HU Extension Assignment 07 E63 Big Data Analytics

Handed out: 10/13/2017 Due by 4:00 PM EST on Saturday, 10/21/2017

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%)

- **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.**
- **Find all null values in all numerical columns. Replace nulls, if any, with average values for respective columns using Spark Data Frame API.**
- **Import data into Spark.**

```
path = "file:////Users/swaite/Stirling/CSIE-63/assignment-7/data/auto_mpg_original-1.csv"
mapped_auto_rdd = sc.textFile(path, use_unicode=False) \
                    .map(lambda x: x.split(","))\
                    .filter(lambda x: (x[0] > 0) & (x[3] > 0))\
                    .map(lambda x: Row(
                        mpg=x[0],
                        cylinders=x[1],
                        displacement=x[2],
                        horsepower=x[3],
                        weight=x[4],
                        acceleration=x[5],
                        model_year=x[6],
                        origin=x[7],
                        car_name=x[8]
                    ))\
                    .toDF()

print(mapped_auto_rdd.show(5))
```

```
+------------+---------+---------+------------+----------+----------+---+------+------+
|acceleration| car_name|cylinders|displacement|horsepower|model_year|mpg|origin|weight|
+------------+---------+---------+------------+----------+----------+---+------+------+
|          12|chevrolet|        8|         307|       130|        70| 18|     1|  3504|
|        11.5|    buick|        8|         350|       165|        70| 15|     1|  3693|
|          11| plymouth|        8|         318|       150|        70| 18|     1|  3436|
|          12|      amc|        8|         304|       150|        70| 16|     1|  3433|
|        10.5|     ford|        8|         302|       140|        70| 17|     1|  3449|
+------------+---------+---------+------------+----------+----------+---+------+------+
only showing top 5 rows

None
```

- **Randomly select 10-20% of you data for testing and use remaining data for training.**

```
training_data, testing_data = mapped_auto_rdd.randomSplit([.8, .2], seed=1234)
print("training_data count: " + str(training_data.count()))
print("testing_data count: " + str(testing_data.count()))
print("mapped_auto_rdd count: " + str(mapped_auto_rdd.count()))
```

```
training_data count: 334
testing_data count: 72
mapped_auto_rdd count: 406
```

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%)

```
### Attached file auto_mpg_original.csv contains a set of data on automobile characteristics
### File auto_mpg_description.csv contains the description of the data.
### Import data into Spark.

path = "file:////Users/swaite/Stirling/CSIE-63/assignment-7/data/auto_mpg_original-1.csv"
df = sc.textFile(path, use_unicode=False) \
                    .map(lambda x: x.split(","))\
                    .map(lambda x: Row(
                        mpg=float(x[0].replace('NA', "0.0")),
                        cylinders=float(x[1].replace('NA', "0.0")),
                        displacement=float(x[2].replace('NA', "0.0")),
                        horsepower=float(x[3].replace('NA', "0.0")),
                        weight=float(x[4].replace('NA', "0.0")),
                        acceleration=float(x[5].replace('NA', "0.0")),
                        model_year=float(x[6].replace('NA', "0.0")),
                        origin=float(x[7].replace('NA', "0.0")),
                        car_name=str(x[8])
                    )) \
                    .filter(lambda x: x[0] > 0.0)\
                    .filter(lambda x: x[3] > 0.0)\
                    .toDF()

print(df.show(5))
```

```
+------------+---------+---------+------------+----------+----------+----+------+------+
|acceleration| car_name|cylinders|displacement|horsepower|model_year| mpg|origin|weight|
+------------+---------+---------+------------+----------+----------+----+------+------+
|        12.0|chevrolet|      8.0|       307.0|     130.0|      70.0|18.0|   1.0|3504.0|
|        11.5|    buick|      8.0|       350.0|     165.0|      70.0|15.0|   1.0|3693.0|
|        11.0| plymouth|      8.0|       318.0|     150.0|      70.0|18.0|   1.0|3436.0|
|        12.0|      amc|      8.0|       304.0|     150.0|      70.0|16.0|   1.0|3433.0|
|        10.5|     ford|      8.0|       302.0|     140.0|      70.0|17.0|   1.0|3449.0|
+------------+---------+---------+------------+----------+----------+----+------+------+
only showing top 5 rows
```

```
#Summary of Input Variable
df2 = df.select(df.horsepower.cast("float"), df.mpg.cast("float"))
print(df2.show(10))
df2.describe().show()
marvar = df2.collect()
```

```
+----------+----+
|horsepower| mpg|
+----------+----+
|     130.0|18.0|
|     165.0|15.0|
|     150.0|18.0|
|     150.0|16.0|
|     140.0|17.0|
|     198.0|15.0|
|     220.0|14.0|
|     215.0|14.0|
|     225.0|14.0|
|     190.0|15.0|
+----------+----+
only showing top 10 rows

None
+-------+------------------+------------------+
|summary|        horsepower|               mpg|
+-------+------------------+------------------+
|  count|               406|               406|
|   mean|104.57881773399015|23.435467978416405|
| stddev|38.956860529340645| 7.855637115477021|
|    min|               0.0|               0.0|
|    max|             230.0|              46.6|
+-------+------------------+------------------+
```

```
# Training the Model
linear_model = LinearRegressionWithSGD.train(training_data, iterations=1000,
step=.0001)

print "\n"
print "--------------"
print "Linear model parameters"
print "--------------"

print(linear_model)

print "\n"
print "--------------"
print "Predictions "
print "--------------"

true_vs_predicted = testing_data.map(lambda p: (p.label,
```

```
linear_model.predict(p.features)))
print("Linear Model preditions: " + str(true_vs_predicted.take(5)))

print "\n"
print "--------------"
print "Model Metrics "
print "--------------"


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

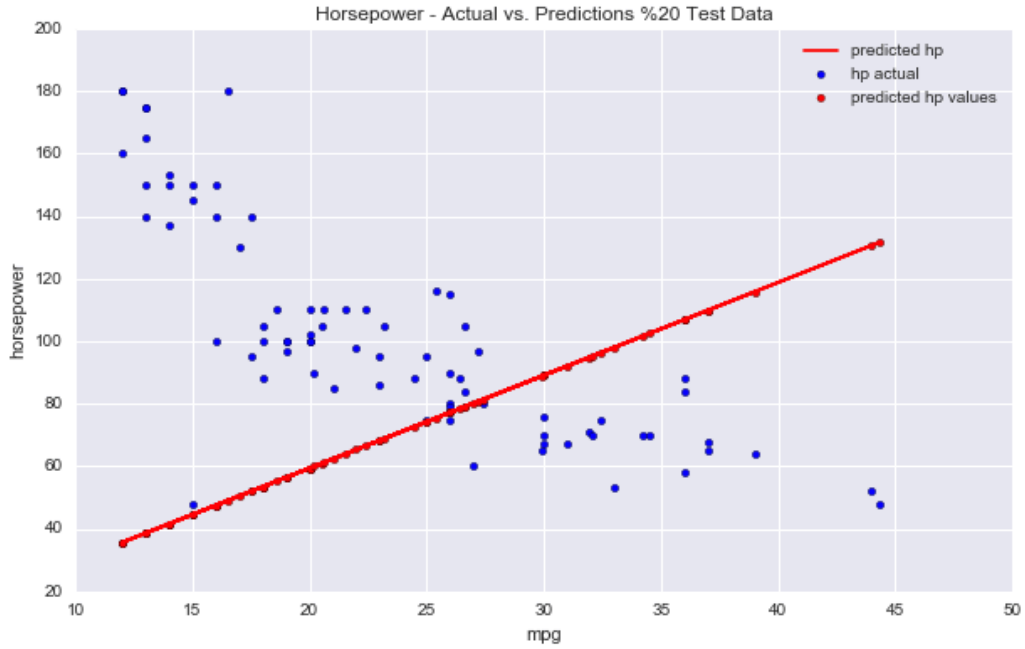
OUTPUT

```
--------------
Linear model parameters
--------------
(weights=[2.96970534459], intercept=0.0)

--------------
Predictions
--------------
Linear Model predictions: [(115.0, 77.212338959343398), (85.0, 62.36
3812236392739), (95.0, 74.24263361475326), (48.0, 44.545580168851956
), (100.0, 53.454696202622351)]

--------------
Model Metrics
--------------
Linear Model - Mean Squared Error: 4130.7654
Linear Model - Mean Absolute Error: 50.8381
Linear Model - Root Mean Squared Log Error: 0.7434
```

Horsepower - Actual vs. Predictions 20% Test Data

```
tvp = testing_data.map(lambda p: (float(p.label), float(linear_model.predict(p.features)),
float(p.features[0]))).toDF().toPandas()
tvp.columns = ['horsepower', 'predicted_horsepower', 'mpg']

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()
```



Horsepower - Actual vs. Predictions 20% Test Data

```
plt.figure(2, figsize = (12, 10))
tvp2 = training_data.map(lambda p: (float(p.label),
float(linear_model.predict(p.features)), float(p.features[0]))).toDF().toPandas()
tvp2.columns = ['horsepower', 'predicted_horsepower', 'mpg']
plt.scatter(tvp["mpg"], tvp["horsepower"], c='b', label="hp .2 sample data")
plt.scatter(tvp2["mpg"], tvp2["horsepower"], c='g', label="hp .8 training data")
plt.plot(tvp["mpg"], tvp["predicted_horsepower"], c='r', label="predicted hp")
plt.scatter(tvp["mpg"], tvp["predicted_horsepower"], c='m', 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%)

```
# ## Extract the variables we want to keep
#
# _All variables:_
#
# `instant,dteday,season,yr,mnth,hr,holiday,weekday,workingday,weathersit,temp,atemp,hum,windspeed,casual,registered,cn
#
# _Variables to keep:_
#
# `season,yr,mnth,hr,holiday,weekday,workingday,weathersit,temp,atemp,hum,windspeed,cnt`

path = "file:////Users/swaite/Stirling/CSIE-63/assignment-7/data/bike-sharing-hour.csv"
bike_rdd = sc.textFile(path, use_unicode=False) \
                    .map(lambda x: x.split(","))
header = bike_rdd.first()
print "header:"
print(header)
bike_rdd_train = bike_rdd.filter(lambda x: x != header)\
                         .cache()
print("\nTraining Data column length: " + str(len(bike_rdd_train.first())))
print(bike_rdd_train.take(5))
```

```
header:
['instant', 'dteday', 'season', 'yr', 'mnth', 'hr', 'holiday', 'weekday', 'workingday', 'weathersit', 'temp', 'atem
p', 'hum', 'windspeed', 'casual', 'registered', 'cnt']

Training Data column length: 17
[['1', '2011-01-01', '1', '0', '1', '0', '0', '6', '0', '1', '0.24', '0.2879', '0.81', '0', '3', '13', '16'], ['2',
 '2011-01-01', '1', '0', '1', '1', '0', '6', '0', '1', '0.22', '0.2727', '0.8', '0', '8', '32', '40'], ['3', '2011-01
-01', '1', '0', '1', '2', '0', '6', '0', '1', '0.22', '0.2727', '0.8', '0', '5', '27', '32'], ['4', '2011-01-01',
 '1', '0', '1', '3', '0', '6', '0', '1', '0.24', '0.2879', '0.75', '0', '3', '10', '13'], ['5', '2011-01-01', '1',
 '0', '1', '4', '0', '6', '0', '1', '0.24', '0.2879', '0.75', '0', '0', '1', '1']]
```

Create feature mappings for categorical features

```
# function to get the categorical feature mapping for a given variable column
def get_mapping(rdd, idx):
    return rdd.map(lambda fields: fields[idx]).distinct().zipWithIndex().collectAsMap()
```

```
# we want to extract the feature mappings for columns 2 - 9
# try it out on column 2 first
print "Mapping of second categorical feature column: %s" % get_mapping(bike_rdd_train, 2)
```

```
Mapping of second categorical feature column: {'11': 0, '10': 6, '12': 7, '1': 1, '3': 2, '2': 8, '5': 3, '4': 9,
 '7': 4, '6': 10, '9': 5, '8': 11}
```

```
# extract all the categorical mappings
mappings = [get_mapping(bike_rdd_train, i) for i in range(2,10)]
print(mappings)
cat_len = sum(map(len, mappings))
num_len = len(bike_rdd_train.first()[11:15])
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
```

```
[{'1': 0, '3': 1, '2': 2, '4': 3}, {'1': 0, '0': 1}, {'11': 0, '10': 6, '12': 7, '1': 1, '3': 2, '2': 8, '5': 3, '4':
9, '7': 4, '6': 10, '9': 5, '8': 11}, {'20': 2, '21': 14, '22': 4, '23': 15, '1': 6, '0': 18, '3': 7, '2': 19, '5':
 8, '4': 20, '7': 9, '6': 21, '9': 10, '8': 22, '11': 0, '10': 12, '13': 1, '12': 13, '15': 11, '14': 23, '17': 3, '1
6': 17, '19': 5, '18': 16}, {'1': 0, '0': 1}, {'1': 0, '0': 3, '3': 1, '2': 4, '5': 2, '4': 5, '6': 6}, {'1': 0, '0':
1}, {'1': 0, '3': 1, '2': 2, '4': 3}]
Feature vector length for categorical features: 57
Feature vector length for numerical features: 4
Total feature vector length: 61
```

```
data = testing_data.map(lambda r: LabeledPoint(extract_label(r), extract_features(r)))
first_point = data.first()
print "Raw data: " + str(first_point)
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: (36.0,[1.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,0.0,
0.0,0.0,0.0,0.0,0.38,0.3939,0.76,0.2537])
Label: 36.0
Linear Model feature vector:
[1.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.38,
0.3939,0.76,0.2537]
Linear Model feature vector length: 61
```

```
# we need a separate set of feature vectors for the decision tree
def extract_features_dt(record):
    return np.array(map(float, record[2:14]))

data_dt = testing_data.map(lambda r: LabeledPoint(extract_label(r), extract_features_dt(r)))
first_point_dt = data_dt.first()
print "Decision Tree feature vector: " + str(first_point_dt.features)
print "Decision Tree feature vector length: " + str(len(first_point_dt.features))
```

```
Decision Tree feature vector: [1.0,0.0,1.0,10.0,0.0,6.0,0.0,1.0,0.38,0.3939,0.76,0.2537]
Decision Tree feature vector length: 12
```

Train a Regression Model on the Bike Sharing Dataset

```
linear_model = LinearRegressionWithSGD.train(data, iterations=10, step=0.1, intercept=False)
true_vs_predicted = data.map(lambda p: (p.label, linear_model.predict(p.features)))
print "Linear Model predictions: " + str(true_vs_predicted.take(5))
```

```
Linear Model predictions: [(36.0, 129.4893833950199), (39.0, 135.07384467007708), (59.0, 126.82225862471493),
 125.96404091718107), (5.0, 125.56387918963046)]
```

```
# we pass in an empty mapping for categorical feature size {}
dt_model = DecisionTree.trainRegressor(data_dt, {})
preds = dt_model.predict(data_dt.map(lambda p: p.features))
actual = data.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: [(36.0, 113.62790697674419), (39.0, 52.753246753246756), (59.0, 113.62790697674419)
0, 113.62790697674419), (5.0, 26.5625)]
Decision Tree depth: 5
Decision Tree number of nodes: 63
```

Perfomance Metrics

```
# set up performance metrics functions

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
```

```
# Calculate at least two performance metrics of your model.
# compute performance metrics for linear model
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: 30491.9925
Linear Model - Mean Absolute Error: 130.1671
Linear Model - Root Mean Squared Log Error: 1.4651
```

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%)

```
path = "file:////Users/swaite/Stirling/CSIE-63/assignment-7/data/auto_mpg_original-1.csv"
df = sc.textFile(path, use_unicode=False) \
                    .map(lambda x: x.split(","))\
                    .filter(lambda x: x[0] > 0.0)\
                    .filter(lambda x: x[3] > 0.0) \

print(df.take(5))
```

```
[['18', '8', '307', '130', '3504', '12', '70', '1', 'chevrolet'], ['15', '8', '350', '165', '3693', '11.5', '70',
  '1', 'buick'], ['18', '8', '318', '150', '3436', '11', '70', '1', 'plymouth'], ['16', '8', '304', '150', '3433', '1
2', '70', '1', 'amc'], ['17', '8', '302', '140', '3449', '10.5', '70', '1', 'ford']]
```

```
## Move RDD Order to:

#     1. mpg:           continuous
#     2. displacement:  continuous
#     3. horsepower:    continuous
#     4. weight:        continuous
#     5. acceleration:  continuous
#     6. model year:    multi-valued discrete
#     7. origin:        multi-valued discrete
#     8. cylinders:     multi-valued discrete
#     9. car name:      string (unique for each instance)

df = df.map(lambda x: [
                        x[0].replace('NA', "0.0"),
                        x[2].replace('NA', "0.0"),
                        x[3].replace('NA', "0.0"),
                        x[4].replace('NA', "0.0"),
                        x[5].replace('NA', "0.0"),
                        x[6].replace('NA', "0.0"),
                        x[7].replace('NA', "0.0"),
                        x[1].replace('NA', "0.0"),
                        x[8].replace('NA', "0.0")
                      ])
print(df.take(5))
```

```
[['18', '130', '3504', '12', '70', '1', '8', '307', 'chevrolet'], ['15', '165', '3693', '11.5', '70', '1', '8', '35
0', 'buick'], ['18', '150', '3436', '11', '70', '1', '8', '318', 'plymouth'], ['16', '150', '3433', '12', '70', '1',
 '8', '304', 'amc'], ['17', '140', '3449', '10.5', '70', '1', '8', '302', 'ford']]
```

Create feature mappings for categorical features

```
# function to get the categorical feature mapping for a given variable column
def get_mapping(rdd, idx):
    return rdd.map(lambda fields: fields[idx]).distinct().zipWithIndex().collectAsMap()
```

```
# extract all the categorical mappings
mappings = [get_mapping(df, i) for i in range(1,4)]
cat_len = sum(map(len, mappings))
num_len = len(df.first()[5:8])
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: 547
Feature vector length for numerical features: 3
Total feature vector length: 550
```

```
data = df.map(lambda r: LabeledPoint(extract_label(r), extract_features(r)))
first_point = data.first()
print "Raw data: " + str(first_point)
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: (18.0,[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
1.0,8.0,307.0])
Label: 18.0
Linear Model feature vector:
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
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,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,8.0,307.0]
Linear Model feature vector length: 550
```

```
# we need a separate set of feature vectors for the decision tree
def extract_features_dt(record):
    return np.array(map(float, record[2:8]))

data_dt = df.map(lambda r: LabeledPoint(extract_label(r), extract_features_dt(r))
first_point_dt = data_dt.first()
print "Decision Tree feature vector: " + str(first_point_dt.features)
print "Decision Tree feature vector length: " + str(len(first_point_dt.features))
```

```
Decision Tree feature vector: [3504.0,12.0,70.0,1.0,8.0,307.0]
Decision Tree feature vector length: 6
```

Train a Regression Model on the Car Dataset ¶

```
linear_model = LinearRegressionWithSGD.train(data, iterations=10, step=0.1, intercept=False)
true_vs_predicted = data.map(lambda p: (p.label, linear_model.predict(p.features)))
print "Linear Model predictions: " + str(true_vs_predicted.take(5))
```

```
Linear Model predictions: [(18.0, -1.0187537112768575e+35), (15.0, -1.1613490090535328e+35), (18.0, -1.0552325260820744e+35), (16.0, -1.0088061513748944e+35), (17.0, -1.0021725267765866e+35)]
```

```
# we pass in an empty mapping for categorical feature size {}
dt_model = DecisionTree.trainRegressor(data_dt, {})
preds = dt_model.predict(data_dt.map(lambda p: p.features))
actual = data.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: [(18.0, 18.133333333333333), (15.0, 18.133333333333333), (18.0, 18.133333333333333), (16.0, 18.133333333333333), (17.0, 18.133333333333333)]
Decision Tree depth: 5
Decision Tree number of nodes: 63
```

Decision Tree Model

```
# compute performance metrics for decision tree model
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: 7.9139
Decision Tree - Mean Absolute Error: 1.8721
Decision Tree - Root Mean Squared Log Error: 0.1912
```

Assessment:

It appears given these results for MPG the MSE is the best result at 7.9

In your solution, please leave the text of every problem as presented here. Add your solution below the problem statement. It is important for us and we will take points if you ignore this request. Please make sure that you provide numeric or textual results of your calculations. If there are no results, we will treat the problem as not addressed. Just providing some code without results will give you 0 point.
You are welcome to implement your solution in any language of your choice.

You are welcome to follow any other instructions and use any other programming or scripting language to accomplish the above goals.

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. 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 as separate files. 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. Please, submit to the class drop box. 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.