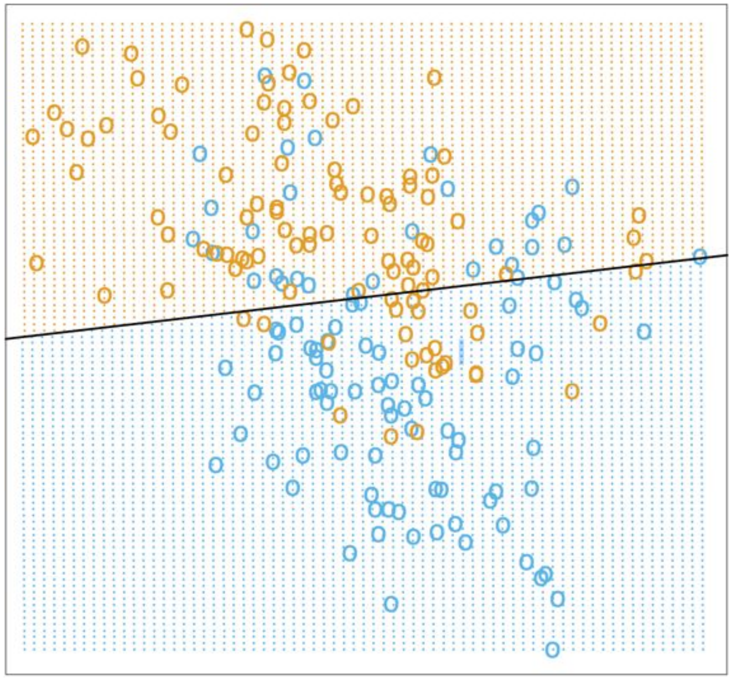
Statistical Decision Theory

Jeremy Irvin and
Daniel Spokoyny

Created from Elements of Statistical Learning (Hastie, Tibshirani, Friedman)

Two Basic Classifiers

- Just as we did in logistic regression, we can learn a linear decision boundary to perform binary classification.



- It seems like a linear assumption is too rigid. Or are errors on our predictions unavoidable?

Two Basic Classifiers

- The errors that we make by assuming a linear decision boundary of course depends on the specific training set we are using:
  - in none of these models have we specified where the data itself comes from.

- Let’s examine two scenarios. The training data in each class were generated from:
  - bivariate Gaussians with uncorrelated components (variance matrix identity) and distinct means.
  - a mixture of 10 low-variance Gaussians, with the means themselves distributed as Gaussian.

Two Basic Classifiers

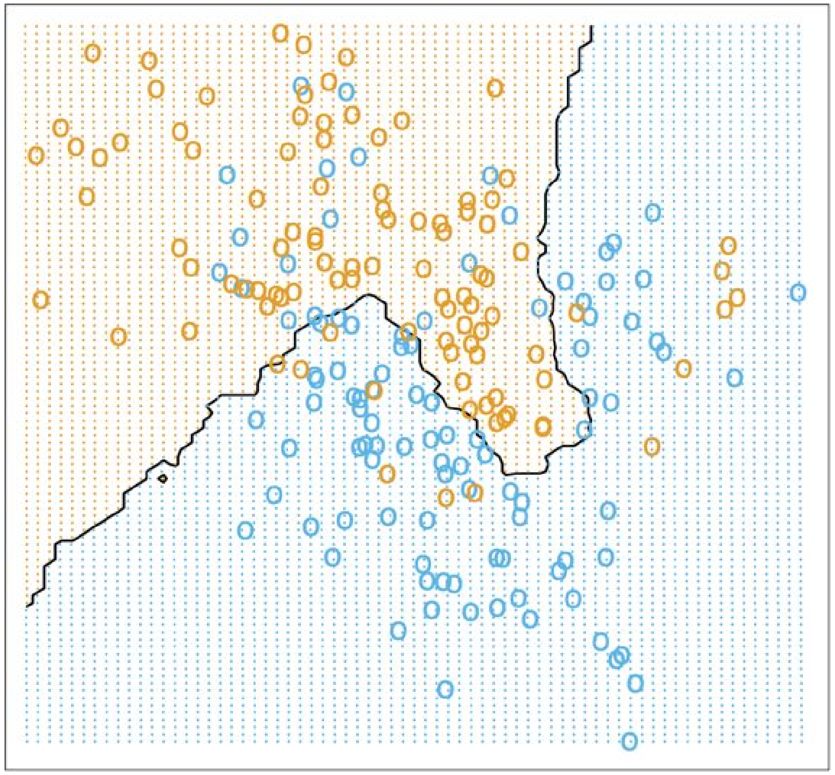
- Think of a mixture of Gaussians in the "generative" sense:
  - Generate a discrete random variable that determines which of the 10 distributions to generate (sample) from
  - Then generate from that chosen distribution

- If the data comes from one Gaussian per class, linear decision boundary is optimal.

- For tightly clustered Gaussians, a linear decision boundary is not optimal - optimal will most likely be linear and disjoint (and therefore difficult to learn).

k-Nearest Neighbors

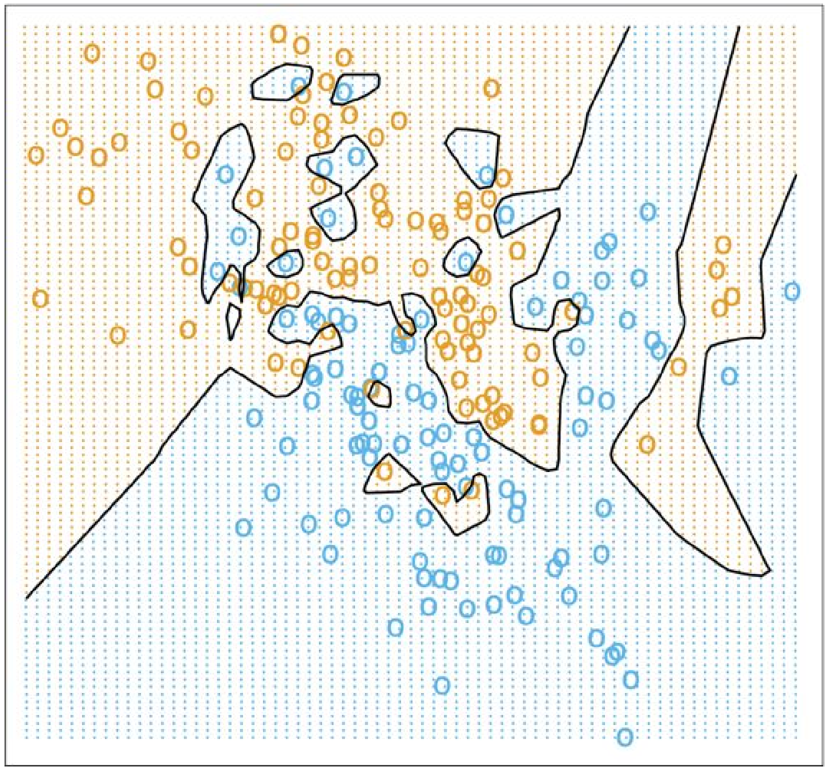
- Can do nearest neighbor methods using majority vote (15-NN):



- Seems much better - but in fact it’s not necessarily a good model. Why?

k-Nearest Neighbors

- This is the decision boundary generated using 1-NN:



- This is a perfect decision boundary for our training set. Why not always use this?

Bias and Variance

- A linear decision boundary is smooth and stable (small changes to our training set won't affect the line), but it relies heavily on the linearity assumption.
  - Low variance, high bias

- k-NN doesn't make any assumptions about the data, and can adapt to it well, but any local region is very susceptible to any change in the training set.
  - High variance, low bias

Statistical Decision Theory

- Let's generalize our original learning formulation:

- Let X denote a random variable which takes on the input values in our training set, and Y a random variable which takes on output values in our training set.

- We want to find h to minimize the value $L(Y, h(X))$ for some loss function L (over the inputs).

- Put another way, we want to discover the joint distribution of the random variables to find the optimal h.

Statistical Decision Theory

- Take the loss function (as before) to be squared loss. Call $\mathcal{T}$ the training set. Then the expected prediction error for h over the training set $\mathcal{T}$ is

$$EPE(h) = \mathbb{E}_{\mathcal{T}}[Y - h(X)]^2$$

- We hope to minimize this error. Turns out we can minimize it pointwise (ie, minimize it for each training example individually):

$$h(x) = \mathbb{E}[Y|X = x]$$

- This is known as the regression function.
- So the best prediction of Y at $X = x$ is the conditional mean when "best" is measured by average squared error.

Statistical Decision Theory

- k-NN in fact attempts to estimate this conditional mean.

- At any input x, the k-NN model yields

$$h(x) = \text{Ave}(y_i | x_i \in N_k(x))$$

Two estimations:

- The expectation is approximated by averaging over sample data.
- Conditioning at a single point is relaxed to condition on a region close to the point.

Statistical Decision Theory

- As the size *N* of our training set increases, these estimations become more and more accurate.
- The points in a neighborhood of x are close to x.
- As the number of neighbors *k* increases, the average will stabilize.
- In fact, it can be shown that if $N, k \to \infty$ with $k/N \to 0$ (the size of the training set increases much faster than the number of neighbors), then

$$h(x) \to \mathbb{E}[Y|X = x]$$

Statistical Decision Theory

- So it seems like we’ve found a universal approximator of this mean, and thus an optimal classifier in this general formulation.

- However, in practice, we often cannot get large enough samples for this approximation to yield good results.

- Additionally, if we know the structure of the data (such as linearity), models with this innate structure will be more stable (but this structure somehow needs to be discovered beforehand).

- Also, as the dimension of the input space becomes large, so does the k-NN neighborhood (the curse of dimensionality), causing the rate of convergence to greatly decrease.

Statistical Decision Theory

- Linear regression similarly approximates this conditional expectation by using the functional model assumption to pool over values of the input space.

- So least squares in this framework amounts to replacing this expectation with averages over the training data, like k-NN.

- Here’s how the two models differ however:
  - least squares assumes h is well approximated by a globally linear function.
  - k-NN assumes h is well approximated by a locally constant function.

Bias-Variance Decomposition

- We can actually express the expected prediction error (using squared loss) as a decomposition into variance and squared bias (here MSE is mean squared error):

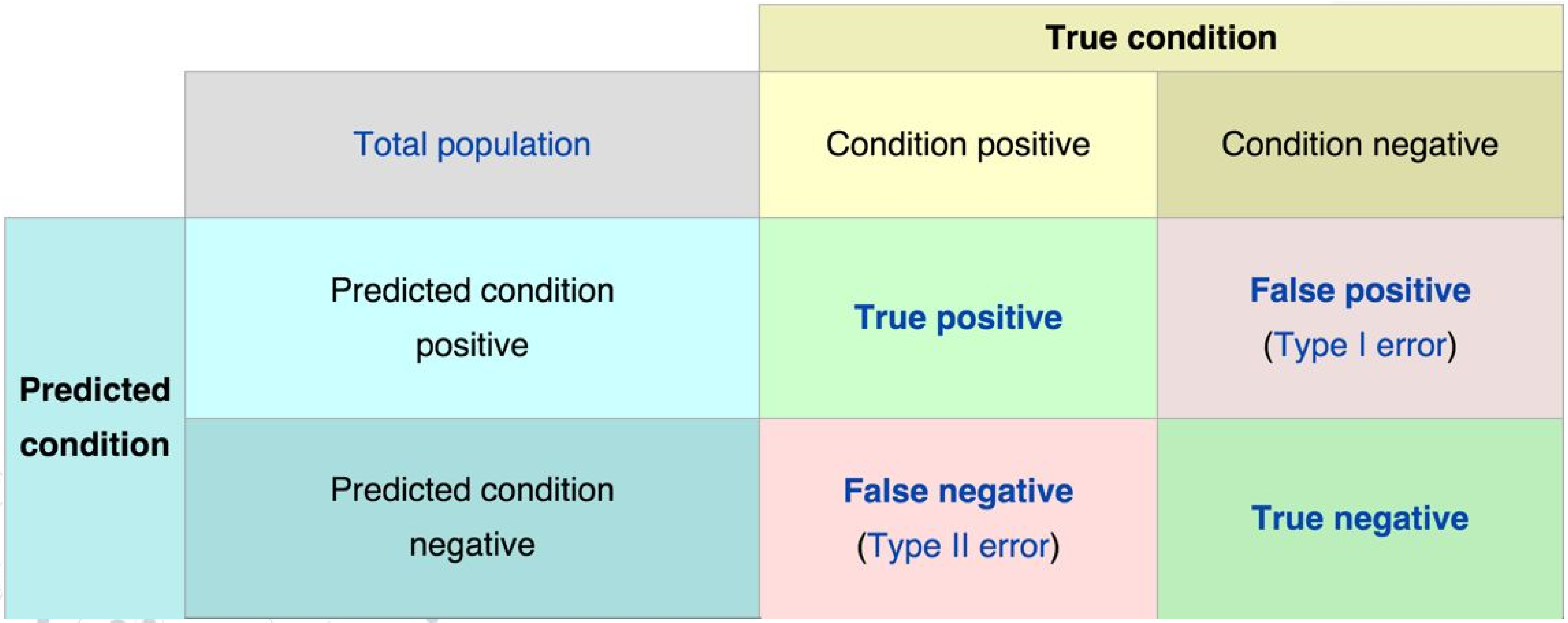
$$\begin{aligned}\mathrm{MSE}(x_0) &= \mathbb{E}_{\mathcal{T}}[\hat{y}_0 - f(x_0)]^2 \\ &= \mathbb{E}_{\mathcal{T}}[\hat{y}_0 - \mathbb{E}_{\mathcal{T}}(\hat{y}_0)]^2 + [\mathbb{E}_{\mathcal{T}}(\hat{y}_0) - f(x_0)]^2 \\ &= \mathrm{Var}_{\mathcal{T}}(\hat{y}_0) + \mathrm{Bias}^2(\hat{y}_0)\end{aligned}$$

- This is known as the <u>bias-variance</u> decomposition.

- This can be used to show (theoretically) the effect of bias and variance on the performance of the model.
  - See <u>Elements of Statistical Learning</u> for more details

Confusion Matrix

- Suppose we are performing binary classification.
- The following is known as the confusion matrix:



		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**	**False positive** (Type I error)
	Predicted condition negative	**False negative** (Type II error)	**True negative**

Precision vs. Recall

- Precision is the number of true positives divided by the total number of positives:

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Recall is the number of true positives divided by the total number of correctly classified points.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- Intuitively, precision is the ability of the classifier not to label as positive a sample that is negative.
- Recall is the ability of the classifier to find all the positive samples.

F1 Score

- One commonly used method of determining the quality of a binary classification model is to use the F1 Score, defined as the harmonic mean of precision and recall:

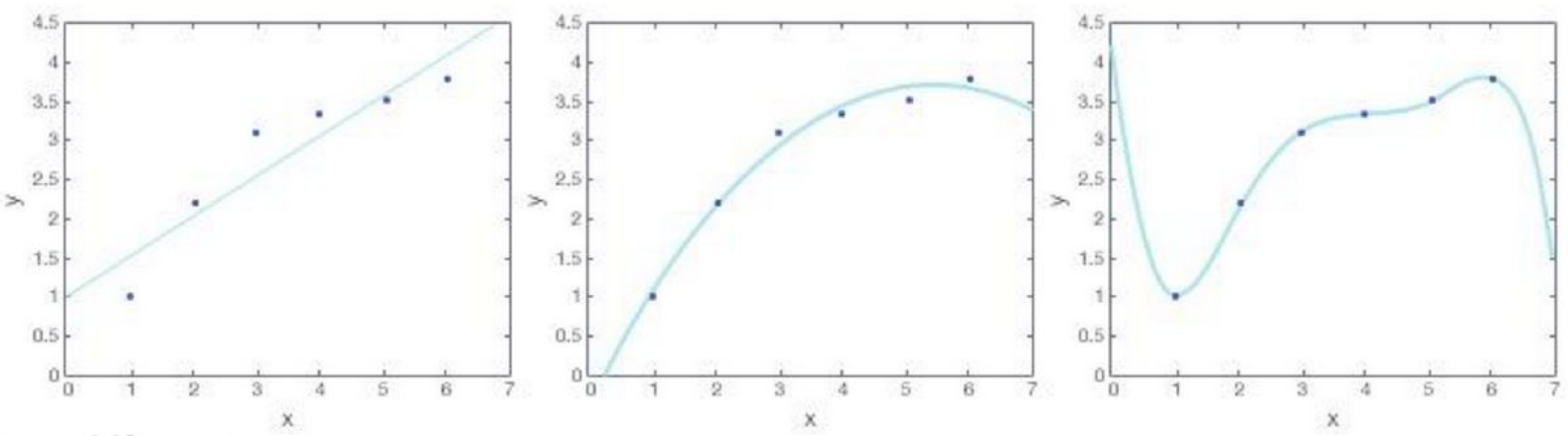
$$F_1 = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

- The best value is 1, the worst is 0.

Overfitting and Underfitting

- We have been discussing ways to evaluate your model.

- Two very common problems with a model are models which overfit and underfit.



Cross-Validation

- In practice, you are given data (let’s say in the supervised setting - so data with labels).

- You hope to build a model and test the model.

- Typically, the data is split into parts - a training set, a test set, and a cross-validation set.

1. The model is first learned using the *training* set.
2. The best performing model (tuning/ choosing hyperparameters) is determined using the *validation* set.
3. The evaluation of the fully trained model is performed using the *test* set (no tuning at this point can occur).

Why separate validation and test sets? To prevent overfitting.

What Just Happened?

- Two Basic Classifiers (linear/ k-NN)
- Bias / Variance Tradeoff and Decomposition
- Confusion Table and F1 Score
- Overfitting and Underfitting
- Cross-Validation