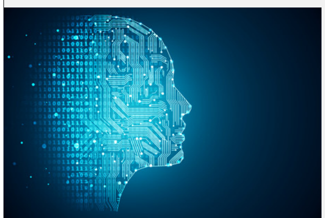


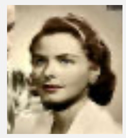
# DS 310 Machine Learning
# Kernel Machines

**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
http://faculty.ist.psu.edu/vhonavar
http://ailab.ist.psu.edu

## What if the data are not linearly separable?

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

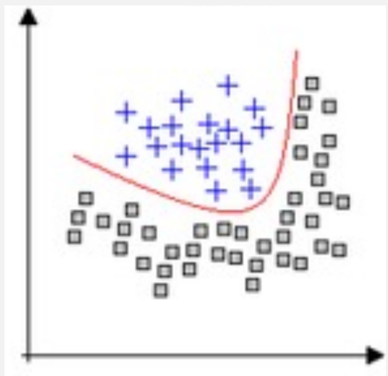
**Example**  vs  No good linear separator in pixel representation

- Classic solutions:
  - Pick a suitably parameterized non-linear function
  - 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

# Classifier engineering

Idea:

- Try out nonlinear decision boundaries, e.g., $\mathbf{w}^T\mathbf{x} + \mathbf{x}^T\mathbf{B}\,\mathbf{x} + b = 0$

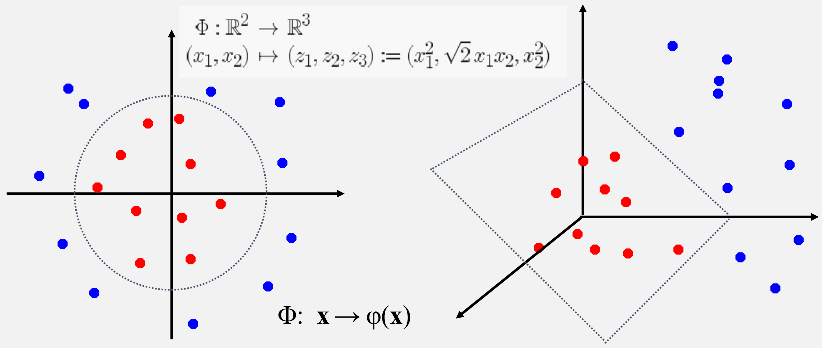


# Dealing with data that are not linearly separable

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

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$



$\Phi$: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$

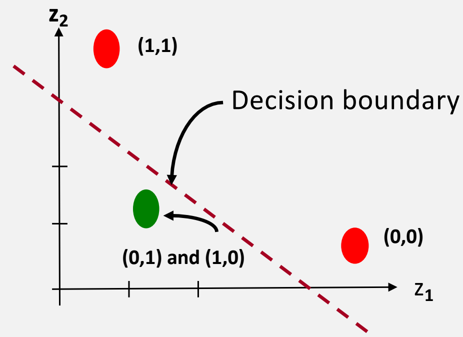
# Exclusive OR revisited

$\mathbf{w}_1=[1,1]^T$
$\mathbf{w}_2=[0,0]^T$

$$\varphi_1(\mathbf{x}) = \varphi_1(x_1, x_2) = e^{\|\mathbf{x}-\mathbf{w}_1\|^2} = z_1$$
$$\varphi_2(\mathbf{x}) = \varphi_2(x_1, x_2) = e^{\|\mathbf{x}-\mathbf{w}_2\|^2} = z_2$$

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

| **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

## Kernel trick


Aizerman

- 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 $\mathbf{w} = [\, w_1, \cdots w_N \,]^T$ 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

# Kernel trick

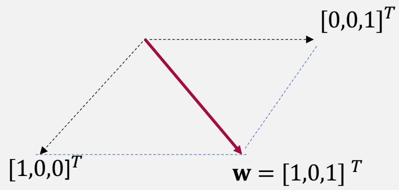
Aizerman

- The final weight vector $\mathbf{w} = [\, w_1, \cdots w_N \,]^T$ 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?
  - $K: D_{\mathbf{x}} \times D_{\mathbf{x}} \rightarrow \Re$ is a kernel function if there exists an implicit mapping $\varphi$ such that $K\left(\mathbf{x}_p, \mathbf{x}_q\right) = \varphi\left(\mathbf{x}_p\right) \cdot \varphi\left(\mathbf{x}_q\right)$
  - 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 $K\left(\mathbf{x}_p, \mathbf{x}_q\right)$

# Kernel induced feature space

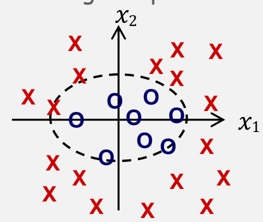
$\mathbf{x}_p = [x_{1p}\ x_{2p}]^T$

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

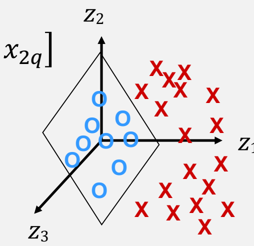
Define

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

original space




ϕ–space




Kernel induced feature space

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

## The magical beauty of Kernel functions

- $K\left(\mathbf{x}_p, \mathbf{x}_q\right)$ 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 $\varphi$

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

- Thus, the kernel function $K\left(\mathbf{x}_p, \mathbf{x}_q\right)$ makes it possible to compute the dot product $\varphi\left(\mathbf{x}_p\right) \cdot \varphi\left(\mathbf{x}_q\right)$ 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

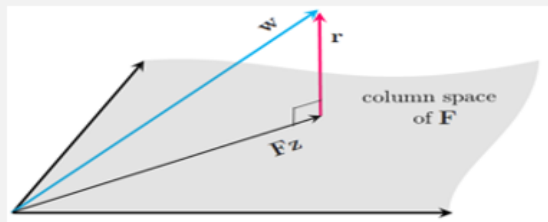
- Kernel trick introduced above can be generalized so as to "kernelize" any machine learning algorithm that uses linear model for classification or regression
  - Support vector machine
  - Linear regression
  - …… and many others
- Key idea: the weight vector learned by linear classifiers, e.g., perceptron, 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 relies on only elementary matrix algebra
  - 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

# A Proposition of the Fundamental Theorem of Linear Algebra

- Consider the decomposition of an $M$-dimensional vector $\mathbf{w}$ over the column space of an $M \times P$ matrix $\mathbf{F}$
- Let $\mathbf{f}_p$ be the $p$th column of $\mathbf{F}$
- Then if $\mathbf{w}$ lies within the column space of $\mathbf{F}$, then
    - $\mathbf{w}$ can be expressed as a linear combination of the column vectors of $\mathbf{F}$
    - $\mathbf{w} = \sum_{p=1}^{P} \mathbf{f}_p z_p$ where $z_p$ denotes the linear coefficient associated with $\mathbf{f}_p$



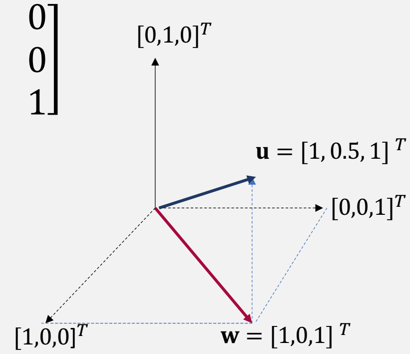

- Let $\mathbf{z} = [z_1 \; z_2 \cdots z_p]^T$
- Then we can write $\mathbf{w} = \mathbf{Fz}$
- If $\mathbf{w}$ lies outside the column space of $\mathbf{F}$, we can decompose it as $\mathbf{w} = \mathbf{Fz} + \mathbf{r}$

# Example

$$\mathbf{F} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

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

$$\mathbf{u} = 1\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + 1\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + 0.5\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$



# 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

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

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

- Let $\mathbf{\Phi} = [\boldsymbol{\varphi}_1 \boldsymbol{\varphi}_2 \cdots \boldsymbol{\varphi}_P]$ ($M \times P$ matrix formed by stacking the kernel-induced feature vectors for the $P$ samples)
- From the preceding proposition of the fundamental theorem of linear algebra, we can write $\mathbf{w} = \ \mathbf{\Phi}\ \mathbf{z} + \mathbf{r}$ where $\mathbf{\Phi}^{\ T}\mathbf{r} = \mathbf{0}_{P \times 1}$ (because r is orthogonal to the space spanned by the columns of $\mathbf{\Phi}$)

# Kernelizing two-class Perceptron with softmax loss

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

$$\mathbf{w} = \ \mathbf{\Phi}\, \mathbf{z} + \mathbf{r} \text{ where } \mathbf{\Phi}^{\,T}\mathbf{r} = \mathbf{0}_{P\times 1}$$

$$E_{smooth}(\mathbf{z}, b) = \sum_p \log \quad \left(1 + e^{-d_p(\boldsymbol{\varphi}_p^T\, (\mathbf{\Phi}\, \mathbf{z}+\mathbf{r})+b)}\right)$$

- Let $\mathbf{K} = \mathbf{\Phi}^T\mathbf{\Phi}$, the $P{\times}P$ kernel matrix ($P \times M$ matrix multiplied by $M \times P$ matrix yielding a $P{\times}P$ matrix)
- Then $\mathbf{k}_p = \mathbf{\Phi}^T \boldsymbol{\varphi}_p$ ($P \times M$ matrix multiplied by $M \times 1$ matrix which gives a $P \times 1$ matrix) is the $p$th column of $\mathbf{K}$
- Hence $\boldsymbol{\varphi}_p^T\, \mathbf{\Phi} = \mathbf{k}_p^T$ ($1 \times P$ matrix)
- Note: Dimensions work out in matrix products

$$\therefore E_{smooth}(\mathbf{z}, b) = \sum_p \log \quad \left(1 + e^{-d_p(\boldsymbol{\varphi}_p^T\, (\mathbf{\Phi}\, \mathbf{z}+\mathbf{r})+b)}\right)$$

# Kernelizing two-class Perceptron with softmax loss

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

Note:

- The loss function is independent of the d imensionality of the kernel-induced feature space
- The weight vector $\mathbf{z}$ has as many components as there are training samples
- In practice, we want to add a regularization term,

e.g., $\frac{\lambda}{2}\|\mathbf{w}\|^2 = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} = \frac{\lambda}{2}(\mathbf{\Phi}\,\boldsymbol{z})^T\,\mathbf{\Phi}\,\boldsymbol{z} = \frac{\lambda}{2}\boldsymbol{z}^T\,\mathbf{\Phi}^T\mathbf{\Phi}\,\boldsymbol{z} = \frac{\lambda}{2}\boldsymbol{z}^T\,\mathbf{K}\,\boldsymbol{z}$

where we have used $\|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w}$ , $(\mathbf{\Phi}\,\boldsymbol{z})^T = \boldsymbol{z}^T\,\mathbf{\Phi}^T$ and $\mathbf{\Phi}^T\mathbf{\Phi} = \mathbf{K}$

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

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

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

$$\nabla_z E^R_{smooth}(\mathbf{z}, b) = \sum_{p=1}^{P} \frac{1}{\left(1+e^{-d_p\left(\mathbf{k}_p^T \mathbf{z}+b\right)}\right)} \nabla_{\mathbf{z}} \left(1 + e^{-d_p\left(\mathbf{k}_p^T \mathbf{z}+b\right)}\right) \quad + \frac{\lambda}{2} \nabla_{\mathbf{z}} (\mathbf{z}^T\, \mathbf{K}\, \mathbf{z}\,)$$

$$= \sum_{p=1}^{P} \frac{1}{\left(1+e^{-d_p\left(\mathbf{k}_p^T \mathbf{z}+b\right)}\right)} \left(0 + e^{-d_p\left(\mathbf{k}_p^T \mathbf{z}+b\right)}\right) \nabla_{\mathbf{z}} \left(-d_p\left(\mathbf{k}_p^T \mathbf{z} + b\right)\right) \quad + \frac{\lambda}{2} 2\mathbf{K}\, \mathbf{z}$$

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

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

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

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

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

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

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

$$b \leftarrow b - \eta\, \nabla_b E^R_{smooth}(\mathbf{z}, b)$$

# How about Kernel SVM?

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

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

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

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

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

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

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

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

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

From kernel trick, we have:

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

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

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

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

# Soft margin Kernel SVM

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

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

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

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

# Exercise: Kernel Linear Regression

Consider linear regression in kernel-induced feature space
$E_{MSE}\,(\mathbf{w}, b) = \sum_{p=1}^{P}\left(d_p - y_p\right)^2$ where $y_p = \boldsymbol{\varphi}_p^T\,\mathbf{w} + b$

Derive the update equations for weights $\mathbf{z}$ and $b$ by kernelizing linear regression.

Hint: Show that the kernelized loss function can be written as:

$$E_{MSE}\,(\mathbf{z}, b) = \sum_{p=1}^{P}\left(d_p - b - \mathbf{k}_p^T\,\mathbf{z}\right)^2$$

Before proceeding to minimize it with respect to $(\mathbf{z}, b)$

# 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

**K** =

| | | | | |
|---|---|---|---|---|
| $K(1,1)$ | $K(1,2)$ | $K(1,3)$ | … | $K(1,P)$ |
| $K(2,1)$ | $K(2,2)$ | $K(2,3)$ | … | $K(2,P)$ |
| | | | | |
| … | … | … | … | … |
| $K(l,1)$ | $K(l,2)$ | $K(l,3)$ | … | $K(P,P)$ |

# 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 **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: $K\left(\mathbf{x}_p, \mathbf{x}_q\right) = \varphi\left(\mathbf{x}_p\right) \cdot \varphi\left(\mathbf{x}_q\right)$

# 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:

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

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

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

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

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

$$\left(\varphi : \mathcal{X} \rightarrow \Re^N; K_3 \text{ is a kernel over } \Re^N \times \Re^N\right)$$

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

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

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
  - $\sum^*$ where $\sum$ 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:

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

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

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

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 $\sum^* \times \sum^*$ as the inner product of the $p$-spectra of $s$ and $t$.

$$s = statistics$$

$$t = computation$$

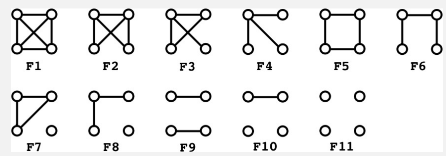
$$p = 3$$

$$\text{Common substrings}: \; tat, \; ati$$

$$K(s,t) = 2$$

## 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 $G_1 \cdots G_M$
- Normalize the histograms $D_{G_1} \cdots D_{G_M}$
- $K(G_i, G_j) = \cos\ \theta(D_{G_i}, D_{G_j})$




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]

$D_{G_i}$

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