Lecture 5: Evaluation and Training

STATS 202: Data Mining and Analysis

Linh Tran

tranlm@stanford.edu

Department of Statistics
Stanford University

July 10, 2023

Announcements

- Python version of textbook is released!
- HW1 hopefully graded by Friday.
  - Some students used old versions. Please use Ed 2.
  - Feel free to use office hours & Piazza.
- HW2 due in 1 week.
- Midterm is in 9 days
  - Requests for accommodations handled through Piazza
- Passing is C- (Credit/No credit) or D- (Letter Grade)

Outline

- Evaluating classification models
  - Confusion matrix
  - Receiver Operating Characteristic curve
- Validation sets
  - Data splitting
  - Leave one out cross-validation
  - K-fold cross-validation
  - Cross-validation theory
  - Ensembles

Recap

Previously, we:

- Defined Multiple Linear Regression
- Generalized to Logistic Regression
- Covered potential issues with linear models
- Covered another linear model (LDA - as well as QDA)

Linear Discriminant Analysis vs Logistic regression

Assume a two-class setting with one predictor

Linear Discriminant Analysis:

$$\log\left[\frac{p_1(x)}{1-p_1(x)}\right] = c_0 + c_1 x \tag{1}$$

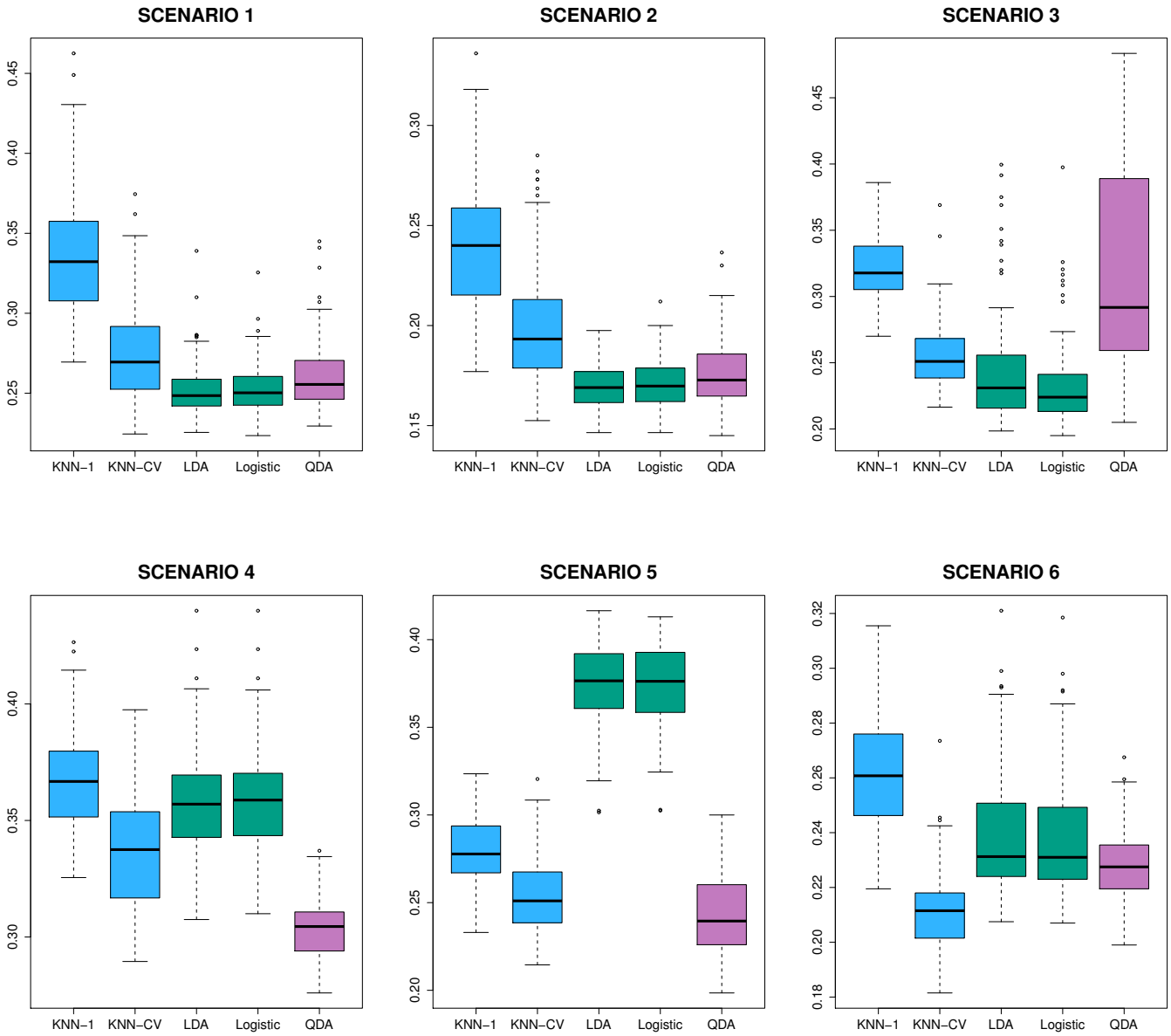
- c_0 and c_1 computed using $\hat{\mu}_0$, $\hat{\mu}_1$, and $\hat{\sigma}^2$

Logistic regression:

$$\log\left[\frac{\mathbb{P}[Y=1|x]}{1-\mathbb{P}[Y=1|x]}\right] = \beta_0 + \beta_1 x \tag{2}$$

- β_0 and β_1 estimated using MLE

Comparison of classification methods



Evaluating a model

Recall: Our standard prediction error functions are

- Classification: Cross-entropy

$$\hat{CE}(\hat{f}_n) \quad = \quad \mathbb{E}_n[-y \log \hat{f}_n(x)] \tag{3}$$

- Regression: Mean squared error

$$\hat{MSE}(\hat{f}_n) = \mathbb{E}_n[y - \hat{f}_n(x)]^2 \tag{4}$$

While MSE has an intuitive interpretation, CE is harder to explain.

- Typically, other losses are used to evaluate classification methods

Evaluating a classification method

Many practitioners use the 0-1 loss:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}[y_i \neq \hat{y}_i] \tag{5}$$

- Can be thought of as $1-$ accuracy

Evaluating a classification method

Many practitioners use the 0-1 loss:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}[y_i \neq \hat{y}_i] \tag{5}$$

- Can be thought of as 1− accuracy
- Possible to make the wrong prediction for some classes more often than others
  - The 0-1 loss doesn't tell you anything about this
- A much more informative error summary is a *confusion matrix*

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

Evaluating a classification method

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

We can calculate a number of statistics from this table, e.g.

- True positive rate (aka Sensitivity, aka Recall)

$$\mathbb{P}[\mathit{Predicted}\ + \,|\, \mathit{True}\ +], \text{i.e. } \mathit{TP}/P \tag{6}$$

Evaluating a classification method

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

We can calculate a number of statistics from this table, e.g.

- True positive rate (aka Sensitivity, aka Recall)

$$\mathbb{P}[\mathit{Predicted}\ + \,|\, \mathit{True}\ +], \text{i.e. } \mathit{TP}/P \tag{6}$$

- True negative rate (aka Specificity)

$$\mathbb{P}[\mathit{Predicted}\ - \,|\, \mathit{True}\ -], \text{i.e. } \mathit{TN}/N \tag{7}$$

Evaluating a classification method

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

We can calculate a number of statistics from this table, e.g.

- True positive rate (aka Sensitivity, aka Recall)

$$\mathbb{P}[\mathit{Predicted} + | \mathit{True} +], \text{i.e. } \mathit{TP}/P \tag{6}$$

- True negative rate (aka Specificity)

$$\mathbb{P}[\mathit{Predicted} - | \mathit{True} -], \text{i.e. } \mathit{TN}/N \tag{7}$$

- Positive predicted value (aka Precision)

$$\mathbb{P}[\mathit{True} + | \mathit{Predicted} +], \text{i.e. } \mathit{TP}/P^* \tag{8}$$

Evaluating a classification method

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

We can calculate a number of statistics from this table, e.g.

- True positive rate (aka Sensitivity, aka Recall)

$$\mathbb{P}[\mathit{Predicted}\ + \,|\, \mathit{True}\ +], \text{i.e. } \mathit{TP}/P \tag{6}$$

- True negative rate (aka Specificity)

$$\mathbb{P}[\mathit{Predicted}\ - \,|\, \mathit{True}\ -], \text{i.e. } \mathit{TN}/N \tag{7}$$

- Positive predicted value (aka Precision)

$$\mathbb{P}[\mathit{True}\ + \,|\, \mathit{Predicted}\ +], \text{i.e. } \mathit{TP}/P^* \tag{8}$$

- Negative predicted value

$$\mathbb{P}[\mathit{True}\ - \,|\, \mathit{Predicted}\ -], \text{i.e. } \mathit{TN}/N^* \tag{9}$$

Evaluating a classification method

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

We can also calculate summary statistics, e.g. the $F1$-score

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{10}$$

Example: Predicting default

Predicting credit card default in a dataset of 10K people

- Predicted "yes" if $\hat{\mathbb{P}}_n[default = yes|\mathbf{X}] > 0.5$

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

Example: Predicting default

Predicting credit card default in a dataset of 10K people

- Predicted "yes" if $\hat{\mathbb{P}}_n[default = yes|\mathbf{X}] > 0.5$

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

- The error rate among people who do not default is very low (i.e. high specificity)

Example: Predicting default

Predicting credit card default in a dataset of 10K people

- Predicted "yes" if $\hat{\mathbb{P}}_n[default = yes|\mathbf{X}] > 0.5$

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

- The error rate among people who do not default is very low (i.e. high specificity)
- The error rate among people who default (false negative rate) is high at 76% (i.e. low sensitivity)
  - i.e. FN/P

Example: Predicting default

Predicting credit card default in a dataset of 10K people

- Predicted "yes" if $\hat{\mathbb{P}}_n[default = yes|\mathbf{X}] > 0.5$

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

- The error rate among people who do not default is very low (i.e. high specificity)
- The error rate among people who default (false negative rate) is high at 76% (i.e. low sensitivity)
  - i.e. FN/P
- It's likely that false negatives are a bigger source of concern

Example: Predicting default

Predicting credit card default in a dataset of 10K people

- Predicted "yes" if $\hat{\mathbb{P}}_n[default = yes|\mathbf{X}] > 0.5$

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

- The error rate among people who do not default is very low (i.e. high specificity)
- The error rate among people who default (false negative rate) is high at 76% (i.e. low sensitivity)
  - i.e. FN/P
- It's likely that false negatives are a bigger source of concern
- **Possible solution**: Change the classifier *threshold*

Example: Predicting default

Predicting credit card default in a dataset of 10K people

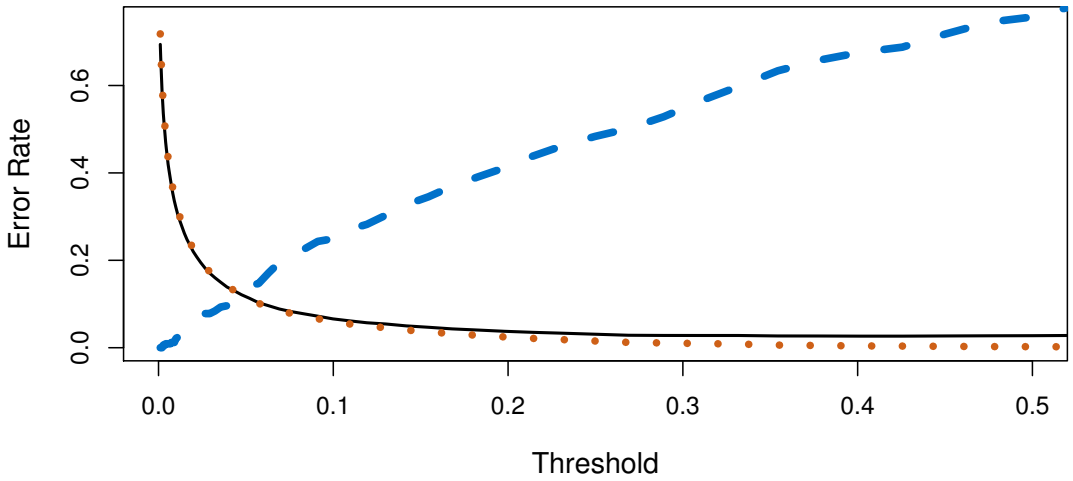
- Predicted "yes" if $\hat{\mathbb{P}}_n[default = yes|\mathbf{X}] > 0.2$

		True default status		
		No	Yes	Total
Predicted default status	No	9,432	138	9,570
	Yes	235	195	430
	Total	9,667	333	10,000

- The false negative rate is now 41%
- The false positive rate has increased (from $< 1\%$ to 2%)
  - So, we're paying a price for reducing the false negative rate (i.e. there's a trade-off)

Example: Predicting default

Viewing the trade-off over different thresholds

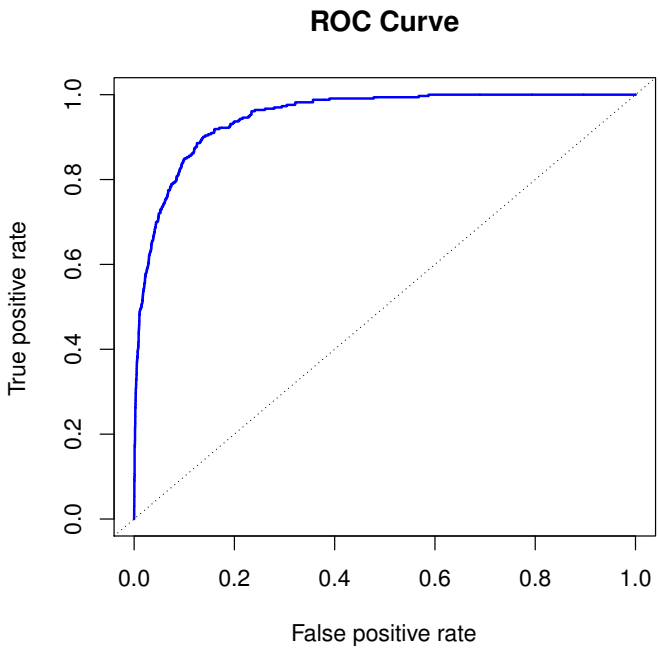


- – – – False negative rate (error for defaulting customers)
- . . . . . False positive rate (error for non-defaulting customers)
- —— 0-1 loss or total error rate

Example: Predicting default

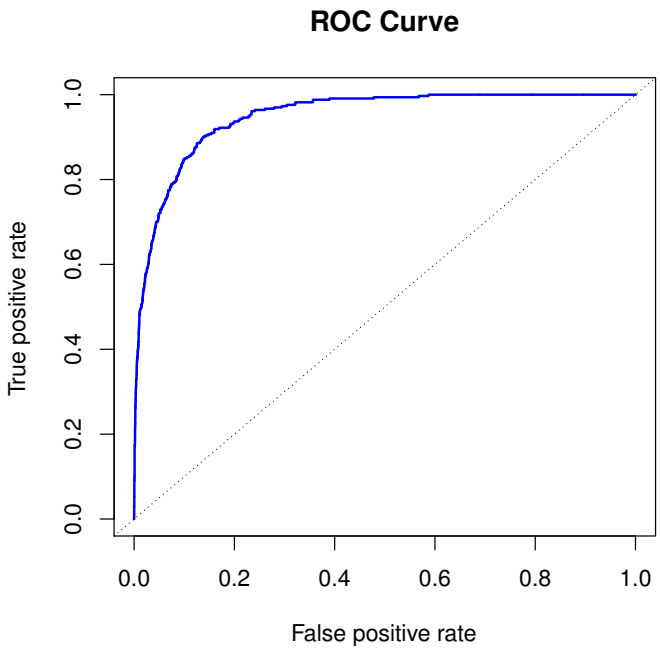
The *Receiver Operating Characteristic (ROC)* curve:

- Displays the performance for every threshold choice.



Example: Predicting default

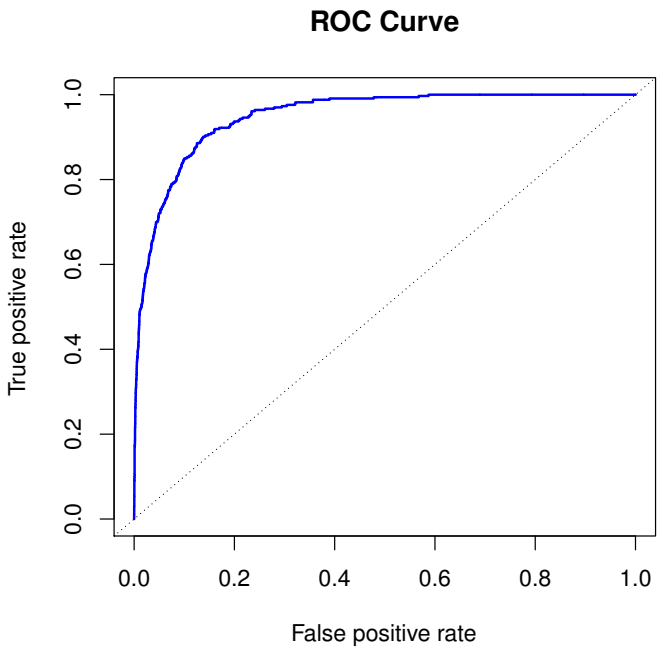
The *Receiver Operating Characteristic (ROC)* curve:



- Displays the performance for every threshold choice.
- The *Area Under the Curve (AUC)* summarizes the classifier performance, e.g.
  - The closer AUC is to 1, the better the performance.

Example: Predicting default

The *Receiver Operating Characteristic (ROC)* curve:



- Displays the performance for every threshold choice.
- The *Area Under the Curve (AUC)* summarizes the classifier performance, e.g.
  - The closer AUC is to 1, the better the performance.
  - $AUC = 0.5$ is equivalent to a random classifier.
    - The minimum value (otherwise, you'd just flip the class predictions).
  - *AUC* is equivalent to the Mann–Whitney U test.

Loss functions

- In general, we want to optimize for loss functions that are specific to our (applied) problem
  - e.g. When predicting credit default, we'll likely care more about the precision or recall
  - i.e. Natural loss function is not the cross-entropy or 0-1 loss
- Even if we use one method which minimizes a certain kind of training error, we can tune it to optimize our true loss function
  - e.g. Find the threshold that brings the recall above an acceptable level

Validation

Recall: Training our model on a dataset and evaluating on the same dataset leads to (overly) optimistic results

- We care about the error on P_0, not P_n

Validation

Recall: Training our model on a dataset and evaluating on the same dataset leads to (overly) optimistic results

- We care about the error on P_0, not P_n

Additionally: We have *hyper-parameters* that we have to tune, e.g.

- The number of neighbors k in k-nearest neighbors
- The number of variables in forward/backward step selection
- The order of a polynomial in polynomial regression

Validation sets

Typically: Datasets are divided into three parts

1. *Training set*: data you use to train your model parameters
2. *Development set*: data you use to tune your model hyper-parameters
3. *Test set*: data you use to evaluate your model after it's been trained

Keeping separate development/test sets prevents the model from over-fitting (providing overly optimistic prediction errors)

Validation sets

Typically: Datasets are divided into three parts

1. *Training set*: data you use to train your model parameters
2. *Development set*: data you use to tune your model hyper-parameters
3. *Test set*: data you use to evaluate your model after it's been trained

Keeping separate development/test sets prevents the model from over-fitting (providing overly optimistic prediction errors)

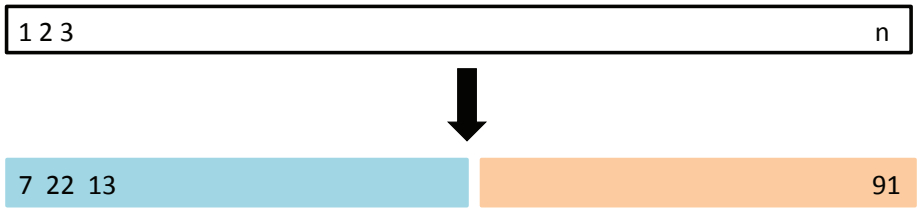
n.b. The terminology will vary across fields, e.g.

- Sometimes, people refer to the "*development*" set as the "*validation*" set
- The course textbook refers to the "*test*" set as the "*validation*" set

Validation sets

In practice, may not need a development set. Consequently, you'll

1. Split the data into two parts (i.e. a train and test set)
2. Train on the first part
3. Compute the error on the second part

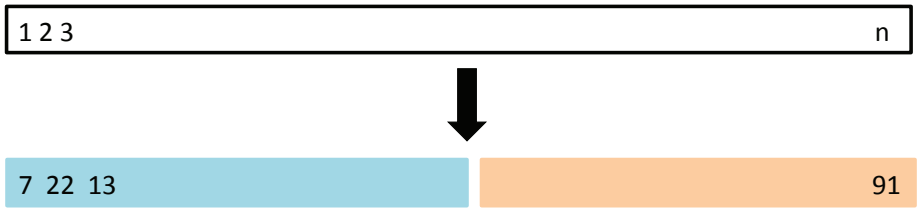


Visualization of data splitting

Validation sets

In practice, may not need a development set. Consequently, you'll

1. Split the data into two parts (i.e. a train and test set)
2. Train on the first part
3. Compute the error on the second part



Visualization of data splitting

n.b. You'll want to split the data randomly

- Avoids possible correlation (e.g. between households)

Validation sets

Comparing scenarios:
Assume $n = 1000$, and consider the three splits

1. $n_{train} = 500$ and $n_{test} = 500$
2. $n_{train} = 50$ and $n_{test} = 950$
3. $n_{train} = 950$ and $n_{test} = 50$

What happens to the model fit and prediction errors?

Validation sets

Comparing scenarios:
Assume $n = 1000$, and consider the three splits

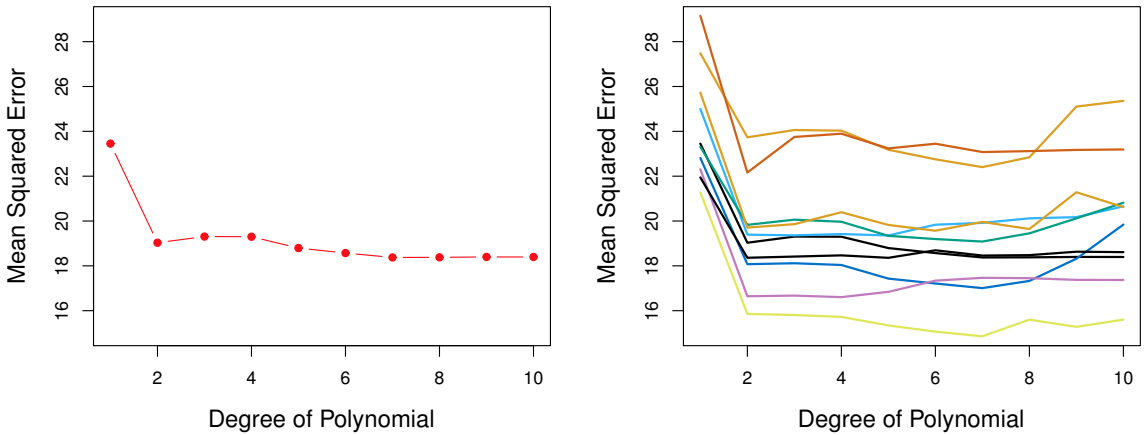
1. $n_{train} = 500$ and $n_{test} = 500$
2. $n_{train} = 50$ and $n_{test} = 950$
3. $n_{train} = 950$ and $n_{test} = 50$

What happens to the model fit and prediction errors?

- Scenario 2 has high model variance, but lower variance in estimates of prediction error
- Scenario 3 has low model variance, but higher variance in estimates of prediction error
- Scenario 1 provides a trade-off between the two

Validation sets

Example: Polynomial regression to estimate mpg from horsepower in the Auto data



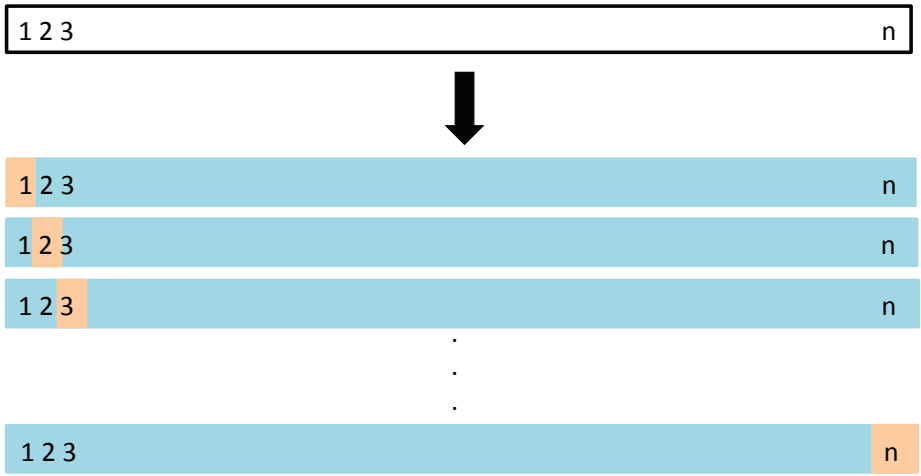
MSE under different samples for the test split

Problem: Every split yields a different estimate of the error

Leave one out cross-validation (LOOCV)

Allows us to use every observation as the test split

1. For $i = 1, 2, ..., n$:
   - Train the model on every point except i
   - Compute the test error on point i
2. Average the test errors



Leave one out cross-validation (LOOCV)

Allows us to use every observation as the test split

1. For $i = 1, 2, ..., n$:
   - Train the model on every point except i
   - Compute the test error on point i
2. Average the test errors

For *regression*:

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i^{(-i)})^2 \tag{11}$$

For *classification*:

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}[y_i \neq \hat{y}_i^{(-i)}] \tag{12}$$

Leave one out cross-validation (LOOCV)

Computing $CV_{(n)}$ can be computationally expensive, since it involves fitting the model n times.

For linear regression, there is a shortcut:

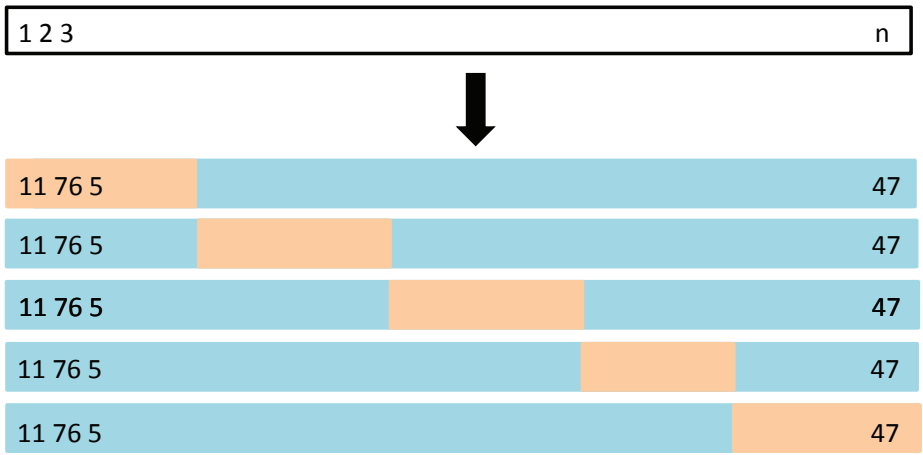
$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - h_{ii}}\right)^2 \tag{13}$$

where h_{ii} is the leverage statistic (Chapter 3).

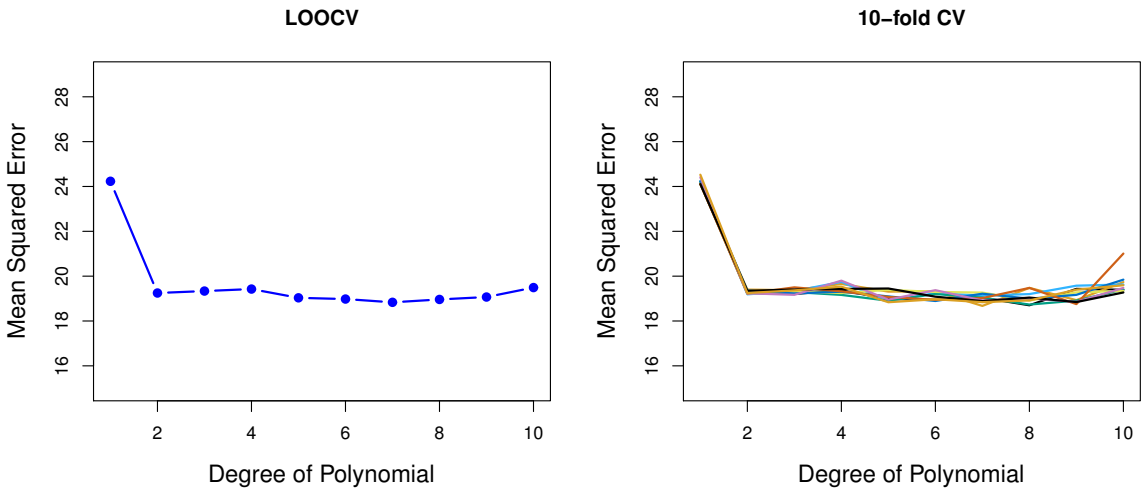
k-fold cross validation

Rather than LOOCV, we can just divide the data into k splits (aka folds)

1. Divide the data into k splits (aka folds)
2. For $i = 1, ..., k$:
    a. Train your model on all the data excluding the i^{th} fold
    b. Compute the prediction error on the i^{th} fold
3. Average the errors over the k splits



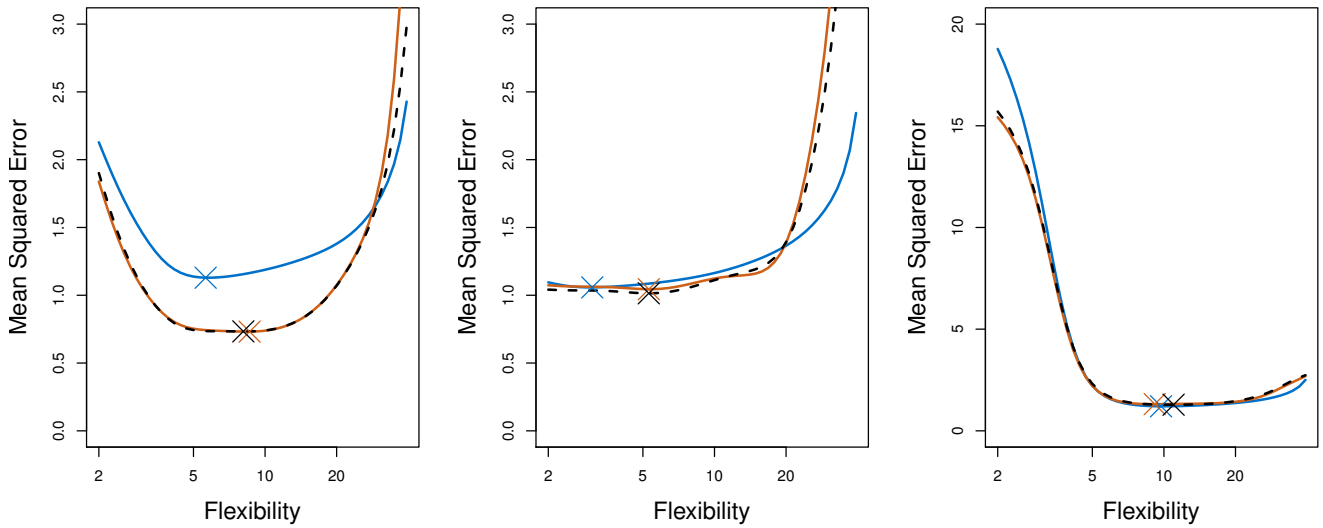
LOO vs k-fold cross validation



- k-fold CV depends on the chosen split
- In k-fold CV, we train the model on less data than what is available.
  - Introduces bias into estimates of the test error
- In LOOCV, the fitted models are very correlated
  - Increases variance of the estimates of the test error

LOO vs k-fold cross validation

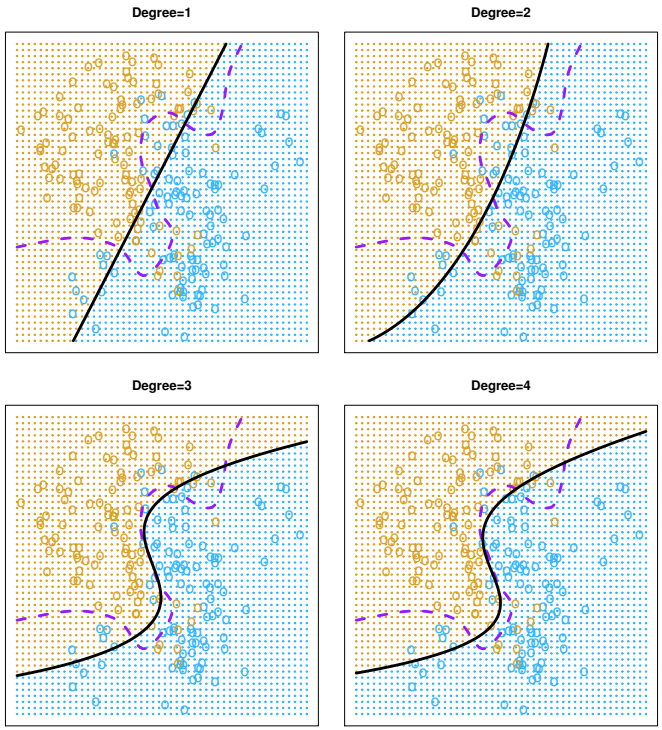
Choosing an optimal model



- Despite the bias-variance tradeoff, the error estimates still (generally) tend to be pretty similar
- Choosing the model with the minimum cross validation error often leads to the method with minimum test error

Choosing an optimal model

In classification, we can take the same approach, e.g.

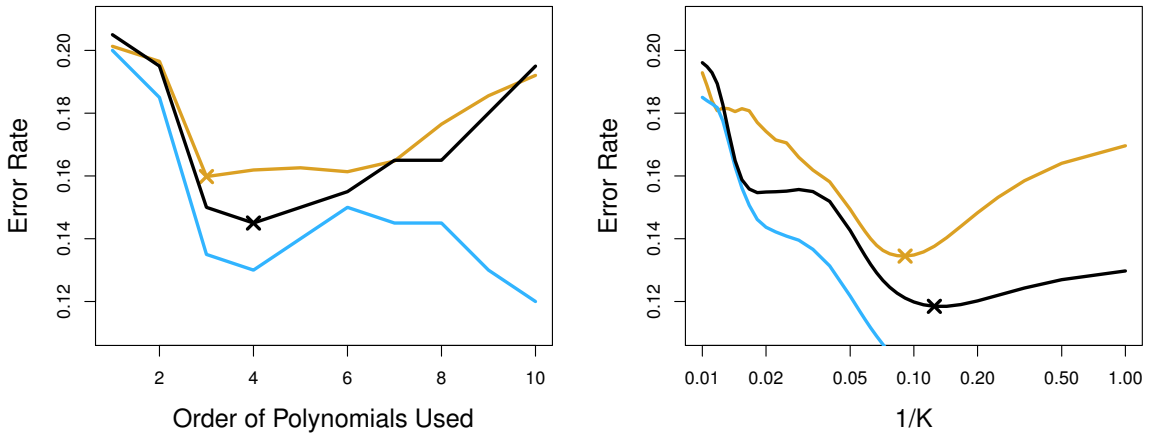


- - - - Bayes boundary

—— Logistic regression with polynomial predictors of increasing degree.

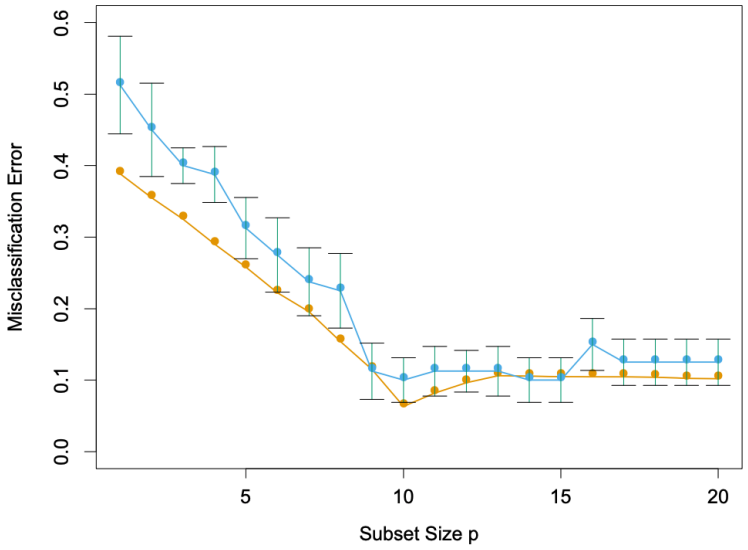
Choosing an optimal model

n.b. We don't know Bayes boundary in practice, but can choose the fit with the lowest error rate, e.g.



The one standard error rule

Forward stepwise selection



Blue: 10-fold cross validation
Yellow: True test error

- A number of models with $9 \leq p \leq 15$ have the same CV error.
- The vertical bars represent 1 standard error in the test error from the 10 folds.
- **Rule of thumb:** Choose the simplest model whose CV error is no more than one standard error above the model with the lowest CV error.

The wrong way to do cross validation

Suppose we want to classify 200 individuals according to whether they have cancer or not.

- We use logistic regression onto $1,000$ measurements of gene expression

Our proposed strategy

- Using all our data, select the 20 most significant genes using z-tests
- Estimate the test error of logistic regression with these 20 predictors via 10-fold cross validation

The wrong way to do cross validation

Suppose we want to classify 200 individuals according to whether they have cancer or not.

Let's simulate some data so that we know the true distribution P_0

- Each gene expression (of the $1,000$) is standard normal and independent of all others
- The response (cancer or not) is sampled from a coin flip — no correlation to any of the "*genes*"

The wrong way to do cross validation

Suppose we want to classify 200 individuals according to whether they have cancer or not.

Let's simulate some data so that we know the true distribution P_0

- Each gene expression (of the $1,000$) is standard normal and independent of all others
- The response (cancer or not) is sampled from a coin flip — no correlation to any of the "*genes*"

Under these settings, the misclassification rate for any classification method using these predictors should be 50%

The wrong way to do cross validation

Runing this simulation gives us CV error rate of 3%!

Why is this?

- Since we only have 200 individuals in total, among $1,000$ variables, at least some will be correlated with the response
- Doing variable selection using **all** the data, means that the variables we select will have some correlation with the response in every subset or fold in the cross validation

The **right** way to do cross validation

- Divide the data into 10 folds
- For $i = 1, ..., 10$:
    a. Using every fold except i, perform **both** the variable selection and model fit with the selected variables
    b. Compute the prediction error on the i^{th} fold
- Average the errors over the 10 splits

The simulation produces an error estimate of close to 50%

The **right** way to do cross validation

- Divide the data into 10 folds
- For $i = 1, ..., 10$:
    a. Using every fold except i, perform **both** the variable selection and model fit with the selected variables
    b. Compute the prediction error on the i^{th} fold
- Average the errors over the 10 splits

The simulation produces an error estimate of close to 50%

Moral of the story: Every aspect of the learning method that involves using the data (variable selection, for example) must be cross-validated

Why cross validation works

Recall: We're interested in estimating some function f_0 within a space of probability functions.

- How do we define '*best*' within this space?

Why cross validation works

Recall: We're interested in estimating some function f_0 within a space of probability functions.

- How do we define '*best*' within this space?

Define $O = (X_1, ..., X_p, Y) \sim P_0$

$$f_0 = \arg\min_{f \in \mathcal{F}} \int L(o, f) \partial P_0(o) \tag{14}$$

where $L(O, f)$ is a bounded loss function with finite variance.

Why cross validation works

Recall: We're interested in estimating some function f_0 within a space of probability functions.

- How do we define '*best*' within this space?

Define $O = (X_1, ..., X_p, Y) \sim P_0$

$$f_0 = \arg\min_{f \in \mathcal{F}} \int L(o, f) \partial P_0(o) \tag{14}$$

where $L(O, f)$ is a bounded loss function with finite variance.

We define the distance between f and f_0 as

$$d_0(f, f_0) \triangleq \mathbb{E}_{P_0}[L(O, f) - L(O, f_0)] \tag{15}$$

The best estimator is the one that minimizes $d_0(f, f_0)$

Why cross validation works

Let $\hat{f}_n^k = \hat{\Psi}_k(P_n) : k = 1, 2, ..., K$ be the K estimators we're evaluating. Our cross-validation selector is

$$\hat{K}(P_n) \triangleq \arg\min_k \mathbb{E}_{B_n} \int L(o, \hat{\Psi}_k(P^0_{n,B_n}))\partial P^1_{n,B_n}(o) \qquad (16)$$

Why cross validation works

Let $\hat{f}_n^k = \hat{\Psi}_k(P_n) : k = 1, 2, ..., K$ be the K estimators we're evaluating. Our cross-validation selector is

$$\hat{K}(P_n) \triangleq \arg\min_k \mathbb{E}_{B_n} \int L(o, \hat{\Psi}_k(P^0_{n,B_n}))\partial P^1_{n,B_n}(o) \qquad (16)$$

Consider an *oracle selector*:

$$\tilde{K}(P_n) \triangleq \arg\min_k \mathbb{E}_{B_n} \int L(o, \hat{\Psi}_k(P^0_{n,B_n}))\partial P_0(o) \qquad (17)$$

Why cross validation works

Let $\hat{f}_n^k = \hat{\Psi}_k(P_n) : k = 1, 2, ..., K$ be the K estimators we're evaluating. Our cross-validation selector is

$$\hat{K}(P_n) \triangleq \arg\min_k \mathbb{E}_{B_n} \int L(o, \hat{\Psi}_k(P^0_{n,B_n}))\partial P^1_{n,B_n}(o) \tag{16}$$

Consider an *oracle selector*:

$$\tilde{K}(P_n) \triangleq \arg\min_k \mathbb{E}_{B_n} \int L(o, \hat{\Psi}_k(P^0_{n,B_n}))\partial P_0(o) \tag{17}$$

Then for any $\delta > 0$,

$$\begin{aligned} \mathbb{E}_0 d_0(\hat{\Psi}_{\hat{K}(P_n)}(P^0_{n,B_n}), f_0) \leq (1 + 2\delta)\mathbb{E}_0 d_0(\hat{\Psi}_{\tilde{K}(P_n)}(P^0_{n,B_n}), f_0) \\ + C(\delta)\frac{\log(K)}{n} \end{aligned} \tag{18}$$

where $C(\delta)$ is a constant.

Why cross validation works

What this means:

- If none of the candidate learners converge at a parametric rate, the cross validated selector performs asymptotically as well (in the risk difference sense) as the oracle selector.

Why cross validation works

What this means:

- If none of the candidate learners converge at a parametric rate, the cross validated selector performs asymptotically as well (in the risk difference sense) as the oracle selector.
- If one of the candidate learners searches within a parametric model (that contains f_0), and thus achieves a parametric rate of convergence, then the cross validated selector achieves the almost parametric rate of convergence $\log n/n$.

Stacking / Ensembling

Question: Rather than just picking one estimator, can we combine them? e.g.

$$\bar{K}(P_n) \triangleq \alpha_1 \hat{\Psi}_1(P^0_{n,B_n}) + \cdots + \alpha_K \hat{\Psi}_K(P^0_{n,B_n}) : \sum_{k=1}^{K} \alpha_k = 1 \quad (19)$$

Each weighted average is a unique candidate algorithm in our '*augmented*' library. e.g.

- Taking the average across the estimators corresponds to $\alpha_k = 1/K$.

Stacking / Ensembling

Question: Rather than just picking one estimator, can we combine them? e.g.

$$\bar{K}(P_n) \triangleq \alpha_1 \hat{\Psi}_1(P^0_{n,B_n}) + \cdots + \alpha_K \hat{\Psi}_K(P^0_{n,B_n}) : \sum_{k=1}^{K} \alpha_k = 1 \quad (19)$$

Each weighted average is a unique candidate algorithm in our '*augmented*' library. e.g.

- Taking the average across the estimators corresponds to $\alpha_k = 1/K$.

We could then apply the cross validated selector to this augmented library.

- Alternatively: could just estimate the α_k's directly.
- Could also make α_k's dependent on our inputs $(X_1, ..., X_p)$.

References

[1] ISL. Chapters 4-5.

[2] ESL. Chapters 7, 8.8.

[3] Super Learner. M.J. van der Laan, E.C. Polley, A.E. Hubbard (2007).