BA-BEAD Big Data Engineering for Analytics

Workshop Series

Spark Machine Learning

Contents

BA-BEAD Big Data Engineering for Analytics 1
List of python scripts and datasets 3
Introduction 3
Machine Learning 3
Supervised Learning 4
Key Steps of Supervised Learning 4
Linear Regression 4
Classification 12
Unsupervised Learning 18
Clustering 18
Summary 23

List of python scripts and datasets

Scripts	Datasets
lr_sample.py	sample_linear_regression_data.txt
lr_ecommerce.py	Ecommerce_Customers.csv
lgr_sample.py	sample_libsvm_data.txt
lgr_titanic.py	titanic.csv
kmeans_sample.py	sample_kmeans_data.txt
kmeans_seed.py	seeds_dataset.csv

Introduction

Machine learning is a field of study that gives computers the ability to learn without being explicitly programmed. There are many successful applications of machine learning, including systems that analyse past sales data to predict customer behaviour, optimizing robot behaviour so that a task can be completed using minimum resources and extracting knowledge from bio-informatics data. With the advantage of big data, maintaining large collections of data is one thing, but extracting useful information from these collections is even more challenging. The ML system should be able to scale on high volumes of data, the accuracy of the models built would also have to be quite high as the training takes place on large data.

Big data and machine learning take place in three steps: collecting, analysing, and predicting. For this purpose, the Spark ecosystem supports a wide range of workloads including batch applications, iterative algorithms, interactive queries, and stream processing. The Spark MLlib component offers a variety of scalable ML algorithms.

The objective of the workshop is to familiarise the learner with Spark Machine Learning concepts. In this workshop, you will work with Spark Machine Learning library.

In this workshop, we will learn how to performing exploratory data analysis by using PySpark. PySpark is a Python API for Spark. PySpark features quite a few libraries for writing efficient programs. MLlib is a wrapper over the PySpark and it is Spark's ML library. The machine-learning API provided by the MLlib library is quite easy to use. MLlib supports many machine-learning algorithms for classification, regression, clustering, collaborative filtering, dimensionality reduction, and underlying optimization primitives.

In previous workshop, we have discussed on how to calculate basic statistics in PyCharm. In this workshop, we will discuss on how to do regression, classification and clustering by using PySpark in PyCharm.

Machine Learning

Spark's ML is mainly designed for Supervised and Unsupervised Learning tasks, with most of its algorithms falling under those two categories. We discuss them in more details.

Supervised Learning

Supervised learning deals with training algorithms with labelled data, inputs for which the outcome or target variables are known. It then predicts the outcome/target with the trained model for unseen future data. For example, historical e-mail data will have individual e-mails marked as ham or spam; this data is then used for training a model that can predict future e-mails as ham or spam. Supervised learning problems can be broadly divided into two major areas: classification and regression. Classification deals with predicting categorical variables or classes; for example, whether an e-mail is ham or spam or whether a customer is going to renew a subscription or not in a post-paid telecom subscription. This target variable is discrete and has a predefined set of values. Regression deals with a target variable, which is continuous. For example, when we need to predict house prices, the target variable price is continuous and doesn't have a predefined set of values.

Key Steps of Supervised Learning

1. Determine the objective
2. Decide the training data
3. Cleaning the training dataset
4. Feature extraction
5. Training the models
6. Validation
7. Evaluation of trained model

Linear Regression

Regression analysis is a type of predictive modelling technique which investigates the relationship between variables. This is widely used for forecasting, time series modelling and finding the effect of relationships between the variables. In this, we try to fit a curve/line for the data points such that the differences between the distances of data points from the curve or line is minimized. Linear regression is the mostly frequently used technique. In this technique, the dependent variable is continuous, independent variables can be continuous or discrete, and the nature of the regression line is linear.

The equation which is used to predict the value of the target variable is based on the following predictor variables:

$$y = aX + E$$

where y = target variable, X = input variable, a = regression coefficient and E = the error term

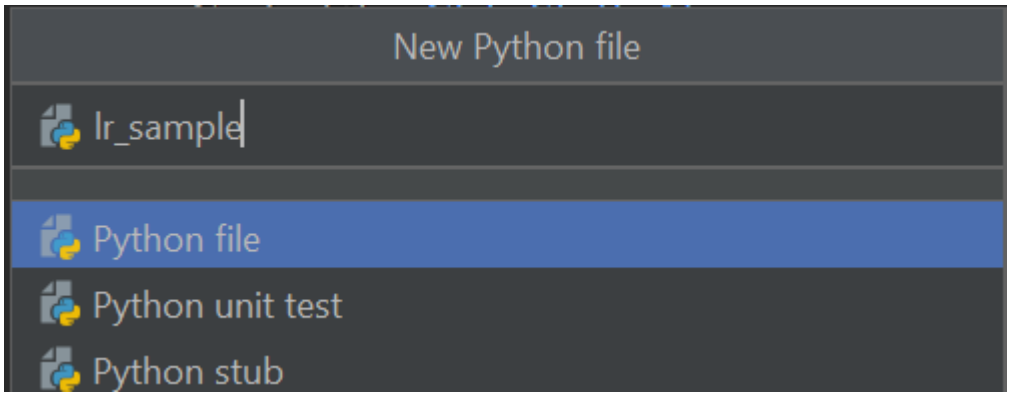
Regression techniques are driven by three metrics-the number of independent variables, the type of dependent variables, and the shape of the regression line. We can also evaluate the model performance using the metric R-square. In this session, we'll see how to apply regression analysis on different data sets.

Linear Regression on sample data set

In this example, we will cover how to implement linear regression using PySpark ML library. The dataset is "*sample_linear_regression_data.txt*".

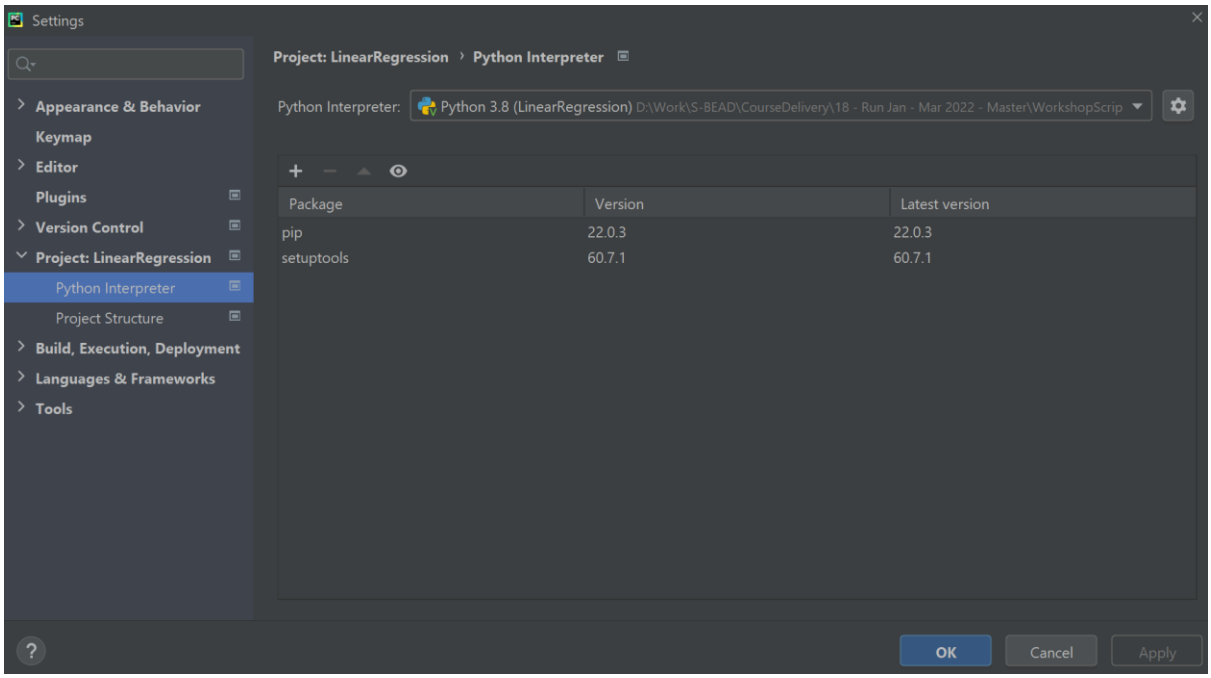
Create a new project named as “LinearRegression” in PyCharm.

Create a new Python file named as “lr_sample”

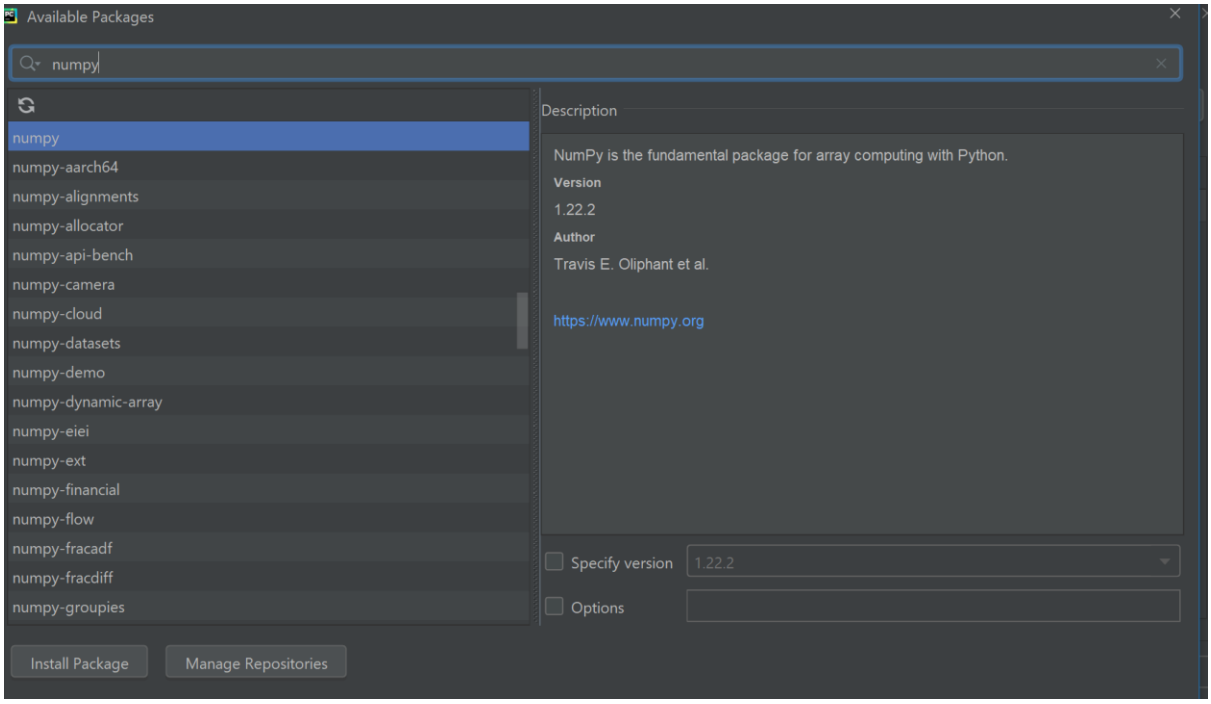


Remember to add content root into this project.

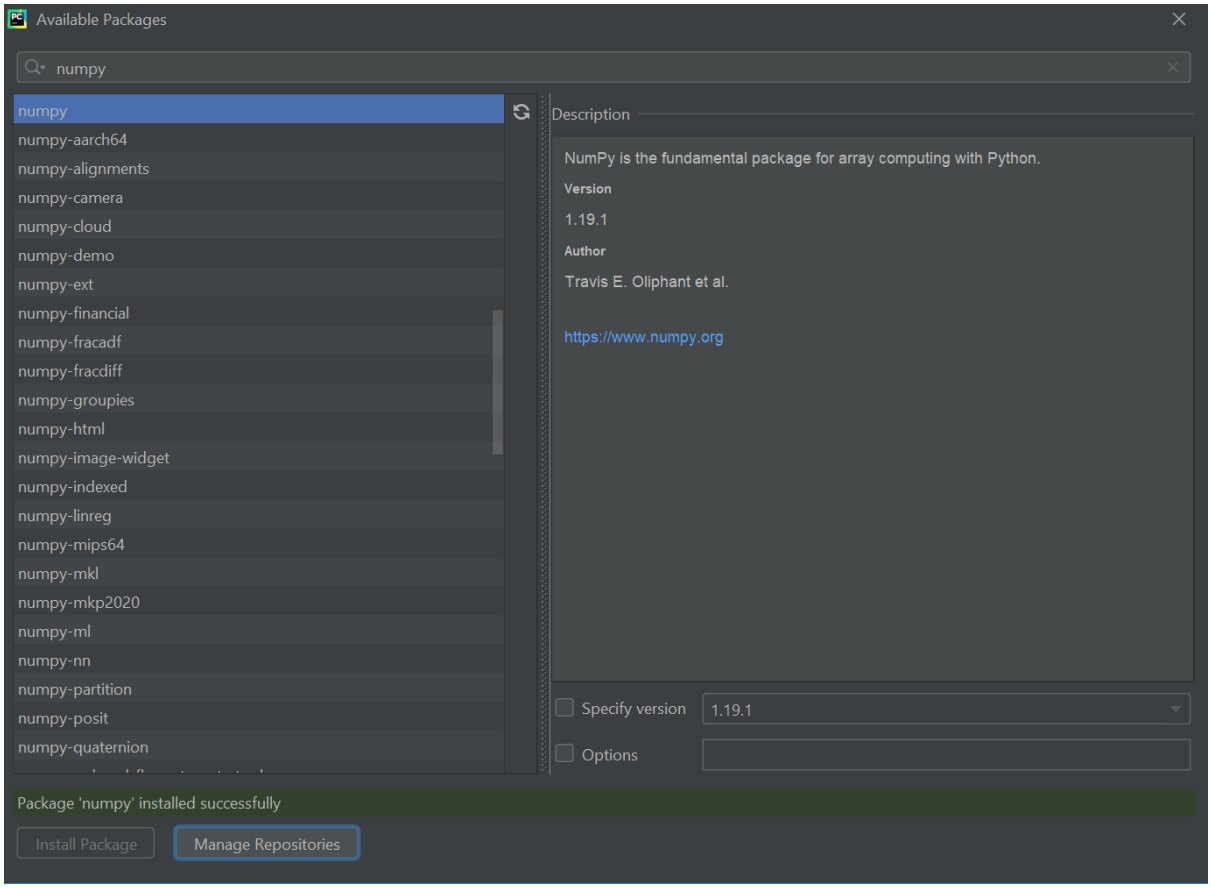
As we will use module “numpy” in this example, we need to install it as well. Go to File -> setting -> Python Interpreter -> “+” -> numpy -> install numpy



Search for “numpy” and click “Install Package”:



After installation, you can see the message "Package 'numpy' installed successfully" at the bottom left.



Create a Python file named as "lr_sample.py":

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('lr_sample').getOrCreate()
```

```
from pyspark.ml.regression import LinearRegression

# Load data
all_data =
spark.read.format("libsvm").option("numFeatures","10").load("sample_linear_regre
ssion_data.txt")
# Split into training data and test data
train_data, test_data = all_data.randomSplit([0.7,0.3])
train_data.show()
test_data.show()
unlabeled_data = test_data.select("features")
unlabeled_data.show()
# These are the default values for the featuresCol, labelCol, predictionCol
lr =
LinearRegression(featuresCol='features',labelCol='label',predictionCol='predicti
on')
# Fit the model
lr_model = lr.fit(train_data)
# Print the coefficients and intercept training data
print("Coefficients: {}".format(str(lr_model.coefficients)))
print("Intercept: {}".format(str(lr_model.intercept)))
# Testing result
test_result = lr_model.evaluate(test_data)
test_result.residuals.show()
print("RMSE: {}".format(test_result.rootMeanSquaredError))

# Prediction
predictions = lr_model.transform(unlabeled_data)
predictions.show()
```

The sample outputs are as follows:

```
Coefficients:                    [-0.03685144892064408,0.5013637840196282,-
0.2994513331115969,1.7121448667038264,0.7823169785289079,1.116224039257927,
-0.20259910893033434,-0.28904721671396433,-
0.5523070755594424,0.3403165673916782]

Intercept: -0.1407464535044517

+-------------------+

|          residuals|

+-------------------+

|-20.307145447701824|

| -16.54596876213889|

|-19.905032588503637|

| -17.14231542253131|

|-18.662398888882144|

|-15.761766896505424|

| -16.55381860430357|

|-16.050547026372247|

|-15.647411905250479|
```

```
|-12.276034745968113|
|-11.554640868465352|
|-13.433850720736965|
| -15.49378714925767|
|-13.190431962391997|
|-12.528843689567609|
| -13.45004874194758|
|-11.377177809330743|
|-11.101695996367665|
|-10.434561844574102|
|-11.415386419518466|
+-------------------+
only showing top 20 rows

RMSE: 10.37798614035672
+--------------------+--------------------+
|            features|          prediction|
+--------------------+--------------------+
|(10,[0,1,2,3,4,5,...|  0.5243826580872867|
|(10,[0,1,2,3,4,5,...|  -2.622323860824872|
|(10,[0,1,2,3,4,5,...|  1.0591101156050553|
|(10,[0,1,2,3,4,5,...|  -0.661310766133206|
|(10,[0,1,2,3,4,5,...|  1.2337243179426398|
|(10,[0,1,2,3,4,5,...| -1.2647253677041244|
|(10,[0,1,2,3,4,5,...|  0.8217303320643229|
|(10,[0,1,2,3,4,5,...|  0.7157795464499044|
|(10,[0,1,2,3,4,5,...|  1.3184333961750356|
|(10,[0,1,2,3,4,5,...| -1.1445600299226442|
|(10,[0,1,2,3,4,5,...| -1.5986947379001792|
|(10,[0,1,2,3,4,5,...| 0.45600199534486086|
|(10,[0,1,2,3,4,5,...|    2.57156404588725|
|(10,[0,1,2,3,4,5,...|  0.6318561735358089|
|(10,[0,1,2,3,4,5,...|0.028069904212553676|
|(10,[0,1,2,3,4,5,...|   0.958606664401166|
|(10,[0,1,2,3,4,5,...| -1.1021024021207544|
|(10,[0,1,2,3,4,5,...| -1.3659603846651946|
|(10,[0,1,2,3,4,5,...|  -1.422788520855323|
|(10,[0,1,2,3,4,5,...|  0.8152560776094325|
```

```
+--------------------+--------------------+
```

Linear Regression on Real-world dataset

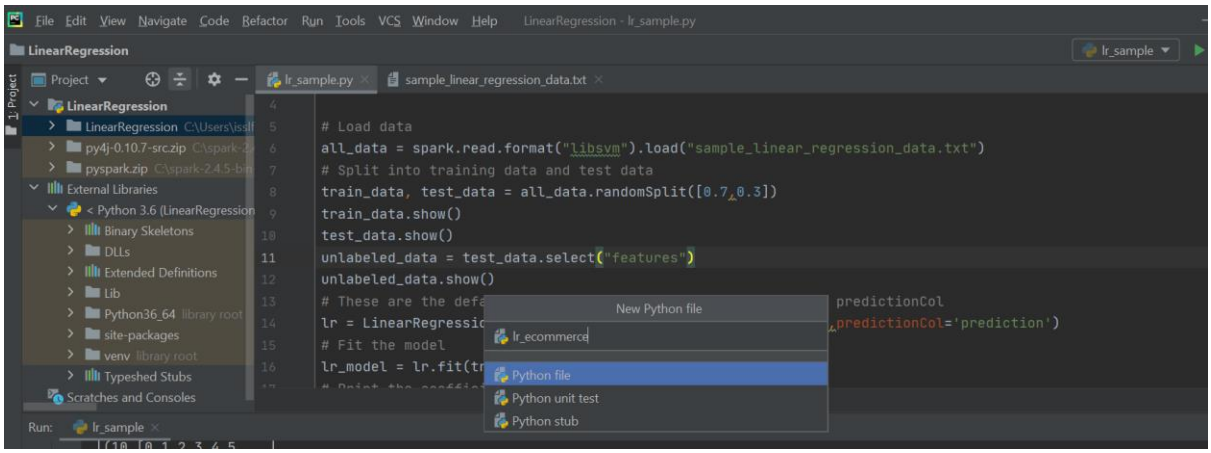
The dataset is *Ecommerce_Customers.csv*. This dataset is for a company websites and mobile app.

Basically what we do here is examine a dataset with Ecommerce Customer Data for a company's website and mobile app. Then we want to see if we can build a regression model that will predict the customer's yearly spend on the company's product.

The attributes are as follows:

Variables	Description
Email	User's email address
Address	User's address
Avatar	User's Avatar
Avg Session Length	Average length user spent on the session
Time on App	Time user spent on this App
Time on Website	Time user spent on the website
Length of Membership	Length of membership
Yearly Amount Spent	How much user spent yearly

Add a new Python file named as "lr_ecommerce" under the project "LinearRegression":



Create a Python file named as "lr_ecommerce.py":

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('lr_ecommerce').getOrCreate()
from pyspark.ml.regression import LinearRegression
# Load dataset into dataframe
data = spark.read.csv("Ecommerce_Customers.csv",inferSchema=True,header=True)
# Print the Schema of the dataframe
data.printSchema()
data.show()
data.head()
for item in data.head():
    print(item)

# Setting up DataFrame for Machine Learning
from pyspark.ml.linalg import Vector
from pyspark.ml.feature import VectorAssembler
data.columns
# We need to change the data into the form of two columns ("label", "features"),
we will only use the numeric columns as features
```

```
assembler = VectorAssembler(inputCols=["Avg Session Length",
"Time on App",
"Time on Website",
"Length of Membership"],
outputCol="features")
output = assembler.transform(data)
output.select("features").show()
output.show()
# Final dataset with two columns ("features", "Yearly Amount Spent")
final_data = output.select("features","Yearly Amount Spent")
# Split into training and testing datasets
train_data, test_data = final_data.randomSplit([0.7,0.3])
train_data.describe().show()
test_data.describe().show()
# Create a linear regression model object
lr = LinearRegression(labelCol='Yearly Amount Spent')
# Fit the model to the data and call this model lrModel
lrModel = lr.fit(train_data,)
# print the coefficients and intercept for linear regression
print("Coefficients: {} Intercept {}".format(lrModel.coefficients,
lrModel.intercept))

# Evaluate the model on a test dataset
test_result = lrModel.evaluate(test_data)
test_result.residuals.show()
print("RMSE:{}".format(test_result.rootMeanSquaredError))
print("MSE: {}".format(test_result.meanSquaredError))
```

Outputs are as follows:

`data.printSchema()`:

```
root
 |-- Email: string (nullable = true)
 |-- Address: string (nullable = true)
 |-- Avatar: string (nullable = true)
 |-- Avg Session Length: double (nullable = true)
 |-- Time on App: double (nullable = true)
 |-- Time on Website: double (nullable = true)
 |-- Length of Membership: double (nullable = true)
 |-- Yearly Amount Spent: double (nullable = true)
```

`output.select("features").show()`:

```
+--------------------+
|            features|
+--------------------+
|[34.4972677251122...|
|[31.9262720263601...|
|[33.0009147556426...|
|[34.3055566297555...|
|[33.3306725236463...|
|[33.8710378793419...|
|[32.0215955013870...|
|[32.7391429383803...|
|[33.9877728956856...|
|[31.9365486184489...|
|[33.9925727749537...|
|[33.8793608248049...|
|[29.5324289670579...|
|[33.1903340437226...|
|[32.3879758531538...|
|[30.7377203726281...|
|[32.1253868972878...|
```

```
|[32.3388993230671...|
|[32.1878120459321...|
|[32.6178560628234...|
+--------------------+
only showing top 20 rows
```

train_data.describe().show():

```
+-------+------------------+
|summary|Yearly Amount Spent|
+-------+------------------+
|  count|               374|
|   mean| 494.77136952135584|
| stddev|  79.32347347332397|
|    min| 256.67058229005585|
|    max|  765.5184619388372|
+-------+------------------+
```

test_data.describe().show()

```
+-------+------------------+
|summary|Yearly Amount Spent|
+-------+------------------+
|  count|               126|
|   mean|  512.7978327643508|
| stddev|  78.05150254141621|
|    min|  275.9184206503857|
|    max|  689.2356997616951|
+-------+------------------+
```

test_results.residuals.show():

```
+--------------------+
|           residuals|
+--------------------+
|  0.0211314098588673|
|   5.133420657904367|
| -22.148174773562573|
|-0.28200824770379995|
|  2.6981651070976795|
|  0.7429079096369264|
|  -3.455359493022627|
| -2.1941889835831034|
|   7.147573069494342|
|   1.206688448485238|
| -12.184920705516333|
|  0.9395486603143581|
|   7.893631818058566|
| -1.6946216546640471|
|  -9.987986847356694|
|  -6.281630483941342|
|  -9.000712269275198|
|  -18.16281648837412|
|  -9.055807893457029|
|    3.71115734739368|
+--------------------+
only showing top 20 rows
```

print("RMSE: {}".format(test_results.rootMeanSquaredError)):

print("MSE: {}".format(test_results.meanSquaredError)):

```
RMSE:10.034230320206236
MSE: 100.68577811894615
```

In this example, we use `VectorAssembler` to generate features. The detailed description of `VectorAssembler` is here: https://spark.apache.org/docs/latest/ml-features#vectorassembler

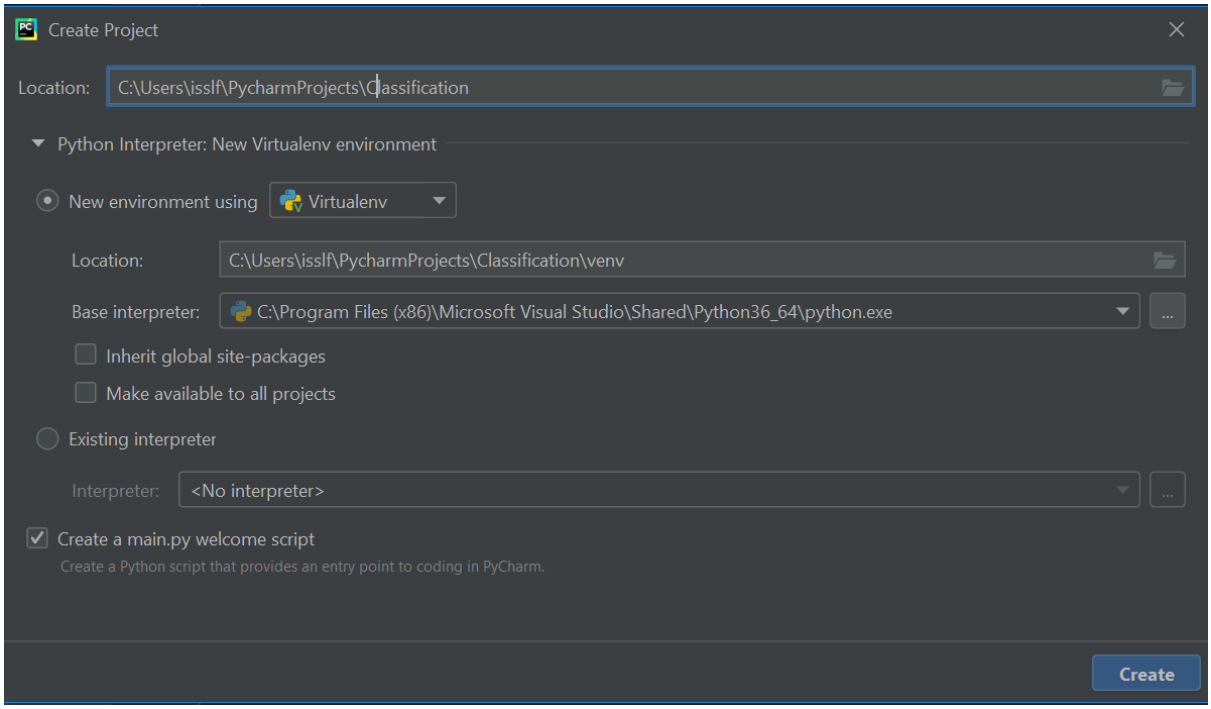
Classification

In the previous examples, we have seen regression problems where the output is a continuous value. But not all the output values are continuous. In order to predict discrete categories, we need to use classification. In this session, we will learn the logistic regression. Logistic regression allows us to solve classification problems, especially for binary classification, which has two classes 0 and 1.

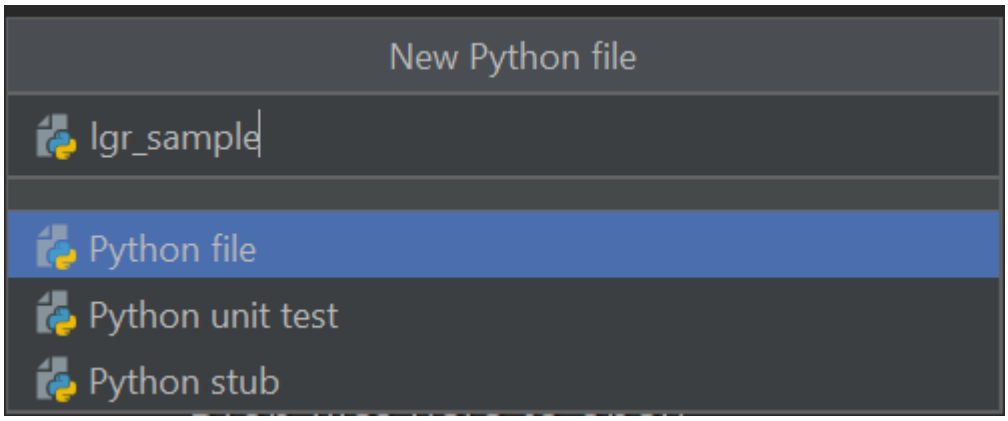
Logistic Regression on Sample Dataset

Let's see an example of how to run a logistic regression with PySpark. We will implement Logistic regression on a sample dataset "*sample_libsvm_data.txt*".

Create a new project named as "Classification" in PyCharm:



Create a new Python file named as "lgr_sample":



Before run the script, similar with the previous example, we need to do necessary settings. Such as add content roots and install numpy.

Create a Python file named as "lgr_sample.py":

```
from pyspark.sql import SparkSession
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import
BinaryClassificationEvaluator,MulticlassClassificationEvaluator


spark = SparkSession.builder.appName("logregdoc").getOrCreate()
# Load training data
data_set = spark.read.format("libsvm").option("numFeatures","
692").load("sample_libsvm_data.txt")
data_set.show()
# Split into training set and testing set
lr_train, lr_test = data_set.randomSplit([0.7,0.3])
# Create LogisticRegression model
lgr_model = LogisticRegression()
# fit model with training set
fitted_model = lgr_model.fit(lr_train)
# evaluate with testing set
prediction_and_labels = fitted_model.evaluate(lr_test)
prediction_and_labels.predictions.show()
prediction_and_labels.predictions.select("probability").show(10, False)
```

Outputs are as follows:

data_set.show():

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|(692,[127,128,129...|
|  1.0|(692,[158,159,160...|
|  1.0|(692,[124,125,126...|
|  1.0|(692,[152,153,154...|
|  1.0|(692,[151,152,153...|
|  0.0|(692,[129,130,131...|
|  1.0|(692,[158,159,160...|
|  1.0|(692,[99,100,101,...|
|  0.0|(692,[154,155,156...|
|  0.0|(692,[127,128,129...|
|  1.0|(692,[154,155,156...|
|  0.0|(692,[153,154,155...|
|  0.0|(692,[151,152,153...|
```

```
|  1.0|(692,[129,130,131...|
|  0.0|(692,[154,155,156...|
|  1.0|(692,[150,151,152...|
|  0.0|(692,[124,125,126...|
|  0.0|(692,[152,153,154...|
|  1.0|(692,[97,98,99,12...|
|  1.0|(692,[124,125,126...|
+-----+--------------------+
```

prediction_and_labels.predictions.show():

```
+-----+--------------------+--------------------+--------------------+----------+
|label|            features|       rawPrediction|         probability|prediction|
+-----+--------------------+--------------------+--------------------+----------+
|  0.0|(692,[123,124,125...|[42.0554835908259...|[1.0,5.4392064445...|       0.0|
|  0.0|(692,[124,125,126...|[45.6830068011262...|[1.0,1.4458466381...|       0.0|
|  0.0|(692,[124,125,126...|[28.2176588760151...|[0.99999999999944...|       0.0|
|  0.0|(692,[124,125,126...|[16.9137584089089...|[0.99999995487179...|       0.0|
|  0.0|(692,[124,125,126...|[22.8014307391275...|[0.99999999987484...|       0.0|
|  0.0|(692,[126,127,128...|[15.4436882161434...|[0.99999980371308...|       0.0|
|  0.0|(692,[126,127,128...|[28.3551575707258...|[0.99999999999951...|       0.0|
|  0.0|(692,[128,129,130...|[47.5585729973203...|[1.0,2.2160225737...|       0.0|
|  0.0|(692,[151,152,153...|[28.0756590015595...|[0.99999999999935...|       0.0|
|  0.0|(692,[152,153,154...|[12.4132829619027...|[0.99999593577313...|       0.0|
|  0.0|(692,[153,154,155...|[29.1853202173902...|[0.99999999999978...|       0.0|
|  0.0|(692,[234,235,237...|[5.62151590573852...|[0.99639390036061...|       0.0|
|  1.0|(692,[97,98,99,12...|[-17.196188583255...|[3.40243763658722...|       1.0|
|  1.0|(692,[119,120,121...|[-0.6293550948436...|[0.34765678247027...|       1.0|
|  1.0|(692,[124,125,126...|[-29.430316648781...|[1.65415376884232...|       1.0|
|  1.0|(692,[124,125,126...|[-21.620972945512...|[4.07502949221533...|       1.0|
|  1.0|(692,[125,126,127...|[-22.723022744416...|[1.35368186213756...|       1.0|
|  1.0|(692,[127,128,129...|[-29.568345824644...|[1.44088914418732...|       1.0|
|  1.0|(692,[127,128,154...|[-15.440050113760...|[1.97002324110452...|       1.0|
|  1.0|(692,[127,128,155...|[-23.766965185734...|[4.76582584627902...|       1.0|
+-----+--------------------+--------------------+--------------------+----------+
```

prediction_and_labels.predictions.select("probability").show(10, False)

```
+----------------------------------------+
|probability                             |
+----------------------------------------+
|[1.0,5.439206444535723E-19]             |
|[1.0,1.445846638169447E-20]             |
|[0.9999999999994438,5.561942028105065E-13]|
```

```
|[0.9999999548717975,4.512820246594277E-8] |
|[0.9999999998748403,1.251596818280533E-10]|
|[0.9999998037130883,1.9628691178709879E-7]|
|[0.9999999999995153,4.847429857481169E-13]|
|[1.0,2.216022573774801E-21]               |
|[0.999999999999359,6.410563697299821E-13] |
|[0.9999959357731348,4.064226865112208E-6] |
+------------------------------------------+
```

The output columns the output column names are: [label | features | rawPrediction | probability | prediction].

RawPrediction is typically the direct probability/confidence calculation. The Probability is the conditional probability for each class. For LogisticRegression, it uses the logistic formula:

class_k probability: 1/(1 + exp(-rawPrediction_k))

The class that have the highest probability in the end is the one predicted.

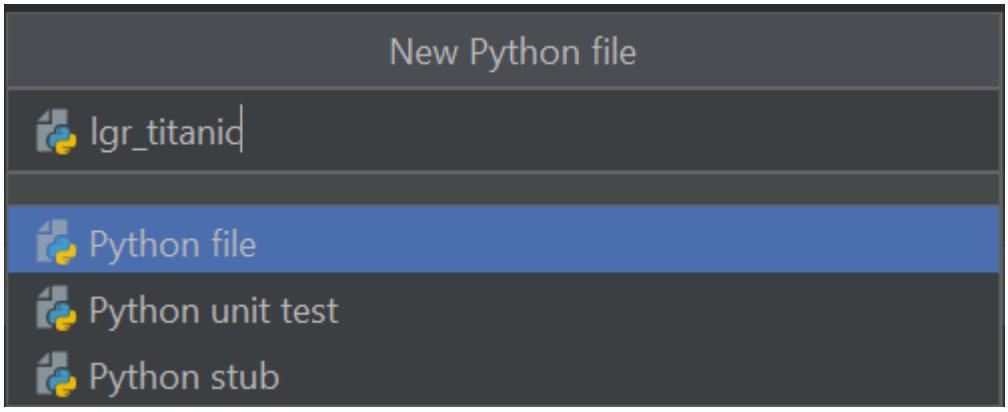
Logistic Regression on Titanic Dataset

The Titanic dataset is a common exercise for classification in machine learning; there are many examples of it online for other machine learning libraries. We will use it to attempt to predict what passengers survived the Titanic crash based solely on passenger's features. The dataset will also have a lot of missing information, so we will need to deal with that as well.

The dataset is *titanic.csv*. The attributes are as follows:

Variables	Description
PassengerId	Passenger identification number
Survived	Srivival (0 = No, 1= Yes)
Pclass	Passenger class
Name	Passenger name
Sex	Passenger gender
Age	Passenger age
SibSp	Number of siblings/Spouse aboard
Parch	Number of parents/Children aboard
Ticket	Ticket number
Fare	Passenger fare
Cabin	Cabin
Embarked	Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

Create a new Python file named as "lgr_titanic":



"lgr_titanic.py":

```
# Titanic logistic regression
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Titanic logistic
regression').getOrCreate()
# Load dataset
data = spark.read.csv('titanic.csv',inferSchema=True,header=True)

data.printSchema()
data.columns

# Use numeric columns
my_cols = data.select(['Survived',
'Pclass',
'Sex',
'Age',
'SibSp',
'Parch',
'Fare',
'Embarked'])
my_final_data = my_cols.na.drop()
# working with categorical columns
from pyspark.ml.feature import
(VectorAssembler,VectorIndexer,OneHotEncoder,StringIndexer)
gender_indexer = StringIndexer(inputCol='Sex',outputCol='SexIndex')
gender_encoder = OneHotEncoder(inputCol='SexIndex',outputCol='SexVec')
embark_indexer = StringIndexer(inputCol='Embarked',outputCol='EmbarkIndex')
embark_encoder = OneHotEncoder(inputCol='EmbarkIndex',outputCol='EmbarkVec')
assembler = VectorAssembler(inputCols=['Pclass',
'SexVec',
'Age',
'SibSp',
'Parch',
'Fare',
'EmbarkVec'], outputCol='features')
from pyspark.ml.classification import LogisticRegression
# Pipelines
from pyspark.ml import Pipeline
log_reg_titantic = LogisticRegression(featuresCol='features',
labelCol='Survived')
pipeline = Pipeline(stages=[gender_indexer,
embark_indexer,
gender_encoder,
embark_encoder,
```

```
assembler,
log_reg_titantic])
train_titanic_data, test_titanic_data = my_final_data.randomSplit([0.7,0.3])
fit_model = pipeline.fit(train_titanic_data)
results = fit_model.transform(test_titanic_data)
from pyspark.ml.evaluation import BinaryClassificationEvaluator
my_eval =
BinaryClassificationEvaluator(rawPredictionCol='prediction',labelCol='Survived')
results.select('Survived','prediction').show()
AUC = my_eval.evaluate(results)
print("AUC is: "+ str(AUC))
```

Outputs are as follows:

data.printSchema():

```
root
 |-- PassengerId: integer (nullable = true)
 |-- Survived: integer (nullable = true)
 |-- Pclass: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Sex: string (nullable = true)
 |-- Age: double (nullable = true)
 |-- SibSp: integer (nullable = true)
 |-- Parch: integer (nullable = true)
 |-- Ticket: string (nullable = true)
 |-- Fare: double (nullable = true)
 |-- Cabin: string (nullable = true)
 |-- Embarked: string (nullable = true)
```

results.select('Survived','prediction').show():

```
+--------+----------+
|Survived|prediction|
+--------+----------+
|       0|       1.0|
|       0|       1.0|
|       0|       1.0|
|       0|       0.0|
|       0|       1.0|
|       0|       0.0|
|       0|       0.0|
|       0|       1.0|
|       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|
+--------+----------+
only showing top 20 rows
```

```
AUC is: 0.828276247207744
```

Unsupervised Learning

Unsupervised learning deals with unlabelled data. The objective is to observe the structure of data and find patterns. Tasks like cluster analysis, association rule mining, outlier detection, dimensionality reduction etc. Those types of problem can be modelled as unsupervised learning problems. As the tasks involved in unsupervised learning vary vastly, there is no single process outline that we can follow.

For Instance, Cluster analysis is a subset of unsupervised learning that aims to create groups of similar items from a set of items. This analysis helps us identify interesting groups of objects that we are interested in. It could be items we encounter in day-to-day life such as movies or songs according to taste, or interests of users in terms of their demography or purchasing patterns.

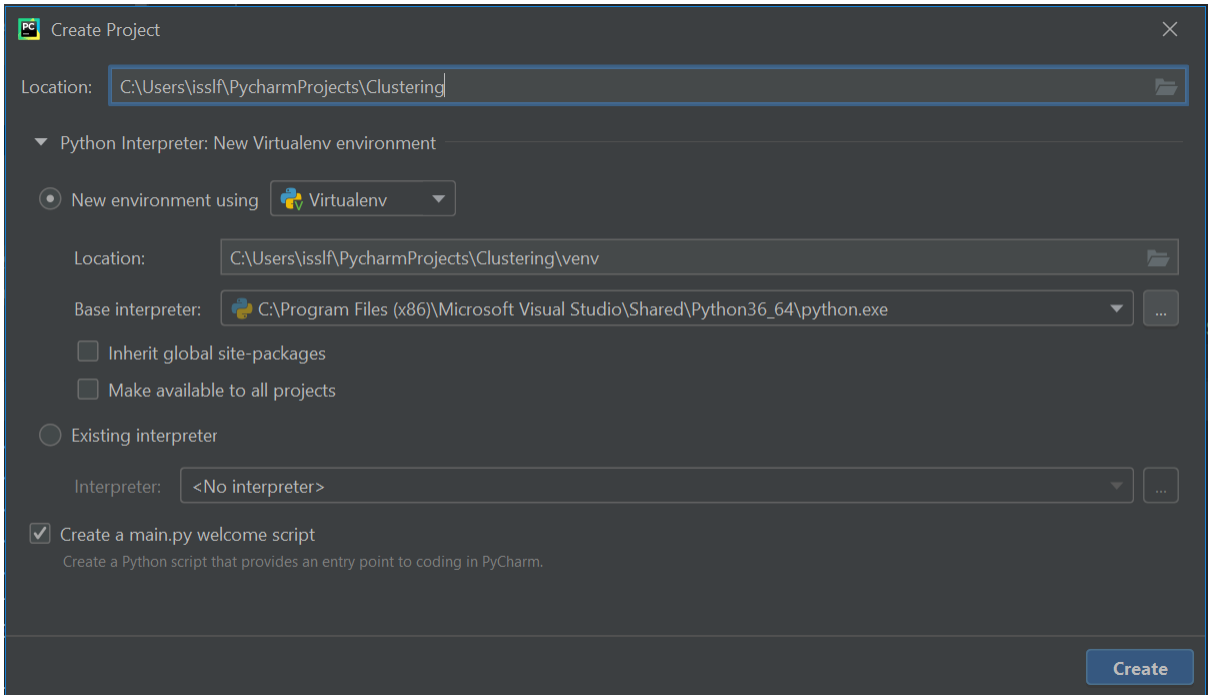
Clustering

K-Means is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters.

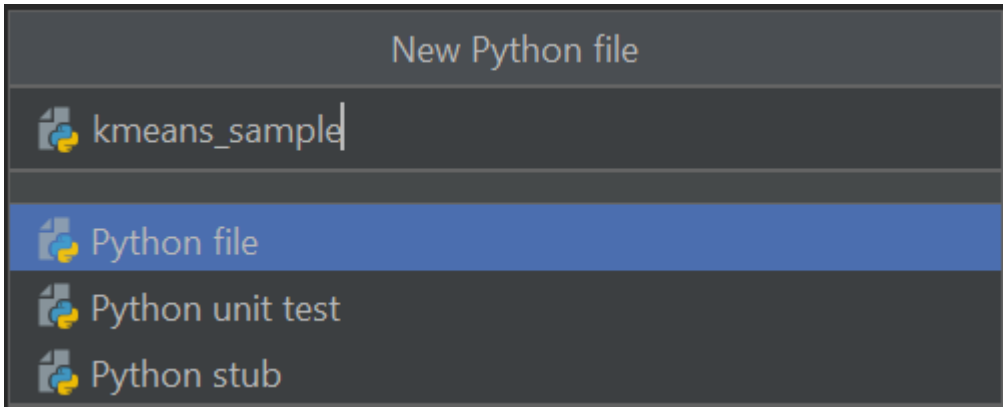
Clustering on a sample data set

Here is a simple K-Means example. The dataset is

Create new Python Project named as “Clustering” in PyCharm:



Create new Python file named as “kmeans_sample”:



kmeans_sample.py are as follows:

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.sql import SparkSession

spark = spark = SparkSession.builder.appName("kmeans").getOrCreate()
# Load data set
dataset = spark.read.format("libsvm").option("numFeatures","
3").load("sample_kmeans_data.txt")
dataset.show()
final_dataset = dataset.select('features')
final_dataset.show()
# Create KMeans model
kmeans = KMeans().setK(3).setSeed(1)
# Fit model
model = kmeans.fit(final_dataset)
# Show centers
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
# Make prediction
predictions = model.transform(final_dataset)
predictions.show()
# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)
print("Silhouette with square euclidean distance =" + str(silhouette))
```

The following is the output:

dataset.show():

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|       (3,[],[])|
|  1.0|(3,[0,1,2],[0.1,0...|
|  2.0|(3,[0,1,2],[0.2,0...|
|  3.0|(3,[0,1,2],[9.0,9...|
```

```
|  4.0|(3,[0,1,2],[9.1,9...|
|  5.0|(3,[0,1,2],[9.2,9...|
+-----+--------------------+
```

final_dataset.show():

```
+--------------------+
|            features|
+--------------------+
|           (3,[],[])|
|(3,[0,1,2],[0.1,0...|
|(3,[0,1,2],[0.2,0...|
|(3,[0,1,2],[9.0,9...|
|(3,[0,1,2],[9.1,9...|
|(3,[0,1,2],[9.2,9...|
+--------------------+
Cluster Centers:
[9.1 9.1 9.1]
[0.05 0.05 0.05]
[0.2 0.2 0.2]
```

predictions.show():

```
+--------------------+----------+
|            features|prediction|
+--------------------+----------+
|           (3,[],[])|         1|
|(3,[0,1,2],[0.1,0...|         1|
|(3,[0,1,2],[0.2,0...|         2|
|(3,[0,1,2],[9.0,9...|         0|
|(3,[0,1,2],[9.1,9...|         0|
|(3,[0,1,2],[9.2,9...|         0|
+--------------------+----------+
Silhouette with square euclidean distance =0.7915403801266928
```

Clustering on a Real-world Dataset

We'll be working with a real data set about seeds, from UCI repository: https://archive.ics.uci.edu/ml/datasets/seeds.

The examined group comprised kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected for the experiment.

Attribute Information:

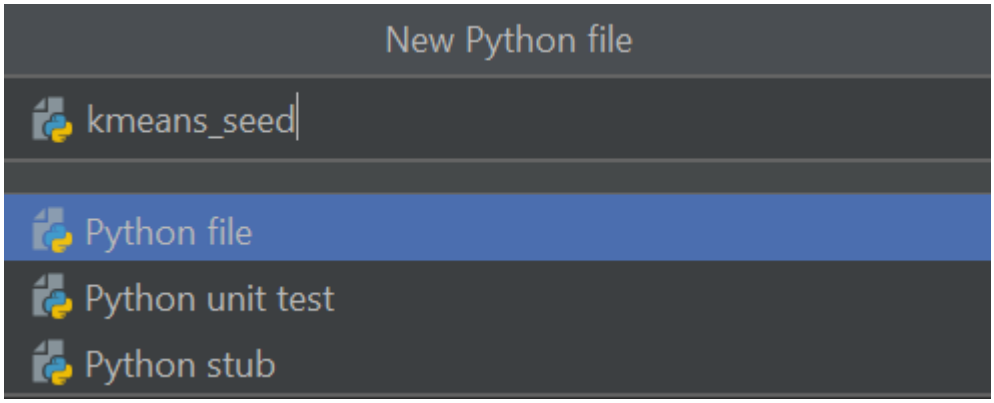
To construct the data, seven geometric parameters of wheat kernels were measured:

- area A,
- perimeter P,
- compactness C = 4piA/P^2,
- length of kernel,
- width of kernel,
- asymmetry coefficient
- length of kernel groove.

All of these parameters were real-valued continuous.

We will cluster them into 3 groups with KMeans.

Create a new Python file named as “kmeans_seed”



kmeans_seed.py:

```
from pyspark.sql import SparkSession
from pyspark.ml.clustering import KMeans

spark = SparkSession.builder.appName('kmeans').getOrCreate()
# Load data set
dataset = spark.read.csv("seeds_dataset.csv",header=True,inferSchema=True)
dataset.head(1)
# Show statistics
dataset.describe().show()

# Format the data
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
dataset.columns
vec_assembler = VectorAssembler(inputCols= dataset.columns,outputCol='features')
final_data = vec_assembler.transform(dataset)

# Scale the data
# It is good to scale our data to deal with the curse of dimensionality
from pyspark.ml.feature import StandardScaler
scaler =
StandardScaler(inputCol="features",outputCol="scaledFeatures",withStd=True,withM
ean=False)
# Compute summary statistics by fitting StandardScaler
scalerModel = scaler.fit(final_data)

# Normalize each feature to hae unit standard deviation
final_data = scalerModel.transform(final_data)
```

```
# Train the model and evaluate
kmeans = KMeans(featuresCol='scaledFeatures', k=3)
model = kmeans.fit(final_data)
# Shows the results
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)

model.transform(final_data).select('prediction').show()
```

The outputs are as follows:

dataset.describe().show():

```
+-------+------------------+------------------+------------------+-------------------+-------
------------+------------------+--------------------+------------------+
|summary|              area|         perimeter|         compactness|   leng
th_of_kernel|   width_of_kernel|asymmetry_coefficient|   length_of_groove|
+-------+------------------+------------------+------------------+-------------------+-------
------------+------------------+--------------------+------------------+
|  count|               210|               210|               210|
        210|               210|                 210|               210|
|   mean|14.847523809523816|14.559285714285718|  0.8709985714285714|  5.628
533333333335| 3.258604761904762|   3.7001999999999997|  5.408071428571429|
|
stddev|2.9096994306873647|1.3059587265640225|0.023629416583846364|0.4430634
7772644983|0.3777144449065867|   1.5035589702547392|0.49148049910240543|
|    min|             10.59|             12.41|              0.8081|
       4.899|              2.63|                0.765|              4.519|
|    max|             21.18|             17.25|              0.9183|
       6.675|             4.033|                8.456|               6.55|
+-------+------------------+------------------+------------------+-------------------+-------
------------+------------------+--------------------+------------------+
```

centers:

```
Cluster Centers:
[ 4.96198582 10.97871333 37.30930808 12.44647267  8.62880781  1.80061978
 10.41913733]
[ 4.07497225 10.14410142 35.89816849 11.80812742  7.54416916  3.15410901
 10.38031464]
[ 6.35645488 12.40730852 37.41990178 13.93860446  9.7892399   2.41585013
 12.29286107]
```

prediction:

```
+----------+
|prediction|
+----------+
|         0|
|         0|
|         0|
|         0|
|         0|
|         0|
|         0|
|         0|
```

```
|          2|
|          0|
|          0|
|          0|
|          0|
|          0|
|          0|
|          0|
|          0|
|          0|
|          0|
|          1|
+-----------+
only showing top 20 rows
```

Summary

In this workshop, we applied different models from Spark ML Lib; Spark has support for other ML algorithms such as Collaborative Filtering for Recommending system, Natural language process, Decision Tree and Random Forests, etc.

The ML library also provides other tools such as:

- Featurization: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating and tuning ML Pipeline
- Persistence: saving and load algorithms, models and Pipeline
- Etc.

There are some advanced topics in terms of optimization, but it requires a lot of statistics, and advanced mathematics, feel free to explore by yourself.

~~ End~~