Predictive Modeling

Chapter 4: Over-fitting and model tuning

STA 6543

The University of Texas at San Antonio

Outline

- Overfitting and Model Tuning
- Data Splitting
- Resampling Methods
  - Cross Validation
  - Repeated Training/Test Splits
  - The Bootstrap
- A Simple Classification Example

A motivating two-class data example

p = 2

n = 208

- Consider the data having two predictors and containing 208 samples that can be either “Class 1”or “Class 2.”
- There are 111 samples in Class 1 and 97 Class 2. Furthermore, there is a significant overlap between the classes.

- **Goal**: develop *a predictive model* to classify new samples. (This is just a simple classification problem)

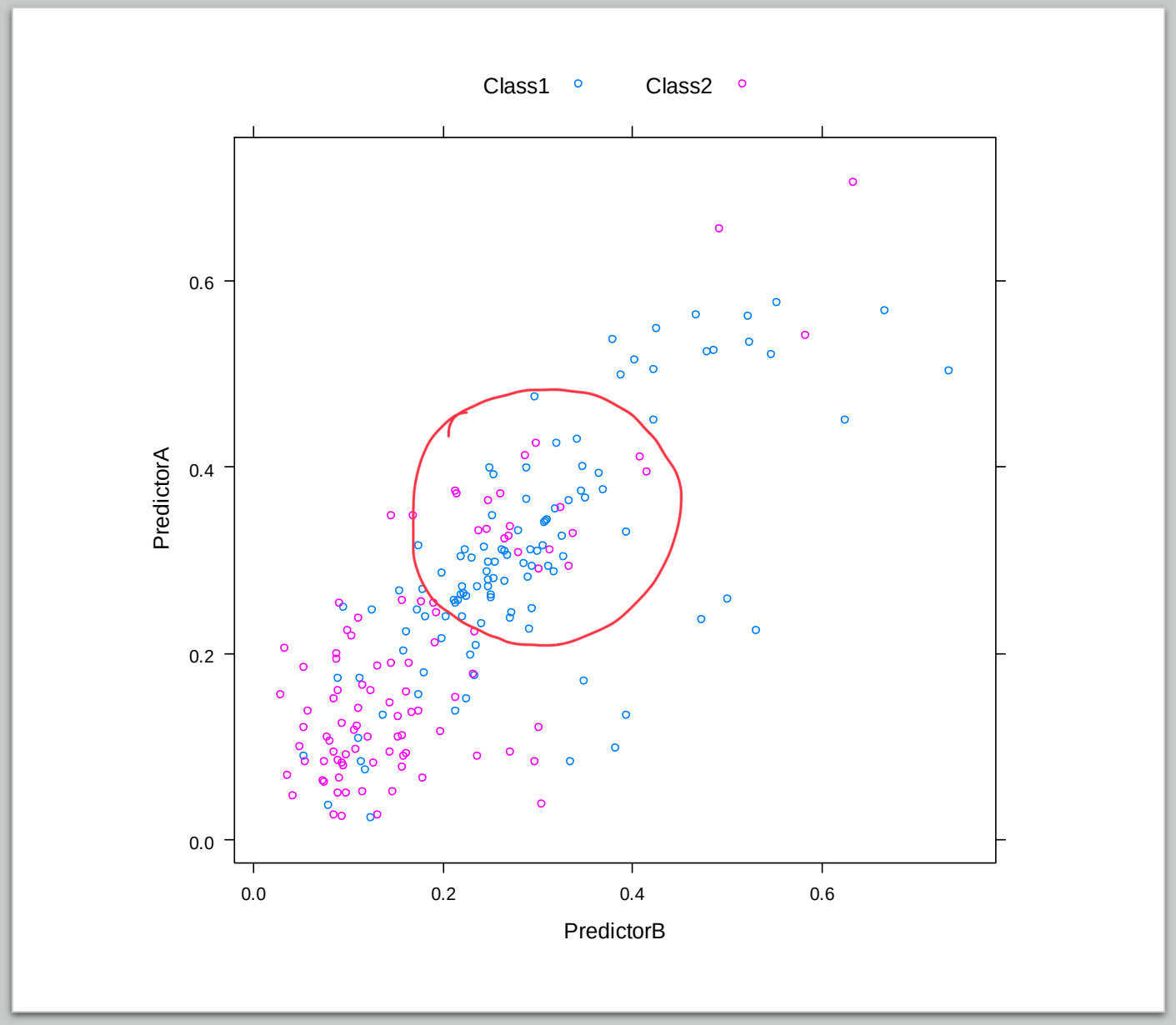
A motivating two-class data example

```
library(AppliedPredictiveModeling)
library(caret)
#Access the two-class data
data(twoClassData)
str(predictors)
str(classes)
data = cbind(predictors, classes)
xyplot(PredictorA~PredictorB, group=classes, data=data,
auto.key=list(column=2))
```

```
> #Access the two-class data
> data(twoClassData)
> str(predictors)
'data.frame':   208 obs. of  2 variables:
 $ PredictorA: num  0.158 0.655 0.706 0.199 0.395 ...
 $ PredictorB: num  0.1609 0.4918 0.6333 0.0881 0.4152 ...
> str(classes)
 Factor w/ 2 levels "Class1","Class2": 2 2 2 2 2 2 2 2 2 2 ...
>
```

A motivating two-class data example

- There is a significant overlap between the classes.

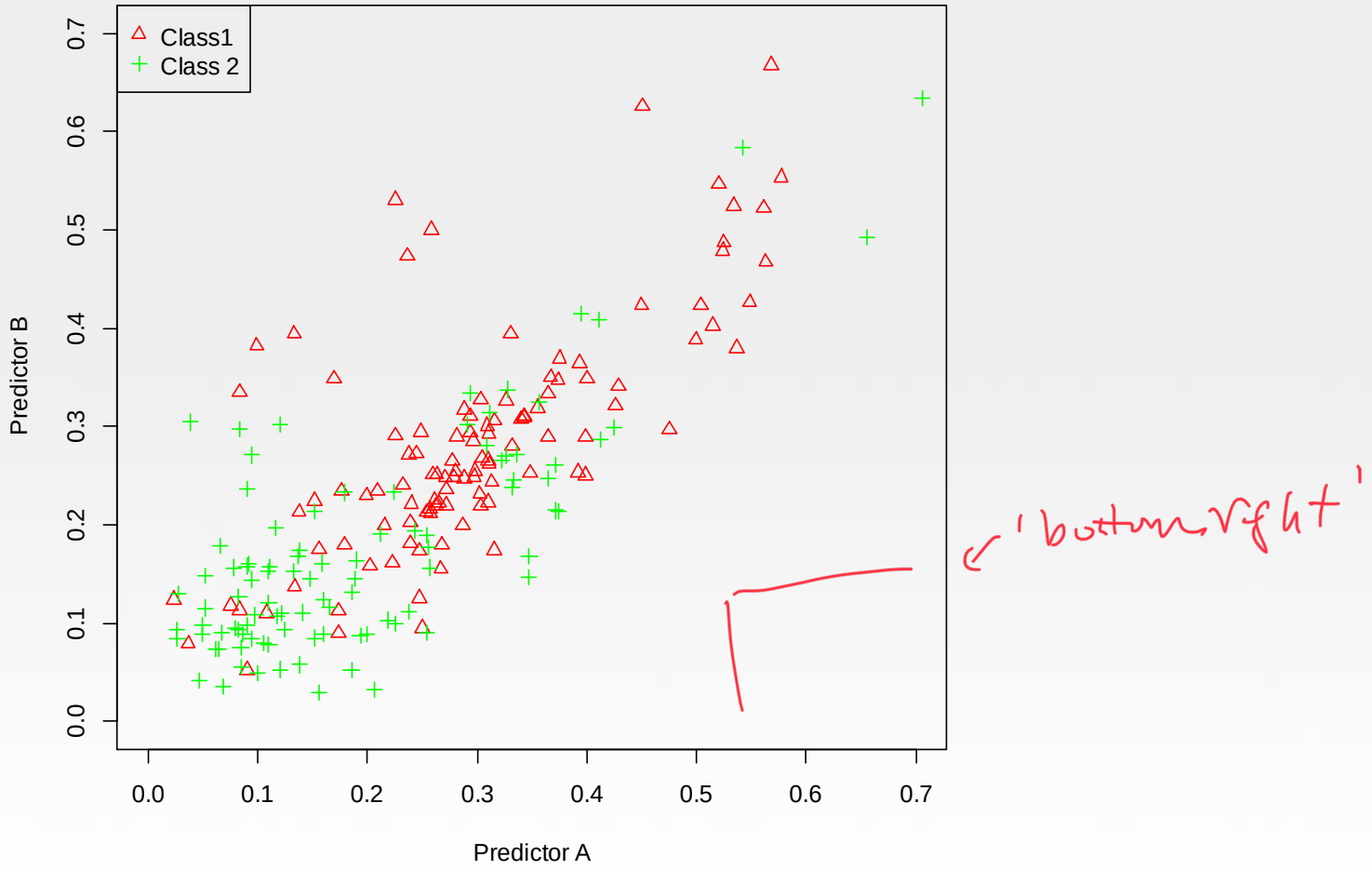


Another way to draw scatter plot

```
library(AppliedPredictiveModeling)
library(caret) #access the train function

#Access the two-class data
data(twoClassData)
str(predictors)
str(classes)

#Use the regular plot function, instead of xyplot
plot(predictors$PredictorA[classes=='Class1'], predictors$PredictorB[classes=='Class1'],
xlab="Predictor A", ylab="Predictor B", xlim=c(0,0.7), ylim=c(0,0.7), col ='red', pch=2)
points(predictors$PredictorA[classes=='Class2'], predictors$PredictorB[classes=='Class2'],
pch=3, col ='green')
legend('topleft', c("Class1", "Class 2"), col=c('red','green' ), pch=c(2,3))
```

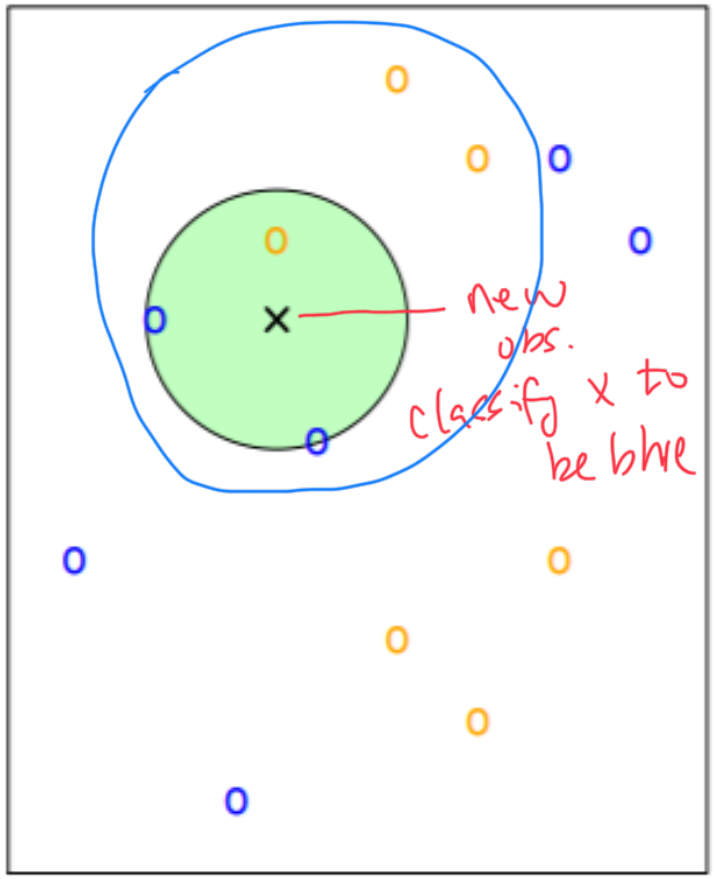
'top right' 'bottom left' 'bottom right' ((0.5, 1.1)

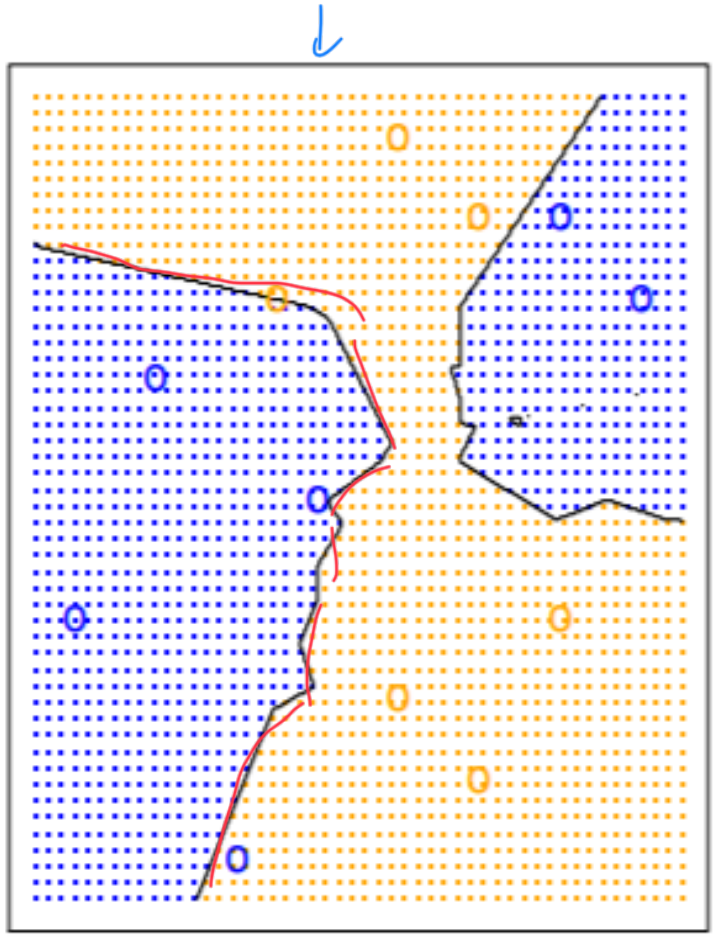
A simple classification example



A predictive model: K-Nearest Neighbors (kNN) with $k = 3$

tuning parameter

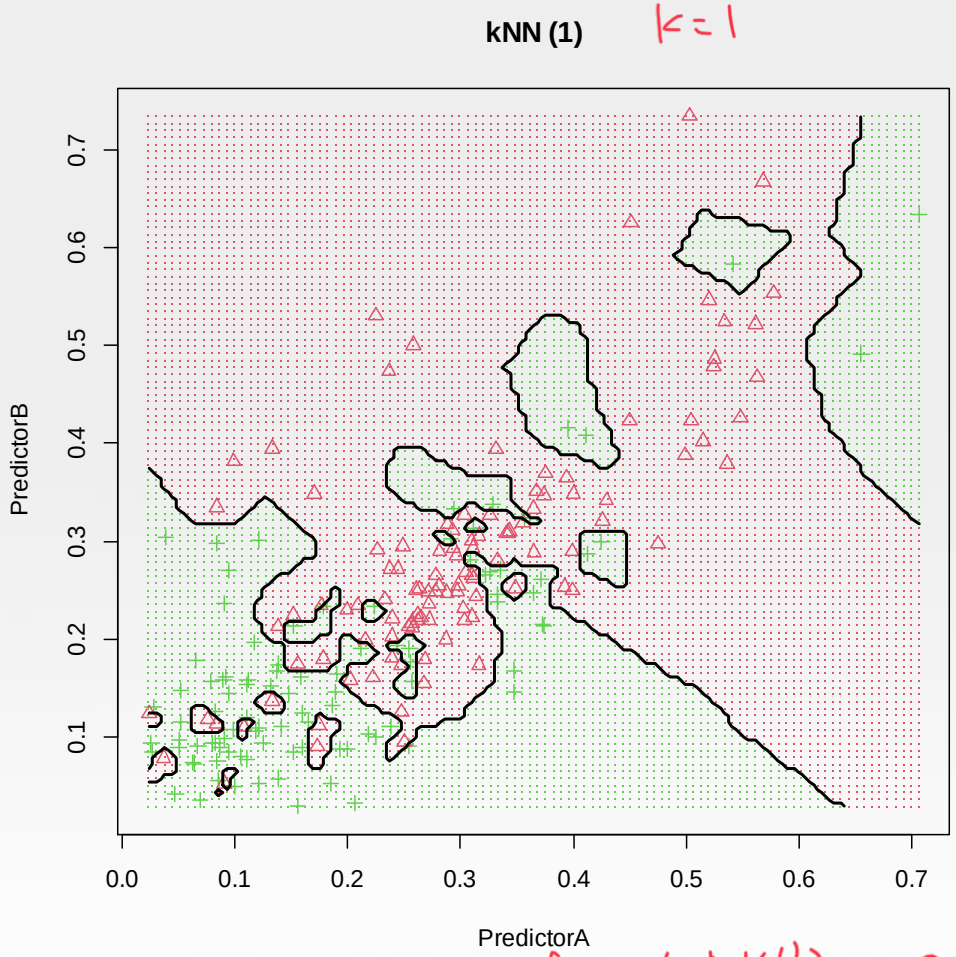


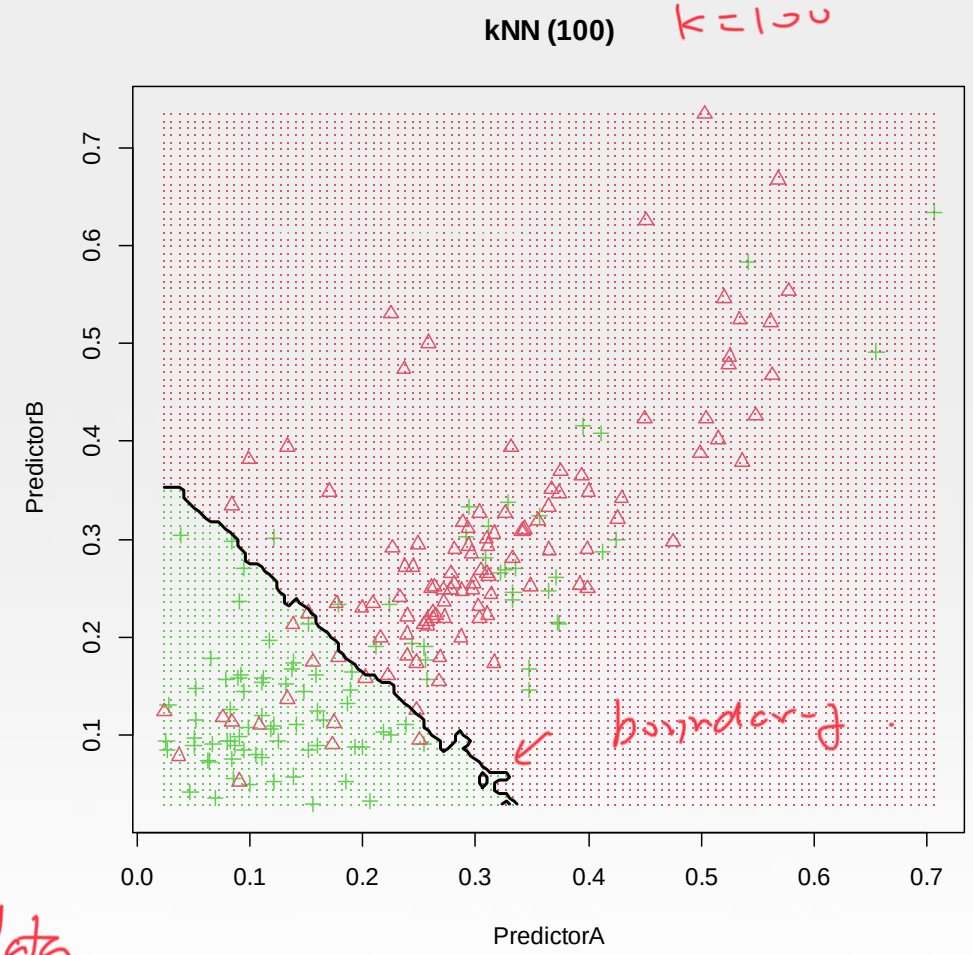


- KNN is often used for Classification problems
- You need to determine the value of the tuning parameter k.

k=5 : 'x' → yellow.

kNN methods (codes available on the blackboard)





classification error for KNN(1) = 0 for training data

Comments on kNN methods

- The previous panels show kNN with different values of k (i.e., *k=1* and *k = 100*) and their associated class boundaries.

- How do we determine an optimal value for *k*?
  - We may apply the KNN method available at the caret's *train* package.

Apply kNN

```
#Apply kNN method available at the caret package
set.seed(1)
ctrl <- trainControl(method="repeatedcv",repeats = 3)
knnFit <- train(classes ~ ., data = data, method = "knn", trControl = ctrl, preProcess = c("center","scale"),
tuneLength = 25)
knnFit
plot(knnFit, print.thres = 0.5, type="S")
```

Apply kNN

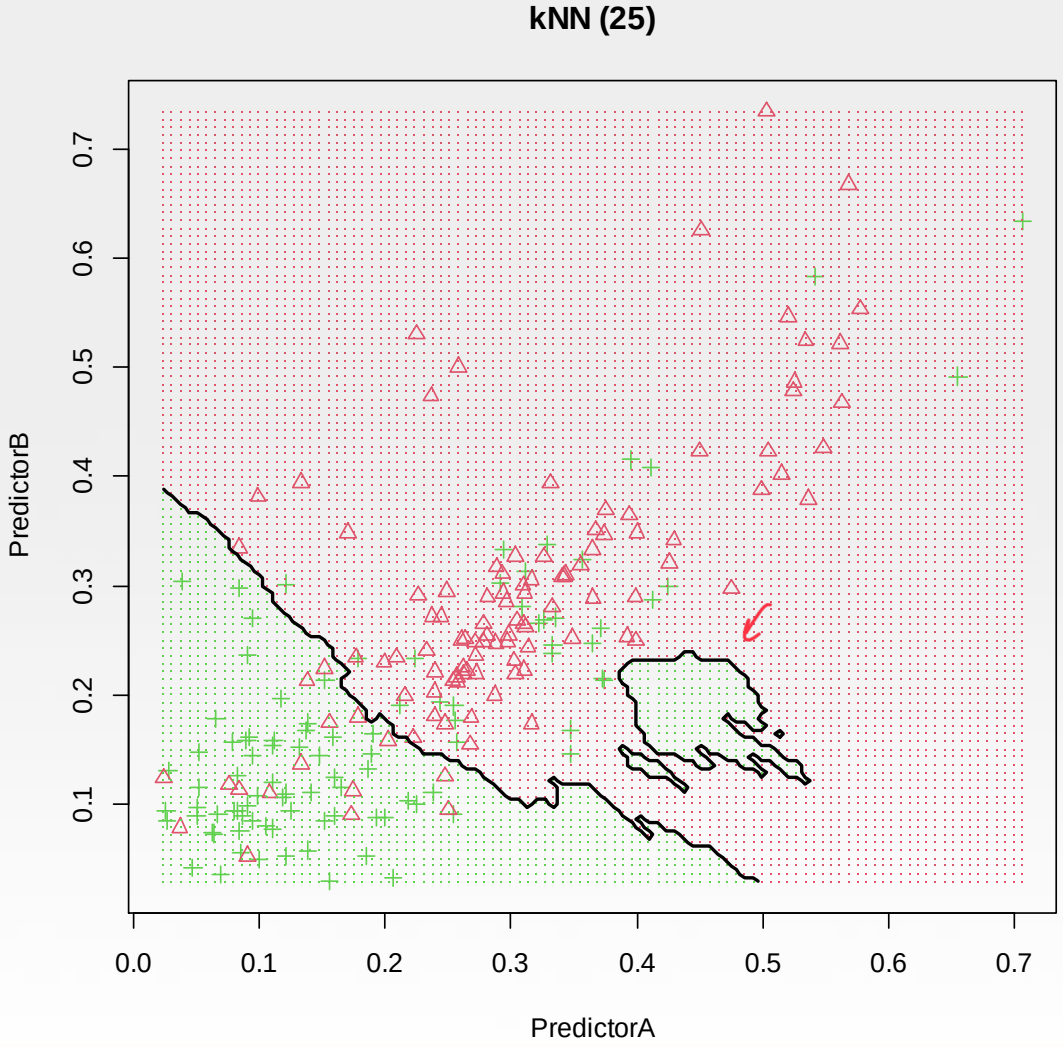
```
> #Apply kNN method available at the caret package
> set.seed(1)
> ctrl <- trainControl(method="repeatedcv",repeats = 3)
> knnFit <- train(classes ~ ., data = data, method = "knn", trControl = ctrl, preProcess = c("center"$
> knnFit
k-Nearest Neighbors

208 samples
  2 predictor
  2 classes: 'Class1', 'Class2'

Pre-processing: centered (2), scaled (2)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 187, 187, 187, 187, 187, 187, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   5  0.7401299  0.4769202

  25  0.7690188  0.5308269
  27  0.7640981  0.5211027

  45  0.7673521  0.5274978
  47  0.7673521  0.5274978
  49  0.7673521  0.5272358
  51  0.7658442  0.5241982
  53  0.7658442  0.5236155

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 25.
```

n = 208

p = 2

The large the value of Accuracy and Kappa, the better the performance of the model.

kNN with k = 25



Which model is preferred? Why?

kNN(1), kNN(100), kNN(25)

Measuring the quality of classifier

- For a classification problem we can use the error rate, i.e.

misclassification

$$Err = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

- where $\hat{y}_i$ is the predicted class label for the *i*th observation using the estimated classifier model.
- $I(y_i \neq \hat{y}_i)$ is an indicator function, which will give 1 if $y_i \neq \hat{y}_i$, otherwise it gives a 0.
- The error rate represents the fraction of misclassifications, or incorrect classifications.

Training vs testing error rates

- If we use the *training* data to estimate the error, which is referred as *apparent performance* of the model (i.e., the apparent error rate or the training error rate.)

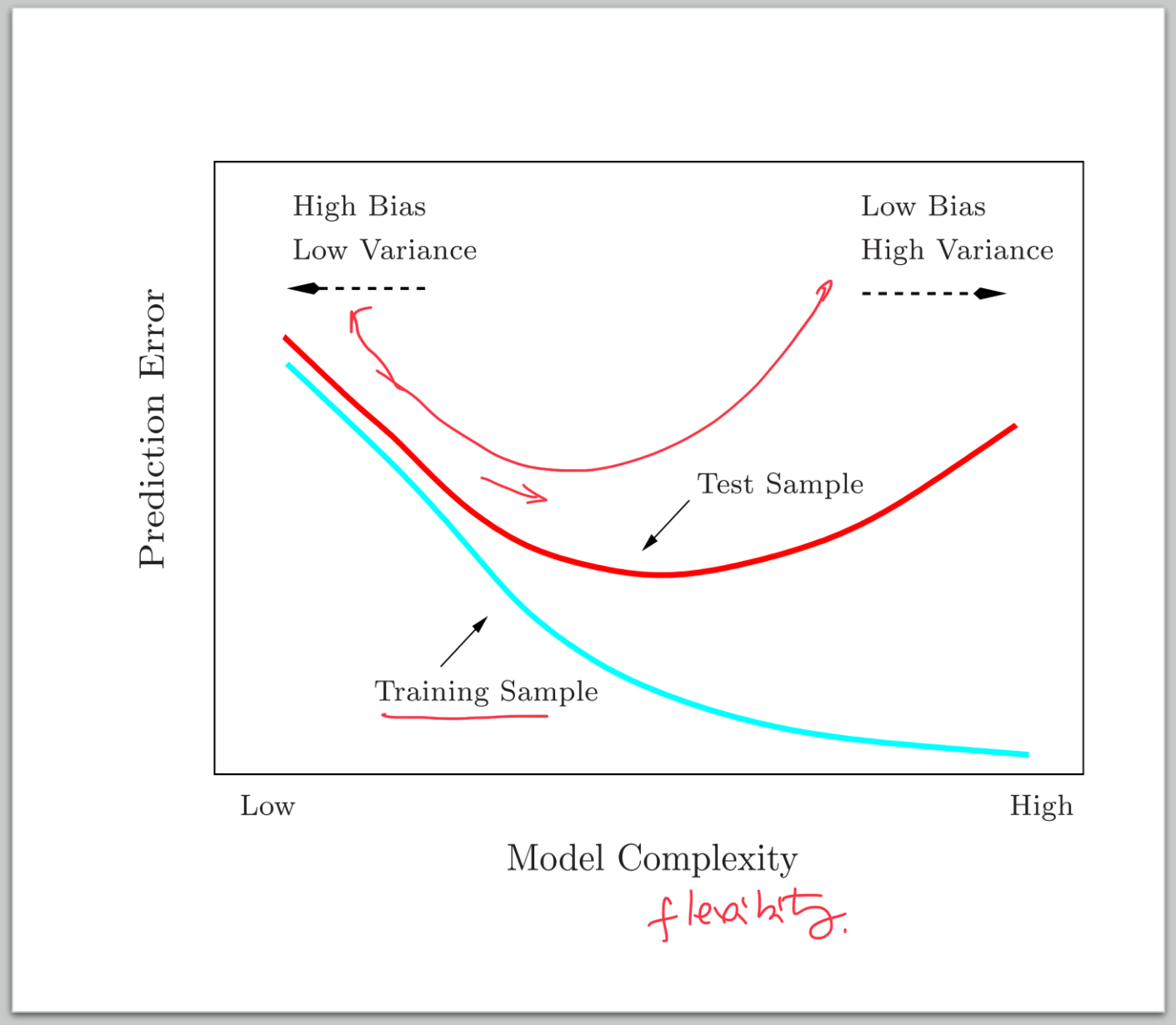
KNN (1) → 0

- If we use the *testing* data to estimate the error, which is referred as *testing error* of the model (i.e., the testing error rate.)

- Questions:
  - (1) How to characterize how much a model is *over-fitting* the training data?
  - (2) How to *split* the data as the training and testing data?

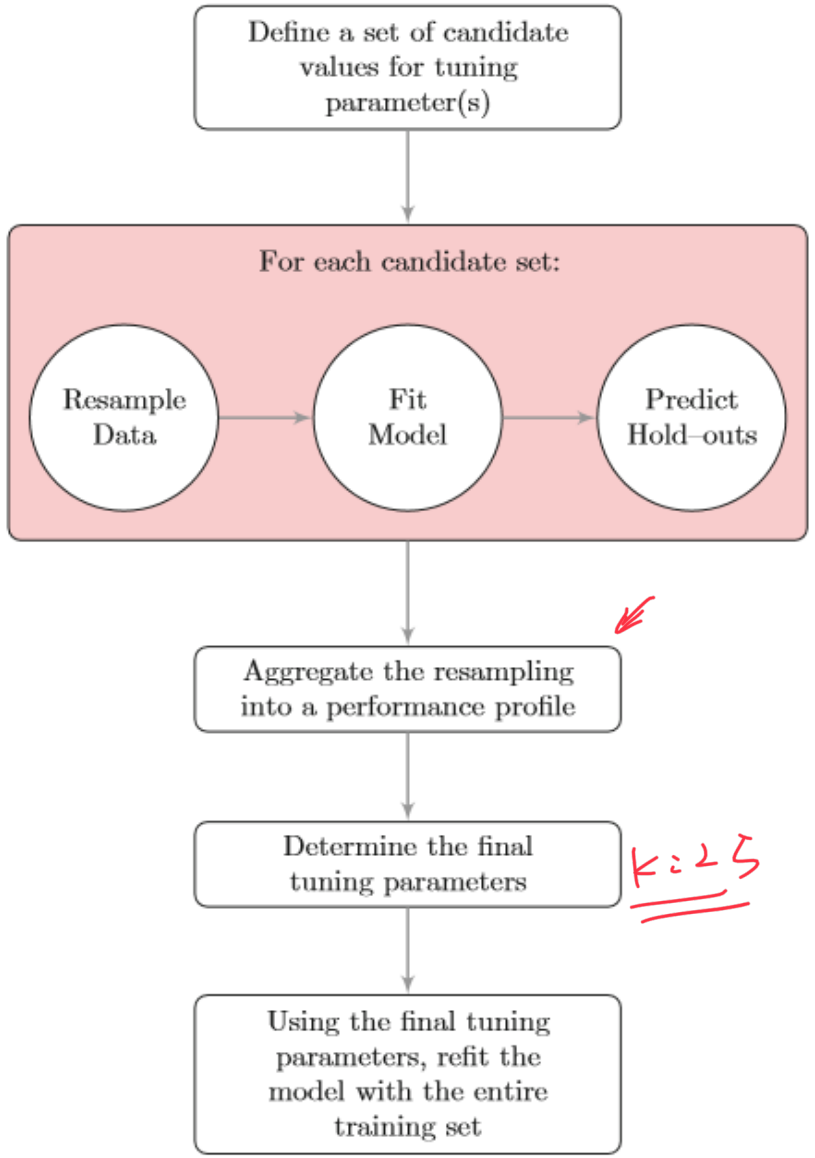
A fundamental picture

- Training errors will always decline in general.
- However, test errors will decline at first (as reductions in bias dominate) but will then start to increase again (as increases in variance dominate).
- *Test error* rate is the key of choosing a learning method.



Model tuning

- Many models have *important model parameters* which cannot be directly estimated from the data.
- This type of parameter is referred to as a *tuning parameter since* there is no analytical formula available to calculate its appropriate value.
- Poor choices for these tuning parameters could result in *overfitting*.
- Different procedures can be used for searching for the best tuning parameters.
- A schematic of the parameter tuning process is given below:

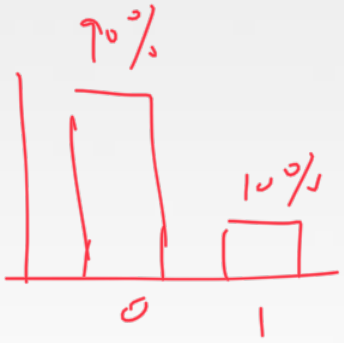



```
> #Apply kNN method available at the caret package
> set.seed(1)
> ctrl <- trainControl(method="repeatedcv",repeats = 3)
> knnFit <- train(classes ~ ., data = data, method = "knn", trControl = ctrl, preProcess = c("
> knnFit
k-Nearest Neighbors

208 samples
  2 predictor
  2 classes: 'Class1', 'Class2'

Pre-processing: centered (2), scaled (2)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 187, 187, 187, 187, 187, 187, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   5  0.7401299  0.4769202

  25  0.7690188  0.5308269
  27  0.7640981  0.5211027

  45  0.7673521  0.5274978
  47  0.7673521  0.5274978
  49  0.7673521  0.5272358
  51  0.7658442  0.5241982
  53  0.7658442  0.5236155

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 25.
```

Data splitting

- To find optimal tuning parameters, we often employ data splitting techniques.
- A few of the common steps in model building are:
  - *Pre-processing* the predictor data
  - Estimating model parameters
  - Selecting predictors for the model
  - Evaluating model performance
  - Fine tuning class prediction rules (via ROC curves, etc.)

- Some common ways for data splitting
  - Simple random splitting (homogeneous samples)
  - Stratified random splitting (keep structure within subgourps)
  - Maximum dissimilarity sampling (on the basis of the predictor values)



but homogenous ?

Data splitting in R

- The base R function *sample* can create simple random splits of the data.
- To create stratified random splits of the data (based on the classes), the *createDataPartition* function in the caret package can be used. (The percent of data that will be allocated to the training set should be specified)
- For maximum dissimilarity sampling, the **caret** function *maxdissim* can be used to sequentially sample the data.