MS&E 226: Fundamentals of Data Science

Lecture 3: Train-validate-test

Ramesh Johari
rjohari@stanford.edu

The prediction problem: Formalism

- $\mathbf{X}, \mathbf{Y}$: Data we are given
- $\vec{X}$: Covariate vector of a *new* data point from the population
- Y: True outcome associated with $\vec{X}$
- $\hat{f}(\cdot)$: Fitted model (input: covariate vector; output: predicted outcome)

Goal: Using $\mathbf{X}$ and $\mathbf{Y}$, construct $\hat{f}$ so that $Y \approx \hat{f}(\vec{X})$.

A "good" fitted model

A "good" fitted model should minimize the expected prediction error on *new* data, also called *generalization* error:

$$\mathbb{E}_{\vec{X},Y}[(Y - \hat{f}(\vec{X}))^2 | \hat{f}].$$

Note that in this definition we condition on the fitted model $\hat{f}$.

The only *randomness* is in the new data point $\vec{X}$ and Y drawn uniformly at random from the population.[1]

[1] There are other forms of generalization error; e.g., you might also assume the new $\vec{X}$ is also known. We will return to this later.

Out-of-sample error

Prediction fundamentally asks whether a model performs well on data that was *not used for fitting the model*.

Out-of-sample error

Prediction fundamentally asks whether a model performs well on data that was *not used for fitting the model*.

This means any assessment of the prediction error of a fitted model relies on data that is *out-of-sample*, i.e., not part of the sample used for fitting (unlike R^2!).

Out-of-sample error

Prediction fundamentally asks whether a model performs well on data that was *not used for fitting the model*.

This means any assessment of the prediction error of a fitted model relies on data that is *out-of-sample*, i.e., not part of the sample used for fitting (unlike R^2!).

This insight is the core of nearly all methodology for fitting, selecting, and evaluating predictive models.

Train-validate-test

Train-validate-test

The most common workflow for prediction is:

1. Separate your data into three groups: *training*, *validation*, *testing*.
2. *Training*: "Fitting models"
3. *Validation*: "Selecting a winning fitted model"
4. *Testing*: "Evaluating the winner"

Note separation of training sample from validation and testing ("out-of-sample").

Commonly, only the first two stages are used;
in such cases, "validation" is often called "testing" (confusing!).

Training

Training involves fitting models using only the training data.

Example: You might fit a range of linear models, using different sets of covariates, higher order terms, interaction terms, transformed or engineered variables, etc.

Validation

The validation step estimates the generalization error of the different models, and chooses the best one.
Formally:

- Suppose samples $(\tilde{\mathbf{X}}_1, \tilde{Y}_1), \ldots, (\tilde{\mathbf{X}}_k, \tilde{Y}_k)$ in the validation set.
- For each fitted model $\hat{f}$, estimate the prediction error as follows:

$$\frac{1}{k} \sum_{i=1}^{k} (\tilde{Y}_i - \hat{f}(\tilde{\mathbf{X}}_i))^2. \tag{1}$$

  This is the *mean squared error* (MSE) on the validation set.
- Choose the model with the smallest MSE on the validation set (the "winner").

Validation

Why does validation ensure we select "good" predictive models?

Intuitively:

As the size of the validation set grows, the validation step *picks a fitted model that has the lowest average prediction error over the population*.

Validation

Why does validation ensure we select "good" predictive models?

Formally:

By the law of large numbers, as $k \to \infty$,

$$\frac{1}{k}\sum_{i=1}^{k}(\tilde{Y}_i - \hat{f}(\tilde{\mathbf{X}}_i))^2 \to$$

$$\mathbb{E}_{\vec{X},Y}[(Y - \hat{f}(\vec{X}))^2 | \hat{f}] \quad \text{(generalization error)}$$

Testing

Suppose that samples $(\tilde{\mathbf{X}}_{k+1}, \tilde{Y}_{k+1}), \ldots, (\tilde{\mathbf{X}}_{\ell}, \tilde{Y}_{\ell})$ are in the test set.

We compute the *test MSE* as:

$$\frac{1}{\ell - k} \sum_{i=k+1}^{\ell} (\tilde{Y}_i - \hat{f}^*(\tilde{\mathbf{X}}_i))^2.$$

Here $\hat{f}^*$ is the winning model.

Testing

If validation already picks a winner, why do we need testing?

Because: In practice, with finite validation data, the error of the winning model is typically an *underestimate* of its true prediction error! (You will investigate why on the problem set.)

The testing stage gives an *unbiased* (i.e., accurate) estimate of the true prediction error of the winning model. (Not all prediction problems require a testing stage.)

Example: Model selection, validation, and testing

For this example, we generate 300 X_1, X_2 as i.i.d. $N(0, 1)$ random variables.

We then generate 300 Y random variables as:

$$Y_i = 1 + 2X_{i1} + 3X_{i2} + \epsilon_i,$$

where ϵ_i are i.i.d. $N(0, 5)$ random variables.
The training, validation, and test separation is 100/100/100 samples, respectively.

Example: Model selection, validation, and testing

We trained the following five models, then ran them through the validation and test set.

For each we computed the square root of the mean squared prediction error (RMSE).[2]

Model	Training	Validation	Test
`Y ~ 1 + X1`	5.37	5.58	
`Y ~ 1 + X2`	4.87	4.95	
`Y ~ 1 + X1 + X2`	4.44	4.67	
`Y ~ 1 + X1 + X2 + I(X1^2) + I(X2^2)`	4.39	4.64	5.80
`Y ~ 1 + X1 + X2 + I(X1^2) + I(X2^2) + . . . I(X1^5) + I(X2^5)`	4.29	4.75	

Example: Model selection, validation, and testing

Uncovering the other entries:

Model	Training	Validation	Test
`Y ~ 1 + X1`	5.37	5.58	6.64
`Y ~ 1 + X2`	4.87	4.95	6.06
`Y ~ 1 + X1 + X2`	4.44	4.67	5.76
`Y ~ 1 + X1 + X2 + I(X1^2) + I(X2^2)`	4.39	4.64	5.80
`Y ~ 1 + X1 + X2 + I(X1^2) + I(X2^2) + . . . I(X1^5) + I(X2^5)`	4.29	4.75	5.91

Regularization

An abundance of features...

Lets return to the problem we discussed at the end of last lecture:

We could keep adding features and rerun OLS (and R^2 would keep increasing as a result). But this risks being a model that generalizes increasingly poorly (as in the last example).

What can we do?

Regularization

One way to achieve good generalization is to *regularize* the objective function:

Add a *penalty* for additional model complexity.

Important note: Always standardize variables prior to running regularized regression methods.

Regularization: Ridge regression

Instead of minimizing SSE, minimize:

$$\text{SSE} + \lambda \sum_{j=1}^{p} |\hat{\beta}_j|^2.$$

where $\lambda > 0$. This is called *ridge regression*.
In practice, the consequence is that it penalizes $\hat{\boldsymbol{\beta}}$ vectors with "large" norms.

Regularization: Lasso

Instead of minimizing SSE, minimize:

$$\text{SSE} + \lambda \sum_{j=1}^{p} |\hat{\beta}_j|$$

where $\lambda > 0$.

This is called the *Lasso*.

In practice, the resulting coefficient vector will be "sparser" (i.e., have fewer nonzero entries) than the unregularized coefficient vector.

Regularization

Both lasso and ridge regression are "shrinkage" methods for covariate selection:

- Relative to OLS, both lasso and ridge regression will yield coefficients $\hat{\boldsymbol{\beta}}$ that have "shrunk" towards zero.
- The covariates that are most "explanatory" of variation in the outcome are the ones that will be retained.
- Lasso typically yields a much smaller subset of nonzero coefficients than ridge regression or OLS (i.e., fewer nonzero entries in $\hat{\boldsymbol{\beta}}$).

An example with regularization: Baseball hitters

Data taken from *An Introduction to Statistical Learning*.

Consists of statistics and salaries for 263 Major League Baseball players.

We use this dataset to:

- Develop the train-test method
- Apply lasso and ridge regression
- Compare and interpret the results

We'll use the `glmnet` package for this example.

Loading the data

`glmnet` uses matrices rather than data frames for model building:

```
> library(ISLR)
> library(glmnet)

> data(Hitters)
> hitters.df = subset(na.omit(Hitters))

> X = model.matrix(Salary ~ 0 + ., hitters.df)
> Y = hitters.df$Salary
```

Standardization

When running lasso and ridge regression, we *standardize* the data:

```
> X = scale(X)
> Y = scale(Y)
```

Note: By default, `glmnet` will internally standardize the covariates, and then return coefficients in the original scale (undoing this standardization). We standardize explicitly in advance here so that we can compare coefficients across covariates.

Training vs. test set

Here is a simple way to construct training and test sets from the single dataset:

```
train.ind = sample(nrow(X), round(nrow(X)/2))
X.train = X[train.ind,]
X.test = X[-train.ind,]
Y.train = Y[train.ind]
Y.test = Y[-train.ind]
```

Ridge and lasso

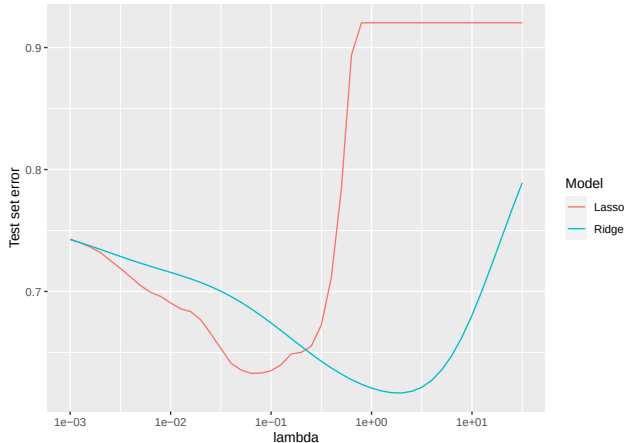
Building a lasso model:

```
> lambdas = 10^seq(-3,3.4,0.1)
> fm.lasso = glmnet(X.train,
     Y.train, alpha = 1,
     lambda = lambdas, thresh = 1e-12)
```

Setting `alpha = 0` gives ridge regression.
Make predictions as follows at $\lambda =$ `lam`:

```
> mean( (Y.test -
    predict(fm.lasso, s = lam, newx = X.test))^2 )
```

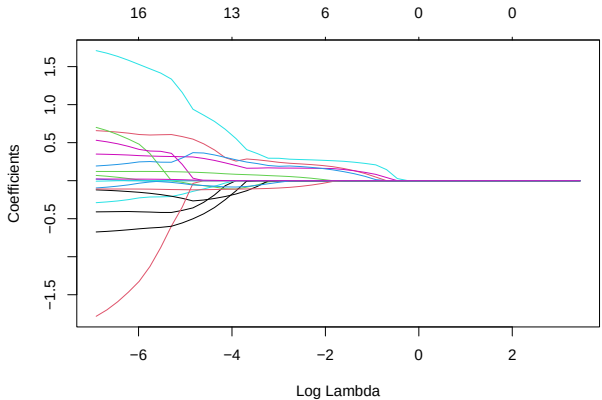
Results



What is happening to lasso?

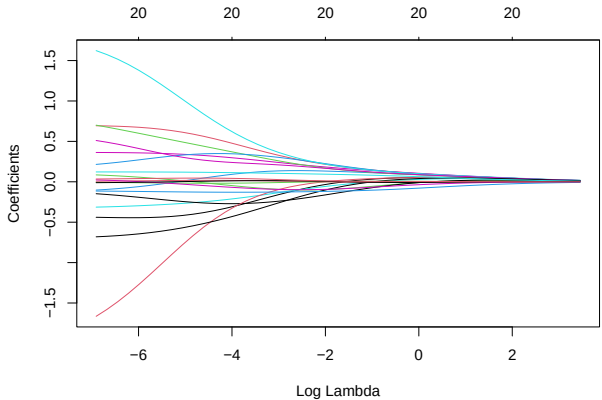
Lasso coefficients

Using `plot(fm.lasso, xvar="lambda")`:



Ridge coefficients

Using `plot(fm.lasso, xvar="lambda")`:



Learning algorithms vs. fitted models

Learning algorithms vs. fitted models

In machine learning, a (supervised) *learning algorithm* is a function or algorithm that takes as input a training data set $\mathbf{X}, \mathbf{Y}$, and produces as output a fitted model $\hat{f}$.

Think of this as code that produces, e.g., a fitted linear regression, lasso regression, ridge regression, etc.

The *fitted model* is the *specific* realization of coefficients or parameters, which we've abbreviated $\hat{f}$.

Example: The R call `fm = lm(data = input, formula = Y ~ 1 + X1 + X2)` takes as input the data frame `input`, and produces as output a fitted model `fm` (i.e., the linear regression coefficients), using the two covariates `X1` and `X2` to predict the outcome.

A plethora of learning algorithms

There are many, many learning algorithms out there! We'll touch on some of these, but not all:

- Linear methods: Linear and logistic regression; regularized regression; quantile regression; generalized linear models
- Tree methods: Decision trees; random forests; gradient boosted trees; XGBoost
- Neural networks: Feed-forward neural networks; convolutional neural networks; recurrent neural networks
- Instance-based methods: k-nearest neighbors; locally weighted regression
- Generative methods: Transformers; diffusion models; variational autoencoders

As supervised learning algorithms, *all* can be put through the train-validate-test framework to obtain high performance fitted models.