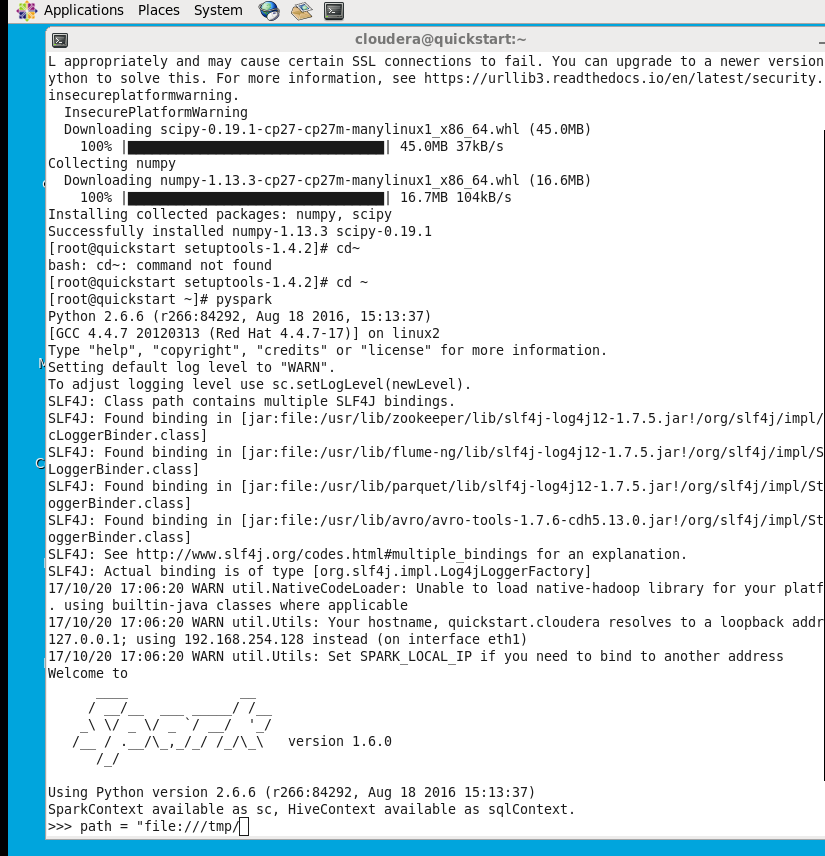
Attached file Regression Analysis with Spark MLlib.py contains all the code discussed in lecture. Be free to adopt that code to your needs**.** This code will work as is on your Cloudera VM with Spark 1.6. On your own VM with Spark 2.2, you might have to modify some functions with newer API calls.You are also welcome to switch from this mostly RDD based code to Data Frame based code offered in Spark 2.2. That is up to you.

**Problem 1.** Attached file auto\_mpg\_original.csv contains a set of data on automobile characteristics and fuel consumption. File auto\_mpg\_description.csv contains the description of the data. Import data into Spark. Randomly select 10-20% of you data for testing and use remaining data for training. Find all null values in all numerical columns. Replace nulls, if any, with average values for respective columns using Spark Data Frame API. (25%)



**#The above python version 2.6.6 does not support numpy installation. I upgraded to python 2.7.6**



>>>import os

>>>import pyspark.mllib

>>>import numpy as np

>>>from pyspark import SparkContext, SparkConf

>>>from pyspark.sql import Row

>>>from pyspark.mllib.regression import LabeledPoint

>>>from pyspark.mllib.regression import LinearRegressionWithSGD

>>>from pyspark.sql.functions import \*

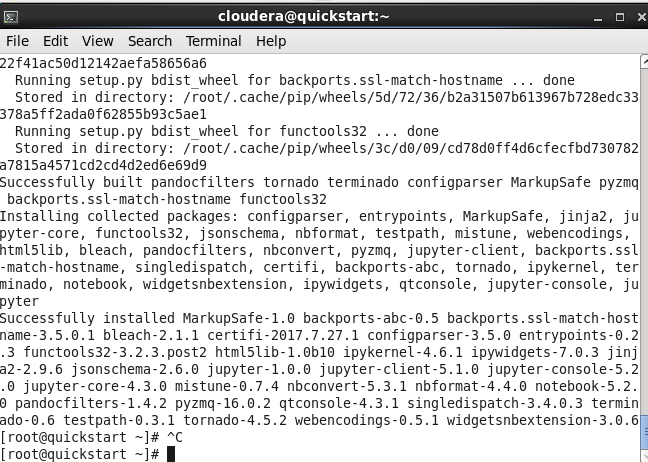
>>>from pyspark.sql import SQLContext

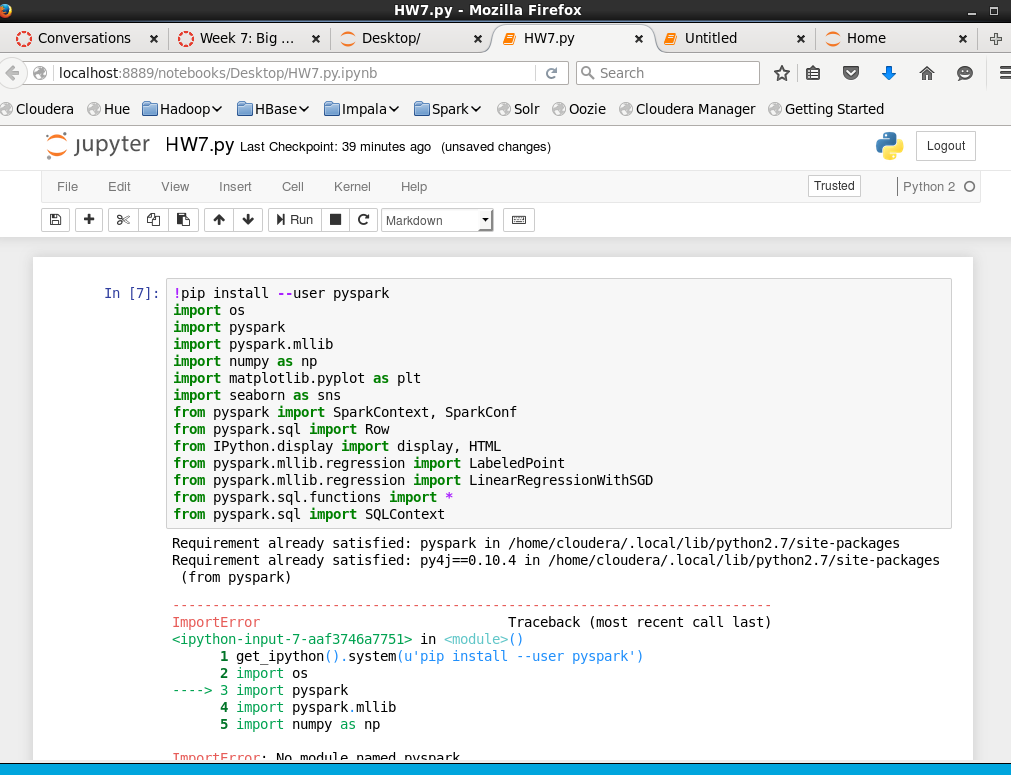
**#Install Jupyter**

[root@quickstart ~]# pip install ipython

[root@quickstart ~]#pip install –upgrade pip

[root@quickstart ~]#pip install jupyter





**#Jupyter does not cooperate, so I go back to Cloudera terminal to do HW, hopefully I can graph in Cloudera.**

**#Import data**

>>> original = sc.textFile('file:///home/cloudera/Desktop/auto\_mpg\_original-1.csv')

>>> num\_data = original.count()

>>> records = original.map(lambda x: x.split(","))

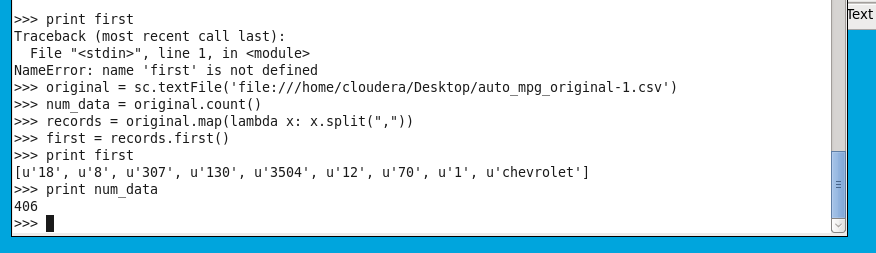
>>> first = records.first()

>>> print first

[u'18', u'8', u'307', u'130', u'3504', u'12', u'70', u'1', u'chevrolet']

>>> print num\_data

406



**#Mapp the RDD Columns in prep to clean up as wellas do preliminary Data Analysis**

>>> mappedrecords = records.map(lambda line: Row(mpg=line[0], cylinders=line[1], displacement=[2], horsepower=line[3], weight=line[4], acceleration=line[5], model\_year=line[6], origin=line[7], car\_name=line[8]))

>>> mappedrecords.take(5)

**#Clean up our Data remove the N/A**

>>> df = mappedrecords.toDF()

>>> df = df.filter(df['mpg'] > 0)

>>> df = df.filter(df['horsepower'] > 0)

>>> num\_data1 = df.count()

>>> print('Total Count:' + str(num\_data1))

Total Count:404

>>> df.take(5)



>>>avg = df.select(mean(df["horsepower"])).collect()

>>> print(avg)

[Row(avg(horsepower)=104.82425742574257)]  
>>>df.na.fill({"horsepower":avg[0][0]["avg(horsepower)"]})



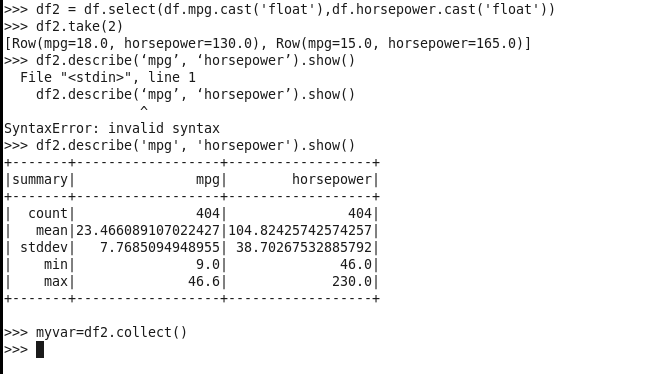
**#Summary of our Input Data**

>>>df2 = df.select(df.mpg.cast('float'),df.horsepower.cast('float'))

>>>df2.take(2)

>>>df2.describe(‘mpg’, ‘horsepower’).show()

>>>myvar=df2.collect()



**#Split data**

>>> test = df2.sample(False, 0.2, 42)

>>> train = df2.subtract(test)

>>> print df2.count()

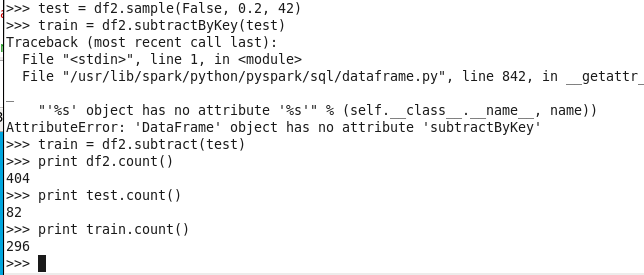
404

>>> print test.count()

82

>>> print train.count()

296



**Problem 2.** Look initially at two variables in the data set from the previous problem: the horsepower and the mpg (miles per gallon). Treat mpg as a feature and horsepower as the target variable (label). Use MLlib linear regression to identify the model for the relationship. Use the test data to illustrate accuracy of the linear regression model and its ability to predict the relationship. Calculate two standard measures of model accuracy. Create a diagram using any technique of convenience to presents the model (straight line), and the original test data. Please label your axes and use different colors for original data and predicted data.

(25%)

>>> from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD, LinearRegressionModel

#plot dataframe

>>> targets = df.select(df.mpg.cast('float')).rdd.flatMap(lambda x: x).collect()

>>> hist(targets, bins=40, color='lightblue', normed=True)

>>>fig = matplot.lib.pyplot.gcf()

>>>fig.set\_size\_inches(8,5)

**#LabeledPoint**

>>>transformed = df2.rdd.map(lambda r: LabeledPoint(r[0], r[1:]))

>>>print transformed.take(5)

**#Split data**

>>>trainingData, testingData = transformed.randomSplit([0.8, 0.2], seed=1234)

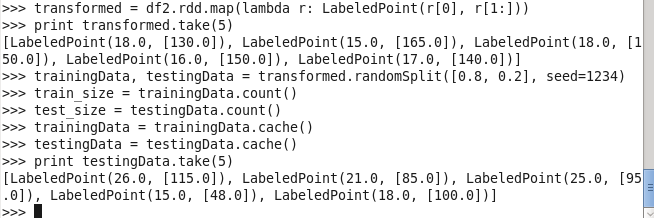
>>>train\_size = trainingData.count()

>>>test\_size = testingData.count()

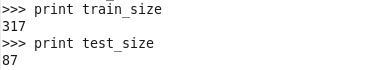
>>>trainingData = trainingData.cache()

>>>testingData = testingData.cache()

>>>print testingData.take(5)



**#Verify train size and test size**



**#Train model**

>>> linearModel = LinearRegressionWithSGD.train(trainingData, iterations=100, step=0.0001)

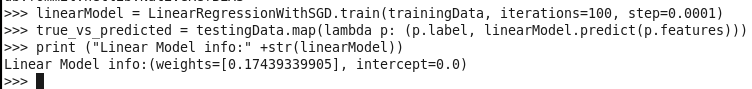
>>> true\_vs\_predicted = testingData.map(lambda p: (p.label, linearModel.predict(p.features)))

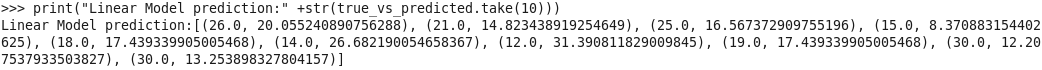
>>> print ("Linear Model info:" +str(linearModel))

**Linear Model info:(weights=[0.17439339905], intercept=0.0)**

>>> print("Linear Model prediction:" +str(true\_vs\_predicted.take(10)))

Linear Model prediction:[(26.0, 20.055240890756288), (21.0, 14.823438919254649), (25.0, 16.567372909755196), (15.0, 8.370883154402625), (18.0, 17.439339905005468), (14.0, 26.682190054658367), (12.0, 31.390811829009845), (19.0, 17.439339905005468), (30.0, 12.207537933503827), (30.0, 13.253898327804157)]



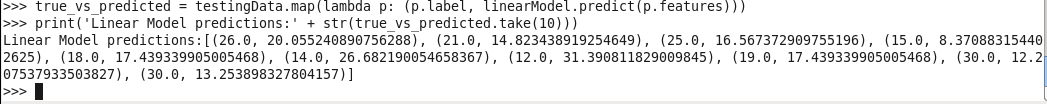


**#Predictions**

>>> true\_vs\_predicted = testingData.map(lambda p: (p.label, linearModel.predict(p.features)))

>>> print('Linear Model predictions:' + str(true\_vs\_predicted.take(10)))

Linear Model predictions:[(26.0, 20.055240890756288), (21.0, 14.823438919254649), (25.0, 16.567372909755196), (15.0, 8.370883154402625), (18.0, 17.439339905005468), (14.0, 26.682190054658367), (12.0, 31.390811829009845), (19.0, 17.439339905005468), (30.0, 12.207537933503827), (30.0, 13.253898327804157)]



**#Gather metrics**

>>>mse = true\_vs\_predicted.map(lambda (t,p): squared\_error(t,p)).mean()

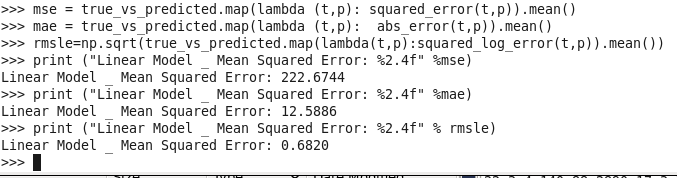
>>>mae = true\_vs\_predicted.map(lambda (t,p): abs\_error(t,p)).mean()

>>>rmsle=np.sqrt(true\_vs\_predicted.map(lambda(t,p):squared\_log\_error(t,p)).mean())

>>>print ("Linear Model \_ Mean Squared Error: %2.4f" %mse)

>>>print ("Linear Model \_ Mean Squared Error: %2.4f" %mae)

>>>print ("Linear Model \_ Mean Squared Error: %2.4f" % rmsle)



**#Plot for 20% of Train Data , in Jupyter (There is Jupyter issue)**

>>>plt.figure(1, figsize=(10,6))

>>>plt.scatter(tvp[‘mpg’], tvp[‘horsepower’], c= ‘b’, label= ‘hp actual’)

>>>plt.plot(tvp[‘mpg’], tvp[‘predicted horsepower’], c= ‘r’, label= ‘predicted hp’)

>>>plt.scatter(tvp[‘mpg’], tvp[‘predicted horsepower’], c= ‘r’, label= ‘predicted hp values’)

>>>plt.xlabel(‘mpg’)

>>>plt.ylabel(‘horsepower’)

>>>plt.title(‘Horsepower – Actual vs. Predictions 20% Test Data’)

>>>plt.legend()

>>>plt.show()

**Problem 3**. Consider attached file Bike-Sharing-Dataset.zip. This is the bike set discussed in class. Do not use all columns of the data set. Retain the following variables: season,yr,mnth,hr,holiday,weekday,workingday,weathersit,temp,atemp,hum,windspeed,cnt. Discard others. Regard cnt as the target variable and all other variables as features. Please note that some of those are categorical variables. Identify categorical variables and use 1-of-k binary encoding for those variables. If there are any null values in numerical columns, replace those with average values for those columns using Spark DataFrame API. Train your model using LinearRegressionSGD method. Use test data (15% of all) to assess the quality of prediction for cnt variable. Calculate at least two performance metrics of your model.

(25%)

**#Import data**

>>> data= sc.textFile('file:///home/cloudera/Desktop/hour.csv')

>>> num\_data = data.count()

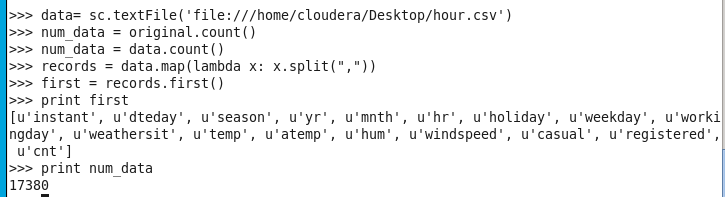
>>> records = data.map(lambda x: x.split(","))

>>> first = records.first()

>>> print first

[u'instant', u'dteday', u'season', u'yr', u'mnth', u'hr', u'holiday', u'weekday', u'workingday', u'weathersit', u'temp', u'atemp', u'hum', u'windspeed', u'casual', u'registered', u'cnt']

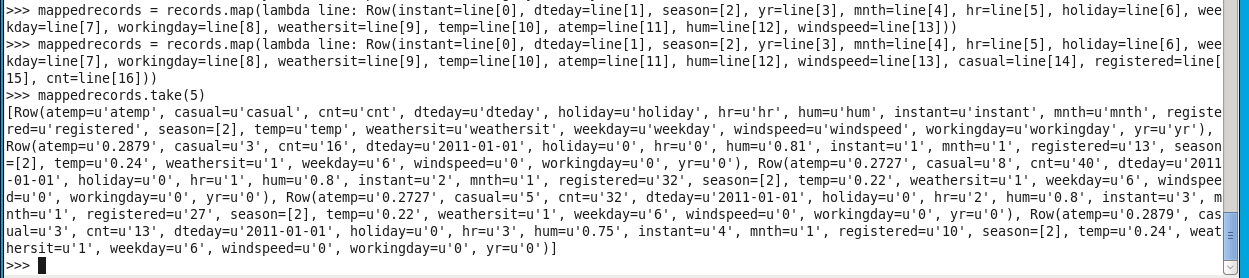
>>> print num\_data

17380

**#Mapp the RDD Columns in prep to clean up as wellas do preliminary Data Analysis**

>>> mappedrecords = records.map(lambda line: Row(instant=line[0], dteday=line[1], season=[2], yr=line[3], mnth=line[4], hr=line[5], holiday=line[6], weekday=line[7], workingday=line[8], weathersit=line[9], temp=line[10], atemp=line[11], hum=line[12], windspeed=line[13], casual=line[14], registered=line[15], cnt=line[16]))

>>> mappedrecords.take(5)



**#Clean up our Data remove the N/A**

>>> df = mappedrecords.toDF()

>>> df = df.filter(df['cnt'] > 0)

>>> df = df.filter(df['season'] > 0)

>>> df = df.filter(df['yr'] > 0)

>>> df = df.filter(df['mnth'] > 0)

>>> df = df.filter(df['hr'] > 0)

>>> df = df.filter(df['holiday'] > 0)

>>> df = df.filter(df['weekday'] > 0)

>>> df = df.filter(df['workingday'] > 0)

>>> df = df.filter(df['weathersit'] > 0)

>>> df = df.filter(df['temp'] > 0)

>>> df = df.filter(df['atemp'] > 0)

>>> df = df.filter(df['hum'] > 0)

>>> df = df.filter(df['windspeed'] > 0)

>>> num\_data1 = df.count()

>>> print('Total Count:' + str(num\_data1))

Total Count:404

>>> df.take(5)

>>>avg = df.select(mean(df["horsepower"])).collect()

>>> print(avg)

[Row(avg(horsepower)=104.82425742574257)]  
>>>df.na.fill({"horsepower":avg[0][0]["avg(horsepower)"]})

**#Summary of our Input Data**

>>>df2 = df.select(df.season.cast('float'),df.cnt.cast('float'))

>>>df3 = df.select(df.yr.cast('float'),df.cnt.cast('float'))

>>>df4 = df.select(df.mnth.cast('float'),df.cnt.cast('float'))

>>>df5 = df.select(df.hr.cast('float'),df.cnt.cast('float'))

>>>df6 = df.select(df.holiday.cast('float'),df.cnt.cast('float'))

>>>df7 = df.select(df.weekday.cast('float'),df.cnt.cast('float'))

>>>df8 = df.select(df.workingday.cast('float'),df.cnt.cast('float'))

>>>df9 = df.select(df.weathersit.cast('float'),df.cnt.cast('float'))

>>>df10 = df.select(df.temp.cast('float'),df.cnt.cast('float'))

>>>df11 = df.select(df.atemp.cast('float'),df.cnt.cast('float'))

>>>df12 = df.select(df.hum.cast('float'),df.cnt.cast('float'))

>>>df13 = df.select(df.windspeed.cast('float'),df.cnt.cast('float'))

>>>myvar2=df2.collect()

>>>myvar3=df3.collect()

>>>myvar4=df4.collect()

>>>myvar5=df5.collect()

**#Split data**

>>> test = df2.sample(False, 0.2, 42)

>>> train = df2.subtract(test)

**#LabeledPoint**

>>>transformed = df2.rdd.map(lambda r: LabeledPoint(r[0], r[1:]))

>>>print transformed.take(5)

**#Split data**

>>>trainingData, testingData = transformed.randomSplit([0.8, 0.2], seed=1234)

>>>train\_size = trainingData.count()

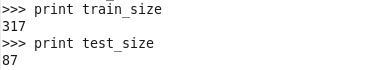
>>>test\_size = testingData.count()

>>>trainingData = trainingData.cache()

>>>testingData = testingData.cache()

>>>print testingData.take(5)

**#Verify train size and test size**



**#Train model**

>>> linearModel = LinearRegressionWithSGD.train(trainingData, iterations=200, step=0.01, intercept=False)

>>> true\_vs\_predicted = testingData.map(lambda p: (p.label, linearModel.predict(p.features)))

>>> print ("Linear Model info:" +str(linearModel))

**Linear Model info:(weights=[0.17439339905], intercept=0.0)**

>>> print("Linear Model prediction:" +str(true\_vs\_predicted.take(10)))

Linear Model prediction:[(26.0, 20.055240890756288), (21.0, 14.823438919254649), (25.0, 16.567372909755196), (15.0, 8.370883154402625), (18.0, 17.439339905005468), (14.0, 26.682190054658367), (12.0, 31.390811829009845), (19.0, 17.439339905005468), (30.0, 12.207537933503827), (30.0, 13.253898327804157)]

**#Predictions**

>>> true\_vs\_predicted = testingData.map(lambda p: (p.label, linearModel.predict(p.features)))

>>> print('Linear Model predictions:' + str(true\_vs\_predicted.take(10)))

Linear Model predictions:[(26.0, 20.055240890756288), (21.0, 14.823438919254649), (25.0, 16.567372909755196), (15.0, 8.370883154402625), (18.0, 17.439339905005468), (14.0, 26.682190054658367), (12.0, 31.390811829009845), (19.0, 17.439339905005468), (30.0, 12.207537933503827), (30.0, 13.253898327804157)]

#Prediction:

True\_vs\_predicted = data.map(lambda p: (p.label, linear\_model.predict(p.features))

Print “Linear Model predictions:” + str(true\_vs\_predicted.take(5))

**#Gather metrics**

>>>mse = true\_vs\_predicted.map(lambda (t,p): squared\_error(t,p)).mean()

>>>mae = true\_vs\_predicted.map(lambda (t,p): abs\_error(t,p)).mean()

>>>rmsle=np.sqrt(true\_vs\_predicted.map(lambda(t,p):squared\_log\_error(t,p)).mean())

>>>print ("Linear Model \_ Mean Squared Error: %2.4f" %mse)

>>>print ("Linear Model \_ Mean Squared Error: %2.4f" %mae)

>>>print ("Linear Model \_ Mean Squared Error: %2.4f" % rmsle)

**Problem 4**. Use a Decision Tree model to predict mpg values in auto\_mpg\_original.txt data. Assess accuracy of your prediction using at least two performance metrics.

(25%)

#Train model

>>>from pyspark.mllib.tree import DecisionTree

>>>from pyspark.mllib.tree import LabeledPoint

>>>dt\_model = DecisionTree.trainRegressor(trainingData,{})

>>>preds = dt\_model.predict(trainingData.map(lambda p: p.features))

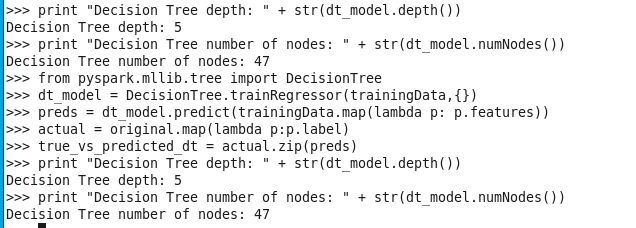
>>> actual = original.map(lambda p:p.label)

>>> true\_vs\_predicted\_dt = actual.zip(preds)

>>>print "Decision Tree predictions: " + str(true\_vs\_predicted\_dt.take(5))

>>>print "Decision Tree depth: " + str(dt\_model.depth())

>>>print "Decision Tree number of nodes: " + str(dt\_model.numNodes())



**Predictions**

>>> true\_vs\_predicted = testingData.map(lambda p: (p.label, linearModel.predict(p.features)))

>>> print('Linear Model predictions:' + str(true\_vs\_predicted.take(10)))

Linear Model predictions:[(26.0, 20.055240890756288), (21.0, 14.823438919254649), (25.0, 16.567372909755196), (15.0, 8.370883154402625), (18.0, 17.439339905005468), (14.0, 26.682190054658367), (12.0, 31.390811829009845), (19.0, 17.439339905005468), (30.0, 12.207537933503827), (30.0, 13.253898327804157)]

**#Gather metrics**

>>>mse = true\_vs\_predicted.map(lambda (t,p): squared\_error(t,p)).mean()

>>>mae = true\_vs\_predicted.map(lambda (t,p): abs\_error(t,p)).mean()

>>>rmsle=np.sqrt(true\_vs\_predicted.map(lambda(t,p):squared\_log\_error(t,p)).mean())

>>>print ("Linear Model \_ Mean Squared Error: %2.4f" %mse)

>>>print ("Linear Model \_ Mean Squared Error: %2.4f" %mae)

>>>print ("Linear Model \_ Mean Squared Error: %2.4f" % rmsle)