HU Extension Assignment 06 E63 Big Data Analytics

Handed out: 03/03/2017 Due by 9:30 AM EST on Saturday, 03/11/2017

**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 20% of you data for testing and use remaining data for training. Look initially at two variables: the horsepower and the displacement. Treat displacement as a feature and horsepower as the target variable (label). Use MLlib linear regression to identify the model for the relationship. Use test data to illustrate accuracy of the linear regression model 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.

**Approach:** Predict horsepower (label) using displacement (feature) by building a linear regression model. Input data is divided into training (80%) and test (20%) data.

**Assumptions**: all ‘NA’ values in the feature or target fields are converted to 0 in the python program. While this approach may not be the best, realized that filtering rows with NA values could be another option.

See attached p1.auto\_regression.py script. Detailed code level comments explain the stages in building the model and measuring its accuracy.

Program output is shown below -

|  |
| --- |
| **[cloudera@quickstart HW6]$** *spark-submit p1.auto\_regression.py*  Sample raw data: [u'18', u'8', u'307', u'130', u'3504', u'12', u'70', u'1', u'chevrolet']  Sample label (horsepower): 130.0  Sample feature (displacement): [307.0]  Sample labeled points: [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])]  Training data size: 330  Test data size: 76  Total data size: 406  17/03/08 18:58:04 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS  17/03/08 18:58:04 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS  Linear model parameters: (weights=[0.2901], intercept=1.0012)  Linear model predications (samples using test data): [(150.0, 92.236732234436531), (150.0, 88.175995595184602), (190.0, 113.12052066487499), (175.0, 111.09015234524902), (160.0, 98.617889810403838)]  Model accuracy using RMSE measure: 53.2866  Model accuracy using MAE measure: 50.2837  Model accuracy using RMSLE measure: 0.7553  *# plot regression line using pylab.plot()*  *from numpy import array*  *from pylab import plot, show*  *xdata = records.map(lambda r:float(r[2]))*  *xi = array(xdata.collect())*  *slope = linear\_model.weights.values[0] # see p1.auto\_regression.py for model params.*  *Intercept = linear\_model.intercept*  *yi = slope \* xi*  *plot(xi, yi, 'r-', xi, yi, 'o')*  *show()* |

**Problem 2.** Consider the entire data set. Regard mpg 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. Train your model using LinearRegressionSGD method. Use test data to assess quality of prediction for mpg variable. Calculate performance metrics of your model.

**Approach:** Predict mpg (target variable) using all available features (like horsepower, cylinders, weight, acceleration etc) by building a linear regression model.

**Assumptions**: all ‘NA’ values in the feature or target fields are converted to 0 in the python program.

Categorical variables: cylinders, model year, origin and car-name.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Categorical Variable** | **Column Index (0 based)** | **Type** | **Values** | **#unique values** |
| Cylinders | 1 | Ordinal | 3,4,5,6,8 | 5 |
| Model Year | 6 | Nominal | 70,71,72... | 13 |
| Origin | 7 | Nominal | 1,2,3 | 3 |
| Car Name | 8 | Nominal | \*Amc, audi, bmw... | 38 |

\*Data issue: chevrolet is misspelt in the data for an entry with chevroelt.

Numerical variables: displacement, horsepower, weight, acceleration

So in total, feature vector length after encoding categorical variables using 1-of-k binary is 5 + 13 + 3 + 38 + 4 = **63**

Input data is divided into training (80%) and test (20%) data.

See attached p2.mpg\_regression.py script. Detailed code level comments explain the stages in building the model and measuring its accuracy.

Program output is shown below -

|  |
| --- |
| **[cloudera@quickstart HW6]$** *spark-submit p2.mpg\_regression.py*  **# study distinct values for categorical variables**  Mapping of categorical feature (Cylinders): {u'8': 2, u'3': 0, u'5': 1, u'4': 3, u'6': 4}  Mapping of categorical feature (Model Year): {u'77': 0, u'76': 7, u'75': 1, u'74': 8, u'73': 2, u'72': 9, u'71': 3, u'70': 10, u'82': 4, u'80': 5, u'81': 11, u'79': 6, u'78': 12}  Mapping of categorical feature (Origin): {u'1': 0, u'3': 1, u'2': 2}  Mapping of categorical feature (Car Name): {u'buick': 31, u'subaru': 11, u'vw': 26, u'citroen': 32, u'chevroelt': 34, u'opel': 19, u'pontiac': 24, u'mercury': 0, u'chevrolet': 25, u'capri': 7, u'audi': 37, u'maxda': 5, u'dodge': 1, u'chrysler': 3, u'cadillac': 22, u'amc': 18, u'honda': 23, u'vokswagen': 13, u'ford': 27, u'mazda': 9, u'toyouta': 12, u'hi': 29, u'bmw': 30, u'mercedes-benz': 33, u'volkswagen': 36, u'peugeot': 17, u'fiat': 8, u'saab': 35, u'renault': 4, u'nissan': 2, u'toyota': 6, u'volvo': 21, u'chevy': 10, u'plymouth': 16, u'oldsmobile': 14, u'datsun': 28, u'mercedes': 15, u'triumph': 20}  **# understand feature vector length**  Feature vector length for categorical variables: 59  Feature vector length for numerical variables: 4  Total feature vector length: 63  **# extract feature vector for one sample input record**  First raw record (features): [u'8', u'307', u'130', u'3504', u'12', u'70', u'1', u'chevrolet']  First label: 18.0  Linear model feature vector: [0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,307.0,130.0,3504.0,12.0]  Linear model feature vector length: 63  **# build training and test data by sampling**  Training data size: 330  Test data size: 76  Total data size: 406  17/03/09 19:24:46 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS  17/03/09 19:24:46 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS  **# build linear regression model**  Linear model parameters: (**weights**=[2.72771404069e-07,1.20813474304e-07,-4.17724340937e-05,9.2559051894e-05,-3.23545909893e-06,2.30136715346e-06,2.5854302195e-08,-1.03925115682e-05,1.64537183446e-06,1.84800402421e-05,1.44670788068e-05,2.39055832126e-06,2.13178391676e-06,3.67075076764e-06,-3.1284761986e-06,-2.55238302765e-06,1.40913198859e-05,4.8139891225e-06,-1.97420870751e-05,4.28776006305e-05,2.48092299796e-05,-1.63443328945e-06,-1.79055741354e-06,8.76806884234e-07,-1.57227495002e-06,3.35433442238e-06,1.47793964368e-07,1.09676823663e-05,3.16036916345e-07,3.16562322974e-06,5.89498724778e-06,-8.2374723675e-07,1.83163754962e-06,2.50662073902e-07,7.0846725772e-07,-1.15257880529e-06,6.93846085496e-08,4.39873234858e-07,1.18772599292e-06,-2.25464522202e-06,7.99538559028e-07,0.0,2.20329011048e-07,-6.20939849518e-07,9.69427136764e-06,-2.56324585799e-06,-3.24821490897e-06,4.98671841998e-06,-2.30193914835e-06,1.32137591784e-05,0.0,6.1270976912e-07,-2.12768037599e-06,2.11130690598e-07,3.3044850334e-07,-4.0774014062e-07,-2.88289356834e-07,7.5218601765e-06,1.92924869872e-06,-0.00652568634513,-0.000566954315282,0.00732962801257,0.000964547573898], **intercept**=0.0)  **# predict using test data. Actual vs predicted for top 5 rows in test data ‘**  Linear model predications (samples using test data): [(18.0, 23.034936842414098), (16.0, 23.105269420263642), (15.0, 25.574461186702504), (28.0, 27.946739547484626), (14.0, 24.150834203120603)]  **# model accuracy**  Model accuracy using RMSE measure: 13.0378  Model accuracy using MAE measure: 11.0718  Model accuracy using RMSLE measure: 0.5669 |

**Problem 3.** Repeat the above analysis in Problem 2 with the decision tree method.Compare quality of the decision tree model and the linear regression technique using Performance metrics.

**Approach:** Predict mpg (target variable) using all available features (like horsepower, cylinders, weight, acceleration etc) by building a decision tree model.

**Assumptions**:

1. ‘NA’ values in the feature or target fields are converted to 0 in the python program.
2. Car-name is a categorical field with nominal string values. This is not suitable for decision tree and so will map each name to a unique integer before passing it in feature vector.

Total features: 8 (cylinders, displacement, horsepower, weight, acceleration, model year, origin, car name)

Input data is divided into training (80%) and test (20%) data.

See attached p3.mpg\_decisiontree.py script. Detailed code level comments explain the stages in building the model and measuring its accuracy.

Program output is shown below -

|  |
| --- |
| **[cloudera@quickstart HW6]$** *spark-submit p3.mpg\_decisiontree.py*  **# Map categorical feature car-name with string values to unique integers**  Mapping of categorical feature (Car Name): {u'buick': 31, u'subaru': 11, u'vw': 26, u'citroen': 32, u'chevroelt': 34, u'opel': 19, u'pontiac': 24, u'mercury': 0, u'chevrolet': 25, u'capri': 7, u'audi': 37, u'maxda': 5, u'dodge': 1, u'chrysler': 3, u'cadillac': 22, u'amc': 18, u'honda': 23, u'vokswagen': 13, u'ford': 27, u'mazda': 9, u'toyouta': 12, u'hi': 29, u'bmw': 30, u'mercedes-benz': 33, u'volkswagen': 36, u'peugeot': 17, u'fiat': 8, u'saab': 35, u'renault': 4, u'nissan': 2, u'toyota': 6, u'volvo': 21, u'chevy': 10, u'plymouth': 16, u'oldsmobile': 14, u'datsun': 28, u'mercedes': 15, u'triumph': 20}  **# sample raw record**  First raw record: [u'8', u'307', u'130', u'3504', u'12', u'70', u'1', u'chevrolet']  First Label: 18.0  Decision tree feature vector: [8.0,307.0,130.0,3504.0,12.0,70.0,1.0,25.0]  Decision tree feature vector length: 8  **# test vs train data split**  Training data size: 330  Test data size: 76  Total data size: 406  **# build decision tree model using train data**  Decision tree model: DecisionTreeModel regressor of depth 5 with 61 nodes  **# predict using test data**  Decision Tree predictions: [(18.0, 17.569230769230771), (16.0, 17.569230769230771), (15.0, 18.333333333333332), (28.0, 14.5), (14.0, 17.569230769230771)]  Decision Tree depth: 5  Desicion Tree number of nodes: 61  **# model accuracy**  Model accuracy using RMSE measure: 6.0329  Model accuracy using MAE measure: 3.2094  Model accuracy using RMSLE measure: 0.4498 |

Compare linear regression results vs decision tree

|  |
| --- |
| **# Linear Regression model accuracy**  Model accuracy using **RMSE** measure: 13.0378  Model accuracy using **MAE** measure: 11.0718  Model accuracy using **RMSLE** measure: 0.5669  **# Decision Tree model accuracy**  Model accuracy using **RMSE** measure: 6.0329  Model accuracy using **MAE** measure: 3.2094  Model accuracy using **RMSLE** measure: 0.4498  **Results**: Decision tree is clearly a better model for predicting car mpg using its numerous features. It is shown by all 3 measures of accuracy above. |

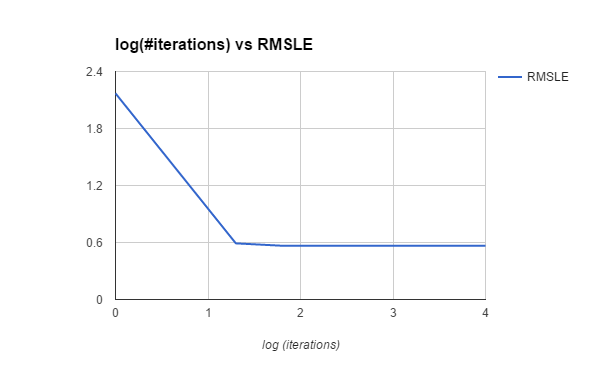
**Problem 4.** Now that your code works, investigate the impact of different parameter settings on model performance. Work with the linear regression model. Collect RMSLEfor several values of the number of iterations and plot those values as a graph. Present the number of iterations as the x-axis. Use the log scale. Present RMSLE as the y-axis. For y-axis use the linear scale. You can use any plotting tool including Excel. Subsequently,once/if you find the most optimal value of the number of iterations, vary the step size and again plot RMSLE on the y-axis and the logarithm of the step size on the x-axis. Tryto find an optimal step size. In production environment you would go back and forthbetween those two analyses. We do not expect you to do that here. Also, we do notexpect you to sweet this out. Several data points on each graph is fine.

**Solution:**

1. **Experiment-1**: Choosing following iterations: 1, 10, 100, 200, 1000, 100000 to evaluate the linear regression model performance. We will keep step size constant at 0.000001. From the results, it is clear that RMSLE quickly drops and stays the same for the model performance regardless of #iterations.

Step-Size: 0.000001

|  |  |
| --- | --- |
| **# iterations** | **RMSLE** |
| 1 | 2.17 |
| 20 | 0.591 |
| 60 | 0.567 |
| 100 | 0.567 |
| 200 | 0.567 |
| 1000 | 0.567 |
| 10000 | 0.567 |
|  |  |



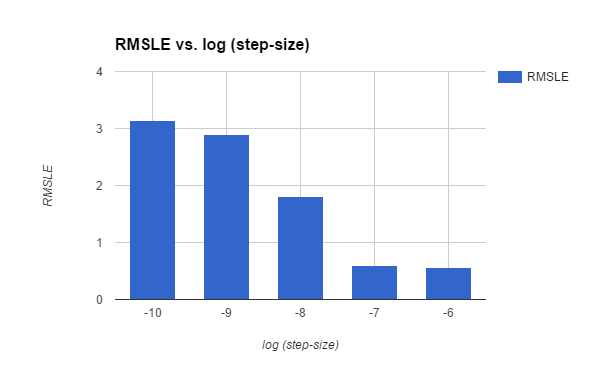
**Code snippet**: repeating below set of statements for various iterations

|  |
| --- |
| *>>> linear\_model = LinearRegressionWithSGD.train(train\_data, iterations=1, step=0.000001, intercept=False)*  *>>> test\_predict = test\_data.map(lambda p: (p.label, linear\_model.predict(p.features)))*  *>>> accuracy\_rmsle = np.sqrt(test\_predict.map(lambda (t, p): squared\_log\_error(t,p)).mean())* |

1. **Experiment-2**: We will keep #iterations constant at 200, but change step-size values. From the results, step-size does have an impact on RMSLE measure. As step-size reduces, RMSLE value actually increasing.

Iterations:200

|  |  |
| --- | --- |
| **Step Size** | **RMSLE** |
| 0.0001 | nan |
| 0.00001 | nan |
| 0.000001 | 0.567 |
| 0.0000001 | 0.588 |
| 0.00000001 | 1.815 |
| 0.000000001 | 2.891 |
| 0.0000000001 | 3.133 |



**Code snippet**: repeating below set of statements for various step sizes

|  |
| --- |
| *>>> linear\_model = LinearRegressionWithSGD.train(train\_data, iterations=200, step=0.000001, intercept=False)*  *>>> test\_predict = test\_data.map(lambda p: (p.label, linear\_model.predict(p.features)))*  *>>> accuracy\_rmsle = np.sqrt(test\_predict.map(lambda (t, p): squared\_log\_error(t,p)).mean())* |