Binary_Classification_using_Logistic_Regression_with_SparkMLib

September 6, 2023

Task 2 - Binary Classification using Logistic Regression

GOAL: The goal of this task is to build a machine learning pipeline including a classification model that predicts the `Attrition` (Yes or No) from the features included in the dataset (income, work years, education level, marital status, job role, and so on), which we used in the Lab 3 and Lab 4.

[0]:
```
#  IMPORTING THE NECESSARY LIBRARIES

# To convert categorical variables to numeric
from pyspark.ml.feature import StringIndexer

# To combine the feature columns into one single column
from pyspark.ml.feature import VectorAssembler

# For logistic regression
from pyspark.ml.classification import LogisticRegression

#For building the Pipeline
from pyspark.ml import Pipeline

# For checking the accuracy
from pyspark.ml.evaluation import BinaryClassificationEvaluator,␣
↪MulticlassClassificationEvaluator



# For Hyperparameter tuning
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
```

2.2 Loading the dataset and displaying the schema

[0]:
```
# Loading the dataset
df1 = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/
↪shared_uploads/clvrashmika@gmail.com/EmployeeAttrition.csv", inferSchema =␣
↪"true")
```

[0]:
```
# Printing the dataset's schema
df1.printSchema()
```

```
root
 |-- Age: integer (nullable = true)
 |-- Attrition: string (nullable = true)
 |-- BusinessTravel: string (nullable = true)
 |-- DailyRate: integer (nullable = true)
 |-- Department: string (nullable = true)
 |-- DistanceFromHome: integer (nullable = true)
 |-- Education: integer (nullable = true)
 |-- EducationField: string (nullable = true)
 |-- EmployeeCount: integer (nullable = true)
 |-- EmployeeNumber: integer (nullable = true)
 |-- EnvironmentSatisfaction: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- HourlyRate: integer (nullable = true)
 |-- JobInvolvement: integer (nullable = true)
 |-- JobLevel: integer (nullable = true)
 |-- JobRole: string (nullable = true)
 |-- JobSatisfaction: integer (nullable = true)
 |-- MaritalStatus: string (nullable = true)
 |-- MonthlyIncome: integer (nullable = true)
 |-- MonthlyRate: integer (nullable = true)
 |-- NumCompaniesWorked: integer (nullable = true)
 |-- Over18: string (nullable = true)
 |-- OverTime: string (nullable = true)
 |-- PercentSalaryHike: integer (nullable = true)
 |-- PerformanceRating: integer (nullable = true)
 |-- RelationshipSatisfaction: integer (nullable = true)
 |-- StandardHours: integer (nullable = true)
 |-- StockOptionLevel: integer (nullable = true)
 |-- TotalWorkingYears: integer (nullable = true)
 |-- TrainingTimesLastYear: integer (nullable = true)
 |-- WorkLifeBalance: integer (nullable = true)
 |-- YearsAtCompany: integer (nullable = true)
 |-- YearsInCurrentRole: integer (nullable = true)
 |-- YearsSinceLastPromotion: integer (nullable = true)
 |-- YearsWithCurrManager: integer (nullable = true)
```

2.3 Splitting the dataset into training and testing sets & Displaying distribution of HourlyRate and Education

[0]:
```
# Splitting the dataset into train and test dataframes
trainDF, testDF = df1.randomSplit([0.8, 0.2], seed=65)
print(trainDF.cache().count()) # Cache because accessing training data multiple␣
↪times
print(testDF.count())
```

```
1204
```

```
[0]: # Checking the distribution of the 'HourlyRate' field in the training dataset␣
     ↪using the summary()
     display(trainDF.select('HourlyRate').summary())
```

```
[0]: # Checking the distribution of the 'Education' field in the training dataset␣
     ↪using groupBY
     display(trainDF.groupBy('Education').count().sort("count", ascending = False))
```

2.4 Feature Processing

```
[0]: #  2.4.1 - Selecting 5 categorical cols from the dataset
     categorical_cols = ["Department", "EducationField", "Gender", "JobRole",␣
     ↪"MaritalStatus"]

     # Coverting the above columns to numerical using stringIndexer
     stringIndexer = StringIndexer(inputCols=categorical_cols, outputCols=[i +␣
     ↪"IndexedCol" for i in categorical_cols])

     # 2.4.2 - Setting the Attritition Feature (Yes/No) as a label
     # Converting to a numeric value
     labelToNum = StringIndexer(inputCol="Attrition", outputCol="NewAttritionCol")
     labelToNum

     #Applying this to the dataset
     stringIndexerModel = stringIndexer.fit(trainDF)

     labelIndexerModel = labelToNum.fit(trainDF)
```

```
[0]: #  2.4.3 and 2.4.4
     #  Combining the feature columns into a new single feature
     numerical_columns = ["Age", "DailyRate", "Education", "DistanceFromHome",␣
     ↪"HourlyRate", "JobInvolvement", "JobLevel", "JobSatisfaction",␣
     ↪"MonthlyIncome", "YearsAtCompany", "YearsInCurrentRole",␣
     ↪"YearsWithCurrManager", "NumCompaniesWorked", "PerformanceRating",␣
     ↪"EnvironmentSatisfaction" ]

     vector_assembler = VectorAssembler(inputCols=numerical_columns,␣
     ↪outputCol="features")
```

2.5 Defining the Model

```
[0]: # Defining the model for Logistic Regression
     log_regression = LogisticRegression(featuresCol="features",␣
     ↪labelCol="NewAttritionCol", regParam=1.0)
```

2.6 - Building the Pipeline

[0]:
```
# Defining the pipeline based on the above created stages
pipeline = Pipeline(stages=[stringIndexer, labelToNum, vector_assembler,␣
↪log_regression])


# Defining the pipeline model
pipelineModel = pipeline.fit(trainDF)


# Apply the pipeline model to the test database
predDF = pipelineModel.transform(testDF)
```

2.6 (Cont.) - Displaying the Predictions

[0]:
```
display(predDF.select("features", "NewAttritionCol", "prediction",␣
↪"probability"))
```

2.7 - Evaluating the Model

[0]:
```
# Plotting the ROC curve
display(pipelineModel.stages[-1], predDF.drop("prediction", "rawPrediction",␣
↪"probability"), "ROC")
```

[0]:
```
# Printing the area under the curve and the accuracy
binary_class_eval = BinaryClassificationEvaluator(metricName="areaUnderROC",␣
↪labelCol="NewAttritionCol")

print("Area under ROC curve: ", binary_class_eval.evaluate(predDF))

multi_class_eval = MulticlassClassificationEvaluator(metricName="accuracy",␣
↪labelCol="NewAttritionCol")

print("Accuracy: ", multi_class_eval.evaluate(predDF))
```

```
Area under ROC curve:  0.7369510015987963
Accuracy:  0.8157894736842105
```

2.8 - Hyperparameter Tuning

[0]:
```
# Using ParamGridBuilder
parameterGrid = (ParamGridBuilder()
                 .addGrid(log_regression.regParam, [0.01, 0.5, 2.0])
                 .addGrid(log_regression.elasticNetParam, [0.0, 0.5, 1.0])
                 .build())
```

[0]:
```
# Using CrossValidator

#Creating a 3-fold CrossValidator
```

```
cross_validator = CrossValidator(estimator=pipeline,␣
↪estimatorParamMaps=parameterGrid, evaluator=binary_class_eval, numFolds=3)

# Running the cross validations to find the best model
cross_validator_model = cross_validator.fit(trainDF)
```

2.9 - Make predictions and Evaluate the model performance

[0]:
```
cvPredDF = cross_validator_model.transform(testDF)


#Evaluating the Model performance
print("Area under ROC curve: ", binary_class_eval.evaluate(cvPredDF))
print("Accuracy: ", multi_class_eval.evaluate(cvPredDF))
```

```
Area under ROC curve:  0.7123107307439106
Accuracy:  0.8157894736842105
```

2.10 Use SQL Commands

[0]:
```
# 2.10.1 Creating a temporary view of the predictions dataset
cvPredDF.createOrReplaceTempView("finalPredictions")
```

2.10.2 Displaying the predictions grouped by JobRole - Bar Chart

[0]:
```
%sql
SELECT JobRole, prediction, count(1||2) as Count
FROM finalPredictions
Group By JobRole, prediction
Order By JobRole
```

```
Output can only be rendered in Databricks
```

2.10.3 Displaying the predictions grouped by Age - Bar Chart

[0]:
```
%sql
SELECT Age, prediction, count(1||2) as Count
FROM finalPredictions
Group By Age, prediction
Order By Age
```

```
Output can only be rendered in Databricks
```

References:

1. Dr. Liao's Code Examples & Tutorials: Blackboard/Liao_PySpark_basic_databricks.html 2. PySpark: https://spark.apache.org/docs/2.4.0/api/python/pyspark.html