COURSE: *Big Data - CTS43135*

Lab Instruction #5:

Machine Learning with Spark MLlib

Lab Objectives:

- Understand and practice data processing with PySpark and Spark MLlib.
- Load and explore the dataset.
- Perform feature preprocessing.
- Define the model and build the pipeline.

Prerequisites

- Basic knowledge of Python, SQL and Machine Learning is recommended.
- This lab runs on Python 3.6+ with Apache Spark 3.x.
- A working environment like Jupyter Notebook or Google Colab is recommended for easier execution.

Activity 1: Load the dataset & Data Exploration

1. **Download the dataset.** We need to first download the Adult Dataset (Census Income Dataset) from the UCI Machine Learning Repository. Run the following command to download the dataset:

```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data
-O adult.csv
```

2. **Display the First Few Lines.** Since the dataset does not include column names, let's display the first few rows to understand its structure:

```
!head -n 5 adult.csv
```

Expected Output:

```
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
```

3. **Initialize a Spark Session.** PySpark requires a Spark session to operate. We create one as follows:

```
# Create a Spark session
spark = SparkSession.builder.appName("lab05").getOrCreate()
```

4. **Load the Dataset.** Because the dataset does not include column names, create a schema to assign column names and datatypes.

```
schema = """`age` DOUBLE,
`workclass` STRING,
`fnlwgt` DOUBLE,
`education` STRING,
`education_num` DOUBLE,
`marital_status` STRING,
`occupation` STRING,
`relationship` STRING,
`race` STRING,
`sex` STRING,
`capital_gain` DOUBLE,
`capital_loss` DOUBLE,
`hours_per_week` DOUBLE,
`native_country` STRING,
`income` STRING"""

dataset = spark.read.csv("adult.csv", schema=schema)
```

5. **Split the dataset into training and test set.** It's best to split the data before doing any preprocessing. This ensures that the test dataset better reflects real-world data during model evaluation. Randomly split data into training and test sets, and set seed for reproducibility.

```
trainDF, testDF = dataset.randomSplit([0.8, 0.2], seed=42)
print(trainDF.cache().count()) # Cache because accessing training data multiple times
print(testDF.count())
```

6. **Data Exploration.** For example, What's the distribution of the number of `hours_per_week`?

```
display(trainDF.select("hours_per_week").summary())
```

Expected Output:

	summary	hours_per_week
1	count	26076
2	mean	40.4284782942169
3	stddev	12.404569739132008
4	min	1.0
5	25%	40.0
6	50%	40.0
7	75%	45.0
8	max	99.0

How about `education` status?

```
display(trainDF
        .groupBy("education")
        .count()
        .sort("count", ascending=False))
```

Expected Output:

	education	count
1	HS-grad	8408
2	Some-college	5860
3	Bachelors	4255
4	Masters	1388
5	Assoc-voc	1102
6	11th	958
7	Assoc-acdm	845
8	10th	748

Showing all 16 rows.

Activity 2: Feature preprocessing

1. **Convert categorical variables to numeric.** Some machine learning algorithms, such as linear and logistic regression, require numeric features. The Adult dataset includes categorical features such as education, occupation, and marital status. The following code block illustrates how to use `StringIndexer` and `OneHotEncoder` to convert categorical variables into a set of numeric variables that only take on values 0 and 1
   - `StringIndexer` converts a column of string values to a column of label indexes. For example, it might convert the values "red", "blue", and "green" to 0, 1, and 2.
   - `OneHotEncoder` maps a column of category indices to a column of binary vectors, with at most one "1" in each row that indicates the category index for that row.

```
from pyspark.ml.feature import StringIndexer, OneHotEncoder
categoricalCols = ["workclass", "education", "marital_status", "occupation",
"relationship", "race", "sex"]

# The following two lines are estimators. They return functions that we will later
apply to transform the dataset.
stringIndexer = StringIndexer(inputCols=categoricalCols, outputCols=[x + "Index" for x
in categoricalCols])
encoder = OneHotEncoder(inputCols=stringIndexer.getOutputCols(), outputCols=[x + "OHE"
for x in categoricalCols])

# The label column ("income") is also a string value - it has two possible values,
"<=50K" and ">50K".
# Convert it to a numeric value using StringIndexer.
labelToIndex = StringIndexer(inputCol="income", outputCol="label")
```

You can call the `.fit()` method to return a `StringIndexerModel`, which you can then use to transform the dataset.

The `.transform()` method of `StringIndexerModel` returns a new DataFrame with the new columns appended. Scroll right to see the new columns if necessary.

```
stringIndexerModel = stringIndexer.fit(trainDF)
```

Expected Output:

	ship	race	sex	capital_gain	capital_loss	hours_per_week	native_country	income	educationIndex	raceIndex	occupationIndex	relationshipIndex	workclassIndex	marital_statusIndex	sexIndex
1	d	White	Male	0	0	20	United-States	<=50K	7	0	7	2	3	1	0
2	d	White	Female	0	0	25	United-States	<=50K	11	0	7	2	3	1	1
3	d	White	Male	0	0	10	United-States	<=50K	5	0	7	2	3	1	0
4	d	White	Female	0	0	30	United-States	<=50K	5	0	7	2	3	1	1
5	d	Black	Female	0	0	40	United-States	<=50K	5	1	7	2	3	1	1
6	d	White	Male	0	0	40	United-States	<=50K	7	0	7	2	3	1	0
7	d	White	Female	0	0	40	United-States	<=50K	7	0	7	2	3	1	1
8															

Showing the first 1000 rows.

2. **Combine all feature columns into a single feature vector.** Most MLlib algorithms require a single features column as input. Each row in this column contains a vector of data points corresponding to the set of features used for prediction. MLlib provides the `VectorAssembler` transformer to create a single vector column from a list of columns. The following code block illustrates how to use `VectorAssembler`.

```
from pyspark.ml.feature import VectorAssembler

# This includes both the numeric columns and the one-hot encoded binary vector
columns in our dataset.
numericCols = ["age", "fnlwgt", "education_num", "capital_gain", "capital_loss",
"hours_per_week"]
assemblerInputs = [c + "OHE" for c in categoricalCols] + numericCols
vecAssembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
```

Activity 3: Define the model & Build the pipeline

1. **Create a logistic regression model.**

```
# Import the LogisticRegression model from PySpark's MLlib
from pyspark.ml.classification import LogisticRegression

# Initialize a Logistic Regression model
# - featuresCol: Specifies the column containing feature vectors
# - labelCol: Specifies the column containing the target label (0 or 1)
# - regParam: Regularization parameter (L2 regularization by default); helps
prevent overfitting
lr = LogisticRegression(featuresCol="features", labelCol="label", regParam=1.0)
```

2. **Build a pipeline.** A `Pipeline` is an ordered list of transformers and estimators. You can define a pipeline to automate and ensure repeatability of the transformations to be applied to a dataset. In this step, we define the pipeline and then apply it to the test dataset.
Similar to what we saw with `StringIndexer`, a `Pipeline` is an estimator. The `pipeline`.fit() method returns a `PipelineModel`, which is a transformer.

```
from pyspark.ml import Pipeline

# Define the pipeline based on the stages created in previous steps.
pipeline = Pipeline(stages=[stringIndexer, encoder, labelToIndex, vecAssembler,
lr])

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

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

This code creates a machine learning pipeline in PySpark, where multiple data transformation and model training steps are chained together into a single workflow. Each stage in the `Pipeline` represents a transformation or an ML model. Let's analyze them:

Stage Name	**Purpose**
`stringIndexer`	Converts categorical variables (strings) into numerical indices.
`encoder`	Applies One-Hot Encoding to categorical variables.

`labelToIndex`	Converts the target label (income column) from string to numeric values (0 or 1).
`vecAssembler`	Combines multiple features into a single vector for ML models.
`lr`	Trains a Logistic Regression model on the processed dataset.

3. **Display the predictions.** The `features` column is a sparse vector, which is often the case after one-hot encoding, because there are so many 0 values.

```
# Display selected columns: features, actual label, predicted label, and
prediction probability
predDF.select("features", "label", "prediction", "probability").show(5,
truncate=False)
```

- `features` → The input feature vector.
- `label` → The actual ground-truth label (0 or 1).
- `prediction` → The model's predicted label.
- `probability` → The probability of the prediction (for logistic regression, it's `[P(0), P(1)]`).
-

	features	label	prediction	probability
1	{"vectorType": "sparse", "length": 59, "indices": [3, 13, 24, 36, 45, 48, 53, 54, 55, 58], "values": [1, 1, 1, 1, 1, 1, 17, 41643, 7, 15]}	0	0	{"vectorType": "dense", "length": 2, "values": [0.9062474976435643, 0.09375250235643576]}
2	{"vectorType": "sparse", "length": 59, "indices": [3, 15, 24, 36, 45, 48, 52, 53, 54, 55, 58], "values": [1, 1, 1, 1, 1, 1, 1, 17, 64785, 6, 30]}	0	0	{"vectorType": "dense", "length": 2, "values": [0.8927691288853388, 0.10723087111466127]}
3	{"vectorType": "sparse", "length": 59, "indices": [3, 13, 24, 36, 45, 48, 53, 54, 55, 58], "values": [1, 1, 1, 1, 1, 1, 17, 80077, 7, 20]}	0	0	{"vectorType": "dense", "length": 2, "values": [0.9041097206748728, 0.09589027932512711]}
4	{"vectorType": "sparse", "length": 59, "indices": [3, 13, 24, 36, 45, 48, 52, 53, 54, 55, 58], "values": [1, 1, 1, 1, 1, 1, 1, 17, 104025, 7, 18]}	0	0	{"vectorType": "dense", "length": 2, "values": [0.8952738661074835, 0.10472613389251656]}
	{"vectorType": "sparse", "length": 59, "indices": [3, 15, 24, 36, 45, 48, 53, 54, 55, 58], "values": [1, 1, 1, 1, 1, 1, 17, 139183, 6,	0	0	{"vectorType": "dense", "length": 2, "values": [0.9087696046250343,

4. **Evaluate the model.** To evaluate the model, we use the `BinaryClassificationEvaluator` to evaluate the area under the ROC curve and the `MulticlassClassificationEvaluator` to evaluate the accuracy.

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator,
```

```
MulticlassClassificationEvaluator

bcEvaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")
print(f"Area under ROC curve: {bcEvaluator.evaluate(predDF)}")

mcEvaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print(f"Accuracy: {mcEvaluator.evaluate(predDF)}")
```

Credits

*This lab is based on the original work from **Databricks**, available at:* 🔗 [Databricks]

Lab Assignment: Sentiment Analysis on IMDB Movie Reviews

Objective

- Load and preprocess an IMDB movie reviews dataset using PySpark MLlib.
- Train a classifier to predict the sentiment of movie reviews as positive or negative.
- Evaluate model performance using Accuracy, Precision, Recall, and F1-score.

Instructions

Download the IMDB Reviews Dataset: 🔗 [IMDB Dataset]

This dataset contains 50,000 movie reviews labeled as positive or negative, which will be used to build a sentiment classification model.

Your goal is to process the dataset and apply machine learning techniques using **Spark MLlib**.

- Load and preprocess the dataset, ensuring valid movie reviews and sentiment labels.
- Convert text labels into binary format (0 = negative, 1 = positive).
- Clean the text data by removing stopwords, punctuation, and lowercasing.
- Convert text reviews into numerical features using TF-IDF or Word2Vec.
- Split the dataset into training (80%) and testing (20%) sets.
- Train a classification model in PySpark MLlib.
- Evaluate the model using Accuracy, Precision, Recall, and F1-score.

Submission

- **Submission deadline:** 2 weeks from the assignment date.
- **Submission Format:** Upload the Executed Notebook (or similar) to LMS (lms.siu.edu.vn).

Suggested Resources

- https://spark.apache.org/docs/latest/api/python/index.html
- https://spark.apache.org/sql/