Skills
Network

Machine Learning with Apache Spark ML

Estimated time needed: **15** minutes

This lab goes introduces Machine Learning using Spark ML Lib (sparkml).

Objectives

Spark ML Library is also commonly called MLlib and is used to perform machine learning operations using DataFrame-based APIs.

After completing this lab you will be able to:

- Import the Spark ML and Statistics Libraries
- Perform basic statistics operations using Spark
- Build a simple linear regression model using Spark ML
- Train the model and perform evaluation

Setup

For this lab, we are going to be using Python and Spark (pyspark). These libraries should be installed in your lab environment or in SN Labs.

In [1]:
```
# When you are executing on SN labs please uncomment the below lines and then run all cells.Nex
!pip3 install pyspark==3.1.2
!pip install findspark
import findspark
findspark.init()
```

```
Collecting pyspark==3.1.2
  Downloading pyspark-3.1.2.tar.gz (212.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 212.4/212.4 MB 2.6 MB/s eta 0:00:0000:0100:01
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9 (from pyspark==3.1.2)
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 198.6/198.6 kB 26.8 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.1.2-py2.py3-none-any.whl size=212880756 sha256=0
591f942523b47aa7b4e4dc40c1efa4b419df1afc06447c972a746b48b6c778f
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/a5/0a/c1/9561f6fecb759579a7d863dcd846d
aaa95f598744e71b02c77
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9 pyspark-3.1.2
Collecting findspark
  Downloading findspark-2.0.1-py2.py3-none-any.whl (4.4 kB)
Installing collected packages: findspark
Successfully installed findspark-2.0.1
```

In [2]:
```python
# Pandas is a popular data science package for Python. In this lab, we use Pandas to load a CSV
import pandas as pd
import matplotlib.pyplot as plt
# pyspark is the Spark API for Python. In this lab, we use pyspark to initialize the spark cont
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
```

Exercise 1 - Spark session

In this exercise, you will create and initialize the Spark session needed to load the dataframes and operate on it

Task 1: Creating the spark session and context

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

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

```
23/08/01 23:24:54 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your plat
form... 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 2: Initialize Spark session

To work with dataframes we just need to verify that the spark session instance has been created. Feel free to click on the "Spark UI" button to explore the Spark UI elements.

In [4]:
```python
spark
```

Out[4]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	`v2.4.3`
Master	`local[*]`
AppName	`pyspark-shell`

Task 2: Importing Spark ML libraries

In this exercise we will import 4 SparkML functions.

1. (Feature library) VectorAssembler(): This function is used to create feature vectors from dataframes/raw data. These feature vectors are required to train a ML model or perform any statistical operations.
2. (Stat library) Correlation(): This function is from the statistics library within SparkML. This function is used to calculate correlation between feature vectors.
3. (Feature library) Normalized(): This function is used to normalize features. Normalizing features leads to better ML model convergence and training results.
4. (Regression Library) LinearRegression(): This function is used to create a Linear Regression model and train it.

In [5]:
```
from pyspark.ml.feature import VectorAssembler, Normalizer, StandardScaler
from pyspark.ml.stat import Correlation
from pyspark.ml.regression import LinearRegression
```

Exercise 2 - Loading the data and Creating Feature Vectors

In this section, you will first read the CSV file into a pandas dataframe and then read it into a Spark dataframe

Pandas is a library used for data manipulation and analysis. Pandas offers data structures and operations for creating and manipulating Data Series and DataFrame objects. Data can be imported from various data sources, e.g., Numpy arrays, Python dictionaries and CSV files. Pandas allows you to manipulate, organize and display the data.

In this example we use a dataset that contains information about cars.

Task 1: Loading data into a Pandas DataFrame

In [6]:
```
# Read the file using `read_csv` function in pandas
cars = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-BD02
```

In [7]:
```
# Preview a few records
cars.head()
```

Out[7]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model	origin	car_name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

For this example, we pre process the data and only use 3 columns. This preprocessed dataset can be found in the `cars2.csv` file.

In [8]:
```
cars2 = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-BD0
cars2.head()
```

Out[8]:

	mpg	hp	weight
0	18.0	130.0	3504.0
1	15.0	165.0	3693.0
2	18.0	150.0	3436.0
3	16.0	150.0	3433.0
4	17.0	140.0	3449.0

Task 2: Loading data into a Spark DataFrame

In [9]:
```
# We use the `createDataFrame` function to load the data into a spark dataframe
sdf = spark.createDataFrame(cars2)
```

In [10]:
```
# Let us look at the schema of the loaded spark dataframe
sdf.printSchema()
```

```
root
 |-- mpg: double (nullable = true)
 |-- hp: double (nullable = true)
 |-- weight: double (nullable = true)
```

Task 3: Converting data frame columns into feature vectors

In this task we use the `VectorAssembler()` function to convert the dataframe columns into feature vectors. For our example, we use the horsepower ("hp) and weight of the car as input features and the miles-per-gallon ("mpg") as target labels.

In [11]:
```
assembler = VectorAssembler(
    inputCols=["hp", "weight"],
    outputCol="features")

output = assembler.transform(sdf).select('features','mpg')
```

We now create a test-train split of 75%-25%

In [12]:
```
train, test = output.randomSplit([0.75, 0.25])
```

Exercise 3 - Basic stats and feature engineering

In this exercise, we determine the correlation between feature vectors and normalize the features.

Task 1: Correlation

Spark ML has inbuilt Correlation function as part of the Stat library. We use the correlation function to determine the different types of correlation between the 2 features - "hp" and "weight".

In [13]:
```
r1 = Correlation.corr(train, "features").head()
print("Pearson correlation matrix:\n" + str(r1[0]))
```

```
[Stage 3:>                                                          (0 + 8) / 8]23/08/01 23:39:4
3 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLA
S
23/08/01 23:39:43 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netli
b.NativeRefBLAS
```

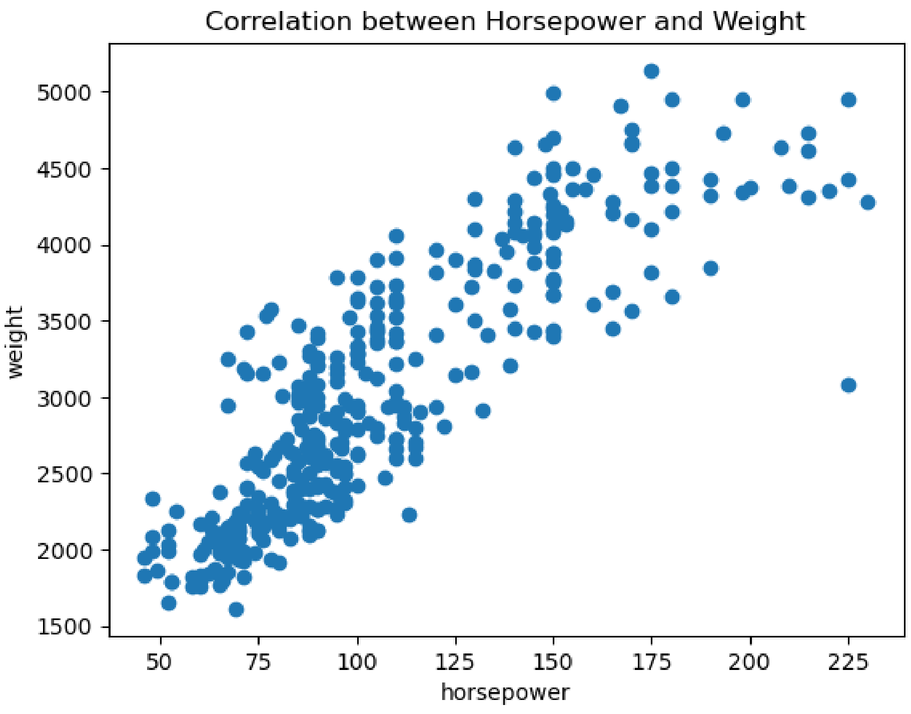
```
Pearson correlation matrix:
DenseMatrix([[1.        , 0.86618724],
             [0.86618724, 1.        ]])
```

In [14]:
```
r2 = Correlation.corr(train, "features", "spearman").head()
print("Spearman correlation matrix:\n" + str(r2[0]))
```

```
Spearman correlation matrix:
DenseMatrix([[1.        , 0.88535846],
             [0.88535846, 1.        ]])
```

We can see that there is a 0.86 (or 86%) correlation between the features. That is logical as a car with higher horsepower likely has a bigger engine and thus weighs more. We can also visualize the feature vectors to see that they are indeed correlated.

In [15]:
```
plt.figure()
plt.scatter(cars2["hp"], cars2["weight"])
plt.xlabel("horsepower")
plt.ylabel("weight")
plt.title("Correlation between Horsepower and Weight")
plt.show()
```



Task 2: Normalization

In order for better model training and convergence, it is a good practice to normalize feature vectors.

In [16]:
```
normalizer = Normalizer(inputCol="features", outputCol="features_normalized", p=1.0)
train_norm = normalizer.transform(train)
print("Normalized using L^1 norm")
train_norm.show(5, truncate=False)
```

```
Normalized using L^1 norm
+-------------+----+-----------------------------------------+
|features     |mpg |features_normalized                      |
+-------------+----+-----------------------------------------+
|[46.0,1835.0]|26.0|[0.024455077086656035,0.9755449229133439]|
|[72.0,2408.0]|22.0|[0.02903225806451613,0.9709677419354839] |
|[85.0,2587.0]|21.0|[0.03181137724550898,0.968188622754491]  |
|[86.0,2220.0]|23.0|[0.0372940156114484,0.9627059843885516]  |
|[88.0,2130.0]|27.0|[0.03967538322813345,0.9603246167718665] |
+-------------+----+-----------------------------------------+
only showing top 5 rows
```

Task 2: Standard Scaling

This is a standard practice to scale the features such that all columns in the features have zero mean and unit variance.

In [17]:
```
standard_scaler = StandardScaler(inputCol="features", outputCol="features_scaled")
train_model = standard_scaler.fit(train)
train_scaled = train_model.transform(train)
train_scaled.show(5, truncate=False)
```

```
+-------------+----+---------------------------------------+
|features     |mpg |features_scaled                        |
+-------------+----+---------------------------------------+
|[46.0,1835.0]|26.0|[1.172826430031535,2.108707458690926]  |
|[72.0,2408.0]|22.0|[1.8357283252667504,2.767175782303951] |
|[85.0,2587.0]|21.0|[2.167179272884358,2.972875310971894]  |
|[86.0,2220.0]|23.0|[2.1926754996241744,2.551133819233709] |
|[88.0,2130.0]|27.0|[2.2436679531038064,2.4477094752107207]|
+-------------+----+---------------------------------------+
only showing top 5 rows
```

In [18]:
```
test_scaled = train_model.transform(test)
test_scaled.show(5, truncate=False)
```

```
+--------------+----+---------------------------------------+
|features      |mpg |features_scaled                        |
+--------------+----+---------------------------------------+
|[87.0,2672.0] |25.0|[2.21817172636399,3.0705538581047165]  |
|[95.0,2372.0] |24.0|[2.422141540282518,2.725806044694756]  |
|[95.0,2375.0] |25.0|[2.422141540282518,2.7292535228288552] |
|[105.0,3439.0]|16.0|[2.6771038076806777,3.9519591010561825]|
|[110.0,2962.0]|18.0|[2.804584941379758,3.403810077734345]  |
+--------------+----+---------------------------------------+
only showing top 5 rows
```

Exercise 4 - Building and Training a Linear Regression Model

In this exercise, we train a Linear Regression model `lrModel` on our training dataset. We train the model on the standard scaled version of features. We also print the final RMSE and R-Squared metrics.

Task 1: Create and Train model

We can create the model using the `LinearRegression()` class and train using the `fit()` function.

In [19]:
```
# Create a LR model
lr = LinearRegression(featuresCol='features_scaled', labelCol='mpg', maxIter=100)

# Fit the model
lrModel = lr.fit(train_scaled)

# Print the coefficients and intercept for linear regression
print("Coefficients: %s" % str(lrModel.coefficients))
print("Intercept: %s" % str(lrModel.intercept))

# Summarize the model over the training set and print out some metrics
trainingSummary = lrModel.summary
#trainingSummary.residuals.show()
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("R-squared: %f" % trainingSummary.r2)
```

```
23/08/01 23:49:17 WARN util.Instrumentation: [e256e9c2] regParam is zero, which might cause nume
rical instability and overfitting.
23/08/01 23:49:18 WARN netlib.LAPACK: Failed to load implementation from: com.github.fommil.netl
ib.NativeSystemLAPACK
23/08/01 23:49:18 WARN netlib.LAPACK: Failed to load implementation from: com.github.fommil.netl
ib.NativeRefLAPACK
[Stage 30:>                                                         (0 + 8) / 8]
Coefficients: [-1.6225032650896452,-5.264306558325767]
Intercept: 45.93396803839196
RMSE: 4.200367
R-squared: 0.718321
```

We see a RMSE (Root mean squared error) of 4.26. This means that our model predicts the `mpg` with an average error of 4.2 units.

Task 2: Predict on new data

Once a model is trained, we can then `transform()` new unseen data (for eg. the test data) to generate predictions. In the below cell, notice the "prediction" column that contains the predicted "mpg".

In [20]:
```
lrModel.transform(test_scaled).show(5)
```

```
+--------------+----+--------------------+------------------+
|      features| mpg|     features_scaled|        prediction|
+--------------+----+--------------------+------------------+
| [87.0,2672.0]|25.0|[2.21817172636399...|26.170640356923705|
| [95.0,2372.0]|24.0|[2.42214154028251...| 27.65455684296369|
| [95.0,2375.0]|25.0|[2.42214154028251...|27.636408261212665|
|[105.0,3439.0]|16.0|[2.67710380768067...|20.786034155520873|
|[110.0,2962.0]|18.0|[2.80458494137975...|23.464820098269797|
+--------------+----+--------------------+------------------+
only showing top 5 rows
```

Question 1 - Correlation

Print the correlation matrix for the test dataset split we created above.

In [ ]:
```
# Code block for learners to answer
```

Double-click **here** for the solution.

Question 2 - Feature Normalization

Normalize the training features by using the L2 norm of the feature vector.

In [ ]:
```
# Code block for learners to answer
```

Double-click **here** for the solution.

Question 3 - Train Model

Repeat the model training shown above for another 100 iterations and report the coefficients.

In [ ]:
```
# Code block for Question 3
```

Authors

Karthik Muthuraman

Other Contributors

Jerome Nilmeier

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-07-14	0.4	Lakshmi Holla	Added code for pyspark
2021-12-22	0.3	Lakshmi Holla	Made changes in scaling
2021-08-05	0.2	Azim	Beta launch
2021-07-01	0.1	Karthik	Initial Draft