Linear Regression in Pyspark

May 12, 2017

1 Introduction:

In this report we will be building a machine learning model which will be capable of predicting the energy consumed by an household on a particular day. We will be utilizing the dataset obtained from UCI web repository. This dataset has more than 2 million records and is sampled at every one minute. This problem belongs under regression category, as we would be predicting a continuous variable. We would be building a simple linear regression model and will evaluate the model performance using appropriate methods.

2 Motivation:

Power is a type of resource which needs to be conserved and hence its utilization should be optimized. The necessity or power conservation at times goes for a toss due to negligence and we end up consuming more power. This negligence comes at huge costs which includes monetory as well as resource loss.

So in order to optimize this situation we need to keep and check and alert the user when the power consumption can be high.

So for this purpose we propose a novel machine learning algorithm which can be used to predict the power consumption levels for a particular day. In the below steps we will look into this problem in much detail and come up with some data backed solutions.

3 Design:

The design of our project can be divided into following steps,

> Importing the libraries and creating a spark session.> Loading and cleaning the dataset.> Explorartory analysis and feature extraction> Building the model and evaluating it.> Conclusion and making inferences.

3.1 Step 1: Importing the libraries and creating a spark session.

In the below step we will load all the required libraries. The description of each library and function has been mentioned in the code chunk.

```
In [1]: from pyspark.sql import SQLContext#For loading the csv files as dataframes
        sqlContext = SQLContext(sc)
        from pyspark.ml.tuning import TrainValidationSplit#For train test split
        from pyspark.ml.regression import LinearRegression#model builder function
        from pyspark.sql import SparkSession#creating spark session
        from pyspark.ml.feature import VectorAssembler#Data structuring
        from pyspark.sql.types import DoubleType,DateType#For defining schema of dataset
        from pyspark.sql.functions import *
        from datetime import datetime#For manipulating the date column
        from pyspark.ml.evaluation import RegressionEvaluator#model evaluator
```

The below step will be used to create the spark session which be responsible for assembing all the hardware and application level configurations.

```
In [2]: if __name__ == "__main__":
            spark = SparkSession \
                .builder \
                .appName("LinearRegression") \
                .getOrCreate()
```

3.2 Step 2: Loading and cleaning the dataset.

The data obtained from UCI is a text file delimited by ";" and the changes for such a type are mentioned in the below code. We will be using sqlContext for loading the dataset as it provides an easy and fast way of loading and manipulating the dataframes in further operations.

```
In [25]: df = sqlContext.read.format('com.databricks.spark.csv').options(header='true',delimiter
             .load('/home/Downloads/household_power_consumption.txt')
```

The sqlContext was not able to infer the schemas of the data correctly hence we had explicitly defined schema of each column in the below steps. Also the date field was also inferred in datetime format which would allow us to extract more information from the field.

```
In [26]: df=df.withColumn("Global_active_power", df.Global_active_power.cast(DoubleType()))
         df=df.withColumn("Global_reactive_power", df.Global_reactive_power.cast(DoubleType()))
         df=df.withColumn("Voltage", df.Voltage.cast(DoubleType()))
         df=df.withColumn("Global_intensity", df.Global_intensity.cast(DoubleType()))
         df=df.withColumn("Sub_metering_1", df.Sub_metering_1.cast(DoubleType()))
         df=df.withColumn("Sub_metering_2", df.Sub_metering_2.cast(DoubleType()))
         df=df.withColumn("Sub_metering_3", df.Sub_metering_3.cast(DoubleType()))

         func =  udf (lambda x: datetime.strptime(x, '%d/%m/%Y'), DateType())
         df = df.withColumn('Date', func(col('Date')))
```

3.3 Step 3: Explorartory analysis and feature extraction

We will be looking at the structure of the dataset by viewing the top 5 rows of the data. We will also work on to remove NAs which can possibly effect our further calculations.

```
In [5]: df.show(5)
```

```
+----------+--------+-------------------+---------------------+-------+----------------+--------
|      Date|    Time|Global_active_power|Global_reactive_power|Voltage|Global_intensity|Sub_mete
```

```
+----------+--------+-------------------+---------------------+-------+----------------+--------
|2006-12-16|17:24:00|              4.216|                0.418| 234.84|            18.4|
|2006-12-16|17:25:00|               5.36|                0.436| 233.63|            23.0|
|2006-12-16|17:26:00|              5.374|                0.498| 233.29|            23.0|
|2006-12-16|17:27:00|              5.388|                0.502| 233.74|            23.0|
|2006-12-16|17:28:00|              3.666|                0.528| 235.68|            15.8|
+----------+--------+-------------------+---------------------+-------+----------------+--------
only showing top 5 rows
```

```
In [6]: df.count()#total observations in our data
```

```
Out[6]: 2075259
```

```
In [7]: df.printSchema()#getting the schema of the data
```

```
root
 |-- Date: date (nullable = true)
 |-- Time: string (nullable = true)
 |-- Global_active_power: double (nullable = true)
 |-- Global_reactive_power: double (nullable = true)
 |-- Voltage: double (nullable = true)
 |-- Global_intensity: double (nullable = true)
 |-- Sub_metering_1: double (nullable = true)
 |-- Sub_metering_2: double (nullable = true)
 |-- Sub_metering_3: double (nullable = true)
```

```
In [8]: df=df.na.drop()#removing NAs
        df.count()#Count after NA removal
```

```
Out[8]: 2049280
```

```
In [27]: df.select('Global_intensity').describe().show()
```

```
+-------+-----------------+
|summary| Global_intensity|
+-------+-----------------+
|  count|          2049280|
|   mean|4.627759310588417|
| stddev|4.444396259786192|
|    min|              0.2|
|    max|             48.4|
+-------+-----------------+
```

In the below steps we will be extracting 4 features from the date parameter which would go into our model building process. We will extract the day of month, Day of the year, the month and year from the dataset. Also we will remove the date and time parameters which are now redundant as the information of these has been extracting and integrated in the form of new features.

```
In [12]: df=df.withColumn('Day', dayofmonth('Date'))#extracting day of month
         df=df.withColumn('Day_y', dayofyear('Date'))#extracting day of year
         df=df.withColumn('month', month('Date'))#extracting month
         df=df.withColumn('year', year('Date'))#extracting year
```

```
In [13]: df=df.drop('Date')#dropping date column
         df=df.drop('Time')#dropping time column
         df.show(2)#Viewing the new records
```

```
+-------------------+---------------------+-------+----------------+--------------+-------------
|Global_active_power|Global_reactive_power|Voltage|Global_intensity|Sub_metering_1|Sub_metering_
+-------------------+---------------------+-------+----------------+--------------+-------------
|              4.216|                0.418| 234.84|            18.4|           0.0|           1.
|               5.36|                0.436| 233.63|            23.0|           0.0|           1.
+-------------------+---------------------+-------+----------------+--------------+-------------
only showing top 2 rows
```

3.4 Step 4: Building the model and evaluating it.

One of the reasons why spark is fast is that it does parallel processing on the data. So for that, there has to be some structuring done on the data and hence allowing spark to operate on it. The linear regression model in spark needs the data to be in the form of features and labels, where features are the vectors of all the independent variables and label is the target (output) variable.In our case Global intensity is the label and all others are the features. For assembling the features we will use a assembler function which will combine all the input features into single feature vector.

```
In [14]: cols=['Global_active_power','Global_reactive_power','Voltage','Sub_metering_1',
         'Sub_metering_2','Sub_metering_3','Day_y','Day','month','year']

         assembler=VectorAssembler(inputCols=cols,outputCol="features")#asembler for all i/p fea
         df=assembler.transform(df)#transforming the data
```

```
In [15]: df=df.select("Global_intensity","features")#combing o/p and i/p columns
         df=df.toDF("label","features")#renaming the columns
```

```
In [16]: df.show(2)#viewing the top 2 rows to understand the structure of data
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
| 18.4|[4.216,0.418,234...|
| 23.0|[5.36,0.436,233.6...|
+-----+--------------------+
only showing top 2 rows
```

In the below steps we will split the data into 75-25 percent for train-test respectively. We will use the train for building and training the model and test for predicitng. We will also look at some model metrics which will help us in evaluating the model.

```
In [17]: train, test = df.randomSplit([0.75, 0.25], seed=121)#Splitting into train and test
         # Define LinearRegression algorithm
         lr = LinearRegression()
         #Fit the model on the train data
         lrModel = lr.fit(train)
```

```
In [18]: # Print the coefficients and intercept for linear regression
         print("Coefficients: %s" % str(lrModel.coefficients))
         print("Intercept: %s" % str(lrModel.intercept))
```

```
Coefficients: [4.17758642536,0.758770641078,-0.0190950101879,0.00289293352559,0.00332667441527,-
Intercept: 11.2073177952
```

For a regression problem root mean square error can be good metric to evaluate the model. It defines the distance of the predicted values from the actual values. In the below output chunk it can be seen that rmse is quite low as compared to the original vales of the global intensity feature. Lower the values of rmse, lower is the error and hence the accuracy of the model is high. It can also be seen that the model is able to explain more than 99

```
In [19]: # Summarize the model over the training set and print out some metrics
         trainingSummary = lrModel.summary
         print("numIterations: %d" % trainingSummary.totalIterations)
         print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))
         trainingSummary.residuals.show()
         print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
         print("r2: %f" % trainingSummary.r2)
```

```
numIterations: 1
objectiveHistory: [0.0]
+--------------------+
|           residuals|
+--------------------+
| -0.2611831341588086|
|-0.25158508365371474|
|-0.25101223334807726|
|-0.23535432499400882|
|-0.23712342132207026|
|  -0.225424919696308|
|-0.22084211725121533|
|-0.20703911637794653|
|-0.21599445221342534|
|-0.21536704019006975|
|-0.22005494976403311|
|-0.21255735037960583|
|-0.21201178093282919|
|-0.21961848784255728|
| -0.2116025998702124|
|-0.21721797909470836|
|-0.21721797909470836|
| -0.2166178479302136|
|-0.20879291005974493|
|-0.21607227848343696|
+--------------------+
only showing top 20 rows

RMSE: 0.170039
r2: 0.998535
```

The rmse of the model on train and test almost remains same and hence it can be inferred that the model performs equally well on unseen data as well and hence the model is good. Now that we have a good model we will dive into the inferences of the given problem and what challenges were faced during the project.

```
In [20]: predictions=lrModel.transform(test)
         evaluator = RegressionEvaluator(metricName="rmse")
         RMSE = evaluator.evaluate(predictions)
         print("Model: Root Mean Squared Error = " + str(RMSE))
```

```
Model: Root Mean Squared Error = 0.170344677796
```

4 Challenges faced:

Following were some of the challenges faced during the project. 1. Finding the problem and getting relevant data.2. The data was not clean and a lot of preprocessing was involved in initial stages.3. Feature extraction from dates and generating features.4. Model building and evaluating

5 Conclusion:

The model built in this report was successfull in predicting the power consumption levels of the households. This model can be utilized to build applications which can help in generating a trigger or alert to inform te user about the power consumption levels and prior steps can be taken to keep the consumption levels under check. This report and the model can thus help a large set of audience and hence protect the environment.