Lecture 7: Regression II: Theory and Practice
Modeling Social Data, Spring 2017
Columbia University

Yi Da Jeremy Ng

March 3, 2017

1 How to Evaluate Models

Currently, we try to find the best parameters by visually inspecting the fit:

1. We start by plotting the *predicted* and *actual* values according to the model
2. We can break this down further using a different *color* for each gender
3. We can color by age, and *wrap* by gender to make this clearer

Note: *Discrete variables* (such as gender) are automatically colored with a discrete color scale, while *continuous variables* (such as age) are automatically colored with a continuous color scale.

2 Modeling Variability

However, all this is done by summarizing the *average* of each gender/age combination, which might cause us to overlook some variability in our observations. For instance, plotting every point shows that variability *within* age is higher than variability *across* age in the figure below), which reveals the deficiencies of our model.

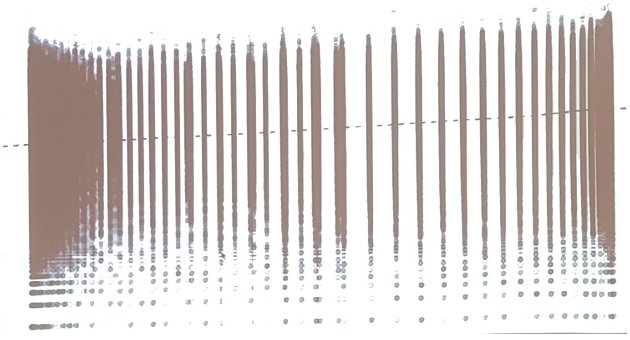

Figure 1: Complete Plot of Observations

Source: Class notes, Jake Hofman

Traditional methods of reporting statistics (by significance value) also overlook this variability, because while it is true that you have explained average changes by age, it is also true that you have *not* explained the rest of the variability in the model. In other words, we can explain *collective* trends, but not *individual* observations that we might see in the real world.

There might also be *unexplored transformations* to the data that might reduce this variability, or *other variables* such as internet connection that have not been recorded.

3 How Good is a Model?

There are two main metrics we can use to evaluate the fit of a model:

- Root mean squared error (RMSE)

$$\sqrt{\frac{1}{N}\sum_{i}^{N}(y_i - \hat{y})^2}$$

  - R syntax: `sqrt(mean((pred - actual)^2))`
  - We take the square root because we are calculating the squared difference between predicted and observed values.
  - To obtain a baseline MSE, we calculate the difference between the each observation and the mean of the observed data (i.e. the variance of y).
  - We then compare this to the MSE of the model with

$$\frac{MSE_{baseline} - MSE_{model}}{MSE_{baseline}} = R^2$$

    where R^2 is also the fraction of variance explained.

- Pearson's correlation coefficient

$$\frac{\sum_i (y_i - \bar{y})(\hat{y_i} - \hat{\bar{y}})}{[\sum_i (y_i - \bar{y})^2]^{\frac{1}{2}}[\sum_i (\hat{y_i} - \hat{\bar{y}})^2]^{\frac{1}{2}}}$$

  - R syntax: `cor(pred, actual)`
  - We calculate the deviations from the mean of observed and actual data, and then scale them accordingly by the denominator
  - The square of Pearson's correlation coefficient is also equal to to R^2 in the MSE, or the fraction of variance explained.

Applying the above tools to our previous model, we get a low R^2 of 0.019, which shows that our model in fact explains *very little* of the data. However, we need to remember that MSE is also highly susceptible to outliers, because a big difference between predicted and actual outcomes may skew the results.

4 Explaining the Past vs. Predicting the Future

As we add degrees to a polynomial, we add accuracy to the predictive power of a model because of the increased flexibility. The `poly(x, degree, raw=true)` function in R automatically adds successive polynomial transformation (until a user-defined k degree) of a variable to a linear model. We need to remember to remove the original variable, because this transformation will also give results for `k=1`.

However, while this succeeds in predicting the *past*, it does not necessarily have a bearing on how accurately we predict the *future*. Polynomial transformations also increase variability in our results, such that different observation sets might lead us to drastically different conclusions (see *Bias and Variance*).

Our models should thus be *complex* enough to explain the past, but *simple* enough to generalize the future. We can achieve this by randomly splitting our data into three sets: training set, validation set, and testing set.

1. Fit the models based on the *training set*
2. Use the *validation set* to evaluate their performance
3. Quote final performance on the *testing set*

If the model does not perform well on the testing set, we should not amend it using the same dataset, but wait for new data to come in. Unfortunately, this is often ignored in practice. A possible solution is a *reusable holdout set* which adds noise to the results of the testing set, rather than revealing the testing data directly.

5 Bias and Variance

As discussed earlier, polynomial models have *high variance* because different observation sets can lead to drastically different predictions. However, there is often a tradeoff between variance (predictiveness of model with *different* observations) and bias (predictiveness of model with *more* observations).

Loosely speaking, the relationship between variance and bias can be captured in the following equation, such that for a given MSE (and irreducible error), an increase in bias will result in a decrease in variance (or vice versa).

$$MSE = Bias^2 + Variance + IrreducibleError$$

While a linear model is *low variance*, it is likely to be *high bias* because there are no guarantees that the relationship is indeed linear. In other words, even with infinite observations, a model may still be wrong. By contrast, a n-degree polynomial model is likely to be *low bias*, because with infinite observations we are more likely to arrive (trivially) at an accurate representation of the world.

A *generalization curve* evaluates the complexity of the model based on its predictive power on the training sample and test sample. Using this, we can identify a point of divergence between the two lines to find a middle ground between bias and variance.

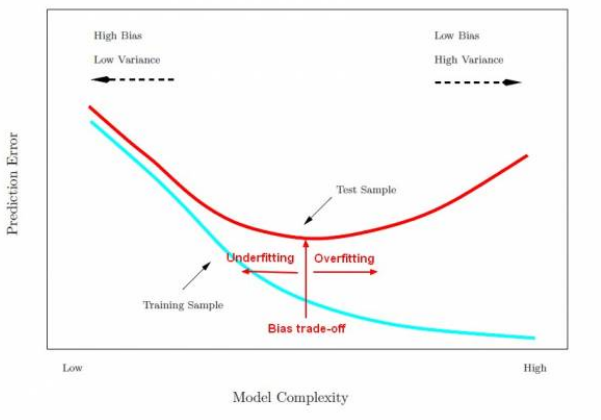

Figure 2: Generalization Curve for Training and Testing/Validation Errors

Source: http://gerardnico.com/wiki/_media/data_mining/model_complexity_error_training_test.jpg

6 Choosing the Training and Validation Sets

When shuffling the data, we can use the `sample()` function in R to select the indices of observations we want to include in either the training or test sets. However, we need to be careful of inherent orderings in the data, which might contain trends (e.g. seasonality) that would be incorrectly eliminated by random sampling.

We notice, however, that our results can vary significantly based on our assignment of the training and validation sets, which brings us to *K-fold cross validation*, which involves taking different subsets of training and validation data such that every observation is at one point part of the validation set. We can then average across all the runs to find the best possible model.

To process this in R, we simply create a variable called `num_folds` and call `sample(1:num_folds)` to split the folds evenly between the observations. We also keep track of both the average test error and standard error across all the folds, to help us evaluate the significance of any reduction in observed error.

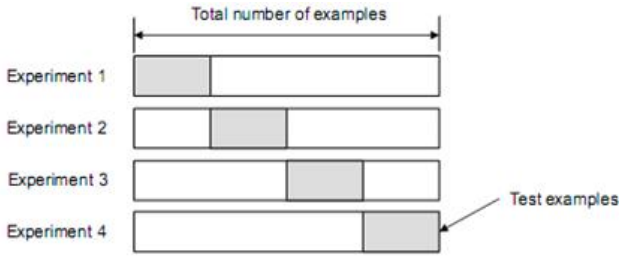

Figure 3: 4-Folds Cross Validation with Training and Testing/Validation Sets

Source: http://www.imtech.res.in/raghava/gpsr/Evaluation_Bioinformatics_Methods_files/image002.jpg

7 Regularization

Regularization involves penalizing complexity in our loss function for a specific choice to map it in the model. We can do so using either Ridge Regularization or Lasso Regularization.

- Ridge Regularization

$$\mathcal{L} = \frac{1}{n}\sum_{i}^{n}(y_i - wx_i)^2 + \lambda||w||^2$$

  As λ increases, the coefficients decrease which reduces the weighting of the model fit.

- Lasso Regularization

$$\mathcal{L} = \frac{1}{n}\sum_{i}^{n}(y_i - wx_i)^2 + ||w_1|| + ||w_2|| + ...$$

  This effectively performs variable selection by ignoring near-zero coefficients.