BIG DATA

"Comparison between different model using Bank Data Set"

Project Report

Group Members:

M.Sheroz Raees 9852

Hassan Noor 9827

Kehkashan 9825

Date Set: Bank

Columns of our Data Set

```
root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- Target: string (nullable = true)
```

Data:

```
data.show(10)
```

```
+---+------------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+------+
|age|         job| marital|education|default|balance|housing|loan|contact|day|month|duration|campaign|pdays|previous|poutcome|Target|
+---+------------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+------+
| 58|  management| married| tertiary|     no|   2143|    yes|  no|unknown|  5|  may|     261|       1|   -1|       0| unknown|    no|
| 44|  technician|  single|secondary|     no|     29|    yes|  no|unknown|  5|  may|     151|       1|   -1|       0| unknown|    no|
| 33|entrepreneur| married|secondary|     no|      2|    yes| yes|unknown|  5|  may|      76|       1|   -1|       0| unknown|    no|
| 47| blue-collar| married|  unknown|     no|   1506|    yes|  no|unknown|  5|  may|      92|       1|   -1|       0| unknown|    no|
| 33|     unknown|  single|  unknown|     no|      1|     no|  no|unknown|  5|  may|     198|       1|   -1|       0| unknown|    no|
| 35|  management| married| tertiary|     no|    231|    yes|  no|unknown|  5|  may|     139|       1|   -1|       0| unknown|    no|
| 28|  management|  single| tertiary|     no|    447|    yes| yes|unknown|  5|  may|     217|       1|   -1|       0| unknown|    no|
| 42|entrepreneur|divorced| tertiary|    yes|      2|    yes|  no|unknown|  5|  may|     380|       1|   -1|       0| unknown|    no|
| 58|     retired| married|  primary|     no|    121|    yes|  no|unknown|  5|  may|      50|       1|   -1|       0| unknown|    no|
| 43|  technician|  single|secondary|     no|    593|    yes|  no|unknown|  5|  may|      55|       1|   -1|       0| unknown|    no|
+---+------------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+------+
only showing top 10 rows
```

Feature Engineering:

Checking if column contains null value or not a number and we will remove it

```
#check the null column count using sql
from pyspark.sql.functions import col,isnan, when, count
data.select([count(when(isnan(a) | col(a).isNull(), a)).alias(a) for a in data.columns]).show()
```

```
+---+---+-------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+------+
|age|job|marital|education|default|balance|housing|loan|contact|day|month|duration|campaign|pdays|previous|poutcome|Target|
+---+---+-------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+------+
|  0|  0|      0|        0|      0|      0|      0|   0|      0|  0|    0|       0|       0|    0|       0|       0|     0|
+---+---+-------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+------+
```

Targeted Values:

Our targeted values were 0 and 1 so we changed NO and Yes into 0s and 1s

df = df.replace('yes','1')
df = df.replace('no','0')

Type Casting:

```
df = df.withColumn("default",df.default.cast('integer'))
df = df.withColumn("loan",df.loan.cast('integer'))
df = df.withColumn("housing",df.housing.cast('integer'))
df = df.withColumn("Target",df.Target.cast('integer'))
```

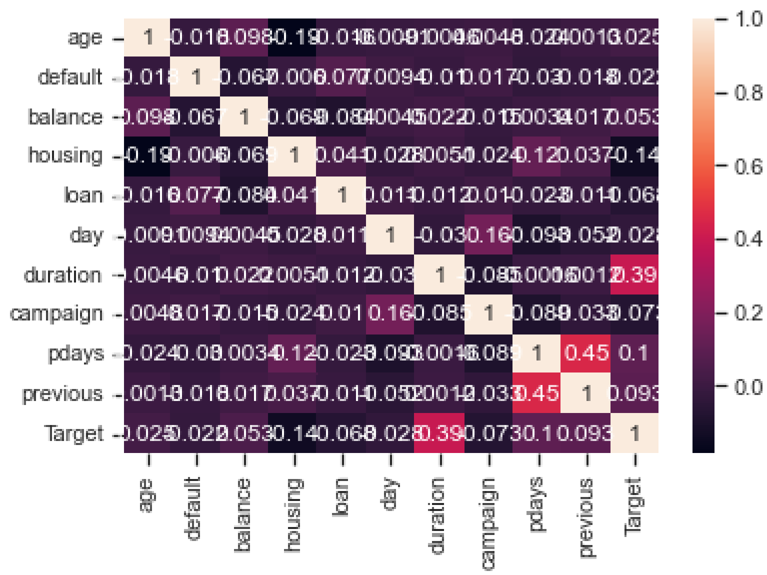
```
root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: integer (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: integer (nullable = true)
 |-- loan: integer (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- Target: integer (nullable = true)
```

Heat Map:

```
sns.heatmap(df.toPandas().corr(), annot = True)
```

```
c:\users\sheroz\appdata\local\programs\python\python36\lib\so
type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 496
  self._sock = None

<AxesSubplot:>
```



Encoding:

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
categoricalColumns = ['job', 'marital', 'education', 'contact', 'month', 'poutcome']
stages = []
for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol = categoricalCol + 'Index')
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
    stages += [stringIndexer, encoder]
label_stringIdx = StringIndexer(inputCol = 'Target', outputCol = 'label')
stages += [label_stringIdx]
numericCols = ['age', 'default', 'balance', 'housing', 'loan', 'day', 'duration', 'campaign', 'pdays', 'previous']
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="Subscribed")
stages += [assembler]
```

encoding of categorical columns (string)

string indexer:

inputCol = categoricalCol

outputCol = categoricalCol + Index

Encoder using OneHotEncoder library:

inputCol = [stringIndexer.getOutputCol()]

outputCol = [categoricalCol + classVec]

now

adding them into **stage**

stages += [stringIndexer, encoder]

label index:

label_stringIdx = StringIndexer(inputCol = 'Target', outputCol = 'label')

using **libarary StringIndexer**

inputCol = 'Target'

outputCol = 'label'

adding it into **stage**

stages += [label_stringIdx]

VectorAssembler:

Create **numericCols** and adding it into **assemblerInput**

assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols

giving input column and output column to vector assembler and saving it into assembler

assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="Subscribed")

inputCols= assemblerInputs

outputCol="Subscribed"

adding assembler into **stages**

stages += [assembler]

Pipeline

```
from pyspark.ml import Pipeline
pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(df)
df = pipelineModel.transform(df)
selectedCols = ['label', 'Subscribed'] + cols
df = df.select(selectedCols)
df.printSchema()
```

Connecting stages through pipeline

Stages= stringIndexer, encoder, label_stringIdx, assembler

Actual schema after pipeline:

```
root
 |-- label: double (nullable = false)
 |-- Subscribed: vector (nullable = true)
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: integer (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: integer (nullable = true)
 |-- loan: integer (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- Target: integer (nullable = true)
```

Taking Transpose

```
pd.DataFrame(df.take(5), columns=df.columns).transpose()
```

```
c:\users\sheroz\appdata\local\programs\python\python36\lib\socket.py:657: ResourceWarning: unclosed <socket.socket fd=5020, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM
proto=0, laddr=('127.0.0.1', 49684), raddr=('127.0.0.1', 49683)>
  self._sock = None
```

	0	1	2	3	4
label	0	0	0	0	0
Subscribed	(0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	(0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, ...	(1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
age	58	44	33	47	33
job	management	technician	entrepreneur	blue-collar	unknown
marital	married	single	married	married	single
education	tertiary	secondary	secondary	unknown	unknown
default	0	0	0	0	0
balance	2143	29	2	1506	1
housing	1	1	1	1	0
loan	0	0	1	0	0
contact	unknown	unknown	unknown	unknown	unknown
day	5	5	5	5	5
month	may	may	may	may	may
duration	261	151	76	92	198
campaign	1	1	1	1	1
pdays	-1	-1	-1	-1	-1
previous	0	0	0	0	0
poutcome	unknown	unknown	unknown	unknown	unknown
Target	0	0	0	0	0

Training and testing data:

We use seed to split data randomly

80 percent data for train

20 percent data for test

```
train, test = df.randomSplit([0.8, 0.2], seed = 2022)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))
```

```
Training Dataset Count: 36061
Test Dataset Count: 9150
```

LogisticRegression

featuresCol = 'Subscribed'
labelCol = 'label'
max Iteration = 15

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol = 'Subscribed', labelCol = 'label', maxIter=15)
lrModel = lr.fit(train)
```

Comparing label and prediction data

Target value= label

```
predictions_LogisticRegression.groupBy('label', 'prediction').count().show()
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  0.0|       1.0|  190|
|  1.0|       0.0|  709|
|  0.0|       0.0| 7868|
|  1.0|       1.0|  383|
+-----+----------+-----+
```

Calculate accuracy, precision and recall:

True negative, True Positive, False negative, False Positive

```
# calculate the elements of the confusion matrix
TN = predictions_LogisticRegression.filter('prediction = 0 AND label = prediction').count()
TP = predictions_LogisticRegression.filter('prediction = 1 AND label = prediction').count()
FN = predictions_LogisticRegression.filter('prediction = 0 AND label <> prediction').count()
FP = predictions_LogisticRegression.filter('prediction = 1 AND label <> prediction').count()
```

```
accuracy_LogisticRegression = (TN + TP) / (TN + TP + FN +FP)
precision_LogisticRegression = TP / (TP + FP)
recall_LogisticRegression = TP/ (TP + FN)

print('n accuracy: %0.3f' % accuracy_LogisticRegression)
print('n precision: %0.3f' % precision_LogisticRegression)
print('n recall: %0.3f' % recall_LogisticRegression)
```

```
n accuracy: 0.902
n precision: 0.668
n recall: 0.351
```

```
y_true = predictions_LogisticRegression.select('label').collect()
y_pred = predictions_LogisticRegression.select('prediction').collect()
```

Confusion Matrix:

```
from sklearn import metrics
```

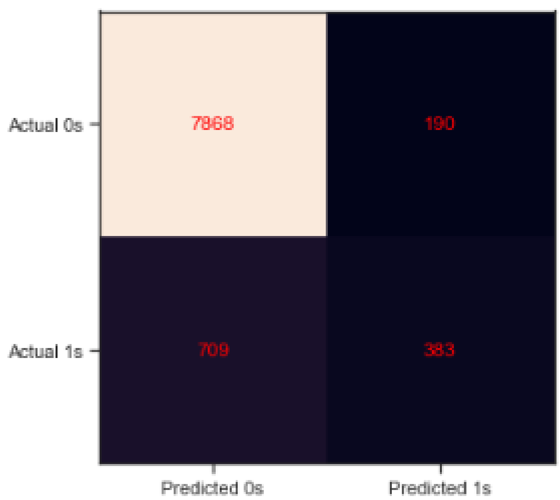
```
cm = metrics.confusion_matrix(y_true, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[7868  190]
 [ 709  383]]
```

Ploting confusion matrix

```
import matplotlib.pyplot as plt
```

```
fig , ax = plt.subplots(figsize=(5,5))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0,1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0,1), ticklabels=('Actual 0s', 'Actual 1s'))
for i in range(2):
    for j in range(2):
        ax.text(j,i,cm[i,j], ha='center', va='center', color='red')
plt.show()
```



DecisionTree

```
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier(featuresCol = 'Subscribed', labelCol = 'label', maxDepth= 3)
dtModel = dt.fit(train)
predictions_DecisionTree = dtModel.transform(test)
```

featuresCol = 'Subscribed'
labelCol = 'label'
maxDepth= 3

<u>Comparing label and prediction data</u>

Target value= label

```
predictions_DecisionTree.groupBy('label', 'prediction').count().show()
```

78] ✓ 0.7s

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  1.0|       1.0|  165|
|  0.0|       1.0|  114|
|  1.0|       0.0|  927|
|  0.0|       0.0| 7944|
+-----+----------+-----+
```

Calculate accuracy, precision and recall:

True negative, True Positive, False negative, False Positive

```
# calculate the elements of the confusion matrix
TN = predictions_DecisionTree.filter('prediction = 0 AND label = prediction').count()
TP = predictions_DecisionTree.filter('prediction = 1 AND label = prediction').count()
FN = predictions_DecisionTree.filter('prediction = 0 AND label <> prediction').count()
FP = predictions_DecisionTree.filter('prediction = 1 AND label <> prediction').count()
```

```
accuracy_DecisionTree = (TN + TP) / (TN + TP + FN +FP)
precision_DecisionTree = TP / (TP + FP)
recall_DecisionTree = TP/ (TP + FN)

print('n accuracy: %0.3f' % accuracy_DecisionTree)
print('n precision: %0.3f' % precision_DecisionTree)
print('n recall: %0.3f' % recall_DecisionTree)
```

```
n accuracy: 0.887
n precision: 0.603
n recall: 0.166
```

```
y_true = predictions_DecisionTree.select('label').collect()
y_pred = predictions_DecisionTree.select('prediction').collect()
```

Confusion Matrix:

```
from sklearn import metrics
```

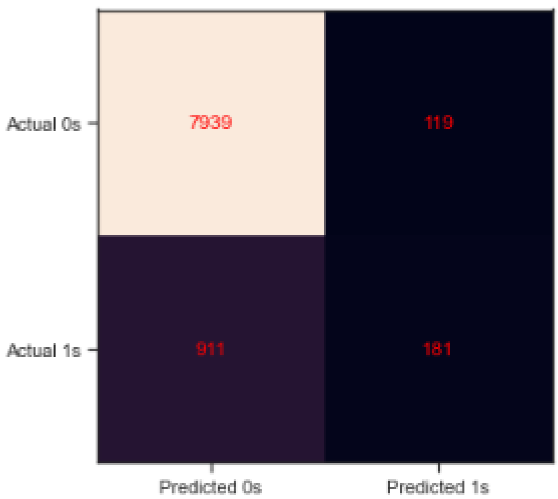
```
cm = metrics.confusion_matrix(y_true, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[7939  119]
 [ 911  181]]
```

Ploting confusion matrix

```
import matplotlib.pyplot as plt
```

```
fig , ax = plt.subplots(figsize=(5,5))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0,1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0,1), ticklabels=('Actual 0s', 'Actual 1s'))
for i in range(2):
    for j in range(2):
        ax.text(j,i,cm[i,j], ha='center', va='center', color='red')
plt.show()
```



Gradient boosting:

```
from pyspark.ml.classification import GBTClassifier
gbt = GBTClassifier(featuresCol = 'Subscribed', labelCol = 'label', maxIter=10)
gbtModel = gbt.fit(train)
predictions_GBT = gbtModel.transform(test)
```

featuresCol = 'Subscribed'
labelCol = 'label'
maxIter=10

Comparing label and prediction data

Target value= label

```
predictions_GBT.groupBy('label', 'prediction').count().show()
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  0.0|       1.0|  212|
|  1.0|       0.0|  674|
|  0.0|       0.0| 7846|
|  1.0|       1.0|  418|
+-----+----------+-----+
```

Calculate accuracy, precision and recall:

True negative, True Positive, False negative, False Positive

```
# calculate the elements of the confusion matrix
TN = predictions_GBT.filter('prediction = 0 AND label = prediction').count()
TP = predictions_GBT.filter('prediction = 1 AND label = prediction').count()
FN = predictions_GBT.filter('prediction = 0 AND label <> prediction').count()
FP = predictions_GBT.filter('prediction = 1 AND label <> prediction').count()
```

```
accuracy_GBT = (TN + TP) / (TN + TP + FN +FP)
precision_GBT = TP / (TP + FP)
recall_GBT = TP/ (TP + FN)

print('n accuracy: %0.3f' % accuracy_GBT)
print('n precision: %0.3f' % precision_GBT)
print('n recall: %0.3f' % recall_GBT)
```

```
n accuracy: 0.903
n precision: 0.663
n recall: 0.383
```

```
y_true = predictions_GBT.select('label').collect()
y_pred = predictions_GBT.select('prediction').collect()
```

Confusion Matrix:

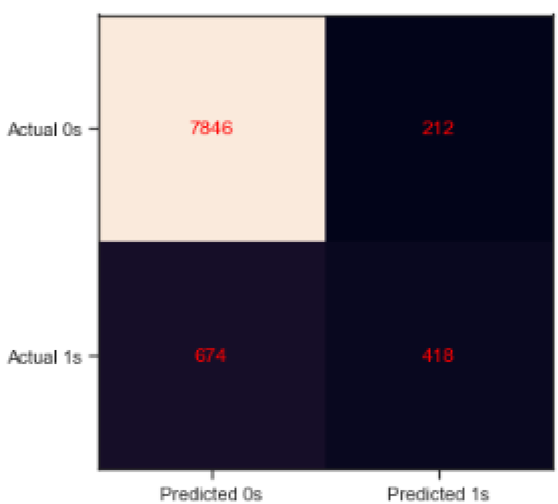
```
from sklearn import metrics
```

```
cm = metrics.confusion_matrix(y_true, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[7846  212]
 [ 674  418]]
```

Ploting confusion matrix

```
fig , ax = plt.subplots(figsize=(5,5))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0,1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0,1), ticklabels=('Actual 0s', 'Actual 1s'))
for i in range(2):
    for j in range(2):
        ax.text(j,i,cm[i,j], ha='center', va='center', color='red')
plt.show()
```



Comparison:

```
print("LogisticRegression")
print('n accuracy: %0.3f' % accuracy_LogisticRegression)
print('n precision: %0.3f' % precision_LogisticRegression)
print('n recall: %0.3f' % recall_LogisticRegression)
print('-----------------------------------')
print("DecisionTree")
print('n accuracy: %0.3f' % accuracy_DecisionTree)
print('n precision: %0.3f' % precision_DecisionTree)
print('n recall: %0.3f' % recall_DecisionTree)
print('-----------------------------------')
print("GBT")
print('n accuracy: %0.3f' % accuracy_GBT)
print('n precision: %0.3f' % precision_GBT)
print('n recall: %0.3f' % recall_GBT)
```

```
LogisticRegression
n accuracy: 0.902
n precision: 0.668
n recall: 0.351
-----------------------------------
DecisionTree
n accuracy: 0.887
n precision: 0.603
n recall: 0.166
-----------------------------------
GBT
n accuracy: 0.903
n precision: 0.663
n recall: 0.383
```