------+---+---------+------+---------+-------------+-------------------+---------------+------------+------+-----+------------
+------------+--------------+--------------+------+--------------------+--------------------+----------+
|label|            features|age|workclass|fnlwgt|education|education_num|     marital_status|     occupation|relationship|  race|  sex|capital_gain
|capital_loss|hours_per_week|native_country|income|       rawPrediction|         probability|prediction|
+-----+--------------------+---+---------+------+---------+-------------+-------------------+---------------+------------+------+-----+------------
+------------+--------------+--------------+------+--------------------+--------------------+----------+
|  0.0|(100,[0,8,23,29,4...| 60|  Private|160625|  HS-grad|            9| Married-civ-spouse| Prof-specialty|     Husband| White| Male|        5013
|           0|            40| United-States| <=50K|[-1.1700061180216...|[0.33006698208408...|       1.0|
|  0.0|(100,[0,8,23,29,4...| 36|  Private|370767|  HS-grad|            9| Married-civ-spouse| Prof-specialty|     Husband| White| Male|           0
|        2377|            60| United-States| <=50K|[-0.1467829521576...|[0.79174805701775...|       0.0|
|  0.0|(100,[0,8,23,29,4...| 29|  Private| 40295|  HS-grad|            9| Married-civ-spouse| Prof-specialty|     Husband| White| Male|           0
|           0|            40| United-States| <=50K|[-0.1467829521576...|[0.79174805701775...|       0.0|
|  0.0|(100,[0,8,23,29,4...| 30|  Private| 83253|  HS-grad|            9| Married-civ-spouse| Prof-specialty|     Husband| White| Male|           0
|           0|            55| United-States| <=50K|[-0.1467829521576...|[0.79174805701775...|       0.0|
|  0.0|(100,[0,8,23,29,4...| 31|  Private| 62374|  HS-grad|            9| Married-civ-spouse| Prof-specialty|     Husband| White| Male|           0
|           0|            50| United-States| <=50K|[-0.1467829521576...|[0.79174805701775...|       0.0|
+-----+--------------------+---+---------+------+---------+-------------+-------------------+---------------+------------+------+-----+------------
+------------+--------------+--------------+------+--------------------+--------------------+----------+
only showing top 5 rows

[Stage 4466:>                                                       (0 + 1) / 1]
Test set accuracy = 0.7918185500202511
```

In [ ]:

```python
# Linear Support Vector Machine
from pyspark.ml.classification import LinearSVC

lsvc = LinearSVC(featuresCol="features", labelCol="label").fit(trainingData)
```

```
# select example rows to display.
predictions = lsvc.transform(testData)
predictions.select(["label", "prediction"]).show(5)

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
accuracy_dict['LSVM'] = round(accuracy, 4)
```

```
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       1.0|
|  0.0|       1.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
+-----+----------+
only showing top 5 rows

[Stage 4701:>                                                    (0 + 1) / 1]
Test set accuracy = 0.8463953017415958
```

In [ ]:

```
# One-vs-Rest
from pyspark.ml.classification import OneVsRest

lr = LogisticRegression(featuresCol="features", labelCol="label", maxIter=10)
ovr = OneVsRest(classifier=lr).fit(trainingData)

# select example rows to display.
predictions = ovr.transform(testData)
predictions.select(["label", "prediction"]).show(5)

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
accuracy_dict['OVR'] = round(accuracy, 4)
```

```
+-----+----------+
|label|prediction|
+-----+----------+
|  0.0|       1.0|
|  0.0|       1.0|
|  0.0|       0.0|
|  0.0|       0.0|
|  0.0|       0.0|
+-----+----------+
only showing top 5 rows

[Stage 4731:>                                                    (0 + 1) / 1]
Test set accuracy = 0.8466990684487646
```

4. Comparison and analysis

In [ ]:

```
# Rank models according to Test set accuracy
import pandas as pd
accuracy_df = pd.DataFrame.from_dict(accuracy_dict, orient='index', columns=['Accuracy'])
accuracy_df.sort_values(by='Accuracy', ascending=False, inplace=True)
accuracy_df
```

Out[ ]:

	Accuracy
GBT	0.8523
Logistic	0.8467
OVR	0.8467
LSVM	0.8464
DecisionTree	0.8342
Random Forest	0.8146
MLP	0.7918
NaiveBayes	0.7806

Result Analysis

The gradient boost tree method has the highest accuracy among all the methods, which is around 85%. Being an ensemble method, GBT uses boosting to put more effort on weak learners. Compared to the random forest, which simply uses bagging and train individual tree separately, GBT is is able to sequentially combine the tree. This helps to model complex structures but also may lead to overfitting very easily.

Other tree methods like OVR, decision treea and random forest also give decent outcomes because the problem's structure is suitable for tree classfication. One-versus-rest performs better than vanilla decision tree and random forest because it . And unexpectedly, the decision tree is better than random forest, which might indicate the data is noisy (or simply under the current random seed setting, the RF happens to be worse, which usually is not the case)

For logistic regression and linear support vector machine, they are all suitable for binary prediction, and both of their performance are great according to the ranking. Intuitively speaking, SVM handles outliers better than logistic regression, but since we are predicting income based on education, occupation etc, the dataset should not have many outliers. These two methods thus have similar performance.

Surprisingly the Multi Layer Perception method performance is not great. Since MLP heavily relies on the structure of the network, adjusting the number of layers and number of neurons within each layer might help to improve the prediction accuracy。