Machine Learning

As organizations create more diverse and more user-focused data products and services, there is a growing need for machine learning, which can be used to develop personalizations, recommendations, and predictive insights. The Apache Spark machine learning library (MLlib) allows data scientists to focus on their data problems and models instead of solving the complexities surrounding distributed data (such as infrastructure, configurations, and so on).

# Load sample data

# Use the Spark CSV datasource with options specifying:

# - First line of file is a header

# - Automatically infer the schema of the data

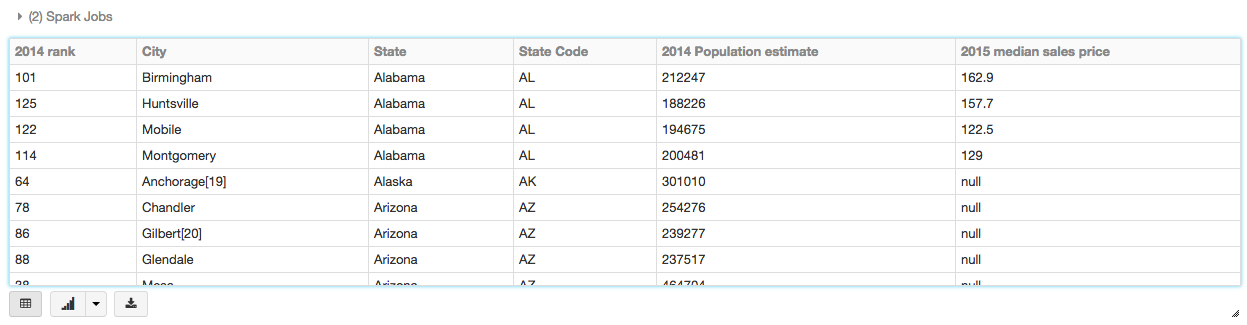
data = spark.read.csv("/databricks-datasets/samples/population-vs-price/data\_geo.csv", header="true", inferSchema="true")

data.cache() # Cache data for faster reuse

data.count()

To view this data in a tabular format, instead of exporting this data to a third-party tool, you can use the display() command in your Databricks notebook.

display(data)



# Prepare and visualize data for ML algorithms

In supervised learning—-such as a regression algorithm—-you typically define a label and a set of features. In this linear regression example, the label is the 2015 median sales price and the feature is the 2014 population estimate. That is, you use the feature (population) to predict the label (sales price).

Drop rows with missing values

data = data.dropna() # drop rows with missing values

data.count()

Rename the feature and label columns, replacing spaces with **\_**.

**from pyspark.sql.functions import col**

**exprs = [col(column).alias(column.replace(' ', '\_')) for column in data.columns]**

**vdata = data.select(\*exprs).selectExpr("2014\_Population\_estimate as population", "2015\_median\_sales\_price as label")**

**display(vdata)**

Select and vectorize the population feature column:

**from** pyspark.ml **import** Pipeline

**from** pyspark.ml.feature **import** VectorAssembler

vdata = data.select(\*exprs).selectExpr("2014\_Population\_estimate as population", "2015\_median\_sales\_price as label")

stages = []

assembler = VectorAssembler(inputCols=["population"], outputCol="features")

stages += [assembler]

pipeline = Pipeline(stages=stages)

pipelineModel = pipeline.fit(vdata)

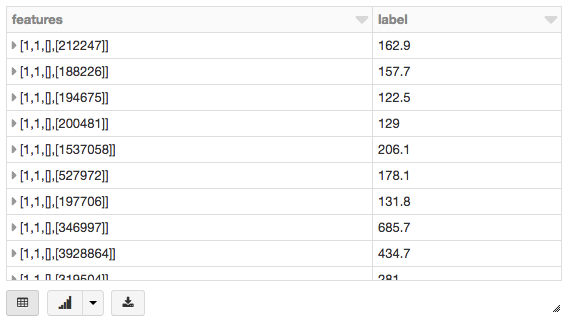
dataset = pipelineModel.transform(vdata)

# Keep relevant columns

selectedcols = ["features", "label"]

Display the selected columns:

display(dataset.select(selectedcols))



# Run the linear regression model

This section runs two different linear regression models using different regularization parameters to determine how well either of these two models predict the sales price (label) based on the population (feature).

**Goal:**

Predict y = 2015 Median Housing Price

Using feature x = 2014 Population Estimate

## Build the model

# Import LinearRegression class

**from** pyspark.ml.regression **import** LinearRegression

# Define LinearRegression algorithm

lr = LinearRegression()

# Fit 2 models, using different regularization parameters

modelA = lr.fit(dataset, {lr.regParam:0.0})

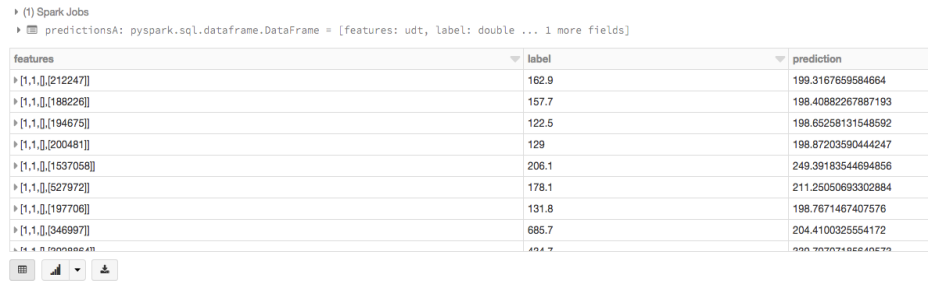
modelB = lr.fit(dataset, {lr.regParam:100.0})

Using the model, you can also make predictions by using the transform() function, which adds a new column of predictions. For example, the code below takes the first model (modelA) and shows you both the label (original sales price) and prediction (predicted sales price) based on the features (population).

# Make predictions

predictionsA = modelA.transform(dataset)

display(predictionsA)



# Scatterplot of the data using ggplot

**import** numpy **as** np

**from** pandas **import** \*

**from** ggplot **import** \*

x = dataset.rdd.map(**lambda** p: (p.features[0])).collect()

y = dataset.rdd.map(**lambda** p: (p.label)).collect()

pydf = DataFrame({'pop':x,'price':y})

p = ggplot(pydf, aes('pop','price')) + \

geom\_point(color='blue') + \

scale\_x\_log10() + scale\_y\_log10()

display(p)

# Evaluate the model

To evaluate the regression analysis, calculate the root mean square error using the RegressionEvaluator. Here is the Python code for evaluating the two models and their output.

Predicted vs. True label

Prediction For Model A

**from** pyspark.ml.evaluation **import** RegressionEvaluator

evaluator = RegressionEvaluator(metricName="rmse")

RMSE = evaluator.evaluate(predictionsA)

**print**("ModelA: Root Mean Squared Error = " + str(RMSE))

# ModelA: Root Mean Squared Error = 128.602026843

Prediction For Model B

predictionsB = modelB.transform(dataset)

RMSE = evaluator.evaluate(predictionsB)

**print**("ModelB: Root Mean Squared Error = " + str(RMSE))

# ModelB: Root Mean Squared Error = 129.496300193

# Plot residuals versus fitted values

display(modelA,dataset)

# Visualize the model

As is typical for many machine learning algorithms, you want to visualize the scatterplot. Since Azure Databricks supports [pandas](https://pandas.pydata.org/) and [ggplot](https://docs.azuredatabricks.net/user-guide/visualizations/matplotlib-and-ggplot.html#matplotlib-and-ggplot), the code below creates a linear regression plot using pandas DataFrame (pydf) and ggplot to display the scatterplot and the two regression models.

**Linear Regression Plots**

# Import numpy, pandas, and ggplot

**import** numpy **as** np

**from** pandas **import** \*

**from** ggplot **import** \*

# Create Python DataFrame

pop = dataset.rdd.map(**lambda** p: (p.features[0])).collect()

price = dataset.rdd.map(**lambda** p: (p.label)).collect()

predA = predictionsA.select("prediction").rdd.map(**lambda** r: r[0]).collect()

predB = predictionsB.select("prediction").rdd.map(**lambda** r: r[0]).collect()

# Create a pandas DataFrame

pydf = DataFrame({'pop':pop,'price':price,'predA':predA, 'predB':predB})

**ggplot figure**

# Visualizing the Model

# Create scatter plot and two regression models (scaling exponential) using ggplot

p = ggplot(pydf, aes('pop','price')) +

geom\_point(color='blue') +

geom\_line(pydf, aes('pop','predA'), color='red') +

geom\_line(pydf, aes('pop','predB'), color='green') +

scale\_x\_log10() + scale\_y\_log10()

display(p)

