Lecture 8: Model complexity and generalization
Modeling Social Data, Spring 2019
Columbia University

Chaim Eisenbach

March 15, 2019

1 Introduction

Deciding how complex your model should be and how to evaluate it. Phrase a linear model as an optimization problem. Today we are discussing the Art of Modeling the claim is that for most social science problems you dont need fancy methods. When the features are really clear (age, income, etc.) linear models are usually good enough. If you understand everything you can do with a linear model then more complex models are just a step away.

2 notes

Check Lecture 7 notebook

A few things to keep in mind when exploring this data set: Looking at a histogram of page views on a log-scale there is no place to put a zero. The really really high page view counts are probably bots or something. If we did not have a log-scale we would have a very low resolution. Another key aspect of modeling is the readability of your plots in order to visualize the data you are manipulating. So splitting the data into different groups can help.

The users: (age, gender, daily views, median daily views) = 225,000 rows. The shaded area around the line, when using geom_smooth, is capturing the uncertainty. Now, instead of having ggplot2 do the modeling for us, let's do it ourselves.

```
model <- lm(log10(daily.views) - age, datamodel = data)
summary(model)
```

The standard error is if you got different samples of the data how much would the coefficient vary. $X \rightarrow \bar{X}_n \quad \sigma \rightarrow \sigma_{se} = \frac{\sigma}{\sqrt{n}}$

Bootstrap.

1. Sample from data you have
2. Estimate coefficient
3. look at standard deviation of distribution of estimates

So you do this a whole bunch of times and you'll build up some distribution where $\hat{w}$ is our estimate and we look at the width of the distribution so the standard deviation of this distribution is the standard error on the coefficient estimate. If you have a small sample the distribution will be wider than if you had a large sample of data. If you have enough data you'll get a really tight distribution and still a small standard error.

There are a bunch of tidyverse functions

```
tidy(model)
glance(model)


```

```
tidy(model) %>%
    ggplot(aes(x = term, ye =estimate) +
    geom_pointrange(aes(ymin = estimate - stderror ...)) +
    facet_wrap(~ term, scale = "free_y")
```

$\hat{y} = w_0 + w_1 x_1 + w_2 ...$
$\hat{y} = w_0 + w_1 age + w_2 age^2$ It's linear in the coefficients. But not necessarily the features themselves (the original features).

```
M <- model.matrix(log10(daily.views) ~ age + I(age^2), model_data)
## '+' means use this set of variables so 'I' forces a new feature which is age^2
head(M)
```

This is the actual set of numbers that go into our X matrix. ($\hat{W} = (X^T X)^{-1} X^T y$). So we are just offloading all this onto a linear algebra library, because we are lazy and don't really have time for that. When we go into a non-linear world our ability to interpret the tables falls apart, but it's useful for plots. Adding gender specific features means that the other gender is only explained by the unspecified features. While the machine can do all these things it's up to you to figure out what to put in. There's a lot of creativity when constructing the model.

Model evaluation

The more features the more crowded the plot and the harder it is to interpret or visualize.

```
ggplot(plot_data, aes(x = pred, y = geom_mean_daily_views, color = age)) +
  geom_point() +
  geom_abline(linetype = "dashed") +
  xlab('Predicted') +
  ylab('Actual') +
  facet_wrap(~ gender, scale = "free")
```

We don't have that many points because we are taking the averages.

But now we can see how terrible this model is

```
In [15]: pred_actual <- model_data %>%
           add_predictions(model) %>%
           mutate(actual = log10(daily.views))

         ggplot(pred_actual, aes(x = 10^pred, y = 10^actual)) +
           geom_point(alpha = 0.1) +
           geom_abline(linetype = "dashed") +
           scale_x_log10(label = comma, breaks = seq(0,100,by=10)) +
           scale_y_log10(label = comma) +
           xlab('Predicted') +
           ylab('Actual')
```

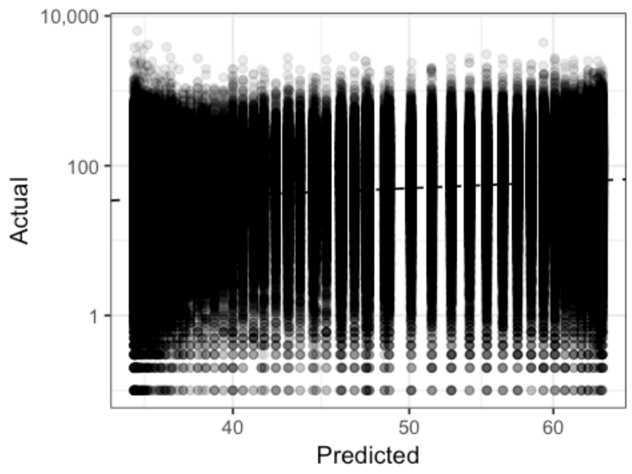


Figure 1: The variance is crazy and this model is good for nothing.

We can also quantify how well this does. MSE-model $= \frac{1}{n}\sum_{i-1}^{n}(y_i - \hat{y}_i)^2$, RMSE-model $= \sqrt{MSE_{model}}$, $(var(y))$ = MSE-baseline(mean) $= \frac{1}{n}\sum_{i=1}(y_i - \bar{y})^2$. $\frac{MSE-baseline-MSE-model}{MSE-baseline} = R^2$ which is the fraction of variance explained.

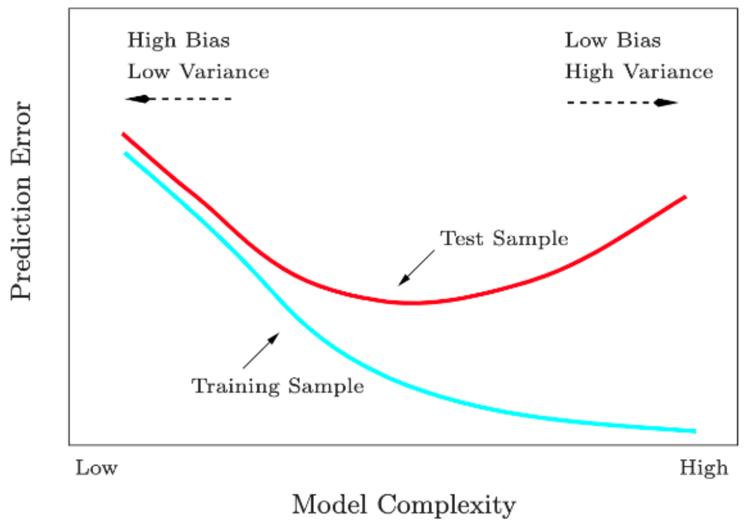
In R:

```
rmse(model, model_data)
rsquare(model, model_data)
```

Check Lecture 8

Bias-Variance tradeoff. A linear model is biased because it can only find linear trends. But it does not vary a lot with different samples of the data. High bias, low variance. As the model gets more complicated you can have a better fit. At some point there's a sweet spot with enough complexity and a good balance of bias-variance.

Bias-variance tradeoff



Simple models may be "wrong" (high bias), but fits don't vary a lot with different samples of training data (low variance)

Cross-validation. You can train it one one subset and test the model on another, never seen, subset of the data.

Complexity control

$\frac{1}{n}\sum_{i=1}^{n}(y_i - wx_i)^2 (which\, is\, the\, fit) + \lambda ||w||^2 (size\, of\, coefficients,\, tradeoff\, between\, fit\, and\, complexity)$

So we have our hyperparameter λ which is what we need to determine to find the best model. We can do this through various testing techniques. Into to glmnet what glmnet does is figure out the paremeters in our fit to the data above So even if you give it something crazy like a 10th degree polynomial it will return something normal.

"Weird R things with the dots, it's just terrible" - Jake Hofman