Thomas Jean

CS 425

Project 1

10/6/2017

Multivariate Linear Regression

Data Exploration:

The auto-mpg data has 398 instances in 9 classes of information from vehicles from the 1970's and 1980's. There are 8 numeric classes with a mixture of discrete and continuous values. The last feature is the unique car name associated with the remaining features that if needed, could be used to find additional features that could be used, such as the MSRP (Manufacturer's Suggested Retail Price) of the car when launched, or tire size as an example. There are 6 missing values for horsepower that are unknown, but all other data are present. The samples are sorted by model year. We are fitting the data of 7 of the numerical classes to the remaining 1, namely the mpg of the vehicle.

Data Preparation:

dataInvestigation.py will produce the statistical information for each feature, and also shows the unique elements of discrete values. This information is used by the data processing algorithm. As an example, the number of cylinders is a discrete value. A car has a specific (discrete?) number of cylinders and the dataset only contains vehicles with 3, 4, 5, 6, and 8 cylinders. So to represent these discrete values, we break the feature into a vector of boolean values. A 3 cylinder car would now have its cylinder value represented as [1,0,0,0,0], a 4 cylinder car as [0,1,0,0,0], and so on. This is done to the other discrete values as well. For continuous values, I normalized by z-score using the following formula:

$$\frac{x_i - \mu}{\sigma}$$

If the quadratic option is set, before normalization the continuous values be used to add squared features to the data set. These values are then normalized as normal. This process is done by the dataProcessing.py script. For the imputation of missing values, I simply filled the missing horsepower with the mean horsepower as the only missing value is the horsepower and throwing away that data would be wasteful.

Implementation:

The particular implementation of linear regression solves for the solution directly using the normal equation:

$$wa = a(X^T X)^{-1} X^T y$$

Where X is the feature matrix and y is the corresponding mpg of the sample, and w is weight to corresponding features in X. We used the pseudo-inverse as an estimate of the inverse. The evaluation of the hypothesis done by two methods, one is the least squares error equation:

$$E(\theta|X) = \frac{1}{2} \sum_{i=0}^{n} [g(x^t|\theta) - y^t]^2$$

and a simple percent error, for another estimation:

$$E(\theta|X) = |\frac{y_{hypothesis} - y_{theortical}}{y_{theortical}}| \times 100$$

The percent error would not be used for optimization, but is a more understandable error function for grading different hypothesis.

The algorithm is tested by fitTest.py

Training and Testing:

I divided the data into two sets, 75 % is used for training and 25 % is used for testing. Specifically, 298 samples will be used for training and 99 for testing. dataFitting.py will shuffle the data to randomly distribute samples before dividing into the two partitions. Then the training set is used and the minimum solution is solved by the normal equation. The training and testing error for this hypothesis calculated by the least squares error and the percent error. The different models that are tested are the linear unnormalized, linear normalized, and a quadratic normalization fit.

For each model, I ran regression about a dozen times to see the general performance. As the data is shuffled each run will produce a different fit based on the different training set. I also modeled an example curve comparing the relation of horsepower and mpg. This is not an accurate representation of the line or curve in the higher dimensional space, but it shows a limited relation and helps visualize the fit. A better solution would be to reduce the data to 1 dimension and plot that, using some reduction method such as PCA or SVD, but as this project is about linear regression I selected the limited representation.

Linear Unnormalized:

This was the worst performing hypothesis with average errors of:

Average Training Square Error approx: 1680	Average Training Percent Error approx: 11.6 %
Average Test Square Error approx: 520	Average Test Percent Error approx: 12.4 %

I think the increased error comes from the discrete variables not being separated properly. The weight has a magnitude in the 1000's while other continuous terms are in the 100's, so normalization would allow them to be treated more equally. Figure 1 shows a potential 1-D example of such a fit.

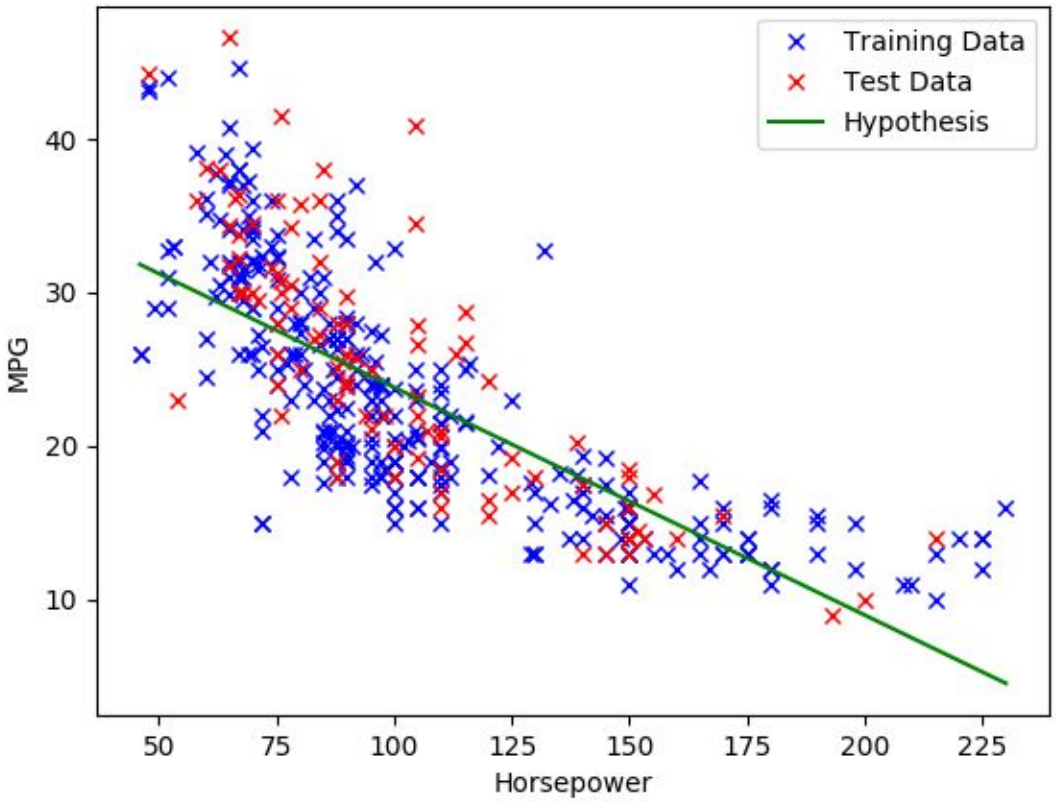

Fig. 1

Linear Normalized:

Average Training Square Error approx: 1180	Average Training Percent Error approx: 9.8 %
Average Test Square Error approx: 420.6	Average Test Percent Error approx: 9.6 %

With the improvement in discrete representation and normalization of continuous values, the average score of these models improved by 2%. The improvements come from the normalization and boolean representation of value. An interesting note is that the percent error of the training set is higher than the test set. I believe that the higher error is due to the high bias of the the linear model. It is so simple that is cannot fit the training set accurately. Figure 2 shows the normalization of continuous terms and how it maps to the change in the x-axis. The mean is now zero with a standard deviation of 1, and Figure 2 show how the 0 sits near the dense grouping of samples. Also as the data is shuffled, the training and testing set are different from the unnormalized example in Figure 1.

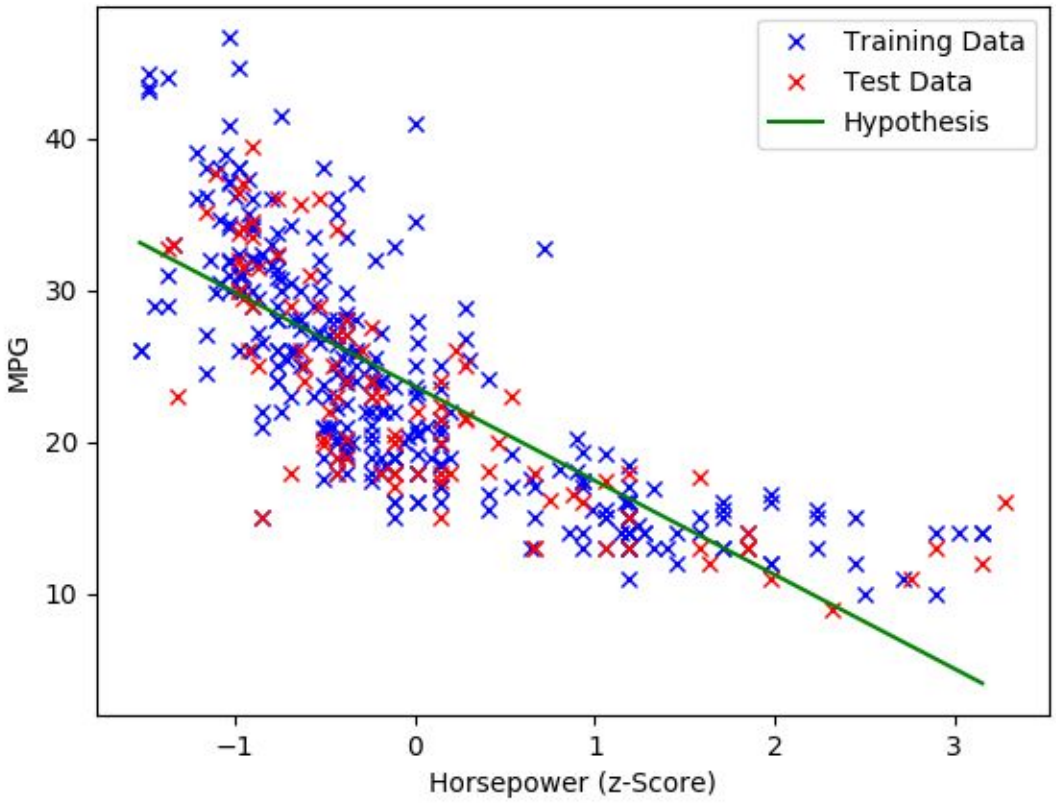

Fig. 2

Quadratic Normalized:

Average Training Square Error	Average Training Percent Error
approx: 924	approx: 7.9 %
Average Test Square Error	Average Test Percent Error
approx: 355.3	approx: 8.3 %

The Quadratic Normalized model improves the hypothesis from the Linear Normalized model by about 1.5%. The introduction of the squared continuous terms allows the hypothesis to vary with respect to the polynomial terms and better fit non-linear data. The training error has now dropped back below the test error showing that the model has enough power to better fit the given data. Figure 3 shows how the addition of polynomial terms better fit the trends in the data.

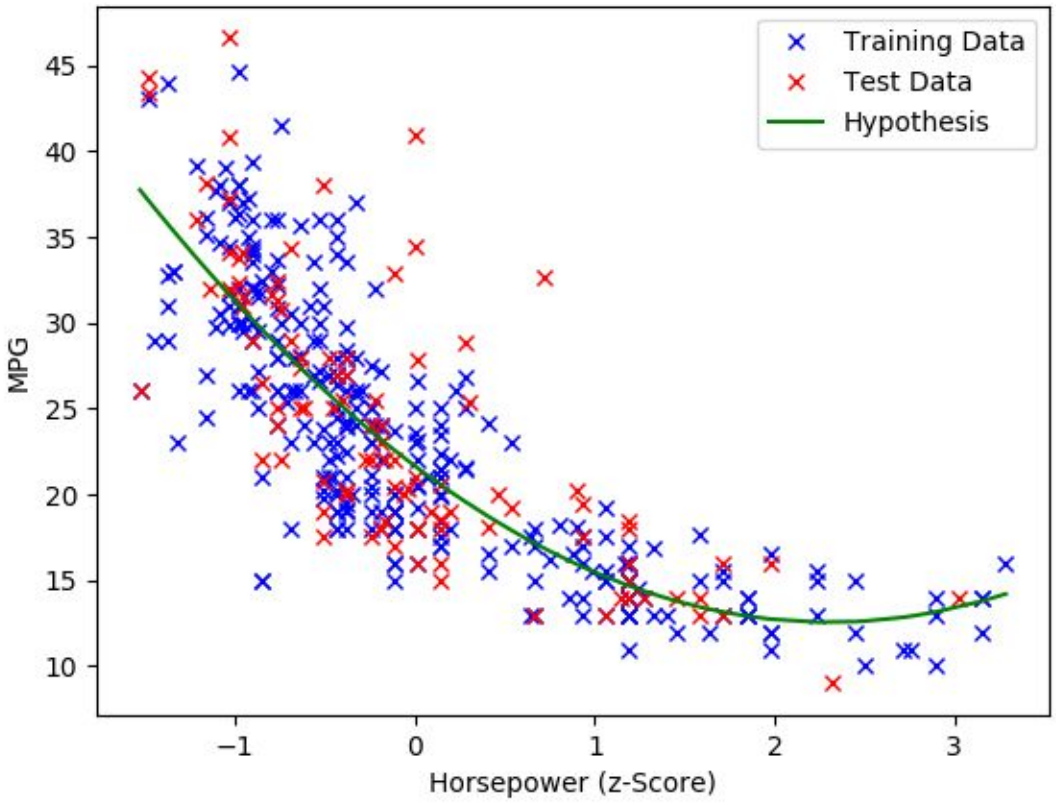

Fig. 3

Data Conclusions:

An an example of the importance of the particular features on one fit of the normalization quadratic polynomial regression the the following w vector was returned.

Weight of Feature	Feature Title	Weight of Feature	Feature Title
14.01	Bias	- 1.78	Model Year 70
- 4.10	3 Cylinder	- 1.67	Model Year 71
3.33	4 Cylinder	- 2.27	Model Year 72
4.96	5 Cylinder	- 2.54	Model Year 73
4.32	6 Cylinder	- 1.05	Model Year 74
5.49	8 Cylinder	- 0.55	Model Year 75
- 6.17	Displacement	- 0.47	Model Year 76
- 2.2	Horsepower	0.65	Model Year 77
- 10.17	Weight	1.47	Model Year 78
- 3.95	Acceleration	3.13	Model Year 79
7.69	Model Year 80	5.19	Origin 3
5.04	Model Year 81	4.65	Displacement^2
6.36	Model Year 82	0.07	Horsepower^2
4.24	Origin 1	7.29	Weight^2
4.57	Origin 2	3.40	Acceleration^2

These results can be analyzed logically, weight has the biggest effect on the mpg as a lighter car would require less energy to move, but as a vehicle gets larger, it may require a larger engine that is more energy efficient. Also, there is a general trend of newer vehicles having better mpg overall. Another note about these features is that they are not independent from each other, a larger engine is likely to produce more horsepower and have larger engine displacement. The effect of these features overlap, so the fundamental truth of the relationship may be obfuscated, but still predictable.

Program Information:

Programs Included:

dataInvestigation.py
dataProcessing.py
dataFitting.py
fitTest.py

Dependences include:
Python 3
pandas
    - some of pandas feature are dependent on scipy
numpy
matplotlib

Sample invocations for the programs:

python3 dataInvestigation.py auto-mpg.data
- produces auto_mpg.csv

python3 dataProcessing -1 auto_mpg.csv
- produces auto_mpg_processed.csv

python3 dataProcessing -2 auto_mpg.csv
- produces auto_mpg_processed_2.csv

python3 fitTest.py

python3 dataFitting.py -r -1 auto_mpg.csv
- produces a non-normalized linear fit

python3 dataFitting.py -p -1 auto_mpg_processed.csv
- produces a normalized linear fit

python3 dataFitting.py -p -2 auto_mpg_processed_2.csv
- produces a normalized quadratic fit

Additional Information:

When developing the pipeline for the project, I attempted to keep the label in the further processed data. This had a negative impact of the code as more of the code had to be special cased based on the number of features. Next time, I should remove all non-numerical data one the data is processed.