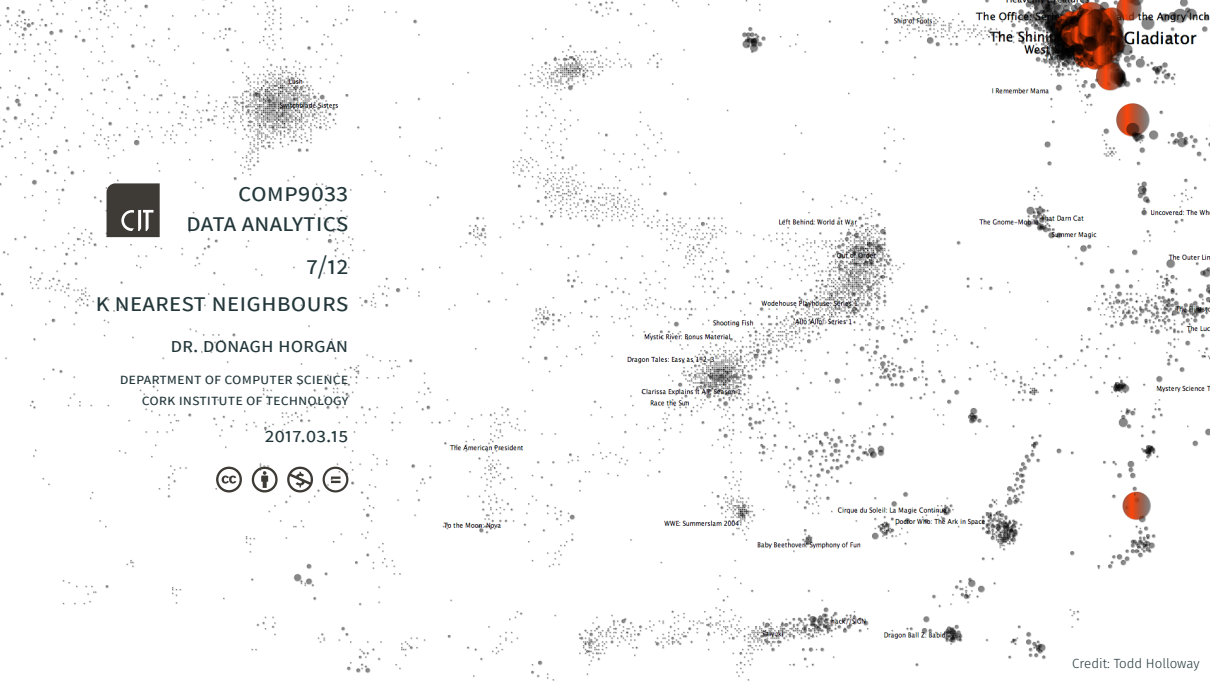


COMP9033

DATA ANALYTICS

7/12

K NEAREST NEIGHBOURS

DR. DONAGH HORGAN

DEPARTMENT OF COMPUTER SCIENCE

CORK INSTITUTE OF TECHNOLOGY

2017.03.15

Credit: Todd Holloway

Overview

1. Linear regression:
   - What it is.
   - How it works.
   - Measuring model error.
2. The least squares technique:
   - The residual sum of squares.
   - The least squares solution.
   - Performance considerations.
3. Shrinkage methods:
   - Problems with least squares.
   - Ridge regression.
   - Hyperparameters.
4. Subset selection:
   - Best subset selection.
   - Forward stepwise selection.
   - Backward stepwise selection.
   - Hybrid methods.

1. *k* nearest neighbours:
   - The *k* nearest neighbours algorithm.
   - Weighted observations.
   - Choosing the number of neighbours.
   - Choosing a distance measure.
   - Choosing a weighting scheme.
   - Advantages and disadvantages.

2. Recommender systems:
   - The long tail phenomenon.
   - Content-based recommenders.
   - User-based collaborative filters.
   - Item-based collaborative filters.
   - Advantages and disadvantages.
   - The Netflix Prize.

k nearest neighbours

- *k* nearest neighbours is supervised machine learning algorithm that can be used to build *both* classification and regression models.
- As with other machine learning algorithms, *k* nearest neighbours combines heuristics with statistics to produce a model.
- Given a set of historical examples, k nearest neighbours predicts new values by identifying the *k* "nearest" examples in the historical data, *e.g.*
  - Application memory usage at 15:00 might be estimated to be the average of the usage between 14:00 and 16:00 (two nearest neighbours) in the historical data.
  - A new credit approval application might be accepted or rejected based on the outcomes of the ten "most similar" applications seen previously.
  - How much a user might like the film Titanic can be estimated as the average of what the hundred users with the "closest" taste in films thought of that film.

- The k nearest neighbours algorithm works as follows:
  1. Gather a labelled data set, *i.e.* a data set whose records consist of one or more explanatory variables ($X = \{x_1, x_2, \ldots, x_n\}$) and an associated target variable (y), which is to be predicted. Store this data for later use.
  2. Select a single unlabelled data record for which a target value is to be predicted, *i.e.* a record consisting of explanatory variables with no associated target variable value.
  3. Determine the k most similar records to the candidate record in the set of stored data by comparing the values of the explanatory variables.
  4. Compute a target variable prediction ($\hat{y}$) for the candidate record based on the k most similar records identified in Step 3.
- Generally, the weighted average is used make predictions in regression models, while the weighted mode is used to make predictions in classification models.

- The *weighted average* is a variation on the average:
  - Multiplicative weights are applied to each data point in the sample.
  - By choosing different weight values, we can increase or decrease the effect of individual points on the final result.
- The weighted average of the sample X is defined as

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}, \tag{7.1}$$

  where w_i denotes the i^{th} weight.
- When all the weights are equal to one (*i.e.* $w_i = 1, \forall\ i \in \{1, 2, \ldots, n\}$), the weighted average is equivalent to the average.

- The *weighted mode* is a variation on the mode:
  - Weights are assigned to each data point in the sample.
  - By choosing different weight values, we can increase or decrease the effect of individual points on the final result.
- The weighted mode of the sample X is defined as the value in the sample with the highest sum of weights.
- When all the weights are equal to one (*i.e.* $w_i = 1, \forall\ i \in \{1, 2, \ldots, n\}$), the weighted mode is equivalent to the mode.

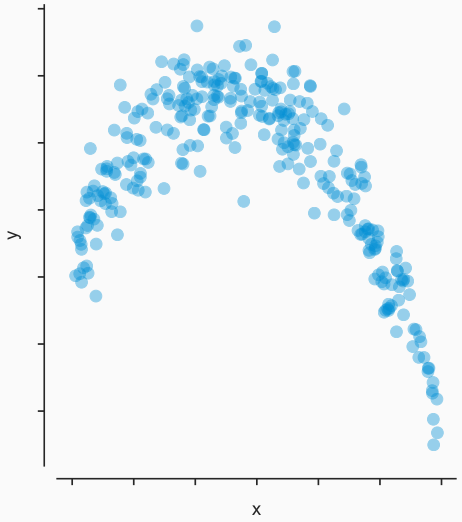
- When applying k nearest neighbours, there are three hyperparameters we must consider:
  1. The number of neighbours to use, *i.e.* k.
  2. A method by which to evaluate how "close" examples are, *i.e.* we must specify some way of determining the distance between pairs of neighbours.
  3. A method by which to weight the k target variables, *i.e.* once we've determined which k neighbours are the nearest, how do we use this information to make our prediction?
- As usual, we can use model selection via cross validation to choose the optimal set of hyperparameters so that our model accuracy is maximised.

- The number of neighbours, k, has a significant effect on the behaviour of the model.
- Choosing a small value of k makes the model more sensitive to data that is "nearby":
  - Neighbours with extreme values have a larger effect on predictions.
  - Can lead to high variance error.
- Choosing a large value of k makes the algorithm more sensitive to data that is "further away".
  - Neighbours with extreme values have a smaller effect on predictions.
  - Need to use more neighbours to make a prediction though — can dilute the effect of valuable nearby neighbours.
  - Can lead to high bias error.
- Typically, it is not obvious what value of k will produce the best results.

- A *distance measure* is a function that *estimates* how similar one sample is to another.
- Distance measures can be applied to quantitative data (*e.g.* Euclidean distance) and categoric data (*e.g.* Jaccard distance).
- Generally, they compute distances in the range $[0, \infty)$, *i.e.*
  - The more similar X and Y are, the smaller their distance is.
  - The more dissimilar X and Y are, the larger their distance is.
- Some measures are more *efficient* than others, depending on the situation.
- As usual, we can use model selection via cross validation to determine which measure is best in a given situation.

- One way to measure the difference between data samples using their Manhattan distance, $M(X, Y)$, *i.e.*

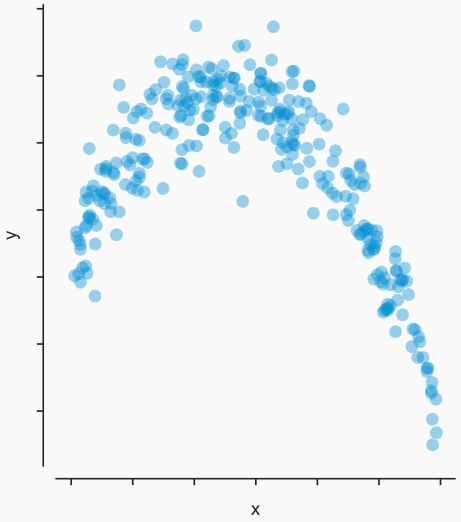
$$M(X, Y) = \sum_{i=1}^{n} |x_i - y_i| \,. \qquad (7.2)$$



- We can also measure the difference between data samples using their Euclidean distance, $E(X, Y)$, *i.e.*

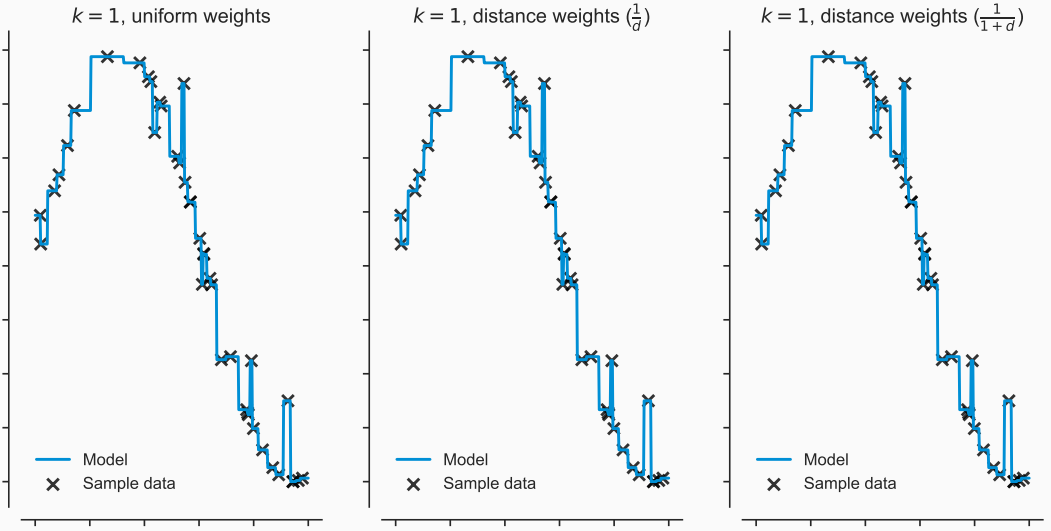
$$E(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \qquad (7.3)$$

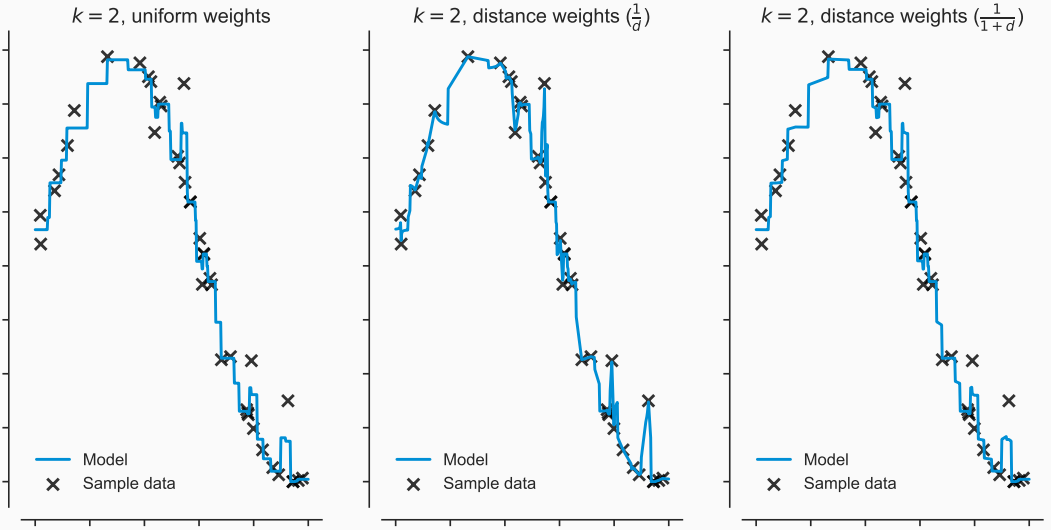
- Euclidean distance is generally larger than Manhattan distance in samples with extreme differences (relies on squares rather than absolute value).

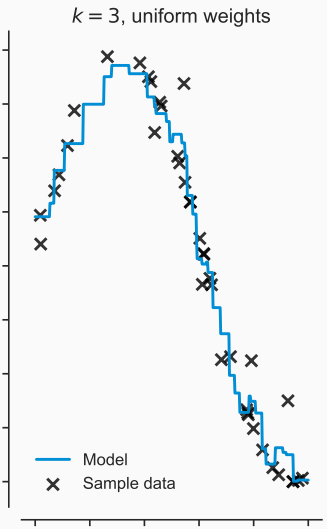


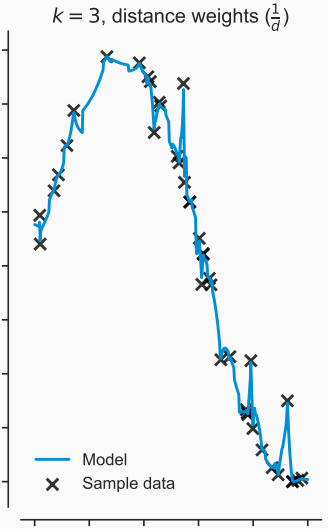
- Distance measurements like Manhattan distance and Euclidean distance are sensitive to the scale of the samples, X and Y.
- If the scales of the samples are very different, then their distance tends to be very large, even though the samples themselves may behave similarly, *e.g.*.
  - *X* is measured in kilobytes while *Y* is measured in gigabytes.
  - *X* is measured in metres while *Y* is measured in kilometres.
- One way to compensate for this is to standardise the samples, so that each sample has zero mean and unit standard deviation.

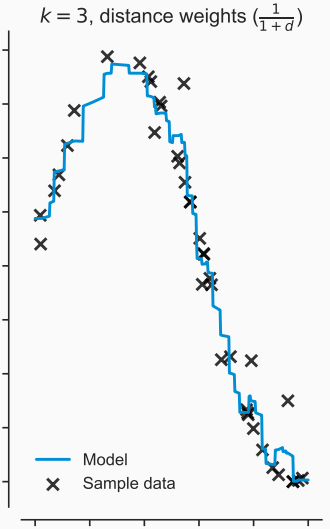
- Many weighting schemes are possible, but uniform and distance-based weighting are commonly used.
- In uniform weighting, all samples count equally when computing the prediction:
  - All weights are equal, *i.e.* $w_i = \frac{1}{n}, \forall\, x_i \in \{x_1, x_2, \ldots, x_n\}$.
  - Effectively, this reduces the weighted average/mode to just a simple average/mode calculation.
- In distance-based weighting, closer samples have a greater influence on the prediction:
  - The weight for each neighbour is set proportionally to the inverse of its distance from the sample under consideration.
  - Weighting by distance means that our prediction is more strongly influenced by neighbours that are closer.

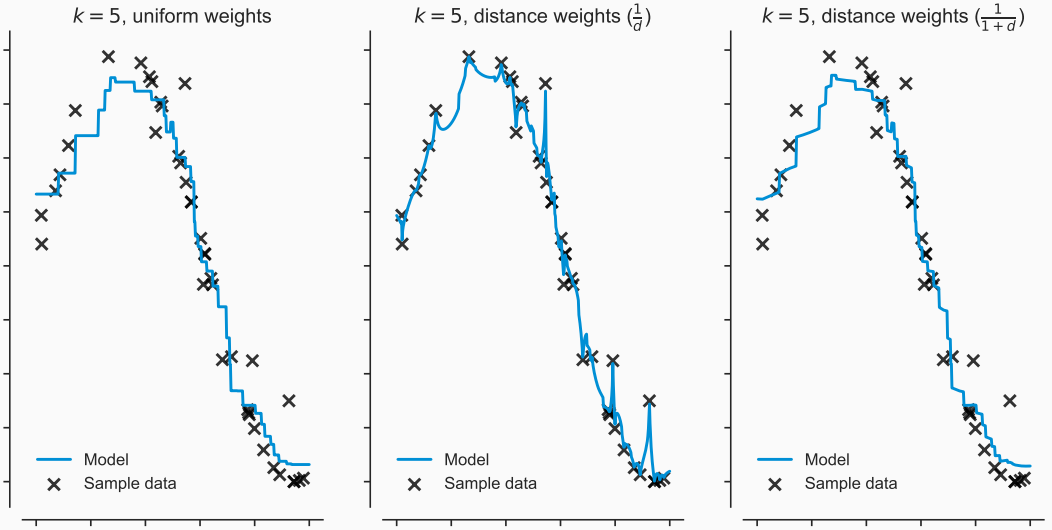

$k = 1$, uniform weights
$k = 1$, distance weights ($\frac{1}{d}$)
$k = 1$, distance weights ($\frac{1}{1+d}$)
Model
Sample data
Model
Sample data
Model
Sample data

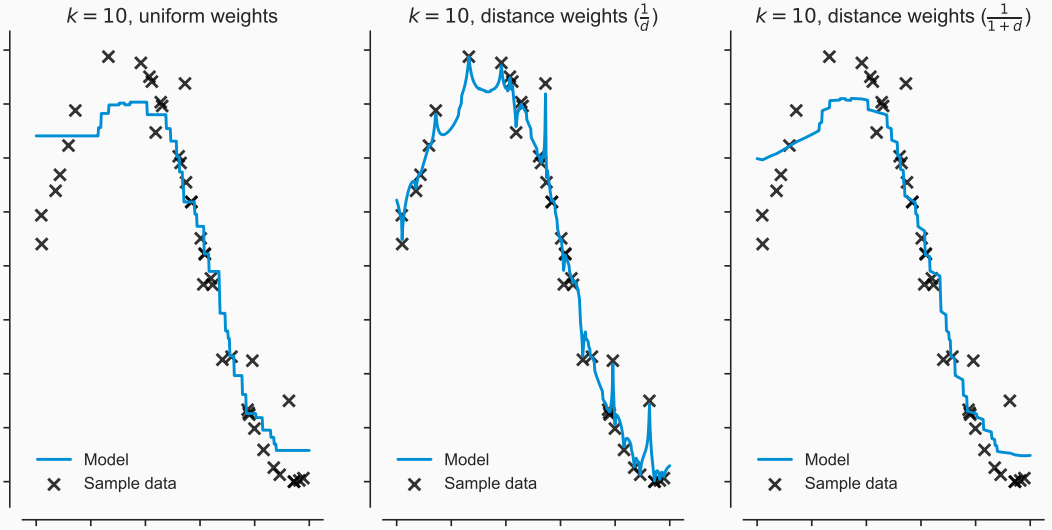

$k = 2$, uniform weights
$k = 2$, distance weights ($\frac{1}{d}$)
$k = 2$, distance weights ($\frac{1}{1+d}$)
Model
Sample data
Model
Sample data
Model
Sample data


$k = 3$, uniform weights
Model
Sample data


$k = 3$, distance weights ($\frac{1}{d}$)
Model
Sample data


$k = 3$, distance weights ($\frac{1}{1+d}$)
Model
Sample data


$k = 5$, uniform weights
$k = 5$, distance weights ($\frac{1}{d}$)
$k = 5$, distance weights ($\frac{1}{1+d}$)
Model
Sample data
Model
Sample data
Model
Sample data


$k = 10$, uniform weights
$k = 10$, distance weights ($\frac{1}{d}$)
$k = 10$, distance weights ($\frac{1}{1+d}$)
Model
Sample data
Model
Sample data
Model
Sample data

- *k* nearest neighbours is a simple and intuitive technique.
- It's useful in situations where we can't make many assumptions about the behaviour of the data:
  - For instance, in linear regression, we assume that new values can be predicted through linear combinations of historical values.
  - However, in many situations, this is not true, and so applying linear regression can lead us to produce models that will make bad predictions.
  - Consequently, in certain situations, using *k* nearest neighbours may be preferable to using another regression or classification technique.

- We must store the entire training set in memory, in order to search through it for neighbours later (can use a lot of memory if the training set is large).
- As the number of historical examples grows, the cost of prediction increases:
  - Each time we want to make a prediction, we must compute distances for *each* of the historical examples, in order to find the k nearest ones.
  - Indexing algorithms (*e.g.* k-dimensional trees) can help to reduce this burden somewhat.
- Prediction quality can be unstable when k is small:
  - We can think of the set of k nearest neighbours as a sample that, ideally, represents the population of the target well.
  - If k is small, then this may not be true.
- In contrast to linear regression and decision trees, a k nearest neighbours model cannot be represented using an equation or diagram.

Recommender systems

- Until relatively recent times, physical systems have dominated our lives, *e.g.*
  - We bought our goods in physical stores.
  - We got our news by reading physical newspapers.
- However, in the past twenty years, things have changed significantly:
  - We buy goods online.
  - We get news, audio and video from online media sources.

- Much of this change has been driven by the limitations of physical systems, *e.g.*
  - Shelf space is limited in a physical store.
  - Page space is limited in a physical newspaper.
- Virtual systems usually don't suffer from these limitations, *e.g.*
  - Products can be stored in different physical locations, but presented to us in a single location.
  - Webpages can be dynamically scaled to fit any content size.

- As a result, book shops only stock the most popular books, video stores only stock the most popular titles, *etc*., while online retailers stock an enormous variety of goods.
- This is known as the long tail phenomenon, as illustrated to the right.
- However, the long tail phenomenon creates a whole new problem: with such huge variety, how can users find items that are relevant to them?

- One solution is the use of *recommender systems*, *i.e.* systems that can make recommendations to users about new items based on information about other users and items.
- By exploiting items' properties and/or hidden relationships in users' behaviour, we can determine which items are the most relevant and present these to the user.

- The term *recommender systems* refers to a class of predictive algorithms that make recommendations by determining similar users or items.
- There are two main varieties:
  1. Content-based recommenders:
     - These determine similarity by examining the properties of items and comparing them to the properties of other items.
  2. Collaborative filtering recommenders:
     - These determine similarity of items by examining the similarity of their ratings from users.
- Each variety has its uses and, in a given situation, may produce better recommendations than the other (can use model selection to determine which is best).

- Content-based systems recommend items based on the similarity of their *properties*, *e.g.*
  - They might recommend Star Wars if you have watched Indiana Jones because they were both written by George Lucas.
  - They might recommend Johnny Cash if you have listened to Hank Williams because they are both classified as country music.
- For a given item, content-based recommendation works as follows:
  1. The properties of the item are profiled and added to a catalogue or database of item properties.
  2. The properties of the item are then compared to the properties of every other item in the database and a similarity measure is computed.
  3. The most similar matches for the item are identified and stored.
  4. When the item is selected by a user, the matches are retrieved and shown as recommendations.

- Content-based systems have several advantages:
  - They can recommend completely new items, *e.g.* new films/products.
  - Usually, the most similar items for a given item only need to be recomputed when new items are added to the system, as the properties of each item are static.
- However, they also have several disadvantages:
  - The properties of each item must be profiled, which can be a time consuming and sometimes manual (*i.e.* human) effort.
  - If the number of properties is small, the variety of forms of similarity can be limited, and so the number of recommendations can be limited too.
  - User preferences are not taken into account, and so items are recommended based *only* on the similarity of their properties.

- Collaborative filtering systems recommend items based on user preferences.
- They come in two varieties:
  1. User-based filters
     - These recommend items to you based on your similarity to other users.
     - For instance, they might recommend Johnny Cash because you like the same variety of music as users who also like Johnny Cash.
  2. Item-based filters
     - These recommend items to you based on the similarity of their ratings.
     - For instance, they might recommend Star Wars because you liked Indiana Jones and users who have seen both, like both.
- We can use k nearest neighbours to determine the k most similar users/items in each case.

- Unlike content-based recommenders, collaborative filtering *requires* user ratings.
- Generally, we can get these in two ways:
  1. Ask users to rate items.
     - Example uses include IMDB, Amazon, Airbnb.
     - Can be effective, if many users rate items.
     - However, in general, users are not willing to provide responses.
     - Can also lead to sampling bias, as those who do respond may be a non-representative sample of the general population of users.
  2. Observe user behaviour and infer preferences.
     - Example uses include Google, Facebook, Spotify.
     - User action is interpreted as a “like” (*e.g.* listening to a song, watching a video).
     - Doesn’t require users to rate items.
     - However, there is usually no way to distinguish inaction from “dislikes”.

- Collaborative filtering systems have several advantages:
  - Like content-based systems, they can recommend completely new items.
  - As the recommendations are based on user preferences, they generally correlate with the perceived quality of items.
- However, they also have several disadvantages:
  - Recommendations can only be made for items that have been rated by users.
    - Items that haven't been rated cannot be recommended.
    - Niche items don't get recommended as often as mainstream items (*i.e.* there is a popularity bias).
  - Recommendations can be poor when there is not a *reasonable* amount of *reliable* data available.
    - For instance, if the number of ratings is small (*i.e.* sparse data), items may appear to be more similar than they actually are.
    - Can impose a minimum number of ratings before recommendations are made to prevent this.

- User-based filtering is simpler when making predictions for just one user:
  - In user-based filtering, if we compute similarity of our target user to the other users, then we can predict ratings for as many items as we like.
  - In item-based filtering, we must compute the similarity of the target item to all the other items for each item we want to predict a rating for.
- Item-based filtering can be significantly faster on large datasets:
  - The similarity between items changes less frequently than the similarity between users, *e.g.* the relative ratings of Raiders of the Lost Ark and The Last Crusade are not likely to change over time, whereas the similarity of two specific users may diverge significantly over their reaction to Kingdom of the Crystal Skull.
  - As a result, item similarities can be precomputed offline and used for long periods of time before needing to be updated.
- Item-based filtering can be a better choice for sparse data sets:
  - Can be easier to find items that are similar because they belong to the same category (*e.g.* Indiana Jones movies) than users that are similar because they share the same taste (*e.g.* they have seen many movies in common).

- In 2006, Netflix offered a million dollar prize for the first algorithm to beat its own movie recommender by 10%.
- Netflix provided 100 million ratings for nearly 500,000 users.
- Netflix's own algorithm was beaten in less than a week, but it took nearly three years to beat it by 10%.
- The team behind BellKor's Pragmatic Chaos algorithm claimed the prize.
- In the end, Netflix didn't implement the winning algorithm because of the effort required!
- Sometimes, increases in accuracy are outweighed by the cost of improvement.

Summary

- *k* nearest neighbours:
  - Used for regression and classification.
  - Three hyperparameters: the number of neighbours, the distance measure, the weighting scheme.
  - Useful when we can't make assumptions about linearity.
  - Cost of prediction becomes high at scale, can be unstable if *k* is small.
  - Can be used to build recommender systems.
- Lab work:
  - Build a *k* nearest neighbours classification model for SMS spam data.
  - Build a *k* nearest neighbours regression model for server load data.
  - Build a user-based recommender system for films.
- Next week: decision tree classification and regression!

1. Hastie et al. *The Elements of Statistical Learning: Data mining, Inference and Prediction.* 2nd edition, February 2009. (stanford.io/1dLkiAv)
2. Ullman et al. *Mining of Massive Data Sets.* Cambridge University Press, 2014. (stanford.io/1qtgAYh)
3. Segaran, Toby. *Programming Collective Intelligence: Building Smart Web 2.0 Applications.* O'Reilly, 2007. (oreil.ly/1nzWODy)
4. Masnick, Mike. *Why Netflix Never Implemented The Algorithm That Won The Netflix $1 Million Challenge.* Techdirt, 13th April 2012. (bit.ly/1BdyZbW)