Data Mining

Module 1: Motivation and distance function

I. Why Data Mining in Biology and Medicine?

A. Data Mining : the search for patterns and statistical dependencies in large datasets

B. Personalised medicine: Tailoring medical treatment to the molecular properties of a patient

II. Data Mining problems in personalised medicine

A. Search for disease-associated loci in the genome

1. Hundreds of new disease-associated genetic loci have been identified
2. The correlations are weak and can not explain the high heritability of these diseases

B. Search for interactions between genetic loci

1. Why interaction search is so difficult?
   a) Human genomes can differ in millions of bases
   b) Without a clever search strategy, we have to consider billions of pairs!

C. Chemoinformatics: Molecule classification

D. Chemoinformatics: Graph classification

1. Why is graph comparison so difficult?
   a) Computational effort grows exponentially with the number of nodes.

III. Data mining's role in the future of medicine

A. The future of datelining in medicine

1. Enormous increase in the amount of data that describes the health state of a person
2. Data Mining
   a) Explore molecular mechanisms underlying diseases
   b) Support when choosing the optimal therapy
   c) Early detection of disease-relevant symptoms
   d) Detection of acute disease symptoms

IV. The Basics of Data Mining

A. Data Mining:

1. The search for reoccurring patterns and statistical dependencies in large datasets

2. Extracting knowledge from large amounts of data

B. Knowledge Discovery Process

1. Data Cleaning: Remove noise and inconsistent data
2. Data integration: Combine multiple data sources
3. Data selection: Retrieving relevant data from database
4. Data mining: Finding reoccurring patterns in data
5. Pattern evaluation: Identifying truly interesting patterns
6. Knowledge presentation: Representing new knowledge for users.

C. Metric

1. For vectors $x_1, x_2, x_3 \in R^d$, a function d is a metric iff
   a) $d(x_1, x_2) \geq 0$
   b) $d(x_1, x_2) = 0$ if and only if $x_1 = x_2$
   c) $d(x_1, x_2) = d(x_2, x_1)$
   d) $d(x_1, x_3) \leq d(x_1, x_2) + d(x_2, x_3)$

D. Distance functions on vectors

1. For vector $x, x' \in R^d$
   a) Manhattan Distance

$$d(x, x') = \sum_{i=1}^{d} |x_i - x_i'|$$

   b) Hamming Distance on binary vectors

$$d(x, x') = \sum_{i=1}^{d} |x_i - x_i'|$$

   c) Euclidean distance

$$d(x, x') = \sqrt{\sum_{i=1}^{d} (x_i - x_i')^2}$$

   d) Chebyshev Distance

$$d(x, x') = \max_i(|x_i - x_i'|)$$

   e) Minkowski Distance

$$d(x, x') = (\sum_{i]1}^{d} |x_i - x_i'|^p)^{\frac{1}{p}}, \text{ where } p \in R^+$$

(1) $p = 1$ -> Manhattan Distance

(2) $p = 2$ -> Euclidean distance

(3) $p \to \infty$ -> Chebyshev distance

(4) The larger p, the more large deviations in one dimension matter

(5) For $p \geq 1$, the Minkowski distance is a metric

E. Similarity measures on sets

1. Finite sets of objects

a) Jaccard coefficient

(1) $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$

(2) Problem: not sensitive to the size of the set

(3) Eg: $|A| = 1, |B| = 100$ and $A \subseteq B$, normally we would think A and B should be similar, however, $j(A, B) = \frac{1}{100} = 1\,\%$

b) Jaccard distance

$$d(A, B) = 1 - j(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

c) Overlap coefficient

$$o(A, B) = \frac{|A \cap B|}{\min(|A|, |B|)}$$

d) Sorensen-Dice coefficient

$$s(A, B) = \frac{2|A \cap B|}{|A| + |B|}$$

2. Sets of vectors

a) Single link distance function

$$d(A, B) = \min_{a \in A, b \in B} d_{vector}(a, b)$$

b) Complete link distance function

$$d(A, B) = \max_{a \in A, b \in B} d_{vector}(a, b)$$

c) Average link distance function

$$d(A, B) = \frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} d_{vector}(a, b)$$

F. Similarity measures on strings

1. K-mer based similarity measures
   a) Goal: Try to quantify the similarity between words w and w'
   b) K-mers: substrings of length k
   c) Represent each string w as a histogram of k-mer frequencies, $h_k(w)$
   d) Spectrum kernel: Count matching pairs of k-mers in w and w'
   e) Example: Similarity between **downtown** and **known**
      (1) $h_3(\textbf{downtown}) = (dow : 1, own : 2, wnt : 1, nto : 1, tow : 1)$
      (2) $h_3(\textbf{known}) = (kno : 1, now : 1, own : 1)$
      (3) Count matching pairs of k-mers: 2

G. Similarity measures on nodes

1. Shortest Path Distance
   a) Objects are nodes in a graph G. Edge weights $w(i, j)$ represent distances between nodes i and j
   b) Goal: Quantify the similarity of an arbitrary pair of nodes
   c) Distance Function: *shortest path length* (the most popular one)
   d) Floyd-Warshall's algorithm: compute all pairs-shortest paths in $O(n^3)$ where n is the number of nodes in G
   e) Floyd-Warshall's Algorithm (G = (V, E, w))
      (1) d(i, j) := w(i, j), $if\ (i, j) \in E$
      (2) d(i, j) := ∞, $if\ (i, j) \notin E$
      (3) For k=1:n do
      (4) For i = 1:n do
      (5) For j = 1:n do
      (6) if d(i, j) > d(i, k) + d(k, j) then
      (7) d(i, j) := d(i, k) + d(k, j)
      (8) Return matrix of shortest path distances D, $D_{ij} = d(i, j)$

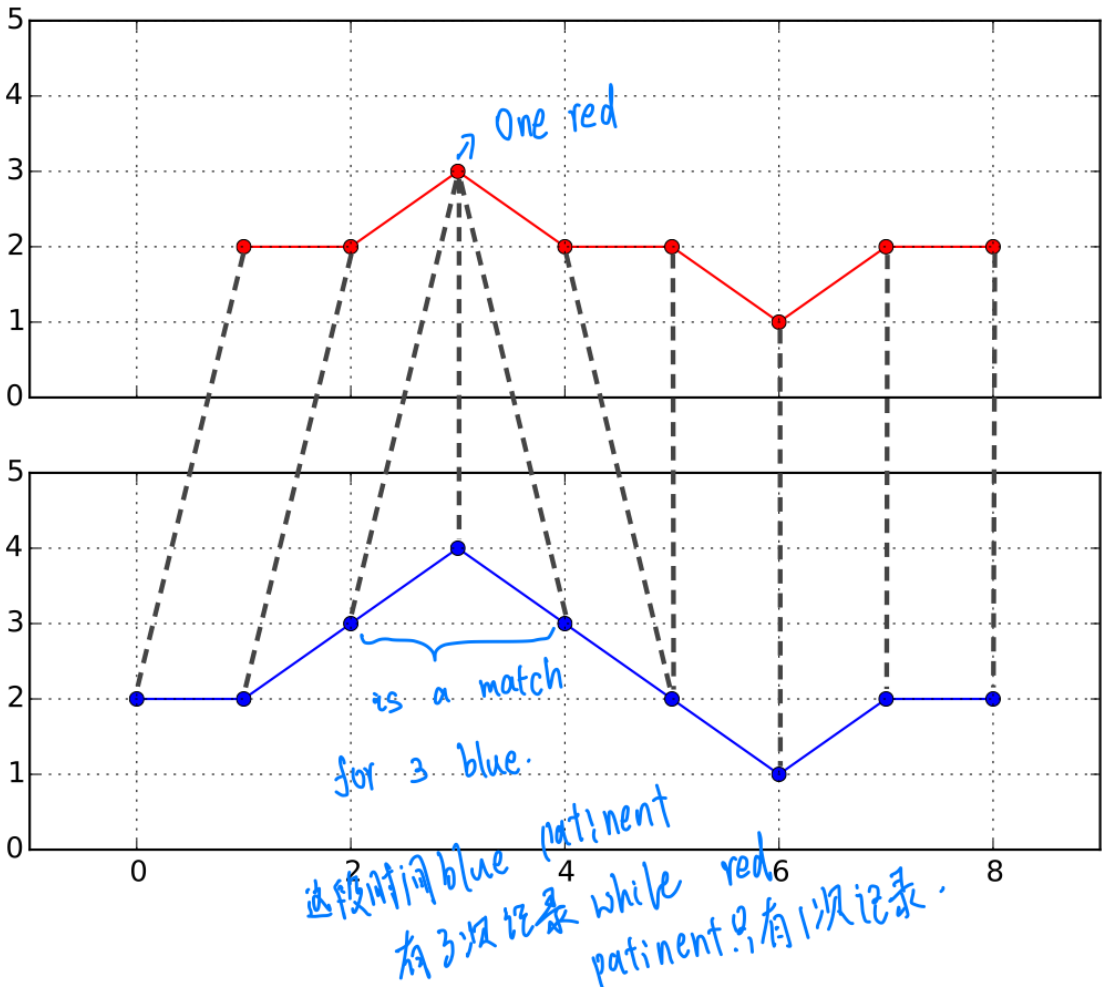
H. Similarity measures on time series

1. Theory and practice:
   a) Ideal setting: If two time series x, x' are vectors of length d and corresponding dimensions represent the same point in time, any vectorial distance function can be used to compare them
   b) In practice:

(1) We often compare time series of different length, $d \neq d'$

(2) The time points at which the time series were observed are not synchronous(同步的)

(3) The time intervals between observations may very within and between time series.



2. Dynamic Time Warping(DTW)

a) A similarity measure for time series of different length, with different intervals between measurements

b) It is the cost of an optimal alignment between the measurements of two time series, x and x'. Individual time points are compared by a base distance function d (e.g. a Minkowski distance)

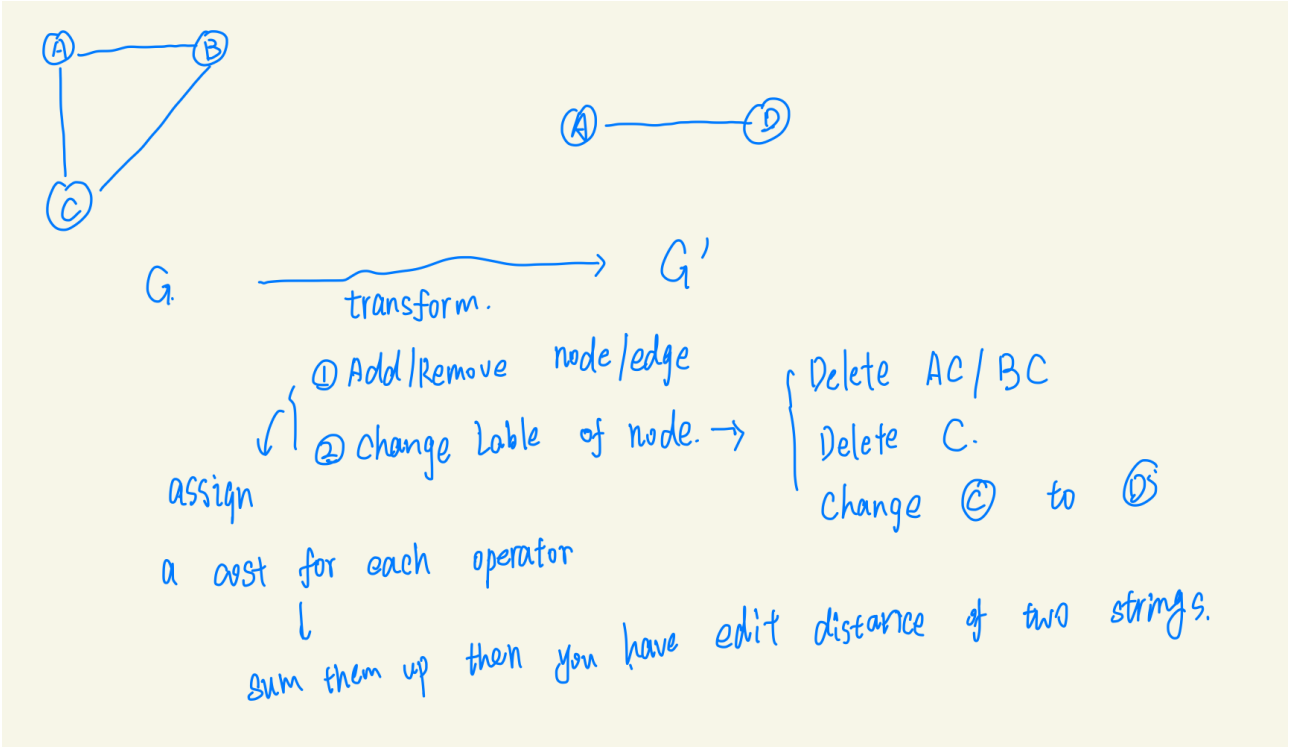
c) The function DTW can be computed recursively as

$$DTW(i,j) = d(x_i, x_j') + min \begin{cases} DTW(i, j-1) & \text{repeat} x_i \\ DTW(i-1, j) & \text{repeat} x_j' \\ DTW(i-1, j-1) & \text{repeat neither} \end{cases}$$

d) $DTW(0,0) = 0$, $DTW(i,0) = \infty$, $DTW(0,j) = \infty$ for all $1 \le i \le d, 1 \le j \le d'$

I. Similarity measures on graphs

1. Approaches to graph comparison
   a) Family 1: Graph isomorphism or subgraph isomorphism tests
      (1) Check whether G' and G are identical
      (2) Check whether G is contained in G' or vice versa
   b) Family 2: Graph edit distances
      (1) Total cost of transforming graph G into graph G'
      (2) Transformations include changing labels, adding or removing nodes or edges



   c) Family 3: Topological vectors
      (1) Map graph to vector
      (2) Then apply vectorial distance functions.
2. Wiener index
   a) Graph representation

(1) Let G be a graph with vertices V and edges ϵ

(2) Let P be the set of shortest paths in G

(3) Then the Wiener index of G is defined as $v(G) = \frac{1}{|P|} \sum_{p \in P} \ell(P)$ where P is the set of all shortest paths and $\ell(p)$ is the length of path p.

b) Graph comparison

(1) The shortest path kernel is a class of similarity measures between two graphs G and G'

(2) The simplest instance of this class is a product between the Wiener indices of G and G': $k(G, G') = v(G)v(G')$

(3) Weisfeiler-Lehman kernel: algorithm

(a) Given labeled graphs G_1 and G_2

(b) For each graph, join the label of node and node's neighbour

(c) Compress each unique label using hash function

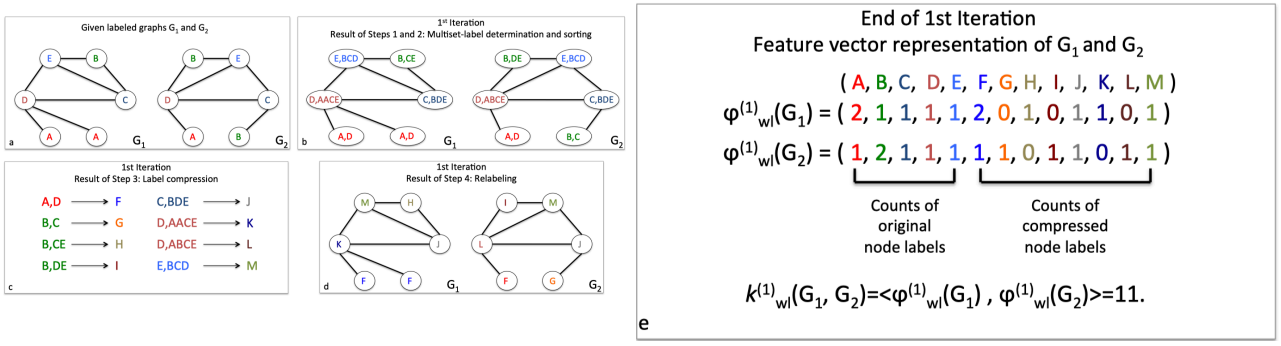
(d) Relabeling both graphs using the output of step (c)

(e) Check the histogram of compressed labels

i) If all the labels are the same: repeat from step (a)

ii) Else stop

(f) Sum up the number of occurrences of those labels which are shared between both graphs G_1 and G_2



(4) Weisfeiler-Lehman kernel: Theoretical runtime properties

(a) Fast Weisfeiler-Lehman kernel:

i) Algorithm: Repeat the following steps h times

(1) Sort: Represent each node v as sorted list L_v of its neighbours (O(m))

(2) Compress: Compress the list into a hash value $h(L_v)$ $(O(m))$

(3) Relabel: Relabel v by the hash value $h(L_v)$ (O(n))

(b) Runtime analysis

i) Per graph pair: Runtime O(m h)

ii) For N graphs: Runtime $O(NmH + N^2nh)$

Module 2: Classification

I. Definition of classification

A. Classification Problem

1. Given an object, which class of objects does it belong to?
2. Given object x, predict its class label y.

B. Problem setting

1. Classification is usually performed in a supervised setting: We are given a **training dataset**
2. A training dataset is a dataset of pairs $\{(x_i, y_i)\}_{i=1}^{n}$, that is objects and their known class labels
3. The test set is a dataset of test points $\{x_i^{'}\}_{i=1}^{d}$ with **unknown** class label
4. The task is to predict the class label $y_i^{'}$ of $x_i^{'}$ via a function f

C. Classes of classification problem

1. if $y \in \{0,1\}$: then we are dealing with a **binary classification** problem
2. if $y \in \{1,...,n\}, (3 \leq n \in \mathbb{N})$: a **multiclass classification** problem
3. if $y \in \mathbb{R}$: a **regression** problem

II. Evaluation of Classifiers

A. Contingency Table

1. In a binary classification problem, one can represent the accuracy of the predictions in a contingency table:

	y = 1	**y = -1**
f(x) = 1	TP	FP
f(x) = -1	FN	TN

2. Accuracy:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

a) Accuracy measures the percentage of the predictions is correct

b) Most common criterion for reporting the performance of a classifier

c) Shortcoming: If the classes are unbalanced, the accuracy on the entire dataset may look high, while being low on the smaller class.

(1) If the TN is way larger than TP, a classifier which always predicts negative would have high accuracy!

3. Precision-Recall

a) Precision:

$$\frac{TP}{TP + FP}$$

b) Recall:

$$\frac{TP}{TP + FN}$$

c) If the positive class if much smaller than the negative class, one should rather use precision and recall to evaluate the classifier.

d) Trade-off between Precision and Recall

(1) By predicting all points to be positive, one can guarantee that recall is 1. However, the precision will then be bad

(2) By only predicting points to be members of positive class for which one is highly confident about the prediction, one will increase precision, but lower recall.

(3) One workaround is to report the precision recall break-even point, that is the value at which precision and recall are identical

4. Dependence on Classification Threshold

a) TP, TN,FP,FN depend on $f(x)$ where $x \in D$

b) The most common definition of $f(x)$ is

$$f(x) = \begin{cases} 1 & \text{if } s(x) \geq \theta \\ -1 & \text{if } s(x) < \theta \end{cases}$$

Where $s : D \to \mathbb{R}$ is a scoring function, and $\theta \in \mathbb{R}$ is a threshold.

c) As the predictions based on f vary with θ, it is important to report results as a function of θ whenever possible, not just for one fixed choice of θ.

5. Report results as a function of θ

a) ROC curves

(1) Receiver Operating Characteristics Curve,

(2) represents the true positive rate versus the false positive rate

(3) True positive rate : Recall

$$\frac{TP}{TP + FN}$$

(4) False positive rate (1 - specificity)

$$\frac{FP}{FP + TN}$$

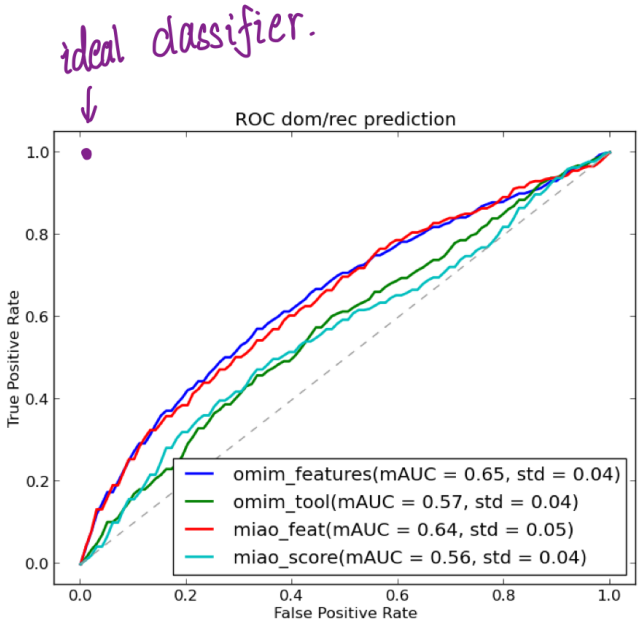
The fraction of negative points that were misclassified.

(5) Each ROC curves starts at (0, 0), if no points are predicted to be positive, then there are no True Positive and False Positives

(6) Each ROC curves ends at (1,1), if all points are predicted to be positive, then there are no TN and FN

(7) The ROC curve of a perfect classifier goes through (0, 1) - it correctly classified all negative points (FP = 0) and correctly classifies all positive points (FN = 0)

(8) ROC curve can go below the diagonal and it means the classifier is doing wrongly classification



(9) AUC : Area under the Receiver Operating Characteristics, a number between 0 and 1

(10) Interpretation: When we present on negative and one positive test point to the classifier, then the AUC is the probability with which the classifier will assign a larger score to the positive than to the negative point.

(11) The larger AUC, the better classifier

(12) The AUC of a perfect classifier can be shown to 1

(13)The AUC of a random classifier is 0.5

(14)The AUC of a misclassifying all points classifier is 0

b) Summarising PR values

(1) 2-D plot of (recall, precision) values for different values of θ

(2) Starts at (0,1). Full precision, no recall. → working convention, no mathematical support

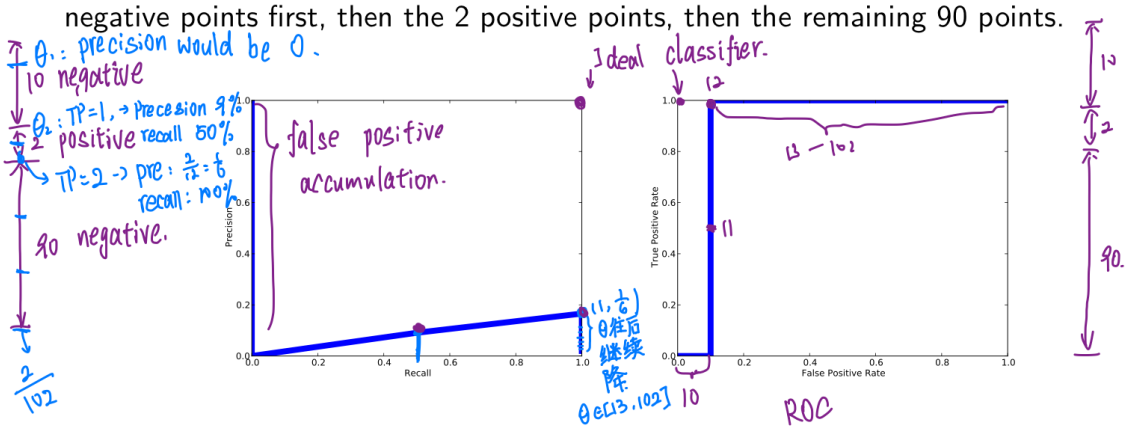
(3) The precision recall break-even point is the point at which the precision-recall-curve intersects the bisecting line

(4) The area under the precision-recall-curve (AUPRC) is another statistic to quantify the performance of a classifier. It is 1 for a perfect classifier, that is, it reaches 100% precision and 100% recall at the same time.

Evaluating Classifiers

Example: The Good and the Bad

- We are given 102 test points, 2 are positive, 100 negative. Our prediction ranks ten negative points first, then the 2 positive points, then the remaining 90 points.



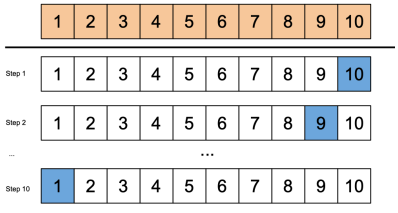
ROC curve judges ON AVERAGE how the true positives rank in the entire dataset

PR Curve requires the true possitive rank in the very top → false possitive accumulation detectable

B. What to do if we only have one dataset for training and testing?

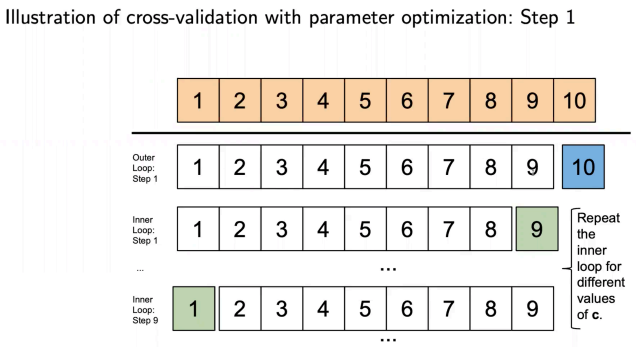
1. Split the dataset into training set and testing set.
2. Splitting the dataset into k subsets and using one of them for testing and the rest for training is referred to as k-fold cross-validation
3. If k = n, cross-validation is referred to as leave-one-out-validation
4. Bootstrapping: Randomly sampling subsets of the data for training and testing and averaging over the results

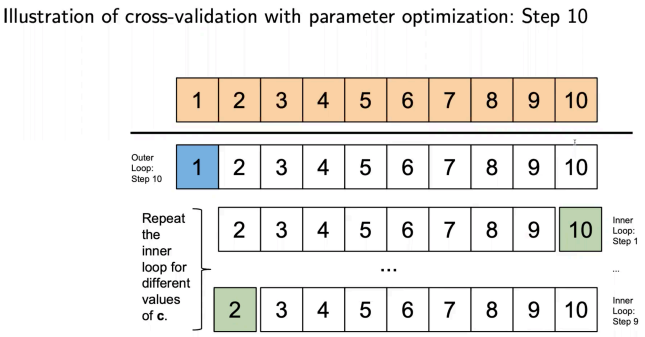
Illustration of cross-validation: 10-fold cross-validation



C. How to optimise the parameters of a classifier?

1. Most classifiers use some parameters c that have to be set
2. It is wrong to optimise these parameters by trying out different values and picking those that perform best on the test set. These parameters are **overfit** on this particular test dataset, and may not generalise to other datasets
3. One needs an internal cross-validation on the training data to optimise c
4. The hyper parameters of one classifier should be chosen on the training set only.





5. Choose the hyper parameter based on internal cross-validation on the training data, and then train the entire training data on the hyper parameter, finally, report the performance on the test dataset.

D. Criteria for choosing a classifier

1. Quality of predictions - there is no optimise classifier for all datasets
2. Runtime and scalability on high-dimensional data and large datasets
3. Interpretability of the classification decision
4. Applicability to diverse types of structured data

III. Nearest Neighbour Classification

A. The actual classification

1. Given x, we predict its label y by

$$x_i = argmin_{x \in D} ||x - x'||^2 \Rightarrow f(x) = y_i$$

Where y_i is the label of x_i

2. The predicted label of x is that of the point closest to it, that is, its nearest neighbour

B. k-NN classification

1. An object is classified to most common among its k nearest neighbours
2. k is a hyperparameter that is chosen by the user

3. The larger k, the lower the influence of single noisy points on the classification
4. In binary classification problems, one usually chooses odd k to avoid ties
5. Instance of instance-based learning and lazy learning
   a) instance based learning: prediction works directly on the training data
   b) lazy learning: No training process
6. Runtime:
   a) compute the distance to all n neighbours in the training datasets $\rightarrow O(n)$
   b) Rank these n distances through sorting $\rightarrow O(nlogn)$
   c) O(n) to find the nearest neighbour in 1-NN
   d) O(n+nlogn) to find the k nearest neighbours in k-NN

C. Challenges in Nearest Neighbour Classification
1. Speed of prediction
2. Choice of k
3. Choice of distance function and weights of dimensions

D. Speed up Nearest Neighbour
1. Triangle inequality:
$$d(x_1, x_2) + d(x_2, x_3) \geq d(x_1, x_3)$$
2. This holds for any metric d
3. Rewrite the triangle inequality
$$d(x_1, x_2) \geq d(x_1, x_3) - d(x_2, x_3)$$
4. $d(x_1, x_3) - d(x_2, x_3)$ provides a lower bound for $d(x_1, x_2)$. If we know a point whose distance to x_1 is smaller than $d(x_1, x_3) - d(x_2, x_3)$, we can avoid calculating $d(x_1, x_2)$

E. Setting hyperparameter k
1. Bootstrapping and cross-validation
2. Cross-validation on the training data
3. Bootstrapping on training data
   a) Split the training data in two subsets T_1 and T_2
   b) For different choices of k, compute the accuracy on T_2 using T_1 as training set.
   c) Repeat the above two steps several times
   d) Pick the k with the highest average accuracy across all iterations

F. Weighting the dimensions

1. The **Mahalanobis distance** d_M takes the $d * d$ covariance structure Σ between features into account

$$d_M(x, x') = \sqrt{(x - x')^T \Sigma^{-1} (x - x')}$$

Where the covariance matrix is defined as

$$\Sigma_{i,j} = cov(X_i, X_j) = \mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)]$$

X_i is a random variable representing feature i and μ_i is its mean.

2. By adding a covariance matrix, we downright the features that are similar to each other.

IV. Naive Bayes Classifier

A. Bayes' Rule

$$P(Y = y \mid X = x) = \frac{P(X = x \mid Y = y)P(Y = y)}{P(X = x)}$$

Here, X is the random variable that describes the features of an object, and Y the random variable that describes its class label. x is an instantiation of X and y an instantiation of Y

B. Naive Bayes Classifier

1. Classify **x** into one of m classes $y_1, \ldots, y_m$:

$$argmax_{y_i} P(Y = yi \mid X = x) = argmax_{y=y_i} \frac{P(X = x \mid Y = y_i)P(Y = y_i)}{P(X = x)}$$

2. Where

a) $P(Y = y_i)$ is the prior probability

b) $P(X = x)$ is the evidence

c) $P(Y = y_i \mid X = x)$ is the posterior probability

d) $P(X = x \mid Y = y_i)$ is the likelihood

C. Simplifications

1. $P(X = x)$ is the same for all classes, ignore this item

$$P(Y = y_i \mid X = x) \propto P(X = x \mid Y = y_i)P(Y = y_i)$$

2. If x is multidimensional with d features $x = (x_1, .., x_d)$, further assume that features are conditionally independent given the class label.

$$P(X = x \mid Y = y_i) = P(X = (x_1, \ldots, x_d) \mid Y = y_i) = \prod_{j=1}^{d} P(X_j = x_j \mid Y = y_i)$$

a) Very strong assumption

b) But usually works in practise

D. Actual Classification

1. The actual classification is performed by computing

$$argmax_{y_i}\{P(Y = y_i \mid X = x)\} \propto P(Y = y_i)\prod_{j=1}^{d} P(X_j = x_j \mid Y = y_i)$$

E. Training the model

1. Estimating $P(Y = y_i)$. Typically, one assumes all classes y_i have the same probability, or one infers the class probability from the class frequencies in the training dataset.
2. Estimating $P(X = x_j \mid Y = y_i)$. Popular choices for $x = (x_1, \ldots, x_d)$ are

a) for binary data: $P(X = x \mid Y = y_i) = \prod_{j=1}^{d} Ber(x_j \mid \theta_{j,i})$, where $\theta_{j,i}$ is the probability that feature x_j has value 1 in class y_i

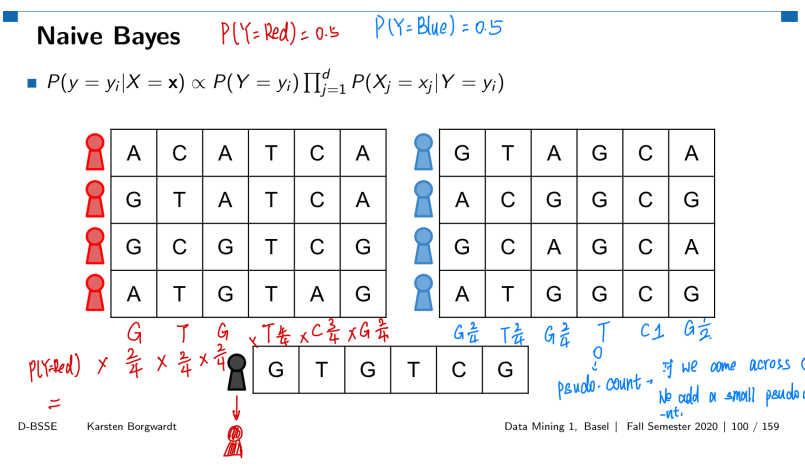
b) for continuous data: $P(X = x \mid Y = y_i) = \prod_{j=1}^{d} N(x_j \mid \mu_{j,i}, \sigma^2_{j,i})$ where $\mu_{j,i}$ is the mean of feature x_j in objects of class i, and $\sigma^2_{j,i}$ is the variance

F. Fundamental Probability Distributions

1. Bernoulli distribution $Ber(x \mid \theta) = \begin{cases} \theta & \text{if x = 1} \\ 1 - \theta & \text{if x = 0} \end{cases}$
2. Gaussian distribution: $N(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$
3. Multivariate Gaussian distribution:

$$N(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} \mid \Sigma \mid^{\frac{1}{2}}} e^{-\frac{1}{2}((x-\mu)^T \Sigma^{-1}(x-\mu))}$$

G. Example:



H. Advantages:

1. Speed: Effort of prediction for on test point is $O(md)$, as we have to compute the class posterior for all m classes
2. Ability to deal with missing data: Missing features x_j can simply be dropped when evaluating the class posteriors, by dropping $P(x_j \mid y_i)$
3. Ability to combine discrete and continuous features: Use discrete or continuous probability distributions for each attribute
4. Practical performance: Despite the unrealistic independence assumption on the features, Naive Bayes often provides good results in practice and is a strong baseline to compare.

V. Linear Discriminant Analysis

A. Underlying assumptions of LDA

1. The goal is to predict a label $y \in \{0,1\}$ from a vector x of d features, that is, $x = \{x_1, \ldots, x_d\}$
2. We assume that $P(Y = 1) = P(Y = 0) = \frac{1}{2}$
3. We assume that the conditional probability of X given Y is a multivariable Gaussian distribution
4. The covariance matrix Σ is the same for both classes, Y = 0 and Y = 1, but they differ in their means, μ_0 and μ_1

B. Log-likelihood ratio

1. The density distribution

$$N(x \mid \mu_y, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} \mid \Sigma \mid^{\frac{1}{2}}} e^{-\frac{1}{2}((x-\mu_y)^T \Sigma^{-1}(x-\mu_y))}$$

The μ_y is different between all the classes

2. We would like to find:

$$argmax_{y \in \{0,1\}} P(Y = y) P(X = x \mid Y = y)$$

3. We predict $f(x) = 1$ if the log-likelihood ratio exceeds zero:

$$\log(\frac{P(Y = 1)P(X = x \mid Y = 1)}{P(Y = 0)P(X = x \mid Y = 0)}) > 0$$

C. Link to linear regression

1. The log likelihood ratio is equivalent to

$$\frac{1}{2}((x - \mu_0)^T \Sigma^{-1}(x - \mu_0)) - \frac{1}{2}((x - \mu_1)^T \Sigma^{-1}(x - \mu_1))$$

2. This can be rewritten as

$$< \vec{w}, \vec{x} > + b = \sum_{i=1}^{d} w_i x_i + b,$$

where

$$\vec{w} = (\mu_1 - \mu_0)^T \Sigma^{-1}$$

and

$$b = \frac{1}{2}(\mu_0^T \Sigma^{-1} \mu_0 - \mu_1^T \Sigma^{-1} \mu_1)$$

3. This shows that linear regression is a Bates-optimal classifier under the aforementioned assumptions
4. To perform LDA, all we have to do is to estimate μ_0, μ_1 and Σ from the data, and use the equations above to determine $\vec{w}$ and b
5. Then the classification is performed via

$$f(x) = sgn(< \vec{w}, \vec{x} > + b)$$

6. This mathematical elegance goes hand in hand with the difficulty to compute Σ^{-1} in very high dimensional spaces.

VI. Logistic Regression

A. Concept

1. Logistic Regression is a *classification* model for binary output variables, $y \in \{-1,1\}$
2. We define an auxiliary target variable z, which is expressed as a linear function of the input variable x, that is $z = \sum_{i=1} w_i x_i + w_0$

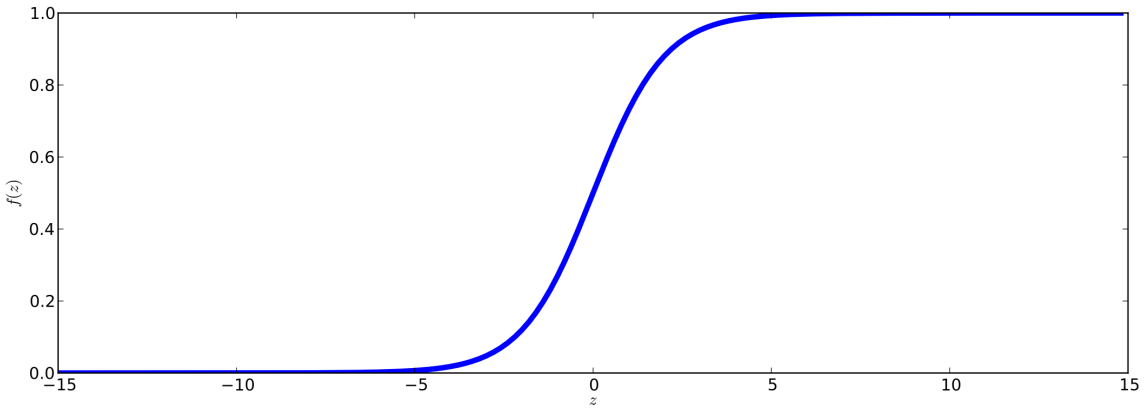
$$z = \sum_{i=1}^{d} w_i x_i + w_0$$

$$= < \tilde{\vec{w}}, \tilde{\vec{x}} >$$

Where $\tilde{\vec{w}} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ w_0 \end{pmatrix}$ and $\tilde{\vec{x}} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ 1 \end{pmatrix}$

3. The logistic function f now maps z to the interval $[0,1]$:

$$f(z) = \frac{exp(z)}{exp(z)+1} = \frac{1}{1+exp(-z)}$$



4. The logistic function can now be written as

$$f_w(x) = f(<\vec{w}, \vec{x}>) = \frac{1}{1+e^{-(w_0+\sum_{i=1}^{d} w_i x_i)}}$$

$f_w(x)$ is the probability that $\vec{x}$ is in class 1

5. The inverse of logistic function $g = f^{-1}$, the logit or log-odds function:

$$g(f_w(x)) = In(\frac{f_w(x)}{1-f_w(x)}) = w_0 + \sum_{i=1}^{d} w_i x_i$$

6. $g \bullet f_w$ clarifies the link to linear regression

B. Training the model

1. We now need an objective that is to be optimised in the training step to learn the weights $\vec{w}$

2. For $y = 1$, the logistic function is $P(y = 1 \mid X = x) = \frac{1}{1+exp(-<\vec{w}, \vec{x}>)}$

   For $y = 0$, the logistic function is $P(y = -1 \mid X = x) = \frac{1}{1+exp(<\vec{w}, \vec{x}>)}$

3. The log probability of each point is therefore

$$\log(\frac{1}{1+exp(-y(\vec{w}, \vec{x})}) = -\log(1+exp(1+exp(-y(\vec{w}, \vec{x}))$$

4. To train the logistic regression model, we minimise the total negative log probability over all points, the *logistic loss function*, which is convex in $\vec{w}$:
   a) convex function: a real-valued function defined on an n-dimensinal interval is called **convex** if the line segment between any two points on the graph of the function lies above the graph between the two points
   b) $$arg \min_{\vec{w} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} log(1 + exp(-y_i < \vec{w}, \vec{x}_i >))$$
5. Several approaches for optimising this object exist, e.g. Maximum Likelihood Estimation.

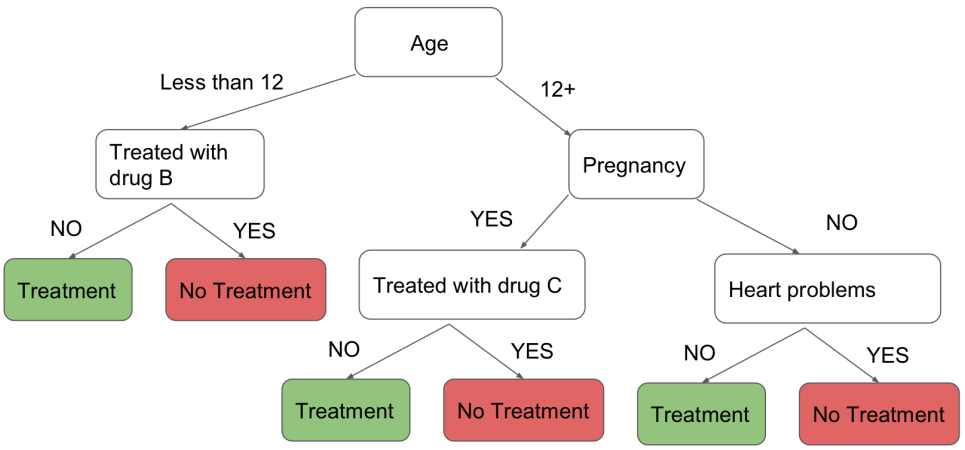
C. Discussion

1. Logistic regression models are easy to fit: The algorithms are easy to implement and fast
   a) There are methods to train Logistic regression models that take time linear in the number of non-zero features in the data, which is the minimal possible time
2. Logistic regression models are easy to interpret, as their output represents the log odd ratio between the poise and the negative class.
3. Several important extensions (multi class classification, nonlinear decision boundaries, other types of output data) are possible

VII. Decision Trees

A. Concept

1. A decision is a flowchart like tree structure with
   a) A root: this is the uppermost node
   b) internal nodes: these represents tests on an attribute
   c) branches: these represent outcomes of a test
   d) leaf nodes: these hold a class label
   e) key idea: recursively split the data space into regions that contain a single class only



B. Classification
   1. Given a test point x
   2. Perform test on the attributes of x at the root
   3. Follow the branch that corresponds to the outcome of this test
   4. repeat this procedure, until you reach a leaf node
   5. predict the label of x according to the label of that leaf node

C. Popularity-Pros
   1. requires no domain knowledge
      a) decision trees are constructed directly from the data
   2. easy to interpret
   3. construction and prediction is fast

D. Construction of Decision Trees
   1. Training Procedure
      a) start with all training examples
      b) select attribute and threshold that gives the “best” split
      c) create child nodes based on split
      d) repeat Step2 and 3 on each child using its data until a stopping criterion is reached
         (1) all examples are in the same class
         (2) number of examples in a node is too small
         (3) the tree gets too large
            (a) overfitting
   2. Central problem: how to choose the “best” attribute? -> Via a cost function
   3. function DecisionTree(D)
      a) If Stopping criterion fulfilled: Predicted class for points in D is the majority class in D
      b) otherwise
      c) $arg \max_{(A,\theta)} = cost(D) - cost(D_{A,\theta})$
      d) $D_{A,<\theta} = \{x \in D \,|\, A < \theta\}$;
      e) $D_{A,\geq\theta} = \{x \in D \,|\, A \geq \theta\}$;
      f) Create two child nodes of D, containing $D_{A,<\theta}$ and $D_{A,\geq\theta}$

g) DecisionTree($D_{A,<\theta}$)

h) DecisionTree($D_{A,\geq\theta}$)

i) end

———————Cost functions for Decision Trees ——————————————

E. Information Gain

1. The information content is defined as

$$Info(D) = -\sum_{i=1}^{m} p(y = y_i | x \in D) log_2(p(y = y_i | x \in D))$$

2. where $p(y = y_i | x \in D)$ is the probability that an arbitrary tuple in D belongs to class y_i and is estimated by $\frac{|\{(x_j, y_j)\} | x_j \in D \cap y_j = y_i|}{|D|}$

3. This is also known as the Shannon entropy of D

4. Assume that attribute A was used to split D into v partitions or subsets, $\{D_1, D_2, \ldots, D_v\}$, where D_j contains those tuples in D that have outcome a_j of A.

   a) Ideally, the D_j would provide a perfect classification, but they seldomly do

   b) How much more information do we need to arrive at an exact classification is quantified by

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{D} info(D_j)$$

5. The information gain is the loss of entropy (increase in information) that is caused by splitting with respect to attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

6. We pick A such that this gain is maximised

F. Gain ratio

1. The information gain is biased towards attributes with a large number of values

   a) For example, an ID attribute maximises the information gain. (We classify every ID as a class)

2. Hence we now extend the information gain to the gain ratio

3. The gain ratio is based on the split information

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \log_2(\frac{|D_j|}{|D|})$$

4. and is defined as

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

5. The attribute with maximum gain ratio is selected as splitting attribute
6. The ratio becomes unstable, as the split information approaches zero
7. A constraint is added to ensure that the information gain of the test selected is at least as great as the average gain over all tests examined.

G. Gini index

1. Gini index measures class impurity as

$$Gini(D) = 1 - \sum_{i=1}^{m} p(y = y_i)^2$$

Where $p(y = y_i)$ is estimated by

$$\frac{|\{(x_j, y_j)\}|x_j \in D \cap y_j = y_i|}{|D|}$$

2. If we split via attribute A into partitions $\{D_1, D_2, \ldots, D_v\}$, the Gini index of this partition is defined as

$$Gini_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} Gini(D_j)$$

3. and the reduction in impurity by a split on A is

$$\Delta Gini(D) = Gini(D) - Gini_A(D)$$

———————-Advantages and Disadvantages of Decision Tree———————————————

H. Advantages

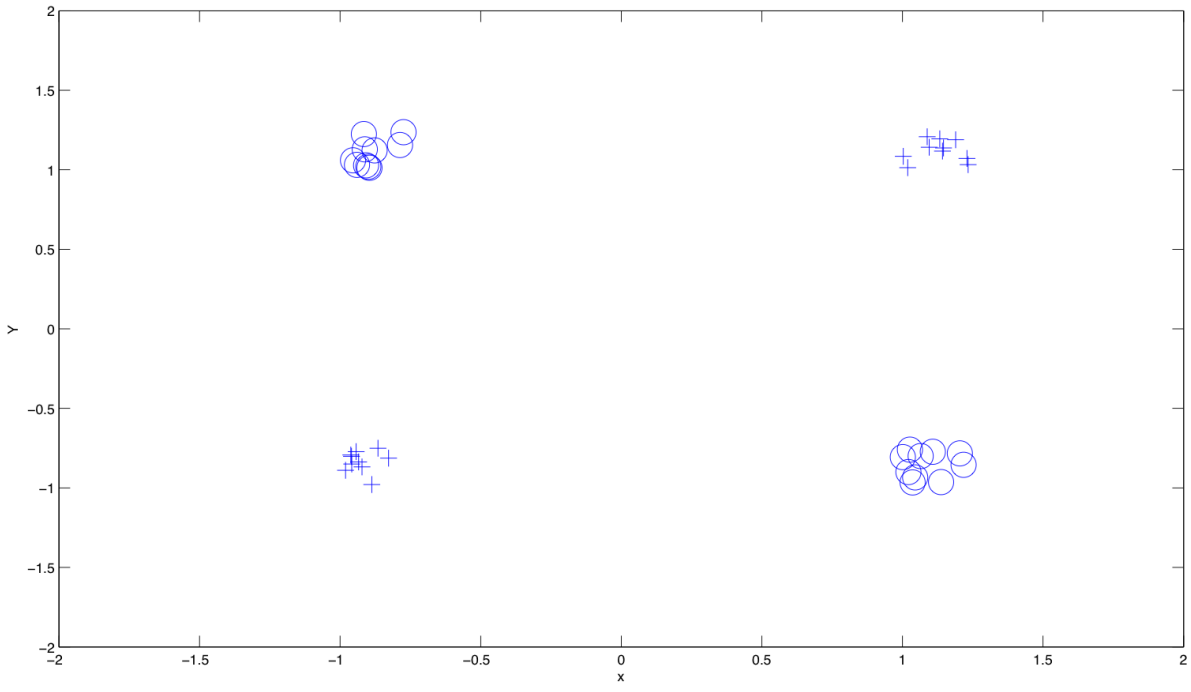
1. Simple to understand and to interpret. Trees can be visualised
2. Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed
3. The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree
4. Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable.
5. Able to handle multi-output problems
6. Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic.
7. Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model

8. Performs well even if its assumptions are somewhat violated by the true model from which the data were generated

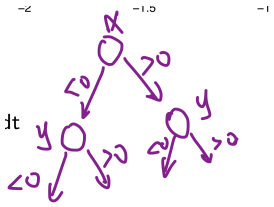
I. Disadvantages

1. Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem
2. Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
   a) For example, drop a feature or one point or two
3. The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms can not guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and apples are randomly sampled with replacement
4. There are concepts that are hard to learn because decision trees do not express then easily, such as XOR, parity or multiplexer problems
5. Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

J. The XOR Problem



1. Since decision trees only use one feature to separate the dataset every time, we can not find a good separation.
   a) When we project the dataset to x or to y, there is no good separation no matter x or y
2. But there do exist a decision tree that can correctly classify this. It's just that the greedy algorithm can not find it.



K. Random Forests

1. To minimize overfitting and the large variability in decision trees, one may use an ensemble of several decision trees.
2. Random Forests: a collection of k decision trees
   a) The idea is to subsample a subset of n' instances and a subset of d' features of the training dataset k times.
   b) On each of these k samples, a decision tree is constructed
   c) Subsequently, the classification of trees is performed by taking a majority vote over the trees.
   d) Three key parameters: number of instances in the subset n', number of features in the subsets d', and number of tree k.

VIII. Support Vector Machines

A. Hyperplane classifiers

1. Vapnik et al. defined a family of classifiers for binary classification problems
2. This family is the class of hyperplanes in some dot product space H

$$< \overrightarrow{w}, \overrightarrow{x} > + b = 0$$

   Where $w \in H, b \in \mathbb{R}$

3. These correspond to decision functions ('classifiers'):

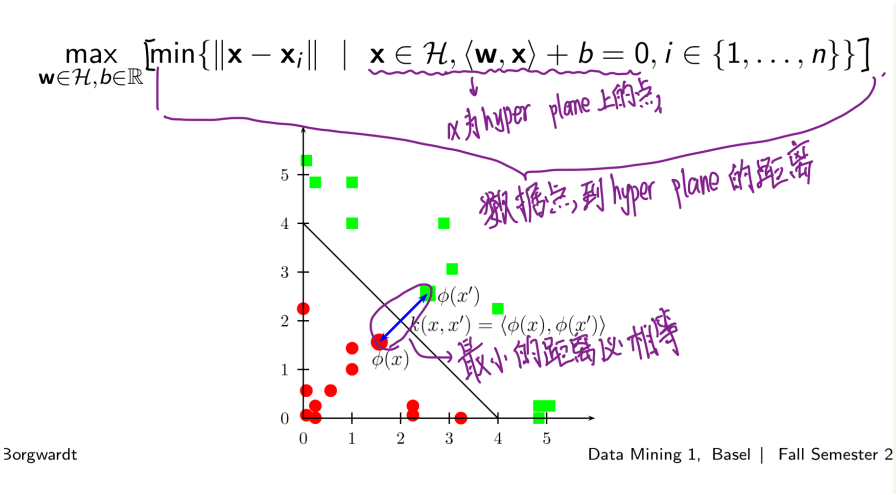
$$f(x) = sgn(< \overrightarrow{w}, \overrightarrow{x} > + b)$$

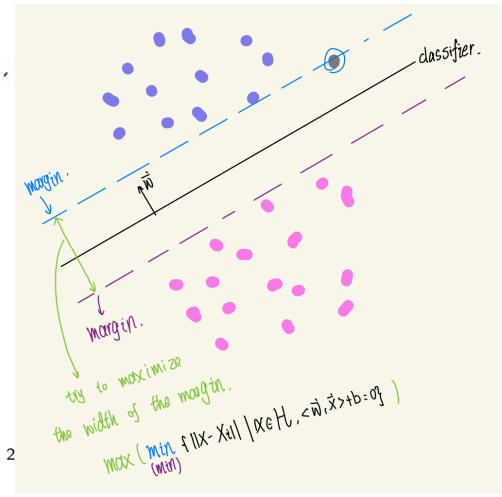
4. Vapnik et cl. proposed a learning algorithm for determining this f from the training dataset.

B. The optimal hyperplane

1. maximises the margin of separation between any training point and the hyperplane

$$\max_{w \in H, b \in \mathbb{R}} [\min\{ ||x - x_i|| \quad | \quad x \in H, < w, x > + b = 0, i \in \{1,...,n\}\}]$$





C. Hard-margin SVM

1. $$\min_{w\in H,b\in\mathbb{R}} \frac{1}{2}||w||^2$$

   subject to $y_i(<w,x_i>+b)\geq 1, \forall i\in\{1,...,n\}$

2. The size of the margin is $\frac{2}{||w||}$. The smaller ||w||, the larger the margin.

3. Constrains $y_i(<w,x_i>+b)\geq 1$ ensure that all training data points of the same class are on the same side of the hyperplane and outside the margin.

D. Soft-margin SVM: C-SVM

1. Points are now allowed to lie inside the margin or in the wrong half space

$$\min_{w\in H,b\in\mathbb{R},\xi\in\mathbb{R}^n} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i$$

   subject to $y_i(<w,x_i>+b)\geq 1-\xi_i, \forall i\in\{1,...,n\} : \xi_i\geq 0$

2. $C\in\mathbb{R}$ is a penalty score parameter that determines the tradeoff between maximising the margin and minimising margin errors

3. ξ is a slack variable, which measures the degree of misclassification of each margin error

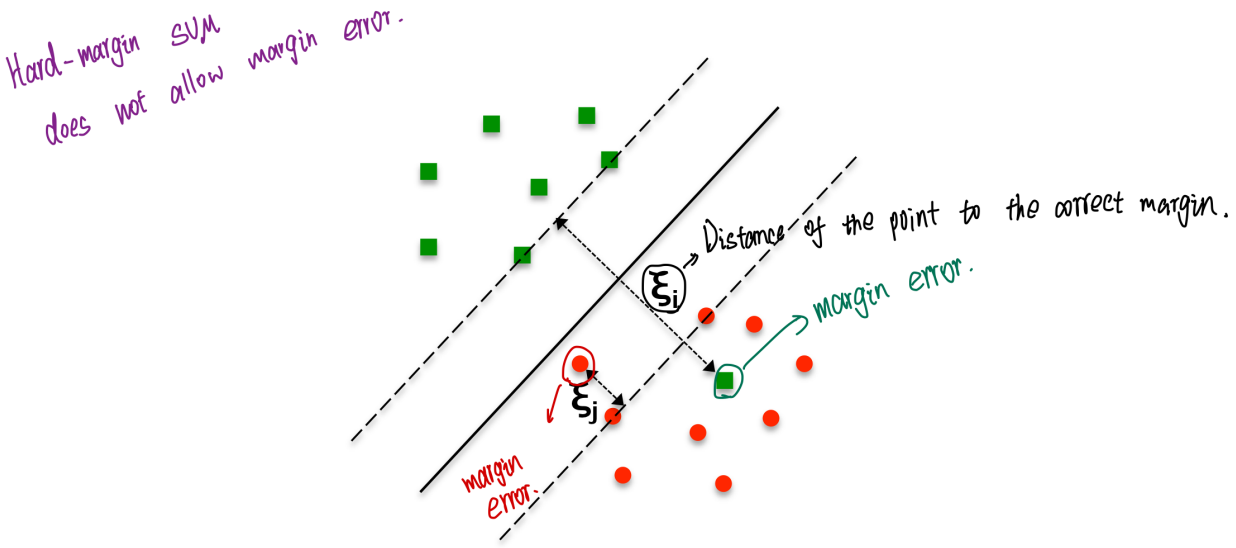
E. Soft-margin SVM: ν-SVM

1. An alternative formulation of soft margin SVMs uses the parameter $\nu\in(0,1]$

$$\min_{w\in H,b\in\mathbb{R},\rho\in\mathbb{R},\xi\in\mathbb{R}^n} \frac{1}{2}||w||^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i - \nu\rho$$

   subject to $y_i(<w,x_i>+b)\geq \rho-\xi_i, \forall i\in\{1,...,n\} : \xi_i\geq 0, \rho\geq 0$

2. ν can be shown to be a lower bound for the fraction of support vectors and an upper bound for the fraction of margin errors among all training points



F. Hard-margin SVM optimisation: The Lagrangian

1. Instead of performing the hard-margin SVM according to the equation on 25.C, we form the Lagrangian:

$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{n} \alpha_i(y_i(< x_i, w > + b) - 1) \qquad (1)$$

2. The Lagrangian is minimised with respect to the primal variables w and b, and maximised with respect to the dual variables α_i.

G. Hard-margin SVM optimisation: Support Vectors

1. At optimality

$$\frac{\partial}{\partial b} L(\vec{w}, b, \vec{\alpha}) = 0 \quad \text{and} \quad \frac{\partial}{\partial \vec{w}} L(\vec{w}, b, \vec{\alpha}) = 0$$

such that

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{and} \quad \vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i \qquad (2)$$

2. Hence the solution vector $\vec{w}$, the crucial parameter of the SVM classifier, has an expansion in terms of the training points and their labels

3. Those training points with $\alpha_i > 0$ are the Support Vectors.

H. Hard-margin SVM optimisation: The dual problem

1. Plugging equation (2) into Lagrangian (1), we obtain the dual optimisation problem that is solved in practice:

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j < x_i, x_j > \quad (3)$$

$$\forall i \in \{1,...n\} : \alpha_i \geq 0$$

I. The Kernel Trick

1. The key insight is that (3) accesses the training data only interns of inner products $< x_i, x_j >$
2. We can plug in an inner product of our choice from any space H here! This is referred to as a kernel k:

$$k(x_i, x_j) = < x_i, x_j >_H$$

3. We assume that data points $x \in \mathbb{R}$ are mapped to a space H via a mapping $\phi : \mathbb{R} \to H$
4. Then the kernel is defined as an inner product $< \cdot, \cdot >: H \times H \to \mathbb{R}$ between data points in this space H:

$$k(x, x') = < \phi(x), \phi(x') >$$

J. SVM: Predict

1. The decision function for Support Vector Machine classification then becomes

$$f(x) = \text{sgn}(\sum_{i=1}^{n} y_i \alpha_i < \phi(x), \phi(x_i) > + b)$$

$$= \text{sgn}(\sum_{i=1}^{n} y_i \alpha_i k(x, x_i) + b);$$

2. The prediction effort is linear in the number of non-zero entries of $\overrightarrow{\alpha}$, that is, in worst-case linear in $O(n)$. In practice, the number of support vectors is often much smaller than n

IX. Kernel Functions

A. Terminology

1. Kernel methods: A machine learning algorithm that access the training data only interns of kernel function
2. kernels and kernel functions: similarity measures
3. kernel matrix: A $m \times n$ matrix, for entry i, j, we compare object i and object j using kernel function.

B. Building kernels in practice

1. In practical applications, one constructs kernels by
   a) Proposing a kernel function and explicitly defining the corresponding mapping ϕ or/and
   b) Combining known kernels in ways that obey the closure properties of kernels

C. Closure properties of kernels

1. Assume we are given two kernels k_1 and k_2. Then $k_1 + k_2$ is a kernel as well
2. Assume we are given two kernel functions k_1 and k_2. Then their product is a kernel $k_1 k_2$ as well
3. Assume we are given a kernel function k and a positive scalar $\lambda \in \mathbb{R}^+$. Then λk is a kernel as well
4. Assume that a kernel k is only defined on a set D. Then its zero-extension k_0 is also a kernel

$$k_0(x, x') = \begin{cases} k(x, x') & \text{if } x \in D \text{ and } x' \in D \\ 0 & \text{otherwise} \end{cases}$$

D. Some prominent kernels

1. linear kernel

$$k(x, x') = \sum_{l=1}^{d} k_l x'_l = x^T x'$$

2. polynomial kernel

$$k(x, x') = (x^T x' + c)^d$$

where $c, d \in \mathbb{R}$,

3. Gaussian Radial Basic Function(RNF) kernel

$$k(x, x') = exp(-\frac{1}{2\sigma^2} ||x - x'||^2)$$

4. the constant ‘all-ones’ kernel

$$k(x, x') = 1$$

5. the delta (Dirac) kernel

$$k(x, x') = \begin{cases} 1 & x = x' \\ 0 & \text{otherwise} \end{cases}$$

E. Kernels on structured data : R-convolution kernels

1. R-Convolution kernels are a famous recipe for constructing kernels on structured data. It is based on decomposing objects X and X' via a relation R into sets of substructures S and S'
2. Their most simple and most widely used instance k_R is the idea to compare all pairs of these substructures of X and X' pairwise:

$$k_R(X, X') = \sum_{s \in S, s' \in S'} k_{base}(s, s')$$

3. For instance, a substructure would be the elements of a set, the nodes of a graph or the substring of a string
4. k_{base} is an arbitrary vectorial kernel, very often even the delta kernel

F. Kernels on strings : String Kernels

1. The spectrum kernel, mentioned above, counts all pairs of matching substrings in two strings.
2. That is, X and X' are strings and S and S' are the sets of all of their substrings.

$$k_R(X, X') = \sum_{s \in S, s' \in S'} k_{base}(s, s')$$

3. k_{base} is a delta kernel, that is, for every pair of matching substrings the kernel increases by one
4. Naively, one has to enumerate all substrings (quadratic effort $O(|X|^2 + |X'|^2)$
5. Via a special data structure called Suffix Trees, it can be done in linear time $O(|X| + |X'|)$

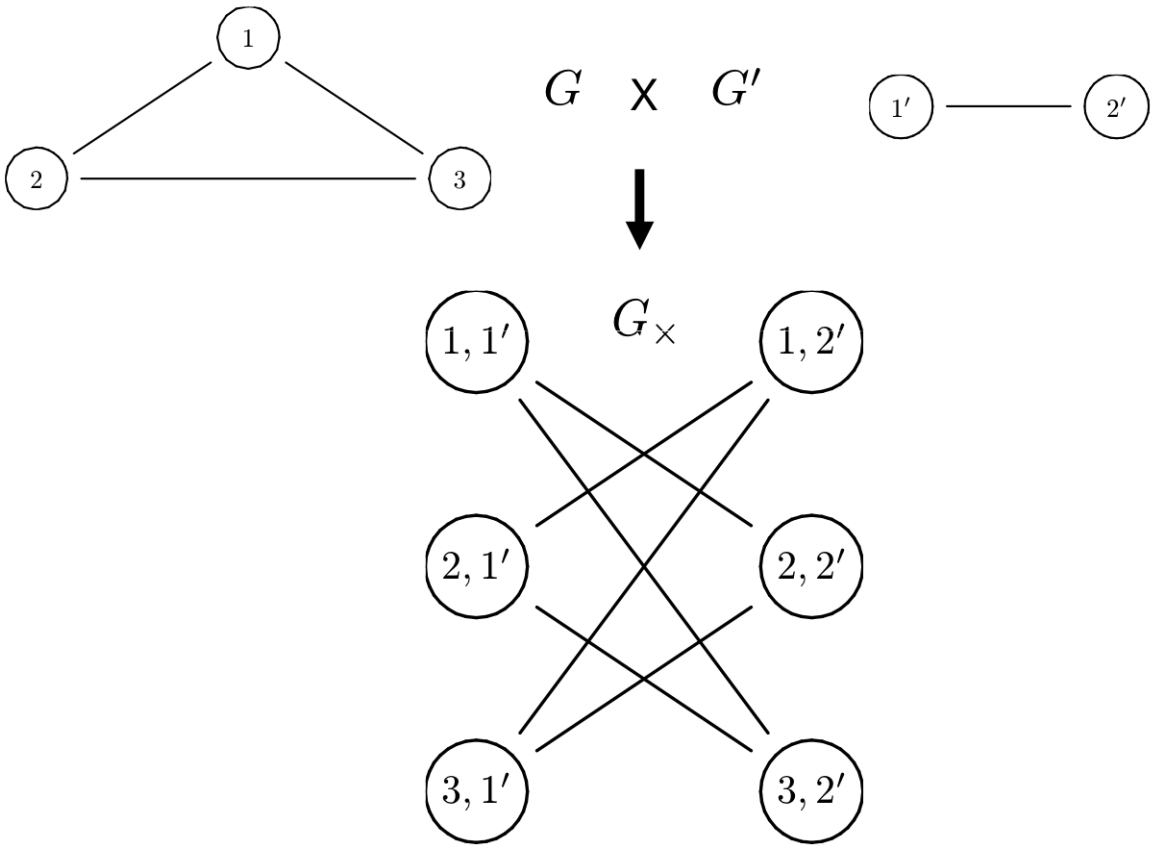
G. Kernels on graphs: Random Walk Kernel

1. Count **matching walks** in G and G': the more matching walks there are, the higher the similarity between G and G'
2. Matching walks are sequences of nodes and edges with matching node labels
3. Elegant Computation
   a) Compute the direct product graph of G and G' and count all walks of length k in it.
   b) Every walk in the product graph corresponds to one walk in G and G'

$$k_x(G, G') = \sum_{i,j=1}^{|V_x|} [\sum_{k=0}^{\infty} \lambda^k A_x^k]_{ij} = e^T (1 - \lambda A_x)^{-1} e$$

Where A_x is the adjacent matrix of the direct product graph and entry i,j in A_x^k is the number of matching walks between node i and node j with length k; λ is a number $\in (0,1)$ used to lower the effect of long walk.

Since A_x is $O(n^2)$ in size, and a naive implementation of calculating an inverse matrix is $O(n^3)$, a naive implementation of this algorithm would be $O(n^6)$



Module 3: Clustering

I. What is Clustering

A. Clustering

1. Class discovery
2. Given a set of objects, group them into clusters (classes that are unknown beforehand)
3. an instance of unsupervised learning

B. In Practice

1. Cluster images to find categories

2. Cluster patient data to find disease subtypes
3. Cluster persons in social networks to detect communities

C. Supervised versus unsupervised learning

1. general inference problem: given x_i, predict y_i by learning a function f
2. training set: set of examples (x_i, y_i) where $y_i = f(x_i)$ (but f is still unknown!)
3. test set: new set of data points x_i where y_i is unknown
4. Supervised: use training data to infer your model, then apply this model to the test data
5. Unsupervised: no training data, learn model and apply it directly on the test data.

II. k-means Clustering

A. Objective:

1. Partition the dataset into k clusters such that intra-cluster variance is minimised

$$V(D) = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2$$

2. Where
   a) V is the variance, S_i is a cluster,
   b) μ_i is its mean, D is the dataset of all points x_j

B. Lloyds algorithm

a) Partition the data into k initial clusters
b) Compute the mean of each cluster
c) Assign each point to the cluster whose mean is closet to the point
d) If any point changed its cluster membership: Repeat from Step 2.

C. Things to note

1. k-means is still the state-of-the-art method for most clustering tasks
2. When proposing a new clustering method, one should always compare to k-means.
3. Lloyds' algorithm has several setbacks:
   a) It is order-dependent
   b) Its results depends on the initialisation of the clusters
   c) Its result may be a local optimum, not the global optimal solution.

D. k-medoid: 'Brother' of k-means

1. Don't use the mean of each cluster but the medoid.
2. The medoid is the point closest to the mean:

$$m_i = argmin_{x_j \in S_i} ||x_j - \mu_i||^2$$

3. One thereby restricts the cluster 'means' to points that are present in the dataset.
4. One only minimises variance with respect to these points
5. m_i is the representative point in that cluster
6. If I have a large number of clusters, we have to store the mean of each classes. But in k-medoid, we only need to add a tag to the medoid.

E. Kernel k-means

1. Kernelised k-means?
   a) It would be attractive to perform clustering using kernels
      (1) can move clustering problem to different feature spaces
      (2) can cluster string and graph data
   b) But we have to be able to perform all steps in k-means using kernels
2. The key step in k-means is to compute the distance between one data point x_1 and the mean of a cluster of points $x_2, \ldots, x_m$:

$$||\phi(x_1) - \frac{1}{(m-1)\sum_{j=1}^{m}\phi(x_j)}||^2 =$$

$$k(x_1, x_1) - \frac{2}{m-1}\sum_{j=1}^{m} k(x_1, x_j) + \frac{1}{(m-1)^2}\sum_{i=2}^{m}\sum_{j=2}^{m} k(x_i, x_j)$$

3. This result is based on the fact that every kernel k induces a distance d:

$$d(x_i, s_j)^2 = ||\phi(x_i) - \phi(x_j)||^2 = k(x_i, x_i) - 2k(x_i, x_j) + k(x_j, x_j)$$

F. Silhouette Coefficients: How to set k in k-means

1. A silhouette coefficient s(x) relates the average distance between a point x and all other points from its cluster C, $d(x, \mu_C)$, to the average distance between a point x and the other points from the second nearest cluster C', $s(x, \mu_{C'})$

$$s(x) = \frac{d(x, \mu_{C'}) - d(x, \mu_C)}{\max(d(x, \mu_{C'}), d(x, \mu_C))}$$

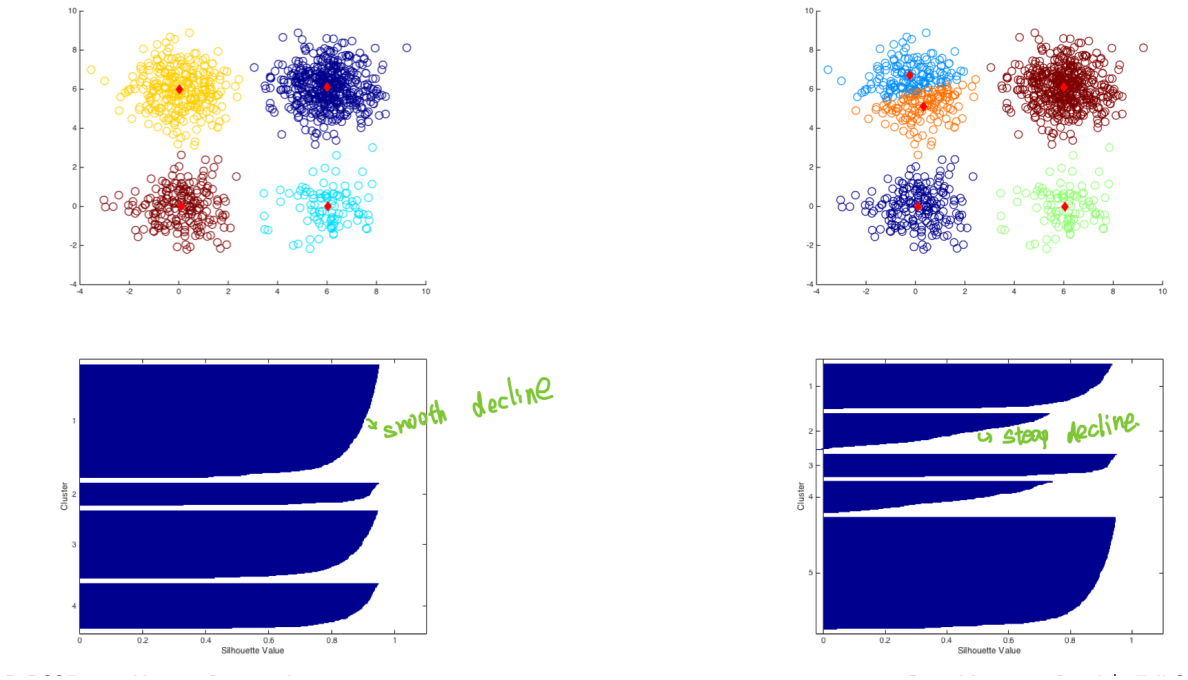
2. Interpretations of $s(x)$

a) $s(x)$ is close to 1, if a point is clearly located in its cluster C

b) $s(x)$ is close to 0, if a point is located between two clusters

c) s(x) is negative, if it is closer to cluster C' than to its current cluster

d) If k is set correctly, the Silhouette Coefficient graph would have a smooth decline, other wise, it would have a steep decline.



III. Graph-based Clustering

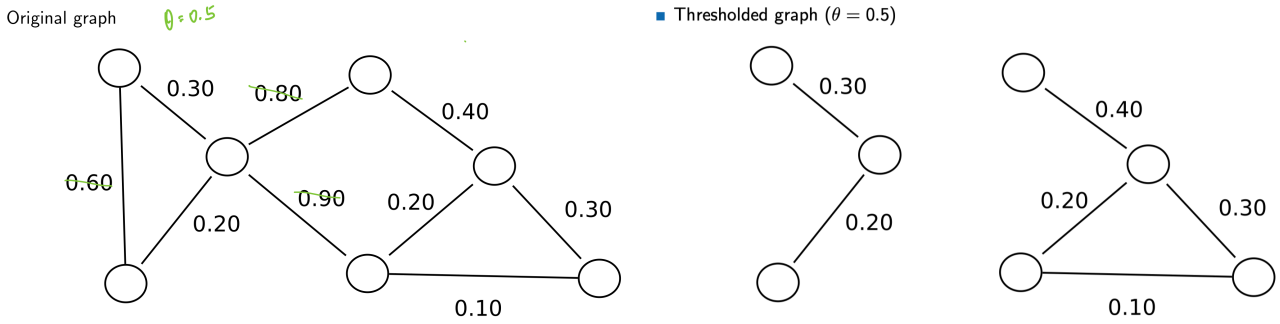
A. Data representation

1. dataset $\mathcal{D}$ is given in terms of a graph $G = (V, E)$
2. a data object v_i is a node in G
3. edge e_{ij} from node v_i to node v_j has weight w_{ij}

B. Graph-based clustering

1. Define a threshold θ
2. Remove all edges e_{ij} from G with weight $w_{ij} > \theta$
3. Each connected component of the graph now corresponds to one cluster
4. Two nodes are in the same connected component if there is a path between them.
5. Graph components can be found by depth-first search in a graph $(O(|V| + |E|))$

C. Example and possible problems of graph-based clustering



1. Single Link Effect: One noisy edge connect two otherwise would be separated cluster into one big cluster
    a) for example, had we miscalculated the weight 0.80 to 0.40, the two separated cluster would become one big cluster
2. This lead to the conclusion that graph-based clustering is not robust to noise

D. Graph representation of data

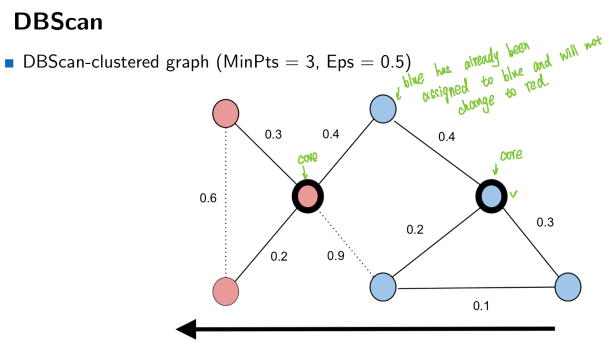
1. Some domains have a natural graph structure.
    a) Telecommunication or social networks
2. Otherwise, obtain the graph structure through a distance function d on the vertices by
    a) connecting each node to its k-nearest neighbours
    b) connecting each node to all nodes in an ϵ-neighbourhood

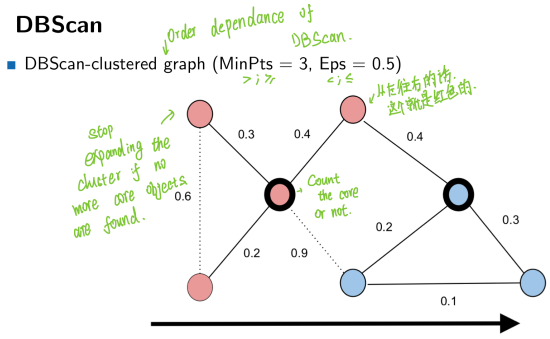
E. DNScan

1. Noise-robust graph-based clustering
    a) Graph-based clustering can suffer from the fact that one noisy edge connects two clusters
    b) DBScan is a noise-robust extension of graph-based clustering
    c) DNScan is short for Density-Based Spatial Clustering of Applications with Noise
2. Core object : Idea of core object/ How DBScan overcome noise?
    a) Two objects v_i and v_j with distance $d(v_i, v_j) < \epsilon$ belong to the same cluster if either v_i or v_j are a core object.
    b) v_i is a core object iff there are *MinPoints* points within a distance of ϵ from v_i
    c) A cluster is defined by iteratively checking this core object property
    d) If two points v_i and v_j are connected by edge $E_{i,j}$, we trust that the $E_{i,j}$ is not a noise if either v_i or v_j (or both) have sufficient near neighbours

3. Oder dependance of DBScan

   a) The result of DBScan can sometimes be different if we scan the graph in different order





4. Properties of DBScan

   a) Cluster assignment of border points is order dependent

   b) Unlike k-means, one does not have to specify the number of clusters as a priori

   c) But one has to set MinPts and Eps

   d) Ester et.cl report that fro 2D examples MinPts=4 is sufficient for good results

   e) They determine Eps by visual inspection of a k-distance plot

   f) How to kernels DBScan?

IV. Pseudocode of DBScan

A. Code: Main

```
1. DBSCAN(SetOfPoints, Eps, MinPts) // SetOfPoints is UNCLASSIFIED
2. ClusterId := nextId(NOISE); // nextId(NOISE) → cluster_0
3. for i FROM 1 TO SetOfPoints.size do
4.     Point := SetOfPoints.get(i);
5.     if Point.ClId = UNCLASSIFIED then
6.         if ExpandCluster(SetOfPoints, Point, ClusterId, Eps, MinPts) then
7.             ClusterId := nextId(ClusterId)
```

B. Code Expand Cluster

```
1. ExpandCluster(SetOfPoints, Point, ClId, Eps, MinPts): Boolean;
2. seeds:= SetOfPoints.regionQuery(Point, Eps); // ϵ-neighbour of points
3. if seeds.size < MinPts then
4.     SetOfPoints.changeClId(Point, NOISE);
5.     RETURN False;
6. else
7.     SetOfPoints.changeClIds(seeds, ClId); // change cluster id for ϵ-neighbours
8.     seeds.delete(Point);
9.     while seeds ≠ Empty
10.        currentP := seeds.first();
11.        result := SetOfPoints.regionQuery(currentP, Eps);
12.        if result.size ≥ MinPts then
13.            for i FROM 1 TO result.size do
14.                resultP = result.get(i)
15.                if resultP.ClId IN (UNCLASSIFIED, NOISE) then
16.                    if resultP.ClId = UNCLASSIFIED then
17.                        seeds.append(resultP)
18.                    SetOfPoints.changeClId(resultP,ClId)
19.        seeds.delete(currentP);
20.    return True
```

V. Spectral Clustering

A. Concept

1. Spectral clustering connects graph-based clustering with k-means
2. But what is the link between these two approaches?
3. To understand this link, we must first familiarise ourselves with cut-based clustering

B. Cut-based clustering

1. Objects are nodes in a graph G with nodes V and edges E. The adjacency matrix W represents the similarities between pairs of nodes.
2. Assume V is partitioned into k subsets: $V = \{C_1, \ldots, C_k\}$.
3. Cut-based clustering tries to minimise the total weight of inter-cluster edges:

$$\min \frac{1}{2} \sum_{a=1}^{k} \sum_{b=1}^{k} k(C_a, C_b)$$

Where $k(C_a, C_b) = \sum_{v_i \in C_a, v_j \in C_b, a \neq b} W_{ij}$ and $k(C_a, C_a) = 0$

C. Link to Graph Laplacian

1. The degree matrix D is defined as

$$D_{ij} = \begin{cases} \sum_{k=1}^{n} W_{ik} & i = j \\ 0 & i \neq j \end{cases}$$

where $\sum_{k=1}^{n} W_{ik}$ is the sum of edges that starts from node i

2. The (unnormalised) Graph Laplacian is defined as $L = D - W$.
3. Let $\overrightarrow{c_a}$ be a vector of size n (number of nodes) that $c_a(i) = \begin{cases} 1 & \text{if } v_i \in C_a \\ 0 & \text{if } v_i \notin C_a \end{cases}$
4. Note that $< \overrightarrow{c_a}, \overrightarrow{c_a} > = ||C_a||^2 = |C_a|$, the size of cluster C_a
5. Furthermore, note that all $\overrightarrow{c_a}$ and $\overrightarrow{c_b}$ are orthogonal, that is $< \overrightarrow{c_a}, \overrightarrow{c_b} > = 0$

$$\begin{aligned} \overrightarrow{c_a}^T L \overrightarrow{c_a} &= \overrightarrow{c_a}^T (D - W) \overrightarrow{c_a} \\ &= \sum_{i,j} c_a(i) c_a(j) (D_{ij} - W_{ij} \\ &= \sum_{i} c_a(i) \left(\sum_{\ell=1}^{n} W_{i\ell} \right) - \sum_{i,j} c_a(i) c_a(j) W_{ij} \\ &= \left(\sum_{b} \sum_{v_i \in C_a, v_j \in C_b} W_{ij} \right) - \sum_{v_i \in C_a, v_j \in C_a} W_{ij} = \sum_{b} \sum_{v_i \in C_a, v_j \in C_b, a \neq b} W_{ij} \\ &= \sum_{b} k(C_a, C_b) \end{aligned}$$

6. It follows that finding the minimum k-cut is identical to minimising

$$\min \frac{1}{2} \sum_{a=1}^{k} \overrightarrow{c_a}^T L \overrightarrow{c_a}$$

D. Ratio Cut

1. Minimum k cut clustering is prone to finding very small clusters
2. Ratio Cut accounts for this problem by dividing the cut size by the size of the cluster

$$\min_{C} \sum_{a=1}^{k} \frac{1}{|C_a|} \sum_{b=1}^{k} k(C_a, C_b)$$

3. This can be rewritten as

$$\min_{C} \sum_{a=1}^{k} \frac{\overrightarrow{c_a}^T L \overrightarrow{c_a}}{||c_a||^2}$$

4. Finding the optimal cut C, that is the optimal binary cluster indicator vectors $\overrightarrow{c_a}$ for $a \in \{1...k\}$ is NP-hard
5. We therefore relax the solution and allow the vector $\overrightarrow{c_a}$ to take any real value, rather than being binary
6. The objective can then be rewritten as

$$\sum_{a=1}^{k} \left(\frac{c_a}{||C_a||} \right)^T L \left(\frac{c_a}{||c_a||} \right) = \sum_{a=1}^{k} u_a^T L u_a$$

   a) where $\mu_a = \frac{c_a}{||c_a||}$ is the unit vector in the direction of vector $\overrightarrow{c_a} \in \mathbb{R}^n$

7. To find the optimal u, we first incorporate the constraint $u_a^T u_a = 1$ via a Lagrange multiplier λ_a into the equation above.

   a) We are trying to find the optimal u with contain that $\overrightarrow{u}$ is a unit vector

   b) $\sum_{a=1}^{k} u_a^T L u_a + \sum_{a=1}^{k} \lambda_a (1 - u_a^T u_a)$

8. Setting the derivative with respect to $\overrightarrow{u_i}$ to zero implies

$$\frac{\partial}{\partial \overrightarrow{u_i}} \left(\sum_{a=1}^{k} u_a^T L u_a + \sum_{a=1}^{k} \lambda_a (1 - u_a^T u_a) \right) = 0$$

$$2L\overrightarrow{u_i} - 2\lambda_i \overrightarrow{u_i} = 0$$

$$L\overrightarrow{u_i} = \lambda_i \overrightarrow{u_i}$$

9. This implies that all $\overrightarrow{u_a}$ are eigenvectors of L.
10. Using Equations above, it follows that

$$\overrightarrow{u_i}^T L \overrightarrow{u_i} = \overrightarrow{u_i}^T \lambda_i \overrightarrow{u_i} = \lambda_a$$

11. This in turn implies that in order to minimise $\sum_{a=1}^{k} u_a^T L u_a$, one should choose the k smallest eigenvalues of L and their corresponding eigenvectors.
12. The eigenvectors represent the relaxed cluster indicator vectors (excluding u_n)
    a) But how should we know which cluster does one node belong to

E. Concept

1. Spectral Clustering solves this problem pragmatically by using the vectors $\overrightarrow{u_a}(a \in [1..k])$ as a new representation of the data points and applying k-means to this new representation after normalisation
2. The new representation is $U = (\overrightarrow{u_n}, \overrightarrow{u_{n-1}}, \ldots, \overrightarrow{u_{n-k+1}})$, a $n \times k$ matrix
3. It is normalised row-by-row to obtain the new k-dimensional representation

$$Y = \begin{pmatrix} \overrightarrow{y_1} \\ \overrightarrow{y_2} \\ \cdots \\ \overrightarrow{y_n} \end{pmatrix}, \text{via } \overrightarrow{y_i} = \frac{1}{\sum_{j=1}^{k} u_{i,n-j+1}^2}(u_{i,n}, u_{i,n-1}, \ldots, u_{i,n-k+1}) \quad (12)$$

Here, $\overrightarrow{y_i}$ is a row vector

F. Pseudocode

1. procedure SPECTRAL CLUSTERING(D, k)
2. Compute the similarity matrix $W \in \mathbb{R}^{n \times n}$ and the Laplacian $L = D - W$;
3. Solve $L\mu_a = \lambda_a \mu_a$ for $a = n, \ldots, n-k+1$, where $\lambda_n \leq \lambda_{n-1} \leq \ldots \leq \lambda_{n-k+1}$
4. $U := (\overrightarrow{u_n}, \overrightarrow{u_{n-1}}, \ldots, \overrightarrow{u_{n-k+1}})$
5. $Y :=$ normilised rows of U via Equation 12
6. $C := \{C_1, \ldots, C_k\}$ via k-means on Y
7. return: C

G. computational complexity

1. The overall worst case runtime is $O(n^3)$ due to the need to compute eigenvectors and eigenvalues
2. For sparse graphs with m edges, this runtime can be improved to $O(mn)$

3. Running $k-$means requires a runtime in $O(tnk^2)$ where t is the number of iterations of k-means until convergence.

H. Spectral clustering connects graph-based clustering with k-means

1. Graph-based: Spectral Clustering is trying to solve a cut-based graph clustering
2. $k-$means: we use k-means on matrix Y

VI. Soft-assignment Clustering

A. Hard Assignment Clustering

1. one point is assigned to one class
2. the clustering algorithm discussed so far are all hard assignment clustering

B. EM clustering

1. Soft k-means
   a) k-means is based on a hard assignment of points to clusters
   b) Wouldn't it be more realistic to work with probabilities of each point to belong to each cluster rather than a hard assignment?
   c) This is the core idea of Expectation Maximisation (EM) Clustering with a Mixture of Gaussian distributions
   d) It is also referred to as soft k-means
2. The General EM algorithm
   a) We are dealing with observed variables (objects X and their features) and latent variables (the cluster membership of the objects Y), and model parameters θ (parameters of the underlying probability distribution)
   b) We would like to maximise $p(X \mid \theta)$
   c) This optimisation is difficult as

$$\log(p(X \mid \theta)) = \log\{\sum_{Y} p(X, Y \mid \theta)\}$$

   That is, we have to sum over the latent variables inside the logarithm, which makes the evaluation of the maximum likelihood extremely challenging

   d) The EM algorithm circumvents this problem in an iterative 2-step procedure.
      (1) No guarantee of global optimum, may find local optimum
   e) Given a joint distribution $p(X, Y \mid \theta)$ over observed variables X and latent variables Y, governed by parameters θ, the goal is to maximise the likelihood function $p(X \mid \theta)$ with respect to θ:

(1) Choose an initial setting for the parameters θ^{old}

(2) Expectation step (E step) : Evaluate $p(Y \mid X, \theta^{old})$.

(3) Maximisation step (M step): Evaluate θ^{new} given by

$$\theta^{new} = \underset{\theta}{\text{argmax}}\, \mathcal{Q}(\theta, \theta^{old}),$$

where $\mathcal{Q}(\theta, \theta^{old}) = \sum_{Y} p(Y \mid X, \theta^{old}) \log p(X, Y \mid \theta)$

(4) Check for convergence of parameters or log likelihood. If not converged, then $\theta_{old} \leftarrow \theta_{new}$ and return to Step 2

f) EM may converge to a local optimum

3. Concept & Example

a) Consider the case of a mixture of k multivariable Gaussians in which θ is a triplet $(c, \{\mu_1, \ldots, \mu_k\}, \{\Sigma_1, \ldots, \Sigma_k\})$

b) where $P_\theta[Y = y] = c_y$ and

$$P_{\theta(t)}[X = x \mid Y = y] = \mathcal{N}(x \mid \mu_y, \Sigma_y) = \frac{1}{(2\pi)^{\frac{d}{2}} \mid \Sigma_y \mid^{\frac{1}{2}}} e^{-\frac{1}{2}((x-\mu_y)^T \Sigma_y^{-1}(x-\mu_y)}$$

c) For simplicity, we assume that $\Sigma_1 = \Sigma_2 = \ldots = \Sigma_k = I$, where I is the identity matrix

d) Expectation Step

(1) For each $i \in \{1,...,n\}$ and $y \in \{1,...,k\}$ we have that

$$P_{\theta(t)}[Y = y \mid X = x_i] = \frac{1}{Z_i} P_{\theta(t)}[Y = y] P_{\theta(t)}[X = x_i \mid Y = y] = \frac{1}{Z_i} c_y^{(t)} exp(-\frac{1}{2} \mid\mid x_i - \mu_y^{(t)} \mid\mid^2)$$

(2) Where Z_i is a normalisation factor which ensures that $\sum_{y} P_{\theta(t)}[Y = y \mid X = x_i] = 1$

e) Maximisation Step

(1) We need to maximise the following expression with respect to μ and c to obtain $\theta^{(t+1)}$:

$$\sum_{i=1}^{n} \sum_{y=1}^{k} P_{\theta(t)}[Y = y \mid X = x_i](log(c_y) - \frac{1}{2} \mid\mid x_i - \mu_y \mid\mid^2)$$

(2) Setting the derivative with respect to μ_y of the term above to zero and rearranging the terms, one obtains:

$$\mu_y = \frac{\sum_{i=1}^{n} P_{\theta(t)}[Y = y \mid X = x_i] x_i}{\sum_{i=1}^{n} P_{\theta(t)}[Y = y \mid X = x_i]}$$

(3) μ_y is the weighted average of the x_i, where the weights are according to the probabilities calculated in the E step.

(4) The maximal c can be shown to be

$$c_y = \frac{\sum_{i=1}^{n} P_{\theta(t)}[Y = y \mid X = x_i]}{\sum_{y'=1}^{k} \sum_{i=1}^{n} P_{\theta(t)}[Y = y \mid X = x_i]}$$

4. Comparison to k-means

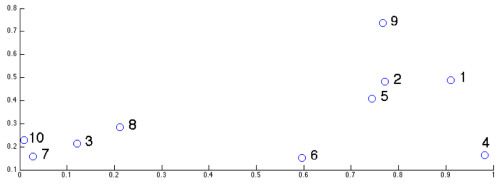
a) k-means assigns each point to a cluster according to the distance to the cluster means. Then the cluster means are updated based on the **examples of this cluster**

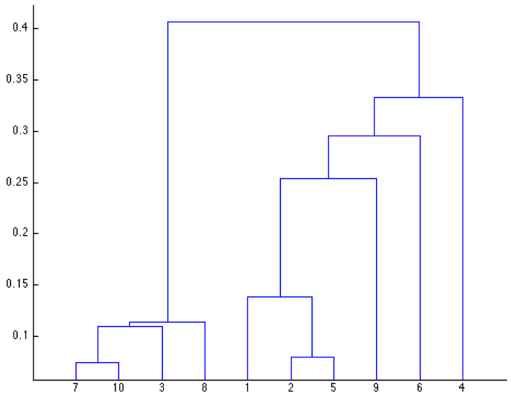
b) EM determines the probability that each example belongs to each cluster. Then the cluster means are updated **based on a weighted sum over all points.**

C. Hierarchical Clustering

1. All other clustering algorithms assume a flat data meaning that there's no super set clusters between clusters and that all points are created equal
2. Join the most similar clusters
   a) Iteratively join the two most similar clusters
   b) But how to measure similarity between clusters?
3. Similarity of clusters
   a) Single Link: $s(C_i, C_j) = \min_{x \in C_i, x' \in C_j} d(x, x')$
   b) Average Link: $s(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i, x' \in C_j} d(x, x')$
   c) Complete Link: $s(C_i, C_j) = \max_{x \in C_i, x' \in C_j} d(x, x')$
4. Pseudocode
   a) procedure HIERARCHICAL CLUSTERING(D, s)
   b) Initialise each point $x_i \in D$ as its own cluster C_i, for $i \in \{1,...,n\}$
   c) repeat
   d) $(i*, j*) = \operatorname{argmin}_{(i,j)} s(C_i, C_j)$
   e) Merge clusters C_{i*} and C_{j*}
   f) until $|C| = 1$

5. Example: Dendrograms as a way to represent hierarchical clusterings





6. Advantage and Disadvantages

a) Advantage

(1) ITs clustering reflects the entire structure of the dataset

b) Disadvantage

(1) It is difficult to make a clear state about cluster membership in hierarchical clustering, as each point belong to a hierarchy of clusters

(2) Stopping Hierarchical Clustering early is an approach to circumvent this cluster-assignment problem. These criteria could be stop the merging of clusters

(a) once a pre-defined number of clusters k has been reached

(b) once the distance between the closest cluster exceeds a threshold ϵ

Module 4 Feature Selection

I. What is Feature Selection?

A. Abundance of features

1. Usually, our output variable Y does not depend on all of our input features X
2. Why is this? X usually includes all features that *could* determine Y according to our prior knowledge, but we do not know for sure
3. In fact, we perform supervised learning to determine this dependence between input variables and out variables
4. (Supervised) feature selection means selecting the relevant subset of features for a particular learning task.
5. Note the difference to feature ranking, which orders features according to their relevance, and feature transformation, which translates the original features into a new feature representation

II. Why Feature Selection?

A. Reasons for feature selection

1. to detect causal features
2. to remove noisy features
3. to reduce the set of features that has to be observed
4. to reduce cost and to improve speed and data understanding

B. Two modes of feature selection

1. Filter approaches: select interesting features a priori, based on a quality function (information criterion)
   a) Select interested features before performing any kind of learning
   b) independent from any kind of classifier
2. Wrapper approaches: select special features that are interesting for one particular classifier
   a) Combine the selection process with one particular classifier
   b) classifier dependent

III. Optimisation Problem

A. Combinatorial Problem

1. Given a set of features D,
2. and a quality function q
3. we try to find the subset S of D of cardinality n' that maximises q

$$\text{argmax}_{S \subset D \wedge |S| = n'} q(S)$$

B. Exponential runtime effort

1. The computation effort for enumerating all possibilities is exponential in n', and hence intractable for large D and n'
2. In practice, we have to find a workaround!

C. Greedy Selection

1. Take the currently best one
   a) Greedy Selection is an alternative to exhaustive enumeration
2. Idea is to iteratively
   a) add the currently most informative feature to the selected set or
   b) remove the currently most uninformative feature from the solution set
3. These two variants of greedy feature selection are referred to as:
   a) forward feature selection
   b) backward elimination
4. Forward Feature Selection
   a) $S^+ \leftarrow \emptyset$
   b) repeat
   c) $j \leftarrow \underset{j}{\text{argmax}}\, q(S^+ \cup j)$
   d) $S^+ \leftarrow S^+ \cup j$
   e) $S \leftarrow S - j$
   f) until $|S^+| = n'$
   g) $j \leftarrow \underset{j}{\text{argmax}}\, q(S^+ \cup j)$ takes $O(n)$ time
   h) the repeat until loop takes $O(n')$ time
   i) in total it would take $O(nn')$ time for general choice of Q
5. Backward Elimination
   a) $S^+ \leftarrow D, S \leftarrow \emptyset$
   b) repeat
   c) $j \leftarrow \underset{j}{\text{argmax}}\, q(S^+ - j)$
   d) $S^+ \leftarrow S^+ - j$
   e) $S \leftarrow S \cup j$
   f) until $|S^+| = n'$

g) $j \leftarrow \underset{j}{\text{argmax}}\, q(S^+ - j)$ takes $O(n)$ time

h) the repeat until loop takes $O(n - n')$ time

i) In total it would take $O((n - n')n)$ time for general choice of Q

j) If $n' < \frac{n}{2}$, we choose forward selection

k) if $n' > \frac{n}{2}$, we use backward elimination

6. Optimality of greedy selection

a) Only optimal if q decomposes over the elements of S

$$q(S) = \sum_{x \in S} q(x)$$

(1) In this case, we can

(2) calculate $q(x)$ for each x

(3) Sort $q(x)$

(4) extract first n' xs

b) Near-optimal if q is submodular

c) Otherwise, there is guarantee for optimality

IV. Correlation Coefficient

A. Definition

1. The correlation coefficient $\rho_{X,Y}$ between two random variables X and Y with expected values μ_X and μ_Y and standard deviations σ_X and σ_Y is defined as:

$$\begin{aligned}\rho_{X,Y} &= \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} \\ &= \frac{\mathbb{E}((X - \mu_X)(Y - \mu_Y))}{\sigma_X \sigma_Y}\end{aligned}$$

2. where $\mathbb{E}$ is the expected value operator and coo means covariant

B. It measures the strength of the relationship between the relative movements of two variables. The value ranges from -1 to 1. -1 means perfect negative correlation while 1 means perfect positive correlation

V. Mutual Information

A. Definition

1. Given two random variable X and Y, we define the mutual information $I(\cdot, \cdot)$ as

$$I(X, Y) = \sum_{y \in Y} \sum_{x \in X} p(X = x, Y = y) \log \left(\frac{p(X = x, Y = y)}{P(X = x) p(Y = y)} \right)$$

2. where
   a) X is the input variable
   b) Y is the output variable
   c) $p(X = x, Y = y)$ is the (joint) probability of observing x and y
   d) $p(X = x)$ and $P(Y = y)$ are the marginal probabilities of observing x and y, respectively
   e) log is usually the logarithm with base 2
3. It means how dependent are the states of these variables from each other
4. How likely it is to observe particular joint values for this two variables in the joint probability.

VI. Hilbert-Schmidt Independence Criterion

A. Definition

1. The Hilbert-Schmidt Independence Criterion (HSIC) measures the dependence of two random variables X and Y
2. An empirical estimate of the HSIC is proportional to

$$\text{trace}(KHLH)$$

3. Where
   a) K is a kernel matrix on X
   b) L is a kernel on Y
   c) H is a centering matrix with $H_{ij} = \delta(i, j) - \frac{1}{n}$
   d) trace is the sum of the diagonal of a matrix
4. $HSIC(X, Y) = 0$ iff X and Y are independent
5. The larger $HSIC(X, Y)$. the larger the dependence between X and Y

VII. Submodular Functions

A. Definition

1. A function q on a set D is said to be submodular if

$$q(S \cup X) - q(S) \geq q(T \cup X) - q(T)$$

2. where

a) $X \in D$: One feature

b) $X \subset D$: Subset of features

c) $T \subset D$: another subset of features

d) $S \subseteq T$: T is larger than S

3. This is referred to as the property of 'diminishing returns'

a) if S is a subset of T

b) then S benefits more from adding X than T

4. $q(S \cup X) - q(S)$ is the quality gain of adding X to S

B. Near-optimality

1. if q is a submodular, nondecreasing set function and $q(\varnothing) = 0$, then the greedy algorithm is guaranteed to find a set S such that

$$q(S) \geq (1 - \frac{1}{e}) \max_{|T|=|S|} q(T)$$

2. This means that the solution of greedy selection reaches at least $\sim 63\,\%$ of the quality of the optimal solution

C. Example: Sensor Placement

1. Imagine our features form a graph $G = (D, E)$

2. Imagine the features are possible locations for a sensor. Each sensor may cover a node v and its neighbourhood $\mathcal{N}(v)$ (including node v), that is $q(S) = |\, U_v \mathcal{N}(v) \,|$

3. Now we want to pick locations in the graph such that our sensors cover as large an area of the graph as possible.

4. q fulfils the following properties

a) $q(\varnothing) = 0$

b) q is non-decreasing

c) q is submodular

5. Hence greedy selection will lead to near-optimal sensor placement!

VIII.Wrapper Methods

A. Two flavours
   1. not-embedded (Wrapper): The learning algorithm is employed as a quality measure
   2. embedded: The selection process is really integrated into the learning algorithm

B. Wrappers
   1. Simple wrapper: do prediction using 1 feature only. Use classification accuracy as measure of quality
   2. Extend this to groups of features by heuristic search strategies (e.g. greedy, Monte-Carlo)

C. Embedded
   1. Typical example: Decision trees and ℓ_0 norm SVM

D. Filter: select most relevant features before putting them into a learning algorithm

E. ℓ_0 norm SVM
   1. Key Idea
      a) Train a regular linear SVM (using ℓ_1-norm or ℓ_2-norm regularisation).
         (1) $SVM : f(\vec{x}) = \text{sgn}(< \vec{w}, \vec{x} > + b)$
      b) Re-scale the input variables x by multiplying them by the absolute values of the components of the weight vector w obtained, that is $x_{new} = w \odot x$ ($\odot$ is the point-wise product)
      c) iterate the first 2 steps until convergences.
         (1) if the set of features that are non-zero are not changing anymore
      d) We use $\vec{w}$ because $\vec{w}$ has large entries in dimensions that correspond to important features

IX. Number of Features

A. What if we do not know a reasonable choice of n'?
   1. Pick all features that are significantly correlated with the output variable
      a) Problem 1: Multiple testing problem: Even some randomly generated features may show a significant correlation
      b) Problem 2: Ignores interactions and correlations between features
   2. Use the probe method
      a) Insert ‘fake’ features (probes) into the set of features
      b) Fake features can be drawn randomly from a Gaussian distribution, or they can be created in a nonparametric manner by randomly shuffling existing features.

c) Stop feature selection when you select the first fake feature or when the proportion of the fake features exceeds a certain threshold

B. Unsupervised Feature selection

1. Problem Setting

a) Even without a target variable Y, we can select features that are informative according to a criterion of interest

2. Criteria

a) Saliency: A feature is salient if it has a high variance or range.

b) Entropy: A feature has high entropy if the distribution of examples is uniform

c) Smoothness: A feature in a time series is smooth if its average local curvature is moderate

d) Density: A feature is in a high-density region if it is connected with many other variables

e) Reliability: A feature is reliable if the measurement error bars are smaller than the variability

X. Feature Selection in Practice

A. Catalog of 10 questions by Guyon and Elisseeff

1. Do you have domain knowledge? If yes, construct a better set of ad-hoc features
2. Are your features commensurate? If no, consider normalising them.
3. Do you suspect interdependence of features? If yes, expand your feature set by constructing conjunctive features or products of features, as much as your computer resources allow you.
4. Do you need to prune the input variables (e.g. for cost, speed or data understanding reasons)? If no, construct disjunctive features or weighted sums of features
5. Do you need to assess features individually (e.g. to understand their influence on the system or because their number is so large that you need to do a first filtering)? If yes, use a variable ranking method; else do it anyway to get baseline results
6. Do you need a predictor? If no, stop
7. Do you suspect your data is “dirty” (has a few meaningless input patterns and/ or noisy outputs or wrong class labels)? If yes, detect the outlier examples using the top ranking variables obtained in step 5 as representation; check and / or discard them
8. Do you know what to try first? If no, use a linear predictor. Use a forward selection method with the “probe” method as a stopping criterion or use the ℓ_0 -norm embedded method. For comparison, following the ranking of step 5, construct a sequence of predictors of same nature using increasing subsets of

features. Can you match or improve performance with a smaller subset? If yes, try a non-linear predictor with that subset.

9. Do you have new ideas, time, computational resources, and enough examples? If yes, compare several feature section methods, including your new idea, correlation coefficients, backward selection and embedded methods. Use linear and non-linear predictors. Select the best approach with model selection.
10. Do you want a stable solution (to improve performance and/or understanding)? If yes, subsample your data and redo your analysis for several “bootstraps”

B. Revealing Examples:

1. Can presumably redundant variables help each other?
   a) Noise reduction and consequently better class separation may be obtained by adding variables that are presumably redundant
2. How does correlation impact variable redundancy?
   a) Perfectly correlated variables are truly redundant in the sense that no additional information is gained by adding them
   b) Very high variable (anti-) correlation does not mean absence of variable complementarity
3. Can a variable that is useless by itself be useful with others?
   a) Two variables that are useless by themselves can be useful together.

Module 5: Applications in Computational Biology

I. Overfitting and Generalisation: Deleteriousness Prediction

A. Deleteriousness Prediction

1. Assessing the impact of missense variants
   a) Given the availability of more and more sequencing data on individual patients, one fundamental question to ask is: Is a variant at a particular position in the genome deleterious?
   b) Even when restricting ourselves to missense variants that cause an amino acid change, one is usually left with tens of thousands of these variants
   c) This motivated the development of a large number of computational tools to predict the deleteriousness of missense variants.
2. Assessing the impact of missense variants
   a) A multitude of definitions of deleteriousness, datasets, and algorithms for deleteriousness prediction (SIFT, POLYPHEN, MUTATIONTASTER, LRT, GERP, FatHMM) exists.
   b) For the practitioner, it is extremely hard to choose among this plethora of approaches

c) Out goal: To provide the cleanest and most comprehensive comparative evaluation of different deleteriousness predictors on a wide variety of datasets

d) We compared 10 methods on 5 widely used datasets

3. Two Major Types of Circularities

a) Type 1 Circularity: The common benchmark datasets used for training and testing tools overlap to a large degree

b) Type 2 Circularity: Most proteins contain only deleterious or only neutral variants. A naive majority class vote within a protein gives (artificially) excellent results.

4. Conclusions:

a) A comparative evaluation of deleteriousness prediction tools is complicated by two types of circularity