Never Stand Still

Classification (1)

COMP9417 Machine Learning & Data Mining

Term 3, 2019

Adapted from slides by Dr Michael Bain

Aims

This lecture will introduce you to machine learning approaches to the problem of classification. Following it you should be able to reproduce theoretical results, outline algorithmic techniques and describe practical applications for the topics:

- outline a framework for solving machine learning problems
- outline the general problem of induction
- describe issues of generalisation and evaluation for classification
- outline the use of a linear model as a 2-class classifier
- describe distance measures and how they are used in classification
  • outline the basic k-nearest neighbour classification method

Introduction

Classification (sometimes called *concept learning*) methods dominate machine learning . . .

. . . however, they often don't have convenient mathematical properties like regression, so are more complicated to analyse. The idea is to learn a *classifier*, which is usually a function mapping from an input data point to one of a set of discrete outputs, i.e., the classes.

We will mostly focus on their advantages and disadvantages as learning methods first, and point to unifying ideas and approaches where applicable. In this and the next lecture we focus on classification methods that are essentially *linear models* . . .

and in later lectures we will see other, more expressive, classifier learning methods.

Classification

Example: Imagine that we want to automate the process of sorting incoming fish in a fish-packing plant. And as a pilot, we start by separating sea bass from salmon using some information collected through sensing.

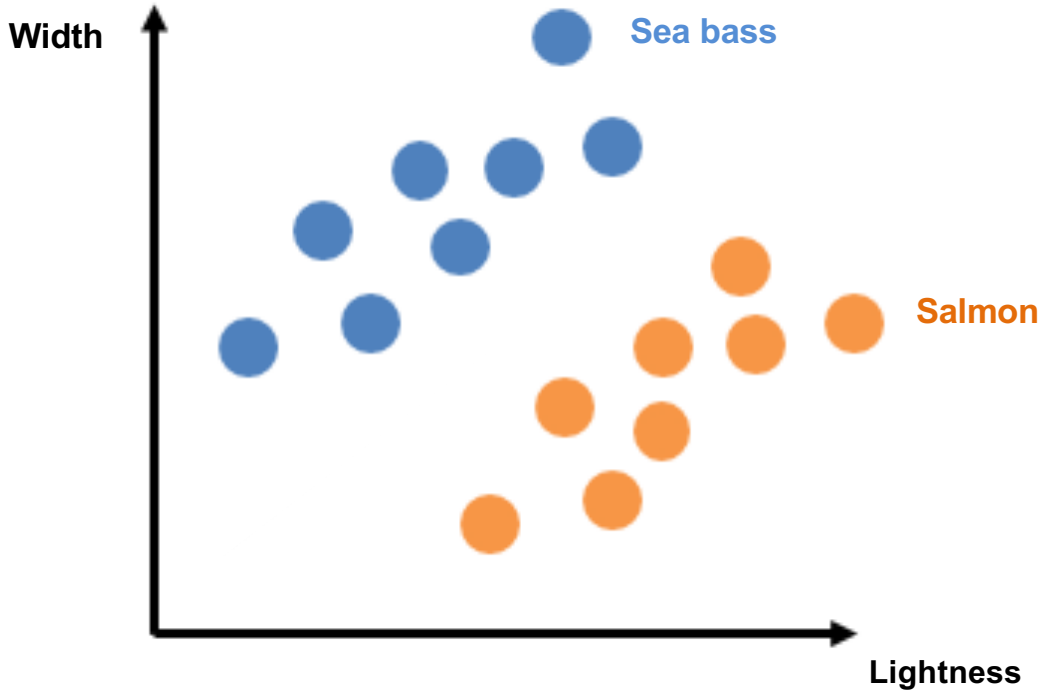
(a) (b)

Classification

Example: classifying sea bass vs. salmon

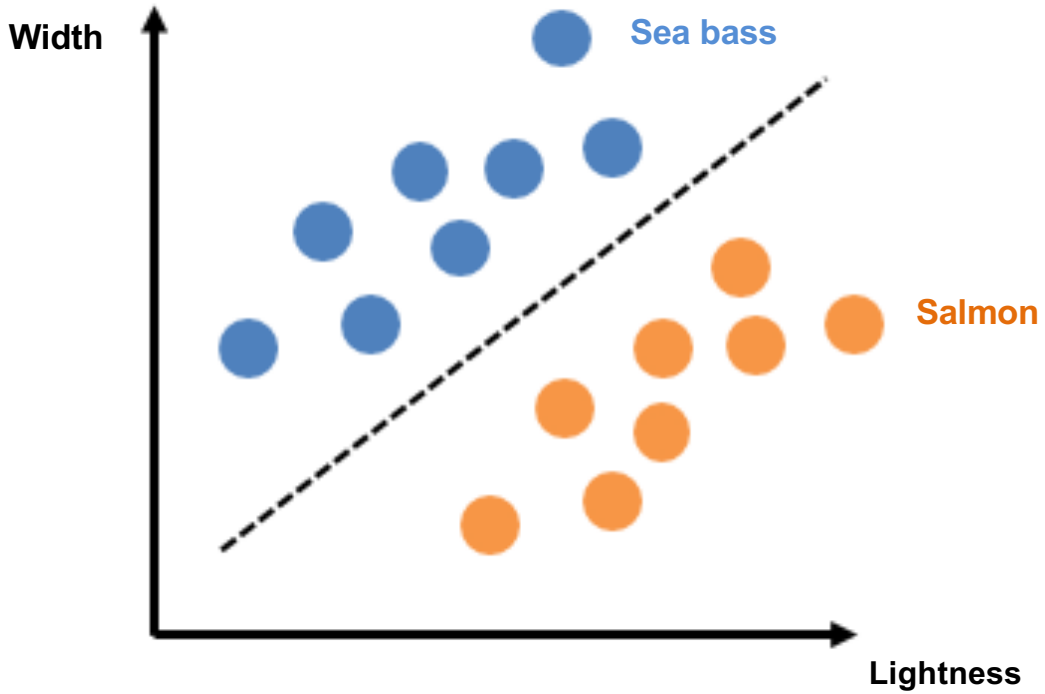
Features that can be used: width, length, weight, lightness, fins, eyes/mouth position, etc.

Question: how to separate these two classes?



Classification

Example: Maybe we can find a line that separates the two classes.



Classification

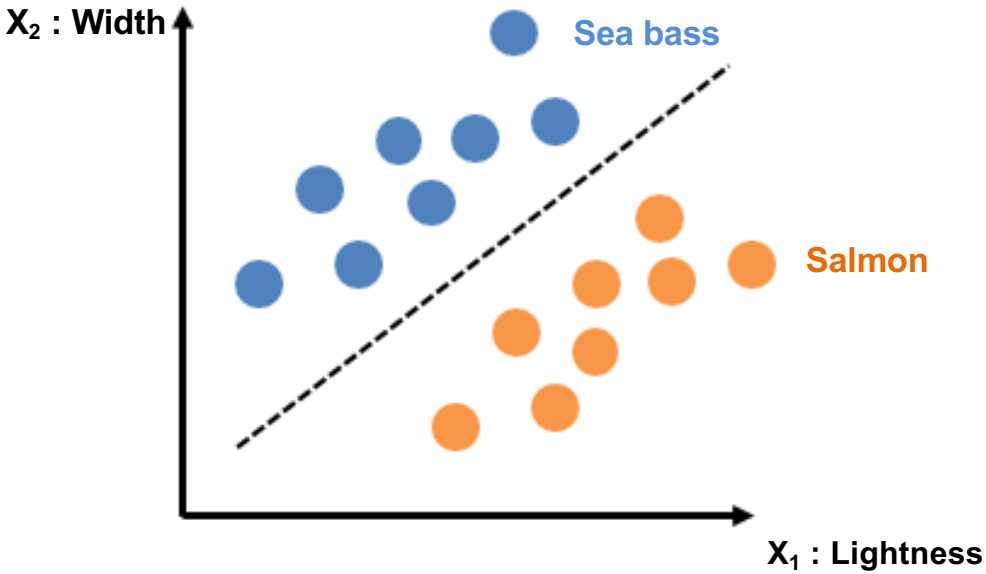
Example: If we find the line that separated the two classes, then how our algorithm makes prediction?

The line equation will look like:
$ax_1 + bx_2 + c = 0$

We can define a, b & c such that:

for any point above the line:
$ax_1 + bx_2 + c > 0$

and for any point below the line:
$ax_1 + bx_2 + c < 0$

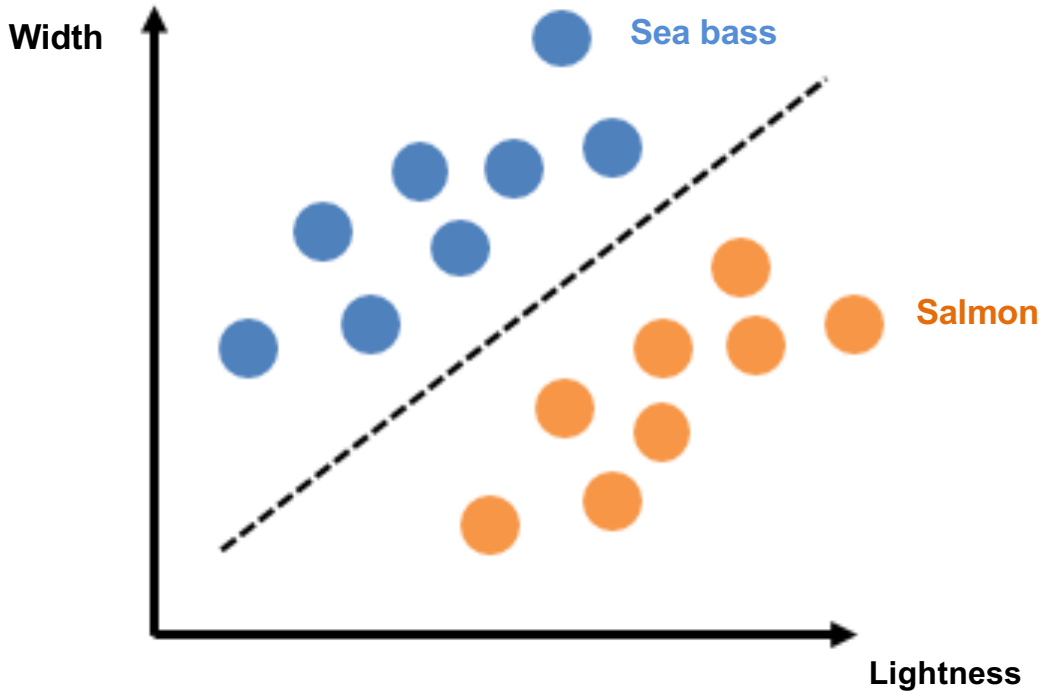


This type of classifier is called *linear classifier*. It is also a type of *discriminative learning* algorithm.

Classification

Example:

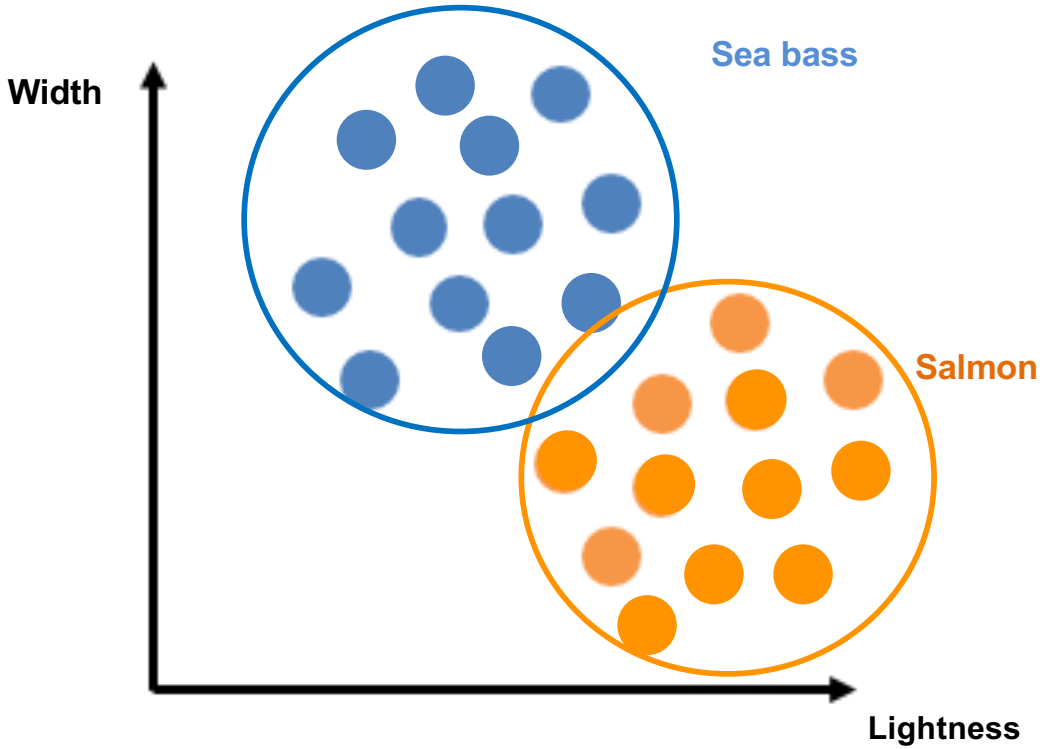
Can we do something different than finding the discriminative line (or some boundary) to be able to separate the two groups?



Classification

Example:

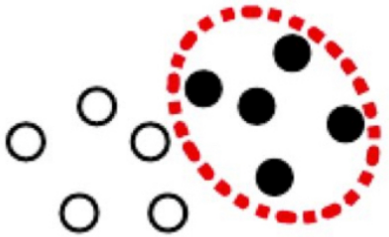
Instead of finding a discriminative line, maybe we can focus on one class at a time and build a model that describes how that class looks like; and then do the same for the other class. This type of models are called *generative learning algorithm*.



Classification

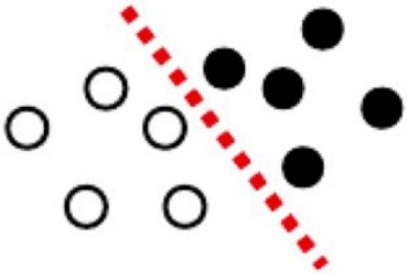
Generative algorithm: builds some models for each of the classes and then makes classification predictions based on looking at the test example and see it is more similar to which of the models.

- Learns $p(x|y)$ (and also $p(y)$, called class prior)
- So we can get $p(x, y) = p(x|y)p(y)$
- It learns the mechanism by which the data has been generated



Discriminative algorithm: Do not build models for different classes, but rather focuses on finding a decision boundary that separates classes

- Learns $p(y|x)$



Classification

- To predict the output for sample x, in generative algorithm, we have to estimate $p(y|x)$:

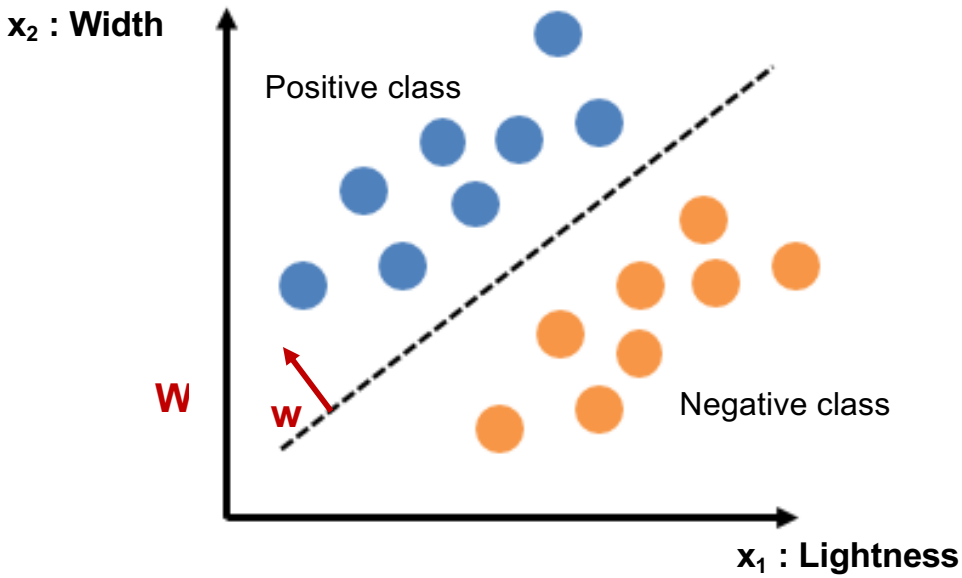
$$p(y=0|x) = \frac{p(x|y=0)p(y=0)}{p(x)}$$

$$p(y=1|x) = \frac{p(x|y=1)p(y=1)}{p(x)}$$

If $p(y=0|x) > p(y=1|x)$, then X belongs to class $y=0$ and otherwise to class $y=1$.

- For discriminate algorithm, we can directly have $p(y=0|x)$ and $p(y=1|x)$ and similar to above, if $p(y=0|x) > p(y=1|x)$, then x belongs to class $y=0$ and otherwise to class $y=1$.

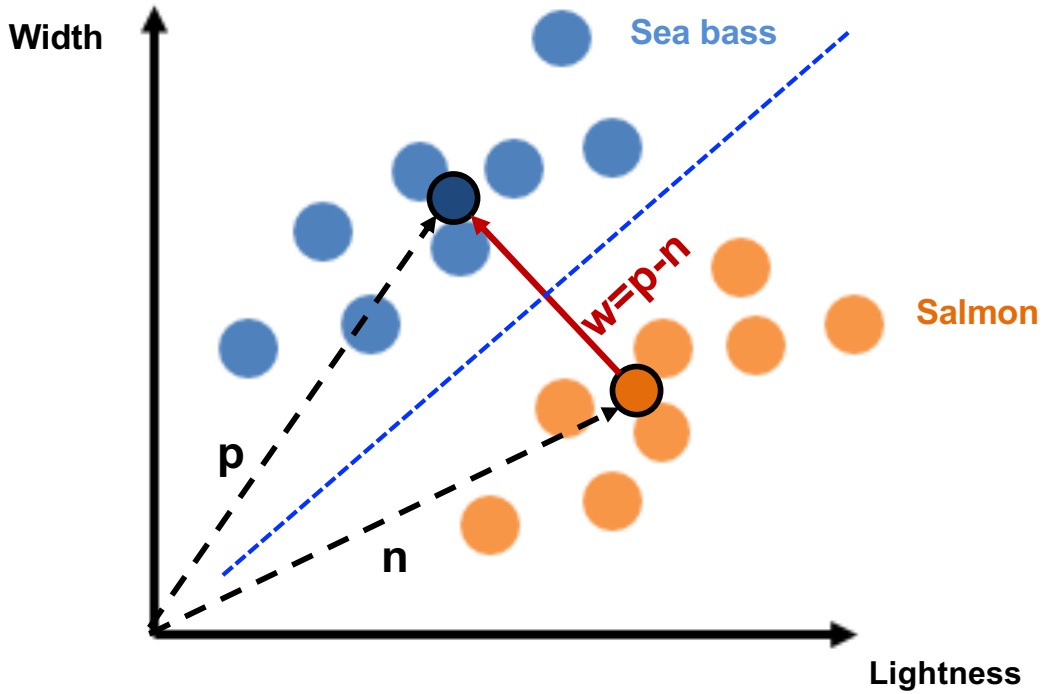
Linear classification in two dimensions



- We find the line that separates the two class: $ax_1 + bx_2 + c = 0$
- We define a weight vector $w^T = [a, b]$, $x^T = [x_1, x_2]$
- So the line can be defined by $x^T w = -c = t$
- w is perpendicular to decision boundary (in direction of positive class)
- t is the decision threshold (if $x^T w > \mathrm{t}$ then x belongs to positive class and if $x^T w < \mathrm{t}$ then x belongs to negative class)

Basic Linear Classifier

The basic linear classifier constructs a decision boundary by half-way intersecting the line between the positive and negative centres of mass.



Basic Linear Classifier

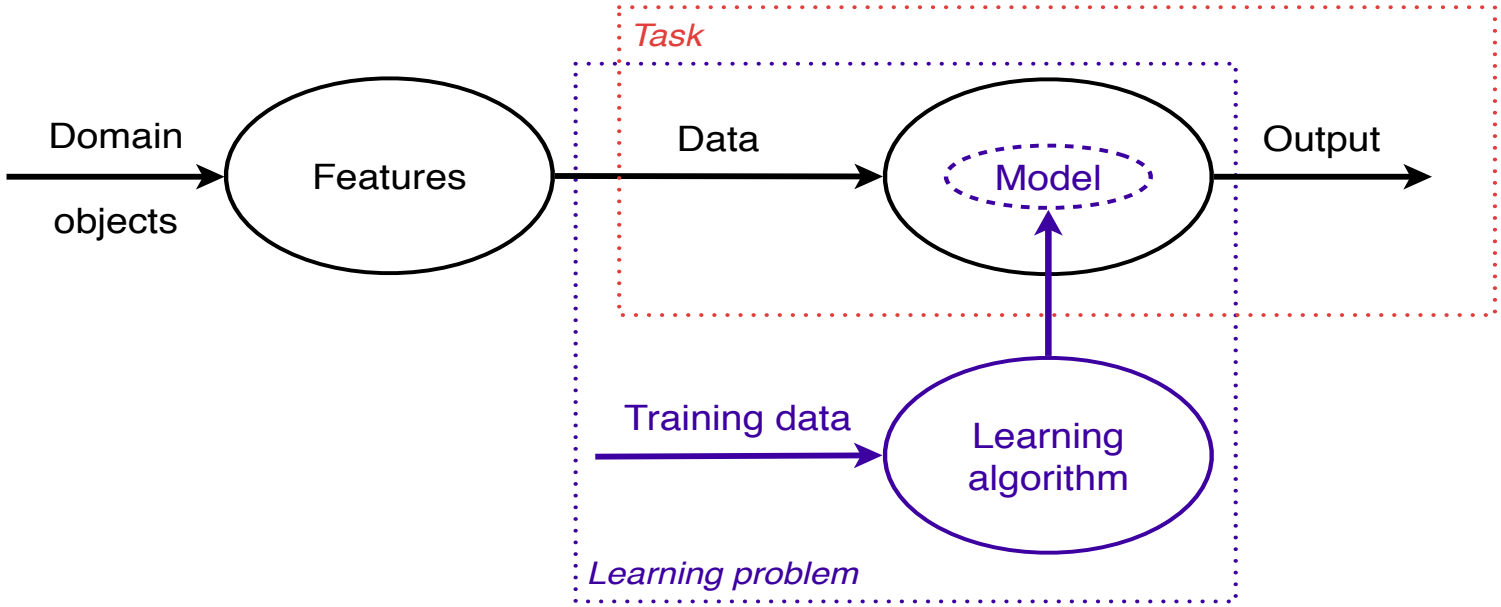
The basic linear classifier is described by the equation $x^T w = t$, and $w = p - n$

As we know, $\frac{p+n}{2}$ is on the decision boundary, so we have:

$$t = (\frac{p+n}{2})^T . (p - n) = \frac{||p||^2 - ||n||^2}{2}$$

Where $||x||$, denotes the length of vector x

How solve a task with machine learning



An overview of how machine learning is used to address a given task. A task (red box) requires an appropriate mapping – a model – from data described by features to outputs. Obtaining such a mapping from training data is what constitutes a learning problem (blue box).

Some terminology

Tasks are addressed by models, whereas learning problems are solved by learning algorithms that produce models.

Some terminology

Machine learning is concerned with using the right features to build the right models that achieve the right tasks.

Some terminology

Models lend the machine learning field diversity, but tasks and features give it unity.

Some terminology

Does the algorithm require all training data to be present before the start of learning ? If yes, then it is categorised as **batch learning** (a.k.a. **offline learning**) algorithm.

If however, it can continue to learn a new data arrives, it is an **online learning** algorithm.

Some terminology

If the model has a fixed number of parameters it is categorised as **parametric**.

Otherwise, if the number of parameters grows with the amount of training data it is categorised as **non-parametric**. They do not make any strong assumption about the underlying model and so they are more flexible.

The philosophical problem

Deduction: derive specific consequences from general theories

Induction: derive general theories from specific observations

Deduction is well-founded (mathematical logic).

Induction is (philosophically) problematic – induction is useful since it often seems to work – an inductive argument !

Generalisation - the key objective of machine learning

What we are really interested in is generalising from the sample of data in our training set. This can be stated as:

The inductive learning hypothesis

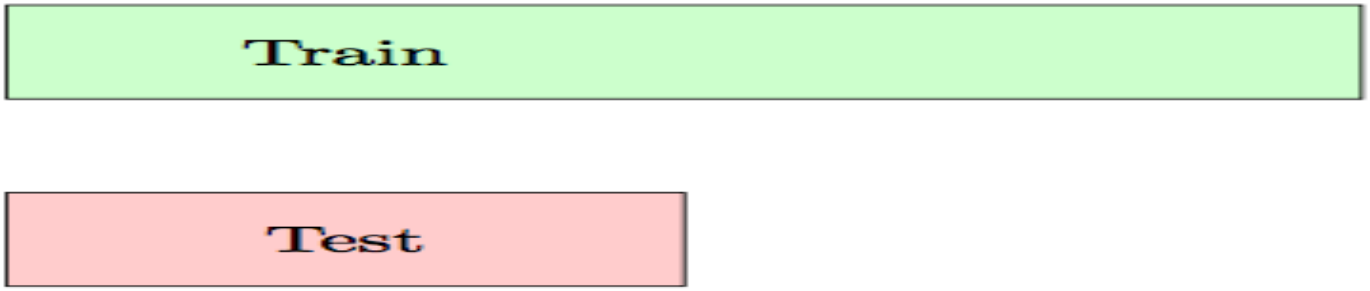
> *Any hypothesis found to approximate the target (true) function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.*

A corollary of this is that it is necessary to make some assumptions about the type of target function in a task for an algorithm to go beyond the data, i.e., generalise or learn.

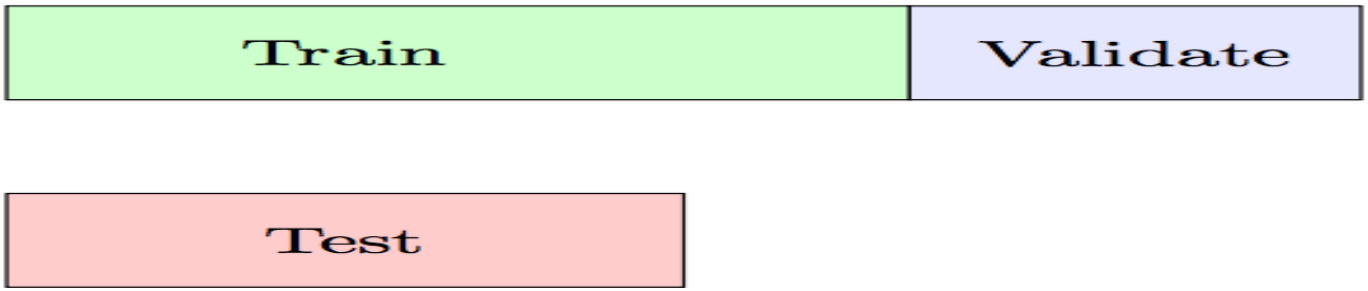
Cross-validation

There are certain parameters that need to be estimated during learning. We use the data, but NOT the training set, OR the test set. Instead, we use a separate *validation* or *development* set.

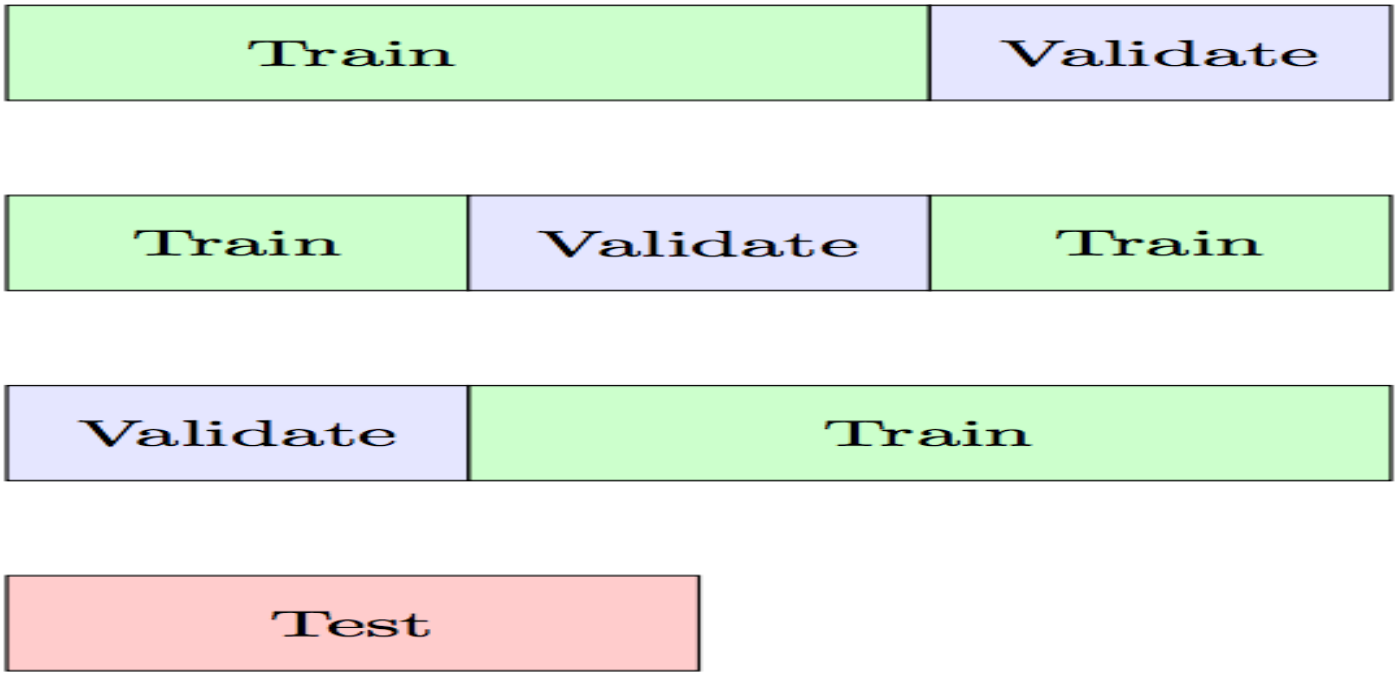
Cross-validation



Cross-validation



Cross-validation



Data Types

In Machine Learning world, in general two types of data is defined:

- **Numerical**: Anything represented by numbers (e.g., integer , floating point)
- **Categorical**: everything that is not numerical (e.g. discrete labeled groups)

In general, for machine learning algorithms, data has to be represented in numeric form

Data Types

Another taxonomy of data types:

1. **Irrelevant**: it might be represented with strings or numbers but has no relationship with the outcome (e.g. participants name or code)
2. **Nominal**: discrete values with no numerical relationship between different categories (e.g. animal types, colors, nationality)
3. **Binary**: discrete data with only two possibilities (e.g cancerous vs. non-cancerous)

Data Types

4. **Ordinal**: discrete integers that can be ranked, but the relative distance between any two number can not be defined (e.g. students rank based on GPA)
5. **Count**: discrete whole numbers without any negatives
6. **Time**: a cyclical, repeating continuous form of data (e.g days, weeks)
7. **Interval**: data that the we can measure the distance between different values. (e.g temperature, income)

Binary Classification task

In a binary classification (or binomial classification) task, we always want to classify the data of a given set into two groups. We usually define one of the classes as positive and one as negative.

- Sometimes the classes are equally important (e.g. recognition of dog vs cat in image classification
- Sometimes misclassification in one of the classes is more costly than misclassification in the other class (e.g. predicting that someone has cancer while (s)he doesn't have vs predicting that someone doesn't have cancer while (s)he has) therefore we may prefer to have better classification in one class in the cost of more errors in the other class

Evaluation of error

If we have a binary classification, then we have two classes of $y \in \{0,1\}$, where we call the class $y = 1$, *positive class* and $y = 0$, *negative class.*

Some evaluation metrics:

- **True positive:** number of instances from class one that have been predicted as one
- **True negative:** number of instances from class zero that have been predicted as zero
- **False positive:** number of instances from class zero that have been predicted as one
- **False negative:** number of instances from class one that have been predicted as zero

Contingency table

For two-class prediction case:

Actual Class	Predicted Class	
	Positive	Negative
Positive	True Positive (TP)	False Negative (FN)
Negative	False Positive (FP)	True Negative (TN)

This is also called *confusion matrix*

Classification Accuracy

Classification Accuracy on a sample of labelled pairs $(X, c(X))$ given a learned classification model that predicts, for each instance X, a class value $\hat{c}(X)$:

$$acc = \frac{1}{|Test|} \sum_{x \in Test} I[\hat{c}(X) = c(X)]$$

where $Test$ is a test set and $I[]$ is the indicator function which is 1 iff its argument evaluates to true, and 0 otherwise.

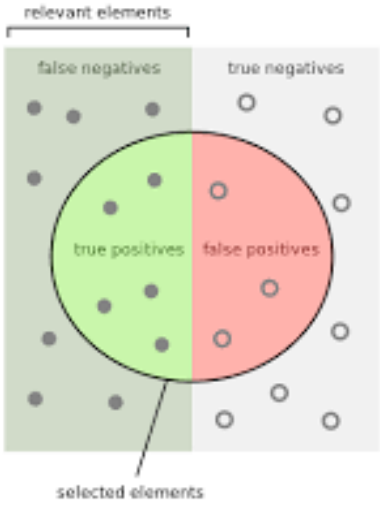
$$Classification\ Error\ is\ = 1 - acc.$$

Other evaluation metrics

Precision/correctness

– is the number of relevant objects classified correctly divided by the total number of relevant objects classified

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



Recall/sensitivity/completeness/true positive rate (TPR)

– is the number of relevant objects classified correctly divided by total number of relevant/correct objects

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

Other evaluation metrics

F_1 score: a measure of accuracy, which is the harmonic mean of precision and recall and is defined as:

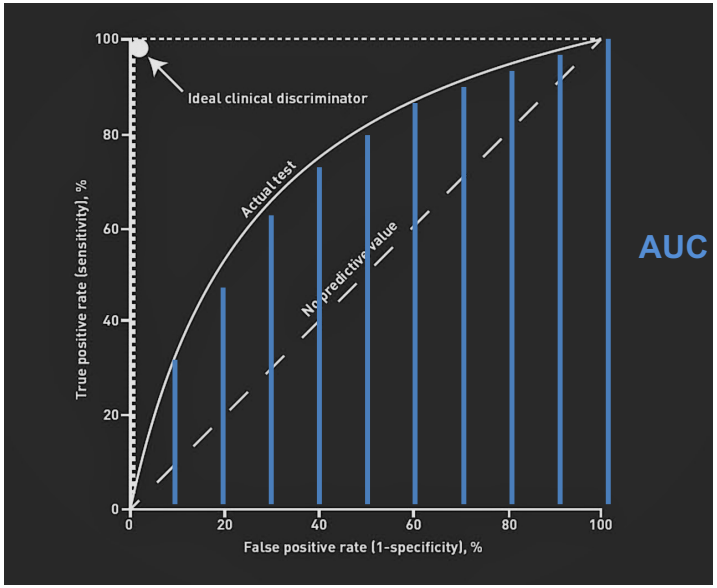
$$F_1 = 2.\frac{precision.recall}{precision + recall}$$

This measure gives equal importance to precision and recall which is sometime undesirable; so we have to decide which metric to use depending on the task and what's important for the task.

Other evaluation metrics

AUC-ROC curve: Area Under the Curve (AUC) – Receiver Operating Characteristics (ROC) curve is one of the most important evaluation metric for performance of classification models. This metric evaluates the model at different threshold settings and can inform us on the capability of the model in distinguishing between classes.

- $TPR = \frac{TP}{TP+FN}$
- $FPR = \frac{FP}{FP+TN}$
- A good model has AUC close to 1
- A very poor model has AUC close to 0
- $AUC = 0.5$ means no class separation



Missing Value: An issue to consider

Missing Values

- In practice it rarely happens that the data is complete and homogenous.
- Why data is incomplete:
  - Human errors
  - Sensor errors
  - Software bugs
  - Faulty preprocessing
  - …

Missing Values

How to handle missing values (common approaches):

- Deleting samples with missing values
- Replacing the missing value with some statistics from the data (mean, median, …)
- Assigning a unique category
- Predicting the missing values
- Using algorithms that support missing values

Missing Values

Deleting samples with missing values:

- Pros:
  - A robust and probably more accurate model
- Cons:
  - Loss of information and data
  - Works poorly if the percentage of missing values is high

Missing Values

Replacing the missing value with mean/median/mode:

- Pros:
  - When the data size is small, it is better than deleting
  - It can prevent data loss
- Cons:
  - Imputing the approximations add variance and bias
  - Works poorly compared to other methods

Missing Values

Assigning a unique category:

- Pros:
  - Less possibilities with one extra category, resulting in low variance after one hot encoding — since it is categorical
  - No loss of data
- Cons:
  - Adds less variance
  - Adds another feature to the model while encoding

Missing Values

Predicting the missing values:

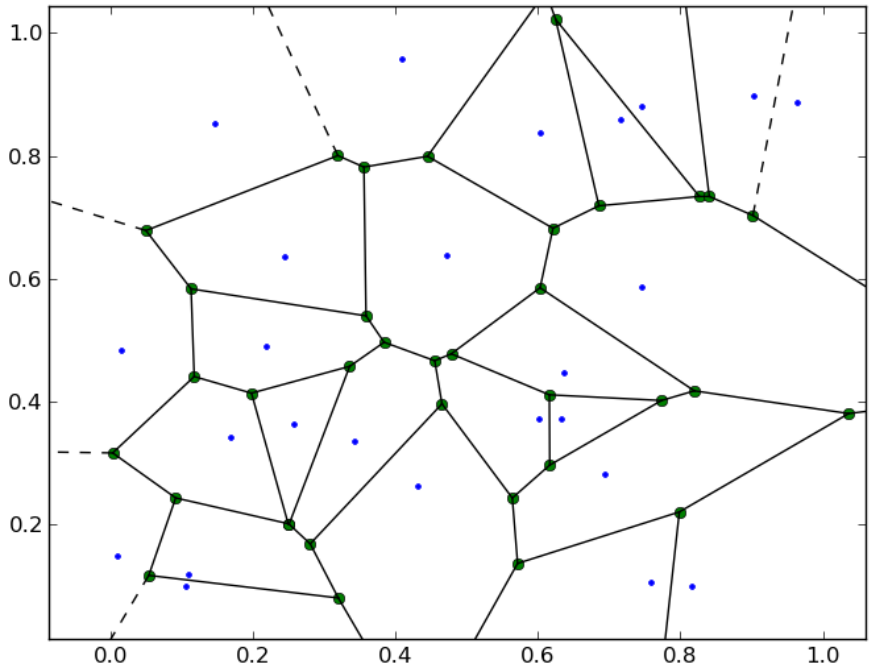
- Pros:
  - Imputing the missing variable is an improvement as long as the bias from it is smaller than the omitted variable bias
  - Yields unbiased estimates of the model parameters
- Cons:
  - Bias also arises when an incomplete conditioning set is used for a categorical variable
  - Considered only as a proxy for the true values

Missing Values

Using algorithms that support missing values:

- Pros:
  - Does not require creation of a predictive model
  - Correlation of the data is neglected
- Cons:
  - Is a very time consuming process and it can be critical in data mining where large databases are being extracted

Nearest Neighbour



Nearest Neighbour is a regression or classification algorithm that predicts whatever is the output value of the nearest data point to some query.

To find the nearest data point, we have to find the *distance* between the query and other points. So we have to decide how to define the *distance*.

Minkowski distance

Minkowski distance If $\mathcal{X} \rightarrow \mathbb{R}^d, x, y \in \chi$, the Minkowski distance of order $p > 0$ is defined as:

$$Dis_p(x,y) = (\sum_{j=1}^{d} |x_j - y_j|^p)^{1/p} = ||\mathrm{x} - \mathrm{y}||_p$$

Where $||z||_p = (\sum_{j=1}^{d} |z_j|^p)^{1/p}$ is the $p - norm$ (sometimes denoted L_p norm) of the vector z.

Minkowski distance

- The *2-norm* refers to the familiar *Euclidean distance*

$$Dis_2(x,y) = \sqrt{\sum_{j=1}^{n}(x_j - y_j)^2} = \sqrt{(x-y)^T(x-y)}$$

- The *1-norm* denotes *Manhattan distance*, also called *cityblock* distance:

$$Dis_1(x,y) = \sum_{j=1}^{n}|x_j - y_j|$$

Minkowski distance

- If we now let p grow larger, the distance will be more and more dominated by the largest coordinate-wise distance, from which we can infer that $Dis_\infty = max_j|x_j - y_j|$; this is also called *Chebyshev distance*.
- You will sometimes see references to the *0-norm* (or L_0 norm) which counts the number of non-zero elements in a vector. The corresponding distance then counts the number of positions in which vectors x and y differ. This is not strictly a Minkowski distance; however, we can define it as:

$$Dis_0(x,y) = \sum_{j=1}^{d}(x_j - y_j)^0 = \sum_{j=1}^{d} I[x_j \neq y_j]$$

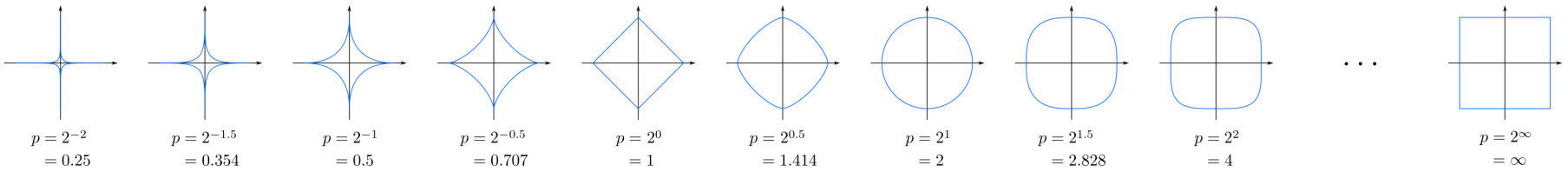
under the understanding that $x^0 = 0$ for $x = 0$ and 1 otherwise.

Minkowski distance

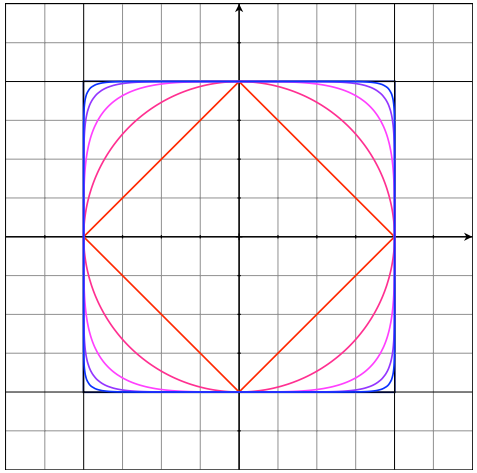
Sometimes the data is not naturally in $\mathbb{R}^d$, but if we can turn it into Boolean features, or character sequences, we can still apply distance measures. For example:

- If x and y are binary strings, this is also called the *Hamming distance*. Alternatively, we can see the Hamming distance as the number of bits that need to be flipped to change x into y.
- For non-binary strings of unequal length this can be generalised to the notion of *edit distance* or *Levenshtein distance*.

Circles and ellipses



Unite circles with different order-p Minkowski distance



- Notice that for points on the coordinate axes all distances agree
- If we require a rotation invariant distance metric, then Euclidean distance is our only choice

Distance metric

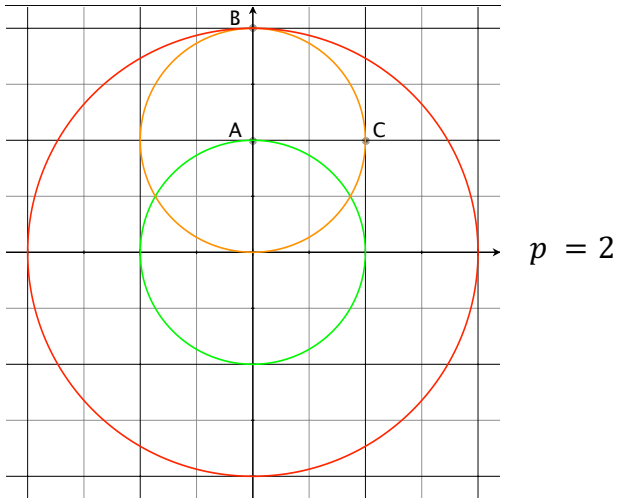
Distance metric Given an instance space $\mathcal{X}$, a distance metric is a function $Dis : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^d$ such that for any $x, y, z \in \mathcal{X}$:

- distances between a point and itself are zero: $Dis(x, x) = 0$
- all other distances are larger than zero: if $x \neq y$ then $Dis(x, y) > 0$
- distances are symmetric: $Dis(y, x) = Dis(x, y)$
- detours can not shorten the distance (triangle inequality):

$$Dis(x, z) \leq Dis(x, y) + Dis(y, z)$$

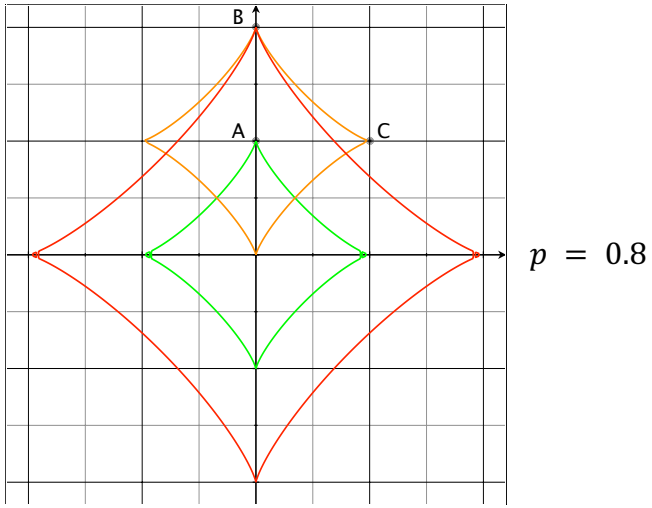
If the second condition is weakened to a non-strict inequality – i.e., $Dis(x, y)$ may be zero even if $x \neq y$ – the function Dis is called a *pseudo-metric*.

The triangle inequality – Minkowski distance for $p = 2$



- If the triangle inequality holds, then $Dis(0, C) \leq Dis(0, A) + Dis(A, C)$
- For $p = 2$, $Dis(A, C) = Dis(A, B)$ (orange circle)
- So $Dis(0, A) + Dis(A, C) = Dis(0, A) + Dis(A, B) = Dis(0, B)$
- Based on the red circle $Dis(0, B) > Dis(0, C)$, therefore the triangle inequality holds for $p = 2$

The triangle inequality – Minkowski distance for $p \leq 1$



- For $p = 1$, $Dis(0, C) = Dis(0, B) = Dis(0, A) + Dis(A, C)$
- For $p < 1$, $Dis(0, C) > Dis(0, B) = Dis(0, A) + Dis(A, C)$, so it violates the triangle inequality in the distance metric properties. So it is not a metric

Means and distances

The arithmetic mean minimises squared Euclidean distance *The arithmetic mean μ of a set of data points D in a Euclidean space is the unique point that minimises the sum of squared Euclidean distances to those data points.*

Proof. We will show that $\arg\min_y \sum_{x \in D} ||x - y||^2 = \mu$, where $||.||$ denotes the *2-norm*. We find this minimum by taking the gradient (the vector of partial derivatives with respect to y) of the sum and setting it to the zero vector:

$$\nabla_y \sum_{x \in D} ||x - y||^2 = -2 \sum_{x \in D} (x - y) = -2 \sum_{x \in D} x + 2|D|y = 0$$

From which we derive $y = \frac{1}{|D|} \sum_{x \in D} x = \mu$

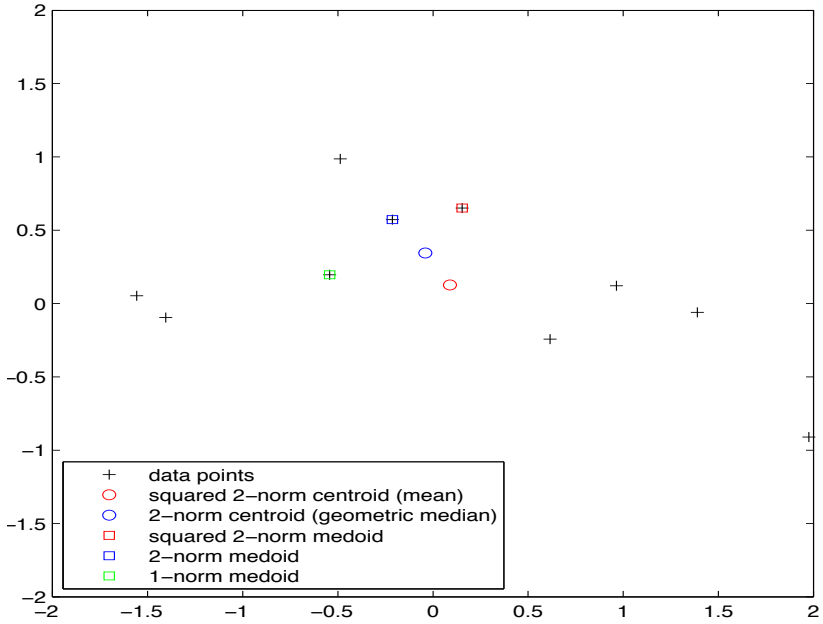
Means and distances

- Notice that minimising the *sum* of squared Euclidean distances of a given set of points is the same as minimising the *average* squared Euclidean distance.
- You may wonder what happens if we drop the square here: wouldn't it be more natural to take the point that minimises total Euclidean distance as exemplar?
- This point is known as the *geometric median*, as for univariate data it corresponds to the *median* or 'middle value' of a set of numbers. However, for multivariate data there is no closed-form expression for the geometric median, which needs to be calculated by successive approximation.

Means and distances

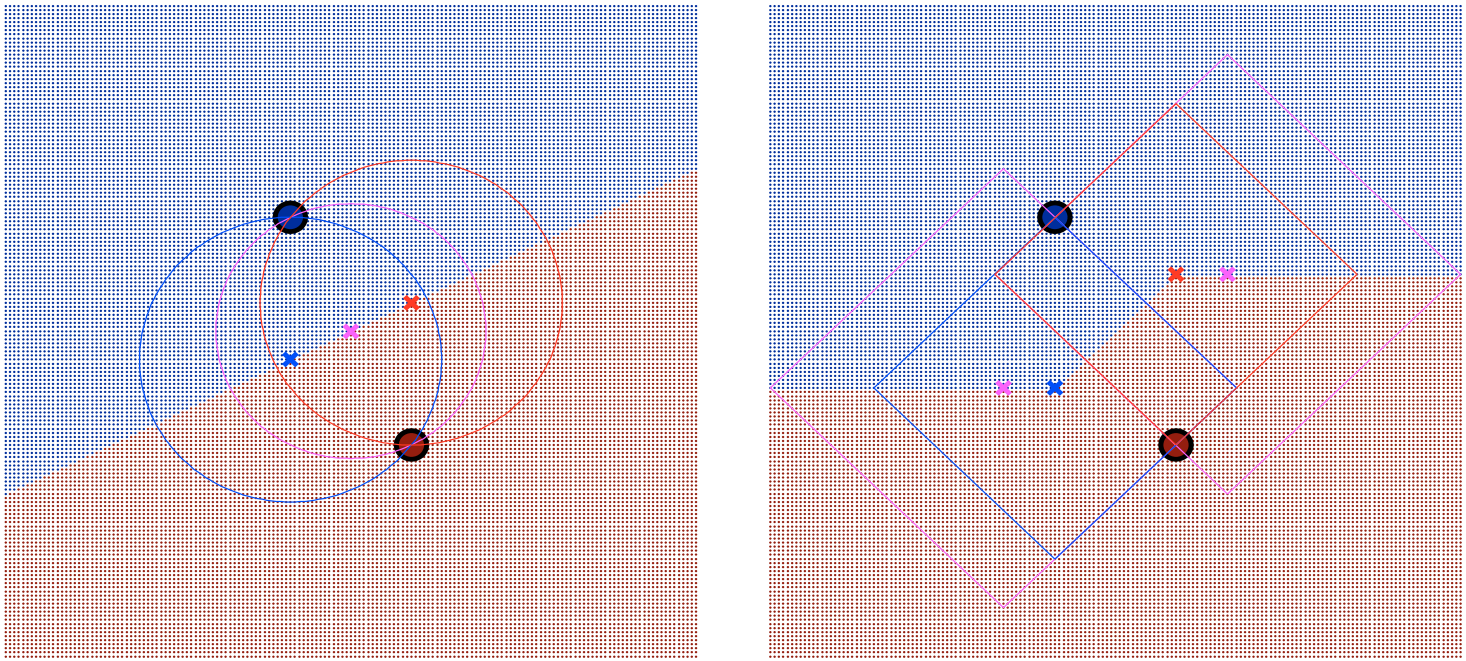
- In certain situations it makes sense to restrict an exemplar to be one of the given data points. In that case, we speak of a *medoid*, to distinguish it from a *centroid* which is an exemplar that doesn't have to occur in the data.
- Finding a medoid requires us to calculate, for each data point, the total distance to all other data points, in order to choose the point that minimises it. Regardless of the distance metric used, this is an $O(n^2)$ operation for n points.
- So for medoids there is no computational reason to prefer one distance metric over another.
- There may be more than one medoid.

Centroids and medoids



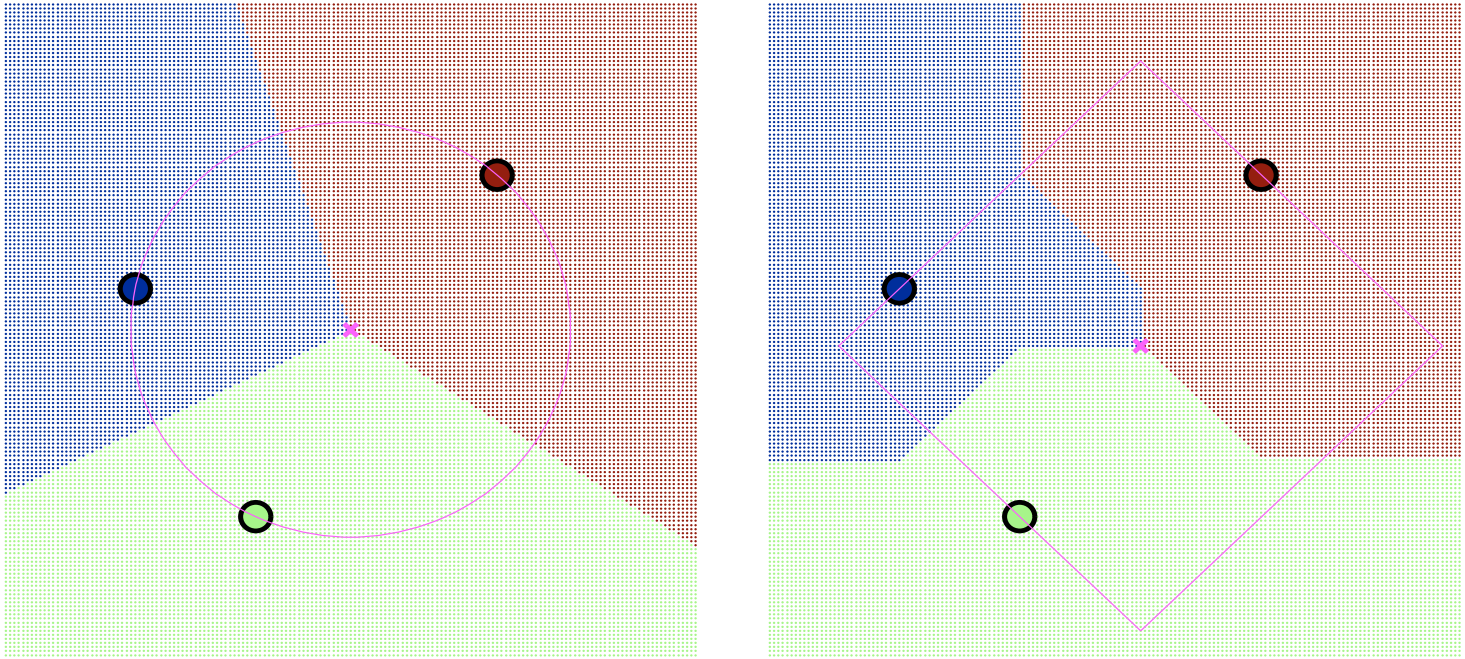
A small data set of 10 points, with circles indicating centroids and squares indicating medoids (the latter must be data points), for different distance metrics. Notice how the outlier on the bottom-right 'pulls' the mean away from the geometric median; as a result the corresponding medoid changes as well.

Two-exemplar decision boundaries



(left) For two exemplars the nearest-exemplar decision rule with Euclidean distance results in a linear decision boundary coinciding with the perpendicular bisector of the line connecting the two exemplars.
(right) Using Manhattan distance the circles are replaced by diamonds.

Three-exemplar decision boundaries



(left) Decision regions defined by the 2-norm nearest-exemplar decision rule for three exemplars. (right) With Manhattan distance the decision regions become non-convex.

Distance-based models

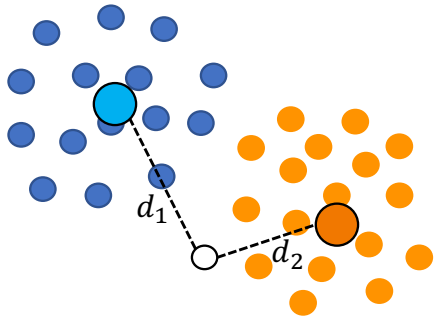
To summarise, the main ingredients of distance-based models are:

- distance metrics, which can be Euclidean, Manhattan, Minkowski or Mahalanobis, among many others;
- exemplars: centroids that find a centre of mass according to a chosen distance metric, or medoids that find the most centrally located data point; and
- distance-based decision rules, which take a vote among the k nearest exemplars.

Nearest Centroid Classifier

Nearest Centroid Classifier

This is a classifier based on minimum distance principle, where the class exemplars are just the centroids (or means)



Training: for training sample pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ where x_i is the feature vector for sample i and y_i is the class label, class centroids are:

$$\mu_k = \frac{1}{|C_k|} \sum_{j \in C_k} x_j$$

Test

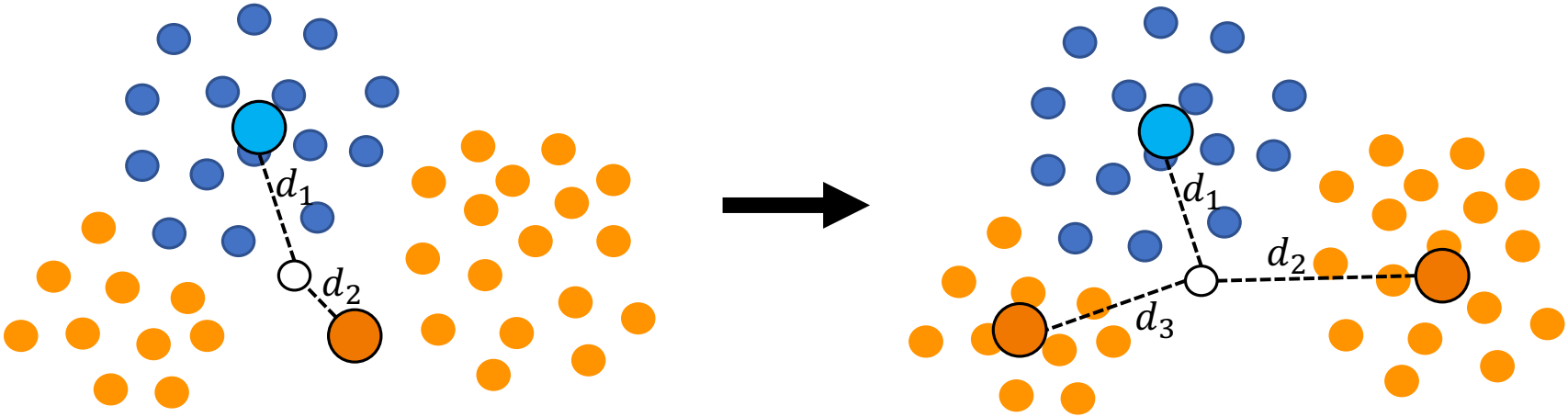
- a new unknown object with feature vector x is classified as class i if it is much closer to the mean vector of class k than to any other class mean vector

Basic Linear Classifier & Nearest Centroid Classifier

- The basic linear classifier is distance-based.
- An alternative, distance-based way to classify instances without direct reference to a decision boundary is by the following decision rule: if x is nearest to $\mu^{\oplus}$ then classify it as positive, otherwise as negative; or equivalently, classify an instance to the class of the nearest exemplar.
- If we use Euclidean distance as our closeness measure, simple geometry tells us we get exactly the same decision boundary.
- So the basic linear classifier can be interpreted from a distance-based perspective as constructing exemplars that minimise squared Euclidean distance within each class, and then applying a nearest-exemplar decision rule.

Nearest Centroid Classifier

- What happens if a class has more than one mode? (similar to the image)
  1. If there is only one centroid per class, then it will perform poorly
  2. If we can somehow find different modes, we can define one centroid per each mode which helps the classifier



(1) (2)

Nearest Centroid Classifier

Advantages:

- Simple
- Fast
- works well when classes are compact and far from each other.

Nearest Centroid Classifier

Disadvantages:

- For complex classes (eg. Multimodal, non-spherical) may give very poor results
- Can not handle outliers and noisy data well
- Can not handle missing data

Nearest neighbour classification

Nearest neighbour classification

- Related to the simplest form of learning: rote learning or memorization
  - Training instances are searched for instance that most closely resembles new or *query* instance
  - The instances themselves represent the knowledge
  - Called: *instance-based*, *memory-based* learning or *case-based* learning; often a form of *local* learning
- The *similarity* or *distance* function defines "learning", i.e., how to go
- beyond simple memorization
- Intuitive idea — instances "close by", i.e., neighbours or *exemplars*, should be classified similarly
- Instance-based learning is lazy learning
- Methods: *nearest-neighbour*, *k-nearest-neighbour*, *lowess*, . . .
- Ideas also important for *unsupervised* methods, e.g., clustering (later lectures)

Nearest Neighbour

Stores all training examples $\langle x^{(j)}, f(x^{(j)}) \rangle$.

Nearest neighbour:

- Given query instance $x^{(q)}$, first locate nearest training example $x^{(n)}$, then estimate $\hat{f}(x^{(q)}) \leftarrow f(x^{(n)})$

k-Nearest neighbour:

- Given $x^{(q)}$, take vote among its k nearest neighbours (if discrete-valued target function) (see next slide)
- take mean of f values of k nearest neighbours (if real-valued)

$$\hat{f}(x^{(q)}) \leftarrow \frac{\sum_{j=1}^{k} f(x^{(j)})}{k}$$

k-Nearest Neighbour Algorithm

Training algorithm:

- For each training example $\langle x^{(j)}, f(x^{(j)}) \rangle$, add the example to the list *training _examples*.

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x^{(1)}, \ldots, x^{(k)}$ be the k instances from *training examples* that are *nearest* to $x^{(q)}$ by the distance function
  - Return

$$\hat{f}(x_q) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x^{(j)}))$$

Where $\delta(a, b) = 1$ if $a = b$ and 0 otherwise.

Distance function again

The distance function defines what is learned.
Instance x is described by a feature vector (list of attribute-value pairs)

$$\langle x_1, \ldots, x_d \rangle$$

Where x_r denotes the value of the rth attribute/feature of x.

Most commonly used distance function is *Euclidean distance* . . .

- distance between two instances $x^{(i)}$ and $x^{(j)}$ is defined to be

$$Dis(x^{(i)}, x^{(j)}) = \sqrt{\sum_{r=1}^{d} (x_r^{(i)} - x_r^{(j)})^2}$$

Distance function again

Many other distance functions could be used . . .

- e.g., *Manhattan* or *city-block* distance (sum of absolute values of differences between attributes)

$$Dis\left(x^{(i)}, x^{(j)}\right) = \sum_{r=1}^{d} |x_r^{(i)} - x_r^{(j)}|$$

Vector-based formalization – use norm L_1, L_2, …

The idea of distance functions will appear again in *kernel methods*.

Normalization and other issues

- Different attributes measured on different scales (for example one attribute/feature may have a range of $[0,100]$ and another have a range of $[-1,1]$)
- Need to be *normalized* (why ?)

$$x'_r = \frac{x_r - \min(x_r)}{max(x_r) - \min(x_r)}$$

where x_r is the actual value of attribute/feature r and x'_r is the normalised value.

- Nominal attributes: distance either 0 or 1

When To Consider Nearest Neighbour

- Instances map to points in $\mathbb{R}^d$
- Less than 20 attributes per instance
    - or number of attributes can be reduced . . .
- Lots of training data
- No requirement for “explanatory” model to be learned

When To Consider Nearest Neighbour

Advantages:

- Statisticians have used k-NN since early 1950s
- Can be very accurate
- Training is very fast
- Can learn complex target functions

When To Consider Nearest Neighbour

Disadvantages:

- Slow at query time: basic algorithm scans entire training data to derive a prediction
- "Curse of dimensionality"
- Assumes all attributes are equally important, so easily fooled by irrelevant attributes
  - Remedy: attribute selection or weights
- Problem of noisy instances:
  - Remedy: remove from data set
  - not easy – how to know which are noisy ?
- Needs homogenous feature type and scale
- Finding the optimal number of neighbors (k) can be challenging

Inductive Bias of k-NN

What is the inductive bias of k-NN ?

- an assumption that the classification of query instance $x^{(q)}$ will be most similar to the classification of other instances that are nearby according to the distance function

Nearest-neighbour classifier

- kNN uses the training data as exemplars, so training is $O(n)$ (but prediction is also $O(n)$!)
- 1NN perfectly separates training data, so low bias but high variance
- By increasing the number of neighbours k we increase bias and
- decrease variance (what happens when $k = n$?)
- Easily adapted to real-valued targets, and even to structured objects (nearest-neighbour retrieval). Can also output probabilities when $k > 1$
- Warning: in high-dimensional spaces everything is far away from everything and so pairwise distances are uninformative (curse of dimensionality)

Distance-Weighted kNN

- Might want to weight nearer neighbours more heavily ...
- Use distance function to construct a weight w_i
- Replace the final line of the classification algorithm by:

$$\hat{f}(x_q) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} w_i \delta(v, f(x^{(j)}))$$

Where

$$w_i = \frac{1}{Dis(x^{(q)}, x^{(i)})^2}$$

$Dis(x^{(q)}, x^{(i)})$ is distance between $x^{(q)}$, $x^{(i)}$

Distance-Weighted kNN

For real-valued target functions replace the final line of the algorithm by:

$$\hat{f}(x^{(q)}) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x^{(i)})}{\sum_{i=1}^{k} w_i}$$

(denominator normalizes contribution of individual weights).

Now we can consider using all the training examples instead of just k:

- using all examples (i.e., when $k = n$ and n is number of training samples) with the rule above is called *Shepard's method*

Evaluation

Lazy learners do not construct an explicit model, so how do we evaluate the output of the learning process ?

- 1-NN – training set error is always zero !
  - each training example is always closest to itself
- k-NN – overfitting may be hard to detect

Solution:

Leave-one-out cross-validation (LOOCV) – leave out each example and predict it given the rest:

$$\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(i-1)}, y^{(i-1)}\right), \left(x^{(i+1)}, y^{(i+1)}\right), \ldots, \left(x^{(n)}, y^{(n)}\right)$$

Error is mean over all predicted examples. Fast – no models to be built !

1

Curse of Dimensionality

Bellman (1960) coined this term in the context of dynamic programming

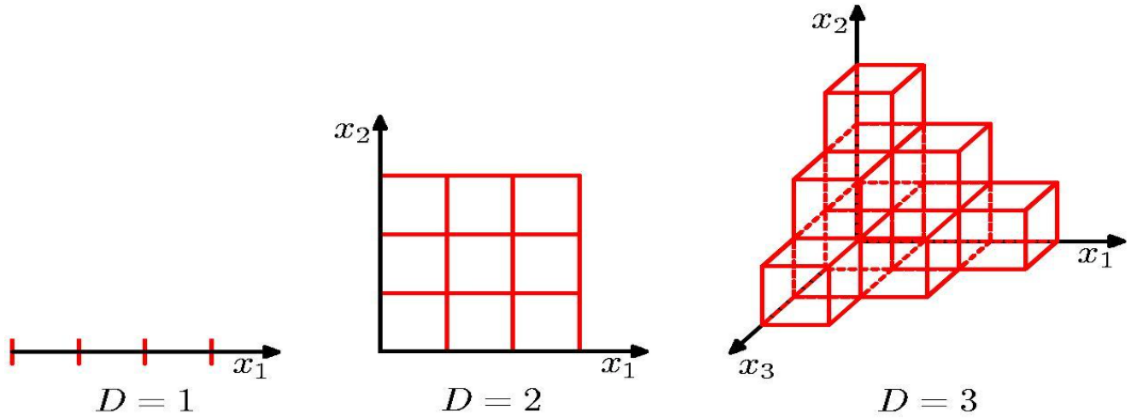
Imagine instances described by 20 attributes, but only 2 are relevant to target function — "similar" examples will appear "distant".

Curse of dimensionality: nearest neighbour is easily mislead when high-dimensional X in terms of the number of features – problem of irrelevant attributes

One approach:

- Stretch jth axis by weight z_j, where $z_1, \ldots, z_d$ chosen to minimize prediction error
- Use cross-validation to automatically choose weights $z_1, \ldots, z_d$
- Note setting z_j to zero eliminates this dimension altogether

Curse of Dimensionality



- number of “cells” in the instance space grows exponentially in the number of features
- with exponentially many cells we would need exponentially many data points to ensure that each cell is sufficiently populated to make nearest-neighbour predictions reliably

Curse of Dimensionality

Some ideas to address this for instance-based (nearest-neighbour) learning

- Euclidean distance with weights on attributes

$$Dis\left(x^{(q)}, x^{(j)}\right) = \sqrt{\sum_{j=1}^{d} z_j \left(x_j^{(q)} - x_j^{(i)}\right)}$$

- updating of weights based on nearest neighbour classification error
  - class correct/incorrect: weight increased/decreased
  - can be useful if not all features used in classification

See Moore and Lee (1994) “Efficient Algorithms for Minimizing Cross Validation Error”

Instance-based (nearest-neighbour) learning

Recap – Practical problems of 1-NN scheme:

- Slow (but fast *k-dimensional tree*-based approaches exist)
  - Remedy: removing irrelevant data
- Noise (but k-NN copes quite well with noise)
  - Remedy: removing noisy instances
- All attributes deemed equally important
  - Remedy: attribute weighting (or simply selection)

Some refinements of instance-based classifiers

- Edited NN classifiers discard some of the training instances before making predictions
- Saves memory and speeds up classification
- IB2: incremental NN learner: only incorporates misclassified instances into the classifier
    - Problem: noisy data gets incorporated
- IB3: store classification performance information with each instance & only use in prediction if above a threshold

Dealing with noise

Use larger values of k (why ?) How to find the “right” k ?

- One way: cross-validation-based k-NN classifier (but slow)
- Different approach: discarding instances that don’t perform well by keeping success records of how well an instance does at prediction (IB3)
  - Computes confidence interval for an instance’s success rate and for default accuracy of its class
  - If lower limit of first interval is above upper limit of second one, instance is accepted (IB3: 5%-level)
  - If upper limit of first interval is below lower limit of second one, instance is rejected (IB3: 12.5%-level)

Summary

- A framework for classification
- Classification viewed in terms of distance in feature space
- Distance-based
- A classifier as a linear model
- Nearest neighbour classifiers
- Later we will see how to extend by building on these ideas

Acknowledgements

- Material derived from slides for the book
  "Elements of Statistical Learning (2nd Ed.)" by T. Hastie, R. Tibshirani & J. Friedman. Springer (2009) http://statweb.stanford.edu/~tibs/ElemStatLearn/
- Material derived from slides for the book
  "Machine Learning: A Probabilistic Perspective" by P. Murphy MIT Press (2012)
  http://www.cs.ubc.ca/~murphyk/MLbook
- Material derived from slides for the book "Machine Learning" by P. Flach Cambridge University Press (2012) http://cs.bris.ac.uk/~flach/mlbook
- Material derived from slides for the book
  "Bayesian Reasoning and Machine Learning" by D. Barber Cambridge University Press (2012)
  http://www.cs.ucl.ac.uk/staff/d.barber/brml
- Material derived from slides for the book "Machine Learning" by T. Mitchell McGraw-Hill (1997)
  http://www- 2.cs.cmu.edu/~tom/mlbook.html
- Material derived from slides for the course "Machine Learning" by A. Srinivasan BITS Pilani, Goa, India (2016)