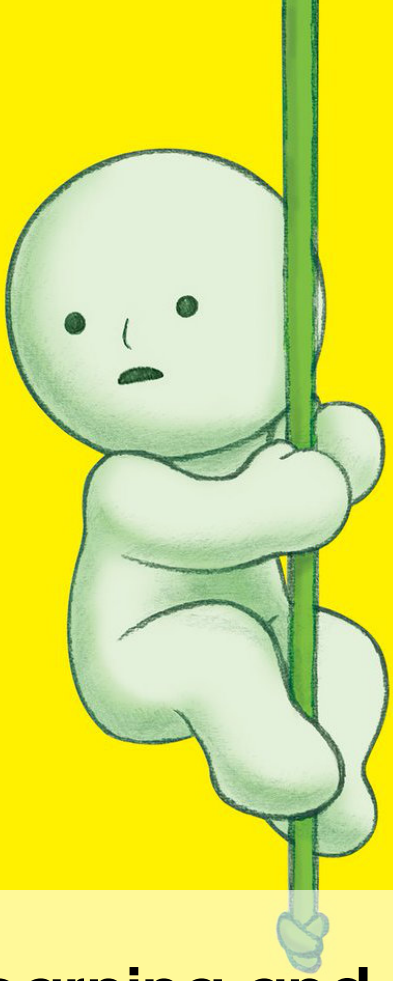


Machine Learning and Optimisation

COMP24111 Course Notes

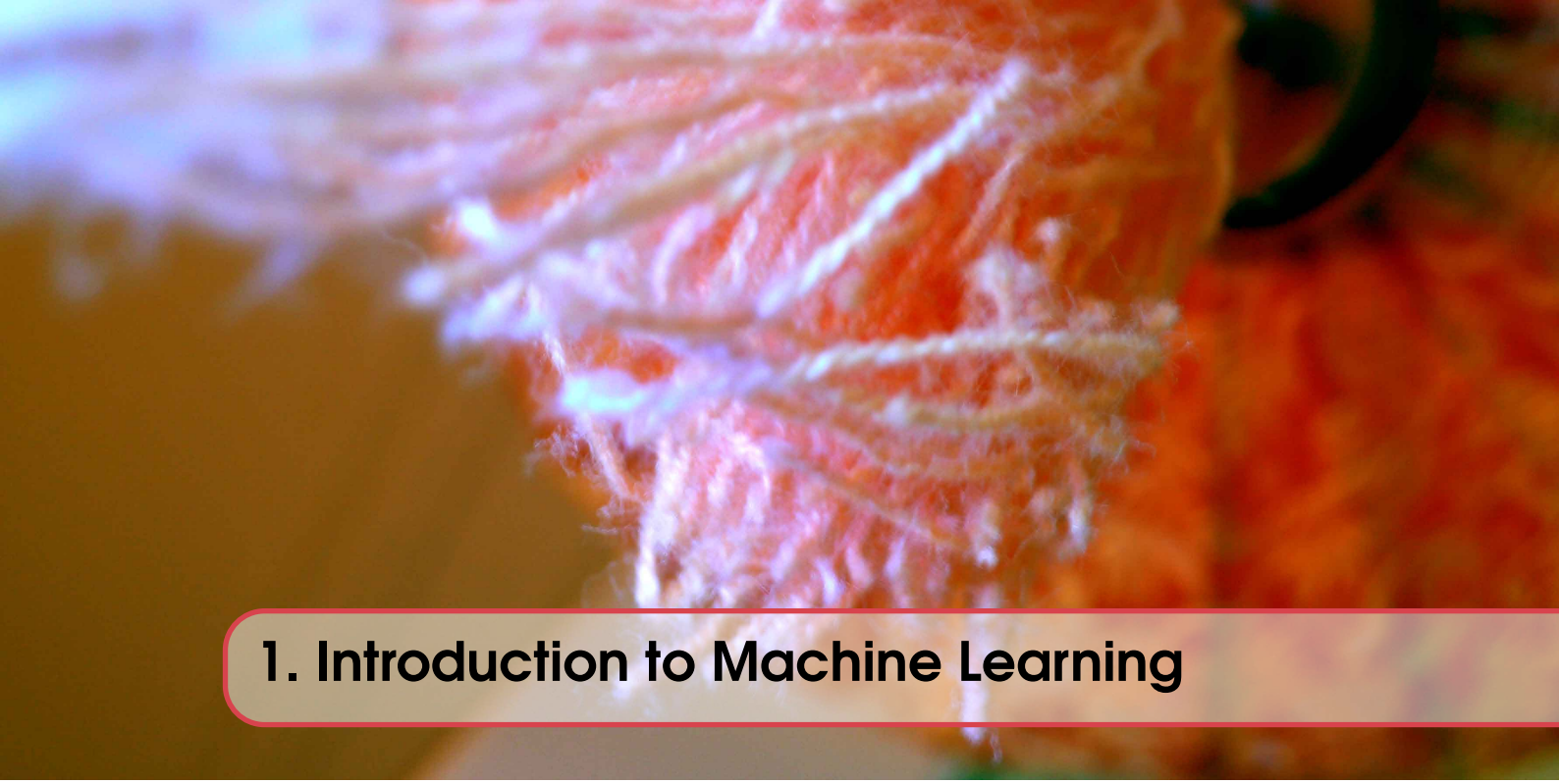
Jonathan Tang

Contributions
Chapter 8: Katie Snell (github.com/katieisnell)

jtang0506.github.io

Contents

Section I



1. Introduction to Machine Learning

1.1 Supervised learning

In **supervised learning** problems, there is an input, X, an target output, Y, provided by a *teacher* and the task is to learn the relationship between the input and the output. A training example in supervised learning is the pair (x, y) where x is the input and y is the target output. The labels provided with the training samples are know as the **ground truth**. We assume a model defined up to a set of parameters, $y = g(x \mid \theta)$ where $g(\cdot)$ is the model and θ are its parameters.

1.1.1 Classification

For a classification problem, the task of the classifier is to assign a class label to a given input. For example, a classification problem may be to decide whether an image contains a picture of a cat or a dog, or neither.

1.1.2 Regression

Suppose we want to predict the salary of a Computer Science graduate role. Inputs are the students attributes - that we believe affects a graduate's worth. The output is the predicted salary. Such problems where the output is a continuous number is known as a **regression** problem.

1.2 Unsupervised learning

Unlike in supervised learning, we do not have a supervisor and we only have input data. The task in **unsupervised learning** is to form a natural understanding of the hidden structure of unlabelled data. There is a structure to the input space such that certain patterns occur more often than others, and we want to see what generally happens and what does not.

1.3 Reinforcement learning

In reinforcement learning, there is a *teacher* who provides feedback on the action of an agent, in terms of reward and punishment.

2. k-nearest Neighbour

The **k-nearest Neighbour** estimation is one of the simplest machine learning algorithms, it is a **non-parametric method** (no parameter needs to be optimised) and requires **no explicit training**.

2.1 k-nearest Neighbour Classification

The k-nearest neighbour classifier assigns an instance to the class most heavily represented among its k neighbours. It is based on the idea that the more similar the instances, the more likely it is that they belong to the same class. We measure how similar two data points are by using a reasonable similarity or distance measure.

2.1.1 k-NN classification rule

```
let x_te be the testing point
for each training data point x_tr
    measure distance(x_te, x_tr)
end
sort distances
select k nearest points
assign most common class to x_te
```

In order to measure the distance $d(x_{te}, x_{tr})$ mentioned in the algorithm, we need to first decide on a distance measure. Euclidean distance and inner product are two measures we can used, that are defined below. An additional distance measure that we can use is cosine similarity, which is described in a later chapter.

Exercise 2.1 The hyper-parameter k should be odd and co-prime to the number of classes in the data set, why? ■

Definition 2.1.1 — Euclidean Distance. Given two d-dimensional data points, $\mathbf{p}$ and $\mathbf{q}$, where $\mathbf{p} = [p_1, p_2, \cdots, p_d]$ and $\mathbf{q} = [q_1, q_2, \cdots, q_d]$, we define the **Euclidean distance** $d(\mathbf{p}, \mathbf{q})$ as:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_d - q_d)^2} = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2}$$

In this case, the k-nearest neighbours are the training points with the lowest Euclidean distances to the testing point.

Definition 2.1.2 — Inner Product. Given two d-dimensional data points, $\mathbf{p}$ and $\mathbf{q}$, where $\mathbf{p} = [p_1, p_2, \cdots, p_d]$ and $\mathbf{q} = [q_1, q_2, \cdots, q_d]$, we define the **inner product** as:

$$s_{inner}(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{d} p_i q_i$$

Since the inner product is a similarity measure, the k-nearest neighbours are the training points possessing the highest similarity values to the testing point.

2.2 k-Nearest Neighbour Regression

For regression, we take a sample's k nearest neighbours to estimate its target output, for example, through the weighted average of these neighbours target output. This is very similar to classification but you need to take into account the distance of this sample from its nearest neighbours.

Worked Example

Since the k-NN classification algorithm is one of the simplest, you should be able to follow it from the lecture slides - hence a worked example has been omitted. It is also trivial to derive the algorithm for k-NN regression.

2.3 Neighbour number k

The neighbour number k is called the **hyper-parameter** and the processing of determining the best value of k to use is called **hyper-parameter selection** or model selection. If our choice of k is too small, we may model noise and if our choice of k is too large, neighbours will include too many samples from other classes.

The number of training samples also plays a role, a small number of training samples will lead to insufficient information about the data set. However, a large number of training samples comes with a tradeoff in time and memory cost for calculating and sorting distances.

3. Linear Classification and Regression

3.1 Linear Regression

In **linear regression**, the model formulates the prediction function as $\hat{y} = f(x) = w_0 + w_1 x$, indeed a straight line. You have come across this problem in high school before, drawing a *line of best fit* for (x, y) pairs so that you can estimate the output for some x. You probably would have been taught to draw the *ideal* line by drawing a line that has equal number of points on each side of the line - in this chapter, we will look at how we optimise this linear regression model.

We have a two parameters to optimise, namely w_0 (the **bias parameter**) and w_1. To find their optimal setting, we can compute the prediction errors and the ideal settings for the parameters would minimise the prediction error.

3.1.1 Sum-of-squares error function

A commonly used error function is the **sum-of-squares** error function:

$$O = (\hat{y_1} - y_1)^2 + (\hat{y_2} - y_2)^2 + \cdots + (\hat{y_n} - y_n)^2 = \sum_{i=1}^{n} (\hat{y_i} - y_i)^2 = \sum_{i=1}^{n} (w_1 x_i + w_0 - y_i)^2$$

It follows that our optimisation problem is to minimise $O(w_1, w_0) = \frac{1}{2} \sum_{i=1}^{n} (w_1 x_i + w_0 - y_i)^2$

Exercise 3.1 The $\frac{1}{2}$ is not a typo, why does the $\frac{1}{2}$ not change our optimisation problem? ∎

The optimal w_1 and w_0 satisfy: $\dfrac{\delta O(w_1, w_0)}{\delta w_0} = 0$ and $\dfrac{\delta O(w_1, w_0)}{\delta w_1} = 0$

We can write $O(w_1, w_0) = \dfrac{1}{2} O_i(w_1, w_0)$ where $O_i(w_1, w_0) = (w_1 x_i + w_0 - y_i)^2$

It follows that $\dfrac{\delta O_i(w_1, w_0)}{\delta w_0} = 2(w_1 x_i + w_0 - y_i)$ and $\dfrac{\delta O_i(w_1, w_0)}{\delta w_1} = 2(w_1 x_i + w_0 - y_i) x_i$

$$\frac{\delta O(w_1,w_0)}{\delta w_0} = \frac{1}{2}\sum_{i=1}^{n}\frac{\delta O_i(w_1,w_0)}{\delta w_0} = \sum_{i=1}^{n}(w_1x_i + w_0 - y_i) = w_1\sum_{i=1}^{n}x_i + nw_0 - \sum_{i=1}^{n}y_i = 0$$

$$\frac{\delta O(w_1,w_0)}{\delta w_1} = \frac{1}{2}\sum_{i=1}^{n}\frac{\delta O_i(w_1,w_0)}{\delta w_1} = \sum_{i=1}^{n}(w_1x_i + w_0 - y_i)x_i = w_1\sum_{i=1}^{n}x_i^2 + w_0\sum_{i=1}^{n}x_i - \sum_{i=1}^{n}x_iy_i = 0$$

Now from the two equations above, we have a pair of simultaneous equations with w_1 and w_0 as our unknowns. We can calculate all the summations as they are the (x,y) values in our training samples and n is simply the number of training examples. This model is known as the **linear least squares** model.

Worked Example

Consider the following data set, where we wish to produce a linear least squares model.

x	4	7	11	10	14
y	62	58	51	58	44

$$\sum_{i=1}^{5}x_i = 4+7+11+10+14 = 46 \quad \text{and} \quad \sum_{i=5}^{n}x_iy_i = 4\cdot 62+7\cdot 58+11\cdot 51+10\cdot 58+14\cdot 44 = 2411$$

$$\sum_{i=5}^{n}y_i = 62+58+51+58+44 = 273 \quad \text{and} \quad \sum_{i=1}^{5}x_i^2 = 4^2+7^2+11^2+10^2+14^2 = 482$$

Now we have the pair of simultaneous equations:

$$46w_1 + 5w_0 - 273 = 0 \quad \text{and} \quad 482w_1 + 46w_0 - 2411 = 0$$

It follows that $w_1 \approx -1.711$ and $w_0 \approx 70.34$ so we get $\hat{y} = -1.711x + 70.34$

3.1.2 Regularised least squares

Overfitting is the problem where an algorithm produced corresponds too closely to the training dataset (in particular, when it is small) so it may be unsuitable to fit additional data. Overfitting can also occur if the algorithm has modelled noise or random data among the training data. In the least squares model, we can add a **regularisation term** to the error function in attempt to control overfitting. Our optimisation problem becomes:

$$O_\lambda(w_1,w_0) = \frac{1}{2}\sum_{i=1}^{n}(w_1x_i + w_0 - y_i)^2 + \frac{\lambda}{2}(w_0^2 + w_1^2)$$

Hyperparameter λ

Notice in the regularised least squares solution, we have a λ in the regularisation term. This is a positive real-valued number, with $\lambda > 0$, set by the user. The purpose of λ is to control the balance between the data dependent error function and the regularisation term and is called a hyper-parameter so it can be distinguished from those model parameters that need to be optimised, namely w_1 and w_0.

3.2 Gradient descent

Another way we can minimise the error function is by **gradient descent**. The gradient of the error function indicates the direction in which the error *ascends* the fastest, so we can simply move across the opposite direction, so the error descends the fastest. Say we have the error function $f(w)$ and an initial value of $\boldsymbol{w} = [w_0, w_1]$, then we can use $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta f'(\boldsymbol{w}^{(t)})$ where $\eta > 0$ is the learning rate.

3.2.1 Sequential (on-line) training

In the gradient descent method described above, it would require processing of the entire training set and this can be computationally costly for large datasets. The idea of **sequential training** is to update model parameters after the presence of one training sample (or a small set of training samples). We will talk about a sequential training process called **stochastic gradient descent**.

Stochastic gradient descent (SGD)

Definition 3.2.1 — Stochastic gradient descent. A way to estimate the gradient of the error function using a single training sample.

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta(\boldsymbol{w}_1^{(t)} x_n + \boldsymbol{w}_0^{(t)} - y_n)(x_n + 1)$$

Definition 3.2.2 — Mini-batch gradient descent (MBGD). A way to estimate the gradient of the error function using a small set of training samples.

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \sum_{n \in I} (\boldsymbol{w}_1^{(t)} x_n + \boldsymbol{w}_0^{(t)} - y_n)(x_n + 1)$$

where I denotes a small set of training samples.

4. Logistic Regression

4.1 Linear basis function model

Suppose the estimated output variable is a linear combination of fixed nonlinear functions (known as basis functions) of the input. For example $\hat{y} = w_0 + w_1 x + w_2 x^2 + w_3 x^3$ is a linear combination of $1, x, x^2$ and x^3 with coefficients w_0, w_1, w_2, w_3 respectively. The purpose of incorporating a **linear basis function model** is so that it is suitable for non-linear classification and regression.

$$\hat{y} = w_0 + w_1\phi_1(\boldsymbol{x}) + w_2\phi_2(\boldsymbol{x}) + \cdots + w_D\phi_D(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x})$$

$$\text{where} \quad \boldsymbol{w} = [w_0, w_1, \cdots, w_D]^T \quad \text{and} \quad \boldsymbol{\phi}(\boldsymbol{x}) = [1, \phi_1(x), \phi_2(x), \cdots, \phi_D(x)]^T$$

4.1.1 Polynomial regression

For a single input variable, if we take $\boldsymbol{\phi}(\boldsymbol{x}) = [1, x, x^2, \cdots, x^D]^T$, our linear basis function model is

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + \cdots + w_D x^D$$

This is known as polynomial regression, and in the case $D = 1$, we get linear regression.

Exercise 4.1 It is typical to set $\phi_0(x) = 1$, why is that? ■

5. Support Vector Machine

5.1 Kernel trick

5.2 Model Validation

5.2.1 Holdout method

With the **holdout method**, the whole dataset is split into two groups, a training set and a testing set. The model is trained using the training set and asked to predict output values for the data in the testing set, which it has never seen before. This evaluation can have a high variance as we may get an unfortunate split as it may depend heavily on which data points end up in the training set and which end up in the testing set. Another drawback is that we may only have a small dataset, so we may not afford to set aside a portion of the dataset for testing.

5.2.2 Random subsampling

In each split of **random subsampling**, we select a fixed number of samples for testing and the remaining samples are used for training. For each data split, the classifier is training from scratch and its error rate is estimated with the testing examples.

5.2.3 k-fold cross validation

k-fold cross validation is one way to improve over the holdout method, the dataset is divided into k partitions and the holdout method is performed k times. Each time, one of the k subsets is used as the testing set and the remaining $k-1$ subsets are combined and used as the training set. We evaluate the error estimate as the average error across the k trials. The advantage of this method is that it matters less how the dataset gets divided and all the examples in the dataset are eventually used for both training and testing, therefore the variance of the resulting estimate is reduced as k is increased. However, the training has to be done k times, which means this method gets expensive to compute as k is increased.

5.2.4 Leave-one-out cross validation

Leave-one-out cross validation is the degenerate case for k-fold cross validation, with $k = N$, the number of data points in the dataset. As before, the average error is computed and used to evaluate the model. The evaluation given by leave-one-out cross validation error is good, but it is very expensive to compute compared to other methods.

5.2.5 Bootstrap

With **bootstrap**, on each iteration, you randomly select, with replacement, a fixed number of examples to use for training. The remaining examples that were not selected are used for testing - this means that the number of testing samples can change over repeats, as training samples are selected with replacement.

Model Selection (hyper-parameter selection)

To select the best model, we run the algorithm with different hyper-parameters, which yields different models and use one of the above validation schemes to calculate an error estimate. It follows that the best model is the one that yields the lowest error estimate.

5.3 Confusion matrix

A confusion matrix is a table with two rows and two columns which allows further analysis rather than just the proportion of correct classifications. For example, if we had a dataset containing 98 examples from Class 1 and 2 examples from Class 2, there may be a classifier which classifies all the observations as Class 1, which yields a overall accuracy of 98%. However, accuracy is not a reliable metric for this classifier as the classifier would have 100% recognition rate for Class 1 but 0% recognition rate for Class 2.

Definition 5.3.1 — True Positives (TP). Both the predicted class and actual class is 'Yes'.

Definition 5.3.2 — True Negatives (TN). Both the predicted class and actual class is 'No'.

Definition 5.3.3 — False Positives (FP). The predicted class is 'Yes' and actual class is 'No'.

Definition 5.3.4 — False Negatives (FN). The predicted class is 'No' and actual class is 'Yes'.

Worked Example

Suppose that we predicted the presence of a disease that a patient and obtain the results below.

n = 165		Predicted Class	
		No	Yes
Actual Class	No	50	10
	Yes	5	100

■ **Example 5.1** For the above example, **TP** = 100, **TN** = 50, **FP** = 10 and **FN** = 5. ■

Some questions we may want to ask about our predictions are:

1. When the patient actually has the disease, how often did we predict this?
2. When the patient does not have the disease, how often did we predict this?
3. When we predict that a patient has a disease, how often are we correct?

First, we define some terms which directly relate to the questions above.

Definition 5.3.5 — Sensitivity / Recall. The proportion of positives that are correctly classified.

$$\frac{\text{True Positives}}{\text{Number of Actual Positives}} = \frac{\text{TP}}{\text{TP+FN}}$$

Definition 5.3.6 — Specificity. The proportion of negatives that are correctly classified.

$$\frac{\text{True Negatives}}{\text{Number of Actual Negatives}} = \frac{\text{TN}}{\text{TN+FP}}$$

Definition 5.3.7 — Precision. The proportion of predicted positives which were positive.

$$\frac{\text{True Positives}}{\text{Number of Predicted Positives}} = \frac{\text{TP}}{\text{TP+FP}}$$

■ **Example 5.2** By definition, the answers to the above questions are sensitivity, specificity and precision respectively. ■

Exercise 5.1 Calculate the sensitivity, specificity and precision for the confusion matrix given above. ■

Definition 5.3.8 — F1-score. The harmonic mean of precision and recall.

$$2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

6. Deep Learning Models

Section II

7. Generative Models and Naive Bayes

Introduction

In machine learning, naive Bayes classifiers are a group of **probabilistic classifiers** which are based on using Bayes' theorem assuming, naivly, that there is a **strong independence** between the features. Naive Bayes are probabilistic, meaning that they calculate the probability of each category for a given sample, and then output the category with the highest probability.

Naive Bayes is popular method for determining the category of text, with word frequencies as the features. Some examples include:

1. Determining what category an email belongs to, i.e. spam or legitimate, fashion or sports
2. Digit recognition
3. Face recognition

Background

There are three methodologies to design a model:

1. Model a classification rule directly, for example k-NN, linear classifier, SVM, neural networks
2. Model the probability of class memberships given input data, for example logistic regression, probabilistic neural networks
3. Make a probabilistic model of data within each class, for example naive Bayes, model-based

This brings up an important way of distinguishing these models based on what we want to achieve with the data we have been given; probabilistic models vs non-probabilistic models and discriminative models vs generative models.

The table shows us the difference between the learning models (classifiers) more clearly.

	Probabilistic	**Non-Probabilistic**
Discriminative	• Logistic regression • Probabilistic neural networks	• k-NN • Linear classifier • SVM • Neural networks
Generative	• Naive Bayes • Model-based (e.g. GMM)	N.A.

7.1 Probability basics

We will define some basic probability notation that will help us to understand how naive Bayes works.

Definition 7.1.1 — Prior probability. The prior probabilities are the probabilities of an example from the training set belonging to each class:

$$P(c_i) = \frac{\text{number of examples that belong to class } i}{\text{total number of examples}} \tag{7.1}$$

Definition 7.1.2 — Conditional Probability. Conditional probability is the probability of x_1 happening if x_2 happens too. We use the notation $P(x_1 \,|\, x_2)$.

These definitions of probabilities will now be clear with an example of using naive Bayes for discrete-valued features.

7.2 Naive Bayes for discrete-valued features

Suppose we want to build our classifier to say **whether a piece of text is about sports or not**. Our training set has 5 sentences:

- "A great game" - Sports
- "The election was over" - Not sports
- "Very clean match" - Sports
- "A clean but forgettable game" - Sports
- "It was a close election" - Not sports

Lets say we want to classify "A very close game". As naive Bayes is a **probabilistic classifier**, we want to calculate the probability that the sentence "A very close game" is Sports and also the probability that it is Not sports.

Once we have done this, we take the largest probability of the two which determines what category our sentence has been classified as (we want this to be Sports in this case). This is an application of the **"Maximum A Posterior"**, or MAP, classification rule.

Definition 7.2.1 — MAP Rule. The MAP rule determines that for an input x, find the largest probability from L probabilities that are outputted by a discriminative probabilistic classifier $P(c_1 \,|\, x), \cdots, P(c_L, \,|\, x)$. Then we assign x to label c^* if $P(c^* \,|\, x)$ is the largest.

In mathematical terms, we want $P(\text{Sports} \,|\, \text{a very close game}) > P(\text{Not sports} \,|\, \text{a very close game})$.

Before we start, we need to decide what we shall use as features. In this case, we are comparing words so we will use **word frequencies**. This means our features will be the counts of each of the words.

Now we look at the **Bayesian rule** (or Baye's Theorem) which is useful when working with conditional probabilities because it provides a way to "reverse" them. This is exactly what we want.

Definition 7.2.2 — Bayesian Rule. We define the Bayesian rule as follows:

$$P(c_i \,|\, \mathbf{x}) = \frac{P(\mathbf{x} \,|\, c_i) \cdot P(c_i)}{P(\mathbf{x})}, \; i = 1, 2, \cdots, L \tag{7.2}$$

In our example, we want to work out $P(\text{Sports} \,|\, \text{a very close game})$, so using the Bayesian rule this gives us:

$$P(\text{Sports} \,|\, \text{a very close game}) = \frac{P(\text{a very close game} \,|\, \text{Sports}) \cdot P(\text{Sports})}{P(\text{a very close game})}$$

Note that in our case, we just want to find out which category (Sports or Not sports) has a bigger probability, so we can ignore the divisor, as it is the same for both categories, and just compare:

$$P(\text{a very close game} \,|\, \text{Sports}) \cdot P(\text{Sports}) \quad \text{and} \quad P(\text{a very close game} \,|\, \text{Not sports}) \cdot P(\text{Not sports})$$

We can calculate these probabilities easily.

Now we count how many times the sentence "A very close game" appears in the Sports category in our training set, and divide it by the total which gives us $P(\text{a very close game} \,|\, \text{Sports})$.

However, "A very close game" does not appear as a whole sentence in our training set so we will get a probability of 0. This means the model is not useful unless every sentence we want to classify appears in our training set, which defeats the point of using it.

So to overcome this issue we make a strong **naive** assumption that every word in the sentence is **independent** of the others in the sentence.

So this means now that "A very close game" is the same as "very game close a". We write this as:

$$P(\text{a very close game}) = P(\text{a}) \cdot P(\text{very}) \cdot P(\text{close}) \cdot P(\text{game})$$

So now we can apply this to the conditional probability we were working with before as so:

$$P(\text{a very close game} \,|\, \text{Sports}) = P(\text{a} \mid \text{Sports}) \cdot P(\text{very} \mid \text{Sports}) \cdot P(\text{close} \mid \text{Sports}) \cdot P(\text{game} \mid \text{Sports})$$

So now we can do some calculations. We first work out P(Sports) and P(Not sports), which is simply the prior probabilities.

So in our example, $P(\text{Sports}) = 3/5$ and $P(\text{Not sports}) = 2/5$.

Next we need to calculate each of the conditional probabilities of each word appearing in a class. So we start by calculating $P(\text{game}|\text{Sports})$ which is the number of times the word "game" appears in Sports examples in the training set, which is 2, divided by the total number of words in the Sports examples, which is 11. Therefore, $P(\text{game}|\text{Sports}) = 2/11$.

However, we have a problem when we work out the probability of $P(\text{close}|\text{Sports})$ as "close" does not appear in any of our Sports examples. This means the probability is 0. This causes a **zero conditional probability problem** when we work out the probability of all the words combined as this 0 will make us end up with a zero probability.

In order to overcome this, we apply Laplace smoothing.

Definition 7.2.3 — Laplace smoothing. Laplace smoothing is the process of adding 1 to every count so we do not have any counts equalling zero.

We adjust our divisor to also add the number of possible words in the whole training set, in this example all possible words are ['a', 'great', 'very', 'over', 'it', 'but', 'game', 'election', 'clean', 'close', 'the', 'was', 'forgettable', 'match'], which is 14.

After we apply smoothing, our probabilities are as follows:

Word	P(Word | Sports)	P(Word | Not sports)
a	2 + 1 / 11 + 14	1 + 1 / 9 + 14
very	1 + 1 / 11 + 14	0 + 1 / 9 + 14
close	0 + 1 / 11 + 14	1 + 1 / 9 + 14
game	2 + 1 / 11 + 14	0 + 1 / 9 + 14

The final step is to multiple all our non-zero probabilities and see whether Sports or Not sports are bigger, and do not forgot to multiple by the prior probability too.

$$P(\text{a}|\text{Sports}) \cdot P(\text{very}|\text{Sports}) \cdot P(\text{close}|\text{Sports}) \cdot P(\text{game}|\text{Sports}) \cdot P(\text{Sports}) = 2.76 \cdot 10^{-5} = 0.0000276$$

Similarly we have $P(\text{a}|\text{Not sports}) = 0.572 \cdot 10^{-5} = 0.00000572$.

So we have classified "A very close game" to be in the Sports category using the naive Bayes classifier for discrete-valued features.

7.3 Naive Bayes for continuous-valued features

When we want to model features which are continuous in nature, such as temperature, then our discrete method will not be appropriate.

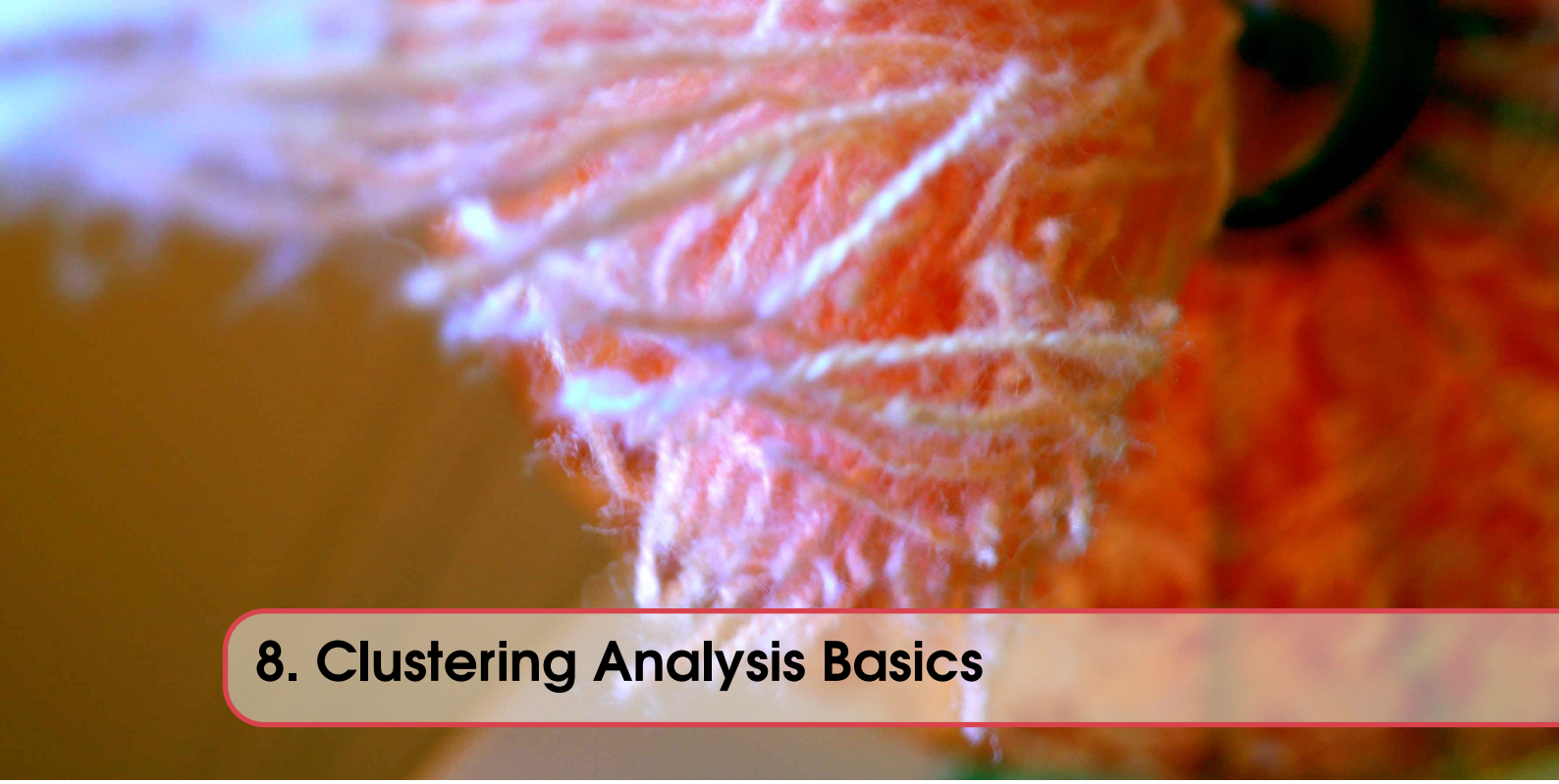
However, the prior probabilities can still be calculated in the same way as the example above.

Instead, we **estimate** the **mean** and **variance** for each class and we can use this with the assumption that the attributes are subject to a **normal distribution** (or Gaussian).

Now when we are in the test phase, we can calculate the conditional probabilities modelled with the Gaussian distribution using the equation in our original Bayesian rule above.

$$P(x_j|c_i) = \frac{1}{\sigma_{ji}\sqrt{2\pi}} e^{-\left(x_j-\mu_{ji}\right)^2 \Big/ 2\sigma_{ji}^2} \tag{7.3}$$

Note that when $\sigma = 0$, we cannot use the equation but instead we realise that this means that there is **no spread of data** hence the attribute has only one value. Therefore, this means the probability of getting that attribute is 1. This means we only need to the equation when $\sigma > 0$.



8. Clustering Analysis Basics

Introduction

Clustering analysis is the process of finding similarities between data according to the characteristics underlying the data and grouping similar data objects into clusters. A **cluster** is a group of data points which are similar to one another within the same group and dissimilar to the points in other groups. Clustering analysis is **unsupervised** as there are no predefined classes for a training data set. The main approach of clustering analysis is to maximise the intra-cluster similarity and minimise the inter-cluster similarity. Typically, clustering analysis is either used as a stand-alone tool to gain an insight into data distribution or used as a preprocessing step of other algorithms in intelligent systems.

The two general tasks of clustering analysis are:

1. Identifying the "natural" number of clusters present in a data set
2. Properly grouping data points into "sensible" clusters

Real Life Applications

- **Marketing** - companies can discover distinct groups in their customer bases and then use this knowledge to develop targeted marketing programs
- **Social network mining** - discovering communities of similar interests in a large group of people
- **Image segmentation** - dividing an image into distinct regions for object recognition

8.1 Data Representation

A data matrix is a $n \times p$ matrix which represents n data points with p dimensions. The data matrix has *two* modes as rows and columns represent different entities.

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

A distance / dissimilarity matrix is a $n \times n$ matrix which represents the distance between each data point. The distance matrix is a symmetric triangular matrix as $\mathbf{d}(x,y) = \mathbf{d}(y,x)$ where x, y are two data points. The distance matrix has *one* mode as the rows and column represent the distance for the same entity.

$$\begin{bmatrix} 0 & & & & \\ d(x_2,x_1) & 0 & & & \\ d(x_3,x_1) & d(x_3,x_2) & 0 & & \\ \vdots & \vdots & \vdots & \ddots & \\ d(x_n,x_1) & d(x_n,x_2) & d(x_n,x_3) & \cdots & 0 \end{bmatrix}$$

8.2 Distance Measures

The **Minkowski distance** is a metric in a normed vector space which can be considered as the generalisation of both the Euclidean distance and the Manhattan distance.

Definition 8.2.1 — Minkowski Distance. The Minkowski distance of order p between two points $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ and $\mathbf{y} = (y_1, y_2, \cdots, y_n)$ is defined as:

$$d(\mathbf{p},\mathbf{q}) = \left(|x_1 - y_1|^p + |x_2 - y_2|^p + \cdots + |x_n - y_n|^p \right)^{\frac{1}{p}}, p > 0$$

The value of p in the Minkowski Distance would be selected depending on the application. The **Manhattan** and **Euclidean** distances are simply Minkowski distances of order 1 and 2 respectively.

Definition 8.2.2 — Manhattan Distance.

$$d(\mathbf{p},\mathbf{q}) = |x_1 - y_1| + |x_2 - y_2| + \cdots + |x_n - y_n|$$

This is also known as the city block distance, the reason becomes clear in the examples below.

Definition 8.2.3 — Euclidean Distance.

$$d(\mathbf{p},\mathbf{q}) = \sqrt{|x_1 - y_1|^2 + |x_2 - y_2|^2 + \cdots + |x_n - y_n|^2}$$

This is the distance that you are already familiar with, in n dimensions.

Exercise 8.1 When is the Manhattan distance between x and y equal to the Euclidean distance between x and y, where x and y are data points of two dimensions? ■

Definition 8.2.4 — Cosine measure. For $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ and $\mathbf{y} = (y_1, y_2, \cdots, y_n)$

$$\cos(\mathbf{x},\mathbf{y}) = \frac{x_1 y_1 + \cdots + x_n y_n}{\sqrt{x_1^2 + \cdots + x_n^2}\sqrt{y_1^2 + \cdots + y_n^2}} \quad \text{and} \quad d(\mathbf{x},\mathbf{y}) = 1 - \cos(\mathbf{x},\mathbf{y})$$

Note that $-1 \leq \cos(\mathbf{x},\mathbf{y}) \leq 1$ and $0 \leq d(\mathbf{x},\mathbf{y}) \leq 2$

Worked Example

Suppose we have some documents that we want to compare to see how similar texts are, without taking into account the order of the words. For this example, we will consider:

(a) Labeeba loves me more than Hani loves me
(b) Hani loves Labeeba more than Labeeba loves Hani
(c) Labeeba likes Hani more than Hani likes Labeeba

First, we make a list of all the words that appear across all texts.

[Hani, Labeeba, likes, loves, me, more, than]

Next, we construct a vector for each document, with the frequency of each word.

a = [1, 1, 0, 2, 2, 1, 1]
b = [2, 2, 0, 2, 0, 1, 1]
c = [2, 2, 2, 0, 0, 1, 1]

Now we can compute the **cosine measures**: cos(**a**, **b**), cos(**a**, **c**) and cos(**b**, **c**).

$$\cos(\mathbf{a},\mathbf{b}) = \frac{(1)(1)+(1)(2)+(0)(0)+(2)(2)+(2)(0)+(1)(1)+(1)(1)}{\sqrt{1^2+1^2+0^2+2^2+2^2+1^2+1^2}\sqrt{2^2+2^2+0^2+2^2+0^2+1^2+1^2}} \approx 0.694$$

By similar calculations, we find that $\cos(\mathbf{a},\mathbf{c}) \approx 0.463$ and $\cos(\mathbf{b},\mathbf{c}) \approx 0.714$. This tells us that sentences **b** and **c** are the most similar pair out the above texts, as they have the highest cosine measure. This is also the reason $d(\mathbf{x},\mathbf{y})$ is defined as $1-\cos(\mathbf{x},\mathbf{y})$ since a larger cosine measure would be converted into a smaller distance.

8.3 Distance for binary features

For binary features, their values can be converted to 1 or 0, then we calculate the contingency table.

		y	
		1	0
x	1	a	b
	0	c	d

Symmetric binary features

Binary features are **symmetric** if both of their states equally valuable and carry the same weight; i.e. no preference on which outcome should be coded as 1 or 0 , e.g. gender.

$$d(\mathbf{x},\mathbf{y}) = \frac{b+c}{a+b+c+d}$$

Asymmetric binary features

Binary features are **asymmetric** if the outcomes of the states not equally important, e.g. the positive and negative outcomes of a disease test; the rarest one is set to 1 and the other is 0. Since the number of negative matches are considered unimportant, they are omitted from the calculations.

$$d(\mathbf{x},\mathbf{y}) = \frac{b+c}{a+b+c}$$

8.4 Distance for nominal features

Nominal features are those that can take more than two states, for example *small, medium, large*. There are two methods to handle variables with nominal features; simple mis-matching and converting them into binary variables.

Simple mis-matching

$$d(\mathbf{x},\mathbf{y}) = \frac{\text{number of mis-matching features between } \mathbf{x} \text{ and } \mathbf{y}}{\text{total number of features}}$$

Converting them into binary variables

We create new binary features for all of its nominal states, for example if our size feature takes three possible nominal states *small, medium, large* then this feature will be expanded into three binary features. *small, medium, large* = 100, 010, 001, then we can use the distance measures for binary features.

Worked Example

Consider the example below where we have two toys, T_1 and T_2, whose features are size, colour and price range.

	Size	Colour	Price Range
T_1	010	100	01
T_2	100	001	01

Simple mis-matching

Only the price range takes the same value for the features between T_1 and T_2 hence

$$d(\mathbf{T_1},\mathbf{T_2}) = \frac{2}{3} \approx 0.66$$

Converting them into binary values

Size: [Small, Medium, Large] = [100, 010, 000]
Colour: [Green, Red, Yellow] = [100, 010, 000]
Price: [Cheap, Expensive] = [10, 01]

Now we do T_1 **XOR** T_2 = 01010001 **XOR** 10000101 = 11010100. Now from here we can get the distance by dividing the number of 1 bits in T_1 **XOR** T_2 by the total number of bits in T_1.

$$d(\mathbf{T_1},\mathbf{T_2}) = \frac{4}{8} = 0.5$$

8.5 Clustering methodologies

- **Partitioning**: construct various partitions and then evaluate them by some criterion, e.g. minimising the sum of squares distance cost
- **Hierarchical**: create a hierarchical decomposition of the set of data using some criterion
- **Density-based**: based on connectivity and density functions
- Model-based: a generative model is hypothesised for each of the clusters and tries to find the best fir of that model to each other
- **Spectral clustering**: convert data set into a weighted graph then cut the graphs into sub-graphs corresponding to clusters via spectral analysis
- **Clustering ensemble**: combine multiple clustering results

9. k-means Clustering

Introduction

The k-means algorithm is the simplest partitioning method for clustering analysis and is widely used in data mining algorithms. It is a **heuristic method** and is based on each clustering being represented by the centre of the cluster and the algorithm will converge to stable centroids of clusters. The goal of this algorithm is find a partition of k clusters to optimise the chosen partition criterion - the global optimum is achieved by exhaustively searching all partitions.

9.1 Partitioning clustering approach

A partitioning clustering approach is done by iteratively partitioning the training data set to learn a partition of the given data space to product several non-empty clusters, where the number of clusters is usually given in advance. In principle, an optimal partition is achieved by minimising the sum of the squared distance to its "representative object" in each cluster.

Exercise 9.1 Why do we have to used the squared distance? ■

9.2 k-means algorithm

Before we can use the k-means algorithm, we must decide the cluster number k and an appropriate distance measure that will be used. We initialise the algorithm by selecting k distinct seed points, randomly.

1. assign each object to the cluster of the nearest point
2. compute new seed points as the centroids of the clusters of the current partition

The two steps are repeated until it converges (membership in each cluster no longer changes). At this point, we have k centroids which partition the whole data space into k mutually exclusive subspaces to form a partition.

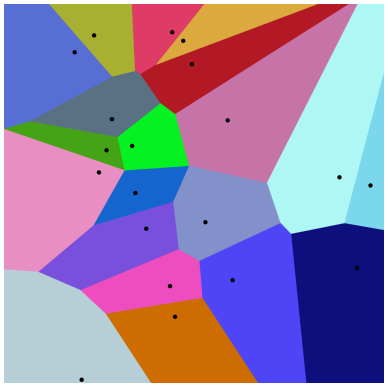


Figure 9.1: Voronoi Diagram

The figure above shows how a data space can be partitioned, where each black dot represents a centroid of a cluster. Notice that the distance used above is Euclidean distance.

9.3 Issues

- This algorithm has computational complexity of $O(\text{tKn})$, where n is the number of data points, K is the number of clusters and t is the number of iterations. Normally, $K, t \ll n$.
- It is sensitive to initialisation, which may result to unwanted solutions
- Is unable to handle noisy data and outliers, which results in an inaccurate partition
- Requires prior knowledge of the cluster number
- Incapable of handling clusters of non-convex shape
- Inapplicable to categorical data, since the mean is not defined
- How do we evaluate the k-mean performance?

Example

The k-Means algorithm is very simple, so an example has be omitted. An example may be added in the future, if I have spare time - but for now, just check the lecture slides.

10. Hierarchical and Ensemble Clustering

10.1 Hierarchical clustering approach

Hierarchical clustering approach is done by sequentially partitioning the data set to construct nested partitions layer by layer, via grouping objects into a tree of clusters. We use a generalised distance matrix as the clustering criteria and we do not need to know the number of clusters in advance.

Approaches to hierarchical clustering

There are two main approaches to hierarchical clustering: **agglomerative clustering** (bottom-up) and **divisive clustering** (top-down). In agglomerative clustering, each data point is initially its own atomic cluster and we merge clusters into larger and larger clusters. Divisive clustering, as the name suggests, is where all data points belong to a single cluster initially, then the cluster is divided into smaller and smaller clusters.

10.2 Cluster distance measures

Definition 10.2.1 — Single link. The **smallest** distance between an element in one cluster and an element in another, i.e. $d(C_i, C_j) = \min\{d(x_{ip}, x_{jq})\}$

Definition 10.2.2 — Complete link. The **largest** distance between an element in one cluster and an element in another, i.e. $d(C_i, C_j) = \max\{d(x_{ip}, x_{jq})\}$

Definition 10.2.3 — Average. The **average** distance between elements in one cluster and elements in the other, i.e. $d(C_i, C_j) = \text{avg}\{d(x_{ip}, x_{jq})\}$

10.3 Agglomerative clustering

Algorithm

1. Select a cluster distance measure to use
2. Convert all object features into a distance matrix
3. Set each object as an atomic cluster (N objects means N clusters at start)
4. Repeat until number of clusters is one (or known # of clusters)
    - Merge two closest clusters
    - Update distance matrix

Example

Again, very simple algorithm - so an example has been omitted as the lecture example is sufficient.

Definition 10.3.1 — Lifetime. The distance between that a cluster is created and that it disappears (merges with other clusters during clustering).

Definition 10.3.2 — k-cluster lifetime. The distance from that K clusters emerge to that K clusters vanish (due to the reduction to $K-1$ clusters).

Relevant Issues

- How do we determine the number of clusters?
    - If the number of clusters is known, then we have a termination condition
    - We can use the **K-cluster lifetime** as a range of threshold values on the dendrogram that leads to the identification of K clusters
    - Heuristic method: cut the dendrogram tree with maximum life time to find a "proper" K
- Major weakness of agglomerative clustering methods
    - Can never undo what was done previously...
    - Sensitive to cluster distance measures, noise and outliers
    - Efficiency of $O(n^2 \log n)$, where n is the number of total objects

10.4 Ensemble Clustering

Ensemble clustering is concerned with the issues that may affect a single clustering algorithm, for example, sensitivity to initialisation, noise, outliers, distance metrics and it may be hard to choose a single algorithm which can handle all types of cluster shapes and sizes. The purpose of ensemble clustering is to utilise the results obtained by multiple clustering analyses for robustness.

Algorithm summary

1. Perform clustering analysis by using either different clustering algorithms or running a single clustering algorithm of different conditions, **leading to multiple partitions**
2. Convert clustering results on different partitions into **binary distance measure**
3. Evidence accumulation: form a **collective distance matrix** based on all the distance matrices
4. Apply a hierarchical clustering algorithm (with a proper cluster distance measure) to collective distance matrix and **use the maximum k-cluster lifetime to decide**

11. Cluster Validation

Introduction

Cluster validation refers to the procedures that evaluate the results of clustering in a *quantitive* and *objective* fashion. In cluster validation, we need to compare clustering algorithms, solve the problem of determining the number of clusters, avoid finding patterns in noise and finding the "best" clusters from the data.

We have two types of criteria, internal and external. An **internal criteria** (index) is used to validate without external information and can be applied to find the "proper" number of clusters in a data set. An **external criteria** (index) may be applied to evaluate the performance of a clustering algorithm against data sets where the ground truth is available, by comparing how similar the partition generated by the clustering algorithm is to the ground truth.

11.1 Internal Index

The internal index we will focus on is the **F-ratio index**, which is a variance based internal index.

Variance-based methods

In **variance-based methods**, the goal is to minimise the intra-cluster variance and to maximise the inter-cluster variance. Assume an algorithm leads to a partition of K clusters where cluster i has n_i data points, c_i is its centroid and $d(\cdot,\cdot)$ is a chosen distance measure.

$$\text{Intra-cluster variance} = SSW(K) = \sum_{i=1}^{K} \sum_{j=1}^{n_i} d^2(\mathbf{x}_{ij}, \mathbf{c}_i)$$

$$\text{Inter-cluster variance} = SSB(K) = \sum_{i=1}^{K} n_i \cdot d^2(\mathbf{c}_i, \mathbf{c})$$

where $\mathbf{c}$ is the global centroid of the whole data set

11.1.1 F-ratio index

The **F-ratio index** is a measure of the inter-cluster variance against the intra-cluster variance. We calculate the F-ratio index for a partition of K clusters as follows.

$$F(K) = \frac{K \cdot SSW(K)}{SSB(K)} = \frac{K \cdot \sum_{i=1}^{K} \sum_{j=1}^{n_i} d^2(\mathbf{x}_{ij}, \mathbf{c}_i)}{\sum_{i=1}^{K} n_i \cdot d^2(\mathbf{c}_i, \mathbf{c})}$$

It is ideal to have a low F-ratio index, so we run clustering algorithms for a range of values of K (the number of clusters) and calculate a F-ratio index for each value of K and each algorithm, then select the algorithm and value of K which yields the lowest F-ratio index.

11.2 External Index

The external index we will focus on is the Rand Index, 1971.

11.2.1 Issues with external indexes

- **cluster-ID permutation**: the cluster IDs in a partition from clustering have been assigned arbitrarily due to unsupervised learning
- **inconsistence in cluster numbers**: the number off clusters from algorithm may be different from the number of classes (in ground truth)
- **point-pair correspondence**: how do we find all possible correspondences between the ground truth and a partition?

11.2.2 Rand Index

The idea behind **Rand Index** is to consider all pairs in the data set by looking into both *agreement* and *disagreement* against the ground truth.

X/Y	Pairs in the same class	Pairs in different classes
Pairs in the same cluster	a	b
Pairs in different clusters	c	d

Definition 11.2.1 — Rand Index. Let $\mathbf{X}$ be the partition from clustering and $\mathbf{Y}$ be the ground truth. The Rand Index is defined as

$$\mathrm{RI}(\mathbf{X}, \mathbf{Y}) = \frac{a+d}{a+b+c+d}$$

To calculate a, b, c, d, we can use a contingency table as follows. k denotes the number of clusters in $\mathbf{X}$, l denotes the number of classes in $\mathbf{Y}$ and each n_{ij} denotes the number of points in both cluster i and class j.

$$\begin{array}{cccc|c} n_{11} & n_{12} & \cdots & n_{1l} & n_{1.} \\ n_{21} & n_{22} & \cdots & n_{2l} & n_{2.} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ n_{k1} & n_{k2} & \cdots & n_{kl} & n_{k.} \\ \hline n_{.1} & n_{.2} & \cdots & n_{.l} & N \end{array}$$

$$a = \frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{l} n_{ij}(n_{ij}-1) \qquad c = \frac{1}{2}(\sum_{i=1}^{k} n_{i.} - \sum_{i=1}^{k}\sum_{j=1}^{l} n_{ij}^2)$$

$$b = \frac{1}{2}(\sum_{j=1}^{l} n_{.j} - \sum_{i=1}^{k}\sum_{j=1}^{l} n_{ij}^2) \qquad d = \frac{1}{2}(N^2 + \sum_{i=1}^{k}\sum_{j=1}^{l} n_{ij}^2 - (\sum_{i=1}^{k} n_{i.} + \sum_{j=1}^{l} n_{.j}))$$

11.3 Weighted Clustering Ensemble

Motivation

The problem with the **clustering ensemble** algorithm is that it does not distinguish between *non-trivial* and *trivial* partitions so all partitions in clustering ensemble are treated **equally important**. This algorithm is also sensitive to the cluster distance used in the hierarchical clustering for reaching a consensus.

Solution

With **weighted clustering ensemble**, multiple internal and external validity indexes reflecting various aspects are applied to different partitions obtained by initial clustering analysis. **Internal indexes** are used directly to measure the importance of a partition, quantitatively and objectively. **External indexes** are used indirectly to measure the importance of a partition via comparing with other partitions used in clustering ensemble for another round of *evidence accumulation*.

The values of these indexes are used as weights to highlight the non-trivial evidence and to diminish the trivial evidence simultaneously. Thus, the weighted clustering ensemble yield better yet robust performance in clustering analysis.

Index