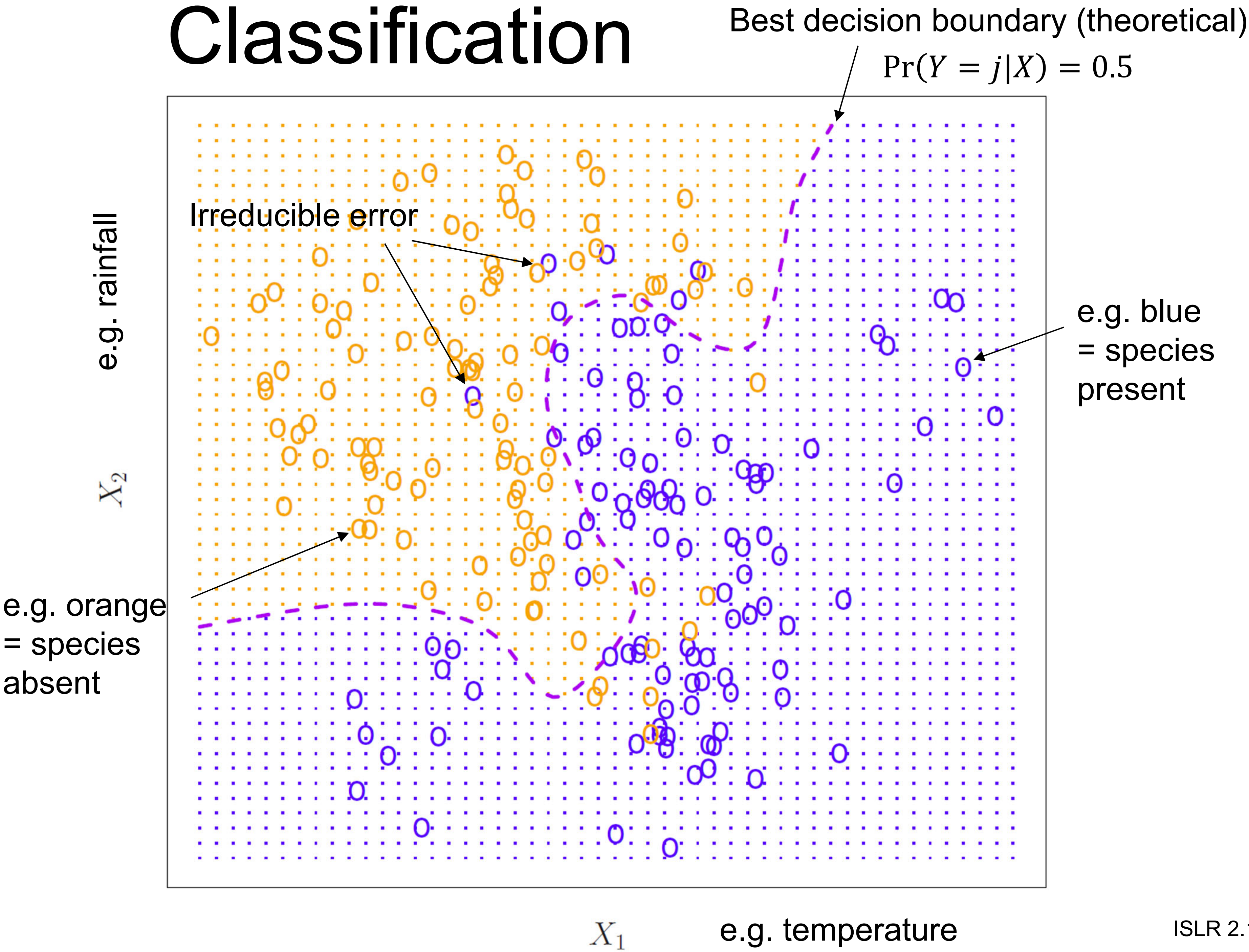
Today

- Classification case

Regression & classification

- Regression:
  - numerical response variable
  - predict a numerical value given x
  - e.g. number of species given latitude
- Classification:
  - categorical response variable
  - predict the category given x
  - e.g. is it a bird, deer, tree, or mountain lion?
  - e.g. is it dead or alive?; present or absent?

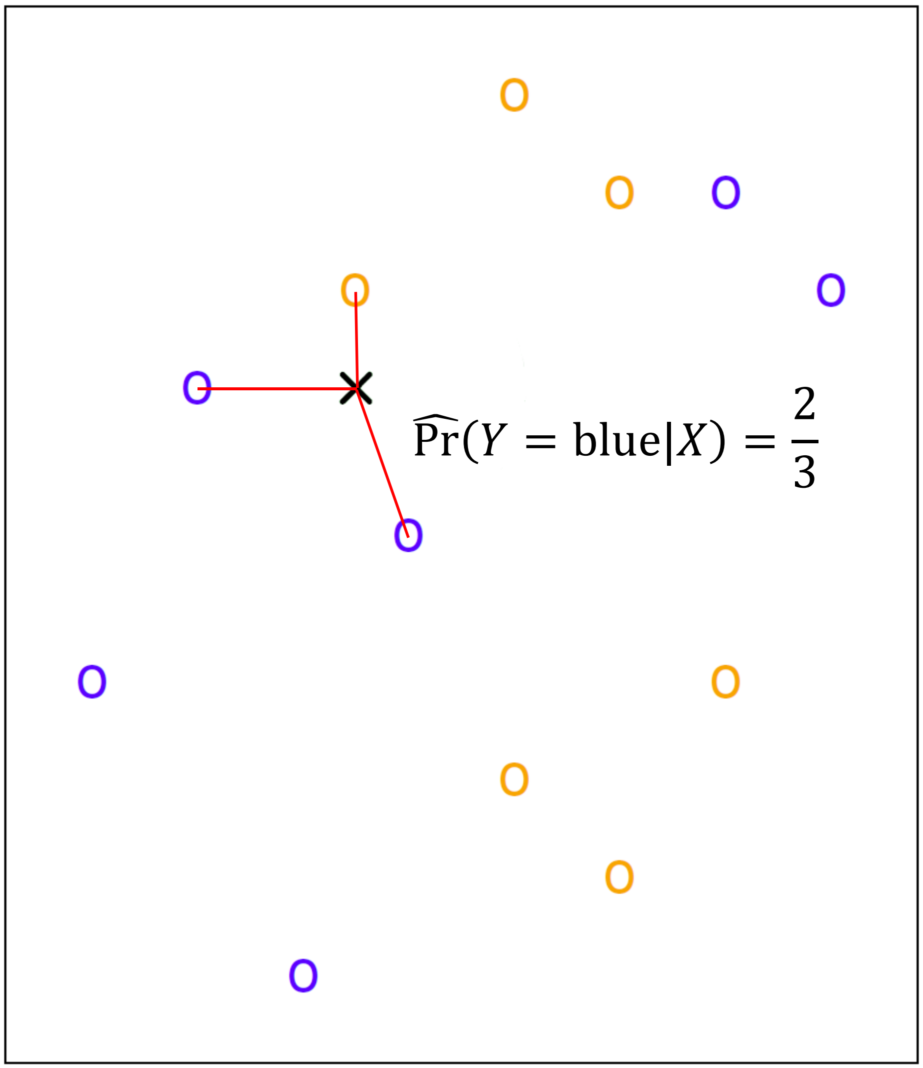
Classification

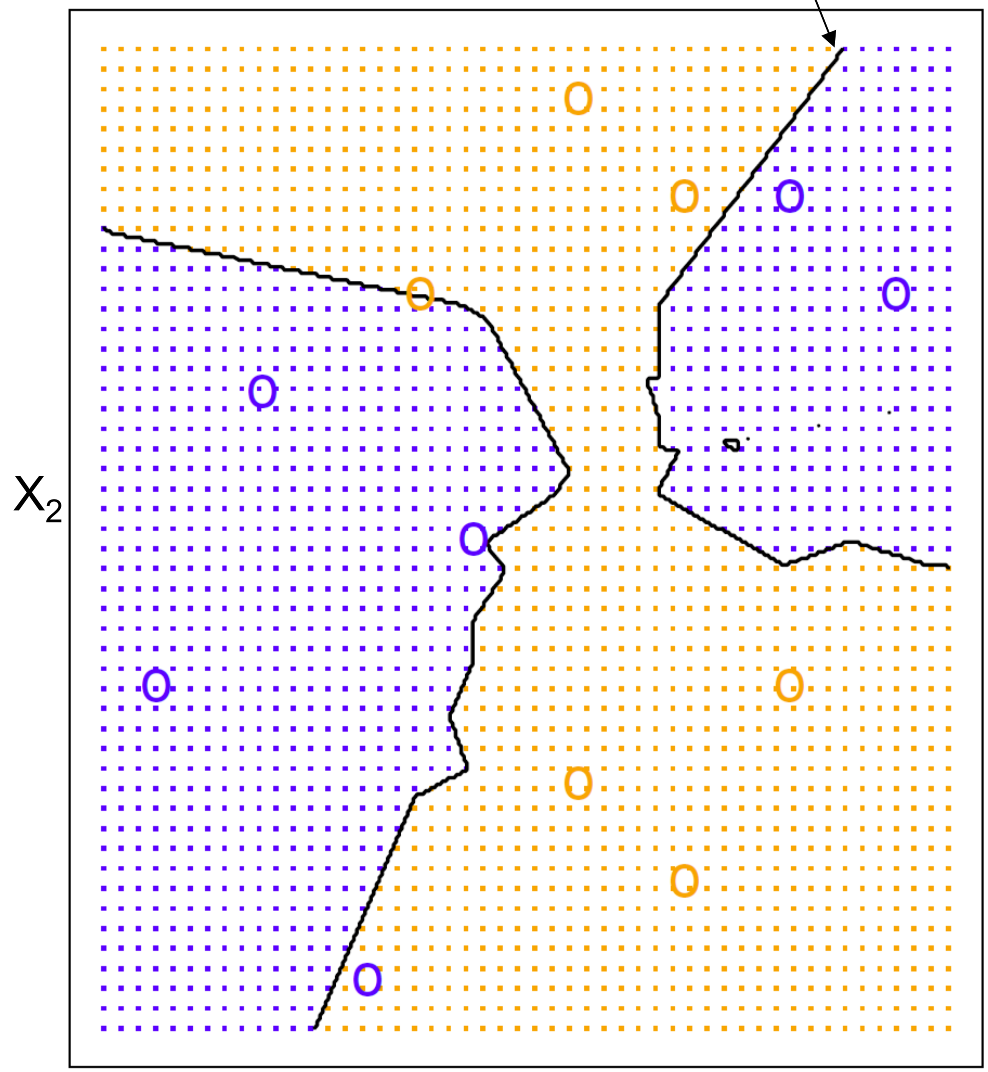


ISLR 2.13

KNN
k = 3

KNN decision boundary
$\widehat{\Pr}(Y = j|X) = 0.5$





ISLR 2.14

Code

- Look at the KNN algorithm
- in
- classification_knn.R
- classification_knn.py

Classification algorithm

Regression algorithm

```r
# KNN function for a vector of x_new
# x:       x data (vector, numeric)
# y:       y data (vector, numeric)
# x_new:   x values at which to predict y (vector, r
# k:       number of nearest neighbors to average (s
# return:  predicted y at x_new (vector, numeric)
#
knn <- function(x, y, x_new, k) {
    y_pred <- NA * x_new
    for ( i in 1:length(x_new) ) {
    #   Distance of x_new to other x
        d <- abs(x - x_new[i])
    #   Sort y ascending by d; break ties randomly
        y_sort <- y[order(d, sample(1:length(d)))]
    #   Mean of k nearest y data
        y_pred[i] <- mean(y_sort[1:k])
    }
    return(y_pred)
}
```

Same

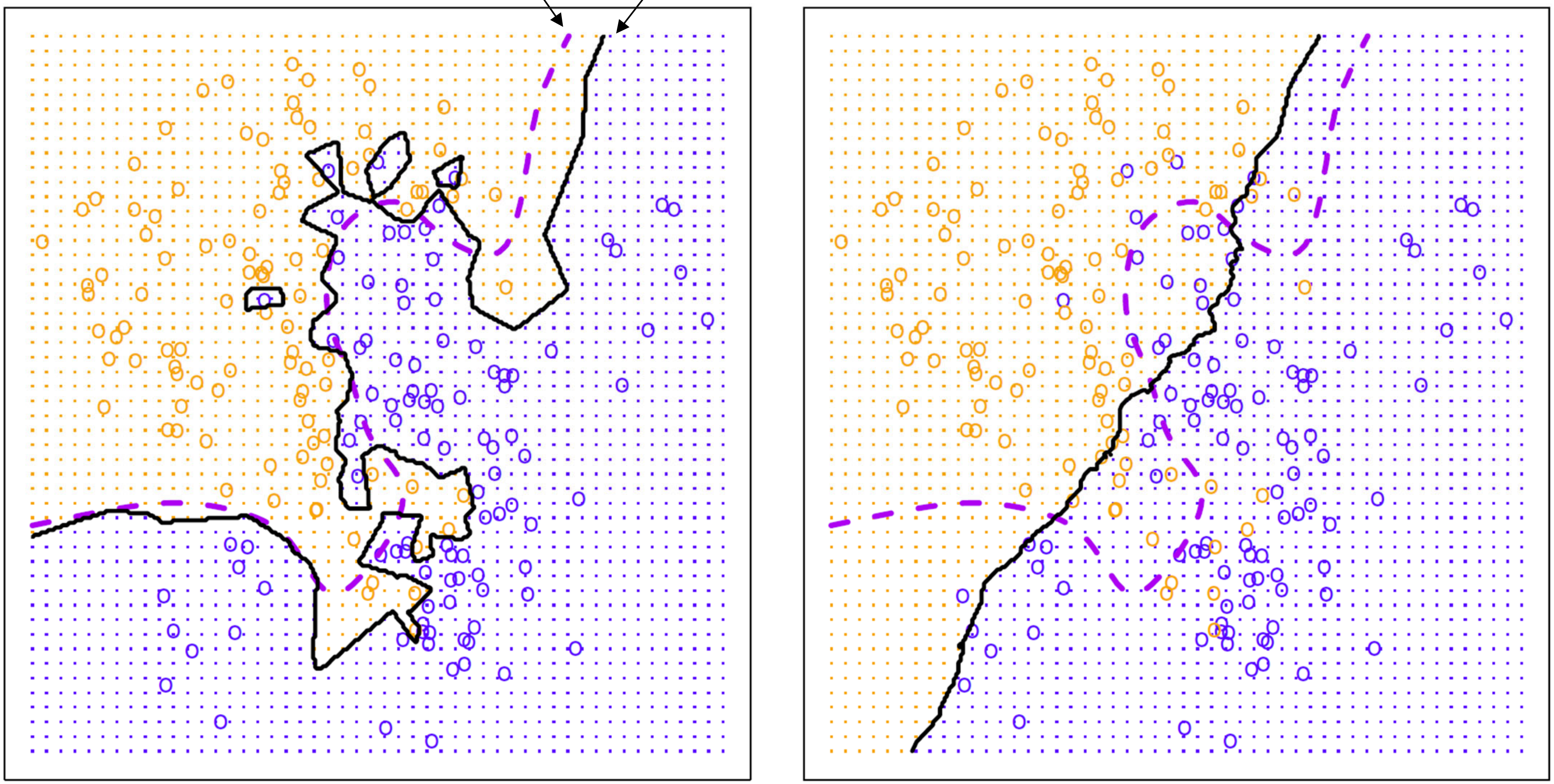
```r
# KNN function for a data frame of x_new
# x:       x data of variables in columns (matrix, numeric)
# y:       y data, 2 categories (vector, character)
# x_new:   values of x variables at which to predict y (matrix, numeric)
# k:       number of nearest neighbors to average (scalar, integer)
# return:  predicted y at x_new (vector, character)
#
knn_classify2 <- function(x, y, x_new, k) {
    category <- unique(y) #get the two category names
    y_int <- ifelse(y == category[1], 1, 0) #convert categories to integer
    nx <- nrow(x)
    n <- nrow(x_new)
    c <- ncol(x_new)
    p_cat1 <- rep(NA, n)
    for ( i in 1:n ) {
    #   Distance of x_new to other x (Euclidean, i.e. sqrt(a^2+b^2+..))
        x_new_m <- matrix(x_new[i,], nx, c, byrow=TRUE)
        d <- sqrt(rowSums((x - x_new_m) ^ 2))
    #   Sort y ascending by d; break ties randomly
        y_sort <- y_int[order(d, sample(1:length(d)))]
    #   Mean of k nearest y data (gives probability of category 1)
        p_cat1[i] <- mean(y_sort[1:k])
    }
    y_pred <- ifelse(p_cat1 > 0.5, category[1], category[2])
    # Break ties if probability is equal (i.e. exactly 0.5)
    rnd_category <- sample(category, n, replace=TRUE) #vector of random la
    tol <- 1 / (k * 10) #tolerance for checking equality
    y_pred <- ifelse(abs(p_cat1 - 0.5) < tol, rnd_category, y_pred)
    return(y_pred)
}
```

KNN
k = 1

Best decision boundary (theory)

KNN decision boundary

KNN
k = 100

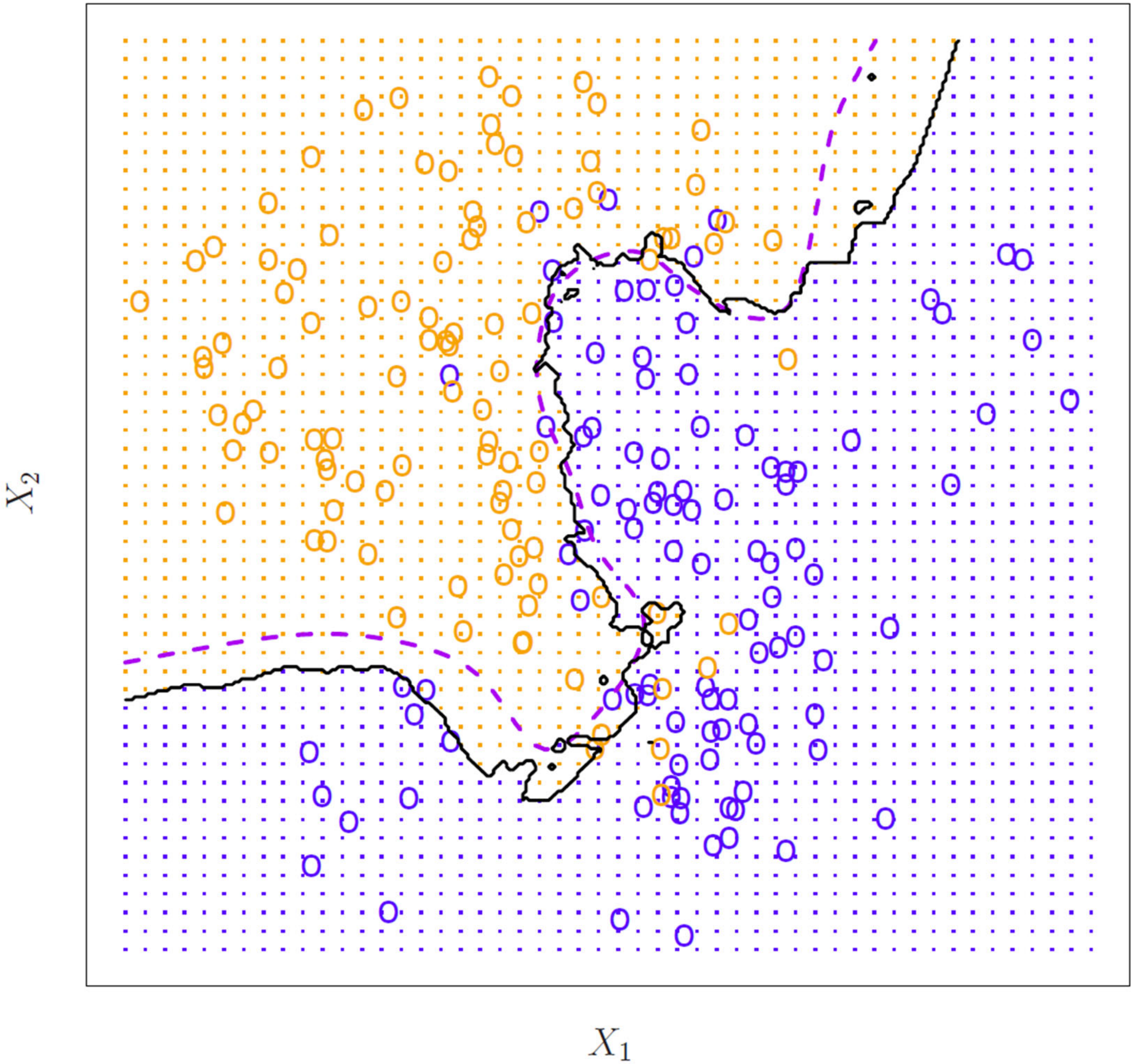


Overfit
high variance

Underfit
high bias

ISLR 2.16

KNN
k = 10



ISLR 2.15

Classification accuracy

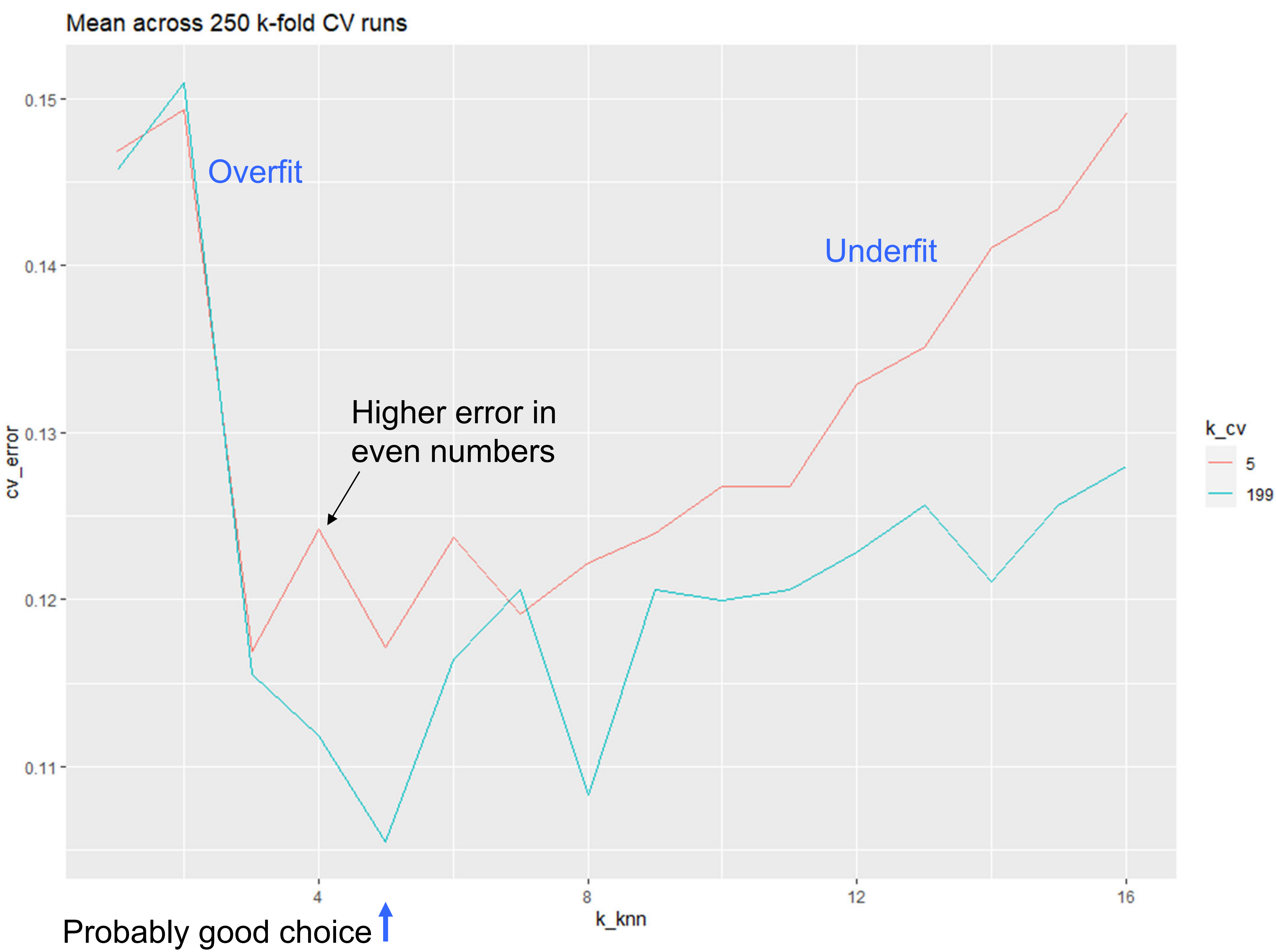
As before: out-of-sample accuracy

A simple measure is the error rate. If we have a *test* dataset of $i = 1 \dots n$ observations, the out-of-sample error rate is:

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

= proportion incorrect

$\hat{y}_i$ is the predicted category for test case i.
$I()$ is an indicator function that equals 1 if the prediction is *incorrect* (i.e. if $y_i \neq \hat{y}_i$) and 0 if the prediction is correct.


Mean across 250 k-fold CV runs
Overfit
Underfit
Higher error in
even numbers
Probably good choice
cv_error
0.15
0.14
0.13
0.12
0.11
k_knn
4
8
12
16
k_cv
5
199