DATA SOCIETY®

Week 7 Day 1 - Intro to classification

"One should look for what is and not what he thinks should be."
-Albert Einstein.

Module completion checklist

Objective	Complete
Summarize the steps & application of kNN	
Clean and transform the data to run kNN	
Define cross validation and how and when it is used	
Implement the kNN algorithm on the training data without cross-validation	
Identify performance metrics for classification algorithms	
Evaluate the optimal number of nearest neighbors to use using cross-validation	
Evaluate performance of optimized kNN model	
Apply the knn and evalute its performance on the CMP dataset	

Directory settings

- First, let's make sure to set our directories correctly, this way, we will not have to worry about this throughout the course

```
# Set `main_dir` to the location of your `hhs-r` folder (for Mac/Linux).
main_dir = "~/Desktop/hhs-r-2020"
# Set `main_dir` to the location of your `hhs-r` folder (for Windows).
main_dir = "C:/Users/[username]/Desktop/hhs-r-2020"
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = paste0(main_dir, "/data")
# Make `plots_dir` from the `main_dir` and
# remainder of the path to plots directory.
plot_dir = paste0(main_dir, "/plots")
```

Loading packages

Let's load the packages we will be using:

```
#install.packages("ROCR")
#install.packages("e1071")
library(e1071)
library(caret)
library(ROCR)
```

kNN : what is it?

- The k-nearest-neighbors (kNN) algorithm is a **supervised learning** algorithm.
- It is primarily used for **Classification**.
- It takes a bunch of **labeled points** and uses them to learn how to label other points.

Classification vs. regression

	Classification	Regression
Target variable	Discrete, usually binary	Continuous
Types	Binary, Multi-Class	Linear
Algorithms	Decision trees, random forest, logistic regression, K-nearest neighbors	Linear regression, regression trees, time-series regression

Classification: assigning to groups

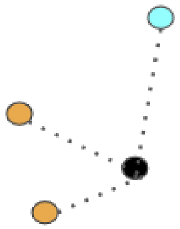
Question to answer	Real world example
What is this object like?	Selecting similar products at the lowest prices
Who is this person like?	Anticipating behavior or preferences of a person based on her similarities with others
What category is this in?	Anticipate if your customer is pregnant, remodeling, just got married, etc.
What is the probability that something is in a given category?	Determine the probabilitythat a piece of equipment will fail, determine the probability that someone will buy your product

Steps of kNN

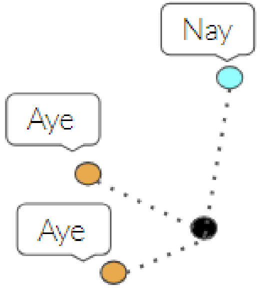
Step 1:
Select
K

k= 4? 16? 3?

Step 2:
Measure
Distance



Step 3:
Majority
Vote

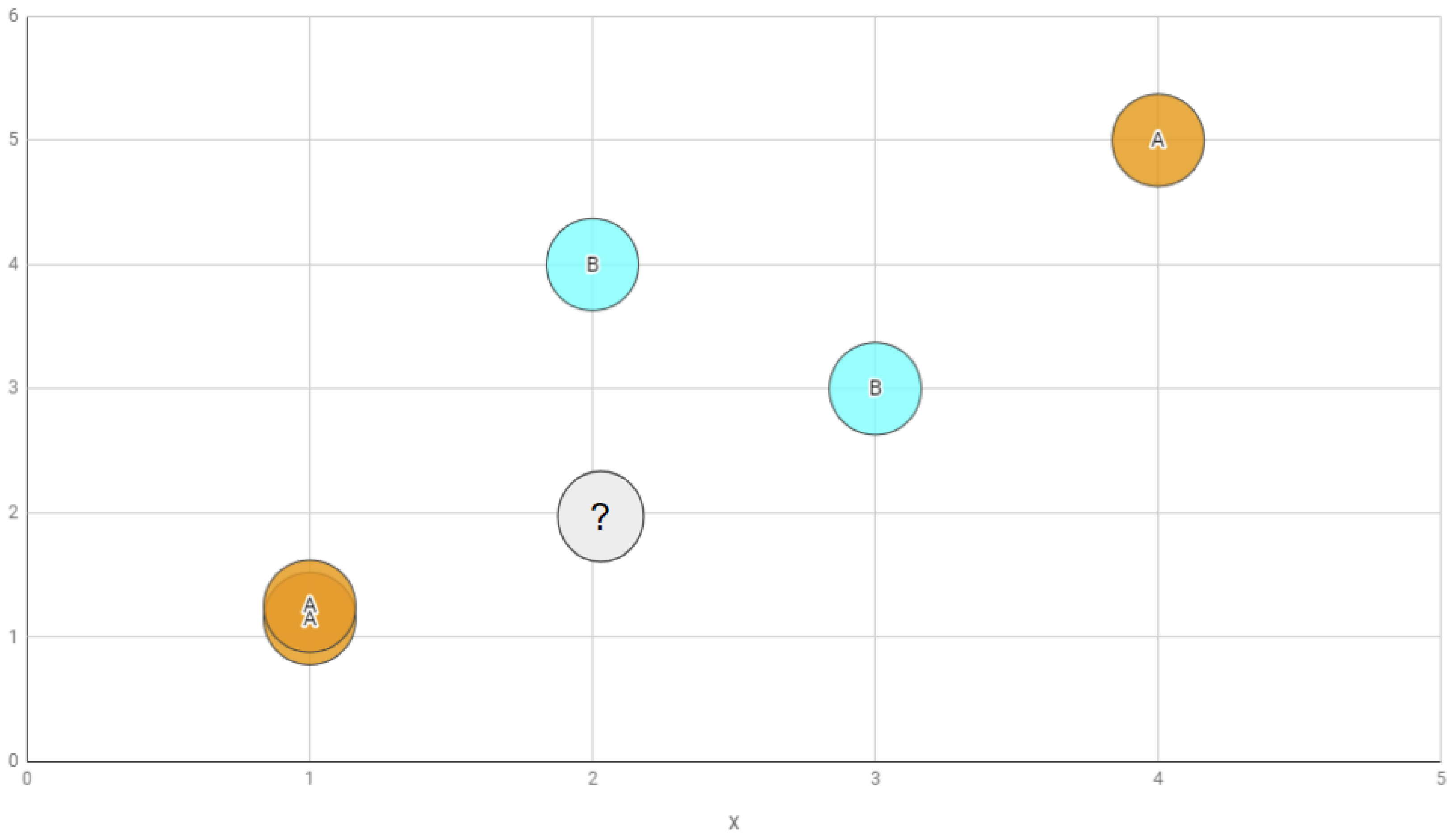


Step 4:
Check
Performance Metrics

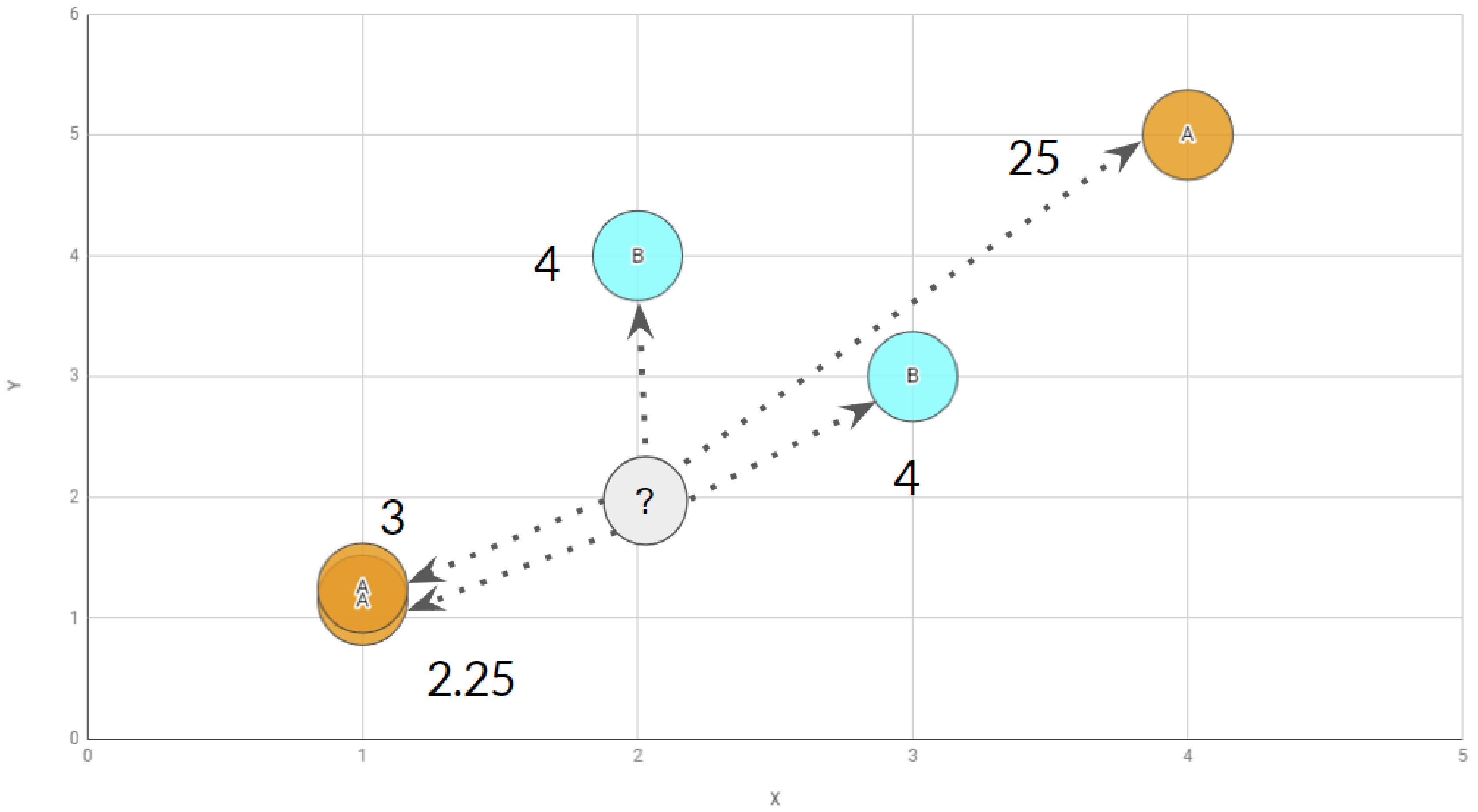
	Act +	Act -	
Pred +			
Pred -			

Step 5:
Find
Best K

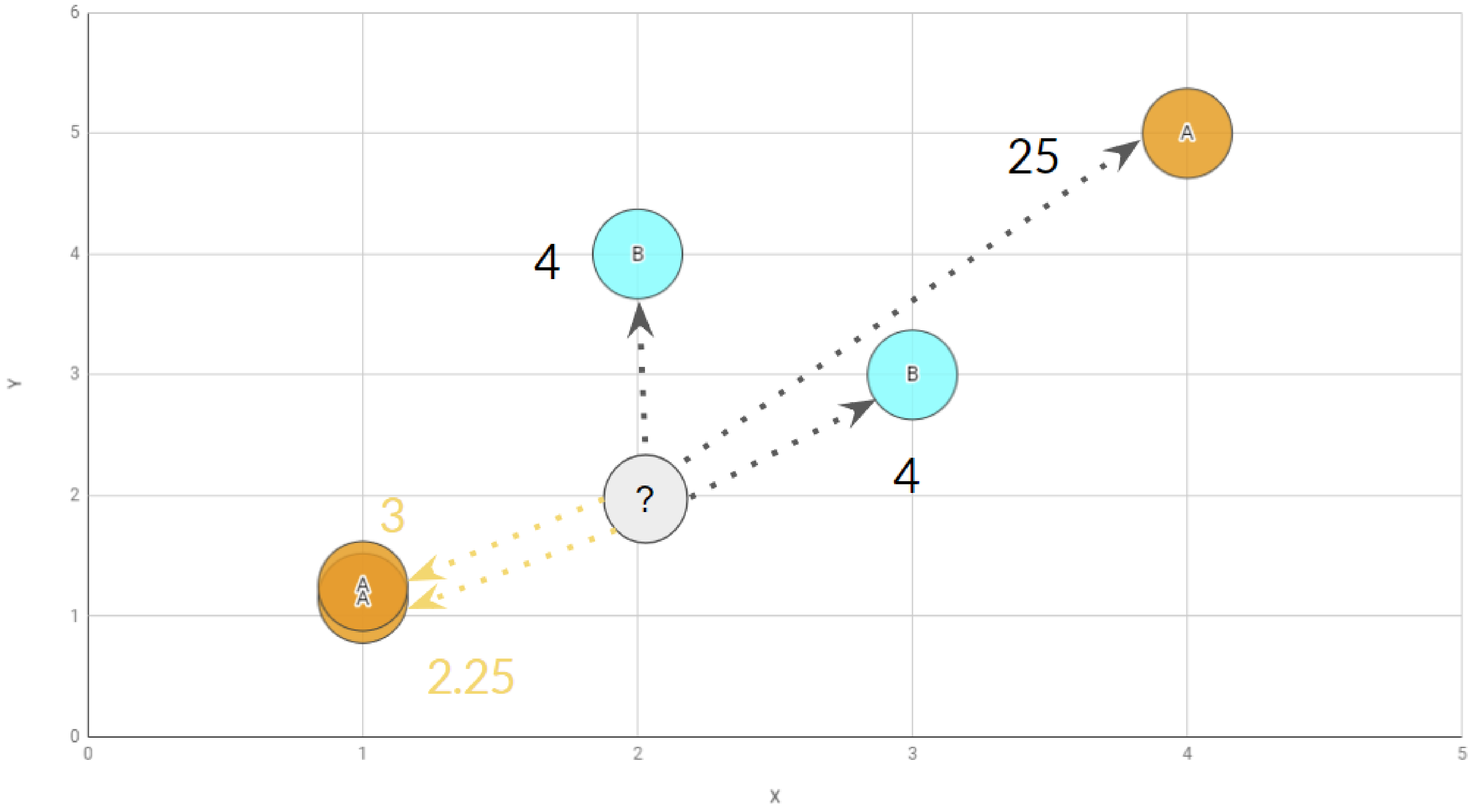
k-Nearest Neighbors: set up



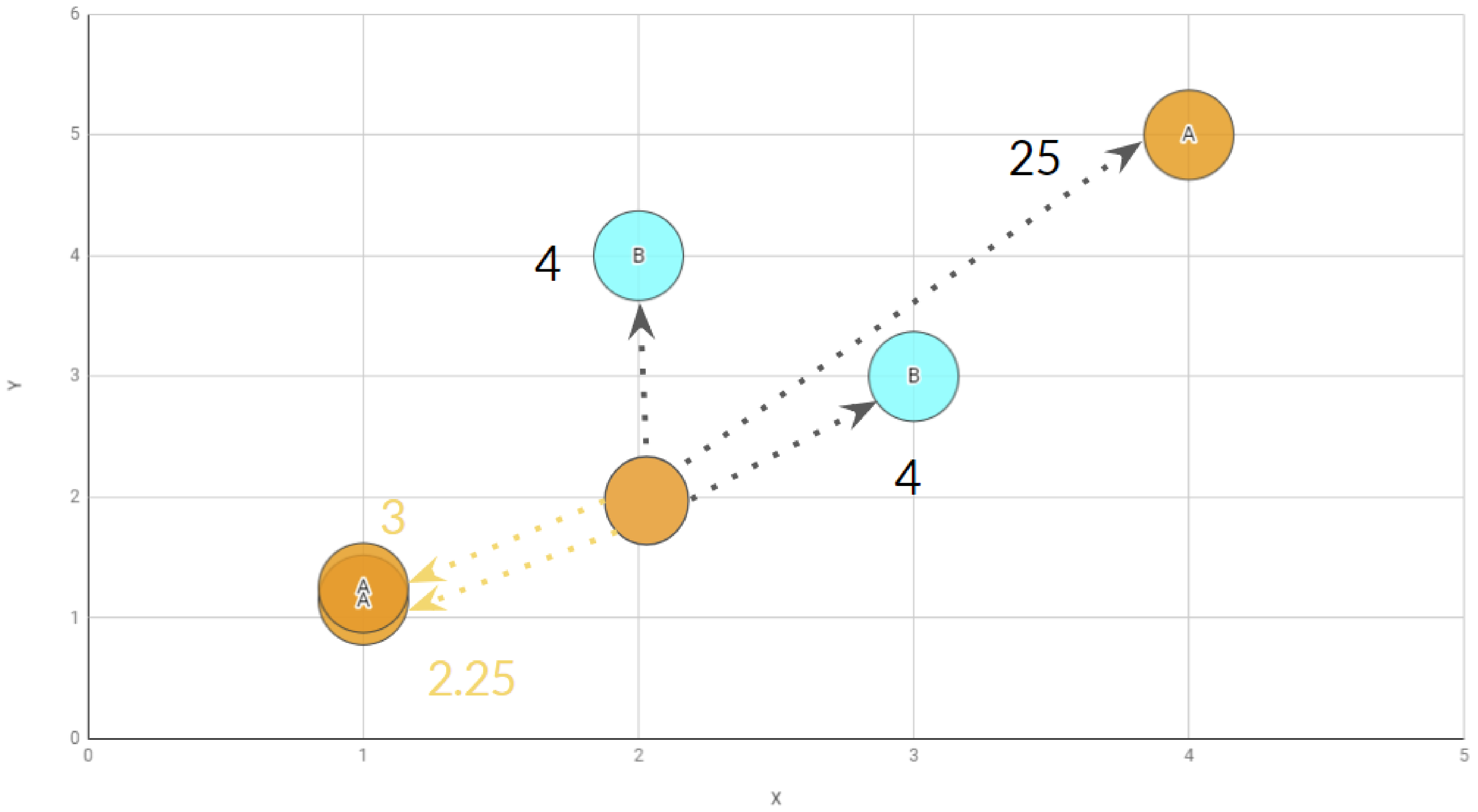
k-Nearest Neighbors: measure



k-Nearest Neighbors: 2-NN for majority vote



k-Nearest Neighbors: label point



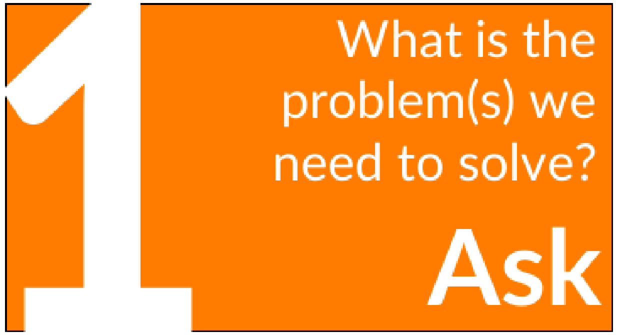
Knowledge check 1

Exercise 1

Module completion checklist

Objective	Complete
Summarize the steps & application of kNN	✔
Clean and transform the data to run kNN	
Define cross validation and how and when it is used	
Implement the kNN algorithm on the training data without cross-validation	
Identify performance metrics for classification algorithms	
Evaluate the optimal number of nearest neighbors to use using cross-validation	
Evaluate performance of optimized kNN model	
Apply the knn and evalute its performance on the CMP dataset	

kNN : example one - business case



A new medicine is out and is being questioned regarding the effect it has on heart rate; the **medication will increase heart rate** and the researchers are now sure it does not vary by gender specifically, but want to **classify gender** for new patients using temperature and heart rate.

We have been given a sample of participants who took the medication for a study. We have:

- Gender,
- Heart rate, and
- Temperature of each participant

We want to use kNN to predict the **classification** of each data point to either Male or Female.

Directory Settings

- In order to maximize the efficiency of your workflow, you may want to encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `hhs-r-2020` folder

```
# Set `main_dir` to the location of your `hhs-r-2020` folder (for Mac/Linux).
main_dir = "~/Desktop/hhs-r-2020"
# Set `main_dir` to the location of your `hhs-r-2020` folder (for Windows).
main_dir = "C:/Users/[username]/Desktop/hhs-r-2020"

# Make `data_dir` from the `main_dir` and remainder of the path to data directory.
data_dir = paste0(main_dir, "/data")
# Make `plots_dir` from the `main_dir` and remainder of the path to plots directory.
plot_dir = paste0(main_dir, "/plots")

# Set directory to data_dir.
setwd(data_dir)
```

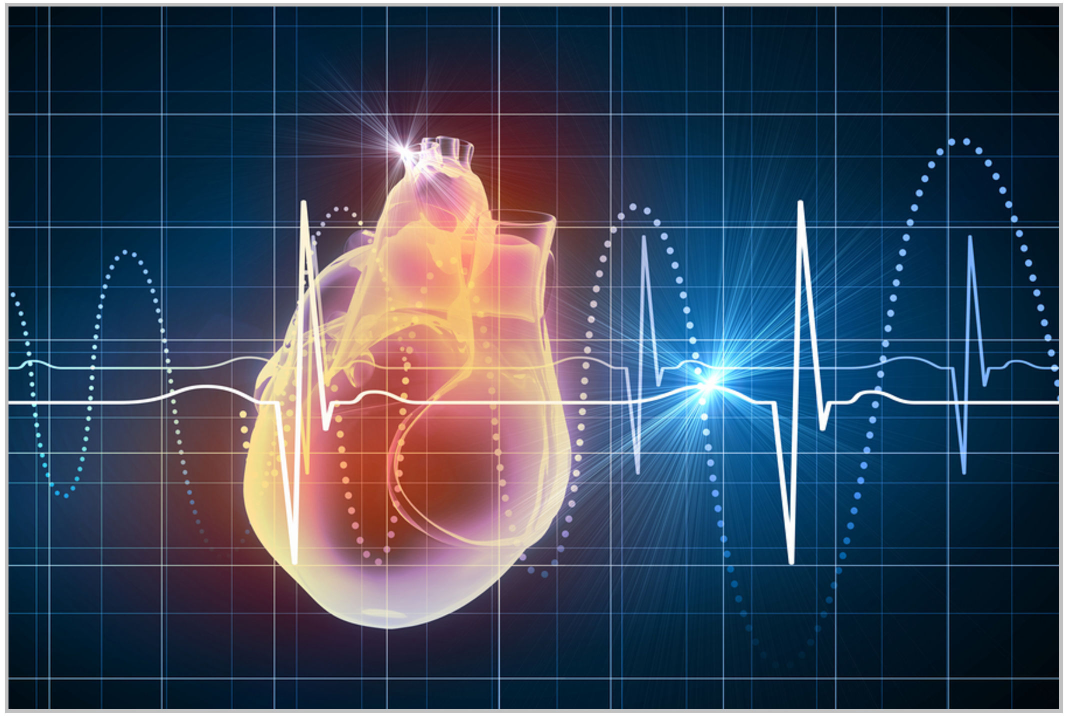
Load the dataset

We will now load the dataset that consists of three variables:

- Gender
- Temperature
- Heart Rate

```
setwd(data_dir)
temp_heart = read.csv("temp_heart_rate.csv",
TRUE)
head(temp_heart)
```

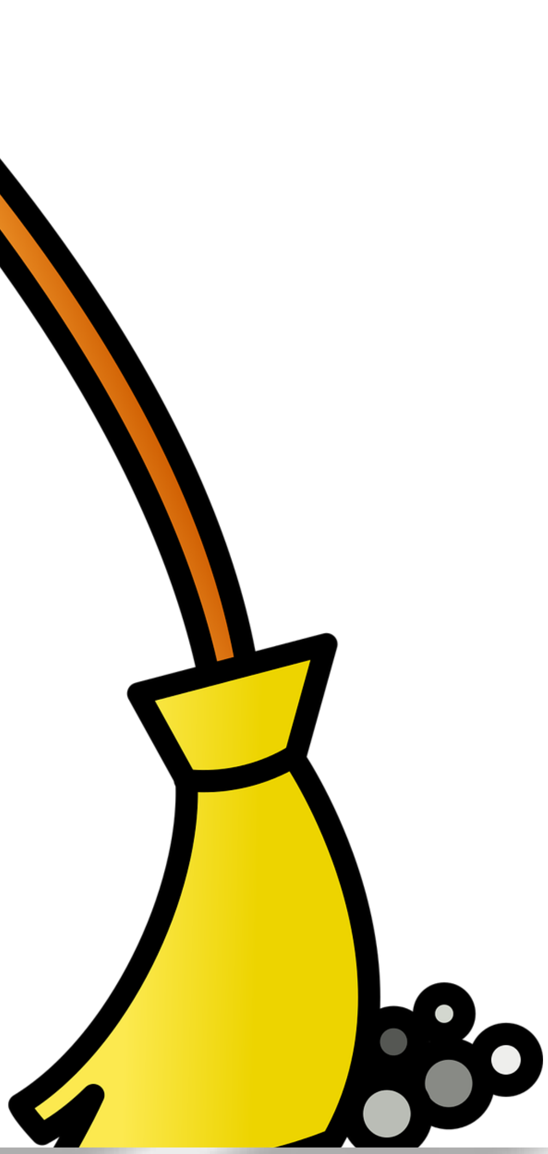
```
  Gender Body.Temp Heart.Rate
1   Male      96.3         70
2   Male      96.7         71
3   Male      96.9         74
4   Male      97.0         80
5   Male      97.1         73
6   Male      97.1         75
```



Data cleaning steps for kNN

Like the other algorithms we have covered, there are a few steps to take before jumping into splitting the data and training the model:

1. Check for NAs
2. Scale the predictors
3. Make sure the target is labeled



The data at first glance

```
# Look at the structure of the data
str(temp_heart)
```

```
'data.frame':	130 obs. of  3 variables:
 $ Gender    : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 2 2 2 2 ...
 $ Body.Temp : num  96.3 96.7 96.9 97 97.1 97.1 97.1 97.2 97.3 97.4 ...
 $ Heart.Rate: int  70 71 74 80 73 75 82 64 69 70 ...
```

```
# Examine the distribution of gender from the sample population

# prop.table creates a table of proportions
prop.table(
  table(                # <- creates a table of counts related to each variable
    temp_heart[,1]))    # <- denotes the variable that is being evaluated
```

```

Female   Male 
   0.5    0.5 
```

Check for NAs

Check for NAs

1. What is the percentage of NAs in our dataset?
2. We need to divide the `sum` of all entries that *are* `NA` by the `sum` of those that *aren't*.

```
# Let's find all NAs in the dataset.
is_NA = is.na(temp_heart)

# Are there NAs in the dataset?
sum(is_NA) / sum(!is_NA)
```

```
[1] 0
```

- Doesn't look like we have to deal with NAs in this dataset

Scaling the predictors

Scale the predictors

1. Let's create a new object, `temp_heart_scaled`, where we will scale the predictors only
2. We then add the target variable `Gender` to the new scaled dataset

```
# Scale the predictors, transform the data to a data frame so
# that we can add gender back in
temp_heart_scaled = as.data.frame(sapply(temp_heart[,2:3],scale))

# Make sure to add gender back to the scaled data set
gender = temp_heart[,1]
temp_heart_scaled$Gender = gender

# Inspect scaled values
head(temp_heart_scaled)
```

```
  Body.Temp  Heart.Rate Gender
1 -2.658586 -0.53263914   Male
2 -2.113020 -0.39103773   Male
3 -1.840237  0.03376649   Male
4 -1.703845  0.88337493   Male
5 -1.567454 -0.10783492   Male
6 -1.567454  0.17536790   Male
```

Defining a target variable

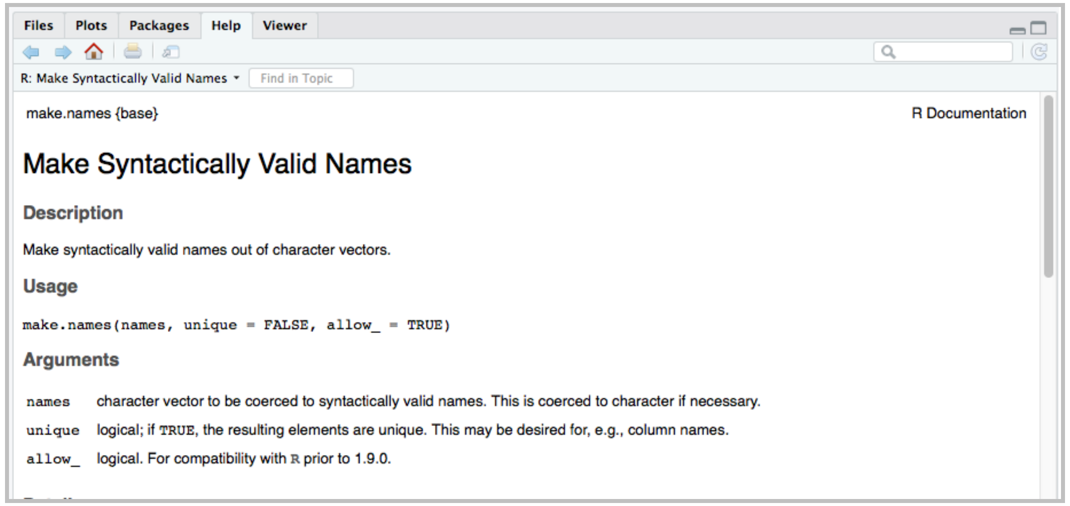
- kNN will have either a **binary** target or a **multi-class** target
- In the case of the heart rate dataset we are dealing with a **binary** target
- Our target variable here is `Gender` which has two levels `Male` and `Female`

- If we have a *continuous* variable as a target, we manipulate it to become **binary** / **multi-class**
- We will address this later today when we introduce kNN on the `CMP` dataset

Prepare for prediction: labeling the target

- In preparing for prediction, the levels of the target variable will be used as variable names for prediction
- To do this, we use the function `make.names`

```
?make.names

make.names(names) #<- character vector to be
coerced syntactically to valid names
```


Files Plots Packages Help Viewer

R: Make Syntactically Valid Names Find in Topic

make.names {base} R Documentation

Make Syntactically Valid Names

Description

Make syntactically valid names out of character vectors.

Usage

```
make.names(names, unique = FALSE, allow_ = TRUE)
```

Arguments

`names`	character vector to be coerced to syntactically valid names. This is coerced to character if necessary.
`unique`	logical; if `TRUE`, the resulting elements are unique. This may be desired for, e.g., column names.
`allow_`	logical. For compatibility with R prior to 1.9.0.

Prepare for prediction: labeling the target

- We need to make sure the levels are valid variable names using the function `make.names`

```
# Setting levels for both training and test data
levels(temp_heart_scaled$Gender) =
  make.names(levels(factor(temp_heart_scaled$Gender)))

levels(temp_heart_scaled$Gender) =
  make.names(levels(factor(temp_heart_scaled$Gender)))
```

Cross-validation: Remember this?

Train

- This is the data that you **train your model on**
- Usually about **70% of your dataset**
- Use a larger portion of the data to train so that the model gets a **large enough sample of the population**
- If there is not a large population on the whole, **cross-validation techniques can be implemented**

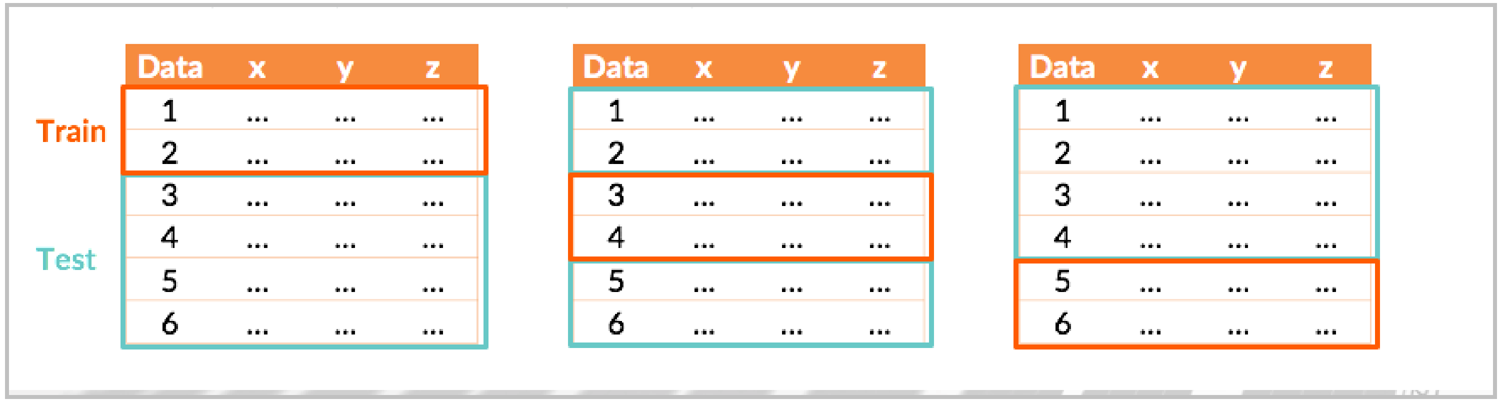
Test

- This is the data that you **test your model on**
- Usually about **30% of your dataset**
- Use a smaller portion to test your trained model on
- If cross-validation is implemented, **small test sets will be held out multiple times**

Cross-validation: n-fold

Earlier, we split into test and train once. Here is the idea now:

1. Split the dataset into several subsets (“n” number of subsets) of equal size
2. **Use each subset as the test data set** and **use the rest of the data as the training dataset**
3. Repeat the process for every subset you create



Train & test: small scale before n-fold

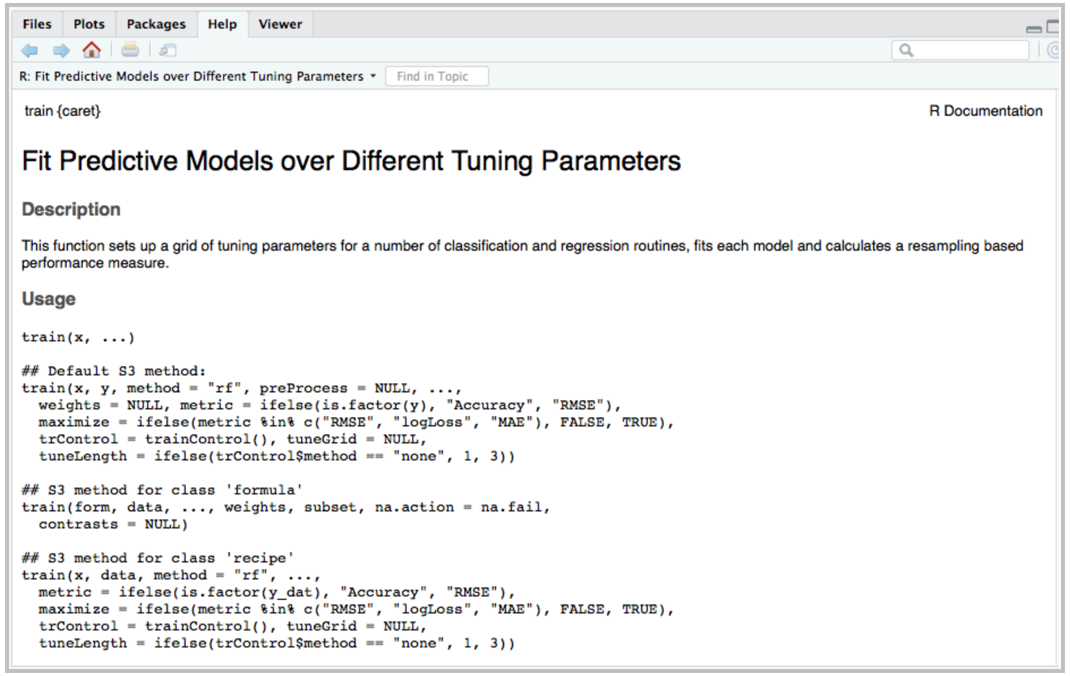
- Before we actually use n-fold cross-validation:
  - We split our data into a train and test set
  - We run kNN initially on the training data

```
# Set the seed.
set.seed(1)

# We will use this to split the temp_heard dataset.
train_index = createDataPartition(temp_heart_scaled$Gender,  #<- outcome variable
                                  list = FALSE,              #<- avoid returning the data as a list
                                  times = 1,                 #<- split 1 time
                                  p = 0.7)                   #<- 70% training, 30% test

# Subset the scaled and cleaned data to include only train set observations.
temp_heart_train = temp_heart_scaled[train_index, ]
# Subset the scaled and cleaned data to include only test set observations.
temp_heart_test = temp_heart_scaled[-train_index, ]
```

kNN: modeling with caret & train

- We will use the kNN function implemented through the `caret` package, using the `train` function

```
?train

train(x,
      y,
      method,
      trControl,
      metric)
```


Files Plots Packages Help Viewer

R: Fit Predictive Models over Different Tuning Parameters Find in Topic

train {caret} R Documentation

Fit Predictive Models over Different Tuning Parameters

Description

This function sets up a grid of tuning parameters for a number of classification and regression routines, fits each model and calculates a resampling based performance measure.

Usage

```
train(x, ...)

## Default S3 method:
train(x, y, method = "rf", preProcess = NULL, ...,
  weights = NULL, metric = ifelse(is.factor(y), "Accuracy", "RMSE"),
  maximize = ifelse(metric %in% c("RMSE", "logLoss", "MAE"), FALSE, TRUE),
  trControl = trainControl(), tuneGrid = NULL,
  tuneLength = ifelse(trControl$method == "none", 1, 3))

## S3 method for class 'formula'
train(form, data, ..., weights, subset, na.action = na.fail,
  contrasts = NULL)

## S3 method for class 'recipe'
train(x, data, method = "rf", ...,
  metric = ifelse(is.factor(y_dat), "Accuracy", "RMSE"),
  maximize = ifelse(metric %in% c("RMSE", "logLoss", "MAE"), FALSE, TRUE),
  trControl = trainControl(), tuneGrid = NULL,
  tuneLength = ifelse(trControl$method == "none", 1, 3))
```

- `x`: The predictors from the training set
- `y`: The target from the training set
- `method`: This will be `knn` here
- `trControl`: We will use this to define cv parameters
- `metric`: The parameter we will use to select the optimal model

kNN: train the model

- Let us run kNN on the training data, using `train`
- Notice that we do not include the `trControl` parameter

```
# We first train the model on just the training
data, without using cross validation
knn_fit = train(Gender ~.,
                 data = temp_heart_train,
                 method = "knn")
```

```
k-Nearest Neighbors

92 samples
 2 predictor
 2 classes: 'Female', 'Male'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 92, 92, 92, 92, 92, 92,
...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.5800895  0.1761705
  7  0.5901166  0.1985280
  9  0.5865995  0.1907089

Accuracy was used to select the optimal model
using the largest value.
The final value used for the model was k = 7.
```

kNN: predict on test

- Now we will take our trained model and predict on the test set

```
test_pred = predict(knn_fit,
                    newdata = temp_heart_test)
```

- What we get is a vector of predicted values
- This is helpful because we have the actual values for this sample
- We can calculate the accuracy of our model using the actual values compared to the predicted values

```
 [1] Male   Male   Male   Male   Male   Male  
Female Female Male   Female
[11] Male   Male   Female Female Male   Female
Female Female Female Male
[21] Male   Male   Male   Male   Male   Male  
Female Male   Male   Female
[31] Female Male   Female Female Female Female
Female Female
Levels: Female Male
```

Knowledge check 2

Exercise 2

Module completion checklist

Objective	Complete
Summarize the steps & application of kNN	✔
Clean and transform the data to run kNN	✔
Define cross validation and how and when it is used	✔
Implement the kNN algorithm on the training data without cross-validation	✔
Identify performance metrics for classification algorithms	
Evaluate the optimal number of nearest neighbors to use using cross-validation	
Evaluate performance of optimized kNN model	
Apply the knn and evalute its performance on the CMP dataset	

Classification: assessing performance

- We have reviewed how to measure *error* when using **regression**
- Because our outcome variable is **binary** we have a different way of measuring error than when the outcome variable was continous
- The following terms are very important to measure performance of a classification algorithm
  - Confusion matrix

Confusion matrix: What is it?

Confusion matrix is what we use to assist us in measuring error

We will use it to calculate Accuracy, Misclassification rate, True positive rate, False positive rate and Specificity

	Female	**Male**	**Predicted Totals:**
Predicted Female	True Positive (TP)	False Positive (FP)	Total Predicted Positive
Predicted Male	False Negative (FN)	True Negative (TN)	Total Predicted Negative
Actual Totals:	Total Positives	Total Negatives	Total

Confusion matrix: for temp heart data

Let's look at the confusion matrix for the `temp_heart_test` predictions - using **7 NN**

	Female	**Male**	**Predicted Totals:**
Predicted Female	13 True Positive (TP)	7 False Positive (FP)	20
Predicted Male	6 False Negative (FN)	12 True Negative (TN)	18
Actual Totals:	19	19	38

Confusion matrix: accuracy

Accuracy: overall, how often is the classifier correct?

TP + TN / **total** = **(13 + 12)** / **38**

	Female	Male	Predicted Totals:
Predicted Female	13 True Positive (TP)	7 False Positive (FP)	20
Predicted Male	6 False Negative (FN)	12 True Negative (TN)	18
Actual Totals:	19	19	38

Confusion matrix: misclassification rate

Misclassification rate (error rate) : overall, how often is the classifier wrong?

FP + FN / **total** = **(7 + 6)** / **38**

	Female	Male	Predicted Totals:
Predicted Female	13 True Positive (TP)	7 False Positive (FP)	20
Predicted Male	6 False Negative (FN)	12 True Negative (TN)	18
Actual Totals:	19	19	38

Confusion matrix: true positive rate

True positive rate (Sensitivity): how often does it predict yes?

TP / **actual yes** = 13 / **20**

	Female	Male	Predicted Totals:
Predicted Female	13 True Positive (TP)	7 False Positive (FP)	20
Predicted Male	6 False Negative (FN)	12 True Negative (TN)	18
Actual Totals:	19	19	38

Confusion matrix: false positive rate

False positive rate: when it's actually no, how often does it predict yes?

FP / **actual no** = **7** / **19**

	Female	Male	Predicted Totals:
Predicted Female	13 True Positive (TP)	7 False Positive (FP)	20
Predicted Male	6 False Negative (FN)	12 True Negative (TN)	18
Actual Totals:	19	19	38

Confusion matrix: specificity

True Negative Rate (Specificity): when it's actually no, how often does it predict no?

TN / **actual no** = **12** / **19**

	Female	Male	Predicted Totals:
Predicted Female	13 True Positive (TP)	7 False Positive (FP)	20
Predicted Male	6 False Negative (FN)	12 True Negative (TN)	18
Actual Totals:	19	19	38

Confusion matrix: model fit for knn

```
k-Nearest Neighbors

92 samples
 2 predictor
 2 classes: 'Female', 'Male'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 92, 92, 92, 92, 92, 92,
...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.5800895  0.1761705
  7  0.5901166  0.1985280
  9  0.5865995  0.1907089

Accuracy was used to select the optimal model
using the largest value.
The final value used for the model was k = 7.
```

The output of the model says that the number of **k=7** was determined by optimizing for **accuracy** within the **training data set**

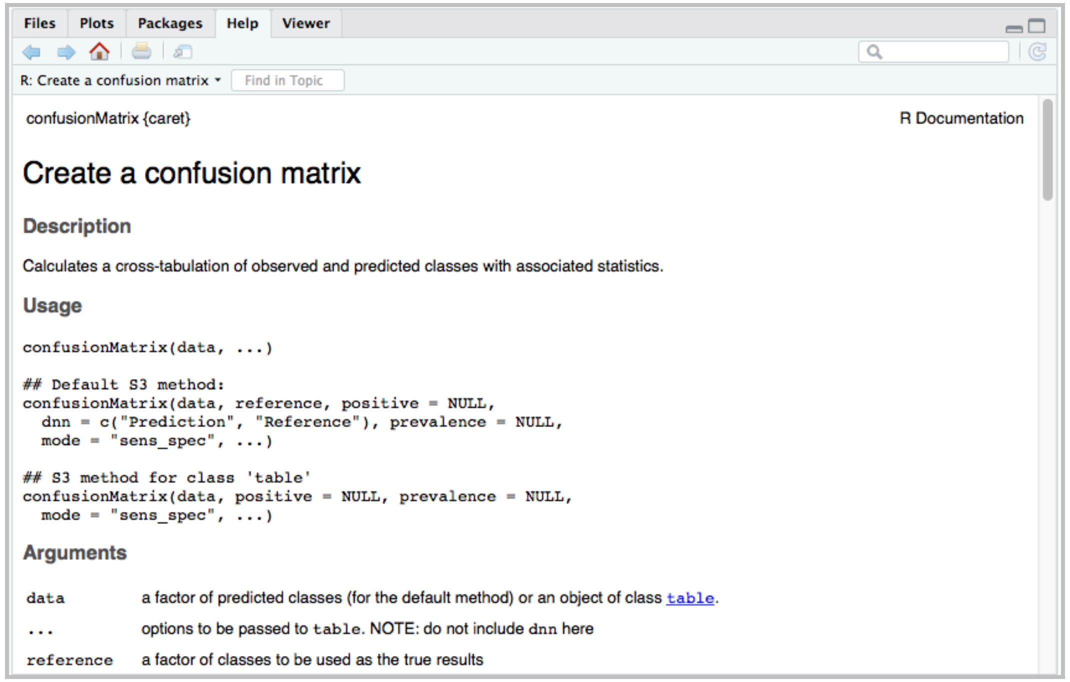
i.e. the algorithm tried different values of **k** to see how k impacted **accuracy** and chose the k with the **highest** accuracy.

The algorithm is using an internal confusion matrix.

Confusion matrix: in R

- Now that we know the metrics behind the madness, let's execute the code to build a confusion matrix in R that **we** can see.
- We use a function `confusionMatrix` from the `caret` package

```
?caret::confusionMatrix

confusionMatrix(data,        #<- the predicted classes
                reference)   #<- the true classes
```



Confusion matrix: calculate in R

- Now we calculate the confusion matrix in R

```
confusionMatrix(test_pred,
                temp_heart_test$Gender)
```

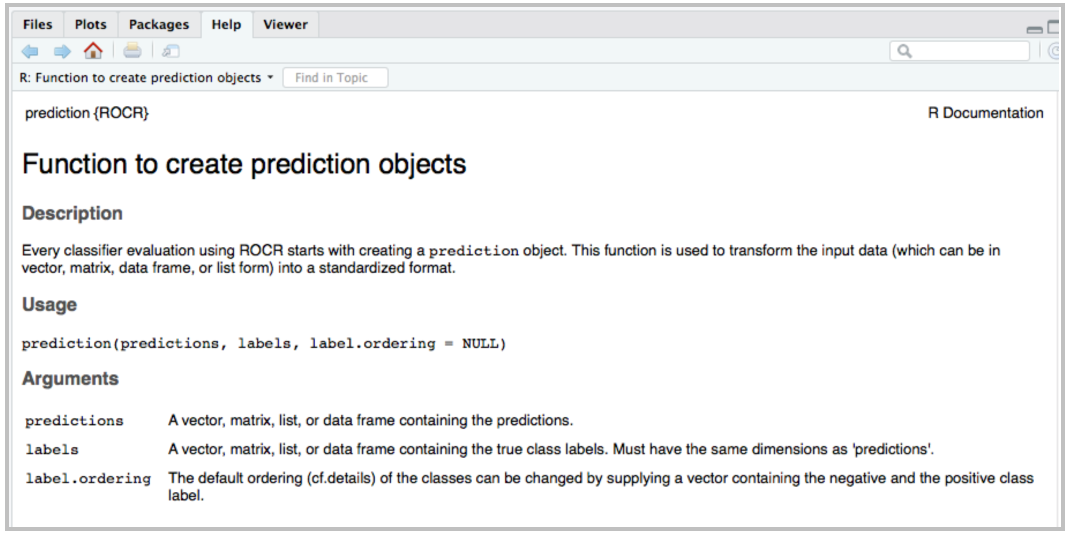
```
Confusion Matrix and Statistics

          Reference
Prediction Female Male
    Female      9    9
    Male       10   10

               Accuracy : 0.5
                 95% CI : (0.3338, 0.6662)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 0.5643

                  Kappa : 0

 Mcnemar's Test P-Value : 1.0000

            Sensitivity : 0.4737
            Specificity : 0.5263
         Pos Pred Value : 0.5000
         Neg Pred Value : 0.5000
             Prevalence : 0.5000
         Detection Rate : 0.2368
   Detection Prevalence : 0.4737
      Balanced Accuracy : 0.5000

       'Positive' Class : Female
```

Performance of our kNN model

Now, let's evaluate our model using the ROC curve:

- We use the function `prediction` from the `ROCR` package to create a prediction object

```
?prediction

prediction(predictions,
           labels)
```


Files Plots Packages Help Viewer

R: Function to create prediction objects Find in Topic

prediction {ROCR} R Documentation

Function to create prediction objects

Description

Every classifier evaluation using ROCR starts with creating a `prediction` object. This function is used to transform the input data (which can be in vector, matrix, data frame, or list form) into a standardized format.

Usage

```
prediction(predictions, labels, label.ordering = NULL)
```

Arguments

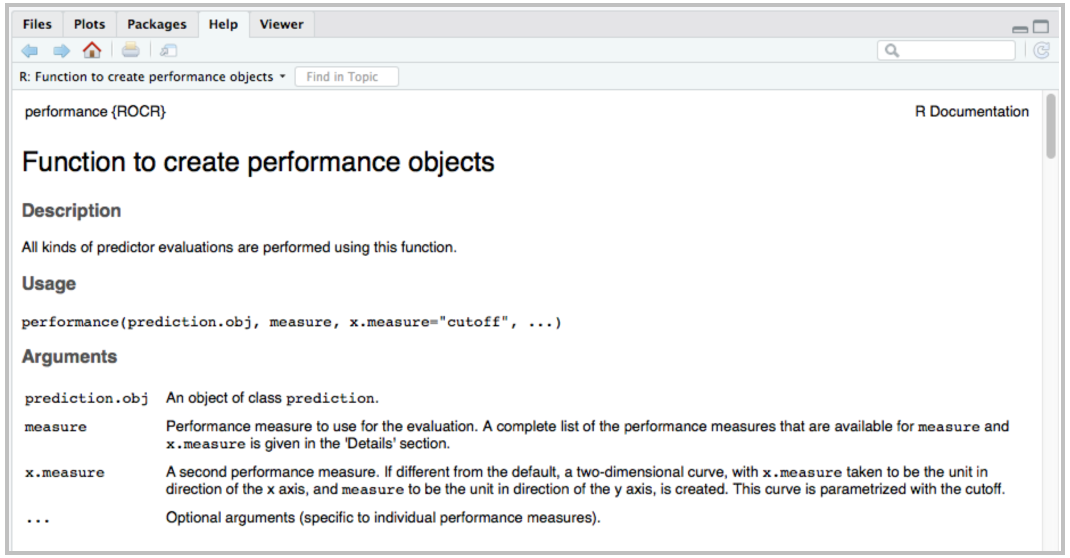
`predictions`	A vector, matrix, list, or data frame containing the predictions.
`labels`	A vector, matrix, list, or data frame containing the true class labels. Must have the same dimensions as 'predictions'.
`label.ordering`	The default ordering (cf.details) of the classes can be changed by supplying a vector containing the negative and the positive class label.

- `predictions`: the vector of class labels predicted by the kNN model
- `labels`: the vector of true class labels

Performance of our kNN model

We can now use the function `performance` from the `ROCR` package for model evaluation

```
?performance

perf = performance(prediction.obj, #<- the
object we just created using `prediction`
            measure,        #<- a performance
measure, in this case, either "tpr"" or "auc"
            x.measure)      #<- a second
performance measure; for ROC curve, use "fpr"
plot(perf)                  #<- plots the ROC
curve
```


Files Plots Packages Help Viewer

R: Function to create performance objects Find in Topic

performance {ROCR} R Documentation

Function to create performance objects

Description

All kinds of predictor evaluations are performed using this function.

Usage

`performance(prediction.obj, measure, x.measure="cutoff", ...)`

Arguments

`prediction.obj`	An object of class `prediction`.
`measure`	Performance measure to use for the evaluation. A complete list of the performance measures that are available for `measure` and `x.measure` is given in the 'Details' section.
`x.measure`	A second performance measure. If different from the default, a two-dimensional curve, with `x.measure` taken to be the unit in direction of the x axis, and `measure` to be the unit in direction of the y axis, is created. This curve is parametrized with the cutoff.
`...`	Optional arguments (specific to individual performance measures).

- `prediction.obj`: the object we just created using `prediction`
- `measure`: a performance measure, in this case, either “tpr” or “auc”
- `x.measure`: a second performance measure; for ROC curve, use “fpr”

Knowledge check 3

Exercise 3

Module completion checklist

Objective	Complete
Summarize the steps & application of kNN	✔
Clean and transform the data to run kNN	✔
Define cross validation and how and when it is used	✔
Implement the kNN algorithm on the training data without cross-validation	✔
Identify performance metrics for classification algorithms	✔
Evaluate the optimal number of nearest neighbors to use using cross-validation	
Evaluate performance of optimized kNN model	
Apply the knn and evalute its performance on the CMP dataset	

Finding optimal k

1. We have now:
   i. Run kNN on our training data
   ii. Reviewed performance metrics for classification algorithms
   iii. Built a confusion matrix for the predicted model
2. We will now:
   i. Use cross-validation and re-run the model on the training data
   ii. Find what our optimal k value is
   iii. Evaluate the new predictions

Finding optimal k

- We will use the `train` function again, but this time we will bring in the `trainControl` parameter
- This allows us to choose `repeatedcv` as the `method` parameter

```
ctrl = trainControl(method="repeatedcv",
                    repeats = 3,
                    classProbs=TRUE)
```

- `method`: use cross-validation
- `repeats`: repeat cross-validation 3 times
- `classProbs`: give us the probabilities for each class

- Run the model on the training data

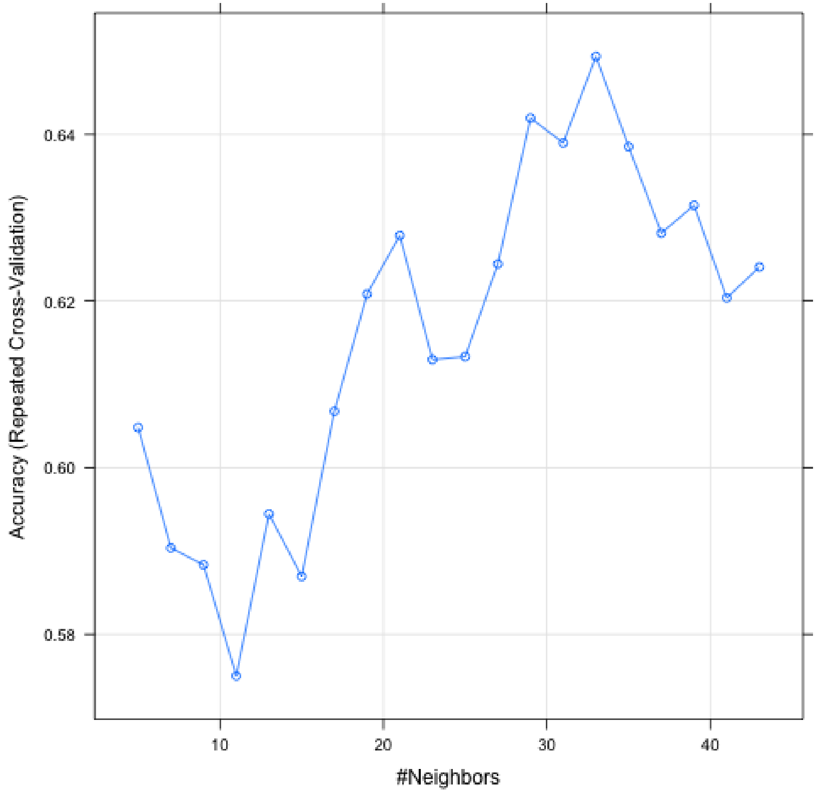
```
knn_fit_cv = train(Gender ~ .,
                   data = temp_heart_train,
                   method = "knn",
                   trControl = ctrl,
                   tuneLength = 20)
```

Finding optimal k

- Look at the results

```
knn_fit_cv

k-Nearest Neighbors

92 samples
 2 predictor
 2 classes: 'Female', 'Male'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 84, 83, 83, 83, 82, 83, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   5  0.5333333  0.06538226
   7  0.5412963  0.07970085
   ...
  39  0.6233333  0.24725393
  41  0.6165741  0.23475116
  43  0.6256481  0.25133563

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was k = 43.
```

Finding optimal k



Using the optimized model to predict

- We can use the model we just built to predict on the test set
- It is optimized because of the cross-validation and will use the optimized k which is `k=43`
- Although it is an optimized model, it does not mean it will perform better than our previous model, it is just more accurate

```
test_pred_cv = predict(knn_fit_cv,newdata =
temp_heart_test )
```

```
#Get the confusion matrix to see accuracy value
and other parameter values
confusionMatrix(test_pred_cv,
                temp_heart_test$Gender )

Confusion Matrix and Statistics

          Reference
Prediction Female Male
    Female      8    6
    Male       11   13

               Accuracy : 0.5526
                 95% CI : (0.383, 0.7138)
                 ...
       'Positive' Class : Female
```

KNN pros and cons

PROs

- easy to use
- can easily handle multiple categories
- there are many options to adjust (which features to use, measurement metric, etc)

CONs

- **There are many options to adjust**
- The correct distance metric is important
- Can be slow with large amounts of data
- Because it is **LAZY**, memorizes and uses every data point when calculating kNN

Module completion checklist

Objective	Complete
Summarize the steps & application of kNN	✔
Clean and transform the data to run kNN	✔
Define cross validation and how and when it is used	✔
Implement the kNN algorithm on the training data without cross-validation	✔
Identify performance metrics for classification algorithms	✔
Evaluate the optimal number of nearest neighbors to use using cross-validation	✔
Evaluate performance of optimized kNN model	✔
Apply the knn and evalute its performance on the CMP dataset	

kNN on CMP

- We are all familiar with the `CMP` dataset by now
- Let's start with data pre-processing
  - scale the data
  - impute NAs using the function we built earlier
  - return a cleaned dataset
- We will then have to convert our target variable `Yield` from **continuous** to **categorical** so that we can use a **classification** algorithm, **kNN**

kNN on CMP: data pre-processing

```
setwd(data_dir)
CMP =
  read.csv("ChemicalManufacturingProcess.csv",
           header = TRUE,
           stringsAsFactors = FALSE)

# Scale `CMP` dataset
CMP_scale = scale(CMP)

# Save it as a data frame
CMP_scale = as.data.frame(CMP_scale)
```

```
# Impute NAs in CMP dataset
CMP_NA_impute = ImputeNAsWithMean(CMP_scale)

# rename it as CMP_cleaned
CMP_cleaned = CMP_NA_impute
```

kNN on CMP: converting Yield

- We covered this early on, when we learned `ifelse`
- Now we see why we would need to convert a variable from continous to categorical
- This is useful, as you now can see how you could use a classification algorithm with a continuous predictor

```
# What are the summary stats of `Yield`?
summary(CMP_cleaned$Yield)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.6692 -0.7716 -0.1119  0.0000  0.7035  3.3394
```

```
# Median
summary(CMP_cleaned$Yield)[3]
```

```
   Median
-0.1119022
```

- Now, we use the median to separate `Yield` into
  - `High yield`
  - `Low yield`

```
# To convert Yield to binary, lets make
everything greater than or equal
# to the median using an ifelse statement
CMP_cleaned$Yield = ifelse(CMP_cleaned$Yield >=
summary(CMP_cleaned$Yield)[3],
                           "High yield",
                           "Low yield" )
# Convert Yield to a factor
CMP_cleaned$Yield = as.factor(CMP_cleaned$Yield)

head(CMP_cleaned$Yield)
```

```
[1] Low yield  High yield High yield High yield
High yield High yield
Levels: High yield Low yield
```

kNN: train model using cv

- Split the data into train and test

```
# Let's split the cleaned dataset into test and train:
# Set the seed.
set.seed(1)

# We will use this to split the actual CMP dataset.
train_index = createDataPartition(CMP_cleaned$Yield,  #<- outcome variable
                                  list = FALSE,       #<- avoid returning the data as a list
                                  times = 1,          #<- split 1 time
                                  p = 0.7)

# Subset the data to include only train set observations.
CMP_train = CMP_cleaned[train_index, ]

# Subset the data to include only test set observations.
CMP_test = CMP_cleaned[-train_index, ]
```

- Name the levels of the target variable `Yield`

```
# Make sure to name the levels
levels(CMP_train$Yield) =
  make.names(levels(factor(CMP_train$Yield)))

levels(CMP_test$Yield) =
  make.names(levels(factor(CMP_test$Yield)))
```

kNN: train model using cv

- Set the parameters for `trainControl`
- Train the model
- Name the model
  `CMP_[algorithm_name]`
- We will be comparing models at the end of the 8 weeks

```
# Run kNN, set cv as train control first,
include two class summary:
CMP_ctrl = trainControl(method="repeatedcv",
                     repeats = 3,
                     classProbs=TRUE,
                     summaryFunction =
twoClassSummary)

CMP_knn = train(Yield ~ ., data = CMP_train,
                method = "knn",
                trControl = CMP_ctrl,
                tuneLength = 20)
```

```
CMP_knn

k-Nearest Neighbors

124 samples
 57 predictors
  2 classes: 'High.yield', 'Low.yield'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3
times)
Summary of sample sizes: 111, 112, 111, 111,
112, 112, ...
Resampling results across tuning parameters:

  k   ROC        Sens       Spec
   5  0.8977986  0.7904762  0.8142857
   7  0.8969482  0.7984127  0.7880952
   ...
   15  0.8984505  0.7880952  0.7730159
   ...
  43  0.8380197  0.6325397  0.7769841

ROC was used to select the optimal model using
the largest value.
The final value used for the model was k = 15.
```

kNN: predict

- Now let's predict on the test set

```
knnPredict = predict(CMP_knn,newdata = CMP_test
)
```

```
#Get the confusion matrix to see accuracy value
and other parameter values
confusionMatrix(knnPredict, CMP_test$Yield )

Confusion Matrix and Statistics

            Reference
Prediction   High.yield Low.yield
  High.yield         16        12
  Low.yield          10        14

               Accuracy : 0.5769
               'Positive' Class : High.yield
```

Knowledge check 4

Exercise 4

Module completion checklist

Objective	Complete
Summarize the steps & application of kNN	✔
Clean and transform the data to run kNN	✔
Define cross validation and how and when it is used	✔
Implement the kNN algorithm on the training data without cross-validation	✔
Identify performance metrics for classification algorithms	✔
Evaluate the optimal number of nearest neighbors to use using cross-validation	✔
Evaluate performance of optimized kNN model	✔
Apply the knn and evalute its performance on the CMP dataset	✔

This completes our module

Congratulations!