Classification, k-Nearest Neighbors.

W. Wang[1]

[1]Department of Mathematics
University of Houston

MATH 4323

Classification Task.

Supervised learning problems can be further divided into regression and classification problems.

The *Wage* data involves predicting a continuous (else known as quantitative), output value $\Longrightarrow$ **regression** task.

In certain cases we wish to predict a categorical (else known as qualitative) output value $\Longrightarrow$ **classification** task.

Example.

- A person arrives at the emergency room with some symptoms that could possibly be attributed to one of three medical conditions. Which of the three conditions does the individual have?
- An online banking service must be able to determine whether or not a transaction being performed on the site is fraudulent, on the basis of the user's IP address, past transaction history, and so forth.

Stock market data: Classification Task.

Example. *Smarket* data set (from ISLR library) contains the daily movements in the Standard & Poor's 500 (S&P) stock index over a 5-year period between 2001 and 2005.

Goal:

- given the past 5 days' %-changes in the index (predictors);
- predict if index will increase or decrease today (outcome).

Hence, we don't predict a numerical value, but only whether stock market will go up or down $\Longrightarrow$ categorical response, with two categories $\Longrightarrow$ classification task.

The classification problems

Here the response variable Y is qualitative or categorical — e.g. email is one of C = (spam; good email), credit rating scale C = {A, B, C, D,E, F}.

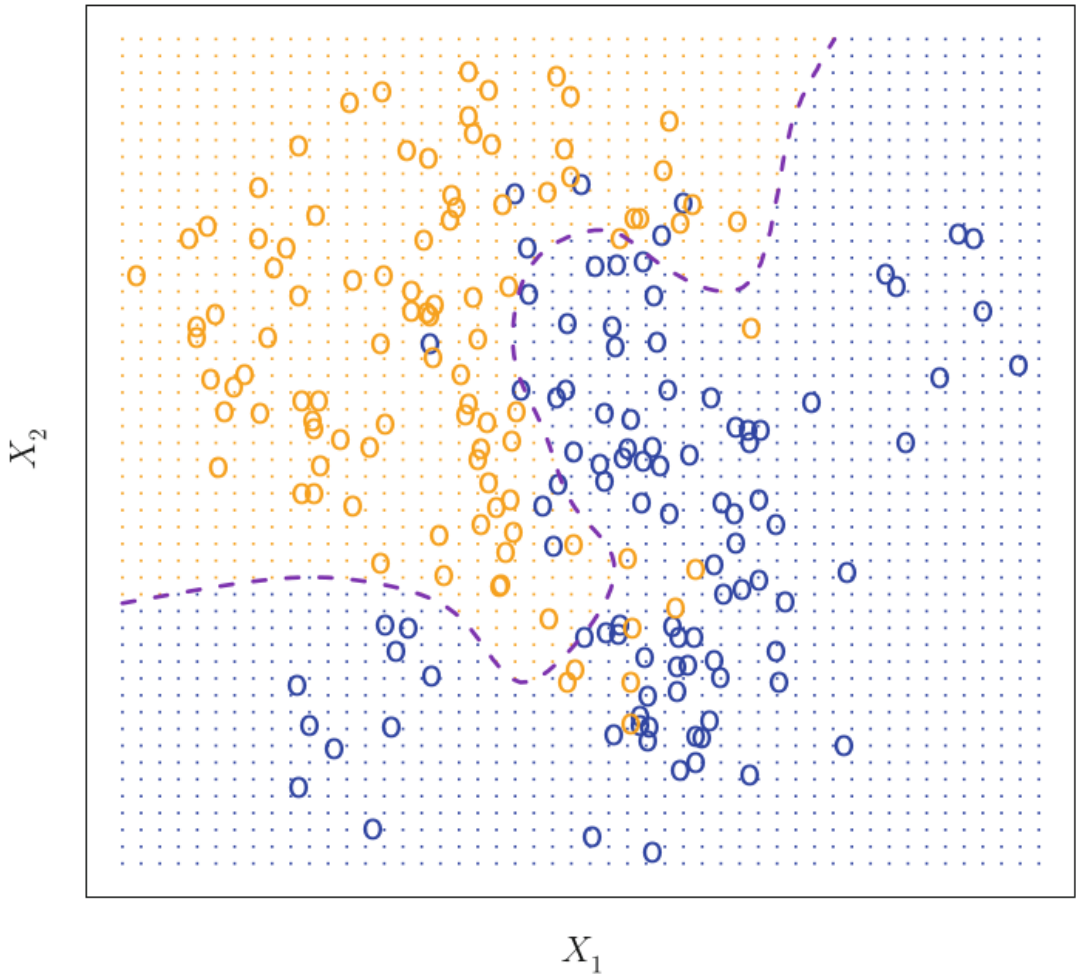
Our goal is:

- Build a classifier $C(X)$ that assigns a class label from C to a future unlabeled observation x.
- Assess the uncertainty in each classification.
- Understand the roles of different predictors among $x = (x1, ..., x_p)^T$.

Classification: Simulated example.

Example. Simulated data set contains:

- two groups (classes) of observations - blue and orange,
- with $n = 100$ observations in each group.
- each observation characterized by $p = 2$ variables - X_1 and X_2.



Goal: classify an observation with predictor values (x_1, x_2) into blue or orange group.

Classification: Main Idea.

In task of classifying an observation, we are generally given:

- predictor values $x_0 = (x_1, \ldots, x_p)$ for the observation to be classified,
- a set of potential classes $\{1, 2 \ldots, C\}$ to which the observation may be classified.

Denoting Y - true class of the observation, we need to calculate

$$P(Y = j \mid X = x_0), \; j = 1, \ldots, C$$

and clearly, $\sum_{j=1}^{j=C} P(Y = j \mid X = x_0) = 1$. Then, identify the most likely class c of the observation, given predictor values x_0:

$$c = \underset{j}{argmax} \; P(Y = j \mid X = x_0)$$

Classification: Main Idea.

Note that

$$P(Y = j \mid X = x_0),\ j = 1, \ldots, C$$

is a conditional probability: it is the probability that $Y = j$, given the observed predictor vector x_0.

This Bayes classifier at x is

$$C(x) = j, \quad \text{if} \quad p_j(x) = max\{p_1(x), \cdots,\ p_J(x)\}.$$

In a two-class problem where there are only two possible response values, say class 1 or class 2, the Bayes classifier corresponds to predicting class 1 if $P(Y = j \mid X = x_0) > 0.5$, and class 2 otherwise.

Classification: Main Idea.

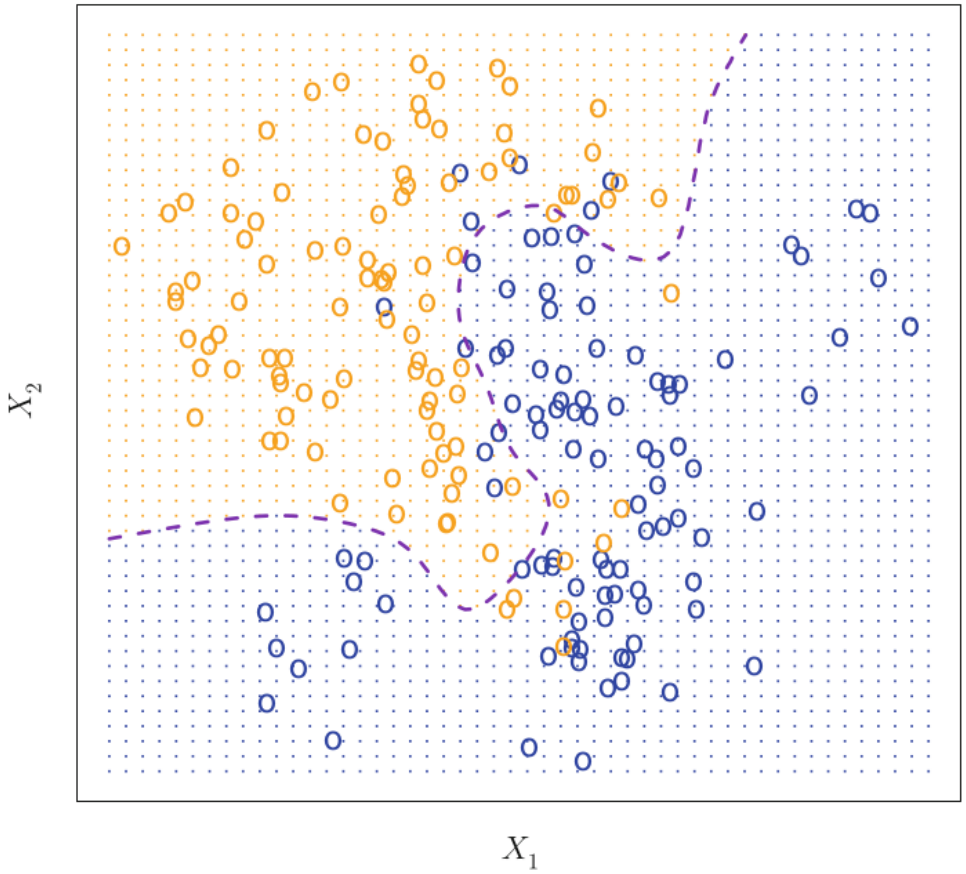
Example (cont'd). In our simulated example:

- Predictor values are: $x_0 = (x_1, x_2)$.
- Potential classes: $\{1, 2\}$, or $\{\text{"}\textit{Blue}\text{"}, \text{"}\textit{Orange}\text{"}\}$.
- Need to calculate: $\begin{cases} P(Y = \text{"}\textit{Blue}\text{"} \mid X = x_0), \\ P(Y = \text{"}\textit{Orange}\text{"} \mid X = x_0). \end{cases}$
- Figure out whether "Blue" or "Orange" yields highest probability.

Example. With the S&P 500 data:

- Predictor vals: $x_0 = (\textit{Lag1}_0, \textit{Lag2}_0, \textit{Lag3}_0, \textit{Lag4}_0, \textit{Lag5}_0, \textit{Volume})$.
- Potential classes: $\{1, 2\}$, or $\{\text{"}\textit{Down}\text{"}, \text{"}\textit{Up}\text{"}\}$.
- Need to calculate: $\begin{cases} P(Y = \text{"}\textit{Down}\text{"} \mid X = x_0), \\ P(Y = \text{"}\textit{Up}\text{"} \mid X = x_0). \end{cases}$
- Figure out whether "Down" or "Up" yields highest probability.

Classification: Simulated example.



The purple dashed line represents the Bayes decision boundary. The orange background grid indicates the region in which a test observation will be assigned to the orange class, and the blue background grid indicates the region in which a test observation will be assigned to the blue class.

Classification: K-Nearest Neighbors (KNN).

In real data setting, the exact conditional probability distribution $P(Y \mid X)$ is unknown, so computing the Bayes classifier is impossible and needs to be estimated.

One probability estimation method is **K-nearest neighbors (KNN)** classifier. Given

- positive integer K, and
- a test observation $x_0 = (x_1, \ldots, x_p)$,

Classification: K-Nearest Neighbors (KNN).

The KNN classifier proceeds to

1. Identify the K points in the training data that are closest to x_0 (this set of points is denoted N_0).
2. Estimate the conditional probability for class j as

   $Pr(Y = j \mid X = x_0) = \{$fraction of points in N_0 whose class equals j$\}$

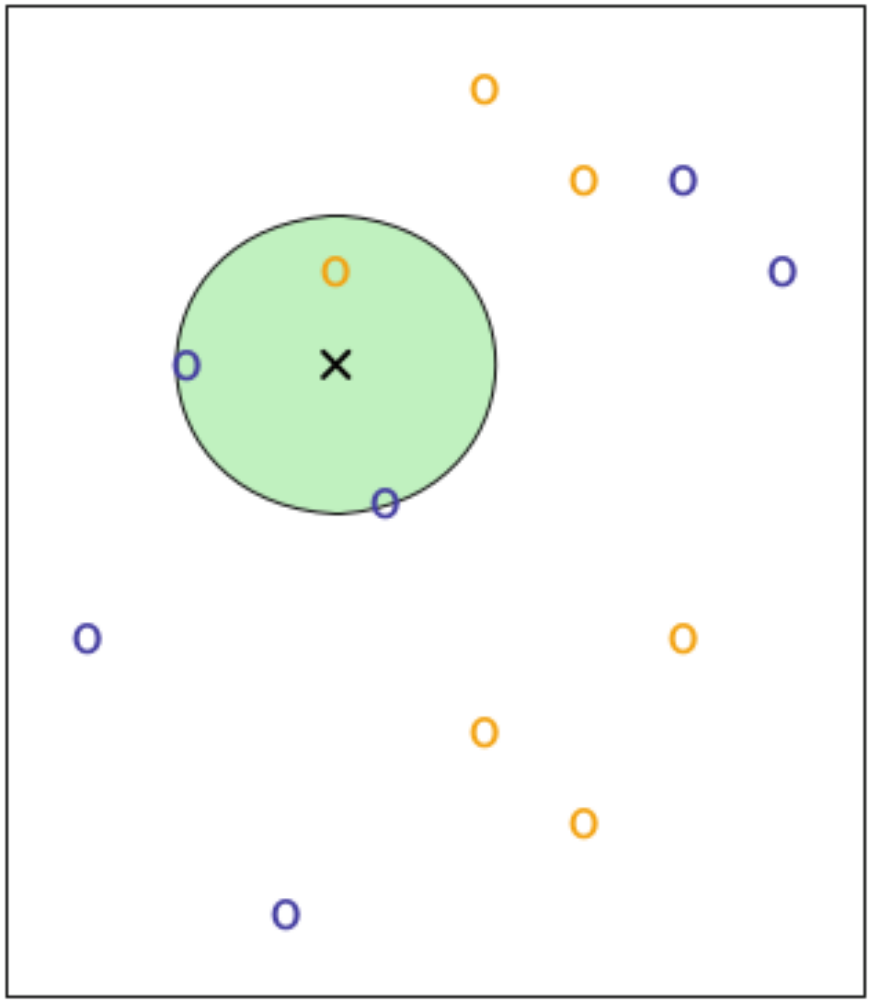
   or, in proper math notation,

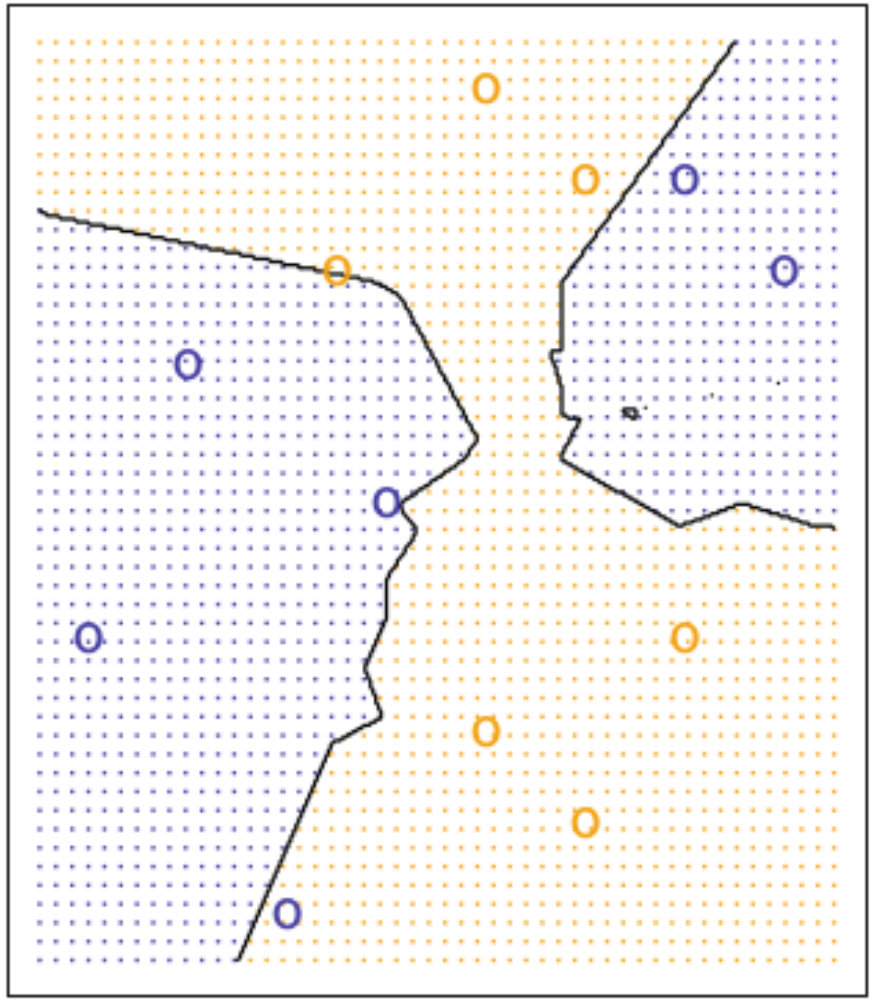
$$Pr(Y = j \mid X = x_0) = \frac{1}{K} \sum_{i \in N_0} \mathbb{I}(y_i = j)$$

3. Classify the obs. to class with highest estimated probability. $\hat{C}_{knn}(x_0) = j$, if $\hat{p}_j(x_0) = max\{\hat{p}_1(x_0), ..., \hat{p}_J(x_0)\}$.

The smaller the K is the more flexible the method will be.

Simple example: KNN with K=3.



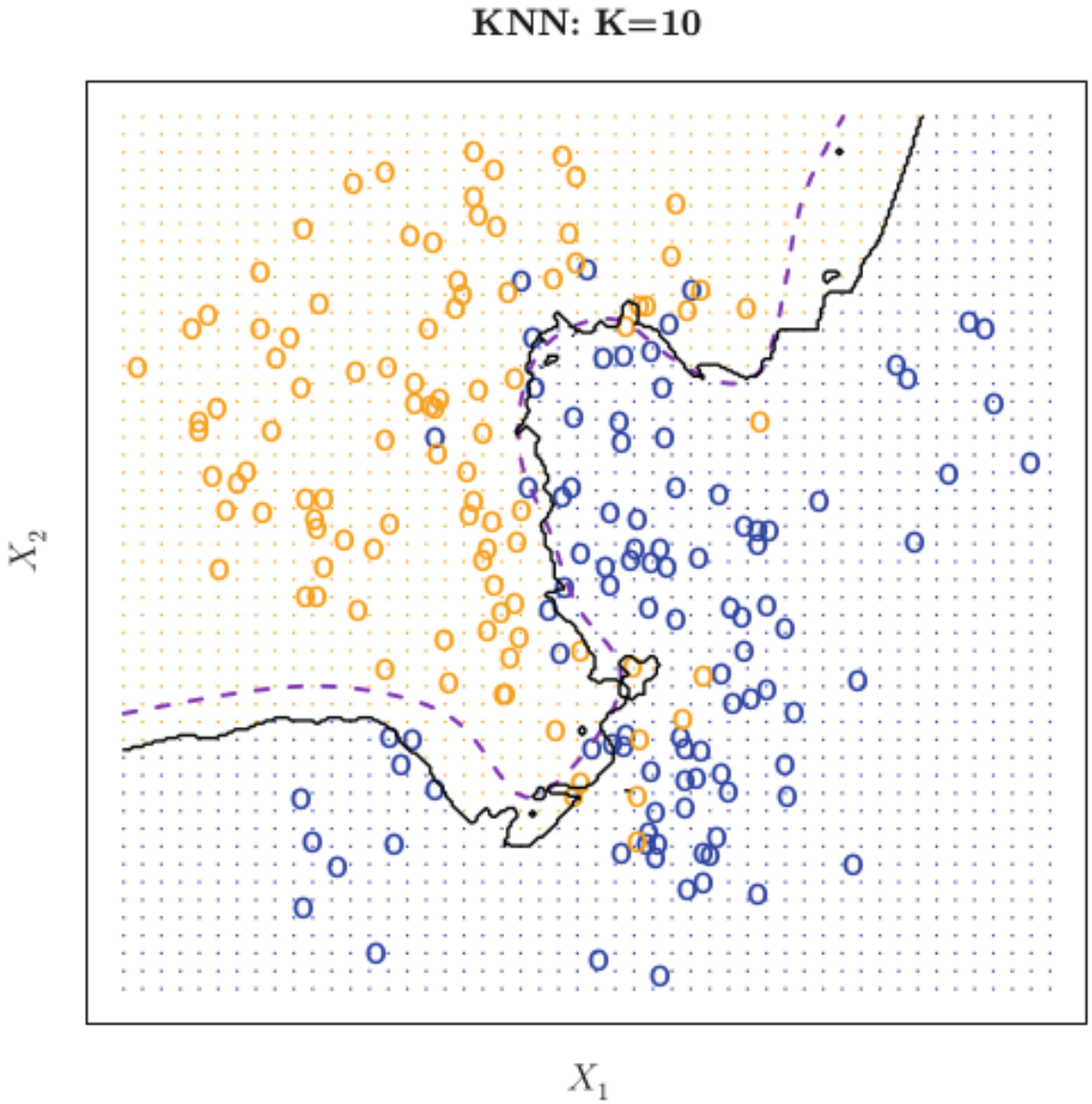


Simulated example: KNN with $K = 10$

Example (cont'd).

Despite it's simplicity, KNN can produce classifiers that are close to the optimal Bayes classifier (the purple dashed line on the plots).

On the right, you may see the boundary resulting from KNN with $K = 10$:



Simulated example: Choice of K for KNN.

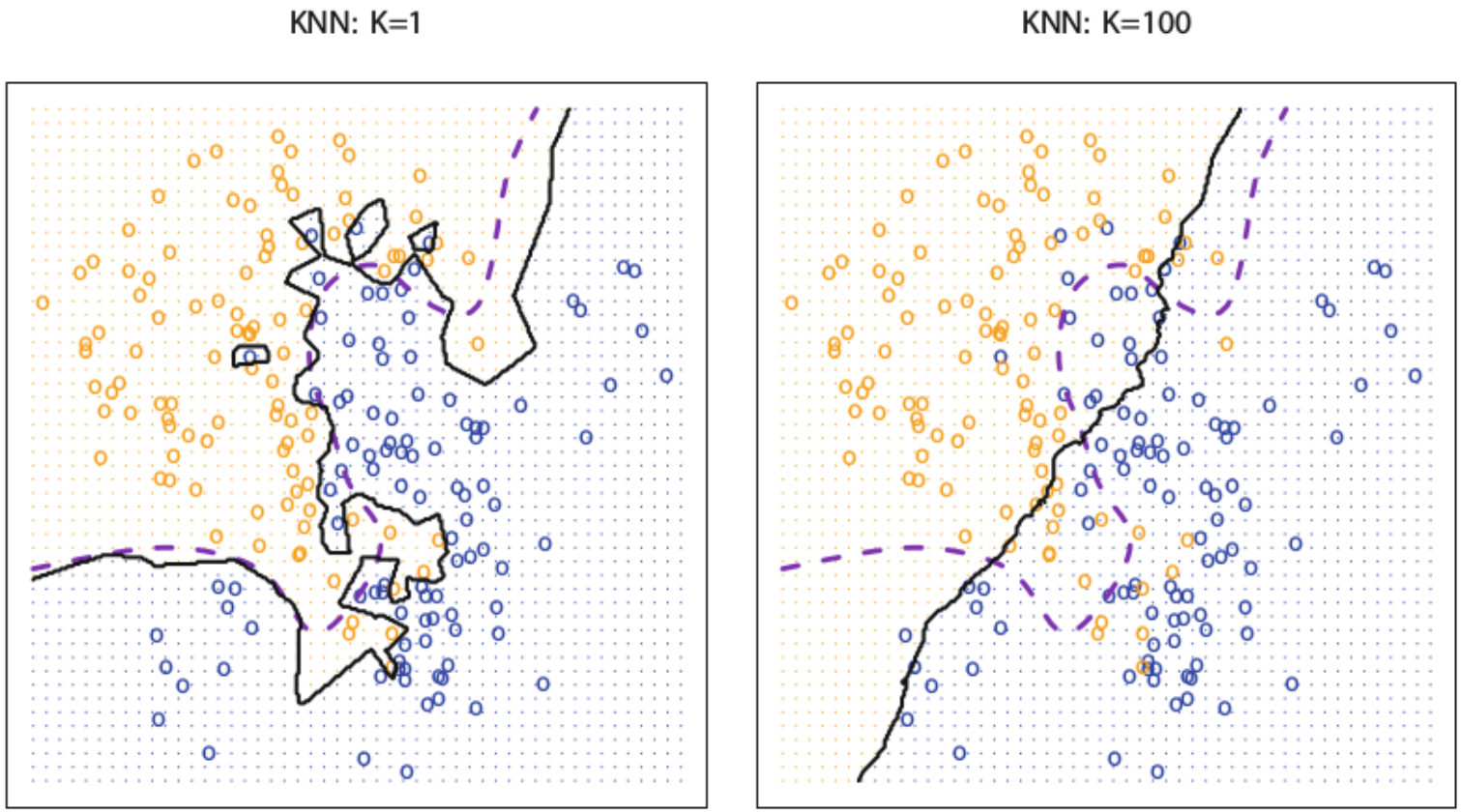


FIGURE 2.16. *A comparison of the KNN decision boundaries (solid black curves) obtained using $K = 1$ and $K = 100$ on the data from Figure 2.13. With $K = 1$, the decision boundary is overly flexible, while with $K = 100$ it is not sufficiently flexible. The Bayes decision boundary is shown as a purple dashed line.*

Assessing model accuracy.

As of now, you've just been introduced to one statistical learning method - K-Nearest Neighbors (KNN), but later we'll also cover Support Vector Machines.

Why is it necessary to introduce multiple different methods? Because there is no single best one: on a particular data set, one specific method may work best, but some other method may work better on a similar but different data set.

Hence it is important to be able to evaluate each model's performance.

Actually, one can even treat two KNN models with **different** K, e.g.

$$K = 1 \text{ versus } K = 100$$

as subjects of comparison.

Measuring the Quality of Fit in Classification.

In classification, the most common approach for quantifying the accuracy of our estimate $\hat{f}$ is the training error rate - the proportion of wrong classifications made when using $\hat{f}$ on the training observations:

$$\frac{1}{n}\sum_{i}\mathbb{I}(y_i \neq \hat{y}_i),$$

where

- n - total # of training observations,
- $\hat{y}_i$ - predicted class label for i^{th} observation via $\hat{f}$, $i = 1, \ldots, n$
- $\mathbb{I}(y_i \neq \hat{y}_i)$ is an indicator function such that

$$\begin{cases} \mathbb{I}(y_i \neq \hat{y}_i) = 1 \text{ if } y_i \neq \hat{y}_i, \text{ wrong prediction}, \\ \mathbb{I}(y_i \neq \hat{y}_i) = 0 \text{ if } y_i = \hat{y}_i, \text{ correct prediction}. \end{cases}$$

Simulated Example: Training Errors.

It's called "training" error because it uses model's performance on the training set of observations - the ones that were used to train the model in the first place.

Example (cont'd). Below are the training errors for the simulated example with blue and orange classes we've discussed:

- $K = 1$: Training Error = 0,
- $K = 10$: Training Error = 0.10,
- $K = 100$: Training Error = 0.17,

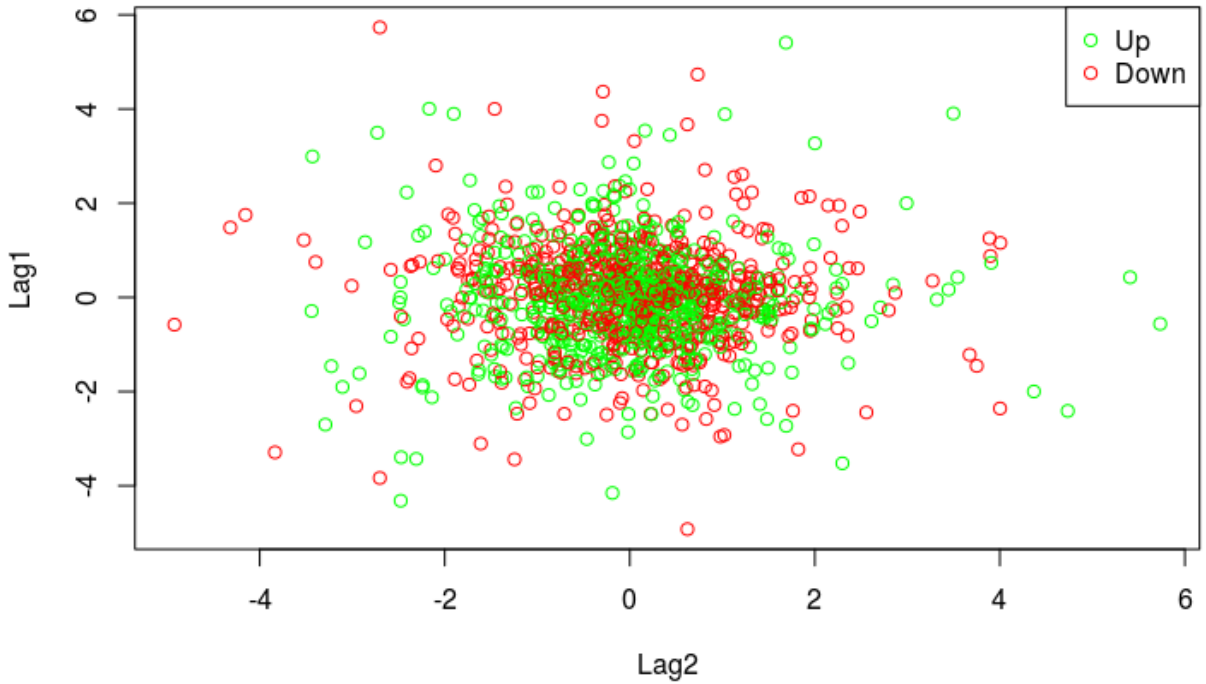
KNN for S&P 500 example.

Finally, onto **real** data:

Example. The same S&P 500 example as before, only now including the trading volume (*Volume*) as another predictor:

- predictors: *Lag*1, *Lag*2, *Lag*3, *Lag*4, *Lag*5, *Volume*,
- response: *Direction* (= "*Down*" /"*Up*")

A bit of exploratory analysis (see source code for details):



KNN for S&P 500 example.

Example (cont'd). We will now perform KNN using the *knn*() function, which is part of the *class* library.

knn() requires four inputs:

1. *train* - a matrix containing the predictors associated with the training data (labeled *train.X* below),
2. *test* - a matrix containing the predictors associated with the data for which we wish to make predictions (labeled *test.X* below),
3. *cl* - a vector containing the class labels for the training observations (labeled *train.Direction* below),
4. *k* - a value for K, the number of nearest neighbors to be used by the classifier.

and as output produces the predicted classes for all observations supplied in *test* argument.

knn() function of library *class*.

```
library(ISLR)
library(class)

train.X <- Smarket[,c("Lag1","Lag2","Lag3","Lag4","Lag5","Volume")
train.Direction <- Smarket[,"Direction"]

## Do it for K=1, 3, 100
K <- 3
knn.obj <- knn(train=train.X,
               test=train.X,
               cl=train.Direction,
               k=K)

## knn.obj contains class predictions
head(knn.obj)
[1] Up   Up   Up   Down Up   Up

## Calculating TRAINING error.
mean(knn.obj != y.train)
[1] 0.2424
```

knn() function of library *class*.

Example (cont'd). Using *knn*() function of *class* library in *R* (see lecture source code for details), the KNN training errors are:

- for $K = 1$: 0
- for $K = 3$: 0.2424
- for $K = 100$: 0.4624

Now, recall that for simulated example we had:

- for $K = 1$: 0
- for $K = 3$: 0.10
- for $K = 100$: 0.17

Question #1: for which K does KNN model yield best training error?

Question #2 (!!!): does this mean that $K = 1$ is the better model?

Test error: Motivation.

In general, we don't really care as much for method's performance on training data, as we do about its performance on previously unseen **test** data.

Example. We develop algorithm to predict stock price based on previous stock returns. We train the model on the past 6 months, but we don't really care how it does on those 6 months. Instead, we prioritize it's performance on future, previously unseen, prices.

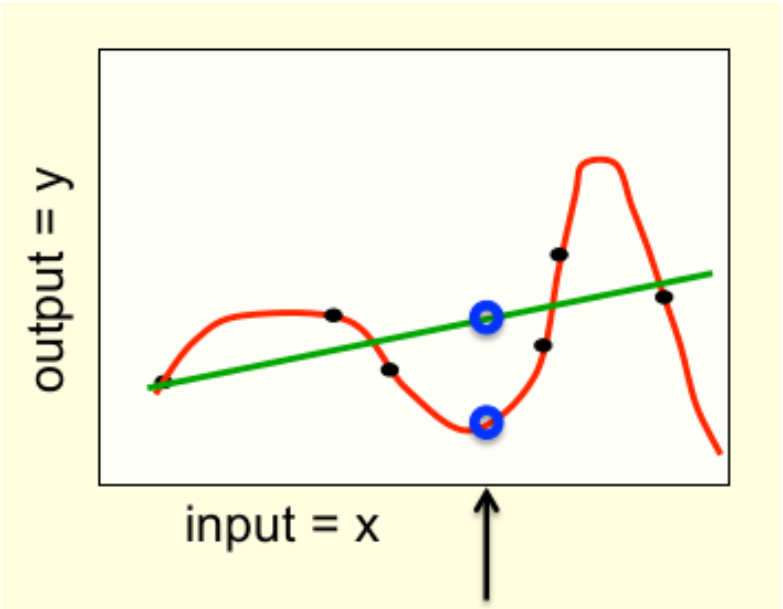
Example. We have patients with clinical measurements (e.g. weight, blood pressure, height, age, family history of disease) as predictors, and response variable being whether a patient has diabetes or not. Statistical learning model is trained on those patients to predict a risk of diabetes given clinical measurements, but we aren't interested in "predicting" the risk diabetes for those patients, as they're already diagnosed. We instead need to diagnose future patients.

Test error. Overfitting.

This leads us to the concept of **test error**.

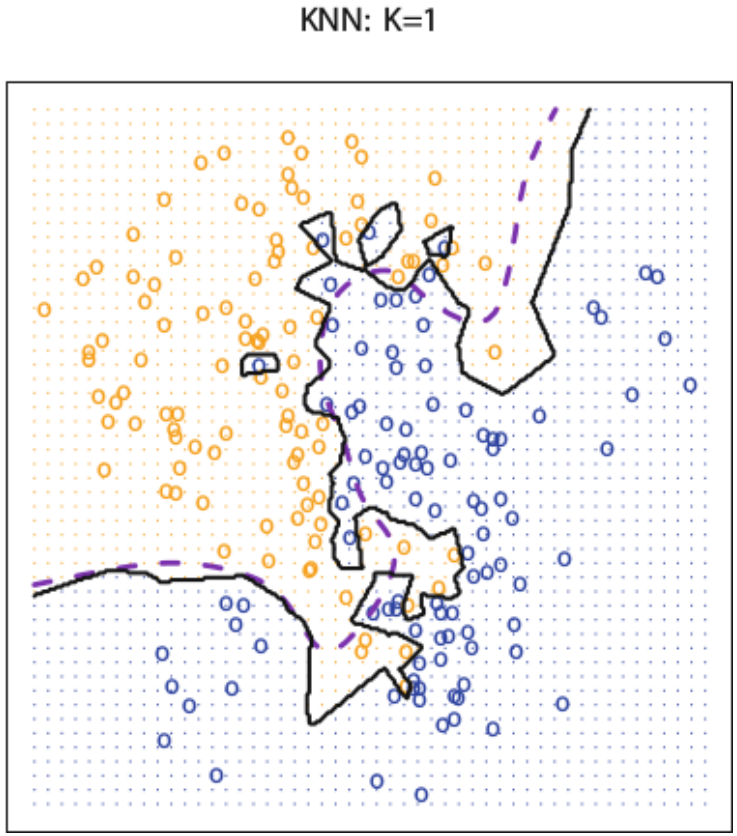
Test error is a much more reasonable metric for evaluation of model performance, compared to training error.

Good training error performance, rather than an indicator of a good generalizable model, instead might be a sign of **overfitting** $\Longrightarrow$ fitting the noise instead of true signal, true relationship between predictors (inputs) & the response (output).



Overfitting: $K = 1$ example.

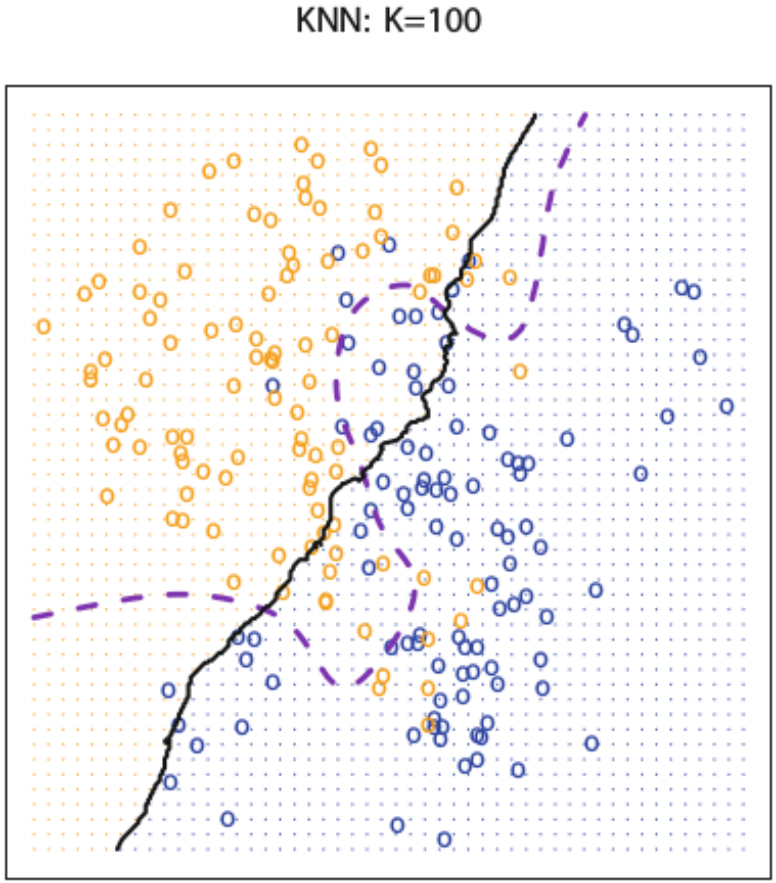
Example (cont'd). Back to our question of whether $K = 1$ might be considered a good model (hint: **it might not**):



Test error rate = 0.1695.

Underfitting: $K = 100$.

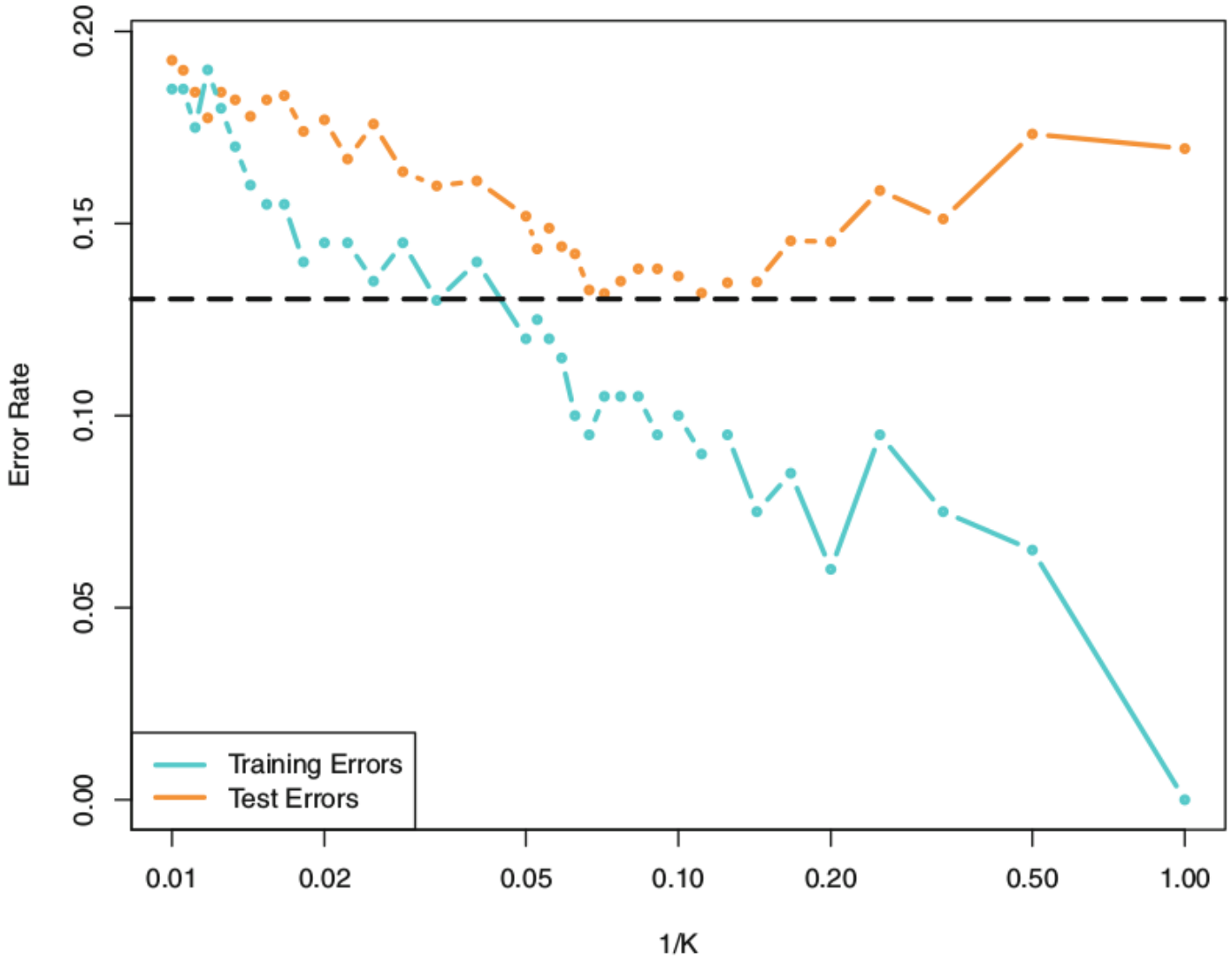
Example (cont'd). While great training error is indicative of overfitting, we don't want to go to another extreme when the training error is very high $\Longrightarrow$ **underfitting**.



Test error rate = 0.1925.

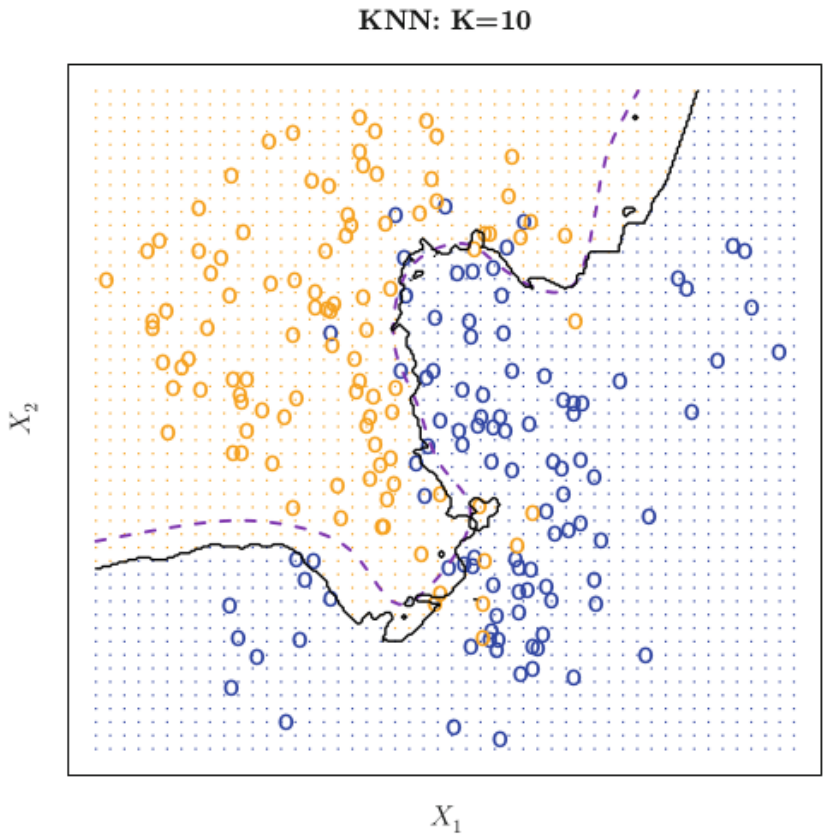
Train & test error dynamic.

Example (cont'd). The dynamic of train and test error as $K \Downarrow$ ($\frac{1}{K} \Uparrow$).



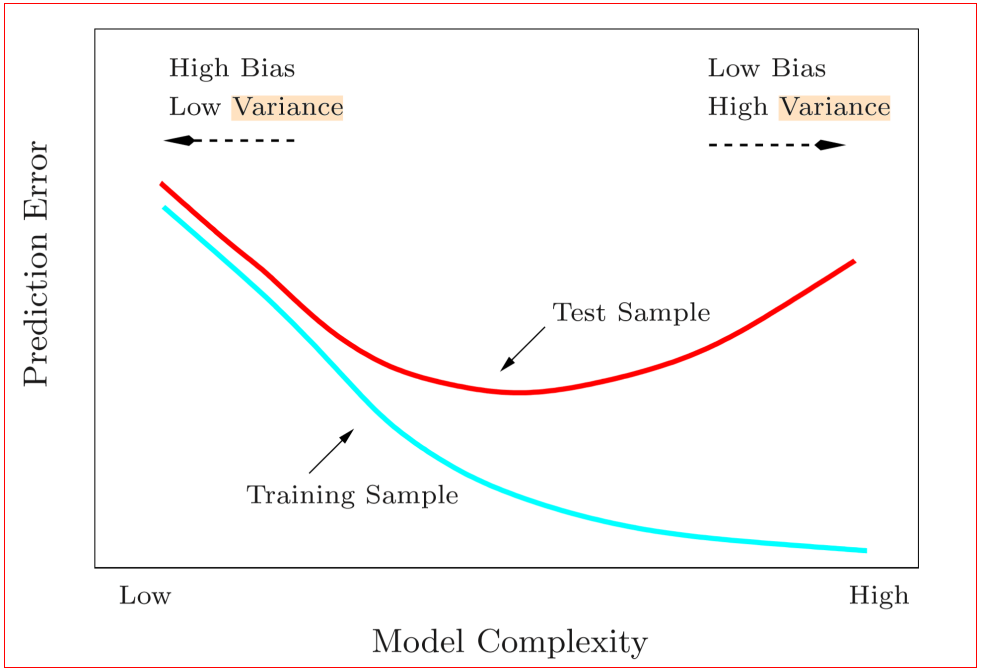
Best model: The One with Smallest Test Error.

Example (cont'd). The best balance can be found by looking at the K that produces smallest **test** error, which is ≈ 10 here.



Test error rate = 0.1363, which is close to the Bayes error rate of 0.1304. Now... **how do we estimate that test error?**

A fundamental picture



- Training errors will always decline in general.
- However, test errors will decline at first but will then start to increase again.
- Keep this picture in mind when choosing a learning method. More flexible/complicated is not always better.

K-Nearest Neighbors Regression.

KNN algorithm can also be applied for regression task. The sole difference is that instead of estimating probability of a class, now we simply take the average of continuous response values of the neighbors:

KNN Regression:

1. Identify the K points in the training data that are closest to x_0 (this set of points is denoted N_0).
2. Estimate $\hat{y}_0 = \hat{f}(x_0)$ using the **average** of all the continuous training responses in N_0 . In other words,

$$\hat{f}(x_0) = \frac{1}{K} \sum_{x_i \in N_0} y_i,$$

K-Nearest Neighbors Regression: Simulated Data.

Example. Below we show two KNN fits on a **simulated** data set with $p = 2$ predictors.

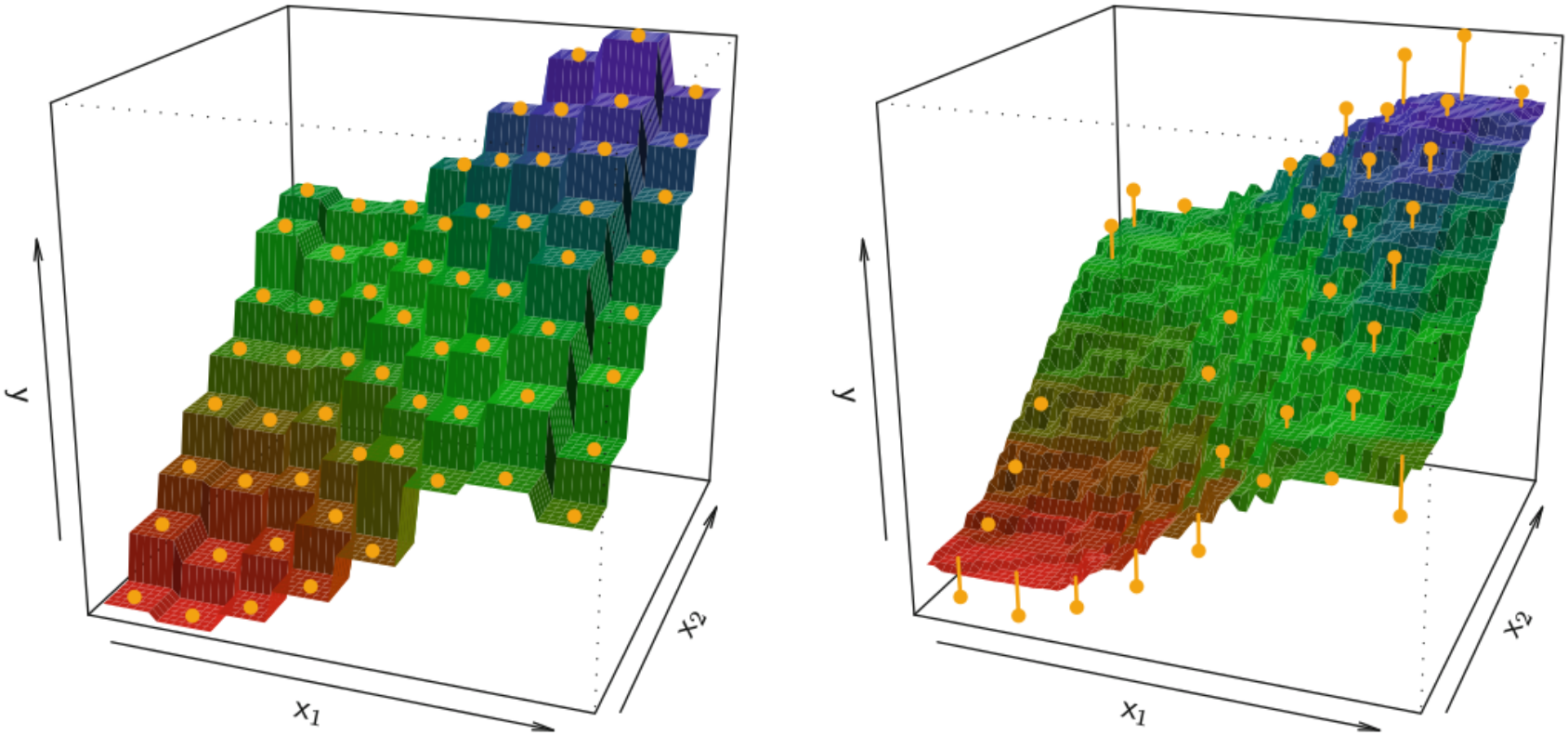


FIGURE 3.16. *Plots of $\hat{f}(X)$ using KNN regression on a two-dimensional data set with 64 observations (orange dots).* Left: $K = 1$ *results in a rough step function fit.* Right: $K = 9$ *produces a much smoother fit.*

K-Nearest Neighbors Regression: Simulated Data.

- When $K = 1$, the KNN fit perfectly interpolates the training observations, and consequently takes the form of a rough step function.
- When $K = 9$, the KNN fit still is a step function, but averaging over nine observations results in much smaller regions of constant prediction $\Longrightarrow$ a smoother fit.

K-Nearest Neighbors - Non-Parametric Method.

K-Nearest Neighbor model (both for classification and regression) is known as a **non-parametric** approach.

For comparison, in case of dealing with two predictors X_1, X_2 and a response Y:

- If we use linear regression, which is a **parametric** approach, we assume the following form of function f in $Y = f(X) + \epsilon$:

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2,$$

  with **parameters** $\beta_0, \beta_1, \beta_2$.
- In KNN regression, we **don't** assume any parametric form on function f. We fully rely on data to dictate that form.

For classification task, same comparison could be made between

- Logistic regression: $f(X) = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2}}{1+e^{\beta_0+\beta_1 X_1+\beta_2 X_2}}$
- KNN classification: no parametric assumption made.

K-Nearest Neighbors - Pros and Cons.

K-NN algorithm is one of the simplest classification algorithm and it is one of the most used learning algorithms.

Pros:

- K-NN is pretty intuitive and simple, with relatively high accuracy.
- K-NN has no assumptions.
- (For curious students) Variety of distance criteria to be chosen from (Euclidean Distance; Hamming Distance; Manhattan Distance; Minkowski Distance)

Cons:

- Computationally expensive — because the algorithm stores all of the training data.
- Does not produce a model, limited ability to understand how the features are related to the output.
- Sensitive to irrelevant features and the scale of the data.