Project 1 Report
Multivariate Regression

Ian Lumsden

October 9, 2018

1 Objective

The goal of this project was to implement multivariate regressors to estimate the miles per gallon (MPG) for a car based on other features (which will be described in section 2). In the first part of the project, a linear regression algorithm was implemented, including imputation and data standardization. Then, to improve the accuracy, a polynomial algorithm was created, and cross validation was used to determine the polynomial degree that produced the most accurate results. Finally, feature importance was analyzed to determine which features were most important. All of the code for this project is contained in "project1.ipynb".

2 Preprocessing

The data for training and testing was provided in "auto-mpg.data". This is a CSV file containing 398 rows, with 9 attributes each. The attributes are listed below with their type of random variable:

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

It was known that the horsepower column of the CSV contains 6 NaN values. This data was read using Python's Pandas library.

Due to its guaranteed uniqueness, the car name feature provided no useful information to the regression algorithm. As a result, it was removed from the dataset. No other data was completely removed from the data.

Then, the NaN values from the horsepower column were replaced to ensure that the numpy library could be used to access the matrix operations that are needed to perform regression. Given the size of the sample, it was fair to assume that the sample mean was representative of the population mean. Therefore, the NaN values were replaced with the mean of all valid values in the horsepower column.

2.1 Standardization

To standardize the data, statistics for each feature first needed to be obtained. The numpy library was used to obtain the following statistics per feature: mean, standard deviation, minimum, maximum, quartiles, and number of elements. Although all these statistics were calculated, only mean and standard deviation were used.

To standardize the data, all elements were substituted for their z-scores with respect to their feature's mean and standard deviation. The equation for z-score is

$$z_i = \frac{x_i - \overline{x}}{\sigma}, \tag{1}$$

where $\overline{x}$ is the mean of the feature and σ is the standard deviation. Since all data was numeric, this strategy was used for all features, regardless if they were continuous or discrete. Other options were considered for discrete features, but they were eventually replaced because using z-scores produced the smallest errors.

Next, the data was separated into training and testing sets. Eighty percent of the data was used for the training set, and the remaining twenty percent was placed in the testing sets. The data was separated randomly using numpy's random.choice function. Then, each set of data was split to separate the input data from the MPG output data. Finally, a column of 1's was added to the end of the input data for both the training and testing datasets. This was required for the algorithms described in the next section.

3 Implementation

For linear regression, the standard multivariate algorithm was used. The algorithm produces the model weights using the matrix $\mathbf{X}$ and the vector $\mathbf{r}$. The preprocessing allowed the input data to directly be used as $\mathbf{X}$ and the output data (MPG) to be used as $\mathbf{r}$. The regression equation is

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{r} \tag{2}$$

It was directly implemented using various numpy matrix and linear algebra functions.

The polynomial regression algorithm was created by analyzing the following multivariate polynomial hypothesis:

$$g(\mathbf{x}^t|w_0, w_1, w_2, ...) = w_0 + w_1x_1^t + w_2(x_1^t)^2 + w_3(x_1^t)^3 + ... + w_{d+1}x_2^t + w_{d+2}(x_2^t)^2 + ... \tag{3}$$

If each power of a variable (i.e. $(x_1^t)^2, (x_1^t)^3$, etc.) were replaced by a different variable, the polynomial regression problem becomes a linear regression problem. For a quadratic polynomial, equation 3 becomes

$$g(\mathbf{x}^t, \mathbf{y}^t|w_0, w_1, w_2, ...) = w_0 + w_1x_1^t + w_2y_1^t + w_3x_2^t + ... \tag{4}$$

Therefore, the implementation of polynomial regression used for this project began with adding columns for the variable powers to the input datasets. For example, for a quadratic regression, one extra column representing the corresponding feature squared was added to the training and testing input datasets. Then, the linear regression algorithm was applied to

the expanded dataset to obtain the weights. Finally, cross validation was used to determine the best polynomial degree. More information about cross validation can be found in section 4.

For both types of regression, prediction is performed by simply taking the dot product of the weights and an instance of input data. Additionally, both algorithms measure their accuracy using mean square error, which is defined by the following equation:

$$E(\mathbf{w}) = \frac{1}{N} \sum_{(x,y)\in S} (f(\mathbf{x}|\mathbf{w}) - r)^2 \tag{5}$$

where $f(\mathbf{x}|\mathbf{w})$ is the model's prediction and r is the expected output. Errors were obtained for both training and testing.

With standardization, the linear regression produced a training error of 0.17902157 and a testing error of 0.18349040. After cross validation and standardization, the polynomial regression produced a training error of 0.12768897 and a testing error of 0.10803149. Although not explicitly listed, the errors for both models without standardization were much greater than those listed above.

4 Cross Validation

The polynomial regression model has a single hyperparameter - the degree of the polynomial. To determine the best degree value, cross validation with ten iterations was used. For each iteration, the training set was randomly distributed between a cross validation training set and a validation set. Three-quarters of the data was placed in the new training set, and the remaining quarter was placed in the validation set. Then, using this newly separated data, training was performed for each valid degree $n = 1, 2, 3, ...$, and training and validation errors were obtained for each of the trained models. Then, the errors for each degree were averaged. These average errors were used to determine which degree produced the most accurate model. In the case of two degrees producing similarly accurate models, the lower degree was selected. The following pseudo-code in algorithm 1 describes the process.

This cross validation algorithm produced the following training and validation errors:

Degree	Training Error	Validation Error
1	0.17946316	0.18433883
2	0.12878658	0.13184075
3	0.13356736	0.15199603
4	0.11362017	0.16263602

The table shows that, in general, the training error decreased with respect to degree. However, the validation error decreased initially before beginning to increase rapidly starting with a degree of 3. This suggests that the model overfits the data for degrees greater than or equal to 3. As a result, the model is most accurate and generalizable with a degree of 2, which is what the cross validation algorithm recommended. Additionally, the testing error for a degree of 2 was 0.10803149, further supporting the conclusion that a degree of 2 is best.

Algorithm 1 Cross Validation Pseudo-Code

```
train_errors ⇐ [[], [], [], [], ...]
valid_errors ⇐ [[], [], [], [], ...]
for i in range(10) do
    X_train ⇐ 75% of input data
    r_train ⇐ 75% of output data
    X_valid ⇐ remaining 25% of input data
    r_valid ⇐ remaining 25% of output data
    for degree in range(1, ...) do
        weights ⇐ Output of Polynomial Training with degree=degree
        Append training error to train_errors[degree]
        Append validation error to valid_errors[degree]
    end for
end for
Average each subarray in train_errors to a single number
Average each subarray in valid_errors to a single number
Create square matrix of differences in average validation errors
for row in difference matrix do
    if row contains one 0 and all other elements are negative then
        Set the best degree as the degree the row represents
        break
    else if row contains negatives and two 0's then
        Find the degree corresponding to the 0 not on the diagonal
        if Degree corresponding to row < Degree corresponding to 0 then
            Set the best degree as the degree the row represents
        else
            Set the best degree as the degree corresponding to 0
        end if
        break
    end if
end for
Return the best degree
```

5 Feature Importance

To determine which features are most important, feature columns were removed from the left of the input matrix one at a time. The polynomial regression model was then trained using the reduced input matrix, and training and validation errors for each reduced matrix were obtained. The errors are shown below:

Features Removed	Training Error	Validation Error
None	0.11869527	0.16804822
Cylinders	0.85693080	1.13923897
Displacement and above	0.93398567	0.85185433
Horsepower and above	0.84706621	1.11311711
Weight and above	0.94897475	0.81267632
Acceleration and above	0.82813251	1.20102170
Model Year and above	0.86448217	1.19860874
Origin and above	0.91718472	0.93357592

From the table, it is clear that dropping the cylinders, horsepower, and acceleration features caused the largest increase in error. This suggests they are the most important features. Several features, namely displacement and weight, also seem to cause large decreases in validation error when dropped. Oddly, dropping these same features seems to increase training error. It is possible that this trend has to do with the purely random way in which the training, validation, and testing data is separated. The remaining features cause relatively minor changes in error. This suggests they do not affect the model very much, and, as a result, they could likely be dropped with minimal affects on the model.

6 Conclusion

From this analysis, one can conclude that the best polynomial regression algorithm for estimating a car's mileage based on the given seven features is a quadratic model trained on a dataset standardized with z-scores. Of the seven features, the number of cylinders, horsepower, and acceleration were shown to be the most important to the accuracy of the model. This also implies that the imputation of the missing values for horsepower is very important. However, some features increased training error while decreasing validation error or vice versa. As a result, it is likely that the algorithm could be improved by determining a different way of separating the training, validation, and testing data.