Data Mining 1 Cheat Sheet

Let's learn some CBB Crew

02/02/2020

Metrics

1 Definition of a metric

We have vectors $x_1, x_2, x_3 \in \mathbb{R}^d$ that form an Euclean space of dimension d. A function is a metric iff

1. $d(x_1, x_2) \geq 0$
2. $d(x_1, x_2) = 0$ if and only if $x_1 = x_2$
3. $d(x_1, x_2) = d(x_2, x_1)$
4. $d(x_1, x_3) \leq d(x_1, x_2) + d(x_2, x_3)$ (Triangle inequality)

2 Similarity measures

2.1 Similarity measures on vectors

We assume that $x, x' \in \mathbb{R}^d$

Manhattan distance

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

In a binary vector this definitions corresponds to the *Hamming distance*.

Euclidean distance

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

Chebyshev distance

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

Minkowski distance

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

where $p \in \mathbb{R}^+ \wedge p \geq 1$. The larger p the more deviations in one dimension matter. For $p \to \infty$, the Minkowski distances converges to the Chebyshev distance.

2.2 Similarity measures on finite sets

Jaccard coefficient

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

Jaccard distance

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

A problem with Jaccard measures occurs when one set is much smaller than the other one. The overlap can be big but the coefficient will still be very small.

Overlap coefficient

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

Here min is the minimal cardinality. It sets the the overlap in relation to the smaller set. (c.f Jaccard)

Sorensen-Dice coefficient

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

Single link distance function

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

Complete link distance function

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

Average link distance function

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

2.3 Similarity measures on strings

k-mer based similarity measures **Goal**: Quantify the similarity between words w and w':

- a *k-mer* is a substring of length k.
- Represent each string w as a histogram of *k-mer* frequencies $h_k(w)$.
- The *spectrum kernel* counts number of matching pairs of *k-mer* in w and w'.

2.4 Similarity measures on nodes

Shortest path distance Objects are nodes in a graph G. Edge weights $w(i,j)$ represent distances between nodes i and j .
Goal: to quantify the similarity of an arbitrary pair of nodes. `Floyd-Warshall's` algorithm allows to compute all pairs-shortest paths in $O(n^3)$, where n is the number of nodes in G.

Algorithm 1: Floyd-Warshall's algorithm ($G = (V, E, w)$)

Input : Graph with vertices V, edges V and weights w.
Output: Matrix of shortest path distance D, $D_{ij} = d(i,j)$,

```
d(i,j) := w(i,j), if (i,j) ∈ E;
d(i,j) := ∞, if (i,j) ∉ E;
for k = 1 : n do
    for i = 1 : n do
        for j = 1 : n do
            if d(i,j) > d(i,k) + d(k,j) then
                d(i,j) := d(i,k) + d(k,j)
            end
        end
    end
end
```

2.5 Similarity measures on time series

Dynamic time warping 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. The function `DTW` can be computed recursively.

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

where $DTW(0,0) = 0$, $DTW(i,0) = \infty$, $DTW(0,j) = \infty \quad \forall \quad 1 \le i \le d, \quad 1 \le j \le d'$.

2.6 Similarity measures on graphs

There are different approaches to graph comparisons:

- Graph isomorphism or subgraph isomorphism test
- Graph edit distance $\rightarrow$ cost of transforming one graph to another graph
- Topological vectors $\rightarrow$ map graph to vectors then apply vectorial distance functions

Let G be a graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$. The **Wiener Index** is a topological approach to represent graphical properties of a molecule:

$$v(G) = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \mathcal{L}(p)$$

where $\mathcal{P}$ is the shortest paths in Graph G and $\mathcal{L}(p)$ is the length of path p.

Shortest path kernel can be used to compare graphs G and G'. The simplest instance of this class is a product between the Wiener indices of G and G'

$$k(G, G') = v(G)v(G')$$

SPKernel on unweighted and undirected graphs With two graphs G_1, G_2 and their corresponding adjacency matrices A_1, A_2 the shortest path kernel can be calculated by:

1. Transforming the adjacency matrices into shortest path matrices S_1, S_2, where $s(i,j)$ is the length of the shortest path between i and j.
2. calculate the SPKernel by:

$$K_{sp}(S_1, S_2) = \sum_{e_1 \in S_1} \sum_{e_2 \in S_2} K^1_{\text{walk}}(e_1, e_2)$$

$$K^1_{\text{walk}}(e_1, e_2) = \begin{cases} 1, & \text{if weight}(e_1) = \text{weight}(e_2) \\ 0, & \text{otherwise} \end{cases}$$

3. e_1 is an edge walk of lenght l in S_1, which is a non-zero entry of S_1. Since S_1 is symmetric, only the upper or lower triangular matrix of S_1 should be considered.

The runtime complexity of the SPkernel is in $\mathcal{O}(n^3)$, where n is the number of nodes (vertices).

The Weisfeiler-Lehman kernel is a procedure that allows one to compute the similarity between graphs efficiently. The following steps are executed on the two graphs:

Algorithm 2: Weisfeiler-Lehmann Kernel

```
for h steps do
    Sort: represent each node v as a sorted list L_v
      of its neighbours
    Compress: the sorted label using a hash value
      h(L_v)
    Relabel: v using the hash value h(L_v)
    Count: the labels obtained in each graph n ⇒
      feature vector representation of the graph:
      φ^(h)(G_n)
    Kernel value is the inner product of the
      vectors: k^(h)(G_1, G_2) = ⟨φ^(h)(G_1), φ^(h)(G_2)⟩
end
```

The runtime complexity of the W-L kernel per graph pair is in $\mathcal{O}(mh)$ and in $\mathcal{O}(Nmh + N^2nh)$ where h is the amount of iterations, m is the size of the list of the unhashed graph, which is equal to the number of edges and n is the size of the labels of the hashed graph.

Classification

A training set is a dataset of pairs $\{(x_i, y_i)\}_{i=1}^n$ which is a set of objects and their known class labels. The test set is a dataset of test points $\{x_i'\}_{i=1}^d$ with unknown class label. The classification function $f(x_i')$ predicts a class label y_i'. There exist *binary* ($y \in \{0, 1\}$), *mulit-class* ($y \in \{1, ..., n\} 3 \leq 3 \in \mathbb{N}$) or *regression* ($y \in \mathbb{R}$) problems.

3 Evaluating classifiers

The Contingency Table is given by:

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

The accuracy is defined as:

$$\text{accuracy} = \frac{\text{TP + TN}}{\text{TP + FP + FN + TN}}$$

The precision is defined as:

$$\text{precision} = \frac{\text{TP}}{\text{TP + FP}} = \frac{\text{TP}}{\text{"actual results"}}$$

The recall is defined as:

$$\text{recall} = \frac{\text{TP}}{\text{TP + FN}} = \frac{\text{TP}}{\text{"predicted results"}}$$

There is a trade-off between precision and recall: by predicting all points to be positive one can guarantee that the recall is 1. However, the precision will then be bad. The `precision recall break-even point` is the value at which precision and recall are identical.

3.1 Dependence on Classification Threshold

TP, FP, TN, FN depend on $f(x)$ where $x \in \mathcal{D}$. $s : \mathcal{D} \to \mathbb{R}$ is a scoring function and $\theta \in \mathbb{R}$ is a threshold.

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

Since predictions based on f vary with θ it is important to report the results as a function of θ. The efficient strategy to compute all solutions as a function of θ is to rank all points x by their score $s(x)$.

3.2 ROC curves

The Receiver Operating Characteristics Curve, which represents the `true positive rate` ($(TP)/(TP + FN)$) versus the `false positive rate` $(FP)/(FP + TN)$.

- the curve goes from (0,0) to (1,1)
- a prefect classifier goes through the point (0,1)
- the ROC curve does not depend on an arbitrarily chosen threshold θ, but it seems difficult to summarize the performance of a classifier in terms of a ROC curve. We need the AUC.

The Area under the Receiver Operating Characteristics (**AUC**), a number between 0 and 1. When we present one 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.

- The larger AUC, the better the classifier.
- The AUC of a perfect classifier can be shown to be 1, the one of a random 0.5 and the one of a stupid 0.

3.3 PR curve

2-D plot of (recall,precision) values for different values of θ. Starts at (0,1): Full precision, no recall. The *precision recall break-even point* is the point at which the precision-recall-curve intersects the bisecting line. 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.

3.4 Evaluating classifiers

It is wrong to optimize parameters of a classifier 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 generalize to other datasets. Instead, one needs an internal cross-validation on the training data to optimize parameters.

Cross-validation Optimize the classifier only on the test set. In every loop chose a test object and train the parameters on it. Finally, take the average/median that gives the overall best result.

Algorithm 3: k-fold Cross validation

```
Partition the dataset into k subsamples.
for k − 1 subsamples do
    use them as training data and test set:
    for k − 1 partitions do
        make sure that one is kept as a test set:
        Train on k − 2 partitions, and evaluate on
         left over partition.
    end
end
Get the value of the hyperparameter that best
 performs on the data that has not been used for
 training.
```

Choosing a classifier

- Quality of predictions - but be aware of the *No-Free-Lunch-Theorem*
- Runtime and scalability on high-dimensional data and large datasets
- Interpretability of the classification decision
- Applicability to diverse types of structured data

4 Nearest Neighbour Classification

Given x we predict its label y by

$$x_i = \underset{x' \in \mathcal{D}}{\text{argmin}} ||x - x'||^2 \Rightarrow f(x) = y_i$$

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

***k*-NN classification** An object is classified to the class most common among its k nearest neighbors ($k \in \mathbb{N}$). The larger k the lower the influence of single noisy points on the classification. Use odd k to avoid ties in binary classification. *k*-NN is *instance based* and *lazy learning*.

Runtime

- $\mathcal{O}(n)$ to find nearest neighbour in 1-NN.
- $\mathcal{O}(n + n\log n)$ to find the *k*-NN.

Speed up *k*-NN Use the triangle inequality:

$$d(x_1, x_2) \geq d(x_1, x_3) - d(x_2, x_3)$$

If you know $d(x_1, x_3), d(x_2, x_3)$ you can provide a lower bound on $d(x_1, x_2)$. If a point is closer to x_1 than $d(x_1, x_3) - d(x_2, x_3)$ you don't need to calculate $d(x_1, x_2)$ $\Rightarrow$ *it is good to have the points ordered beforehand.*

Set the parameter *k* Use Bootstrapping and cross-validation and the trainingset with different choices of *k* and pick the one with the highest accuracy.

Weight the dimensions The *Mahalanobis distance* takes the covariance structure between features into account.

$$d_M(x, x') = \sqrt{(x - x')^\top \text{Cov}(X_i, X_j)^{-1}(x - x')}$$

With the covariance matrix defined as (X_i random variable, μ_i its mean):

$$\text{Cov}(X_i, X_j) = \mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)]$$

5 Naive Bayes

The Naive Bayes classification classifies x into one of *m* classes $y_1, ..., y_m$:

$$\arg\max_{y_i} P(Y = y_i | X = x)$$
$$= \arg\max_{y_i} \frac{P(X = x | Y = y_i) P(Y = y_i)}{P(X = x)}$$

The actual classifier We simplify by assuming:

1. $P(X = x)$ is the same for all classes ignoring the evidence (denominator)
2. *Naive* assumption: if x is multidimensional we assume that the features are conditionally independent given the class label (often not true in the life sciences)

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

Training the model

- One has to estimate $P(Y)$: typically one assumes that all classes have the same probability, or one infers the class probabilities from the class frequencies in the training dataset.
- One has to estimate $P(X|Y)$: popular choices for binary data is the Bernoulli distribution and the Normal distribution for continuous data.

Advantages

- Speed: Effort of prediction for one test point is $\mathcal{O}(md)$, as we have to compute the class posterior for all m classes.
- Ability to deal with missing data: Missing features can simply be dropped when evaluating the class posteriors
- Ability to combine discrete and continuous features: Use discrete or continuous probability distributions for each attribute
- Practical performance: Despite the unrealistic independence assumption on the features, Naive Bayes often provides good results in practice.

6 Linear Discriminant Analysis

We are in a two-class classification problem. Both classes are equally likely, i.e. their prior probabilities are equal. We assume that:

- our features, i.e their conditional probability $P(X \mid Y)$, follow a multivariate normal distribution.
- their covariance matrices are equal: instead of working with Σ_0 and Σ_1, we only have to care about a *single* matrix Σ
- two means μ_0, μ_1 that may be different

Given $y \in \{0, 1\}$, the probability density distribution for our features is thus given by:

$$f_{\mathcal{N}(\boldsymbol{\mu_y}, \Sigma)}(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}}$$
$$\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu_y})^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu_y})\right)$$

Predicting class 1 if the log-likelihood ratio is greater than 0 or, more generally, greater than some positive threshold.

$$\log \mathcal{L}(\mathbf{x}) = \log\left(\frac{P(Y = 1) P(X = \mathbf{x} \mid Y = 1)}{P(Y = 0) P(X = \mathbf{x} \mid Y = 0)}\right)$$

The denominator and numerator are parts of the right-hand side of the theorem but the evidence term is ignored:
$P(Y \mid X) = \frac{P(X|Y)P(Y)}{P(X)}$
Plugging in the normal distribution into the log likelihood ratio, we see that only the exponential terms remain because we assume that $\Sigma_0 = \Sigma_1 = \Sigma$. Hence, we are left with an expression of the following form:

$$\log\left(\frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^\top\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)\right)}{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_0)^\top\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_0)\right)}\right)$$

Since $\log\left(\frac{x}{y}\right) = \log(x) - \log(y)$, and $\log(\exp(x)) = x$, the above equation simplifies to:

$$-\frac{1}{2}((\mathbf{x}-\boldsymbol{\mu}_1)^\top\Sigma^{-1}\mathbf{x}-\boldsymbol{\mu}_1) + \frac{1}{2}((\mathbf{x}-\boldsymbol{\mu}_0)^\top\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_0))$$

Switching the term for 0 and 1 gives the desired form of the equation. This expression can be brought into a simpler form by collecting terms that depend on x and terms that do *not* depend on x.
However, this classifier suffers from computational inefficiencies in higher dimensions, because inverting Σ is very costly. ($\mathcal{O}(n^{2.373})$)

7 Logistic Regression

Logistic regression is a classification for binary output variables $y \in \{-1, 1\}$. We define a auxillary variable z, expressed as a linear function of the input x, $z = \sum_{i=1} w_i x_i + w_0$. The logistic function maps to the interval [0,1]:

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

It can be rewritten such that $f_w(x)$ is the probability that x is in class 1.

$$f_w(x) = f(\langle w, x\rangle) = \frac{1}{1+\exp\left(-(w_0 + \sum_{i=1}^d w_i x_i)\right)}$$

The inverse of the logisitc function $g = f^{-1}$, the *logit* or *log-odds* function clarifies the link to linear regression ($g \circ f_w$).

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

Training the model The log probability of each point is

$$\log\left(\frac{1}{1+\exp(-y\langle w, x_i\rangle)}\right) = -\log(1+\exp(-y\langle w, x_i\rangle))$$

To train the model we minimize the total negative log probability over all points, the *logistic loss function* which is convex in w:

$$\arg\min_{w\in R^d} \frac{1}{n}\sum_{i=1}^n \log(1+\exp(-y_i\langle w, x_i\rangle))$$

Advantages

- easy to fit: The algorithms are easy to implement and fast.
- 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.
- easy to interpret, as their output represents the log odd ratio between the positive and the negative class.
- extensions are possible

8 Decision Tree

The idea is to recursively split the data space into regions that contain a single class only. A decision tree is a flowchart like tree structure with a rootm internal nodes (attributes), branches (outcome of a test) and leaf nodes (class label). It is easy to interpret and really fast.

Classification perform test on the attributes of x at the root and follow the branches that correspond to the outcome of the test. This procedure is repeated until a leaf node is reached. The label of x is the label of this leaf node.

Construction Training Procedure

1. start with all training examples
2. select attribute and threshold that gives "best" split
3. create child node based on split
4. repeat step 2 and 3 on each child using its data until a stopping criterion is fulfilled (e.g. all examples in the same class, number of examples in node too small, tree too large)

Algorithm 4: Decision tree ($\mathcal{D}$)

```
if Stopping Criterion fulfilled then
    Predicted class for points in D is the majority class in D
end
else
    arg max_(A,θ) = cost(D) - cost(D_{A,θ})
    D_{A,<θ} = {x ∈ D | A < θ}
    D_{A,≥θ} = {x ∈ D | A ≥ θ}
    Create two child nodes of D, containing D_{A,<θ} and D_{A,≥θ}
    Decision Tree(D_{A,<θ})
    Decision Tree(D_{A,≥θ})
end
```

Quantify the Information gain The Shannon entropy of $\mathcal{D}$ is given by

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

Attribute A was used to split $\mathcal{D}$ into v subsets $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_v\}$, where D_j contains those tuples in D that have outcome a_j of A. How much more information do we need to arrive at an exact classification?

$$\text{Info}_A(\mathcal{D}) = \sum_{j=1}^{v} \frac{|\mathcal{D}_j|}{|\mathcal{D}|} \text{Info}(\mathcal{D}_j)$$

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

$$\text{Gain}(A) = \text{Info}(\mathcal{D}) - \text{Info}_A(\mathcal{D})$$

The information gain is biased towards attributes with a large number of values. The gain ratio is based on the split information

$$\text{SplitInfo}_A(\mathcal{D}) = -\sum_{j=1}^{v} \frac{|\mathcal{D}_j|}{|\mathcal{D}|} \log_2(\frac{|\mathcal{D}_j|}{|\mathcal{D}|})$$

$$\text{GainRatio}(A) = \frac{\text{Gain}(A)}{\text{SplitInfo}_A(\mathcal{D})}$$

In the Gain ratio the attribute with maximum gain ratio is selected as the splitting attribute. The ratio becomes unstable, as the split information approaches zero. 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.

Gini index The Gini index measures class impurity as $\text{Gini}(\mathcal{D}) = 1 - \sum_{i=1}^{m} p(y = y_i)^2$. If we split via attribute A into partitions $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_v\}$, the Gini index of this partitioning is defined as:

$$\text{Gini}_A(\mathcal{D}) = \sum_{j=1}^{v} \frac{|\mathcal{D}_j|}{|\mathcal{D}|} \text{Gini}(\mathcal{D}_j)$$

the reduction in impurity by a split on A is:

$$\Delta\text{Gini}(\mathcal{D}) = \text{Gini}(\mathcal{D}) - \text{Gini}_A(\mathcal{D})$$

Advantages of Decision trees

- Simple, easy to interpret, white box model, can be visualized
- little data preparation, can handle all kinds of variables
- cost of using the tree is logarithmic in the number of data points used to train the tree.
- can handle multi output problems, output can be validated by statistical tests, performs well even if assumptions are violated

Disadvantages of Decision trees

- over-complex trees that overfit data (use pruning to overcome this) and biased trees if some classes dominate (balance dataset)
- trees can be unstable because small variations in the data might result in a completely di fferent tree being generated
- learning an optimal decision tree is known to be NP-complete under several aspects of optimality
- concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems

Random Forests To minimize overfitting and the large variability in decision trees, one may use an ensemble of several decision trees.

9 Support Vector Machines

Hyperplane classifiers Given a hyperplane in some dot product space $\langle \mathbf{w}, x \rangle + b = 0$ with $\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}$ the classifier is given by:

$$f(x) = \text{sgn}(\langle \mathbf{w}, x \rangle + b)$$

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

$$\max_{\mathbf{w} \in H, b \in R} \min\{\|x - x_i\| \; | x \in \mathcal{H}, \langle \mathbf{w}, x \rangle + b = 0, i \in \{1, ..., n\}\}$$

9.1 Hard-margin SVM

$$\min_{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1 \quad \forall i \in \{1, ..., n\}$$

The size of the margin is $\frac{2}{\|\mathbf{w}\|}$. The constraint $y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1$ ensures that all training points of the same class are in the same side outside of the margin.

9.2 Soft-margin SVM

C-SVM Points are now allowed to lie inside the margin or in the wrong halfspace (margin errors):

$$\min_{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \quad \text{subject to} \quad y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1 - \xi_i \quad \forall i \in \{1, ..., n\} : \xi_i \geq 0$$

$C \in \mathbb{R}$ is a penalty score parameter that determines the tradeoff between maximizing the margin and minimizing margin errors. ξ is is a slack variable, which measures the degree of misclassification of each margin error. Can be extended such that the margin errors in different classes are penalized differently (two parameters C^+ and C^- instead of one).

υ-SVM

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

$$\text{subject to} \quad y_i(\langle\mathbf{w},x_i\rangle+b)\geq\rho-\xi_i$$
$$\forall i\in\{1,...,n\}:\xi_i\geq 0,\rho\geq 0$$

Is an alternative to C-SVM. $\upsilon\in(0,1]$ 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.

9.3 Hard-margin SVM optimization

Optimizing using the Lagrangian we get

$$\text{maximise}_{\alpha\in\mathbb{R}^n}\sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j\langle x_i,x_j\rangle$$
$$\forall i\in\{1,...,n\}_{\alpha i}\geq 0$$

Since we access the training data only in terms of the inner product $\langle x_i,x_j\rangle$ we can plug in any inner product from any space $\mathcal{H}$. This is referred to as a kernel k:

$$k(x_i,x_j)=\langle x_i,x_j\rangle_{\mathcal{H}}$$

Kernels We assume that the data points $x\in\mathbb{R}$ are mapped to a space $\mathcal{H}$ via mapping $\phi:\mathbb{R}\to\mathcal{H}$. The the kernel is defined as an inner product $\langle\cdot,\cdot\rangle:\mathcal{H}\times\mathcal{H}\to\mathbb{R}$ between two data points in $\mathcal{H}$:

$$k(x,x')=\langle\phi(x),\phi(x')\rangle$$

The decision function for *SVM* classification becomes

$$f(x)=\text{sgn}(\sum_{i=1}^{n}y_i\alpha_i\langle\phi(x),\phi(x')\rangle+b)$$
$$=\text{sgn}(\sum_{i=1}^{n}y_i\alpha_i k(x,x')+b)$$

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

10 Kernel functions

In practical applications, one constructs kernels by

- Proposing a kernel function and explicitly defining the corresponding mapping ϕ or/and
- Combining known kernels in ways that obey the closure properties of kernels.

It is crucial to know the **closure properties** of kernels:

1. given two kernels $k_1,k_2\Rightarrow k_1+k_2$ is a kernel
2. given two kernels $k_1,k_2\Rightarrow k_1\cdot k_2$ is a kernel
3. given a kernel and a positive scalar $k,\lambda\in\mathbb{R}^+\Rightarrow\lambda k$ is a kernel
4. if a kernel k is defined on a set, the zero-extension k_0 is also a kernel:

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

Kernel matrix the Gram or kernel Matrix of k is defined as an $n\times n$ matrix K

$$K_{ij}=k(x_i,x_j)$$

A matrix K is *positive semi-definite*, if for all $c_i\in\mathbb{R}$ K satisfies the following inequality:

$$\sum_{i,j}c_i c_j K_{i,j}\geq 0$$

If the Gram matrices K of the function k are all positive semi-definite, then k is a semi-definite kernel function or kernel.

10.1 Some famous and useful kernels

- linear kernel

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

- polynomial kernel

$$k(x,x')=(x^\top x'+c)^d \quad c,d\in\mathbb{R}$$

- Gaussian Radial Basis Function (RBF) kernel

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

- the constant 'all-ones' kernel

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

- the delta (Dirac) kernel

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

10.2 R-convolution kernels

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'. The most widely used instance k_R is the idea to compare all pairs of these substructures (elements of a set, nodes of a graph, substring of a string) of X and X' pairwise:

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

k_{base} is an arbitrary vectorial kernel, very often the delta kernel.

String kernels Use the spectrum kernel and k_{base} the *delta kernel* to count all pairs of matching substrings in two strings. One has to enumerate all substrings (using suffix tree can be done in linear time $\mathcal{O}(|X|+|X|)$)

Random walk kernel Counts matching walks in G and G': the more matching walks there are, the higher the similarity. Matching walks are sequences of nodes and edges with matching node labels. An elegant (but not necessarily very efficient) computation is the direct product graph of G and G' and count all walks of length k in it. Every walk in the product graph corresponds to one walk in G and G':

$$k_x(G, G') = \sum_{i,j=1}^{|V_x|} \left[\sum_{k=0}^{\infty} \lambda^k A_x^k \right]_{ij} = e^\top (1 - \lambda A_x)^{-1} e$$

Clustering

Given a set of objects, group them into clusters (classes that are unknown beforehand). It allows us to discover classes. Is is an instance of *unsupervised learning* (no training set).

11 k-means Clustering

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

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

where:

1. $\mathcal{D}$ The data set with all points x_i.
2. V($\mathcal{D}$) is the variance which needs to be minimised.
3. S_i is a cluster.
4. μ_i is the mean of cluster i. w

Algorithm 5: Lloyds' algorithm

Partition data into k initial clusters (guess)
Compute the mean μ of each cluster
Assign each point to the cluster whose mean is closest to the point
if *any point changed its cluster membership* **then**
  repeat from step 2
end

Remark on Lloyds' algorithm The algorithm is order dependent and the results depend on initialisation. Further the result may only be a local optimum.

Different implementations

1. Pairwise implementation
   (a) If one point changes, recalculate cluster (jump to point 2).
2. Stepwise implementation
   (a) Recalculate every point first. Then jump to point 2.

11.1 k-Medoid Clustering

Not the mean of each cluster is used but the medoid. The medoid is the point closest to the mean.

$$m_i = \text{argmin}_{x_j \in S_i} \|x_j - \mu_i\|^2$$

With this method one can optimise storage space. Less points need to be stored because the medoids are already data points.

11.2 Kernel k-means

Move the problem to different feature spaces. Graph and string data can be clustered. (**Note**: A Kernel is a similarity measure (inner product in a Hilbert space). It is not a distance)

Kernelising k-means

$$\begin{aligned} V(\mathcal{D}) &= \|\phi(\vec{x_1}) - \frac{1}{m-1} \sum_{j=2}^{m} \phi(\vec{x_j})\|^2 \\ &= \langle \phi(\vec{x_1}) - \vec{\mu}, \phi(\vec{x_1}) - \vec{\mu} \rangle \\ &= \langle \phi(\vec{x_1}), \phi(\vec{x_1}) \rangle + \langle \vec{\mu}, \vec{\mu} \rangle - \langle \phi(\vec{x_1}), \vec{\mu} \rangle - \langle \vec{\mu}, \phi(\vec{x_1}) \rangle \\ &= \langle \phi(\vec{x_1}), \phi(\vec{x_1}) \rangle + \langle \vec{\mu}, \vec{\mu} \rangle - 2\langle \phi(\vec{x_1}), \vec{\mu} \rangle \\ &= \kappa(\vec{x_1}, \vec{x_1}) - \frac{2}{m-1} \sum_{j=2}^{m} \kappa(\vec{x_1}, \vec{x_j}) \\ &\quad + \frac{1}{(m-1)^2} \sum_{i=2}^{m} \sum_{j=2}^{m} \kappa(\vec{x_i}, \vec{x_j}) \end{aligned}$$

The symmetry of the innerproduct was used to compute the result above.

Remark Kernel *Def:* A kernel needs to be a bilinear function that satisfies symmetry and is positive definite.

1. $k(x, y) = k(y, x)$
2. $\sum_{i,k} c_i c_j k(x_i, x_j) \geq 0$

Which holds for every c_i, $c_j \in$ and x_i, x_j in the domain of the kernel. Every kernel k induces a distance d:

$$\begin{aligned} d(x_i, x_j)^2 &= \|\phi(x_j) - \phi(x_i)\|^2 \\ &= \kappa(x_i, x_i) - 2\kappa(x_i, x_j) + \kappa(x_j, x_j) \end{aligned}$$

11.3 k-means: Silhouette Coefficients

A **silhouette coefficient** $s(x)$ relates the average distance between a point x and and all others 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 $C', d(x, \mu_{C'})$:

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

12 Graph-based Clustering

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

Graph-based clustering

1. Define a threshold θ
2. Remove all edges e_{ij} from $\mathcal{G}$ 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 ($\mathcal{O}(|V|+|E|)$)

Graph-based clustering can suffer from the fact that one noisy edge connects two clusters.

12.1 DBScan

Is short for Density-based Spatial Clustering of Applications with Noise. Two parameters have to be set MinPts $\in \mathbb{N}$ and ϵ. Two objects v_i and v_j belong th to same cluster if their distance is smaller than ϵ and either v_j or v_i is a core object. This alogrithm classifies the nodes of a graph into 3 distinct objects.

1. p is a Core point iff:
   $|N_\epsilon(p)| \geq$ MinPts
2. p is a border point iff:
   $|N_\epsilon(p)| <$ MinPts and $\exists q \in N_\epsilon(p) : |N_\epsilon(q)| \geq$ MinPts
3. p is a noise point iff:
   p is neither a core point or a border point

The clusters are defined by iteratively checking the core point property of a point p. The algorithm terminates after it won't find any new core points. DBScan is order dependent. Checking the points in different order can lead to different clustering results.

13 Spectral Clustering

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

13.1 Cut-based clustering

Objects are nodes in a graph G be a graph with nodes $\mathcal{V}$ and edges $\mathcal{E}$ and the adjacency matrix $\mathcal{W}$. Assume $\mathcal{V}$ is partitioned into k subsets: $\mathcal{V} = \{C_1, ..., C_k\}$. Cut-based clustering tries to minimize the total weight of *inter-cluster edges*:

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

where $\kappa(C_a, C_b) = \sum_{v_i \in C_a, v_j \in C_b, a \neq b} \mathcal{W}_{ij}$ and $\kappa(C_a, C_a) = 0$

Link to the graph Laplacian The degree matrix $\mathcal{D}$ is defined as:

$$\mathcal{D}_{ij} = \begin{cases} \sum_{j=1}^{n} \mathcal{W}_{ij} & i = j \\ 0 & i \neq j \end{cases}$$

The (unnormalized) Graph Laplacian is defined as $\mathcal{L} = \mathcal{D} - \mathcal{W}$. Further, let c_a be a vector of size n such that

$$c_a(i) = \begin{cases} 1 & \text{if} \quad v_i \in C_a \\ 0 & \text{if} \quad v_i \notin C_a \end{cases}$$

Note that $\langle c_a, c_a \rangle = \|c_a\|^2 = \|C_a\|$, the size of cluster C_a. Further, note that if c_a and c_b are orthogonal, $\langle c_a, c_a \rangle = 0$. It follows that finding the minimum k-cut is identical to minimizing:

$$\min \frac{1}{2} \sum_{a=1}^{k} c_a^\top \mathcal{L} c_a$$

13.2 Ratio Cut

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

$$\min_C \sum_{a=1}^{k} \frac{1}{|C_a|} \sum_{b=1}^{k} \kappa(C_a, C_b) = \min_C \sum_{a=1}^{k} \frac{c_a^\top \mathcal{L} c_a}{\|c_a\|^2}$$

However, finding the optimal $\mathcal{C}$, the binary cluster indicator vectors c_a for $a \in \{1, ..., k\}$ is NP-hard. We allow the vectors c_a to take any real value, rather than being binary. It follows that:

$$u_a^\top \mathcal{L} u_a = u_a^\top \lambda_a u_a = \lambda_a$$

where u_a are Eigenvectors of $\mathcal{L}$. This implies that in order to minimize the Objective above, one should choose the k smallest Eigenvalues of $\mathcal{L}$ and their corresponding Eigenvectors. The Eigenvectors represent the relaxed cluster indicator vectors (excluding u_n).

13.3 The Spectral Clustering algorithm

Spectral Clustering solves this problem pragmatically by using the vectors u_a as a new representation of the data points and applying k-means to this new representation after normalization. The new representation is $U = (u_n, u_{n-1}, ..., u_{n-k+1})$, a $n \times k$ matrix. It is normalized row-by-row to obtain the new k-dimensional representation: $Y = (y_1, y_2, ..., y_n)^\top$ via:

$$y_i = \frac{1}{\sqrt{\sum_{j=1}^{k} u_{i,n-j+1}^2}} (u_{i,n}, u_{i,n-1}, ..., u_{i,n-k+1})^\top$$

Algorithm 6: Spectral Clustering $(\mathcal{D}, k)$

1. Compute the similarity matrix $\mathcal{W} \in \mathbb{R}^{n \times n}$ and the Laplacian $\mathcal{L} = \mathcal{D} - \mathcal{W}$
2. Solve $\mathcal{L} u_a = \lambda_a u_a$ for $a = n, ...n - k + 1$, where $\lambda_n \leq \lambda_{n-1} \leq ...\lambda_{n-k+1}$
3. $U := (u_n, u_{n-1}, ..., u_{n-k+1})$
4. $Y :=$ normalized rows of U via Equation for y_i
5. $\mathcal{C} := \{C_1, ..., C_k\}$ via k-means on Y
6. **return**: $\mathcal{C}$

Computational Complexity The overall worst case runtime is $\mathcal{O}(n^3)$ due to the need to computer Eigenvectors and Eigenvalues. For sparse graphs with m edges, this runtime can be improved to $\mathcal{O}(mn)$. Running k-means requires a runtime in $\mathcal{O}(tnk^2)$, where t is the number of iterations of k-means until convergence.

14 Soft-assignment Clustering

Soft k-means k-means is based on a hard assignment of points to clusters. The core idea of Expectation Maximization (EM) Clustering with a Mixture of Gaussian distributions is to work with the probabilities of each point to belong to each cluster rather than a hard assignment.

14.1 EM Clustering

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). We would like to maximize $p(X|\theta)$. However, this is difficult as

$$\log p(X|\theta) = \log\left(\sum_Y p(X,Y|\theta)\right)$$

we have to sum over the latent variables inside the logarithm, which makes the evaluation of the maximum likelihood extremely challenging. The EM algorithm circumvents this problem in an iterative 2-step procedure.
Given a joint distribution $p(X,Y|\theta)$ the goal is to maximize the likelihood function $p(X|\theta)$ with respect to θ:

1. Choose an initial setting for the parameters θ^{old}
2. Expectation step (E step): Evaluate $p(Y|X,\theta^{\text{old}})$
3. Maximization step (M step): Evaluate θ^{new} given by:
$$\theta^{\text{new}} = \arg\max_\theta \mathcal{Q}(\theta,\theta^{\text{old}})$$
where $\mathcal{Q}(\theta,\theta^{\text{old}}) = \sum_Y p(Y|X,\theta^{\text{old}})\log p(X,Y|\theta)$
4. check for convergence of parameters or log likelihood. If not converged, then $\theta^{\text{old}} \rightarrow \theta^{\text{new}}$ and return to Step 2.

EM may converge to a local optimum.

Comparison to k-means

- 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 in this cluster.
- EM determines the probability that each example belongs to each cluster. Then the cluster means are updated based on a weighted sum over all data points.

15 Hierarchical Clustering

What if clusters contain clusters themselves? Then we need hierarchical clustering! Iteratively join the two most similar clusters based on similarity between clusters $s(C_i,C_j)$:

- Single link: $\min_{x\in C_i, x'\in C_j} d(x,x')$
- Average link: $\frac{1}{|C_i||C_j|}\sum_{x\in C_i,x'\in C_j} d(x,x')$
- Complete link: $\max_{x\in C_i, x'\in C_j} d(x,x')$

Algorithm 7: Hierarchical Clustering $(\mathcal{D}, s)$

Initialize each point $x_i \in \mathcal{D}$ as its own cluster C_i for $i \in \{1,...,n\}$
repeat
  $(i^*,j^*) = \arg\min_{ij} s(C_i,C_j)$
  Merge clusters C_{i^*} and C_{j^*}
until $|\mathcal{C}|= 1$;

Advantage Its clustering reflects the entire structure of the dataset.

Disadvantage (1) It is difficult to make a clear statement 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 to stop the merging of clusters.