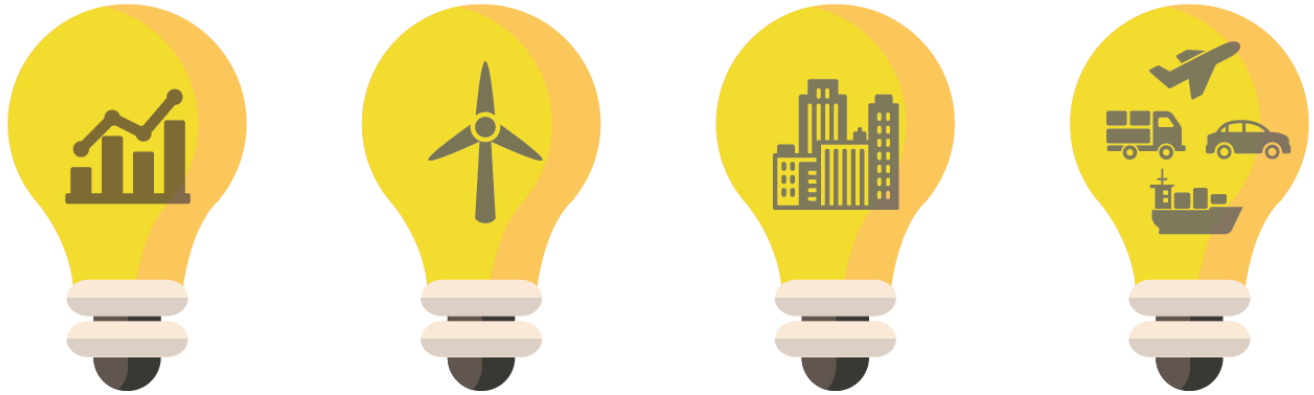
BENV0091 Lecture 6: Unsupervised Learning

Patrick de Mars



Lecture Overview

- Principal components analysis and clustering smart meter data
- Model selection techniques
- Kaggle winners!

Unsupervised Learning

Unsupervised Learning

- We have covered supervised learning, which deals with finding a mapping between a set of features $X_1, X_2, \ldots, X_N$ and a response variable Y
- Unsupervised learning is concerned with problems where we only have the features $X_1, X_2, \ldots, X_N$
- It is often used as a part of exploratory data analysis
- Examples include:
  - Clustering
  - Principal components analysis

Principal Components Analysis (PCA)

- PCA can be used to find a low-dimensional representation of higher-dimensional data
- It is useful for:
  - Clustering (which can be challenging/computationally expensive in high dimensional space)
  - Feature engineering
  - Visualising high-dimensional data

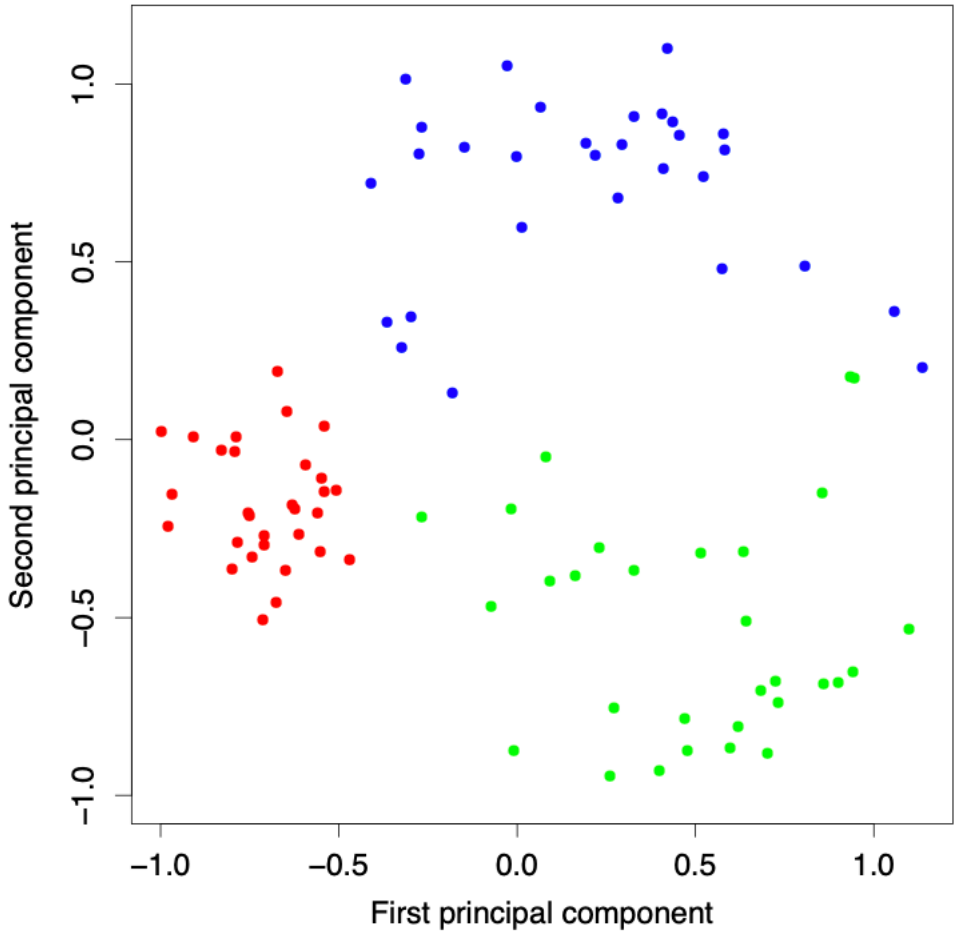


Figure: The Elements of Statistical Learning, Hastie et al.

Clustering Smart Meter Data

- We will use K-means clustering to cluster smart meter profiles
- We will look at Winter only (Q3 and Q4), and look to create archetypal weekly customer profiles (at hourly resolution)
- Each observation will have 7*24 = 168 features: a lot of dimensions!
- Hence, we'll use **PCA** to reduce the number of dimensions and cluster the data in lower-dimensional space

Average weekly profiles

Customer ID	Monday 00:00	Monday 01:00	…	Sunday 23:00
0001	0.10 kWh	0.05 kWh		0.15 kWh
0002	0.12 kWh	0.07 kWh		0.11 kWh

Initial Data Preparation

- First, some initial preparation:
  - Read the data with `read_csv2()`
  - Name the first column `timestamp`
  - Ensure the timestamp column is a datetime (lubridate!)
  - Create `wday` and `hour` columns
  - Convert all character columns to numeric
  - Filter the data to just Q3 and Q4
- Task: confirm that there are no missing values
- Task: confirm that no columns have 0 variance (i.e. no activity)

`read_csv2()` reads CSV files where columns are separated with semi-colons and commas are used for decimal points

`mutate_if(condition, function)` can be used to applied a function to columns meeting a condition (such as `is.character`)

Transforming the Data

- In order to cluster the data, we must have the same dimensionality for all customers
- There are several options for how to cluster the data:
  - Raw data (c. 17,000 features)
  - Average day (24 features)
  - Average week (7*24 features)
  - Average weekday, average weekend day (2*24 features)
- We want to transform the data such that:
  - Each row is a customer
  - Each column is a 1 hour period (weekday/hour)
  - Each entry is mean power (kW)
- Task: transform the data:
  - Pivot the data longer, excluding timestamp, wday and hour
  - Group by customer, wday and hour and calculate the mean power (kW)
  - Pivot the data wider by wday and hour (see output right)

Desired output

	customer	`1_0`	`1_1`	`1_2`	`1_3`	`1_4`	`1_5`
	<chr>	*<dbl>*	*<dbl>*	*<dbl>*	*<dbl>*	*<dbl>*	*<dbl>*
1	MT_001	4.26	4.32	4.29	4.29	4.32	4.27
2	MT_002	20.6	18.7	16.9	16.7	16.7	17.4
3	MT_003	1.44	1.17	1.00	1.36	1.36	1.43
4	MT_004	93.1	82.9	70.6	66.6	62.3	59.0
5	MT_005	43.2	39.3	34.4	31.9	31.1	30.6
6	MT_006	133.	119.	103.	96.6	91.3	88.2
7	MT_007	6.51	6.08	5.74	5.64	5.62	5.55

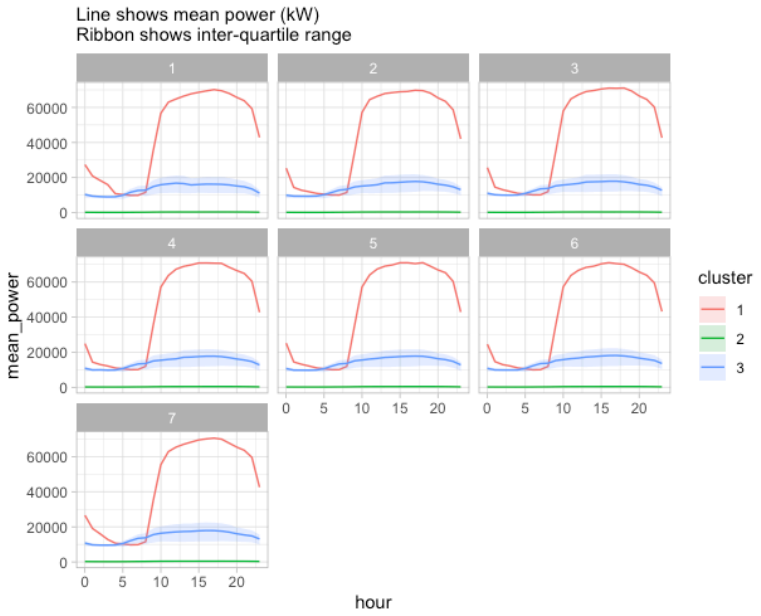
Normalising the Data

- For smart meter data, **normalising** each observation can be useful if we are predominantly interested in **patterns**, rather than **absolute** consumption
- Task: normalise each row of your data

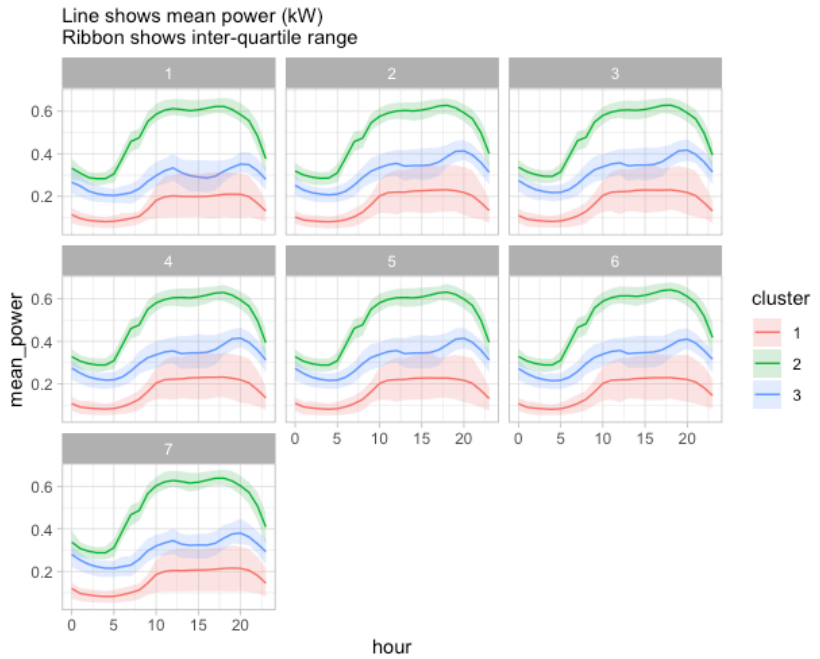
Normalising between 0 and 1

$$\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

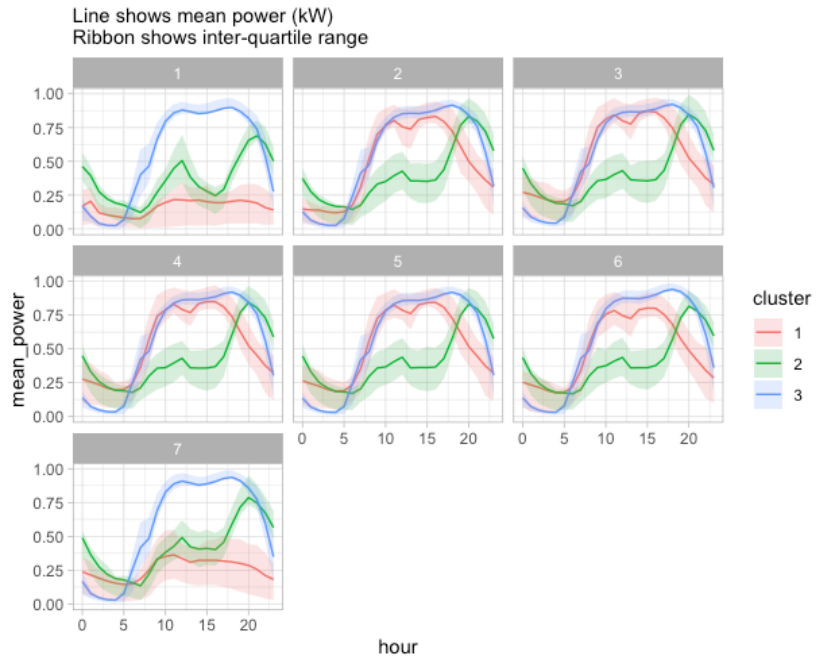
No normalisation



Normalisation of raw data

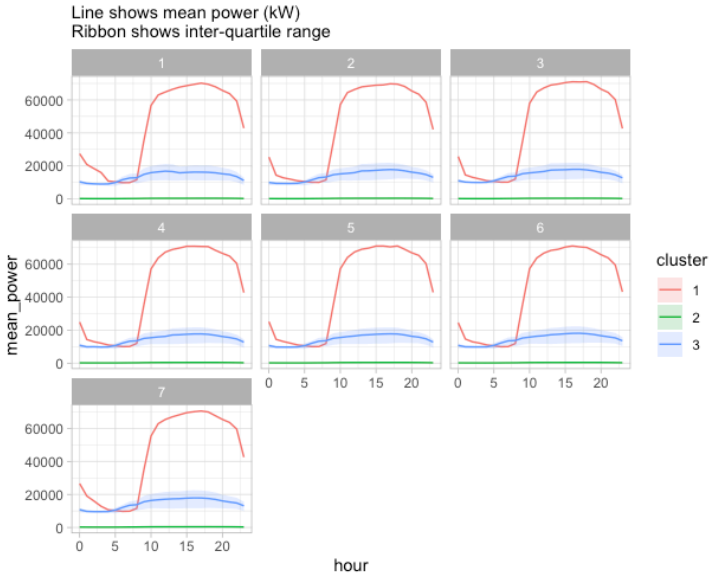


Normalisation of weekly average



Normalising the Data

- The plot on the right shows clustered profiles (k=3) when the data is not normalised
- The clusters are mainly split by consumption **volume** rather than **pattern**
- This may be useful for some purposes, but doesn't show the customer archetypes



Normalising between 0 and 1

$$\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Calculating the Principal Components

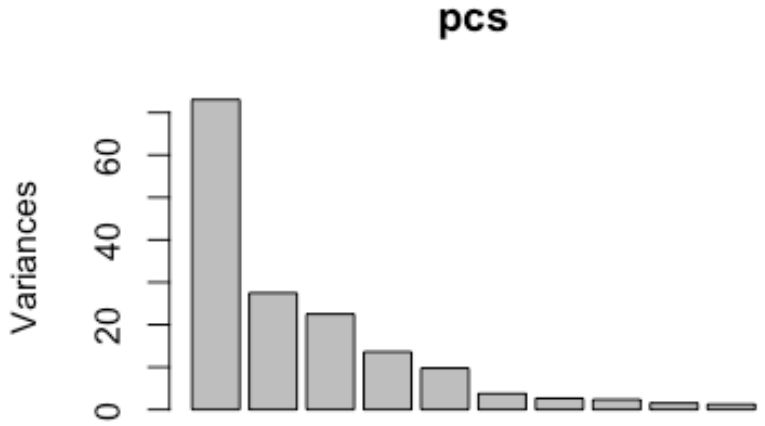
- PCA is sensitive to the magnitude of each **column** and data should be **standardised** (mean 0, standard deviation 1) to prevent variables with large magnitude from dominating the principal components
- Task: calculate the principal components:
  - Drop any non-numeric columns (e.g. customer ID)
  - Standardise each column with `scale()`
  - Calculate the principal components with `prcomp()`
  - Use `tidy()` to tidy the output of your PCA in a data frame called `tidy_pcs`

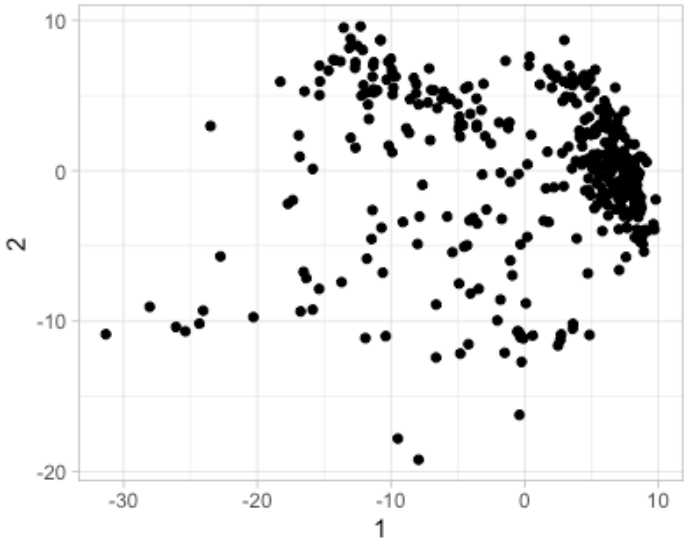
Use `scale(x)` to standardise each column of a data frame or matrix

Use `prcomp(x)` to perform PCA on numeric data x (matrix or data frame)

Plotting Principal Components

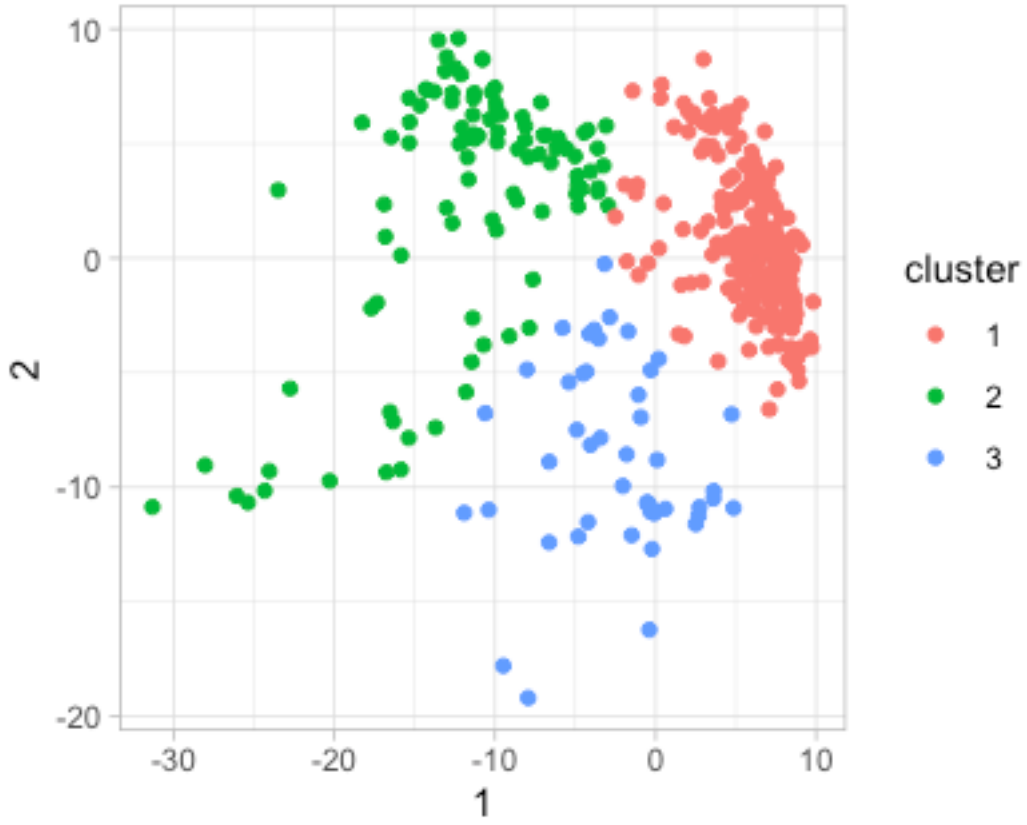
- Running `plot(pcs)` plots the variance of each of the principal components (AKA **scree plot**)
- Task: create the scree plot
- Plotting the two principal components against one another can be useful to identify the spread of the data
  - Right plot: we can see that there is one dominant cluster of customers
- Task: plot the first two principal components in a scatter plot





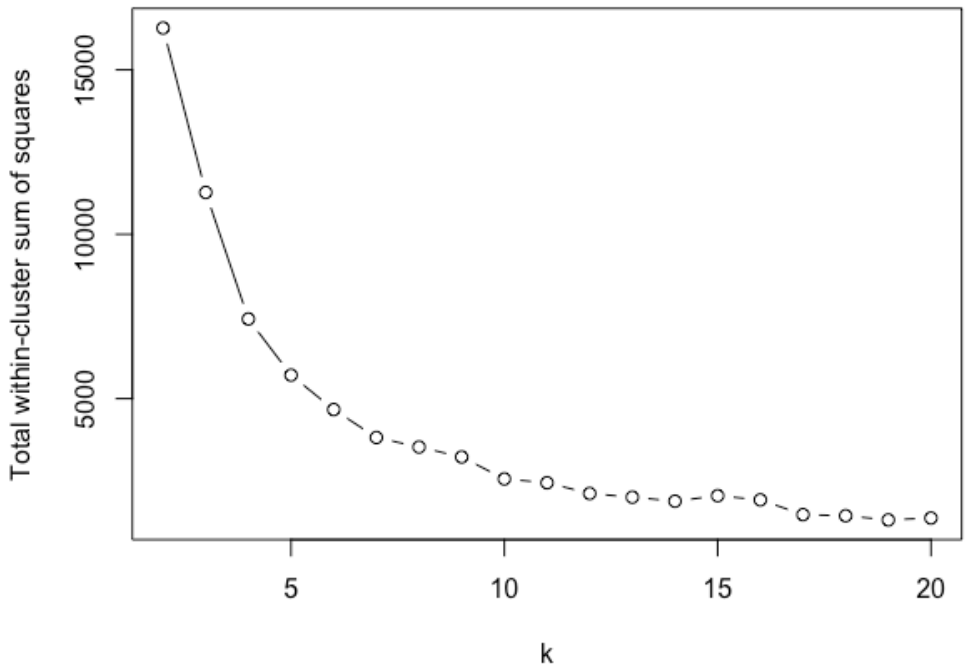
Clustering

- Task: cluster the data with k=3 based on the first two principal components
- Task: reproduce the plot on the right (cluster IDs may vary)
- Task: count how many customers belong to each cluster
- Task: experiment with producing the right-hand plot with different settings of k



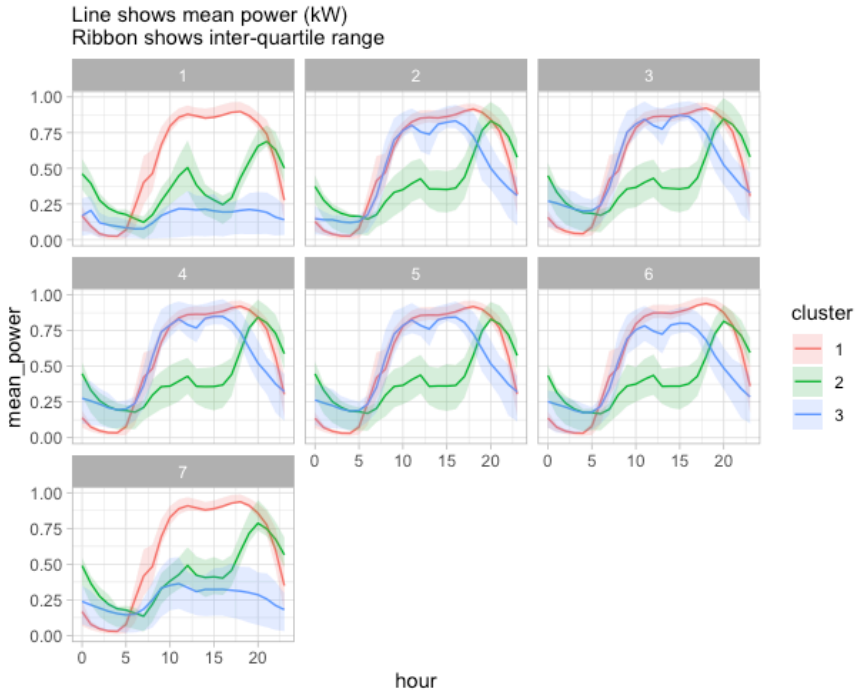
Choosing k?

- A common method for choosing the number of clusters is the **elbow method**
- The right-hand plot shows the amount of variance that is captured with each setting of k
- The **elbow** is the point where the curve flattens (point of diminishing returns)
- Task: use a for loop to calculate `tot.withinss` for each setting of k = 2—20
- Task: create the plot on the right hand side
- The elbow method is only one way to determine k and it is ambiguous. Often domain knowledge of visualisation are just as helpful! **In our anaylsis, we will stick to k=3**



Plotting Time Series By Cluster

- Finally, we will return back to the time series space and investigate the distribution of profiles in each cluster
- Task: plot the mean consumption profile for each cluster (see right)
- Task: add the inter-quartile range for each cluster
- How would you classify each customer type?



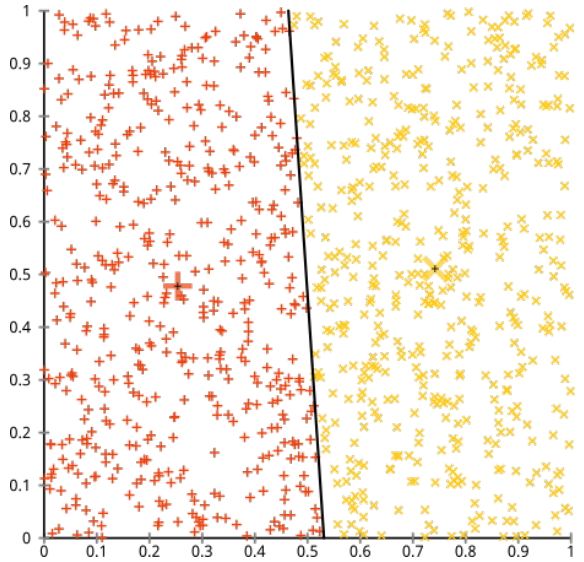
Note: 1 → Sunday, 2 → Monday etc.

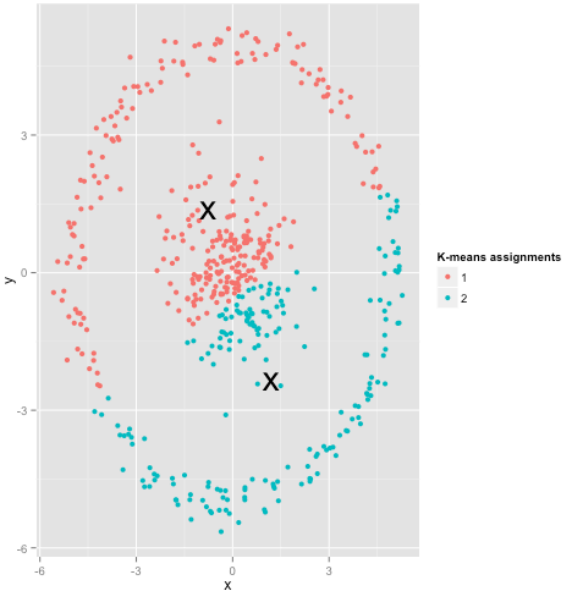
Limitations of k-means Clustering

- Clustering can be a very appealing and intuitive way to gain insight about data
- It is also very general and makes no assumptions about the data
- However, this generality comes with drawbacks:
  - Some data do not have clusters!
  - It assumes spherical clusters

[How to understand the drawbacks of k-means]

This data has no clusters!



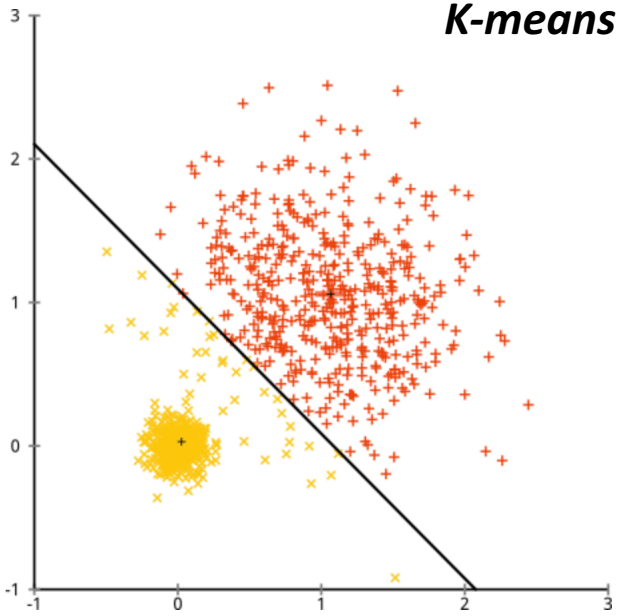


This data has non-spherical clusters!

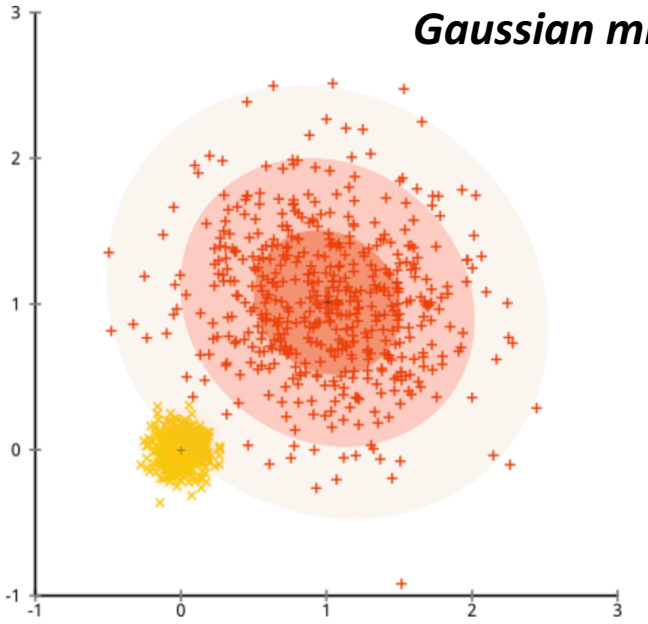
Other Clustering Algorithms

- Clustering algorithms trade-off computation time and number of parameters
- Alternatives to k-means include:
  - Gaussian mixtures: can incorporate information about covariance structure of the data (see right)
  - Hierarchical clustering: progressively combines similar clusters in a dendrogram; doesn't require number of clusters to be specified
- Further reading:
  - *An Introduction to Statistical Learning (Chapter 12), James et al.*
  - *Scikit-learn: Clustering*

K-means



Gaussian mixtures

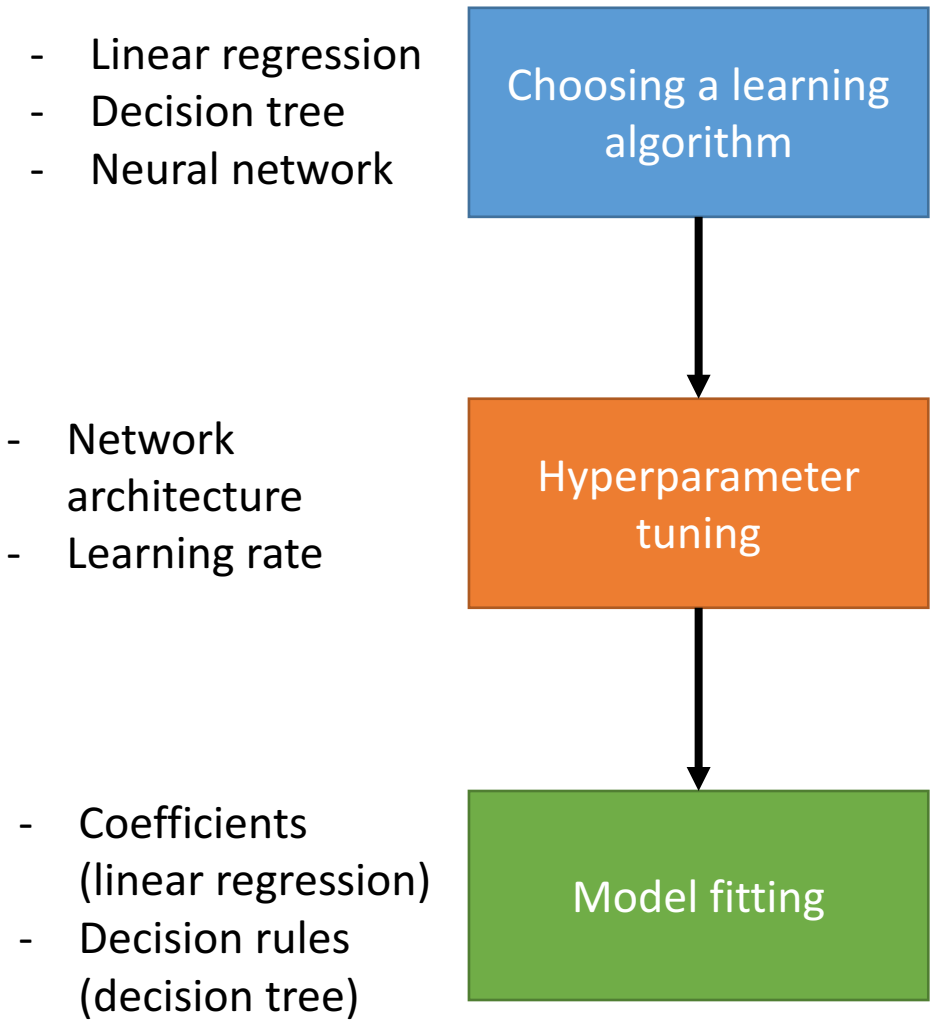


Supervised Learning: Model Selection

Model Selection

- There are infinitely many models that can be trained for a given supervised learning task, but not all will be equally strong predictors
- We want to choose a model that gives the best performance on held-out data and has appropriate characteristics for the task
- Model selection can broadly be split into 3 components:
  1. Choosing a learning algorithm
  2. Hyperparameter tuning
  3. Model fitting
- **Model fitting** is always done computationally using methods such as **gradient descent** while the other choices might be made manually based on knowledge of the problem or model requirements

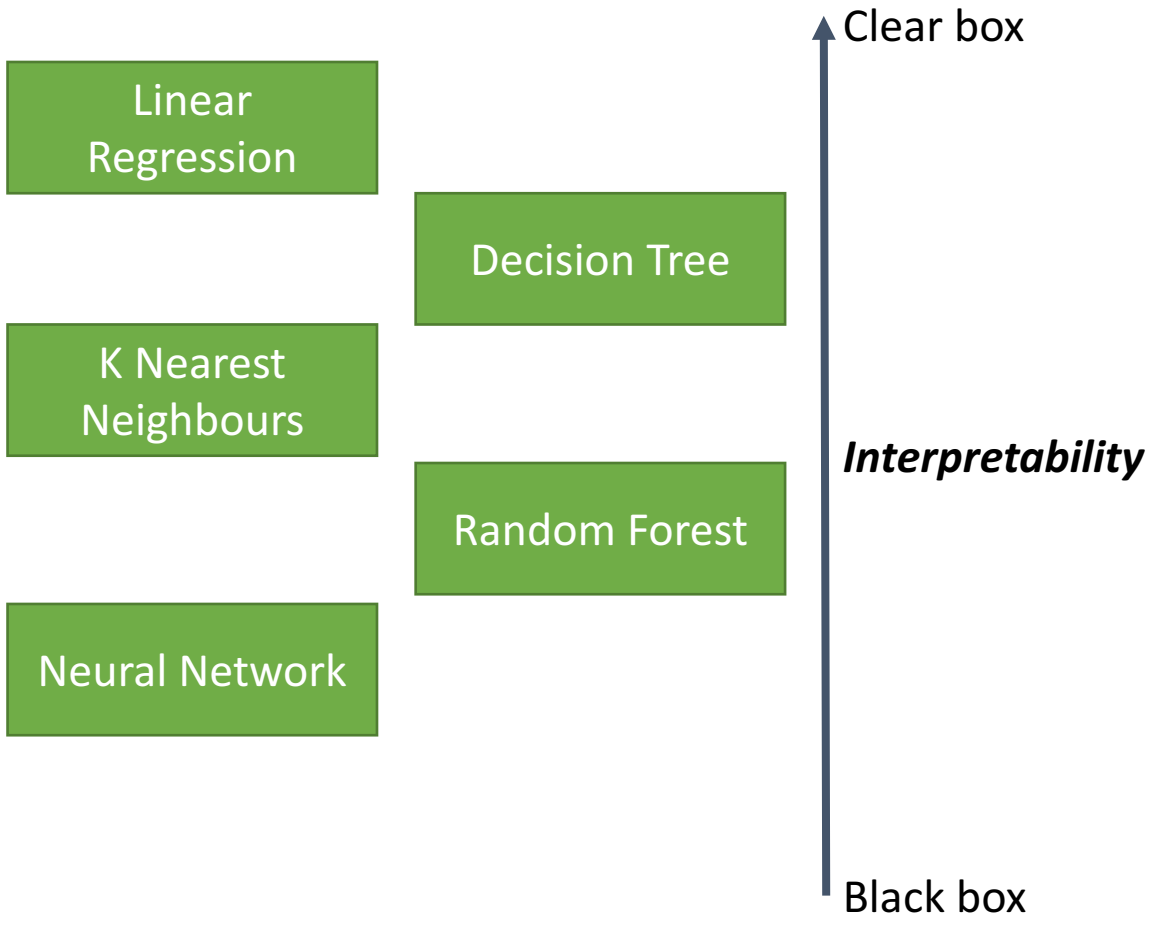
Hierarchical decisions in model selection



*Fitting is the process of adjusting model parameters to achieve the best performance - usually minimising a **loss function***

Choosing a Learning Algorithm

- Choosing the learning algorithm automatically can be **very computationally expensive,** so is often done semi-manually based on:
  - The model's predictive power
  - Interpretability
  - Computational expense
  - Prior experience!
- **Feature selection** is also an important part of choosing a learning algorithm

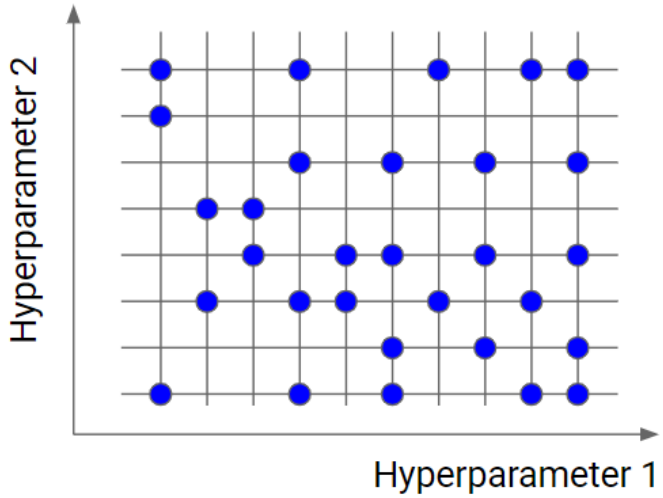


Hyperparameter Tuning

- Each learning method has its own hyperparameters which can significantly impact predictive power and convergence time
- Fitting and evaluating (e.g. with a validation set) multiple models with different hyperparameters often yields significant performance improvements
- One way to choose hyperparameters is to use **grid search**:
  - Manually choose a set of values for the parameters to tune
  - Fit and evaluate models for all combinations of the parameters
- Other methods include **random search** and **Bayesian optimisation**

Example hyperparameters

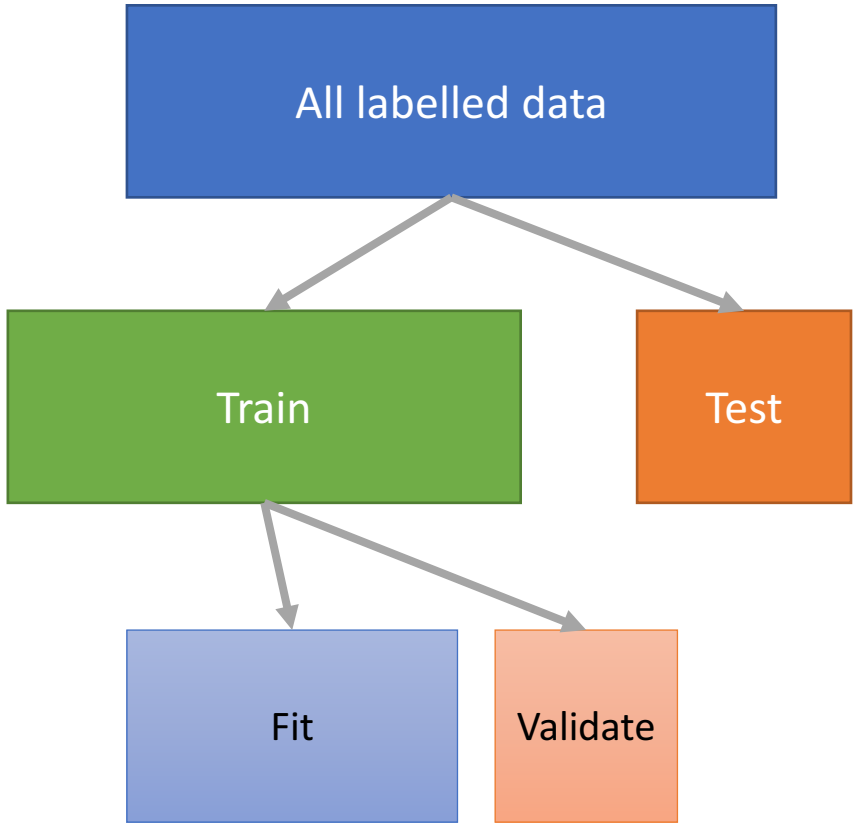
Decision tree	• Min. bucket size • Max. depth
Random forest	• # trees • Max. depth
Neural network	• Learning rate • # hidden layers



Train, Validate, Test

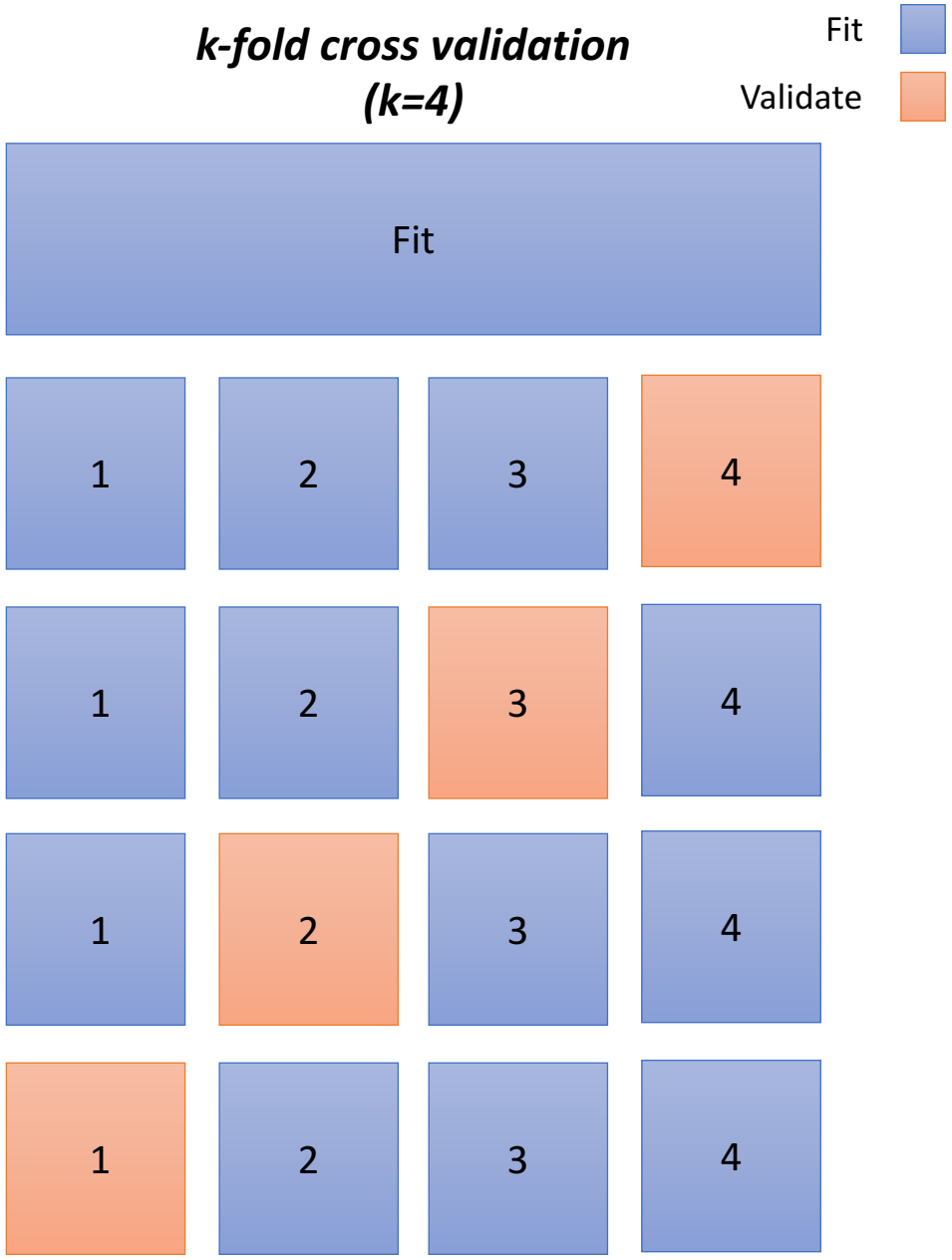
- Models which don't fit the training data well enough have been **underfitted**
- Models which fit the training data too well have been **overfitted**
- Overfitting is often more of a problem than underfitting
- In order to prevent overfitting, data is often split into training and validation sets
- **The testing data should only be used once** to evaluate the final model
- The validation data can be used to evaluate different models while controlling for overfitting during model selection

Common approach to splitting data for supervised learning



Cross-Validation

- The vanilla fit/validate split approach has two disadvantages:
  - The validation error can be very sensitive to how the data is split
  - As a large proportion of the data must be held-back for validation, we cannot use all the data for training and the validation error can over-estimate the test error
- **Cross-validation** methods fit multiple models with different fit/validate splits:
- The most widely used method is **k-fold cross validation**
  - Data is split into k 'folds'
  - Then we fit k models, each time leaving out 1 fold for validation, and calculate the average error



Cross-Validation: Example

- We will fit logistic regression models with different predictors using 10-fold CV, to predict the use of the washer dryer
- Task: read the training data and create a new column (as a **factor**) `washer_dryer` which is:
  - 1 if `appliances` contains 'washer_dryer'
  - 0 otherwise
- Task: drop NAs
- Task: run the code on the right-hand side
  - The code uses `cv.glm()` from the `boot` package to implement k-fold CV for generalised linear models
- Task: determine the strongest model based on the CV errors

Raw appliances column

"+fridge+tumble_dryer+washer_dryer"	"+fridge+washing_machine+washer_dryer"
"+fridge"	"+fridge"
"+fridge"	"+fridge"
"+fridge+washing_machine+washer_dryer"	"+fridge+washing_machine+washer_dryer"
"+fridge+washing_machine+washer_dryer"	"+fridge+washing_machine+washer_dryer"

Use `str_detect(string, pattern)` to determine pattern is found in string

`if_else(condition, true_val, false_val)` can be used with `mutate()` to create a new column with an if/else statement

k-fold CV for Logistic Regression

```
# Defining formulae to try
formulae <- c('washer_dryer ~ activePower + transient1',
              'washer_dryer ~ activePower + transient1 + transient2',
              'washer_dryer ~ current * voltage + transient1',
              'washer_dryer ~ activePower + current + voltage')

# K-fold CV
errors <- numeric()
for (f in formulae){
  fit <- glm(as.formula(f), family = 'binomial', data = train)
  cv_fit <- cv.glm(train, fit, K = 10)
  errors <- c(errors, cv_fit$delta[1])
}
```

Tools for Cross-Validation and Model Selection

- CV can require quite a lot of code (and iteration) to implement
- Several packages include functions to facilitate k-fold CV and other resampling techniques such as **bootstrapping**
- In particular: **modelr** and **rsample** (part of the **tidymodels** family)

Kaggle Competition – The Winners!

Jess Steinemann (73.2%)

Alfian Rahman (74.5%)

Ayokunle Arikawe (72.8%)