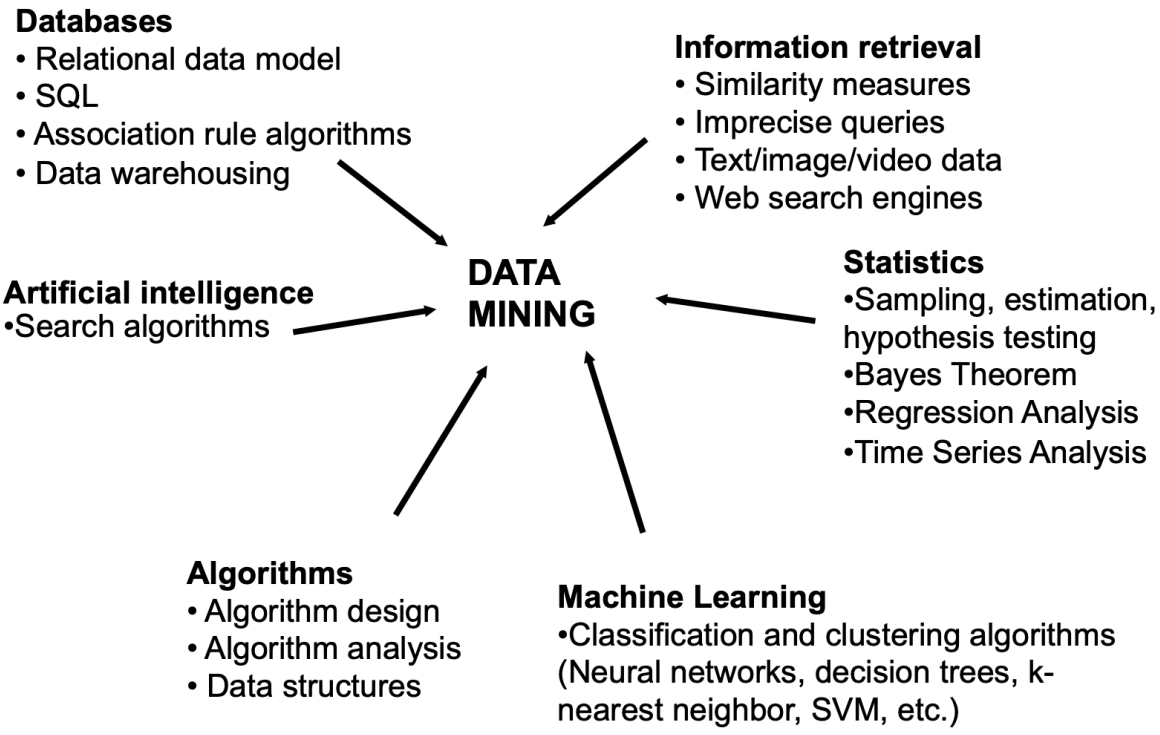
COMP5318 Notes

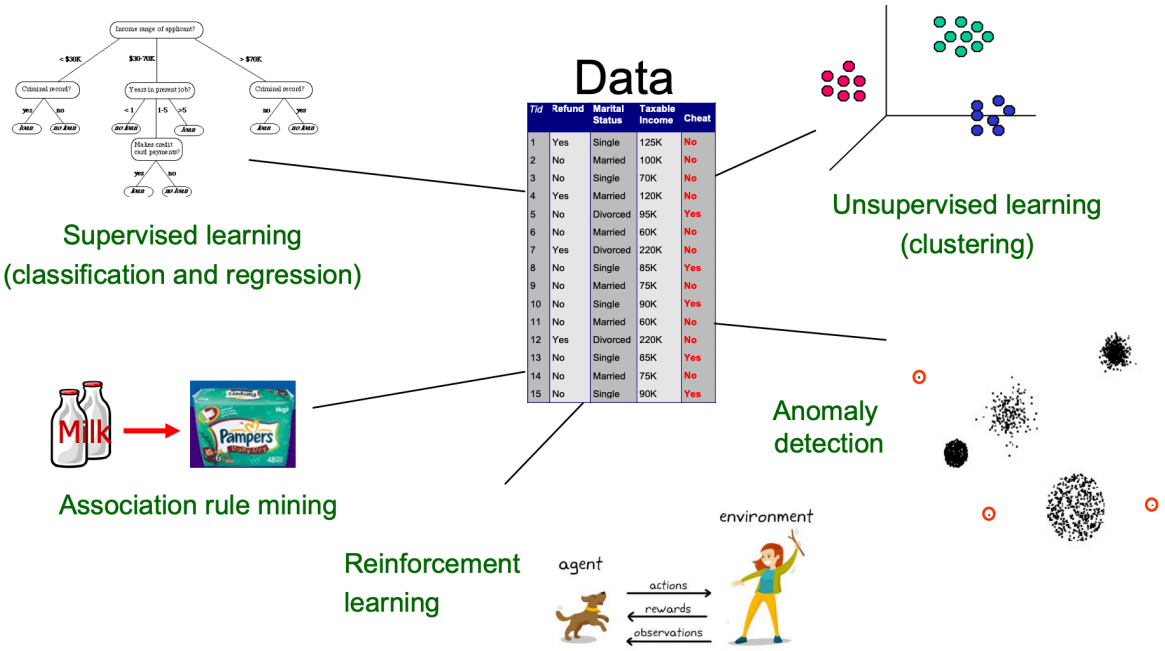
Lecture 1 - Introduction and Data Preprocessing

- **Data explosion** - society produces and stores huge amounts of data.
    - Due to automated data collection tools and sensors, mature database technology, cheaper and more powerful computers.
- **Current trend:** gather whatever data you can, whenever and wherever possible.
    - **Expectation:** it will be useful either for the purpose being collected or another purpose not yet envisioned.
- Raw data is useless on its own.
- Machine Learning (ML) and Data Mining (DM) are concerned with extracting knowledge (useful patterns) from data.
    - Patterns are meaningful, useful and actionable.
    - The process is automatic or semi-automatic.

- **ML vs DM:**
    - ML is a core part of artificial intelligence.
    - Most of the algorithms used for DM have been developed in ML.
    - DM deals with large and multidimensional data, ML not necessarily.
    - DM can be seen as applied ML - we use ML algorithms to do DM.



ML and DM Tasks



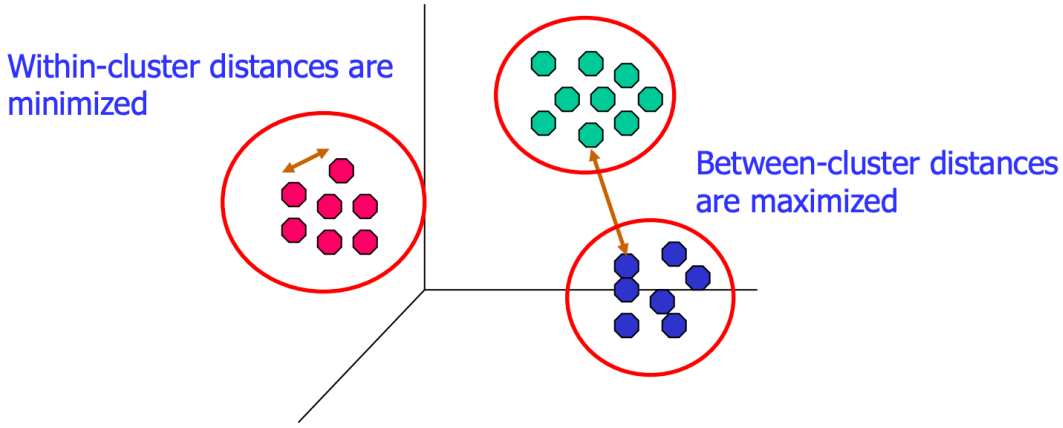
- Emphasis in this course on supervised, unsupervised and reinforcement learning.

Supervised Learning

- **Given**: a set of pre-classified (labelled) examples $\{x, y\}$ where x is the input vector and y is the target output.
- **Task**: learn a function (classifier, model) that maps $x \rightarrow y$ and can be used predictively.
  - i.e. predict the value of y given the values of x for new, unseen examples.
- It is called supervised learning since it learnt from labelled data.
- **Two Types of Supervised Learning:**
  - Classification: the variable to be predicted is categorical i.e. its values belong to a pre-specified, finite set of possibilities e.g. fraud detection in credit card transactions.
    - Given a training set of **labelled** examples with class values, build a model that can be used to predict the class of new examples.
    - To evaluate performance, we measure the accuracy on test data - porportion of correctly classified test examples.
  - Regression: the variable to be predicted is numeric e.g. predict electricity demand based off previous demand, weather data, weather forecast.

Unsupervised Learning/Clustering

- **Given**: a set of examples containing only input vectors x and no target outputs y.
- **Task**: group (cluster) the examples into a finite number of clusters so that the examples:
  - from each cluster are similar to each other
  - from different clusters are dissimilar to each other



- Examples include targeted marketing, gene clustering, document clustering, patent assessment.

Association Rule Mining

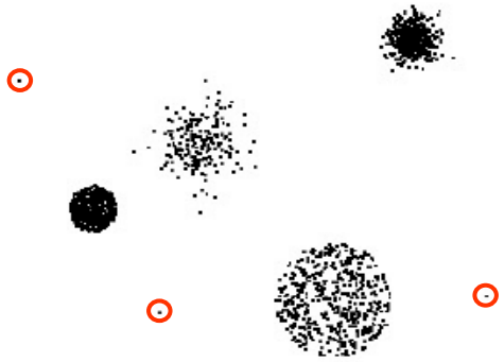
- Find combinations of items that occur together, also called **market-basket analysis**.
- Assumes transaction data.
- Sequential version of association rule mining finds frequent sequences in data
    - e.g. insurance fraud when unusual combination of insurance claims made, medical informatics where combination of patient symptoms and test results are associated with certain diseases.

TID	*Items*
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Potato Chips, Milk
4	Beer, Bread, Potato Chips, Milk
5	Coke, Potato Chips, Milk

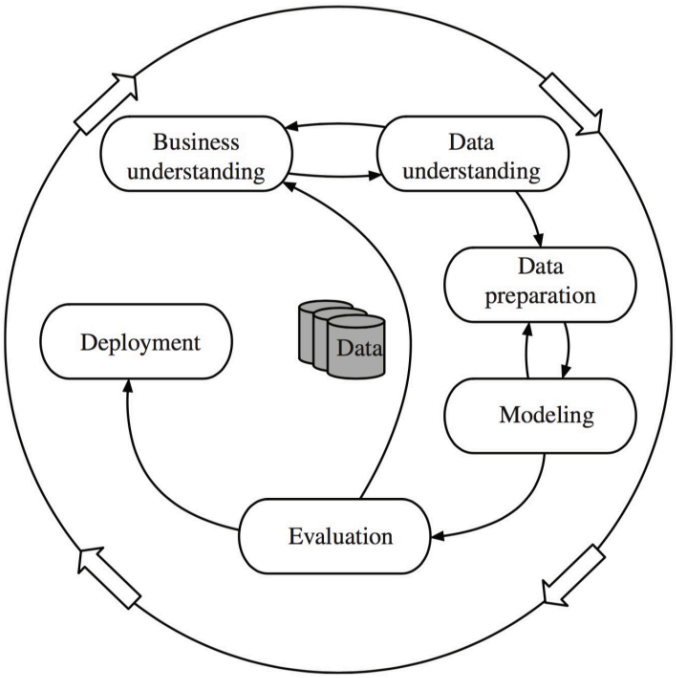
Rules discovered:
{Milk} --> {Coke}
{Potato Chips, Milk} --> {Beer}

Outlier Detection

- Outliers are examples that are significantly different than the others i.e. far away from others.
- In statistics, an outlier is typically defined as an example that differs more than 3 standard deviations from the mean of all examples.
- Detecting outliers is important for two reasons:
    - Outliers are noise and should be removed before data analysis.
    - Outliers are the goal of our DM analysis - to detect unusual behaviour such as credit card fraud detection or intrusion detection (for computer security).



The Data Mining Process



1. **Business Understanding**
    - Investigating the business objectives and requirements.
    - Deciding whether DM can be applied to meet them.
    - Determining what kind of data can be collected to build a deployable model.
2. **Data Understanding**
    - Get an initial dataset; is it suitable for further processing?
    - If the data quality is poor, collect more data based on more stringent criteria.
    - Gain insights from data and review the objective - can DM be applied?
3. **Data Preparation**
    - Preprocessing the data so that ML algorithms can be applied. This involves cleaning and various transformations.
        - Cleaning, since data in the real world is **incomplete, noisy, inconsistent**.
        - Fill in missing values, smooth noisy data by removing errors or outliers, resolve inconsistencies in codes and names.
        - Transformation: convert to common format, transform to new format, perform normalisation, dimensionality reduction and feature selection.
4. **Modelling**

- Building ML models such as a prediction model.

NOTE: 3 and 4 go hand in hand and there are many iterations e.g. model informs the use of different preprocessing, use different feature selection and dimensionality reduction and rebuiild the model.

---

5. **Evaluation**
    - How good is the performance? Accuracy, F1 measure etc.
    - Are the patterns meaningful and useful, or just reflecting spurious regularities?
    - If the performance is poor, reconsider the project and return to step 1)
    - If the performance is good, deploy it in practice.
6. **Deployment**
    - Typically requires integration into a larger software system by software engineers.
    - May be necessary to re-implement the model in a different programming language.

Data Basics

- Data is a collection of examples (also called instances, records, observation, objects).
- Examples are described with attributes (features, variables).

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- **Nominal (Categorical)** - their values belong to a pre-specified, finite set of possiblities.
- **Numeric (Continuous)** - their values are numbers.
- Other types of data include:

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

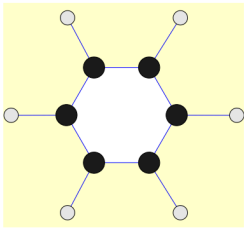
data matrix

TID	Items
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

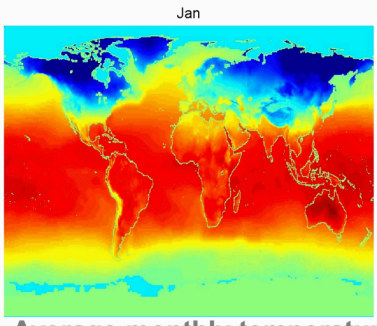
transaction data

```
GGTTCCGCCTTCAGCCCCGCGCC
CGCAGGGCCCGCCCCGCGCCGTC
GAGAAGGGCCCGCCTGGCGGGCG
GGGGGAGGCGGGGCCGCCCGAGC
CCAACCGAGTCCGACCAGGTGCC
CCCTCTGCTCGGCCTAGACCTGA
GCTCATTAGGCGGCAGCGGACAG
GCCAAGTAGAACACGCGAAGCGC
TGGGCTGCCTGCTGCGACCAGGG
```

Sequential
(e.g. genetic sequence)



graph
(e.g. molecular structure)



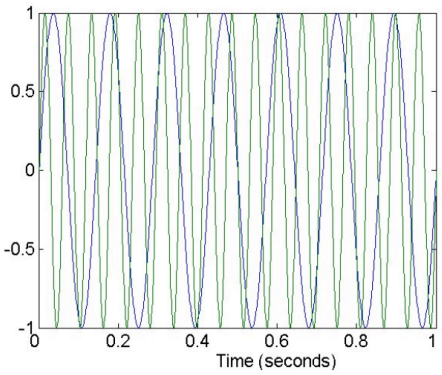
spatio-temporal

Data Cleaning

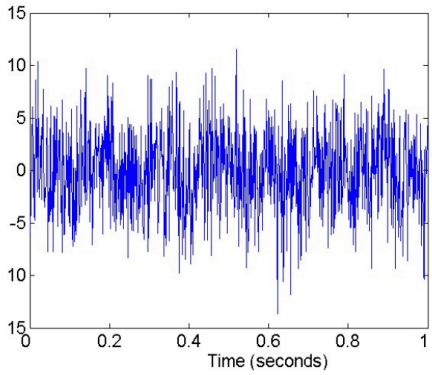
- Data is not perfect, and human errors when entering data or limitations of measuring instruments, flaws in the data collection process all impact the quality.

Noise

- **Distortion of Values:** higher distortion may cause the shape of the signal to be lost e.g. distortion of human voice when talking on a poor phone line.



voice



voice + noise

- **Addition of Spurious Examples**: some are far from the other examples, being outliers, others are mixed with the non-noisy data.

(a) Three groups of points. (b) With noise points (+) added.

- **Inconsistent and Duplicate Data**: e.g. negative weight and height values, non-existing zip codes, 2 records for the same person - need to be detected and corrected.
    - Typically easier to detect and correct than the other two types of noise

---

- REDUCING NOISE TYPES 1) AND 2):
    - Using signal and image processing and outlier detection techniques before DM.
    - Using ML algorithms that are more robust to noise - give acceptable results in presence of noise.

Dealing with Missing Values

1. If there is a small % with missing values, ignore all examples.
2. Estimate the missing values using the remaining values
    - **Nominal Attributes** - replace the missing values for attribute A with:
        - the most common value for A
        - the most common value among the examples with the same class, given supervised learning.
    - **Numerical** - replace with the average value of the nearest neighbour i.e. the most similar examples.

Data Preprocessing

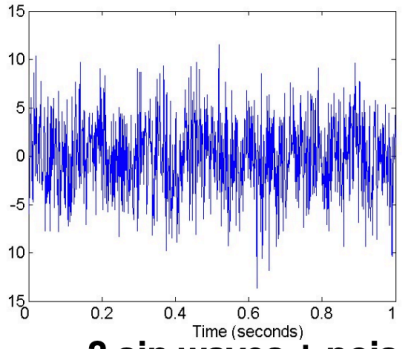
- Also includes dimensionality reduction.

Data Aggregation

- Combining two or more attributes into one. **Purpose:**
    - Data reduction - less memory and computation time; may allow the use of computationally more expensive ML algorithms.
    - Change of scale - provides higher level view e.g. cities aggregated into states or countries. '
    - More stable data - aggregated data is less variable than non-aggregated e.g. consumed daily food into weekly food to get a more reliable understanding of diet (carbohydrates, fat, protein etc).
    - **Disadvantage**: potential loss of interesting details.

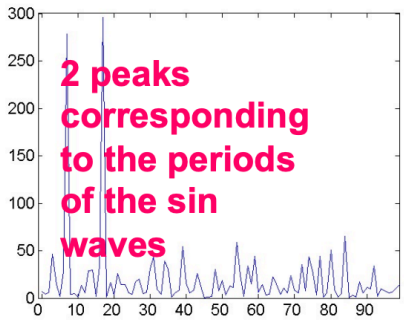
Feature Extraction

- Feature extraction is the creation of features from raw data; it requires domain expertise e.g. classifying images into outdoors or indoors.
    - Raw Data: colour value for each pixel
    - Extracted Features: colour histogram, dominant colour, edge histogram etc.
- May require mapping data into a new space, then extracting features. The new space may reveal important characteristics.



2 sin waves + noise

Fourier transform →



power spectrum

Feature Subset Selection

- The process of removing irrelevant and redundant features and selecting a small set of features that are necessary and sufficient for good classification.
    - Very important for successful classification.
- Good feature selection typically improves accuracy.
- **Using less features also means:**
    - Faster building of the classifiers i.e. reduces computational cost.
    - (Often) more compact and easier to interpret classification rule.

Feature Subset Selection Methods

- Brute Force - try all possible combinations of features as input to a ML algorithm and select the best one (impossble).
- Embedded - some ML algorithms can automatically select features e.g. decision trees.
- Filter - select features before the ML algorithm is run; the feature selection is independent of the ML algorithm.
    - Based on statistical measures e.g. information gain, mutual information, odds ratio etc.
    - Correlation-based feature selection, Relief.
- Wrapper - select the best subset for a given ML algorithm; use the ML algorithm as a black box to evaluate different subsets and select the best.

Feature selection is well studied in ML and there are many excellent methods.

Feature Weighting

- Can be used instead of feature reduction or in conjunction with it.
- The more important features are assigned a higher weight, the less important, lower.
    - Manually, based on domain knowledge
    - Automatically, some classification algorithms do it e.g. boosting, or may do it if this option is selected (k-nearest neighbour).
- **Key Idea**: features with higher weights play more important role in the construction of the ML model.

Converting Attributes From One Type To Another

- Converting numeric attributes to nominal (**discretization**).
- Converting numeric and nominal attributes to binary attributes (**binarization**).
    - Needed as some ML algorithms work only with numeric, nominal or binary attributes.

Binarization

- Converting categorical and numeric attributes into binary.
    - There is no best method, the best one is the one that works best for a given ML algorithm - all possibilities cannot be evaluated.
- Simple technique
    - Categorical attribute $\rightarrow$ Integer $\rightarrow$ Binary
    - Numeric Attribute $\rightarrow$ Categorical $\rightarrow$ Integer $\rightarrow$ Binary

Table 2.5. Conversion of a categorical attribute to three binary attributes.

Categorical Value	Integer Value	x_1	x_2	x_3
awful	0	0	0	0
poor	1	0	0	1
OK	2	0	1	0
good	3	0	1	1
great	4	1	0	0

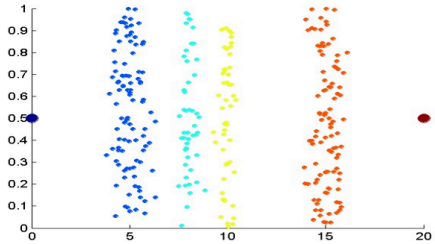
Table 2.6. Conversion of a categorical attribute to five asymmetric binary attributes.

Categorical Value	Integer Value	x_1	x_2	x_3	x_4	x_5
awful	0	1	0	0	0	0
poor	1	0	1	0	0	0
OK	2	0	0	1	0	0
good	3	0	0	0	1	0
great	4	0	0	0	0	1

Discretization

- Converting numeric attributes into nominal.
- Two Types:
    - **Unsupervised**: class information is not used.
    - **Supervised**: class information is used.

- Decisions to be taken include how many categories (intervals), where should the splits be?



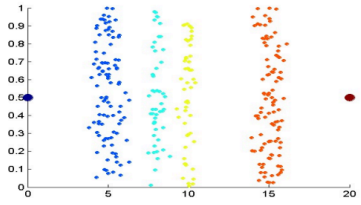
2-dim data; x and y are numeric attributes

Goal: convert x from numeric to nominal

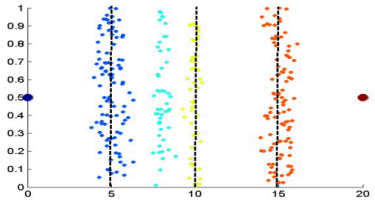
Unsupervised Discretization

- How many intervals?
  - The user specifies them, e.g. 4
- Where should the splits be?
  - 3 methods

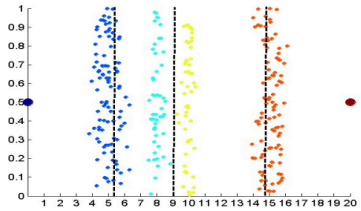
- *equal width* – 4 intervals with the same width - [0,5), [5, 10) etc.
- *equal frequency* – 4 intervals with the same number of points in each of them
- *clustering (e.g. k-means)* – 4 intervals determined by a clustering method



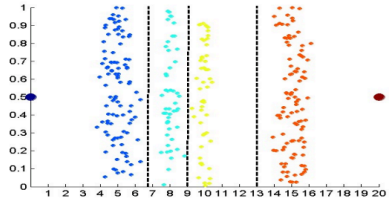
Data



Equal width



Equal frequency



K-means

Supervised Discretization (Entropy Based)

- Splits are placed so that they maximise the purity of the intervals.
- Entropy is a measure of the purity of a dataset (interval) S. The higher the entropy, the lower the purity of the dataset.

$$\text{entropy}(S) = -\sum_i P_i \cdot \log_2 P_i \text{ where } P_i \text{ is the proportion of examples from class } i$$

- Consider a split between 70 and 71. What is the entropy of the left and right datasets (intervals)?

64	65	68	69	70	71	72	73	74	75	80	81	83	85
yes	no	yes	yes	yes	no	no	no	yes	yes	no	yes	yes	no

$$entropy(S_{left}) = -\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5} = 0.722\,bits$$

$$entropy(S_{right}) = -\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9} = 0.991\,bits$$

- Total entropy of the split = weighted average of the interval entropies.

- where w_i - proportion of values in interval i, n - number of intervals.

$$\text{totalEntropy} = \sum_{i}^{n} w_i \text{ entropy}(S_i)$$

- **Algorithm**: evaluate all possible splits and choose the best one (with the lowest total entropy); repeat recursively until stopping criteria are satisfied (e.g. user specified number of splits is reached).

Normalisation and Standardisation

- Attribute transformation to a new range e.g. $[0, 1]$.
- Used to avoid the dominance of attributes with large values over attributes with small values.
- Required for distance based ML algorithms; some other algorithms also work better with normalised data.
  - E.g. Age (in years) and Annual Income (in dollars).
  - $A = [20, 40000], B = [40, 60000], D(A.B) = [20 - 40] + [40000 - 60000] = 20020$
  - Difference in income dominates and age doesn't contribute.
- **Solution**: first normalise and standardise, then calculate distance.

Formulas

- Performed for **each attribute**:
- **Normalisation** (also called min-max scaling):

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

- **Standardisation**:

$$x' = \frac{x - \mu(\text{x})}{\sigma(\text{x})}$$

- Where:
  - x is the original value.
  - x' is the new value.
  - x is all the values of the attribute, a vector.
  - $\min(\text{x})$ and $\max(\text{x})$ are the min and max values of the attribute of the vector x.
  - The other two are the mean and standard deviation of the attribute.

Examples with 2 attributes: *age* and *income*:
A=[20, 40 000]
B=[40, 60 000]
C=[25, 30 000]
…
Suppose that:
for *age*: min = 0, max=100
for *income*: min=0, max=100 000

After normalization:
A=[0.2, 0.4]
B=[0.4, 0.6]
C=[0.25, 0.3]
…
D(A,B)=|0.2-0.4| + |0.4-0.6|=0.4, i.e. *income* and *age* contribute equally

Similarity Measures

- Many ML algorithms require you to measure the similarity between 2 examples.
  - Distance
  - Correlation

Distance Measurement

- Distance measures for numeric attributes.
  - A, B are examples with attribute values $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$.
- **Euclidian Distance (L2 norm)**
  - Most frequently used.

$$D(A,B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2}$$

- **Manhattan Distance (L1 norm)**

$$D(A,B) = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|$$

- **Minkowski Distance**
  - Generalisation of Euclidian and Manhattan, where q is a positive integer

$$D(A,B) = (|a_1 - b_1|^q + |a_2 - b_2|^q + \ldots + |a_n - b_n|^q)^{1/q}$$

- **Weighted Distance**
  - Each attribute is assigned a weight according to its importance (requires domain knowledge).
  - Weighted Euclidian:

$$D(A,B) = \sqrt{w_1|a_1 - b_1|^2 + w_2|a_2 - b_2|^2 + \ldots + w_n|a_n - b_n|^2}$$

- **For Nominal Attributes**

- Various options depending on the task and type of data type; requires domain expertise. Example:

  difference =0 if attribute values are the same
  difference =1 if they are not

  Example: 2 attributes = *temperature* and *windy*
  - *temperature* values: low and high
  - *windy* values: yes and no

  A={high, no}
  B={high, yes}
  d(A,B) =$(0+1)^{1/2}$=1 (Euclidean distance)

Similarity Between Binary Vectors

- Consider the example:
  - $\mathrm{A} = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$
  - $\mathrm{B} = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1]$

- **Hamming Distance**
  - Manhattan Distance for binary vectors.
  - Counts the number of different bits.
    - $D(A, B) = 3$

$$D(A, B) = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|$$

- **Similarity Coefficients**
  - f00: number of matching 0-0 bits
  - f01: number of matching 0-1 bits
  - f10: number of matching 1-0 bits
  - f11: number of matching 1-1 bits
    - f00 = 7, f01 = 2, f10 = 1, f11 = 00
- **Simple Matching Coefficient (SMC)**
  - SMC = (f11 + f00) / num. attributes = (f11 + f00) / (f00 + f01 + f10 + f11)
  - SMC = (0+7) / (2+1+0+7) = 0.7
- **Jaccard Coefficient**
  - Counts matching f11 and ignores matching 00
  - (f11) / (f01 + f10 + f11)
  - J = 0 / (2 + 1 + 0) = 0

SMC vs Jaccard

Task: Suppose that A and B are the supermarket bills of 2 customers. Each product in the supermarket corresponds to a different attribute.

- attribute value = 1 – product was purchased
- attribute value = 0 - product was not purchased

SMC is used to calculate the similarity between A and B. Is there any problem using SMC?

Yes, SMC will find all customer transactions (bills) to be similar

Reason: The number of products that are not purchased in a transaction is much bigger than the number of products that are purchased

=> f00 will be very high (not purchased products)

- f11 will be low (purchased products)
- f00 will be much higher than f11 and its effect will be lost

SMC = (f11+f00)/(f01+f10+f11+f00)

=> More generally, the problem is that the 2 vectors A and B contain many 0s, i.e. are very *sparse* => SMC is not suitable for sparse data

A = [1 0 0 0 0 0 0 0 0 0]
B = [0 0 0 0 0 0 1 0 0 1]

- Comparing with Jaccard, J = 0 indicates that A and B are dissimilar.
  - SMC with 0.7 indicates that A and B are highly similar, which is evidently incorrect.
  - **Conclusion**: SMC is not suitable for sparse data.

Cosine Similarity

- Useful for spare data (both binary and non-binary).
  - Widely used for classification of text documents.
  - Where • refers to vector dot product and $||A||$ is the length/norm of vector A i.e. square root of $A \bullet A$

$$cos(A, B) = \frac{A \bullet B}{||A||||B||}$$

- **Geometric representation:** measures the angle between A and B.
  - Cosine similarity = 1 where angle(A, B) = 0°
  - Cosine similarity = 0 where angle(A, B) = 90°
  - Cosine similarity = -1 where angle(A, B) = 180°

Example

- Two document vectors:
  - $d_1 = 3\ 2\ 0\ 5\ 0\ 0\ 0\ 2\ 0\ 0$
  - $d_2 = 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 2$

$d_1 \bullet d_2 = 3*1 + 2*0 + 0*0 + 5*0 + 0*0 + 0*0 + 0*0 + 2*1 + 0*0 + 0*2 = 5$

$||d_1|| = (3*3+2*2+0*0+5*5+0*0+0*0+0*0+2*2+0*0+0*0)^{1/2} = (42)^{1/2} = 6.481$

$||d_2|| = (1*1+0*0+0*0+0*0+0*0+0*0+0*0+1*1+0*0+2*2)^{1/2} = (6)^{1/2} = 2.245$

$\Rightarrow \quad \cos(d_1, d_2) = 0.3150$

Correlation

- Measures linear relationship between numeric attributes.
- **Pearson Correlation Coefficient** between data objects (instances) x and y with dimensionality n.

$$\text{corr}(\mathbf{x}, \mathbf{y}) = \frac{\text{covar}(\mathbf{x}, \mathbf{y})}{\text{std}(\mathbf{x})\,\text{std}(\mathbf{y})}$$

where:

$$mean(\mathbf{x}) = \frac{\sum_{k=1}^{n} x_k}{n} \qquad std(\mathbf{x}) = \sqrt{\frac{\sum_{k=1}^{n} (x_k - mean(\mathbf{x}))^2}{n-1}}$$

$$\text{co var}(\mathbf{x}, \mathbf{y}) = \frac{1}{n-1} \sum_{k=1}^{n} (x_k - mean(x))(y_k - mean(y))$$

- Range: $[-1, 1]$
  - -1: perfect negative correlation
  - +1: perfect positive correlation
  - 0: no correlation

Ex1: corr(x,y)=?
x=(-3, 6, 0, 3, -6)
y=(1,-2, 0,-1, 2)

corr(x,y) = -1
perfect negative linear correlation

Ex2: corr(x,y)=?
x=(3, 6, 0, 3, 6)
y=(1, 2, 0, 1, 2)

corr(x,y) = +1
perfect positive linear correlation

Ex3: corr(x,y)=?
x=(-3, -2, -1, 0, 1, 2, 3)
y=(9, 4, 1, 0, 1, 4, 9)

corr(x,y) = 0
no linear correlation
However, there is a non-linear relationship: $y=x^2$

- Visual Evaluation of Correlation:

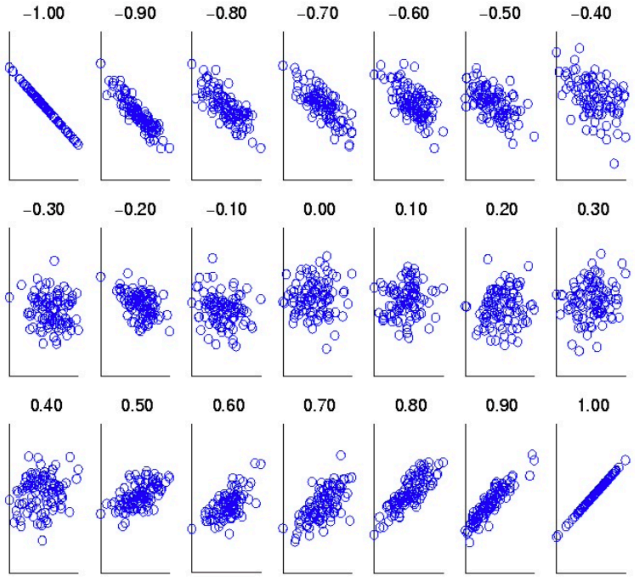


Figure 5.11. Scatter plots illustrating correlations from -1 to 1.

Lecture 2 - KNN

- In general:
  - Often very accurate.
  - Used in statistics since the 1950s.
  - Slow for big datasets.
  - Distance-based and requires normalisation.
  - Produces arbitrarily shaped decision boundaries defined by a subset of Voronoi edges.
    - Not effective for high-dimensional data (data with many features).
      - The notion of nearness is not effective in high dimensional data, although this can be solved with dimensionality reduction and feature selection.
  - Sensitive to the value of k.

1NN

- Remember all training examples and make a prediction for a new unlabelled examples by finding the nearest training example using a distance measure.
  - The class label of the nearest neighbour will be the predicted label for the new example.

Example

The dataset below consists of 4 examples described with 3 numeric features (a1, a2 and a3); the class has 2 values: yes and no.

What will be the prediction of the Nearest Neighbor algorithm using the Euclidean distance for the following new example: a1=2, a2=4, a3=2?

	a1	a2	a3	class
1	1	3	1	yes
2	3	5	2	yes
3	3	2	2	no
4	5	2	3	no

D(new, ex1) = sqrt[(2-1)^2+(4-3)^2+(2-1)^2]=sqrt(3) yes
D(new, ex2) = sqrt[(2-3)^2+(4-5)^2+(2-2)^2]=sqrt(2) yes
D(new, ex3) = sqrt[(2-3)^2+(4-2)^2+(2-2)^2]=sqrt(5) no
D(new, ex4) = sqrt[(2-5)^2+(4-2)^2+(2-3)^2]=sqrt(14) no

The closest nearest neighbor is ex. 2, hence the nearest neighbor algorithm predicts class=yes for the new example

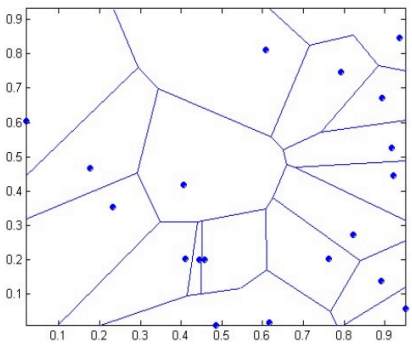
Computational Complexity

- Training is fast since no model is built; just storing training examples.
- Classification (prediction for a new example):
  - Compare each unseen example with each training example.
  - If m training examples with dimensionality $n \rightarrow$ lookup for 1 unseen example takes $m \times n$ computations i.e. $O(mn)$.
- **Memory Requirements:** you need to remember each training example, $O(mn)$.
- Impractical for large data due to time and memory requirements.
  - However, there are variations allowing you to find the nearest neighbours more efficiently.
  - This is done using more efficient data structures such as KD-trees and ball trees.

Decision Boundary of 1NN

- Nearest neighbour classifiers porduce decision boundaries with an arbitrary shape.
  - 1NN boundary is formed by the edges of the Voronoi diagram that separates the points of the two classes.
- **Voronoi region:** each training example has an associated Voronoi region; it contains the data points for which this is the closest example.

Voronoi diagram



KNN

- k-Nearest Neighbour uses k closest examples and majority voting is used to take the decision.
- k-Nearest Neighbour is very sensitive to the value of k.
  - Rule of Thumb: $k \leq \sqrt{\#\text{training\_examples}}$
- Using more nearest neighbours increases the robustness to noisy examples.
  - k-Nearest Neighbour can be used not only for classification but for regression as well.
    - The prediction is the average value of the class values of the nearest neighbnours.

Example

The dataset below consists of 4 examples described with 3 numeric features (a1, a2 and a3); the class has 2 values: yes and no.

What will be the prediction of the 3-Nearest Neighbor algorithm using the Euclidean distance for the following new example: a1=2, a2=4, a3=2?

	a1	a2	a3	class
1	1	3	1	yes
2	3	5	2	yes
3	3	2	2	no
4	5	2	3	no

D(new, ex1) = sqrt[(2-1)^2+(4-3)^2+(2-1)^2]=sqrt(3) yes
D(new, ex2) = sqrt[(2-3)^2+(4-5)^2+(2-2)^2]=sqrt(2) yes
D(new, ex3) = sqrt[(2-3)^2+(4-2)^2+(2-2)^2]=sqrt(5) no
D(new, ex4) = sqrt[(2-5)^2+(4-2)^2+(2-3)^2]=sqrt(14) no

The closest 3 nearest neighbors are ex.2 (yes), ex.1 (yes) and ex.3 (no); the majority class is yes. Hence, 3-Nearest Neighbor predicts class =yes

Weighted Nearest Neighbour

- **Idea**: Closer neighbours should count more than distant neighbours.
- **Algorithm**:
  - Find the k nearest neighbours.
  - Weight their contribution based on their distance to the new example.
    - Biggest weight if closer, smaller weight if they are further.
  - E.g. the vote can be weighted according to the distance-weight $w = 1/d^2$.

KNN Example Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

##### SETTING UP AND SPLITTING #####
```

```
from sklearn.datasets import load_iris
iris_dataset = load_iris() # loading iris dataset

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)

##### VISUALISATION #####

# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# create a scatter matrix from the dataframe, color by y_train
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15),
                           marker='o', hist_kwds={'bins': 20}, s=60,
                          alpha=.8);

##### NORMALISATION #####

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler() # creating an object
scaler.fit(X_train) # calculate min and max value of the training data

# apply normalization to the training set
X_train_norm = scaler.transform(X_train)

# Important: MinMaxScaler (and the other scalers) always apply the same transformation
to the training and test set, based on the min and max values of the training set
X_test_norm = scaler.transform(X_test)

##### STANDARDISATION #####
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler() # creating an object
scaler.fit(X_train) # calculate mean and sd value of the training data

# apply standardization to the training set
X_train_standard = scaler.transform(X_train)

# Apply standardization to the test set
X_test_standard = scaler.transform(X_test)

##### KNN #####

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Accuracy on test set: {:.2f}".format(accuracy_score(y_test, y_pred)))
```

```
##### PREDICTING NEW EXAMPLE #####

X_new = np.array([[5, 2.9, 1, 0.2]])
prediction = knn.predict(X_new)
print("Prediction:", prediction)
```

Lecture 3 - Linear and Logistic Regression, Overfitting and Regularisation

- Go to Lecture Slides

Lecture 4 - Naive Bayes, Numerical Logistic Regression

Bayes Theorem

- Bayesian classifiers are statistical classifiers. They can predict the **class membership probability** i.e. the probability that a given example belongs to a particular class, based on the Bayes Theorem.
- **Given a hypothesis H and evidence E for this hypothesis, then the probability of H given E is**:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

- For instances of fruit, let E is red and round and H is the hypothesis that E is an apple.
  - $P(H)$ is the probability that any given example is an apple, regardless of how it looks. Called the **prior** probability of H.
  - $P(E)$ is the prior probability of E, that an example from the fruit data set is red and round.
  - $P(H|E)$ is the probability that E is an apple, given that we have seen E is red and round. Known as the posteriori probability of H conditioned on E. This is based on more information than the prior probability, which is independent of E.
  - $P(E|H)$ is the posteriori probability of E conditioned on H - the probability that E is red and round given that we know that E is an apple.

Naive Bayes Algorithm

- The Bayes Theorem can be applied for classification tasks - Naive Bayes algorithm.
- While 1R makes decisions based on a single attribute, Naive Bayes uses all attributes and allows them to make contributions to the decision that are equally important and independent of one another.
- **Assumptions of Naive Bayes Algorithm**:
  - **Independence assumption:** the values of the attributes are conditionally independent of each other, given the class (for each class value).
  - **Equal importance assumption:** all attributes are equally important.
- These are unrealistic assumptions (hence why it is called Naive Bayes), but these assumptions lead to a simple method which works surprisingly well in practice.

Weather Example

- **Given**: weather data.

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

- **Task**: use Naive Bayes to predict the class (yes or no) of the new example.

outlook=sunny, temperature=cool,
humidity=high, windy=true

- H is play = yes, and also play = no i.e. there are two hypotheses.
- E Is the new example.

How Do We Use the Bayes Theorem for Classification?

- Calculate $P(H|E)$ for each H (class) i.e. $P(\text{yes}|E)$ and $P(\text{no}|E)$.
- Compare them and assign E to the class with the highest probability.
- To calculate the three parts of Bayes Theorem, we use the given data (this is the training phase of the classifier).

$$P(yes \mid E) = \frac{P(E \mid yes)P(yes)}{P(E)}$$
$$P(no \mid E) = \frac{P(E \mid no)P(no)}{P(E)}$$

where E
outlook=sunny,temperature=cool,
humidity=high, windy=true

1) How to calculate P(E|yes) and P(E|no) ?
Let's split the evidence E into 4 smaller pieces of evidence:

- **E1 =** **outlook=sunny,** **E2 =** **temperature=cool**
- **E3 =** **humidity=high,** **E4 =** **windy=true**

- Using the Naive Bayes independence assumption, $E1$ to $E4$ are independent given the class. Their combined probability is obtained by the multiplication of per-attribute probabilities. The final equation is thus:

$$P(\text{yes}|E) = \frac{P(E_1|\text{yes})P(E_2|\text{yes})P(E_3|\text{yes})P(E_4|\text{yes})P(\text{yes})}{P(E)}$$
$$P(\text{no}|E) = \frac{P(E_1|\text{no})P(E_2|\text{no})P(E_3|\text{no})P(E_4|\text{no})P(\text{no})}{P(E)}$$

- Numerator: the probabilities will be estimated from the data.
- Denominator: the two denominators are the same, and since we are comparing the two fractions, we **DO NOT** need to calculate $P(E)$.

E1 = outlook=sunny, E2 = temperature=cool
E3 = humidity=high, E4 = windy=true

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

	outlook		temperature			humidity			windy			play	
	yes	no		yes	no		yes	no		yes	no	yes	no
sunny	2	3	hot	2	2	high	3	4	false	6	2	9	5
overcast	4	0	mild	4	2	normal	6	1	true	3	3		
rainy	3	2	cool	3	1								
sunny	2/9	3/5	hot	2/9	2/5	high	3/9	4/5	false	6/9	2/5	9/14	5/14
overcast	4/9	0/5	mild	4/9	2/5	normal	6/9	1/5	true	3/9	3/5		
rainy	3/9	2/5	cool	3/9	1/5								

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

proportions of days when play is yes

proportions of days when humidity is normal and play is yes
i.e. the probability of humidity to be normal given that play=yes

$P(yes \mid E) = ?$ $P(yes \mid E) = \frac{P(E_1 \mid yes)P(E_2 \mid yes)P(E_3 \mid yes)P(E_4 \mid yes)P(yes)}{P(E)}$

	outlook		temperature			humidity			windy			play	
	yes	no		yes	no		yes	no		yes	no	yes	no
sunny	2	3	hot	2	2	high	3	4	false	6	2	9	5
overcast	4	0	mild	4	2	normal	6	1	true	3	3		
rainy	3	2	cool	3	1								
sunny	2/9	3/5	hot	2/9	2/5	high	3/9	4/5	false	6/9	2/5	9/14	5/14
overcast	4/9	0/5	mild	4/9	2/5	normal	6/9	1/5	true	3/9	3/5		
rainy	3/9	2/5	cool	3/9	1/5								

⇒ **$P(E_1|yes)=P(outlook=sunny|yes)=2/9$**
$P(E_2|yes)=P(temperature=cool|yes)=3/9$
$P(E_3|yes)=P(humidity=high|yes)=3/9$
$P(E_4|yes)=P(windy=true|yes)=3/9$

- **P(yes) =? - the probability of a yes without knowing any E, i.e. anything about the particular day; the prior probability of yes; P(yes) = 9/14**

- By substituting the respective evidence probabilities, and then similarly calculating $P(\text{no}|E)$:

$$P(\text{yes}|E) = \frac{\frac{2}{9}\frac{3}{9}\frac{3}{9}\frac{3}{9}\frac{9}{14}}{P(E)} = \frac{0.0053}{P(E)}$$

$$P(\text{no}|E) = \frac{\frac{3}{5}\frac{1}{5}\frac{4}{5}\frac{3}{5}\frac{5}{14}}{P(E)} = \frac{0.0206}{P(E)}$$

$$P(\text{no}|E) > P(\text{yes}|E)$$

$$\text{Therefore, play} = \text{no is more likely than play} = \text{yes.}$$

Another Example

- Consider a volleyball game between team A and team B.
  - Team A has won 65% of the time and team B has won 35%.
  - Among the games won by team A, 30% were when playing on team B's court.
  - Among the games won by team B, 75% were when playing at home.
- If team B is hosting the next match, which team is most likely to win?

Solution

- Host: $\{A, B\}$
- Winner: $\{A, B\}$
- Using NB, the task is to compute and compare the two probabilities:

$$P(winner = A \mid host = B) = \frac{P(host = B \mid winner = A)P(winner = A)}{P(host = B)}$$

$$P(winner = B \mid host = B) = \frac{P(host = B \mid winner = B)P(winner = B)}{P(host = B)}$$

P(winner=A)= ? //probability that A wins =0.65

P(winner=B)=? //probability that B wins =0.35

P(host=B|winner=A)=? //probability that team B hosted the match, given that team A won =0.30

P(host=B|winner=B)=? //probability that team B hosted the match, given that team B won =0.75

$$P(winner = A \mid host = B) = \frac{P(host = B \mid winner = A)P(winner = A)}{P(host = B)} = \\ = \frac{0.3 * 0.65}{P(host = B)} = 0.195$$

$$P(winner = B \mid host = B) = \frac{P(host = B \mid winner = B)P(winner = B)}{P(host = B)} = \\ = \frac{0.75 * 0.35}{P(host = B)} = 0.2625$$

=>NB predicts team B

Three More Things About Naive Bayes

Problem: Probability Values of 0

- Suppose that the training data was different: `outlook=sunny` had always occured together with `play=no`.
- Then $P(\text{outlook}=\text{sunny}|\text{yes}) = 0$ and so:

$$P(yes \mid E) = \frac{\underset{=0}{P(E_1 \mid yes)} P(E_2 \mid yes) P(E_3 \mid yes) P(E_4 \mid yes) P(yes)}{P(E)}$$

- Regardless of other probabilities, the final probability will always be 0.
- This is not good since the other probabilities are completely ignored due to the multiplication by 0. The prediction for `outlook=sunny` will always be no, regardless of the other attributes.

Solution

- Assume that our training data is so large that adding 1 to each count would not make a difference in calculating the probabilities, but will avoid the case of 0 probability.
- This is called the Laplace Correction/Estimator.

Example

- Add 1 to the numerator and k to the denominator, where k is the number of attribute values for the given attribute.
- Example:
  - A dataset with 2000 examples, 2 classes: `buy_Mercedes=yes` and `buy_Mercedes=no`; 1000 examples in each class.

- One of the attributes is `income` with 3 values: `low`, `medium` and `high`.
- For class `buy_Mercedes=yes`, there are 0 examples with `income=low`, 10 with `income=medium` and 990 with `income=high`.

- Probabilities without the Laplace correction for class `yes`: 0/1000=0, 10/1000=0.01, 990/1000=0.99.
- Probabilities with the Laplace correction: 1/1003=0.001, 11/1003=0.011, 991/1003=0.988.
- The correct probabilities are close to the adjusted probabilities, yet the 0 probability value is avoided!

Another Example

	yes	...
sunny	0	...
overcast	4	...
rainy	3	...
		...
sunny	0/9	...
overcast	4/9	...
rainy	3/9	...

P(sunny|yes)=0/9 →problem
P(overcast|yes)=4/9
P(rainy|yes)=3/9

Laplace correction

- **Assumes that there are 3 more examples from class *yes*, 1 for each value of *outlook***
- **This results in adding 1 to the numerator and 3 to the denominator of all probabilities**
- **Ensures that an attribute value which occurs 0 times will receive a nonzero (although small) probability**

$$P(sunny \mid yes) = \frac{0+1}{9+3} = \frac{1}{12}$$

$$P(overcast \mid yes) = \frac{4+1}{9+3} = \frac{5}{12}$$

$$P(rainy \mid yes) = \frac{3+1}{9+3} = \frac{4}{12}$$

Generalisation of the Laplace Correction: M-Estimate

- Add a small constant m to each denominator and mp_i to each numerator, where p_i is the prior probability of the i values of the attribute.

$$P(sunny \mid yes) = \frac{2+mp_1}{9+m} \quad P(overcast \mid yes) = \frac{4+mp_2}{9+m} \quad P(rainy \mid yes) = \frac{3+mp_3}{9+m}$$

- Note that $p_1 + p_2 + \ldots + p_m = 1$ where m is the number of attribute values.
- Advantage of using prior probabilities: it is rigorous.
- Disadvantage: computationally expensive to estimate prior probabilities.
- Large m: the prior probabilities are very important compared with the new evidence coming in from the training data; small m is less important.
- Typically, we assume that each attribute value is equally probable i.e. $p_1 = p_2 = \ldots = p_n = 1/n$. The Laplace correction is a special case of the m-estimate, where $p_1 = p_2 = \ldots = p_n = 1/n$ and $m = n$. Thus, 1 is added to the numerator and m to the denominator.

Handling Missing Values

- If there are missing attribute values in the new example, do not include this attribute.

- **e.g. outlook=?, temperature=cool, humidity=high, windy=true**
- **Then:**

$$P(yes \mid E) = \frac{\frac{3}{9}\cdot\frac{3}{9}\cdot\frac{3}{9}\cdot\frac{9}{14}}{P(E)} = \frac{0.0238}{P(E)} \qquad P(no \mid E) = \frac{\frac{1}{5}\cdot\frac{4}{5}\cdot\frac{3}{5}\cdot\frac{5}{14}}{P(E)} = \frac{0.0343}{P(E)}$$

- **outlook is not included. Compare these results with the previous results!**
- **As one of the fractions is missing, the probabilities are higher but the comparison is fair - there is a missing fraction in both cases**

- Missing attribute value in a training example - do not include this value in the counts. Calculate the probabilities based on the number of values that actually occur and not on the total number of training examples.

Handling Numeric Attributes

outlook			temperature		humidity		windy			play	
	yes	*no*	*yes*	*no*	*yes*	*no*		*yes*	*no*	*yes*	*no*
sunny	2	3	83	85	86	85	false	6	2	9	5
overcast	4	0	70	80	96	90	true	3	3		
rainy	3	2	68	65	80	70					
			64	72	65	95					
			69	71	70	91					
			75		80						
			75		70						
			72		90						
			81		75						
sunny	2/9	3/5	*mean*	73	74.6	*mean*	79.1	86.2	false	6/9	2/5	9/14	5/14
overcast	4/9	0/5	*std dev*	6.2	7.9	*std dev*	10.2	9.7	true	3/9	3/5		
rainy	3/9	2/5											

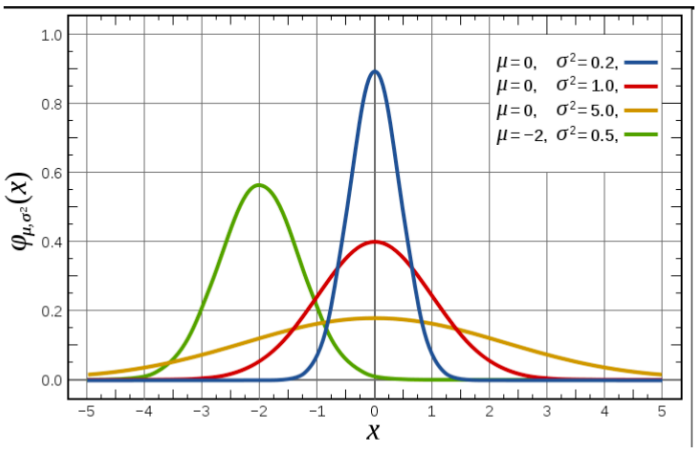
numeric

• We would like to classify the following new example:
outlook=sunny, temperature=66, humidity=90, windy=true

• Question: How to calculate
P(temperature=66|yes)=?, P(humidity=90|yes)=?
P(temperature=66|no)=?, P(humidity=90|no) ?

- Answer: by assuming the numerical values have a normal (Gaussian, bell curve) probability distribution and using the probability density function.
- For a normal distribution with mean μ and standard deviation σ, the probability function is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$



- A probability density function of a continuous random variable is closely related to probability but not exactly the probability (e.g. the probability that x is exactly 66 is 0).
- The probability that a given value $x \in (x - \epsilon/2, x + \epsilon/2)$ is $\epsilon \times f(x)$.
- Given that 73 represents the mean for temperature for `class=yes`:

$$f(temperature = 66 \mid yes) = \frac{1}{6.2\sqrt{2\pi}} e^{-\frac{(66-73)^2}{2*6.2^2}} = 0.034$$

μ for temp. for play=yes

σ for temp. for play=yes

- Similarly: $f(humidity = 90 \mid yes) = 0.0221$

$$P(yes|E) = \frac{\frac{2}{9}\,0.034\;0.0221\;\frac{3}{9}\;\frac{9}{14}}{P(E)} = \frac{0.000036}{P(E)}$$

$$P(no|E) = \frac{\frac{3}{5}\,0.0279\;0.038\;\frac{3}{5}\;\frac{5}{14}}{P(E)} = \frac{0.000137}{P(E)}$$

- P(no|E) >P(yes|E) => Naïve Bayes predicts play=no

Advantages of NB

- Simple approach – the probabilities are easily computed due to the independence assumption.
- Clear semantics for representing, using and learning probabilistic knowledge.
- Excellent computational complexity:
  - Requires 1 scan of the training data to calculate all statistics (for both nominal and continuous attributes assuming normal distribution).
  - $O(pk)$, p - # training examples, k-valued attributes.
- In many cases outperforms more sophisticated learning methods - always try the simple method first!
- Robust to isolated noise points as such points are averaged when estimating the conditional probabilities from data.

Disadvantages of NB

- Correlated attributes reduce the power of Naive Bayes.
  - Violation of the independence assumption.
  - Solution: apply feature selection beforehand to identify and discard correlated (redundant) attributes.
- Normal distribution assumption for numeric attributes - many features are not normally distributed. Solutions:
  - Discretize the data first, i.e. numerical $\rightarrow$ nominal attributes.
  - Use other probability density functions, e.g. Poisson, binomial, gamma, etc.
  - Transform the attribute using a suitable transformation into a normally distributed one (sometimes possible).
  - Use kernel density estimation – doesn't assume any particular distribution.

Evaluating and Comparing Classifiers

- Our goal with classifiers are that they generalise well on new data i.e. they correctly classify new data, unseen during traing.

How to Evaluate the Performance of Classifiers

Accuracy and Error Rate

- **Accuracy**: proportion of correctly classified examples.
- **Error Rate**: complimentary to accuracy, the proportion of incorrectly classified examples.
- These two sum to 1, and since it is typically in percentage, thus sum to 100%.
- They are evaluated on the training and test set.
    - Accuracy on training data is overly optimistic, not a good indicator of performance on future data.
    - Accuracy on test data is the performance measure used.

Making the Most Out of Data

- Generally:
    - The larger the training data, the better the classifier.
    - The larger the test data, the better the accuracy estimate.
- Dilemma: ideally, we want to use as much data as possible for
    - Training to get a good classifier
    - Testing to get a good accuracy estimate
- Once the evaluation is completed, all the data can be used to build the final classifier.
    - Training, validation and test sets are joined together, a classifier is built using all of them for actual use by a customer.
    - The accuracy of the classifier must be quoted to the customer based on the test data.

Holdout Procedure

- Split data into two independent (non-overlapping) sets: training and test (usually 2/3 and 1/3).
- Use the training data to build the classifier.
- Use the test data is evaluate how good the classifier is by calculating performance measures such as accuracy.

outlook	temp.	humidity	windy	play
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	86	false	yes
rainy	70	96	false	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	false	yes
sunny	75	70	true	yes
overcast	73	90	true	yes
overcast	81	75	false	yes
rainy	71	91	true	no

training data: 9 examples (2/3)

test data: 5 examples (1/3)

Validation Set

- Sometimes we need a 3rd set: the validation set. Some classification methods including decisioin trees and neural networks operate in two stages:
    - Stage 1: build the classifier.
    - Stage 2: tune its parameters.
- The test data should not be used in any way to create the classifier, including parameter tuning.
- Proper procedure uses three non-overlapping data sets.
    - 1. Training set - to build the classifier
    - 2. Validation set - to tune the parameters
    - 3. Test set - to evaluate accuracy
- Examples:
    - DTs: training set used to build the tree, validation set for pruning, test set for performance.
    - NNs: validation set to stop the training and prevent overtraining.

Stratification

- An improvement of the holdout method.
- **Problem**: the hldout method reserves a certain amount for testing and uses the remainder for testing, however, the training set may not be representative of all classes (if all examples of a certain class is missing in the training set, the classifier cannot learn to predict this class).
- **Solution**: stratification ensures that each class is represented with approximately equal proportions in both data sets (training and test).
- Stratification is used together with the evaluation method e.g. holdout or cross validation.

Repeated Holdout Method

- Holdout can be made more reliable by repeating the process:
    - E.g. ten iterations, in each, a certain proportion (e.g. 2/3) is randomly selected for training (possibly with stratification) and the remainder is used for testing.
    - The accuracy on the different iterations are average to yield an overall accuracy.
- Can be improved by ensuring that the test sets do not overlap using cross validation.

Cross Validation

- S-fold cross validation:
    - Data is split into S subsets of equal size.
    - A classifier is built S times. Each time, the testing is on one segment and the training is on the remaining $S-1$ segments.
    - Average accuracies for each run to calculate the overall accuracy.
- The standard is 10-fold cross-validation.

Leave-One-Out Cross-Validation

- n-fold cross-validation, where n is the number of examples in the data set. We thus need to build the classifier ten times.
- **Advantages**:
  - The greatest possible amount of data is used for training, increasing the chance of building an accurate classifier.
  - Deterministic procedure - no random sampling is involved (no point in repeating the procedure since the same results are obtained).
- **Disadvantage**: High computational cost, although it is useful for small data sets.

How to Compare Classifiers

- Given two classifiers $C1$ and $C2$, we want to find out which one is better on a given task. Comparing the 10-fold CV estimates might give a general picture, but we aren't able to definitely say whether the difference is significant.
- A paired t-test can be used.

	C1	C2	
Fold 1	**95%**	**91%**	d_1=|95-91|=4
…			
Fold 2	**82%**	**85%**	d_2=|82-85|=3
mean	**91.3%**	**89.4%**	d_{mean}=3.5

$$\sigma = \sqrt{\frac{\sum_i^k (d_i - d_{mean})^2}{k-1}}$$

1. **Calculate the differences d_i**
2. **Calculate the standard deviation of the differences (an estimate of the true standard deviation)**
   If k is sufficiently large, d_i is normally distributed
3. **Calculate the confidence interval Z:**

$$Z = d_{mean} \pm t_{(1-\alpha)(k-1)} \frac{\sigma}{\sqrt{k}}$$

t is obtained from a probability table
1-α – confidence level
k-1 – degree of freedom

4. **Interval contains 0 – difference not significant, else significant**

Suppose that:

- **We use 10 fold CV => k=10**
- d_{mean}=3.5; $\frac{\sigma}{\sqrt{k}} = 0.5$
- **We are interested in significance at 95% confidence level**
- **Then:** $Z = 3.5 \pm 2.26 \times 0.5 = 3.5 \pm 1.13$

=> The interval does not contain 0 => the difference is statistically significant

	$(1-\alpha)$				
$k-1$	0.8	0.9	0.95	0.98	0.99
1	3.08	6.31	12.7	31.8	63.7
2	1.89	2.92	4.30	6.96	9.92
4	1.53	2.13	2.78	3.75	4.60
9	1.38	1.83	2.26	2.82	3.25
14	1.34	1.76	2.14	2.62	2.98
19	1.33	1.73	2.09	2.54	2.86
24	1.32	1.71	2.06	2.49	2.80
29	1.31	1.70	2.04	2.46	2.76

Confusion Matrix

2 class prediction (classes *yes* and *no*) – 4 different outcomes

Confusion matrix:

examples	# assigned to class yes	# assigned to class no
# from class yes	true positives (tp)	false negatives (fn)
# from class no	false positives (fp)	true negatives (tn)

- The accuracy is thus:

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

- Information retrieval (IR) also uses recall (R), precision (P), and their combination, F1 measure (F1) as performance measures.

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

$$\text{F1} = \frac{2 \times P \times R}{P + R}$$

- Typically we can maximise one of recall and precision but not both.

Extreme example 1: all e-mails are classified as spam and blocked

email	# classified as spam	# classified as not spam
# spam	25 (tp)	0 (fn)
# not spam	75 (fp)	0 (tn)

Spam precision = 25%, Spam recall =100%, Accuracy = 25%

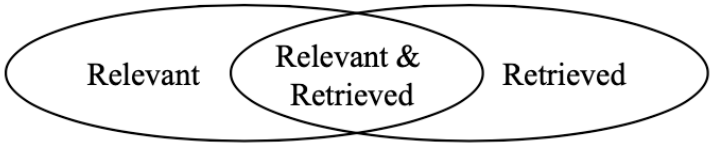
Extreme example 2: all but one e-mail are classified as not-spam

email	# classified as spam	# classified as not spam
# spam	1 (tp)	79 (fn)
# not spam	0 (fp)	20 (tn)

Spam precision = 100%, Spam recall =12.5%, Accuracy = 21%

IR Example 2: Text Retrieval

- Given a query, a text retrieval system retrieves a number of documents.
  - Retrieved - the number of all retrieved documents.
  - Relevant - the number of all documents that are relevant.



- Recall, Precision and F1 are used to address the accuracy of the retrieval.

$$\text{Precision} = \frac{\text{Relevant and Retrieved}}{\text{Retrieved}}$$

$$\text{Recall} = \frac{\text{Relevant and Retrieved}}{\text{Relevant}}$$

Numerical Logistic Regression

- Spam Classification

	free	bank	meet	time	y
Email 1	5	3	1	1	Spam
Email 2	4	2	1	1	Spam
Email 3	2	1	2	3	Ham
Email 4	1	2	3	2	Ham

Perform gradient descent to learn weights $\mathbf{w}$

- Feature vector for email 1: $\mathbf{x}_1 = [1, 5, 3, 1, 1]^\top$
- Let $\boldsymbol{X} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4]^\top$ the matrix of all feature vectors.
- Initial weights $\mathbf{w} = [0.5, 0.5, 0.5, 0.5, 0.5]^\top$
- Prediction

  $[\sigma(\mathbf{w}^\top \mathbf{x}_1), \sigma(\mathbf{w}^\top \mathbf{x}_2), \sigma(\mathbf{w}^\top \mathbf{x}_3), \sigma(\mathbf{w}^\top \mathbf{x}_4)]^\top = [0.996, 0.989, 0.989, 0.989]^\top$

  which can be obtained by computing $\boldsymbol{X}\mathbf{w}$ and then apply $\sigma(\cdot)$ entrywise,

  which we abuse the notation and write $\sigma(\boldsymbol{X}\mathbf{w})$.

- Batch Gradient Descent

Perform gradient descent to learn weights $\mathbf{w}$

- Prediction $\sigma(\boldsymbol{X}\mathbf{w}) = [0.996, 0.989, 0.989, 0.989]^\top$
- Difference from labels $\boldsymbol{y} = [1, 1, 0, 0]^\top$ is

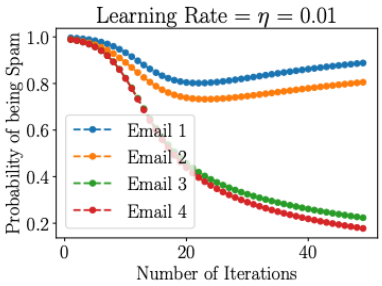
$$\sigma(\boldsymbol{X}\mathbf{w}) - \boldsymbol{y} = [-0.004, -0.011, 0.989, 0.989]^\top$$

- Gradient of the first email,

$$\boldsymbol{g}_1 = (\sigma(\mathbf{w}^\top \mathbf{x}_1) - y_1)\mathbf{x}_1 = -0.004[1, 5, 3, 1, 1]^\top$$

- $\mathbf{w} \leftarrow \mathbf{w} - \underbrace{0.01}_{\text{learning rate}} \sum_n \boldsymbol{g}_n = \mathbf{w} - \eta \boldsymbol{X}^\top(\sigma(\boldsymbol{X}\mathbf{w}) - \boldsymbol{y})$

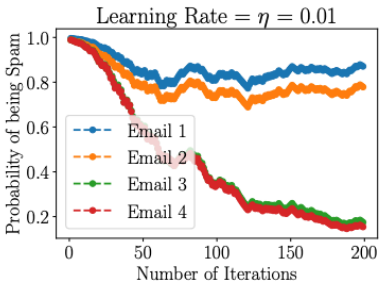
notice the similarity with linear regression



Predictions for Emails 3 and 4 are initially close to 1 (spam), but they converge towards the correct value 0 (ham)

- Stochastic Gradient Descent

  - Prediction $\sigma(\mathbf{w}^\top \mathbf{x}_r) = 0.996$ for a randomly chosen email r
  - Difference from label $y = 1$ is -0.004
  - Gradient is $\boldsymbol{g}_r = (\sigma(\mathbf{w}^\top \mathbf{x}_n) - y)\mathbf{x}_r = -0.004\mathbf{x}_r$
  - $\mathbf{w} \leftarrow \mathbf{w} - 0.01\boldsymbol{g}_r$



Predictions for Emails 3 and 4 are initially close to 1 (spam), but they converge towards the correct value 0 (ham)

  - Final $\mathbf{w} = [0.187, 0.482, 0.179, -0.512, -0.524]^\top$ after 50 batch gradient descent iterations.
  - Given a new email with feature vector $\mathbf{x} = [1, 1, 3, 4, 2]$, the probability of the email being spam is estimated as $\sigma(\mathbf{w}^\top \mathbf{x}) = \sigma(-1.889) = 0.13$.
  - Since this is less than 0.5 we predict ham.

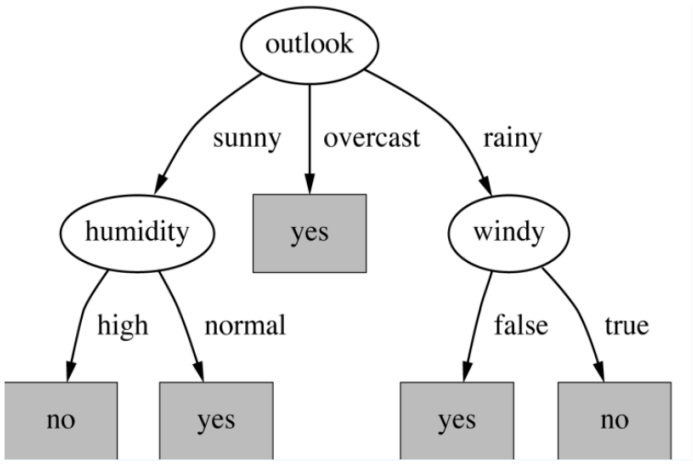
Lecture 5 - Decision Trees

- DT's are supervised classifiers. They were developed in parallel in ML by Ross Quinlan (USYD) and in statistics by Breiman, Friendman, Olshen and Stone.
    - Quinlan has refined the DT algorithm over the years - ID3 in 1986, C4.5 in 1993 (commercial version is used in many DM packages).

Properties of DTs

- Easy to implement. Efficient:
    - Cost of building the tree $O(mn \log n)$, n instances and m attributes.
    - Cost of pruning the the tree with sub-tree replacement: $O(n)$.
    - Cost of pruning by sub-tree lifting: $O(n\ (\log n)^2)$.
    - Total cost: $O(mn \log n) + O(n\ (\log n)^2)$.
- Interpretable:
    - The output of the DT (tree = a set of rules) is considered easier to understand by humans and use for decision making than the output of other algorithms e.g. neural network, support vector machines.

Decision Tree Specifics



- **DT representation**:
    - each non-leaf node represents an attribute.
    - each branch corresponds to an attribute value.
    - each leaf node assigns a class.
- To predict the class for a new example: start from the root and test the values of the attributes until you reach a leaf node, return the class of the leaf node.
- DTs thus can express any function of the input attributes - there is a consistent DT for any training set with 1 path to a leaf i.e. lookup table, explicitly encoding the training data.
    - This DT will no generalise well to new examples. We prefer to find more compact decision trees.
- DTs can be expressed as a set of mutually exclusive rules (each rule is a conjunction of tests with 1 rule = 1 path in the tree). DTs are thus a disjunction of conjunctions.

Constructing Decision Trees

- Top down in a recursive divide and conquer fashion:
  - The best attribute is selected for a root node and a branch is created for each possible attribute value.
  - The examples are split into subsets, one for each branch extending from the node.
  - The procedure is repeated recursively for each branch, using only the examples that reach the branch.
  - Stop if all examples have the same class; make a leaf node corresponding to this class.

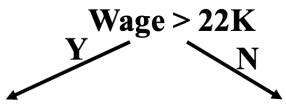
Stopping Criteria

- Stop if:
  - The typical case: **all examples in the subset following the branch have the same class** $\rightarrow$ make a leaf node corresponding to this class.
  - Additional condition: **cannot split any further** - all examples in the subset have the same attribute values but different classes (there's noise in the data) $\rightarrow$ make a leaf node and label it with the majority class of the subset.
  - Additional condition: **subset is empty** (e.g. there are no examples with the attribute value for the branch) $\rightarrow$ create a leaf node and label it with the majority class of the subset of the parent.

Example

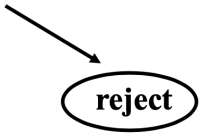
- Assume that we know how to select the best attribute at each step.

name	wage	dependents	sex	loan
Jim	40K	7	M	reject
Jill	20K	1	F	reject
Jack	58K	5	M	accept
Jane	23K	3	F	accept



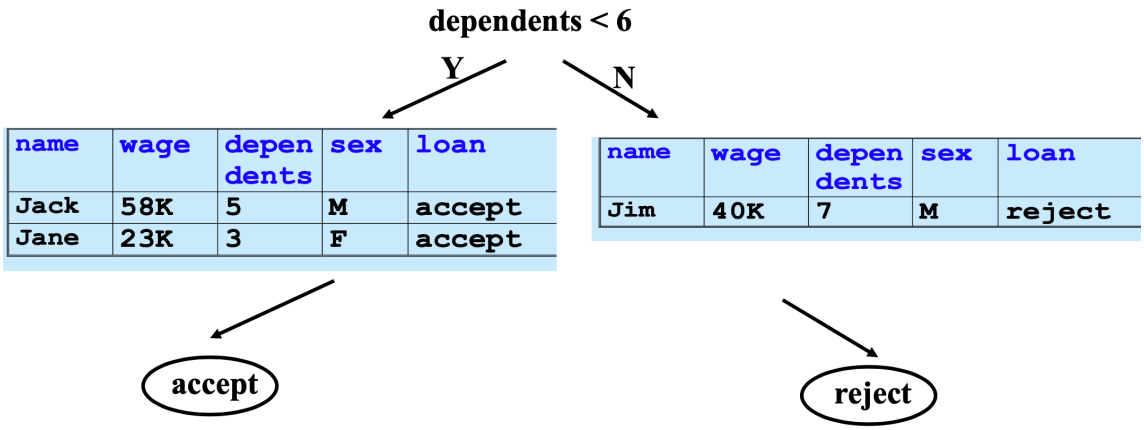
name	wage	dependents	sex	loan
Jim	40K	7	M	reject
Jack	58K	5	M	accept
Jane	23K	3	F	accept

name	wage	dependents	sex	loan
Jill	20K	1	F	reject



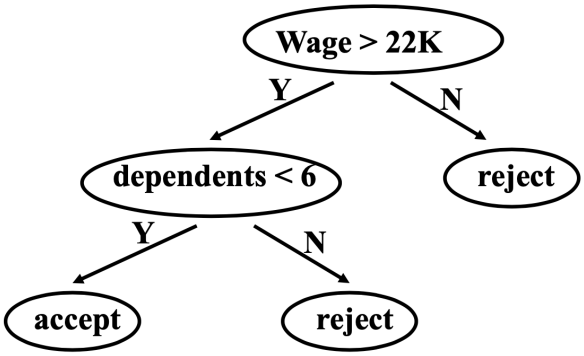
- From here, we look at the leftward node.

name	wage	dependents	sex	loan
Jim	40K	7	M	reject
Jack	58K	5	M	accept
Jane	23K	3	F	accept

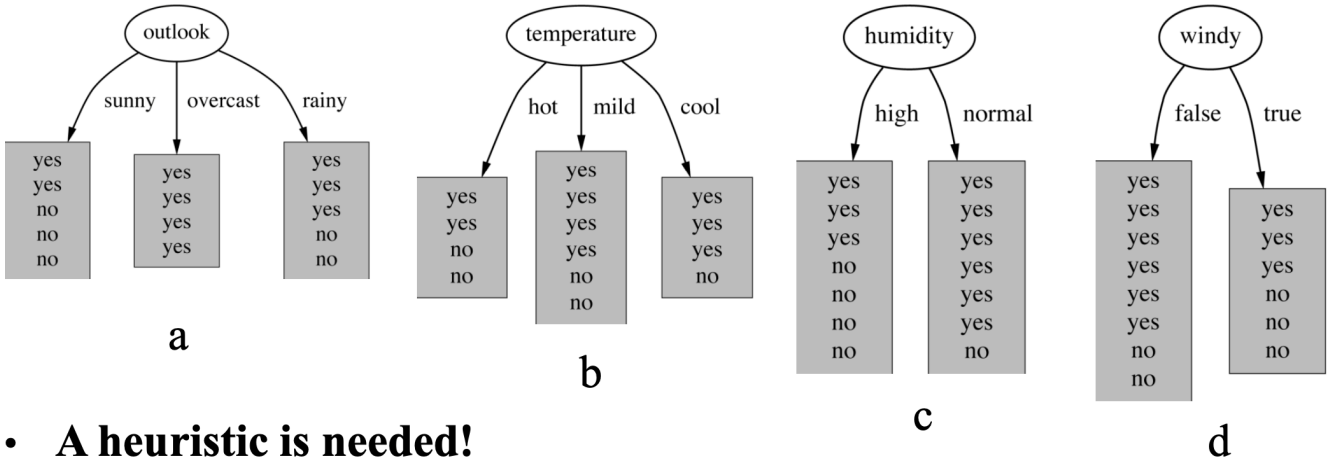


- Finally:

name	wage	dependents	sex	loan
Jim	40K	7	M	reject
Jill	20K	1	F	reject
Jack	58K	5	M	accept
Jane	23K	3	F	accept



How to Find the Best Attribute?



- **A heuristic is needed!**

- A measure of 'purity' of each node would be a good choice, as the 'pure' subsets (containing only yes or no) will not have to be split further and the recursive process will terminate.
    - At each step, we can choose the attribute which produces the purest children nodes.

Entropy

- Given a set of examples with their class e.g. the weather data with 9 examples from class yes and 5 examples from class no.

Information Content

- **Entropy** measures the homogeneity (purity) of a set of examples with respect to their class.
- The smaller the entropy, the greater the purity of the set. It is the standard measure in signal compression, information theory, physics.

Formal Definition

- Entropy $H(S)$ of data set S, where P_i is the proportion of examples that belong to class i.

$$H(S) = I(S) = -\sum_i P_i \times log_2 P_i$$

- For our example:

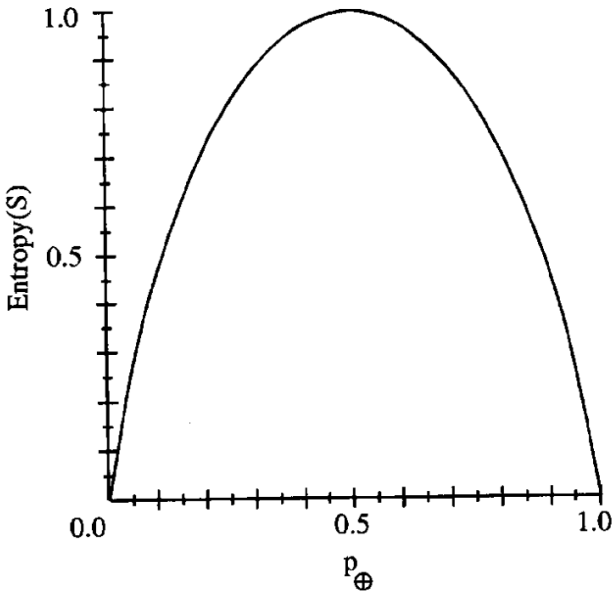
$$H(S) = -P_{yes}\log_2 P_{yes} - P_{no}\log_2 P_{no} = I(P_{yes}, P_{no}) = I(\frac{9}{14}, \frac{5}{14}) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940\ bits$$

- Entropy is measured in bits so we used log to the base 2. Note: $log(0)$ is undefined, so we just assume that it is 0 when we calculate entropy.

Range of Entropy for Binary Classification

- 2 classes: yes and no. On the x-axis is the proportion of positive examples (with the proportion of negative examples being 1-p). On the y-axis is the entropy $H(S)$.

$$H(S) = I(p, 1-p) = -p\log_2 p - (1-p)\log_2(1-p)$$



- $H(S) \in [0, 1]$.
- $H(S) = 0$ means that all examples in S belong to the same class (no disorder, min entropy).
- $H(S) = 1$ means that there are an equal number of yes and no (S is as disordered as it can be, max entropy).

Information Gain

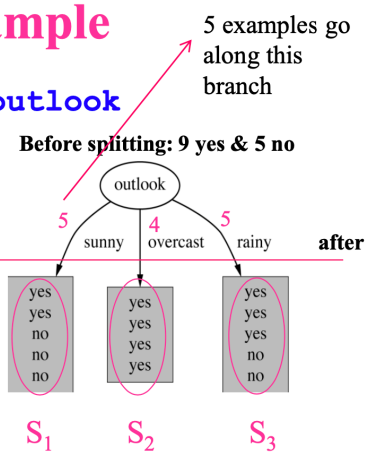
- Information gain measures of the effectiveness of an attribute to classify the training data.
  - It measures the reduction in entropy caused by using this attribute to partition the set of training examples.
  - The best attribute is the one with the highest information gain i.e. largest reduction in the entropy of the parent node, and the one that is expected to lead to pure partitions fastest.

Example

Information Gain - Example

- **Let's calculate the information gain of the attribute `outlook`**
- **Information gain measures reduction in entropy**
- **It is a difference of 2 terms: *T1 –T2***
- ***T1* is the entropy of the set of examples S associated with the parent node before the split**

$$T1 = H(S) = I(\frac{9}{14}, \frac{5}{14}) = 0.940 \; bits$$



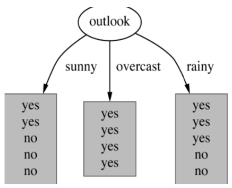
- ***T2* is the remaining entropy in S, after S is split by the attribute (e.g. `outlook`)**
  - **It takes into consideration the entropies of the child nodes and the distribution of the examples along each child node**
  - **E.g. for a split on `outlook`, it will consider the entropies of S1, S2 and S3 and the proportion of examples following each branch (5/14, 4/14, 5/15):**

$$T2 = H(S \mid outlook) = \frac{5}{14} \cdot H(S_1) + \frac{4}{14} \cdot H(S_2) + \frac{5}{14} \cdot H(S_3)$$

- Therefore:

$$Gain(S \mid A) = H(S) - \sum_{j \in values(A)} P(A = v_j) H(S \mid A = v_j) =$$

$$= H(S) - \sum_{j \in values(A)} \frac{|S_v|}{|S|} H(S \mid A = v_j)$$



Gain(S|A) is the information gain of an attribute A relative to S

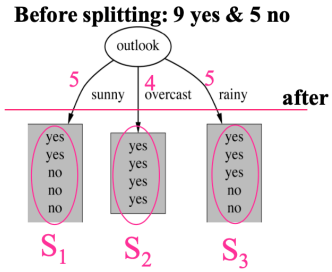
Values(A) is the set of all possible values for A

S_v is the subset of S for which A has value v

=entropy of the original data set S

=conditional entropy H(Y|A);
=expected value of the entropy after S is partitioned by A
=called *Reminder* in Russell and Norvig

- Calculating the entropy of the child:



$$H(S \mid outlook = sunny) = I(\frac{2}{5}, \frac{3}{5}) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971 bits$$

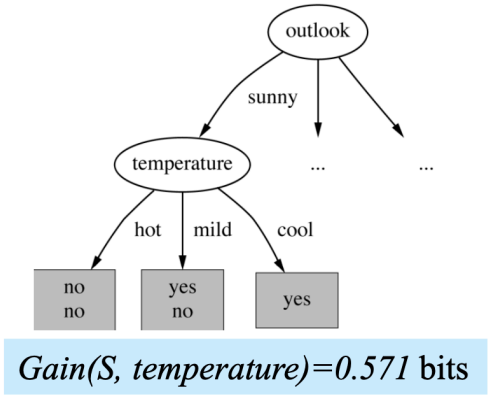
$$H(S \mid outlook = overcast) = I(\frac{4}{4}, \frac{0}{4}) = -\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4} = 0 bits$$

$$H(S \mid outlook = rainy) = I(\frac{3}{5}, \frac{2}{5}) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.971 bits$$

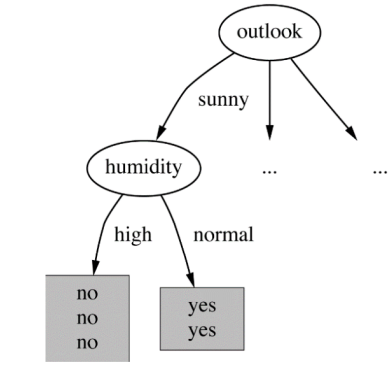
$$H(S \mid outlook) = \frac{5}{14} \cdot 0.971 + \frac{4}{14} \cdot 0 + \frac{5}{14} \cdot 0.971 = 0.693 bits$$

- The $Gain(S|outlook) = 0.247$ bits. The other three attributes are temperature $= 0.029$, humidity $= 0.152$, windy $= 0.048$.

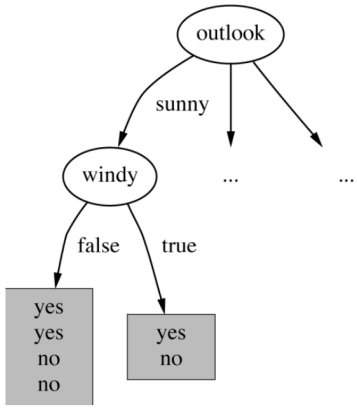
Continuing the Split From Here On



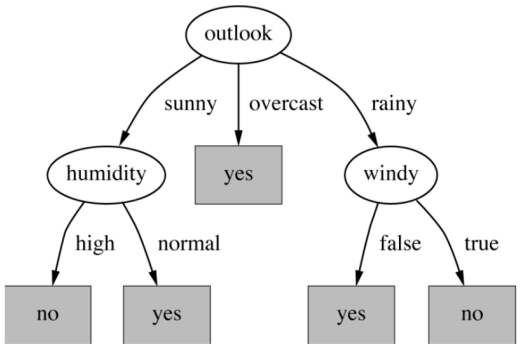
Gain(S, temperature)=0.571 bits



Gain(S, humidity)=0.971 bits



Gain(S, windy)=0.020 bits

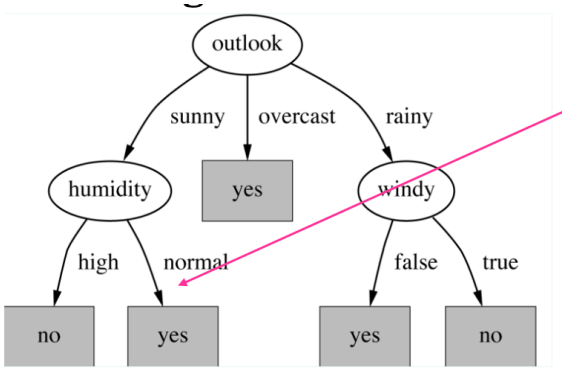


Final DT

Overfitting in DTs - Reasons

- A DT grows each branch of the tree deeply enough to perfectly classify the training examples.
- In doing this (and depending on the data), the tree may become very specific (like a look-up table) and not represent patterns, but only memorize the data.
- Problems with the training data also leads to problems with the induced DT
  - Training data is too small $\rightarrow$ not enough representative examples to build a model that can generalize well on new data
  - Noise in the training data, e.g. incorrectly classified examples $\rightarrow$ DT learns them but they are incorrect and will incorrectly classify new examples.

Overfitting Due to Noise



- Let's say that the following positive example is incorrectly labelled as negative: `outlook=sunny`, `temperature=hot`, `humidity=normal`, `wind=yes`, `play=no`.
- The new tree will test below the highlighted node, becoming more complex. We still expect the original tree to outperform the new tree on the subsequent data.

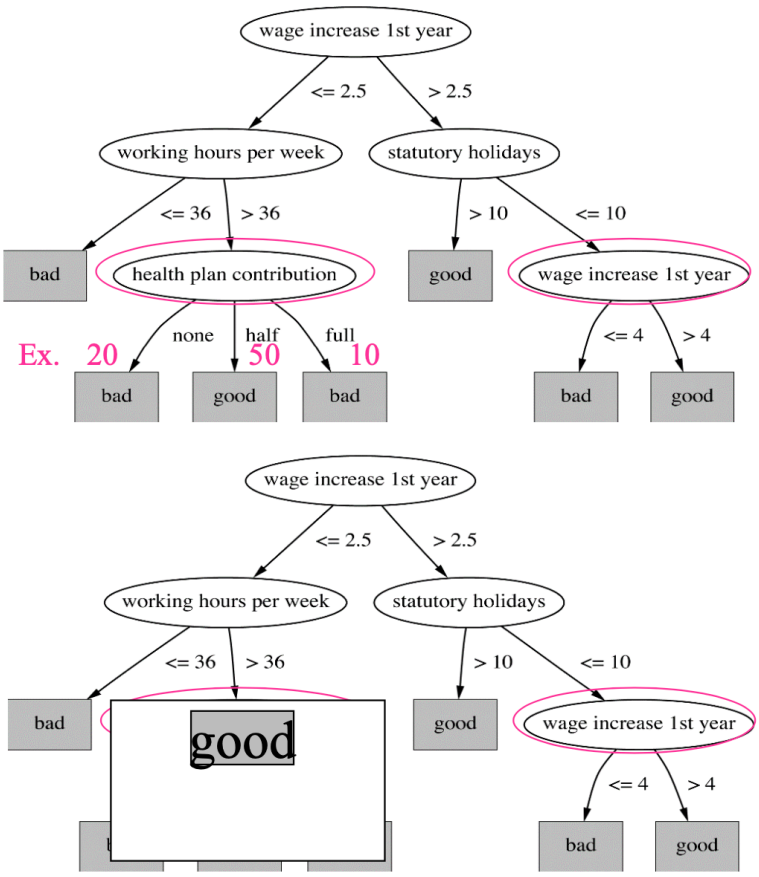
Tree Pruning

- Used to avoid overfitting in DTs.
  - Pre-pruning: stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data.
  - Post-pruning: fully grow the tree (allowing it to perfectly over the training data) and then prune it. Since this is the **more successful and widely used method**, there are two main approaches.
    - Tree-pruning - directly prune the tree.
      - Sub-tree replacement
      - Sub-tree raising
    - Rule pruning - convert the three into a set of rules and then prune them.

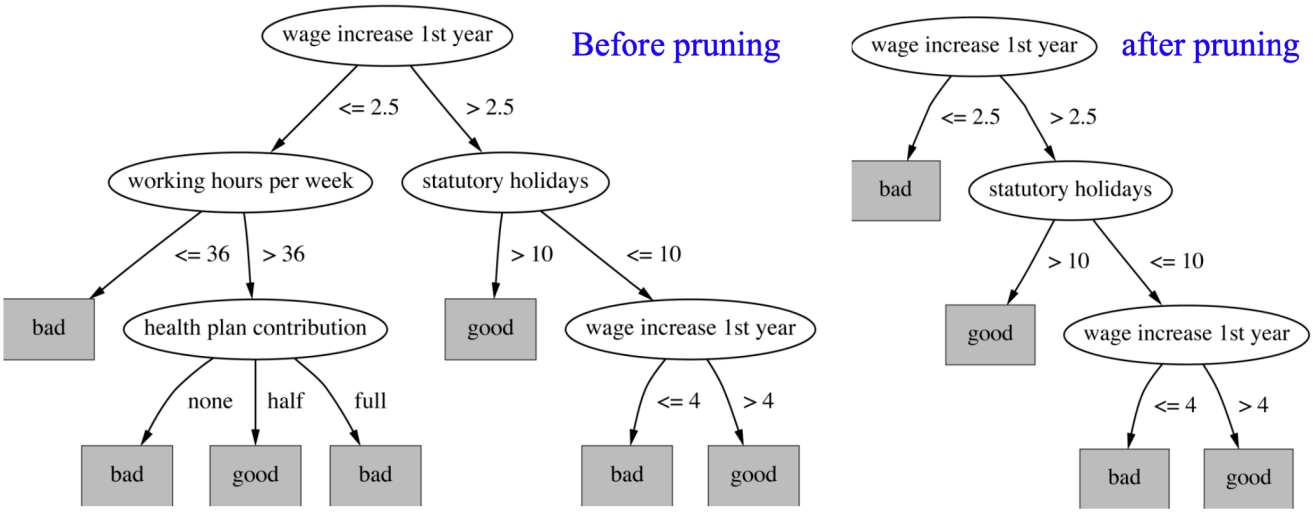
Using Validation Sets

- We determine when to stop pruning by estimating the accuracy using validation sets. We could use the training data, but this is too optimistic (statistical underpinning of heuristic is rather weak).
- Available data is separated into three sets:
  - Training set - used to build the DT.
  - Validation set - to evaluate the impact of pruning and decide when to stop.
  - Test data - to evaluate how good the final DT is.
- **Motivation**:
  - Even though the DT may be misled by random errors and coincidental regularities within the training set, the validation set is unlikely to exhibit the same random fluctuations $\rightarrow$ the validation set can provide a safety check against overfitting of the training set.
  - The validation set should be large enough; typically 1/2 of the available examples are used as training set, 1/4 as validation set, and 1/4 as test set.
- **Disadvantage:** The tree is built on less data.
  - When the data is limited, withholding part of it for validation reduces even further the examples available for training.

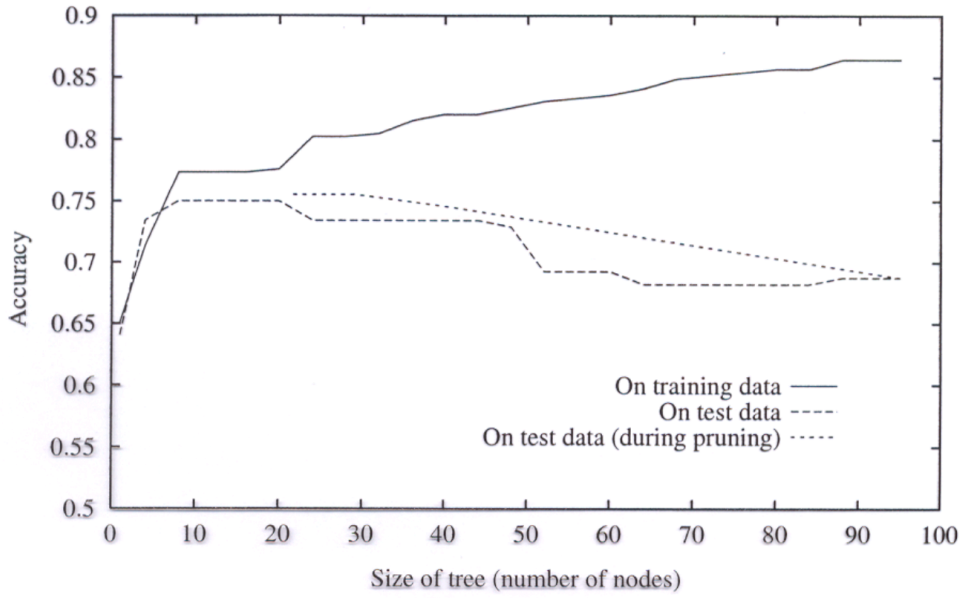
Tree Post-Pruning by Sub-Tree Replacement



- Each non-leaf node (i.e. each test node) is a candidate for pruning.
- Start from the leaves and work towards the root.
- For each candidate node:
  - Remove the sub-tree rooted at it.
  - Replace it with a leaf with the class being the majority class of examples that go along the candidate node.
  - Compare the new tree with the old tree by calculating the accuracy on the validation set for both.
  - If the accuracy of the new tree is better or the same as the accuracy of the old tree, keep the new tree (we say that the candidate node is pruned).
- At each step, we will evaluate all candidate nodes and accept the best pruning – i.e. the one that will improve the accuracy most. No pruning if the new tree is worse than the old tree.



- The accuracy of test data increases as nodes are pruned.



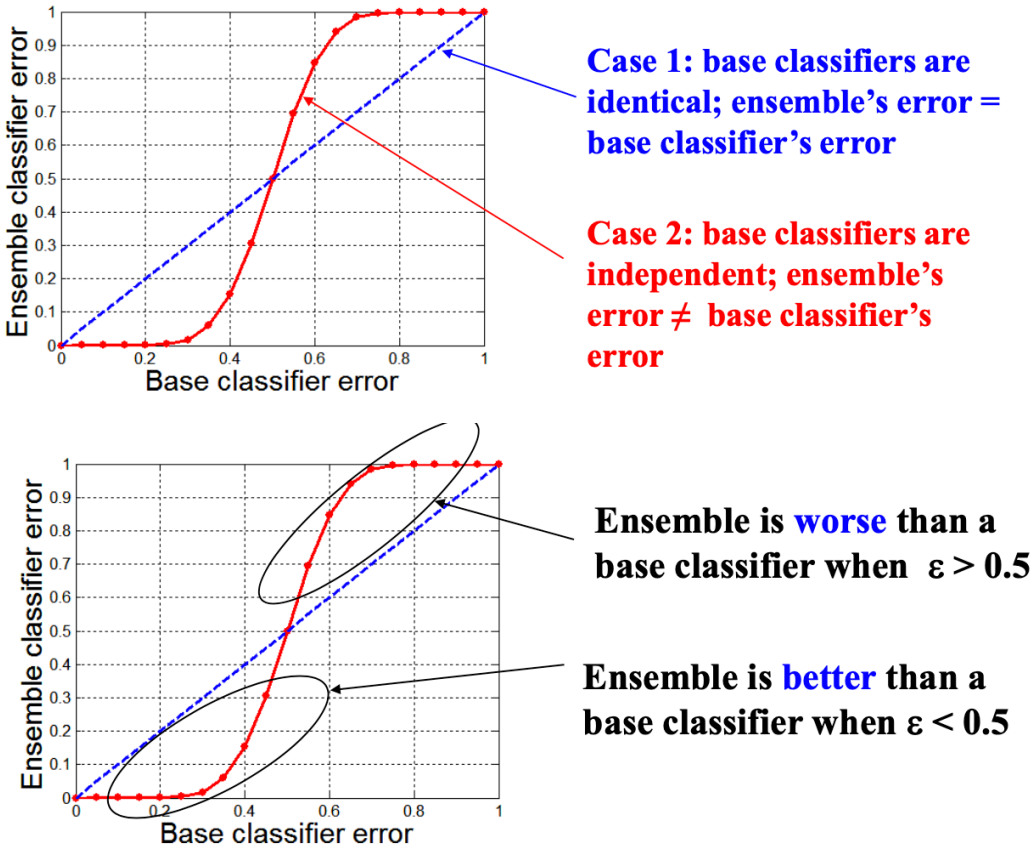
Ensemble Methods

- An ensemble of classifiers combines the predictions of multiple classifiers (called base classifiers) in some way to classify new instances i.e. a **committee** of classifiers.
  - An ensemble of 50 backpropagation NNs, all trained on the same dataset but with different parameters, architecture, initialisation, learning rate etc.
  - To classify a new example, the individual predictions are combined by taking the majority vote.
- In real life:
  - If a certain medical diagnosis occurs more than the others, he chooses it as the final diagnosis i.e. the majority vote (intuition behind bagging).
  - Instead of trusting equally, weights are assigned to the value of each diagnosis, based on the accuracy of past diagnoses. The final one is a weight combination (intuition behind boosting).

Motivation Behind Ensembles

- Combining multiple classifiers helps when:
  - The base classifiers do not make the same mistakes i.e. they complement each other, each is an expert in a part of the domain where the others don't perform well.
  - Each base classifier is reasonably accurate.
- Given 25 base classifiers, binary classification task, an error rate of $\epsilon = 0.35$ on the test set for each base classifier, and majority voting:
  - If base classifiers are identical and make the same mistakes, the error rate for the ensemble is $\epsilon = 0.35$.
  - If the base classifiers are independent (errors not correlated), a new example will be misclassified if more than half of the base classifiers predict incorrectly. Mathematically, however, the error is much less than the individual error rate:

$$e_{\text{ensemble}} = \sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} = 0.06$$



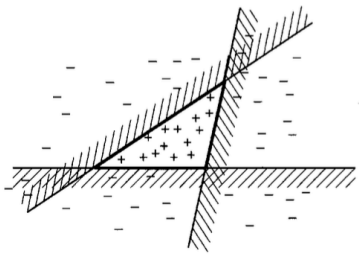
- In essence, if the base classifier is better than a random guess, an ensemble will perform better.

Summary of Conditions

- Ensembles perform better than a single classifier when:
  - The base classifiers are **good enough/highly correct** i.e. better than random guessing.
  - The base classifiers are **independent** of each other. Although this is difficult in practice, good results have been achieved with slightly correlated classifiers.

Ensemble Enlarges Hypothesis Space

- Ensembles do well because they are able to learn much more expressive hypotheses without much additional computational expense compared to an individual classifier.



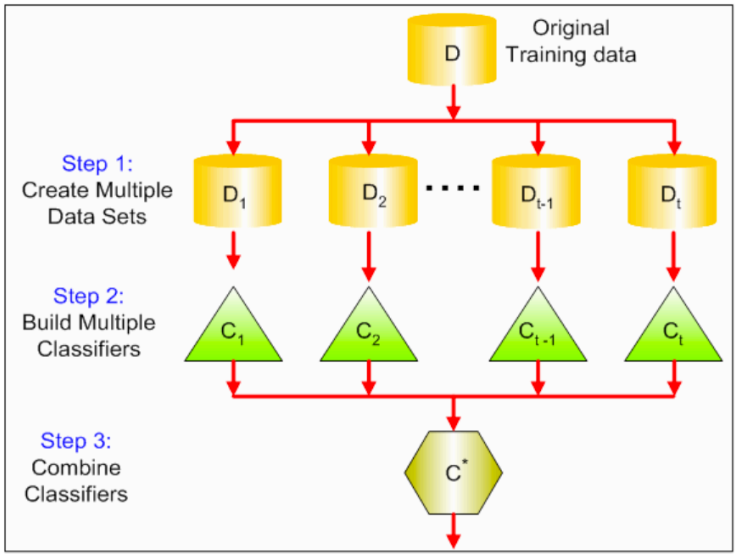
Example:

• 3 linear binary hypotheses, e.g. generated by 3 perceptrons

• An ensemble of 3 perceptrons can learn the resulting triangular region - a hypothesis not expressible in the original hypothesis space (of the perceptron)

Methods for Creating Ensembles

- Focus on generating disagreement among base classifiers by:
  - Manipulating the **training data** - creating multiple training sets by resampling the original data and creating a classifier for each training set i.e. Bagging and Boosting



  - Manipulating the **attributes** - using a subset of input features i.e. Random Forest and Random Subspace.
  - Manipulating the **class labels** - e.g. output coding.
  - Manipulating the learning algorithm - e.g. building a set of classifiers with different parameters.

Bagging - Bootstrap Aggregation

- Create M bootstrap samples.
  - Given a dataset D with n examples, a bootstrap sample D' contains n examples, randomly chosen from D with replacement.
  - On average, 63% of examples in D will also appear in D'.

	Dataset with 10 examples:									
Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Use each sample to build a classifier.
- To classify a new example: get the predictions of each classifier and combine them with a majority vote.

```
model generation
Let n be the number of instances in the training data.
For each of M iterations:
  Sample n instances with replacement from training data.
  Apply the learning algorithm to the sample.
  Store the resulting model (classifier).

classification
For each of the M models:
  Predict class of testing instance using model.
Return class that has been predicted most often.
```

When is Bagging Useful?

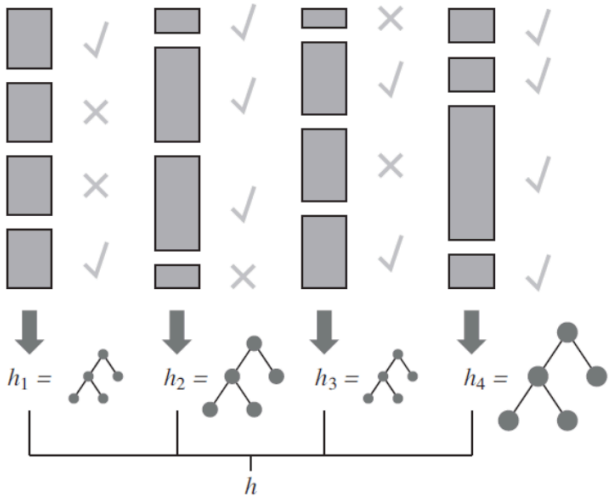
- Typically performs significantly better than the single classifier and is never substantially worse.
- Effective for unstable classifiers, in which small changes in the training set result in large changes in predictions e.g. DTs, neural networks.
- Bagging to numerical prediction/regression:
  - Individual predictions are averaged.
  - Shown theoretically that averaging over multiple models always reduces the expected value of the mean-squared error (not proved for classification).

Boosting

- The most widely used ensemble method, the idea is to make classifiers complement each other.
  - The next classifier should be created using examples that were difficult for the previous classifiers.
- This is achieved by using a weighted training set, where higher weights are examples that have proven difficult to classify previously, and thus should be selected with higher probability.

Algorithm

- Set equal weights for all example e.g. $1/m$ where m is the number of training examples.
- Generate first classifier $h_1 \rightarrow$ we want the next classifier to develop complementary expertise by classifying the misclassified examples.
- Increase the weights of misclassified and decrease the weights of correctly classified examples.
- There is a mechanism for selecting examples for the training set of the next classifier such that higher weights are more likely to be selected $\rightarrow$ generate h_2.
- Continue until K classifiers are generated, and combine using a weighted vote based on how well each classifier performed on the training set e.g. h_4 would have a higher weight vote than h_1.



1 rectangle corresponds to 1 example
The height of the rectangle corresponds to the weight of the example
√ and X show how the example was classified by the current hypothesis (classifier)
The size of the DT corresponds to the weight of that hypothesis in the final ensemble

AdaBoost Algorithm

- Uses a weighted training set. Each training example has an associated weight ≥ 0.
  - The higher the weight, the more difficult the example was to classify by previous classifiers.
  - Examples with higher weight wil have a higher chance to be selected in the training set for the next classifier.
- Note that AdaBoost is a meta-algorithm and can be used with any type of classifier, since it never specifies how we get $h_t^*(x)$.

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing

$$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

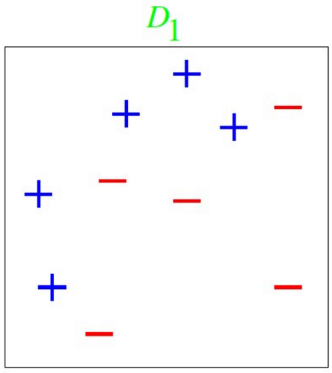
  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights on training points

$$w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

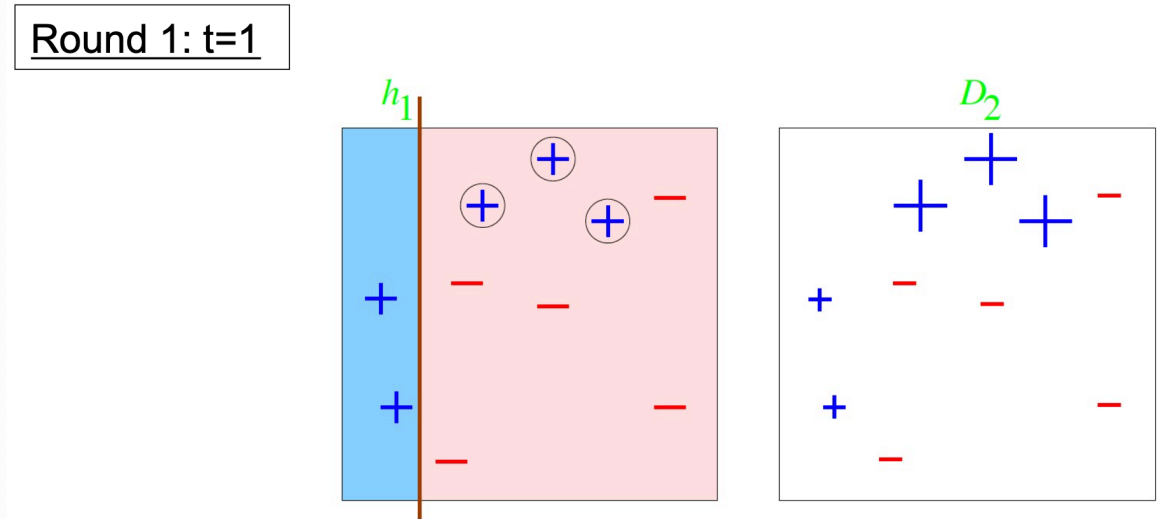
  and normalize them such that $\sum_n w_{t+1}(n) = 1$
- Output the final classifier

$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x})\right]$$

Example

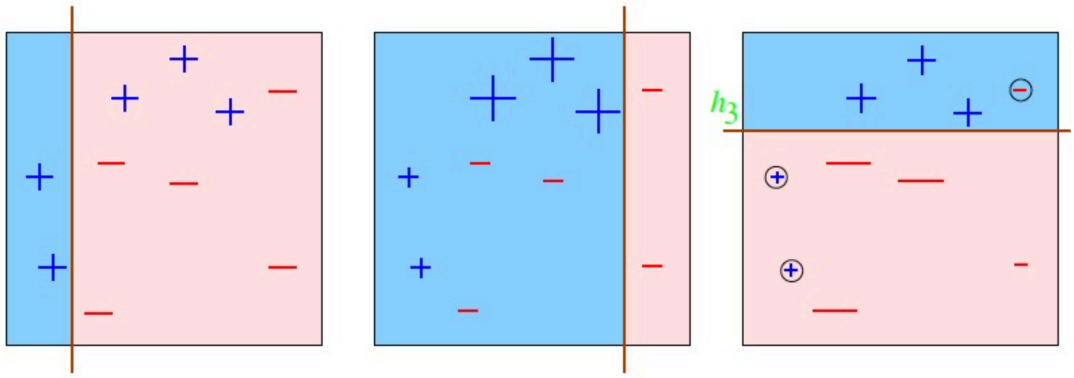


- The data pouints are clearly not linearly separable.
  - In the beginning, all data points have equal weights (size of data markers).
  - The base classifier $H(\cdot)$: horizontal or vertical lines i.e. decision stumps.

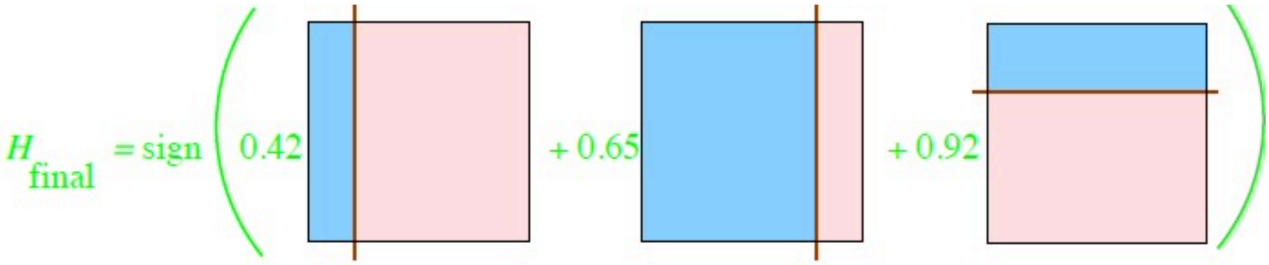


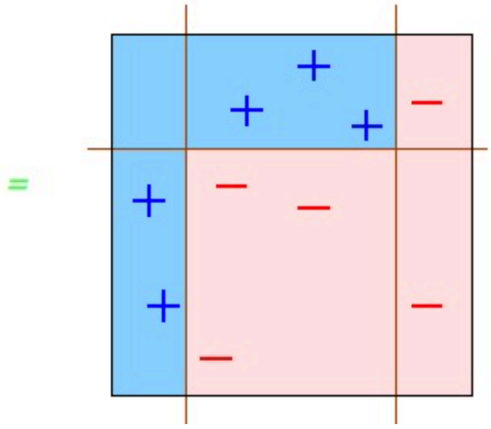
- The 3 misclassified leads to an $\epsilon_1 = 0.3 \rightarrow \beta_1 = 0.42$, where β_1 is the contribution of this classifier.

t=3



- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, hence our error is low. Why?
  - Since they have been consistently classified correctly, this round's mistake will hopefully not have a huge impact on the overall prediction





Why Does Adaboost Work?

- It minimises a loss function related to classification error.
- Classification Loss
  - Suppose we want to have a classifier,

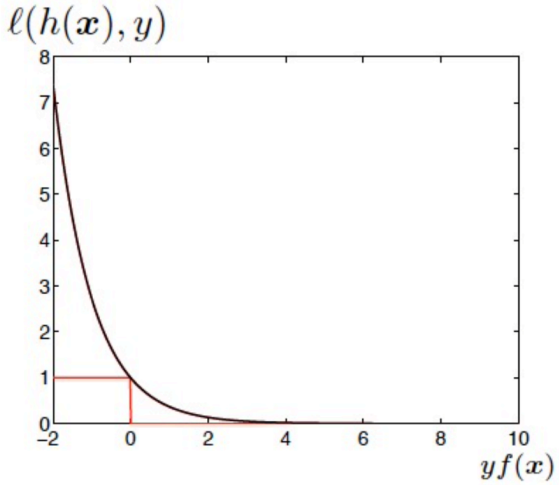
$$h(\boldsymbol{x}) = \text{sign}[f(\boldsymbol{x})] = \begin{cases} 1 & \text{if } f(\boldsymbol{x}) > 0 \\ -1 & \text{if } f(\boldsymbol{x}) < 0 \end{cases}$$

- One seemingly natural loss function is 0-1 loss:

$$\ell(h(\boldsymbol{x}), y) = \begin{cases} 0 & \text{if } yf(\boldsymbol{x}) > 0 \\ 1 & \text{if } yf(\boldsymbol{x}) < 0 \end{cases}$$

- However, 0-1 loss function $l(h(x), y)$ is non-convex and difficult to optimise.
- We can use a surrogate loss such as exponential loss.

$$\ell^{\text{EXP}}(h(\boldsymbol{x}), y) = e^{-yf(\boldsymbol{x})}$$



How to Choose t-classifier?

Suppose a classifier $f_{t-1}(\boldsymbol{x})$, and want to add a weak learner $h_t(\boldsymbol{x})$

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

Note: $h_t(\cdot)$ outputs -1 or 1, as does sign $[f_{t-1}(\cdot)]$

How can we 'optimally' choose $h_t(\boldsymbol{x})$ and combination coefficient β_t?

Adaboost greedily *minimizes the exponential loss function*!

$$\begin{aligned}(h_t^*(\boldsymbol{x}), \beta_t^*) &= \text{argmin}_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n e^{-y_n f(\boldsymbol{x}_n)} \\ &= \text{argmin}_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n e^{-y_n [f_{t-1}(\boldsymbol{x}_n) + \beta_t h_t(\boldsymbol{x}_n)]} \\ &= \text{argmin}_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)}\end{aligned}$$

where we have used $w_t(n)$ as a shorthand for $e^{-y_n f_{t-1}(\boldsymbol{x}_n)}$

The New Classifier

We can decompose the *weighted* loss function into two parts

$$\begin{aligned}
&\sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(\boldsymbol{x}_n)] \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]) \\
&= (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}$$

We have used the following properties to derive the above

- $y_n h_t(\boldsymbol{x}_n)$ is either 1 or -1 as $h_t(\boldsymbol{x}_n)$ is the output of a binary classifier
- The indicator function $\mathbb{I}[y_n = h_t(\boldsymbol{x}_n)]$ is either 0 or 1, so it equals $1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$

Finding the Optimal Weak Learner

$$\begin{aligned}
(h_t^*(\boldsymbol{x}), \beta_t^*) &= \text{argmin}_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \text{argmin}_{(h_t(\boldsymbol{x}), \beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \\
&\quad + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}$$

What term(s) must we optimize?

We need to minimize the entire objective function with respect to β_t!

We can take the derivative with respect to β_t, set it to zero, and solve for β_t. After some calculation and using $\sum_n w_t(n) = 1$...

$$\beta_t^* = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

Updating the Weights

Once we find the optimal weak learner we can update our classifier:

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x})$$

We then need to compute the weights for the above classifier as:

$$w_{t+1}(n) \propto e^{-y_n f(\boldsymbol{x}_n)} = e^{-y_n[f_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x}_n)]}$$

$$= w_t(n) e^{-y_n \beta_t^* h_t^*(\boldsymbol{x}_n)} = \begin{cases} w_t(n) e^{\beta_t^*} & \text{if } y_n \neq h_t^*(\boldsymbol{x}_n) \\ w_t(n) e^{-\beta_t^*} & \text{if } y_n = h_t^*(\boldsymbol{x}_n) \end{cases}$$

Intuition Misclassified data points will get their weights increased, while correctly classified data points will get their weight decreased

Other Boosting Algorithms

AdaBoost is by far the most popular boosting algorithm.

Gradient boosting generalizes AdaBoost by substituting another (smooth) loss function for exponential loss.

- Squared loss, logistic loss, ...
- Choose the candidate learner that greedily minimizes the error (mathematically, it should maximize the gradient of the residuals calculated with this loss function).
- Reduce overfitting by constraining the candidate learner (e.g., computing the residuals only over a sample of points).

LogitBoost minimizes the logistic loss instead of exponential loss.

Gentle AdaBoost bounds the step size (β) of each learner update.

Bagging and Boosting Comparison

Similarities

- Uses voting for classification and averaging for prediction to combine the outputs of individual learners.
- Combine classifiers of the same type e.g. DTs.

Differences

- The ensemble members are built separately in bagging; they are built iteratively in boosting (influenced by performance of previous ensemble members).
    - A new ensemble member is encouraged to become an expert that complement each other i.e. for examples incorrectly classified by previous ensemble members.
- Combing the opinions of individual ensemble members:
    - Bagging - equal weighting (all experts equally influential).

- Boosting - weighed based on performance (i.e. some more influential than others).

Random Forest

- Combines decision trees and uses 1) bagging and 2) subset of features (during decision tree building, when selecting the most important attribute).

n - number of training examples, *m* – number of all features, *k* – number of features to be used by each ensemble member (k<m), *M* – number of ensemble members

Model generation:
For each of *M* iteration
1. Bagging – generate a bootstrap sample
   Sample *n* instances with replacement from training data
2. Random feature selection for selecting the best attribute
   Grow decision tree without pruning. At each step select the best feature to split on by considering only *k* randomly selected features and calculating information gain

Classification:
Apply the new example to each of the *t* decision trees starting from the root. Assign it to the class corresponding to the leaf. Combine the decisions of the individual trees by majority voting.

Discussion

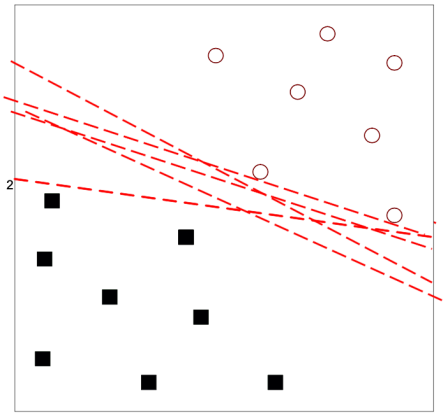
- Performance depends on the accuracy of the individual trees and their correlation with each other.
  - Ideally, we would like highly accurate individual trees with little correlation.
- Bagging and random feature selection used to generate diversity and reduce correlation.
- As the number of features k increases, both the strength and correlation increase.
  - Rule of Thumb: $k = \log_2 m$ where m is number of features.
- Random Forest typically outperforms a single DT.
- Random Forests have been proven to not overfit.
- Random Forests are faster than AdaBoost and give comparably accurate results.

Lecture 6 - SVM

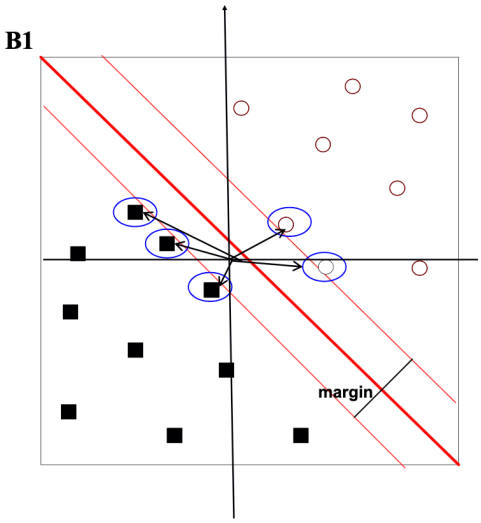
- Very popular, "off-the-shelf" classification method rooted in statistical learning theory, as it only has a few parameters that are easy to tune.
  - Maximise the margin of the decision boundary.
  - Transform data into a higher dimensional space where it is more likely to be linearly separable.
  - Kernel trick allows you to do calculations in the original, not the higher dimensional space.
- **Advantages**:
  - Can form arbitrary decisions both linear and non-linear.
  - The decision boundary is a maximum margin hyperplane i.e. has the highest possible distance to the training examples from the two classes $\rightarrow$ this helps to generalise well on new examples.
  - The decision boundary is defined by a subset of the training vectors called support vectors $\rightarrow$ resistant to overfitting.

Separation by a Hyperplane

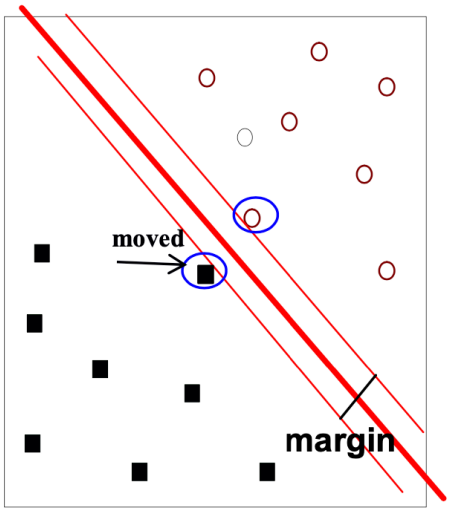
- Given a 2 class problem, squares and circles, find a linear boundary that separates the data.



- **Given a decision boundary B1:**
  - Support Vectors: the data points that lie closest to the decision boundary. It is possible to have more than 1 support vector for each class e.g. 5 total, 3 square, 2 circle $\rightarrow$ remember that training examples are given as input vectors, where each dimension corresponds to 1 feature.
  - Margin: the separation between the boundary and the closest examples (or the width the boundary can be increased before touching an example). The boundary is in the middle of the margin.



- Support vectors define the decision boundary. If we move a support vector, the decision boundary changes. This is not the case when a non support vector is moved.



- The hyperplane with the bigger margin is "better" and should be selected.

Maximum Margin Hyperplane

- The hyperplane/decision boundary (separator) with the biggest margin is called the **maximum margin hyperplane (separator)**.
  - A SVM selects the maximum margin hyperplane, since they are typically more accurate on new examples.
- Intuitively, it feels safer. We don't know where new examples will be but we assume they will be drawn from the same distribution as training examples.
- If we make small errors in the location of the boundary, the chances of causing misclassifications will be smaller if the margin is big $\rightarrow$ big margin = less sensitive to noise and overfitting.

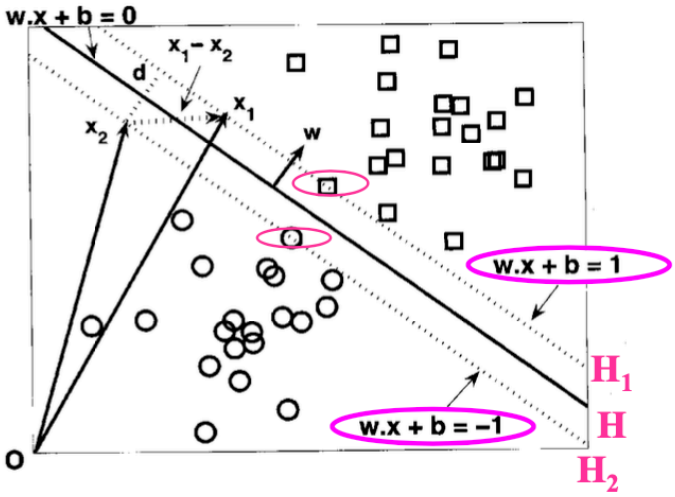
Maximum Margin - Formal Justification

- Structural risk minimisation principle from computational learning theory:
  - Generalisation error (error on new examples) depends on the training error and model complexity (also called model capacity).
  - If the training error is the same, the classifier with the lower complexity will generalise better.
- Small/big margin = high/low complexity = higher/lower generalisation error.
- If the margin is small, then any slight change in the hyperplane or the training examples at the boundary is more likely to affect the classification $\rightarrow$ more susceptible to overfitting.

Quick Revision - Linear Decision Boundary

- A decision boundary of a linear classifier is $w \cdot x + b = 0$ where w and b are parameters.
- Determining the sign will give us the classification of an example x_i.

Problem Statement for SVM



- Our separating hyperplane is H, which is between two others H_1 and H_2 defined as:
  - $H_1 : w \cdot x + b = 1$
  - $H_2 : w \cdot x + b = -1$
- d is the margin of H, with the point lying on the two hyperplanes above being the support vectors.
- It can be shown by calculating the distance between a point from H and the hyperplane H_1 and then multiplying by 2, that d can be expressed as:

$$d = \frac{2}{||w||}$$

- To maximise the margin d, we must minimise $||w||$, i.e. minimise $\frac{1}{2}||w||^2$.
- The optimisation problem can be transformed into its dual optimisation problem:

$$\max \mathbf{w}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

Dot product of pairs of training vectors

Target value (class value) of the training vectors

$$subject\ to\ \lambda_i \geq 0, \sum_{i=1}^{N} \lambda_i y_i = 0$$

- This is a Quadratic Programming (QP) problem and can be solved (i.e. the λs can be estimated) using a standard numerical procedure.
  - Global maximum of the λs (Lagrange multipliers) always exist.
  - w i.e. the optimum decision boundary:

$$w = \sum_{i=1}^{N} \lambda_i y_i x_i$$

All Combined

- Given N labelled training examples

$$(x_i, y_i), i = 1, \ldots, N$$
$$x_i = (x_{i1}, \ldots, x_{im})^T, y_i = \{-1, 1\}$$

- Minimise $\frac{1}{2}||w||^2$ subject to the linear constraint $y_i(w \cdot x_i + b) \geq 1, \forall i$, explained further 10a 18.

    - Learn the maximum margin hyperplane such that all training examples are classified correctly.
- Using QP, where λ are Lagrange multipliers:

$$w = \sum_{i=1}^{N} \lambda_i y_i x_i$$

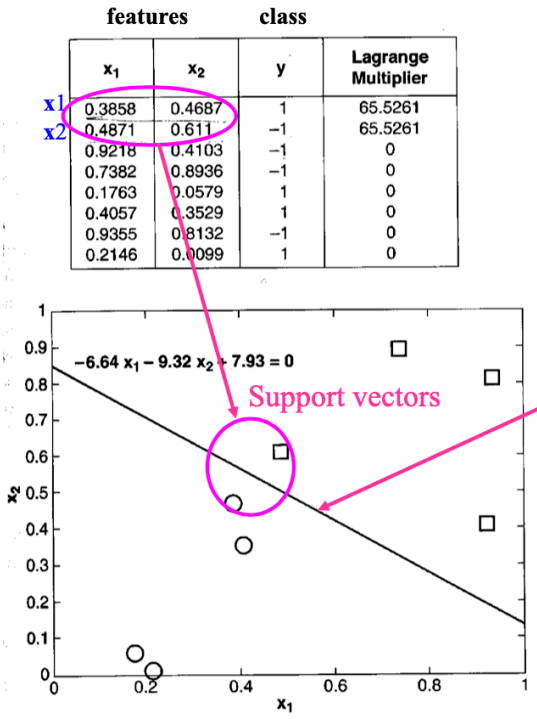
- Since many λs are 0, the optimal decision boundary w is a linear combination ^ of support vectors, which have a non-zero λ_i.
- To classify a new example z:

$$f = \mathbf{w} \cdot \mathbf{z} + b = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \cdot \mathbf{z} + b$$

Dot product of the new vector and the support vectors

$sign(f)$ **i.e. the new example belongs to class 1, if f>0 or class -1 if f<0**

- For example:



x_1	x_2	y	Lagrange Multiplier
0.3858	0.4687	1	65.5261
0.4871	0.611	−1	65.5261
0.9218	0.4103	−1	0
0.7382	0.8936	−1	0
0.1763	0.0579	1	0
0.4057	0.3529	1	0
0.9355	0.8132	−1	0
0.2146	0.0099	1	0

- **8 2-dim. training examples; 2 classes: -1,1**
- **After solving the problem with QP we find the λs and only 2 of them are non-zero and they correspond to the support vectors for the data (x1 & x2)**
- **Using the λs , the weights (defining the decision boundary are):**

$$w_1 = \sum_{i=1}^{2} \lambda_i y_i \mathbf{x}_{i1} = 65.5261(1*0.3858 - 1*0.4871) = -6.64$$

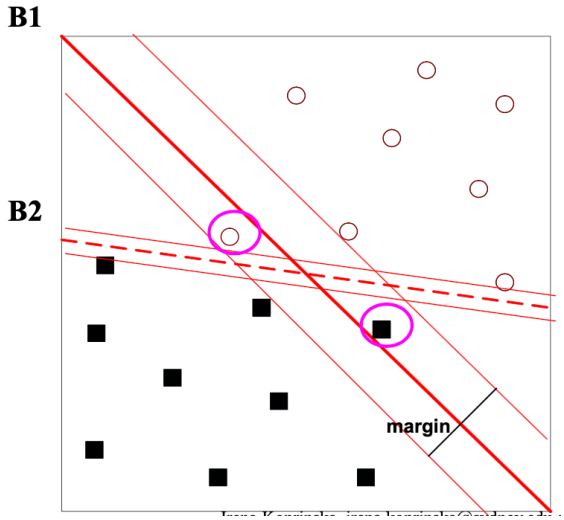
$$w_2 = \sum_{i=1}^{2} \lambda_i y_i \mathbf{x}_{i2} = 65.5261(1*0.4687 - 1*0.611) = -9.32$$

$b = 7.93$ //there is a formula for b (not shown)

- **Classifying new examples:**
    - **above the decision boundary: class 1**
    - **below: class -1**

Soft Margins

- The method above constructs decision boundaries that are free of misclassifications. We can allow some misclassifications.



- B_2 classifies these new examples correctly unlike B_1, but these examples may be noise, and still prefer the wide margin and lesser sensitivity to overfitting and noise of B_1.
- The optimisation problem formulation is similar for a **soft margin solution**, but there is an extra parameter C in the function that we want to maximise, corresponding to the tradeoff between error and margin.
    - The solution is the same as the hard margin, but there is an upper bound C on the values of λs.
- The modified method will still construct a linear boundary even if the data is not linearly separable.

The old formulation:

Find $\mathbf{w}$ and b such that
$\Phi(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Modified formulation incorporates slack variables:

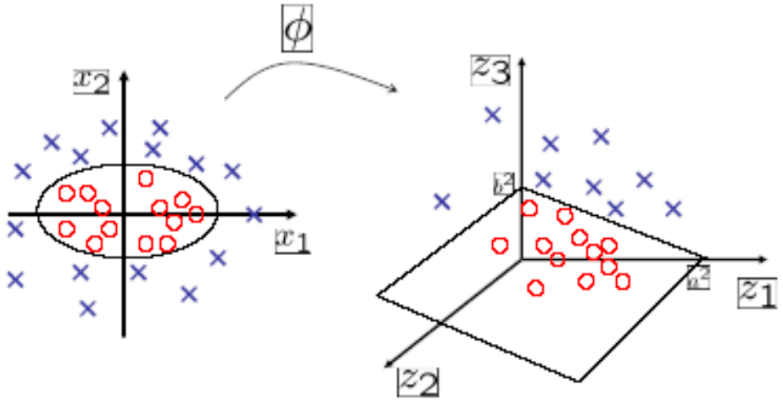
Find $\mathbf{w}$ and b such that
$\Phi(\mathbf{w}) = \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$, $\xi_i \geq 0$

Parameter C can be viewed as a way to control overfitting: it "trades off" the relative importance of maximizing the margin and fitting the training data.

Non-Linear SVM

- Most problems are linearly non-separable. In order to find a non-linear boundary, we want to transform the data from its original feature space to a new space where a linear boundary can be used to separate the data.
    - Cover, 1965: A complex classification problem cast in high dimensional space non-linearly is more likely to be linearly separable than in a low dimensional space.
- The transformation has two properties:
    - It is non-linear.
    - It is to a higher dimensional space.

Non-linearly separable data $\mathbf{x} = (x_1, x_2)$ in the original space and linearly separable in the new space



$\phi : (x_1, x_2) \longrightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2)$ transformation from old to new space

$\left(\frac{x_1}{a}\right)^2 + \left(\frac{x_2}{b}\right)^2 = 1 \longrightarrow \frac{z_1}{a^2} + \frac{z_3}{b^2} = 1$ decision boundaries in old and new space

original space (2-dim): (x_1, x_2)

new space (3-dim): $(x_1^2, \sqrt{2}x_1x_2, x_2^2)$

- Intuitively for example above, we transform our 2D shape into a hill, and use a plane to cut through.

Issue 1

- How do we choose the dimensionality of the new space so that the data is linearly separable in the new space?
  - Theorem: If a dataset is mapped to a space of sufficiently high dimension, it will always be linearly separable.
  - If we use a space of infinite dimensionality though, we run the risk of overfitting.
  - A line in a d-dimensional space is defined by an equation with d parameters $\rightarrow$ if $d \approx N$, overfitting i.e. restrictions on dimension defined by data.
- We also deal with overfitting by constructing the maximum margin hyperplane in the new space (structural risk minimisation principle).

Issue 2

- Even if we know what the transformation Φ should be, solving a QP task in a new, higher dimensional space is computational expensive.
  1. Transform the 2-dim training data into 3-dim using Φ.

$$\mathbf{x}_i \overset{\Phi}{\rightarrow} \Phi(\mathbf{x}_i), \mathbf{x}_j \overset{\Phi}{\rightarrow} \Phi(\mathbf{x}_j)$$

$\mathbf{x}_1, \mathbf{x}_2$ – a pair of training vectors in the original 2-dim feature space

  2. Feed the 3-dim vector to the QP solver. Recall that QP computes dot product pairs of training vectors:

$$\max W(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

Before (in the original 2-dim space)

$$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

Now (in the new 3-dim space)

- The computation increases dramatically for even more dimensions.

Kernel Trick

- We learn in higher dimensional space by computing kernel functions in lower dimensional space, instead of transforming each vector into the higher dimensional space and computing dot products between vectors.
- We need the dot product of the features in the new space (the transformed features):

$$\Phi(x_i) \cdot \Phi(x_j)$$

- We do NOT want to do the following:

$$1) x_i \overset{\Phi}{\rightarrow} \Phi(x_i), x_j \overset{\Phi}{\rightarrow} \Phi(x_j)$$
$$2) \Phi(x_i) \cdot \Phi(x_j)$$

- Instead, we want to specify K in the original space i.e. indirectly specifying Φ, and perform the dot product computation in the original space which has smaller dimensionality to find a linear boundary in the new higher dimensional space.

$$K(u, v) = \Phi(u) \cdot \Phi(v)$$

Method

2 dim original and 3 dim new space, $\Phi : (x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$

Consider a pair of vectors in original space: u, v (they are 2-dim). They were transformed into the (3 dim.) vectors Φ(u), Φ(v) in the new space, i.e.

$\mathbf{u} \overset{\Phi}{\rightarrow} \Phi(\mathbf{u}), \mathbf{v} \overset{\Phi}{\rightarrow} \Phi(\mathbf{v})$

Let's calculate the dot product of Φ(u) and Φ(v)

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (u_1^2, \sqrt{2}u_1u_2, u_2^2) \cdot (v_1^2, \sqrt{2}v_1v_2, v_2^2) =$$
$$= u_1^2v_1^2 + 2u_1u_2v_1v_2 + u_2^2v_2^2 = (u_1v_1)^2 + (u_2v_2)^2 + 2u_1u_2v_1v_2 =$$
$$= (u_1v_1 + u_2v_2)^2 = (\mathbf{u} \cdot \mathbf{v})^2$$

The dot product in the new space can be expressed via the dot product in the original space! $\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^2$ **Nice property!**

This means that the dot product in the new space can be computed without first computing Φ for each input vector! Instead we will compute a function in the original space to evaluate the dot product in the new space!

- In this case, the circled expression is called a kernel function $K(u, v)$.
- NOTE: functions K for which this is true need to satisfy Mercer's Theorem, which restricts the class of functions K we can use:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p - polynomial\ kernel$$
$$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{\|x-y\|^2}{2\sigma^2}} - RBF$$
$$K(\mathbf{x}, \mathbf{y}) = \tanh(k\mathbf{x} \cdot \mathbf{y} - \theta) - tanget\ hyperbolic$$
$$(satisfies\ Mercer's\ Th. only\ for\ some\ k\ and\ \theta)$$

- In terms of **TRAINING**:
  - **Original, without kernel function**

$$\max \mathbf{w}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$subject\ to\ \lambda_i \geq 0, \sum_{i=1}^{N} \lambda_i y_i = 0$$

solution

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i$$

Optimal hyperplane in the original space

  - **With kernel function**

$$\max \mathbf{w}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$subject\ to\ \lambda_i \geq 0, \sum_{i=1}^{N} \lambda_i y_i = 0$$

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \Phi(\mathbf{x}_i)$$

Optimal hyperplane in the new space

- When classifying **NEW EXAMPLES** (note it's still a summation over support vectors):

Classify new example z as sign(f), i.e. class 1 if f>0 and class -1 if f<0:

Original, without kernel function

$$f = \mathbf{w} \cdot \mathbf{z} + b = \sum_{i=1}^{N} \lambda_i y_i (\mathbf{x}_i \cdot \mathbf{z}) + b$$

Dot product of the new vector and the support vectors

With kernel function

$$f = \mathbf{w} \cdot \mathbf{z} + b = \sum_{i=1}^{N} \lambda_i y_i K(\mathbf{x}_i, \mathbf{z}) + b$$

Kernel function of the new vector and the support vectors

Furthermore

- Kernels have been designed for strings, trees and non-numeric data.
- Kernalisation can be applied to all algorithms that can be re-formulated to work only with dot products (this is then replaced with a kernel function) e.g. k-nearest neighbour etc.

SVM Summary and Comparison

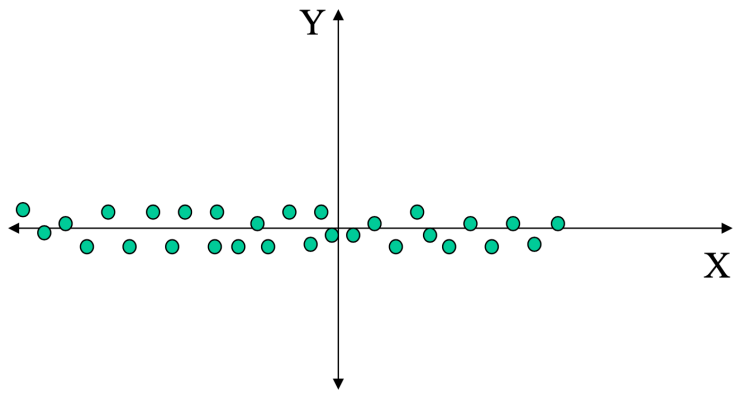
- Linear SVM - no kernel function, finds the optimal linear decision boundary between the positive and negative examples in the ORIGINAL feature space.
- Non-Linear SVM - uses kernel function, finds the optimal linear boundary in the NEW space.
  - This boundary when mapped back to the original feature space corresponds to a non linear decision boundary between positive and negative examples.
  - Scales well in high dimensions.
  - Multi-class problems need to be transformed into 2-class problems.
- Perceptrons: simple and efficient training algorithm but can only learn linear decision boundaries.
- Backpropagation NNs: hard to train with too many parameters, gets stuck in local minima, but can learn non-linear boundaries.
- SVM: relatively efficient training algorithm that can learn non-linear decision boundaries.

Dimensionality Reduction

- Some ML problems involve thousands of data, which leads to:
  - Slow training.
  - Unreliable classification given that examples are far away from each other; high dimensional data is very sparse.
  - Overfitting is more likely in high dimensional data.
  - Building interpretable models is not possible - we would like to build compact and easier to interpret classification models.
  - Visualising - humans can only interpret in low dimensional data e.g. max 3 dimensions.
  - Not all features are important - it is desirable to find a smaller set of features that are necessary and sufficient for good classification.
- Dimensionality reduction removes redundant and highly correlated features and reduces noise in the data.

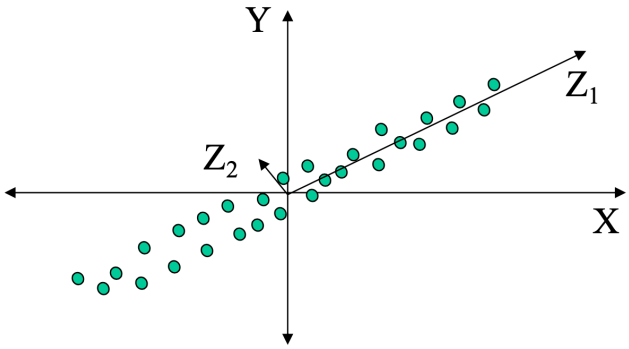
Principal Component Analysis

- PCA is the most popular dimensionality reduction method.
  - It is often called a **feature projection** method.
- The main idea is to find a new set of dimensions (axes) and project the data into it.
  - The dimensionality of the new space is smaller than the dimensionality of the original space.
  - The new axes capture the essence of the data (the variability of the data).
- The resulting dataset (the projection) can be used as an input to train a ML algorithm.
- Consider below, what axis has the maximum variability?



Answer: X

- What axis has the maximum variability?



Answer: Along another axis - Z_1

Max variability along Z_1, some variability along Z_2

- In this case, $Var(Z_1) > Var(Z_2)$ are a linear combination of X and Y.
  - $Z_1 = 1X + 1Y$
  - $Z_2 = -1X + 1Y$

Main Idea

- **Given**: N examples with dimensionality m i.e. m features
- **Find**: m

Note, Content on Face Recognition and Image Compression in Lecture

Summary

- PCA is a method for dimensionality reduction.
  - It is an unsupervised method - it doesn't use class information.
- The new axes are ordered based on how much variance they capture. They are called principal components.
  - We use only the first principal components since they have the highest variance.
- PCA can also be used for compression.