Multivariate Polynomial Regression

COSC528 - Fall 2018
Project 1
10/08/18

Rekesh Ali

1 Introduction

Given a list of cars with several attributes including mpg, horsepower, and weight to name a few, the goal of this project is to be able to predict a car's gas mileage (mpg) using it's other specifications. To do this, we must build a proprietary polynomial regression tool that allows us to test different feature combinations against the gas mileage in order to zone in on a model that best predicts a car's gas mileage. Presented here are first and second degree polynomial trials. In addition to error estimates this report includes discussion of feature selection and the effects of data standardization on the model.

2 Data Preparation

The data acquired describes 398 instances split into 9 classes described as the following: mpg, cylinders, displacement, horsepower, weight, acceleration, model year, origin, and car name. Since we are predicting the car's mpg, it will be used as our solution vector during model training. The attribute 'car name' is unique for each instance and entirely arbitrary with no basis on a car's mpg, so it will be ignored in the model's training stage. The other specifications may have a direct physical influence on a car's mpg, so it would be unwise to discard any additional attributes. Thus, the best model will implement a combination of the 7 remaining features fitted to the car's mpg.

Upon manual inspection of the data, it appears the 'horsepower' feature has 6 missing values in the data set. There are a few different strategies available for imputation. The simplest would be to replace these values with the mean for the other instances. The dataset is large enough that the mean is a fair representation of the expected values, so this is a safe option. Another option is to use the 'car name' attribute to search for the actual values. Although much more accurate this method requires a bit more work, which for the amount missing in this set is no large problem, but difficulty will scale as more values turn up missing in future datasets. Finally, one could train a model against the cars' horsepower, and predict it that way. For simplicity, we will impute the missing values with the dataset's average.

Since there are a mixture of continuous and discrete data types in the remaining 7 attributes, the dataset needs to be tweaked. The continuous data need not be touched, but the discrete values must be split into sub-attributes. For example: any car within the dataset can have a value for origin of 1, 2, or 3 and since it is not physically correct to be in-between these values, we will split the attributes into 'origin1', 'origin2', and 'origin3'. Now that there are 3 sub-attributes for this attribute, any car will have either a 0 or a 1 for their instance according to what their origin value is. Keep in mind this does not add any features to the dataset, and if origin is selected as a feature, it will load all of the sub-origins and their respective values.

To mitigate potential bias in the data collection process, the data will be randomized before being split into a training and a test set. The dataset will be allocated 75% and 25%

to the training and testing sets, respectively.

To account for multicollinearity in the dataset and interaction of values in the higher orders of the second degree polynomial model, the samples for each continuous attribute will, in addition to the raw data analysis, be standardized according to z-normalization. Specifically, a sample's z-score is computed according to **Eq.1**, where μ_{ij} is the sample mean for the attribute and σ_{ij} is the standard deviation. This study will look at the effects of normalizing with respect to the full dataset and with respect to the individual training or test sets.

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j},\ i = 1 : N_{samples},\ j = 1 : N_{attributes} \tag{1}$$

3 Implementation

3.1 Model

There are four separate but very similar models considered in this project. There are two models that take raw data and two that take standardized data, each model corresponding to a polynomial order 1 or 2. We will only be going up to 2 because the literature suggests that linear models generally do very well and higher order models will show strong bias against a test set. The model that regresses raw data follows the standard regression protocol. The features in question are lined up side by side in a matrix **X** with the first column being a column of ones for the intercept weight. The solution values are given their own single column matrix, **R**, and a corresponding weight matrix is of the size features + 1 (to account for the intercept), denoted **W**. The general model is shown below, and **Eq.2** shows us how to solve for the unknown weights.

$$\begin{aligned}\mathbf{X}\mathbf{w} &= \mathbf{r}\\ \mathbf{X} = [1, \mathbf{x}_1, ..., \mathbf{x}_j],\ \mathbf{x}_j &= [x_{1j}, ..., x_{ij}]^T\\ \mathbf{w} = [w_0, ..., w_j]^T,\ \mathbf{r} &= [r_1, ..., r_i]^T\\ \mathbf{w} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{r}\end{aligned} \tag{2}$$

The above formulation is an example of the linear model, the quadratic one is slightly different. It has all the same values as the linear **X** and **w** matrices, but it also has all 2nd order multiples of the individual features as well. This leads to an additional $j!/(2(j-2)!)$ combinations of chosen features to account for all possible dependencies. Thus, finding all combinations for **X** is done iteratively, and **w** is trivial and will take care of itself through **Eq.2**. The solution matrix **r** remains the same.

The reason the normalized data requires a different model is because it does not have intercept values $\mathbf{x}_0 = [1_1, ..., 1_i]^T$ and the corresponding weight w_0. This is because all values are normalized with their average equal to zero. Thus, the matrices above lose the corresponding dimension. Everything else regarding implementation of a 2nd order model remains the same.

The mean squared error, E_{LS}, is what is being minimized by the above configuration, so it will be used to check the capability of our model. **Eq.3&4** show that for the raw and standardized data sets, respectively. Error is also estimated using a percent error formula, **Eq.5**. The error will be computed for each instance and averaged for a simpler view of the performance. The errors are also averaged over many iterations on randomized data for a better statistical representation of the model. The normalized error, $E_{z\%}$, is shown in **Eq.6**.

$$E_{LS} = \frac{\sum(\mathbf{r}_{test} - \mathbf{r}_{true})^2}{N_{samples}} \tag{3}$$

$$E_{zLS} = \frac{\sum(\sigma\mathbf{r}_{test} - \sigma\mathbf{r}_{true})^2}{N_{samples}} \tag{4}$$

$$E_{\%} = 100 \times mean\left(\frac{\mathbf{r}_{true} - \mathbf{r}_{test}}{\mathbf{r}_{true}}\right) \tag{5}$$

$$E_{z\%} = 100 \times mean\left(\sigma\frac{\mathbf{r}_{z/true} - \mathbf{r}_{z/test}}{\sigma\mathbf{r}_{z/true} + \mu}\right) \tag{6}$$

3.2 Program

To get a really good estimate of the model's capabilities, the data is randomized and trained 100 times before settling on an error estimate. Throughout each iteration, every single possible combination of features is trained and tested, where at the end of the iteration only the top 5 with the lowest error are chosen to represent the model. The program does this by equating the feature indices to a binary array, where there are $2^j - 1$ possible combinations. This is very computationally intensive, and there is certainly a simpler way to do pinpoint relevant features, but it was a good exercise and piqued curiosity. Once finished, a tally of which features appeared in the top 5 per iteration, and how many times, is created and divided by the total number of iterations times the total possible amount of times it could appear (5). The result is a probability array showing for each n order model what the likelihood is that a certain feature will appear in the most superior models. In addition to the likelihood array, the program counts how many times a combination of features shows up in the top 5 and outputs the one that showed up the most along with its average errors and number of appearances. main.py calls these error modules from tools.py. tools.py also contains the automated routines that run through the binary array for feature selection. explore.py compiles the raw data from auto-mpg.data into a dictionary using keys from auto-mpg.names. prepare.py contains modules that split discrete key value pairs into subsets as well as modules for imputation, statistics, randomization, and normalization. The matrix building, feature set training, and feature set testing are all handled in implement.py. The current build is available on github at rekeshali/multi-regress.

4 Results

4.1 Polynomial Order

Table 1 shows a quick snippet of the error differences, where each row is a different set of 100 iterations. The light green and red bands indicate min and max values of the averaged error for that combination of hyper-parameters (order and norm). Also, the two errors were not computed concurrently, so they are not officially correlated with each other. However, given the number of training iterations, it should be a good statistical representation (1000 iterations were also tried with little change).

The first and second order model seem very close to each other in terms of percent error, with the difference between the worst and best being roughly 1-2%. However, the real indicator of performance is the mean squared error, and we see that for both normalized and non-normalized datasets the increase in polynomial order results in a positive change in performance, as demonstrated by the consistent decrease in E_{LS} across the board. The percent difference in max mean squared error between polynomials of order 1 and 2 for a normalized dataset, for example, is quite staggering at a whopping 20% change. Again, an increase into even higher order regression may cause extreme fitting as mentioned by the literature. Thus from these error estimates we can conclude that an increase of polynomial order from 1 to 2 enhances the model's predictive capability.

Min	Max	Run 1	Run 2
Order	Norm	E_{LS}	E%
1	No	894.68	10.28%
1	No	885.22	9.92%
1	Yes	934.90	10.16%
1	Yes	874.77	9.87%
2	No	862.43	9.28%
2	No	846.18	9.24%
2	Yes	781.61	8.72%
2	Yes	779.22	8.69%

Table 1: Mean square and percent error of the test cases for $O(1, 2)$, standardized and not.

4.2 Normalization Effects

Also in **Table 1** are the normalized and non-normalized min and max errors for each polynomial order. For the first degree fit, we can observe that standardizing our data does not do much to augment the performance of our model, in fact it is clear that it can even hurt the first degree fit as indicated by the increase in max mean square error due to normalization. However, looking at the normalization trend in the rest of the table, this increase in error could be attributed to the randomized datasets not being robust enough. Even the errors that improved didn't do so by much, so it seems the strongest argument for normalizing the linear dataset is that it reduces the weight vector by one and can save on computation time.

The real benefits from z-normalizing the data seem to reveal themselves very strongly for the 2nd order polynomial. For example, the max mean squared error decreases by nearly 10% and the minimum by 8%. Comparing this to the increase in the max and the meager decrease in the minimum of the 1st order fit, it is clear that due to the interactions between higher order terms, the quadratic terms benefit more from the standardized datasets than the linear terms.

4.3 Feature Analysis

Table 2 shows the feature arrays that represent the likelihood a feature will appear in the top 5 best models across a number of iterations, where the probabilities are color coded for easier trend analysis. As demonstrated in the previous sections, it seems polynomial order is more dominant over what features capture the best curve. Notice the color code validates this even further by highlighting the similarities between models of the same order and the differences between models of different order. The feature combination does not change much with data standardization in the first order which matches the error analysis from above, for how could the error change much if the model features are the same? Again just like above, data normalization on a 2nd degree fit really shifts the image of the model, and the error along with it. For example, the decreased appearance of the acceleration as a strong feature is clear evidence that z-normalization relieves collinear influence from the model. This makes sense because a lot of these features physically depend on each other. I.e. more acceleration may be the result of a higher horsepower or a lower weight or a bigger engine so on and so forth. Finally, it seems that the really strong indicators of gas mileage are ubiquitous throughout both polynomial orders, and going down the table yields better returns for much less effort.

Order	Norm	E_{LS}	Cylinders	Displace	HorsePow	Weight	Accel	Year	Origin
1	No	885.22	89.40%	50.00%	72.00%	98.80%	43.80%	100.00%	84.20%
1	Yes	874.77	87.40%	49.60%	68.20%	97.40%	44.80%	100.00%	84.40%
2	No	846.18	42.20%	27.80%	46.80%	78.00%	86.40%	100.00%	58.00%
2	Yes	779.22	45.80%	31.60%	43.00%	88.40%	44.60%	100.00%	30.20%

Table 2: Likelihoods of a feature appearing in a model for $O(1,2)$, standardized and not.

5 Conclusion

There are many interesting things to note as a result of this study. First, linear models seem to have higher dependence on the dimensionality of the dataset; the 2nd order polynomial can get by with fewer features, indicating an increased flexibility to fit the underlying trends. Second, z-normalization decouples and reduces redundant influence from highly collinear attributes, a perk that becomes more apparent with the model's flexibility. Lastly, the above results indicate that the best model for predicting a car's mpg is a 2nd degree polynomial with a normalized dataset, where the most determinative features are weight and model year. This agrees with logic because more work is required to drive a heavier car, which leads to more overlapping compensation from the other features, and innovation in the industry leads to fuel efficiency that gets introduced as the model years become more recent. The advantages of such a model include a very condensed subspace for simpler data acquisition, analysis, and a dramatic reduction in computational effort. As a final validation of this logic, this exact model was trained 10 times at 1000 randomizations each, yielding about a 9% error overall from just these two features. The end result demonstrates how in-depth analysis can reveal certain characteristics and make representation of phenomena much simpler by exploiting the knowledge tucked away within endless data.