PredictNextMinLoad

October 26, 2017

```
In [3]: from pyspark.sql import SparkSession

        # Build the SparkSession
        spark = SparkSession.builder \
            .master("local") \
            .appName("Machine learning for Load Prediction") \
            .config("spark.executor.memory", "1gb") \
            .getOrCreate()
        sc = spark.sparkContext
        # Load the data by creating rdd
        rdd = sc.textFile('/home/hassan/Side_Projects/WeblogChallenge/data/2015_07_22_mktplace_s
        # split the data into columns
        rdd = rdd.map(lambda line: line.split(" "))
        # ====================================
        # Manipulating data
        # ====================================
        from pyspark.sql import Row
        from pyspark.sql.types import *
        from pyspark.sql.functions import *

        #Map the RDD to a DF for better performance
        mainDF = rdd.map(lambda line: Row(timestamp=line[0], ipaddress=line[2].split(':')[0],url
        # convert timestamps from string to timestamp datatype
        mainDF = mainDF.withColumn('timestamp', mainDF['timestamp'].cast(TimestampType()))

        #get count of hit within window of 60 every seccond
        loadperMinDF = mainDF.select(window("timestamp", "60 seconds").alias('timewindow'),'time
        # get count of hit per IP
        countdDF = mainDF.select(window("timestamp", "60 seconds").alias('timewindow'),'timestam
        countdDF.show(20)
```

```
+--------------------+---------------+---------+
|          timewindow|      ipaddress|HitperMin|
+--------------------+---------------+---------+
|[2015-07-22 05:00...|117.200.191.192|        1|
|[2015-07-22 05:00...| 106.77.203.224|        1|
|[2015-07-22 05:00...| 122.15.164.218|        3|
|[2015-07-22 05:00...|  107.150.4.153|        1|
|[2015-07-22 05:00...|  103.242.156.9|        1|
|[2015-07-22 05:00...|   106.67.99.24|        1|
|[2015-07-22 05:00...|107.167.108.212|        2|
|[2015-07-22 05:00...| 122.166.231.76|        1|
|[2015-07-22 05:00...| 182.68.216.254|        2|
|[2015-07-22 05:00...|117.234.213.177|        1|
|[2015-07-22 05:00...|  49.205.99.169|        1|
|[2015-07-22 05:00...| 117.211.43.234|        2|
|[2015-07-22 05:00...|  49.207.236.65|        1|
|[2015-07-22 05:00...|107.167.108.131|        2|
|[2015-07-22 05:00...|  128.185.3.222|        1|
|[2015-07-22 05:01...| 115.113.117.48|       14|
|[2015-07-22 05:01...|     1.39.13.51|        3|
|[2015-07-22 05:01...|  203.122.41.18|        5|
|[2015-07-22 05:01...| 117.196.181.12|        6|
|[2015-07-22 05:01...|  116.203.5.226|        5|
+--------------------+---------------+---------+
only showing top 20 rows
```

```
In [4]: # computing mean ,std and max of hit counts per IP within 60 seccond window as
        # features of each 60 seccond window
        # these features can be used for perdicting the next load in next minute
        Feature1 = countdDF.groupBy("timewindow").agg(stddev('HitperMin').alias("stdOfHitPerMinP
        Feature2 = countdDF.groupBy("timewindow").agg(mean('HitperMin').alias("meanOfHitPerMinPe
        Feature3 = countdDF.groupBy("timewindow").agg(max('HitperMin').alias("maxOfHitPerMinPerI

        Features = Feature1.join(Feature2,["timewindow"])
        Features = Features.join(Feature3,["timewindow"])
        Features = Features.join(loadperMinDF,["timewindow"])
        Features = Features.orderBy('timewindow', ascending=True)
        Features.show(20,False)
```

```
+---------------------------------------------+-------------------+--------------------+--------
|timewindow                                   |stdOfHitPerMinPerIP|meanOfHitPerMinPerIP|maxOfHit
+---------------------------------------------+-------------------+--------------------+--------
|[2015-07-21 22:40:00.0,2015-07-21 22:41:00.0]|37.63654992233117  |6.183619550858652   |978
|[2015-07-21 22:41:00.0,2015-07-21 22:42:00.0]|45.16269677704943  |8.580278128950695   |846
|[2015-07-21 22:42:00.0,2015-07-21 22:43:00.0]|41.00957913182911  |7.097839898348157   |915
|[2015-07-21 22:43:00.0,2015-07-21 22:44:00.0]|21.876291841819434 |5.738181818181818   |366
|[2015-07-21 22:44:00.0,2015-07-21 22:45:00.0]|19.665877050690277 |5.591397849462366   |374
|[2015-07-21 22:45:00.0,2015-07-21 22:46:00.0]|2.0763215336529917 |1.8888888888888888  |18
|[2015-07-22 01:09:00.0,2015-07-22 01:10:00.0]|NaN                |1.0                 |1
|[2015-07-22 01:10:00.0,2015-07-22 01:11:00.0]|13.95908526268474  |5.097930338213024   |355
|[2015-07-22 01:11:00.0,2015-07-22 01:12:00.0]|16.928535755315508 |6.049766718506999   |346
|[2015-07-22 01:12:00.0,2015-07-22 01:13:00.0]|22.19877473979011  |6.37948984903696    |539
|[2015-07-22 01:13:00.0,2015-07-22 01:14:00.0]|25.967580688670626 |6.771067415730337   |467
```

```
|[2015-07-22 01:14:00.0,2015-07-22 01:15:00.0]|18.09106237421149  |5.919556840077071   |575
|[2015-07-22 01:15:00.0,2015-07-22 01:16:00.0]|6.7884838863006856 |2.982490272373541   |75
|[2015-07-22 02:54:00.0,2015-07-22 02:55:00.0]|0.0                |1.0                 |1
|[2015-07-22 02:55:00.0,2015-07-22 02:56:00.0]|24.397729320063686 |5.4246753246753245  |1058
|[2015-07-22 02:56:00.0,2015-07-22 02:57:00.0]|13.954862803139653 |5.335841584158416   |493
|[2015-07-22 02:57:00.0,2015-07-22 02:58:00.0]|11.228685780753388 |5.507252382925818   |152
|[2015-07-22 02:58:00.0,2015-07-22 02:59:00.0]|10.784985916123745 |5.006719865602688   |274
|[2015-07-22 02:59:00.0,2015-07-22 03:00:00.0]|11.13514600581905  |5.062182741116751   |278
|[2015-07-22 03:00:00.0,2015-07-22 03:01:00.0]|4.046688058258602  |2.400749063670412   |59
+---------------------------------------------+-------------------+--------------------+--------
only showing top 20 rows
```

```
In [5]: # Divide hit coutns by 60 to become hit per seccond
        Features = Features.withColumn("stdOfHitPerSecPerIP", col("stdOfHitPerMinPerIP")/60.0) \
           .withColumn("meanOfHitPerSecPerIP", col("meanOfHitPerMinPerIP")/60.0) \
           .withColumn("maxOfHitPerSecPerIP", col("maxOfHitPerMinPerIP")/60.0) \
           .withColumn("HitperSec", col("HitperMin")/60.0)
        Features = Features.select("timewindow","stdOfHitPerSecPerIP","meanOfHitPerSecPerIP","ma
        Features.show(5)
```

```
+--------------------+-------------------+--------------------+-------------------+-------------
|          timewindow|stdOfHitPerSecPerIP|meanOfHitPerSecPerIP|maxOfHitPerSecPerIP|         Hitp
+--------------------+-------------------+--------------------+-------------------+-------------
|[2015-07-21 22:40...| 0.6272758320388528|  0.1030603258476442|               16.3| 78.016666666
|[2015-07-21 22:41...| 0.7527116129508239| 0.14300463548251158|               14.1|113.116666666
|[2015-07-21 22:42...| 0.6834929855304852| 0.11829733163913596|              15.25|
|[2015-07-21 22:43...| 0.3646048640303239| 0.09563636363636364|                6.1|
|[2015-07-21 22:44...| 0.3277646175115046|  0.0931899641577061|  6.233333333333333|
+--------------------+-------------------+--------------------+-------------------+-------------
only showing top 5 rows
```

```
In [6]: # get id for each window
        Features = Features.withColumn("tagId", monotonically_increasing_id().cast("double"))
        Features.show(10)
```

```
+--------------------+-------------------+--------------------+--------------------+------------
|          timewindow|stdOfHitPerSecPerIP|meanOfHitPerSecPerIP| maxOfHitPerSecPerIP|           H
+--------------------+-------------------+--------------------+--------------------+------------
|[2015-07-21 22:40...| 0.6272758320388528|  0.1030603258476442|                16.3|   78.016666
|[2015-07-21 22:41...| 0.7527116129508239| 0.14300463548251158|                14.1|   113.116666
|[2015-07-21 22:42...| 0.6834929855304852| 0.11829733163913596|               15.25|
|[2015-07-21 22:43...| 0.3646048640303239| 0.09563636363636364|                 6.1|
|[2015-07-21 22:44...| 0.3277646175115046|  0.0931899641577061|   6.233333333333333|
|[2015-07-21 22:45...|0.03460535889421653| 0.03148148148148148|                 0.3|   5.3833333
|[2015-07-22 01:09...|                NaN|0.016666666666666666|0.016666666666666666|0.0166666666
```

```
|[2015-07-22 01:10...|0.23265142104474565| 0.08496550563688372|   5.916666666666667|  168.316666
|[2015-07-22 01:11...| 0.2821422625885918| 0.10082944530844998|   5.766666666666667|
|[2015-07-22 01:12...|0.36997957899650186| 0.10632483081728267|   8.983333333333333|
+--------------------+-------------------+--------------------+--------------------+------------
only showing top 10 rows
```

```
In [7]: # get hit per sec of each next 60 sec window
        # we will use this as the label for training model
        from pyspark.sql.window import Window
        w = Window.orderBy('tagId').rowsBetween(1,1)
        avgHit = avg(Features['HitperSec']).over(w)
        NextloadDf = Features.select(Features['stdOfHitPerSecPerIP'],Features['meanOfHitPerSecPe
        NextloadDf.show(10)
```

```
+-------------------+--------------------+--------------------+--------------------+------------
|stdOfHitPerSecPerIP|meanOfHitPerSecPerIP| maxOfHitPerSecPerIP|           HitperSec|   LoadInNex
+-------------------+--------------------+--------------------+--------------------+------------
| 0.6272758320388528|  0.1030603258476442|                16.3|   78.01666666666667|  113.116666
| 0.7527116129508239| 0.14300463548251158|                14.1|  113.11666666666666|
| 0.6834929855304852| 0.11829733163913596|               15.25|                93.1|
| 0.3646048640303239| 0.09563636363636364|                 6.1|                78.9|
| 0.3277646175115046|  0.0931899641577061|   6.233333333333333|                78.0|   5.3833333
|0.03460535889421653| 0.03148148148148148|                 0.3|   5.383333333333334|0.0166666666
|                NaN|0.016666666666666666|0.016666666666666666|0.016666666666666666|  168.316666
|0.23265142104474565| 0.08496550563688372|   5.916666666666667|  168.31666666666666|
| 0.2821422625885918| 0.10082944530844998|   5.766666666666667|               194.5|
|0.36997957899650186| 0.10632483081728267|   8.983333333333333|              204.25|
+-------------------+--------------------+--------------------+--------------------+------------
only showing top 10 rows
```

```
In [8]: # removing null values
        NextloadDf = NextloadDf.na.drop(subset=["stdOfHitPerSecPerIP"])
        NextloadDf = NextloadDf.na.drop(subset=["LoadInNextOnedMin"])
        NextloadDf = NextloadDf.na.drop(subset=["HitperSec"])

        from pyspark.ml.linalg import DenseVector
        # Define the `input_data`
        input_data = NextloadDf.rdd.map(lambda x: (x[4], DenseVector(x[:4])))
        dataFrameInputdata = spark.createDataFrame(input_data, ["label", "features"])
        dataFrameInputdata.first()

Out[8]: Row(label=113.11666666666666, features=DenseVector([0.6273, 0.1031, 16.3, 78.0167]))
```

```
In [22]: # training a linear regression Model
         train_data, test_data = dataFrameInputdata.randomSplit([.8,.2])
         from pyspark.ml.regression import LinearRegression,RandomForestRegressor

         rf = RandomForestRegressor(numTrees=100, maxDepth=10)
         linearModel = rf.fit(train_data)

         #lr = LinearRegression(labelCol="label", maxIter=100, regParam=0.3, elasticNetParam=0.8
         # Fit the data to the model
         #linearModel = lr.fit(train_data)


         predicted = linearModel.transform(test_data)
         predictions = predicted.select("prediction").rdd.map(lambda x: x[0])
         labels = predicted.select("label").rdd.map(lambda x: x[0])
         predictionAndLabel = predictions.zip(labels).collect()


In [23]: import numpy as np
         error =[]
         for a in predictionAndLabel:
            error.append(np.abs(a[0]-a[1]))

         print 'mean abs error is: ',np.mean(error)

mean abs error is:  65.400781239


In [28]: # predicting the load for the next minute
         # becuase the last data record is the last 60 seccond of data
         # we predict for the last record for predicting the next minute load

         with_id = NextloadDf.withColumn("_id", monotonically_increasing_id())
         i = with_id.select(max("_id")).first()[0]
         last_item = with_id.where(col("_id") == i).drop("_id")
         input_data = last_item.rdd.map(lambda x: (x[4], DenseVector(x[:4])))
         dataFrameInputdata = spark.createDataFrame(input_data, ["label", "features"])
         predicted = linearModel.transform(dataFrameInputdata)
         predictions = predicted.select("prediction").rdd.map(lambda x: x[0])
         labels = predicted.select("label").rdd.map(lambda x: x[0])
         predictions = predictions.collect()
         predictions[:]

Out[28]: [69.80983333333323]

In [ ]:
```