Data Mining
W4240 Sections 001, 003/004

Lauren A. Hannah

Columbia University, Department of Statistics

October 14, 2014

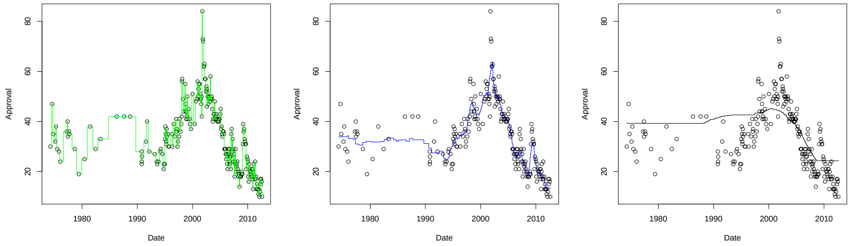
Some Estimators

Recall:

- kNN:

$$\hat{y} = \frac{1}{k} \sum_{i: x_i \in N_k(x)} y_i$$

- k controls the tradeoff between neighborhood size (bias) and estimator noise (variance)



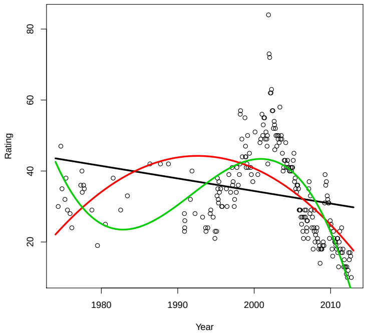
Some Estimators

Recall:

- Polynomial regression:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_m x^m$$

- Maximal degree m controls the tradeoff between estimator flexibility (bias) and estimator noise (variance)



Tunable Parameters

m and k are called **tunable parameters**

Their values affect how well a method works

- the “right” value should minimize the bias-variance tradeoff
- ...but the “right” value depends on the data

So how do we find the “right” value for a given dataset?

This problem is called **model selection**

Some More Estimators

Let's go back to kNN:

- suppose that I have chosen k and I now have an estimator
- I want to know how good this estimator is (i.e. what is the error on a new dataset?)

How would I estimate that?

This problem is called **model assessment**

Traditional Statistics

Ronald Aylmer Fisher (1890-1962)[1]

- 1. study problem, 2. propose model, 3. fit model, 4. check assumptions, 5. go back to 2. if assumptions not met
- tunable parameters changed to meet assumptions (e.g. Gaussian residuals)
- if assumptions met, theoretical properties describe model behavior on new datasets (e.g. confidence intervals)

[1] Photo credit: Wikipedia

Data Mining

Data Mining (Larry Page and Sergey Brin)[2]

- 1. look at data, 2. propose model, 3. select tunable parameters, 4. fit model, 5. assess model
- data most likely does not meet assumptions for parametric models
- usually care more about prediction than inference

[2]Photo credit: money.cnn.com

Generalization

Modeling for prediction:

1. get data
2. choose a model
3. fit the model
4. make predictions for new data

Generalization: making high quality predictions for new data

Expected Predictive Error

Tunable parameters α

Model with parameters α, $\hat{f}_\alpha(x)$

Goals for expected predictive error:

- **Model selection:** estimating the performance of different models in order to choose the best one (best α).
- **Model assessment:** having chosen a final model, estimating its prediction error (generalization error) on new data.

Expected Predictive Error

Training data: $\mathcal{T} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$

New data: X^0, Y^0

Generalization error:

$$\mathrm{Err}_{\mathcal{T}} = \mathbb{E}_{X^0, Y^0}[L(Y^0, \hat{f}(X^0)) \,|\, \mathcal{T}]$$

Here, $L(Y, f(X))$ is a **loss function**

- usually squared error for regression

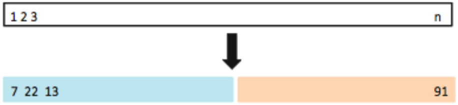
$$L(Y, f(X)) = (Y - f(X))^2$$

- usually 0/1 loss (misclassification rate, Hamming distance) for classification

$$L(Y, f(X)) = \mathbf{1}_{\{Y \neq f(X)\}}$$

Validation Sets

We could use a **validation set**[3]



- randomly divide the data into a training set and a validation set
- fit model on training set with differing values of α
- pick the best one for model
- fit model on entire dataset (depends on your level of philosophical purity...)

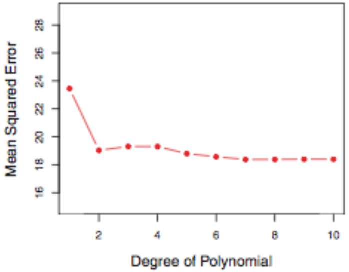
Doesn't use data twice!

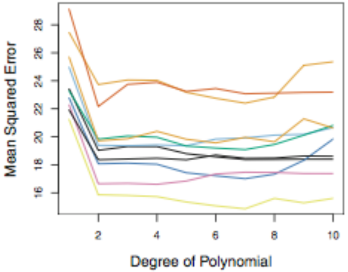
[3]Some images from *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani

Validation Sets

But there are two problems:

- estimates depend heavily on the validation set (high variance)





- estimate of error is probably *higher* than error for full model

Both of these problems get worse with small n

Cross-Validation

What if we use the data twice?

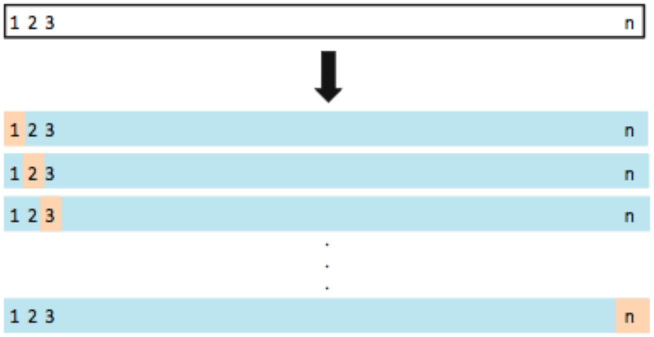
Problems with using the data twice:

- introduce *a lot* of bias if we fit model on (x_i, y_i) and then use $L(y_i, \hat{f}_\alpha(x_i))$ to estimate error
- introduce *a little* bias if we don't fit model on (x_i, y_i), use $L(y_i, \hat{f}_\alpha(x_i))$ to select α, and then fit $\hat{f}_\alpha$ on training *and* validation sets

Let's try:

- use one element as a validation set at the rest as a training set
- do this for all elements
- average results

Leave-One-Out-Cross-Validation



$$\mathrm{Err}_{\mathcal{T}} = \mathbb{E}_{X^0,Y^0}[L(Y^0, \hat{f}_\alpha(X^0)) \mid \mathcal{T}]$$
$$\approx \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}_\alpha^{(-i)}(x_i))$$

Leave-One-Out-Cross-Validation

Let's do this for kNN regression with the following data set:

x_1	x_2	y
1	1	2
-1	-1	-2
2	0	2
-3	2	1

Do this for $k = 1, 2, 3$.

Leave-One-Out-Cross-Validation

Model Selection:

1. For all values of α, estimate generalization error with LOOCV:

$$\mathrm{Err}_{\alpha}^{LOOCV} = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{f}_{\alpha}^{(-i)}(x_i))$$

2. Choose α with lowest $\mathrm{Err}_{\alpha}^{LOOCV}$ (if many are similar, choose one in the middle of the range)
3. Fit $\hat{f}_{\alpha^*}$ to $(x_1, y_1), \ldots, (x_n, y_n)$

Model Assessment:

1. For a fixed value of α, estimate generalization error with LOOCV:

$$\mathrm{Err}_{\alpha}^{LOOCV} = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{f}_{\alpha}^{(-i)}(x_i))$$

2. Can use $\{L((y_i, \hat{f}_{\alpha}^{(-i)}(x_i))\}_{i=1}^{n}$ to approximate distribution of predictive loss for given model

Generalized Cross-Validation

Fun fact: for *ordinary least squares linear regression* we actually only need to fit the model once

$$\begin{aligned}
\mathrm{Err}^{LOOCV} &= \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}^{(-i)}(x_i)\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}(x_i)}{1 - H_{ii}}\right)^2 \\
&\approx \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}(x_i)}{1 - \mathrm{trace}(\mathbf{H})/n}\right)^2
\end{aligned}$$

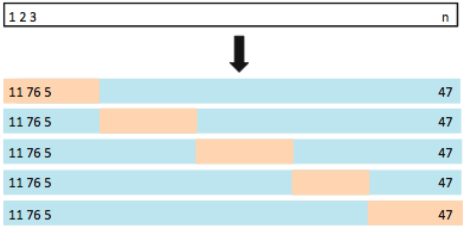
where $\hat{y} = \mathbf{H}y$, so $\mathbf{H} = X(X^TX)^{-1}X^T$. Note that $H_{ii} = h_i$, the leverage statistic.

Leave-One-Out-Cross-Validation

LOOCV sounds great! What could go wrong?

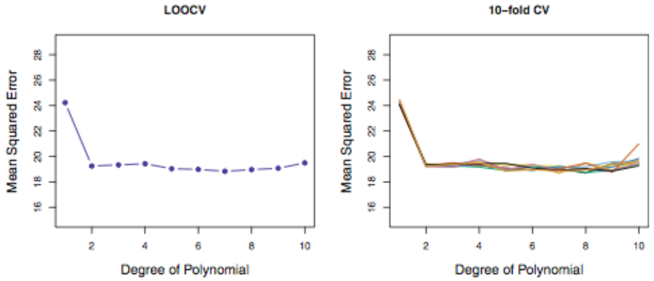
- suppose that I am training f on the entire Wikipedia corpus (> 3 million articles). I write efficient code, but it still takes about 12 hours to do one fit. Would LOOCV work? How could we fix it?
- suppose that we have *multiple tunable parameters*, i.e. $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m)$. How would LOOCV change? Would it work, and if not, how could we fix it?

K-Fold Cross-Validation



1. separate training set into K different, equally sized sets (folds)
2. for each tunable parameter value $\alpha = \alpha_1, \ldots, \alpha_M$:
    - for $k = 1, \ldots, K$:
        - use all of the data except fold k as a training set to fit the function with parameter α
        - use fold k as a testing set
        - estimate squared error on fold k
    - average errors to approximate expected predictive error
3. compare error values; pick parameter with lowest error

K-Fold Cross-Validation

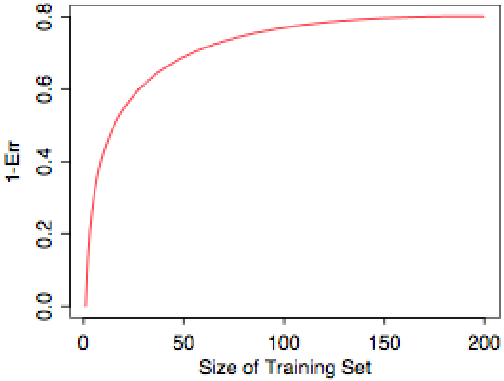


It turns out that K-fold CV is almost as good as LOOCV

K-Fold Cross-Validation

How big should K be?

- need to do computation K times
- ...but larger K is more accurate estimator



K-Fold Cross-Validation

Why is 10-fold CV so close to LOOCV?

Intuition: Central Limit Theorem

- n iid random variables, $\mathbb{E}[X_i] = \mu$, $\mathrm{Var}(X_i) = \sigma^2 < \infty$
- $\bar{\mu} = \frac{1}{n} \sum_{i=1}^{n} X_i$
- By CLT, $\bar{\mu} \sim N(\mu, \sigma^2/n)$
- So that means
$$|\bar{\mu} - \mu| \approx C \frac{1}{\sqrt{n}}$$

Intuition: difference between $n = 5$ and $n = 10$ is much bigger than $n = 495$ and $n = 500$.

- $n = 500$: Expected error for 5-fold CV is 0.05, for 10-fold CV is 0.047 and for LOOCV is 0.045
- $n = 5,000$: Expected error for 5-fold CV is 0.016, for 10-fold CV is 0.015 and for LOOCV is 0.014

Data Preprocessing

Common data problems:

- missing data: not all data has all values
- high dimensional: p is too large to effectively use most methods

To deal with these problems, we often do **preprocessing**

- missing data: remove or impute values
- high dimensional: reduce dimensionality

Data Preprocessing

Ways to reduce dimensionality:

- select a set of covariates that are "highly predictive"
  - highly correlated with response
  - have large marginal information gains/variance reductions
- make some combination(s) of the covariates (like $0.7X_1 - 3.8X_2$ or PCA loadings) that is "highly predictive," select components based on correlation with Y

Finding a set of predictors before fitting an estimator is called **screening**.

The simplest method is selecting the K covariates that are the most correlated with Y.

Cross-Validation

Consider a problem with many predictors (ex: microarray data). A possible strategy:

1. Screen the predictors to find a "good" subset (e.g. choose the best subset based on data)
2. Using this set of predictors, build a multivariate classifier
3. Use cross-validation to select tunable parameters and estimate model error

Is this the right way to use cross-validation?

Cross-Validation

Consider this classification problem:

- $n = 50$, $p = 5000$
- $X \sim N_{5000}(0, I)$, $Y \sim Bernoulli(0.5)$
- true error rate: 50% (labels independent from covariates)

Let's screen to select 100 most predictive covariates and then use 1-nn prediction.

What happens?

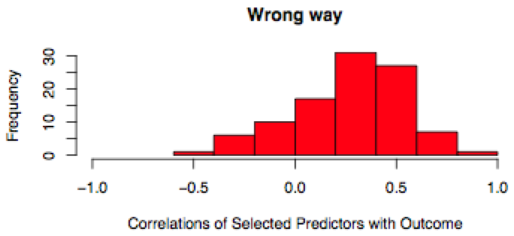
Cross-Validation

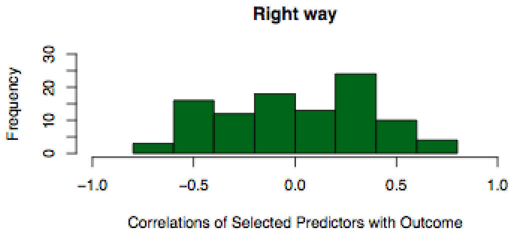
Right way to do cross-validation:

1. Divide samples into K cross-validation folds *at random*
2. For each fold $k = 1, \ldots, K$
    a. Reserve fold k for test, use other folds for training
    b. Find a subset of “good” predictors from training
    c. Using this subset of predictors, build a multivariate classifier on training set
    d. Use the classifier to predict labels on fold k

Your screening method is a part of your model!

Cross-Validation


Wrong way
Frequency
Correlations of Selected Predictors with Outcome


Right way
Frequency
Correlations of Selected Predictors with Outcome