Phase 2 - Machine Learning with Apache Spark

Build - 3 new ML models using Apache Spark framework

- Logistic Regression
- Random Forest and
- GBM

Recap

Info about model evaluation - accuracy metric vs recall

- The global metric accuracy will be used to evaluate the models between all frameworks (xgb, lgbm, sklearn, h2o.ai and Apache Spark)

The last notebook build ml models using python will provide some additional techniques, such as:

- Unbalanced classification and class weight
- Smote technique for oversampling the training dataset
- Standard Scale vs. default data and
- Finally, exchange the global metric accuracy and use recall metric < recall or Sensitivity or True positive rate (TPR) >
  Recall metric is a better metric than accuracy to evaluate this type of scenario (customer churn)

Additional info: [https://spark.apache.org/ (https://spark.apache.org/)](https://spark.apache.org/)

Starting process...

In [1]:

```
## Spark Session
from pyspark.sql import SparkSession

## Data Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline

# ML Models
from  pyspark.ml.classification import LogisticRegression, RandomForestClassifier, GBTC
lassifier

## Metrics
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

Create Spark Session and load data into Spark

In [2]:

```
spark = SparkSession.builder.appName('Customer_support_ML').getOrCreate()
```

In [3]:

```
!ls data/WA_Fn-UseC_-Telco-Customer-Churn.csv
```

```
data/WA_Fn-UseC_-Telco-Customer-Churn.csv
```

Load data into Spark

In [4]:

```
## Load csv into Spark
path = 'data/WA_Fn-UseC_-Telco-Customer-Churn.csv'
dataset = spark.read.csv(path, header=True, inferSchema=True)

## Show 1 record sample
dataset.take(1)
```

Out[4]:

```
[Row(customerID='7590-VHVEG', gender='Female', SeniorCitizen=0, Partner='Y
es', Dependents='No', tenure=1, PhoneService='No', MultipleLines='No phone
service', InternetService='DSL', OnlineSecurity='No', OnlineBackup='Yes',
DeviceProtection='No', TechSupport='No', StreamingTV='No', StreamingMovies
='No', Contract='Month-to-month', PaperlessBilling='Yes', PaymentMethod='E
lectronic check', MonthlyCharges=29.85, TotalCharges='29.85', Churn='No')]
```

Data prep to run the ML models with Apache Spark

In [5]:

```
## Filter columns, fill values and convert columns for correct data type
dataset.createOrReplaceTempView('v_data')

df_spark = spark.sql("""
SELECT
    tenure,
    MonthlyCharges as monthly_charges,
    CAST ((case when TotalCharges == ' ' then 0 else TotalCharges end) as float) as tot
al_charges,
    gender,
        case when PaymentMethod == 'Electronic check' then 'ElectronicCheck'
                when PaymentMethod == 'Mailed check' then 'MailedCheck'
                when PaymentMethod == 'Bank transfer (automatic)' then 'BankTransferAut
omatic'
                when PaymentMethod == 'Credit card (automatic)' then 'CreditCardAutomat
ic'
                else 'Not_mapped'
        end as payment_method,
        case when Contract == 'Month-to-month' then 'MonthToMonth'
                 when Contract == 'One year' then 'OneYear'
                 when Contract == 'Two year' then 'TwoYear'
                else 'NotMapped'
        end as contract,
    CAST ((case when Churn == 'Yes' then 1 else 0 end) as integer) as churn
FROM v_data
""")

df_spark.printSchema()
```

```
root
 |-- tenure: integer (nullable = true)
 |-- monthly_charges: double (nullable = true)
 |-- total_charges: float (nullable = true)
 |-- gender: string (nullable = true)
 |-- payment_method: string (nullable = false)
 |-- contract: string (nullable = false)
 |-- churn: integer (nullable = false)
```

Read some data from Spark Dataframe

In [6]:

```
df_spark.take(3)
```

Out[6]:

```
[Row(tenure=1, monthly_charges=29.85, total_charges=29.850000381469727, ge
nder='Female', payment_method='ElectronicCheck', contract='MonthToMonth',
churn=0),
 Row(tenure=34, monthly_charges=56.95, total_charges=1889.5, gender='Mal
e', payment_method='MailedCheck', contract='OneYear', churn=0),
 Row(tenure=2, monthly_charges=53.85, total_charges=108.1500015258789, gen
der='Male', payment_method='MailedCheck', contract='MonthToMonth', churn=
1)]
```

Evaluate the distribution and labels applied in the Data Prep above

In [7]:

```
df_spark.groupby('gender').count().show()
```

```
+------+-----+
|gender|count|
+------+-----+
|Female| 3488|
|  Male| 3555|
+------+-----+
```

In [8]:

```
df_spark.groupby('contract').count().show()
```

```
+------------+-----+
|    contract|count|
+------------+-----+
|     OneYear| 1473|
|MonthToMonth| 3875|
|     TwoYear| 1695|
+------------+-----+
```

In [9]:

```
df_spark.groupby('payment_method').count().show()
```

```
+--------------------+-----+
|      payment_method|count|
+--------------------+-----+
|     ElectronicCheck| 2365|
|         MailedCheck| 1612|
| CreditCardAutomatic| 1522|
|BankTransferAutom...| 1544|
+--------------------+-----+
```

Machine Learning models build and data pipeline

- data pipeline and one hot encode for categorial features
- 3 ML models creation and evaluation

In [10]:

```
## set seed to reproduce similar results between executions
SEED = 12345

## ML models
lm_model = LogisticRegression(featuresCol='features',labelCol='churn')
rf_model = RandomForestClassifier(featuresCol='features', labelCol='churn', numTrees=10
0, seed=SEED)
gbm_model = GBTClassifier(featuresCol='features', labelCol='churn', seed=SEED)

## split data into train and test for prediction
train , test = df_spark.randomSplit([0.75, 0.25], seed=SEED)
```

Data pipeline

In [11]:

```
## SELECT COLUMNs
# target = 'Churn'
# current_features = ['tenure', 'monthly_charges', 'total_charges', 'gender', 'payment_
method' , 'churn', 'contract']

gender_idx = StringIndexer(inputCol='gender',outputCol='gender_idx')
gender_vec = OneHotEncoder(inputCol='gender_idx', outputCol='gender_vec')

contract_idx = StringIndexer(inputCol='contract',outputCol='contract_idx')
contract_vec = OneHotEncoder(inputCol='contract_idx', outputCol='contract_vec')

payment_method_idx = StringIndexer(inputCol='payment_method', outputCol='payment_method
_idx')
payment_method_vec = OneHotEncoder(inputCol='payment_method_idx', outputCol='payment_me
thod_vec')

assembler = VectorAssembler(inputCols=['tenure', 'monthly_charges', 'total_charges', 'g
ender_vec',
                                       'payment_method_vec' , 'contract_vec'
                                       ],
                            outputCol='features')
```

ML - Logistic Regression

- Accuracy: 79,32%

In [12]:

```
## data pipeline
pipeline_lm = Pipeline(stages=[gender_idx, gender_vec,
                               payment_method_idx, payment_method_vec,
                               contract_idx, contract_vec,
                               assembler, lm_model])

## Model Creation and prediction
fit_model = pipeline_lm.fit(train)
predict_lm = fit_model.transform(test)

# print(type(predict_lm))
# predict_lm.select('tenure', 'monthly_charges', 'total_charges', 'gender', 'payment_me
thod' , 'churn', 'contract',
#                    'probability', 'prediction').show()

## Model evaluation
eval_acc = MulticlassClassificationEvaluator(predictionCol='prediction',labelCol='chur
n', metricName='accuracy')
log_acc = eval_acc.evaluate(predict_lm)

# type(log_acc)
print('Logistic Regression - accuracy: {}'.format(log_acc))
```

```
Logistic Regression - accuracy: 0.7932960893854749
```

ML - Random Forest

- Accuracy: 79,72%

In [13]:

```
## data pipeline
pipeline_rf = Pipeline(stages=[gender_idx, gender_vec,
                               payment_method_idx, payment_method_vec,
                               contract_idx, contract_vec,
                               assembler, rf_model])

## Model Creation and prediction
fit_model = pipeline_rf.fit(train)
predict_rf = fit_model.transform(test)
# print(type(predict_rf))
# predict_rf.select('tenure', 'monthly_charges', 'total_charges', 'gender', 'payment_me
thod' , 'churn', 'contract',
#                    'probability', 'prediction').show()

## Metrics - accuracy evaluation
eval_acc = MulticlassClassificationEvaluator(predictionCol='prediction',labelCol='chur
n', metricName='accuracy')
log_acc = eval_acc.evaluate(predict_rf)

# type(log_acc)
print('Random Forest - accuracy: {}'.format(log_acc))
```

```
Random Forest - accuracy: 0.7972067039106145
```

ML - GBM

- Accuracy: 79,83%

In [14]:

```
## data pipeline
pipeline_gbm = Pipeline(stages=[gender_idx, gender_vec,
                               payment_method_idx, payment_method_vec,
                               contract_idx, contract_vec,
                               assembler, gbm_model])

## Model Creation and prediction
fit_model = pipeline_gbm.fit(train)
predict_gbm = fit_model.transform(test)

# print(type(predict_gbm))
# predict_gbm.select('tenure', 'monthly_charges', 'total_charges', 'gender', 'payment_m
ethod' , 'churn', 'contract',
#                    'probability', 'prediction').show()

## Model evaluation
eval_acc = MulticlassClassificationEvaluator(predictionCol='prediction',labelCol='chur
n', metricName='accuracy')
log_acc = eval_acc.evaluate(predict_gbm)

# type(log_acc)
print('GBM - accuracy: {}'.format(log_acc))
```

```
GBM - accuracy: 0.7983240223463687
```

Print confusion matrix for GBM

- All 3 models have similar accuracy score, however GBM has the bigger one with small higher accuracy

In [15]:

```
## Metrics - Classification
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import numpy as np

## Function to print Confusion Matrix and metrics
def print_confusion_matrix(y_true, y_pred):
    """Print metrics"""
    report = classification_report(y_true, y_pred)
    confusion_matrix_rpt = confusion_matrix(y_true, y_pred)
    accuracy_score_rpt = accuracy_score(y_true, y_pred)

    print('--   Confusion Matrix')
    print(confusion_matrix_rpt)
    print('')
    print('--   Accuracy')
    print(accuracy_score_rpt)
    print('')
    print('--   Metrics report')
    print(report)

## Get values to generate the confusion matrix
y_true = np.array(predict_gbm.select('churn').collect())
y_pred = np.array(predict_gbm.select('prediction').collect())

print('')
print('GBM results...')
print('')
print_confusion_matrix(y_true, y_pred)
```

```
GBM results...

--   Confusion Matrix
[[1184  112]
 [ 249  245]]

--   Accuracy
0.7983240223463687

--   Metrics report
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1296
           1       0.69      0.50      0.58       494

    accuracy                           0.80      1790
   macro avg       0.76      0.70      0.72      1790
weighted avg       0.79      0.80      0.79      1790
```

Summary - Apache Spark

- These 3 models above provide almost identical results, and GBM provide the best accuracy with 79,83%

This notebook along with others using Python show the creation of various Machine Learning models

- Frameworks: Sklearn, H2O.ai, Apache Spark, LightGBM and XGBoost
- Models: GLM-Generalized Linear Model, Logistic Regression, ..., Random Forest, GBM and Xgboost

Info

- Until now the GBM build with sklearn provide better results - 80% of accuracy

Let's move on...

In [1]:

```
!jupyter nbconvert --to html Phase_2_Build_ML_models_with_Python_4x6_Apache_Spark.ipynb
```

In [ ]:

```

```