2018

Machine Learning and Real-World Data Notes

UNIVERSITY OF CAMBRIDGE, PART IA

ASHWIN AHUJA

Table of Contents

Machine Learning and Real-World Data

Statistical Classification

Introduction to Machine Learning

- Machine Learning involve the following three aspects
    - 1) Task
    - 2) Data
    - 3) Algorithm
- Therefore, we consider
    - 1) Data Acquisition and Preparation
    - 2) Feature Extraction
    - 3) Evaluation
- **Task**
    - An abstraction from a real problem, or a piece of a larger architecture
        - In research, the large architecture is often hypothetical.
    - End-user systems often very different from experimental systems
        - Generally, research concerns standard tasks
        - Which don't really matter much in the real world
        - Reason why lots of high profile uses games – this is a task which is very easy to define.
        - Sometimes, subtle change in task makes a huge difference to research progress
            - Became easier to make progress in statistical machine translation (SMT) when the task was redefined as producing the closes match to a reference translation as measured by the BLEU score
- **Data**
    - Split into three parts
        - Training Set
        - Development (also known as the validation or tuning set)
        - Evaluation (also known as test set)
    - Acquiring Data
        - For a language processing, the data will be text (a corpus), and could be scraped from the web.
        - Can also have additional materials, such as translations into another language (parallel text)
        - For supervised learning, the available labelled data is usually quite limited – the data must be split to be used in the first three phases (Training, Development, Testing)
            - Split because we want to have new data for testing and tuning on different sets of data
            - Avoid overtraining

- How to pick and pre-process data
    - What type of data?
    - Where should data come from?
    - How much data?
    - Is annotation needed for training or evaluation?
    - What sort of pre-processing?

- **Supervision**
    - Can have varied degrees of supervision
        - Supervised – training data is labelled with the desired outcome
            - Can be annotated manually or using a found annotation (star ratings for movie reviews).
        - Unsupervised
        - Semi-supervised
- **Features**
    - Once data has been acquired, features are extracted by the algorithm
    - External resources may be used
- **Algorithm**
    - An algorithm should be chosen which is appropriate for the task, given the available data and features.
    - Fast and dumb may be better than slow and sophisticated – especially if large amounts of data are available.
    - For any practical application, robustness to unexpected data and consistency across datasets may be more important than obtaining the highest score on evaluation dataset.
- **Evaluation**
    - Allows one to see how well a chosen algorithm performs on a given task
    - Many different metrics and approaches for different tasks
    - Evaluation may be done using humans, but for standard tasks, there are standardised metrics and test sets
    - Important to note that the best performance will depend on the details of the task

Introduction to Sentiment Classification

- We are using reviews from IMDb (Internet Movie Data Base) which has about 4.7 million titles. They have reviews which are written in natural language by the general public.
- **Sentiment Classification –** Task of automatically deciding whether a review is positive or negative based on the text – this is a standard task in NLP (Natural Language Processing)
- One possible method is using data about individual words to find the sentiment
    - **Using a lexicon** which lists over 8000 words as positive or negative
    - **Hypothesis:** A review that contains more positive than negative words is overall positive.

- It is clear that the system is not perfect, therefore important to find out how accurate we are.
- We can find out that whether it is actually positive or negative based on the star rating
    - $Accuracy = \frac{numberCorrect}{numberCorrect + numberIncorrect}$
- Accuracy of 0.635
    - Using first word to break ties - 0.6356
    - Ignoring 'weakly positive' and 'weakly negative' words – 0.664
    - Using 'weakly positive' and 'weakly negative' as half as good as 'strongly negative' and 'strongly positive' – 0.643
- Problems with using a lexicon
    - It is built using human intuition
    - Requires many hours of human labour to build
    - Is it limited to the words the humans decided to include
    - It is static – words could have different meanings in different demographics

- **Tokenization**
    - Since we are looking at words, we must divide the text into words – splitting on whitespace is often not enough
        - Words with upper case
        - Punctuation
    - **Type vs Token**
        - A token is any word, repeated or not
            - Could say that it is a word in a specific context, but can also just be a word, depending on how simplified our model is.
        - A type is a different word to those seen before
        - *The old horse went to stand next to the old man has 11 tokens but only 8 types*

Naïve Bayes Parameter Estimation

- **What is Machine Learning?**
    - A program that learns from data
    - Adapts after having been exposed to new data
    - Learns implicitly
    - Without explicit programming
- **Feature**
    - Easily observable (and not necessarily obviously meaningful) properties of the data
    - In this case, words of the review
- **Classes**
    - Meaningful labels associated with the data

- Positive or Negative in our case
- Classification is a function that maps from features to a target class
  - A function mapping from the words in the review to a sentiment is what we are looking for.
- Probabilistic Classifiers
  - Given a set of input features, a probabilistic classifier returns the probability of each class.
  - Therefore, our prediction, is the highest probability one, given the features.

$$\hat{c} = \operatorname*{argmax}_{c \in C} P(c|O)$$

- Naïve Bayes Classifier
  - Liner Classifier
    - Uses a linear combination of the inputs.
  - Generative Classifier
    - They build a model of each class
    - Given an observation, they return the class most likely to have generated the observation
    - **Discriminative Classifiers are the other type**
      - They instead learn what features from the input are most useful to discriminate between the different possible classes
  - Based on Bayes' Theorem
    - Literally is basically just Bayes' Theorem – the trick is removing P(O)
  - $P(c|O) = \frac{P(c)\,P(O|c)}{P(O)}$ *where O are the set of Observed Features*
  - $predicted\ class\ (c_{NB}) = argmax_{c \epsilon C} \frac{P(c)P(O|c)}{P(O)}$
    - Argmax is the one with the maximum posterior probability
  - We can remove P(O) because it will be constant during a given classification and therefore will not affect the result of argmax
  - Therefore:
    - $predicted\ class\ (c_{NB}) = argmax_{c \epsilon C} P(c)P(O|c)$
    - P(c) is the prior probability and P(O|c) is the likelihood of the Observed features
  - For us P(O|c) = P(w1, w2, …, wn | c)
    - W1 is a word
  - **We assume there is strong independence assumption between the observed features – Naïve Bayes assumption**
    - P(O|c) ≅ P(w1|c) x P(w2|c) x … x P(wn | c)
    - So,

$$c_{NB} = \operatorname*{argmax}_{c \in C} P(c) \prod_{i=1}^{n} P(w_i|c)$$

- In the **training phase,** we find whatever information is needed to calculate P(wi|c) and P(c)
- In the **testing phase,** we apply the formula to find the classifiers decision
- **This is Supervised ML, since you use information about the classes during training**

- **Testing vs Training**
  - Testing: the process of making observations about some known data set
  - Testing: the process of applying the knowledge from training into some, new unseen data
- Implementing Naïve Bayes
  - Getting word probabilities

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}$$

    - Where count (wi, c) is the number of times wi occurs with class c and V is the vocabulary of all words
  - Getting class probabilities

$$\hat{P}(c) = \frac{N_c}{N_{rev}}$$

    - Where Nc is the number of reviews with class c and Nrev is total number of reviews
  - **This all uses Maximum Likelihood Estimation (MLE) as a method to estimate the parameters of a statistical model given observations**
  - Unknown Words
    - Ignore them is the best strategy
  - **Stop Words**
    - These are very frequent words, like ‘the’ and ‘a’ which really don’t mean anything at all.
    - We can ignore these by sorting the vocabulary by frequency and getting rid of the first 10.
    - However, doesn’t always work very well.
  - **USING LOGS**
    - In practise we use logs to deal with multiplying lots of very small probabilities together
      - Avoid underflow and increase speed
    - So

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}}\ logP(c) + \sum_{i=1}^{n} logP(w_i|c)$$

    - Since log is monotonically increasing, we can still simply look for argmax

        - However, if the count is 0 for a word for a class, we will attempt to say that the probability of that word being in that class is 0 (and add negative infinity in the case of the logs), so it disappears entirely.
        - Fix this by adding smoothing:

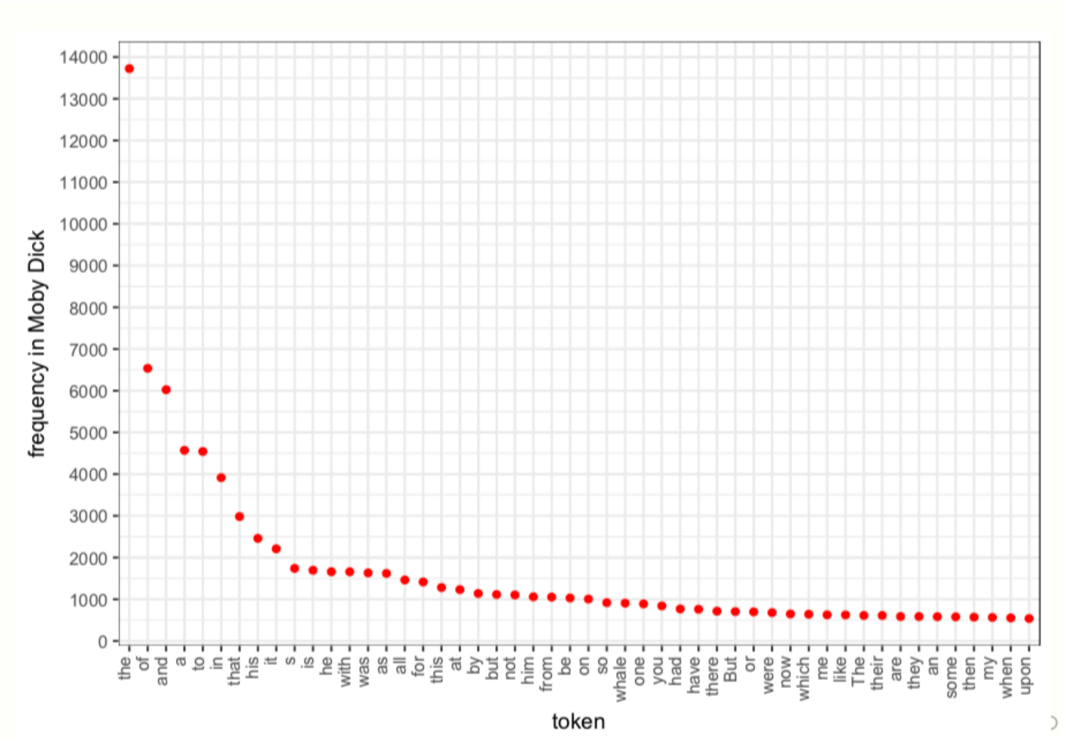
- **Laplace (Add-One) Smoothing**

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V}(count(w, c) + 1)} = \frac{count(w_i, c) + 1}{(\sum_{w \in V} count(w, c)) + |V|}$$

    - This fixes the problem with discovering new words
- **Binary Multinomial Bayes**
    - Same as previously, but for each review, we remove all duplicate words before concatenating them into the single big one to go through
- Further information on Naïve Bayes
    - Naïve Bayes is generally used as an effective baseline in modelling human languages
    - It clearly has assumptions which are wrong, but this is true of all formal models of human language
    - We are forced to use statistical models which are wrong because of two main things
        - Computational Tractability
        - Acquisition of Training Data
    - The Naïve Bayes assumption assumes that everything is independent from each other
        - Example of 1000 x 1000 binary pixels on a monitor (which have too many states to model exhaustively)
        - Naïve Bayes assumption is equivalent to saying that the pixels on the monitor all behave independently of each other.
        - Naïve Bayes works when we are interested in an overall characterization of some collection of sub-entities – the overall shade of the monitor – not the fact that it is displaying a cat.
    - More intelligent algorithms will take advantage of interactions between states – working with correlations which exist.

Statistical Laws of Language

- Word Frequency Distributions Power Law
    - Small number of very high-frequency words
    - Large number of low-frequency words

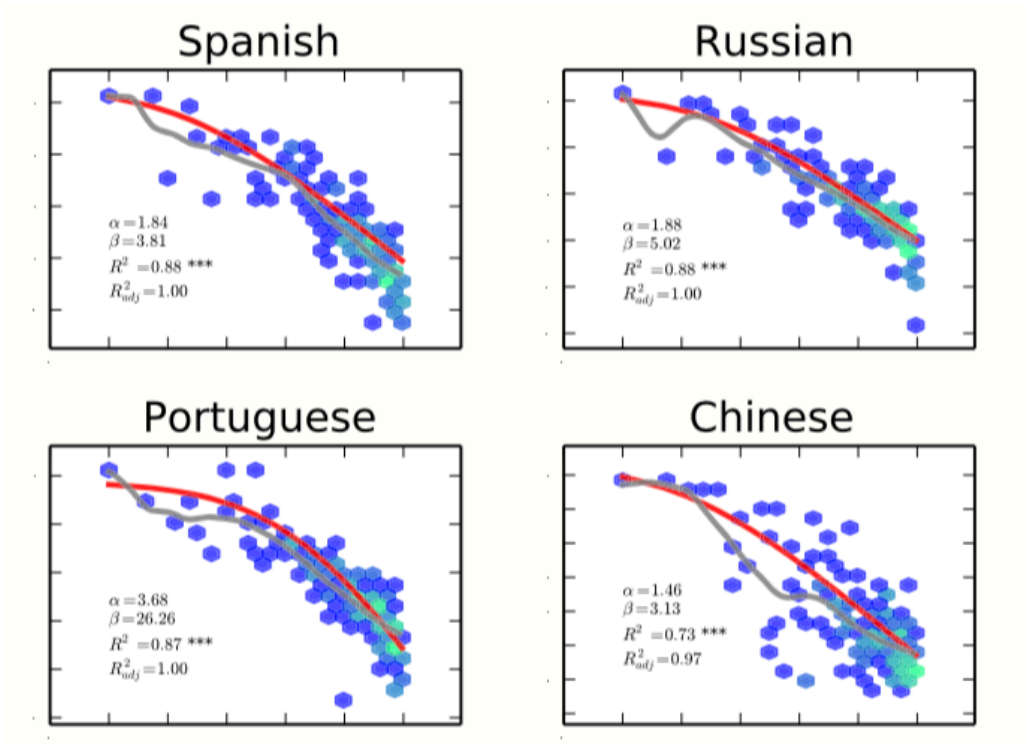


- The distribution obeys a power law – **Zipf's Law**
  - The nth most frequent word has a frequency proportional to 1/n
  - "A word's frequency in a corpus is inversely proportional to its rank"
  - The parameters of Zipf's Law are language dependent
  - $f_w \approx k/r_w^\alpha$
  - f_w is the frequency of word w
  - r_w is the frequency rank of the word w
  - α, k are constants (which vary with the language)
    - Alpha is around 1 for English
    - 1.3 for German
  - **Actually**
    - **Mandelbrot proposed a generalisation that more closely fits the frequency distribution in languages by shifting the range by an amount**

$$f_w \approx \frac{k}{(r_w + \beta)^\alpha}$$

    - $\alpha \cong 1, \beta \cong 2.7$
  - Zipfian distributions occur in many collections
    - Almost all languages
      - **Even in extinct and yet-untranslated languages like Meroitic**
    - Sizes of settlements

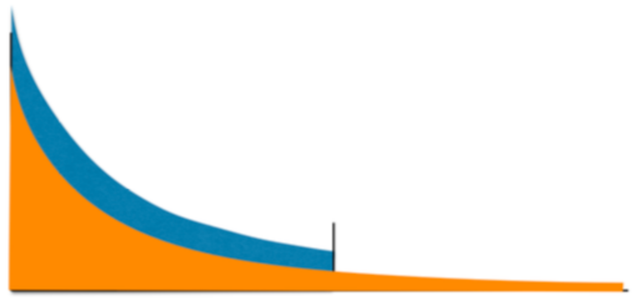
    - Frequency of access to web pages
    - Size of earthquakes
    - Word senses per word
    - Notes in musical performances
    - Machine instructions
    - **If a monkey randomly bangs on a typewriter, you also get this.**
- Very useful to plot in log space



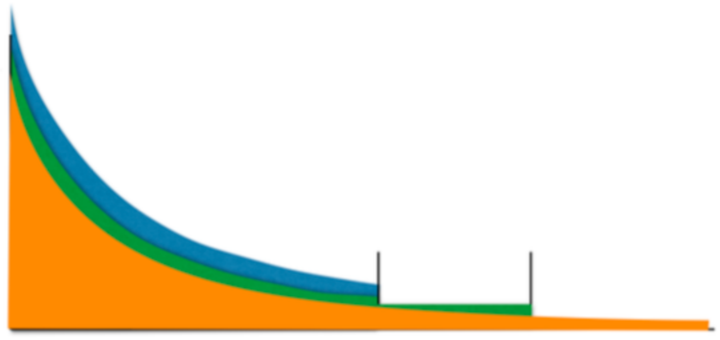
- Why does it happen?
    - No-one really knows!
        - One theory is that it can be described by communicative optimization principles
        - 'trade-off between speakers' and listeners' efforts'
        - Maybe it's just a 'normal-esque' distribution, given how common it is.
    - It is important to say that the apparent simplicity of the distribution of how the distribution is plotted.
    - The standard method of visualising the word frequency distribution is to count how often each word appears in a corpus and to sort the word frequency by decreasing magnitude.
    - The frequency of the rth most frequent word is then plotted against the frequency rank, typically yielding a mostly linear curve on a log-log plot.

        - However, the frequency and frequency rank are estimated on the same corpus, leading to correlated errors between the x-location and y-location
        - We can fix this by using two independent corpora of data to plot it.

- **Heap's Law**
    - Describes the relationship between the size of a vocabulary and the size of text that gave rise to it
    - $u_n = kn^{\beta}$
        - Where u_n is the number of types
        - N is the total number of tokens (text size)
        - Beta and k are constants (language-dependent)
            - Beta is around ½
            - $30 \leq k \leq 100$
    - Useful to plot in log-space
- **Effect of Zipf's Law and Heap's Law on Classifier**
    - With MLE, only seen types get a probability estimate:



    - Therefore, the estimate of the seen types is too high
    - With laplace smoothing, we redistribute the probability mass:



Statistical Tests for Classification Tasks

- Very important to be able to see which system is better and which is worse in an effective manner
    - Ensure that we do not draw conclusions and take actions based on evidence which is too weak.

- Sometimes apparent effects may be due to natural variation in the test data
- First define a Null Hypothesis – that two result sets come from the same distribution
  - System 1 Is equally as good as system 2
- Then we choose a significance level (α) at 1% or 5% for example (0.01 or 0.05)
  - When we accept at 1% we claim that is we run the baseline system 100 times we would only once get as good performance as our new system.
- We then try to reject the null hypothesis with confidence $1-\alpha$ $(0.99 \text{ or } 0.95 \text{ in these}$ cases)
- Rejecting the null hypothesis means that the observed result is very unlikely to have occurred by chance (proof beyond 'reasonable doubt').
  - We can say: "The difference between System 1 and System 2 is statistically significant at $\alpha = 0.01$"
  - Any other statements based on raw accuracy differences alone are strictly speaking meaningless.
- **Sign Test (non-parametric, paired)**
  - **The sign test uses a binary event model**
  - Here events correspond to documents
  - They have binary outcomes
    - Positive: System 1 beats System 2
    - Negative: System 2 beats System 1
    - Tie: They do equally well – this doesn't technically exist, we distribute the ties among the positive and negative.
  - Uses Binary Distribution:
    - Call the probability of a negative outcome q (here we say it is always 0.5)
      - If the null hypothesis is true, the baseline and the new system are actually equally good, but just happen to give slightly different results on particular tests.
      - Therefore, the observed counts of plus and minus give a binary distribution with a mean of 0.5n under the null hypothesis.
      - **Sign test checks how likely it is that the actual observations are a binomial distribution**
    - Probability of observing X=k negative events out of N

$$P_q(X = k|N) = \binom{N}{k} q^k (1-q)^{N-k}$$

    - At most k negative events:

$$P_q(X \leq k|N) = \sum_{i=0}^{k} \binom{N}{i} q^i (1-q)^{N-i}$$

  - If the probability of observing our events under the Null Hypothesis is very small (below α) then we can safely reject the Null hypothesis

- **Two tailed tests**
  - So far tested difference in a specific direction
    - Probability that at most 753/2000 binary events is negative
  - However, we generally want to test for statistically significant difference regardless of direction
  - This is given by $2P(X \leq k)$ because the distribution is symmetric
- When is it appropriate?
  - The sign test is appropriate when we have paired data, so it is a natural fit for a situation where we have tried a baseline system on some test data and want to compare a new system with the baseline.
  - We **make the assumption that the individual tests on data items are not linked**

- **Dealing with ties**
  - Split between the two
  - We go for the ceiling of the split as this makes it more unlikely that the test passes.
- **Things to learn**
  - 95% confidence = 1.96σ from the mean
  - We can check whether the observed value is more or less than two standard deviations from the mean.

Cross-Validation and Test sets

- In order to check whether a system works on all data, it is important to test the system on data that is a completely separate piece of test data.
- However, it is important to note that **overtraining** that can still happen even if we use separate test data.
  - When you think you are making improvements (performance on test data goes up) but in reality, you are making classifier worse because it generalises less well to data other than test data.
    - Indirectly also picked up accidental properties of the small test data
  - Until deployed to real unseen data, there is a danger that overtraining will go unnoticed
  - Also known as:
    - **Overfitting**
    - **Type III errors (rejecting the null hypothesis, but for the wrong reason)**
  - Ways to know if overtraining
    - Not overtraining if large amounts of test data, and use new and large enough test data each time you make an improvement
    - Not sure if incremental improvements to classifier and optimise system based on its performance on same small test data
    - You can inspect most characteristic features for each class and get suspicious when you find features unlikely to generalise
  - **Time Effects (Wayne Rooney Effect)**

    - Time changes public opinion on words / particular people
    - Therefore, important to test system on data from different time

- **Cross-Validation**
  - Why?
    1. Can't afford to get new test data every time
    2. Must never test on training set
    3. Want to use as much training material as possible
    4. Use every little bit of training data for testing
  - N-Fold Cross-Validation
    - Split data randomly into N folds
    - For each fold – use all other folds for training, test on that fold only
    - Final performance is the average of the performances for each fold.
  - Significance Test
    - Gained more usable test data and therefore are more likely to find a difference (pass the test)
  - **Stratified Cross-Validation**
    - Cross-Validation where each split is done in a way such that it mirrors the distribution of classes observed in the overall data
  - **Analysis**
    - Find average and **variance:**

$$var = \frac{1}{n}\sum_{i}^{n}(x_i - \mu)^2$$

- **How good is it?**
  - From testing – according to the textbook, the average of the cross-validation may not necessarily get the right true accuracy – with the system having a higher degree of flexibility.
  - In fact, importantly it gets an accuracy which is higher.
  - Therefore, we should only be interested in the minimum point of the error curve and trying to get that to apparently increase rather than necessarily getting the average to be higher.

Uncertainty and Human Agreement

- **Adding more classes**
  - So far only contained positive or negative reviews
  - Need to also consider neutral reviews
- **Finding true category**
  - We can use **human agreement** as a source of truth

        - Something is 'true' is sever humans agree on their judgement, independently from each other
        - The more they agree, the truer it is
        - We can find the observed agreement:

$$P(A) = \text{MEAN}\left(\frac{\text{observed rater–rater pairs in agreement}}{\text{possible rater–rater pairs}}\right)$$

- **P(A) observed agreement**
    - Pairwise observed agreement: average ratio of observed to possible rater-rater agreements
    - There are n_C2 = ½n(n-1) possible pairwise combinations between n judges
    - P(A) is the mean of the proportion of prediction pairs which are in agreement for all items (sum up the ratios for all items and divide by the number of items)
    - **However,** we want to see how much better this is than chance
        - **The probability of two independent judges choosing a class blindly – following the observed distribution of the classes is:**
        - $$P(E) = P(\text{both choose POSITIVE or both choose NEGATIVE}) = P(\text{POSITIVE})^2 + P(\text{NEGATIVE})^2$$
    - **Fleiss' Kappa measures reliability of agreement**
        - $$\kappa = \frac{P(A) - P(E)}{1 - P(E)}$$
        - Degree of agreement over that which would be expected by chance
        - P(A) (Observed Agreement): average ration of observed to possible pairwise agreements
        - P(E) (Chance Agreement): sum of squares of probabilities of each category
        - P(A) – P(E) gives the agreement above chance
        - 1 – P(E) gives the agreement that is attainable above chance
        - **Analysis**
            - If k = 1 then complete agreement
            - If k = 0 then no agreement beyond chance
            - If k < 0, then observed agreement less than expected by chance
            - 0.8 indicates very good agreement
            - **But, no universally accepted interpretation**
            - Size of k is affected by the number of categories and may be misleading with a small sample size.
            - **Weighted Kappa was created that allows you to weight different degrees of disagreement.**
            - Issues in two specific ways as well

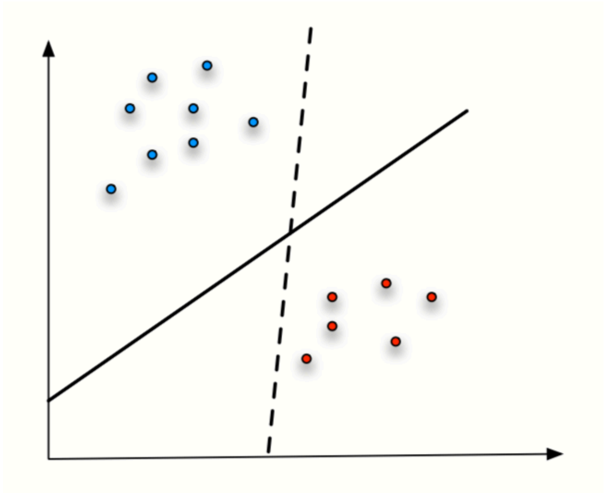
        - **1) Trait Prevalence**
            - Occurs when the characteristic being measured is not distributed uniformly across items
            - Since the P_e is vastly increased for times when there is more of one class than another.
            - The value of Fleiss' Kappa not only depends on actual agreement (and the accuracy of the system), but also the distribution of the **Gold Standard** scores: **these are the correct classes**
        - **2) Marginal Homogeneity**
            - Second problem is that the difference in the marginal probabilities (probabilities distribution of the results) affects the coefficient considerably
            - Comes from the assumption that marginal probabilities are classification propensities that are fixed.
            - Kappa could lead to learning functions that favour assigning distributions that are different to that of the gold standard.
            - **Tends to affect weighted kappa less**

- **Other Measurement Metrics**
    - Pearson's Product Moment Correlation
        - Parametric measure of association that quantifies the degree of linear dependence between two variables and describes the extent to which the variables co-vary relative to the degree to which they vary independently.
        - The greater the association, the more accurately one can use the value of one to predict the other.
        - However, outliers can destroy the coefficient.
    - Spearman's Rank Correlation Coefficient
        - Non-parametric measure of coefficient
        - Calculating by ranking the variables and computing Pearson on the ranks rather than the raw values
        - Unlike Pearson, has robustness to outliers
        - But, not really an appropriate measurement for his task, where we would like actual agreement with respect to the scores
            - Also, do not account for any systematic biases in the data.
            - High correlation can be observed in even if the predicted scores are 0.01 higher than the gold standard.
    - Scott's Pi Metric
        - Sensitive to trait prevalence but not to marginal
    - Agreement Coefficient AC and Brennan Prediger Coefficient

        - Both estimate P_e more conservatively using more plausible assumptions.
    - **Desirable Features for a Metric**
        - **1) Robustness to trait prevalence**
        - **2) Robustness to marginal homogeneity**
        - **3) Sensitivity to magnitude of misclassification**
        - **4) Robustness to score range**
    - According to the paper, the best is the AC coefficient with quadratic weights - with Cohen's Kappa generally being bad with 1, 2 and 4.
        - However, it is independent on the type of data that it is being used on, that is, whether there is a categorical or ordinal (gold standard) scale.

Introduction to other classifiers

- Naïve Bayes is a probabilistic classifier – we find the probability of features and classes based on data and therefore pick the item with the highest probability.
- **Support Vector Machine (SVM)** is a non-probabilistic binary linear classifier
    - SVMs assign new examples to one category or the other
    - SVMs can reduce the amount of labelled data required to gain good accuracy
    - A linear SVM can be considered to be a base-line for non-probabilistic approaches
    - SVMs can be efficiently adapted to perform non-linear classification
    - SVMs find hyper-planes that separate classes



    - It finds the maximum-margin hyperplane in noisy data such that distance from it to the nearest data point from each class is maximised
    - **Efficient and Effective**
        - When learning from large number of features
        - When learning from small amounts of labelled data
        - We can choose how many points to involve (size of margin) when choosing the plane (tuning vs overfitting)
        - We can separate non-linear boundaries by increasing the feature space (using a kernel functions)
- **Decision Tree**

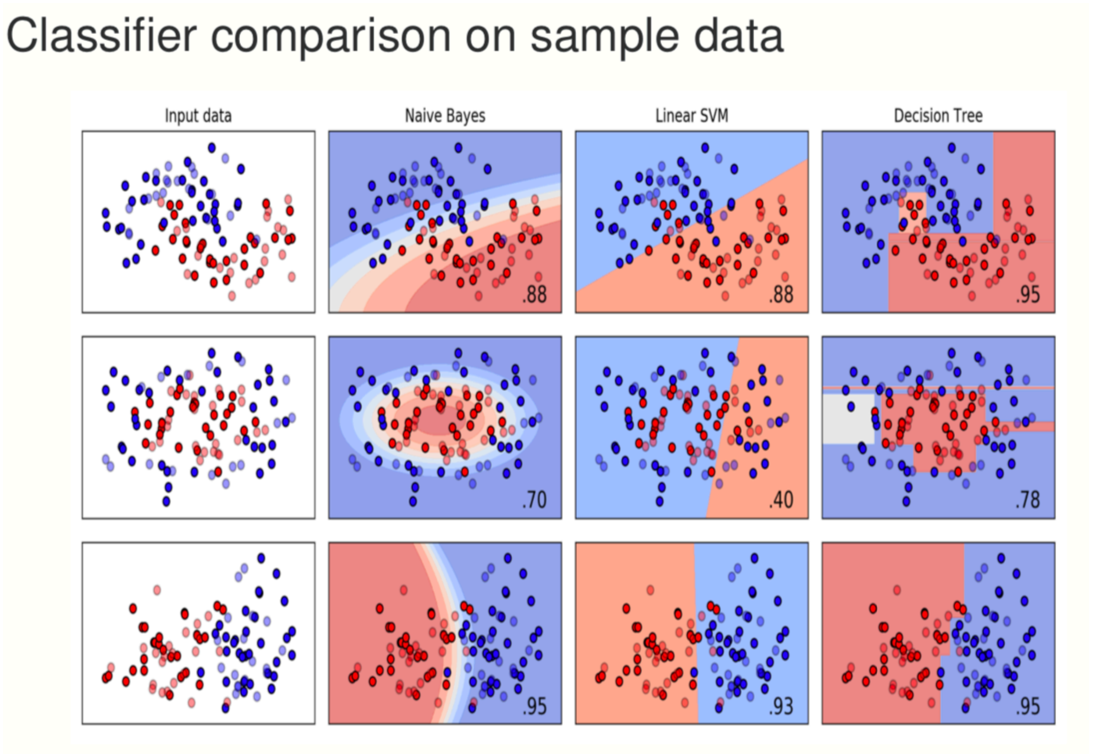
- A decision tree can be used to visually represent classifications
- It is very simple to interpret, and you can mix numerical and categorical data.
- You can specify the parameters of the tree (maximum depth, number of items and lead nodes)
- However, finding the optimal decision tree can be np-complete
- **Information Gain**
    - It is defined in terms of entropy H

$$H(n) = -\sum_{p} p_i \log_2 p_i$$

        - Where p's are the fraction of each class at node n
    - Used to decided which feature to split on at each step in building the tree

$$I(n, D) = H(n) - H(n|D)$$

        - Defines the information gain where H(n|D) is the weighted entropy of the child nodes.



Sequence Analysis

Hidden Markov Models and HMM training

- Idea is that we can use a history of observations to predict the next one.
- **Markov Assumption (first order)**
    - **The next state is ONLY based on the previous state – Limited Horizon**

$$P(w_t \mid w_{t-1}, w_{t-2}, \ldots, w_1) \approx P(w_t \mid w_{t-1})$$

- Using data, we can come up with a transition probability matrix, such as this one:

$$\begin{array}{llcc} & & \multicolumn{2}{c}{\text{Tomorrow}} \\ & & Rainy & Cloudy \\ \text{Today} & Rainy & 0.7 & 0.3 \\ & Cloudy & 0.3 & 0.7 \end{array}$$

- **Markov Chain is a stochastic process that embodies the Markov Assumption**
  - Can be viewed as a probabilistic finite-state automaton.
  - The states are fully observable, finite and discrete; the transitions are labelled with transition probabilities.
  - It models sequential problems – your current situation depends on what happened in the past.
  - **It is useful for modelling the probability of a sequence of events that can be unambiguously observed**
- **Hidden Markov Model**
  - **Nothing more than a probabilistic function of a Markov process**
  - **Assumptions**
    - Limited Horizon – Markov Assumption
    - Time Invariant
      - Probability of something happening at one time is equal to it happening at any time.
    - Output independence
      - Probability of an output observation depends only on the state that produced the observation and not on any other states or any other observations
  - There are hidden states, which we can't see. There is also a set of observed states, which we can.
  - But the underlying Markov Chain is over the hidden states.
  - We have no 1-1 mapping between observations and hidden states.
  - We now have to infer the sequence of hidden states that correspond to a sequence of observations.
  - **We tend to create a fast state and a final state (with no observations)**
    - It can in many ways be thought of as a non-deterministic finite state automaton with probabilities attached to each arc.
    - The Markov properties ensure that we have a finite state automaton.
    - There are no long-distance dependencies, and where one ends up next depends simply on what state one is in.
  - **Definitions:**

$$\begin{array}{rl} S_e = \{s_1, \ldots, s_N\} & \text{a set of } N \text{ emitting hidden states,} \\ s_0 & \text{a special start state,} \\ s_f & \text{a special end state.} \end{array}$$

$K = \{k_1, \dots k_M\}$	an output alphabet of M observations ("vocabulary").
k_0	a special start symbol,
k_f	a special end symbol.
$O = O_1 \dots O_T$	a sequence of T observations, each one drawn from K.
$X = X_1 \dots X_T$	a sequence of T states, each one drawn from S_e.

- **Assumptions**
  - MARKOV ASSUMPTION: Transitions depend only on the current state
  - OUTPUT INDEPENDENCE: Probability of an output observation depends only on the current state and not on any other states or any other observations
- **Implementation**
  - Create a state transition probability matrix of size (N+2) x (N+2) where N is the number of states
    - A_{ij} is the probability of moving from state s_i to state s_j
  - Create an emission probability matrix of size (M + 2) x (N + 2) where M is the number of observations
    - $B_i(k_j)$ is the probability of emitting vocabulary item k_j from state s_i
  - Our hidden Markov Model is defined by its parameters μ = (State Transition Probability Matrix, Emission Probability Matrix)
  - **Problem 1 is Labelled Learning – getting two matrices**
  - **Problem 2 is Unlabelled Learning – learning using only the observation sequence and the set of emitting states**
  - **Problem 3 is Likelihood – given a HMM and an observation sequence, find out its likelihood**
  - **Problem 4 is Decoding – given an observation sequence and a HMM, find the most likely hidden state sequence**
- **Applications**
  - Speech Recognition
    - Observations: audio signal
    - States: phonemes
  - Part-of-speech tagging (Noun, Verb, etc)
    - Observations: words
    - States: part-of-speech tags
- **Why us a HMM?**
  - Very useful when one can think of underlying events probabilistically.
  - One of a class of models for which there are efficient methods of training through using **Expectation Maximization algorithm**

            - Algorithm lets us automatically learn the model parameters that best account for the observed data.
        - We can use HMMs to generate parameters for linear interpolation of n-gram models.
            - Use the hidden states being whether to use unigram, bigram, trigram, etc probabilities
            - The HMM will determine the optimal weight to give to the arcs entering each of these hidden states, which represents the amount of the probability mass which should be determined by each n-gon model, by setting their parameters.
    - **Arc-Emission HMM vs State Emission HMM**
        - **Arc-Emission HMM:** The symbol emitted at time t depends on both the state at time t and at time t+1
        - **State Emission HMM:** Symbol emitted at time t depends only on the state at time t.
    - **Fully Connected HMM (Ergodic)**
        - HMM where there is a non-zero probability of transitioning between any two states.
    - **Bakis HMM (left-to-right)**
        - HMM where there are no transitions between a higher numbered state to a lower numbered state – only transitions going upwards.
    - **Fundamental Questions for HMMs**
        - **1) Likelihood:** Given a model μ = (A, B), how do we efficiently compute how likely a certain observation is, that is P(O|μ) (**Forward Algorithm)**
        - **2) Decoding:** Given the observation sequence O, and a model μ, how do we choose a state sequence that best explains the observations **(Viterbi)**
        - **3) Learning:** Given the observation sequence O, and a space of possible models found by varying the model parameters, how do we find the model that best explains the observed data?

The Viterbi Algorithm

- **Decoding:** finding the most likely hidden sequence X that explains the observation O given the HMM parameters μ

$$\begin{aligned}\hat{X} &= \underset{X}{\operatorname{argmax}}\, P(X, O|\mu)\\ &= \underset{X}{\operatorname{argmax}}\, P(O|X,\mu)P(X|\mu)\\ &= \underset{X_1 \dots X_T}{\operatorname{argmax}} \prod_{t=1}^{T} P(O_t|X_t)P(X_t|X_{t-1})\end{aligned}$$

    - **Viterbi Algorithm**

- The Viterbi algorithm helps us to accomplish this in an efficient manner, using Dynamic Programming
- We can use Dynamic Programming if
    - There is an optimal substructure property
        - An optimal state sequence X1, … Xj, … Xt contains within in the sequence X1 … Xj which is also optimal
    - Overlapping Subsolutions Property
        - If both Xt and Xu are on the optimal path, with u > t, then the calculation of the probability for being in Xt is part of each of the many calculations for being in state Xu
- Do the calculation of the probability of reaching each stage once for each time step
- Then memoise this probability in a Dynamic Programming table – **trellis**
    - (N + 2) x (T + 2) with states j as rows and time steps as columns
    - Each cell records the Viterbi Probability of the most likely path that ends at that time (being in that state, having made the correct observations):

$$\delta_j(t) = \max_{1 \leq i \leq N} [\delta_i(t-1)\, a_{ij}\, b_j(O_t)]$$

- This probability is calculated by maximising over the best ways of going to s_j for each s_i.
- a_{ij}: the transition probability from s_i to s_j
- $b_j(O_t)$: the probability of emitting O_t from destination state s_j

δ(t-1) – The **previous Viterbi Path Probability** from the previous time step

A_{ij} – **Transition probability**

$B_j(o_t)$ – **State observation likelihood**

- You should set the probability of starting the sequence at t=0 as 1 for start state and 0 for everything else.
- It is also useful to have a table that will help you to back trace and find the actual most likely path, which should simply tell you which state came last.
- This reduces our effort to $O(N^2T)$

```
function VITERBI(observations of len T, state-graph of len N) returns best-path

  create a path probability matrix viterbi[N+2,T]
  for each state s from 1 to N do                          ; initialization step
      $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
      $backpointer[s,1] \leftarrow 0$
  for each time step t from 2 to T do                      ; recursion step
    for each state s from 1 to N do
      $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
      $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
  $viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$          ; termination step
  $backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$     ; termination step
  return the backtrace path by following backpointers to states back in
         time from $backpointer[q_F,T]$
```

- We can also give a formal definition of the Viterbi recursion as follows:

1. **Initialization:**

$$\begin{aligned} v_1(j) &= a_{0j}b_j(o_1) \quad 1 \leq j \leq N \\ bt_1(j) &= 0 \end{aligned}$$

2. **Recursion** (recall that states 0 and q_F are non-emitting):

$$\begin{aligned} v_t(j) &= \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T \\ bt_t(j) &= \operatorname{argmax}_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T \end{aligned}$$

3. **Termination:**

$$\text{The best score:} \quad P* = v_T(q_F) = \max_{i=1}^{N} v_T(i) * a_{iF}$$

$$\text{The start of backtrace:} \quad q_T* = bt_T(q_F) = \operatorname{argmax}_{i=1}^{N} v_T(i) * a_{iF}$$

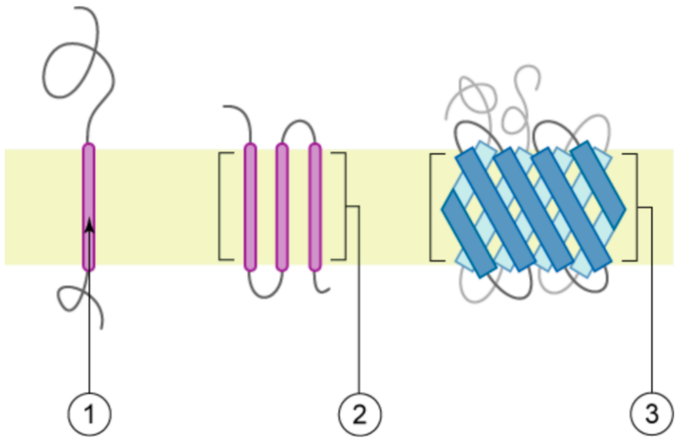
- **Precision and Recall**
  - The human labels we define as correct are called **gold labels**
  - **So, therefore, we have true positives, false positive, false negatives and false positives**

- We have accuracy – ratio of observations the system labelled correctly.
- This doesn't work very well when the test data is not very well distributed
    - When we have 900 and 100 of another class, if we simply randomly predict everything is in the first class, then we get an accuracy of 0.9, even though the system is clearly terrible
- So, we define precision and recall
    - $Precision = \frac{true\ positives}{true\ positives + false\ positives}$
    - $Recall = \frac{true\ positives}{true\ positives + false\ negatives}$
- We combine the two metrics, precision and recall into a single metric called **F-measure**
    - $\frac{(\beta^2+1)PR}{\beta^2 P+R}$
    - We set β depending on what we want to weight more heavily
    - B > 1 favours recall
    - B = 1 precision and recall are equally balanced
    - B < 1 favours precision
    - We generally often use F_1 measure
        - $F_1 = \frac{2PR}{P+R}$
    - F-measure comes from a weighted harmonic mean of precision and recall.
        - The harmonic mean of a set of numbers is the reciprocal of the arithmetic mean of reciprocals.
        - It is used because it is a conservative metric – the harmonic mean of two values is closer to the minimum of the two values than the arithmetic mean is.

Using an HMM in a biological application

- Cell membranes are lipid bilayers: inside the membrane is hydrophobic, the two sides are hydrophilic
    - They have let things in an out of the cell
    - Proteins which are part of the cell membrane allow this
- In cell metabolism: proteins make sure the right thing happens in the right place at the right time. The proteins are made of amino acid sequences – with each amino acid having very different side chains.
- The sequences fold into very complex 3D structures.
- **Transmembrane Proteins**
    - These are proteins which go through the membrane one or more times
    - The regions of the protein which lie inside and outside the cell tend to have more hydrophilic amino acids.
    - The regions inside the membranes tend to have more hydrophobic amino acids
    - Many of them proteins involve alpha-helixes in the membrane

- The channel formed by the proteins allows ions to pass through in a controlled way



1. a single transmembrane α-helix (bitopic membrane protein)
2. a polytopic transmembrane α-helical protein 3. a polytopic transmembrane β-sheet protein

- We can image and get an accurate structure using x-ray crystallography, but it is difficult and time-consuming, and we don't know the membrane location

- **HMM**
  - **Hidden Sequence is inside the membrane or inside or outside the cell**
  - **Observed sequence is amino acid**
  - This is much quicker and easier than x-ray crystallography
  - It helps us distinguish the interior of the membrane from the inside / outside of the cell

Monte Carlo Methods

- **Protein Folding**
  - **Anfinsen's Hypothesis:** The structure a protein forms in nature is the global minimum of the free energy and is determined by the amino acid sequence
  - **Levinthals's Paradox:** Protein folding takes milliseconds – not enough time to explore the space and find the global minimum. Therefore, a kinetic function must be important.
  - **Finding Protein Structure:** The Primary structure may be determined directly or from DNA sequencing
    - **The Secondary and Tertiary structure** can be determined by x-ray crystallography and other direct methods, but difficult, time-consuming
    - We want to be able to predict the structure given the amino acid sequence
    - Secondary structure prediction is relatively tractable, various prediction methods, including HMMs
    - But, tertiary structure prediction is very difficult
- **Protein Tertiary Structure Prediction**

- Modelling fully is hugely computationally intensive
- Methods:
  - Molecular Dynamics (MD): modelling chemistry: use home computers to run simulations.
  - Foldit: Get lots of humans to work on the problem (an example of gamification)
  - **Use Monte Carlo Methods (repeated random sampling to explore possibilities)**

- **Monte Carlo methods in protein structure prediction**
  - Objective: find lowest energy state of protein
  - Idea
    - Start with secondary structure
    - Try pseudo(random) move – see if the result is lower energy and repeat
  - Problem
    - **Local Minima – locally good move might not be part of best solution**
      - So also sometimes accept a move that increases energy
  - Specific Approach
    - **Metropolis-Hastings – a type of Markov Chain Monte Carlo method**
- **Monte Carlo methods in general**
  - Using random sampling to solve intractable numerical problems
  - Buffon's needle for estimating pi
  - Physicists developed modern Monte Carlo methods at Los Alamos: programmed into ENIAC by von Neumann
  - Bayesian statistical inference did not come until 1993 – this is essential for many modern machine learning approaches

Social Networks

Properties of Networks

- What are we looking for?
  - What role does a node (entity / person) play in this network?
  - Is there a substructure?
    - Neighbourhood areas / cliques
- Uses
  - Academic investigation of human behaviour (sociology, economics)
  - Disease transmission in epidemics
  - Modelling information flow – who sees a picture, reads a piece of news (or fake news)
  - Identifying links between sub-communities
    - Erdös Number

            - Steps in a path between Paul Erdos and a researcher, counting co-authorship of papers as links
      - Finding countries with powerful positions in an international trade graph
      - Split online journalism into clusters
         - Turns out it splits well into liberal and conservative
         - Then can pick the most used pages
      - **Networks in the natural world**
         - Food webs
            - Allows us to consider cascades of removing nodes with cascading extinctions
         - Brain structure
         - Metabolism networks
- **Degree:** Number of neighbours a node has in a graph
   - The distribution of node degrees may be very skewed
- **Distance**
   - Length of shortest path between two nodes
- **Cycle**
   - Informally a ring structure
   - Formally, a path with at least three edges in which the first and last nodes are the same, but otherwise all nodes are distinct.
- **Diameter**
   - Maximum distance between any pair of nodes
   - **Small World Phenomenon: Six Degrees of Separation**
      - Chain Links was a short story by Karinthy in 1929 (book says it was a play by John Guare), which claimed that any two individuals in the world could be connected via at most 5 personal acquaintances.
      - Hard to prove, but has been shown in settings where we do have full data available on the network structure – Erdos Number, Bacon Number (very hard to find people with numbers above 6!)
         - However, have been some experiments on a small scale which have showed this idea working.
      - All comes down to very small distances to the Giant Component
- Natural networks tend to have closely clustered regions connected only by a few links between them – these are often week ties.
- **Giant Component:** A connected component containing a significant proportion of the nodes in a graph
   - This is a deliberately vague term
   - ‘When a network contains a giant component, it almost always contains only one’
      - Unlikely to have two coexisting giant components
      - At some point, they would get joined.
- **Weak and Strong Ties:** The closeness of a link – how many connections

  - Strong ties tend to be embedded in tightly-linked regions of the network
  - Weak ties, representing more casual connections, tend to cross these regions
    - More crucial to the structure
  - We can think about social networks in terms of the strongly tied networks and the weak connections that connect them.
- **Structural Balance:** How local breaks in connections can affect a network
- **Structural Holes:** Empty space in the network between two sets of nodes that do not otherwise interact closely.
  - Purposefully vague definition
- **Connectivity**: Graph is connected if, for every pair of nodes, there is a path between them
  - If a graph is not connected, then it breaks apart naturally into a set of pieces – components.
  - **Connected Component:** Subset of nodes such that every node in the subset has a path to each other and the subset is not part of some larger set which is connected.
- **Bridge:** Edge connecting two components which would otherwise be unconnected (removing the edge would make one component into two)
  - Bridges are generally very rare in real social networks
- **Local Bridge:** An edge joining two nodes that have no other neighbours in common. Cutting a local bridge increases the length of the shortest path between the nodes.
  - **Span of a local bridge:** The distance the two nodes would be if we remove the node.
    - No local bridge of span 2
- **Triadic Closure:** Triangle of nodes.
  - **Thought of as a dynamic property: if A knows B and A knows C, relatively likely B and C will get to know each other**
  - A, B and C are a triangle in the network
  - **Reasons for Triadic Closure**
    - In human systems
      - Large opportunity to meet them
      - And there is a basis for trust
      - "incentive for the common node to bring them together as it will be a source of latent stress'
- **Strong Triadic Closure Property**
  - Triadic closure is especially likely to occur if the two connections are strong ties.
  - "Node A violates the Strong Triadic Closure property if it has strong or weak ties to two other nodes B and C, and there is no edge at all (strong or weak) between B and C"
  - If a node A in a network satisfies the Strong Triadic Closure property and is involved in at least two strong ties, then any local bridge it is involved in must be a weak tie.
- **Clustering Coefficient:** A measure of the amount of triadic closure in a network

- Probability that two randomly selected neighbours of a node are also neighbours of each other.
- More strongly the triadic closure process operates, the higher the coefficient
- **Relevance in human networks**
  - Very high correlation between having a low clustering coefficient and high rates of suicide in a group

- **Braess's Paradox**
  - Idea that interactions among people's behaviour can lead to counterintuitive effects
  - Adding resources to a transportation network can create incentives which undermine efficiency
- **Embeddedness:** The number of common neighbours shared between two endpoints
- **Neighbourhood Overlap**
  - Defined for an edge from A to B
  - $\frac{\textit{number of nodes who are neighbours of both A and B}}{\textit{number of neighbours who are neighbours of at least one of A and B}}$
- **Random Networks**
  - Is the small world phenomenon surprising?
    - Exponential number of connections means we *should* reach other people very quickly
    - Since we have triadic closure, this limits the number of paths you have to take to get to someone nearby
    - It is clear that long weak ties are crucial
  - Watt and Strogatz: Randomly generated graph with triangles at close range plus a few long random links
    - **Mathematically Defined**
    - Describes a graph $G_{n, k, p}$ = (V, E) where n is |V|, k is the initial degree of a node (an even integer) and p is a rewiring probability
    - It starts with a ring of nodes, with each node connected to its k nearest neighbours with undirected edges – **it is a ring lattice**
    - Each of the starting edges (u, v) is visited in turn (u < v, starting with a circuit of the edges to the nearest neighbours, then to the nest nearest and so on) and replaced with a new edge (u, v') with probability p, with v' chosen randomly.
    - If p is 1, every edge is rewired randomly and is p is 0, there is no rewiring.
    - Watt and Strogatz also say that n >> k >> ln(n) >> 1 where k >> ln(n) guarantees that the graph will be connected.
  - Erdós-Renyi Model
    - Random undirected graph generation where $G_{n, p}$ = (V, E) where n is the number of vertices |V| and the probability of forming an edge (u, v) is given by p

        - A variance is to generate all the possible graphs with the given number of vertices and edges and select between them with uniform probability, but this is not much used.
- **Implementation of Task**
    - **How to find diameter of the network**
        - BFS from any one point and find the node it finds last
        - Then BFS from that node and find the node it finds last (and the distance) – that is the diameter
        - **Alternatively**
            - Breadth-first all-pairs shortest path – O ($V^3 + EV^2$)

Betweeness Centrality

- **Aim:** Find gatekeeper nodes via the betweeness centrality
    - Certain nodes / edges are most crucial in lining densely connected regions of the graph
    - Cutting these edges isolates the cliques / clusters
- **Local Bridge:** An edge which increased the shortest paths if cut
- **Nodes which are intuitively the gatekeepers can be determined by the betweeness centrality.**
- **Betweeness Centrality:**
    - Proportion of shortest paths between all pairs of nodes that go through the node
    - **Naïve Approach**
        - For every node, use a BFS to find all the shortest path between s and all the nodes. Store all these paths for each pair s, t
        - For each vertex, count the number of shortest paths which go through the node.
            - Given a directed graph G = <V, E>
            - σ (s, t) is the number of shortest paths between nodes s and t
            - σ (s, t | v) is the number of shortest paths that pass-through v
            - c_b (v) is the betweeness centrality of v
                - $c_b(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$
                - If s = t, then σ(s, t) = 1
                - If v ϵ s, t then σ(s, t | v) = 0
            - σ(s, t) can be calculated recursively

$$\sigma(s,t) = \sum_{u \in Pred(t)} \sigma(s,u)$$

$Pred(t) = \{u : (u,t) \in E, d(s,t) = d(s,u) + 1\}$
predecessors of t on shortest path from s
$d(s,u)$: Distance between nodes s and u

Therefore, we can run a Breadth First Search from each node as a source once, for a total complexity of O(V^2 + EV)

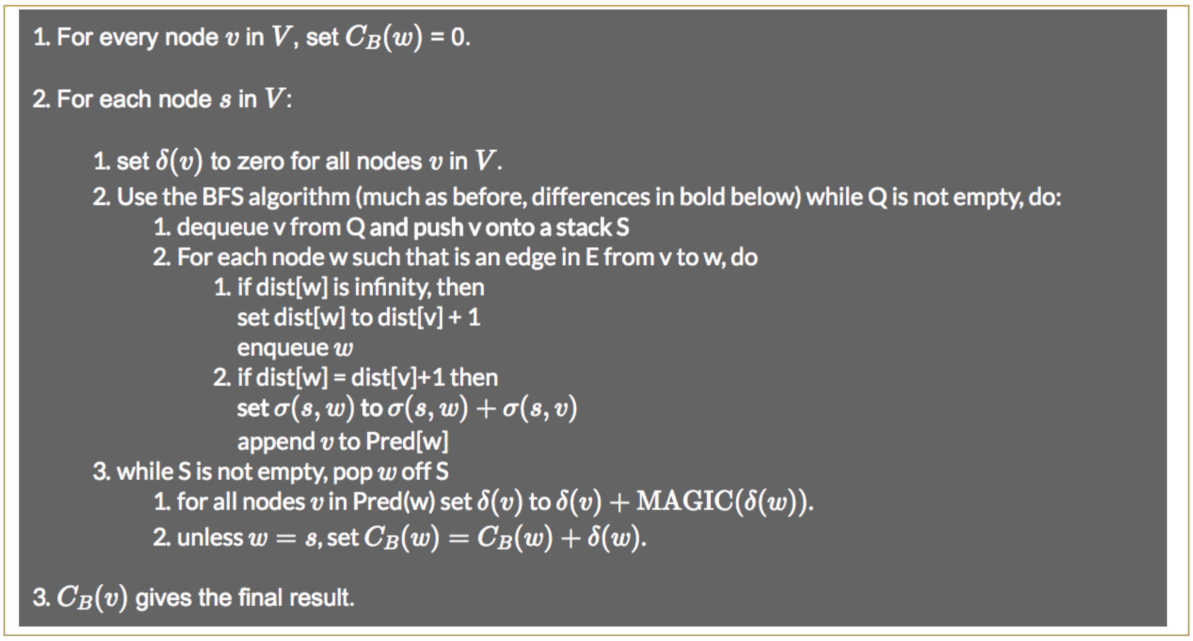
- The naïve algorithm uses a lot of space
    - We shouldn't really need to save anything about the paths between s and t once we've updated $C_B(v)$ for each vertex v on those paths
    - So, we could (Storage Efficient Algorithm)

```
1. For every node v in V, set C_b(w) = 0
2. For each node s in V
     ▪ Use a BFS algorithm to find all the shortest paths
       between s and all other nodes. Store the paths for
       each target t
     ▪ For each t, for each vertex that occurs on one of the
       stored paths, count the number of times w appears
       in total, and divide by the total number of paths
       between s and t.
     ▪ Add this result to C_b(w)
3. C_b(w) gives the final result
```

- Integration with Shortest Paths Algorithm
    - Easier to add things going backwards, rather than checking if every node is on the path
    - For every node v, set $C_b(v) = 0$
    - For every node s in V
        - Set S(v, t) to zero for all nodes v and t in V
        - Use the BFS algorithm, as above, to reach target t from
        - In the backward phase, increment S(u, t) as appropriate when each node v is reached, rather than creating full paths
        - At the end, divide each S(u, t) by the total number of paths between s and t
        - Add the result to $C_b(v)$
    - $C_b(w)$ gives the final result

- Recursive method
    - The main difference between what we have above and Brandes' efficient algorithm is that the latter make use of a recursive step on the backwards phase to allow direct calculations of the ration for each v on the basis of its successor nodes on the shortest paths to every t.
    - For now, we assume we have such a function, and a value δ(v) for which the following hold

      - δ(t) = 0 if t is a terminal node (no nodes below them on shortest paths)
      - We can increment δ(v) via this function every time we reach v from a node w on the backwards phase
      - After we have finished with all the w values, δ(v) can be straightforwardly be accumulated into Cb(v)



```
1. For every node $v$ in $V$, set $C_B(w)$ = 0.

2. For each node $s$ in $V$:

    1. set $\delta(v)$ to zero for all nodes $v$ in $V$.
    2. Use the BFS algorithm (much as before, differences in bold below) while Q is not empty, do:
        1. dequeue v from Q and push v onto a stack S
        2. For each node w such that is an edge in E from v to w, do
            1. if dist[w] is infinity, then
               set dist[w] to dist[v] + 1
               enqueue $w$
            2. if dist[w] = dist[v]+1 then
               set $\sigma(s,w)$ to $\sigma(s,w) + \sigma(s,v)$
               append $v$ to Pred[w]
    3. while S is not empty, pop $w$ off S
        1. for all nodes $v$ in Pred(w) set $\delta(v)$ to $\delta(v) + \mathrm{MAGIC}(\delta(w))$.
        2. unless $w = s$, set $C_B(w) = C_B(w) + \delta(w)$.

3. $C_B(v)$ gives the final result.
```

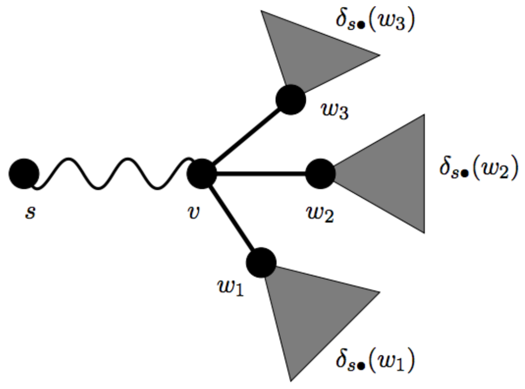
    - It turns out this function requires us to know the number of shortest paths between s and each node v so we create these values in the forward phase of the BFS as shown.
    - We also need to make sure that the nodes are visited in the correct order on the backward step, which we do with the stack
  - Brandes' Algorithm
    - Ratio between the shortest paths between s and t that go through v and the total number of shortest paths between s and t is:
      - $$\delta(s,t|v) = \frac{\sigma(s,t|v)}{\sigma(s,t)}$$
    - Therefore:
      - $$C_B(v) = \sum_{s,t \in V} \delta(s,t|v)$$
    - If we define **one-sided dependencies as:**

$$\delta(s|v) = \sum_{t \in V} \delta(s,t|v)$$

- And:

$$C_B(v) = \sum_{s \in V} \delta(s|v)$$

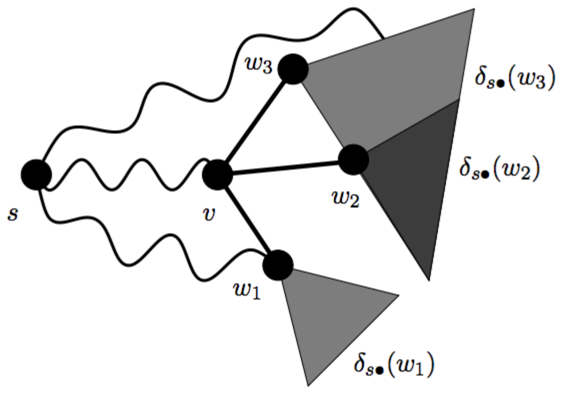
- **Assuming only one shortest path (tree condition)**
  - If the vertices and edges of all the shortest paths form a tree, it is clear that δ(v) can be computed simply



  - In a bit more detail
    - For any vertex v and any target t, either v lies on the unique shortest path between s and t, in which case δ(s, t|v) = 1, or it doesn't, δ(s, t|v) = 0
    - For any vertex w, such that v immediately precedes on shortest paths from s, v will lie on the shortest path from s to w
    - So, for any node x, st w immediately precede x on shortest paths, v will lie on the shortest paths.
    - Therefore, we can total all the one-sided dependencies:

$$\delta(v) = \sum_{w}(1 + \delta(w))$$

- **Multiple shortest paths**
  - We need to find a way to determine the ration of shortest paths which will go through v



- Then,

$$\delta_{s\bullet}(v) = \sum_{t\in V} \delta_{st}(v) = \sum_{t\in V} \sum_{w\,:\,v\in P_s(w)} \delta_{st}(v, \{v, w\}) = \sum_{w\,:\,v\in P_s(w)} \sum_{t\in V} \delta_{st}(v, \{v, w\}).$$

-
- Let w be any vertex with $v \in P_s(w)$. Of the σ_{sw} shortest paths from s to w, many first go from s to v and then use {v, w}
- Therefore,
  - $\frac{\sigma_{sv}}{\sigma_{sw}} \cdot \sigma_{st}(w)$
  - Shortest paths from s to some t contain v and {v, w}.
  - It follows that the pair dependency of s and t on v and {v, w} is:

$$\delta_{st}(v, \{v, w\}) = \begin{cases} \frac{\sigma_{sv}}{\sigma_{sw}} & \text{if } t = w \\ \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \frac{\sigma_{st}(w)}{\sigma_{st}} & \text{if } t \neq w \end{cases}$$

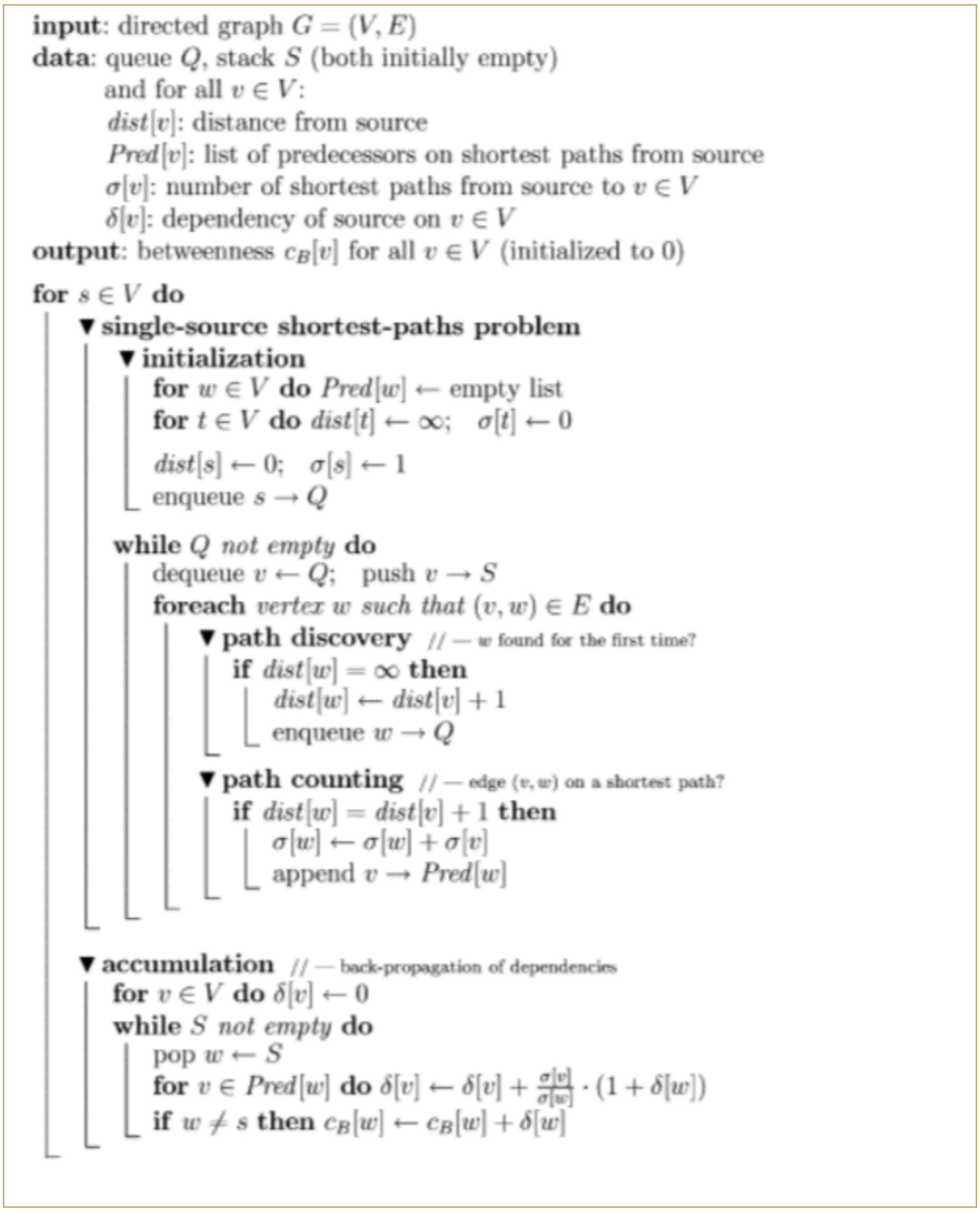
- Putting this into the above equation gives:

$$\begin{aligned} \sum_{w\,:\,v\in P_s(w)} \sum_{t\in V} \delta_{st}(v, \{v, w\}) &= \sum_{w\,:\,v\in P_s(w)} \left(\frac{\sigma_{sv}}{\sigma_{sw}} + \sum_{t\in V\setminus\{w\}} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \frac{\sigma_{st}(w)}{\sigma_{st}} \right) \\ &= \sum_{w\,:\,v\in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \left(1 + \delta_{s\bullet}(w)\right). \end{aligned}$$

- **Intuition**
  - Several cases to consider depending on whether the bypassing edge arrives at one of the nodes w or beyond it
  - The intuition is that only the situation where the bypassing edge arrives at one of the w that we have to worry about.
  - Otherwise, δs of the ws already incorporate the ratios of the shortest paths accordingly.
  - Therefore, only need to look at the ratio of paths going between s and v compared to those going between s and w to capture the extent to which there are bypassing paths going to w.

- **Brandes' Algorithm**
  - *Intended for directed graph, but works the same for undirected graph, except that each pair is considered twice, once in each direction*
  - *Therefore, halve the scores at the end for undirected graphs.*
    - Iterate over all vertices s in V
    - Calculate δ(s | v) for all V

- Breadth-first search, calculating distances and shortest path counts from s, push all vertices onto stack as they're visited.
- Visit all vertices in reverse order (pop off stack) aggregating dependencies according to the equation.

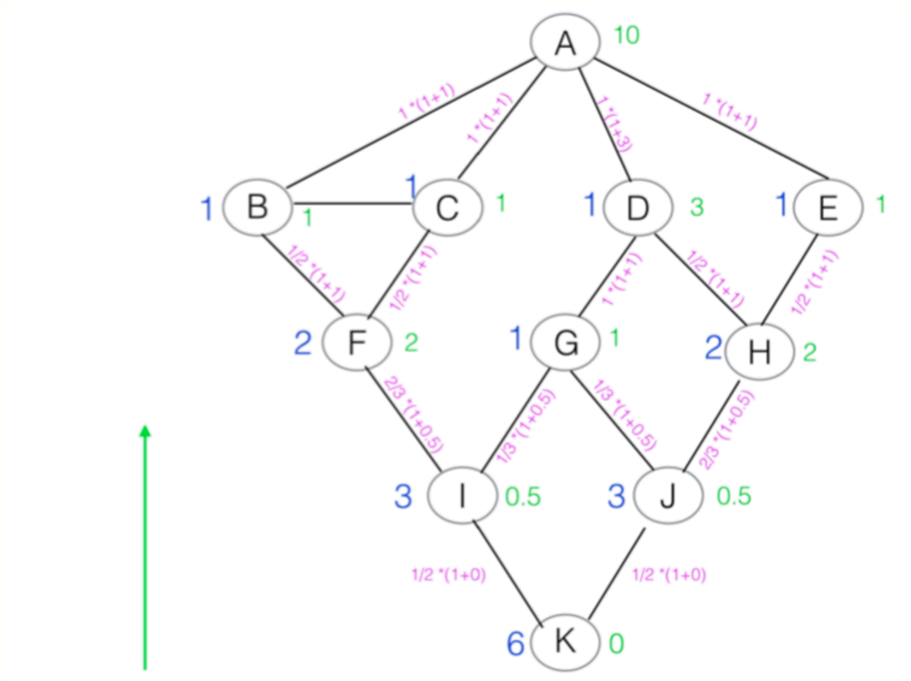


```
input: directed graph $G = (V, E)$
data: queue $Q$, stack $S$ (both initially empty)
      and for all $v \in V$:
      $dist[v]$: distance from source
      $Pred[v]$: list of predecessors on shortest paths from source
      $\sigma[v]$: number of shortest paths from source to $v \in V$
      $\delta[v]$: dependency of source on $v \in V$
output: betweenness $c_B[v]$ for all $v \in V$ (initialized to 0)

for $s \in V$ do
    ▼ single-source shortest-paths problem
        ▼ initialization
            for $w \in V$ do $Pred[w] \leftarrow$ empty list
            for $t \in V$ do $dist[t] \leftarrow \infty$;   $\sigma[t] \leftarrow 0$
            $dist[s] \leftarrow 0$;   $\sigma[s] \leftarrow 1$
            enqueue $s \rightarrow Q$
        while $Q$ not empty do
            dequeue $v \leftarrow Q$;   push $v \rightarrow S$
            foreach vertex $w$ such that $(v, w) \in E$ do
                ▼ path discovery // — $w$ found for the first time?
                    if $dist[w] = \infty$ then
                        $dist[w] \leftarrow dist[v] + 1$
                        enqueue $w \rightarrow Q$
                ▼ path counting // — edge $(v, w)$ on a shortest path?
                    if $dist[w] = dist[v] + 1$ then
                        $\sigma[w] \leftarrow \sigma[w] + \sigma[v]$
                        append $v \rightarrow Pred[w]$
    ▼ accumulation // — back-propagation of dependencies
        for $v \in V$ do $\delta[v] \leftarrow 0$
        while $S$ not empty do
            pop $w \leftarrow S$
            for $v \in Pred[w]$ do $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w])$
            if $w \neq s$ then $c_B[w] \leftarrow c_B[w] + \delta[w]$
```

- **Single-Source Shortest Path Algorithm**
    - Given a source, effectively breadth first search, keeping track of the distances
    - Every time you see something new, add it to the queue and set its distance

        - If you see a node which will keep shortest path algorithm – distance to that is distance to current node + 1, add one to the shortest paths going through this edge.
        - Add this node to the list of predecessors for that node.
        - Also, add this to the stack when you process a node, so you can do the back propagation of dependencies
    - **Accumulation for one start!**



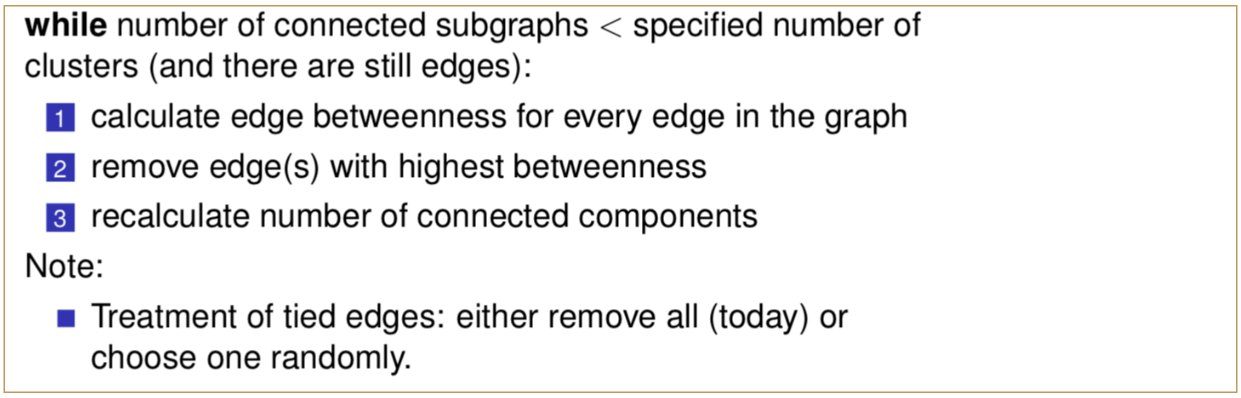
$$\delta(s|v) = \sum_{\substack{(v,w)\in E \\ w:\ d(s,w)=d(s,v)+1}} \sigma(s,v)/\sigma(s,w).(1+\delta(s|w))$$

        - Go backwards through the nodes, and simply apply the formula
    - **Accumulation**
        - Perform the V iterations, once with each node as the source and sum up the δ(s|v) for each node, giving the node's betweeness centrality.

Clustering using betweeness centrality

- **Clustering:** Automatically grouping data according to some notion of closeness or similarity
    - **Agglomerative Clustering** works bottom up
        - "glue regions together"
    - **Divisive Clustering** works top-down by splitting the graph
- **Newman-Girvan method**
    - Form of divisive clustering

- Criterion for breaking links is **edge betweeness centrality**
- When do we know when to stop breaking links?
    - Use prior knowledge when to stop based on numbers of clusters
    - Inherent 'goodness of clustering' metric – using modularity



```
while number of connected subgraphs < specified number of
clusters (and there are still edges):
  1 calculate edge betweenness for every edge in the graph
  2 remove edge(s) with highest betweenness
  3 recalculate number of connected components
Note:
  ■ Treatment of tied edges: either remove all (today) or
    choose one randomly.
```

- **Determining Connected Components**
    - Depth-First search, start at an arbitrary node and mark the other nodes you reach
    - Repeat with unvisited nodes until all are visited
- **Alternative, Manual Method**
    - Supplement the network with numerical estimates of tie strength for the edges, based on empirical evidence
    - Delete edges of minimum total strength
- **Edge Betweeness Centrality**
    - With Node betweeness, we measure the number of shortest paths between two nodes going through a particular node.
    - This time we are measuring the number going through a particular edge
    - The algorithm only changes in the bottom-up (accumulation) phase.
    - $\delta(v)$ is as before, but $c_b(v, w)$

```
for $s \in V$ do
    ▼ single-source shortest-paths problem
        ▼ initialization
            for $w \in V$ do $Pred[w] \leftarrow$ empty list
            for $t \in V$ do $dist[t] \leftarrow \infty$;   $\sigma[t] \leftarrow 0$
            $dist[s] \leftarrow 0$;   $\sigma[s] \leftarrow 1$
            enqueue $s \rightarrow Q$

        while $Q$ not empty do
            dequeue $v \leftarrow Q$;   push $v \rightarrow S$
            foreach vertex $w$ such that $(v, w) \in E$ do
                ▼ path discovery   // — $w$ found for the first time?
                    if $dist[w] = \infty$ then
                        $dist[w] \leftarrow dist[v] + 1$
                        enqueue $w \rightarrow Q$
                ▼ path counting   // — edge $(v, w)$ on a shortest path?
                    if $dist[w] = dist[v] + 1$ then
                        $\sigma[w] \leftarrow \sigma[w] + \sigma[v]$
                        append $v \rightarrow Pred[w]$

    output: betweenness $c_B[q]$ for $q \in V \cup E$ (initialized to 0)
    ▼ accumulation
        for $v \in V$ do $\delta[v] \leftarrow 0$
        while $S$ not empty do
            pop $w \leftarrow S$
            for $v \in Pred[w]$ do
                $c \leftarrow \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w])$
                $c_B[(v, w)] \leftarrow c_B[(v, w)] + c$
                $\delta[v] \leftarrow \delta[v] + c$
            if $w \neq s$ then $c_B[w] \leftarrow c_B[w] + \delta[w]$
```

- **Facebook Circles Dataset**
    - Designed to allow experimentation with automatic discovery of circles
        - Facebook friends from a particular social group
    - Profile and network data from 10 Facebook ego-networks
        - Networks emanating from one person are referred to as an ego
    - Gold-standard circles – manually identified by the egos themselves
    - Average of 19 circles per ego
    - Complete network has 4,039 nodes in 193 circles
    - When doesn't Newman-Girvan work?
        - Nodes may be in multiple circles
        - Sub-categories within categories

    - Nodes not in any category
- **Evaluating Simple Clustering**
  - We assume data sets with gold standard or ground truth clusters
  - But we do not generally have labels for clusters, the number of clusters found may not equal true classes
  - **Purity:** Assign label corresponding to majority class found in each cluster, then count correct assignments, divide by total elements