Spark MLlib
Exercise 3

Today you'll learn about Spark MLlib
How to perform operations and train learners using Spark MLlib
(Sources: https://spark.apache.org/;
Spark: The Definitive Guide by Bill Chambers, Matei Zaharia)
https://spark.apache.org/docs/latest/ml-classification-regression.html
NB. Text and images are directly copied from these sources.

In this exercise, we will cover a basic overview of Spark's advanced analytics capabilities. The term advanced analytics refers to a variety of techniques, which are acquired to solve the core problem of getting insights, predictions, and recommendations based on raw data. It includes:

- Building supervised/unsupervised learning model to predict target variable/label
- From that, build a follow-up recommendation engine to suggest/give recommendations to users
- Others: data clustering, topic modeling, graph, finding patterns, etc.,

Advanced Analytics Process:

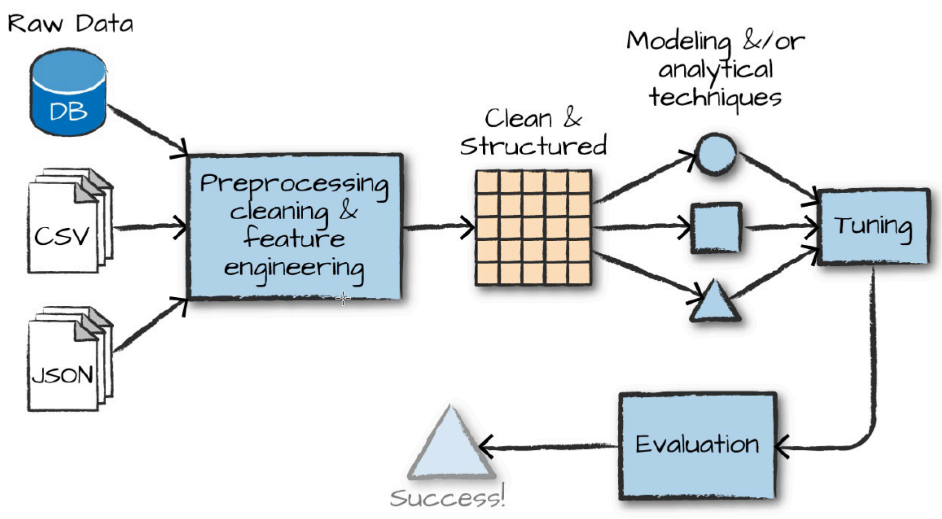

Raw Data
DB
CSV
JSON
Preprocessing cleaning & feature engineering
Clean & Structured
Modeling &/or analytical techniques
Tuning
Evaluation
Success!

Figure source: Spark: The Definitive Guide by Bill Chambers, Matei Zaharia

The overall process involves, the following steps (with some variation):

- Gathering and collecting the relevant data for your task.
- Cleaning and inspecting the data to better understand it.
- Performing feature engineering to allow the algorithm to leverage the data in a suitable

form (e.g., converting the data to numerical vectors).

- Using a portion of this data as a training set to train one or more algorithms to generate some candidate models.
- Evaluating and comparing models against your success criteria by objectively measuring results on a subset of the same data that was not used for training. This allows you to better understand how your model may perform in the wild.
- Leveraging the insights from the above process and/or using the model to make predictions, detect abnormalities or solve more general business challenges.

What is MLlib?
MLlib is a package, built on and included in Spark, that provides interfaces for gathering and cleaning data, feature engineering and feature selection, training and tuning large-scale supervised and unsupervised machine learning models, and using those models in production.

Why to use MLlib?
First, you want to leverage Spark for preprocessing and feature generation to reduce the amount of time it might take to produce training and test sets from a large amount of data. Then you might leverage single-machine learning libraries to train on those given data sets. Second, when your input data or model size becomes too difficult or inconvenient to put on one machine, use Spark to do the heavy lifting. Spark makes distributed machine learning very simple.

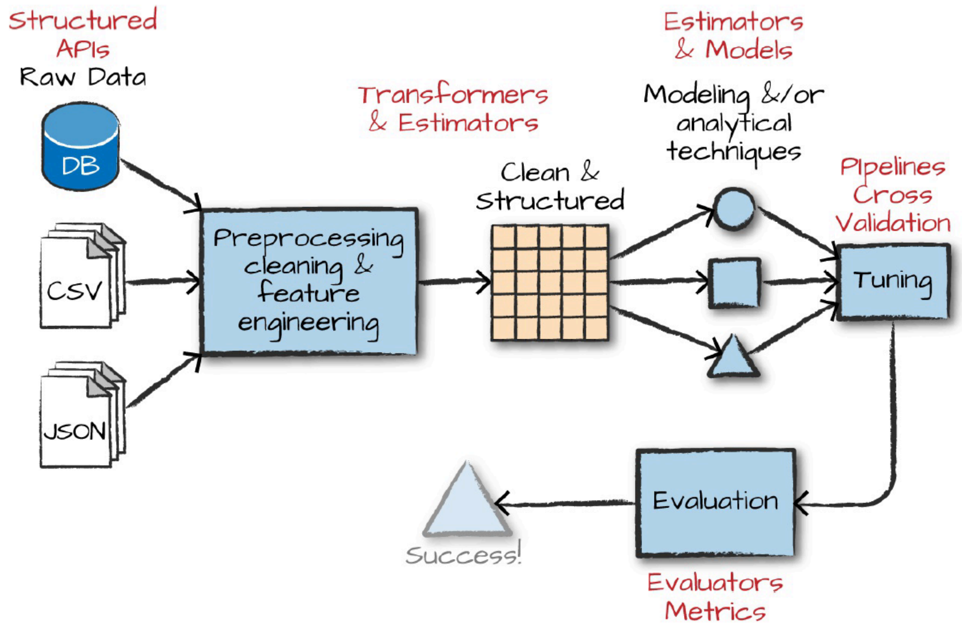

Structured APIs
Raw Data
DB
CSV
JSON
Transformers & Estimators
Preprocessing cleaning & feature engineering
Clean & Structured
Estimators & Models
Modeling &/or analytical techniques
Pipelines Cross Validation
Tuning
Evaluation
Evaluators Metrics
Success!

Figure source: Spark: The Definitive Guide by Bill Chambers, Matei Zaharia

In MLlib there are several fundamental “structural” types: transformers, estimators, and pipelines.

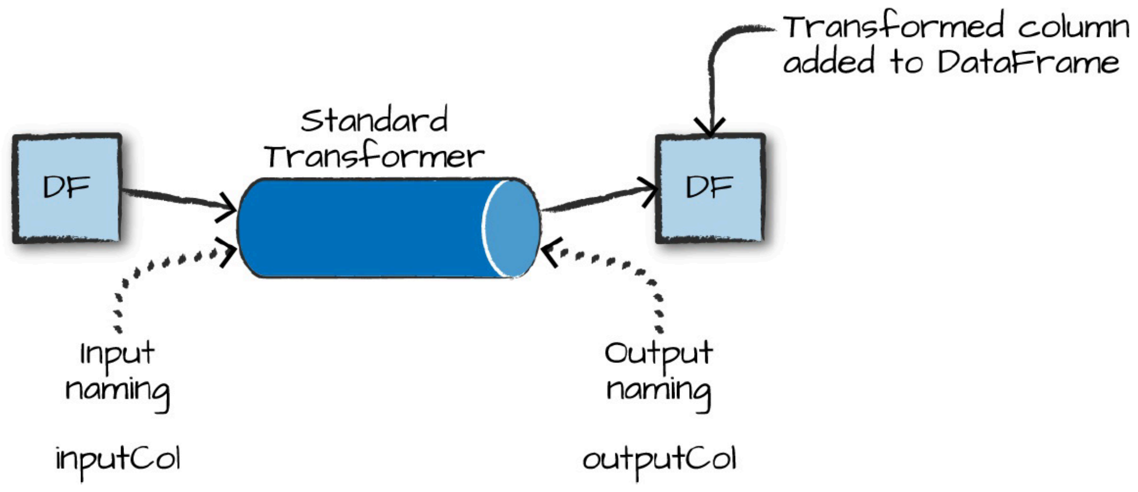


Figure source: Spark: The Definitive Guide by Bill Chambers, Matei Zaharia

Transformer: an algorithm that transforms the data in some way, making one DataFrame into another.
(E.g. parse data, create new interaction variable (from 2+ other variables), normalize data, or simply convert the datatype to feed as input into a model - converts string categorical variables into numerical values, etc.,)

Estimator: an algorithm that can fit or train on a DataFrame to produce a Transformer.
(E.g., a machine learning algorithm or an indexer)

Pipeline: chains multiple Transformers and Estimators together to create a workflow.

Let's proceed to a practical example. Today, we will try to predict atmospheric temperature using different independent observations:

Metainformation:
date: - Date and Time (utc)
"rain: - Precipitation Amount (mm) "
"temp: - Air Temperature (C)"
wetb: - Wet Bulb Temperature (C)
dewpt: - Dew Point Temperature (C)
"vappr: - Vapour Pressure (hPa) "
rhum: - Relative Humidity (%)
msl: - Mean Sea Level Pressure (hPa)
wdsp: - Mean Wind Speed (kt)
wddir: - Predominant Wind Direction (deg)

1) Reading the data
We will use the same data from previous exercises containing observations of weather conditions. However, you are free to use any data, as you like.

```
#Reference:
https://spark.apache.org/docs/latest/ml-classification-regression.htm
l#decision-tree-regression

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
pd.set_option('display.max_rows', 20000)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', 500)
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import sys
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .master("local") \
    .appName("Exercise3")\
    .getOrCreate()

df = spark.read.csv('/home/jovyan/work/exampleData.csv', header =
True)
```

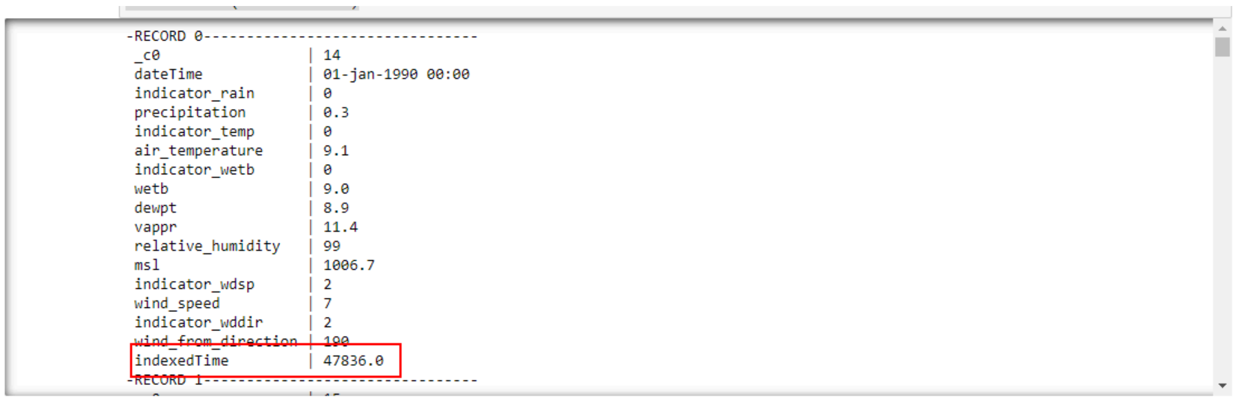
2) Exploring schema
Now, we will check the data types of parameters present in the read file. Also, if there is categorical data, how should we proceed with it?

```
df.dtypes
```

3) Indexing the data with StringIndexer
StringIndexer encodes a string column to a column with indices. In this data, we have a datetime parameter with actual string values. So to make the learner (model) understand the string values, we will use a string indexer.

```
from pyspark.ml.feature import StringIndexer

indexer = StringIndexer(inputCol="dateTime" ,outputCol="indexedTime")
indexed = indexer.fit(df).transform(df)
indexed.show(vertical=True)
```



```
-RECORD 0-------------------------------
 _c0                 | 14
 dateTime            | 01-jan-1990 00:00
 indicator_rain      | 0
 precipitation       | 0.3
 indicator_temp      | 0
 air_temperature     | 9.1
 indicator_wetb      | 0
 wetb                | 9.0
 dewpt               | 8.9
 vappr               | 11.4
 relative_humidity   | 99
 msl                 | 1006.7
 indicator_wdsp      | 2
 wind_speed          | 7
 indicator_wddir     | 2
 wind_from_direction | 190
 indexedTime         | 47836.0
-RECORD 1-------------------------------
```

Q: Can we use any other alternative approach than StringIndexer? If yes, what is the main advantage?

4) Change and select the types of parameters to vectorize the data for further training

For today's exercise, we will use independent variables as DateTime, relative_humidity, wind_speed, msl parameters to predict the target variable - air_temperature.

```
# Change the data types of parameters to correct ones to vectorize
parsed_data = indexed.selectExpr("cast(dateTime as string)dateTime",
                                 "cast(relative_humidity as float)
relative_humidity","cast(wind_speed as float) wind_speed",
                                          "cast(indexedTime as double)
indexedTime","cast(air_temperature as float) label","cast(msl as
float) msl")
```

5) Vectorize data and generate features using Spark MLlib's Vector Assembler

```
from pyspark.ml.feature import VectorAssembler
vectorAssembler = VectorAssembler(inputCols = ['relative_humidity',
'wind_speed', 'indexedTime', 'msl'], handleInvalid="skip",outputCol =
'features')
vectorized_df = vectorAssembler.transform(parsed_data)
dataset = vectorized_df.select("features", "label")
```

Q: What does parameter → HandleInvalid do in the vector assembler?
Q: Find alternative approaches to do the same job

5) Splitting data

```
# Split the data into training and test sets (30% held out for
testing)
(trainingData, testData) = dataset.randomSplit([0.7, 0.3])
```

6) Creating a learner using Spark ML

```
from pyspark.ml.regression import GBTRegressor
gbt = GBTRegressor(featuresCol="features", maxIter=30, maxDepth = 11)

# Train model.  This also runs the indexer.
model = gbt.fit(trainingData)

# Make predictions.
prediction = model.transform(testData)
```

Q: Can we opt for k-fold cross validation in pyspark? How?
Q: Try to perform weather prediction using DateTime separately as Day(Mon, Tue etc.), Time (in hours , 11:00, 12:00 etc.)

7) Try Spark ML Pipelines

MLlib standardizes APIs for machine learning algorithms to make it easier to combine multiple algorithms into a single pipeline, or workflow. This section covers the key concepts introduced by the Pipelines API, where the pipeline concept is mostly inspired by the scikit-learn project.

```
from pyspark.ml.regression import GBTRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline


featureIndexer =VectorIndexer(inputCol="features",
outputCol="indexedFeatures", handleInvalid="skip").fit(dataset)

# Split the data into training and test sets (30% held out for
testing)
(trainingData, testData) = parsed_data.randomSplit([0.7, 0.3])
# Train a GBT model.
gbt = GBTRegressor(featuresCol="indexedFeatures", maxIter=30,
maxDepth = 11)

# Chain indexer and GBT in a Pipeline
```

```
pipeline = Pipeline(stages=[featureIndexer, gbt])

# Train model.  This also runs the indexer.
model_gbt = pipeline.fit(trainingData)

# Make predictions.
preds = model_gbt.transform(testData)

# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="r2")
rmse = evaluator.evaluate(preds)
print("R Squared on test data = %g" % rmse)
```

8) Evaluate using different metrics

MAE:

```
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="mae")
mae= evaluator.evaluate(preds)
print("MAE on test data = %g" % mae)
```

RMSE:

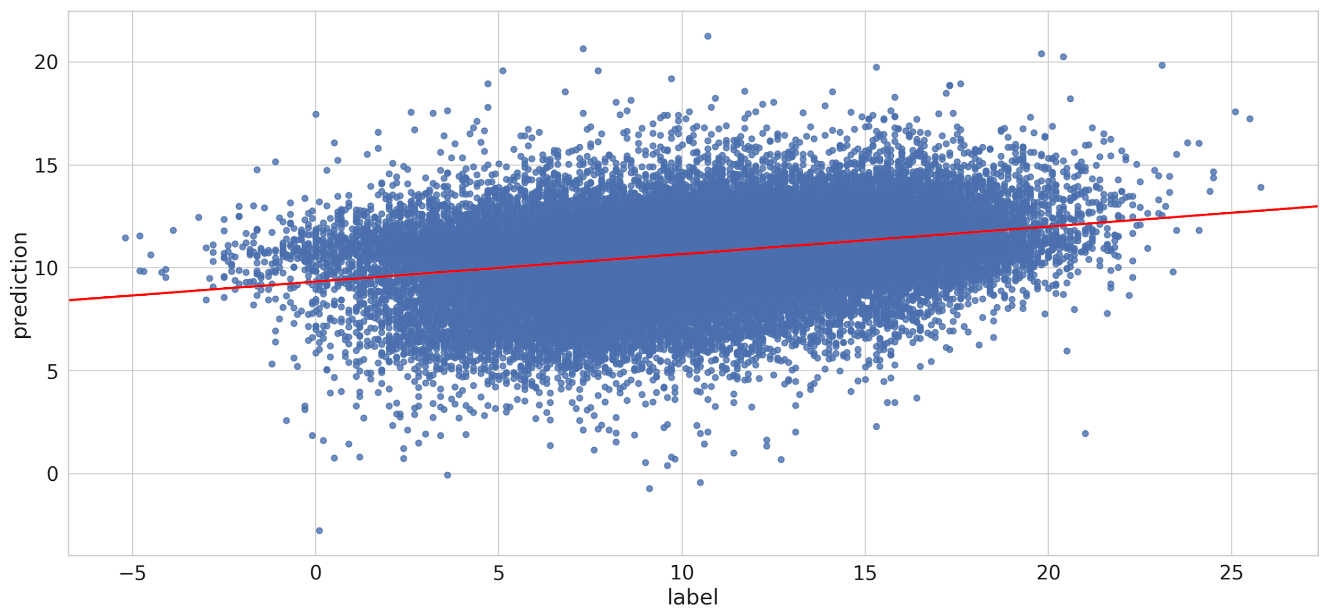
```
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(preds)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

9) Plotting results

```
pred_pd = preds.toPandas()
import seaborn
from scipy.stats import *
seaborn.set(style="whitegrid", font_scale = 1.8)

fig, ax = plt.subplots()
seaborn.set(color_codes=True)
seaborn.set(rc={'figure.figsize':(20, 10)})

seaborn.regplot(x="label",  y="prediction",  fit_reg=False,  ax=ax,data
=pred_pd,scatter_kws={"color": "b"});
seaborn.regplot(x="label",  y="prediction",scatter=False,  ax=ax,  data
=pred_pd, line_kws={"color": "red"});
```



Questions for the Exercise 3 report

Please answer the following questions. Remember to put your student ID and your name.

1. Statement: *"A big benefit of using ML Pipelines is hyperparameter optimization"*.
   Check Spark docs and explain why is that. Give us **<u>your</u>** justification for this statement. (Provide the answer in your own words).

<u>NOTE:</u>

1. Always return in **<u>PDF format.</u>**
2. Try to give **brief and concise** answers *(just enough to solve the question).*
3. **<u>Do not</u>** copy and share your answer with others.
4. Any questions/concerns need to be **<u>sent/cc-ed to all TA members</u>**, not to one specific person.