Task 10 - Random Forest Classification

In this task, we build an end to end pipeline that reads in data in parquet format, converts it to

CSV and loads it into a dataframe, trains a model and perform hyperparameter tuning.

1.Spark initialization

In [1]:
```
%%bash
export version=`python --version |awk '{print $2}' |awk -F"." '{print $1$2}'`

echo $version

if [ $version == '36' ] || [ $version == '37' ]; then
    echo 'Starting installation...'
    pip3 install pyspark==2.4.8 wget==3.2 pyspark2pmml==0.5.1 > install.log 2> install.log
    if [ $? == 0 ]; then
        echo 'Please <<RESTART YOUR KERNEL>> (Kernel->Restart Kernel and Clear All Outputs)'
    else
        echo 'Installation failed, please check log:'
        cat install.log
    fi
elif [ $version == '38' ] || [ $version == '39' ]; then
    pip3 install pyspark==3.1.2 wget==3.2 pyspark2pmml==0.5.1 > install.log 2> install.log
    if [ $? == 0 ]; then
        echo 'Please <<RESTART YOUR KERNEL>> (Kernel->Restart Kernel and Clear All Outputs)'
    else
        echo 'Installation failed, please check log:'
        cat install.log
    fi
else
    echo 'Currently only python 3.6, 3.7 , 3.8 and 3.9 are supported, in case you need a different
    exit -1
fi
```

```
37
Starting installation...
Please <<RESTART YOUR KERNEL>> (Kernel->Restart Kernel and Clear All Outputs)
```

In [1]:
```
from pyspark import SparkContext, SparkConf, SQLContext
from pyspark.sql import SparkSession
import os
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark2pmml import PMMLBuilder
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import MinMaxScaler
import logging
import shutil
import site
import sys
import wget
import re
import pandas as pd
```

In [2]:
```
if sys.version[0:3] == '3.9':
    url = ('https://github.com/jpmml/jpmml-sparkml/releases/download/1.7.2/'
           'jpmml-sparkml-executable-1.7.2.jar')
    wget.download(url)
    shutil.copy('jpmml-sparkml-executable-1.7.2.jar',
```

```
                site.getsitepackages()[0] + '/pyspark/jars/')
elif sys.version[0:3] == '3.8':
    url = ('https://github.com/jpmml/jpmml-sparkml/releases/download/1.7.2/'
           'jpmml-sparkml-executable-1.7.2.jar')
    wget.download(url)
    shutil.copy('jpmml-sparkml-executable-1.7.2.jar',
                site.getsitepackages()[0] + '/pyspark/jars/')
elif sys.version[0:3] == '3.7':
    url = ('https://github.com/jpmml/jpmml-sparkml/releases/download/1.5.12/'
           'jpmml-sparkml-executable-1.5.12.jar')
    wget.download(url)
elif sys.version[0:3] == '3.6':
    url = ('https://github.com/jpmml/jpmml-sparkml/releases/download/1.5.12/'
           'jpmml-sparkml-executable-1.5.12.jar')
    wget.download(url)
else:
    raise Exception('Currently only python 3.6 , 3.7, 3,8 and 3.9 is supported, in case '
                    'you need a different version please open an issue at '
                    'https://github.com/IBM/claimed/issues')
```

In [3]:
```
# Creating a spark context class
sc = SparkContext()

# Creating a spark session
spark = SparkSession \
        .builder \
        .appName("Python Spark Random Forest Classification") \
        .getOrCreate()
        # .config("spark.some.config.option", "some-value") \
```

```
23/08/04 21:09:31 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platfor
m... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
```

Task 10.1 : Reading the parquet file we created as part of Task 3

In [10]:
```
data_parquet = 'data.parquet'
data_csv = 'randomforest.csv'
data_dir = './component-library/data/'
df = spark.read.parquet(data_dir + data_parquet)
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
~/conda/envs/python/lib/python3.7/site-packages/pyspark/sql/utils.py in deco(*a, **kw)
     62         try:
---> 63             return f(*a, **kw)
     64         except py4j.protocol.Py4JJavaError as e:

~/conda/envs/python/lib/python3.7/site-packages/py4j/protocol.py in get_return_value(answer, gateway
_client, target_id, name)
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:

Py4JJavaError: An error occurred while calling o196.parquet.
: org.apache.spark.sql.AnalysisException: Path does not exist: file:/resources/labs/BD0231EN/compone
nt-library/component-library/deploy/component-library/data/data.parquet;
        at org.apache.spark.sql.execution.datasources.DataSource$$anonfun$org$apache$spark$sql$execu
tion$datasources$DataSource$$checkAndGlobPathIfNecessary$1.apply(DataSource.scala:558)
        at org.apache.spark.sql.execution.datasources.DataSource$$anonfun$org$apache$spark$sql$execu
tion$datasources$DataSource$$checkAndGlobPathIfNecessary$1.apply(DataSource.scala:545)
        at scala.collection.TraversableLike$$anonfun$flatMap$1.apply(TraversableLike.scala:241)
        at scala.collection.TraversableLike$$anonfun$flatMap$1.apply(TraversableLike.scala:241)
        at scala.collection.immutable.List.foreach(List.scala:392)
        at scala.collection.TraversableLike$class.flatMap(TraversableLike.scala:241)
        at scala.collection.immutable.List.flatMap(List.scala:355)
        at org.apache.spark.sql.execution.datasources.DataSource.org$apache$spark$sql$execution$data
sources$DataSource$$checkAndGlobPathIfNecessary(DataSource.scala:545)
        at org.apache.spark.sql.execution.datasources.DataSource.resolveRelation(DataSource.scala:35
9)
        at org.apache.spark.sql.DataFrameReader.loadV1Source(DataFrameReader.scala:223)
        at org.apache.spark.sql.DataFrameReader.load(DataFrameReader.scala:211)
        at org.apache.spark.sql.DataFrameReader.parquet(DataFrameReader.scala:641)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
        at py4j.Gateway.invoke(Gateway.java:282)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.GatewayConnection.run(GatewayConnection.java:238)
        at java.lang.Thread.run(Thread.java:745)


During handling of the above exception, another exception occurred:

AnalysisException                         Traceback (most recent call last)
/tmp/ipykernel_1003/1997914230.py in <module>
      2 data_csv = 'randomforest.csv'
      3 data_dir = './component-library/data/'
----> 4 df = spark.read.parquet(data_dir + data_parquet)

~/conda/envs/python/lib/python3.7/site-packages/pyspark/sql/readwriter.py in parquet(self, *paths)
    314         [('name', 'string'), ('year', 'int'), ('month', 'int'), ('day', 'int')]
    315         """
--> 316         return self._df(self._jreader.parquet(_to_seq(self._spark._sc, paths)))
    317
    318     @ignore_unicode_prefix

~/conda/envs/python/lib/python3.7/site-packages/py4j/java_gateway.py in __call__(self, *args)
   1255         answer = self.gateway_client.send_command(command)
   1256         return_value = get_return_value(
-> 1257             answer, self.gateway_client, self.target_id, self.name)
   1258
   1259         for temp_arg in temp_args:

~/conda/envs/python/lib/python3.7/site-packages/pyspark/sql/utils.py in deco(*a, **kw)
     67                                             e.java_exception.getStackTrace()))
     68             if s.startswith('org.apache.spark.sql.AnalysisException: '):
```

```
---> 69                 raise AnalysisException(s.split(': ', 1)[1], stackTrace)
     70             if s.startswith('org.apache.spark.sql.catalyst.analysis'):
     71                 raise AnalysisException(s.split(': ', 1)[1], stackTrace)

AnalysisException: 'Path does not exist: file:/resources/labs/BD0231EN/component-library/component-l
ibrary/deploy/component-library/data/data.parquet;'
```

Task 10.2 : Converting the parquet file to CSV format.

In [ ]:
```
if os.path.exists(data_dir + data_csv):
os.remove(data_dir + data_csv)
df.coalesce(1).write.option("header", "true").csv(data_dir + data_csv)
file = glob.glob(data_dir + data_csv + '/part-*')
shutil.move(file[0], data_dir + data_csv + '.tmp')
shutil.rmtree(data_dir + data_csv)
shutil.move(data_dir + data_csv + '.tmp', data_dir + data_csv)
```

Task 10.3 : Loading the CSV file into a dataframe

In [ ]:
```
# Reading the file using `read_csv` function in pandas
pd_df_csv = pd.read_csv('./component-library/data/randomforest.csv')

# pd_df_csv.head()   -- to view the first frew rows of the dataframe

# useing the `createDataFrame` function to load the data into a spark dataframe
sdf = spark.createDataFrame(pd_df_csv)
```

Task 10.4 : Creating a 80-20 training and test split with seed=1.

In [ ]:
```
# casting feature columns to doubletype
sdf = sdf.withColumn("x", sdf.x.cast(DoubleType()))
sdf = sdf.withColumn("y", sdf.y.cast(DoubleType()))
sdf = sdf.withColumn("z", sdf.z.cast(DoubleType()))

# spliting dataframe into training and testing subsets
splits = sdf.randomSplit([0.8, 0.2], seed=1)
df_train = splits[0]
```

Task 10.5 : Train a Random Forest model with different hyperparameters listed below and report the best performing hyperparameter combinations.

Hyper parameters:

- number of trees : {10, 20}

- maximum depth : {5, 7}

- use random seed = 1 wherever needed

In [ ]:
```
# indexing classes
indexer = StringIndexer(inputCol="class", outputCol="label")
input_columns = ['x', 'y', 'z']

# aggregating feature columns into vector
vectorAssembler = VectorAssembler(inputCols=input_columns,outputCol="features")

# normalizing features
normalizer = MinMaxScaler(inputCol="features", outputCol="features_norm")

# creating pandas dataframe to keep predictions accuracy
pd_df = pd.DataFrame(columns = ['n_trees', 'max_depth', 'accuracy'])

# hyperparameter testing
```

```
for n_trees in [10, 20]:
    for max_depth in [5, 7]:

        rf = RandomForestClassifier(numTrees=n_trees, maxDepth=max_depth, featuresCol="features_nor

        pipeline = Pipeline(stages=[indexer, vectorAssembler, normalizer, rf])
        rf_model = pipeline.fit(df_train)
        predictions = rf_model.transform(df_test)

        # evaluate predictions
        evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="predictions"
        accuracy = evaluator.evaluate(predictions)

        # print accuracy
        print("# Trees = %s" % (n_trees))
        print("Max Depth = %s" % (max_depth))
        print("Accuracy = %s" % (accuracy))

        # add entry to pandas dataframe
        pd_df = pd_df.append({'n_trees' : n_trees, 'max_depth' : max_depth, 'accuracy' :accuracy,ig
```

Task 10.6 : Use the accuracy metric when evaluating the model with different hyperparameters

In [ ]:
```
# print parameters with highest accuracy

pd_df[pd_df['accuracy'] == pd_df['accuracy'].max()]
```