Data Mining II: Advanced Methods and Techniques

Lecture 5

Natasha Balac, Ph.D.

Today

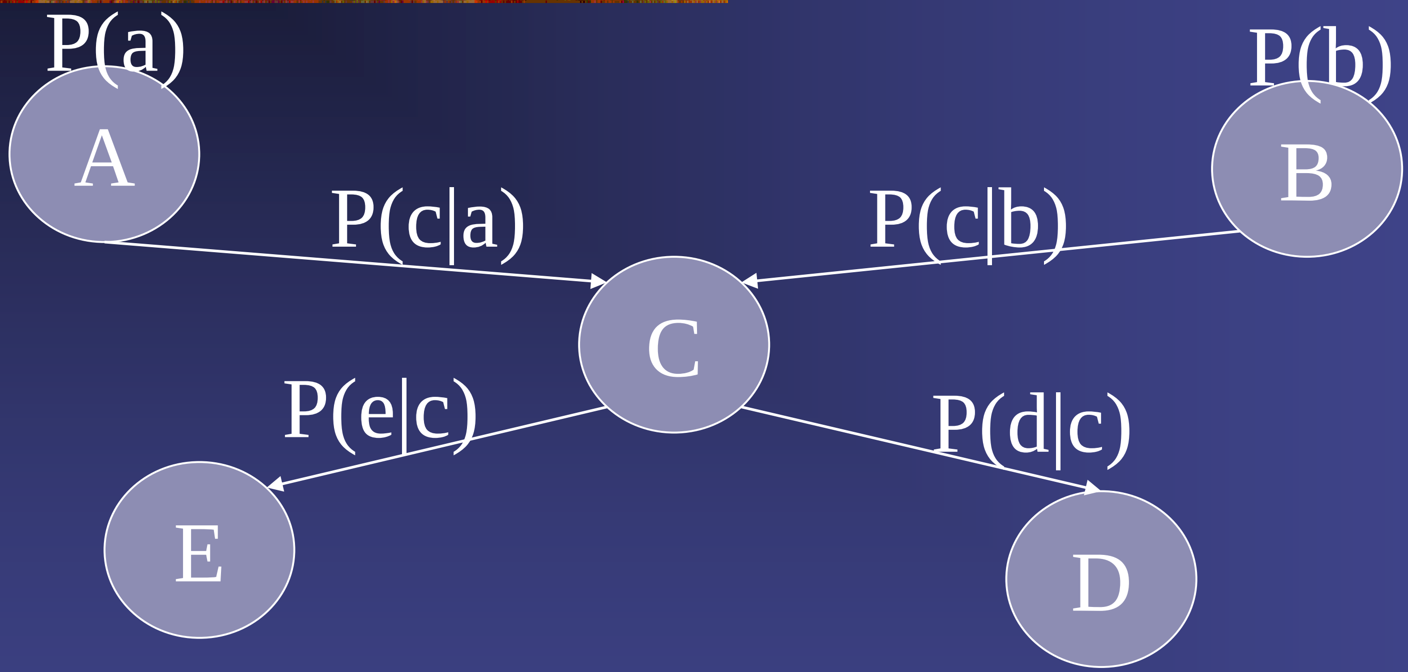
- Bayesian Learning
- HMM
- SVM

Bayesian Decision Theory

Bayesian Decision Theory

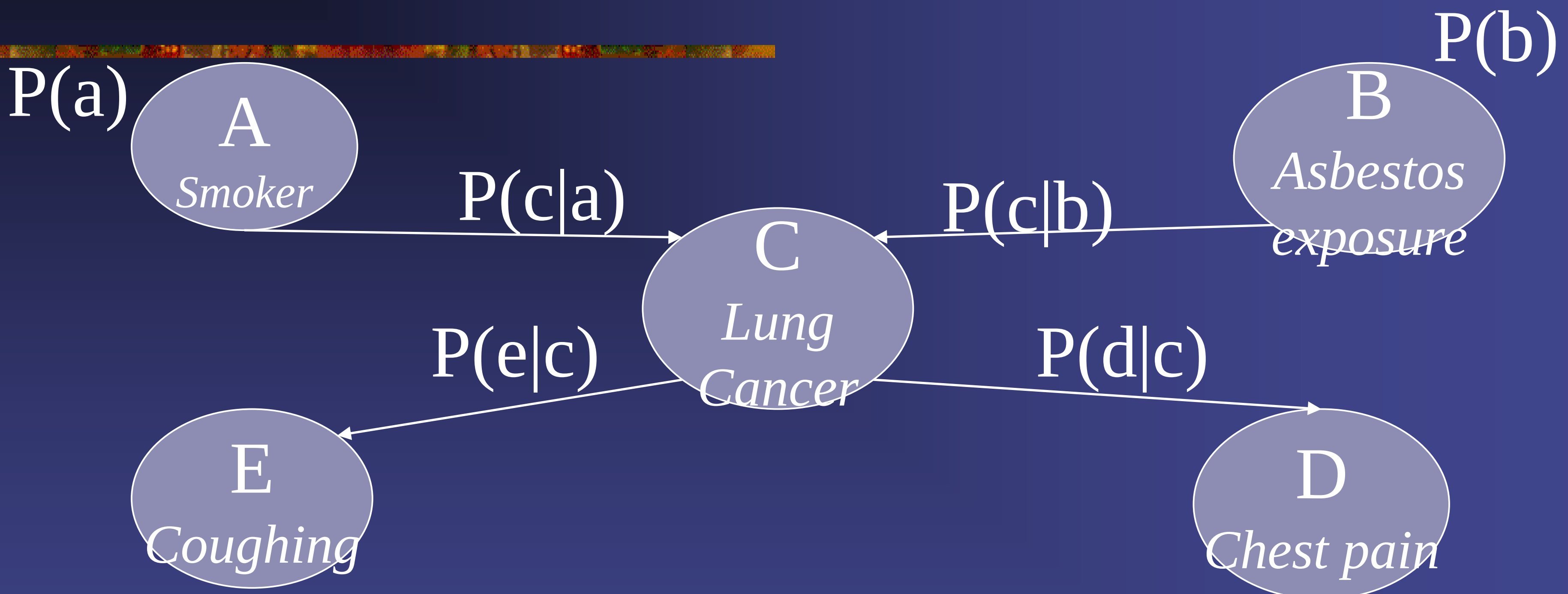
- Originally developed by Thomas Bayes in 1763
- The general idea is that the likelihood of a future event occurring is based on the past probability that it occurred
- A simplified Bayes Theorem simply tells us that in the absence of other evidence
  - the likelihood of an event is equal to its past likelihood

Bayesian Belief Networks



- **Consists of nodes labeled by their discrete states**
- **The links between nodes represent conditional probabilities**
- **Links are directional**
  - **when A points at B, A is said to *influence* B**

Bayesian Belief Network Example



- Over-simplified network
  - how lung cancer is influenced by other states
  - and how the presence of particular symptoms might be influenced by lung cancer

Bayesian Belief Networks Example

A human expert might provide matrices of all the probabilities in the network

P(cancer|smoking):

	cancer	healthy
Never	0.001	0.999
former	0.005	0.995
heavy	0.06	0.94
light	0.04	0.96

…and so forth for the 4 other nodes

- **If we have complete matrices for belief network we can make predictions for any state in the network given a set of input variables**

Bayesian Belief Networks Example

We could now answer questions such as:

- What is the likelihood a person will have lung cancer given that they are a heavy smoker and have been exposed to asbestos?
- A person has severe coughing, chest pain and is a smoker. What is the likelihood of a cancer diagnosis?
- What is the likelihood of past asbestos exposure given that a person has been diagnosed with cancer?

Bayesian Belief Networks

The probability of a particular state is the product of the probabilities of all the states, given their prior states

$$p(\omega_k \mid \mathrm{x}) \propto \prod_{i=1}^{d} p(x_i \mid \omega_k)$$

Markov Chain

- A Markov chain is a type of belief network where you have a sequence of states (x_1, x_2 … x_i) where the probability of each state is dependent only on the previous state

$$P(x_1,x_2,\ldots xi,x_{i+1})$$
$$=P(x_{i+1}|x_1,x_2,\ldots x_i)P(x_1,x_2,\ldots x_i)$$

Hidden Markov Models

- In a Hidden Markov Model (HMM) the a Markov Chain is expanded to include the idea of hidden states
- Given a set of observations $x_1, x_2 \ldots x_n$ and a set of hidden underlying states $s_1, s_2 \ldots s_n$, there is now a *transition probability* for moving between the hidden states:

$$a_{kl} = P(s_i = l \mid s_{i-1} = k)$$

…where l and k are the states at positions I and i-1

Emission Probabilities

- At each state, there is a probability of “emitting” a particular observation
- We define this as

$$e_{kb} = P(x_i = b \mid s_i = k)$$

…where e is the probability that the state *k* at position *i* emits observation *b*

and s_i is the state at position *i* and x_i is the observation at that point

Probability of a Path

The probability of an individual path through a sequence of hidden states in terms of Bayes theorem

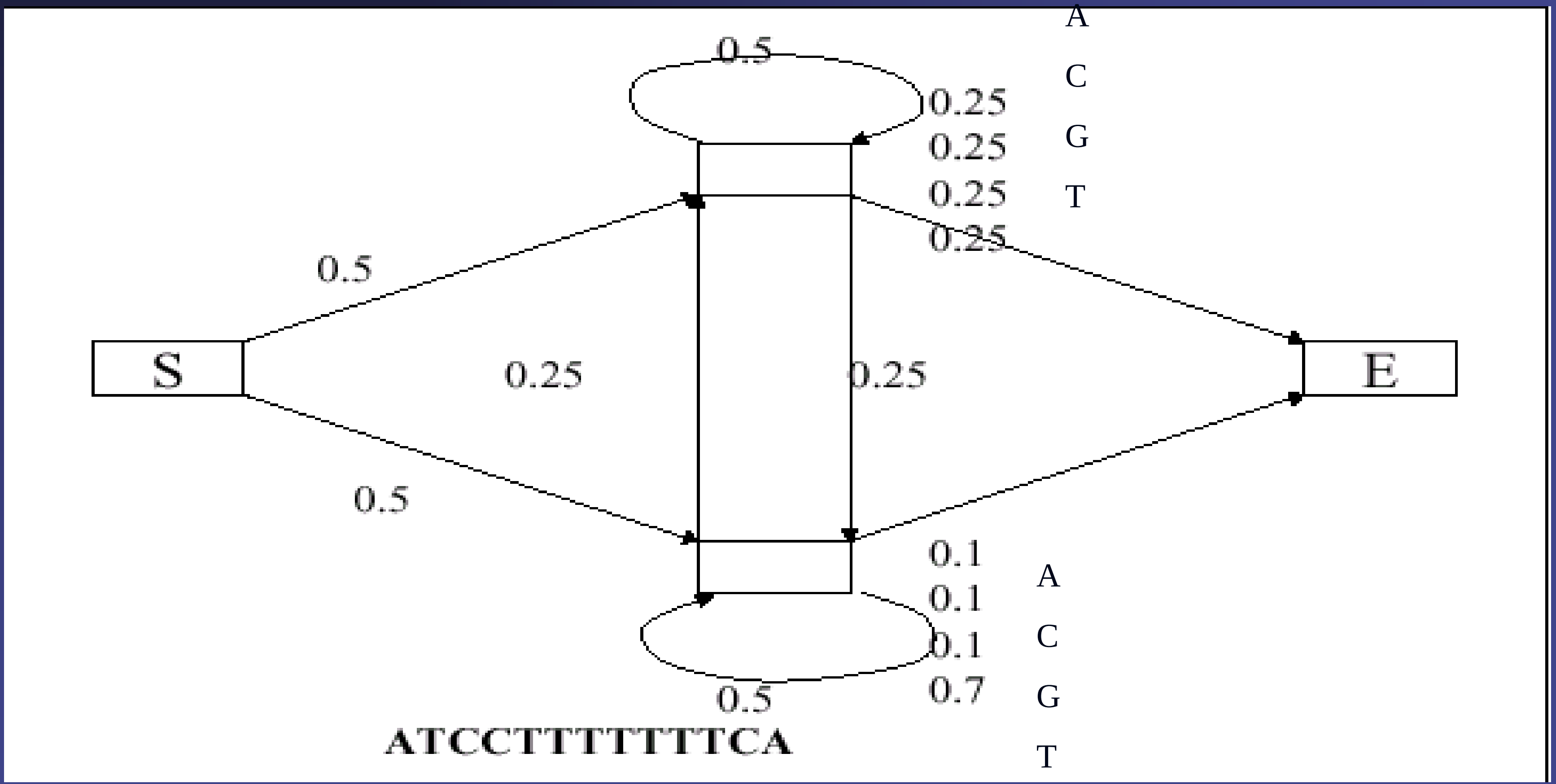
$$P(x,s) = a_{0s_1} \prod_i e_{s_i x_i} a_{s_i s_{i+1}}$$

Probability that we observe the sequence of visible states is equal to the product of the conditional probability that the system has made a particular transition multiplied by the probability that it emitted the observation in our target sequence

HIDDEN MARKOV MODELS (HMM)

- First order discrete HMM: stochastic generative mode for time series
  - ***S*** **: set of states**
  - ***A*** **: discrete alphabet of symbols**
  - ***T*** **: probability transition matrix**
  - ***E*** **: probability emission matrix**
  - **First order assumption: The emission and transition depend on the current state only, not on the entire previous states**
- Meaning of "Hidden"
  - Only emitted symbols are observable
  - Random walk between states are hidden

HMM EXAMPLE



0.5
A
C
G
T
0.25
0.25
0.25
0.25
0.5
S
0.25
0.25
E
0.5
0.1
0.1
0.1
0.7
A
C
G
T
0.5
ATCCTTTTTTTCA

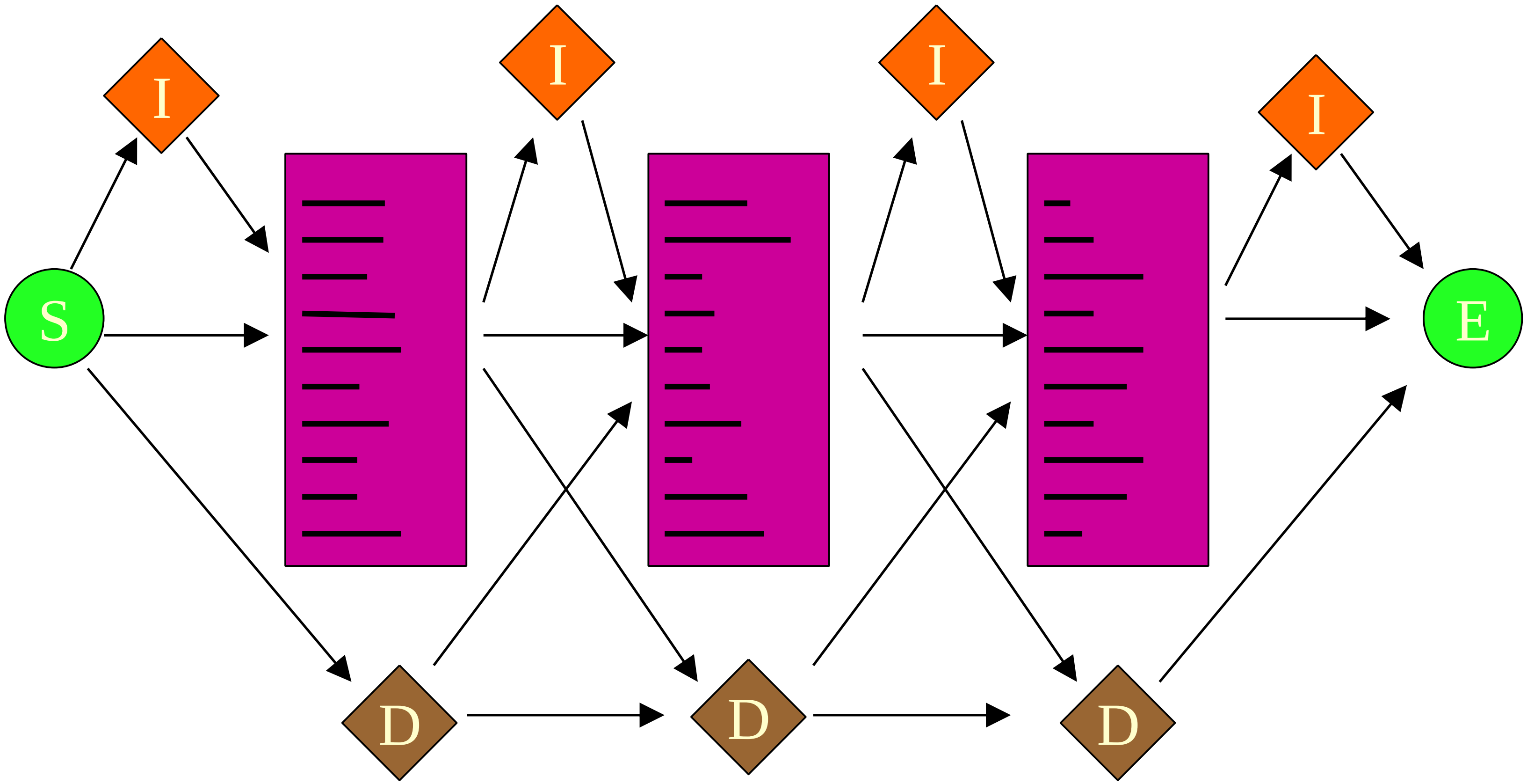
HMMs FOR BIOLOGICAL SEQUENCES

- Most common use
  - To model sequence families

Standard HMM architecture

- start, end
- main states
- insert states
- delete states
- *N* : length of model, typically average length of the sequences in the family

THE STANDARD HMM ARCHITECTURE



3 HMM QUESTIONS

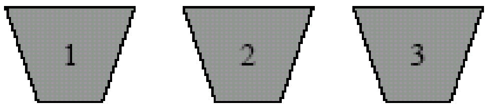
- Likelihood question:
  - How likely is this sequence for this HMM?
- Decoding question:
  - What is the most probable sequence of transitions and emissions through the HMM underlying the production of this particular sequence?
- Learning question:
  - How should their values be revised in light of the observed sequence?

APPLICATION OF HMMs

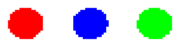
- For any given sequence
  - The computation of its probability according to the model as well as its most likely associated path
  - Analysis of the model structure
- Applications
  - Multiple alignments
  - Database mining and classification of sequence and fragments
  - Structural analysis and pattern discovery

HMM Example

3 Urns



3 Colors of Marbles



Each urn contains a mix of red, blue, and green marbles

$$\Pi = \begin{bmatrix} 0.3 & 0.2 & 0.5 \end{bmatrix}$$

Prob of starting at Urn 1 is 30%

$$A = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$

Prob of staying at Urn 1 is 40%

Prob of trans from Urn 1 to Urn 2 is 30%

Prob of trans from Urn 1 to Urn 3 is 30%

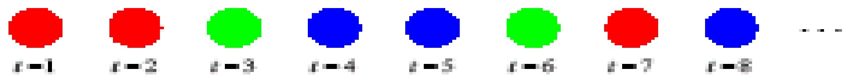
$$B = \begin{bmatrix} 0.2 & 0.8 & 0.0 \\ 0.4 & 0.4 & 0.2 \\ 0.3 & 0.3 & 0.4 \end{bmatrix}$$

20% of marbles in Urn 1 are red

80% of marbles in Urn 1 are blue

No green marbles in Urn 1

Possible sequence of observations



Given a set of observations (marbles), we don't know the state sequence (sequence of urns), that produced the observations.

Hence, it is a *Hidden* Markov Model

Support Vector Machines (SVM)

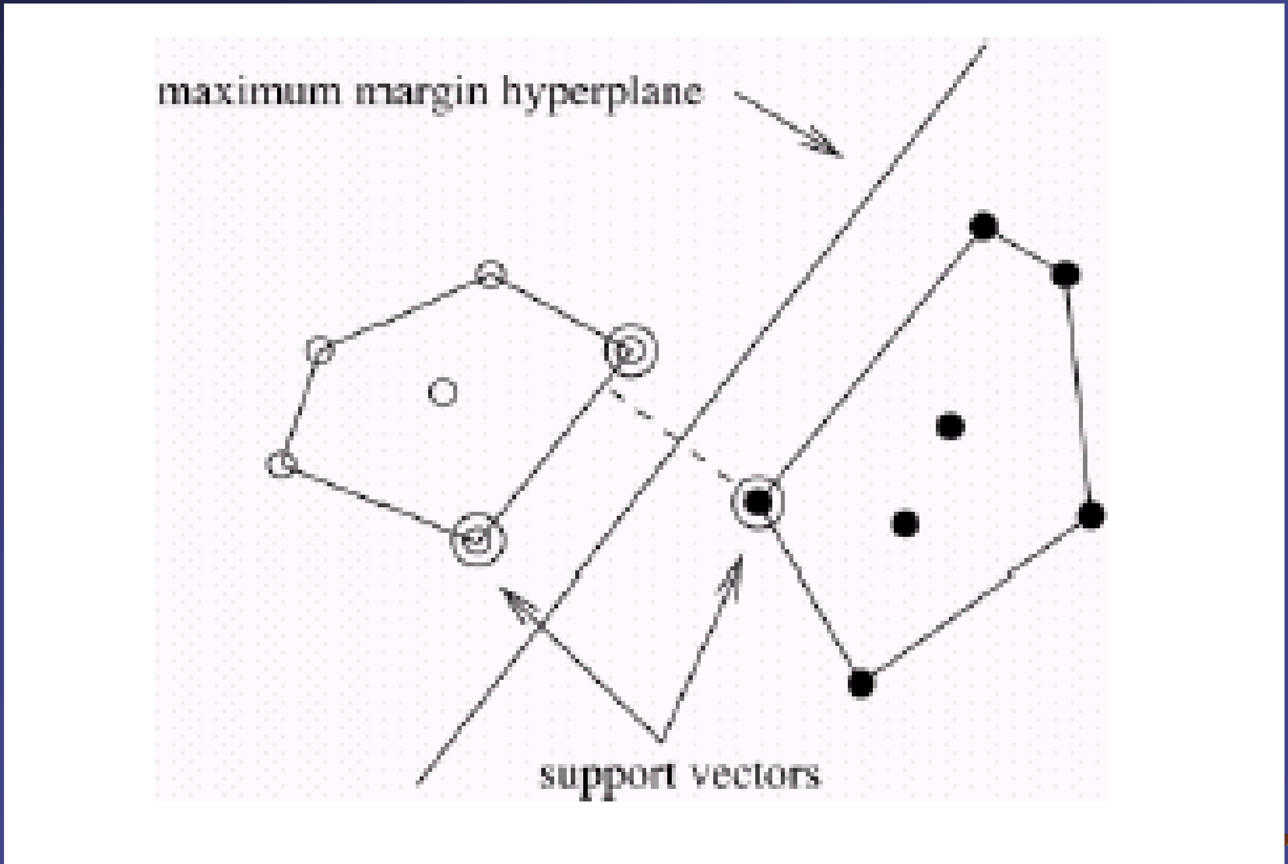
Support Vector Machines (SVM)

- Blend of linear modeling and instance based learning
- SVM select a small number of critical boundary instances called support vectors from each class and build a linear discriminant function that separates them as widely as possible
- They transcend the limitations of linear boundaries by making it practical to include extra nonlinear terms in the calculations
  - making it possible to form quadratic, cubic, higher-order decision boundaries

Support vector machines

- Algorithms for learning linear classifiers
- Resilient to overfitting because they learn a particular linear decision boundary
  - *The maximum margin hyperplane*
- They are fast in the nonlinear case
  - Employ a clever mathematical trick to avoid the creation of "pseudo-attributes"
  - Nonlinear space is created implicitly

The maximum margin hyperplane



Support vectors

- The instances closest to the maximum margin hyperplane are called support vectors
- Important observation: the support vectors define the maximum margin hyperplane!
  - All other instances can be deleted without changing the position and orientation of the hyperplane
- Hyperplane $x = w_0 + w_1 a_1 + w_2 a_2$

  can be written as

$$x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i \mathbf{a}(i) \bullet \mathbf{a}$$

Finding support vectors

- Support vector: training instance for which $\alpha_i > 0$
- Determining α_i and *b* all and is a constrained quadratic optimization problem
  - There are off-the-shelf tools for solving these problems
  - Some special-purpose algorithms are faster
    - Example: Platt's sequential minimal optimization algorithm (implemented in WEKA)
- So far we assumed linearly separable data

Nonlinear SVMs

- Same trick can be applied
  - “pseudo attributes” representing attribute combinations
- Overfitting is unlikely to occur because maximum margin hyperplane is stable
  - There are usually few support vectors relative to the size of the training set
- Computation time still a problem
  - Every time an instance is classified it’s dot product with all support vectors must be calculated

Kernel trick

- Avoid computing the “pseudo attributes”
- We can compute the dot product before the nonlinear mapping is performed
- Example: instead of computing

$$x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i \mathbf{a}(i) \bullet \mathbf{a}$$

- we can compute

$$x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i (\mathbf{a}(i) \bullet \mathbf{a})^n$$

- This corresponds to a map into the instance space spanned by all products of n attributes

Noise

- So far we have assumed that the data is separable (in original or transformed space)
- SVMs can be applied to noisy data by introducing a “noise” parameter C
- C bounds the influence of any one training instance on the decision boundary
- Corresponding constraint: $0 \le \alpha_i \le C$
- Still a quadratic optimization problem C has to be found by experimentation

Sparse data

- SVM algorithms can be sped up dramatically if the data is sparse (many values are 0)
- Why? Because they compute lots and lots of dot products
- With sparse data dot products can be computed very efficiently
  - We just need to iterate over the values that are non-zero
- SVMs can process sparse data sets with tens of thousands of attributes

Applications

- Machine vision: face identification
  - Outperforms alternative approaches (1.5% error)
- Handwritten digit recognition: USPS data
  - Comparable to best alternative (0.8% error)
- Bioinformatics: prediction of protein secondary structure
- Text classification
- Algorithm can be modified to deal with numeric prediction problems

Differences between MLP and SVM

- In MLPs complexity is controlled by keeping number of hidden nodes small
- In SVM complexity is controlled independently of dimensionality
- The mapping means that the decision surface is constructed in a very high (often infinite) dimensional space
- Curse of dimensionality (makes finding the optimal weights difficult) is avoided by using the notion of an inner product kernel and optimizing the weights in the input space

SVM Strengths

- Complexity/capacity is independent of dimensionality of the data thus avoiding curse of dimensionality
- Statistically motivated
  - Can get bounds on the error
- Finding the weights is a quadratic programming problem - guaranteed to find a minimum of the error surface
  - Thus the algorithm is efficient and SVMs generate near optimal classification and are insensitive to overtraining
- Obtain good generalization performance due to high dimension of feature space

SVM Strengths

- SVMs are a superclass of network containing both MLPs and RBFNs
  - both can be generated using the SV algorithm
- By using a suitable kernel SVM automatically computes all network parameters for that kernel
- Example:
  - RBF SVM: automatically selects the number and position of hidden nodes (and weights and bias)

Weaknesses

- Slow training (compared to RBFNs/MLPs) computationally intensive solution especially for large amounts of training data => need special algorithms
- Generates complex solutions
  - normally > 60% of training points are used as support vectors - especially for large amounts of training data
- Example (from Haykin's paper) increase in performance of 1.5% over MLP
  - MLP used 2 hidden nodes and SVM used 285
- Difficult to incorporate prior knowledge

Summary

- The SVM was proposed by Vapnik and colleagues in the 70's but has only recently become popular early 90's
- It (and other kernel techniques) is currently a very active (and trendy) topic of research

More information:

http://www.kernel-machines.org

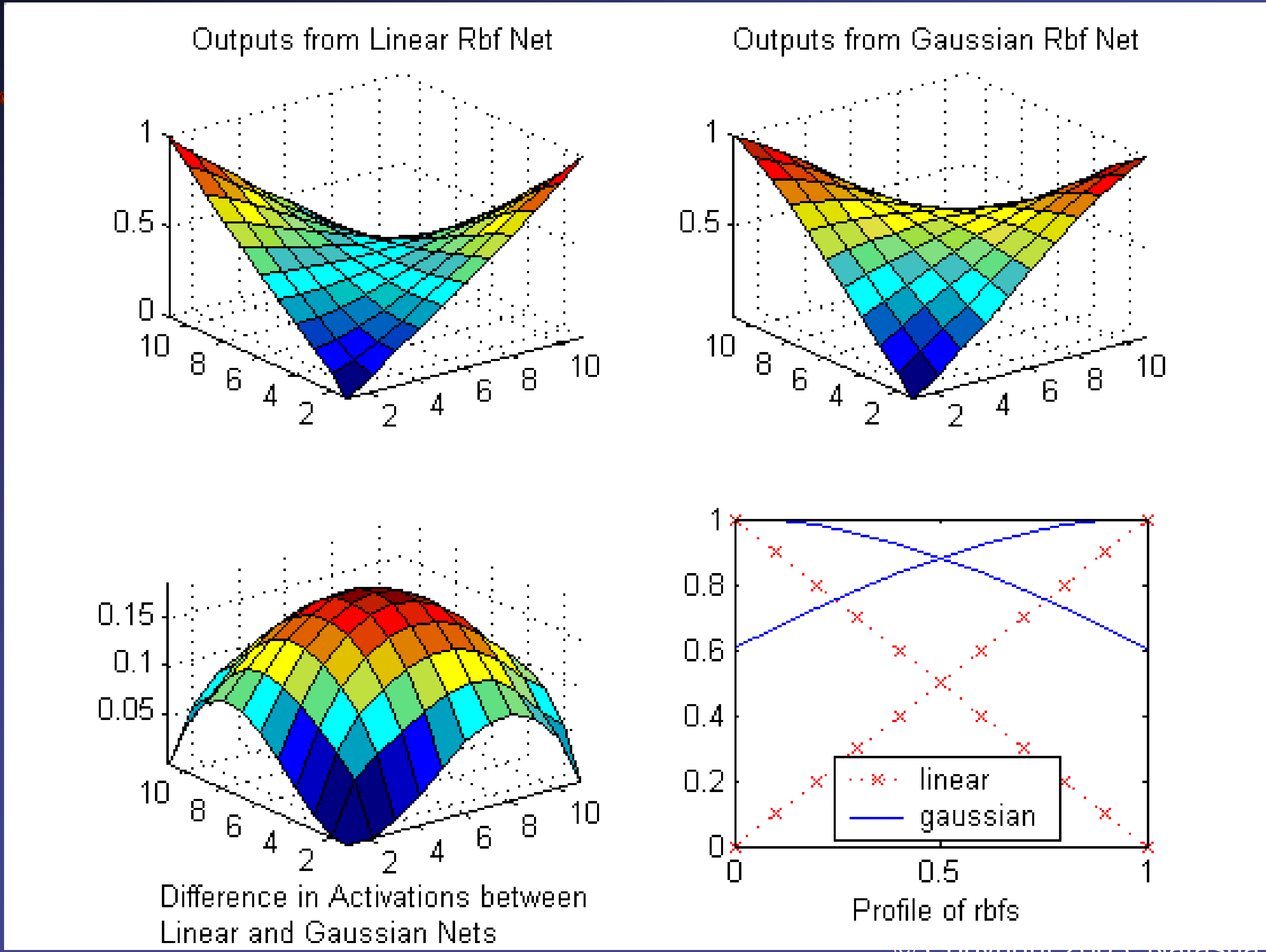
book:

AN INTRODUCTION TO SUPPORT VECTOR MACHINES (and other kernel-based learning methods). N. Cristianini and J. Shawe-Taylor, Cambridge University Press. 2000. ISBN: 0 521 78019 5

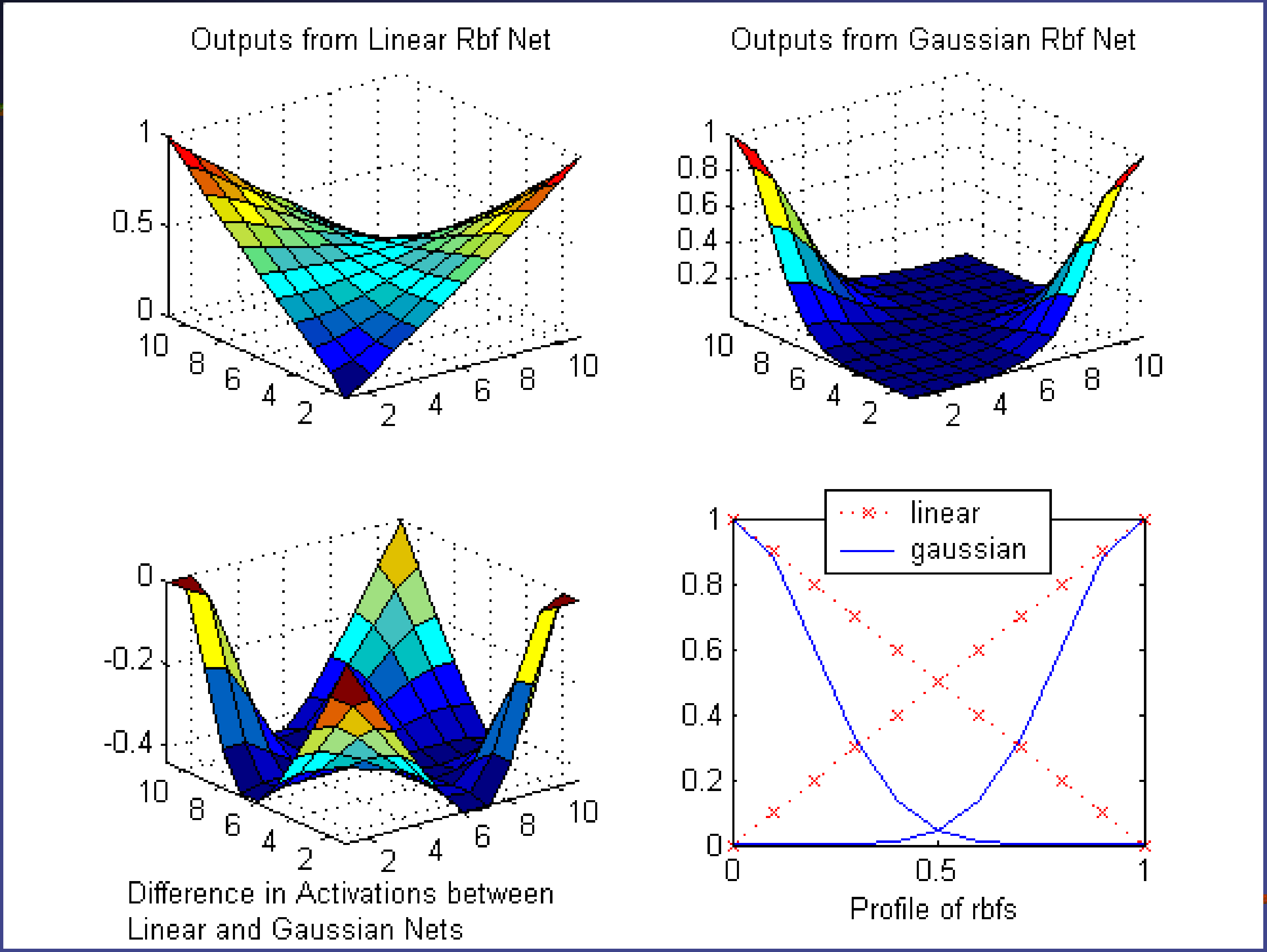
LAB Time!!

- Lab #3

Large σ = 1

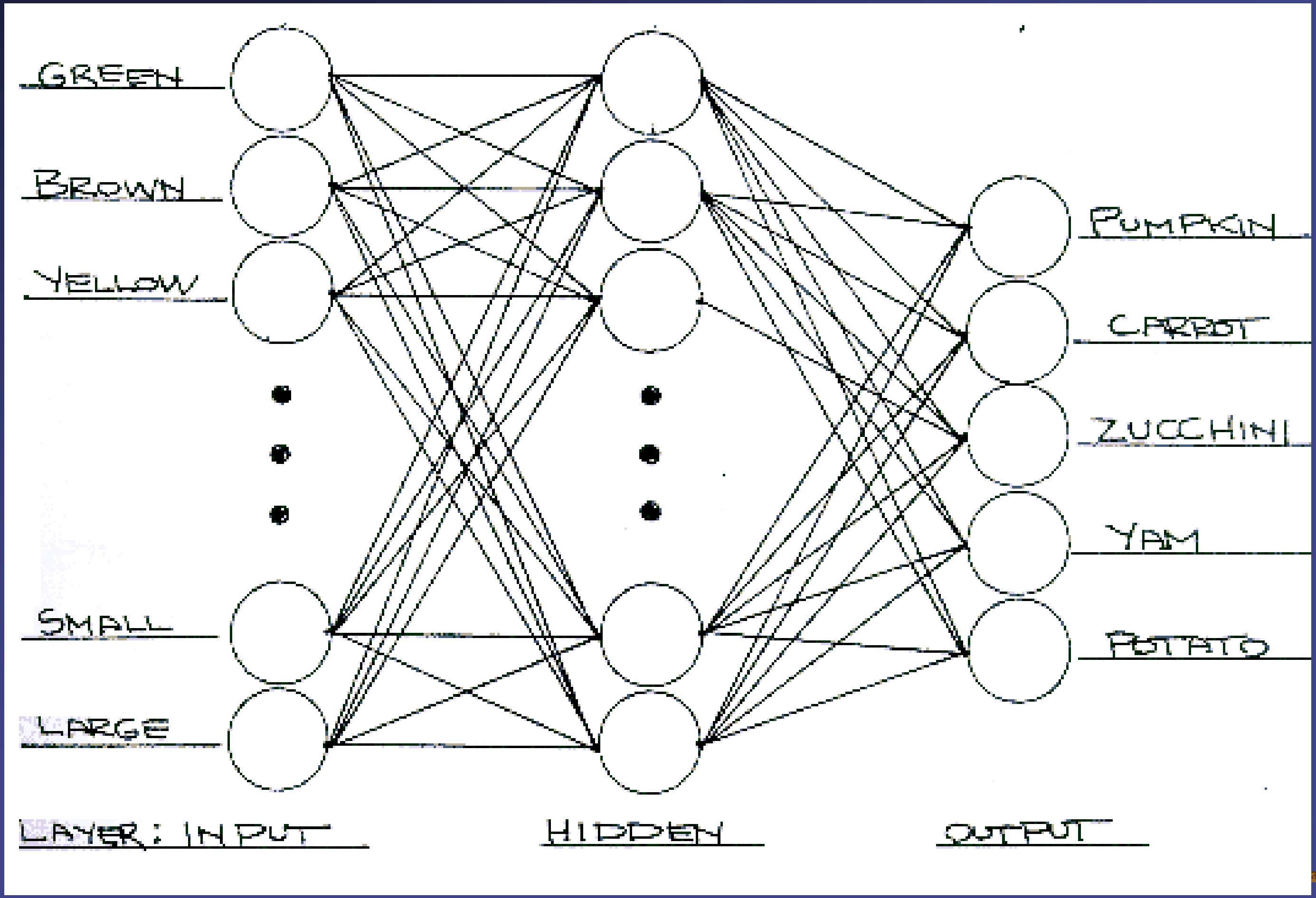

Outputs from Linear Rbf Net
Outputs from Gaussian Rbf Net
Difference in Activations between
Linear and Gaussian Nets
linear
gaussian
Profile of rbfs

Small σ = 0.2



Outputs from Linear Rbf Net
Outputs from Gaussian Rbf Net
linear
gaussian
Profile of rbfs
Difference in Activations between
Linear and Gaussian Nets

Food Recognition Neural Network

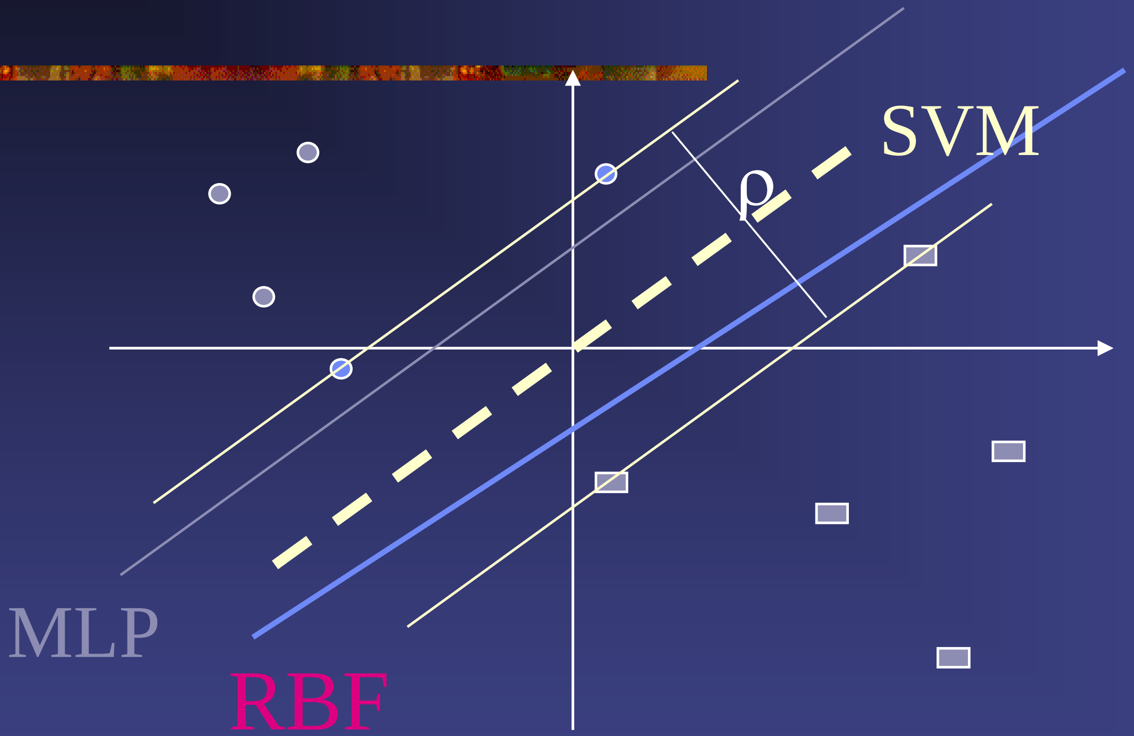


From Introduction to Neural Networks by J. Lawrence

Support Vector Machines (SVMs)

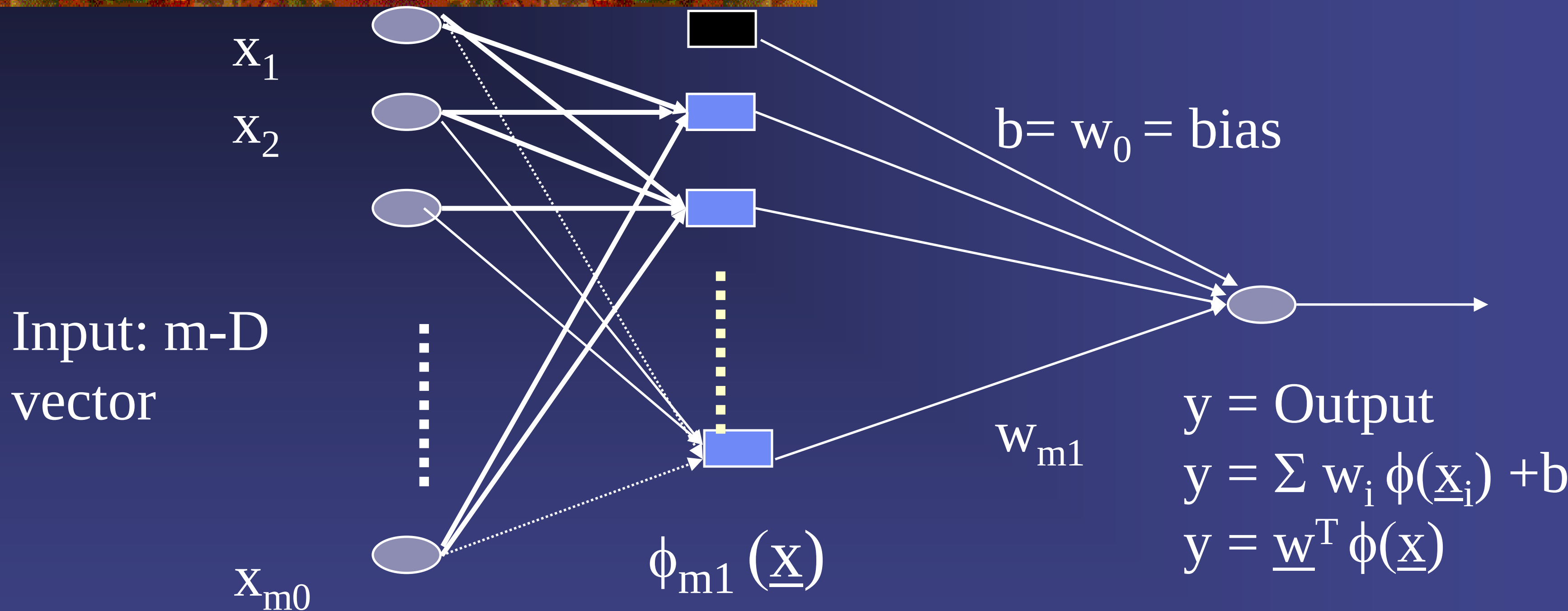
1. Transform the data with a non-linear mapping f so that it is linearly separable
2. Cover's theorem: non-linearly separable data can be transformed into a new feature space which is linearly separable if
    1. mapping is non-linear
    2. dimensionality of feature space is high enough
3. Construct the 'optimal' hyperplane (linear weighted sum of outputs of first layer) which maximizes the degree of separation (the *margin of separation* - r) between the 2 classes

Minimizing Error



- MLPs and RBFN stop training when all points are classified correctly
- Decision surfaces are not optimized in the sense that the generalization error is not minimised

First layer

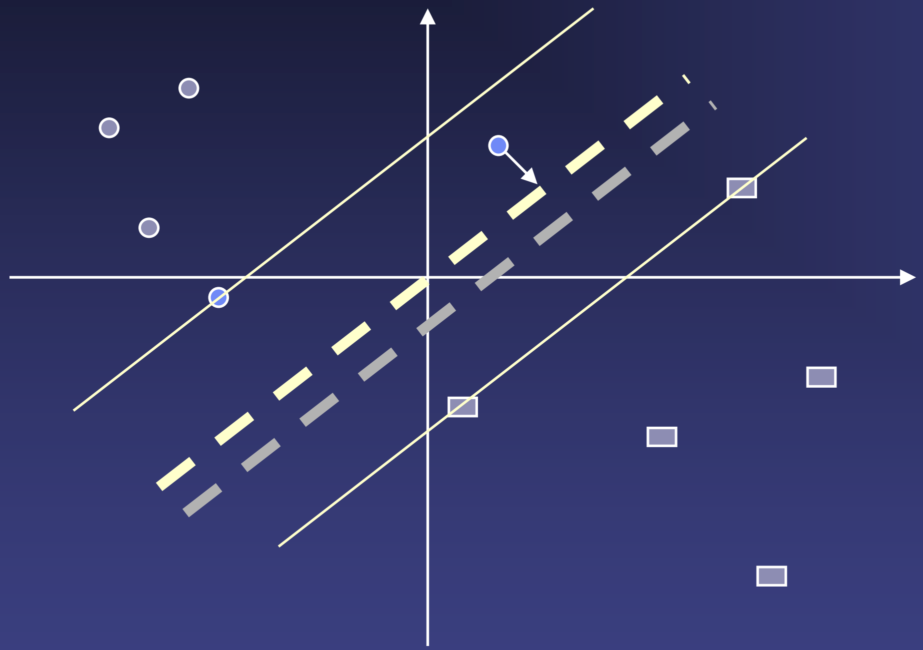


First layer: mapping performed from the input space into a feature space of higher dimension where the data is now linearly separable using a set of m_1 non-linear functions (RBFNs)

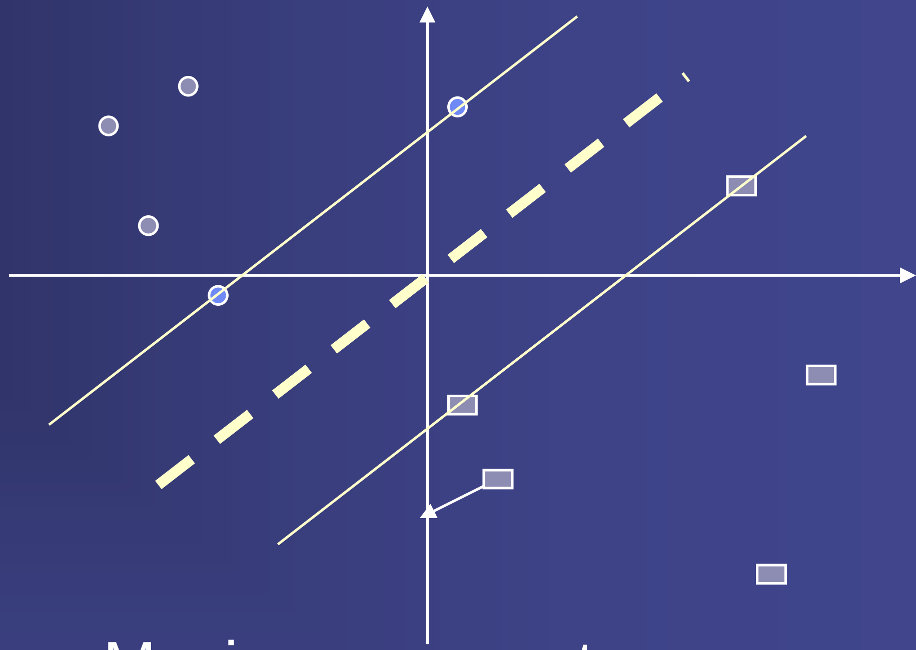
After learning

1. RBFN /MLP decision surfaces might not be at optimal position
   1. Example, as shown in the figure, both learning rules will not perform further iterations (learning) since the error criterion is satisfied
2. In contrast the SVM algorithm generates the optimal decision boundary (the dotted line) by maximizing the distance between the classes r
   1. which is specified by the distance between the decision boundary and the nearest data points
3. Points which lie exactly r/2 away from the decision boundary are known as ***Support Vectors***
   1. Most important points since moving them moves the decision boundary

Decision Boundary



Moving the other vectors has no effect



Moving a support vector moves the decision boundary

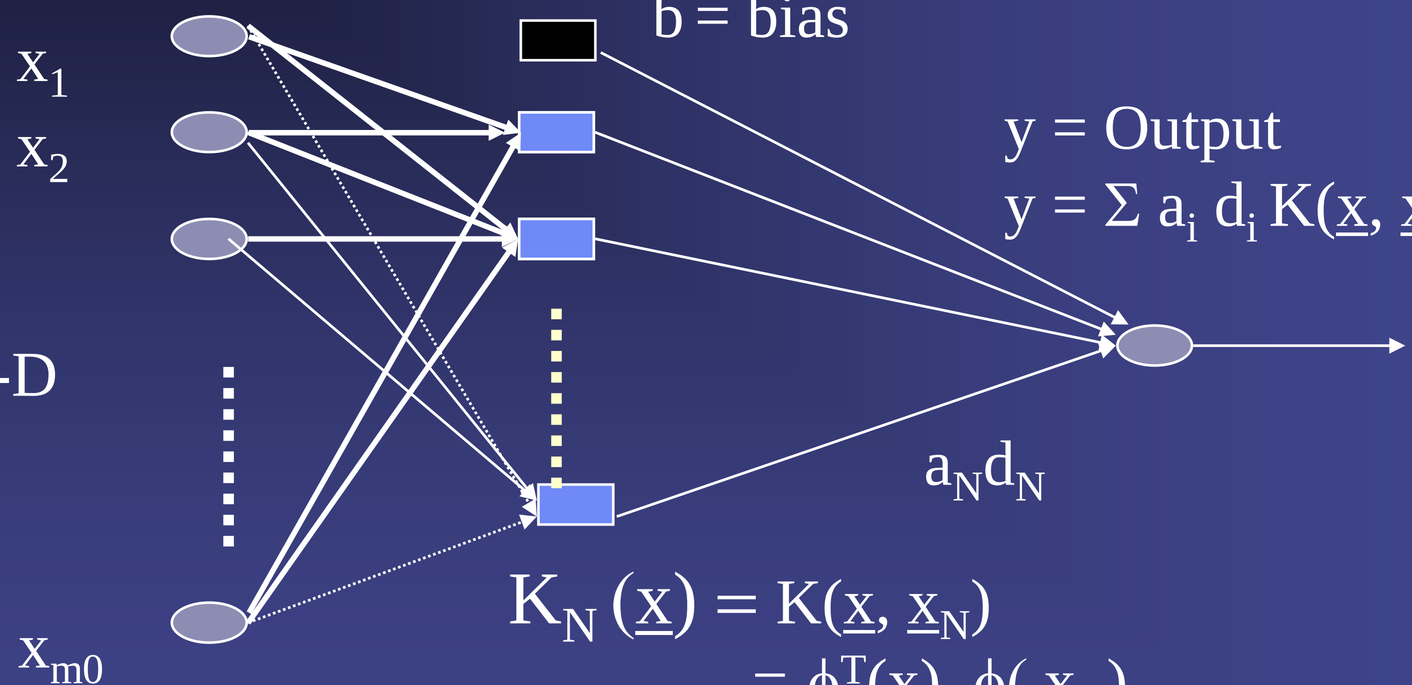
The algorithm to generate the weights proceeds in such a way that ***only*** the support vectors determine the weights and thus the boundary

Output of the SVM

- Output of the SVM can also be interpreted as a weighted sum of the inner (dot) products of the
  - images of the input $\underline{x}$
  - and the support vectors $\underline{x}_i$ in the feature space
- which is computed by an inner product kernel function $K(\underline{x},\underline{x}_m)$

Output of the SVM

Input: m-D vector



y = Output

$y = \Sigma\, a_i\, d_i\, K(\underline{x}, \underline{x}_i) + b$

$K_N(\underline{x}) = K(\underline{x}, \underline{x}_N)$

$= \underline{\phi}^T(\underline{x}).\ \underline{\phi}(\underline{x}_N)$

Where: $\underline{\phi}^T(\underline{x}) = [\phi_1(\underline{x}), \phi_2(\underline{x}), .., \phi_{m1}(\underline{x})]^T$ I.e. image of x in feature space

and d_i = +/- 1 depending on the class of $\underline{x}_i$