**Introduction should be re-written**

Zettabytes of raw data is available in the world today. New data is also generating at exponential rate, handling and utilizing such huge data is a challenge today. In many environments such as chemical, pharmaceuticals, Biological etc. So there is a need of a system which can handle and process this data effectively and efficiently, also provide the maximum speed through parallelization and distribution of tasks and also provide the reliability when data is lost in between of processing. In past there were several systems developed for processing of big data that were mostly based on map-reduce framework. Each framework had several disadvantages for example, all frameworks were using hard drive for saving and retrieving the results which is very costly is terms of time and speed.

**What is Big Data?**

Big Data can be defined as a dataset so huge in size in which normal data processing cannot be applied. As I already said that data is generating at exponential rate, this huge amount of data is being generated by different sources which can be social media, sensors or mobile devices. If you take the example of social media, [Facebook](http://wersm.com/category/facebook), alone has [1.4 billion active monthly users](http://wersm.com/facebook-now-has-over-1-4-billion-monthly-active-users/), who like millions of post every minute and over 100 million posts per hour. Facebook alone generating huge amount of data, if we aggregate the data of all social media sites, sensors and mobile devices (or any existing source which generates data) this will become a huge amount of data. Analysing and extracting useful information from it, is a challenge today. Data plays very important role now a days and every successful industry in the world today using data to take decisions to have a growth in a competitive environment. For example, Organization can get buyer behaviour data to get better understanding of changing habits of customers so that it perfectly deals with time to market and competition environment.

Apache Hadoop and Apache Spark is the well-known example of distributed system. Hadoop and Spark are design for the distributed processing of large data sets across clusters of computers. Although we use Hadoop for fast distributed computing but it had several disadvantages. For example, it does not use In-memory computation which is nothing but keeping the data in RAM instead of Hard Disk for fast processing. In-memory computation is needed when there is a need of fast processing in any environment; When Apache Spark developed it overcome this problem. Apache Spark uses In-memory computation for fast computing. Apache Spark is not replacement of Hadoop. Spark is designed to run on top of Hadoop.

**Challenges while working with big data**

The major challenges associated with big data are as follows:

**Capturing:** Capturing data is tough task today because now a day it comes with large volume and high velocity. Today, both the speed and volume of data collected have exploded. There are millions of data sources are available which are generating data at high speed.

**Storage:** It is also one of the biggest challenge now a days industries are facing, how to store their data efficiently.

**Querying and Analyzing**: In this task people want to have useful information or insights from data in less time or you can say real time. If you look at the current scenario, you can say that this is more challenging compare to capturing and storing data.

**Transferring:** If data is stored in different places how to transfer the data from one place to another place more efficiently.

**Methods to deal with these challenges**

Pending

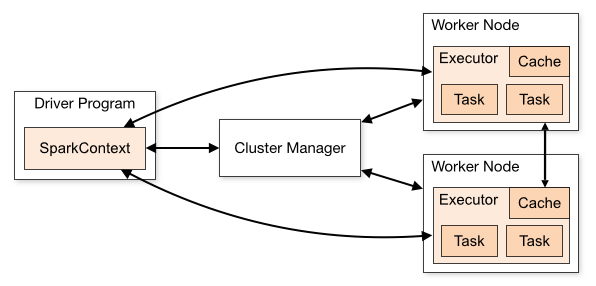
talk about In-memory computation, network, cluster, distributed computing, cloud computing, map-reduce, one map-reduce

**What is Apache Spark?**

Apache Spark is a fast cluster computing framework which is used for processing, querying and analysing big-data. It is based on In-memory computation which is also an advantage of Apache Spark over several big-data Frame work. Apache spark is an open source and one of the most famous big-data framework. It runs tasks up to 100 times faster when it utilizes the in memory computations and 10 times faster when it uses disk than traditional map-reduce task. In terms of speed, Apache Spark is much faster than map-reduce when there is a need of more than one map-reduce for a task.

An Apache Spark cluster is divided into cluster of node where there is one driver program and many workers which execute the operations parallel.

**Key terms used in Apache Spark:**



* **Driver and Worker:** A driver is theprocess running the main() function of an application and creating the Spark Context and a worker is any node that can run application code in the cluster. If a process is launched for an application, then this application acquires executors at worker node.
* **Cluster Manager**: Cluster manager allocate resources to each applications in driver program. There are three types of cluster managers supported by Apache Spark, which are Standalone, Mesos and YARN. Apache Spark is agnostic to the underlying cluster manager, so we can install any cluster manager, each has its own unique advantages depending upon the goal. They all are different in terms of scheduling, Security, Monitoring. Once Spark Context connect to cluster manager it acquires executors on a cluster node, these executors are worker’s node on cluster which work independently on each tasks and interact with each other.
* **Spark Context**: It is a connection which holds a connection to Spark cluster manager. All Spark applications run as an independent set of process, coordinate by a SparkContext in a program.

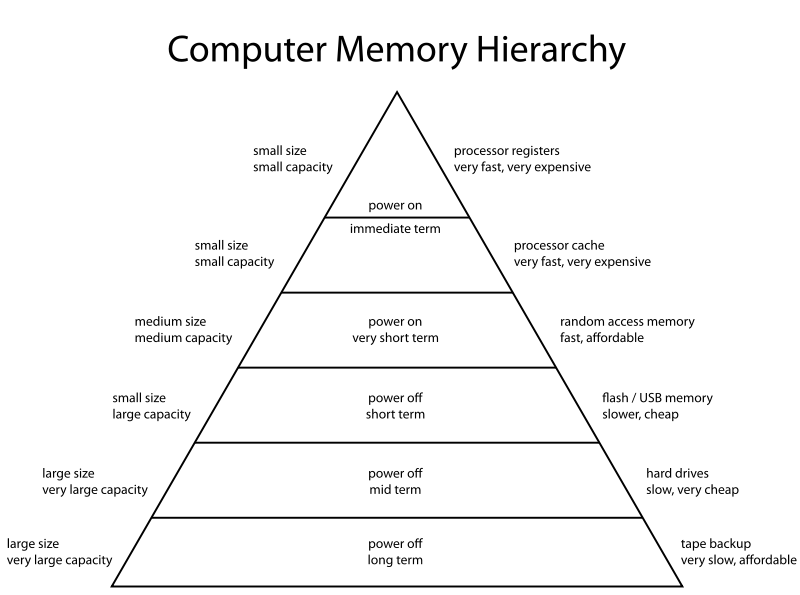
**History of Apache Spark**

Apache Spark originally started at the u[niversity of California, Berkeley](https://en.wikipedia.org/wiki/UC_Berkeley)'s [AMPLa](https://en.wikipedia.org/wiki/AMPLab)b in 2009, the Spark [code bas](https://en.wikipedia.org/wiki/Codebase)e was later donated to the [Apache Software Foundatio](https://en.wikipedia.org/wiki/Apache_Software_Foundation)n. It was open sourced in 2010. Spark mostly written in Scala language. It has some code written in Java, Python and R. Apache spark provides several APIs for programmers which include Java, Scala, R and Python.

**Source “https://en.wikipedia.org/wiki/Apache\_Spark”**

**How Apache Spark is better than traditional big data framework**

**In-memory computation:**  It saves and loads the data in and from the RAM rather than from the disk (Hard Drive). If we talk about memory hierarchy, it is already experimented that RAM has higher processing speed than Hard Drive.



Sparks uses in-memory computations to speed up 100 times faster than Hadoop framework. It recently gains a momentum when memory prices are started decreasing.

In Hadoop, tasks distribute among the all nodes in the cluster that saves the data in disk. When that data is required for processing, each node has to load the data from the disk and save the data into disk after performing operation, this process needed cost in terms of speed and time. Because disk is far slower than RAM. It also requires time to convert the data in a particular format when writing the data into disk from RAM, this conversion is known as **Serialization** and reverse is **Deserialization.**

To understand more about in-memory computation we can take the example of map reduce system. Suppose there are several map-reduce task happening one after another. Initially when computation started, map task in both scenario (Hadoop and Apache Spark) read the data from the hard drive. In case of Hadoop it Performs the map operation and save the results back to hard drive but in case of Apache spark it saves the results in RAM. So in next step, in Hadoop when reduce operation started it again reads the saved data from the Hard drive but in case of Apache Spark instead of reading the data from hard drive it will read the saved data from the RAM. As we have seen that Accessing the data from the hard disk is slower as compare to accessing the data from RAM. So if there are multiple map-reduce operations, significant time deference to do a task can be seen at the end of task completion.

**Language Support:** Apache Spark has API support for major data science languages like Python, R, Scala and Java.

**Supports of Real time and Batch processing:** Apache Spark supports “Batch data” processing where a group of transactions is collected over a period of time, It also support real time data processing where data is continuously coming from the source. For example, the weather information can be processed by Apache Spark which is coming from the sensors in real time.

**lazy operation:** It is used to optimize the solution in Apache Spark. I will discuss about lazy evaluation in later part of this article. For now, we can think that there are some operations which do not execute until we require their result. So for doing that Spark remember the all operations w.r.t. their order on data.

**Map-reduce:** Another advantage of Apache spark over Hadoop is that Hadoop support only map-reduce but Apache Spark support many transformations and actions including map-reduce.

There are more advantages Apache Spark has in comparison with Hadoop. for example, Map side shuffling and Reduce side shuffling which also makes Apache Spark faster. I am not talking about shuffling here, as this needed one more article for covering it, for now you can think that it involves with less transmission as much as possible of data from machine to another machine in a cluster.

**Installation of Apache Spark with pyspark.**

We can install Apache Spark in many different ways. One of the easiest way to install Apache spark is to install it on a single machine. We can install Apache Spark on different Operating Systems. For installing in a single machine we need to have certain prerequisite for different versions of Apache Spark. I’m sharing steps to install for Ubuntu:

**OS:** Ubuntu 14.04, 64 bit

**Software Required:** Java 7+, Python 2.6+, R 3.1+

For the Scala API, Spark 1.6.0 uses Scala 2.10.X

**Installation Steps**

Add installation steps or provide link for Mac and Windows (Optional)

**Step 0**: Open the terminal.

**Step 1**: Install java

$ sudo apt-add-repository ppa:webupd8team/java

$ sudo apt-get update

$ sudo apt-get install oracle-java7-installer

We need to check wether Java has installed successfully or not. To check the java version and installation. Type.

$ java -version

**Step 2** : Install Scala

$ cd ~/Downloads

$ wget http://www.scala-lang.org/files/archive/scala-2.11.7.deb

$ sudo dpkg -i scala-2.11.7.deb

$ scala –version

This will show you the version of Scala installed

**Step 3**: Install git

We need to build the Spark from the source which requires git to be installed.

$ sudo apt-get -y install git

**Step 4**: **Install py4j**

Py4J is used on the driver for local communication between the Python and Java SparkContext objects; large data transfers are performed through a different mechanism.

$ sudo pip install py4j

**Step 5**: **Install Spark.**

As we have already installed the dependencies which require to install Apache Spark. We need to download than extract Spark source tar. We can also refer the latest versions of Apache Spark.

$ cd ~/Downloads

$ wget http://d3kbcqa49mib13.cloudfront.net/spark-1.6.0.tgz

$ tar xvf spark-1.6.0.tgz

**Step 6**: **Compile extracted source**

By using sbt we can build out Apache Spark. sbt is an [open sourc](https://en.wikipedia.org/wiki/Open_source)e build tool for [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)) and Java projects which is similar to Java's Maven.

$ cd ~/Downloads/spark-1.6.0

$ sbt/sbt assembly

This will take some time to install Spark. After installing we can check whether spark is running correctly or not by typing.

$ ./bin/run-example SparkPi 10

this will produce the output:

Pi is roughly 3.14042

**Step 7**: **Create environment variables**

To set the environment variables in bashrc by opening the file in any editor.

$ nano ~/.bashrc

To set the SPARK\_HOME and PYTHONPATH

export SPARK\_HOME= “where your spark”

export PYTHONPATH= “where is your python”

Now, restart bashrc.

$ . ~/.bashrc

**Step 8:** **Start with Pyspark**

Lets start pyspark shell for writing the first program in spark by typing command in root directory:

$ ./bin/pyspark --packages

We can also start ipython notebook in shell by typing:

$ PYSPARK\_DRIVER\_PYTHON=ipython ./bin/pyspark

When we launch the shell in pyspark, it will automatically load spark Context as sc and SQLContext as sqlContext.

**RDD: Prerequisite to work with Spark functionality**

After Installing and configuring pyspark, we can start programming using spark in python but to use spark functionality we must use RDD. RDD (Resilient Distributed Database) is a collection of elements that can be divided across multiple nodes in a cluster to run parallel processing. It is also a fault tolerant collection of elements which means it can automatically recover from failures. RDD is immutable, we can create RDD once but can’t change it. We can apply any number of operation on it and can create another RDD by applying some transformations.

**RDD has 2 parallel operations:**

**Transformation:** Transformation refer to the operation which can apply on RDD and create new RDD. As I already discussed that RDD is immutable in nature so we can transform it but can’t change it.

**Action:** Actions refer to an operation which also apply on RDD that perform computation and send the result back to driver.

**Example:** In Map(Transformation), it performs operation on each element of RDD and return a new RDD but in case of Reduce(Action), it reduce/ aggregate the output of a map result by applying some functions (Reduce by key). There are many transformations and actions are defined in Apache Spark documentation, I will discuss these in later article.

**RDD uses Shared Variable:**

The parallel operations in Apache spark uses shared variable. It means that whenever a task sends by a driver to executers program in a cluster, a copy of shared variable sends to each node in a cluster so that they can use this variable while performing task. **Accumulator** and **Broadcast** are the two types of shared variables supported by Apache spark.

**Broadcast:** We can use the Broadcast variable to save the copy of data across all node.

**Accumulator:** In Accumulator variables are used for aggregating the information.

**How to Create RDD in Apache spark**

**Existing storage:** When want to create the RDD though existing storage in driver program which we would like to parallelized. For example, converting a list to RDD which is already created in a driver program.

**External sources:**  When we want to create the RDD though external sources such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

**Write first program in Apache Spark**

I have discussed that RDD supports two type of operations, which are transformation and action.

**Step1: Create Spark Context**

First step in any Apache programming is to create a Spark Context. Spark Context is needed when we want to execute operations in a cluster. Spark Context tells Spark how and where to access a cluster. It is a first step to connect with Apache Cluster. If we are using Spark Shell we will find that it is already created. Otherwise we can create the Spark Context by importing, initializing and provide the configuration settings. For example:

**from** **pyspark** **import** SparkContext

sc = SparkContext()

We can also set the configuration properties by passing the configuration parameters object in Apache Spark Context. For example:

**from** **pyspark** **import** SparkConf, SparkContext

conf = (SparkConf().setMaster("local").setAppName("My app")

sc = SparkContext(conf = conf)

“My app” is a name of our application which will be showing in cluster User Interface. setMaster(“local”), the master URL to connect to, such as “local” to run locally with one thread, “local[2]”to run locally with 2 thread. We can check more about configuration from documentation. Link is <http://spark.apache.org/docs/latest/configuration.html>

**Step2: Create RDD**

I have already discussed that we can create RDD in two ways first from existing storage and second from external storage. Let’s create the first RDD. SparkContext has parallelize method which is used for creating the spark RDD from the iterable which are present in driver program. We can also provide the number of partition to parallelize method if we will not give number of partition perimeter than spark will automatically set the number of partition in a cluster. The number of partition can be set manually by passing second parameter to parallelize method. For example, sc.parallelize(data, 10)), where data is an existing data in driver program and 10 is the number of partition set by spark.

Lets create the first spark RDD called rdd.

Data = range(1,1000)

rdd = sc.parallelize(data)

We have a collect method to see the content of RDD.

rdd.collect()

To see the first n element of a RDD we have a method take.

rdd.take(2) # It will print first 2 elements of rdd

We have 2 parallel operations in RDD which are Transformation and Action. Transformation and Action is already discussed. So let’s see how transformation works. As I have already dicussed about immutable property of a RDD which mean that we can’t change our RDD but we can apply transformation on it. Let’s see an example of map transformation to demonstrate how transformation works.

**Step 3: Map transformation.**

Map transforamtion Return a Mapped RDD by applying function to each elements of base RDD. Let’s repeat the first step of creating a RDD from existing source, For example,

Data = ['Hello' , 'I' , 'AM', 'Ankit ', 'Gupta']

Rdd = sc.parallelize(data)

Now a RDD(name is 'Rdd') is created from the existing source which is a list of string in a driver program. Apply lambda function to each element of Rdd and return the mapped(tranformed) RDD (word,1) pair in the Rdd1.

Rdd1 = Rdd.map(lambda x: (x,1))

Let’s see the out of this map operation.

Rdd1.collect()

output: [('Hello', 1), ('I', 1), ('AM', 1), ('Ankit ', 1), ('Gupta', 1)]

If you noticed that after applying the lambda function on Rdd1 nothing happen(we won't see any computation happening in a cluster). That is called the **lazy operation**. All Transformation operations in Spark are lazy which means that we will not see any computations on RDD until we apply any action on it. Transformation operations not computed immediately. Spark will remember that which transformation is applied to which RDD with the help of DAG(Directed a Cyclic Graph). The lazy evaluation helps Spark to optimize the solution because spark will get a time to see the DAG before actually execute the operations on RDD. This enables Spark to run operations more efficiently and efficiently. Now we can see that collect and take are the examples of an action.

There are many number of transformation defined in Apache Spark we will talk about more these once, we develop the basic understanding in Apache Spark.

**Working with DataFrame and solving a machine learning problem**

We can also create the dataframe in Apache spark with the help of SQLContext. Which is a distributed collection of observations(rows) with column name, like a table. We can also create the data frame by reading the csv file. Let’s see how can we do that. For reading the csv file in Apache spark we need to specify the library to our program in python shell or we can set the path in “bashrc”. Lets read the the data from a csv files to create the dataframe and apply some data science skills on this dataframe like we do in Pandas.

To work on dataframe, I have taken the data set of “Practice Problem: Black Friday”. You can participate in this challenge here (“<https://datahack.analyticsvidhya.com/contest/black-friday/>”). You can see the description from this link about the dataframe and problem statement, because later in this post we will apply a machine learning model on these data sets.

As I have already discussed that sparkContext and sqlContext always created at the time of lauch of our spark shell. So for reading the csv file directly and for creating the dataframe, we need to use the sqlContext. Lets see how we can do that.

**Open a Spark Shell**

First open a pyspark shell and include the package(“spark-csv\_2.10:1.3.0”) in home directory of our spark. We can also use the latest version of spark-csv package.

$ ./bin/pyspark --packages com.databricks:spark-csv\_2.10:1.3.0

**Loading csv files (Train and Test)**

Lets load the training and testing data from our csv files with the help of sparkContext.

train = sqlContext.load(source="com.databricks.spark.csv", path = 'PATH/train.csv', header = True,inferSchema = True)

test = sqlContext.load(source="com.databricks.spark.csv", path = 'PATH/test-comb.csv', header = True,inferSchema = True)

Here PATH is the location of folder where your training and testing csv files are located. Header is True, it means that the csv files contain the header. We are using inferSchema is True for telling sqlContext to automatically detect the data type of each column in data frame. For example, a categorical column can be referring to as String type.

**Analyze the data type**

To see the types of columns in dataframe we can use the method called printSchema(). Lets apply printschema on train which will *Print the schema in a tree format.*

**train.printSchema()**

**Summarizing data set**

So after reading the csv files, we can see the various summary Statistics of a dataframe columns which can be perform by calling describe method which only shows numerical statistics. To show the results we need to call show method.

train.describe().show()

To see the first n rows of a dataframe we have head method in pyspark like pandas in python. We need to provide an argument(number of rows) insight the head method. Lets see first 10 rows of train

train.head(10)

To see the number of rows in a data frame we need to call a method count(). Lets check the number of rows in train. The count method in pandas and spark are different.

train.count()

To see the columns name in train by calling typing a train.columns, like we use in pandas dataframe.

train.columns

**Sub setting Columns**

Let’s select a column called 'User\_ID' from a train, we need to call a method 'select' and pass the column name which we want to select. The select method will show a result for selected column. We can also select more than two columns from a data frame by providing a more than two columns insight the select method.

train.select('User\_ID').show()

We can also perform many operations on spark dataframe columns. We will discuss these operations in my future posts.

**You should select all the required columns here only**

to predict purchasing behavior of a user based on Gender, Age, Occupation, Marital\_Status and product categories.

**Impute Missing values**

Let’s start applying a machine learning algorithm Let’s check number of not null observations in train and test by calling drop method. By default, drop method will drop a row if it contains any null value. By default, it set to 'any', we can also pass 'all'' to drop a row only if all its values are null.

train.na.drop().count(),test.na.drop('any').count()

For creating a basic ML model here, I am imputing null values in train and test with -1. it is not recommended that impute the null with -1, we have several algorithm to impute null values but for the simplicity i am imputing null with constant value. we can transform our base train,test dataframe after applying this imputation . For imputing constant value we have fillna method. Lets fill the -1 in-place of null in all columns.

train = train.fillna(-1)

test = test.fillna(-1)

**Analyzing categorical features**

For now I want to demonstrate a random forest model to predict the purchase. As we have seen that Purchase has type Integer so this problem comes under regression. So start to a build the model we should also see the categorical features values to check the categories distributions in train and test. Here I am showing this for only Product\_ID but we can also do the same for any categorical feature. Lets see the number of distinct categories in train and test Product\_ID. Which we can do by applying methods distinct than count.

train.select('Product\_ID').distinct().count(), test.select('Product\_ID').distinct().count()

After counting the number of distict values for train and test we can see the test has 60 more catagories than train. Lets check what are those catagories by applying subtract method which will give you the results of catagories which are present in train but not in test. We can also do the same for all. The method subtract method will not genrate the distict result we need to again count the distict values same way as we did above.

diff\_cat\_in\_train\_test = test.select('Product\_ID').subtract(train.select('Product\_ID'))

diff\_cat\_in\_train\_test.distinct().count()# For distict count

After seeing the difference (40) we can either raise the error or skip the rows in test for those categories(invalid category) which are not in train. We also need to transform this column to proper format by applying StringIndexer Transformation on Product\_ID which will encodes the Product\_ID column of labels to a column of label indices. You can see more about this from the link below

https://spark.apache.org/docs/latest/ml-features.html#stringindexer”

To skip or raise the error we have setHandleInvalid method in StringIndexer. By default it will raise the error but for simplicity i am not using this column in my ML model. So for converting Product\_ID into indexed column we need to import StringIndexer from spark.ml.feature. How we can do this is shown below. We also need to tell StringIndexer that which categorical variable we want to transform and what is the will be the name after transform, in our case Product\_ID is the input column name and the output column name is product\_ID('p' is small).

from pyspark.ml.feature import StringIndexer

plan\_indexer = StringIndexer(inputCol = 'Product\_ID', outputCol = 'product\_ID')

model = plan\_indexer.setHandleInvalid('skip').fit(train)

Than we need to build a 'model' which we will need to create by applying fit on a train dataframe that will transform our train and test. Lets transform our train and test dataframe with the help of model. We need to call transform method for doing that. Here we are taking the transformation result in Train1 and Test1.

Train1 = model.transform(train)

Test1 = model.transform(test)

Lets check the resulting Train1 dataframe.

Train1.show()

The show method on Train1 dataframe will show that we successfully added one transformed column product\_ID in our previous train dataframe. Lets try to create a formula for Machine learning model like we do in R. First we need to import RFormula from the pyspark.ml.feature than we need to specify the dependent and independent column insight this formula. We also have to specify the names for features column and label column.

from pyspark.ml.feature import RFormula

formula = RFormula(formula="Purchase ~ Age+ Occupation +City\_Category+Stay\_In\_Current\_City\_Years+Product\_Category\_1+Product\_Category\_2+ Gender",featuresCol="features",labelCol="label")

After creating the formula we need to fit this formula on our Train1 and transform Train1,Test1 through this formula. Lets see how to do this and after fitting transform train1,Test1 in train1,test1.

model1 = formula.fit(Train1)

train1 = t1.transform(Train1)

test1 = t1.transform(Test1)

We can see the transformed train1,test1.

train1.show()

After applying the formula we can see that train1,test1 has 2 extra columns called features and label which we have specified in formula( featuresCol="features" and labelCol="label"). The intuition is that all categorical variables in the features column in train1 and test1 are transformed to the numerical and the numerical variables are same as before for applying ML. Purchase variable will transom to label column. We can also look at the columns features and label in train1 and test1.

train1.select('features').show()

train1.select('label').show()

After applying the RFormula and transformed the dataframe, we now need to develop the machine learning model on this data. I want to apply a random forest regressor for this task. Lets import a random forest regressor which is defined in pyspark.ml.regression and than create a model called rf. Here i am using a default parameters for randomforest algorithm.

from pyspark.ml.regression import RandomForestRegressor

rf = RandomForestRegressor()

After creating a model rf we need to divide our train1 data to train\_cv and test\_cv for cross validation.

Here we are dividing train1 dataframe in 70% for train\_cv and 30% test\_cv.

(train\_cv, test\_cv) = train1.randomSplit([0.7, 0.3])

Now build the model on train\_cv and predict on test\_csv. The results of prediction will save predictions.

model1 = rf.fit(train\_cv)

predictions = model1.transform(test\_cv)

If you check the columns in predictions dataframe there is one column called prediction which has prediction result for test\_csv.

model1 = rf.fit(train\_cv)

predictions = model1.transform(test\_cv)

Lets evaluate our predictions on test\_cv and see what is the mean sqaue error. To evaluate our model we need to import RegressionEvaluator from the pyspark.ml.evaluation and have to create a object of this. There is a method called evaluate for evaluator which will evaluate the model. We need to specify the metrics for that.

from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator()

mse = evaluator.evaluate(predictions,{evaluator.metricName:"mse" })

After evaluation we can see that our root mean square error is 3773.1460883883865 which is a square root of mse.

import numpy as np

np.sqrt(mse) ,mse

Now we will implement the same process on full train1 dataset.

model = rf.fit(train1)

predictions1 = model.transform(test1)

After prediction we need to select those columns which are require in Black Friday competition submission.

df = predictions1.selectExpr("User\_ID as User\_ID", "Product\_ID as Product\_ID", 'prediction as Purchase')

Now we need to write the df in csv format for submission. To save the results in single partition we need to pass the coalesce(1), other wise we would have many partitions for a result.

df.coalesce(1).write.format("com.databricks.spark.csv").option("header", "true").save('submission.csv')