Lecture notes

Knowledge Discovery in Databases

Summer Semester 2012

Lecture 6: Classification III

Lecture: Dr. Eirini Ntoutsi
Tutorials: Erich Schubert

http://www.dbs.ifi.lmu.de/cms/Knowledge_Discovery_in_Databases_I_(KDD_I)

Sources

- Previous KDD I lectures on LMU (Johannes Aßfalg, Christian Böhm, Karsten Borgwardt, Martin Ester, Eshref Januzaj, Karin Kailing, Peer Kröger, Jörg Sander, Matthias Schubert, Arthur Zimek)
- Jiawei Han, Micheline Kamber and Jian Pei, *Data Mining: Concepts and Techniques, 3rd ed.,* Morgan Kaufmann, 2011.
- Tan P.-N., Steinbach M., Kumar V., *Introduction to Data Mining*, Addison-Wesley, 2006
- Boosting tutorial by Robert Schapire, Machine Learning Summer School (MLSS), Chicago 2005 (http://videolectures.net/mlss05us_schapire_b/)
- Support Vector and Kernel Machines, Nello Cristianini, http://www.support-vector.net/icml-tutorial.pdf

Outline

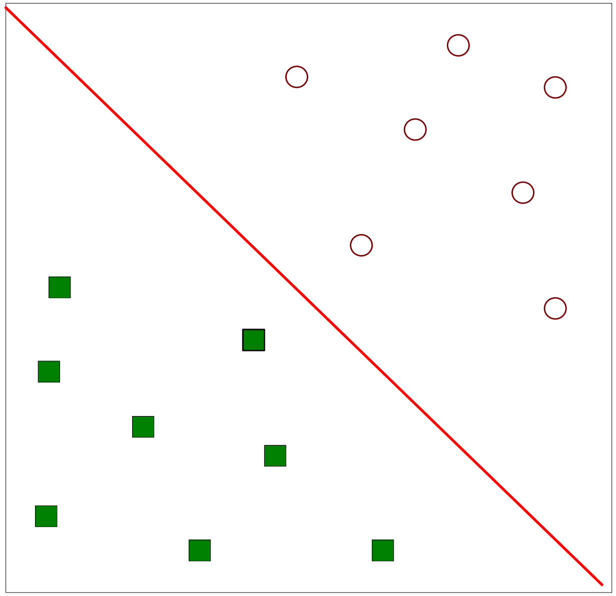
- Introduction
- Support Vector Machines
- Ensembles of classifiers
- An overview of classification
- Things you should know
- Homework/tutorial

Support Vector Machines (SVM)

- A popular classification method
- Its roots are in statistical learning theory
- Promising results in many applications, e.g., handwritten text classification, text categorization
- The decision boundary is represented using a subset of the training examples, support vectors

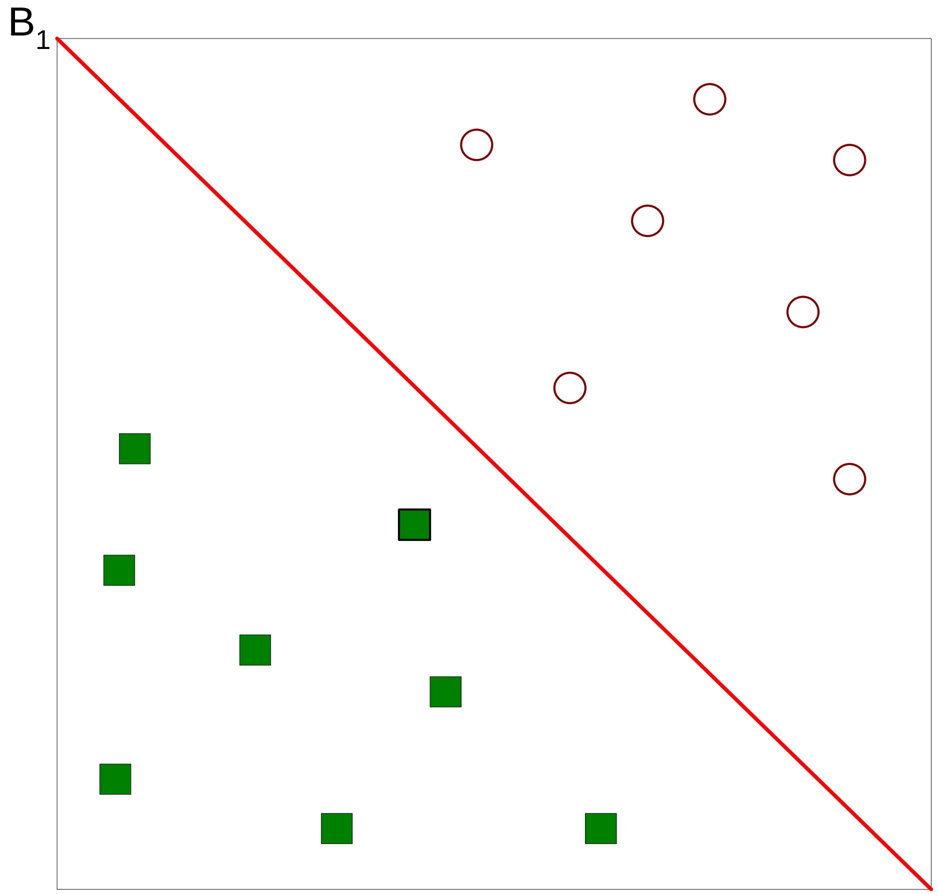
Basic idea

Lets start with a simple 2 class problem



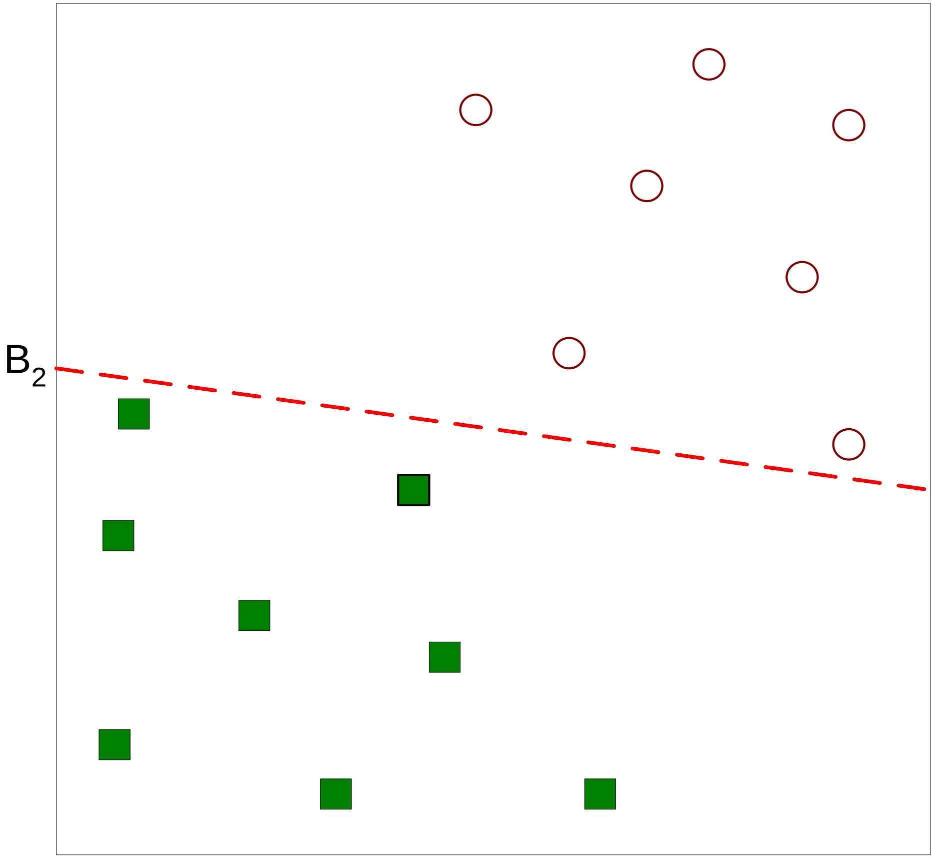
- Goal: find a hyperplane (decision boundary) that will separate the data based on their class
  - In 2D this is just a straight line

Finding a hyperplane I



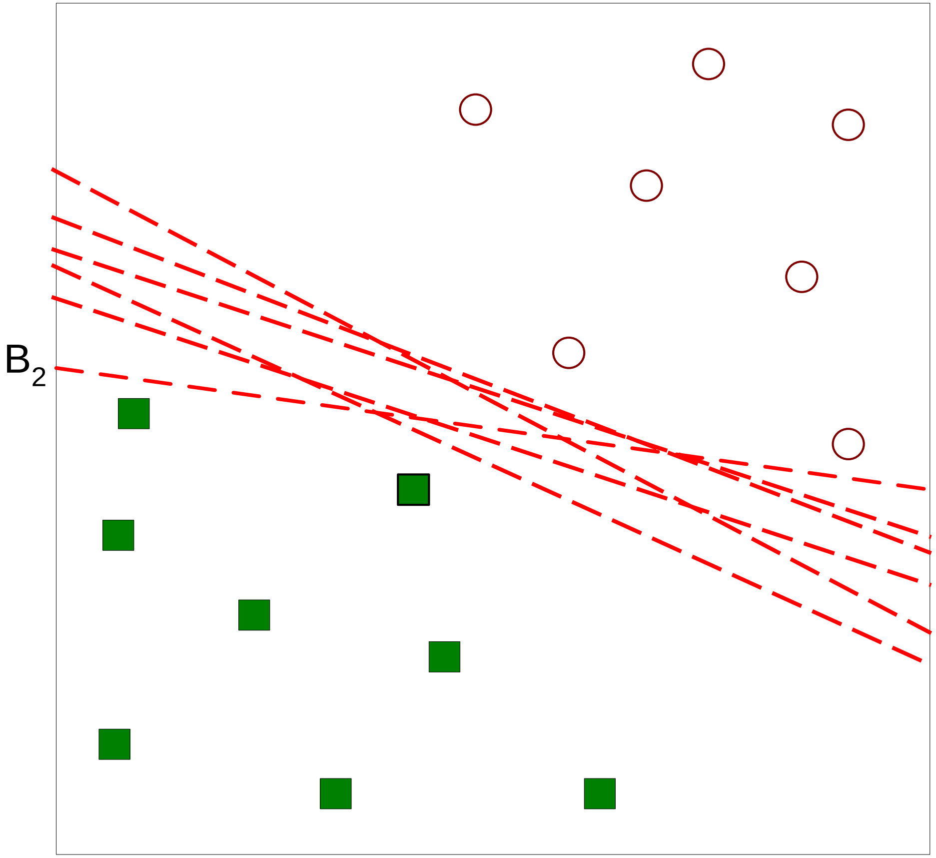
One possible solution

Finding a hyperplane II



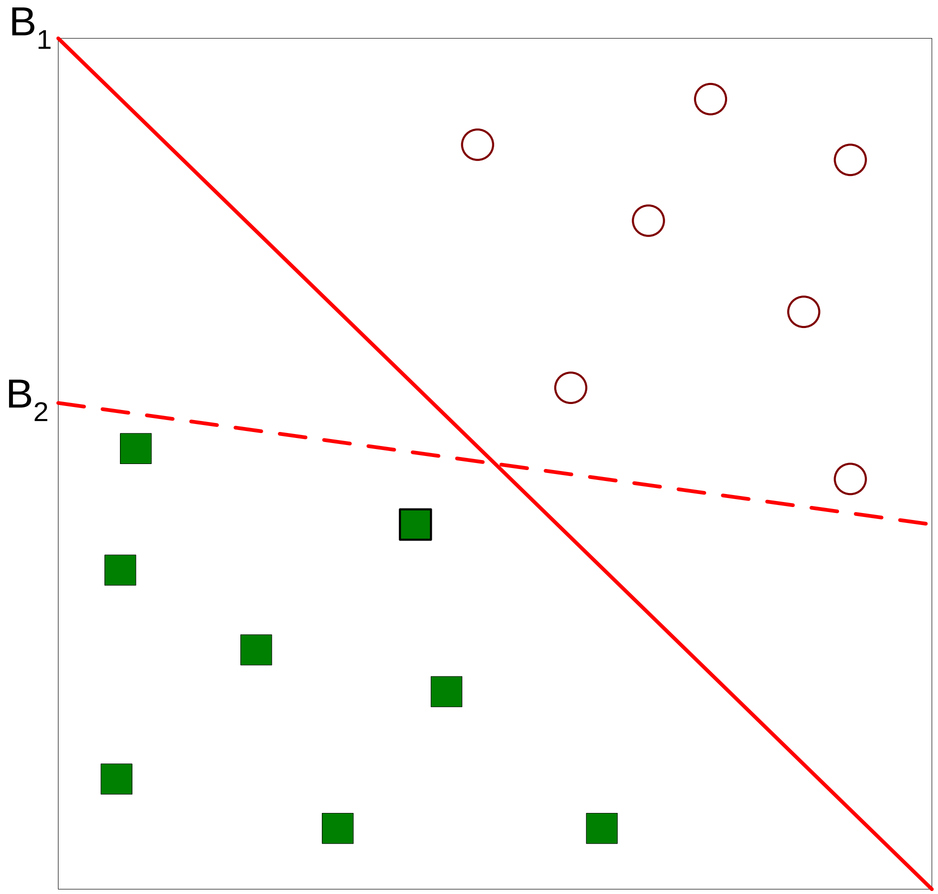
Another possible solution

Finding a hyperplane III



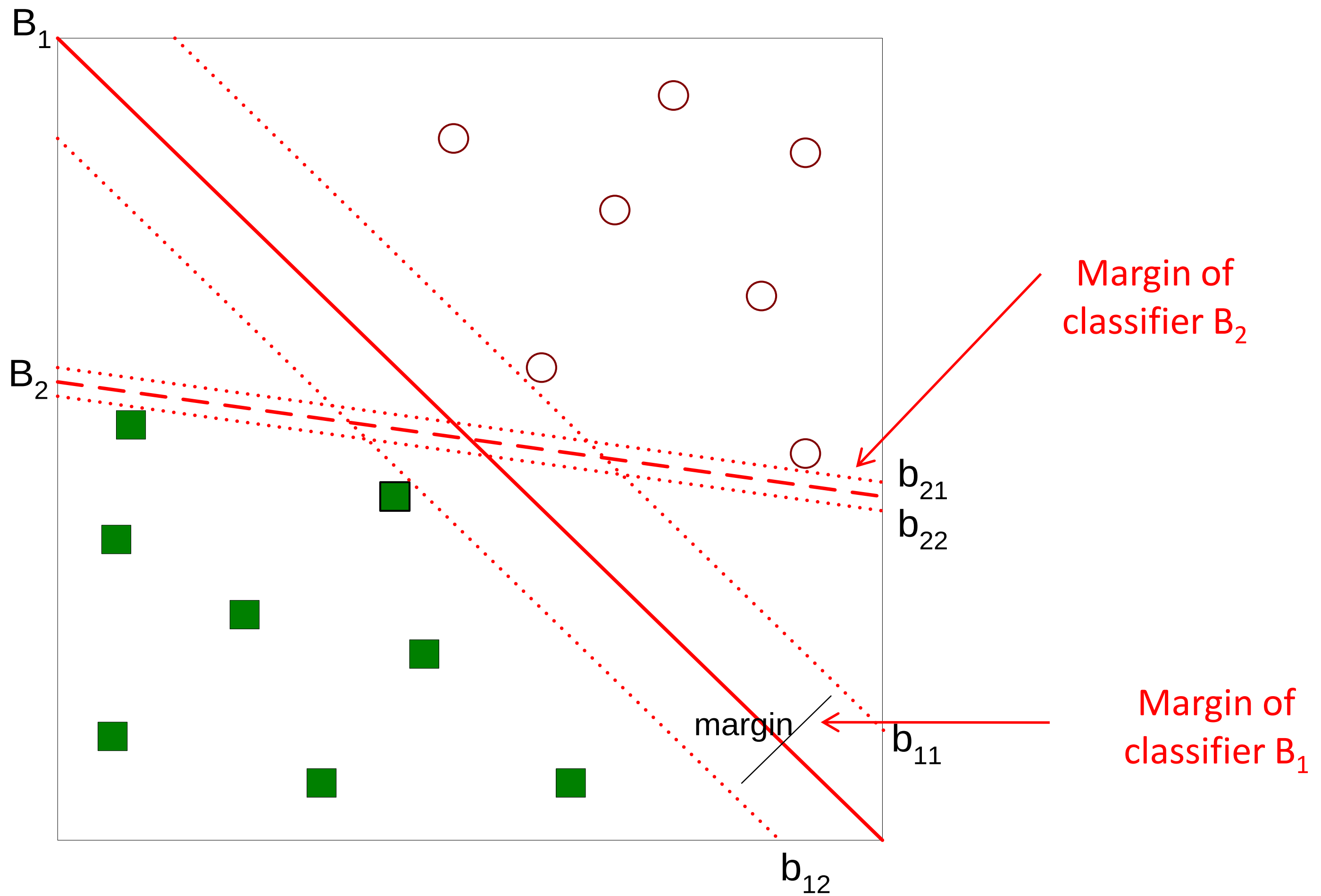
Other possible solutions

Choosing a hyperplane I



- Which hyperplane is better?
- How do you define better?

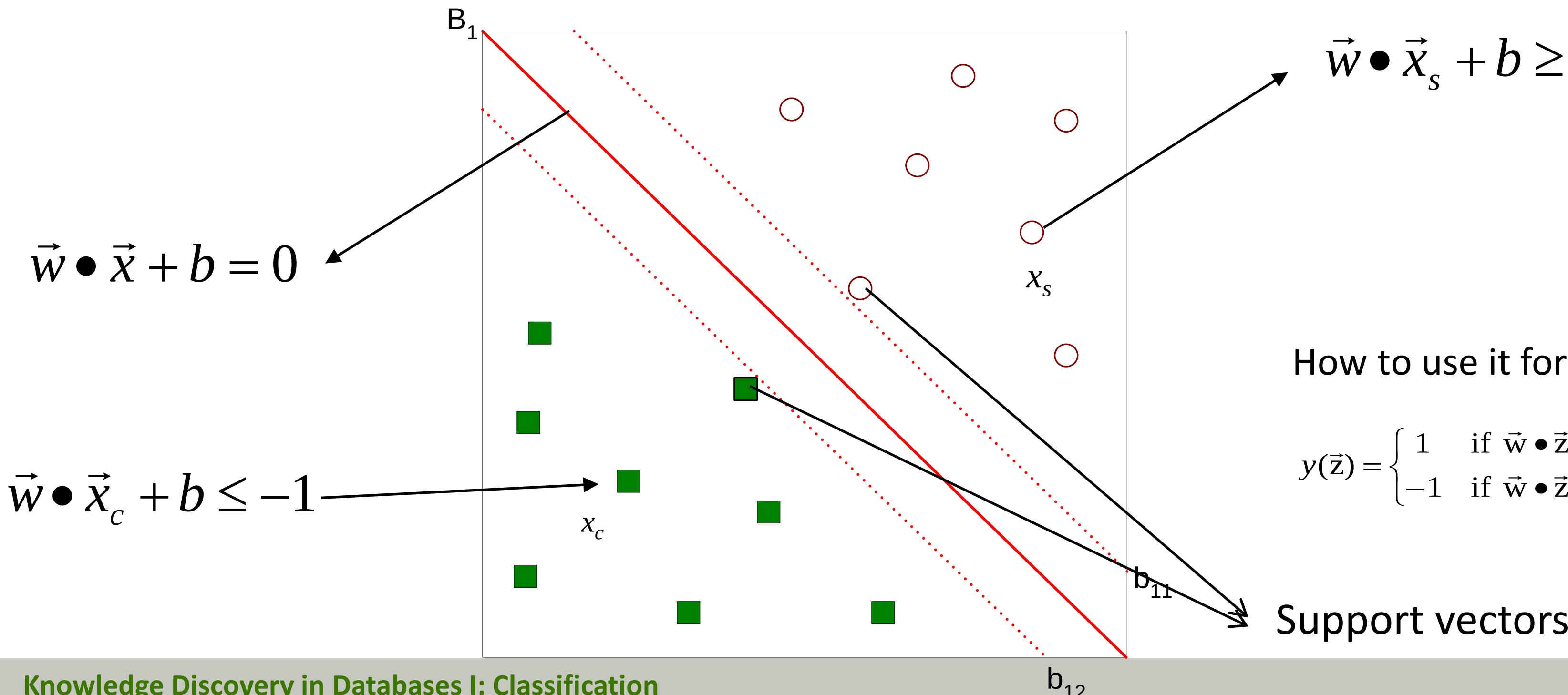
Choosing a hyperplane II



Find hyperplane that maximizes the margin => B_1 is better than B_2

Linear SVM I

- A linear SVM searches for a hyperplane that maximizes the margin (maximal margin classifier)
- Consider a simple 2 class problem. Let D=(x_i) and y_i={-1,1}
- We can represent a linear classifier by: $\vec{w} \bullet \vec{x} + b = 0$
  - w is a weight vector and b a scalar (bias)



How to use it for prediction?

$$y(\vec{z}) = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{z} + b \geq +1 \\ -1 & \text{if } \vec{w} \bullet \vec{z} + b \leq -1 \end{cases}$$

Linear SVM II

- The margin of B_1 is given by the distance between the two hyperplanes b_{11}, b_{12}.
- Let x_1, x_2 be two points in b_{11}, b_{12} respectively.

$$\vec{w} \bullet \vec{x}_1 + b = +1$$

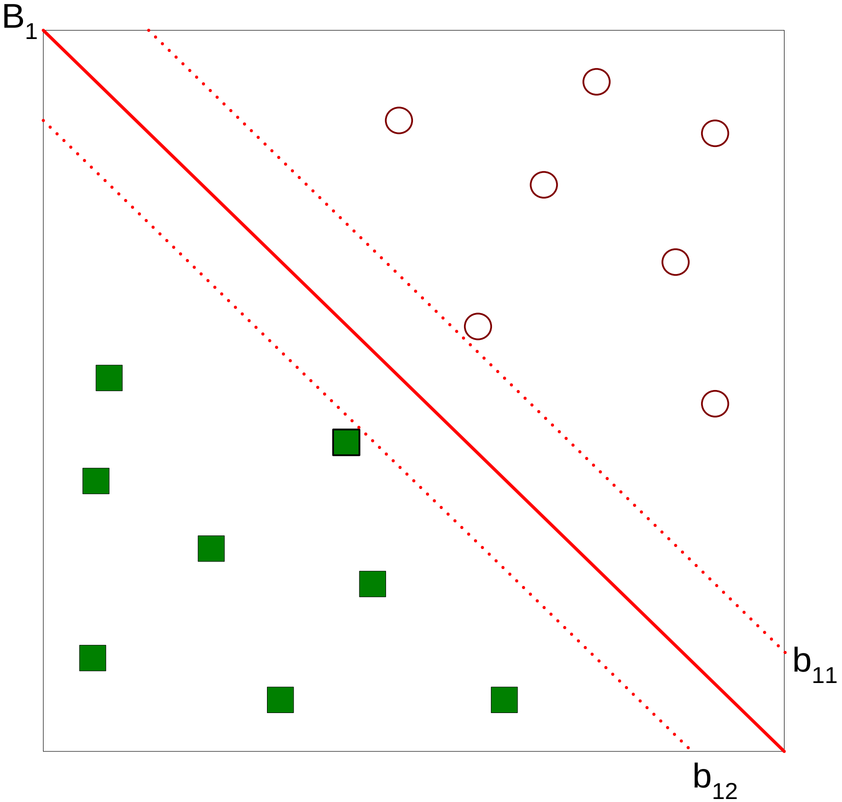
$$\vec{w} \bullet \vec{x}_2 + b = -1$$

$$\vec{w} \bullet (\vec{x}_1 - \vec{x}_2) = 2$$

$$\Rightarrow \text{margin d} = \frac{2}{\| \vec{w} \|}$$

- We want to maximize this margin

$$\| \vec{w} \| = \sqrt{\vec{w} \bullet \vec{w}}$$



Linear SVM III

- We want to maximize $d = \dfrac{2}{\|\vec{w}\|}$
  - This is equivalent to minimizing the following objective function:

$$\min_{w} \frac{\|\vec{w}\|}{2} \Leftrightarrow \min_{w} \frac{\|\vec{w}\|^2}{2}$$

This allows us to perform quadratic programming optimization latter on

  - but, subject to the following constraints

$$y_i = \begin{cases} 1 & \text{if } \vec{\mathrm{w}} \bullet \vec{\mathrm{x}}_{\mathrm{i}} + \mathrm{b} \geq 1 \\ -1 & \text{if } \vec{\mathrm{w}} \bullet \vec{\mathrm{x}}_{\mathrm{i}} + \mathrm{b} \leq -1 \end{cases} \Bigg\} \quad y_i(\vec{\mathrm{w}} \bullet \vec{\mathrm{x}}_{\mathrm{i}} + \mathrm{b}) \geq 1$$

Linear SVM IV

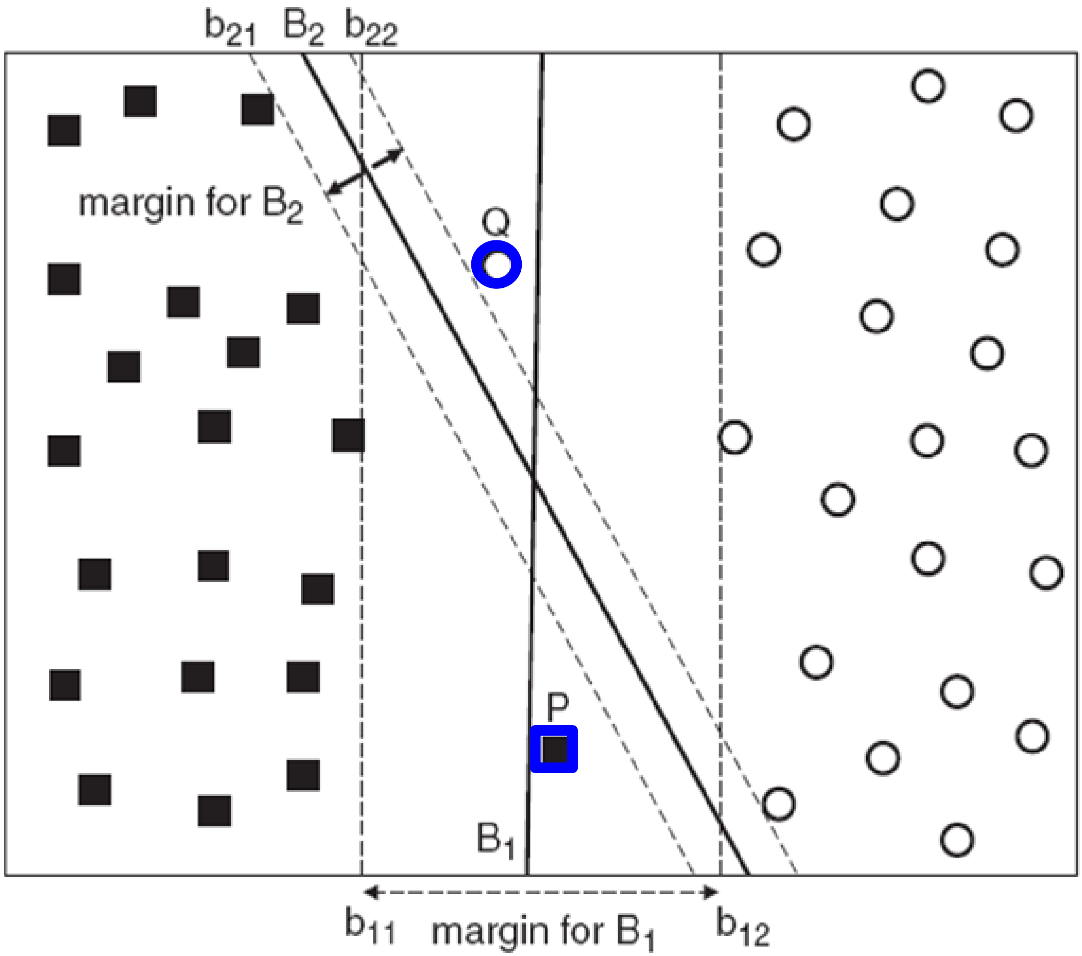
- This is a constrained quadratic optimization problem
  - The constraints are rewritten using a Lagrangian formulation
- The solution (trained SVM) consists of
  - The support vectors
  - The parameters w, b of the decision boundary
- How can I classify a new instance?

$$y(z) = sign(wz + b) = sign\left(\sum_{i=1}^{N} \lambda_i y_i x_i z + b \right)$$

  - λ_i: Lagrange multipliers
  - x_i: is the support vector
  - y_i: is the class of x_i

Linear SVM: nonseparable cases

What if the problem is not linearly separable?



- B_1 should be preferred over B_2
  - it has a wider margin → less susceptible to overfitting
- but, the so far SVM formulation is error free
  - > Soft margin approach

Soft margin approach I

- Learn a decision boundary that is tolerable to small training errors
- Allows SVM to construct a decision boundary even in cases where the classes are not linearly separable
- **Idea:** trade-off between the width of the margin and the misclassification errors committed by the linear decision boundary

Original optimization problem

$$\min \frac{\|\vec{w}\|^2}{2}$$

subject to :

$$y_i(\vec{w} \bullet \vec{x}_i + b) \geq 1$$

violated

Idea:

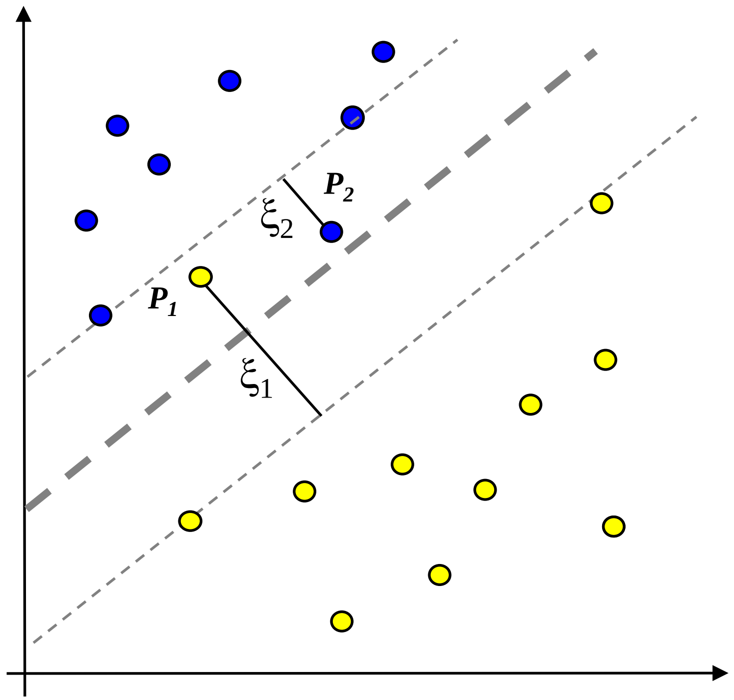
- Relax the constraints to accommodate nonlinearly separable data
- Introduce positive-valued slack variables ξ_i

Soft margin approach II

- Relaxing by introducing slack variables ξ_i, $\xi_i \geq 0$

$$\cancel{y_i = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x}_i + b \geq 1 \\ -1 & \text{if } \vec{w} \bullet \vec{x}_i + b \leq -1 \end{cases}} \Rightarrow y_i = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x}_i + b \geq +1 - \xi_i \\ -1 & \text{if } \vec{w} \bullet \vec{x}_i + b \leq -1 + \xi_i \end{cases}$$

  - The slack variable ξ_i measures the degree of missclassification of instance x_i
  - Intuitively, data points on the incorrect side of the margin boundary have a penalty that increases with the distance from it.



Soft margin approach III

If no constrains on # mistakes, we might end up with a very wide margin with many misclassification errors

Updated definition

- Need to minimize: $\frac{\|\vec{w}\|^2}{2} + C\left(\sum_{i=1}^{N} \xi_i^k\right)$

- Subject to the following constraints:

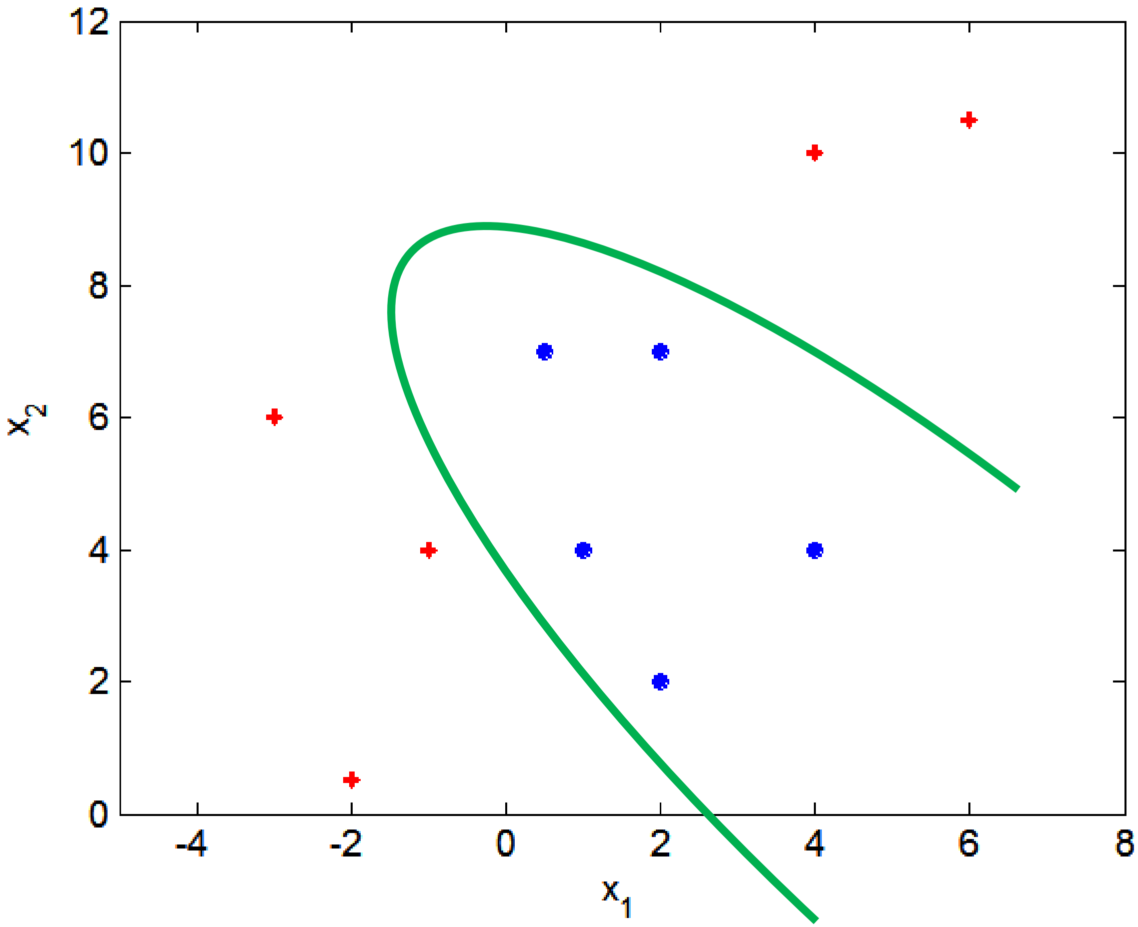
$$y_i = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x}_i + b \geq +1 - \xi_i \\ -1 & \text{if } \vec{w} \bullet \vec{x}_i + b \leq -1 + \xi_i \end{cases}$$

C, k are user-specified parameters representing the penalty of missclassifying the training instances

- Can be solved used quadratic programming
  - This way we can learn the parameters w, b of the decision boundary

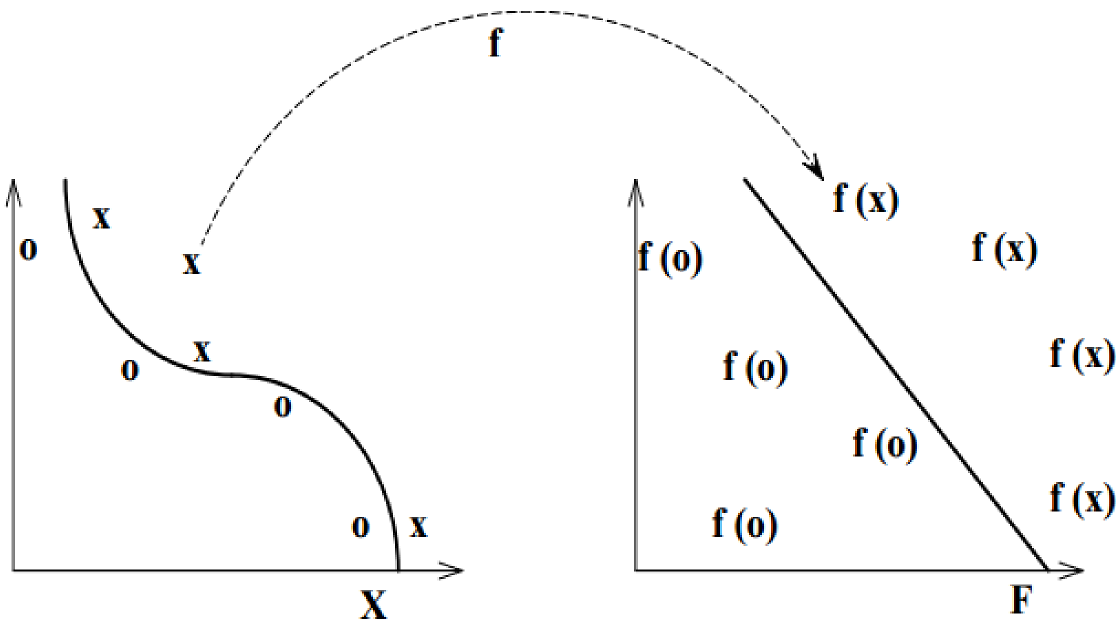
Nonlinear SVM

What if the decision boundary is not linear?



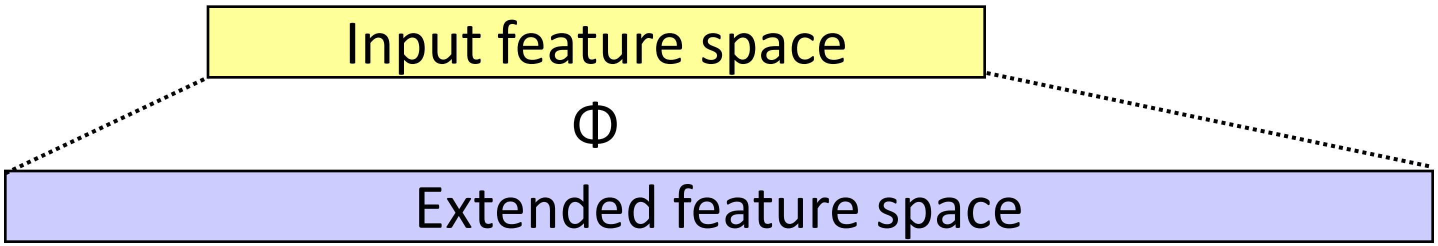
Idea I

- Trick: transform the data from its original space x into a new space Φ(x) so that a linear decision boundary can be used to separate the instances in the transformed space
- In Φ(x), we can apply the same methodology as before to find a linear decision boundary

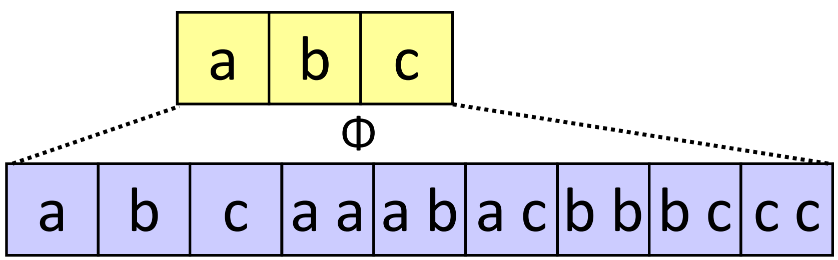


Idea II

- Intuitively, we extend the hypothesis space



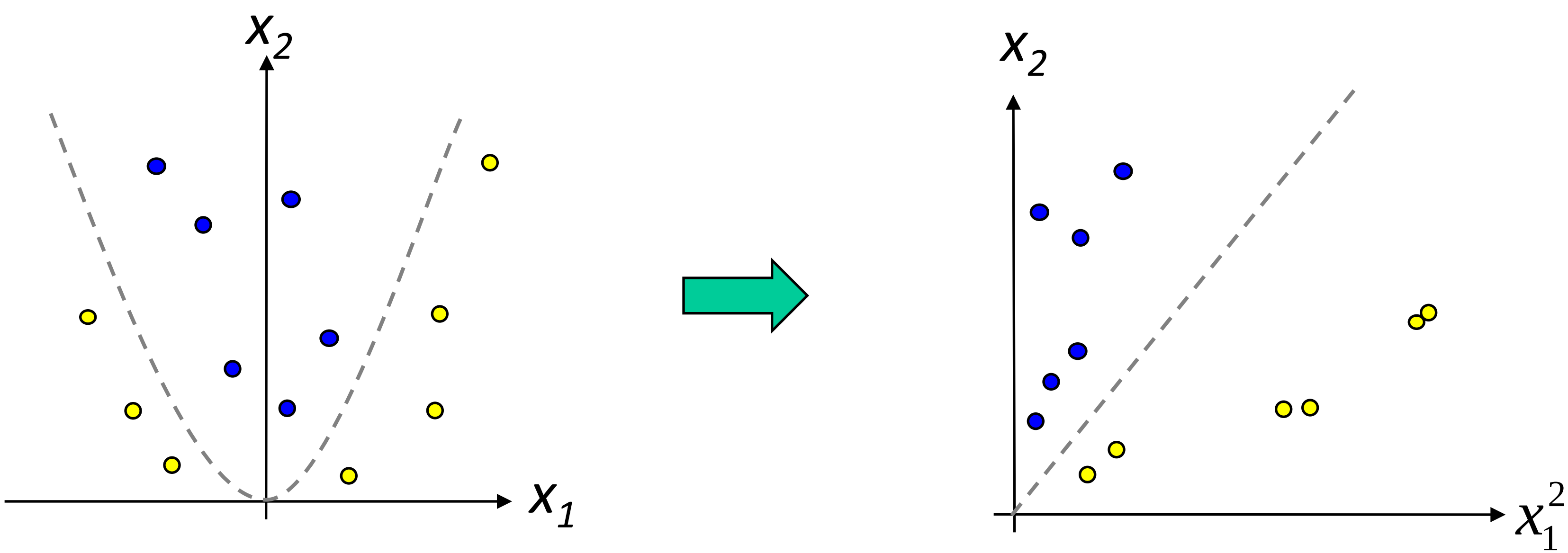
- e.g.,



Example I

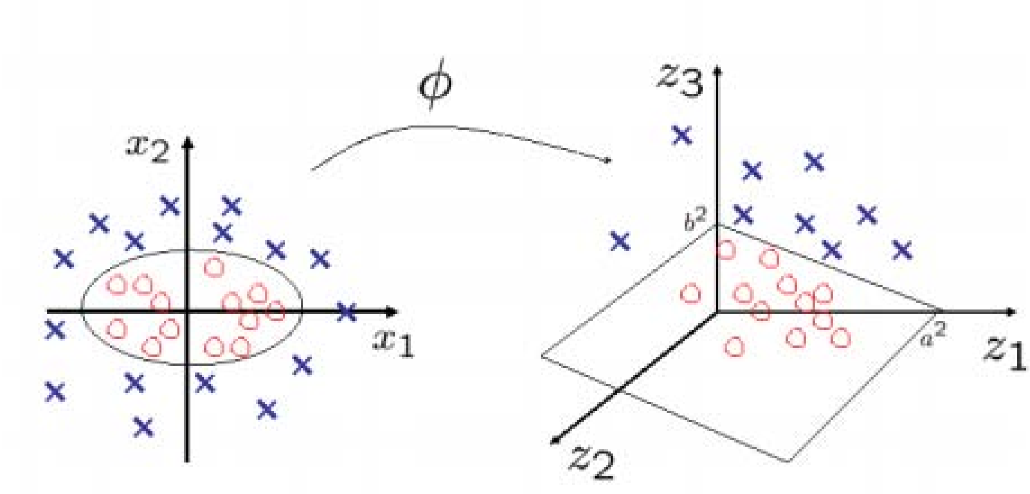
Input space: $\vec{x} = (x_1, x_2)$ (2 Attribute)

Extended space (6 Attributes) $\phi(\vec{x}) = \left(x_1^2, x_2^2, \sqrt{2} \cdot x_1, \sqrt{2} \cdot x_2, \sqrt{2} \cdot x_1 \cdot x_2, 1\right)$



Example II

Elliptical boundary in the input space becomes linear in the transformed space



$$\phi : [x_1, x_2]^T \rightarrow [x_1^2, \sqrt{2}x_1x_2, x_2^2]^T$$

Nonlinear SVM definition

Updated definition

- Need to minimize: $\min_{w} \frac{\|\vec{w}\|^2}{2}$
- Subject to the following constraints:

$$y_i(\vec{\mathrm{w}} \bullet \Phi(\vec{\mathrm{x}}_{\mathrm{i}}) + \mathrm{b}) \geq 1$$

- Can be solved used quadratic programming
  - This way we can learn the parameters w, b of the decision boundary

- Classifying a new instance z (through the transformed space)

$$f(z) = sign(\mathrm{w} \bullet \Phi(\mathrm{z}) + \mathrm{b}) = \mathrm{sign}(\sum_{\mathrm{i}=1}^{\mathrm{N}} \lambda_{\mathrm{i}} \mathrm{y}_{\mathrm{i}} \Phi(\mathrm{x}_{\mathrm{i}})\Phi(\mathrm{z}) + \beta)$$

Involves calculating of the dot product in the transformed space.

- computational problem (very large vectors)

- curse of dimensionality

Kernel trick

The kernel trick is a method for computing similarity between two instances in the transformed feature space using the original attribute set.

- e.g., consider the mapping: $\Phi : (x_1, x_2) \rightarrow \left(x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1\right)$
- The dot product between 2 input vectors u, v in the transformed space is:

$$\Phi(u)\Phi(v) = \left(u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, 1\right) * \left(v_1^2, v_2^2, \sqrt{2}v_1, \sqrt{2}v_2, 1\right)$$
$$= u_1^2 v_1^2 + u_2^2 v_2^2 + 2u_1 v_1 + 2u_2 v_2 + 1$$
$$= (uv + 1)^2$$

- So, we can express the dot product in Φ(x) in terms of a similarity function in the original feature space

kernel function

$$K(u,v) = \Phi(u)\Phi(v) = (uv + 1)^2$$

A function that returns the dot product between the images of two vectors

Kernels

The main requirement for a kernel function in nonlinear SVM:

There must exist a transformation such that the kernel function computed for two vectors is equivalent to the dot product between these vectors in the transformed space.

Mercer's Theorem:

A kernel function K can be expressed as:

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

if and only if, for any function g(x) such that $\int g(x)^2 dx$ is finite, then

$$\int K(x,y)g(x)g(y)dxdy \geq 0$$

These functions are called positive definite kernel functions

Kernel functions

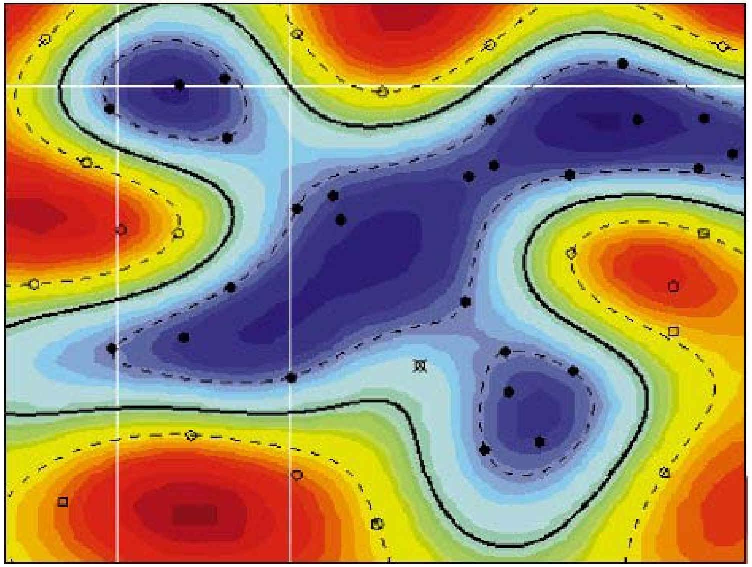
Popular kernel functions:

- Linear $K(\vec{x},\vec{y}) = \langle \vec{x},\vec{y} \rangle$
- Polynomial $K(\vec{x},\vec{y}) = \left(\langle \vec{x},\vec{y} \rangle + c \right)^d$
- Gaussian kernel $K(\vec{x},\vec{y}) = \exp\left(-\frac{\left\| \vec{x} - \vec{y} \right\|^2}{2\sigma^2} \right)$
- Radial basis function kernel $K(\vec{x},\vec{y}) = \exp\left(-\gamma \cdot \left| \vec{x} - \vec{y} \right|^2 \right)$

Choosing the right kernel depends on the problem at hand

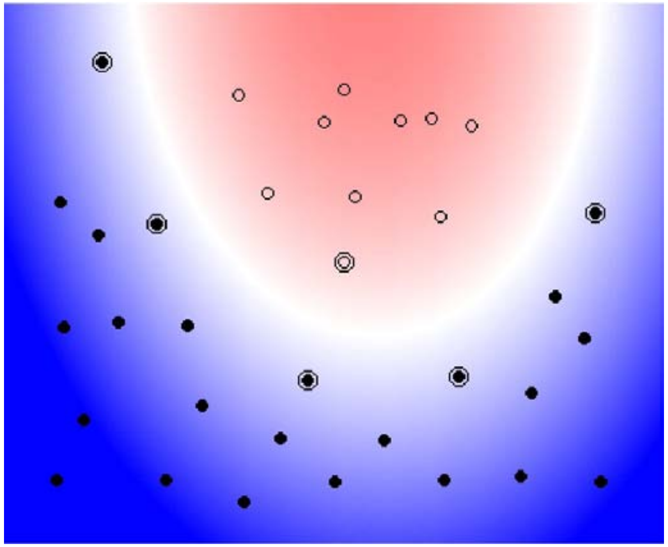
- a linear kernel allows us to model hyperplanes / a polynomial kernel allows us to model feature conjunctions / radial basis functions allows us to model hyperspheres
- Parameter settings is also important!

Kernel Machines



Radial Basis Kernel

Polynomial kernel (degree 2)



SVM: overview

- \+ High accuracy classifiers
- \+ Relatively weak tendency to overfitting
- \+ Efficient classification of new objects
- \+ Compact models

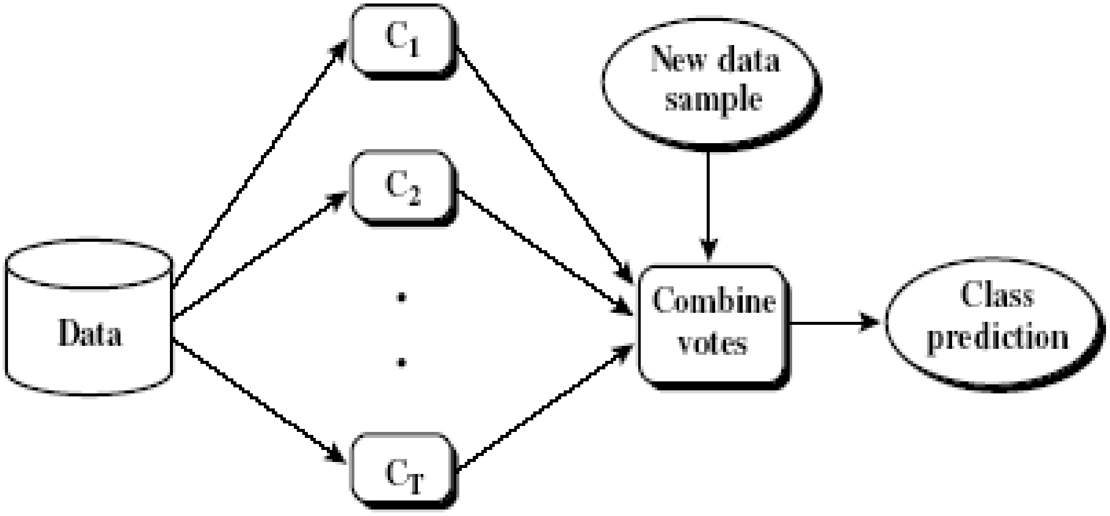
- – Costly implementation
- – sometimes long training times
- – found models difficult to interpret

Outline

- Introduction
- Support Vector Machines
- Ensembles of classifiers
- An overview of classification
- Things you should know
- Homework/tutorial

Ensemble of classifiers

- Idea:
  - Instead of a single model, use a combination of models to increase accuracy
  - Combine a series of T learned models, M_1, M_2, …, M_T, with the aim of creating an improved model M*
  - To predict the class of previously unseen records, aggregate the predictions of the ensemble



How to generate ensembles of classifiers I

- By manipulating the training set
  - Multiple training sets are created by resampling the original training data
  - A classifier is built from each training set using some learning algorithm
  - e.g., bagging, boosting

- By manipulating the input features
  - A subset of features is chosen to form each training set (randomly or by domain experts)
  - A base classifier is built from each training set using some learning algorithm
  - e.g., random forests

How to generate ensembles of classifiers II

- By manipulating the class labels
  - Transform into a binary classification problem by randomly partitioning the class labels in two disjoint subsets A_0, A_1. Training examples who belong to A_0 are assigned to class 0, the rest to class 1.
  - The relabeled examples are used to train a base classifier.
  - Repeat the class-relabeling and model-building steps multiple times to derive the ensemble
  - During testing, if the test instance is predicted as class 0 (1), all classes in A_0 (A_1) will receive a vote
- By manipulating the learning algorithm
  - Many learning algorithms can be manipulated such that applying the same algorithm in the same data might result in different models
  - e.g., insert randomness in the tree-growing process
    - e.g., instead of choosing the best splitting attribute choose randomly

Bagging/ Bootstrap aggregation (Breiman, 1996)

- Analogy: Diagnosis based on multiple doctors' majority vote
- Training: Given a training set D of d tuples
  - In each iteration i: i=1, … , T
    - Randomly sample with replacement from D a training set D_i of *d* tuples (i.e., boostrap)
      - On avg, the bootstrap sample contains approximately 63% of the original D
    - Train a chosen "base model" M_i (e.g. neural network, decision tree) on the sample D_i
- Testing
  - For each test example
    - Get the predicted class from each trained base model M_1, M_2, … M_T
    - Final prediction by majority voting

Bagging example I

Training set

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

x <= 0.65 ==> y = 1
x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Bagging example II

Combining the predictions

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Bagging overview

- The performance of bagging depends on the stability of base learners
  - If the base learner is unstable, bagging helps to reduce the errors associated with random fluctuations in the training data
  - If a base learner is stable, i.e., robust to minor perturbations of the training set, bagging may not be able to improve the performance of the base learners significantly.
    - It may even degrade the overall performance because the size of each dataset is ~37% smaller than the original data
- It is less susceptible to model overfitting when applied to noisy data
  - since it does not focus on any particular instance of the training data

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all N records are assigned equal weights
  - Unlike bagging, weights may change at the end of boosting round
    - Records that are wrongly classified will have their weights increased
    - Records that are classified correctly will have their weights decreased

- Adaptive boosting; each classifier is dependent on the previous one and focuses on the previous one's errors

- Adaboost

Adaboost (Freund and Schapire, 1995)

- Given a training set D of *d* instances ($\mathbf{X_1}$, y_1), …, ($\mathbf{X_d}$, y_d)
- Initially, all instances have the same weight: 1/d
- A weak learner is trained and its error is computed
- The weights are updated based on the weak learner error
  - If a tuple is misclassified, its weight is increased, o.w. it is decreased
- The new weights are used in the next round
- The final decision (upon the arrival of a new test instance) is a linear combination of the weak learners decisions; the decision of each weak learner is by its error

Adaboost (Freund and Schapire, 1995)

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i] .$$

Error of classifier M_t

- Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$.

the weight of classifier M_t

- Update:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \end{aligned}$$

Weights update

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

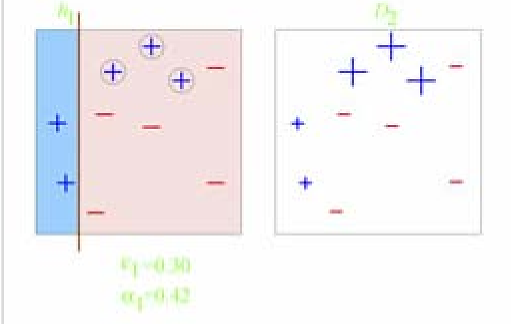
Output the final hypothesis:

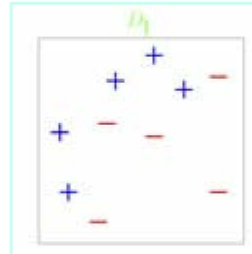
$$H(x) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right) .$$

Adaboost example

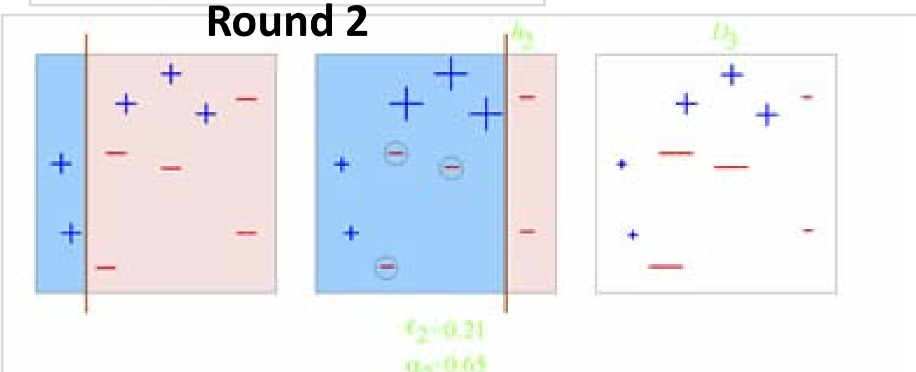
From: http://videolectures.net/mlss05us_schapire_b/

Round 1

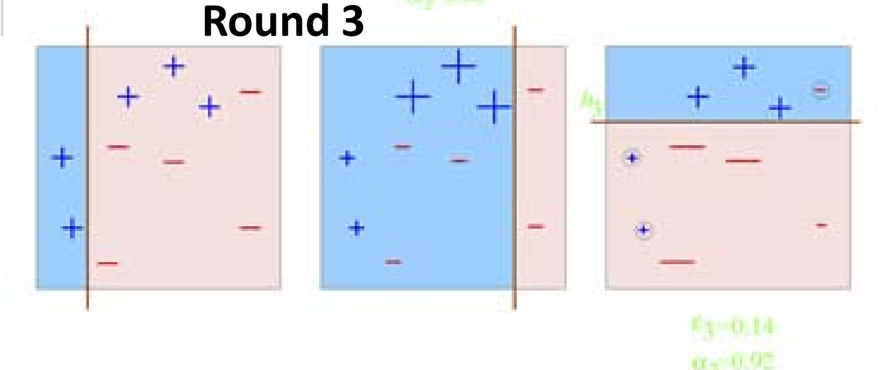




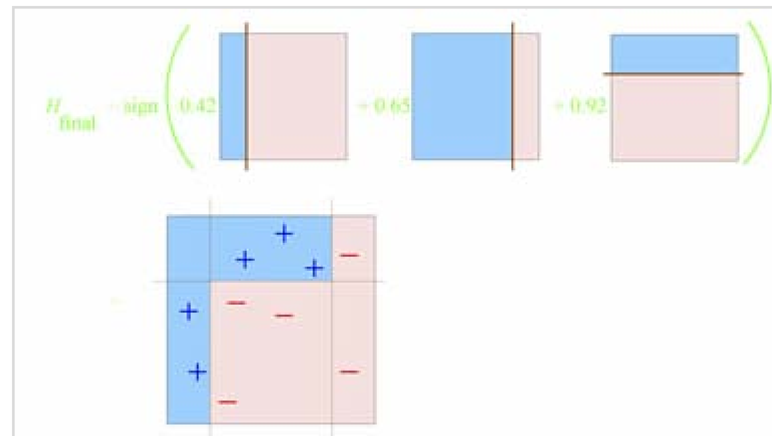
Round 2



Round 3



Final classifier



Boosting overview

- Concentrates on more difficult examples
- Can be quite susceptible to overfitting
  - since it focuses on training examples that are wrongly classified

- Comparing to bagging: boosting tends to achieve greater accuracy, but it also risks overfitting the model to misclassified data

Ensembles: overview

- Pros
  - Better classification performance than individual classifiers
  - More resilience to noise
- Cons
  - Time consuming
  - Overfitting
- Necessary conditions
  - The base classifiers should be independent of each other
  - The base classifiers should do better than a classifier that performs random guessing

Outline

- Introduction
- Support Vector Machines
- Ensembles of classifiers
- An overview of classification
- Things you should know
- Homework/tutorial

Overview of the classification process

- **Model construction:**
  - Based on a training set
  - The class label for each training instance is known
  - The output of this step is a model :
    - e.g. a decision tree, Naïve Bayes etc
- **Model evaluation:**
  - Based on a test set
  - The class label for each testing instance is known and is compared with the model prediction
  - The output of this step are some quality measures:
    - e.g. accuracy
- **Model usage:**
  - If the quality is acceptable, use the model to classify data tuples whose class labels are not known

predefined class values

Class attribute: tenured={yes, no}

Training set

NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

known class label attribute

Test set

NAME	RANK	YEARS	TENURED	PREDICTED
Maria	Assistant Prof	3	no	no
John	Associate Prof	7	yes	no
Franz	Professor	3	yes	yes

known class label attribute

predicted class value by the model

NAME	RANK	YEARS	TENURED	PREDICTED
Jeff	Professor	4	?	yes
Patrick	Associate Prof	8	?	yes
Maria	Associate Prof	2	?	no

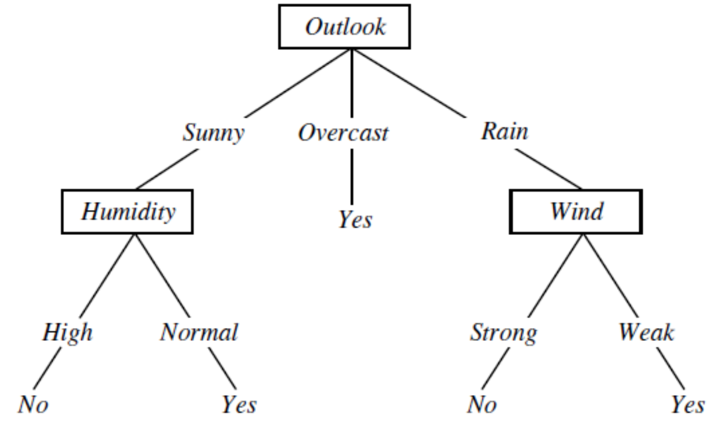
unknown class label attribute

predicted class value by the model

Decision tree classifiers

- A partition-based method

Day	Outlook	Temperature	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



- Selecting the best attribute for splitting
- Avoiding overfitting

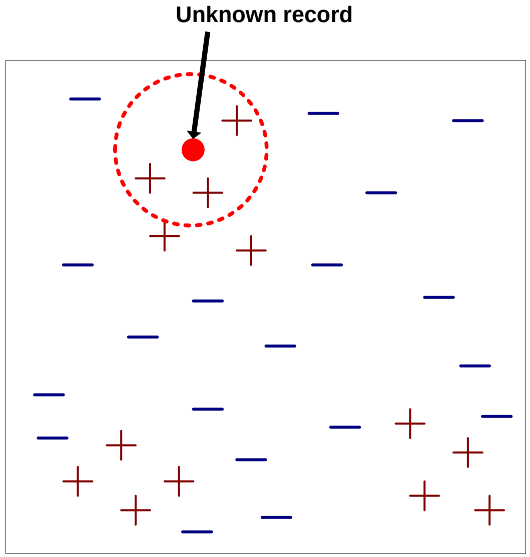
Naïve Bayes classifiers

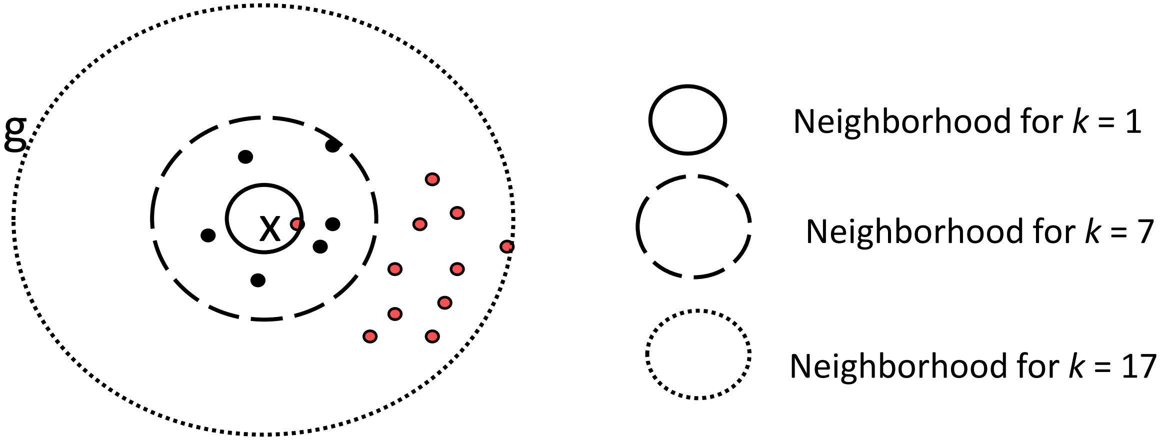
- A statistical method
- Maximum likelihood classification $c = \arg\max_{c \in C} P(c \mid X)$
- Bayes Rule $c = \arg\max_{c \in C} \frac{P(X \mid c)P(c)}{P(X)} = \arg\max_{c \in C} P(X \mid c)P(c)$
- Independency assumption: $\mathrm{P}(\mathrm{X} \mid \mathrm{c}) = \mathrm{P}(\mathrm{A}_1\mathrm{A}_2 \ldots \mathrm{A}_\mathrm{n} \mid \mathrm{c}) = \prod P(A_i \mid c)$
- Estimating:
  - P(c)
  - P(A_i|c)
- Dealing with 0 probabilities

kNN classifiers

- A similarity-based method
- Learning from your neighbors
- Lazy learner

- Distance function
- # of neighbors (k)
- Voting
  - Majority voting
  - Weighted voting

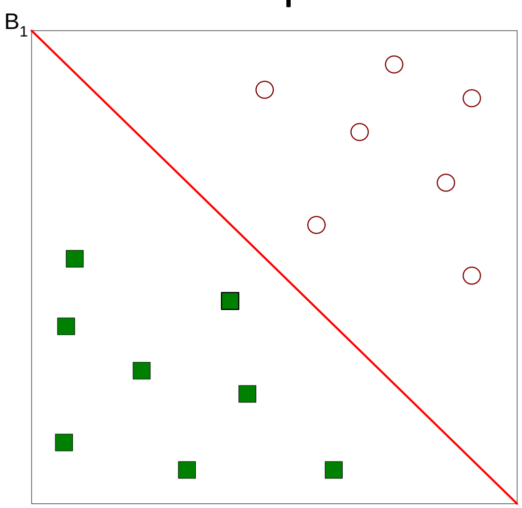




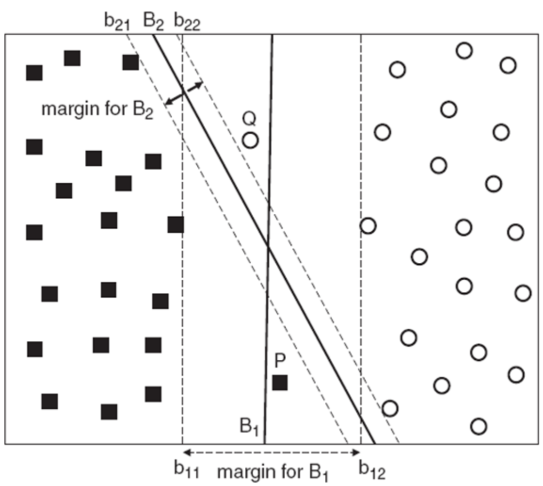
Support Vector machines

- A statistical method
- Maximizes the margin of the decision boundary

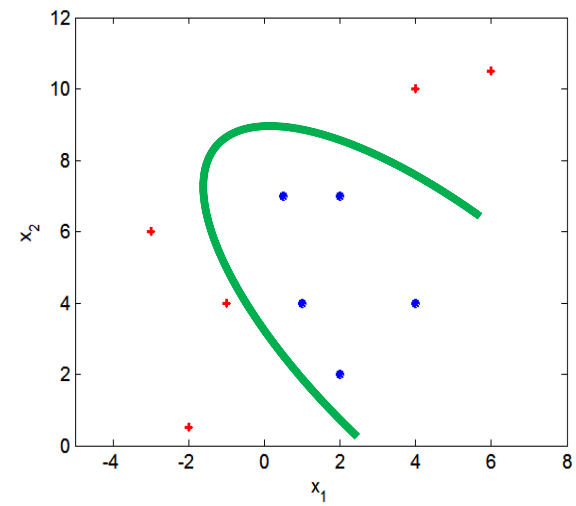
Linear separable



Linear nonseparable



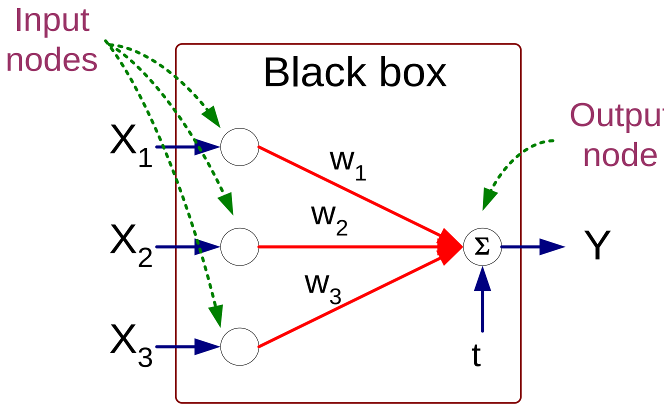
Non linear



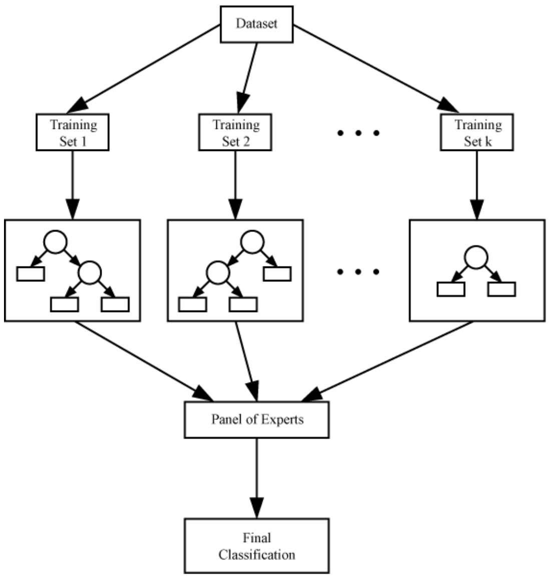
- Kernel functions

More methods

- Neural networks



- Ensembles of classifiers
  - Bagging
  - Boosting



http://en.wikibooks.org/wiki/Proteomics/Protein_Identification_-_Mass_Spectrometry/Data_Analysis/_Interpretation

Evaluation of classifiers: quality measures

Confusion Matrix

Predicted class

Actual class

	C_1	C_2	totals
C_1	TP (true positive)	FN (false negative)	P
C_2	FP(false positive)	TN (true negative)	N
Totals	P'	N'	

Different quality measures:

- Accuracy - Error rate
- Sensitivity - Specificity
- Precision - Recall
- F_1 score/ F-score/ F-measure

Evaluation of classifiers: train – test sets

- Hold-out method
  - Random sampling

- Cross-validation
  - Leave-one-out
  - Stratified cross-validation

- Bootstrap
  - .632 bootstrap

Outline

- Introduction
- Support Vector Machines
- Ensembles of classifiers
- An overview of classification
- Things you should know
- Homework/tutorial

Things you should know

- Support Vector Machines
  - Formulation
  - Linear separable case
  - Linear nonseparable cases
  - Kernel functions
- Ensemble methods
  - Boosting
  - Bagging

Homework/ Tutorial

Tutorial: this Thursday tutorial on

- Decision trees/ Support Vector Machines

Homework:

- Repeat the classification methods learned

Suggested reading:

- Han J., Kamber M., Pei J. *Data Mining: Concepts and Techniques 3rd ed., Morgan Kaufmann, 2011* (Chapters 8, 9)
- Tan P.-N., Steinbach M., Kumar V., *Introduction to Data Mining*, Addison-Wesley, 2006 (Chapter 5).
- Support Vector Machines tutorial by Chih-Jen Lin, Machine Learning Summer School (MLSS), Taipei 2006 (http://videolectures.net/mlss06tw_lin_svm/)
- Boosting tutorial by Robert Schapire, Machine Learning Summer School (MLSS), Chicago 2005 (http://videolectures.net/mlss05us_schapire_b/)