

Support Vector Machines

Mark Craven and David Page
Computer Sciences 760
Spring 2018

www.biostat.wisc.edu/~craven/cs760/

Some of the slides in these lectures have been adapted/borrowed from materials developed by Tom Dietterich, Pedro Domingos, Tom Mitchell, David Page, and Jude Shavlik

Goals for the lecture

you should understand the following concepts

- the margin
- slack variables
- the linear support vector machine
- hinge loss
- nonlinear SVMs
- the kernel trick
- the primal and dual formulations of SVM learning
- support vectors
- the kernel matrix
- valid kernels
- polynomial kernel
- Gaussian kernel
- string kernels
- support vector regression



Burr Settles, UW CS PhD

Four key SVM ideas

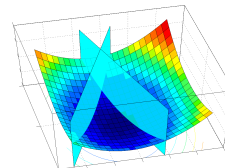
- Maximize the margin
don't choose just *any* separating hyperplane



- Penalize misclassified examples
use soft constraints and slack variables



- Use optimization methods to find model
linear programming
quadratic programming



- Use kernels to represent nonlinear functions and handle complex instances (sequences, trees, graphs, etc.)



Some key vector concepts

the *dot product* between two vectors w and x is defined as:

$$w \cdot x = w^T x = \sum_i w_i x_i$$

for example

$$\begin{bmatrix} 1 \\ 3 \\ -5 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ -2 \\ -1 \end{bmatrix} = (1)(4) + (3)(-2) + (-5)(-1) = 3$$

the 2-norm (Euclidean length) of a vector x is defined as:

$$\|x\|_2 = \sqrt{\sum_i |x_i|^2}$$

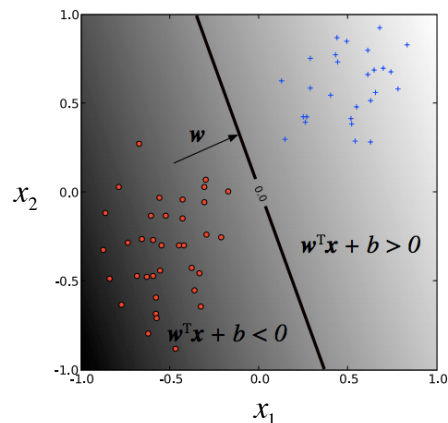
Linear separator learning revisited

suppose we encode our classes as $\{-1, +1\}$ and consider a linear classifier

$$h(x) = \begin{cases} 1 & \text{if } \left(\sum_{i=1}^n w_i x_i \right) + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

an instance $\langle x, y \rangle$ will be classified correctly if

$$y(w^T x + b) > 0$$



Large margin classification

- Given a training set that is linearly separable, there are infinitely many hyperplanes that could separate the positive/negative instances.
 - Which one should we choose?
- In SVM learning, we find the hyperplane that maximizes the margin.

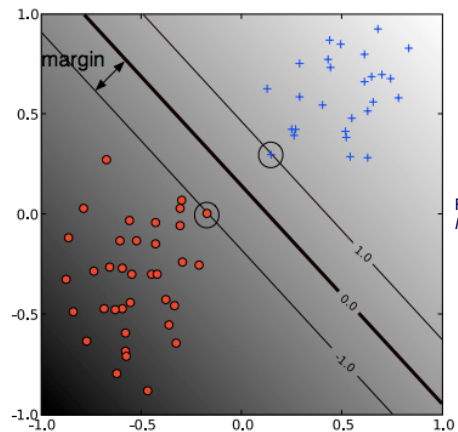
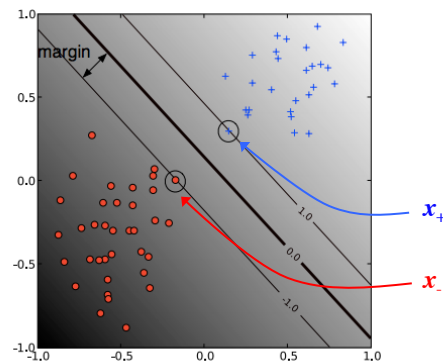


Figure from Ben-Hur & Weston,
Methods in Molecular Biology 2010

The margin

- suppose we learn a hyperplane h given a training set D
- let x_+ denote the closest instance to the hyperplane among positive instances, and similarly for x_- and negative instances
- since $w^T x + b = 0$ and $c(w^T x + b) = 0$ define the same hyperplane, we have the freedom to choose the normalization of w
- choose normalization such that $w^T x_+ + b = 1$ and $w^T x_- + b = -1$ for positive and negative support vectors respectively
- then the margin is given by

$$\begin{aligned}
 \text{margin}_D(h) &= \frac{1}{2} \frac{w}{\|w\|_2} \cdot (x_+ - x_-) \\
 &= \frac{w^T(x_+ - x_-)}{\|w\|_2} \\
 &= \frac{1}{\|w\|_2}
 \end{aligned}$$



The hard-margin SVM

- given a training set $D = \{ \langle \mathbf{x}^{(1)}, y^{(1)} \rangle, \dots, \langle \mathbf{x}^{(m)}, y^{(m)} \rangle \}$
- we can frame the goal of maximizing the margin as a constrained optimization problem

$$\begin{array}{c} \text{minimize} \quad \frac{1}{2} \|\mathbf{w}\|_2^2 \\ \text{adjust these parameters} \quad \mathbf{w}, b \quad \leftarrow \quad \text{to minimize this} \end{array}$$

subject to constraints: $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1$
for $i = 1, \dots, m$

correctly classify $\mathbf{x}^{(i)}$ with room to spare

- and use standard algorithms to find an optimal solution to this problem

The soft-margin SVM

[Cortes & Vapnik, *Machine Learning* 1995]

- if the training instances are not linearly separable, the previous formulation will fail
- we can adjust our approach by using *slack variables* (denoted by ξ) to tolerate errors

$$\text{minimize}_{\mathbf{w}, b, \xi^{(1)} \dots \xi^{(m)}} \quad \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{i=1}^m \xi^{(i)}$$

$$\text{subject to constraints: } y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1 - \xi^{(i)}$$

$$\xi^{(i)} \geq 0$$

$$\text{for } i = 1, \dots, m$$

- C determines the relative importance of maximizing margin vs. minimizing slack



The effect of C in a soft-margin SVM

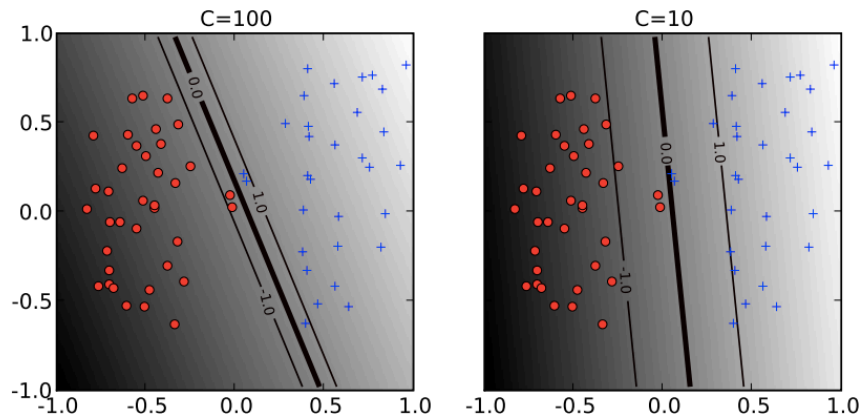
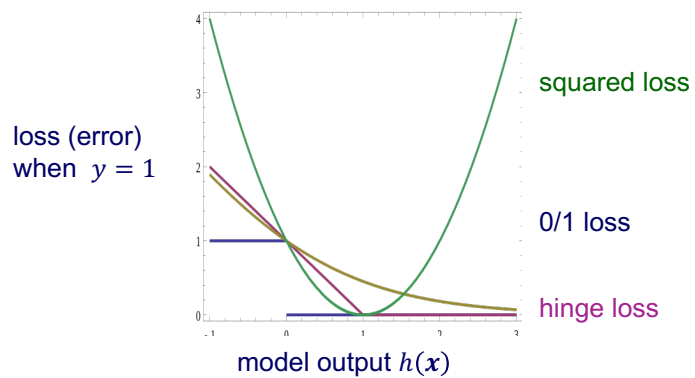


Figure from Ben-Hur & Weston,
Methods in Molecular Biology 2010

Hinge loss

- when we covered neural nets, we talked about minimizing squared loss and cross-entropy loss
- SVMs minimize *hinge loss*



Nonlinear classifiers

- What if a linear separator is not an appropriate decision boundary for a given task?
- For any data set, there exists a mapping ϕ to a higher-dimensional space such that the data is linearly separable

$$\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_k(\mathbf{x}))$$

- Example: mapping to quadratic space

$$\mathbf{x} = \langle x_1, x_2 \rangle \quad \text{suppose } \mathbf{x} \text{ is represented by 2 features}$$

$$\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$

- now try to find a linear separator in this space

Nonlinear classifiers

- for the linear case, our discriminant function was given by

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

- for the nonlinear case, it can be expressed as

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } \hat{\mathbf{w}} \cdot \phi(\mathbf{x}) + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

where $\hat{\mathbf{w}}$ is a higher dimensional vector

SVMs with polynomial kernels

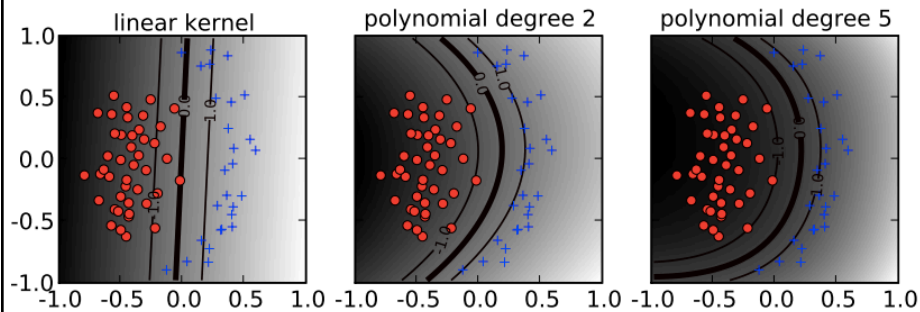


Figure from Ben-Hur & Weston,
Methods in Molecular Biology 2010

The kernel trick

- explicitly computing this nonlinear mapping does not scale well
- a dot product between two higher-dimensional mappings can sometimes be implemented by a *kernel function*
- example: quadratic kernel

$$\begin{aligned}
 k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x} \cdot \mathbf{z} + 1)^2 \\
 &= (x_1 z_1 + x_2 z_2 + 1)^2 \\
 &= x_1^2 z_1^2 + 2x_1 x_2 z_1 z_2 + x_2^2 z_2^2 + 2x_1 z_1 + 2x_2 z_2 + 1 \\
 &= (x_1^2, \sqrt{2}x_1 x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1) \cdot \\
 &\quad (z_1^2, \sqrt{2}z_1 z_2, z_2^2, \sqrt{2}z_1, \sqrt{2}z_2, 1) \\
 &= \phi(\mathbf{x}) \cdot \phi(\mathbf{z})
 \end{aligned}$$

The kernel trick

- thus we can use a kernel to compute the dot product without explicitly mapping the instances to a higher-dimensional space

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^2 = \phi(\mathbf{x}) \cdot \phi(\mathbf{z})$$

But why is the kernel trick helpful?

Using the kernel trick

- given a training set $D = \{ \langle \mathbf{x}^{(1)}, y^{(1)} \rangle, \dots, \langle \mathbf{x}^{(m)}, y^{(m)} \rangle \}$
- suppose the weight vector can be represented as a linear combination of the training instances

$$\mathbf{w} = \sum_{i=1}^m \alpha_i \mathbf{x}^{(i)}$$

- then we can represent a linear SVM as

$$\sum_{i=1}^m \alpha_i \mathbf{x}^{(i)} \cdot \mathbf{x} + b$$

- and a nonlinear SVM as

$$\begin{aligned} & \sum_{i=1}^m \alpha_i \phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}) + b \\ &= \sum_{i=1}^m \alpha_i k(\mathbf{x}^{(i)}, \mathbf{x}) + b \end{aligned}$$

Can we represent a weight vector as a linear combination of training instances?

- consider perceptron learning, where each weight can be represented as

$$w_j = \sum_{i=1}^m \alpha_i x_j^{(i)}$$

- proof: each weight update has the form $w_j(t) = w_j(t-1) + \eta \delta_t^{(i)} y^{(i)} x_j^{(i)}$

$$\delta_t^{(i)} = \begin{cases} 1 & \text{if } \mathbf{x}^{(i)} \text{ misclassified in update } t \\ 0 & \text{otherwise} \end{cases}$$

$$w_j = \sum_t \sum_{i=1}^m \eta \delta_t^{(i)} y^{(i)} x_j^{(i)}$$

$$w_j = \sum_{i=1}^m \left(\sum_t \eta \delta_t^{(i)} y^{(i)} \right) x_j^{(i)}$$

$$w_j = \sum_{i=1}^m \alpha_i x_j^{(i)}$$

Note: this is for the case in which $y \in \{-1, 1\}$

The *primal* and *dual* formulations of the hard-margin SVM

primal

$$\text{minimize}_{\mathbf{w}, b} \quad \frac{1}{2} \|\mathbf{w}\|_2^2$$

$$\text{subject to constraints: } y^{(i)} (\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1$$

$$\text{for } i = 1, \dots, m$$

dual

$$\text{maximize}_{\alpha_1, \dots, \alpha_m} \quad \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{j=1}^m \sum_{k=1}^m \alpha_j \alpha_k y^{(j)} y^{(k)} (\mathbf{x}^{(j)} \cdot \mathbf{x}^{(k)})$$

$$\text{subject to constraints: } \alpha_i \geq 0 \quad \text{for } i = 1, \dots, m$$

$$\sum_{i=1}^m \alpha_i y^{(i)} = 0$$

The *dual* formulation with a kernel (hard margin version)



primal

$$\text{minimize}_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|_2^2$$

subject to constraints: $y^{(i)}(\mathbf{w}^\top \phi(\mathbf{x}^{(i)}) + b) \geq 1$
for $i = 1, \dots, m$

dual

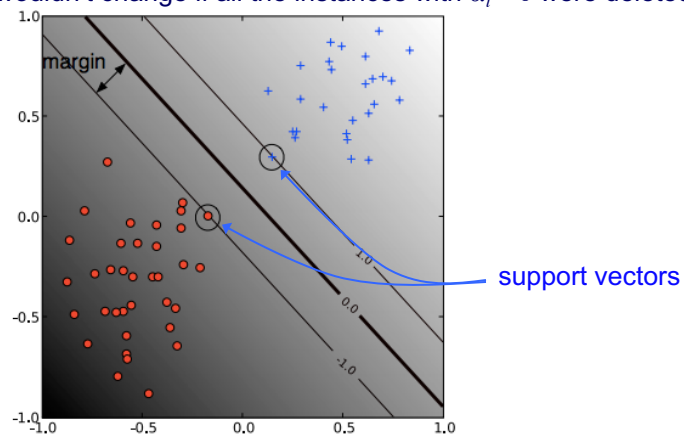
$$\text{maximize}_{\alpha_1, \dots, \alpha_m} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{j=1}^m \sum_{k=1}^m \alpha_j \alpha_k y^{(j)} y^{(k)} k(\mathbf{x}^{(j)}, \mathbf{x}^{(k)})$$

subject to constraints: $\alpha_i \geq 0$ for $i = 1, \dots, m$

$$\sum_{i=1}^m \alpha_i y^{(i)} = 0$$

Support vectors

- the solution to the dual formulation is a sparse linear combination of the training instances
- those instances having $\alpha_i > 0$ are called *support vectors* – they lie on the margin boundary
- the solution wouldn't change if all the instances with $\alpha_i = 0$ were deleted



The kernel matrix

- the kernel matrix (a.k.a. Gram matrix) represents pairwise similarities for instances in the training set

$$\begin{bmatrix} k(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & k(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) & \dots & k(\mathbf{x}^{(1)}, \mathbf{x}^{(m)}) \\ k(\mathbf{x}^{(2)}, \mathbf{x}^{(1)}) & \ddots & & \\ \vdots & & & \\ k(\mathbf{x}^{(m)}, \mathbf{x}^{(1)}) & & & k(\mathbf{x}^{(m)}, \mathbf{x}^{(m)}) \end{bmatrix}$$

- it represents the information about the training set that is provided as input to the optimization process

Some common kernels

- polynomial of degree d

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^d$$

- polynomial of degree up to d

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^d$$

- radial basis function (RBF) (a.k.a. Gaussian)

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{z}\|^2\right)$$

The RBF kernel

- the feature mapping ϕ for the RBF kernel is infinite dimensional!
- recall that $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{z})$

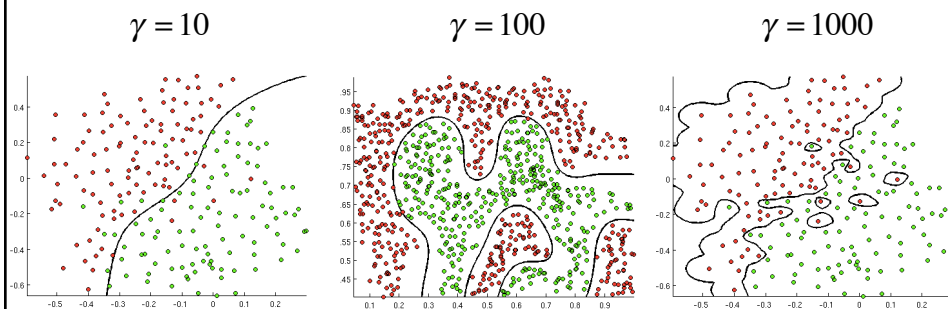
$$k(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{1}{2}\|\mathbf{x} - \mathbf{z}\|^2\right) \quad \text{for } \gamma = \frac{1}{2}$$

$$= \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right) \exp\left(-\frac{1}{2}\|\mathbf{z}\|^2\right) \exp(\mathbf{x} \cdot \mathbf{z})$$

$$= \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right) \exp\left(-\frac{1}{2}\|\mathbf{z}\|^2\right) \left(\sum_{n=0}^{\infty} \frac{(\mathbf{x} \cdot \mathbf{z})^n}{n!}\right)$$

from the Taylor series
expansion of $\exp(\mathbf{x} \cdot \mathbf{z})$

The RBF kernel illustrated



Figures from openclassroom.stanford.edu (Andrew Ng)

What makes a valid kernel?

- $k(\mathbf{x}, \mathbf{z})$ is a valid kernel if there is some ϕ such that

$$k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{z})$$
- this holds for a symmetric function $k(\mathbf{x}, \mathbf{z})$ if and only if the kernel matrix \mathbf{K} is positive semidefinite for any training set (Mercer's theorem)

definition of positive semidefinite (p.s.d): $\forall \mathbf{v} : \mathbf{v}^\top \mathbf{K} \mathbf{v} \geq 0$

Kernel algebra

- given a valid kernel, we can make new valid kernels using a variety of operators

kernel composition

$$k(\mathbf{x}, \mathbf{v}) = k_a(\mathbf{x}, \mathbf{v}) + k_b(\mathbf{x}, \mathbf{v})$$

$$k(\mathbf{x}, \mathbf{v}) = \gamma k_a(\mathbf{x}, \mathbf{v}), \gamma > 0$$

$$k(\mathbf{x}, \mathbf{v}) = k_a(\mathbf{x}, \mathbf{v}) k_b(\mathbf{x}, \mathbf{v})$$

$$k(\mathbf{x}, \mathbf{v}) = \mathbf{x}^\top \mathbf{A} \mathbf{v}, \quad \mathbf{A} \text{ is p.s.d.}$$

$$k(\mathbf{x}, \mathbf{v}) = f(\mathbf{x}) f(\mathbf{v}) k_a(\mathbf{x}, \mathbf{v})$$

mapping composition

$$\phi(\mathbf{x}) = (\phi_a(\mathbf{x}), \phi_b(\mathbf{x}))$$

$$\phi(\mathbf{x}) = \sqrt{\gamma} \phi_a(\mathbf{x})$$

$$\phi_l(\mathbf{x}) = \phi_{ai}(\mathbf{x}) \phi_{bj}(\mathbf{x})$$

$$\phi(\mathbf{x}) = L^\top \mathbf{x}, \quad \text{where } \mathbf{A} = LL^\top$$

$$\phi(\mathbf{x}) = f(\mathbf{x}) \phi_a(\mathbf{x})$$

The power of kernel functions

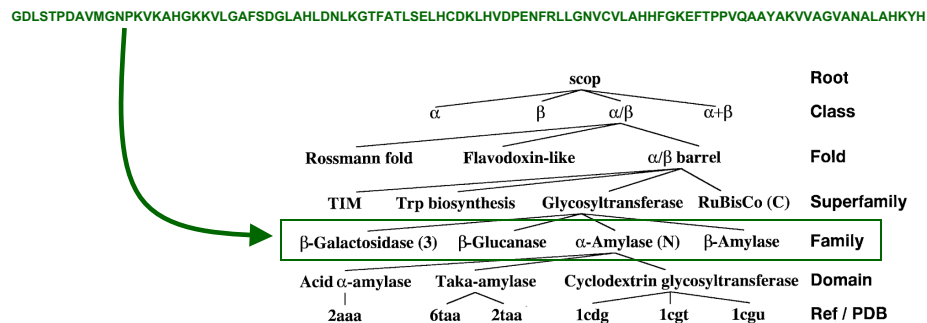
- kernels can be designed and used to represent complex data types such as
 - strings
 - trees
 - graphs
 - etc.
- let's consider a specific example



The protein classification task

Given: amino-acid sequence of a protein

Do: predict the *family* to which it belongs



The k -spectrum feature map

- we can represent sequences by counts of all of their k -mers

$x = \text{AKQDYYYYEI}$



$k = 3$

$\phi(x) = (0, 0, \dots, 1, \dots, 1, \dots, 2)$
 $\text{AAA} \text{ AAC} \dots \text{AKQ} \dots \text{DYY} \dots \text{YYY}$

- the dimension of $\phi(x) = |A|^k$ where $|A|$ is the size of the alphabet
 - using 6-mers for protein sequences, $|20|^6 = 64$ million
 - almost all of the elements in $\phi(x)$ are 0 since a sequence of length l has at most $l-k+1$ k -mers

The k -spectrum kernel

- consider the k -spectrum kernel applied to x and z with $k = 3$

$x = \text{AKQDYYYYEI}$

$z = \text{AKQIAKQYEI}$

$\phi(x) = (0, \dots, 1, \dots, 1, \dots, 0)$
 $\text{AAA} \dots \text{AKQ} \dots \text{YEI} \dots \text{YYY}$

$\phi(z) = (0, \dots, 2, \dots, 1, \dots, 0)$
 $\text{AAA} \dots \text{AKQ} \dots \text{YEI} \dots \text{YYY}$

$$\phi(x) \cdot \phi(z) = 2 + 1 = 3$$

(k, m) -mismatch feature map

- closely related protein sequences may have few exact matches, but many near matches
- the (k, m) -mismatch feature map uses the k -spectrum representation, but allows up to m mismatches

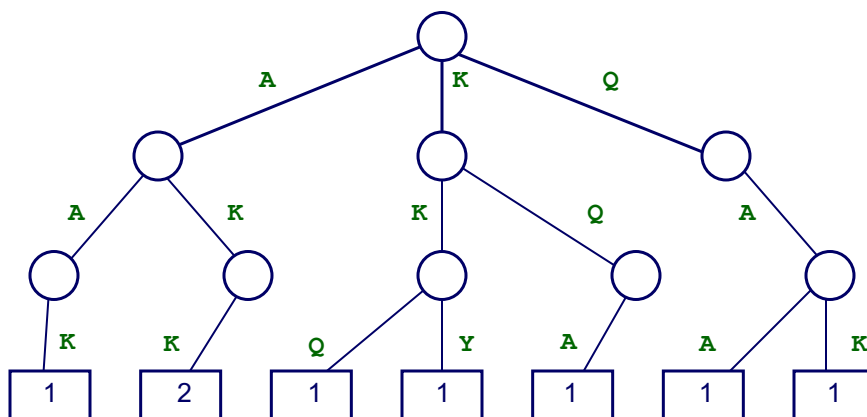
$x = \text{AKQ}$

$\downarrow k=3, m=1$

$\phi(x) = (0, \dots, 1, \dots, 1, \dots, 1, \dots, 0)$
 $\text{AAA} \quad \text{AAQ} \quad \dots \quad \text{AKQ} \quad \dots \quad \text{DKQ} \quad \dots \quad \text{YYY}$

Using a trie to represent 3-mers

- example: representing all 3-mers of the sequence QAACKQAQKY



Computing the kernels efficiently with tries [Leslie et al., *NIPS* 2002]

***k*-spectrum kernel**

- for each sequence
 - build a trie representing its k -mers
- compute kernel $\phi(x) \cdot \phi(z)$ by traversing trie for x using k -mers from z
 - update kernel function when reaching a leaf

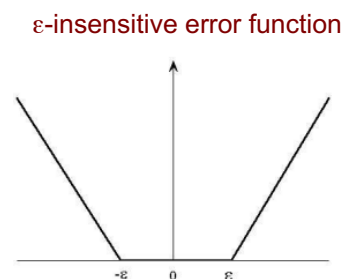
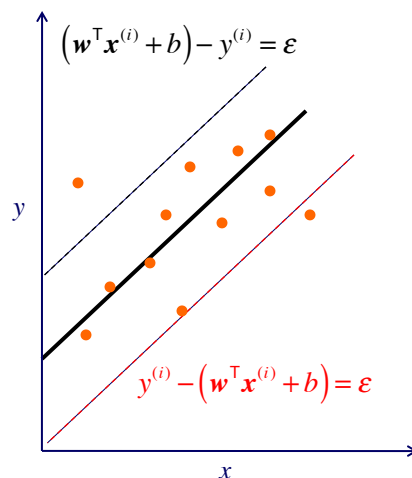
***(k, m)*-mismatch kernel**

- for each sequence
 - build a trie representing its k -mers and also k -mers with at most m mismatches
- compute kernel $\phi(x) \cdot \phi(z)$ by traversing trie for x using k -mers from z
 - update kernel function when reaching a leaf

scales linearly with sequence length: $O(k^{m+1}|A|^m(|x|+|z|))$

Support vector regression

- the SVM idea can also be applied in regression tasks
- an ε -insensitive error function specifies that a training instance is well explained if the model's prediction is within ε of $y^{(i)}$



Support vector regression

$$\text{minimize}_{\mathbf{w}, b, \xi^{(1)} \dots \xi^{(m)}, \hat{\xi}^{(1)} \dots \hat{\xi}^{(m)}} \quad \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{i=1}^m (\xi^{(i)} + \hat{\xi}^{(i)})$$

$$\text{subject to constraints: } (\mathbf{w}^\top \mathbf{x}^{(i)} + b) - y^{(i)} \leq \varepsilon + \xi^{(i)}$$

$$y^{(i)} - (\mathbf{w}^\top \mathbf{x}^{(i)} + b) \leq \varepsilon + \hat{\xi}^{(i)}$$

$$\xi^{(i)}, \hat{\xi}^{(i)} \geq 0$$

$$\text{for } i = 1, \dots, m$$

slack variables allow predictions for some training instances to be off by more than ε

Learning theory justification for maximizing the margin

$$\text{error}_D(h) \leq \text{error}_D(h) + \sqrt{\frac{VC \left(\log \frac{2m}{VC} + 1 \right) + \log \frac{4}{\delta}}{m}}$$

\nearrow error on true distribution \nearrow training set error \nearrow VC-dimension of hypothesis class

Vapnik showed there is a connection between the margin and VC dimension

$$VC \leq \frac{4R^2}{\text{margin}_D(h)^2} \quad \leftarrow \text{constant dependent on training data}$$

thus to minimize the VC dimension (and to improve the error bound) \rightarrow maximize the margin

Comments on SVMs

- we can find solutions that are globally optimal (maximize the margin)
 - because the learning task is framed as a convex optimization problem
 - no local minima, in contrast to multi-layer neural nets
- there are two formulations of the optimization: *primal* and *dual*
 - dual represents classifier decision in terms of support vectors
 - dual enables the use of kernel functions
- we can use a wide range of optimization methods to learn SVM
 - standard quadratic programming solvers
 - SMO [Platt, 1999]
 - linear programming solvers for some formulations
 - etc.

Comments on SVMs

- kernels provide a powerful way to
 - allow nonlinear decision boundaries
 - represent/compare complex objects such as strings and trees
 - incorporate domain knowledge into the learning task
- using the kernel trick, we can implicitly use high-dimensional mappings without explicitly computing them
- one SVM can represent only a binary classification task; multi-class problems handled using multiple SVMs and some encoding
 - one class vs. rest
 - ECOC
 - etc.
- empirically, SVMs have shown state-of-the art accuracy for many tasks
- the kernel idea can be extended to other tasks (anomaly detection, etc.)