

DECISION TREES, RANDOM FOREST, SVM

Gerhard Widmer

Linz, Austria

www.cp.jku.at/people/widmer

Do not distribute!



May 5, 2025

Goals of this Lecture

- ▶ Introduce some popular learning algorithms for classification
- ▶ Review their relative advantages and shortcomings
- ▶ Give you a deeper understanding of what the algorithms you are using in your experiments actually do

Another important family of learning algorithms – those that learn to predict probabilities (specifically: Neural Networks) – will be the topic of the next chapter.

Outline

1 Recap: Simple 'Baseline' Classifiers

The Default Classifier

Naïve Bayes

The k -Nearest Neighbour Classifier

2 Other Popular Classifiers

A Classic: Decision Trees and the ID3 Algorithm

Robust Learning via Ensemble Building: Random Forest

Robustness via Linear Classification in High-dimensional Spaces: Support Vector Machines

3 Literature


The Baseline / Default Accuracy

From Section 4 of this Class:

Definition

The **Baseline Accuracy** (or **Default Accuracy**) of a dataset \mathcal{D} is

- = the prediction accuracy one would achieve by always predicting the class that is most frequent in the training set \mathcal{D}
- = the proportion of the most frequent class in the training data

 **Your classifier should be substantially better than the baseline**
(otherwise the added complexity of learning is not worth it)

The Naïve Bayes Classifier

Algorithm:

👉 see Section 3 (“Density Estimation”)

Advantages:

- ▶ Simple and efficient
- ▶ Fast learning and classification
- ▶ Can be made to learn incrementally ('on-line')
- ▶ Directly applicable to multi-class problems
- ▶ Outputs probabilities with its predictions
- ▶ Can be used in discrete domains and in very high-dimensional spaces

Disadvantages / Problems:

- ▶ "Naïve": independence assumption is rarely satisfied
- ▶ May give poor class probability estimates in domains with correlated features
- ▶ Very restricted model class (*high bias*): cannot take into account feature interactions

The k -Nearest-Neighbour Classifier

Algorithm:

👉 see Section 3 (“Density Estimation”)

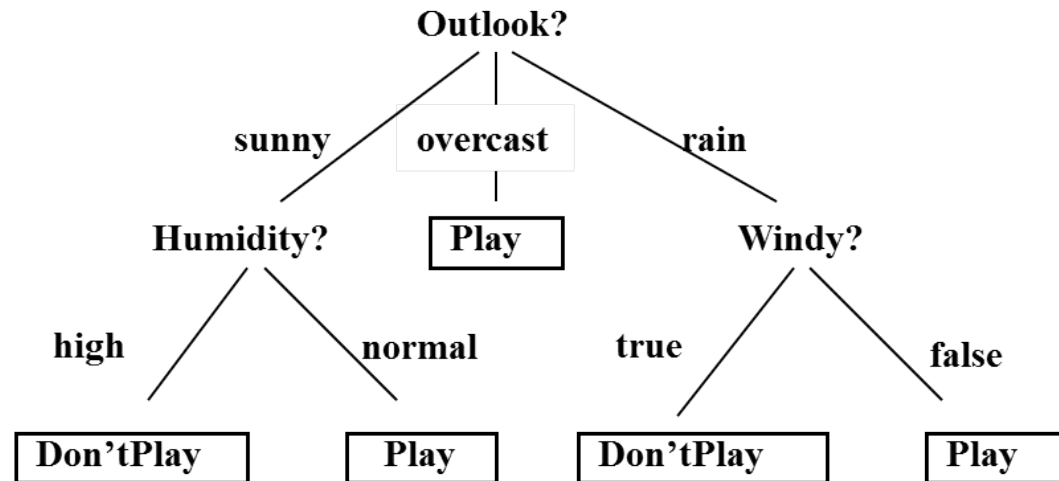
Advantages:

- ▶ Extremely simple
- ▶ Extremely short training time (none ...)
- ▶ Can learn incrementally ('on-line')
- ▶ Can represent arbitrarily complex decision boundaries (low *bias*)

Disadvantages / Problems:

- ▶ Slow at classification time (grows linearly with $|\mathcal{D}|$)
 - ▶ Does not produce an interpretable model
 - ▶ Sensitive to irrelevant features (*Curse of Dimensionality*)
- ⇒ problematic in high-dimensional feature spaces

A Classic: Decision Trees



Idea:

- ▶ A classification model that explicitly describes which combination of features the prediction depends on
- ▶ Decision tree \equiv a set of logical rules (implications)

$$\langle Conditions \rangle \Rightarrow Class$$

- ▶ Can be applied to new instances by testing conditions on features

A Little Discrete Toy Problem

Task:¹

- ▶ Learn, from the following example observations \mathcal{D} , to predict in what weather conditions a specific person is going to play tennis (for whatever reason ...)

Day	Outlook	Temp.	Humidity	Windy?	Class	
1	sunny	hot	high	false	Don't Play	-
2	sunny	hot	high	true	Don't Play	-
3	overcast	hot	high	false	Play	+
4	rain	mild	high	false	Play	+
5	rain	cool	normal	false	Play	+
6	rain	cool	normal	true	Don't Play	-
7	overcast	cool	normal	true	Play	+
8	sunny	mild	high	false	Don't Play	-
9	sunny	cool	normal	false	Play	+
10	rain	mild	normal	false	Play	+
11	sunny	mild	normal	true	Play	+
12	overcast	mild	high	true	Play	+
13	overcast	hot	normal	false	Play	+
14	rain	mild	high	true	Don't Play	-

¹from the classic decision tree paper (Quinlan, 1986)

Learning Decision Trees from Examples

Task:

- ▶ Given training set \mathcal{D} , find/construct a decision tree that optimises some criterion relative to \mathcal{D} (see below)
 - ▶ Given a fixed set of discrete features and classes, there is a **huge number** of possible decision trees one could build²
- ⇒ Decision tree learning is a **complex search problem**

Desirable Properties of a Decision Tree:

- ▶ Consistent with given training examples³
⇒ predict the correct class for all examples $x_i \in \mathcal{D}$
- ▶ More general than the set of training examples
⇒ make predictions on new, unseen cases!
- ▶ As **simple** as possible (simple will also tend to be general)



Our Goal: Find the simplest tree consistent with given data \mathcal{D}

²In fact, constructing a minimal decision tree is an **NP-complete problem** (Hyafil & Rivest, 1976)

³Will relax this later, when we talk about pruning ...

Excursus: Simplicity and Occam's Razor

Ockham's Principle ("Occam's Razor"):

“Non sunt multiplicanda entia praeter necessitatem.”
(Entities should not be multiplied beyond necessity.)

William of Ockham (1287? – 1347?)



General Interpretation of Occam's Razor in Science:

“Among the theories consistent with observed phenomena, one should prefer the simplest theory (all other things being equal).”

Not to be confused with Newton's (stronger) statement:

“Natura enim simplex est, et rerum causis superfluis non luxuriat.”
(For nature is simple, and does not afford the pomp of superfluous causes.)

Isaac Newton, Preface to "Principia"

(cited after Li & Vitányi, 2013)

The ID3 Algorithm for Learning Decision Trees⁴

Basic Ideas / Properties:

- ▶ **Recursive** algorithm: generate decision tree step by step, starting with the empty tree
- ▶ **Heuristic** algorithm: aim at obtaining a simple tree by heuristically choosing features to use for splitting at tree nodes
- ▶ **Greedy** algorithm: at each step, take decision (select feature) that maximises a local optimality criterion.

Consequences:

- ▶ **Non-optimal:** cannot guarantee finding the simplest possible tree
- ▶ **Short-sighted:** blind to features that are predictive only in combination (because heuristic evaluates features only individually)
- ▶ ... but **efficient** (fast)

⁴ *Iterative Dichotomizer* (Quinlan, 1986).

The ID3 Algorithm for Learning Decision Trees

Schema of the ID3 Algorithm

Start with root node containing all training examples: $\mathcal{D}_{Root} = \mathcal{D}$

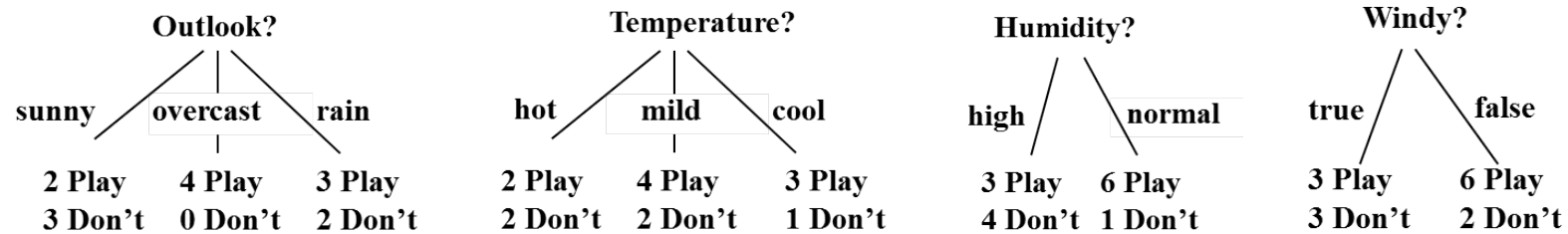
ID3:

- 1 If all examples \mathcal{D}_N in current node N belong to the same class ω
 \Rightarrow make current node a leaf, label with class ω , and EXIT
- 2 If no more features available for splitting in current node N ^a
 \Rightarrow make current node a leaf, label with majority class in N , and EXIT
- 3 Select “best” feature A_N to split instance set \mathcal{D}_N
- 4 Create a branch + successor node N_i for each possible value $v_i \in \text{Val}(A_N)$
- 5 Split training examples \mathcal{D}_N into subsets according to their value for feature A_N ; assign each subset to its respective branch/subnode
- 6 For each subnode: recursively call ID3

^aonly in discrete-feature domains

Central Question: What is the “best” feature to split on?

What is the 'Best' Feature?



Intuition:

- ▶ Best feature to split on is the one that is *most informative*
 - ▶ i.e., the one that best discriminates between the classes in the data
 - ▶ Expectation: is most likely to produce example subsets that are easily separable w.r.t. the class
- ⇒ simple tree (hopefully)

Heuristic evaluation function that formalises this intuition:

Information Gain

Entropy as a Measure of Uncertainty of a Distribution

Definition

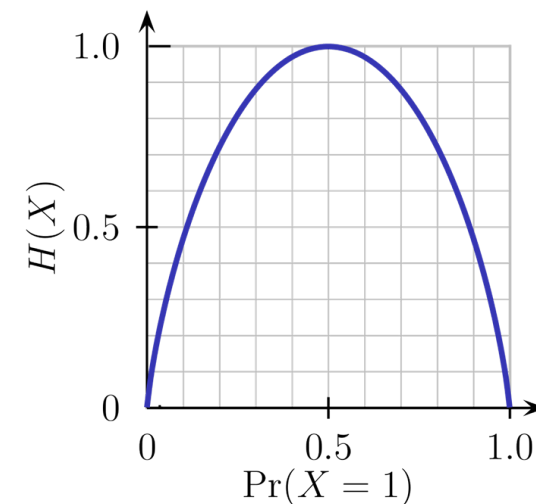
The **(Shannon) Entropy** of a distribution $P(X)$ over a discrete variable X is

$$H(X) = - \sum_{x \in X} P(x) \log P(x)$$

For a **binary** class distribution $\Omega = \{p, n\}$:

$$H(\Omega) = -P(p) \log P(p) - P(n) \log P(n)$$

Entropy of a Bernoulli trial as a function of success probability, often called the **binary entropy function**. The entropy is maximised at 1 bit per trial when the two possible outcomes are equally probable, as in an unbiased coin toss (or uniform binary class distribution):



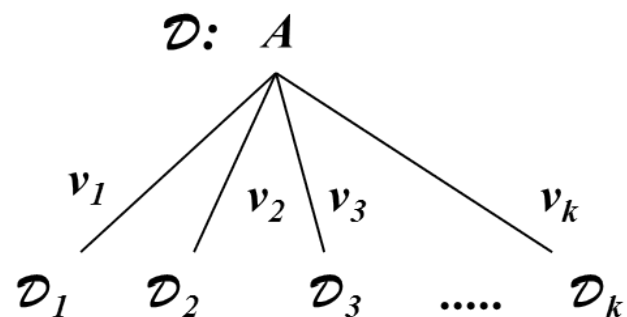
The Heuristic Measure: Information Gain

Definition

The **Information Gain** of a feature A on a dataset \mathcal{D} is the **expected reduction of uncertainty (entropy)** about the class labels, if we split \mathcal{D} according to A :

$$IG_{\mathcal{D}}(A) = E(\mathcal{D}) - \sum_{v_i \in A} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} E(\mathcal{D}_i)$$

where \mathcal{D}_i is the subset of all examples in \mathcal{D} that have value v_i for feature A .

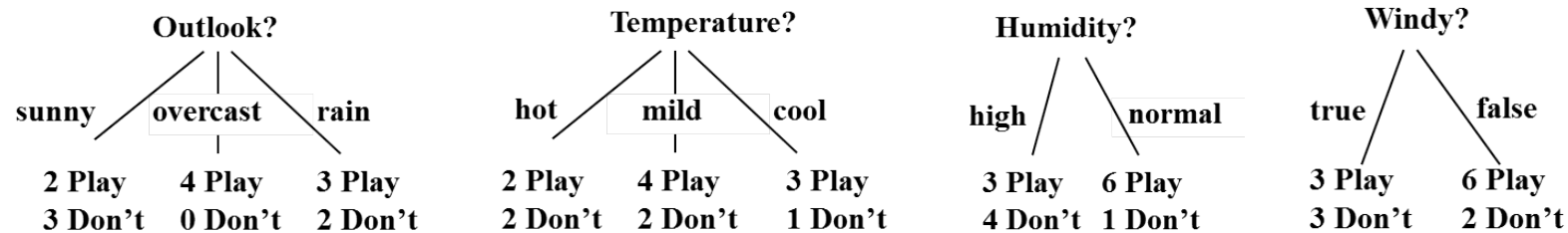


Note: $E(\mathcal{D})$ is independent of (i.e., the same for all) A

⇒ minimise 2^{nd} term (mean entropy of successors) instead of maximising IG

An Example: Selecting the Root Feature

$$IG_{\mathcal{D}}(A) = E(\mathcal{D}) - \sum_{v_i \in A} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} E(\mathcal{D}_i)$$



Class distribution in the root (full training set \mathcal{D}): 9 Play, 5 Don't Play

$$\Rightarrow E(\mathcal{D}) = -9/14 \cdot \log_2(9/14) - 5/14 \cdot \log_2(5/14) = 0.940 \text{ [bits]}$$

Information Gains:

$IG(Outlook)$	$= .940 - 5/14 \cdot .971 - 4/14 \cdot 0.00 - 5/14 \cdot .971$	$= \underline{0.246}$ [bits]
$IG(Temperature)$	$= .940 - 4/14 \cdot 1.00 - 6/14 \cdot .918 - 4/14 \cdot .811$	$= 0.029$ [bits]
$IG(Humidity)$	$= .940 - 7/14 \cdot .985 - 7/14 \cdot .592$	$= 0.151$ [bits]
$IG(Windy)$	$= .940 - 6/14 \cdot 1.00 - 8/14 \cdot .811$	$= 0.048$ [bits]

ID3: The Full Algorithm

The ID3 Algorithm

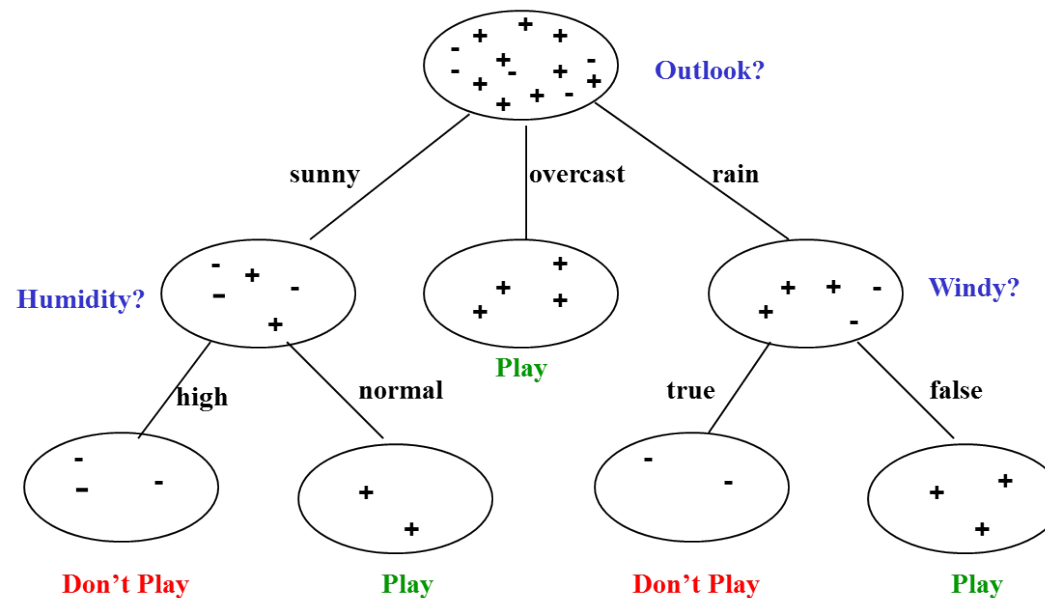
Start with root node containing all training examples: $\mathcal{D}_{Root} = \mathcal{D}$

ID3:

- 1 If all examples \mathcal{D}_N in current node N belong to the same class ω
 \Rightarrow make current node a leaf, label with class ω , and EXIT
- 2 If no more features available for splitting in current node N
 \Rightarrow make current node a leaf, label with majority class in N , and EXIT
- 3 Calculate $IG_{\mathcal{D}_N}(A_i)$ for all features A_i
(i.e., try out all features for splitting)
- 4 Select feature $A = \arg \max_{A_i} IG(A_i)$ with maximum information gain
- 5 Create a branch + successor node N_i for each value $v_i \in Val(A)$
- 6 Split training examples \mathcal{D}_N into subsets according to their value for feature A ; assign each subset to its respective branch/subnode
- 7 For each subnode: recursively call ID3

Example: The Final Tree

Day	Outlook	Temp.	Humidity	Windy?	Class	
1	sunny	hot	high	false	Don't Play	-
2	sunny	hot	high	true	Don't Play	-
3	overcast	hot	high	false	Play	+
4	rain	mild	high	false	Play	+
5	rain	cool	normal	false	Play	+
6	rain	cool	normal	true	Don't Play	-
7	overcast	cool	normal	true	Play	+
8	sunny	mild	high	false	Don't Play	-
9	sunny	cool	normal	false	Play	+
10	rain	mild	normal	false	Play	+
11	sunny	mild	normal	true	Play	+
12	overcast	mild	high	true	Play	+
13	overcast	hot	normal	false	Play	+
14	rain	mild	high	true	Don't Play	-



Numeric Features

Day	Outlook	Temp.	Humidity	Windy?	Class	
1	sunny	85	high	false	Don't Play	-
2	sunny	80	high	true	Don't Play	-
3	overcast	83	high	false	Play	+
4	rain	70	high	false	Play	+
5	rain	68	normal	false	Play	+
6	rain	65	normal	true	Don't Play	-
7	overcast	64	normal	true	Play	+
8	sunny	72	high	false	Don't Play	-
9	sunny	69	normal	false	Play	+
10	rain	75	normal	false	Play	+
11	sunny	75	normal	true	Play	+
12	overcast	72	high	true	Play	+
13	overcast	81	normal	false	Play	+
14	rain	71	high	true	Don't Play	-

👉 How to split on Temperature?

👉 Don't want to create a branch for each possible numeric value ...

Numeric Features

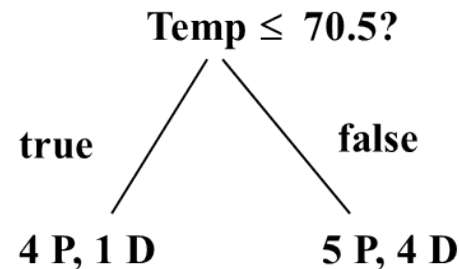
Common Solution:

- ▶ Create binary tests $A \leq t$ (for some threshold t)

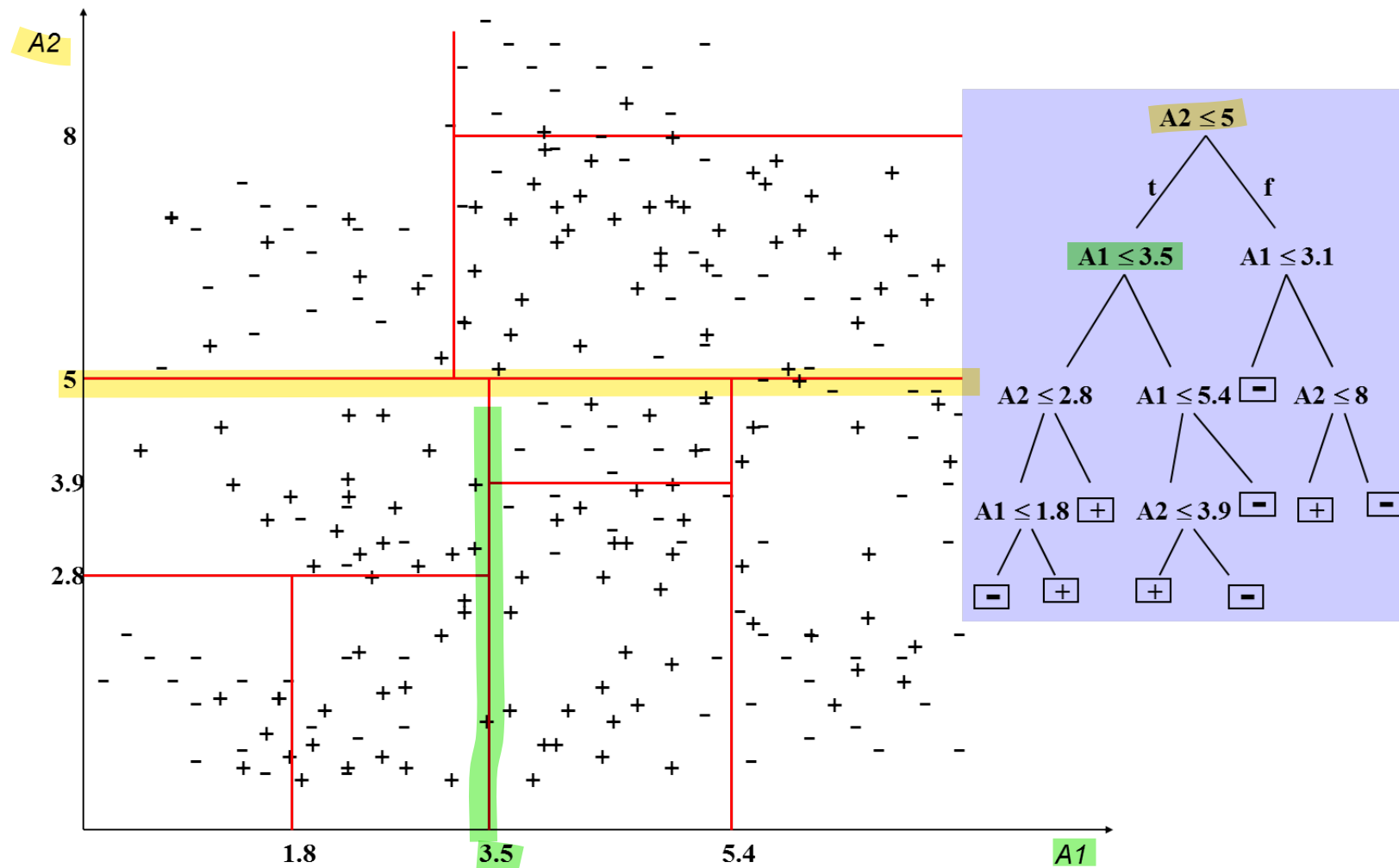
Finding the best t for feature A :

- ▶ Sort examples in current node N according to A
- ▶ Try each mid-point x_i between two values as possible split point
- ▶ Compute $IG(A \leq x_i)$ that would be produced by this split
- ▶ Select $t_A = \arg \max_{x_i} IG(A \leq x_i)$
- ▶ This test competes against all other features A for the best split overall

Temperature	64	65	68	69	70	71	72	75	80	81	83	85
<i>#Class Play :</i>	1	0	1	1	1	0	1	2	0	1	1	0
<i>#Class Don't :</i>	0	1	0	0	0	1	1	0	1	0	0	1



Decision Boundaries of Decision Trees: Hyper-Rectangles



👉 The standard ID3 algorithm would not stop splitting here ...

Overfitting Avoidance in Decision Trees: 'Pruning'

1. Pre-Pruning:

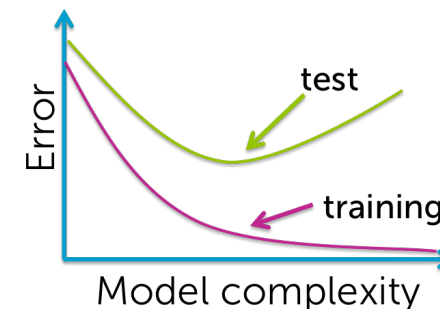
- ▶ Limit tree complexity already during construction
- ▶ Stop splitting a node further (even if it still contains examples of several classes) if remaining features don't seem to say much about the class any more (to be determined by some appropriate statistical test)

2. Post-Pruning:

- ▶ First construct (possibly complex) tree that is maximally consistent with (has minimum error on) the training data
- ▶ Then simplify the tree by cutting off branches/subtrees that seem harmful

Effect:

- ▶ Simpler trees
- ▶ with higher error on training data
- ▶ but possibly lower error on new data



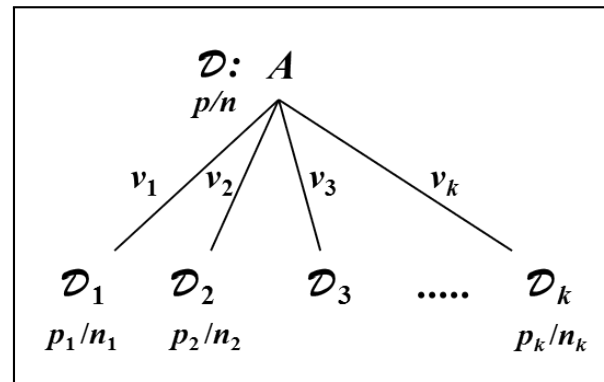
Pre-Pruning via Significance Testing

Question:

- In a given node N with set of instances \mathcal{D} , do any of the features still available for splitting carry significant information about the class?

Consider binary classification scenario:

(p/n ... number of positive/negative instances in a node)



If A were completely irrelevant (unrelated) to the class of objects in \mathcal{D} :

- ▶ Expected value p'_i (n'_i) of p_i (n_i) would be

$$p'_i = p \cdot \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \qquad n'_i = n \cdot \frac{|\mathcal{D}_i|}{|\mathcal{D}|}$$

Pre-Pruning via Significance Testing

If A were completely irrelevant (unrelated) to the class of objects in \mathcal{D} :

$$p'_i = p \cdot \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \quad n'_i = n \cdot \frac{|\mathcal{D}_i|}{|\mathcal{D}|}$$

- The larger the differences between expected and actually encountered values $(p_i - p'_i)$, the less likely it is that A is completely irrelevant

⇒ The statistic

$$S = \sum_{i=1}^k \frac{(p_i - p'_i)^2}{p'_i} + \frac{(n_i - n'_i)^2}{n'_i}$$

approximately follows a χ^2 **distribution** with $k - 1$ degrees of freedom

- Perform χ^2 **Significance Test**: S large enough for A being relevant?

👉 **Prune** (don't split current node) if there is no A that seems relevant at a given significance level. Predict **majority (most frequent) class** in this node

Post-Pruning via 'Reduced Error Pruning'⁵

Reduced Error Pruning (REP)

- 1 Randomly split training examples \mathcal{D} into a **Training Set** \mathcal{D}_{train} and a **Pruning (Validation) Set** \mathcal{D}_{val}
- 2 Learn a (possibly complex) tree \mathcal{T} from \mathcal{D}_{train} with minimum error (i.e., one that possibly overfits on \mathcal{D}_{train})
- 3 Iteratively simplify the tree, step by step:
 - for each subtree \mathcal{T}_i of \mathcal{T} , tentatively replace \mathcal{T}_i by a leaf (with majority class) and test/compare accuracy on validation set \mathcal{D}_{val}
 - if there is no \mathcal{T}_i that improves accuracy when removed: **EXIT**
 - otherwise: choose \mathcal{T}_i with maximum improvement and replace with leaf
- 4 Go to 3.

Question:

- ▶ Why additional validation set \mathcal{D}_{val} ?
- ▶ Why not use original training set \mathcal{D} for making pruning decisions?

⁵see (Quinlan, 1987)

ID3 and Decision Trees: Pros and Cons

Advantages:

- ▶ Efficient training algorithm
- ▶ Very fast prediction
- ▶ Produces an interpretable classification model
- ▶ Can represent arbitrarily complex decision boundaries (low *bias*)

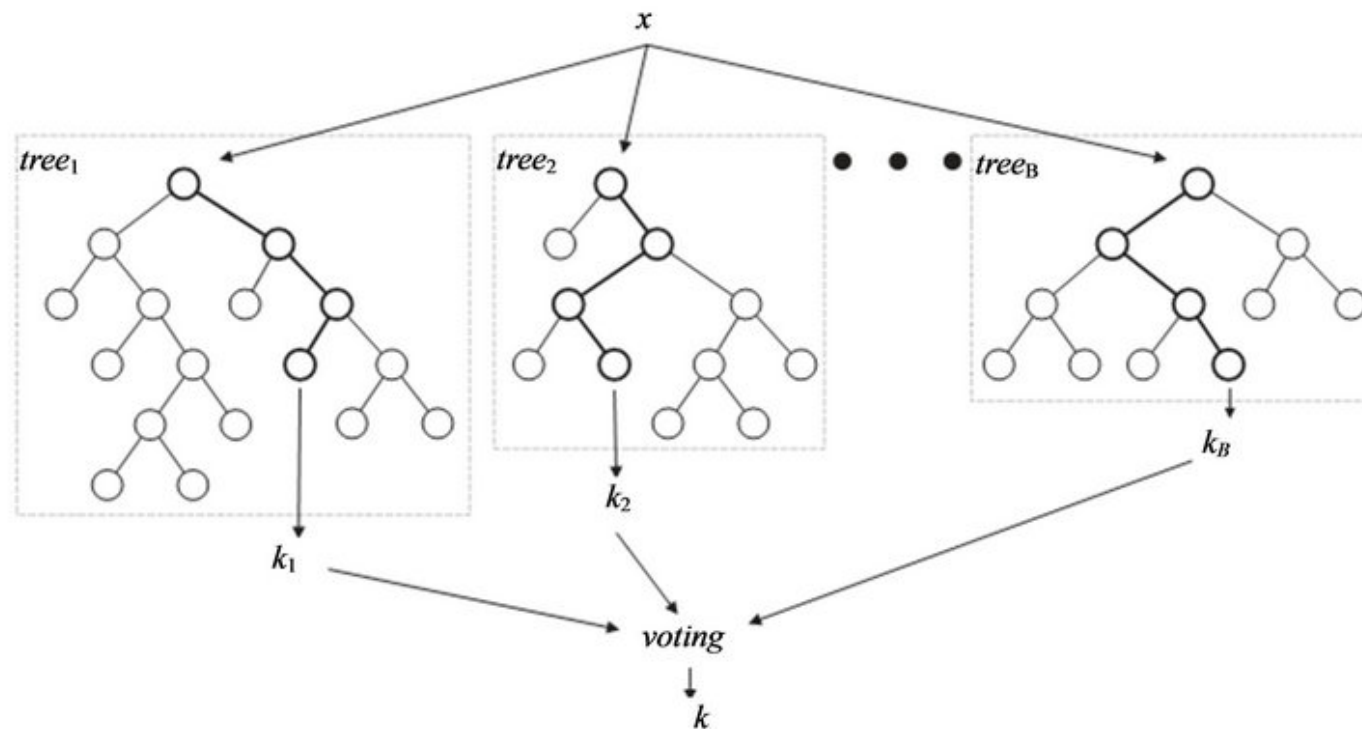
Problems:

- ▶ Cannot learn incrementally
- ▶ Blind to features that are relevant only **in combination** (because features are evaluated and split on individually, in isolation)
⇒ may use irrelevant features for splitting (Example: XOR-type concepts ...)
- ▶ High *variance* ⇒ Can overfit easily

Robust Learning via Ensembles: The Random Forest Classifier⁶

Basic Idea:

- ▶ Instead of learning one decision tree, build N (a large number of) **different trees** from the same data (👉 a *forest*)
- ▶ and combine their predictions on a new object by **voting**



⁶see (Breiman, 2001)

The Basic Idea

Motivation:

- ▶ Decision trees are prone to overfitting – have low bias, high variance
- ▶ Research in statistics shows: variance/overfitting can be reduced (with some increase in bias) by combining several high-variance (overfitting) models
- ▶ Important precondition: models should be **uncorrelated** (i.e., make their prediction errors on *different* examples)!

Central Question:

- ▶ How to generate a set of uncorrelated trees from one training set \mathcal{D} ?

Answer: Introduce **Randomisation**

- 1 Generate N (slightly) different training sets $\mathcal{D}'_1, \dots, \mathcal{D}'_N$ by **random sampling of training examples** from given set \mathcal{D} (often: Sampling *with replacement* ➡ **“Bagging”**); or: = Bootstrap Aggregating
- 2 Generate N trees from same training set \mathcal{D} , but introduce randomisation in node splitting process: at each node, select and **consider only a random subset of the features** as candidates for splitting.

Statistical Analysis Shows:

- ▶ Feature randomisation is more effective in producing uncorrelated trees.

The Random Forest Algorithm

Random Forest Learning Algorithm

Given: Set of training examples \mathcal{D}

Do:

- ▶ Learn N decision trees from \mathcal{D}
- ▶ without any pruning ...
- ▶ ... but with **randomised feature selection** at every learning step:
- ▶ At each node, randomly select a subset of k features to consider for information gain and splitting
- ▶ Common heuristic: $k = \sqrt{d}$ where d is the total number of features

Important:

- ▶ No (or little) pruning of individual trees
- ⇒ Individual trees *overfit* the training data *in different ways*
- ⇒ **High variance, low correlation** (as desired)

Random Forest: Pros and Cons

Advantages:

- ▶ Robust against overfitting and noise in the data
- ▶ Fast classification: prediction of N trees can be parallelised

Disadvantages:

- ▶ No interpretable model
- ▶ Cannot learn incrementally

Robustness via Linear Classification: Support Vector Machines

Starting Observations:

- ▶ Many real-world classification problems have complex decision boundaries
⇒ cannot be adequately learned by high-bias learners (e.g., Naïve Bayes)
- ▶ But fitting complex boundaries with low-bias/high-variance classifiers is dangerous (⇒ *Overfitting!*) ...
- ▶ ... while restricted, high-bias classifiers (e.g., linear discriminant) would be robust against noise and have simple and efficient learning algorithms

Question:

- ▶ How can we combine the robustness of simple (linear) classifiers with the need for complex decision boundaries?

Central Insight:

- ▶ Many classification problems, when projected into a **high-dimensional feature space**, become **linearly separable** (or nearly)

Non-linear Mapping of Feature Spaces⁷

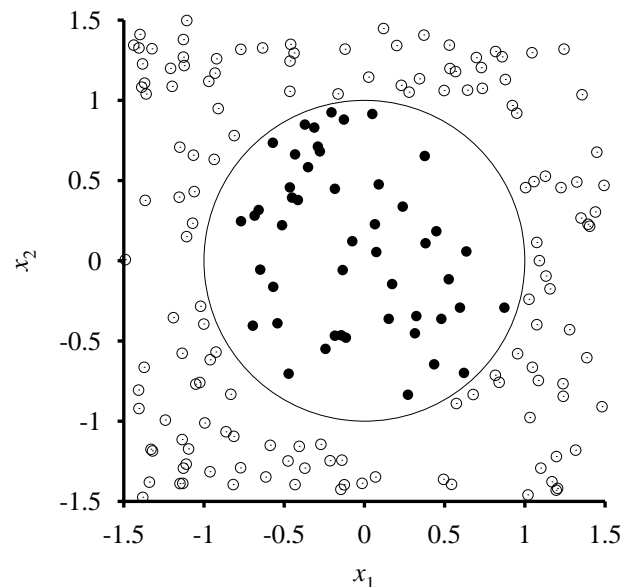


Figure : A 2-dimensional training set with positive examples as black circles and negative examples as white circles. The true (non-linear) decision boundary $x_1^2 + x_2^2 = 1$ is also shown.

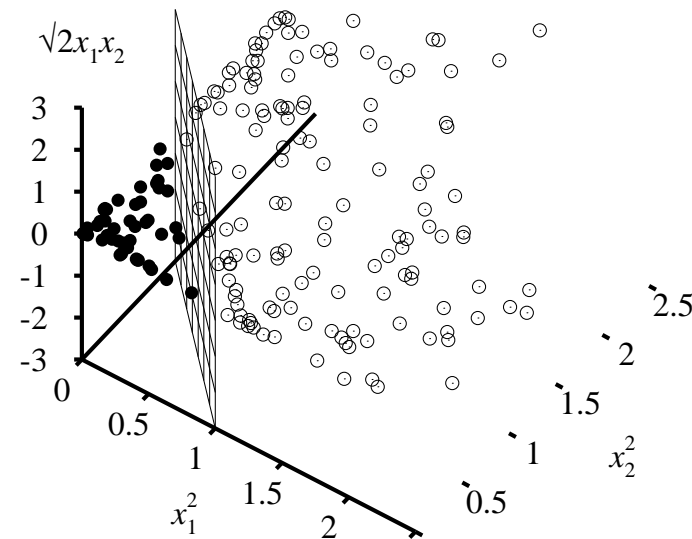


Figure : The same data after mapping into a new (3D) feature space, with features $a_1 = x_1^2$, $a_2 = x_2^2$, $a_3 = \sqrt{2}x_1x_2$. The circular decision boundary in 2D becomes a linear boundary (a plane) in these three dimensions.

⁷Figures from (Russell & Norvig, 2002)

Support Vector Machines (SVMs)⁸

Basic Idea of SVMs (for binary classification tasks):

- ▶ Map data into a (very) high-dimensional space by a transform
 $\varphi : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}, x \mapsto \varphi(x)$ with $d_1 < d_2$
- ▶ Find the optimal linear decision boundary between the two classes in this high-dimensional space
- ▶ 'Optimal' = the linear boundary that maximises the distance (the **Margin**) to the nearest examples (the **Support Vectors**) of the two classes

Maximum Margin Hyperplane

- ▶ If training points are not linearly separable, find hyperplane that minimises the number & size of errors ( **Soft Margin Method**)

Feasible for *very* high dimensions d_2 :

- ▶ Implicitly define new features from the given ones via a **(non-linear) Kernel Function**
- ▶ Through the **Kernel Trick**, high-dimensional features need never be explicitly computed ... (see below)

⁸see (Cortes & Vapnik, 1995)

Support Vectors and Maximum Margin Hyperplane⁹

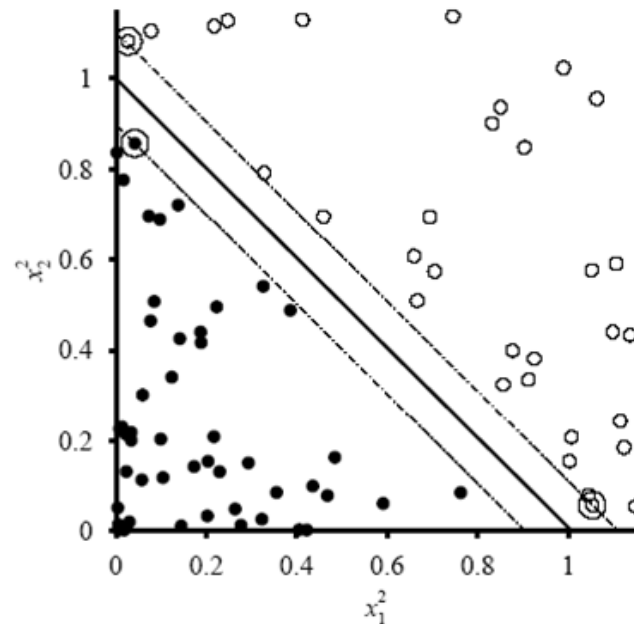


Figure : A close-up, projected onto the first two dimensions, of the optimal separator plane shown in the previous figure. The separator is shown as a heavy line, with the closest points – the **support vectors** – marked with circles. The **margin** is the space separating the positive from the negative examples. The separator hyperplane chosen by the algorithm is the one with the largest margin – the **maximum margin hyperplane**. Note that only the examples closest to the margin – the support vectors – are needed to determine the exact position and direction of the separating hyperplane; vectors “hidden” behind these cannot influence the positioning, and are not needed for computing it.

⁹Figure from (Russell & Norvig, 2002)

Simplified Scenario: Optimal Separation in Original Space

- ▶ Given: Feature space \mathbf{X} defined by a set of d features $\mathcal{X} = \{X_1, \dots, X_d\}$
- ▶ Given: Training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ with $\mathbf{x}_i \in \mathbf{X}$ and $y_i \in \Omega = \{-1, +1\}$ ¹⁰
- ▶ Find: A (hyper)plane in feature space \mathbf{X} that separates the two classes

Linear Classifier:

- ▶ Hyperplane = linear function of the features: all points that satisfy

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

for some vector of coefficients (weights) $\mathbf{w} \in \mathbb{R}^d$ and some offset $b \in \mathbb{R}$

- ▶ $\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x}$ is the *dot (inner, scalar) product*
- ▶ \mathbf{w} is a **normal vector** to the plane
- ▶ The hyperplane is used as a **classifier (linear discriminant function)**:

$$\hat{y}(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

In Words: The class label \hat{y} predicted for an example vector \mathbf{x} is the sign (+1 or −1) of the output of the linear function for \mathbf{x} . (Examples \mathbf{x} directly on the plane produce 0)

¹⁰SVMs are limited to binary classification problems; we encode the two classes as −1 and +1

Linear SVM: Hard-margin Separation¹¹

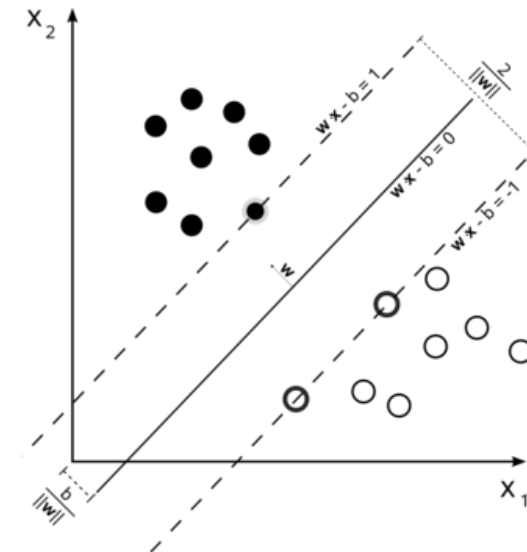
If training data are **linearly separable**:

- ▶ Can select two parallel hyperplanes h_1, h_2 that separate the two classes, so that distance between them is as large a possible
- ▶ Can be described (for some appropriate w, b) as

$$h_1 : \quad w \cdot x + b = 1$$

$$h_2 : \quad w \cdot x + b = -1$$

- ▶ Distance between these is the **margin**
(of width $\frac{2}{\|w\|}$)
- ▶ Max margin hyperplane lies halfway between these



¹¹ Figure from https://en.wikipedia.org/wiki/Support_vector_machine

The Learning Task as an Optimisation Problem

Optimisation Criterion: Maximise the Margin

- ▶ Distance (margin) between hyperplanes h_1, h_2 is $\frac{2}{\|w\|}$
- ▶ To maximise the margin, we need to minimise $\|w\|$ (or $\|w\|^2$)

Constraints: Agreement with Training Examples

- ▶ Training examples (x_i, y_i) must not lie inside or on wrong side of the margin
- ▶ For each $(x_i, y_i) \in \mathcal{D}$:

$$\begin{aligned} w \cdot x_i + b &\geq 1 & \text{if } y_i = +1 \\ w \cdot x_i + b &\leq -1 & \text{if } y_i = -1 \end{aligned}$$

which can be summarised as:

$$y_i(w \cdot x_i + b) \geq 1 \quad \text{for all } x_i \in \mathcal{D}$$

Resulting Optimisation Problem:

Find w and b that minimise $\|w\|^2$
 subject to the constraints $y_i(w \cdot x_i + b) \geq 1$, for $i = 1, \dots, n$

Linear SVM: The Soft-margin Case

If the classes are **not** linearly separable:

- ▶ Introduce variables ξ_i that quantify prediction errors for examples (x_i, y_i) :

$$\xi_i = \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b)) \quad (\text{"hinge loss"})$$

- ▶ $\xi_i = 0$ if constraint for example i is satisfied (x_i is on correct side of margin), proportional to the error (distance from the margin) otherwise.

Resulting Optimisation Problem:

Find \mathbf{w} and b that minimise

$$\left[\frac{1}{n} \sum_{i=1}^n \xi_i \right] + \lambda \cdot \|\mathbf{w}\|^2$$

where parameter λ controls the trade-off between maximising the margin size and minimising classification errors.



Learning problem can be solved by standard quadratic optimisation.

The Full Story: SVM with Non-linear Data Projection

(... now it gets difficult to tell the story without going into the details of optimisation ...)

Shortened Version of the Story:

Goal:

- ▶ Want to **map the data** from the original feature space \mathbb{R}^{d_1} to a higher-dimensional space \mathbb{R}^{d_2} , with $d_2 \gg d_1$, via a transform function $\varphi : \mathbf{x} \mapsto \varphi(\mathbf{x})$, so that transformed problem becomes linearly separable
- ▶ Example: φ might construct all pairwise products of features
- ▶ Could do this explicitly: map training data \mathcal{D} to $\varphi(\mathcal{D})$ and solve above optimisation problem
- ▶ Obtain linear classifier (with d_2 -dimensional weight vector \mathbf{w}) in high-dimensional space
- ▶ To classify a new object \mathbf{x} : map \mathbf{x} into new space and apply high-dimensional classifier to it: $\hat{y}(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \varphi(\mathbf{x}) + b)$

Problem:

- ▶ Dimensionality d_2 of new space (and thus size of \mathbf{w} and $\varphi(\mathbf{x})$) may be huge!

The Full Story: SVM with Non-linear Data Projection

Solution:

- ▶ We transform the optimisation problem into its equivalent *dual formulation* ...
... where it turns into something that involves example vectors x, y only as part of **dot (scalar) products** ($x \cdot y$)
- ▶ Consequence: also the **classification rule** for a query object x changes:

$$\hat{y}(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \varphi(\mathbf{x}) + b) = \text{sgn} \left(\sum_{i=1}^n \alpha_i y_i (\varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x})) + b \right)$$

where x_i are the training examples, and $\alpha_i \neq 0$ only for the **support vectors** (which are identified in the learning process)...

And the most important point: **The Kernel Trick**

- We replace the dot product ($\varphi(\mathbf{x}) \cdot \varphi(\mathbf{y})$) by an appropriate **kernel function** $K(\mathbf{x}, \mathbf{y})$ that has the central property

$$K(\boldsymbol{x}, \boldsymbol{y}) = (\varphi(\boldsymbol{x}) \cdot \varphi(\boldsymbol{y}))$$

In Words: The kernel function implicitly corresponds to a specific feature mapping φ , and computing the kernel function in the original (low-dimensional) space (on x, y) corresponds to the dot product of the corresponding vectors $\varphi(x), \varphi(y)$ in the high-dimensional space.

The Full Story: SVM with Non-linear Data Projection

$$K(\mathbf{x}, \mathbf{y}) = (\varphi(\mathbf{x}) \cdot \varphi(\mathbf{y}))$$

Consequence:

- ▶ We can **completely avoid computation in the high-dimensional space**, replacing the needed high-dimensional dot products with kernel computations in the original space!

Class Prediction:

- ▶ The classification rule using the learned hyperplane changes from

$$\hat{y}(\mathbf{x}) = \text{sgn} \left(\sum_{i=1}^n \alpha_i y_i (\varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x})) + b \right)$$

to

Class Prediction in SVM

$$\hat{y}(\mathbf{x}) = \text{sgn} \left(\sum_{i=1}^n \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \right)$$

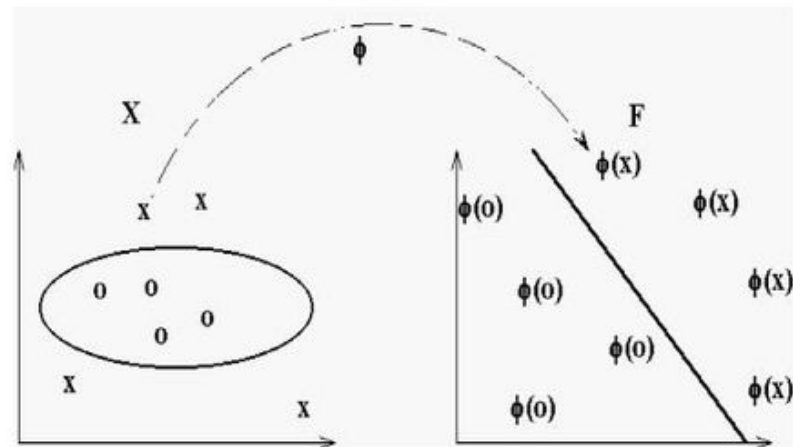
An Example: The Polynomial Kernel

The Polynomial Kernel

$$K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x} \cdot \boldsymbol{y} + c)^k$$

- ▶ Can be easily computed for given vectors and yields a number (scalar)
- ▶ *Implicitly* introduces a large number of products and powers of features
- ▶ $c \geq 0$ is a parameter that trades off the influence of higher-order vs. lower-order terms in the polynomial

Figure : Illustration of the mapping φ : On the left a set of samples in the input space, on the right the same samples in the feature space mapped to by the polynomial kernel $K(\mathbf{x}, \mathbf{y})$ (for some values of the parameters c and k). The hyperplane learned by an SVM in the new feature space is an **ellipse** in the input space.



An Example: The Polynomial Kernel

Simple Case: $k = 2$ ('Quadratic Kernel')

The Quadratic Kernel

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) &= (\mathbf{x} \cdot \mathbf{y} + c)^2 \\ &= \left(\sum_{i=1}^n x_i y_i + c \right)^2 \\ &= \sum_{i=1}^n x_i^2 y_i^2 + \sum_{i=2}^n \sum_{j=1}^{n-1} (\sqrt{2} x_i x_j)(\sqrt{2} y_i y_j) + \sum_{i=1}^n (\sqrt{2} c x_i)(\sqrt{2} c y_i) + c^2 \end{aligned}$$

Corresponding Feature Mapping:

Mapping Function φ of the Quadratic Kernel

$$\begin{aligned} \varphi(\mathbf{x}) = & \{x_n^2, \dots, x_1^2, \\ & \sqrt{2}x_n x_{n-1}, \dots, \sqrt{2}x_n x_1, \dots, \sqrt{2}x_{n-1} x_{n-2}, \dots, \sqrt{2}x_{n-1} x_1, \dots, \sqrt{2}x_2 x_1, \\ & \sqrt{2}c x_n, \dots, \sqrt{2}c x_1, c\} \end{aligned}$$

Got it?

$$\underline{K(\mathbf{x}, \mathbf{y})} = K\left(\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}\right) = x_1^2 y_1^2 + x_2^2 y_2^2 + \dots \\ + (\sqrt{2} x_1 x_2)(\sqrt{2} y_1 y_2) + (\sqrt{2} x_1 x_3)(\sqrt{2} y_1 y_3) + \dots \\ + (\sqrt{2c} x_1)(\sqrt{2c} y_1) + (\sqrt{2} x_2)(\sqrt{2} y_2) + \dots + c^2$$

$$= \begin{pmatrix} x_1^2 \\ x_2^2 \\ \vdots \\ \sqrt{2} \, x_1 x_2 \\ \sqrt{2} \, x_1 x_3 \\ \vdots \\ \sqrt{2c} \, x_1 \\ \sqrt{2c} \, x_2 \\ \vdots \\ c \end{pmatrix} \cdot \begin{pmatrix} y_1^2 \\ y_2^2 \\ \vdots \\ \sqrt{2} \, y_1 y_2 \\ \sqrt{2} \, y_1 y_3 \\ \vdots \\ \sqrt{2c} \, y_1 \\ \sqrt{2c} \, y_2 \\ \vdots \\ c \end{pmatrix} = \underline{(\varphi(\mathbf{x}) \cdot \varphi(\mathbf{y}))}$$

- ▶ with $\varphi(\mathbf{x}) = \{x_1^2, x_2^2, \dots, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \dots, \sqrt{2}cx_1, \sqrt{2}cx_2, \dots, c\}$
- ▶ and where for concrete feature vectors \mathbf{x}, \mathbf{y} , the numeric value of $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + c)^2$ can be easily calculated without expansion.

Support Vector Machines: Pros and Cons

Advantages:

- ▶ Can approximate highly complex decision boundaries (*low bias*)
- ▶ Robust against overfitting and noise in the data
- ▶ Can be used in very high-dimensional classification problems (e.g., document classification)

Disadvantages:

- ▶ No interpretable model
- ▶ Only directly applicable to 2-class problems
- ▶ Relatively slow in training \Rightarrow problematic for huge datasets
- ▶ Cannot learn incrementally

What to expect in the exam: A few typical questions ...

Consider a set \mathcal{D} of training examples, described in terms of N numeric features f_1, \dots, f_N . Assume we create a new training set $\mathcal{D}1$ from \mathcal{D} by duplicating one of these features, f_i , 10 times; in other words: by adding 10 new features f_{N+1}, \dots, f_{N+10} to the data that are all identical to f_i . (That is, we change the data set, not by adding new objects, but by adding 10 copies of a feature to the description of each object.)

When we replace the training set \mathcal{D} by the new training set $\mathcal{D}1$, which of the following learning algorithms may be affected by this feature duplication (in terms of learning a classifier that now predicts differently):

- ▶ k -NN with $k = 1$ and Euclidean distance
- ▶ k -NN with $k > 1$ and Euclidean distance
- ▶ Standard Decision Tree Learner (ID3)
- ▶ A restricted version of ID3 that can only learn “decision stumps” (i.e., a one-level tree with a feature at the root and leaves below that)
- ▶ Naïve Bayes
- ▶ A feed-forward neural network without hidden layer (i.e., logistic regression)
- ▶ A Support Vector Machine with the Quadratic Kernel as explained in the lecture slides
- ▶ A maximum margin linear classifier using only the given features

