COSC 528 Project 1 Report
Linear and Polynomial Regression

Devanshu Agrawal

October 4, 2017

1 Objective

The objective of this project was to build linear and polynomial regression models to predict a car's gas mileage from eight other car features including cylinders, displacement, horsepower, acceleration, origin, model year, weight, and car name.

2 Preprocessing the Data

Since the "car name" feature is unique for every instance in the data, then it provides no information that could allow our model to generalize to unseen data, and hence we discarded this feature. This left us with seven features to use as possible predictors for gas mileage.

The "horsepower" feature had six missing values. We imputed these with the average of all other horsepower values that were given; this imputation strategy assumes that all given horsepower values comprise a sufficiently large sample so that its sample mean agrees with the population mean. Under this assumption, our imputation strategy is unbiased. We saved the imputed data as a numpy array in the file "Data.npy".

See Section 3.2 for an explanation of how we standardized the data.

3 Implementation

3.1 The Model

For linear regression, we constructed the augmented matrix $\mathbf{A}$ by concatinating a column vector of 1's to the beginning of the data matrix $\mathbf{X}$.

For polynomial regression, additional columns must be appened to $\mathbf{X}$. Consider a degree-n polynomial $f(\cdot; \mathbf{w})$ in the feature variables $x_1, \ldots, x_D$. A term of degree k in this polynomial has the form

$$w_{i_1 i_2 \ldots i_D} x_{i_1} x_{i_2} \cdots x_{i_D},$$

where $1 \leq i_j \leq D$ for all j. Since the variables x_{i_j} commute, then we may assume without loss of generality that $i_1 \leq i_2 \leq \ldots \leq i_D$. We recursively generated all valid sets $\{i_1, \ldots, i_D\}$

(see "Terms.py"). For each generated set, we multiplied the corresponding columns of $\mathbf{X}$ element-wise and appended the resulting product to the end of $\mathbf{X}$. Also appending a column of 1's to the beginning of $\mathbf{X}$ gives the augmented matrix $\mathbf{A}$.

In polynomial regression, the parameters of the model $f(\cdot; \mathbf{w})$ satisfy the equation

$$\mathbf{A}\mathbf{w} = \mathbf{y}.$$

The solution to this equation is

$$\mathbf{w} = (\mathbf{A}^\top A)^+ \mathbf{A}^\top \mathbf{y}$$

(see "Regressors.py").

3.2 Training and Validation

We randomly selected 20% of the data instances to be set aside as a test set. We further split the remaining 80% of the data into a training set and validation set (60% and 20% of the original data set respectively). We computed the mean and standard deviation of every feature in the training set and used them to standardize (i.e., z-normalize) the training set. At validation and test time, we used the same training mean and standard deviation to standardize the validation and test sets.

We fit the model $f(\cdot; \mathbf{w})$ to the training set by following the implementation outlined above. We then evaluated the trained model on the training and validation sets by computing the mean squared error

$$E(\mathbf{w}) = \frac{1}{N} \sum_{(\mathbf{x},y)\in S} [f(\mathbf{x}; \mathbf{w}) - y)^2,$$

where S is either the training or validation set. It should be kept in mind that we often evaluated our model in terms of its performance averaged over several random training-validation splits

4 Hyperparameter Optimization

We used cross validation to optimize the single hyperparameter in our model– the polynomial degree. For each degree $n = 1, 2, 3, \ldots$, we trained our model and recorded its average mean squared error on both the training set and validation set over 10 random training-validation splits of the data.

We specifically implemented the following pseudocode:

```
1. set aside 20% of the data as a test set
2. for count in range(10):
3. split data into training and validation sets
4. for degree in {1, 2, 3, 4}:
5. train regressor R(degree) on training set
6. compute training and validation errors
7. compute average training and validation errors
8. # average is taken over the 10 training-validation splits
```

(see "HyperParameterOpt-new.py" for the complete code).

The training error strictly decreases with increasing degree (see table). The degree-4 polynomial fits almost perfectly to the training set. However, although the validation error decreases initially, we see that it reverts and rapidly increases starting at degree 3. This means that the cubic and quartic models severely overfit to the training set. The validation error is lowest at degree 2, and hence we conclude that a quadratic polynomial provides the best fit in the sense that it achieves low training error and is simultaneously able to generalize to unseen validation data as indicated by its low validation error.

We confirmed our choice of a quadratic model by evaluating its performance on the test set; the trained quadratic model produced an error of 0.11940 on the test set. The quadratic model therefore performs comparably well on the test set – relative to the training and validation sets – and hence exhibits strong generalizability.

degree	training error	validation error
1	0.1780	0.17248
2	0.10477	0.13056
3	0.04465	2.2651
4	3.5301e-16	189.49

Note the tabulated data is also stored in "HyperparameterOpt-new-out.txt".

5 Feature Selection

We were interested to see if we could reduce the dimension of the parameter space and simplify our model by eliminating unnecessary features. Our approach was simply to delete one feature at a time and train our model on all remaining features. The feature whose deletion results in the lowest training and validation errors is then the least important feature (since it least affects the model). A recursion of this feature deletion let us rank the features in order of increasing importance.

We specifically implemented the following pseudocode:

```
01. set aside 20% of the data as a test set
02. let F be the set of seven features
03. while F is not empty:
04. for count in range(20):
05. split the non-test data into training and validation set
06. for feature f in F:
07. train quadratic regressor R on training set ignoring f
08. compute training and validation errors
09. compute average training and validation errors TE(f) and VE(f)
10. # the averages are taken over 20 training-validation splits
11. let f_* = argmin(TE(f)**2 + VE(f)**2)
12. # assumption: we weight TE(f) and VE(f) equally
13.  delete f_* from F
```

(see "FeatureSelection-new.py" for the complete code).

We found that we can drop five of the seven features without a significant increase in error. The error increases significantly only after we drop "model year" and then "weight" (see table). The order in which the first five features were dropped varied over several runs of our program; this is consistent with the result that the errors do not change significantly as the first five features are dropped.

It appears then that "model year" and "weight" are the most important features for predicting gas mileage. We confirmed this by applying a quadratic model trained only on "model year" and "weight" to the test set; we obtained a testing error of 0.12986. This is comparable to the model's performance on the training and validation sets. We therefore conclude that the model year and weight of a car are sufficient predictors for a car's gas mileage.

Features Dropped	Training Error	Validation Error
none	0.09646	0.15233
disp.	0.10583	0.12670
cyl. and above	0.10733	0.13861
accel. and above	0.11096	0.13329
orig. and above	0.12705	0.12649
HP and above	0.13809	0.15005
MY and above	0.28113	0.30239
weight and above	1.0000	1.0340

Note the tabulated data is also stored in "FeatureSelection-new-out.txt".

6 Conclusions

We conclude from our analysis that the best polynomial regressor for predicting a car's gas mileage given seven other features is a quadratic model trained on a standardized (i.e., z-normalized) training set. In addition, we found that the model year and weight of a car provide sufficient information for accurately predicting its gas mileage. This significantly simplifies our model. In particular, the imputation of missing values for horespower in the data set is now irrelevant.

Recall that the mean squared errorof our quadratic model trained only on model year and weight and evaluated on the test set was approximately 0.13 (reported in Section 5). The root mean sqquare (RMS) error is therefore $\sqrt{0.13} \approx 0.36$. But recall also that this is error in the standardized gas mileage. To convert this error into the original units of miles per gallon, we need to multiply the error 0.36 with the standard deviation of gas mileage over the training set. The training standard deviation varies with every random training-validation split. But let us approximate the training standard deviation with the standard deviation in gas mileage over the entire original data set, which is approximately 7.8 $\frac{\text{mi}}{\text{gal}}$. The prediction error of our model is therefore approximately

$$0.36 \times 7.8 \quad \frac{\text{mi}}{\text{gal}} \approx 1.0 \quad \frac{\text{mi}}{\text{gal}}.$$

Our model therefore predicts gas mileage with an uncertainty on the order of about 1 $\frac{\text{mi}}{\text{gal}}$.