HU Extension Assignment 11 E63 Big Data Analytics

Handed out: 04/16/2016 Due by 11:30PM EST on Friday, 04/22/2016

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document with explanations of the purpose of those commands. We cannot retype text that is in JPG images. Please, always submit a separate copy of the original, working scripts and/or class files you used. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portions of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. For issues and comments visit the class Discussion Board. You are not obliged to use Java or Eclipse. You are welcome to use any language and any IDE of your choice.

Problem 1. Remove the header of the attached Samll_Car_Data.csv file and then import it into Spark. Randomly select 10% of you data for testing and use remaining data for training. Look initially at horsepower and displacement. Treat displacement as a feature and horsepower as the target variable. Use MLlib linear regression to identify the model for the relationship.
Use test data to illustrate accuracy of your ability to predict the relationship.

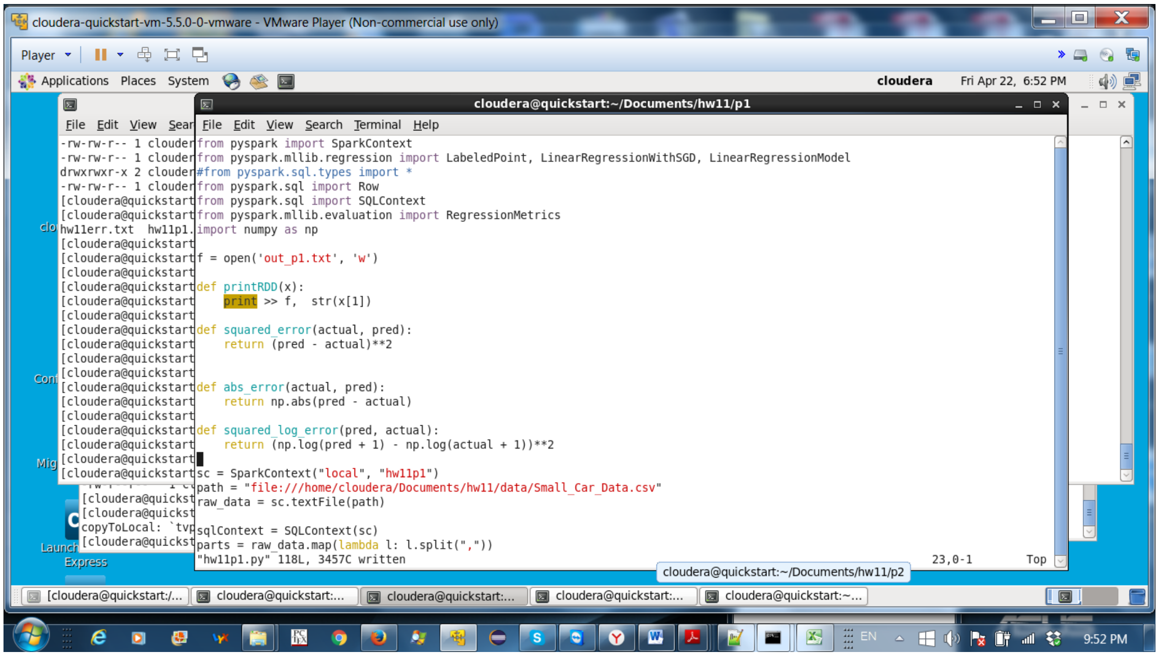
```
# Solution for hw11 problem 1
# Author: S.K.
# Last Modified 22 April 2016
# Problem:
# Remove the header of the attached Samll_Car_Data.csv file and then
# import it into Spark. Randomly select 10% of you data for testing and use remaining data
# for training. Look initially at horsepower and displacement. Treat displacement as a
# feature and horsepower as the target variable. Use MLlib linear regression to identify the model
# for the relationship.


from pyspark import SparkContext
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD, LinearRegressionModel
from pyspark.sql import Row
from pyspark.sql import SQLContext
from pyspark.mllib.evaluation import RegressionMetrics
import numpy as np

#open a file
f = open('out_p1.txt', 'w')


#####################################################################
#### Some functions for evaluations of the results ###############
#####################################################################

#print content of RDD for debugging purposes
def printRDD(x):
```

```
    print >> f,  str(x[1])

# Compute the square of the distance
def squared_error(actual, pred):
    return (pred - actual)**2

# Compute absolute error
def abs_error(actual, pred):
    return np.abs(pred - actual)

# Compute log of absolute error
def squared_log_error(pred, actual):
    return (np.log(pred + 1) - np.log(actual + 1))**2




###############  Start Spark #################################
# get context, data and split the data
sc = SparkContext("local", "hw11p1")
path = "file:///home/cloudera/Documents/hw11/data/Small_Car_Data.csv"
raw_data = sc.textFile(path)

sqlContext = SQLContext(sc)
parts = raw_data.map(lambda l: l.split(","))

pre_df=parts.map(lambda p: Row(displacement = p[3],hspower = p[4]))

# create dataframe for cleaning the data later on
df=sqlContext.createDataFrame(pre_df)

# Count the number of rows before cleaning the data ( via filtering)
print >> f, "Before filtering count="
print >> f, df.count()

# cleaning the data
dff = df.where( (df.displacement != 'NaN')  &  ( df.hspower != 'NaN'))

# Count the number of rows after cleaning the data ( via filtering)
print >> f, "After filtering count="
print >> f,dff.count()


# inspect the data
dff.show(300)


#leave this line
#df_lp=dff.map(lambda line: LabeledPoint(line[0], [line[1:]]))

# create a dataframe with labeledpoints, which are the input to spark regressionss
df_lp=dff.map(lambda line: LabeledPoint(line.hspower, [line.displacement]))

# inspect the data
print >> f, df_lp.take(4)
```

```
# split the data into training and testing parts
trainingData, testingData = df_lp.randomSplit([.9,.1],seed=1234)

#evaluate the regression
model = LinearRegressionWithSGD.train( trainingData  , iterations=2000, step=0.0001,
initialWeights=[1.0], intercept=True  )



# print out the regression results
print >> f,("############################## MODEL ESTIMATION  RESULTS1 starts
################################")
print >> f,(model)
print >>f,("############################## MODEL ESTIMATION  RESULTS1 ends
################################")


# compute different measures of predictions
true_vs_predicted = df_lp.map(lambda p: (p.label, model.predict(p.features)))

valuesAndPreds = df_lp.map(lambda p: (float(model.predict(p.features)), p.label))

true_vs_predicted_testing = testingData.map(lambda p: (p.label, model.predict(p.features)))



# compute additional metrics of regression quality
metrics = RegressionMetrics( valuesAndPreds  )

print >> f, "metrics.r2="
print >> f,  metrics.r2

mse = true_vs_predicted_testing.map(lambda (t, p): squared_error(t, p)).mean()
mae = true_vs_predicted_testing.map(lambda (t, p): abs_error(t, p)).mean()

print >> f,  "Linear Model - Mean Squared Error: %2.4f" % mse
print >> f,  "Linear Model - Mean Absolute Error: %2.4f" % mae


# save the results of regressions and predictions in the hdfs dfs

true_vs_predicted.map(lambda r: [r] ).saveAsTextFile("true_vs_predicted_p1")
true_vs_predicted_testing.map(lambda r: [r] ).saveAsTextFile("true_vs_predicted_testing_p1")
valuesAndPreds.map(lambda r: [r] ).saveAsTextFile("valuesAndPreds_p1")

# close the file for intermediate output
f.close()
```

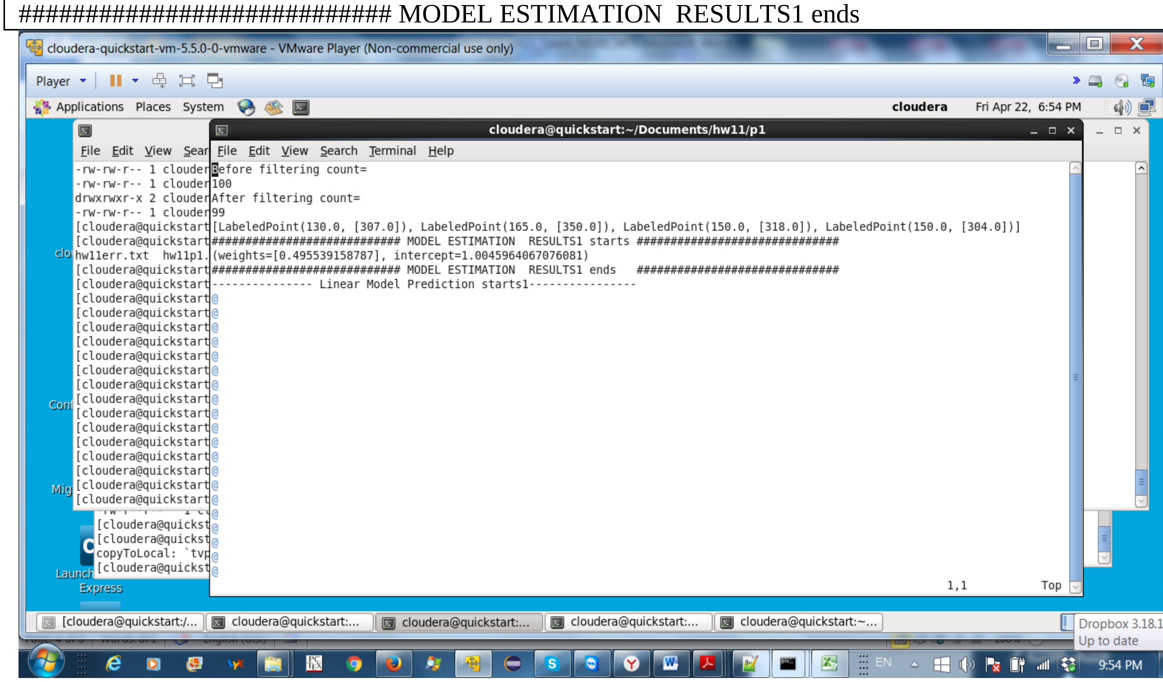
TO run:

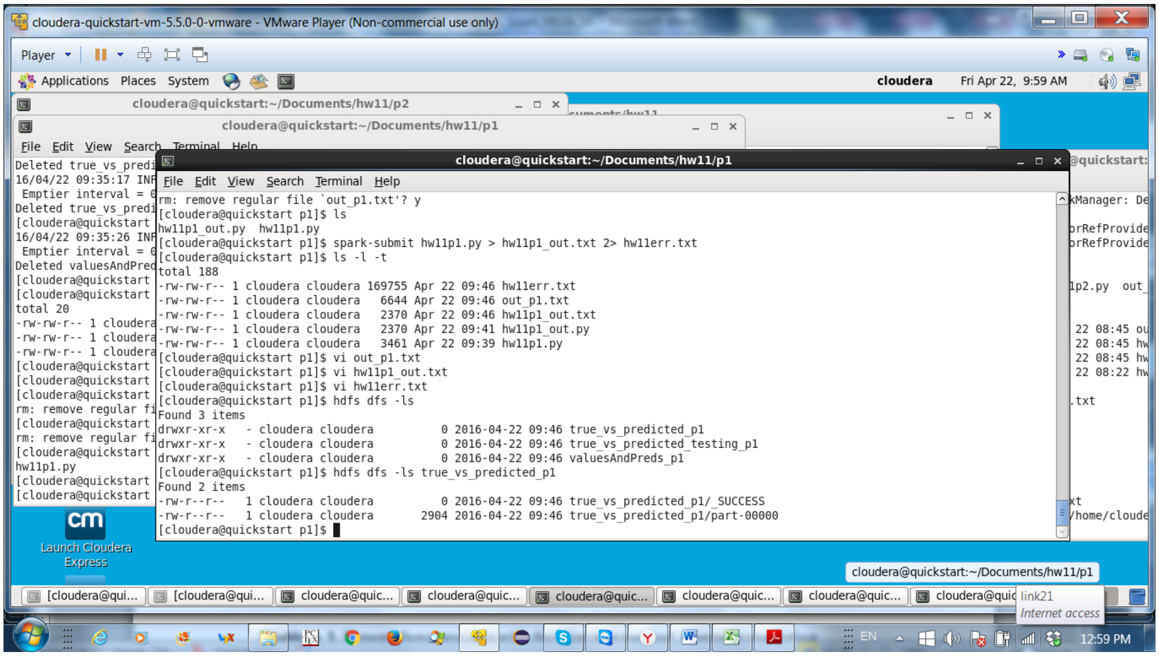
```
spark-submit hw11p1.py > hw11p1_out.txt
```



OUTPUT:

```
############################## MODEL ESTIMATION  RESULTS1 starts
(weights=[0.495539158787], intercept=1.0045964067076081)
############################## MODEL ESTIMATION  RESULTS1 ends
```





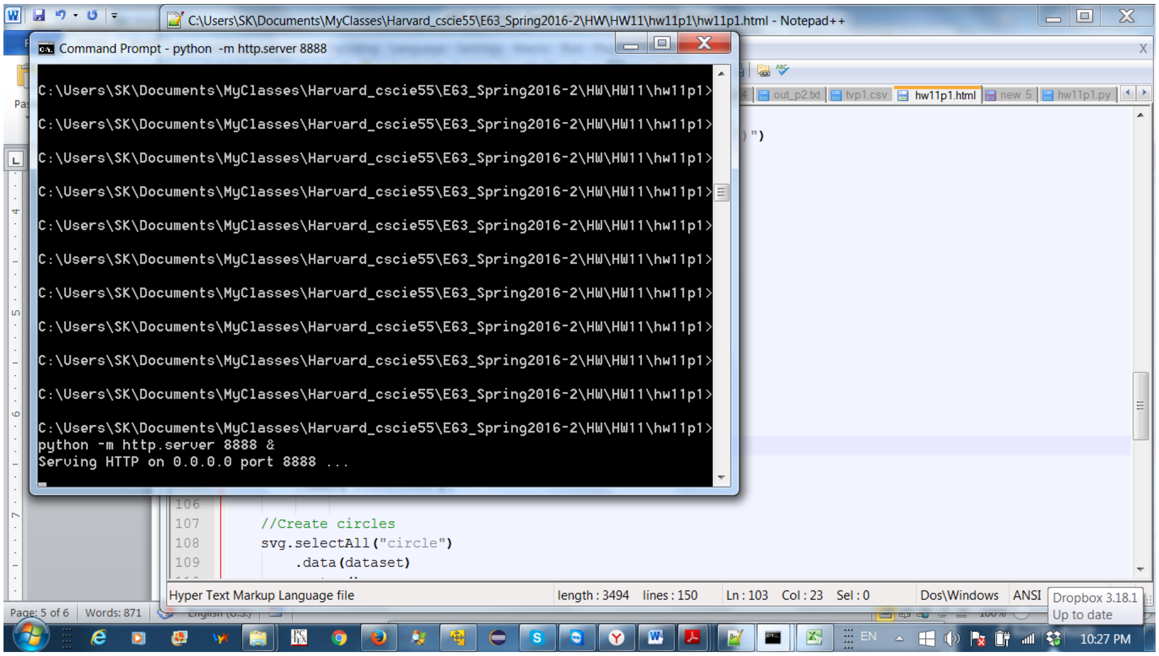
Prediction metrics:

--------------Linear Model Prediction ends1 -------------------
metrics.r2=
0.709215359906
Linear Model - Mean Squared Error: 681.3341
Linear Model - Mean Absolute Error: 22.9580

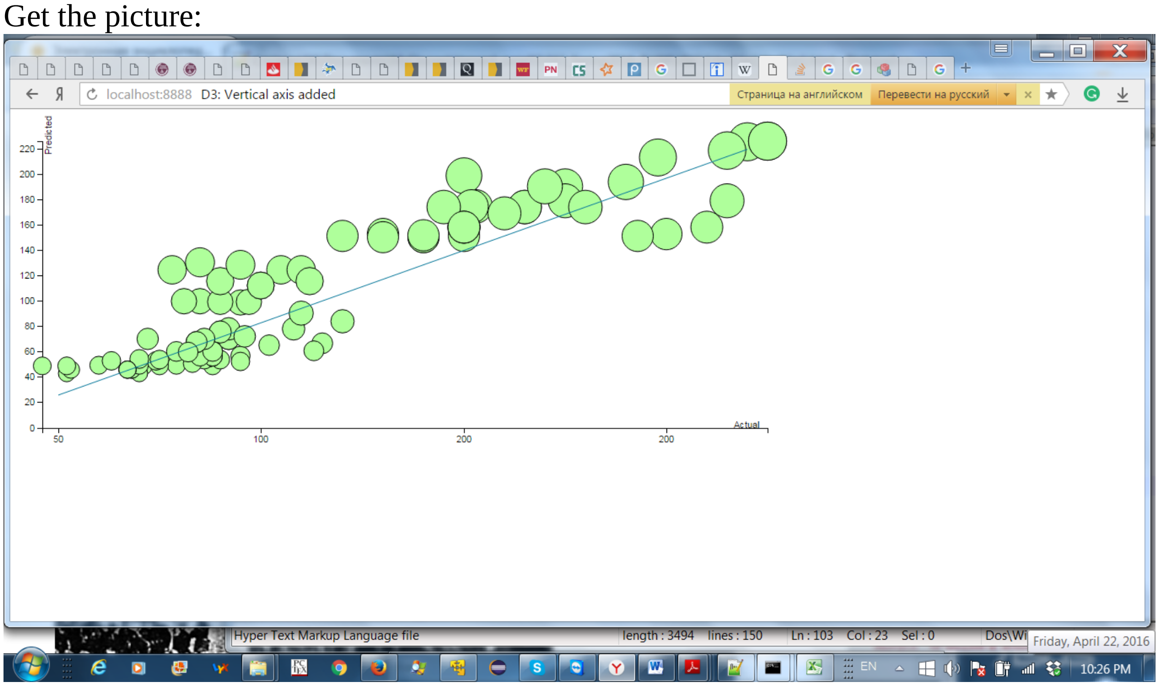
Create a diagram using D3 which presents the model (straight line), original test data and predictions of your analysis. Please label your axes and use different colors for original data and predicted data.

Start server

```
python -m http.server 8888 &
```



Get the picture:




```
<!DOCTYPE html>
<html lang="en">
        <head>
                <meta charset="utf-8">
                <title>D3: Vertical axis added</title>
                <script type="text/javascript" src="d3/d3.js"></script>
```

```
		<style type="text/css">

			.axis path,
			.axis line {
				fill: none;
				stroke: black;
				shape-rendering: crispEdges;
			}

			.axis text {
				font-family: sans-serif;
				font-size: 11px;
			}

		</style>
	</head>
	<body>
		<script type="text/javascript">



			//Width and height
			var w = 950;
			var h = 400;
			var padding = 30;

			//Create scale functions
			var xScale = d3.scale.linear();
			var yScale = d3.scale.linear();
			var rScale = d3.scale.linear();

			//Format X axis
			//var f = d3.formatPrefix(",.0", 1e+2);
			//var formatAsPercentage = d3.format(".1%");
			var xAxis = d3.svg.axis();


			//Define Y axis
			var yAxis = d3.svg.axis();

			//Create SVG element
			var svg = d3.select("body")
						.append("svg")
						.attr("width", w)
						.attr("height", h);




d3.csv("tvp1.csv",function(d)
{
	return{
		Actual: +d.Actual ,
		Predicted:+d.Predicted,
	};
},function(error, dataset) {
```

```
        xScale.domain([d3.min(dataset, function(d) { return d.Actual; }),
                                        d3.max(dataset, function(d) { return d.Actual; }) ])
                                        .rangeRound([padding, w - padding * 2]);

        yScale.domain([0, d3.max(dataset, function(d) { return d.Predicted; })])
        .range([h - padding, padding]);

        rScale.domain([0, d3.max(dataset, function(d) { return Math.abs(d.Predicted/d.Actual-1); })])
        .range([0.1, 1]);

        //Continue creating scale for X axis
        xAxis.scale(xScale)
                .orient("bottom")
                .tickFormat(d3.format(".1s"))
                .ticks(5, "k");
        //Continue creating  scale for Y axis
        yAxis.scale(yScale)
                .orient("left")
                .ticks(10);

        //Create X axis
        svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(0," + (h - padding) + ")")
                .call(xAxis)
                .append("text")
    //.attr("x", 16)
    .attr("dx", "80em")
    .style("text-anchor", "end")
    .text("Actual");

        //Create Y axis
        svg.append("g")
                .attr("class", "axis")
                .attr("transform", "translate(" + padding + ",0)")
                .call(yAxis)
                .append("text")
    .attr("transform", "rotate(-90)")
    .attr("y", 0)
    .attr("dy", "1em")
    .style("text-anchor", "end")
    .text("Predicted");

        //Create circles
        svg.selectAll("circle")
                .data(dataset)
                .enter()
                .append("circle")
                .attr("stroke", "black")
                .attr("fill", function(d){
                return "rgba(175, 255, 155,1)";
                })
                .attr("cx", function(d) {
                        return xScale(parseFloat(d.Actual));
                })
```

```
                    .attr("cy", function(d) {
                              return yScale(parseFloat(d.Predicted));
                    })
                    .attr("r", function(d) {
                              return rScale(Math.sqrt(d.Predicted));
                    });

        svg.append("line")
                  .attr("x1", xScale(50))
                   .attr("y1", yScale(26))
                  .attr("x2", xScale(220))
                  .attr("y2", yScale(220))
                  .attr("stroke-width", 1)
                  .attr("stroke", "rgb(6,120,155)");


           //Hover the mouse
           d3.selectAll("circle")
                    .data(dataset)
                    .on("mouseover", function(d)
                    {
                              d3.select(this)
                              .append("title")
                              .text(function(d) {
                               return Math.abs(d.Predicted/d.Actual-1);
                               });
                    });
});
</script>

</body>
</html>
```

Problem 2. Treat: cylinders, displacement, manufacturer, model_year, origin and weight as features and use linear regression to predict two target variable: horsepower and acceleration. Please note that some of those are categorical variables.
The code:

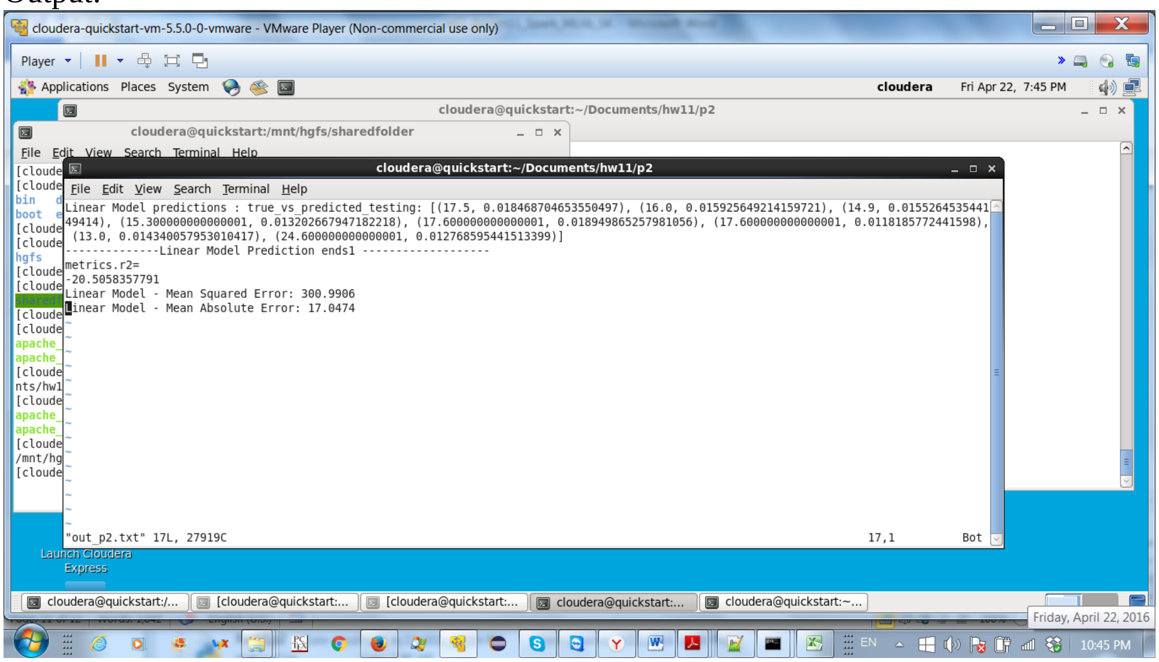
```
from pyspark import SparkContext
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD, LinearRegressionModel
from pyspark.sql import Row
from pyspark.sql import SQLContext
from pyspark.mllib.evaluation import RegressionMetrics
import numpy as np

f = open('out_p2.txt', 'w')

def printRDD(x):
   print >> f,  str(x[1])

def squared_error(actual, pred):
   return (pred - actual)**2
```

```
def abs_error(actual, pred):
    return np.abs(pred - actual)

def squared_log_error(pred, actual):
    return (np.log(pred + 1) - np.log(actual + 1))**2




sc = SparkContext("local", "hw11p1")
path = "file:///home/cloudera/Documents/hw11/data/Small_Car_Data.csv"
raw_data = sc.textFile(path)

sqlContext = SQLContext(sc)
parts = raw_data.map(lambda l: l.split(","))

pre_df=parts.map(lambda p: Row( accel = p[1], cyl=p[2],displacement = p[3],hspower = p[4], manuf= 
p[5], myear=p[7], origin=p[9], weight=p[10]))


df=sqlContext.createDataFrame(pre_df)

print >> f, "Before filtering count="
print >> f, df.count()


dff = df.where( (df.accel != 'NaN')  &   (df.displacement != 'NaN')  &  ( df.hspower != 'NaN') &  ( df.cyl != 
'NaN') &  ( df.manuf != 'NaN') &  ( df.myear != 'NaN')   & (df.origin != 'NaN') & (df.weight != 'NaN')  )



def get_mapping(rdd, idx):
    return rdd.map(lambda fields: fields[idx]).distinct().zipWithIndex().collectAsMap()



print "Mapping of first categorical feature column-Manuf : %s" % get_mapping(dff, 4)
print "Mapping of first categorical feature column-ModelYear : %s" % get_mapping(dff, 5)
print '\n'
print "Mapping of first categorical feature column-Origin : %s" % get_mapping(dff, 6)

mappings = [get_mapping(dff, 4), get_mapping(dff, 5),get_mapping(dff, 6)  ]

cat_len = sum(map(len, mappings))
num_len = 3
total_len = num_len + cat_len

print "Feature vector length for categorical features: %d" % cat_len
print "Feature vector length for numerical features: %d" % num_len
print "Total feature vector length: %d" % total_len

def extract_features(record, cat_len ):
    cat_vec = np.zeros(cat_len)
    i = 0
```

```
    step = 0
    for field in [record[4], record[5], record[6]  ] :
        m = mappings[i]
        idx = m[field]
        cat_vec[idx + step] = 1
        i = i + 1
        step = step + len(m)
    num_vec = np.array([float(field) for field in [record[1], record[2], record[7] ] ])
    return np.concatenate((cat_vec, num_vec))



def extract_label(record):
    return record[0]



df_lp  = dff.map(lambda r: LabeledPoint( extract_label(r)   ,extract_features(r,cat_len  )))


trainingData, testingData = df_lp.randomSplit([.9,.1],seed=1234)

print "trainingData.take(4)="
print trainingData.take(20)


model = LinearRegressionWithSGD.train( trainingData  , iterations=20, step=0.00000000001,
initialWeights= [0.000005 for x in range(1, 41)], intercept=False )




print >> f,("############################ MODEL ESTIMATION  RESULTS1 starts ")
print >> f,(model)
print >>f,("############################ MODEL ESTIMATION  RESULTS1 ends   #")


true_vs_predicted = df_lp.map(lambda p: (p.label, model.predict(p.features)))

valuesAndPreds = df_lp.map(lambda p: (float(model.predict(p.features)), p.label))

true_vs_predicted_testing = testingData.map(lambda p: (p.label, model.predict(p.features)))


print >> f, ("--------------- Linear Model Prediction starts1----------------")
print >> f,  "Linear Model predictions : true_vs_predicted: " + str(true_vs_predicted.take(200))

print >> f,  "Linear Model predictions : valuesAndPreds: " + str(valuesAndPreds.take(200))

print >> f,  "Linear Model predictions : true_vs_predicted_testing: " +
str(true_vs_predicted_testing.take(200))



metrics = RegressionMetrics( valuesAndPreds  )
```

```
print >> f, "metrics.r2="
print >> f,  metrics.r2

mse = true_vs_predicted_testing.map(lambda (t, p): squared_error(t, p)).mean()
mae = true_vs_predicted_testing.map(lambda (t, p): abs_error(t, p)).mean()

print >> f,  "Linear Model - Mean Squared Error: %2.4f" % mse
print >> f,  "Linear Model - Mean Absolute Error: %2.4f" % mae


f.close()
```

Output:



Use test data to assess quality of prediction for both target variables. Which of two target variables is easier to predict, in the sense that predicted values differ less from the original values.

Response:
It is easier to predict using the model in problem1 because this model does not provide reliable results and might suffer from overidentification. Also I am not sure that the method of dealing with categorical data as presented in the lecture was correct from statistical point of view.

Problem 3. Repeat above analysis with decision tree method. Compare predicting ability/quality of this technique with that of the linear regression.

Code:

```
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.util import MLUtils

from pyspark import SparkContext
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD, LinearRegressionModel
from pyspark.sql import Row
from pyspark.sql import SQLContext
from pyspark.mllib.evaluation import RegressionMetrics
import numpy as np


path = "file:///home/cloudera/Documents/hw11/data/Small_Car_Data.csv"
sc = SparkContext("local", "hw11p3")

# Load and parse the data file into an RDD of LabeledPoint.

data = sc.textFile(path)
sqlContext = SQLContext(sc)

parts = data.map(lambda l: l.split(","))
pre_df=parts.map(lambda p: Row(displacement = p[3],hspower = p[4]))

df=sqlContext.createDataFrame(pre_df)

dff = df.where( (df.displacement != 'NaN')  &  ( df.hspower != 'NaN'))

df_lp=dff.map(lambda line: LabeledPoint(line.hspower, [line.displacement]))

# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = df_lp.randomSplit([0.7, 0.3])

# Train a DecisionTree model.
#  Empty categoricalFeaturesInfo indicates all features are continuous.
model = DecisionTree.trainRegressor(trainingData, categoricalFeaturesInfo={}, impurity='variance', maxDepth=5,
maxBins=32)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testMSE = labelsAndPredictions.map(lambda (v, p): (v - p) * (v - p)).sum()/float(testData.count())
print('Test Mean Squared Error = ' + str(testMSE))
print('Learned regression tree model:')
print(model.toDebugString())
```

Output:

```
Test Mean Squared Error = 729.854731639
Learned regression tree model:
DecisionTreeModel regressor of depth 5 with 37 nodes
  If (feature 0 <= 258.0)
   If (feature 0 <= 105.0)
    If (feature 0 <= 97.0)
     If (feature 0 <= 85.0)
      Predict: 61.0
     Else (feature 0 > 85.0)
      If (feature 0 <= 90.0)
       Predict: 70.0
      Else (feature 0 > 90.0)
       Predict: 64.625
    Else (feature 0 > 97.0)
```

```
    If (feature 0 <= 101.0)
     If (feature 0 <= 98.0)
      Predict: 69.66666666666667
     Else (feature 0 > 98.0)
      Predict: 83.0
    Else (feature 0 > 101.0)
     Predict: 68.5
  Else (feature 0 > 105.0)
   If (feature 0 <= 119.0)
    If (feature 0 <= 110.0)
     If (feature 0 <= 107.0)
      Predict: 83.66666666666667
     Else (feature 0 > 107.0)
      Predict: 78.5
    Else (feature 0 > 110.0)
     If (feature 0 <= 113.0)
      Predict: 88.8
     Else (feature 0 > 113.0)
      Predict: 81.5
   Else (feature 0 > 119.0)
    If (feature 0 <= 130.0)
     Predict: 107.5
    Else (feature 0 > 130.0)
     If (feature 0 <= 140.0)
      Predict: 85.0
     Else (feature 0 > 140.0)
      Predict: 96.29411764705883
 Else (feature 0 > 258.0)
  If (feature 0 <= 400.0)
   If (feature 0 <= 305.0)
    If (feature 0 <= 302.0)
     Predict: 136.66666666666666
    Else (feature 0 > 302.0)
     Predict: 145.0
   Else (feature 0 > 305.0)
    If (feature 0 <= 307.0)
     Predict: 200.0
    Else (feature 0 > 307.0)
     If (feature 0 <= 351.0)
      Predict: 161.5
     Else (feature 0 > 351.0)
      Predict: 179.16666666666666
  Else (feature 0 > 400.0)
   Predict: 225.0
```

Decision Tree and Regressin model: prediction of perfomance
The decision tree has Test Mean Squared Error = 729.854731639, but the regression model has Mean Squared Error: 681.3341. These numbers are statistically quite close, so I would say that both methods produce equally valid results.