

DATA2002

Nearest neighbours

Garth Tarr



Nearest neighbours
Case study: diabetes

Microchip test data

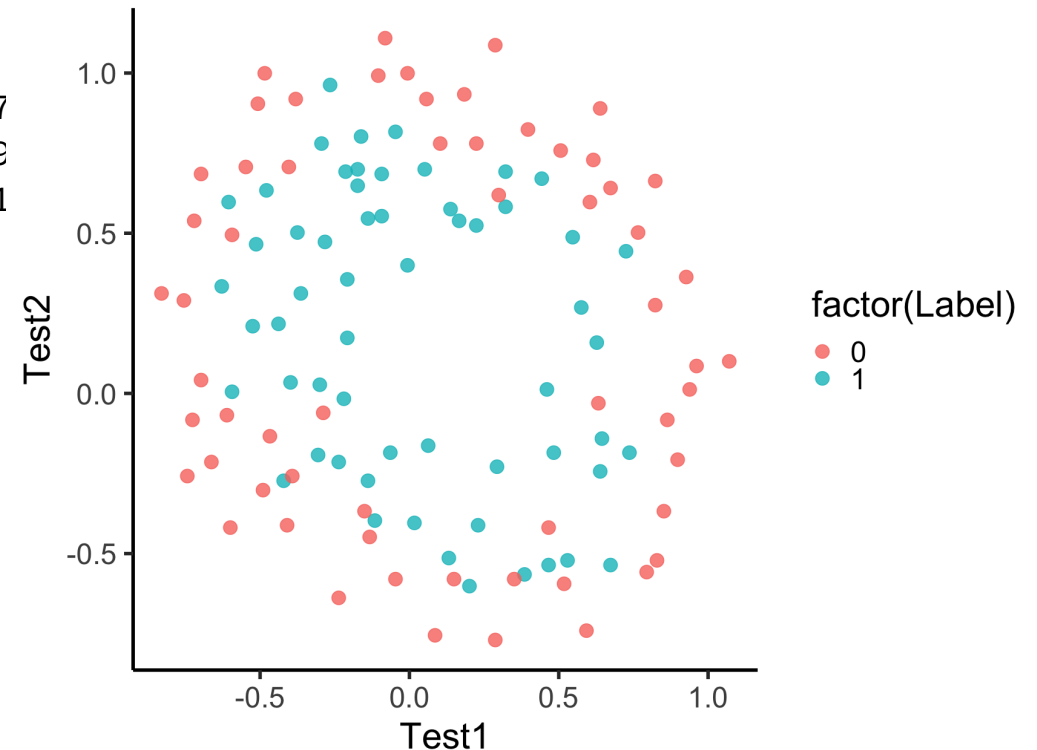
```
library(tidyverse)
data = read_csv("https://raw.githubusercontent.com/
glimpse(data)
```

```
## Rows: 118
## Columns: 3
## $ Test1 <dbl> 0.051267, -0.092742, -0.213710, -0.37
## $ Test2 <dbl> 0.699560, 0.684940, 0.692250, 0.50219
## $ Label <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

The Microchips dataset contains test scores Test1 and Test2, and Label indicating whether or not the chip passed the test.

Why **can't** we use logistic regression or decision trees to predict the class for a new data point?

```
ggplot(data, aes(x=Test1, y=Test2, color=factor
geom_point(size=5, alpha = 0.8) +
theme_classic(base_size=30)
```



Different data types require different approaches

We can use logistic regression to **classify** binary observations, this isn't feasible when:

- We have high class separation in our data (perfect separation)
- We have a non-linear combination of predictors influencing our response

We can use decision trees to **classify** observations into 2 or more classes, but this

- Can be complicated
- Might overfit
- Only draws boundaries parallel to axes

So, what other options do we have?

The k-nearest neighbours algorithm

Data with p attributes (explanatory variables) are points in a p -dimensional space. We can calculate **distances** between these points.

- **k-nearest neighbours (kNN)** is a non-parametric algorithm (doesn't assume the data follows any particular shape).
- In kNN, we vote on the class of a new data point by looking at the majority class of the **k** nearest neighbours.

k-nearest neighbours

The Euclidean distance between any two points is the square root of the sum of the squared deviations:

$$d(\mathbf{x}, \mathbf{z}) = \sqrt{(x_1 - z_1)^2 + (x_2 - z_2)^2 + \dots + (x_p - z_p)^2}$$

- It makes sense to talk about the distance between two observations (rows in the data frame) in p dimensional space.
- **k-nearest neighbours** uses these distances and the understanding that points "close" to each other probably have similar outcomes.

- Suppose we have a set of training data (\mathbf{X}, \mathbf{y}) .
- For some positive integer k , a k -nearest neighbour algorithm classifies a new observation \mathbf{x}^* by:
 1. Finding the k observations in the training data \mathbf{X} , that are closest to \mathbf{x}^* according to some distance metric (usually Euclidian),
 2. Denote the set of k closest observations as $D(\mathbf{x}^*) \subseteq (\mathbf{X}, \mathbf{y})$.
 3. Define an aggregation function f (e.g. the majority rules aggregation function) and compute $y^* = f(y)$ for the k values of y in $D(\mathbf{x}^*)$.

k-nearest neighbours

Note that the kNN classifier doesn't need to pre-process the training data to make predictions - it can do this on the fly.

The distance metric only makes sense for quantitative variables (not categorical variables).

Advantages

- kNN is easy to understand
- Doesn't require any preprocessing time

Disadvantages

- Predictions can be slow (as data is processed at that time)
- Usefulness depends on the geometry of the data: are the points clustered together? What is the distribution of differences among each variable? A wider scale on one variable can dwarf a narrow scale on another variable.

Choosing k

- An appropriate choice of k will depend on the application and the data.
- Cross validation can be used to optimise the choice of k .
- Misclassification rate increases as k increases. This implies that if you're looking to minimise the misclassification rate on a particular data set, the optimal value of k is 1.

Voroni

kNN in R

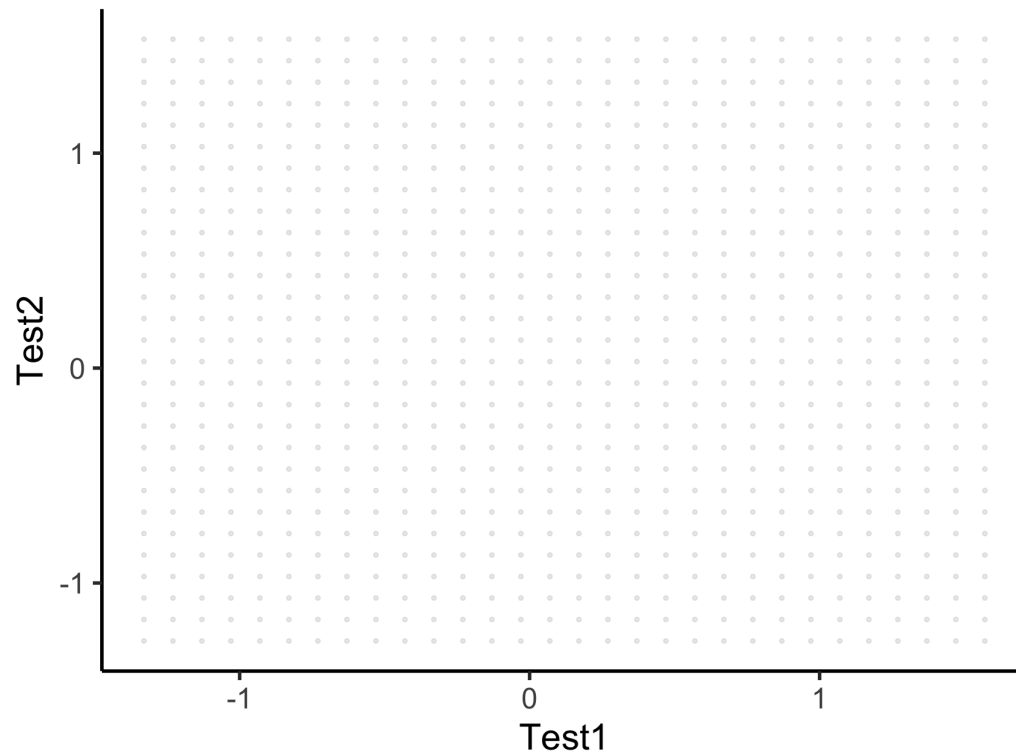
Split our data into **predictors** (X), and response (cl for *class*),

```
X = data %>% dplyr::select(Test1, Test2)
cl = as.factor(data$Label)
```

We will give the function a grid of new data points.

```
new.X = expand.grid(
  Test1 = seq(min(X$Test1 - 0.5),
              max(X$Test1 + 0.5), by = 0.1),
  Test2 = seq(min(X$Test2 - 0.5),
              max(X$Test2 + 0.5), by = 0.1))
```

```
p = ggplot(new.X) +
  geom_point(aes(x=Test1, y=Test2), alpha=0.1)
  theme_classic(base_size = 30)
p
```

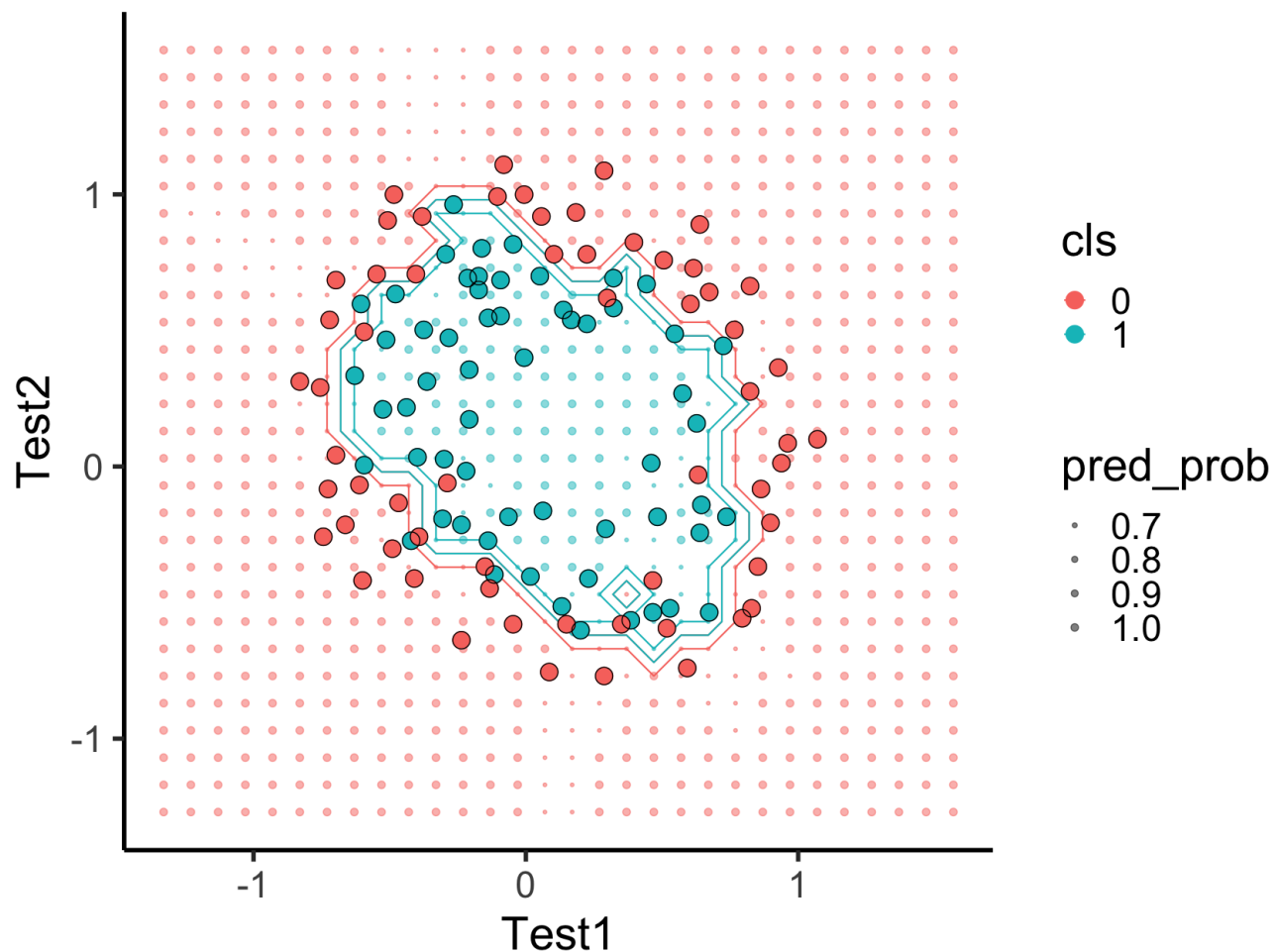


kNN in R

Now we call the `knn` function, putting in our original observations and their observed the classes as well as the grid of "new points" that we'll be predicting.

We also need to specify how many neighbours we want to learn from. For now let $k = 3$.

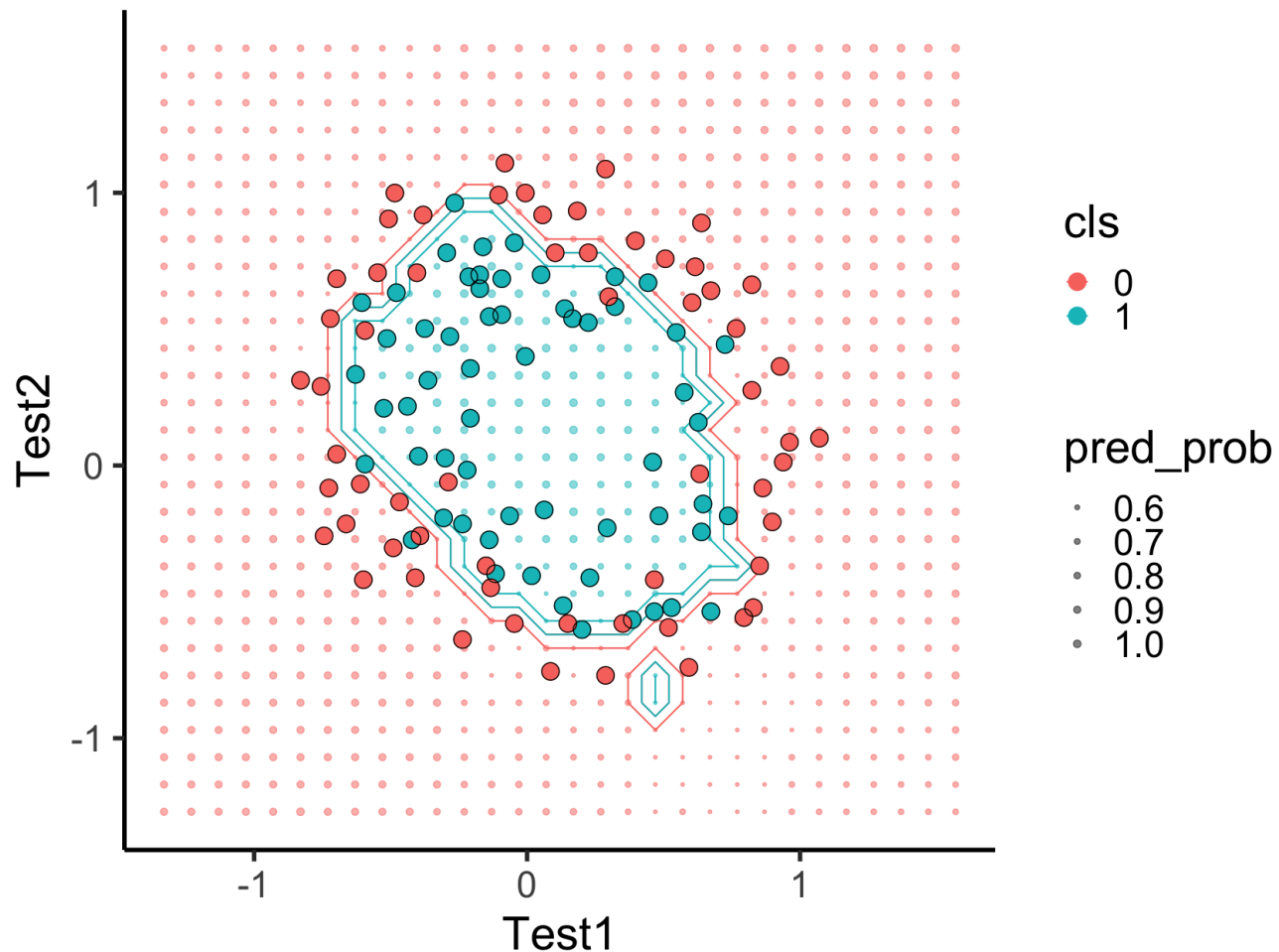
```
library(class)
pred_cl_k3 = knn(X, new.X, cl,
                 k = 3, prob=TRUE)
pred_prob_k3 = attr(pred_cl_k3,
                    "prob")
```



kNN in R

We can also see how our decisions change based on the number of neighbours we consider! Here, we consider $k=9$.

```
pred_cl_k9 = knn(X, new.X, cl,  
                 k = 9, prob=TRUE)  
pred_prob_k9 = attr(pred_cl_k9,  
                    "prob")
```



Performance assessment

Let's compare k=3 and k=9

In sample confusion matrix

```
library(caret)
```

k=3

```
k3 = knn(train = X, test = X, cl = cl, k = 3)  
confusionMatrix(k3, cl)$table
```

```
##           Reference  
## Prediction  0  1  
##           0 50  6  
##           1 10 52
```

```
confusionMatrix(k3, cl)$overall[1] %>% round(2)
```

```
## Accuracy  
##      0.86
```

k=9

```
k9 = knn(train = X, test = X, cl = cl, k = 9)  
confusionMatrix(k9, cl)$table
```

```
##           Reference  
## Prediction  0  1  
##           0 51 11  
##           1  9 47
```

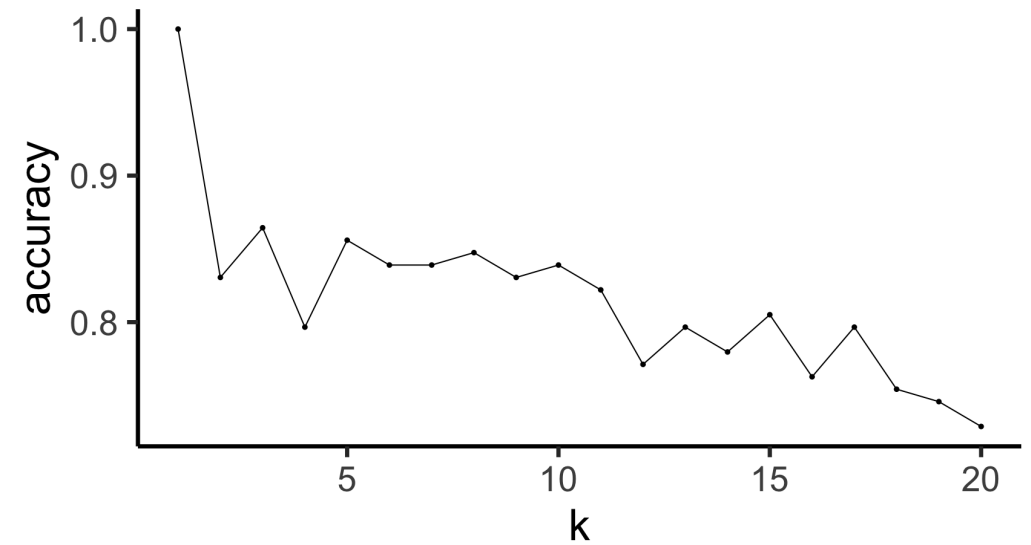
```
confusionMatrix(k9, cl)$overall[1] %>% round(2)
```

```
## Accuracy  
##      0.83
```

Let's compare lots of k! (Still in-sample)

```
res = data.frame(  
  k = 1:20,  
  accuracy = NA  
)  
for(k in res$k){  
  k_temp = knn(train = X, test = X,  
               cl = cl, k = k)  
  cmat = confusionMatrix(k_temp, cl)  
  res$accuracy[k] = cmat$overall[1]  
}
```

```
ggplot(res, aes(x = k, y = accuracy)) +  
  geom_point() + geom_line() +  
  theme_classic(base_size = 36)
```



What's with the saw-tooth pattern?!?!?

Out of sample 5 fold CV

For $k = 3$

```
n = length(cl)
n
```

```
## [1] 118
```

```
n/5
```

```
## [1] 23.6
```

```
set.seed(1)
fold_id = sample(c(1,2,3,rep(1:5, each = 23)))
length(fold_id)
```

```
## [1] 118
```

```
cv_acc = NA
```

```
for(i in 1:5){
  k_temp = knn(train = X[fold_id!=i,],
               test = X[fold_id==i,],
               cl = cl[fold_id!=i], k = 3)
  cmat = confusionMatrix(k_temp,
                        cl[fold_id==i])
  cv_acc[i] = cmat$overall[1]
}
mean(cv_acc)
```

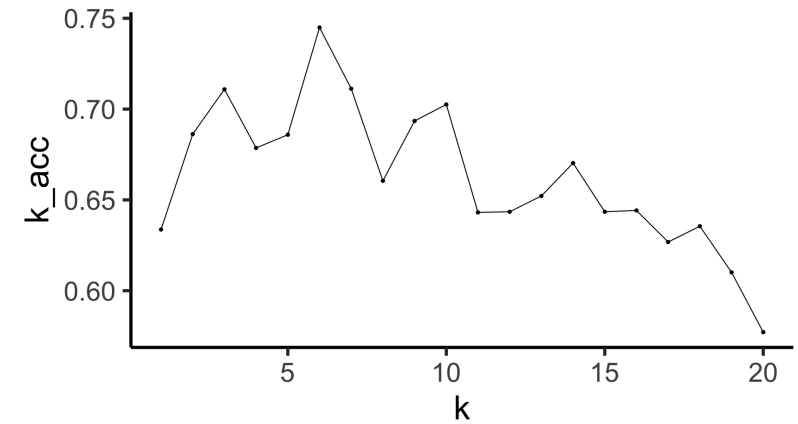
```
## [1] 0.7108696
```

Out of sample 5 fold CV

To help choose k

```
set.seed(1)
fold_id = sample(c(1,2,3,rep(1:5, each = 23)))
cv_acc_k = matrix(nrow = 20, ncol = 5)
for(k in 1:20){
  for(i in 1:5){
    k_temp = knn(train = X[fold_id!=i,],
                  test = X[fold_id==i,],
                  cl = cl[fold_id!=i], k = k)
    cmat = confusionMatrix(k_temp, cl[fold_id==i])
    cv_acc_k[k,i] = cmat$overall[1]
  }
}
cv_res = data.frame(
  k = 1:20,
  k_acc = apply(cv_acc_k, 1, mean)
)
```

```
ggplot(cv_res, aes(x = k, y = k_acc))
  geom_point() + geom_line() +
  theme_classic(base_size = 36)
```

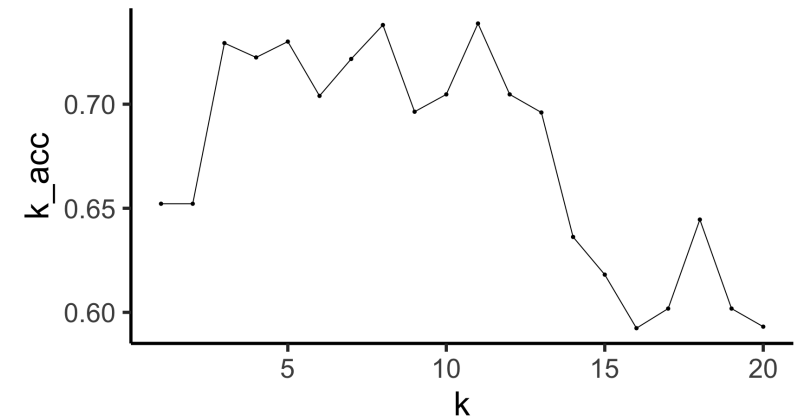


Out of sample 5 fold CV

To help choose k

```
set.seed(2)
fold_id = sample(c(1,2,3,rep(1:5, each = 23)))
cv_acc_k = matrix(nrow = 20, ncol = 5)
for(k in 1:20){
  for(i in 1:5){
    k_temp = knn(train = X[fold_id!=i,],
                  test = X[fold_id==i,],
                  cl = cl[fold_id!=i], k = k)
    cmat = confusionMatrix(k_temp, cl[fold_id==i])
    cv_acc_k[k,i] = cmat$overall[1]
  }
}
cv_res = data.frame(
  k = 1:20,
  k_acc = apply(cv_acc_k, 1, mean)
)
```

```
ggplot(cv_res, aes(x = k, y = k_acc))
  geom_point() + geom_line() +
  theme_classic(base_size = 36)
```

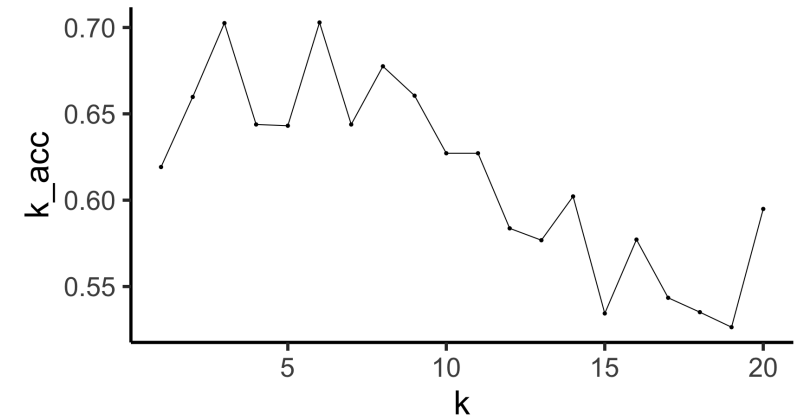


Out of sample 5 fold CV

To help choose k

```
set.seed(3)
fold_id = sample(c(1,2,3,rep(1:5, each = 23)))
cv_acc_k = matrix(nrow = 20, ncol = 5)
for(k in 1:20){
  for(i in 1:5){
    k_temp = knn(train = X[fold_id!=i,],
                  test = X[fold_id==i,],
                  cl = cl[fold_id!=i], k = k)
    cmat = confusionMatrix(k_temp, cl[fold_id==i])
    cv_acc_k[k,i] = cmat$overall[1]
  }
}
cv_res = data.frame(
  k = 1:20,
  k_acc = apply(cv_acc_k, 1, mean)
)
```

```
ggplot(cv_res, aes(x = k, y = k_acc))
  geom_point() + geom_line() +
  theme_classic(base_size = 36)
```

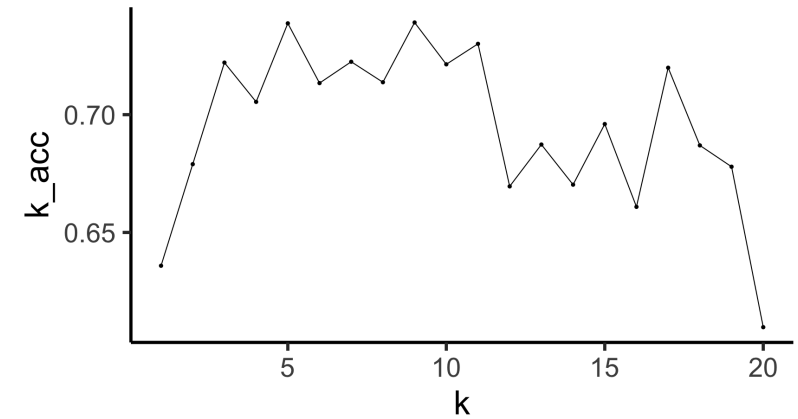


Out of sample 5 fold CV

To help choose k

```
set.seed(4)
fold_id = sample(c(1,2,3,rep(1:5, each = 23)))
cv_acc_k = matrix(nrow = 20, ncol = 5)
for(k in 1:20){
  for(i in 1:5){
    k_temp = knn(train = X[fold_id!=i,],
                  test = X[fold_id==i,],
                  cl = cl[fold_id!=i], k = k)
    cmat = confusionMatrix(k_temp, cl[fold_id==i])
    cv_acc_k[k,i] = cmat$overall[1]
  }
}
cv_res = data.frame(
  k = 1:20,
  k_acc = apply(cv_acc_k, 1, mean)
)
```

```
ggplot(cv_res, aes(x = k, y = k_acc))
  geom_point() + geom_line() +
  theme_classic(base_size = 36)
```

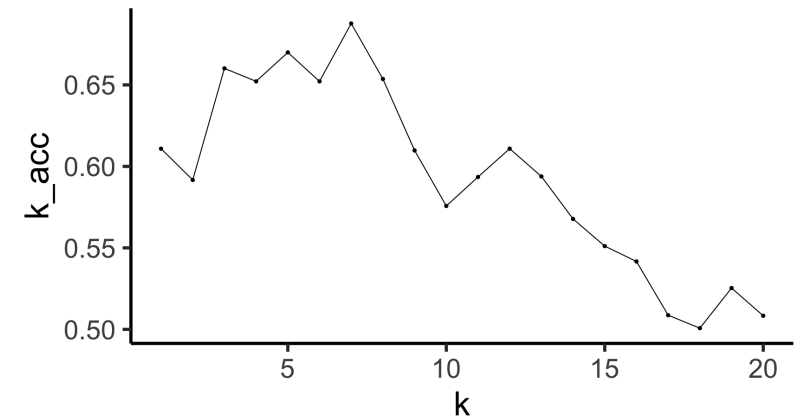


Out of sample 5 fold CV

To help choose k

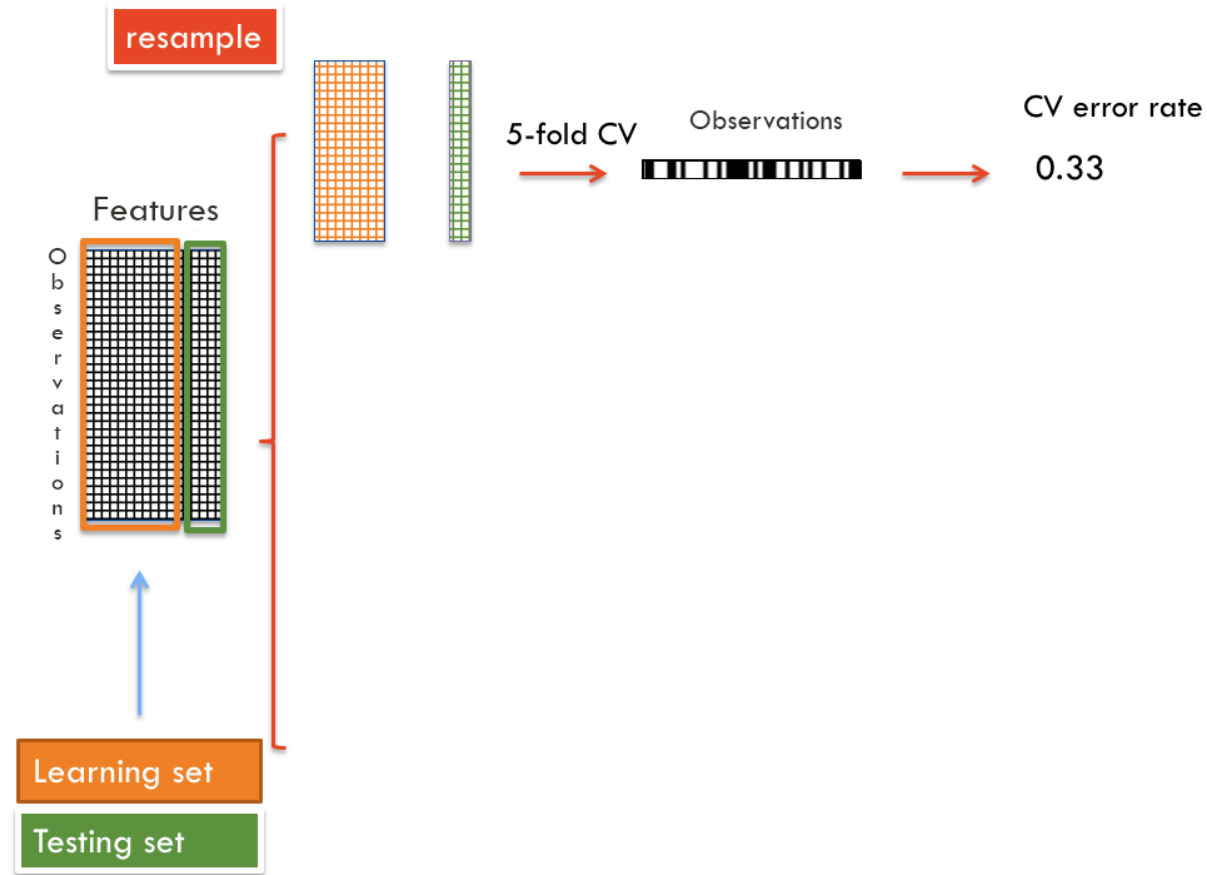
```
set.seed(5)
fold_id = sample(c(1,2,3,rep(1:5, each = 23)))
cv_acc_k = matrix(nrow = 20, ncol = 5)
for(k in 1:20){
  for(i in 1:5){
    k_temp = knn(train = X[fold_id!=i,],
                  test = X[fold_id==i,],
                  cl = cl[fold_id!=i], k = k)
    cmat = confusionMatrix(k_temp, cl[fold_id==i])
    cv_acc_k[k,i] = cmat$overall[1]
  }
}
cv_res = data.frame(
  k = 1:20,
  k_acc = apply(cv_acc_k, 1, mean)
)
```

```
ggplot(cv_res, aes(x = k, y = k_acc))
  geom_point() + geom_line() +
  theme_classic(base_size = 36)
```



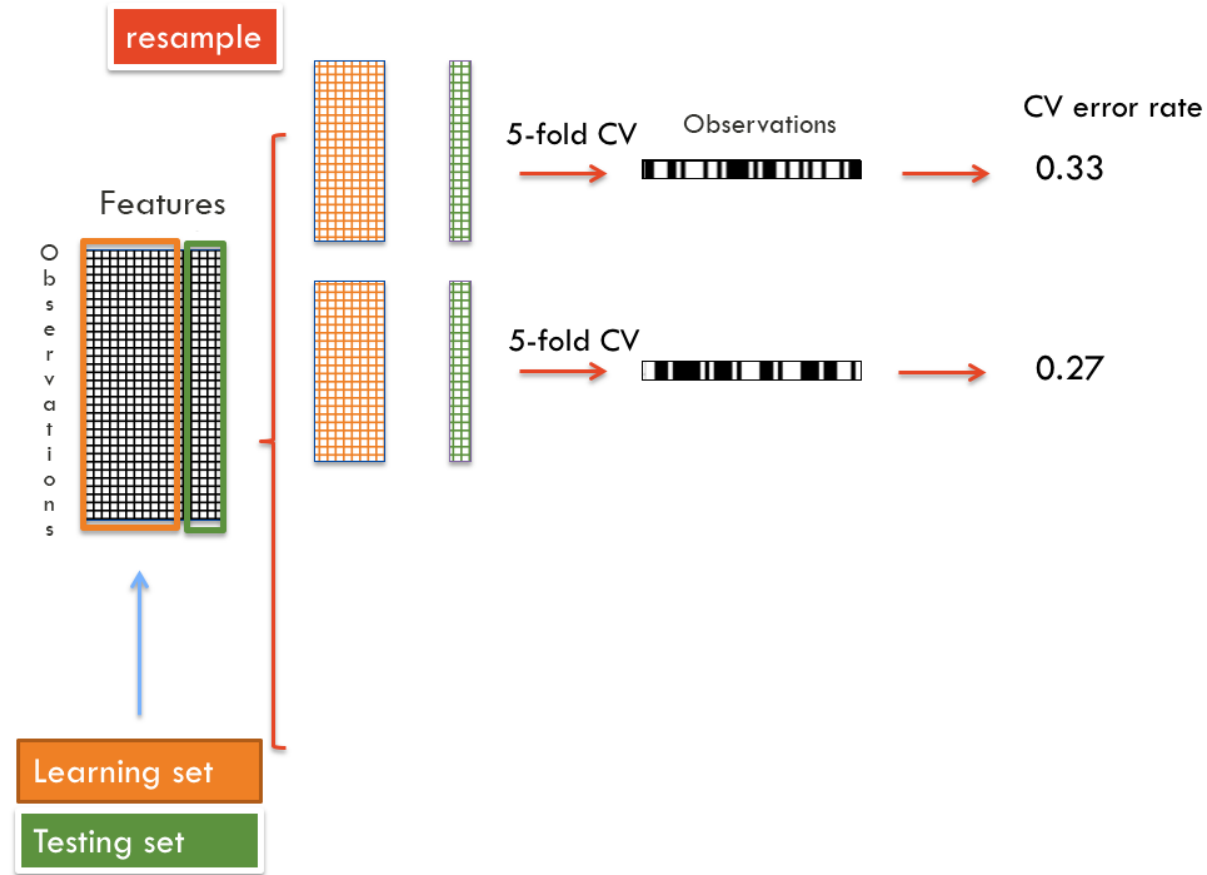
Repeated K fold CV

Often we want to repeat the CV process, to get a range of error values.



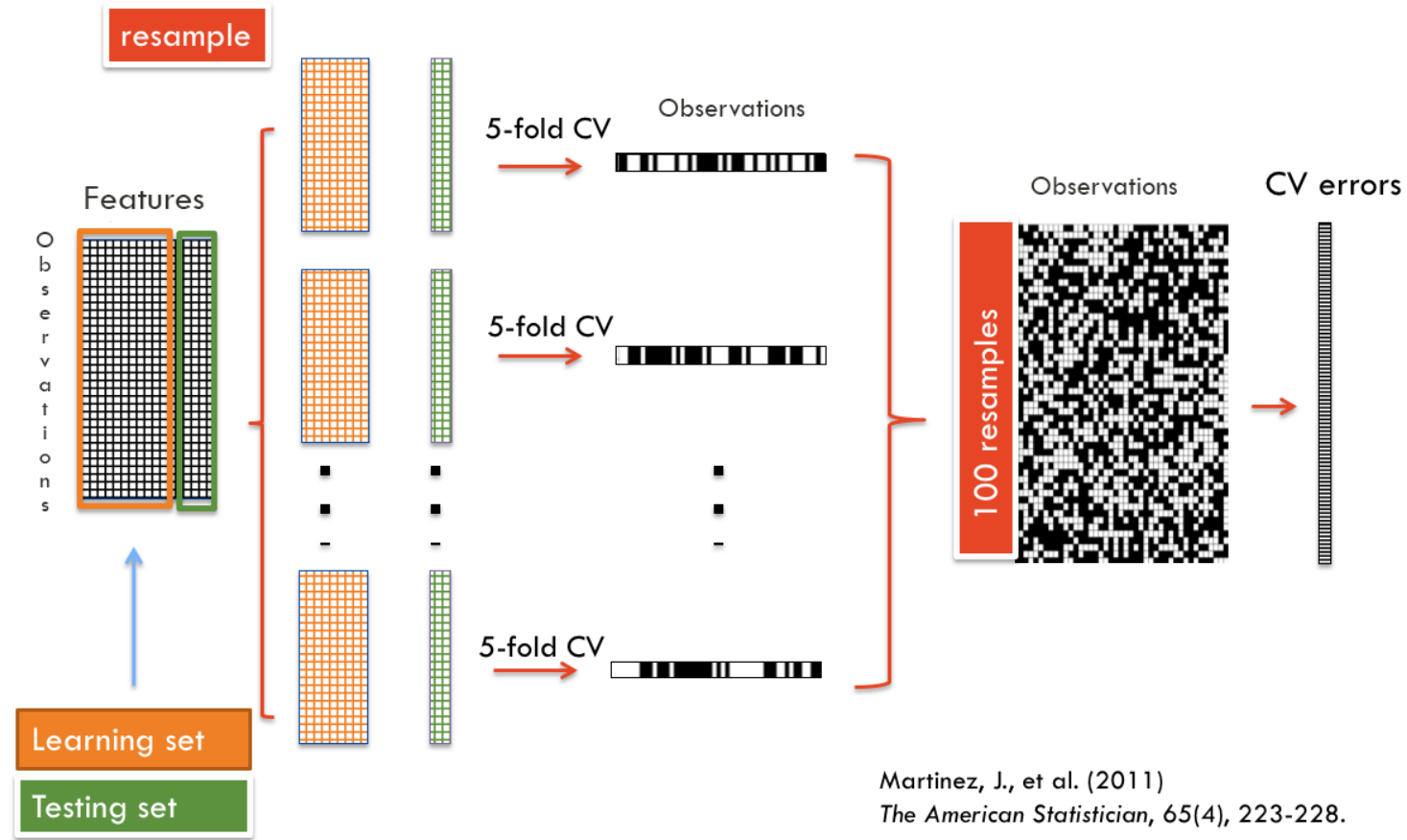
Repeated K fold CV

Often we want to repeat the CV process, to get a range of error values.



Repeated K fold CV

Often we want to repeat the CV process, to get a range of error values.

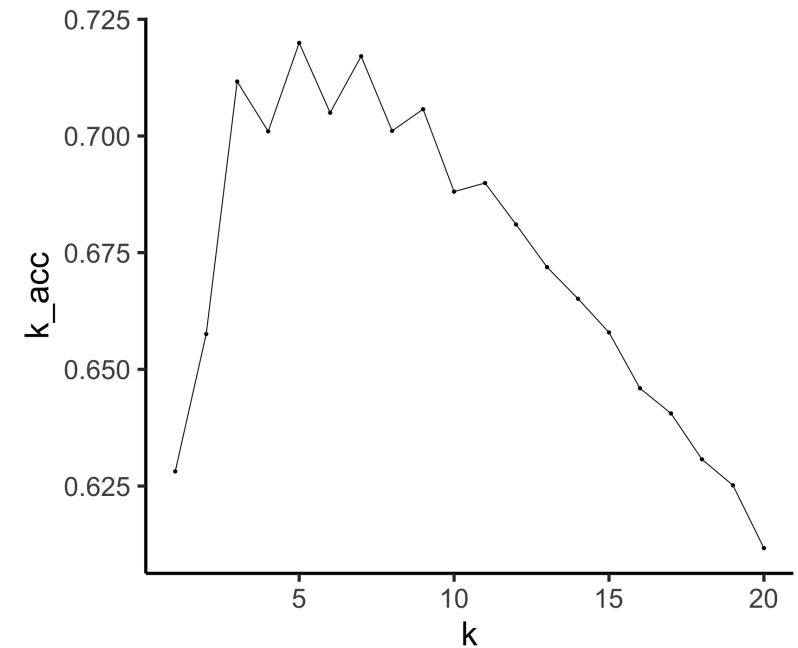


Repeated 5 fold CV

To help choose k

```
set.seed(1)
cv_acc_k = array(dim = c(20, 5, 100))
for(j in 1:100){
  fold_id = sample(c(1,2,3,rep(1:5, each = 23)))
  for(k in 1:20){
    for(i in 1:5){
      k_temp = knn(train = X[fold_id!=i,],
                   test = X[fold_id==i,],
                   cl = cl[fold_id!=i], k = k)
      cmat = confusionMatrix(k_temp, cl[fold_id==i])
      cv_acc_k[k,i,j] = cmat$overall[1]
    }
  }
}
cv_res = data.frame(
  k = 1:20,
  k_acc = apply(cv_acc_k, 1, mean),
  k_acc_n = apply(cv_acc_k, 1, length)
)
```

```
ggplot(cv_res, aes(x = k, y = k_acc))
  geom_point() + geom_line() +
  theme_classic(base_size = 36)
```

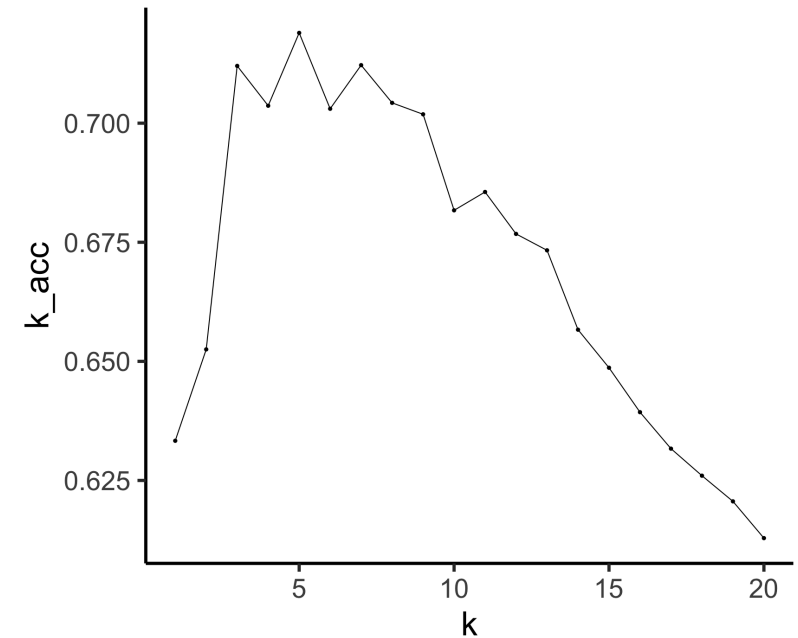


Repeated 5 fold CV

To help choose k

```
set.seed(2)
cv_acc_k = array(dim = c(20, 5, 100))
for(j in 1:100){
  fold_id = sample(c(1,2,3,rep(1:5, each = 23)))
  for(k in 1:20){
    for(i in 1:5){
      k_temp = knn(train = X[fold_id!=i,],
                  test = X[fold_id==i,],
                  cl = cl[fold_id!=i], k = k)
      cmat = confusionMatrix(k_temp, cl[fold_id==i])
      cv_acc_k[k,i,j] = cmat$overall[1]
    }
  }
}
cv_res = data.frame(
  k = 1:20,
  k_acc = apply(cv_acc_k, 1, mean),
  k_acc_n = apply(cv_acc_k, 1, length)
)
```

```
ggplot(cv_res, aes(x = k, y = k_acc))
  geom_point() + geom_line() +
  theme_classic(base_size = 36)
```



Using the caret package

```
library(caret)
fitCtrl = trainControl(method = "repeatedcv", number = 5, repeats = 100)
set.seed(1)
knnFit1 = caret::train(factor(Label) ~ ., data = data, method = "knn", trControl = fitCtrl)
knnFit1
```

```
## k-Nearest Neighbors
##
## 118 samples
## 2 predictor
## 2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 100 times)
## Summary of sample sizes: 94, 95, 94, 95, 94, 94, ...
## Resampling results across tuning parameters:
##
##  k  Accuracy  Kappa
##  5  0.7254891  0.4507026
##  7  0.7228406  0.4455525
##  9  0.7162029  0.4325306
##
## Accuracy was used to select the optimal model using
```

```
confusionMatrix(knnFit1)
```

```
## Cross-Validated (5 fold, repeated 100 times) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    0     1
##           0 36.8 13.4
##           1 14.0 35.7
##
## Accuracy (average) : 0.7255
```

Case study: Diabetes data

Case study: Diabetes data

```
diabetes = read_csv("https://raw.githubusercontent.com/DATA2002/data/master/diabetes.csv")
glimpse(diabetes)
```

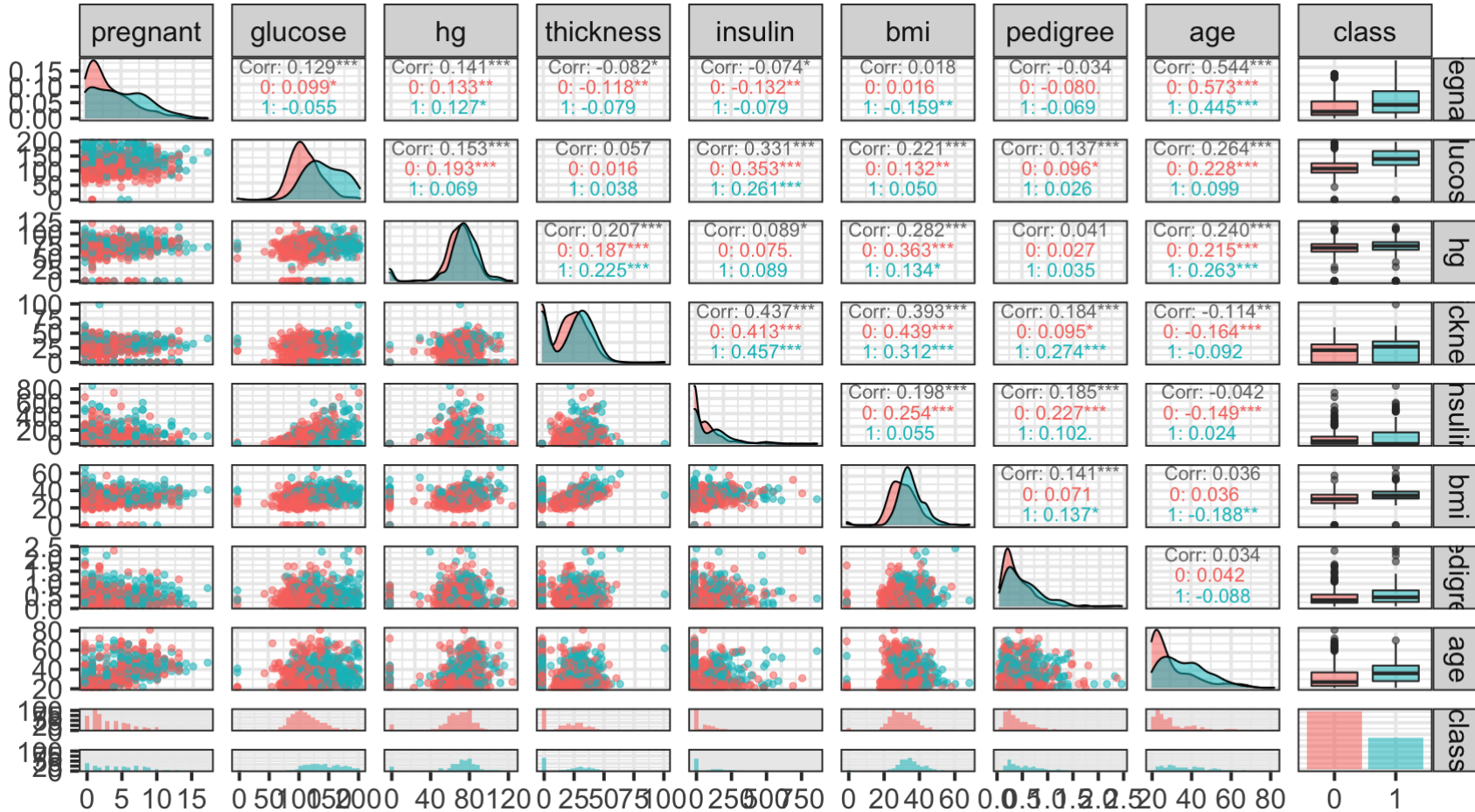
```
## Rows: 768
## Columns: 9
## $ pregnant <dbl> 6, 1, 8, 1, 0, 5, 3, 10, 2, 8, 4, 10, 10...
## $ glucose <dbl> 148, 85, 183, 89, 137, 116, 78, 115, 197...
## $ hg <dbl> 72, 66, 64, 66, 40, 74, 50, 0, 70, 96, 9...
## $ thickness <dbl> 35, 29, 0, 23, 35, 0, 32, 0, 45, 0, 0, 0...
## $ insulin <dbl> 0, 0, 0, 94, 168, 0, 88, 0, 543, 0, 0, 0...
## $ bmi <dbl> 33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31.0...
## $ pedigree <dbl> 0.627, 0.351, 0.672, 0.167, 2.288, 0.201...
## $ age <dbl> 50, 31, 32, 21, 33, 30, 26, 29, 53, 54, ...
## $ class <dbl> 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1...
```

```
diabetes$class = factor(diabetes$class)
```

Aim: to build a model to predict whether or not the patients in the dataset have diabetes (`class`).

```
library(GGally)
```

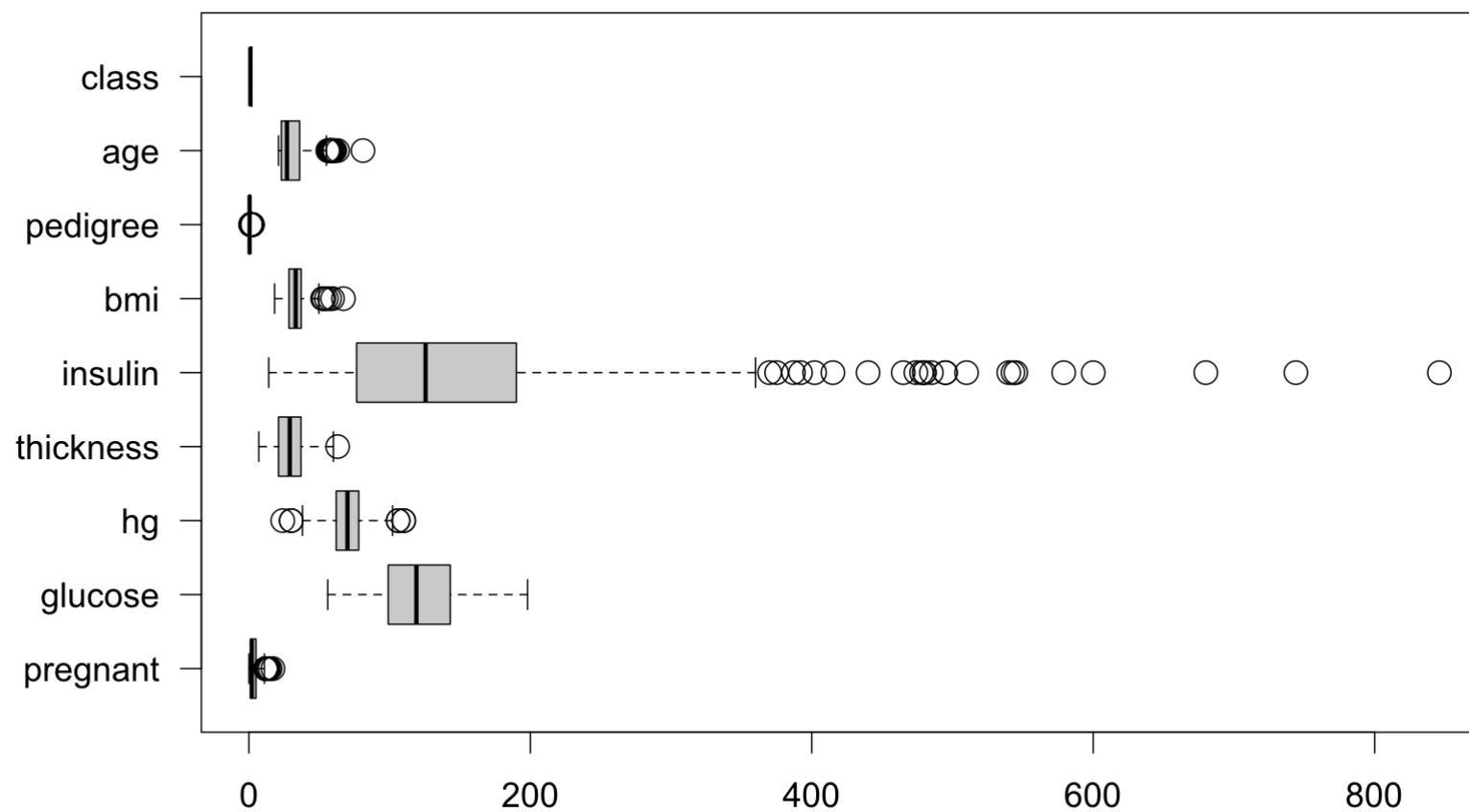
```
ggpairs(diabetes, aes(colour=class, alpha = 0.5)) + theme_bw(base_size = 20)
```



Some unusual zero values...

```
diabetes = diabetes %>%  
  mutate(across(  
    c(glucose, hg, thickness, insulin, bmi), .fns = ~ na_if(., 0))) %>%  
  drop_na()  
ggpairs(diabetes, aes(colour=class, alpha = 0.5)) + theme_bw(base_size = 20)
```

```
par(mar = c(4,8,1,1), cex = 1.5)  
boxplot(diabetes, horizontal = TRUE, las = 1)
```



Normalisation

We need to **normalise** our predictors:

```
norm_fn = function(x) { (x - min(x))/(max(x) - min(x)) }  
d_norm = diabetes %>%  
  mutate(across(where(is.numeric), .fns = norm_fn))  
par(mar = c(4,8,1,1), cex = 2)  
boxplot(d_norm, horizontal = TRUE, las = 1)
```

5-fold CV for Diabetes knn model

Let's perform a 5-fold CV for the knn model. First, we split up our response and predictors.

```
library(class) # for kNN function
K = 5 # number of folds
n = nrow(d_norm) # number of data points
X = d_norm %>% select(-class) # predictors
y = d_norm %>% select(class) %>% pull()
```

K-fold CV in R using `cvTools`

Using the `cvTools` package, we can use the function `cvFolds` to separate our n data points into K partitions for cross validation.

```
library(cvTools)
set.seed(1)
cvSets = cvFolds(n,K)
glimpse(cvSets)
```

```
## List of 5
## $ n      : num 392
## $ K      : num 5
## $ R      : num 1
## $ subsets: int [1:392, 1] 324 167 129 299 270 187 307 85 277 362 ...
## $ which  : int [1:392] 1 2 3 4 5 1 2 3 4 5 ...
## - attr(*, "class")= chr "cvFolds"
```

- The important outputs are `subsets` and `which`.
- For each fold (given by `which`), `subset` tells us which index of the data belongs to what fold. E.g., observation 204 belongs to fold 1, etc

K-fold CV in R using cvTools

```
error.fold = NA
for(j in 1:K){

  test.inds = cvSets$subsets[cvSets$which == j]

  X.test = X[test.inds,]
  X.train = X[-test.inds,]
  y.test = y[test.inds]
  y.train = y[-test.inds]

  fit = knn(X.train, X.test, y.train, k=9)

  error.fold[j] = sum(fit != y.test)
}

cv.error = sum(error.fold)/n
cv.error
```

```
## [1] 0.2244898
```

```
glimpse(cvSets)
```

```
## List of 5
## $ n      : num 392
## $ K      : num 5
## $ R      : num 1
## $ subsets: int [1:392, 1] 324 167 129
## $ which  : int [1:392] 1 2 3 4 5 1 2
## - attr(*, "class")= chr "cvFolds"
```

Task 1

We want to know which value of k (neighbours) from `k.vals = 1:50` fits the data best. Embed the CV loop on the previous slide in another loop to determine which value of k gives the lowest error.

Task 2

Fit a CV loop for `randomForest`. Does it perform better than the best value of k you found in Task 1?

Repeated K-fold CV

```
set.seed(1)
reps = 10
cv.error = c()
for(i in 1:reps){
  cvSets = cvFolds(n,K)

  error.fold = c()
  for(j in 1:K){

    fold.j = which(cvSets$which==j)
    test.inds = cvSets$subsets[fold.j]

    X.test = X[test.inds,]
    X.train = X[-test.inds,]
    y.test = y[test.inds]
    y.train = y[-test.inds]

    fit = knn(X.train, X.test, y.train, k=9)

    error.fold[j] = sum(fit!=y.test)
  }
  cv.error[i] = sum(error.fold)/n
}
```

- To perform a **repeated** cross validation, all we do is embed our existing CV loop into another loop that repeats the process `reps` times.
- We can then easily visualise our repeated cross validation errors!



Task 3

Perform a 10 repeat, 5 fold Cross Validation, for `randomForest`.

Task 4

Boxplot the results of `knn` (with best `k`) vs `randomForest`. Which model performs better?

Using the caret package

- Using the caret package, we first set up details of CV, including how many repeats we want, and how many folds.
- Next, we specify the response and its predictors, as well as what method we want to use, and the training parameters.

```
fitCtrl = trainControl(  
  method = "repeatedcv",  
  number = 5,  
  repeats = 10)  
set.seed(1)  
knnFit1 = train(  
  class ~ ., data = d_norm,  
  method = "knn",  
  trControl = fitCtrl)
```

knnFit1

```
## k-Nearest Neighbors  
##  
## 392 samples  
## 8 predictor  
## 2 classes: '0', '1'  
##  
## No pre-processing  
## Resampling: Cross-Validated (5 fold, repeated 10 times)  
## Summary of sample sizes: 314, 314, 314, 313, 313, 313, ...  
## Resampling results across tuning parameters:  
##  
## k Accuracy Kappa  
## 5 0.7671081 0.4496161  
## 7 0.7681207 0.4473379  
## 9 0.7648458 0.4347491  
##  
## Accuracy was used to select the optimal model using  
## the largest value.  
## The final value used for the model was k = 7.
```


kNN

- kNN is considered a **lazy** learning algorithm
- Data processing is deferred until it receives a request to classify an unlabelled example.
- The constructed algorithm is discarded along with any intermediate results

Advantages

- Analytically tractable and simple implementation
- Performance improves as the sample size grows
- Uses **local** information, and is highly adaptive
- Can be easily parallelised

Disadvantages

- Large storage requirements (need to store full data set)
- Computationally intensive when predicting new observations

References

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning*. 2nd ed. Springer Series in Statistics. New York, NY, USA: Springer. URL: <https://web.stanford.edu/~hastie/ElemStatLearn/>.


James, G., D. Witten, T. Hastie, and R. Tibshirani (2017). *An Introduction to Statistical Learning: With Applications in R*. New York: Springer. URL: <https://www-bcf.usc.edu/~gareth/ISL/>.

Jed Wing, M. K. C. from, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, the R Core Team, M. Benesty, et al. (2018). *caret: Classification and Regression Training*. R package version 6.0-80. URL: <https://CRAN.R-project.org/package=caret>.

Liaw, A. and M. Wiener (2002). "Classification and Regression by randomForest". In: *R News* 2.3, pp. 18-22. URL: <https://CRAN.R-project.org/doc/Rnews/>.

Therneau, T. and B. Atkinson (2018). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-13. URL: <https://CRAN.R-project.org/package=rpart>.

Acknowledgements

Special thanks to [Sarah Romanes](#)  for providing content inspiration from her R Ladies talks on machine learning:

- sarahromanes.github.io/r-ladies-ML-1
- sarahromanes.github.io/r-ladies-ML-2