CISC 5950 - Big Data Programming

Project 2 - Part 3 Report

Team Members:

Revathi Bhuvaneswari

Tianying Luo

Youfei Zhang

Yue Zeng

Spring 2019

P3: Logistic Regression on Census Income Data

Bash Script: Both train + test data are copied from local and sent as input.

```
#!/bin/bash
source ../../env.sh
/usr/local/hadoop/bin/hdfs dfs -rm -r /part3/input/
/usr/local/hadoop/bin/hdfs dfs -rm -r /part3/output/
/usr/local/hadoop/bin/hdfs dfs -mkdir -p /part3/input/
/usr/local/hadoop/bin/hdfs dfs -copyFromLocal ../../data/adult.data.csv /part3/input/
/usr/local/hadoop/bin/hdfs dfs -copyFromLocal ../../data/adult.test.csv /part3/input/
/usr/local/spark/bin/spark-submit --master=spark://$SPARK_MASTER:7077 ./p3.py hdfs://$SPARK_MASTER:9000/part3/input/
```

Python Script:

Stage 1: The train / test data are read from the CSV files and loaded into dataframes.

```python
# read the train and test CSV datafiles into a dataframes
train_df = sqlContext.read.load(train_file_path, format = 'com.databricks.spark.csv',
                                header = 'true', inferSchema = 'true',
                                ignoreLeadingWhiteSpace='true', ignoreTrailingWhiteSpace='true')
n_rows = train_df.count()
n_cols = len(train_df.columns)
train_df.show(5, False)
print('# Initial Train Rows :', n_rows, '\t# Initial Train Columns :', n_cols, '\n')

test_df = sqlContext.read.load(test_file_path, format = 'com.databricks.spark.csv',
                               header = 'true', inferSchema = 'true',
                               ignoreLeadingWhiteSpace='true', ignoreTrailingWhiteSpace='true')
n_rows = test_df.count()
n_cols = len(test_df.columns)
test_df.show(5, False)
print('# Initial Test Rows :', n_rows, '\t# Initial Test Columns :', n_cols, '\n')
```

Stage 2: The train / test data are cleaned, and 'income' label is encoded as: **{<=50K: 0, >50K: 1}**

```python
# finding frequent items in train_df to fill missing values in train and test dataframe
freq_items = train_df.freqItems(['native-country', 'workclass', 'occupation'], support = 0.6).collect()
train_df = clean_df(train_df, freq_items)
train_df = train_df.withColumn('income', when(train_df['income'] == '<=50K', 0).otherwise(1))
n_rows = train_df.count()
n_cols = len(train_df.columns)
train_df.show(5, False)
train_df.printSchema()
print('\n# Final Train Rows :', n_rows, '\t# Final Train Columns :', n_cols, '\n')

test_df = clean_df(test_df, freq_items)
test_df = test_df.withColumn('income', when(test_df['income'] == '<=50K.', 0).otherwise(1))
n_rows = test_df.count()
n_cols = len(test_df.columns)
test_df.show(5, False)
test_df.printSchema()
print('\n# Final Test Rows :', n_rows, '\t# Final Test Columns :', n_cols, '\n')
```

The cleaning process is done by first dropping the unwanted columns of ‘education’ & ‘fnlwgt’. The missing values in ‘native-country’, ‘workclass’ and ‘occupation’ are filled by the frequent value from each of these columns of train DF. Additionally, ‘native-country’ column is altered by converting it into a binary column: ‘United-States’ and ‘Not-US’ to make things simpler.

```python
# returns a cleaned dataframe
def clean_df(df, freq_items):
    # drop unwanted columns
    columns_to_drop = ['education', 'fnlwgt']
    df = df.drop(*columns_to_drop)
    # replace missing values with most frequent values
    df = df.withColumn('native-country',
                       when(df['native-country'] == '?', freq_items[0][0][0]).otherwise(df['native-country']))
    df = df.withColumn('workclass',
                       when(df['workclass'] == '?', freq_items[0][1][0]).otherwise(df['workclass']))
    df = df.withColumn('occupation',
                       when(df['occupation'] == '?', freq_items[0][2][0]).otherwise(df['occupation']))
    # convert native-country to be a two value column
    df = df.withColumn('native-country',
                       when(df['native-country'] != 'United-States', 'Not-US').otherwise(df['native-country']))
    # one-hot-encode all categorical columns
    df = ohe(df)
    return df
```

The following function is used to one-hot-encode all category columns, and get vector representation of all features. String Indexer and One Hot Encoder Estimator are used for this.

```python
# returns one-hot-encoded stages
def ohe(df):
    cols = df.columns
    # One-hot-encode category columns
    categoricalColumns = ['workclass', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'native-country']
    stages = []
    for categoricalCol in categoricalColumns:
        stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol = categoricalCol + 'Index')
        encoder = OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
        stages += [stringIndexer, encoder]

    # define income as label column
    label_stringIdx = StringIndexer(inputCol = 'income', outputCol = 'label')
    stages += [label_stringIdx]

    # vectorize numeric columns
    numericCols = ["age", "education-num", "capital-gain", "capital-loss", "hours-per-week"]
    assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
    assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="initial_features")
    stages += [assembler]

    # perform final pipeline to get vectorized feature column
    pipeline = Pipeline().setStages(stages)
    pipelineModel = pipeline.fit(df)
    df = pipelineModel.transform(df)
    selectedCols = ['label', 'initial_features'] + cols
    df = df.select(selectedCols)

    # convert sparse vector initial_feature column to dense vector
    sparse_to_array_udf = udf(sparse_to_array, T.ArrayType(T.FloatType()))
    df = df.withColumn('dense_vector_features', sparse_to_array_udf('initial_features'))
    ud_f = udf(lambda r : Vectors.dense(r), VectorUDT())
    df = df.withColumn('features', ud_f('dense_vector_features'))
    return df
```

Stage 3: In this stage, the cleaned train / test data frames are used to extract just the 'label' and 'features' columns to create a new data frame. A Logistic Regression model is created with some suitable parameters, and the features as well as the label columns are set accordingly. In the end, this model is trained with the newly created train data frame, and the coefficients / intercepts of the trained model are obtained.

```
# create new dataframe with just two columns - features and income(label)
vtrain_df = train_df.select(['label', 'features'])
vtrain_df.show(5, False)
vtrain_df.printSchema()
print('\n\tFinal Changed Train Dataframe for Logistic Regression\n')
vtest_df = test_df.select(['label', 'features'])
vtest_df.show(5, False)
vtest_df.printSchema()
print('\n\tFinal Changed Test Dataframe for Logistic Regression\n')
```

```
# initializing Logistic Regression Model
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter = 10)

# fitting the model
lrModel = lr.fit(vtrain_df)

# printing the coefficients and intercept for logistic regression
print('\nCoefficients: ', lrModel.coefficients)
print('\nIntercept: ', lrModel.intercept, '\n')
```

Stage 4: In this last stage, the trained Logistic Regression model is used to make predictions by applying the transform() function on both the train as well as the test data frames. The Area Under the ROC and Area Under the PR is calculated using the Binary Classification Evaluator by passing in the actual predictions data frame, which has both the labels and predictions.

```
evaluator = BinaryClassificationEvaluator()
train_pred = lrModel.transform(vtrain_df)
train_pred.show(5, False)
print('\nEntire Train Predictions DataFrame\n')
train_roc = evaluator.setMetricName('areaUnderROC').evaluate(train_pred)
train_pr = evaluator.setMetricName('areaUnderPR').evaluate(train_pred)
condensed_train_pred = train_pred.select(['label', 'prediction'])
train_acc = round(get_accuracy_rate(condensed_train_pred), 2)
condensed_train_pred.show(5, False)
print('\nCondensed Train Predictions DataFrame\n')
print('Train Accuracy Rate :', train_acc)
print('Train Area Under ROC :', train_roc)
print('Train Area Under PR :', train_pr, '\n')
```

```python
# getting test predictions and accuracy rate / area under ROC / area under PR
test_pred = lrModel.transform(vtest_df)
test_pred.show(5, False)
print('\nEntire Test Predictions DataFrame\n')
test_roc = evaluator.setMetricName('areaUnderROC').evaluate(test_pred)
test_pr = evaluator.setMetricName('areaUnderPR').evaluate(test_pred)
condensed_test_pred = test_pred.select(['label', 'prediction'])
test_acc = round(get_accuracy_rate(condensed_test_pred), 2)
condensed_test_pred.show(5, False)
print('\nCondensed Test Predictions DataFrame\n')
print('Test Accuracy Rate :', test_acc)
print('Test Area Under ROC :', test_roc)
print('Test Area Under PR :', test_pr, '\n')
```

The accuracy rate of the predictions is obtained by using a custom function, which finds the number of correct predictions through comparing label and prediction column.

Accuracy Rate = [# Correct Predictions / # Total Predictions] * 100

```python
# returns accuracy rate of given prediction dataframe
def get_accuracy_rate(pred_df):
    accuracy_rate = 0.0
    num_total_preds = pred_df.count()
    pred_df = pred_df.withColumn('isSame', when(pred_df['label'] == pred_df['prediction'], 1.0).otherwise(0.0))
    num_correct_preds = pred_df.select(sum('isSame')).collect()[0][0]
    accuracy_rate = (float(num_correct_preds) / float(num_total_preds)) * 100.0
    return accuracy_rate
```

Output:

```
2019-04-23 02:23:33 INFO  DAGScheduler:54 - Job 3 finished: showString at NativeMethodAccessorImpl.java:0, took 0.365396 s
+---+----------------+------+---------+-------------+------------------+-----------------+-------------+-----+------+------------+------------+--------------+--------------+------+
|age|workclass       |fnlwgt|education|education-num|marital-status    |occupation       |relationship |race |sex   |capital-gain|capital-loss|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

# Initial Train Rows : 32561    # Initial Train Columns : 15
```

Figure 1: Initial Train DataFrame / Dimensions

```
+---+---------+------+------------+-------------+------------------+-----------------+------------+-----+------+------------+------------+--------------+--------------+------+
|age|workclass|fnlwgt|education   |education-num|marital-status    |occupation       |relationship|race |sex   |capital-gain|capital-loss|hours-per-week|native-country|income|
+---+---------+------+------------+-------------+------------------+-----------------+------------+-----+------+------------+------------+--------------+--------------+------+
|25 |Private  |226802|11th        |7            |Never-married     |Machine-op-inspct|Own-child   |Black|Male  |0           |0           |40            |United-States |<=50K.|
|38 |Private  |89814 |HS-grad     |9            |Married-civ-spouse|Farming-fishing  |Husband     |White|Male  |0           |0           |50            |United-States |<=50K.|
|28 |Local-gov|336951|Assoc-acdm  |12           |Married-civ-spouse|Protective-serv  |Husband     |White|Male  |0           |0           |40            |United-States |>50K. |
|44 |Private  |160323|Some-college|10           |Married-civ-spouse|Machine-op-inspct|Husband     |Black|Male  |7688        |0           |40            |United-States |>50K. |
|18 |?        |103497|Some-college|10           |Never-married     |?                |Own-child   |White|Female|0           |0           |30            |United-States |<=50K.|
+---+---------+------+------------+-------------+------------------+-----------------+------------+-----+------+------------+------------+--------------+--------------+------+
only showing top 5 rows

# Initial Test Rows : 16281     # Initial Test Columns : 15
```

Figure 2: Initial Test DataFrame / Dimensions

```
root
 |-- label: double (nullable = false)
 |-- initial_features: vector (nullable = true)
 |-- age: integer (nullable = true)
 |-- workclass: string (nullable = true)
 |-- education-num: integer (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: integer (nullable = true)
 |-- capital-loss: integer (nullable = true)
 |-- hours-per-week: integer (nullable = true)
 |-- native-country: string (nullable = true)
 |-- income: integer (nullable = false)
 |-- dense_vector_features: array (nullable = true)
 |    |-- element: float (containsNull = true)
 |-- features: vector (nullable = true)


# Final Train Rows : 32561      # Final Train Columns : 17
```

Figure 3: Schema / Dimensions of cleaned Train DataFrame

```
root
 |-- label: double (nullable = false)
 |-- initial_features: vector (nullable = true)
 |-- age: integer (nullable = true)
 |-- workclass: string (nullable = true)
 |-- education-num: integer (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: integer (nullable = true)
 |-- capital-loss: integer (nullable = true)
 |-- hours-per-week: integer (nullable = true)
 |-- native-country: string (nullable = true)
 |-- income: integer (nullable = false)
 |-- dense_vector_features: array (nullable = true)
 |    |-- element: float (containsNull = true)
 |-- features: vector (nullable = true)


# Final Test Rows : 16281       # Final Test Columns : 17
```

Figure 4: Schema / Dimensions of cleaned Test DataFrame

```
+-----+--------------------------------------------------------------------------------+
|label|features                                                                        |
+-----+--------------------------------------------------------------------------------+
|0.0  |[0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,1.0,39.0,13.0,2174.0,0.0,40.0]|
|0.0  |[0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,1.0,50.0,13.0,0.0,0.0,13.0]  |
|0.0  |[1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,1.0,38.0,9.0,0.0,0.0,40.0]   |
|0.0  |[1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,53.0,7.0,0.0,0.0,40.0]   |
|0.0  |[1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,28.0,13.0,0.0,0.0,40.0]  |
+-----+--------------------------------------------------------------------------------+
only showing top 5 rows

root
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)


        Final Changed Train Dataframe for Logistic Regression
```

Figure 5: Final Train DataFrame used for Logistic Regression

```
+-----+--------------------------------------------------------------------------------+
|label|features                                                                        |
+-----+--------------------------------------------------------------------------------+
|0.0  |[1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,25.0,7.0,0.0,0.0,40.0]   |
|0.0  |[1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,1.0,38.0,9.0,0.0,0.0,50.0]   |
|1.0  |[0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,1.0,28.0,12.0,0.0,0.0,40.0]  |
|1.0  |[1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,44.0,10.0,7688.0,0.0,40.0]|
|0.0  |[1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,18.0,10.0,0.0,0.0,30.0]  |
+-----+--------------------------------------------------------------------------------+
only showing top 5 rows

root
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)


        Final Changed Test Dataframe for Logistic Regression
```

Figure 6: Final Test DataFrame used for Logistic Regression

```
Coefficients:
[-1.376, -1.716, -1.573, -1.692, -1.221, -0.792, -5.316, 0.092, -2.018, -0.975, -1.309,
-0.793, -1.137, -0.753, 0.037, -0.481, 0.171, -0.246, -2.299, -1.078, -0.647, -2.007,
-1.584, 0.048, 0.167, -3.33, 0.073, -0.283, -2.468, -1.205, 0.879, -1.9, -1.876,
-2.246, -2.594, 0.339, -0.063, 0.018, 0.245, 0.0, 0.001, 0.029]

Intercept:  -1.75361636115
```

Figure 7: Coefficients & Intercept of Logistic Regression Model

```
+-----+----------+
|label|prediction|
+-----+----------+
|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       |
|1.0  |0.0       |
|1.0  |1.0       |
|1.0  |1.0       |
+-----+----------+
only showing top 10 rows


Condensed Train Predictions DataFrame

Train Accuracy Rate : 84.98
Train Area Under ROC : 0.9013
Train Area Under PR : 0.7578
```

```
+-----+----------+
|label|prediction|
+-----+----------+
|0.0  |0.0       |
|0.0  |0.0       |
|1.0  |0.0       |
|1.0  |1.0       |
|0.0  |0.0       |
|0.0  |0.0       |
|0.0  |0.0       |
|1.0  |1.0       |
|0.0  |0.0       |
|0.0  |0.0       |
+-----+----------+
only showing top 10 rows


Condensed Test Predictions DataFrame

Test Accuracy Rate : 83.77
Test Area Under ROC : 0.8882
Test Area Under PR : 0.724
```

Figure 8: Train and Test Prediction Evaluation Results