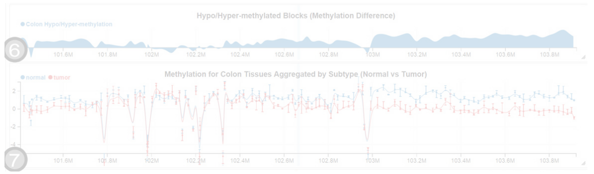


Introduction to Data Science: Classifier Evaluation and Model Selection

Héctor Corrada Bravo

University of Maryland, College Park, USA

CMSC320: 2020-04-26

Classifier evaluation

How do we determine how well classifiers are performing?

One way is to compute the *error rate* of the classifier, the percent of mistakes it makes when predicting class

Classifier evaluation

```
logis_fit <- glm(default ~ balance, data=Def
logis_pred_prob <- predict(logis_fit, type="
logis_pred <- ifelse(logis_pred_prob > 0.5,
print(table(predicted=logis_pred, observed=D

# error rate
mean(Default$default != logis_pred) * 100

# dummy error rate
mean(Default$default != "No") * 100
```

```
##            observed
## predicted    No  Yes
##       No   9625  233
##       Yes    42  100

## [1] 2.75

## [1] 3.33
```

Classifier evaluation

We need a more precise language to describe classification mistakes:

	True Class +	**True Class -**	**Total**
Predicted Class +	True Positive (TP)	False Positive (FP)	P*
Predicted Class -	False Negative (FN)	True Negative (TN)	N*
Total	P	N	

Classifier evaluation

Using these we can define statistics that describe classifier performance

Name	Definition	Synonyms
False Positive Rate (FPR)	FP / N	Type-I error, 1-Specificity
True Positive Rate (TPR)	TP / P	1 - Type-II error, power, sensitivity, **recall**
Positive Predictive Value (PPV)	TP / P*	**precision**, 1-false discovery proportion
Negative Predicitve Value (NPV)	FN / N*	

Classifier evaluation

In the credit default case we may want to increase **TPR** (recall, make sure we catch all defaults) at the expense of **FPR** (1-Specificity, clients we lose because we think they will default)

Classifier evaluation

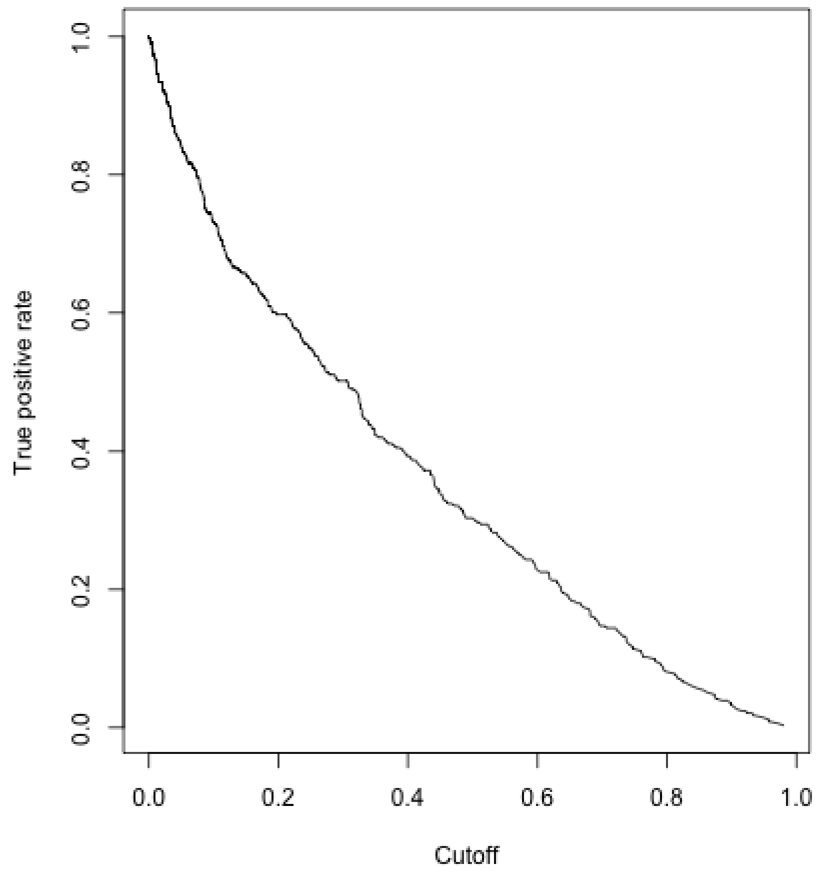
This leads to a natural question: Can we adjust our classifiers TPR and FPR?

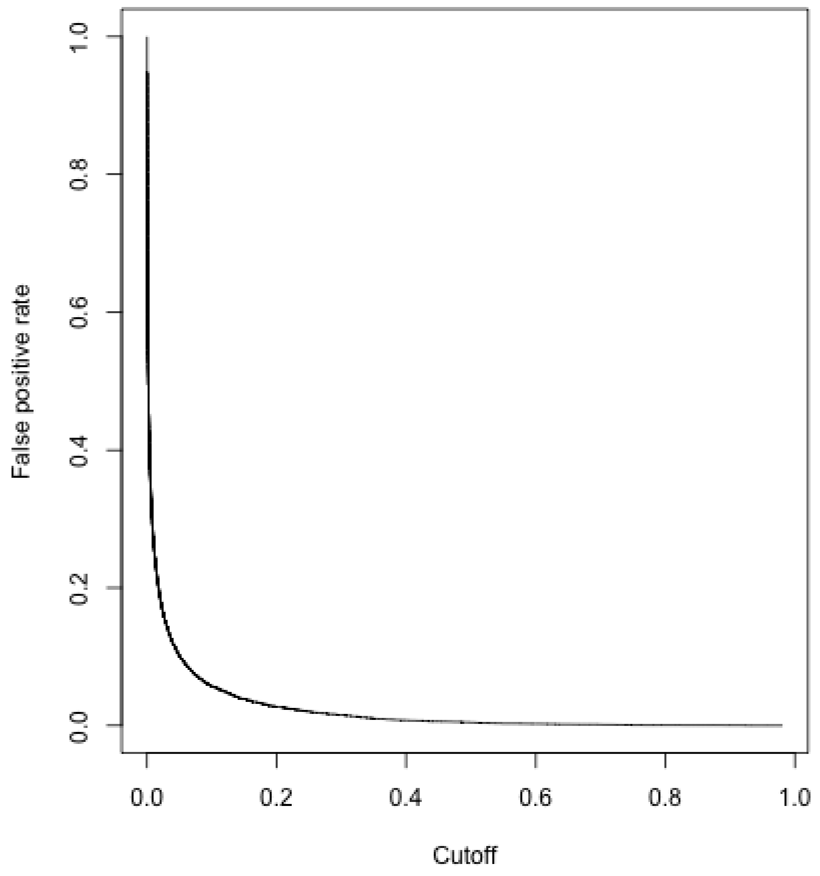
Remember we are classifying `Yes` if

$$\log \frac{P(Y=\texttt{Yes}|X)}{P(Y=\texttt{No}|X)} > 0 \Rightarrow P(Y=\texttt{Yes}|X) > 0.5$$

What would happen if we use $P(Y=\texttt{Yes}|X) > 0.2$?

Classifier evaluation

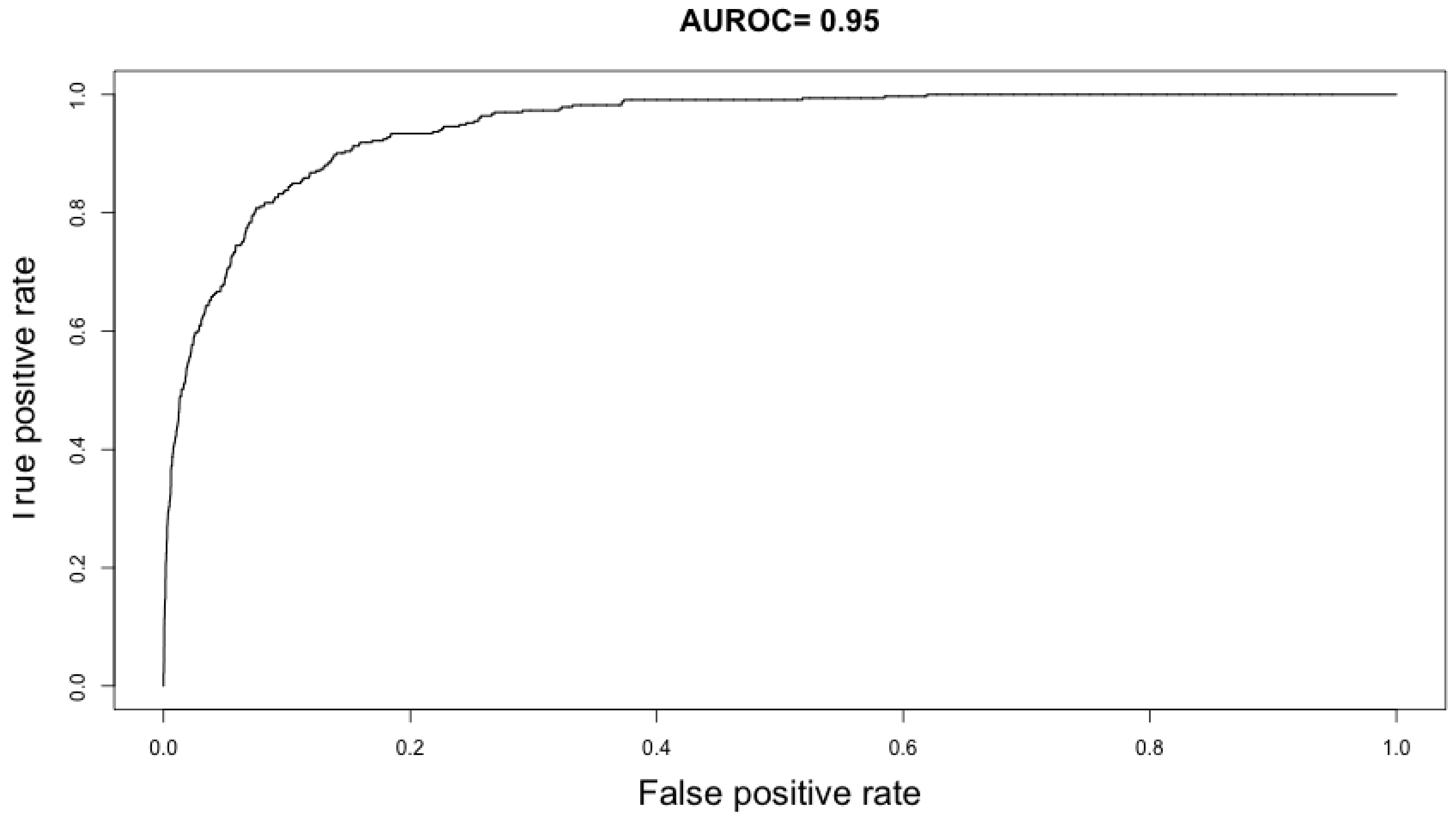




Classifier evaluation

A way of describing the TPR and FPR tradeoff is by using the **ROC curve** (Receiver Operating Characteristic) and the **AUROC** (area under the ROC)

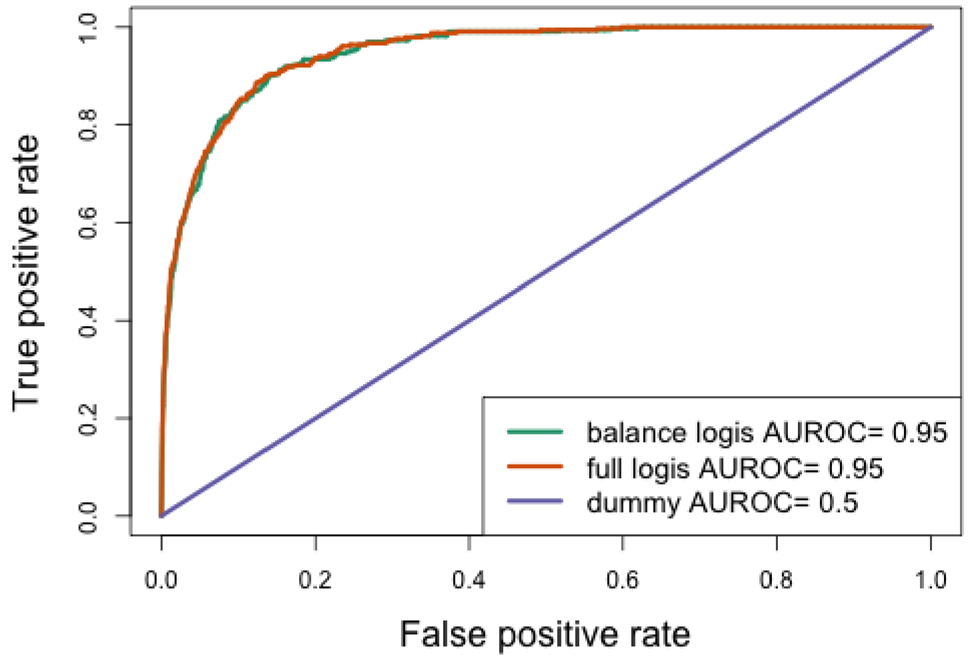
Classifier evaluation



Classifier evaluation

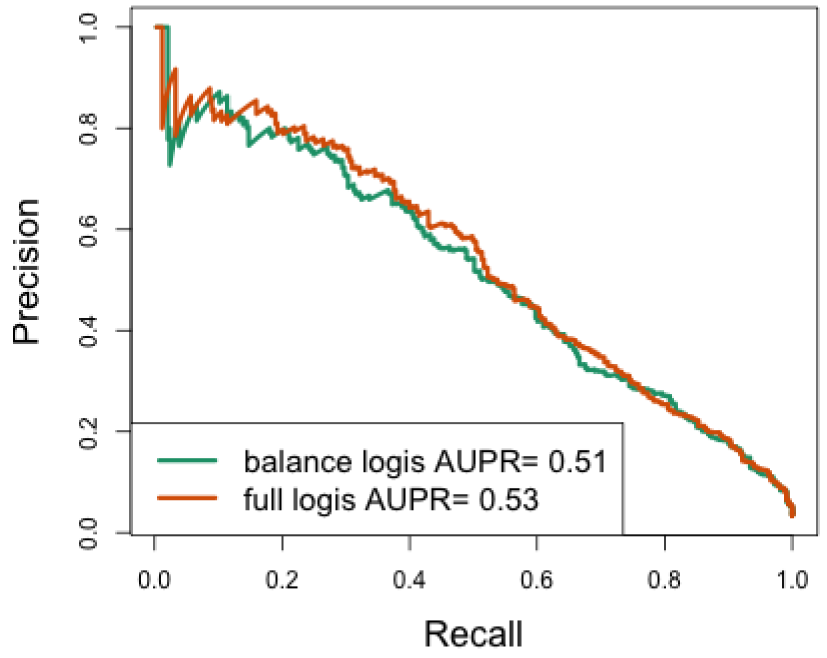
Consider comparing a logistic regression model using all predictors in the dataset, including an interaction term between balance and student.

```
default ~
balance*student
+ income
```



Classifier evaluation

Another metric that is frequently used to understand classification errors and tradeoffs is the precision-recall curve:



Classifier evaluation

The bigger model shows a slightly higher precision at the same recall values and slightly higher area under the precision-recall curve.

This is commonly found in datasets where there is a skewed distribution of classes (e.g., there are many more "No" than "Yes" in this dataset).

The area under the PR curve tends to distinguish classifier performance than area under the ROC curve in these cases.

Model Selection

Our goal when we use a learning model like linear or logistic regression, decision trees, etc., is to learn a model that can predict outcomes for new unseen data.

Model Selection

We should therefore think of model evaluation based on *expected predicted error*: what will the prediction error be for data *outside* the training data.

Model Selection

We should therefore think of model evaluation based on *expected predicted error*: what will the prediction error be for data *outside* the training data.

How then, do we measure our models' ability to predict unseen data, when we only have access to training data?

Cross-validation

The most common method to evaluate model **generalization** performance is *cross-validation*.

It is used in two essential data analysis phases: *Model Selection* and *Model Assessment*.

Cross-validation

Model Selection

Decide what kind, and how complex of a model we should fit.

Cross-validation

Model Selection

Decide what kind, and how complex of a model we should fit.

Consider a regression example: I will fit a linear regression model, what predictors should be included?, interactions?, data transformations?

Cross-validation

Model Selection

Decide what kind, and how complex of a model we should fit.

Consider a regression example: I will fit a linear regression model, what predictors should be included?, interactions?, data transformations?

Another example is what classification tree depth to use.

Cross-validation

Model Selection

Decide what kind, and how complex of a model we should fit.

Consider a regression example: I will fit a linear regression model, what predictors should be included?, interactions?, data transformations?

Another example is what classification tree depth to use.

Which kind of algorithm to use, linear regression vs. decision tree vs. random forest

Cross-validation

Model Assessment

Determine how well does our selected model performs as a **general** model.

Cross-validation

Model Assessment

Determine how well does our selected model performs as a **general** model.

Ex. I've built a linear regression model with a specific set predictors. How well will it perform on unseen data?

Cross-validation

Model Assessment

Determine how well does our selected model performs as a **general** model.

Ex. I've built a linear regression model with a specific set predictors. How well will it perform on unseen data?

The same question can be asked of a classification tree of specific depth.

Cross-validation

Cross-validation is a *resampling* method to obtain estimates of **expected prediction error rate** (or any other performance measure on unseen data).

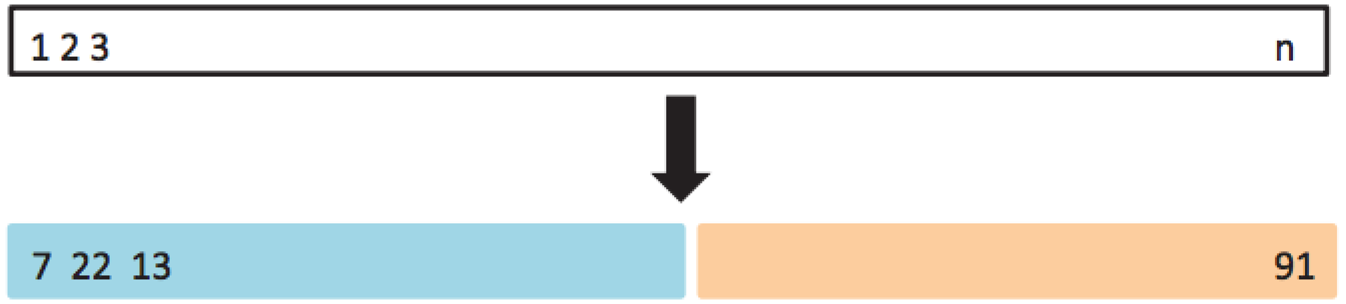
In some instances, you will have a large predefined test dataset **that you should never use when training**.

In the absence of access to this kind of dataset, cross validation can be used.

Validation Set

The simplest option to use cross-validation is to create a *validation* set, where our dataset is **randomly** divided into *training* and *validation* sets.

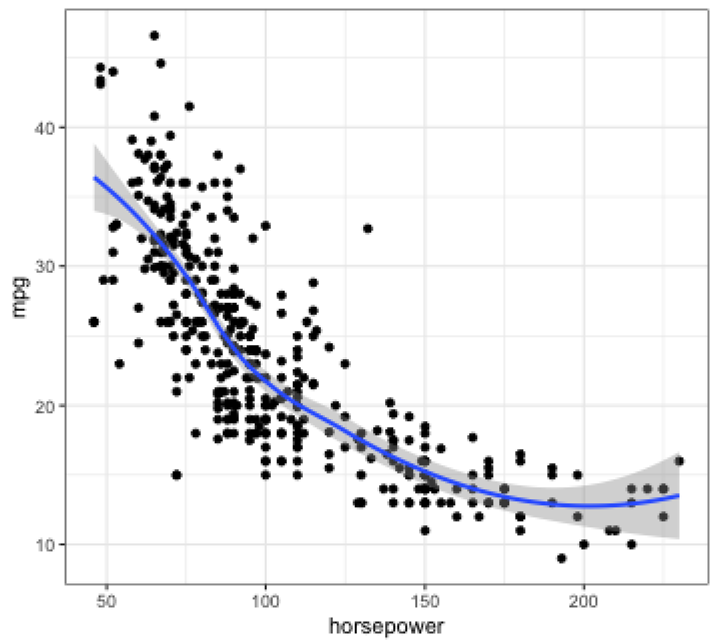
Then the *validation* is set aside, and not used at until until we are ready to compute **test error rate** (once, don't go back and check if you can improve it).



Validation Set

Let's look at our running example using automobile data, where we want to build a regression model to predict miles per gallon given other auto attributes.

A linear regression model was not appropriate for this dataset. Use *polynomial* regression as an illustrative example.



Validation Set

For polynomial regression, our regression model (for a single predictor X) is given as a d degree polynomial.

$$\mathbb{E}[Y|X = x] = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d$$

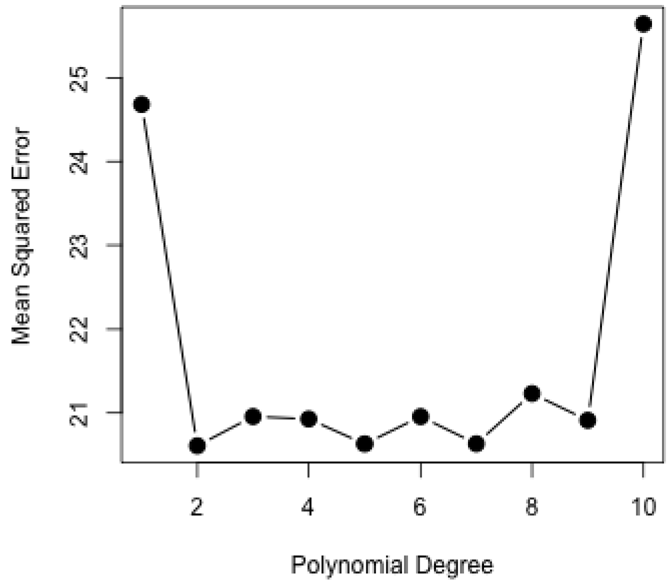
For *model selection*, we want to decide what degree d we should use to model this data.

Validation Set

Using the *validation set* method, split our data into a training set,

fit the regression model with different polynomial degrees d on the training set,

measure test error on the validation set.



Resampled validation set

The validation set approach can be prone to sampling issues.

It can be highly variable as error rate is a random quantity and depends on observations in training and validation sets.

Resampled validation set

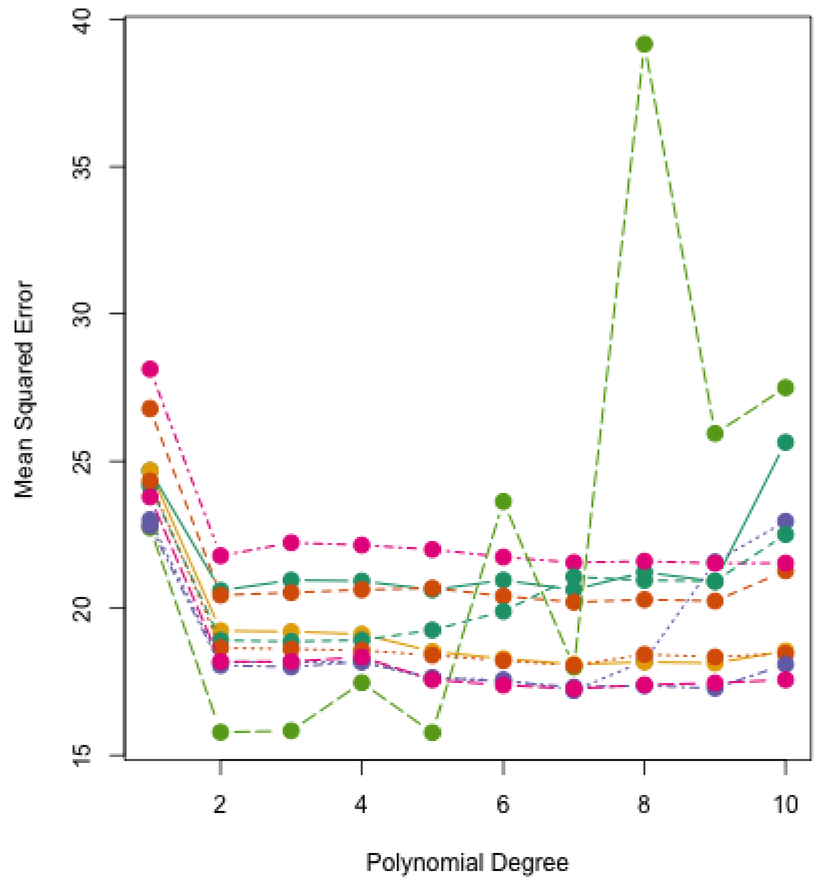
The validation set approach can be prone to sampling issues.

It can be highly variable as error rate is a random quantity and depends on observations in training and validation sets.

We can improve our estimate of *test error* by averaging multiple measurements of it (remember the law of large numbers).

Resampled validation set

Resample validation set 10 times (yielding different validation and training sets) and averaging the resulting test errors.



Leave-one-out Cross-Validation

This approach still has some issues.

Each of the training sets in our validation approach only uses 50% of data to train, which leads to models that may not perform as well as models trained with the full dataset and thus we can overestimate error.

Leave-one-out Cross-Validation

This approach still has some issues.

Each of the training sets in our validation approach only uses 50% of data to train, which leads to models that may not perform as well as models trained with the full dataset and thus we can overestimate error.

To alleviate this situation, we can extend our approach to the extreme: Make each single training point it's own validation set.

Leave-one-out Cross-Validation

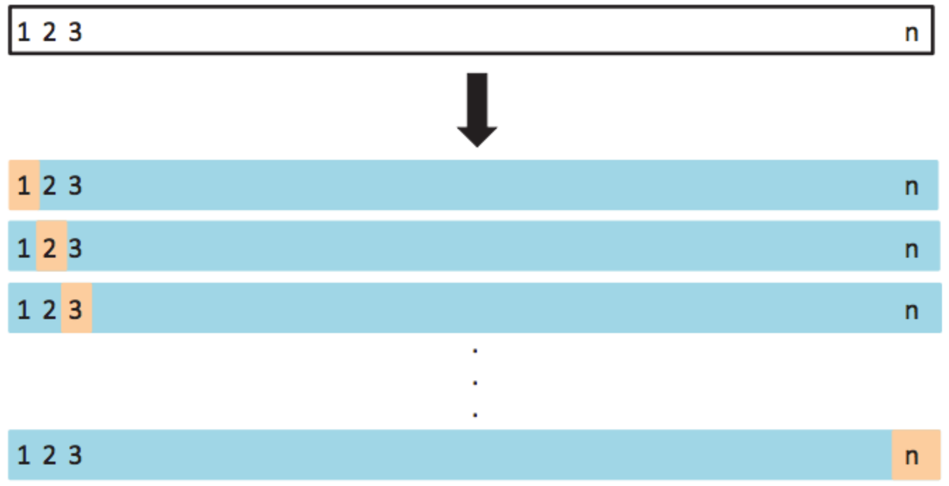
Procedure:

For each observation i in data set:

a. Train model on all but i-th observation

b. Predict response for i-th observation

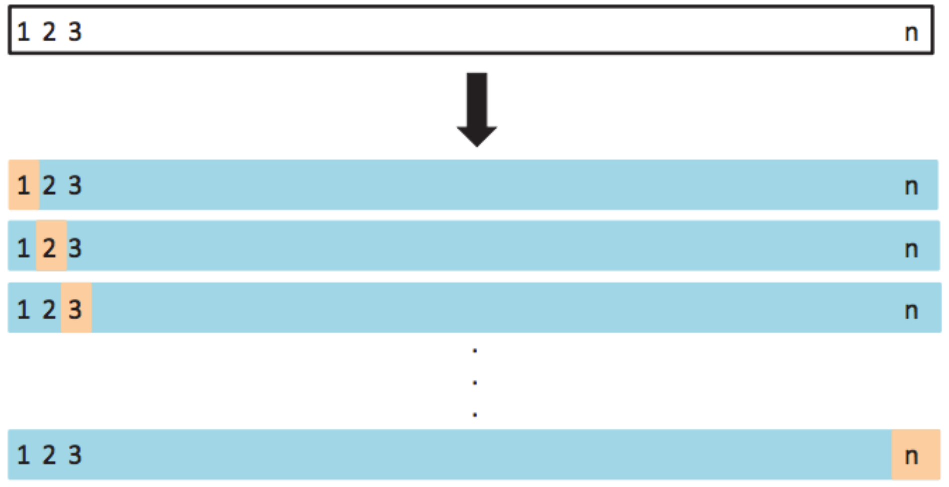
c. Calculate prediction error



Leave-one-out Cross-Validation

This gives us the following *cross-validation* estimate of error.

$$CV_{(n)} = \frac{1}{n} \sum_i (y_i - \hat{y}_i)^2$$



Leave-one-out Cross-Validation

- use $n - 1$ observations to train each model
- no sampling effects introduced since error is estimated on each sample

Leave-one-out Cross-Validation

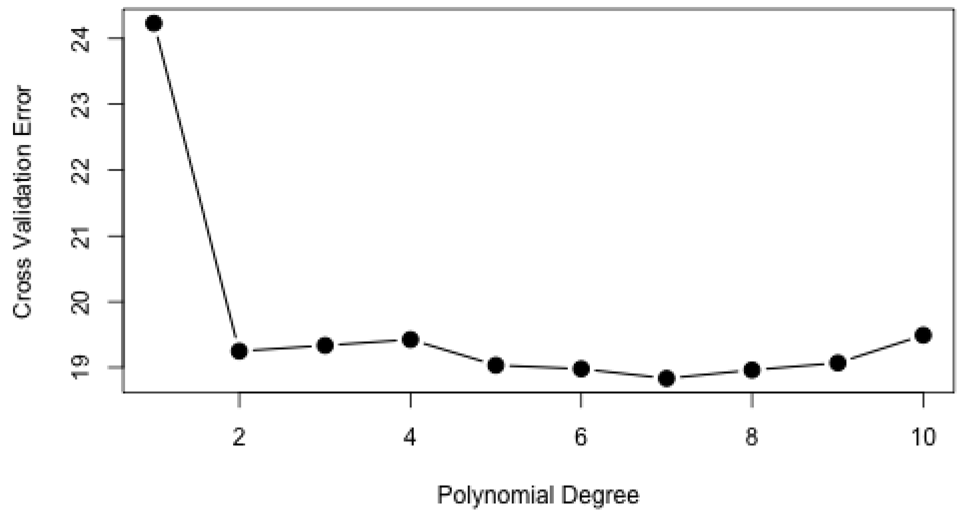
- use $n-1$ observations to train each model
- no sampling effects introduced since error is estimated on each sample

Disadvantages:

- Depending on the models we are trying to fit, it can be very costly to train $n-1$ models.
- Error estimate for each model is highly variable (since it comes from a single datapoint).

Leave-one-out Cross-Validation

On our running example



k-fold Cross-Validation

This discussion leads us to the most commonly used cross-validation approach *k-fold Cross-Validation*.

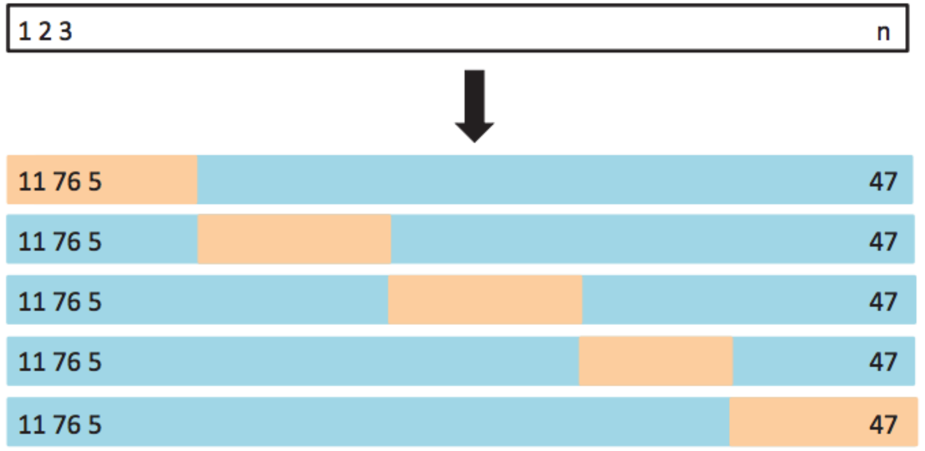
k-fold Cross-Validation

Procedure:

Partition observations randomly into k groups (folds).

For each of the k groups of observations:

- Train model on observations in the other $k-1$ folds
- Estimate test-set error (e.g., Mean Squared Error) on this fold



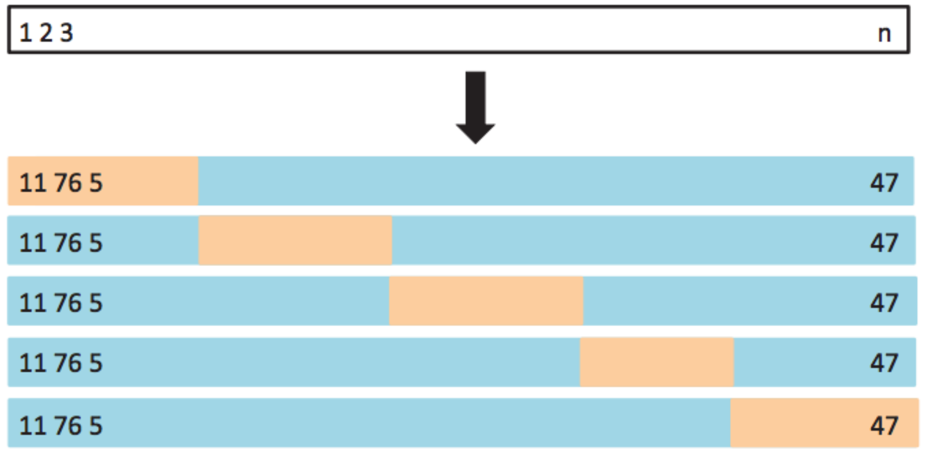
k-fold Cross-Validation

Procedure:

Compute average error across k folds

$$CV_{(k)} = \frac{1}{k} \sum_i MSE_i$$

where MSE_i is mean squared error estimated on the i-th fold



k-fold Cross-Validation

- Fewer models to fit (only k of them)
- Less variance in each of the computed test error estimates in each fold.

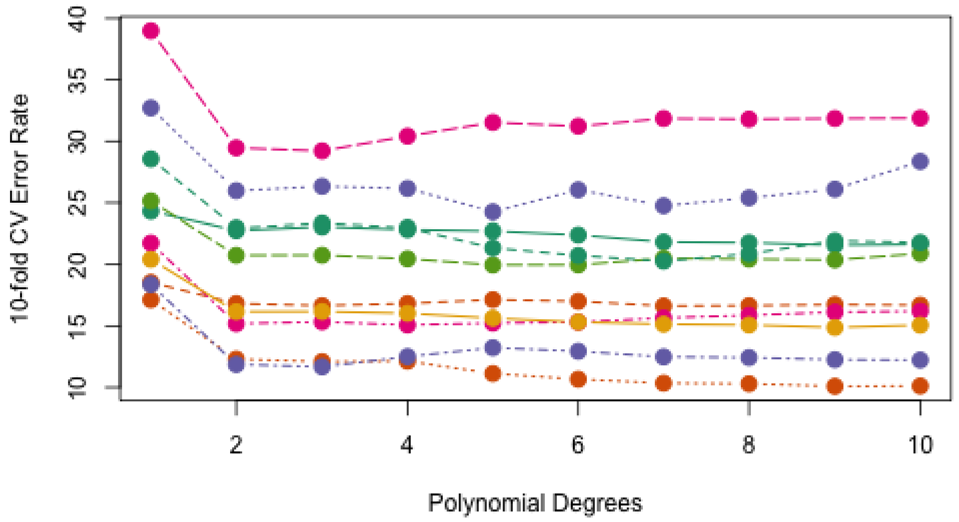
k-fold Cross-Validation

- Fewer models to fit (only k of them)
- Less variance in each of the computed test error estimates in each fold.

It can be shown that there is a slight bias (over estimating usually) in error estimate obtained from this procedure.

k-fold Cross-Validation

Running Example



Cross-Validation in Classification

Each of these procedures can be used for classification as well.

In this case we would substitute MSE with performance metric of choice.

E.g., error rate, accuracy, TPR, FPR, AUROC.

Cross-Validation in Classification

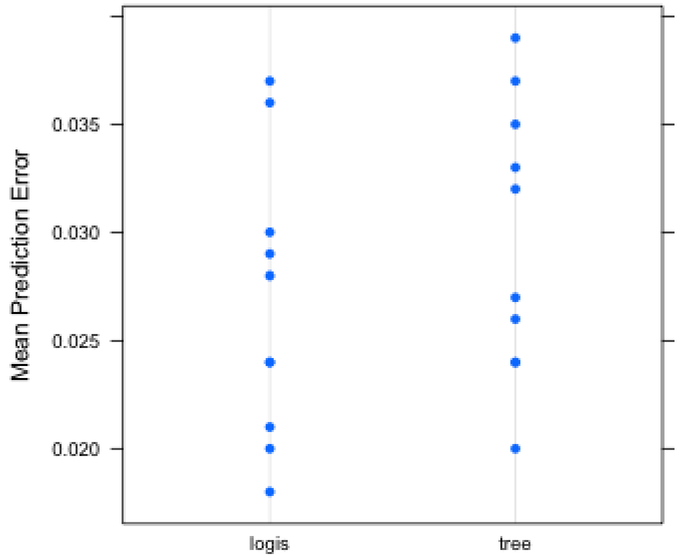
Each of these procedures can be used for classification as well.

In this case we would substitute MSE with performance metric of choice. E.g., error rate, accuracy, TPR, FPR, AUROC.

Note however that not all of these work with LOOCV (e.g. AUROC since it can't be defined over single data points).

Comparing models using cross-validation

Suppose you want to compare two classification models (logistic regression vs. a decision tree) on the `Default` dataset. We can use Cross-Validation to determine if one model is better than the other, using a t-test for example.



Comparing models using cross-validation

Using hypothesis testing:

term	estimate	std.error	statistic	p.value
(Intercept)	0.0267	0.0020306	13.148828	0.0000000
methodtree	0.0030	0.0028717	1.044677	0.3099998

In this case, we do not observe any significant difference between these two classification methods.

Summary

Model selection and assessment are critical steps of data analysis.

Error and accuracy statistics are not enough to understand classifier performance.

Classifications can be done using probability cutoffs to trade, e.g., TPR-FPR (ROC curve), or precision-recall (PR curve).

Area under ROC or PR curve summarize classifier performance across different cutoffs.

Summary

Resampling methods are general tools used for this purpose.

k-fold cross-validation can be used to provide larger training sets to algorithms while stabilizing empirical estimates of expected prediction error