

# Data Science for Researchers and Scholars

**Vasant G. Honavar**

Dorothy Foehr Huck and J. Lloyd Huck Chair in Biomedical Data Sciences and Artificial Intelligence  
Professor of Data Sciences, Informatics, Computer Science and Engineering, Bioinformatics & Genomics,  
Public Health Sciences and Neuroscience  
Director, Center for Artificial Intelligence Foundations and Scientific Applications  
Associate Director, Institute for Computational and Data Sciences  
Pennsylvania State University


[vhonavar@psu.edu](mailto:vhonavar@psu.edu)  
<http://faculty.ist.psu.edu/vhonavar>  
<http://ailab.ist.psu.edu>

**PennState**  
Institute for Computational  
and Data Sciences

Center for Artificial Intelligence Foundations & Scientific Applications  
Artificial Intelligence Research Laboratory

**PennState**  
Clinical and Translational  
Science Institute

# Maximum Margin Classifiers

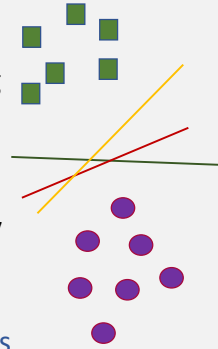
**PennState**  
College of Engineering  
Science and Technology


Data Science for Researchers and Scholars

50  
Vasant Honavar, Fall 2023


## Linear classifiers revisited

- We have considered algorithms that can find a linear hyperplane that correctly labels the training data when the data are linearly separable.
- Perceptron algorithm finds a hyperplane (from among an infinite number of choices) that correctly classifies the training samples when they are linearly separable
- However, our goal is not just to correctly label the training samples, but to correctly predict the labels of data samples not seen during training!
- Is one of the infinite number of hyperplanes optimal in the sense that it yields the lowest error on samples not seen in the training data (assuming they are also linearly separable)?

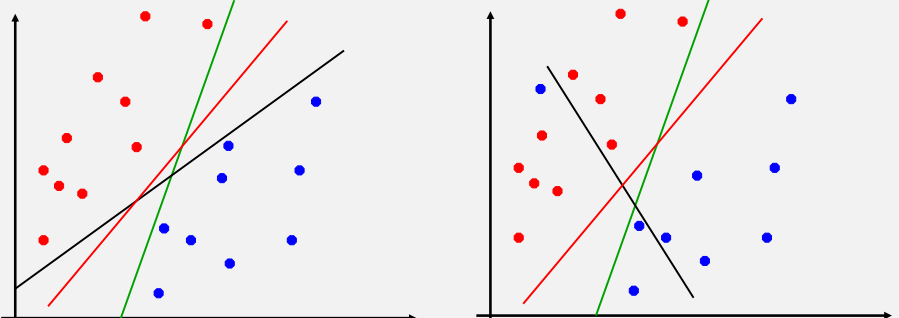


**PennState**  
Institute for Computational  
and Data Sciences

Center for Artificial Intelligence Foundations & Scientific Applications  
Artificial Intelligence Research Laboratory


**PennState**  
Clinical and Translational  
Science Institute

## Which hyperplane to choose?



Linear classifiers differ with respect to:

- which hyperplane they choose when data are linearly separable
- how they handle data that are not linearly separable

**PennState**  
College of Engineering  
Science and Technology

Data Science for Researchers and Scholars

Vasant Honavar, Fall 2023

## Which hyperplane to choose?

- When data are separable, the perceptron algorithm finds **some** hyperplane that separates the data
  - Can we do better?
  - What does it mean to do better?
  - Minimize the error on the unseen samples?
  - How can we do so when the samples are unseen?
- **Is one of the infinite number of hyperplanes optimal in the sense that it yields the lowest error on samples not seen in the training data (assuming that the data are separable)?**
- This question occupied Vladimir Vapnik for nearly 30 years!
- After 30 years of painstaking work by Vapnik and others, we know the answer to this question
- The resulting insights led to rapid progress in machine learning during 1990-2010

## History of Key Developments

1958 Perceptron (Rosenblatt)

1963 Margin (Vapnik)



1964 Kernel Trick (Aizerman)

1965 Optimization formulation (Mangasarian)



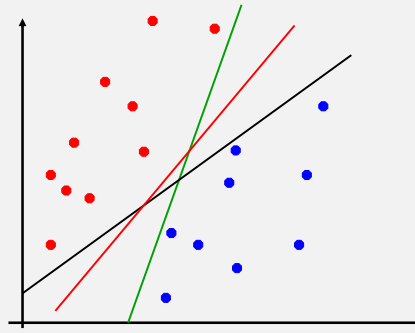
1971 Kernels (Wahba)



1992 SVM (Vapnik)

1996 – present      Rapid growth, numerous applications  
Extensions to other problems

Which hyperplane would you choose?



## The Generalization Problem

- The learning problem is ill posed
  - There are infinitely many hyperplanes that separate the training data
  - Need a principled approach to choose an optimal hyperplane
  - What does it mean for a hyperplane to be optimal?
  - Minimize the true error of classification



## Notation

In what follows, for consistency with the literature, we will use  $\mathbf{w}$  to denote  $[w_1 \cdots w_N]^T$  and  $b$  to denote  $w_0$  and  $\mathbf{x}$  to denote  $[x_1 \cdots x_N]^T$

The linear hyperplane (or threshold function) is given by

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

**PennState**  
Institute for Computational  
and Data Sciences

**Center for Artificial Intelligence Foundations & Scientific Applications**  
Artificial Intelligence Research Laboratory

**PennState**  
Clinical and Translational  
Science Institute

## A Little Learning Theory (without proofs)

Suppose:

- We are given  $P$  training examples  $(\mathbf{x}_i, d_i)$
- Train and test samples drawn randomly from some unknown probability distribution
- The machine learns the function  $h_{\mathbf{w},b}(\mathbf{x})$
- A particular choice of  $\mathbf{w}, b$  specifies “trained machine”
- The expectation  $E(\mathbf{w}, b)$  of the test error is the probability that  $h_{\mathbf{w},b}(\mathbf{x})$  incorrectly labels samples drawn according to the distribution that generated the training data
- The empirical estimate of  $\hat{E}(\mathbf{w}, b) = \frac{1}{P} \sum_p \max\{-d_p h_{\mathbf{w},b}(\mathbf{x}_p), 0\}$

**PennState**  
College of Engineering,  
Science and Technology

Data Science for Researchers and Scholars

Vasant Honavar, Fall 2023

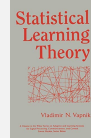
## Bounding Generalization Error

Choose some  $\delta$  such that  $0 < \delta < 1$ . With probability  $1 - \delta$ , the following distribution independent bound holds (Vapnik, 1995):

$$E(\mathbf{w}, b) \leq \hat{E}(\mathbf{w}, b) + \sqrt{\frac{d\left(\log\left(\frac{2P}{d}\right) + 1\right) - \log\frac{\delta}{4}}{P}}$$

where  $d \geq 0$  is called Vapnik-Chervonenkis (VC) dimension is a measure of “capacity” of machine (proof omitted) and  $P$  is the number of training samples

- VC dimension is a key notion in the theory of machine learning
- VC dimension impacts the number of samples needed to reliably learn a classifier of a certain type (e.g., a linear hyperplane)



## Bounding Generalization Error

Choose some  $\delta$  such that  $0 < \delta < 1$  With probability  $1 - \delta$ , the following distribution independent bound holds (Vapnik, 1995):

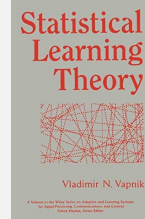
$$E(\mathbf{w}, b) \leq \hat{E}(\mathbf{w}, b) + \sqrt{\frac{d(\log(\frac{2P}{d}) + 1) - \log \frac{\delta}{4}}{P}}$$

where  $d \geq 0$  is called VC dimension is a measure of “capacity” of machine

- If the training data are linearly separable, it is possible to achieve  $\hat{E}(\mathbf{w}, b) = 0$ .

- Then  $E(\mathbf{w}, b) = \sqrt{\frac{d(\log(\frac{2P}{d}) + 1) - \log \frac{\delta}{4}}{P}}$

or  $E(\mathbf{w}, b)$  can be minimized by controlling  $d$  and increasing  $P$  independent of the dimensionality of the data (number of features)



## Margin

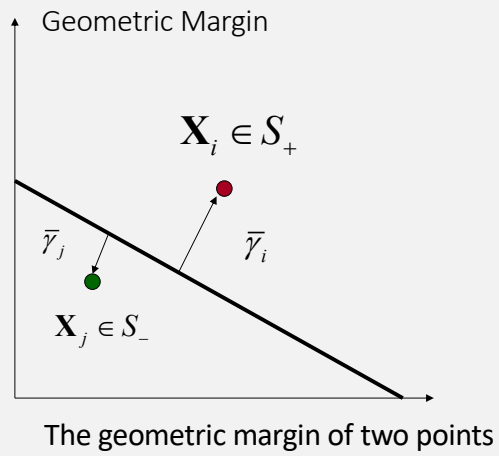
- The **functional margin** of a linear hyperplane specified by  $(\mathbf{w}, b)$  w.r.t. a labeled pattern  $(\mathbf{x}_i, d_i)$  is defined as

$$\gamma_i = d_i (\mathbf{w} \cdot \mathbf{x}_i + b)$$

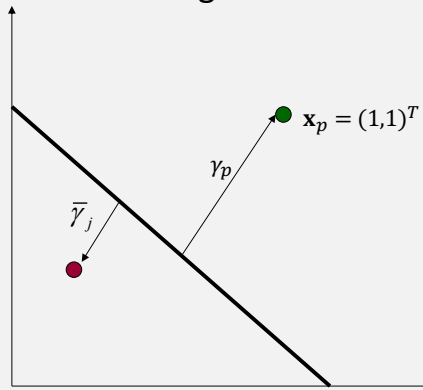
- If the functional margin is negative, then the pattern is incorrectly classified, if it is positive then the classifier predicts the correct label.
- The larger  $|\gamma_i|$ , the further away  $\mathbf{x}_i$  is from the hyperplane
- This is made more precise in the notion of the **geometric margin**

$$\bar{\gamma}_i = \frac{\gamma_i}{\|\mathbf{w}\|}$$

which measures the Euclidean distance of  $\mathbf{x}_i$  from the decision boundary



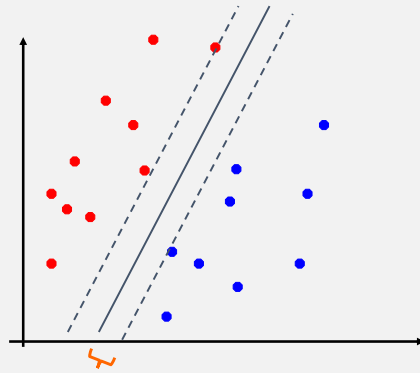
## Geometric Margin



### Example

$$\begin{aligned}
 \mathbf{w} &= (1,1)^T \\
 b &= -1 \\
 \mathbf{x}_p &= (1,1)^T \\
 d_p &= 1 \\
 \gamma_p &= (d_p)(\mathbf{w} \cdot \mathbf{x}_p + b) \\
 &= (1)(1 + 1 - 1) \\
 &= 1 \\
 \bar{\gamma}_p &= \frac{1}{\|\mathbf{w}\|} = \frac{1}{\sqrt{2}}
 \end{aligned}$$

## Margin of a hyperplane with respect to a data set



$$\bar{\gamma} = \min_p \frac{d_p(\mathbf{w} \cdot \mathbf{x}_p + b)}{\|\mathbf{w}\|}$$



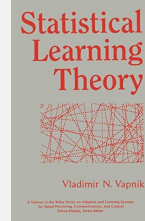
## Margin based bound on generalization error

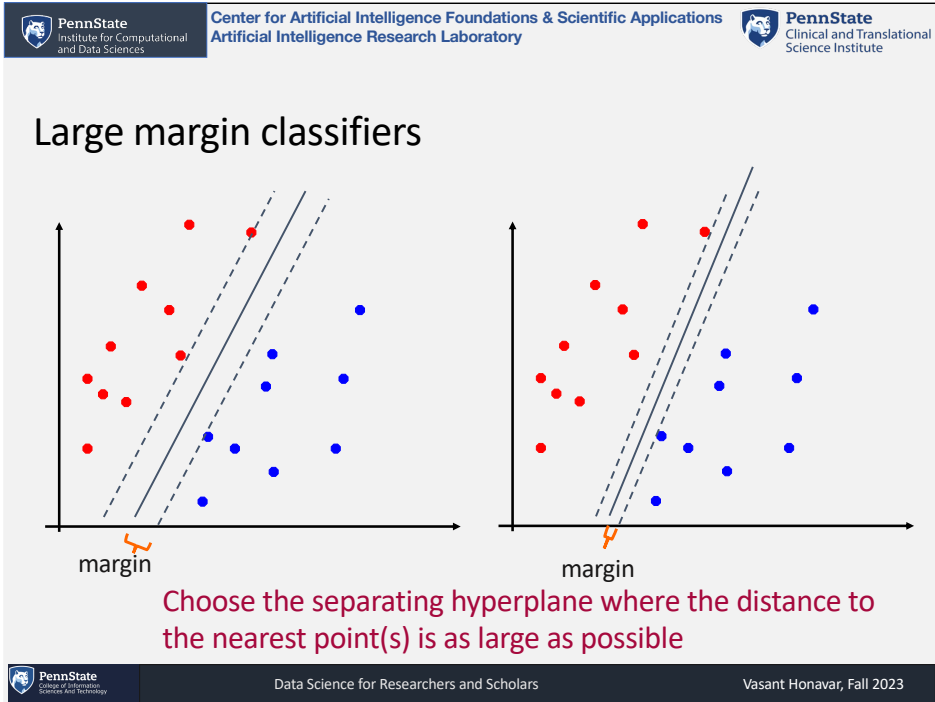
Margin based bound


$$\bar{\gamma} = \min_p \frac{d_p(\mathbf{w} \cdot \mathbf{x}_p + b)}{\|\mathbf{w}\|}$$

$$L = \max_p \|\mathbf{x}_p\| \quad \varepsilon = O\left(\frac{1}{P} \left(\frac{L}{\bar{\gamma}}\right)^2\right)$$

- Error  $\varepsilon$  of the classifier trained on a separable data set is inversely proportional to its margin, the number of training samples, regardless of the dimensionality of the input space! (proof omitted)
- If you want to minimize the error of the classifier on data unseen during training, you should pick a hyperplane with the largest margin on the training data!








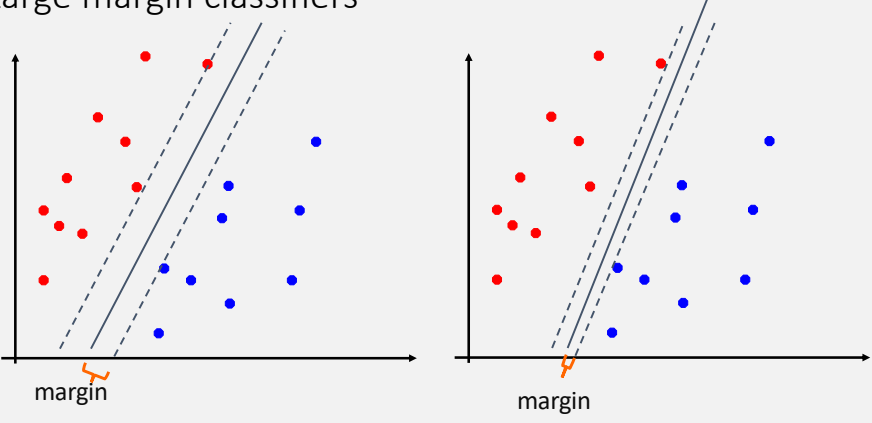
**PennState**  
Institute for Computational  
and Data Sciences

**Center for Artificial Intelligence Foundations & Scientific Applications**  
Artificial Intelligence Research Laboratory




**PennState**  
Clinical and Translational  
Science Institute

## Large margin classifiers




- The **margin** of a classifier is the distance to the closest points of either class
- Large margin classifiers attempt to maximize this




**PennState**  
College of Engineering  
Science and Technology

Data Science for Researchers and Scholars

Vasant Honavar, Fall 2023

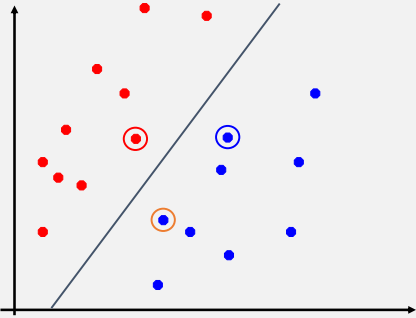
**PennState**  
Institute for Computational  
and Data Sciences

Center for Artificial Intelligence Foundations & Scientific Applications  
Artificial Intelligence Research Laboratory


**PennState**  
Clinical and Translational  
Science Institute

## Support vectors

- For any separating hyperplane, there exist some set of “closest points”
- These are called the support vectors because they fully specify the maximum margin separating hyperplane



A margin maximizing classifier was named the Support Vector Machine by Vapnik

**PennState**  
College of Engineering  
Science and Technology

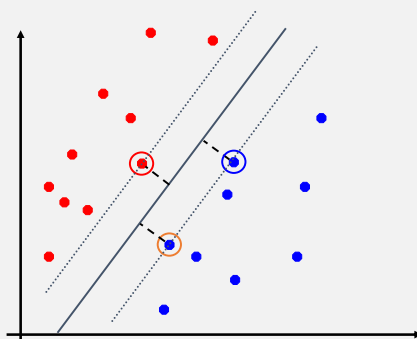
Data Science for Researchers and Scholars

Vasant Honavar, Fall 2023

- they are vectors (i.e. points) and they support/define the line

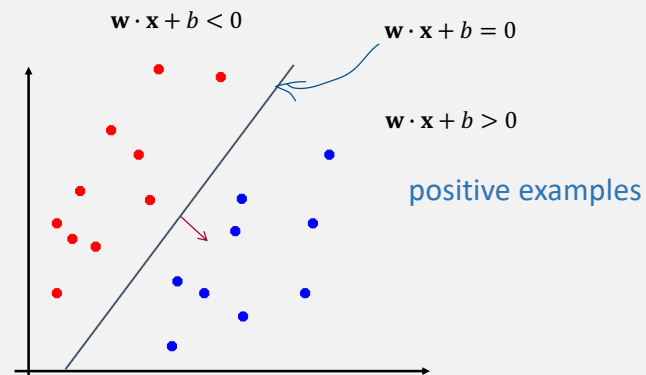
## Measuring the margin

The margin of a hyperplane is the distance to the support vectors, i.e. the closest points, on either side of the hyperplane

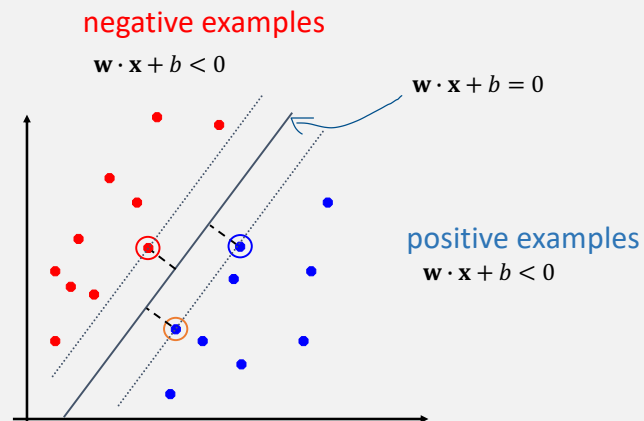


## Measuring the margin

negative examples



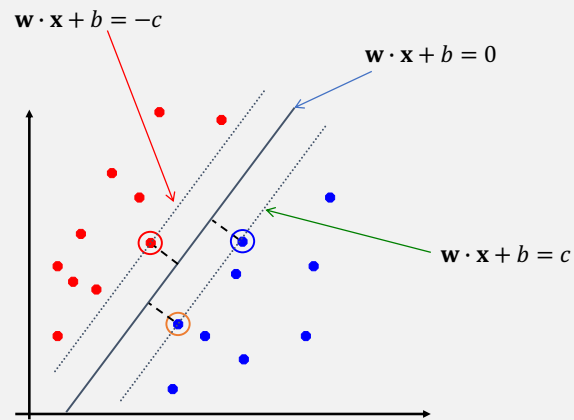
What defines the margin?



## Maximum margin hyperplane

Place the hyperplane equidistant from the nearest points of the two classes

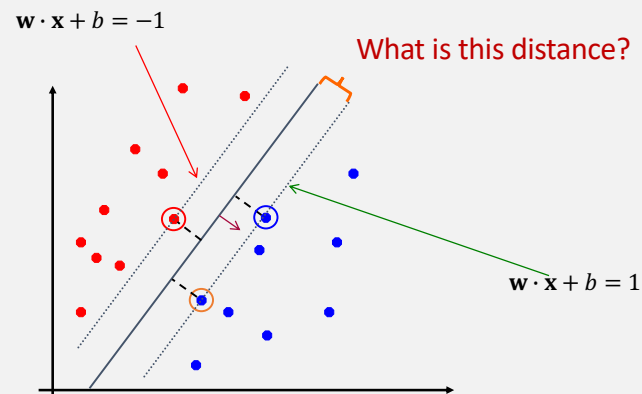
- What is  $c$ ?
- It depends!
- If we scale  $\mathbf{w}$ , we can vary  $c$  without changing the separating hyperplane!





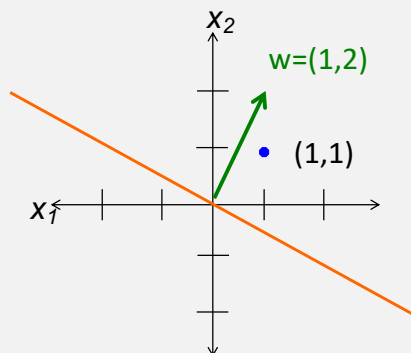
## Maximum margin hyperplane

- If we scale  $\mathbf{w}$ , we can vary  $c$  without changing the separating hyperplane!
- So how about if we choose  $c = 1$ ?



## Distance from the hyperplane

How far away is the point (1,1) from the hyperplane?

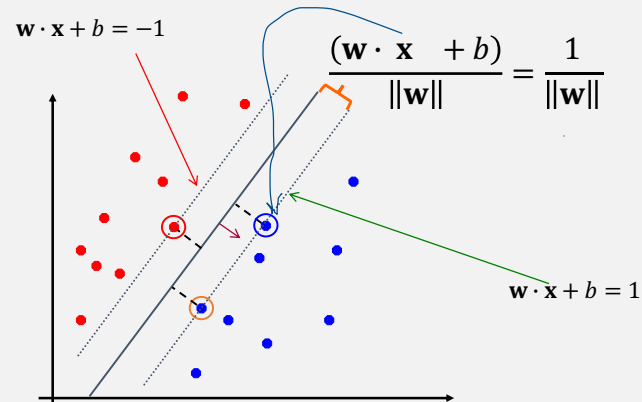



$$\frac{(w \cdot x_i + b)}{\|w\|} = \frac{(1)(1) + (1)(2) + 0}{\sqrt{1^2 + 2^2}} = 1.34$$

- Note that the magnitude of the length of the weight vector does not matter!
- Only its direction and the value of  $b$  do.

## Maximum margin hyperplane


- If we scale  $\mathbf{w}$ , we can vary  $c$  without changing the separating hyperplane!
- So how about if we choose  $c = 1$ ?





**PennState**  
Institute for Computational  
and Data Sciences

**Center for Artificial Intelligence Foundations & Scientific Applications**  
**Artificial Intelligence Research Laboratory**



**PennState**  
Clinical and Translational  
Science Institute

## Maximum margin classifier

Select, among all separating hyperplanes, one that correctly classifies the training samples with the largest margin over the training data


This yields a **constrained optimization problem**:

$$\max_{\mathbf{w}, b} \text{margin}(\mathbf{w}, b)$$

Subject to:

$$\forall p \quad d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \geq 1$$

Because  $\text{margin}(\mathbf{w}, b) = \frac{1}{\|\mathbf{w}\|}$ , maximizing the margin is equivalent to minimizing  $\|\mathbf{w}\|$  subject to the constraints  $\forall p \quad d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \geq 1$




**PennState**  
College of Engineering,  
Science and Technology

Data Science for Researchers and Scholars


Vasant Honavar, Fall 2023

All points are classified correctly AND they have a prediction  $\geq 1$ .



**PennState**  
Institute for Computational  
and Data Sciences

**Center for Artificial Intelligence Foundations & Scientific Applications**  
**Artificial Intelligence Research Laboratory**



**PennState**  
Clinical and Translational  
Science Institute

## Maximum margin classifier


Select, among all separating hyperplanes, i.e., one that correctly classifies the training samples with the largest geometric margin

- Because  $\text{margin}(\mathbf{w}, b) = \frac{1}{\|\mathbf{w}\|}$ , maximizing the margin is equivalent to minimizing  $\|\mathbf{w}\|$ , or equivalently,  $\|\mathbf{w}\|^2$  subject to the constraints  $\forall p \quad d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \geq 1$
- This yields the following constrained optimization problem:

$$\min_{\mathbf{w}, b} \|\mathbf{w}\|^2$$

Subject to:

$$\forall p \quad d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \geq 1$$



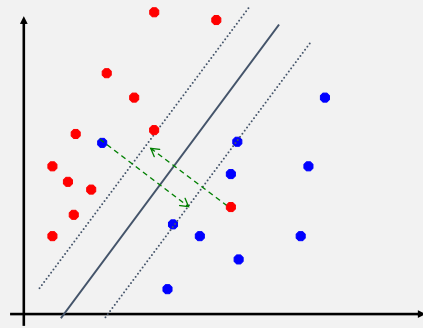
**PennState**  
College of Engineering,  
Science and Technology

Data Science for Researchers and Scholars

Vasant Honavar, Fall 2023

All points are classified correctly AND they have a prediction  $\geq 1$ .

## Accommodating misclassifications: Slack variables



$$\min_{\mathbf{w}, b} \|\mathbf{w}\|^2 + C \sum_p \zeta_p$$

Subject to:

$$\forall p \quad d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \geq 1 - \zeta_p$$

and  $\zeta_p \geq 0$

slack variables penalize misclassified samples

## How do slack variables help?

Maximize margin

trade-off between margin maximization and

$$\min_{\mathbf{w}, b} \|\mathbf{w}\|^2 + C \sum_p \zeta_p$$

Penalize by distance on wrong side of the margin

Subject to:

$$\forall p \quad d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \geq 1 - \zeta_p$$

and  $\zeta_p \geq 0$

Allow an occasional misclassification

## Understanding the Soft Margin SVM

$$\min_{\mathbf{w}, b} \left( \|\mathbf{w}\|^2 + C \sum_p \zeta_p \right)$$

Subject to:

$$\forall p \quad d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \geq 1 - \zeta_p$$

and  $\zeta_p \geq 0$

$$\zeta_p = \begin{cases} 0 & \text{if } d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \geq 1 \\ 1 - d_p(\mathbf{w} \cdot \mathbf{x}_p + b) & \text{otherwise} \end{cases}$$

In other words

$$\zeta_p = \max\{0, (1 - d_p(\mathbf{w} \cdot \mathbf{x}_p + b))\}$$



## Understanding the Soft Margin SVM

$$\min_{\mathbf{w}, b} \left( \|\mathbf{w}\|^2 + C \sum_p \zeta_p \right)$$

Subject to:

$$\forall p \quad d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \geq 1 - \zeta_p$$

and  $\zeta_p \geq 0$

The above implies

$$\zeta_p = \max \left\{ 0, \left( 1 - d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \right) \right\}$$

Do we still need the constraints?

Not really, because we can substitute  $\zeta_p$  into the objective function!

## Understanding the Soft Margin SVM

The result is an unconstrained optimization problem

$$\min_{\mathbf{w}, b} \left( \|\mathbf{w}\|^2 + C \sum_p \max \left\{ 0, \left( 1 - d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \right) \right\} \right)$$

## Solving the Soft Margin SVM

We want to find  $\mathbf{w}, b$  that minimize

$$E(\mathbf{w}, b) = \|\mathbf{w}\|^2 + C \sum_p \max\left\{0, \left(1 - d_p(\mathbf{w} \cdot \mathbf{x}_p + b)\right)\right\}$$

We note that this objective function is convex and is differentiable everywhere except where  $1 - d_p(\mathbf{w} \cdot \mathbf{x}_p + b) = 0$

So we can

- find the (sub) gradients with respect to  $\mathbf{w}$  and  $b$  and iteratively update them in the direction of their negative gradients

## Solving the Soft Margin SVM

We want to find  $\mathbf{w}, b$  that minimize

$$E(\mathbf{w}, b) = \|\mathbf{w}\|^2 + C \sum_p \max\{0, (1 - d_p(\mathbf{w} \cdot \mathbf{x}_p + b))\}$$

- When  $d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \geq 1$ 
  - the second term in  $E(\mathbf{w}, b)$  is 0
  - $\frac{\partial E(\mathbf{w}, b)}{\partial \mathbf{w}} = 2 \mathbf{w}$  and  $\frac{\partial E(\mathbf{w}, b)}{\partial b} = 0$
- When  $d_p(\mathbf{w} \cdot \mathbf{x}_p + b) < 1$ 
  - the second term in  $E(\mathbf{w}, b)$  is  $C \sum_p (1 - d_p(\mathbf{w} \cdot \mathbf{x}_p + b))$
  - $\frac{\partial E(\mathbf{w}, b)}{\partial \mathbf{w}} = 2 \mathbf{w} - C \sum_p d_p \mathbf{x}_p$  and  $\frac{\partial E(\mathbf{w}, b)}{\partial b} = -C \sum_p d_p$

## Solving the Soft Margin SVM

Combining the two cases, the weight update rule is

- $\mathbf{w} \leftarrow \mathbf{w} + \eta \left( I \left( d_p \left( \mathbf{w} \cdot \mathbf{x}_p + b \right) < 1 \right) \mathcal{C} \sum_p d_p \mathbf{x}_p - 2 \mathbf{w} \right)$
- $b \leftarrow b + \eta I \left( d_p \left( \mathbf{w} \cdot \mathbf{x}_p + b \right) < 1 \right) \mathcal{C} \sum_p d_p$

Where  $I(z)$  is the indicator function.  $I(z) = 1$  if  $z$  is True, and 0 otherwise.

- $\mathcal{C}$  is tuned using cross-validation.

## Remarks on the soft margin SVM

- Soft margin SVM finds the maximum margin classifier if the data are linearly separable
- Theory tells us that the maximum margin classifier will generalize optimally on unseen samples if the classes are linearly separable
- If the data are not linearly separable, soft margin SVM produces a classifier implements a good compromise
- We used the hinge loss, that is,  $\max \left\{ 0, \left( 1 - d_p \left( \mathbf{w} \cdot \mathbf{x}_p + b \right) \right) \right\}$  in our objective function
- We can replace it by a smooth differentiable approximation as we saw in the case of the perceptron and derive the update equations.
- Later we will generalize SVMs to incorporate kernel functions so they can be applied to data that do not necessarily live in a Euclidian space and/or may not be linearly separable

**PennState**  
Institute for Computational  
and Data Sciences

Center for Artificial Intelligence Foundations & Scientific Applications  
Artificial Intelligence Research Laboratory

**PennState**  
Clinical and Translational  
Science Institute

# Kernel Machines

**PennState**  
College of Engineering,  
Science and Technology

Data Science for Researchers and Scholars

Vasant Honavar, Fall 2023133

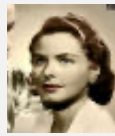
## What if the data are not linearly separable?

**Suppose samples from two classes are not** linearly separable in the most natural feature representation.

**Example**



vs



No good linear separator  
in pixel representation

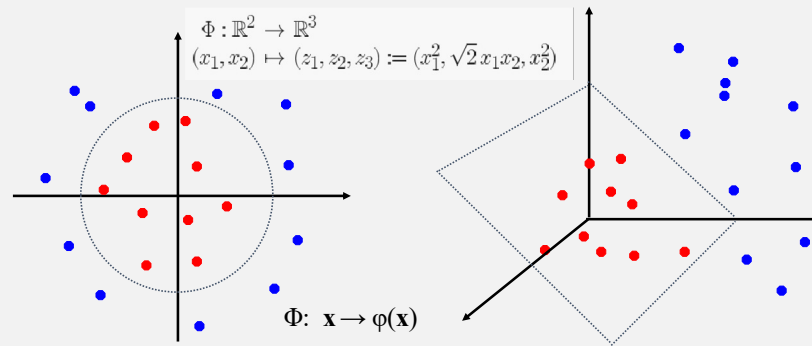
- **Classic solutions:**
  - Pick a suitably parameterized non-linear function, e.g., high order polynomial
  - Engineer features such that the data are likely to become separable in the feature space
  - Both approaches involve too much ad hoc trial and error
- **Modern solutions:**
  - Use kernel trick and maximize margin (or regularize)
  - Representation learning



## Dealing with data that are not linearly separable

### Idea:

- Map data samples from the original input space to some higher-dimensional feature space where they become separable and learn a separating hyperplane in the feature space



## Exclusive OR revisited

$$\varphi_1(\mathbf{x}) = \varphi_1(x_1, x_2) = e^{\|\mathbf{x} - \mathbf{w}_1\|^2} = z_1$$

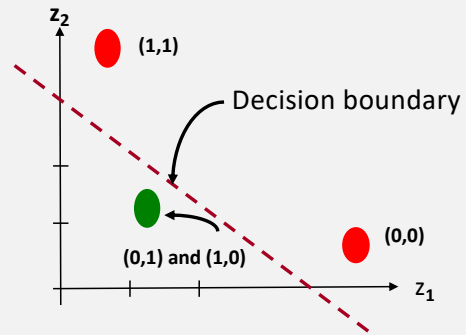
$$\varphi_2(\mathbf{x}) = \varphi_2(x_1, x_2) = e^{\|\mathbf{x} - \mathbf{w}_2\|^2} = z_2$$

$$\mathbf{w}_1 = [1, 1]^T$$

$$\mathbf{w}_2 = [0, 0]^T$$

- In feature space  $[z_1, z_2]$  the two classes 2 become linearly separable.

$\mathbf{x}$	$z_1$	$z_2$
00	$e^2$	1
01	$e^1$	$e^1$
10	$e^1$	$e^1$
11	1	$e^2$



- So EXOR is learnable in the feature space  $[z_1, z_2]$

## Dealing with non-separable data

- Map input data to feature space where the classes become separable
  - The resulting feature space often is of a higher dimension than the original input space
  - Sometimes, linear separability requires the feature space to be infinite dimensional
- Learn a separating hyperplane in the feature space

### Challenges

- How to cope with a high dimensional, perhaps even infinite dimensional feature space (how do you compute  $\mathbf{w} \cdot \mathbf{x}$  when the two vectors are infinite dimensional?)
- How to ensure good generalization on samples not present in the training data ?

## Dealing with non-separable data

### Challenges

- How to cope with high dimensional, perhaps even infinite dimensional feature space (How do you compute  $\mathbf{w} \cdot \mathbf{x}$  when the two vectors are infinite dimensional?)
  - **Kernel trick:** Allows dot product in the feature space to be computed using dot product in the input space
- How to ensure good generalization on samples not present in the training data ?
  - **Maximum margin separating hyperplane:** Find a separating hyperplane that maximizes the margin of separation between the classes in the kernel induced feature space
- These two ideas came together for the first time in support vector machines, and revolutionized machine learning



Aizerman

## Kernel trick

- All of the algorithms we have considered for learning a separating hyperplane, e.g., perceptron, work by adding or subtracting (depending on the sign of the difference between the desired and actual output) a misclassified sample to an arbitrary weight vector.
- The final weight vector  $\mathbf{w} = [w_1, \dots, w_N]^T$  is a linear combination of training samples

$$\mathbf{w} = \sum_{p=1}^P \alpha_p d_p \mathbf{x}_p$$

- The  $\alpha_p$  are positive coefficients, proportional to the number of times the misclassification of  $\mathbf{x}_p$  has caused the weight vector  $\mathbf{w}$  to be updated, and  $d_p$  the sign of the net contribution of  $\mathbf{x}_p$  to the weight update.
- **Key observation:**  $\mathbf{w}$  lies in the space spanned by the training samples



Aizerman

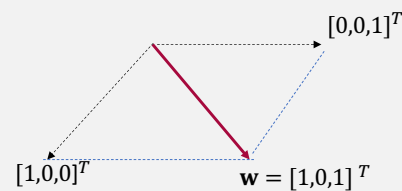
## Kernel trick

- The final weight vector  $\mathbf{w} = [w_1, \dots, w_N]^T$  is a linear combination of training samples

$$\mathbf{w} = \sum_{p=1}^P \alpha_p d_p \mathbf{x}_p$$

- Key observation:**  $\mathbf{w}$  lies in the space spanned by the training samples

$$\mathbf{w} = 1 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + 1 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$



## Kernel trick



Aizerman

- What is a kernel?
  - $K: D_{\mathbf{x}} \times D_{\mathbf{x}} \rightarrow \mathfrak{R}$  is a kernel function if there exists an implicit mapping  $\varphi$  such that  $K(\mathbf{x}_p, \mathbf{x}_q) = \varphi(\mathbf{x}_p) \cdot \varphi(\mathbf{x}_q)$
  - That is, a kernel function implicitly defines the dot product between two samples in a (kernel induced) feature space.
  - Hence, we can get linear machines to operate in a kernel induced feature space by replacing the pairwise dot product between data samples in the input space by  $K(\mathbf{x}_p, \mathbf{x}_q)$

## Kernel induced feature space

$$\mathbf{x}_p = [x_{1p} \ x_{2p}]^T$$

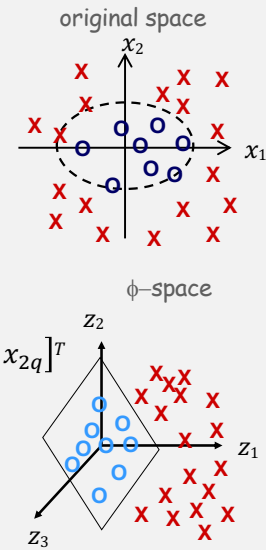
$$\mathbf{x}_q = [x_{1q} \ x_{2q}]^T$$

Define

$$\begin{aligned} K(\mathbf{x}_p, \mathbf{x}_q) &= (\mathbf{x}_p \cdot \mathbf{x}_q)^2 \\ &= (x_{1p}x_{1q} + x_{2p}x_{2q})^2 \\ &= x_{1p}^2 x_{1q}^2 + x_{2p}^2 x_{2q}^2 + 2 x_{1p}x_{1q} x_{2p}x_{2q} \\ &= [x_{1p}^2 \ x_{2p}^2 \ \sqrt{2}x_{1p}x_{2p}]^T \cdot [x_{1q}^2 \ x_{2q}^2 \ \sqrt{2}x_{1q}x_{2q}]^T \\ &= \varphi(\mathbf{x}_p) \cdot \varphi(\mathbf{x}_q) \end{aligned}$$

Kernel induced feature space

$$(x_1, x_2) \rightarrow \varphi(\mathbf{x}) = (x_1^2 \ x_2^2 \ \sqrt{2}x_1x_2)$$





## The magical beauty of Kernel functions

- $K(\mathbf{x}_p, \mathbf{x}_q)$  is expressed as a function of the dot product  $\mathbf{x}_p \cdot \mathbf{x}_q$  in the input space
- Yet it implicitly yields the dot product between the images of  $\mathbf{x}_p$  and  $\mathbf{x}_q$  in the feature space  $\varphi$

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

- Thus, the kernel function  $K(\mathbf{x}_p, \mathbf{x}_q)$  makes it possible to compute the dot product  $\varphi(\mathbf{x}_p) \cdot \varphi(\mathbf{x}_q)$  between high-dimensional, even infinite dimensional feature vectors efficiently using the dot product  $\mathbf{x}_p \cdot \mathbf{x}_q$  in the low, finite dimensional input space
- Given a function  $K$ , it is possible to verify that it is a kernel function (we will return to this later).

## Kernel trick

- Kernel trick introduced above can be generalized so as to “kernelize” any machine learning algorithm that uses linear model for classification or regression
  - Support vector machine
  - Linear regression
  - Logistic regression
  - ..... and many others
- **Key idea:** the weight vector learned by linear classifiers, e.g., perceptron, SVM, lies in the space spanned by the data samples

## Kernelizing a broad class of loss functions

- We will introduce an approach to kernelizing any loss function that is expressed as a function of the dot product of  $\mathbf{w}$  and  $\mathbf{x}$
- **Key idea:** the weight vector learned by linear classifiers, e.g., perceptron, SVM, lies in the space spanned by the data samples
- This treatment has several advantages over the standard treatment of kernel trick in SVM
  - It does not require us to get into constrained convex optimization
  - It is very general and can be used to kernelize a broad class of loss functions used in machine learning for classification, regression, dimensionality reduction, and clustering

## Kernelizing two-class Perceptron with softmax loss

- Let  $\varphi_p = [\varphi_1(\mathbf{x}_p) \varphi_2(\mathbf{x}_p) \cdots \varphi_M(\mathbf{x}_p)]^T$  be the kernel-induced feature representation of sample  $\mathbf{x}_p$
- Let  $\mathbf{w} = [w_1 w_2 \cdots w_M]^T$  be the corresponding weight vector
- Recall  $\mathbf{w} \cdot \varphi_p + b = \varphi_p^T \mathbf{w} + b$ .
- So the perceptron loss in the kernel induced feature space is

$$E_{soft}(\mathbf{w}, b) = \sum_p \max\{0, -d_p(\varphi_p^T \mathbf{w} + b)\}$$

$$E_{smooth}(\mathbf{w}, b) = \sum_p \log \left\{ e^0 + e^{-(\varphi_p^T \mathbf{w} + b)d_p} \right\} = \sum_p \log \left\{ 1 + e^{-(\varphi_p^T \mathbf{w} + b)d_p} \right\}$$

- Let  $\Phi = [\varphi_1 \varphi_2 \cdots \varphi_P]$  ( $M \times P$  matrix formed by stacking the kernel-induced feature vectors for the  $P$  samples)

## Kernelizing two-class Perceptron with softmax loss

We can show (using some basic results of linear algebra) that minimizing  $E_{smooth}(\mathbf{w}, b)$  is equivalent to minimizing

$$E_{smooth}(\mathbf{z}, b) = \sum_p \log \left( 1 + e^{-d_p(\mathbf{k}_p \cdot \mathbf{z} + b)} \right)$$

where  $\mathbf{k}_p$  is the  $p$ th column of the  $P \times P$  kernel matrix.

- The loss function is independent of the dimensionality of the kernel-induced feature space
- The weight vector  $\mathbf{z}$  has as many components as there are training samples
- In practice, we add a regularization term,  $\frac{\lambda}{2} \|\mathbf{w}\|^2 = \frac{\lambda}{2} \mathbf{z}^T \mathbf{K} \mathbf{z}$

$$E_{smooth}^R(\mathbf{z}, b) = \nabla_{\mathbf{z}} \sum_p \log \left( 1 + e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)} \right) + \frac{\lambda}{2} \mathbf{z}^T \mathbf{K} \mathbf{z}$$

## Minimizing the Kernelized two-class Perceptron with softmax loss

$$\begin{aligned}\nabla_{\mathbf{z}} E_{smooth}^R(\mathbf{z}, b) &= \nabla_{\mathbf{z}} \sum_{p=1}^P \log \left( 1 + e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)} \right) + \nabla_{\mathbf{z}} \left( \frac{\lambda}{2} \mathbf{z}^T \mathbf{K} \mathbf{z} \right) \\ \nabla_{\mathbf{z}} E_{smooth}^R(\mathbf{z}, b) &= \sum_{p=1}^P \frac{1}{(1 + e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)})} \nabla_{\mathbf{z}} \left( 1 + e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)} \right) + \frac{\lambda}{2} \nabla_{\mathbf{z}} (\mathbf{z}^T \mathbf{K} \mathbf{z}) \\ &= \sum_{p=1}^P \frac{1}{(1 + e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)})} \left( 0 + e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)} \right) \nabla_{\mathbf{z}} \left( -d_p(\mathbf{k}_p^T \mathbf{z} + b) \right) + \frac{\lambda}{2} 2 \mathbf{K} \mathbf{z} \\ &= - \sum_{p=1}^P \left( \frac{e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)}}{1 + e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)}} \right) (d_p \mathbf{k}_p) + \lambda \mathbf{K} \mathbf{z}\end{aligned}$$

$$\mathbf{z} \leftarrow \mathbf{z} - \eta \nabla_{\mathbf{z}} E_{smooth}^R(\mathbf{z}, b)$$

Note:  $\mathbf{z}$  is a  $P \times 1$  matrix (column vector), and so are  $\mathbf{k}_p$  and  $\mathbf{K} \mathbf{z}$

## Minimizing the Kernelized two-class Perceptron with softmax loss

$$\begin{aligned}
 E_{smooth}^R(\mathbf{z}, b) &= \sum_{p=1}^P \log \left( 1 + e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)} \right) + \left( \frac{\lambda}{2} \mathbf{z}^T \mathbf{K} \mathbf{z} \right) \\
 \nabla_b E_{smooth}^R(\mathbf{z}, b) &= \nabla_b \sum_{p=1}^P \log \left( 1 + e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)} \right) + \nabla_b \left( \frac{\lambda}{2} \mathbf{z}^T \mathbf{K} \mathbf{z} \right) \\
 &= \sum_{p=1}^P \left( \frac{e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)}}{1 + e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)}} \right) \nabla_b \left( -d_p(\mathbf{k}_p^T \mathbf{z} + b) \right) + 0 \\
 &= - \sum_{p=1}^P \left( \frac{e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)}}{1 + e^{-d_p(\mathbf{k}_p^T \mathbf{z} + b)}} \right) d_p \\
 b &\leftarrow b - \eta \nabla_b E_{smooth}^R(\mathbf{z}, b)
 \end{aligned}$$

## How about Kernel SVM?

We use the kernel trick

- Wherever we have the dot product of the feature vector for the  $p$ th sample with weight vector  $N$ -dimensional  $\mathbf{w}$ , we replace it with the dot product of the  $p$ th column of the kernel matrix with a different  $P$ -dimensional weight vector  $\mathbf{z}$
- Note that
  - $N$  is the number of features
  - $P$  is the number of training samples



## Recall the SVM in the input space (no kernel)

The problem was to find  $\mathbf{w}, b$  that minimize

$$E(\mathbf{w}, b) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{p=1}^P \max \left\{ 0, \left( 1 - d_p(\mathbf{w} \cdot \mathbf{x}_p + b) \right) \right\}$$

Recall that  $\max\{a, b\} \approx \log(e^a + e^b)$

$$E(\mathbf{w}, b) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{p=1}^P \log \left( e^0 + e^{(1 - d_p(\mathbf{w} \cdot \mathbf{x}_p + b))} \right)$$

$$E(\mathbf{w}, b) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{p=1}^P \log \left( 1 + e^{(1 - d_p(\mathbf{w} \cdot \mathbf{x}_p + b))} \right)$$

When we introduce the kernel,  $\mathbf{w} \cdot \mathbf{x}_p$  is replaced by  $\mathbf{w} \cdot \varphi(\mathbf{x}_p)$  where  $\varphi(\mathbf{x}_p)$  is the kernel-induced feature space.

## Soft margin Kernel SVM

$$E(\mathbf{w}, b) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_p \max \left\{ 0, \left( 1 - d_p(\varphi_p^T \mathbf{w} + b) \right) \right\}$$

$$\approx \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{p=1}^P \log \left( e^0 + e^{(1 - d_p(\varphi_p^T \mathbf{w} + b))} \right)$$

From kernel trick, we have:

$$E(\mathbf{z}, b) \approx \frac{1}{2} \mathbf{z}^T \mathbf{K} \mathbf{z} + C \sum_{p=1}^P \log_p \left( 1 + e^{(1 - d_p(\mathbf{k}_p^T \mathbf{z} + b))} \right)$$

$$\nabla_{\mathbf{z}} E(\mathbf{z}, b) = \mathbf{K} \mathbf{z} + C \sum_{p=1}^P \nabla_{\mathbf{z}} \left( \log \left( 1 + e^{(1 - d_p(\mathbf{k}_p^T \mathbf{z} + b))} \right) \right)$$

$$= \mathbf{K} \mathbf{z} - \sum_{p=1}^P \left( \frac{C e^{(1 - d_p(\mathbf{k}_p^T \mathbf{z} + b))}}{1 + e^{(1 - d_p(\mathbf{k}_p^T \mathbf{z} + b))}} \right) (d_p \mathbf{k}_p)$$

$$\mathbf{z} \leftarrow \mathbf{z} - \nabla_{\mathbf{z}} E(\mathbf{z}, b)$$

## Soft margin Kernel SVM

$$E(\mathbf{z}, b) \approx \frac{1}{2} \mathbf{z}^T \mathbf{K} \mathbf{z} + C \sum_{p=1}^P \log \left( 1 + e^{(1-d_p(\mathbf{k}_p^T \mathbf{z} + b))} \right)$$

$$\nabla_b E(\mathbf{z}, b) = 0 + \nabla_b \left( C \sum_{p=1}^P \log \left( 1 + e^{(1-d_p(\mathbf{k}_p^T \mathbf{z} + b))} \right) \right)$$

$$= - \sum_{p=1}^P \left( \frac{C e^{(1-d_p(\mathbf{k}_p^T \mathbf{z} + b))}}{1 + e^{(1-d_p(\mathbf{k}_p^T \mathbf{z} + b))}} d_p \right)$$

$$b \leftarrow b - \nabla_b E(\mathbf{z}, b)$$

## The power of kernels

- Kernels allow us to learn non-linear decision or regression surfaces in the input space which correspond to linear surfaces in kernel-induced feature space
- Kernel trick allows us to generalize machine learning methods for classification and regression designed for data that live in fixed dimensional vector input spaces to work with arbitrary data – sequences, graphs, documents ...
- Kernels provide a means of injecting domain knowledge (useful notions of similarity) into predictive models trained using machine learning
- Kernelization can be used to upgrade any linear model for classification or regression to work with kernel-induced feature spaces
- The resulting loss functions are independent of the dimensionality of the feature space – allows working with even infinite dimensional feature spaces
- Generalization in high-dimensional kernel induced feature spaces requires regularization (e.g., maximizing margin in the case of SVM)

## The Kernel Matrix

Kernel matrix is a  $P \times P$  matrix of pair-wise dot products between kernel-induced feature vectors that encode the training samples

$$\mathbf{K} = \begin{array}{|c|c|c|c|c|} \hline K(1,1) & K(1,2) & K(1,3) & \dots & K(1,P) \\ \hline K(2,1) & K(2,2) & K(2,3) & \dots & K(2,P) \\ \hline & & & & \\ \hline \dots & \dots & \dots & \dots & \dots \\ \hline K(l,1) & K(l,2) & K(l,3) & \dots & K(P,P) \\ \hline \end{array}$$

## Properties of Kernel Matrices

It is easy to show that the kernel matrix is

- A Square matrix
- Symmetric ( $\mathbf{K}^T = \mathbf{K}$ )
- Positive semi-definite (all eigenvalues of  $\mathbf{K}$  are non-negative
  - Recall that Eigen values of a square matrix  $\mathbf{A}$  are given by values of  $\lambda$  that satisfy  $|\mathbf{K} - \lambda \mathbf{I}| = 0$ )

Any symmetric positive semi definite matrix can be regarded as a kernel matrix, that is, as an inner product matrix in some feature space  $\Phi$ .

## Mercer's Theorem: Characterization of Kernel Functions

A function  $K : D_{\mathbf{x}} \times D_{\mathbf{x}} \rightarrow \Re$  is said to be (finitely) positive semi-definite if

- $K$  is a symmetric function:  $K(\mathbf{x}_p, \mathbf{x}_q) = K(\mathbf{x}_q, \mathbf{x}_p)$
- Matrices formed by restricting  $K$  to any finite subset of the domain  $D_{\mathbf{x}}$  are positive semi-definite

### Characterization of Kernel Functions

Every (finitely) positive semi definite, symmetric function is a kernel: i.e., there exists a mapping  $\varphi$  such that it is possible to write:  $K(\mathbf{x}_p, \mathbf{x}_q) = \varphi(\mathbf{x}_p) \cdot \varphi(\mathbf{x}_q)$

## Modularity of Kernels

The set of kernels is closed under some operations. If

$K_1, K_2$  are kernels over  $\mathcal{X} \times \mathcal{X}$ , then the following are kernels:

$$K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z}) + K_2(\mathbf{x}, \mathbf{z})$$

$$K(\mathbf{x}, \mathbf{z}) = aK_1(\mathbf{x}, \mathbf{z}), \quad a > 0$$

$$K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z})K_2(\mathbf{x}, \mathbf{z})$$

$$K(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z}); \quad f: \mathcal{X} \rightarrow \mathbb{R}$$

$$K(\mathbf{x}, \mathbf{z}) = K_3(\phi(\mathbf{x}), \phi(\mathbf{z}));$$

$$(\phi: \mathcal{X} \rightarrow \mathbb{R}^N; K_3 \text{ is a kernel over } \mathbb{R}^N \times \mathbb{R}^N)$$

$$K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{B} \mathbf{z};$$

$$(\mathbf{B} \text{ is a symmetric positive definite } n \times n \text{ matrix and } \mathcal{X} \subseteq \mathbb{R}^n)$$

We can make complex kernels from simple ones: modularity!



## Kernels for different types of data

- We can define Kernels over arbitrary instance spaces including
  - Finite dimensional vector spaces,
  - Boolean spaces
  - $\Sigma^*$  where  $\Sigma$  is a finite alphabet
  - Documents, graphs, molecular structures, etc.
- Kernels need not always be expressed by a closed form formula.
- Many useful kernels can be computed by complex algorithms (e.g., diffusion kernels over graphs)
- This allows machine learning methods designed for data encoded by fixed dimensional feature vectors to be upgraded to work with arbitrary data types – sequences, graphs, etc.

## Kernels on Sets and Multi-Sets

Let  $X \subseteq 2^V$  for some fixed domain  $V$

Let  $A_1, A_2 \in X$

Then  $K(A_1, A_2) = 2^{|A_1 \cap A_2|}$  is a kernel.

Example:

$$V = \{A, B, C, D, E\}$$

$$S_1 = \{A, B, C, D\}, S_2 = \{B, C, D, E\}$$

$$S_1 \cap S_2 = \{B, C, D\}$$

$$K(S_1, S_2) = 2^3 = 8$$

Exercise: Define a Kernel on the space of multi-sets  
whose elements are drawn from a finite domain  $V$

## String Kernel ( $p$ -spectrum Kernel)

- The  $p$ -spectrum of a string is the histogram – vector of number of occurrences of all possible contiguous substrings – of length  $p$
- We can define a kernel function  $K(s, t)$  over  $\Sigma^* \times \Sigma^*$  as the inner product of the  $p$ -spectra of  $s$  and  $t$ .


$s = \text{statistics}$

$t = \text{computation}$

$p = 3$


Common substrings:  $tat, ati$

$K(s, t) = 2$



**PennState**  
Institute for Computational  
and Data Sciences

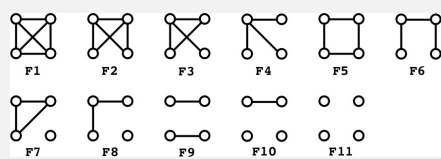
**Center for Artificial Intelligence Foundations & Scientific Applications**  
Artificial Intelligence Research Laboratory



**PennState**  
Clinical and Translational  
Science Institute

## Graphlet kernel for graphs

- Count the number of occurrences of each graphlet (subgraph with a specified structure) of a given size (4 in the example) in graphs
- Constructs a vector from the histogram of graphlets of size  $k$  in each of the graphs  $G_1 \dots G_M$
- Normalize the histograms  $D_{G_1} \dots D_{G_M}$
- $K(G_i, G_j) = \cos \theta(D_{G_i}, D_{G_j})$



All Graphlets of size 4


↓

[15, 17, 1, 85, 9, 4, 3, 5, 7, 2, 99]

↓ normalize

[0.06, 0.07, 0, 0.34, 0.04, 0.02, 0.01, 0.02, 0.03, 0.01, 0.40]

$D_{G_i}$



**PennState**  
College of Engineering,  
Science and Technology

Data Science for Researchers and Scholars

Vasant Honavar, Fall 2023

## How to design good kernels?

- The purpose of a kernel function is to map data into a suitable kernel induced feature space where it is easier to learn than in the original space
- How can we design good kernels?
  - Kernel function  $K(\mathbf{x}_p, \mathbf{x}_q) = \varphi(\mathbf{x}_p) \cdot \varphi(\mathbf{x}_q)$  is a measure of similarity between pairs of data samples
  - We can inject domain knowledge into a kernel function
  - There is no algorithm that can provide us an optimal kernel for any given problem or data set
- Can we tell if a proposed kernel is a good kernel?
  - Examine the Kernel matrix
    - Is it mostly diagonal (non-zero entries only along the diagonal and zeros everywhere else?)
      - Then the kernel does not provide any useful notion of similarity that the machine learning algorithm can exploit

## Kernels – the good, the bad, and the ugly

- **Bad kernel** – A kernel when applied to the data set, yields a kernel matrix that is mostly diagonal
  - No data sample is similar to any other!
- In mapping in a space with too many irrelevant features, kernel matrix becomes diagonal
  - Need some prior knowledge of target so choose a good kernel
- A diagonal kernel matrix implies that there is no regularity to be exploited by the learning algorithm

1	0	0	...	0
0	1	0	...	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

## How to craft good kernel functions for specific applications?

- There is no general algorithm for designing an optimal kernel function for a given data set or machine learning problem
- However, it is easy to determine whether a given kernel is a reasonable kernel for a given data set
  - Examine the Kernel matrix
    - Does the kernel matrix exhibit a block diagonal structure where the blocks define subsets of data that are similar and share class labels

## Kernels – the good, the bad, and the ugly

- Good kernel – Corresponds to a Gram (kernel) matrix in which subsets of data points belonging to the same class are similar to each other, and hence the machine can detect hidden structure in the data

3	2	0	0	0
2	3	0	0	0
0	0	4	3	3
0	0	3	4	2
0	0	3	2	4

 Class 1

 Class 2



## The power of kernels

- Kernels allow us to learn non-linear decision or regression surfaces in the input space which correspond to linear surfaces in kernel-induced feature space
- Kernel trick allows us to generalize machine learning methods for classification and regression designed for data that live in fixed dimensional vector input spaces to work with arbitrary data – sequences, graphs, documents ...
- Kernels provide a means of injecting domain knowledge (useful notions of similarity) into predictive models trained using machine learning
- Kernelization can be used to upgrade any linear model for classification or regression to work with kernel-induced feature spaces
- The resulting loss functions are independent of the dimensionality of the feature space – allows working with even infinite dimensional feature spaces
- Generalization in high-dimensional kernel induced feature spaces requires regularization (e.g., maximizing margin in the case of SVM)

## Kernel Trick

- Map input data to a high dimensional feature space learning becomes easy

### Challenges

- How to cope with high dimensional, perhaps even infinite dimensional feature space
  - How do you compute the dot product between weights and features when the kernel induced feature space is high dimensional?
  - Use the Kernel trick which makes the dimensionality of the weights independent of the dimensionality of the feature space
- How to ensure good generalization on samples not present in the training data?
  - Regularize the weights in the kernel induced feature space