CS 277, Data Mining

Review of Predictive Modeling

Padhraic Smyth
Department of Computer Science
Bren School of Information and Computer Sciences
University of California, Irvine

Typical Predictive Modeling Tasks

- Classification: predicting a categorical outcome
  - Spam email
  - Automatically classifying the sentiment of a product review
- Regression:
  - Predicting the lifetime value of a customer
- Ranking, e.g., by P(y | x)
  - Ranking which Web advertisement a user is likely to click on
  - Credit scoring

Data as a Matrix…..

Rows -> objects/individuals
Columns -> measurements/variables

ID	Income	Age	….	Monthly Debt	Good Risk?
18276	65,000	55	….	2200	Yes
72514	28,000	19	….	1500	No
28163	120,000	62	….	1800	Yes
17265	90,000	35	….	4500	No
…	…	…	….	…	…
…	…	…	….	…	…
61524	35,000	22	….	900	Yes

Notation for Predictive Modeling

- **Training data, pairs** $\underline{x}(i)$, $y(i)$, $i = 1....N$
  - N times d matrix of data
  - Each row corresponds to a d-dimensional vector $\underline{x}(i)$, $i= 1,....N$
  - Additional column of target values $y(i)$, $i= 1,....N$ (e.g., y real-valued, y binary, etc)

- **We want to learn a model that can predict y given $\underline{x}$**
  - $f(\underline{x} ; \underline{\alpha})$ is our model: some scalar function of $\underline{x}$
  - $\underline{\alpha}$ is a p-dimensional parameter vector
  - e.g., $f(\underline{x} ; \underline{\alpha}) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + ... \alpha_d x_d$ (linear predictor, $p = d+1$)

- **We can measure the quality of any particular model with an error function**
  - Squared Error score (as an example: we could use other scores)

$$E_{train}(\underline{\alpha}) = \sum_i [y(i) - f(\underline{x}(i) ; \underline{\alpha})]^2$$

where $f(\underline{x}(i) ; \underline{\alpha})$ is the prediction of y by the model using parameters $\underline{\alpha}$

Empirical Learning

- Minimize the total error $E_{train}(\underline{\alpha})$ on the training data

$$\text{e.g.,}\quad E_{train}(\underline{\alpha}) = \Sigma_i \, [y(i) - f(\underline{x}(i) \, ; \, \underline{\alpha}) \,]^2$$

- Note that in minimizing we treat $E_{train}(\underline{\alpha})$ as a scalar function of a p-dimensional unknown parameter vector $\underline{\alpha}$, with known fixed $\underline{x}$'s and y's.

- Minimizing $E_{train}(\underline{\alpha})$ as a function of $\underline{\alpha}$ is an optimization problem
  - Occasionally there is a direct solution, e.g., via linear algebra
  - More typically we must use some form of local iterative search
    - E.g., gradient descent, starting from a random starting point in $\underline{\alpha}$ space

Generalization to New Data

Will minimizing $E_{train}(\underline{\alpha})$ give us the best possible predictor?

No: we really want to find the $f(x; \underline{\alpha})$ that best predicts y on **future** data,

i.e., in minimizing expected value of $[y - f(\underline{x} ; \underline{\alpha})]^2$ on future data

In practice we can only use $E_{train}(\underline{\alpha})$ as a surrogate for performance on future data

Empirical learning

- Minimize $E(\underline{\theta})$ on the training data D_{train}
- If D_{train} is large and model is simple we are assuming that the best f on training data is also the best predictor f on future test data D_{test}

- Note that the test error $E_{test}(\underline{\alpha})$ on new test data might be significantly larger than $E_{train}(\underline{\alpha})$ on training data

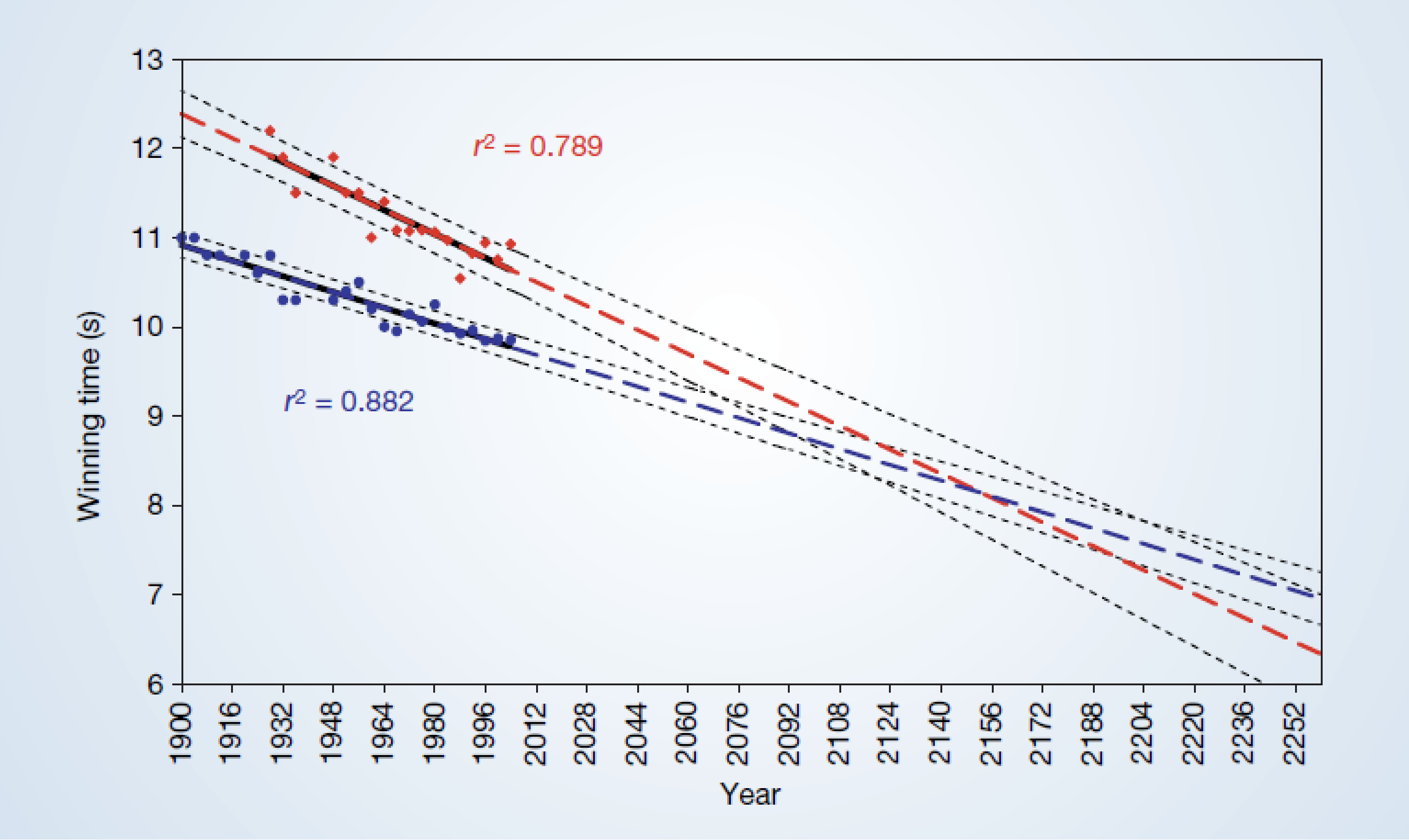


Figure 1 The winning Olympic 100-metre sprint times for men (blue points) and women (red points), with superimposed best-fit linear regression lines (solid black lines) and coefficients of determination. The regression lines are extrapolated (broken blue and red lines for men and women, respectively) and 95% confidence intervals (dotted black lines) based on the available points are superimposed. The projections intersect just before the 2156 Olympics, when the winning women's 100-metre sprint time of 8.079 s will be faster than the men's at 8.098 s.

From Tatem et al., Nature 2004.
(see also response letters at http://faculty.washington.edu/kenrice/natureletter.pdf)

Models, Loss Functions, and Optimization Methods

- Models:
  - Linear weighted sums of the input variables (linear regression)
  - Non-linear functions of linear weighted sums (logistic regression, neural networks, GLMs)
  - Thresholded functions (decision trees)

- Objective functions:
  - Regression: Squared (L2) error, Absolute (L1) error, Robust loss, log-loss/log-likelihood
  - Classification: classification error, margin, log-loss/log-likelihood

- Optimization methods
  - For linear models, squared error loss: solving sets of linear equations in p parameters
    - Tend to scale as $O(N d^2 + d^3)$
  - More complex models
    - Iterative search methods, e.g., gradient-based, 2nd-order (Newton), etc

Important point: most learning algorithms are a combination of

Model + Objective Function + Optimization Method

Method	Model, f(x|α)	Objective Function	Optimization Method

Method	Model, $f(x\|\alpha)$	Objective Function	Optimization Method
Linear regression	Weighted sum	Squared error	Linear system of equations

Method	Model, $f(x\|\alpha)$	Objective Function	Optimization Method
Linear regression	Weighted sum	Squared error	Linear system of equations
Logistic regression	h(weighted sum)	Log-Likelihood	Iterative, system of equations

Method	Model, $f(x\|\alpha)$	Objective Function	Optimization Method
Linear regression	Weighted sum	Squared error	Linear system of equations
Logistic regression	h(weighted sum)	Log-Likelihood	Iterative, system of equations
Neural network	Weighted sum of logistic regressions	Squared error	Gradient-based

Method	Model, $f(x\|\alpha)$	Objective Function	Optimization Method
Linear regression	Weighted sum	Squared error	Linear system of equations
Logistic regression	h(weighted sum)	Log-Likelihood	Iterative, system of equations
Neural network	Weighted sum of logistic regressions	Squared error	Gradient-based
Support vector machine	Sparse weighted sum	Margin	Convex optimization

Method	Model, $f(x\|\alpha)$	Objective Function	Optimization Method
Linear regression	Weighted sum	Squared error	Linear system of equations
Logistic regression	h(weighted sum)	Log-Likelihood	Iterative, system of equations
Neural network	Weighted sum of logistic regressions	Squared error	Gradient-based
Support vector machine	Sparse weighted sum	Margin	Convex optimization
Decision trees	Binary tree	Classification error	Greedy search over trees

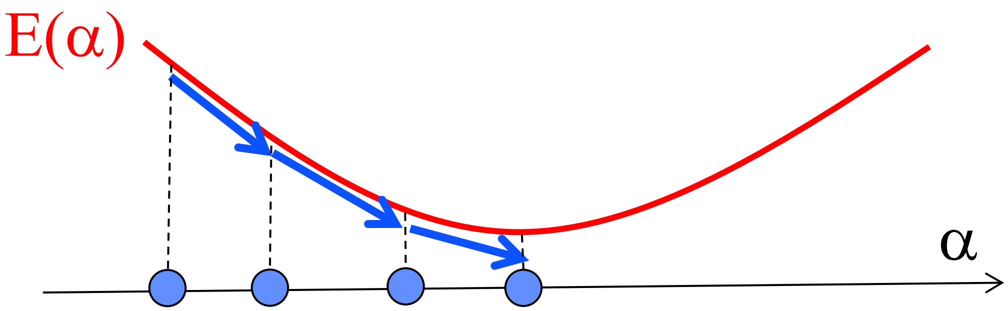
Gradient Descent

- Linear regression requires O(d^3) matrix inversion
  - If d = 10k, 100k, 1 million, -> very slow!

- Alternative
  - Directly minimize loss function via *gradient descent*
  - Algorithm:
    - Start at some $\underline{\alpha}$, e.g., randomly selected
    - Compute local gradient
    - Move downhill in space some small distance
    - If not converged, compute gradient and move again

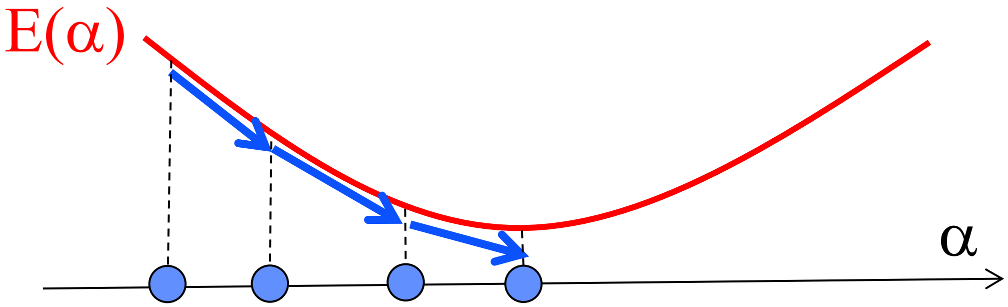
  - If E is convex then this is usually works well
    - e.g., linear models + squared error = convex problem

Optimization

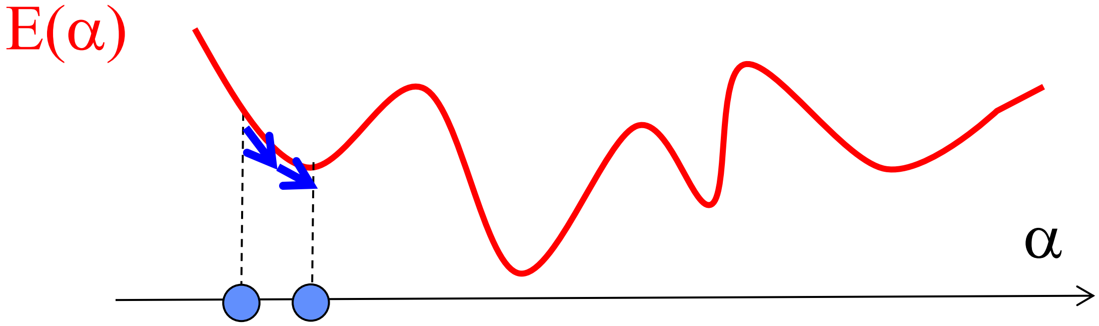


Easy
(convex)

Optimization



Easy
(convex)



Hard
(non-convex)

Stochastic Gradient Descent (SGD)

- Standard gradient descent
  - Per iteration, compute gradient by summing over N gradient terms

- Stochastic gradient descent
  - Compute gradient **per case**, and update parameters
  - Cycle through cases in turn or in random order
  - Usually far fewer iterations than standard method

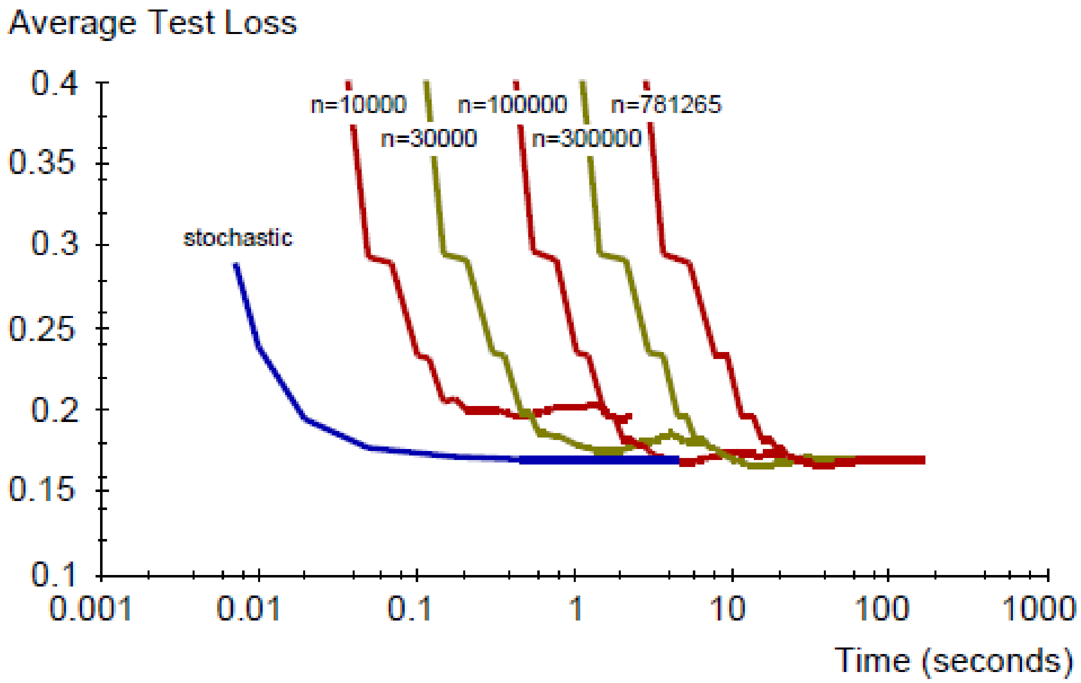
  - O(N d) per iteration can be O(N r) for sparse data

    (r = average # non-zero elements per row, r << d)

  - Many variants
    - "mini-batch": update based on gradient from small sets of cases
    - "online": continuously update model as new cases arrive

SGD Example

Binary Text Classification Problem
d = 47k word features
N = 781k documents
Various forms of linear classifiers with regularization



Log-loss problem

Batch Conjugate Gradient on various training set sizes

Stochastic Gradient on the full set

From Leon Bottou, Stochastic Gradient Learning, MLSS Summer School 2011, Purdue University, http://learning.stat.purdue.edu/mlss/_media/mlss/bottou.pdf

A Simple Baseline Model: k-Nearest Neighbor Regression

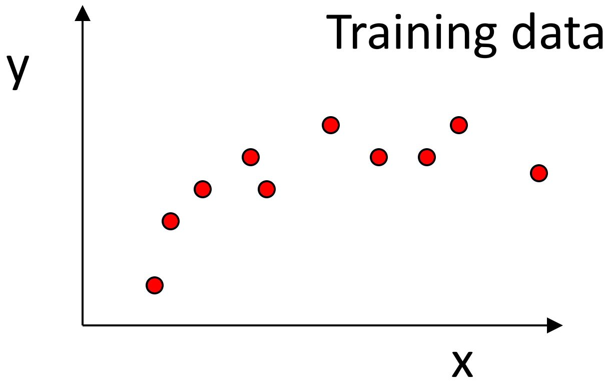
- Given a training data set D_{train} with $\{\underline{x}, y\}$ training pairs
  - To predict y for a new vector $\underline{x}^*$?
  - Find the k nearest neighbors to $\underline{x}^*$ in the training data D_{train}
    - e.g., using Euclidean distance (be careful about scaling!)
  - Prediction of y is y* = the average y values of the k-nearest neighbors

- Comments on k-NN regression
  - Extremely simple method, can be a useful baseline
  - The predictions for y are piecewise constant functions of $\underline{x}$
  - An example of a “non-parametric” method
  - k can be chosen automatically by cross-validation (see later slides)
  - Same idea can be used for classification (when y is a categorical class variable)

- Weaknesses
  - Performance tends to degrade as d (dimensionality of x) increases
  - Can be very sensitive to how distance is defined
  - Requires that all of training data D_{train} be stored in memory and accessed at prediction time

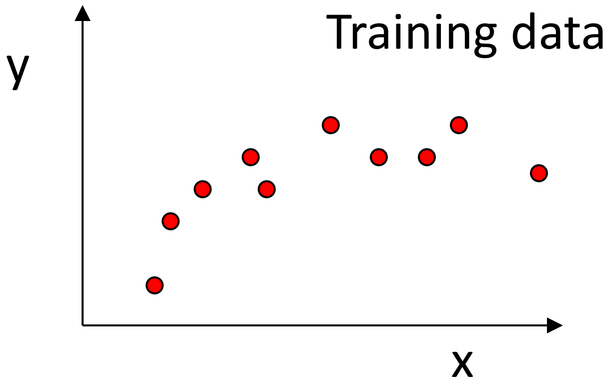
Picking the Best Model from a Set of Models

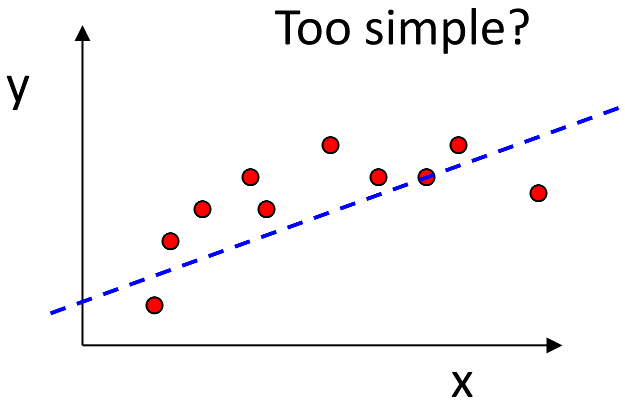
Complexity versus Goodness of Fit



Complexity versus Goodness of Fit

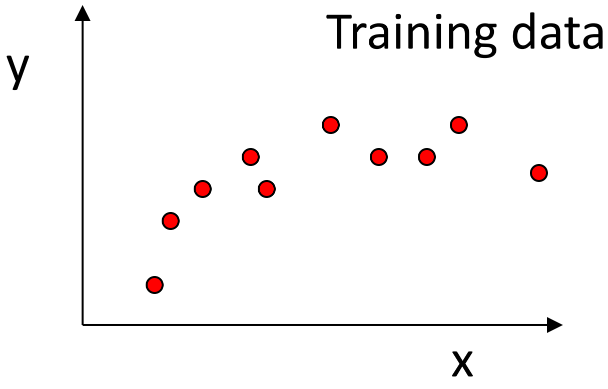
Here the red circles represent training data and the blue curves are models fitted to the data

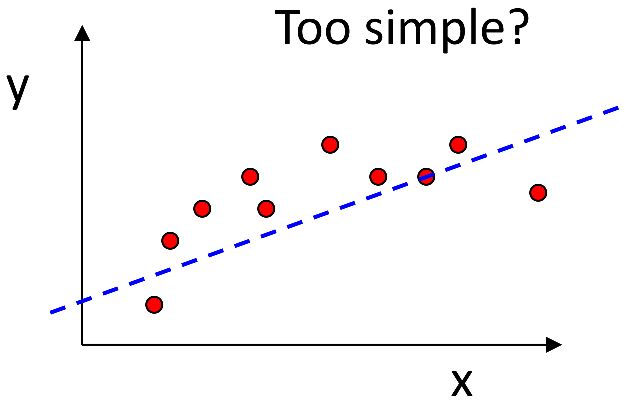


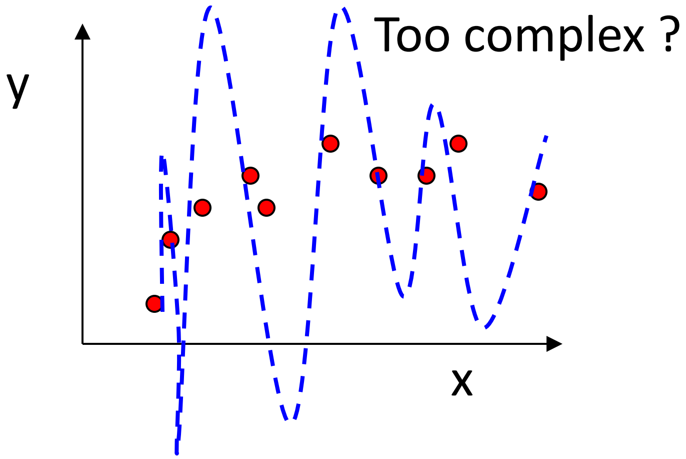


Complexity versus Goodness of Fit

Here the red circles represent training data and the blue curves are models fitted to the data

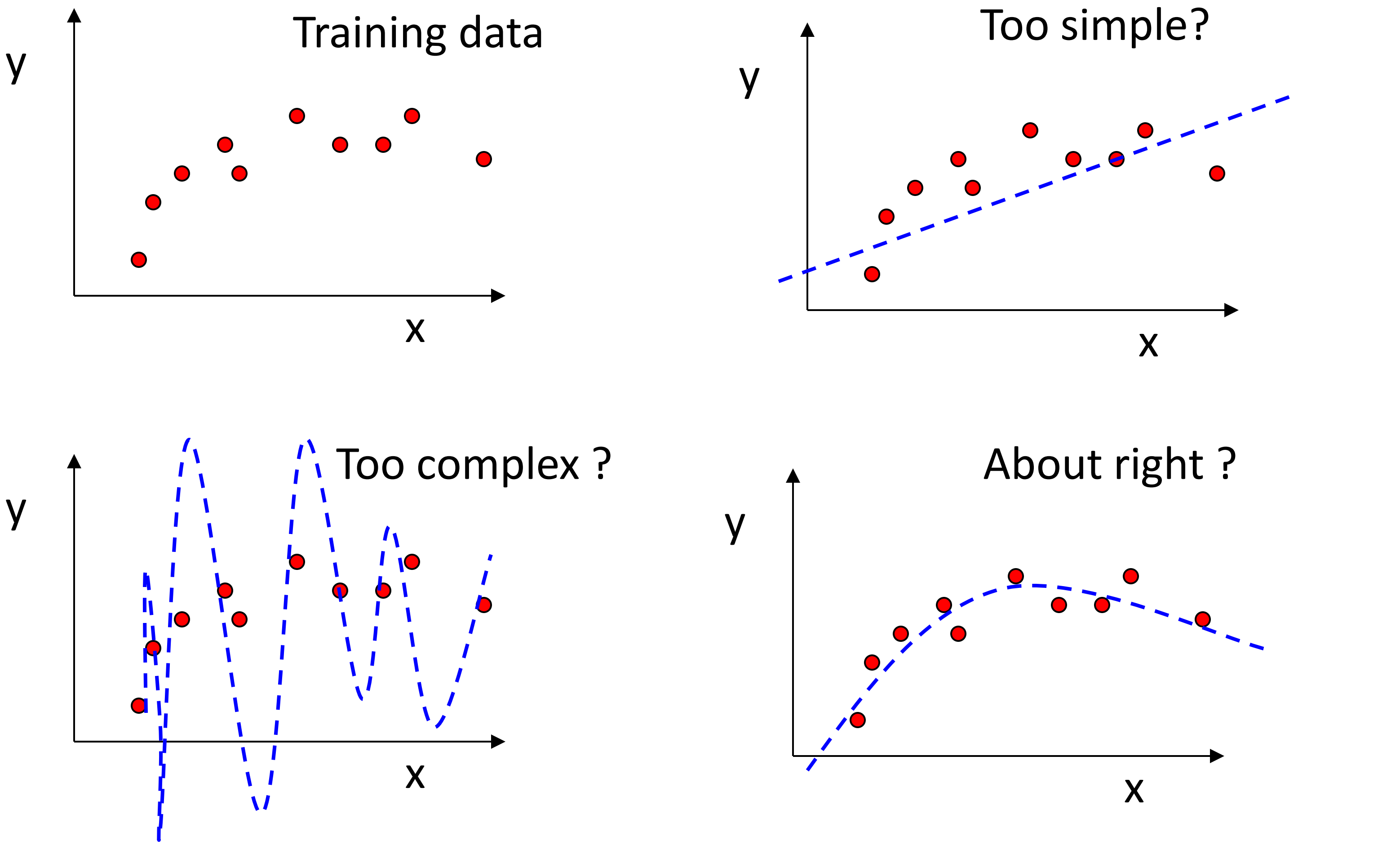

Training data


Too simple?

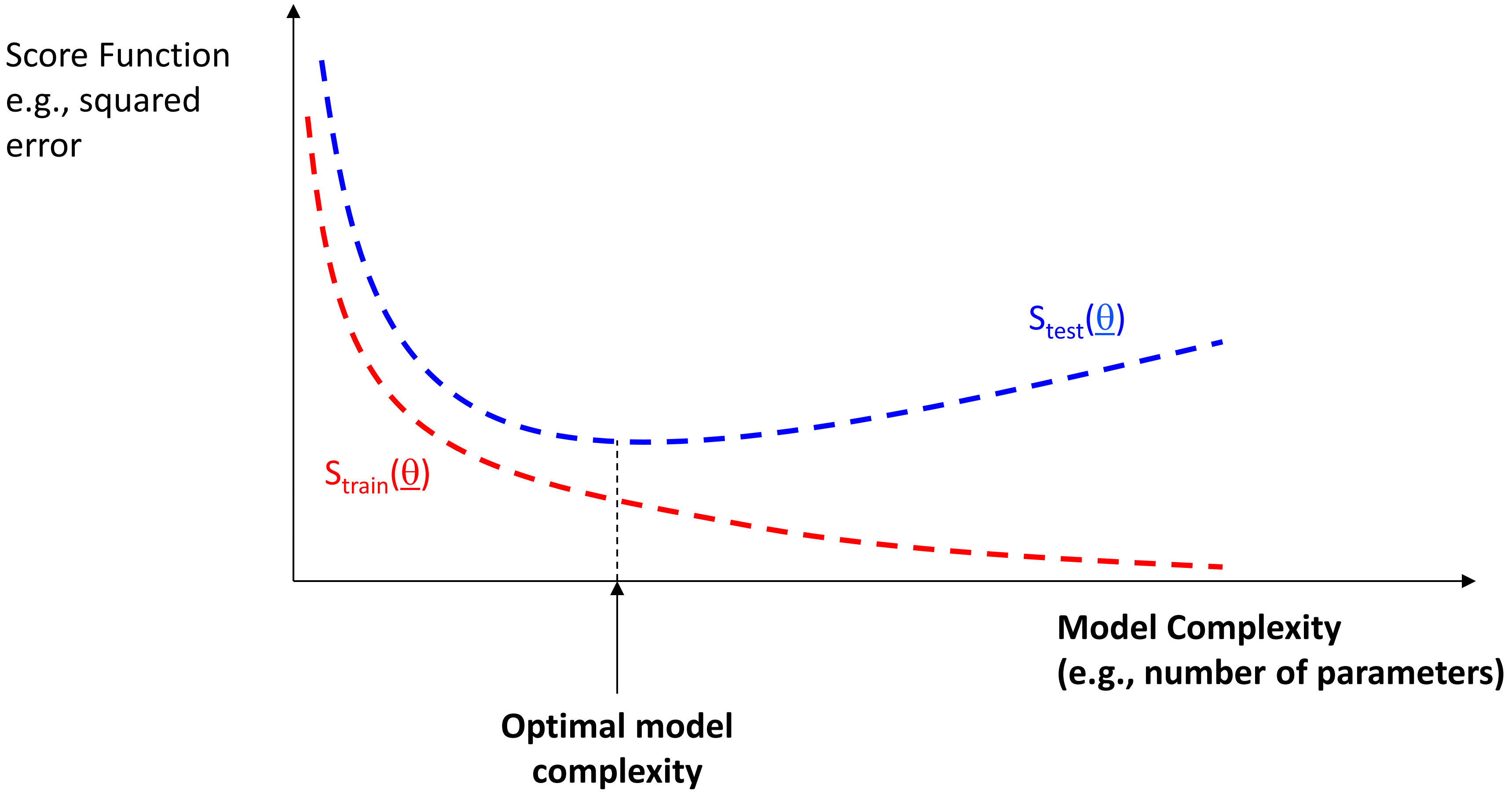

Too complex ?

Complexity versus Goodness of Fit

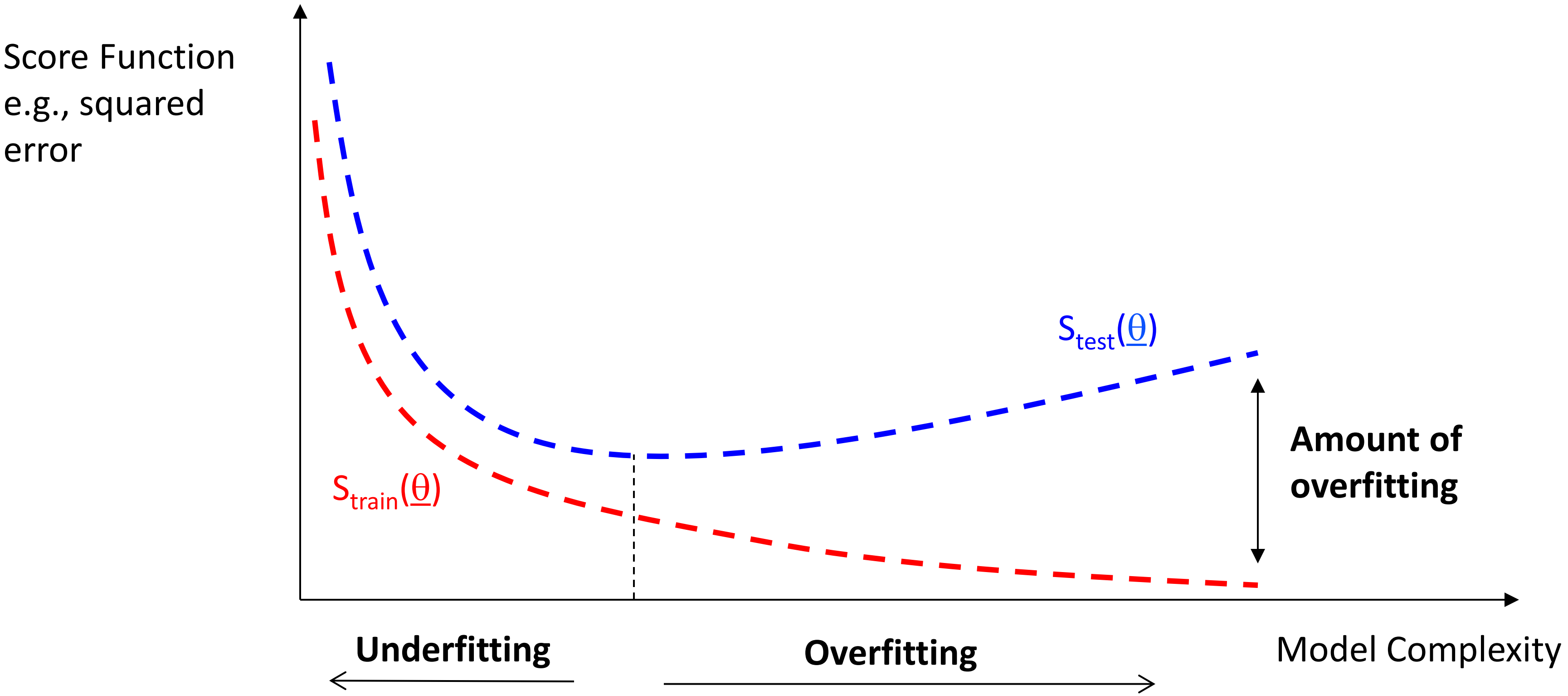
Here the red circles represent training data and the blue curves are models fitted to the data



Complexity and Generalization



Complexity and Generalization



Bias and Variance in Model Fitting

Bias:

Expected difference between our model's predictions and the true targets

Can be reduced by making the model more complex

High-bias -> model with few parameters, e.g., linear predictor

Low-bias -> model with many parameters, e.g., large neural network

Variance:

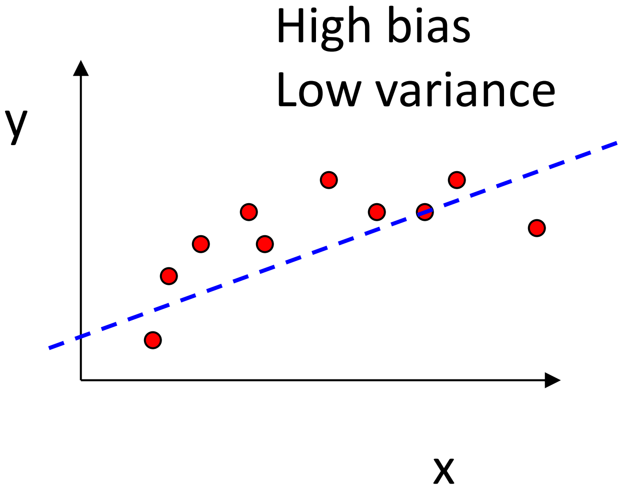
Variability in our model's predictions across data sets of size N

Can be reduced by increasing N

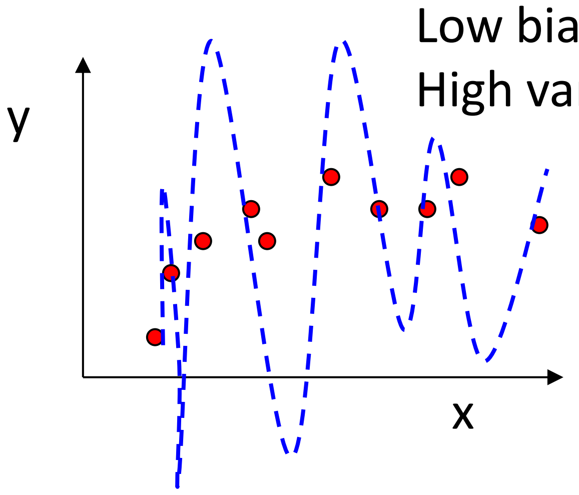
High-variance -> model with many parameters, e.g., large neural network

Low-variance -> model with few parameters, e.g., linear predictor

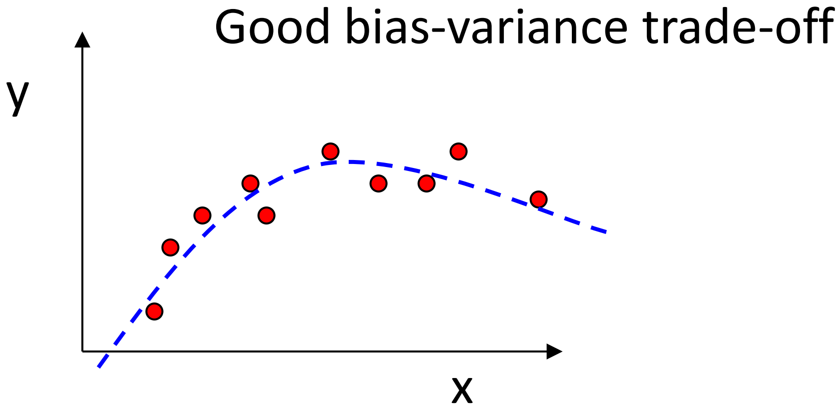
Bias-Variance Examples



High bias
Low variance



Low bias
High variance



Good bias-variance trade-off

Here the red circles represent training data and the blue curves are models fitted to the data

Classic Result: The Bias-Variance Tradeoff

Expected Squared-error on Test Data = Bias^2 + Variance

There is a fundamental tradeoff between bias and variance ("no free lunch")

Typically when we decrease bias we increase variance, and vice-versa

-> strive to find a model that balances both terms

Classic Result: The Bias-Variance Tradeoff

Expected Squared-error on Test Data = Bias^2 + Variance

There is a fundamental tradeoff between bias and variance ("no free lunch")

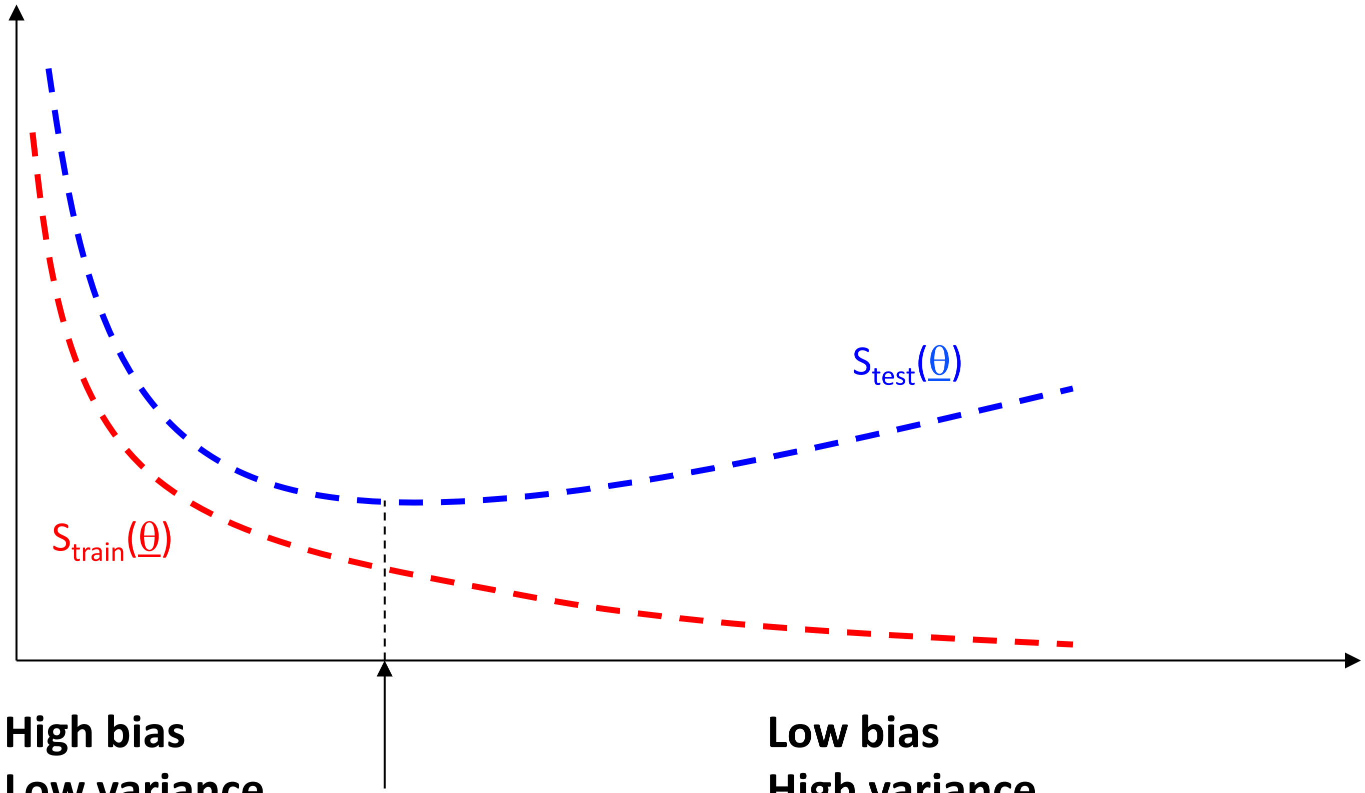
Typically when we decrease bias we increase variance, and vice-versa

-> strive to find a model that balances both terms

Question: why don't we just measure Bias and Variance on the training data and then select the model that minimizes Bias^2 + Variance ?

Complexity and Generalization

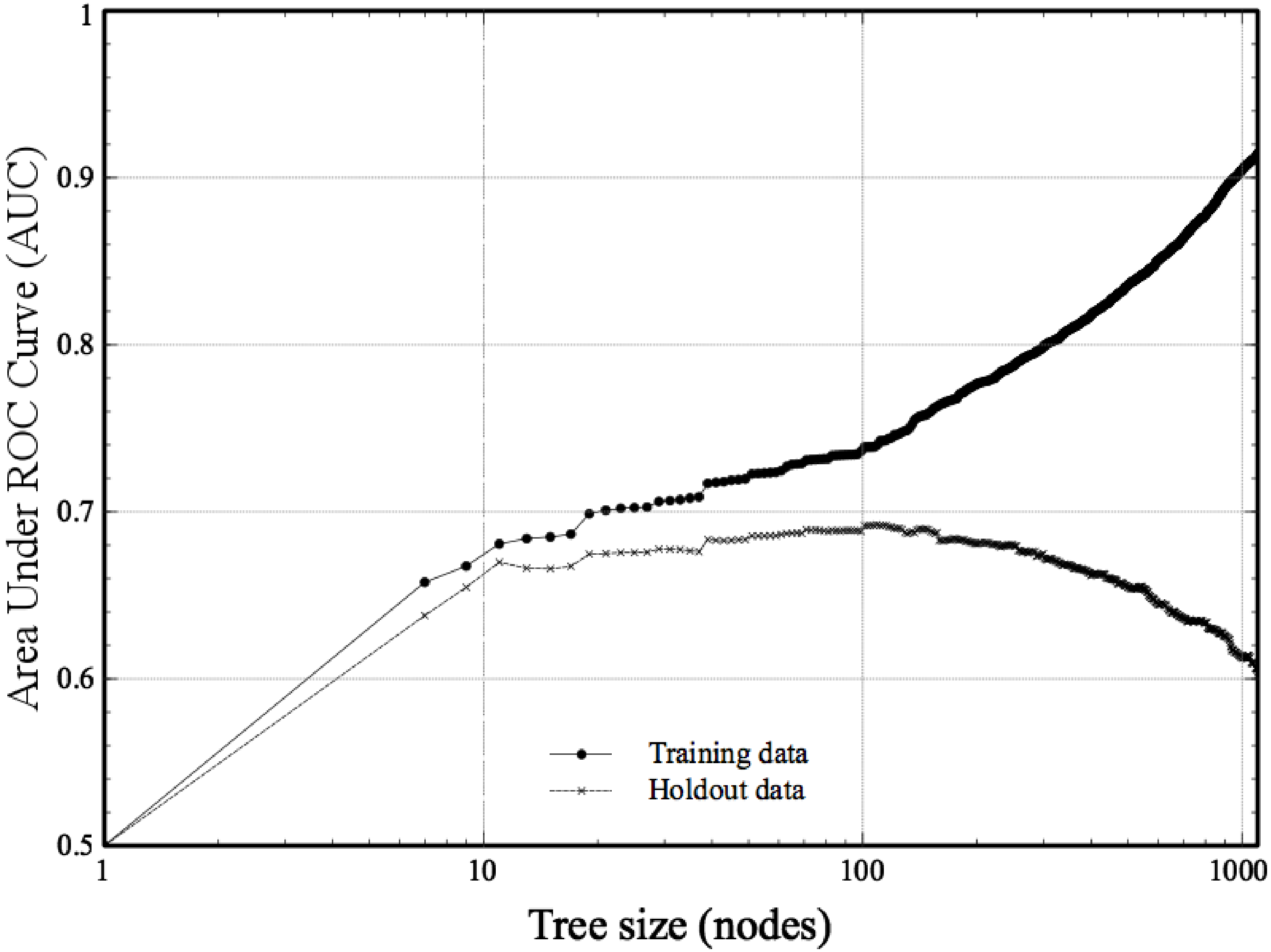
Score Function
e.g., squared
error



High bias
Low variance

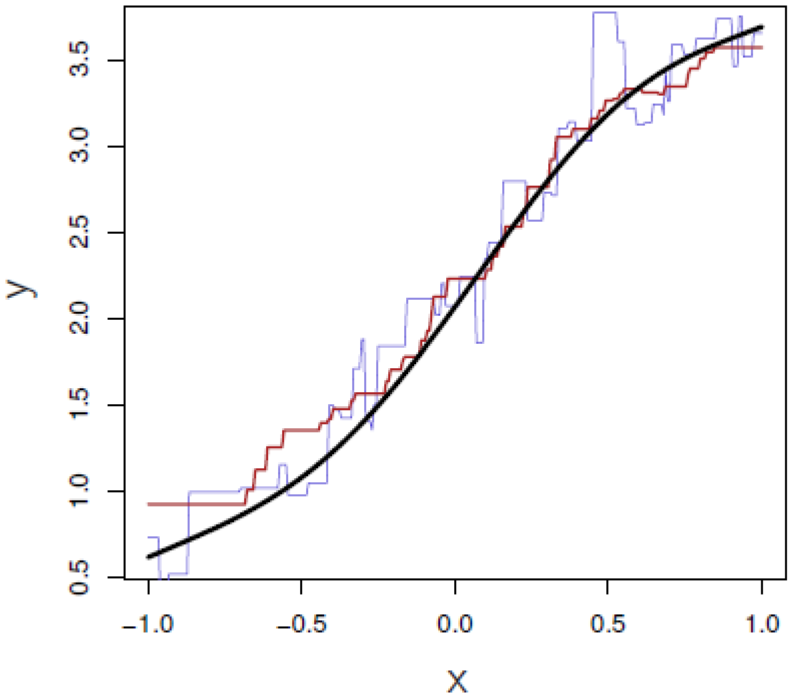
Optimal model
complexity

Low bias
High variance



Holdout (Test) Data performance (as measured by AUC) as a function of the number of nodes in a decision tree classifier, for a "churn" classification problem, N =47,000

From Chapter 8: Visualizing Model Performance, in Data Science for Business (O Reilly, 2013),
with permission from the authors, F. Provost and T. Fawcett

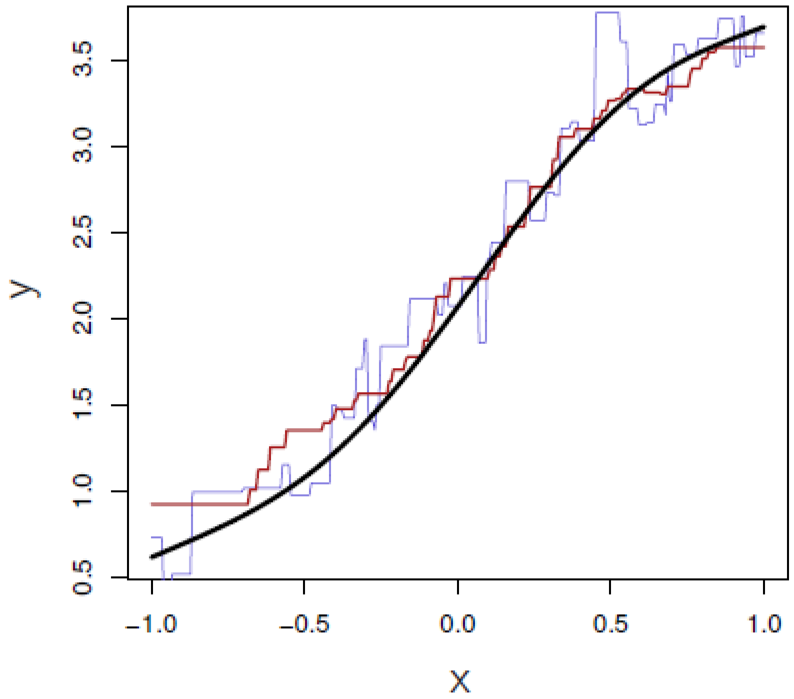


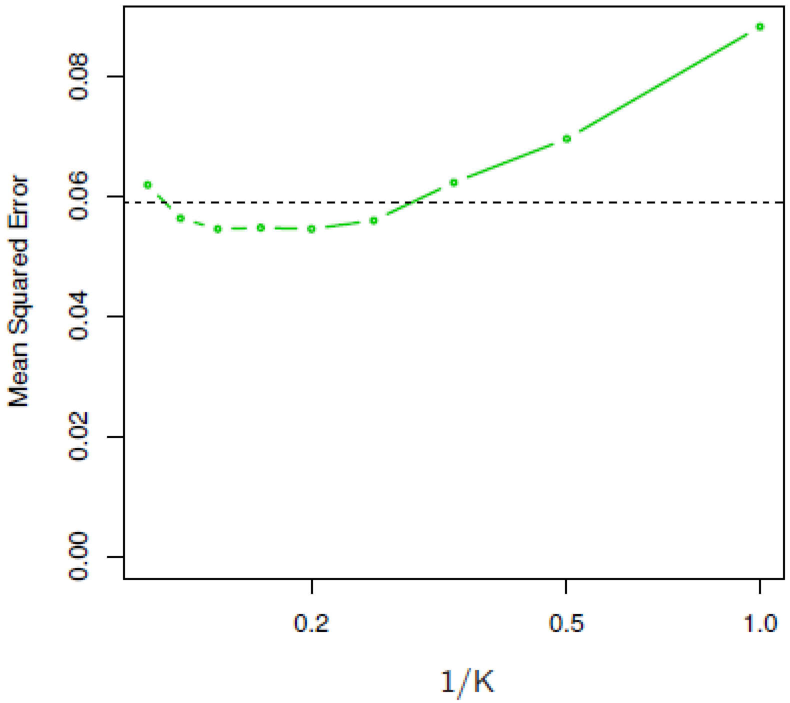
True relationship between X and Y is in black

Predictions of KNN regression with K=1 is in blue and K=9 in red. Note that both are piecewise constant as a function of x and red (K=9) is smoother

N = 100 training data points

From Chapter 3 on Linear Regression in *An Introduction to Statistical Learning, with Applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani (text available online)





True relationship between X and Y is in black

Predictions of KNN regression with K=1 is in blue and K=9 in red. Note that both are piecewise constant as a function of x and red (K=9) is smoother

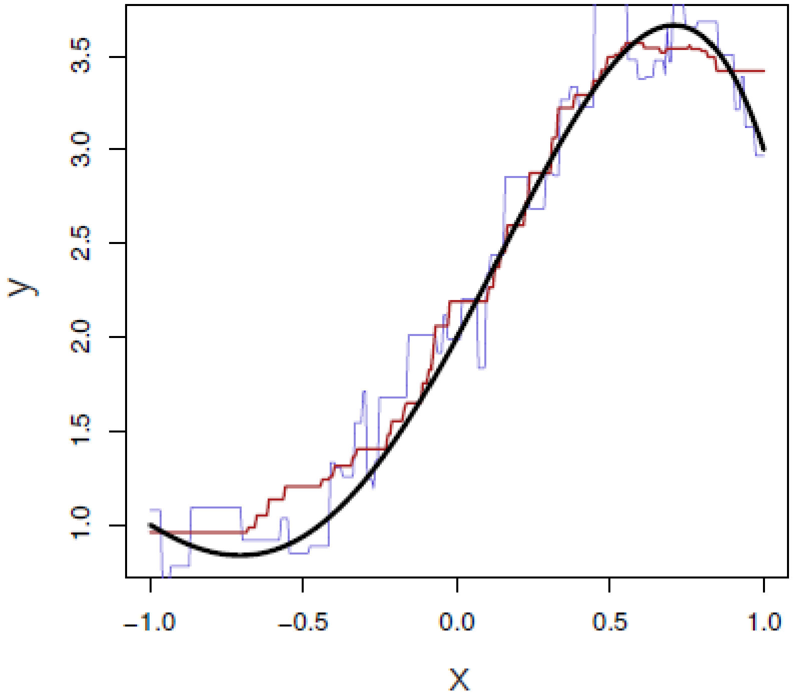
N = 100 training data points

Test mean-squared error (MSE) for the problem on the left

X-axis is 1/K where K is the value used for predictions from KNN regression

Dotted black line is for linear regression

From Chapter 3 on Linear Regression in *An Introduction to Statistical Learning, with Applications in R* (Springer, 2013) with permission from the authors:
G. James, D. Witten, T. Hastie and R. Tibshirani (text available online)

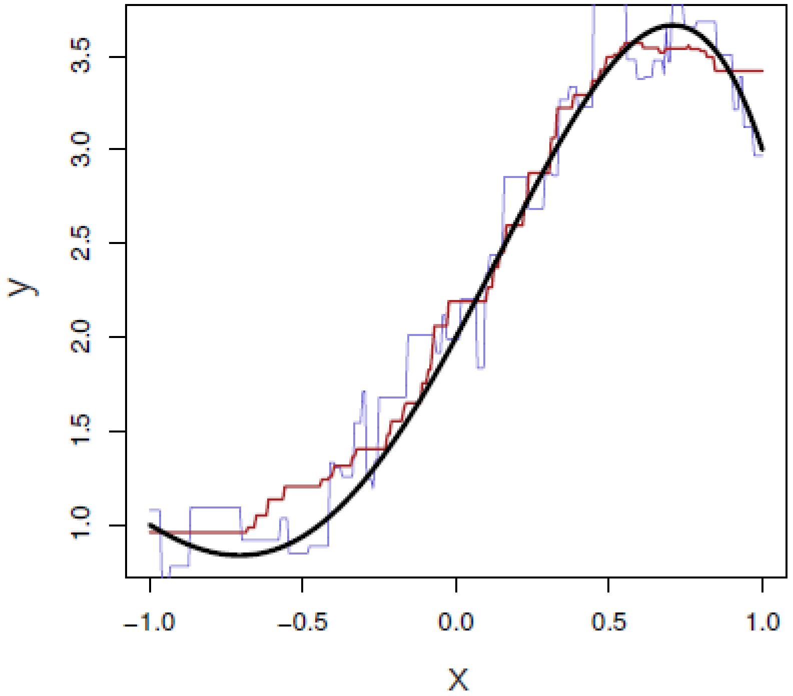


A more non-linear relationship, in black

Predictions of KNN regression with K=1 is in blue and K=9 in red.

N = 100 training data points

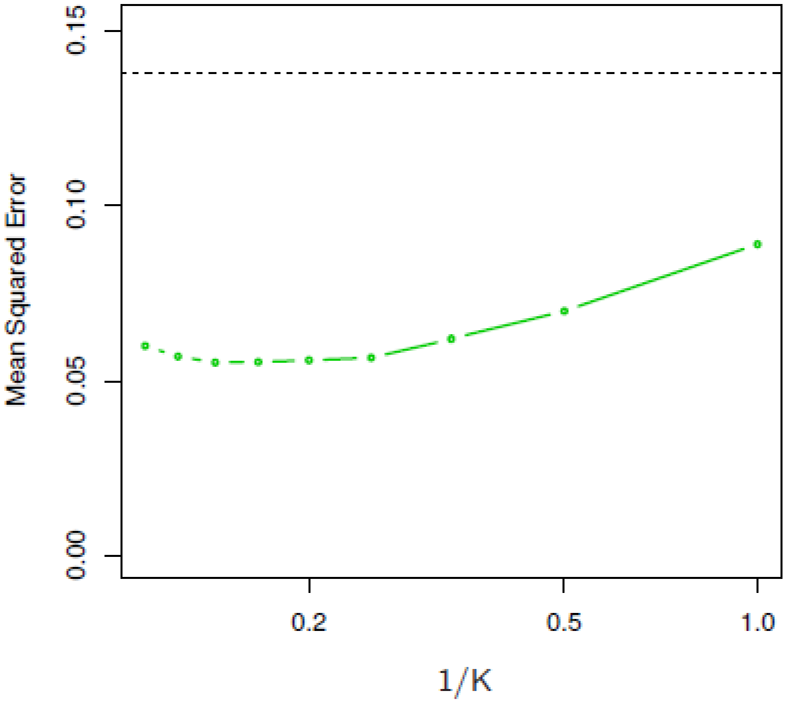
From Chapter 3 on Linear Regression in *An Introduction to Statistical Learning, with Applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani (text available online)



A more non-linear relationship, in black

Predictions of KNN regression with K=1 is in blue and K=9 in red.

N = 100 training data points

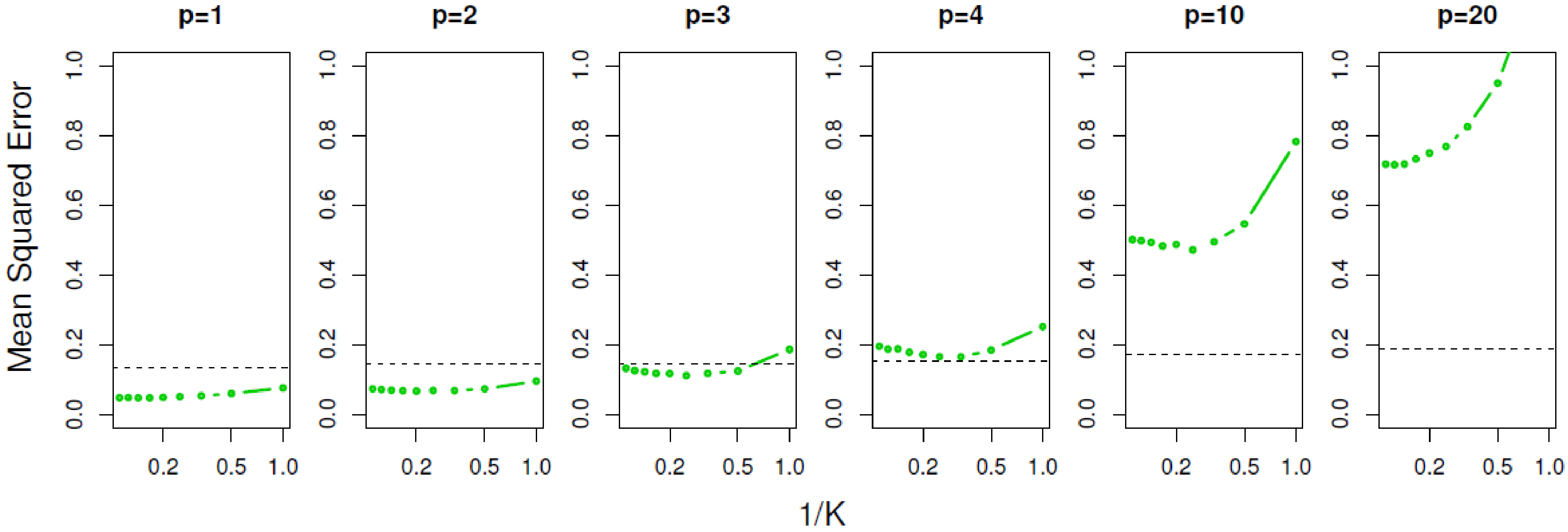


Test mean-squared error (MSE) for the problem on the left

X-axis is 1/K where K is the value used for predictions from KNN regression

Dotted black line is for linear regression

From Chapter 3 on Linear Regression in *An Introduction to Statistical Learning, with Applications in R* (Springer, 2013) with permission from the authors:
G. James, D. Witten, T. Hastie and R. Tibshirani (text available online)



Panels show how the error increases as the number of variables/parameters in the problem increases

Dotted line is MSE for linear regression, green lines are MSE for KNN as a function of 1/K

p=1 corresponds to the previous slide with 1 variable. P=2, 3, ..20 are problems with p variables, where All except the first variable are noise (i.e., y only depends on the first variable).

Note how kNN's performance decreases rapidly as p increases, compared to linear regression

A clear example of a low-variance model (linear regression) outperforming a high-variance model (KNN)

From Chapter 3 on Linear Regression in *An Introduction to Statistical Learning, with Applications in R* (Springer, 2013) with permission from the authors:
G. James, D. Witten, T. Hastie and R. Tibshirani (text available online)

Defining what "best" means

- How do we measure "best"?
  - Best performance on the training data?
    - K = p will be best (i.e., use all variables), e.g., p=10,000
    - So this is not useful in general
  - Performance on the training data will in general be optimistic

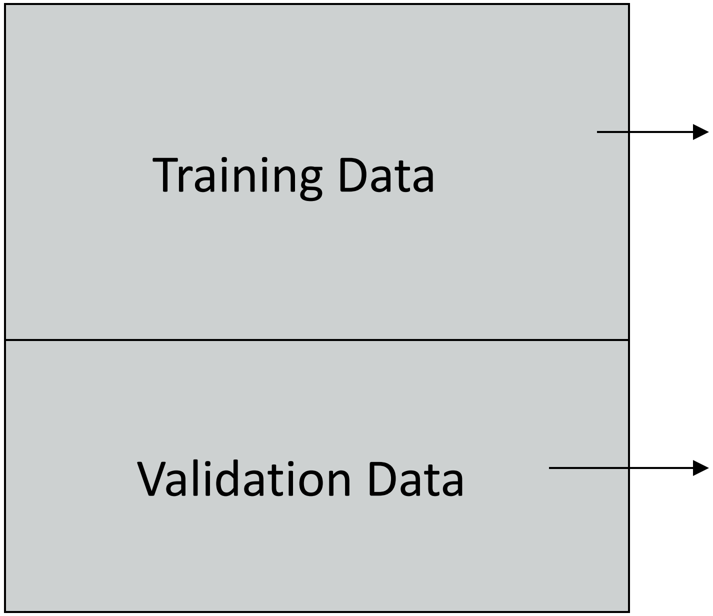
- Practical Alternatives:
  - Measure performance on a single validation set
  - Measure performance using multiple validation sets
    - Cross-validation
  - Add a penalty term to the score function that "corrects" for optimism
    - E.g., "regularized" regression: SSE + λ sum of weights squared

Training Data

Training Data

Use this data to find the best $\underline{\theta}$ for each model $f_k(\underline{x} ; \underline{\theta})$

Validation Data



Training Data

Validation Data

Use this data to find the best $\underline{\theta}$ for each model $f_k(\underline{x} ; \underline{\theta})$

Use this data to

(1) calculate an estimate of $E_k(\underline{\theta})$ for each $f_k(\underline{x} ; \underline{\theta})$ and

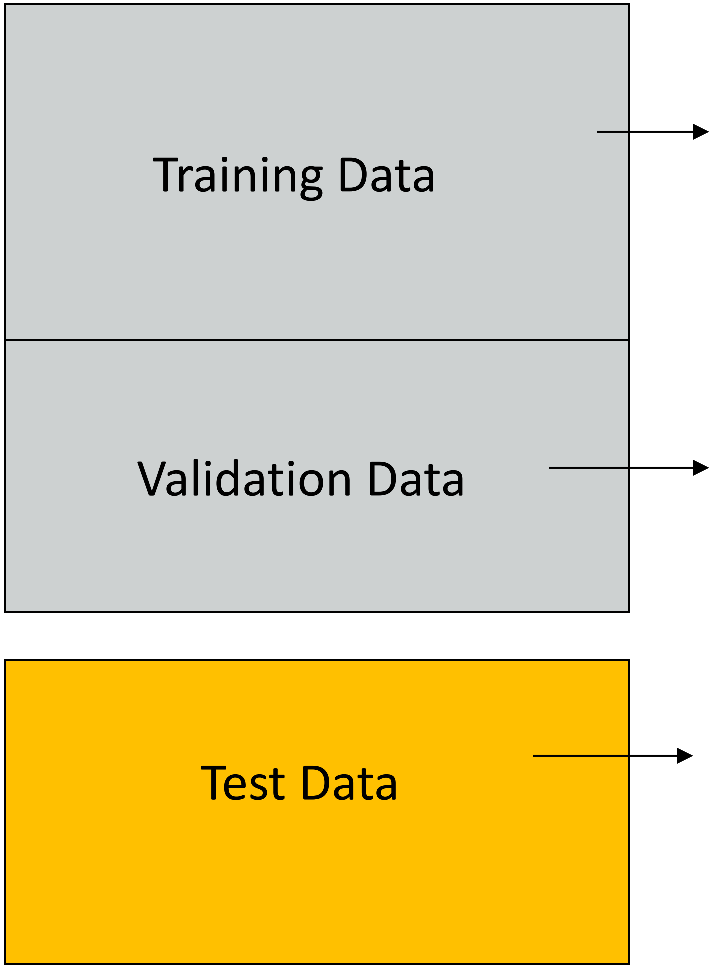
(2) select $k^* = \arg\min_k E_k(\underline{\theta})$

2 different (but related) issues here

1. Finding the function f that minimizes E($\underline{\theta}$) for future data

2. Getting a good estimate of E($\underline{\theta}$), using the chosen function, on future data,
   – e.g., we might have selected the best function f, but our estimate of its performance will be optimistically biased if our estimate of the score uses any of the same data used to fit and select the model.

Test Data



Training Data

Use this data to find the best $\underline{\theta}$ for each model $f_k(\underline{x}\ ;\ \underline{\theta})$

Validation Data

Use this data to

(1) calculate an estimate of $E_k(\underline{\theta})$ for each $f_k(\underline{x}\ ;\ \underline{\theta})$ and

(2) select $k^* = \arg\min_k E_k(\underline{\theta})$

Test Data

Use this data to calculate an unbiased estimate of $E_k(\underline{\theta})$ for the selected model

K-fold Cross-Validation

- In the previous slides we used a single training data set and a single validation data set

- Our results might be sensitive to the particular choice of validation set
  - Particularly for small data sets

- Better approach is K-fold Cross-Validation
  - Divide the data into K disjoint randomly selected subsets (e.g., K = 10)
  - Using each of the K subset as a validation set, with N/K data points
    - Train your models on the other N(K-1)/K data points
    - Evaluate the prediction of each model on the kth validation set
  - For each model average its scores over the K validation sets
  - Select the model with the best average cross-validated score

  - Widely used in practice

Evaluation Methods

Evaluation Methods in General

- When we build a predictive model how can we evaluate the model?
- As mentioned before, the standard approach is to keep a holdout or test data set (that we haven’t used in any aspect of model training or model selection) and evaluate our error or objective function on that, e.g.,
  - Squared error or classification error
- But this is often only part of the story….its important to look at other aspects of performance, particularly if we want to further improve our model
- In the next few slides we will discuss different aspects of evaluating or “diagnosing” model performance
  - These techniques can be very useful in practical applications

Comparing to a Baseline: Classification

Classification: Comparing to a Constant Predictor

Simplest baseline is to always predict the most likely class c* (ignore x)

Easy to show that the average classification error of this strategy is 1 – P(c*)

In binary problems P(c*) can often be quite high,

e.g., probability a random patient does not have cancer ~ 0.99

e.g., probability a random Web page visitor does not click on an ad, ~ 0.9999

It is important to evaluate the accuracy of a classifier relative to this base rate, e.g.,

Relative error reduction = 1 – [E_{test} / (1 – P(c*))],

where E_{test} is the error rate of our classifier

e.g., 1 – P(c*) = 0.01, and $\mathbf{E_{test}}$ = 0.008,

=> Relative error reduction = 1 – (0.008/0.01) = 0.2, i.e., 20%

Whether a reduction of 20% in error rate is useful or not will depend on the application

Classification: Confusion Matrices

Count the pairs of (predicted, actual) class labels in test data set
Patterns in the off-diagonal cells can be informative

		True Class Labels		
		Class 1	Class 2	Class 3
Model’s Predictions	Class 1	400	120	40
	Class 2	2	980	50
	Class 3	5	10	10

Binary Classification: Ranking Metrics

- In many applications we have a set of test items/vectors x, and want to generate a score to rank

- Often rank by P(C = 1 | x) , where C=1 indicates the class of interest
  - And where P(C=1 | x) is produced by our prediction model, e.g., by logistic regression

- Examples:
  - Ranking loan applicants by likelihood of repaying the loan
  - Ranking Web users by likelihood of clicking on an ad
  - Ranking patients by likelihood that they need surgery
  - And so on.

- In practice we might then select the top K-ranked items, e.g., to award a loan to
  - Sidenote: there are algorithms that “learn to rank” directly, not discussed here

Binary Classification: Ranking Metrics

- To evaluate using a ranking metric we do the following
  - take a test set with N data vectors x
  - compute a score for each item , say P(C = 1 | x), using our prediction model
  - sort the N items from largest to smallest score

- This gives us 2 lists, each of length N
  - A list of predictions, with decreasing score values
  - A corresponding list of "ground truth" values, 0's and 1's for binary class labels

- A variety of useful evaluation metrics can be computed based on these 2 lists
  - Precision/recall
  - Receiver-operating characteristics
  - Lift curves
  - And so on.

Ranking Terminology

		True Labels	
		Positive	Negative
Model’s Predictions	Positive	TP True positive	FP False positive
	Negative	FN False negative	TN True negative

Precision = TP/ (TP + FP) = ratio of correct positives predicted to total positive predicted

Recall = TP / (TP + FN) = ratio of correct positives predicted to actual number of positives

Typically will get high precision for low recall, and low precision at high recall

Binary Classification: Simple Example of Precision and Recall

- Test set with 10 items, binary labels, 5 from each class (TP + FN = 5)

Scores from the Model	True Label
0.97	1
0.95	1
0.93	0
0.81	1
0.55	0
0.28	1
0.17	0
0.15	1
0.10	0
0.03	0

Binary Classification: Simple Example of Precision and Recall

- Test set with 10 items, binary labels, 5 from each class (TP + FN = 5)

Scores from the Model	True Label
0.97	1
0.95	1
0.93	0
0.81	1
0.55	0
0.28	1
0.17	0
0.15	1
0.10	0
0.03	0

Threshold

TP = 2, FP = 0
Precision = TP/(TP+FP) = 100%
Recall = TP/(TP + FN) = 2/5 = 40%

Binary Classification: Simple Example of Precision and Recall

- Test set with 10 items, binary labels, 5 from each class (TP + FN = 5)

Scores from the Model	True Label
0.97	1
0.95	1
0.93	0
0.81	1
0.55	0
0.28	1
0.17	0
0.15	1
0.10	0
0.03	0

Threshold

TP = 3, FP = 2
Precision = TP/(TP+FP) = 60%
Recall = TP/(TP + FN) = 3/5 = 60%

Binary Classification: Simple Example of Precision and Recall

- Test set with 10 items, binary labels, 5 from each class (TP + FN = 5)

Scores from the Model	True Label
0.97	1
0.95	1
0.93	0
0.81	1
0.55	0
0.28	1
0.17	0
0.15	1
0.10	0
0.03	0

Threshold

TP = 5, FP = 8
Precision = TP/(TP+FP) = 62%
Recall = TP/(TP + FN) = 5/5 = 100%

How are Precision and Recall used?

- Precision @ K is often used as a metric, where K is the top K items or the top K% of the sorted prediction list
  - E.g., useful for information retrieval

- Precision-recall curves can be plotted for varying threshold
  - Will return to this when we discuss text classification in later lectures

ROC Plots

		True Labels	
		Positive	Negative
Model’s Predictions	Positive	TP True positive	FP False positive
	Negative	FN False negative	TN True negative

TPR = True Positive Rate = TP / (TP + FN)
= ratio of correct positives predicted to actual number of positives
(same as recall, sensitivity, hit rate)

FPR = False Positive Rate
= FP / (FP + TN) = ratio of incorrect negatives predicted to actual number of negatives
(same as false alarm rate)

Receiver Operating Characteristic: plots TPR versus FPR as threshold varies

As we decrease our threshold, both the TPR and FPR will increase, both ending at [1, 1]

ROC for Binary Classification

- Test set with 10 items, binary labels, 5 from each class (TP + FN = 5)

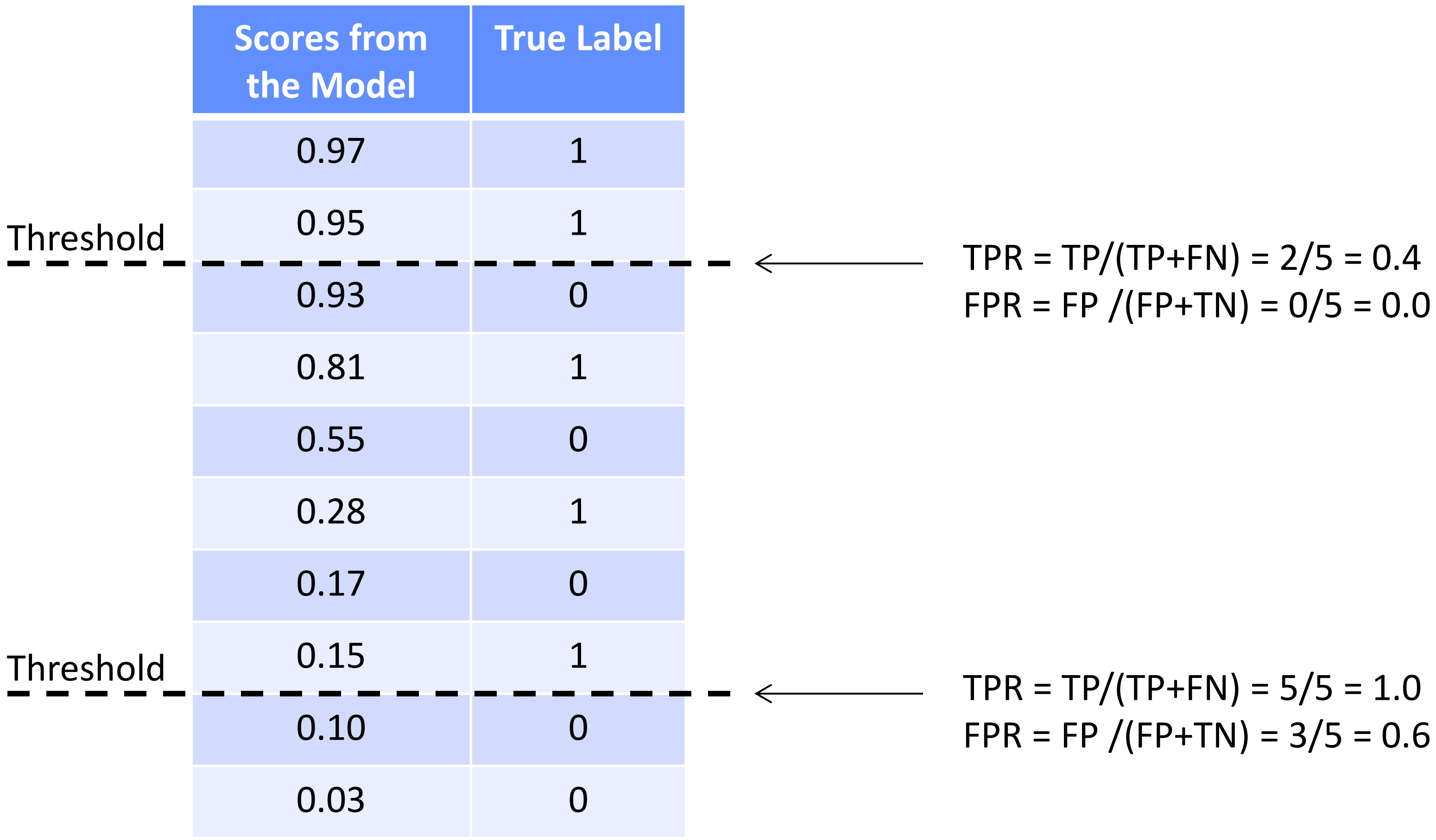
Scores from the Model	True Label
0.97	1
0.95	1
0.93	0
0.81	1
0.55	0
0.28	1
0.17	0
0.15	1
0.10	0
0.03	0

Threshold

TPR = TP/(TP+FN) = 2/5 = 0.4
FPR = FP /(FP+TN) = 0/5 = 0.0

ROC for Binary Classification

- Test set with 10 items, binary labels, 5 from each class (TP + FN = 5)



Scores from the Model	True Label
0.97	1
0.95	1
0.93	0
0.81	1
0.55	0
0.28	1
0.17	0
0.15	1
0.10	0
0.03	0

Threshold

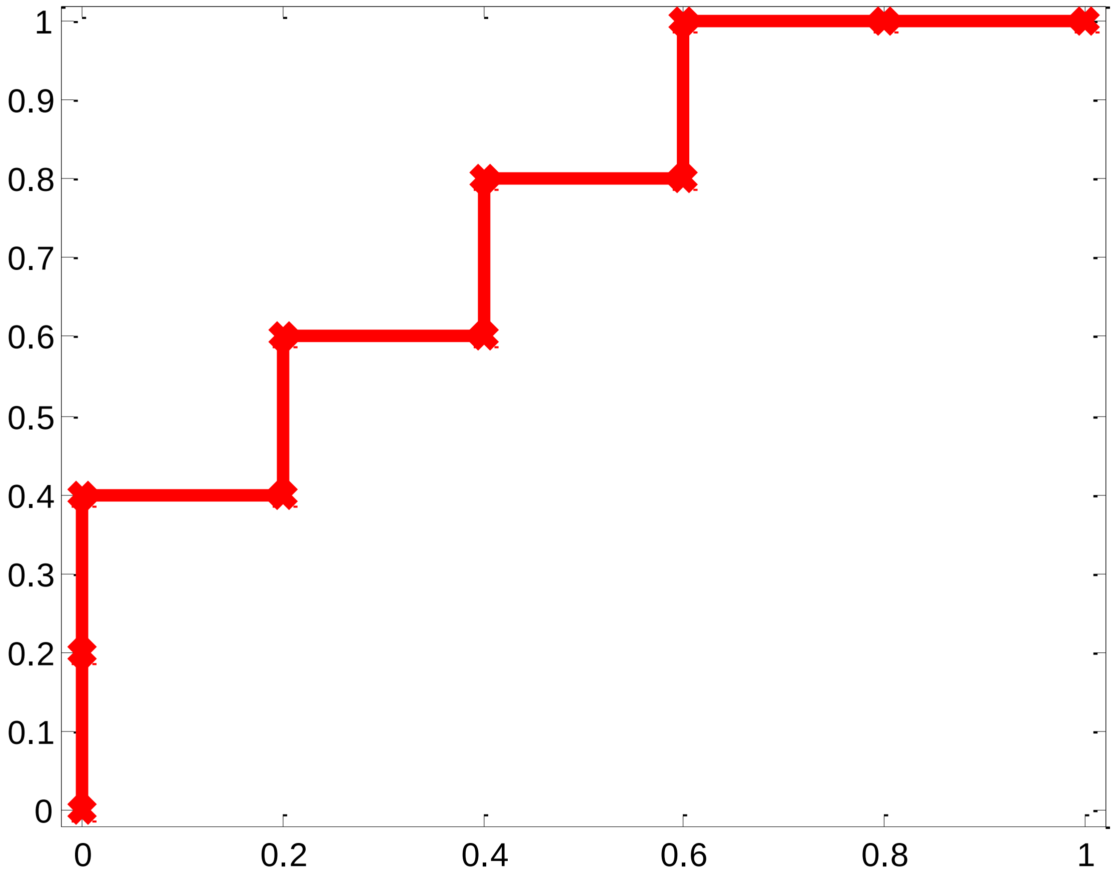
TPR = TP/(TP+FN) = 2/5 = 0.4
FPR = FP /(FP+TN) = 0/5 = 0.0

Threshold

TPR = TP/(TP+FN) = 5/5 = 1.0
FPR = FP /(FP+TN) = 3/5 = 0.6

ROC Plot for Example on Previous Slide

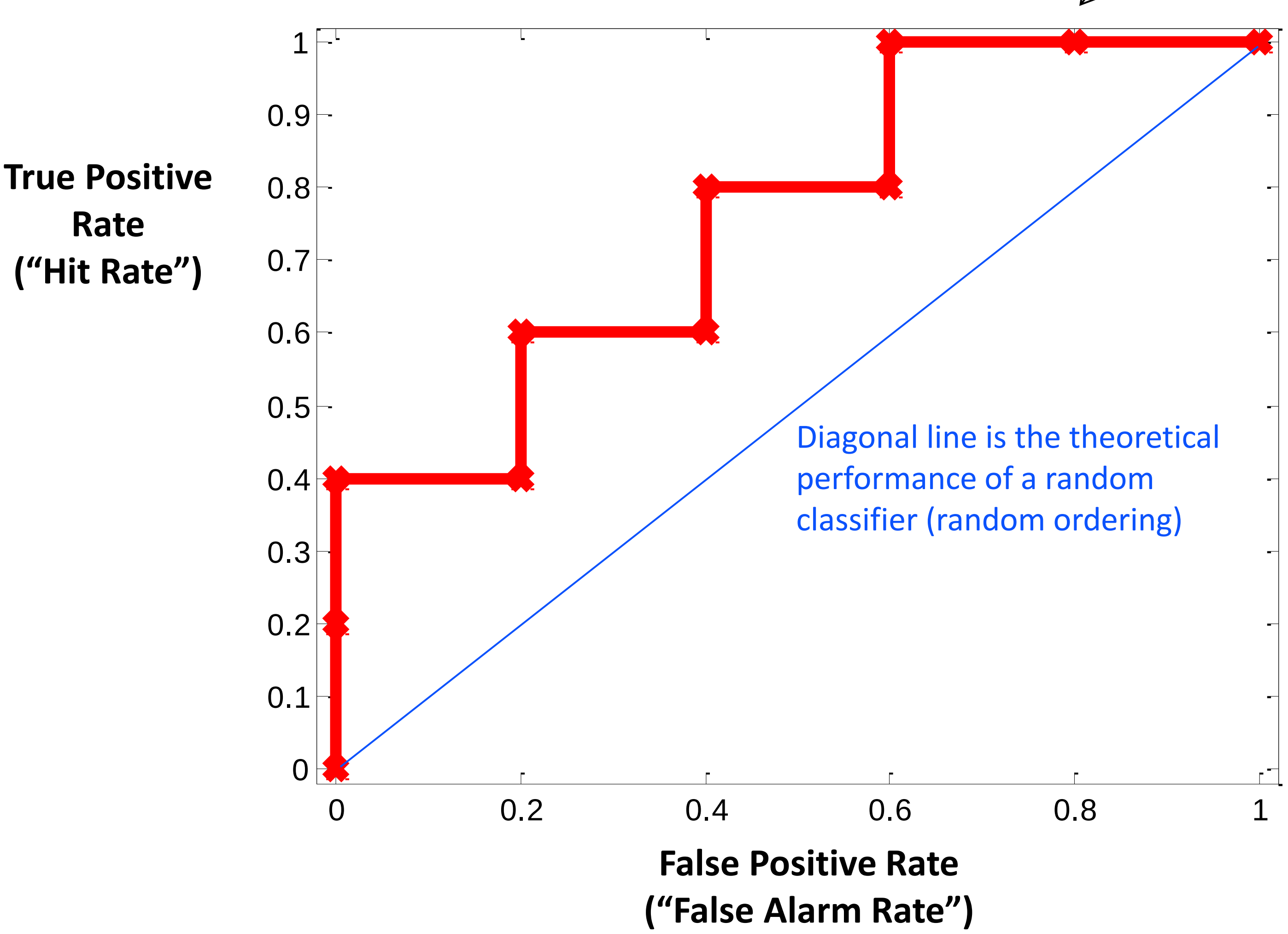
Each point corresponds to a particular operating threshold (N+1 points)



True Positive Rate ("Hit Rate")

False Positive Rate ("False Alarm Rate")

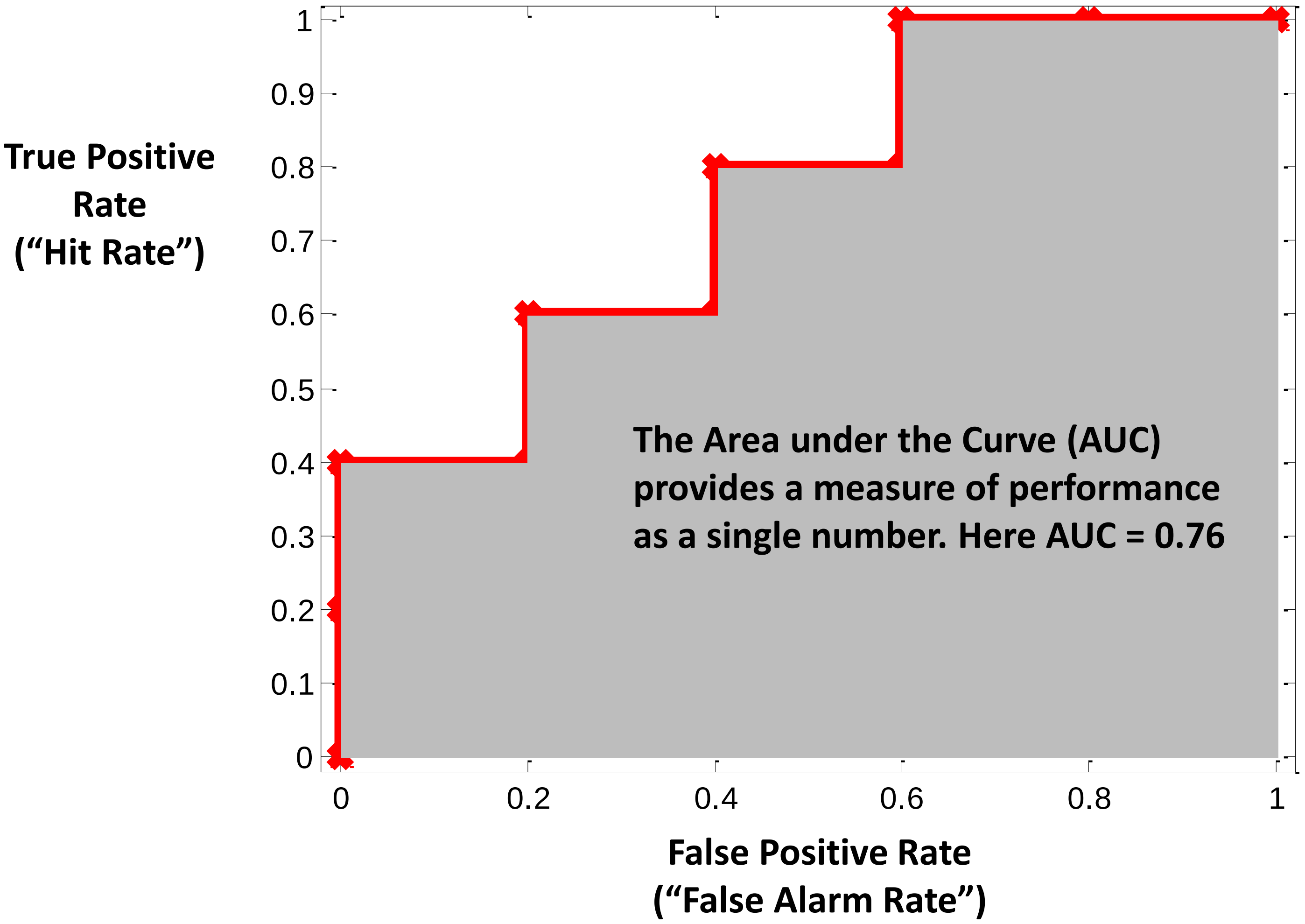
ROC Plot for Example on Previous Slide



Each point corresponds to a particular operating threshold (N+1 points)

Diagonal line is the theoretical performance of a random classifier (random ordering)

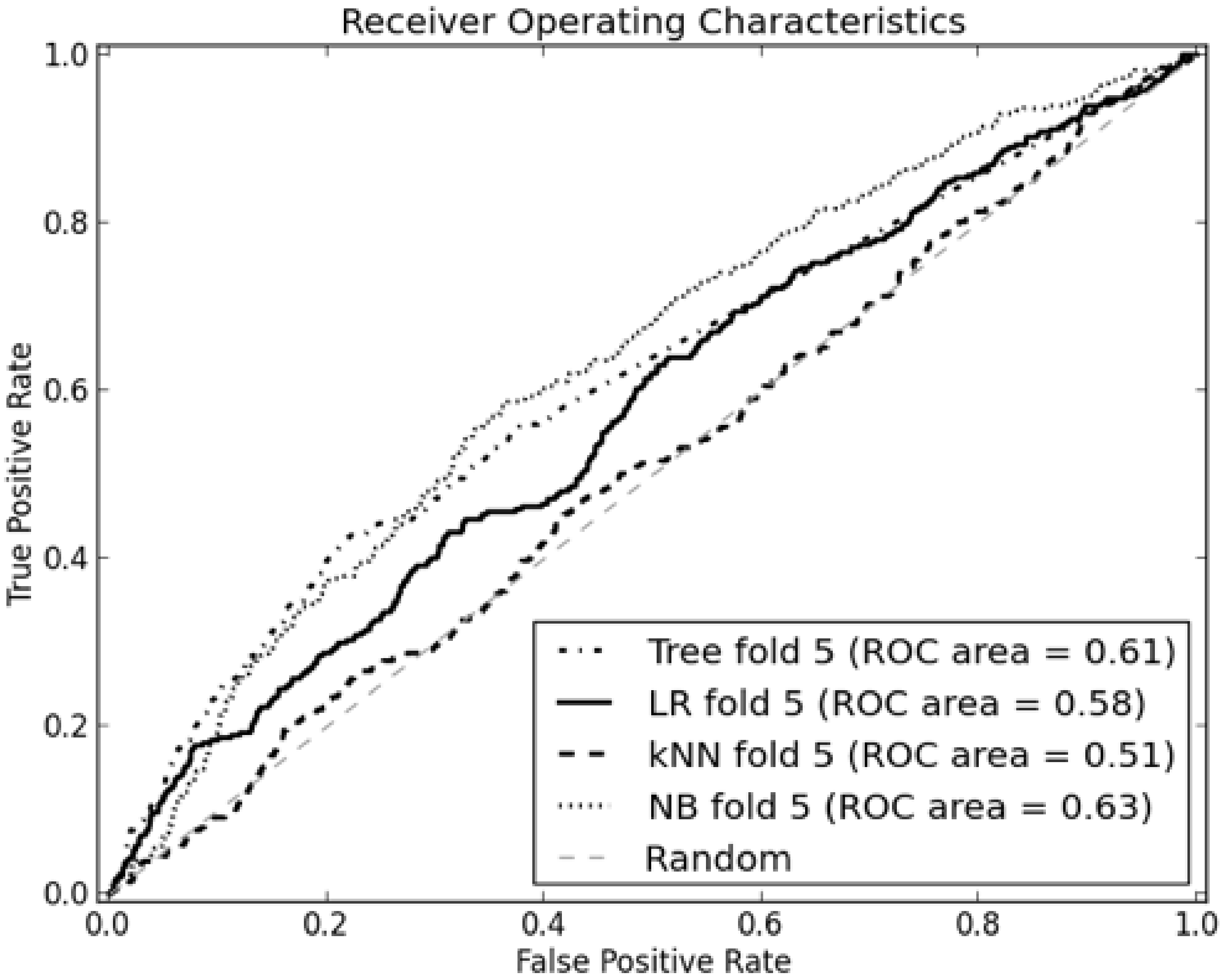
ROC Plot for Example on Previous Slide



Each point corresponds to a particular operating threshold (N+1 points)

The Area under the Curve (AUC) provides a measure of performance as a single number. Here AUC = 0.76

Example of an Actual ROC



From Chapter 8: Visualizing Model Performance, in Data Science for Business (O Reilly, 2013), with permission from the authors, F. Provost and T. Fawcett

Binary Classification: Lift Curves

Sort test examples by their predicted score

For a particular threshold compute

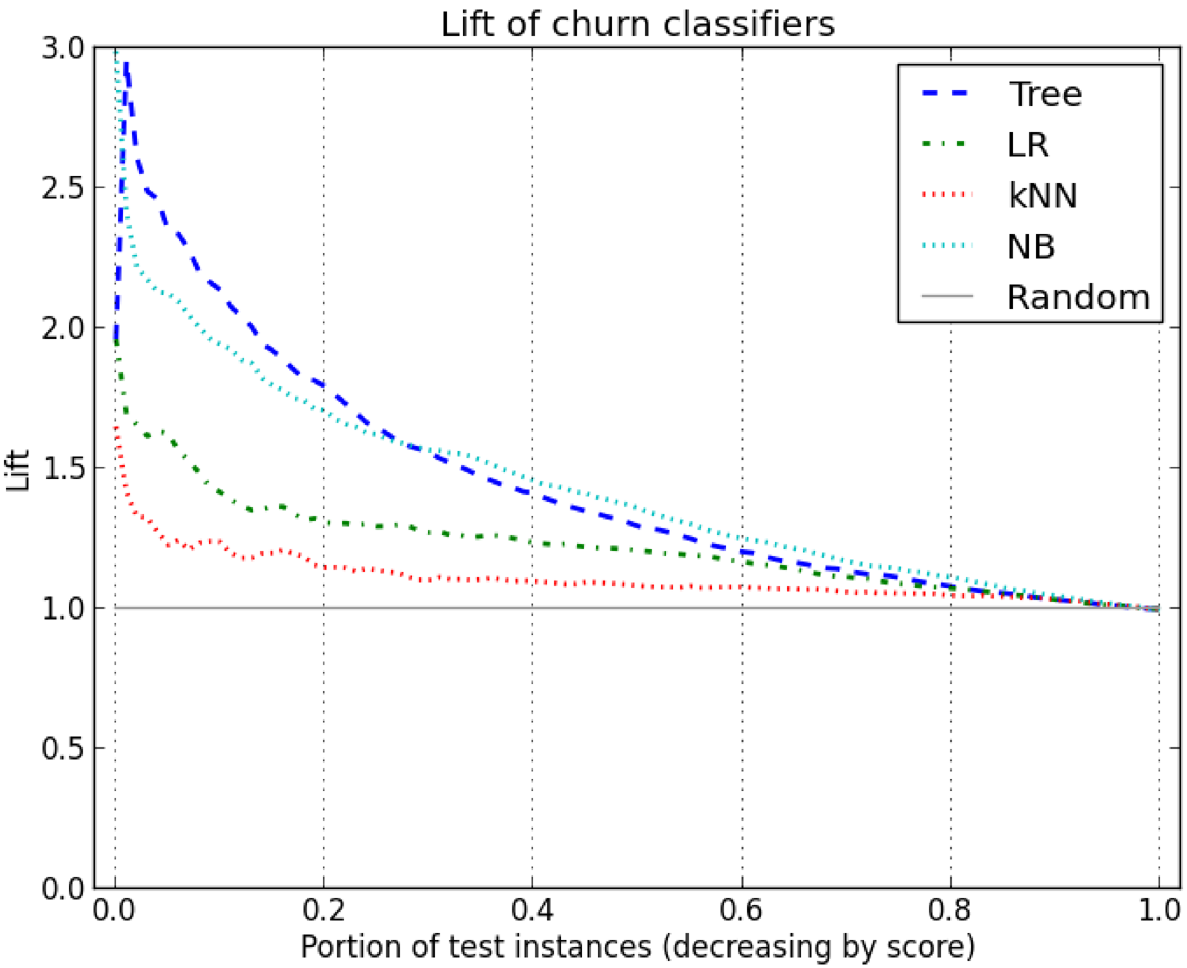
(1) NP = number of actual positive examples detected by the model

(2) NR = number of actual positive examples that would be detected by random ordering

Lift = NP/NR

Lift curve = Lift as a function of number of examples above the threshold, as the threshold is varied

Expect that good models will start with high lift (and will eventually decay to 1)



From Chapter 8: Visualizing Model Performance, in Data Science for Business (O Reilly, 2013), with permission from the authors, F. Provost and T. Fawcett

Costs and Benefits

	Truth	
Predictions	TP	FP
	FN	TN

Consider a binary classification problem where the costs and benefits of misclassification are not symmetric

Example:

A bank uses a predictive model to predict which applicants will repay a loan or not. It wants to compute the expected profit for different thresholds on the score P(C=1 | x).

An applicant who gets a loan and repays it earns the company $1000

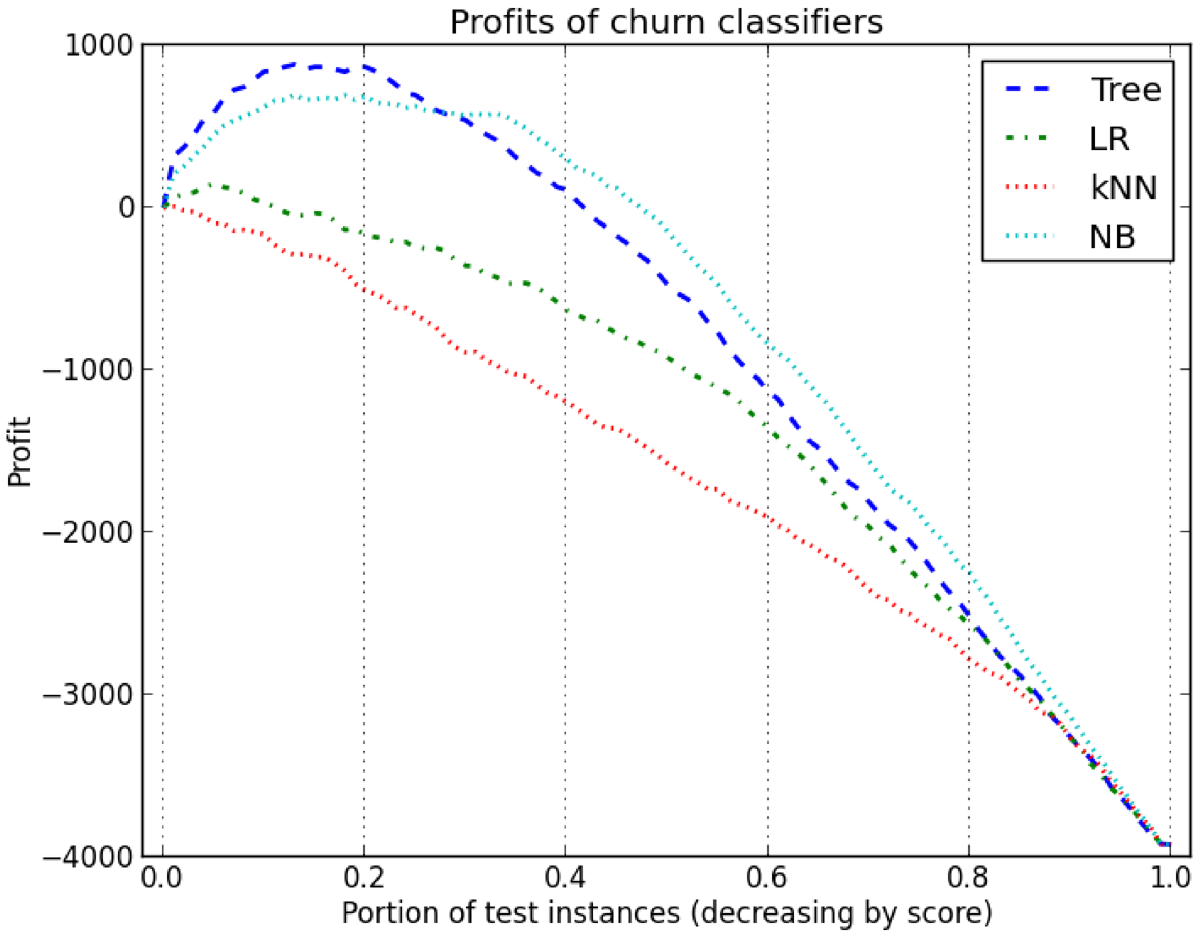
An applicant who gets a loan and doesn't repay it costs the company $5000

Empirical benefit measured on a test data set = N(TP) * $1000 – N(FP) * $5000

The numbers N(TP) and N(FP) vary with the threshold selected, so we can plot the empirical benefit as a function of percentage of applicants selected

Example of an Empirical "Profit Curve"

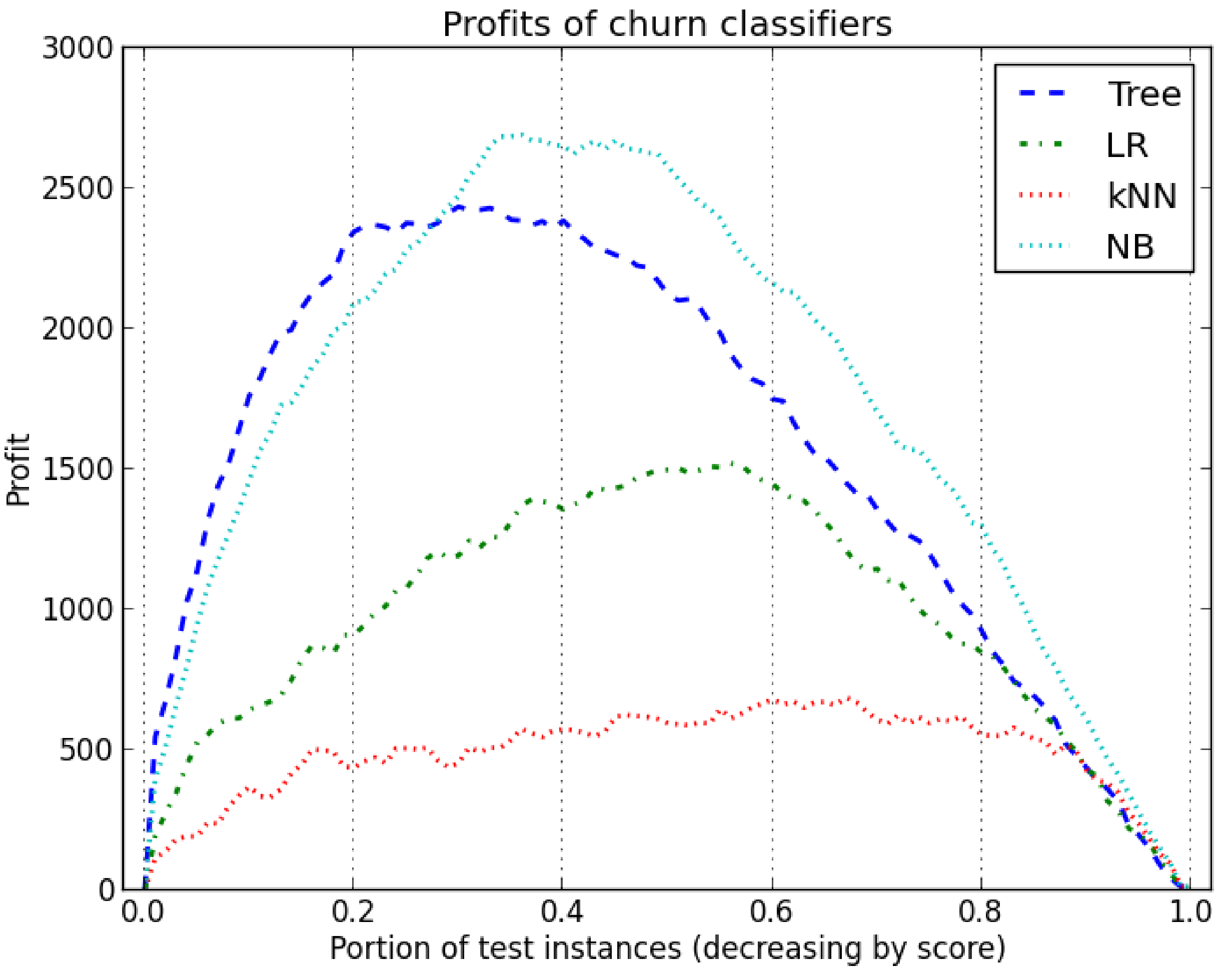
9:1 benefit/cost ratio



From Chapter 8: Visualizing Model Performance, in Data Science for Business (O Reilly, 2013), with permission from the authors, F. Provost and T. Fawcett

Example of an Empirical "Profit Curve"

12:1 benefit/cost ratio
(more lucrative)



From Chapter 8: Visualizing Model Performance, in Data Science for Business (O Reilly, 2013),
with permission from the authors, F. Provost and T. Fawcett

UCIrvine
University of California, Irvine

Calibration

- In addition to ranking we may be interested in how accurate our estimates of P(C=1|x) are,
  - i.e., if the model says P(C=1|x) = 0.9, how accurate is this number?

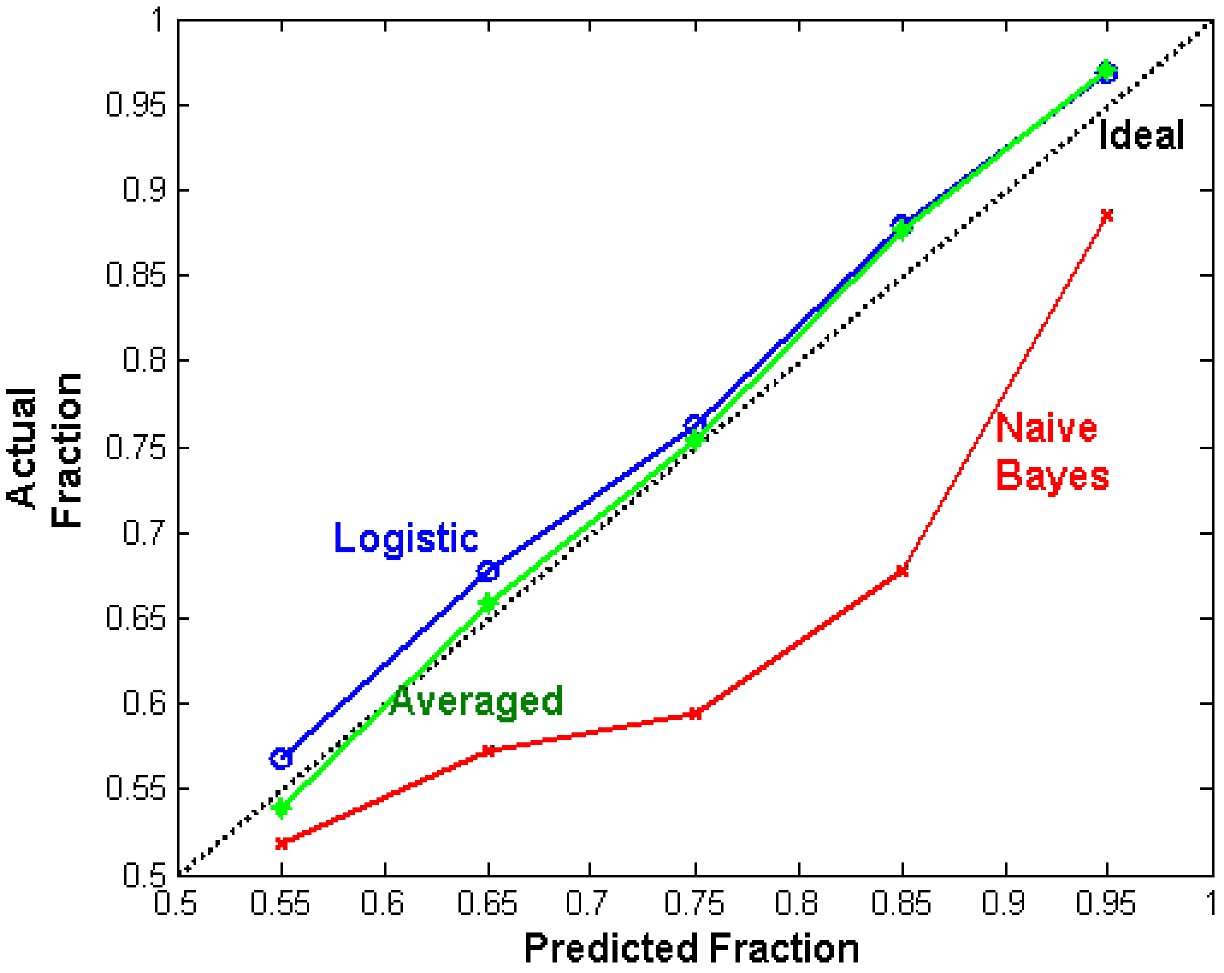
- Calibration:
  - a model is well-calibrated if its probabilistic predictions match real-world empirical frequencies
  - i.e., if a classifier predicts p(c|x) = 0.9 for 100 examples, then on average we would expect about 90 of these examples to belong to class c, and 10 not to.

  - We can empirically estimate calibration curves by binning a classifier's probabilistic predictions, and comparing to the empirical probability
    - i.e., measuring the fraction of test data points that have C=1 for examples predicted to be in that bin

Calibration in Probabilistic Prediction

Example from a real classification problem.

The logistic model tends is often well calibrated and naïve Bayes often is not



Comparing to a Baseline: Regression

Regression: Comparing to a Constant Predictor

Simplest baseline is to predict the mean value of Y (ignore $\underline{x}$) - this is the best "constant predictor" to use if we want to minimize squared error

Squared error of this "predictor" is :

$$\mathbf{Var(y) = \Sigma\, [y - \mu_y]^2}$$

where the y's are in the test set and μ_y is the average value of y in the training data.

$\mathbf{E_{test} = \Sigma\, [y - f(\underline{x}\, ;\, \underline{\alpha})\,]}^2$ = the error of our prediction model on the same test data

Then E_{test} / Var(y) = fraction reduction in variance provided by our model

$\mathbf{R^2 = 1 - [E_{test} / Var(y)\ ]}$

= proportion of variance "explained" by our model, $0 <= R^2 <= 1$

Note that our interpretation of R^2 may depend on the application, e.g.,

- a low R^2 might still be enough to systematically do better than the competition, e.g., online advertising, financial investing

Review of Basic Concepts in Regression and Classification Algorithms

Algorithm = Model + Error Function + Optimization Method

Important point: most learning algorithms are a combination of

Model + Error Function + Optimization Method

- Models:
  - Linear weighted sums of the input variables (linear regression)
  - Non-linear functions of linear weighted sums (logistic regression, neural networks, GLMs)
  - Thresholded functions (decision trees)

- Error functions:
  - Regression: Squared (L2) error, Absolute (L1) error, Robust loss, log-loss/log-likelihood
  - Classification: classification error, margin, log-loss/log-likelihood

- Optimization methods
  - For linear models, squared error loss: solving sets of linear equations in p parameters
    - Tend to scale as $O(N d^2 + d^3)$
  - More complex models
    - Iterative search methods, e.g., gradient-based, 2nd-order (Newton), etc

Example: Multivariate Linear Regression

Task: predict real-valued y, given real-valued vector $\underline{x}$

Model: linear $f(\underline{x}\ ;\ \underline{\alpha}) = \alpha_0 + \sum \alpha_j\, x_j$

Model parameters = $\underline{\alpha} = \{\alpha_0, \alpha_{1,} \ldots\ldots \alpha_p \}$

Error function: e.g., least squares is often used

$$E(\underline{\alpha}) = \sum_i [\underbrace{y(i)}_{\text{target value}} - \underbrace{f(\underline{x}(i)\ ;\ \underline{\alpha})}_{\text{predicted value}}\ ]^2$$

Optimization:

How do we solve the optimization problem, i.e., minimize $E(\underline{\alpha})$?

Note that we can write

$$E(\alpha) = \Sigma_i \, [y(i) - \Sigma \, \alpha_j \, x_j]^2$$

$$= \Sigma_i \, e_i^2$$

$$= \underline{e}' \, \underline{e}$$

$$= (y - X \, \alpha)' \, (y - X \, \alpha)$$

where $\underline{e} = y - X \, \alpha$

y = N x 1 vector
of target values

X = N x (p+1) vector
of input values

(d+1) x 1 vector
of parameter values

$$E(\alpha) = \Sigma e^2 = e' e = (y - X \alpha)' (y - X \alpha)$$

$$= y' y - \alpha' X' y - y' X \alpha + \alpha' X' X \alpha$$

$$= y' y - 2 \alpha' X' y + \alpha' X' X \alpha$$

Taking derivative of $E(\alpha)$ with respect to the components of α gives….

$dE/d \alpha = -2 X' y + 2 X' X \alpha$

Set this to 0 to find the extremum (minimum) of E as a function of α …

Set to 0 to find the extremum (minimum) of E as a function of α …

$\Rightarrow$ $- 2 X' y + 2 X' X \alpha = 0$

$\Rightarrow$ $X' X \alpha = X' y$ (known in statistics as the Normal Equations)

Letting $X' X = C$, and $X' y = b$,

we have $C \alpha = b$, i.e., a set of linear equations

We could solve this directly, e.g., by matrix inversion

$$\alpha = C^{-1} b = (X' X)^{-1} X' y$$

↑

X' X = d by d matrix

Solving for the α's

- Problem is equivalent to inverting the X’ X matrix (dimension d x d)
  - Inverse does not exist if matrix is not of full rank
    - E.g., if 1 column is a linear combination of another (collinearity)
    - Note that X’X is closely related to the covariance of the X data
      - So we are in trouble if 2 or more variables are perfectly correlated
    - Numerical problems can also occur if variables are almost collinear

- Equivalent to solving a system of d linear equations
  - Many good numerical methods for doing this, e.g.,
    - Gaussian elimination, LU decomposition, etc
  - These are numerically more stable than direct inversion
  - Time complexity = $O(N\ d^2 + d^3)$

- Alternative: gradient descent
  - Compute gradient and move downhill

Comments on Multivariate Linear Regression

- Prediction of the model is a linear function of the parameters

- Error function: quadratic in predictions and parameters
  - ⇒ Derivative of error is linear in the parameters
  - ⇒ Leads to a linear algebra optimization problem, i.e., $C \theta = b$

- Model structure is simple….
  - d-1 dimensional hyperplane in p-dimensions
  - Linear weights => interpretability

- Often useful as a baseline model

- Note: even if it’s the wrong model for the data (e.g., a poor fit) it can still be useful for prediction

Limitations of Linear Regression

- True relationship of X and Y might be non-linear
  - Suggests generalizations to non-linear models
- Complexity:
  - $O(N\ d^2 + d^3)$ - problematic for large d
- Correlation/collinearity among the X variables
  - Can cause numerical instability (C may be ill-conditioned)
  - Problems in interpretability (identifiability)
- Includes all variables in the model…
  - But what if d=1000 and only 3 variables are actually related to Y?

Non-Linear Terms, but Linear in Parameters

- We can add additional polynomial terms in our equations, e.g., all “2nd order” terms

$$f(\underline{x}\ ;\ \underline{\alpha}) = \alpha_0 + \sum \alpha_j x_j + \sum \beta_{ij} x_i x_j$$

- Note that it is a non-linear functional form, but it is linear in the parameters (so still referred to as “linear regression”)
  - We can just treat the $x_i\, x_j$ terms as additional fixed inputs
  - In fact we can add in any non-linear input functions!, e.g.

$$f(\underline{x}\ ;\ \underline{\alpha}) = \alpha_0 + \sum \alpha_j f_j(\underline{x})$$

Comments:

- Same linear algebra optimization problem (with more parameters)
- Number of parameters has now exploded -> greater chance of overfitting
  - Ideally would like to select only the useful quadratic terms
  - Can generalize this idea to higher-order interactions

Models that are Non-Linear in both Inputs and Parameters

- We can generalize further to models that are nonlinear in all aspects

$$f(\underline{x}\,;\,\underline{\theta}) = \alpha_0 + \sum \alpha_k\, g_k\big(\beta_{k0} + \sum \beta_{kj}\, x_j\,\big)$$

where the g’s are non-linear functions with fixed functional forms.

In machine learning this is called a neural network

In statistics this might be referred to as a generalized linear model or projection-pursuit regression

For almost any score function of interest, e.g., squared error, the score function is a non-linear function of the parameters.

Closed form (analytical) solutions are rare.

Thus, we have a multivariate non-linear optimization problem
(which may be quite difficult!)

Optimization in the Non-Linear Case

- We seek the minimum of a function in d dimensions, where d is the number of parameters (d could be large!)

- There are a multitude of heuristic search techniques
  - Steepest descent (follow the gradient)
  - Newton methods (use 2nd derivative information)
  - Conjugate gradient
  - Line search
  - Stochastic search
  - And more….

- Two cases:
  - Convex (nice -> means a single global optimum)
  - Non-convex (multiple local optima => need multiple starts)

Other non-linear models

- Splines
  - "patch" together low-order polynomials over different parts of the x-space
  - Works well in low dimensions, less well in higher dimensions

- Memory-based models

  $y' = \sum w_{(x',x)} y$, where y's are from the training data
  $w_{(x',x)}$ = function of distance of x from x'

- Local linear regression

  $y' = \alpha_0 + \sum \alpha_j x_j$, where the alpha's are fit at prediction time just to the (y,x) pairs that are close to x'

Predicting a Binary Variable

What if our y variable is binary?, e.g., is this search result relevant or not

A nice theoretical result: if we train any model $f(\underline{x}(i) \, ; \underline{\alpha})$, with some parameters , to minimize squared error with binary targets y,

$$E(\underline{\alpha}) = \sum_i [y(i) - f(\underline{x}(i) \, ; \underline{\alpha}) \,]^2$$

then the learned function $f(\underline{x}(i) \, ; \underline{\alpha})$ will try to approximate $P(y=1 \mid x)$

This is why for example the outputs of a model like a neural network (which is just a particular choice for our f function), will often seem to be more like probabilities between 0 and 1, than “hard predictions” 0 and 1.

So in principle we can use any model we like for f, train it on binary targets, and it will learn to produce estimates of $P(y=1 \mid \underline{x})$ which we can use for ranking, etc

Logistic Regression

Candidate model:

$$f(x \mid \alpha) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \ldots. + \alpha_d x_d$$

Problem: this can give predictions that are negative, > 1, etc.

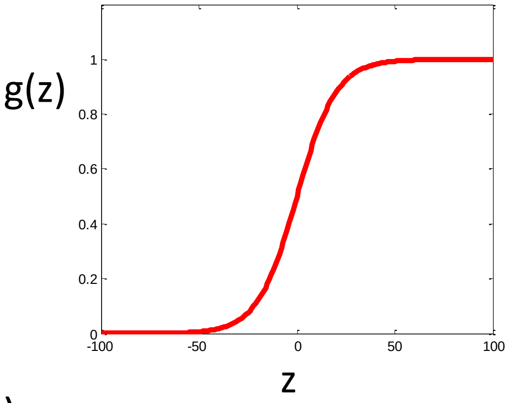
Logistic Regression

Candidate model:

$$f(x \mid \alpha) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_d x_d$$

Problem: this can give predictions that are negative, > 1, etc.

Example of a Logistic Function



Better approach:

$$f(x \mid \alpha) = g\,(\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_d x_d\,)$$

where $g(z) = 1 / [\,1 + e^{-z}\,]$

This is known as a logistic regression model in statistics

(a special case of a generalized linear model, or GLM)

A linear model where the linear function is transformed to lie between 0 and 1

Log-Loss Function

Log-Loss function, for binary targets y_i

Let $f_i = f(\underline{x}(i) ; \underline{\alpha})$, the prediction of our model for input $\underline{x}(i)$,
where for example f is our logistic model. Assume $0 < f_i < 1$.

$$L(\underline{\alpha}) = \sum_{i:\, y = 1} \log f_i + \sum_{i:\, y = 0} \log (1 - f_i)$$

This function $L(\underline{\alpha})$ encourages the f function to be close to 1 for positive examples, $y_i = 1$ and close to 0 for negative examples, $y_i = 0$.

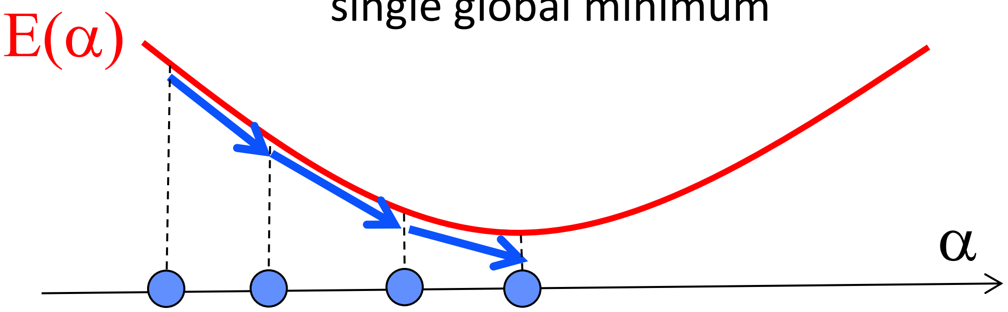
Since the f_i predictions are between 0 and 1, the $\log f_i$ and $\log (1 - f_i)$ terms will all be negative and the highest possible value for the log-loss function is 0. This is achieved only if the model predicts 1 for every positive example and 0 for every negative example.

We can turn this into an error function by defining $E(\underline{\alpha}) = - L(\underline{\alpha})$

Logistic Regression

How can we maximize the log-loss L($\underline{\alpha}$) as a function of $\underline{\alpha}$?

- no direct solution (as with linear regression)
- but the loss function is convex, so relatively easy to optimize
- can use iterative gradient methods



Convex function with a single global minimum

Optimizing the Log-Loss Function in Logistic Regression

Classical statistical approach

- use iterative 2nd order gradient (Newton) methods
- known as "iterative reweighted least-squares" (IRLS)
- requires solution of d x d linear set of equations at each iteration, O(d^3)
- this will be very slow in high dimensions

More modern approaches for large data sets

- avoid 2nd order gradient methods (which is where the d^3 comes from)
- use 1st order gradient information instead, O(d)
- can get further speeds up by using sparse (noisy) gradient by updating the parameters based on small subsets of the data rather than the full data

Logistic Regression

The output of the logistic function f provides direct estimates of p(y = 1 | x)

- this is very useful for ranking

Logistic regression is widely used in practice – the “workhorse” of predictive modeling

- Document classification (emails)
- Click prediction for Web users
- Credit scoring
- Fraud detection
- And many more….
- Google, Yahoo!, Microsoft, eBay, Amazon, Yelp, Linkedin, Facebook, Experian, etc

Building Predictive Models with Many Input Variables

- In many problems we have a very large number of input variables d

- We know that many of these are probably not relevant to predicting y
  - Likely to cause overfitting if included in the model
  - Will also make the fitted model harder to interpret

- Strategies for reducing the number of variables
  - Feature selection
    - Either based on individual feature predictive ability or integrated with the learning algorithm, e.g., via forward or backward search
  - Dimension reduction methods
    - Linear or non-linear projections: but may be suboptimal since they are only modeling x and do not take into account the dependence of y on x
  - Algorithms that automatically do feature selection
    - E.g., classification and regression trees
  - Regularization
    - Add a penalty term to the error function to drive weights to 0

Regularization

- Modified error function with a penalty term

$$E_\lambda(\underline{\alpha}) = E(\underline{\alpha}) + \lambda \Sigma \alpha_j^{\ 2}$$

e.g., $E_\lambda(\underline{\alpha}) = \sum_i [y(i) - f(\underline{x}(i) \,;\, \underline{\alpha})\,]^2 + \lambda \Sigma \alpha_j^{\ 2}$

- The second term is for "regularization"
  - When we minimize -> encourages keeping the α_j's near 0
  - Size of λ is important: can optimize value of λ via cross-validation

- L1 regularization

$$E_\lambda(\underline{\alpha}) = \sum_i [y(i) - f(\underline{x}(i) \,;\, \underline{\alpha})\,]^2 + \lambda \Sigma \mid \alpha_j \mid$$

Basis of popular "Lasso" method,

e.g., see Rob Tibshirani's page on lasso methods: http://www stat.stanford.edu/~tibs/lasso.html