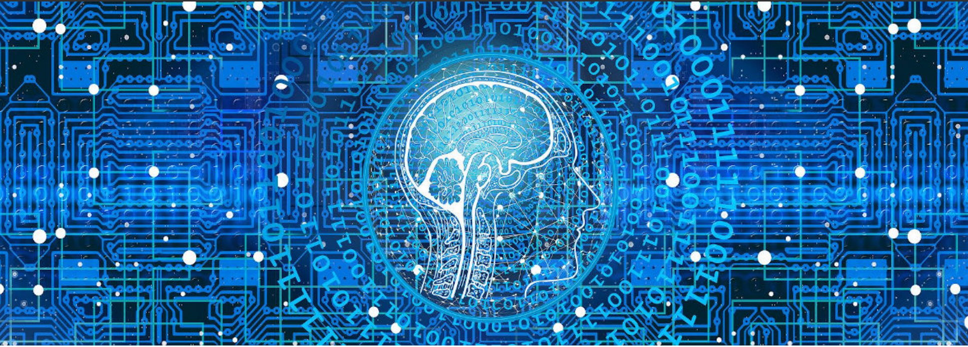


Chapter 13: Machine Learning

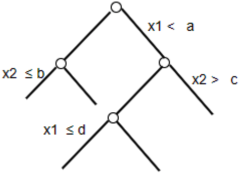
References

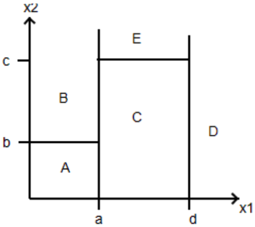
(A) https://cogsys.uni-bamberg.de/teaching/ws0506/ml/slides/cogsysII-3.pdf

Introduction

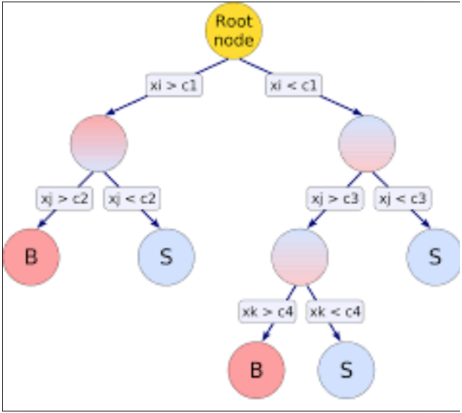
- The field of machine learning received significant attention in recent years and we have many dedicated lectures on this topic (Machine Learning, Neural Networks, ...)
- In this course we will just give a brief introduction to one of the most common tasks in machine-learning: Classification
  - Classification is a task, where an algorithm should decide if a given input pattern belongs a certain class, e.g. if a picture shows an apple, a face or a tree
- We will only discuss two classifiers, which turned out to be performant before the "deep neural network" revolution
  - (Boosted) Decision Trees
  - Support Vector Machines

What is a decision Tree (1/2)





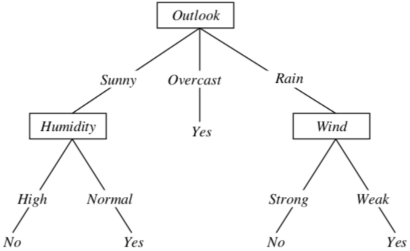
What is a decision Tree (2/2)



Decision Tree Representation

- classification of instances by sorting them down the tree from the root to some leaf note
    - node ≈ test of some attribute
    - branch ≈ one of the possible values for the attribute
- decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances
    - i.e., $(\ldots\wedge\ldots\wedge\ldots)\vee(\ldots\wedge\ldots\wedge\ldots)\vee\ldots$
- equivalent to a set of if-then-rules
    - each branch represents one if-then-rule
- if-part: conjunctions of attribute tests on the nodes
- then-part: classification of the branch

Decision Tree Representation



- This decision tree is equivalent to:
  - if $(Outlook = Sunny) \wedge (Humidity = Normal)$ then Yes;
  - if $(Outlook = Overcast)$ then Yes;
  - if $(Outlook = Rain) \wedge (Wind = Weak)$ then Yes;

Appropriate Problems

- Instances are represented by attribute-value pairs, e.g. (Temperature, Hot)
- Target function has discrete output values, e.g. yes or no
- Disjunctive descriptions may be required
- Training data may contain errors
- Training data may contain missing attribute values

ID3-Algorithm to build decision trees

- learns decision trees by constructing them top-down
- employs a greedy search algorithm without backtracking through the space of all possible decision trees
  - → finds the shortest but not necessarily the best decision tree

$$Gain(S,A) = \underbrace{Entropy(S)}_{\text{original entropy of S}} - \underbrace{\sum_{v \in values(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)}_{\text{relative entropy of S}}$$

- key idea:
  - selection of the next attribute according to a statistical measure
  - all examples are considered at the same time (simultaneous covering)
  - recursive application with reduction of selectable attributes until each training example can be classified unambiguously

Full ID3 algorithm

$ID3(Examples, Target\_attribute, Attributes)$

- Create a $Root$ for the tree
- If all examples are positive, Return single-node tree $Root$, with label $= +$
- If all examples are negative, Return single-node tree $Root$, with label $= -$
- If $Attributes$ is empty, Return single-node tree $Root$, with label $=$ most common value of $Target - attribute$ in $Examples$
- otherwise, Begin
  - A $\leftarrow$ attribute in $Attributes$ that best classifies $Examples$
  - decision attribute for $Root \leftarrow A$
  - For each possible value v_i of A
- Return $Root$

The best classifier

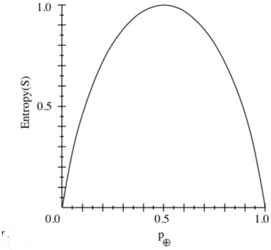
- central choice: Which attribute classifies the examples best?
- ID3 uses the information gain
  - statistical measure that indicates how well a given attribute separates the training examples according to their target classification
- interpretation:
  - denotes the reduction in entropy caused by partitioning S according to A
  - alternative: number of saved yes/no questions (i.e., bits)
- $\rightarrow$ attribute with $max_A Gain(S, A)$ is selected!

Entropy

What is entropy?

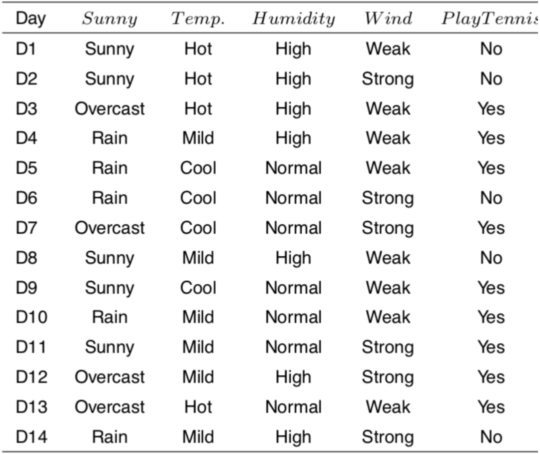
- statistical measure from information theory that characterizes (im-)purity of an arbitrary collection of examples S
  - for binary classification: $H(S) := -p \bigoplus log_2 p \bigoplus -p \bigoplus log_2 p \bigoplus$
  - for n-ary classification: $H(S) := \sum_i -p_i log_2 p_i$
- interpretation:
  - specification of the minimum number of bits of information needed to encode the classification of an arbitrary member of S
  - alternative: number of yes/no questions

Entropy



- minimum of $H(S)$
  - for minimal impurity → point distribution
  - $H(S) = 0$
- maximum of H(S)
  - for maximal impurity → uniform distribution
  - for binary classification: $H(S) = 1$
  - for n-ary classification: $H(S) = log_2 n$

Example



Day	*Sunny*	*Temp.*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example

- entropy of S

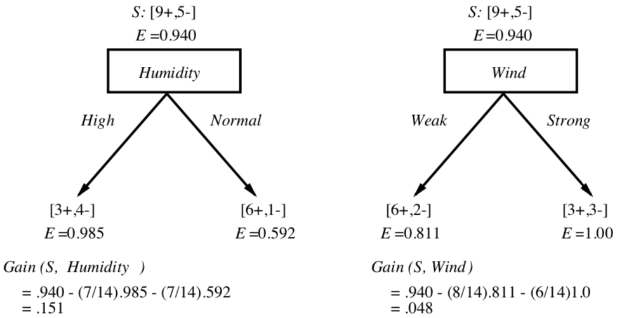
$$S = \{D1, ..., D14\} = [9+, 5-]$$
$$H(S) = -\tfrac{9}{14} \cdot \log_2 \tfrac{9}{14} - \tfrac{5}{14} \cdot \log_2 \tfrac{5}{14} = 0.940$$

- information gain (e.g. Wind)
  - $S_{Weak} = \{D_1, D_3, D_4, D_5, D_8, D_9, D_{10}, D_{13}\} = [6+, 2-]$
  - $S_{Strong} = \{D_2, D_6, D_7, D_{11}, D_{12}, D_4\} = [3+, 3-]$

$$\begin{aligned} Gain(S, Wind) &= H(S) - \sum_{v \in Wind} \frac{|S_v|}{|S|} \cdot H(S_v) \\ &= H(S) - \frac{8}{14} \cdot H(S_{Weak}) - \frac{6}{14} \cdot H(S_{Strong}) \\ &= 0.940 - \frac{8}{14} \cdot 0.811 - \frac{6}{14} 1.000 \\ &= 0.048 \end{aligned}$$

Example

- Which attribute is the best classifier?



S: [9+,5-]
E =0.940

Humidity

High *Normal*

[3+,4-]
E =0.985

[6+,1-]
E =0.592

Gain (*S*, *Humidity*)

= .940 - (7/14).985 - (7/14).592
= .151

S: [9+,5-]
E =0.940

Wind

Weak *Strong*

[6+,2-]
E =0.811

[3+,3-]
E =1.00

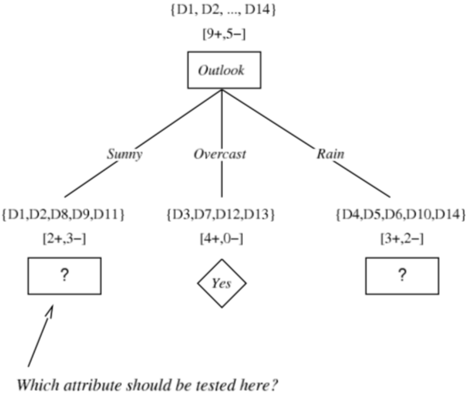
Gain (*S*, *Wind*)

= .940 - (8/14).811 - (6/14)1.0
= .048

Example

- informations gains for the four attributes:
  - Gain(S, Outlook) = 0.246
  - Gain(S, Humidity) = 0.151
  - Gain(S, Wind) = 0.048
  - Gain(S, Temperature) = 0.029
- Outlook is selected as best classifier and is therefore Root of the tree
- now branches are created below the root for each possible value
  - because every example for which Outlook = Overcast is positive, this node becomes a leaf node with the classification Yes
- the other descendants are still ambiguous
- hence, the decision tree has to be further elaborated below these nodes

Example



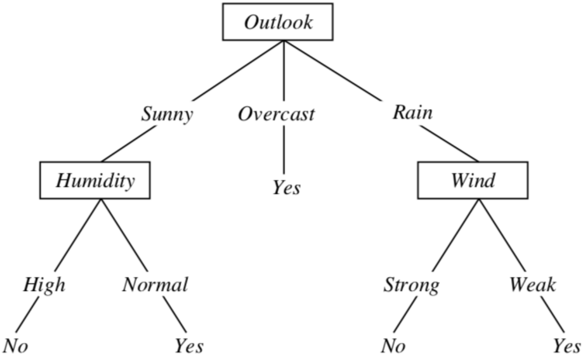
S_{sunny} = {D1,D2,D8,D9,D11}

$Gain\ (S_{sunny}\ ,\ Humidity) = .970 - (3/5)\ 0.0 - (2/5)\ 0.0 = .970$

$Gain\ (S_{sunny}\ ,\ Temperature) = .970 - (2/5)\ 0.0 - (2/5)\ 1.0 - (1/5)\ 0.0 = .570$

$Gain\ (S_{sunny}\ ,\ Wind) = .970 - (2/5)\ 1.0 - (3/5)\ .918 = .019$

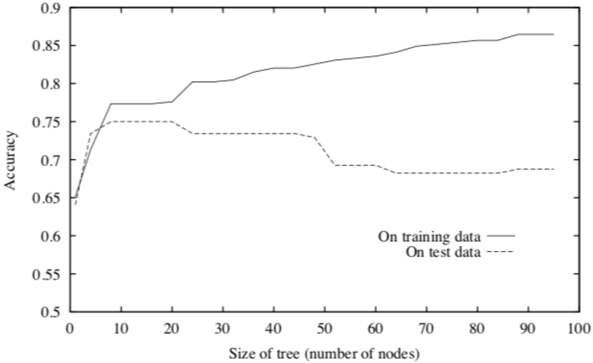
Example: Resulting decision tree



Inductive Bias

- Inductive bias: Shorter trees are preferred to longer trees. Trees that place high information gain attributes close to the root are also preferred.
- Why prefer shorter hypotheses?
  - Occam's Razor: Prefer the simplest hypothesis that fits the data!
  - e.g., if there are two decision trees, one with 500 nodes and another with 5 nodes, the second one should be preferred
  - → better chance to avoid overfitting

Overfitting (1/2)



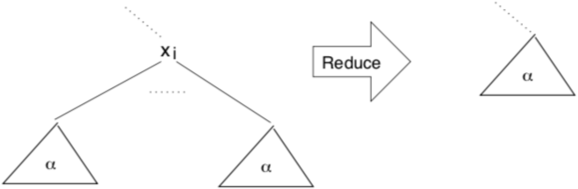
- Given a hypothesis space H, a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training, but h' has smaller error than h over the entire distribution of instances.

Overfitting (2/2)

- reasons for overfitting:
  - noise in the data
  - number of training examples is to small too produce a representative sample of the target function
- how to avoid overfitting:
  - stop the tree grow earlier, before it reaches the point where it perfectly classifies the training data
  - allow overfitting and then post-prune the tree (more successful in practice!)
- how to determine the perfect tree size:
  - separate validation set to evaluate utility of post-pruning
  - apply statistical test to estimate whether expanding (or pruning) produces an improvement

Reduced Error Pruning

- each of the decision nodes is considered to be candidate for pruning



- pruning a decision node consists of removing the subtree rooted at the node, making it a leaf node and assigning the most common classification of the training examples affiliated with that node
- nodes are removed only if the resulting tree performs not worse than the original tree over the validation set
- pruning starts wit the node whose removal most increases accuracy and continues until further pruning is harmful
- also other algorithms of pruning, e.g. Rule Post-Pruning

Random Forests

- Grow K trees on datasets sampled from the original dataset with replacement (bootstrap samples), p = number of features.
  - Draw K bootstrap samples of size N
  - Grow each Decision Tree, by selecting a random set of m out of p features at each node, and choosing the best feature to split on.
  - Aggregate the predictions of the trees (most popular vote) to produce the final class.
- Typically m might be e.g. $\sqrt{p}$ but can be smaller.

Random Forests

Principles: we want to take a vote between different learners so we don't want the models to be too similar. These two criteria ensure diversity in the individual trees:

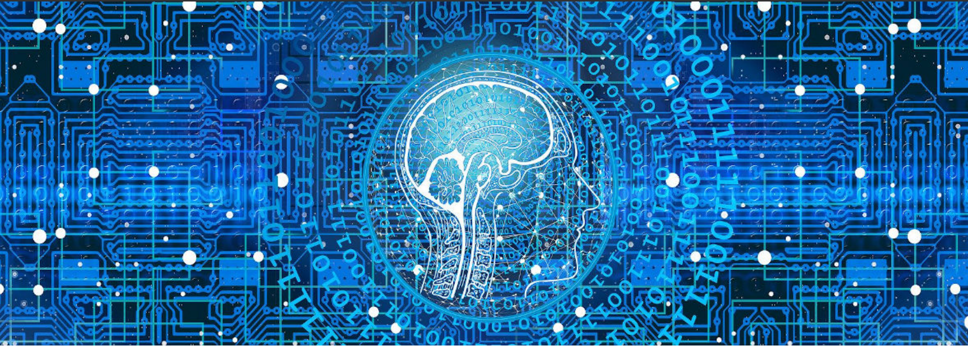
- Draw K bootstrap samples of size N:
- Each tree is trained on different data.
- Grow a Decision Tree, by selecting a random set of m out of p features at each node, and choosing the best feature to split on.
- Corresponding nodes in different trees (usually) cant use the same feature to split.

Random Forests

- Very popular in practice, probably the most popular classifier for dense data (← a few thousand features)
- Easy to implement (train a lot of trees). Good match for MapReduce.
- Parallelizes easily (but not necessarily efficiently).
- Not quite state-of-the-art accuracy
- DNNs generally do better, and sometimes gradient boosted trees.
- Needs many passes over the data - at least the max depth of the trees. (« boosted trees though)
- Easy to overfit - hard to balance accuracy/fit tradeoff.

Boosted Decision Trees

- A recently-developed alternative to random Forests:
- In contrast to RFs whose trees are trained independently, BDT trees are trained sequentially by boosting: Each tree is trained on weighted data which emphasizes incorrectly-labeled instances by the previous trees.
- Both methods can produce very high-quality models. Superiority of one method or the other is very dataset- dependent.
- Resource requirements are very different as well, so its actually non-trivial to compare the methods (what resources do you fix during the experiment?).



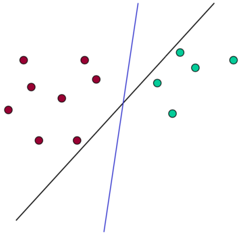
Support Vector Machine

References

(A) web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf

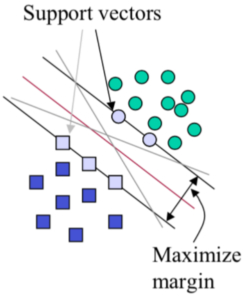
Introduction to Support Vector Machines

- Recall from 1-layer nets : Which Separating Hyperplane?



- In general, lots of possible solutions for a,b,c (an infinite number!)
- Support Vector Machine (SVM) finds an optimal solution

Introduction to Support Vector Machines



- SVMs maximize the margin (Winston terminology: the 'street') around the separating hyperplane.
- The decision function is fully specified by a (usually very small) subset of training samples, the support vectors.
- This becomes a Quadratic programming problem that is easy to solve by standard methods

Separation by Hyperplanes

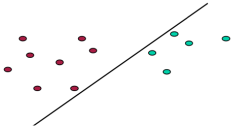
- Assume linear separability for now (we will relax this later)
- in 2 dimensions, can separate by a line - in higher dimensions, need hyperplanes

Training Concept of SVM

General input/output for SVMs just like for neural nets, but for one important addition...

- Input: set of (input, output) training pair samples; call the input sample features $x_1, x_2...x_n$, and the output result y. Typically, there can be lots of input features x_i.
- Output: set of weights w (or w_i), one for each feature, whose linear combination predicts the value of y.
    - (So far, just like neural nets...)
    - Important difference: we use the optimization of maximizing the margin ('street width') to reduce the number of weights that are nonzero to just a few that correspond to the important features that 'matter' in deciding the separating line(hyperplane)...these nonzero weights correspond to the support vectors (because they'support' the separating hyperplane)

2-D Case



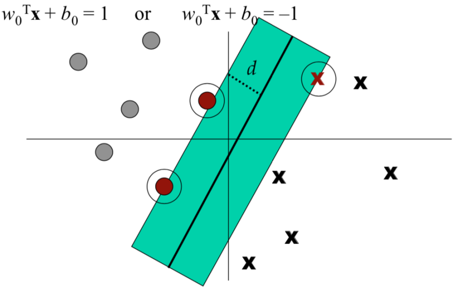
- Find a,b,c, such that
  - $ax + by \geq c$ for red points
  - $ax + by \leq (or <)c$ for green points.
- Which Hyperplane to pick?
  - Lots of possible solutions for a, b, c.
  - Some methods find a separating hyperplane, but not the optimal one (e.g., neural net)
  - But: Which points should influence optimality?
- All points?
  - Linear regression, Neural nets
  - Or only “difficult points” close to decision boundary: Support vector machines

Support Vectors again for linearly separable case

- Support vectors are the elements of the training set that would change the position of the dividing hyperplane if removed.
- Support vectors are the critical elements of the training set
- The problem of finding the optimal hyper plane is an optimization problem and can be solved by optimization techniques (we use Lagrange multipliers to get this problem into a form that can be solved analytically).

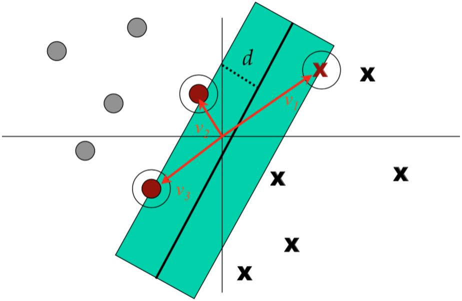
SVM: Explanation (1/2)

- Support Vectors: Input vectors that just touch the boundary of the margin (street)
  - circled below, there are 3 of them (or, rather, the 'tips' of the vectors



SVM: Explanation (2/2)

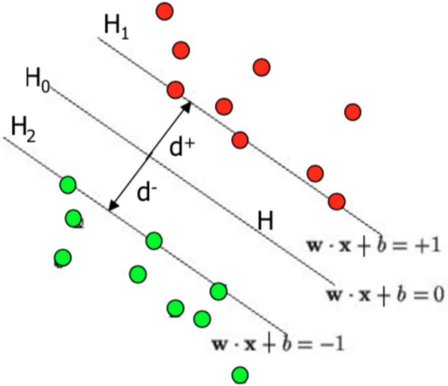
- Here, we have shown the actual support vectors, v_1, v_2, v_3, instead of just the 3 circled points at the tail ends of the support vectors. d denotes 1/2 of the street-'width'



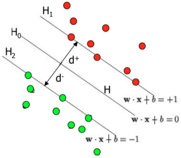
Definitions (1/2)

Define the hyperplanes H such that:

- $w \cdot x_i + b \geq +1$ when $y_i = +1$
- $w \cdot x_i + b \leq -1$ when $y_i = -1$


H$_1$
H$_0$
H$_2$
d$^+$
d$^-$
H
$\mathbf{w} \cdot \mathbf{x} + b = +1$
$\mathbf{w} \cdot \mathbf{x} + b = 0$
$\mathbf{w} \cdot \mathbf{x} + b = -1$

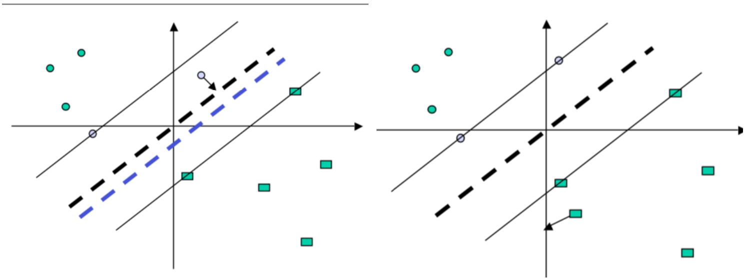
Definitions (2/2)



- H_1 and H_2 are the planes:
  - H_1: $w \cdot x_i + b = +1$
  - H_2: $w \cdot x_i + b = -1$
  - The points on the planes H_1 and H_2 are the tips of the Support Vectors
  - The plane H_0 is the median in between, where $w \cdot x_i + b = 0$
  - d_+ = the shortest distance to the closest positive point
  - d_- = the shortest distance to the closest negative point
  - The margin (gutter) of a separating hyperplane is $d_+ + d_-$.

Support Vectors and other Vectors

- The optimization algorithm to generate the weights proceeds in such a way that only the support vectors determine the weights and thus the boundary
  - Moving a support vector moves the decision boundary
  - Moving the other vectors has no effect



Defining the separating Hyperplane

- Form of equation defining the decision surface separating the classes is a hyperplane of the form:

$$w^T x + b = 0$$

  - w is a weight vector
  - x is input vector
  - b is bias
- Allows us to write
  - $w^T x + b \geq 0$ for $d_i = +1$
  - $w^T x + b < 0$ for $d_i = -1$
- Margin of Separation (d): the separation between the hyperplane and the closest data point for a given weight vector w and bias b.
- Optimal Hyperplane (maximal margin): the particular hyperplane for which the margin of separation d is maximized.

What do we want to achieve?

Maximizing the margin (aka street width)

- We want a classifier (linear separator) with as big a margin as possible.
- Recall the distance from a point (x_0, y_0) to a line: $Ax + By + c = 0$ is:
  - $|Ax_0 + By_0 + c|/\sqrt{A_2 + B_2}$, so, The distance between H_0 and H_1 is then:

$$|w \cdot x + b|/||w|| = 1/||w||$$

- The total distance between H_1 and H_2 is thus: $2/||w||$
- In order to maximize the margin, we thus need to minimize $||w||$. With the condition that there are no datapoints between H_1 and H_2:

$$\vec{x}_i \cdot \vec{w} + b \geq +1$$

when $y_i = +1$

$$\vec{x}_i \cdot \vec{w} + b \leq -1$$

when $y_i = -1$

- Can be combined into: $y_i.(\vec{x}_i \cdot \vec{w}) \geq 1$

SVM Training (1/3)

We now must solve a quadratic programming problem

- Problem is: minimize $||w||$, under the condition that discrimination boundary is obeyed, i.e., $minf(x)$ s.t. $g(x) = 0$, which we can rewrite as:
  - $min_f : 1/2||w||^2$ (Note this is a quadratic function) under the condition that g: $y_i(\vec{w} \cdot \vec{x}_i) - \vec{b} = 1$ or $[y_i(w \cdot x_i) - b] - 1 = 0$
- This is a constrained optimization problem
- It can be solved by the Lagrangian multipler method
- Because it is quadratic, the surface is a paraboloid, with just a single global minimum (thus avoiding a problem we had with neural nets!)

SVM Training (2/3)

How to do this?
We can reformat the problem. We have two constraints

- 1. Parallel normal constraint (= gradient constraint on f, g s.t. solution is a max, or a min)
- 2. g(x)=0 (solution is on the constraint line as well)

We now recast these by combining f, g as the new Lagrangian function by introducing new'slack variables' denoted

- $L(x, a) = f(x) - ag(x)$
- $\nabla L(x, a) = 0$

SVM Training (3/3)

General case

- The general Lagrangian is

$$L(x, a) = f(x) + \sum_i a_i g_i(x)$$

- a function of n + m variables. n for the x's, m for the a. Differentiating gives n + m equations, each set to 0. The n equations differentiated wrt each x_i give the gradient conditions; the m equations differentiated wrt each a_i recover the constraints g_i
- so in the SVM problem, the problem can be formulated in the "Lagrangian-way" is

$$minL_p = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{l} a_i y_i(\vec{x}_i \cdot \vec{w} + b) + \sum_{i=1}^{l} a_i$$

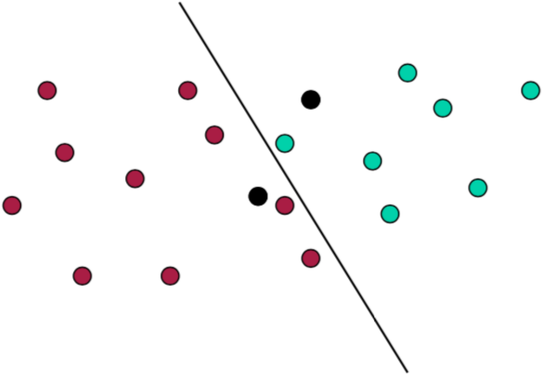
- with the condition that for all i, $a_i \geq 0$, where l is the number of training points-
- We deriving w.r.t. to w and b, we get our initial conditions back

Similarity and the inner product

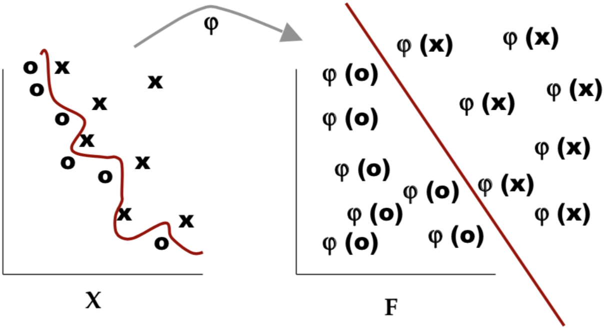
- Intuition is that inner products provide some measure of 'similarity'
- Inner product in 2D between 2 vectors of unit length returns the cosine of the angle between them = how'far apart' they are e.g. $x = [1, 0]^T$ and $y = [0, 1]^T$
- if they are parallel their inner product is 1 (completely similar) $x^T y = x \cdot y = 1$
- if they are perpendicular (completely unlike) their inner product is 0 (so should not contribute to the correct classifier) $x^T y = x \cdot y = 0$

SVM in non linear separable problems

- Find a line that penalizes points on “the wrong side”

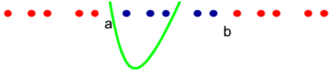


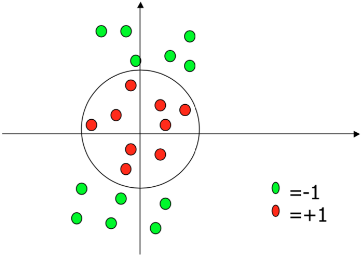
Transformation to separate



Non-Linear SVMs (1/2)

- The idea is to gain linearly separation by mapping the data to a higher dimensional space
- The following set can't be separated by a linear function, but can be separated by a quadratic one

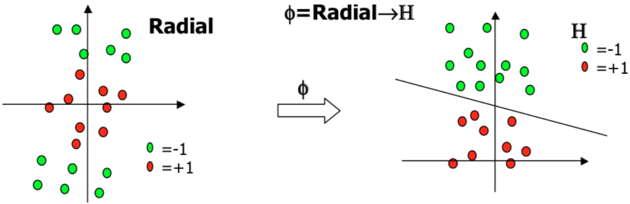




- So if we map $x \rightarrow (x^2, x)$ we gain linear separation

Non-Linear SVMs (2/2)

- What if the decision function is not linear? What transform would separate these?



Training of non-linear SVM

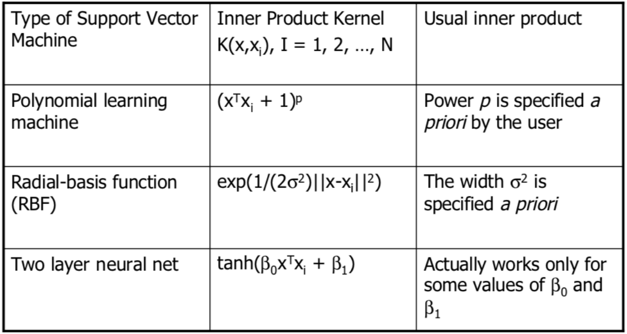
We want to optimize:

$$L_d = \sum_i a_i - \frac{1}{2} \sum a_i a_j y_i y_j (\vec{x}_i \cdot \vec{x}_j)$$

- where $(\vec{x}_i \cdot verx_j)$ is the dot product
- the cool thing is: we dont even have to compute the dot-product of the transformed variables - it is enough to define a kernel function, such that $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$

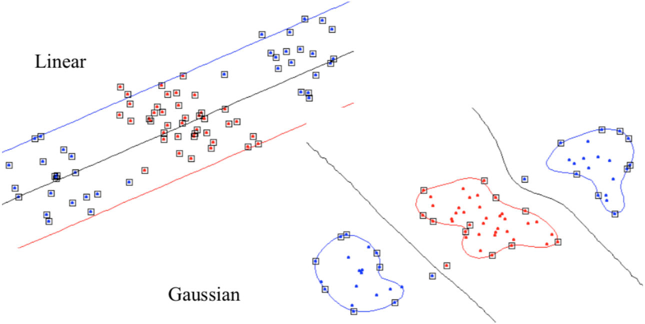
Kernel Functions

- Some examples of popular inner product Kernel Functions



Type of Support Vector Machine	Inner Product Kernel $K(x,x_i)$, $I = 1, 2, \dots, N$	Usual inner product
Polynomial learning machine	$(x^Tx_i + 1)^p$	Power *p* is specified *a priori* by the user
Radial-basis function (RBF)	$\exp(1/(2\sigma^2)\|\|x-x_i\|\|^2)$	The width σ^2 is specified *a priori*
Two layer neural net	$\tanh(\beta_0 x^T x_i + \beta_1)$	Actually works only for some values of β_0 and β_1

Examples for Non Linear SVMs 2 - Gaussian Kernel



Advantages and Disadvantages of Support Vector Machones

Advantages

It works really well with clear margin of Separation

It is effective in high dimensional spaces

It is effective in cases where number of dimensions is greater than the number of samples (i.e. works good when you dont have lots of data)

very sparse features

It uses a subset of training Points in the decision function (the support vectors), so it is memory efficient

Disadvantage:

usually only two classes (but can be extended in principle)

it doesnt perform well, when we have large data sets because the training time is much higher

it doesnt perform will, when the data has more noise, i.e. target classes are