Hanjiao Qiu
CSCIE63 Big Data Analytics
Assignment 11

Assignment 11 Solution

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. 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.

Solution:

1. Install Anaconda and IPython. Setup the environment variables in ".bash_profile". If we set "IPYTHON_OPTS = notebook" and run "pyspark", a web server will be started and the default browser will pop-up open at port 8888.

```
[cloudera@localhost Downloads]$ bash Anaconda2-4.0.0-Linux-x86_64.sh

Welcome to Anaconda2 4.0.0 (by Continuum Analytics, Inc.)

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
================
Anaconda License
================

Copyright 2016, Continuum Analytics, Inc.

All rights reserved under the 3-clause BSD License:

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
```

```
cloudera@localhost:~
File  Edit  View  Search  Terminal  Help
# User specific environment and startup programs

PATH=$PATH:$HOME/binVA_HOME=/usr/java/jdk1.8.0_60
export JAVA_HOME

M2_HOME=/usr/local/apache-maven-3.3.9
export M2_HOME

SPARK_HOME=/usr/lib/spark
export SPARK_HOME

ANACONDA_HOME=/home/cloudera/anaconda2
export ANACONDA_HOME

PATH=$PATH:$HOME/bin:$JAVA_HOME/bin:$M2_HOME/bin:$SPARK_HOME/bin:$ANACONDA_HOME/
bin
export PATH

IPYTHON=1
export IPYTHON
IPYTHON_OPTS=notebook
export IPYTHON_OPTS
```

2. Remove the header of the attached Samll_Car_Data.csv file and then import it into Spark.

```
[cloudera@localhost Week11_SparkML]$ sed 1d Small_Car_Data.csv > car_noheader.csv
```

```
localhost:8888/notebooks/linearRegressionSimple.ipynb
jupyter  linearRegressionSimple  Last Checkpoint: 13 hours ago (unsaved changes)
File  Edit  View  Insert  Cell  Kernel  Help
Code  CellToolbar

In [1]: import pyspark.mllib
        from pyspark.mllib.regression import LinearRegressionWithSGD
        from pyspark.mllib.regression import LabeledPoint
        from pyspark.sql.functions import *
        import numpy as np

In [2]: path = "file:///home/cloudera/Documents/hw11/Week11_SparkML/car_noheader.csv"

In [3]: raw_data = sc.textFile(path)

In [4]: num_data = raw_data.count()

In [5]: raw_data.take(5)

Out[5]: [u'1,12,8,307,130,chevrolet    ,chevrolet chevelle malibu        ,70,18,USA    ,3504',
         u'2,11.5,8,350,165,buick        ,buick skylark 320              ,70,15,USA    ,3693',
         u'3,11,8,318,150,plymouth     ,plymouth satellite             ,70,18,USA    ,3436',
         u'4,12,8,304,150,amc          ,amc rebel sst                  ,70,16,USA    ,3433',
         u'5,10.5,8,302,140,ford         ,ford torino                    ,70,17,USA    ,3449']
```

In [6]:
```
records = raw_data.map(lambda x: x.split(","))
```

In [7]:
```
first = records.first()
```

In [8]:
```
print first
```

```
[u'1', u'12', u'8', u'307', u'130', u'chevrolet    ', u'chevrolet chevelle malibu        ', u'70', u'18', u'USA    ', u'3
504']
```

In [9]:
```
print num_data
```

```
99
```

In [10]:
```
records.take(1)
```

Out[10]:
```
[[u'1',
  u'12',
  u'8',
  u'307',
  u'130',
  u'chevrolet    ',
  u'chevrolet chevelle malibu        ',
  u'70',
  u'18',
  u'USA    ',
  u'3504']]
```

3. Treat displacement as a feature and horsepower as the target variable. Create LabeledPoint using Linear Regression model. Since the displacement is a numerical feature, we don't need to do the binary encoding here. Randomly select 10% of the data for testing and the remaining 90% for training.

In [11]:
```
data = records.map(lambda line:LabeledPoint(line[4], [line[3]]))
data.take(5)
```

Out[11]:
```
[LabeledPoint(130.0, [307.0]),
 LabeledPoint(165.0, [350.0]),
 LabeledPoint(150.0, [318.0]),
 LabeledPoint(150.0, [304.0]),
 LabeledPoint(140.0, [302.0])]
```

In [18]:
```
trainingData, testingData = data.randomSplit([.9,.1],seed=42)
print trainingData.count()
print testingData.count()
print trainingData.take(10)
```

```
90
9
[LabeledPoint(130.0, [307.0]), LabeledPoint(165.0, [350.0]), LabeledPoint(150.0, [318.0]), LabeledPoint(150.0, [304.0]),
LabeledPoint(140.0, [302.0]), LabeledPoint(198.0, [429.0]), LabeledPoint(220.0, [454.0]), LabeledPoint(215.0, [440.0]), L
abeledPoint(225.0, [455.0]), LabeledPoint(190.0, [390.0])]
```

4. Predict the horsepower and calculate the accuracy.

In [21]:
```
linear_model = LinearRegressionWithSGD.train(trainingData, iterations=100, step=0.00001)
true_vs_predicted = testingData.map(lambda p: (p.label, linear_model.predict(p.features)))
print "Linear Model predictions: " + str(true_vs_predicted.take(9))
```

```
Linear Model predictions: [(175.0, 180.53933874710663), (85.0, 100.29963263728146), (152.0, 176.02585527842896), (70.0, 4
5.134834686776657), (150.0, 159.4764158932775), (145.0, 175.52435711524254), (74.0, 52.6573071345772761), (67.0, 45.636332
849963061), (85.0, 131.3925187548387)]
```

In [22]:
```
def abs_error(actual, pred):
    return np.abs(pred - actual)

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

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

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 Absolute Error: %2.4f" % mae
print "Linear Model - Root Mean Squared Log Error: %2.4f" % rmsle
```

```
Linear Model - Mean Squared Error: 616.2225
Linear Model - Mean Absolute Error: 22.0922
Linear Model - Root Mean Squared Log Error: 0.2819
```

Source code:

LinearRegressionSimple.py

```
from pyspark import SparkContext
import pyspark.mllib
from pyspark.mllib.regression import LinearRegressionWithSGD
from pyspark.mllib.regression import LabeledPoint
from pyspark.sql.functions import *
import numpy as np

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

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

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

sc = SparkContext(appName = "LinearRegressionSimple")

path = "file:///home/cloudera/Documents/hw11/Week11_SparkML/car_noheader.csv"
raw_data = sc.textFile(path)

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

data = records.map(lambda line:LabeledPoint(line[4], [line[3]]))

trainingData, testingData = data.randomSplit([.9,.1],seed=42)

linear_model = LinearRegressionWithSGD.train(trainingData, iterations=100,
step=0.00001)
true_vs_predicted = testingData.map(lambda p: (p.label,
linear_model.predict(p.features)))
print "Linear Model predictions: " + str(true_vs_predicted.take(9))

# linear_model.save(sc,
"file:///home/cloudera/Documents/hw11/Week11_SparkML/linear_model")
true_vs_predicted.saveAsTextFile("file:///home/cloudera/Documents/hw11/Week11_S
parkML/output")

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 Absolute Error: %2.4f" % mae
print "Linear Model - Root Mean Squared Log Error: %2.4f" % rmsle
```

Note that the iterations and step in Linear Regression model may affect the accuracy.

```
[cloudera@localhost Desktop]$ spark-submit LinearRegressionSimple.py
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/St
aticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/Sta
ticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
16/04/21 21:20:11 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform...
using builtin-java classes where applicable
```

```
Linear Model predictions: [(175.0, 180.53933874710663), (85.0, 100.29963263728146), (152.0, 176.025
85527842896), (70.0, 45.134834686776657), (150.0, 159.4764158932775), (145.0, 175.52435711524254),
(74.0, 52.657307134572761), (67.0, 45.636332849963061), (85.0, 131.3925187548387)]
Linear Model - Mean Squared Error: 616.2225
Linear Model - Mean Absolute Error: 22.0922
Linear Model - Root Mean Squared Log Error: 0.2819
```

5. Present the model using D3.

Original data VS. Predictions:

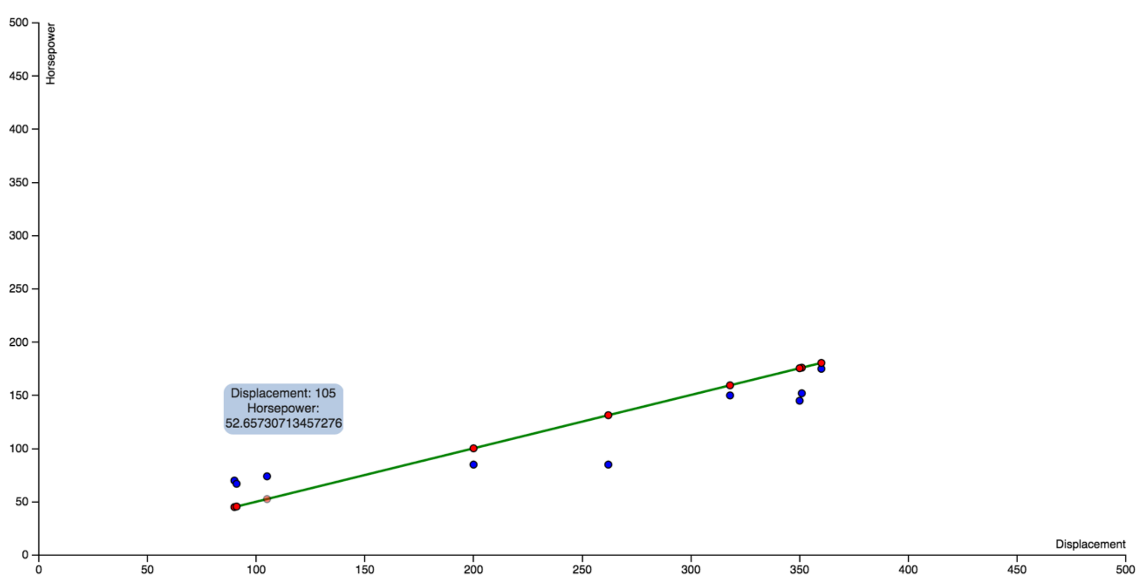
In [26]:
```
trainingData, testingData = data.randomSplit([.9,.1],seed=42)
print testingData.take(9)
```

```
[LabeledPoint(175.0, [360.0]), LabeledPoint(85.0, [200.0]), LabeledPoint(152.0, [351.0]), LabeledPoint(70.0, [90.0]), Lab
eledPoint(150.0, [318.0]), LabeledPoint(145.0, [350.0]), LabeledPoint(74.0, [105.0]), LabeledPoint(67.0, [91.0]), Labeled
Point(85.0, [262.0])]
```

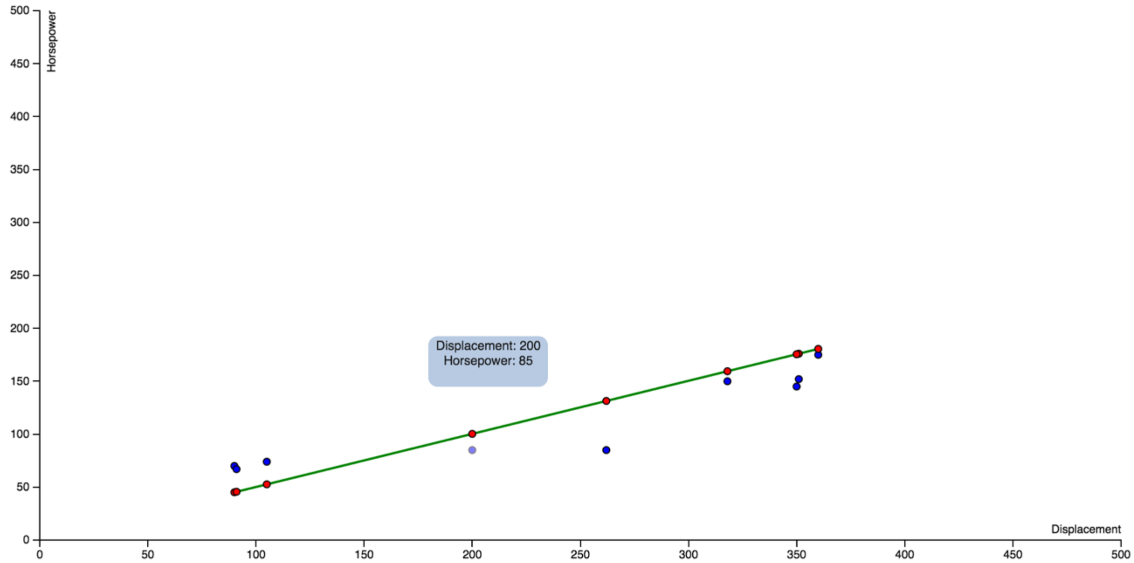
In [27]:
```
linear_model = LinearRegressionWithSGD.train(trainingData, iterations=100, step=0.00001)
true_vs_predicted = testingData.map(lambda p: (p.label, linear_model.predict(p.features)))
print "Linear Model predictions: " + str(true_vs_predicted.take(9))
```

```
Linear Model predictions: [(175.0, 180.53933874710663), (85.0, 100.29963263728146), (152.0, 176.02585527842896), (70.0, 4
5.134834686776657), (150.0, 159.4764158932775), (145.0, 175.52435711524254), (74.0, 52.657307134572761), (67.0, 45.636332
849963061), (85.0, 131.3925187548387)]
```

Predict the horsepower from displacement



Predict the horsepower from displacement



Start the HTTP server:

```
hqiu@bos-mp9cx>> python -m SimpleHTTPServer 8888
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<title>Predict the horsepower from displacement</title>
<head>
    <meta charset="utf-8">
    <script type="text/javascript" src="../d3/d3.js"></script>
    <style type="text/css">
        #chart {
            margin-left: -40px;
            height: 506px;
        }

        text {
            font: 10px sans-serif;
        }

        .dot {
            stroke: #000;
        }

        .axis path, .axis line {
            fill: none;
            stroke: #000;
            shape-rendering: crispEdges;
        }

        div.tooltip {
```

```
            position: absolute;
            text-align: center;
            width: 100px;
            height: 40px;
            padding: 2px;
            font: 11px sans-serif;
            background: lightsteelblue;
            border: 0px;
            border-radius: 8px;
            pointer-events: none;
        }

    </style>
</head>

<h1>Predict the horsepower from displacement</h1>

<p id="chart"></p>

<body>
<script type="text/javascript">
    // Various accessors that specify the five dimensions of data to visualize.
    function x(d) { return d.displacement; }
    function yOrig(d) { return d.horsepowerOrig; }
    function yPredict(d) { return d.horsepowerPredict; }

    // Chart dimensions.
    var margin = {top: 19.5, right: 19.5, bottom: 19.5, left: 70},
            width = 960 - margin.right,
            height = 500 - margin.top - margin.bottom;

    // Various scales. These domains make assumptions of data, naturally.
    var xScale = d3.scale.linear().domain([0, 500]).range([0, width]),
            yScale = d3.scale.linear().domain([0, 500]).range([height, 0]);

    // The x & y axes.
    var xAxis = d3.svg.axis().orient("bottom").scale(xScale).ticks(12,
d3.format(",d")),
            yAxis = d3.svg.axis().scale(yScale).ticks(12,
d3.format(",d")).orient("left");

    // Create the SVG container and set the origin.
    var svg = d3.select("#chart").append("svg")
            .attr("width", width + margin.left + margin.right)
            .attr("height", height + margin.top + margin.bottom)
            .append("g")
            .attr("transform", "translate(" + margin.left + "," + margin.top +
")");

    // Add the x-axis.
    svg.append("g")
            .attr("class", "x axis")
            .attr("transform", "translate(0," + height + ")")
            .call(xAxis);

    // Add the y-axis.
    svg.append("g")
            .attr("class", "y axis")
            .call(yAxis);

    // Add an x-axis label.
```

```
    svg.append("text")
            .attr("class", "x label")
            .attr("text-anchor", "end")
            .attr("x", width)
            .attr("y", height - 6)
            .text("Displacement");

    // Add a y-axis label.
    svg.append("text")
            .attr("class", "y label")
            .attr("text-anchor", "end")
            .attr("y", 6)
            .attr("dy", ".75em")
            .attr("transform", "rotate(-90)")
            .text("Horsepower");

    // Define the div for the tooltip
    var div = d3.select("body").append("div")
            .attr("class", "tooltip")
            .style("opacity", 0);

    // Load the data.
    d3.csv("Small_Car_Data.csv", function(error, data) {

        data.forEach(function(d) {
            d.displacement = +d.displacement;
            d.horsepowerOrig = +d.horsepowerOrig;
            d.horsepowerPredict = +d.horsepowerPredict;
            console.log(d);
        });

        var dot = svg.append("g")
                .attr("class", "dots")
                .selectAll("circle")
                .data(data)
                .enter()
                .append("g");

        // Plot the original values
        dot.append("circle")
                .attr("class", "dot")
                .style("fill", "#0000ff")
                .call(positionOrig)
                .on("mouseover", function(d) {
                    d3.select(this).style("opacity", 0.5);

                    div.transition()
                            .duration(200)
                            .style("left", d3.select(this).attr("cx") + "px")
                            .style("top", d3.select(this).attr("cy") + "px")
                            .style("opacity", .9);

                    div.html("Displacement: " + d.displacement + "<br/>" +
                             "Horsepower: " + d.horsepowerOrig)
                })
                .on("mouseout", function(d) {
                    d3.select(this).style("opacity", 1);

                    div.transition()
                            .duration(500)
                            .style("opacity", 0);
```

```
                });

        function positionOrig(dot) {
            dot.attr("cx", function (d) {
                        return xScale(x(d));
                    })
                    .attr("cy", function (d) {
                        return yScale(yOrig(d));
                    })
                    .attr("r", 3);
        }

        // Plot the predict values
        dot.append("circle")
                .attr("class", "dot")
                .style("fill", "#ff0000")
                .call(positionPredict)
                .on("mouseover", function(d) {
                    d3.select(this).style("opacity", 0.5);

                    div.transition()
                            .duration(200)
                            .style("left", d3.select(this).attr("cx") + "px")
                            .style("top", d3.select(this).attr("cy") + "px")
                            .style("opacity", .9);

                    div.html("Displacement: " + d.displacement + "<br/>" +
                            "Horsepower: " + d.horsepowerPredict)
                })
                .on("mouseout", function(d) {
                    d3.select(this).style("opacity", 1);

                    div.transition()
                            .duration(500)
                            .style("opacity", 0);
                });

        function positionPredict(dot) {
            dot.attr("cx", function (d) {
                        return xScale(x(d));
                    })
                    .attr("cy", function (d) {
                        return yScale(yPredict(d));
                    })
                    .attr("r", 3);
        }

    });

    // Draw the line
    svg.append("line")
            .attr("x1", xScale(90))
            .attr("y1", yScale(45.134834686776657))
            .attr("x2", xScale(360))
            .attr("y2", yScale(180.53933874710663))
            .attr("stroke-width", 2)
            .attr("stroke", "green");

</script>
</body>
```

To draw the line of the linear regression model, we just need to find the max and min value on line and concatenate two dots using a line.

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. 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.

Solution:

1. Now we have more features and some of them are categorical variables. We need to do the binary encoding for the categorical variables.

```
In [11]: records.cache()

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

         print "Mapping of first categorical feature column: %s" % get_mapping(records, 5)
         print "Mapping of first categorical feature column: %s" % get_mapping(records, 9)
```

```
Mapping of first categorical feature column: {u'fiat         ': 0, u'mercury      ': 1, u'honda        ': 2, u'bmw
   ': 3, u'plymouth     ': 4, u'volvo        ': 5, u'dodge        ': 6, u'volkswagen   ': 7, u'ford         ': 8, u'chevr
olet    ': 9, u'peugeot      ': 10, u'renault      ': 11, u'chrysler     ': 12, u'mazda        ': 13, u'oldsmobile   ': 1
4, u'pontiac      ': 15, u'nissan       ': 16, u'ih           ': 17, u'toyota       ': 18, u'mercedes-benz': 19, u'audi
       ': 20, u'cadillac     ': 21, u'buick        ': 22, u'saab         ': 23, u'datsun       ': 24, u'opel         ': 2
5, u'amc          ': 26, u'citroen      ': 27}
Mapping of first categorical feature column: {u'Sweden ': 0, u'Italy  ': 1, u'USA    ': 2, u'Germany': 3, u'Japan  ': 4,
u'France ': 5}
```

```
In [12]: mappings = [get_mapping(records, i) for i in [5, 9]]
         cat_len = len(get_mapping(records, 5)) + len(get_mapping(records, 9))
         num_len = 4
         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
```

```
Feature vector length for categorical features: 34
Feature vector length for numerical features: 4
Total feature vector length: 38
```

Extracted mappings to convert the categorical features to binary-encoded features. Concatenate with the numerical features and compose feature vectors.

```
In [13]: def extract_features(record):
             cat_array = [record[5], record[9]]
             cat_vec = np.zeros(cat_len)
             i = 0
             step = 0
             for field in cat_array:
                 m = mappings[i]
                 idx = m[field]
                 cat_vec[idx + step] = 1
                 i= i + 1
                 step = step + len(m)
             num_array = [record[2], record[3], record[7], record[10]]
             num_vec = np.array([float(field) for field in num_array])
             return np.concatenate((cat_vec, num_vec))

         def extract_label(record):
             return float(record[1])
```

```
In [14]: data = records.map(lambda r: LabeledPoint(extract_label(r),extract_features(r)))
         data.take(1)

Out[14]: [LabeledPoint(12.0, [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
         0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,8.0,307.0,70.0,3504.0])]
```

Use acceleration as the label.

In [15]:
```
first_point = data.first()
print "Raw data: " + str(first[1:])
print "Label: " + str(first_point.label)
print "Linear Model feature vector:\n" + str(first_point.features)
print "Linear Model feature vector length: " + str(len(first_point.features))
```

```
Raw data: [u'12', u'8', u'307', u'130', u'chevrolet    ', u'chevrolet chevelle malibu        ', u'70', u'18', u'USA    ',
 u'3504']
Label: 12.0
Linear Model feature vector:
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
1.0,0.0,0.0,0.0,0.0,8.0,307.0,70.0,3504.0]
Linear Model feature vector length: 38
```

In [16]:
```
trainingData, testingData = data.randomSplit([.9,.1],seed = 42)
print trainingData.count()
print testingData.count()
```

```
90
9
```

In [17]:
```
linear_model = LinearRegressionWithSGD.train(trainingData, iterations=100, step=0.000001)
true_vs_predicted = testingData.map(lambda p: (p.label, linear_model.predict(p.features)))
print "Linear Model predictions: " + str(true_vs_predicted.take(5))
```

```
Linear Model predictions: [(11.0, 16.65709973869798), (16.0, 11.40339222655504), (12.8, 18.410937773285795), (14.2, 8.822
7527152946585), (13.2, 17.259522141685977)]
```

In [18]:
```
def abs_error(actual, pred):
    return np.abs(pred - actual)

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

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

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 Absolute Error: %2.4f" % mae
print "Linear Model - Root Mean Squared Log Error: %2.4f" % rmsle
```

```
Linear Model - Mean Squared Error: 29.6892
Linear Model - Mean Absolute Error: 5.3580
Linear Model - Root Mean Squared Log Error: 0.3846
```

Use horsepower as the label.

```
def extract_label(record):
    return float(record[4])
```

In [20]:
```
data = records.map(lambda r: LabeledPoint(extract_label(r),extract_features(r)))
data.take(1)
```

Out[20]:
```
[LabeledPoint(130.0, [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,8.0,307.0,70.0,3504.0])]
```

In [21]:
```
first_point = data.first()
print "Raw data: " + str(first[1:])
print "Label: " + str(first_point.label)
print "Linear Model feature vector:\n" + str(first_point.features)
print "Linear Model feature vector length: " + str(len(first_point.features))
```

```
Raw data: [u'12', u'8', u'307', u'130', u'chevrolet    ', u'chevrolet chevelle malibu        ', u'70', u'18', u'USA    ',
 u'3504']
Label: 130.0
Linear Model feature vector:
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
1.0,0.0,0.0,0.0,0.0,8.0,307.0,70.0,3504.0]
Linear Model feature vector length: 38
```

In [22]:
```
trainingData, testingData = data.randomSplit([.9,.1],seed = 42)
print trainingData.count()
print testingData.count()
```

```
90
9
```

In [23]:
```
linear_model = LinearRegressionWithSGD.train(trainingData, iterations=100, step=0.000001)
true_vs_predicted = testingData.map(lambda p: (p.label, linear_model.predict(p.features)))
print "Linear Model predictions: " + str(true_vs_predicted.take(5))
```

```
Linear Model predictions: [(175.0, 148.05431351555734), (85.0, 98.664927651101024), (152.0, 161.30622033559189), (70.0, 7
2.72794305040442), (150.0, 150.58598193717597)]
```

```
In [24]: def abs_error(actual, pred):
             return np.abs(pred - actual)

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

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

         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 Absolute Error: %2.4f" % mae
         print "Linear Model - Root Mean Squared Log Error: %2.4f" % rmsle

         Linear Model - Mean Squared Error: 234.2946
         Linear Model - Mean Absolute Error: 11.3977
         Linear Model - Root Mean Squared Log Error: 0.1347
```

The horsepower is easier to predict since it has lower RMSLE value (percentage errors).

Source code:

LinearRegression.py

```
from pyspark import SparkContext
import pyspark.mllib
from pyspark.mllib.regression import LinearRegressionWithSGD
from pyspark.mllib.regression import LabeledPoint
from pyspark.sql.functions import *
import numpy as np

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

def extract_features(record):
    cat_array = [record[5], record[9]]
    cat_vec = np.zeros(cat_len)
    i = 0
    step = 0
    for field in cat_array:
        m = mappings[i]
        idx = m[field]
        cat_vec[idx + step] = 1
        i= i + 1
        step = step + len(m)
    num_array = [record[2], record[3], record[7], record[10]]
    num_vec = np.array([float(field) for field in num_array])
    return np.concatenate((cat_vec, num_vec))

def extract_label(record):
    # return float(record[1])
      return float(record[4])

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

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

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

sc = SparkContext(appName = "LinearRegression")

path = "file:///home/cloudera/Documents/hw11/Week11_SparkML/car_noheader.csv"
raw_data = sc.textFile(path)

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

records.cache()

mappings = [get_mapping(records, i) for i in [5, 9]]
cat_len = len(get_mapping(records, 5)) + len(get_mapping(records, 9))
num_len = 4
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

data = records.map(lambda r:
LabeledPoint(extract_label(r),extract_features(r)))

first_point = data.first()
print "Label: " + str(first_point.label)
print "Linear Model feature vector:\n" + str(first_point.features)
print "Linear Model feature vector length: " + str(len(first_point.features))

trainingData, testingData = data.randomSplit([.9,.1],seed = 42)

linear_model = LinearRegressionWithSGD.train(trainingData, iterations=100,
step=0.000001)
true_vs_predicted = testingData.map(lambda p: (p.label,
linear_model.predict(p.features)))
print "Linear Model predictions: " + str(true_vs_predicted.take(5))

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 Absolute Error: %2.4f" % mae
print "Linear Model - Root Mean Squared Log Error: %2.4f" % rmsle
```

Results:

```
[cloudera@localhost Desktop]$ spark-submit LinearRegression.py
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/St
aticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/Sta
ticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
```

Use acceleration as label.

```
Feature vector length for categorical features: 34
Feature vector length for numerical features: 4
Total feature vector length: 38
Label: 12.0
Linear Model feature vector:
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,8.0,307.0,70.0,3504.0]
Linear Model feature vector length: 38
```

```
Linear Model predictions: [(11.0, 16.65709973869798), (16.0, 11.40339222655504), (12.8, 18.41093777
3285795), (14.2, 8.8227527152946585), (13.2, 17.259522141685977)]
Linear Model - Mean Squared Error: 29.6892
Linear Model - Mean Absolute Error: 5.3580
Linear Model - Root Mean Squared Log Error: 0.3846
```

Use horsepower as label.

```
Feature vector length for categorical features: 34
Feature vector length for numerical features: 4
Total feature vector length: 38
Label: 130.0
Linear Model feature vector:
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,8.0,307.0,70.0,3504.0]
Linear Model feature vector length: 38
```

```
Linear Model predictions: [(175.0, 148.05431351555734), (85.0, 98.664927651101024), (152.0, 161.306
22033559189), (70.0, 72.72794305040442), (150.0, 150.58598193717597)]
Linear Model - Mean Squared Error: 234.2946
Linear Model - Mean Absolute Error: 11.3977
Linear Model - Root Mean Squared Log Error: 0.1347
```

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

1. Use the Decision Tree model.

localhost:8888/notebooks/DecisionTreeSimple.ipynb

jupyter DecisionTreeSimple Last Checkpoint: 17 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Help

CellToolbar

In [1]:
```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import DecisionTree
from pyspark.mllib.linalg import SparseVector
import numpy as np
```

In [2]:
```
path = "file:///home/cloudera/Documents/hw11/Week11_SparkML/car_noheader.csv"
```

In [3]:
```
raw_data = sc.textFile(path)
```

In [4]:
```
num_data = raw_data.count()
```

In [5]:
```
raw_data.take(5)
```

Out[5]:
```
[u'1,12,8,307,130,chevrolet    ,chevrolet chevelle malibu        ,70,18,USA    ,3504',
 u'2,11.5,8,350,165,buick        ,buick skylark 320              ,70,15,USA    ,3693',
 u'3,11,8,318,150,plymouth      ,plymouth satellite             ,70,18,USA    ,3436',
 u'4,12,8,304,150,amc           ,amc rebel sst                  ,70,16,USA    ,3433',
 u'5,10.5,8,302,140,ford         ,ford torino                    ,70,17,USA    ,3449']
```

In [6]:
```
records = raw_data.map(lambda x: x.split(","))
```

In [11]:
```
data = records.map(lambda line:LabeledPoint(line[4], [line[3]]))
data.take(5)
```

Out[11]:
```
[LabeledPoint(130.0, [307.0]),
 LabeledPoint(165.0, [350.0]),
 LabeledPoint(150.0, [318.0]),
 LabeledPoint(150.0, [304.0]),
 LabeledPoint(140.0, [302.0])]
```

In [12]:
```
trainingData, testingData = data.randomSplit([.9,.1],seed=42)
print trainingData.count()
print testingData.count()
```

```
90
9
```

In [13]:
```
dt_model = DecisionTree.trainRegressor(trainingData,{})
preds = dt_model.predict(testingData.map(lambda p: p.features))
actual = testingData.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())
```

```
Decision Tree predictions: [(175.0, 184.0), (85.0, 93.6), (152.0, 184.0), (70.0, 64.6), (150.0, 164.0)]
Decision Tree depth: 5
Decision Tree number of nodes: 41
```

In [14]:
```
def abs_error(actual, pred):
    return np.abs(pred - actual)

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

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

mse_dt = true_vs_predicted_dt.map(lambda (t, p): squared_error(t, p)).mean()
mae_dt = true_vs_predicted_dt.map(lambda (t, p): abs_error(t, p)).mean()
rmsle_dt = np.sqrt(true_vs_predicted_dt.map(lambda (t, p): squared_log_error(t, p)).mean())
print "Decision Tree - Mean Squared Error: %2.4f" % mse_dt
print "Decision Tree - Mean Absolute Error: %2.4f" % mae_dt
print "Decision Tree - Root Mean Squared Log Error: %2.4f" % rmsle_dt
```

```
Decision Tree - Mean Squared Error: 524.9527
Decision Tree - Mean Absolute Error: 16.9519
Decision Tree - Root Mean Squared Log Error: 0.1864
```

DecisionTreeSimple.py

```
from pyspark import SparkContext
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import DecisionTree
from pyspark.mllib.linalg import SparseVector
import numpy as np

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

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

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

sc = SparkContext(appName = "DecisionTreeSimple")

path = "file:///home/cloudera/Documents/hw11/Week11_SparkML/car_noheader.csv"
raw_data = sc.textFile(path)

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

data = records.map(lambda line:LabeledPoint(line[4], [line[3]]))

trainingData, testingData = data.randomSplit([.9,.1],seed=42)

dt_model = DecisionTree.trainRegressor(trainingData,{})
preds = dt_model.predict(testingData.map(lambda p: p.features))
actual = testingData.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())

mse_dt = true_vs_predicted_dt.map(lambda (t, p): squared_error(t, p)).mean()
mae_dt = true_vs_predicted_dt.map(lambda (t, p): abs_error(t, p)).mean()
rmsle_dt = np.sqrt(true_vs_predicted_dt.map(lambda (t, p): squared_log_error(t,
p)).mean())
print "Decision Tree - Mean Squared Error: %2.4f" % mse_dt
print "Decision Tree - Mean Absolute Error: %2.4f" % mae_dt
print "Decision Tree - Root Mean Squared Log Error: %2.4f" % rmsle_dt
```

```
Decision Tree predictions: [(175.0, 184.0), (85.0, 93.6), (152.0, 184.0), (70.0, 64.6), (150.0, 164
.0)]
Decision Tree depth: 5
Decision Tree number of nodes: 41
Decision Tree - Mean Squared Error: 524.9527
Decision Tree - Mean Absolute Error: 16.9519
Decision Tree - Root Mean Squared Log Error: 0.1864
```

```
Linear Model predictions: [(175.0, 180.53933874710663), (85.0, 100.29963263728146), (152.0, 176.025
85527842896), (70.0, 45.134834686776657), (150.0, 159.4764158932775), (145.0, 175.52435711524254),
(74.0, 52.657307134572761), (67.0, 45.636332849963061), (85.0, 131.3925187548387)]
Linear Model - Mean Squared Error: 616.2225
Linear Model - Mean Absolute Error: 22.0922
Linear Model - Root Mean Squared Log Error: 0.2819
```

So the Decision Tree has better performance.

2. Use more features.

In [11]:
```
records.cache()

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

print "Mapping of first categorical feature column: %s" % get_mapping(records, 5)
print "Mapping of first categorical feature column: %s" % get_mapping(records, 9)
```

```
Mapping of first categorical feature column: {u'fiat         ': 0, u'mercury      ': 1, u'honda        ': 2, u'bmw
   ': 3, u'plymouth     ': 4, u'volvo        ': 5, u'dodge        ': 6, u'volkswagen   ': 7, u'ford         ': 8, u'chevr
olet    ': 9, u'peugeot      ': 10, u'renault      ': 11, u'chrysler     ': 12, u'mazda        ': 13, u'oldsmobile   ': 1
4, u'pontiac      ': 15, u'nissan       ': 16, u'ih           ': 17, u'toyota       ': 18, u'mercedes-benz': 19, u'audi
       ': 20, u'cadillac     ': 21, u'buick        ': 22, u'saab         ': 23, u'datsun       ': 24, u'opel         ': 2
5, u'amc          ': 26, u'citroen      ': 27}
Mapping of first categorical feature column: {u'Sweden ': 0, u'Italy  ': 1, u'USA    ': 2, u'Germany': 3, u'Japan  ': 4,
u'France ': 5}
```

When composing the feature vectors for Decision Tree, we still need to convert the categorical features into numeric values. But we don't need to convert categorical features into a binary vector encoding. That's why we only have 6 features, not 38 compared to Linear Regression model.

In [12]:
```
mappings = [get_mapping(records, i) for i in [5, 9]]
cat_len = len(mappings)
num_len = 4
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
```

```
Feature vector length for categorical features: 2
Feature vector length for numerical features: 4
Total feature vector length: 6
```

Use acceleration as label.

In [13]:
```
def extract_features_dt(record):
    cat_array = [record[5], record[9]]
    cat_vec = np.zeros(cat_len)
    i = 0
    step = 0
    for field in cat_array:
        m = mappings[i]
        idx = m[field]
        cat_vec[i] = idx
        i = i + 1
    num_array = [record[2], record[3], record[7], record[10]]
    num_vec = np.array([float(field) for field in num_array])
    return np.concatenate((cat_vec, num_vec))

def extract_label(record):
    return float(record[1])
```

In [14]:
```
data_dt = records.map(lambda r: LabeledPoint(extract_label(r), extract_features_dt(r)))
```

In [15]:
```
first_point_dt = data_dt.first()
print "Raw data: " + str(first[1:])
print "Label: " + str(first_point_dt.label)
print "Decision Tree feature vector: " + str(first_point_dt.features)
print "Decision Tree feature vector length: " + str(len(first_point_dt.features))
```

```
Raw data: [u'12', u'8', u'307', u'130', u'chevrolet    ', u'chevrolet chevelle malibu        ', u'70', u'18', u'USA    ',
 u'3504']
Label: 12.0
Decision Tree feature vector: [9.0,2.0,8.0,307.0,70.0,3504.0]
Decision Tree feature vector length: 6
```

In [17]:
```
trainingData, testingData = data_dt.randomSplit([.9,.1],seed = 42)
print trainingData.count()
print testingData.count()
```

```
90
9
```

In [18]:
```
dt_model = DecisionTree.trainRegressor(trainingData,{})
preds = dt_model.predict(testingData.map(lambda p: p.features))
actual = testingData.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())
```

```
Decision Tree predictions: [(11.0, 11.333333333333334), (16.0, 15.575000000000001), (12.8, 11.333333333333334), (14.2, 20
.25), (13.2, 13.733333333333334)]
Decision Tree depth: 5
Decision Tree number of nodes: 55
```

In [19]:
```
def abs_error(actual, pred):
    return np.abs(pred - actual)

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

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

mse_dt = true_vs_predicted_dt.map(lambda (t, p): squared_error(t, p)).mean()
mae_dt = true_vs_predicted_dt.map(lambda (t, p): abs_error(t, p)).mean()
rmsle_dt = np.sqrt(true_vs_predicted_dt.map(lambda (t, p): squared_log_error(t, p)).mean())
print "Decision Tree - Mean Squared Error: %2.4f" % mse_dt
print "Decision Tree - Mean Absolute Error: %2.4f" % mae_dt
print "Decision Tree - Root Mean Squared Log Error: %2.4f" % rmsle_dt
```

```
Decision Tree - Mean Squared Error: 7.2174
Decision Tree - Mean Absolute Error: 1.7139
Decision Tree - Root Mean Squared Log Error: 0.1502
```

Use horsepower as label.

```
def extract_label(record):
    return float(record[4])
```

In [21]:
```
data_dt = records.map(lambda r: LabeledPoint(extract_label(r), extract_features_dt(r)))
```

In [22]:
```
first_point_dt = data_dt.first()
print "Raw data: " + str(first[1:])
print "Label: " + str(first_point_dt.label)
print "Decision Tree feature vector: " + str(first_point_dt.features)
print "Decision Tree feature vector length: " + str(len(first_point_dt.features))
```

```
Raw data: [u'12', u'8', u'307', u'130', u'chevrolet    ', u'chevrolet chevelle malibu        ', u'70', u'18', u'USA    ',
 u'3504']
Label: 130.0
Decision Tree feature vector: [9.0,2.0,8.0,307.0,70.0,3504.0]
Decision Tree feature vector length: 6
```

In [23]:
```
trainingData, testingData = data_dt.randomSplit([.9,.1],seed = 42)
print trainingData.count()
print testingData.count()
```

```
90
9
```

In [24]:
```
dt_model = DecisionTree.trainRegressor(trainingData,{})
preds = dt_model.predict(testingData.map(lambda p: p.features))
actual = testingData.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())
```

```
Decision Tree predictions: [(175.0, 165.0), (85.0, 92.71428571428571), (152.0, 152.6), (70.0, 62.875), (150.0, 152.6)]
Decision Tree depth: 5
Decision Tree number of nodes: 55
```

In [25]:
```
def abs_error(actual, pred):
    return np.abs(pred - actual)

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

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

mse_dt = true_vs_predicted_dt.map(lambda (t, p): squared_error(t, p)).mean()
mae_dt = true_vs_predicted_dt.map(lambda (t, p): abs_error(t, p)).mean()
rmsle_dt = np.sqrt(true_vs_predicted_dt.map(lambda (t, p): squared_log_error(t, p)).mean())
print "Decision Tree - Mean Squared Error: %2.4f" % mse_dt
print "Decision Tree - Mean Absolute Error: %2.4f" % mae_dt
print "Decision Tree - Root Mean Squared Log Error: %2.4f" % rmsle_dt
```

```
Decision Tree - Mean Squared Error: 103.1044
Decision Tree - Mean Absolute Error: 8.3886
Decision Tree - Root Mean Squared Log Error: 0.0984
```

DecisionTree.py

```
from pyspark import SparkContext
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import DecisionTree
from pyspark.mllib.linalg import SparseVector
from pyspark.sql.functions import *
import numpy as np

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

def extract_features_dt(record):
    cat_array = [record[5], record[9]]
```

```
    cat_vec = np.zeros(cat_len)
    i = 0
    step = 0
    for field in cat_array:
        m = mappings[i]
        idx = m[field]
        cat_vec[i] = idx
        i = i + 1
    num_array = [record[2], record[3], record[7], record[10]]
    num_vec = np.array([float(field) for field in num_array])
    return np.concatenate((cat_vec, num_vec))

def extract_label(record):
    # return float(record[1])
       return float(record[4])

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

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

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

sc = SparkContext(appName = "DecisionTree")

path = "file:///home/cloudera/Documents/hw11/Week11_SparkML/car_noheader.csv"
raw_data = sc.textFile(path)

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

records.cache()

mappings = [get_mapping(records, i) for i in [5, 9]]
cat_len = len(mappings)
num_len = 4
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

data_dt = records.map(lambda r: LabeledPoint(extract_label(r),
extract_features_dt(r)))

first_point_dt = data_dt.first()
print "Label: " + str(first_point_dt.label)
print "Decision Tree feature vector: " + str(first_point_dt.features)
print "Decision Tree feature vector length: " +
str(len(first_point_dt.features))

trainingData, testingData = data_dt.randomSplit([.9,.1],seed = 42)
```

```
dt_model = DecisionTree.trainRegressor(trainingData,{})
preds = dt_model.predict(testingData.map(lambda p: p.features))
actual = testingData.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())

mse_dt = true_vs_predicted_dt.map(lambda (t, p): squared_error(t, p)).mean()
mae_dt = true_vs_predicted_dt.map(lambda (t, p): abs_error(t, p)).mean()
rmsle_dt = np.sqrt(true_vs_predicted_dt.map(lambda (t, p): squared_log_error(t,
p)).mean())
print "Decision Tree - Mean Squared Error: %2.4f" % mse_dt
print "Decision Tree - Mean Absolute Error: %2.4f" % mae_dt
print "Decision Tree - Root Mean Squared Log Error: %2.4f" % rmsle_dt
```

```
Feature vector length for categorical features: 2
Feature vector length for numerical features: 4
Total feature vector length: 6
Label: 12.0
Decision Tree feature vector: [9.0,2.0,8.0,307.0,70.0,3504.0]
Decision Tree feature vector length: 6
Decision Tree predictions: [(11.0, 11.333333333333334), (16.0, 15.575000000000001), (12.8, 11.333333
333333334), (14.2, 20.25), (13.2, 13.733333333333334)]
Decision Tree depth: 5
Decision Tree number of nodes: 55
Decision Tree - Mean Squared Error: 7.2174
Decision Tree - Mean Absolute Error: 1.7139
Decision Tree - Root Mean Squared Log Error: 0.1502
```

```
Feature vector length for categorical features: 2
Feature vector length for numerical features: 4
Total feature vector length: 6
Label: 130.0
Decision Tree feature vector: [9.0,2.0,8.0,307.0,70.0,3504.0]
Decision Tree feature vector length: 6
Decision Tree predictions: [(175.0, 165.0), (85.0, 92.71428571428571), (152.0, 152.6), (70.0, 62.875
), (150.0, 152.6)]
Decision Tree depth: 5
Decision Tree number of nodes: 55
Decision Tree - Mean Squared Error: 103.1044
Decision Tree - Mean Absolute Error: 8.3886
Decision Tree - Root Mean Squared Log Error: 0.0984
```

Comparing the accuracy, horsepower is easier to predict and Decision Tree has better performance.