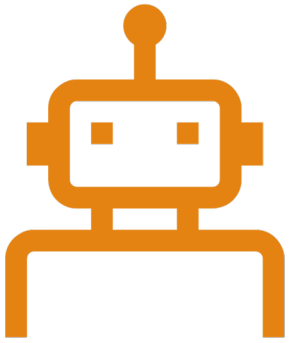


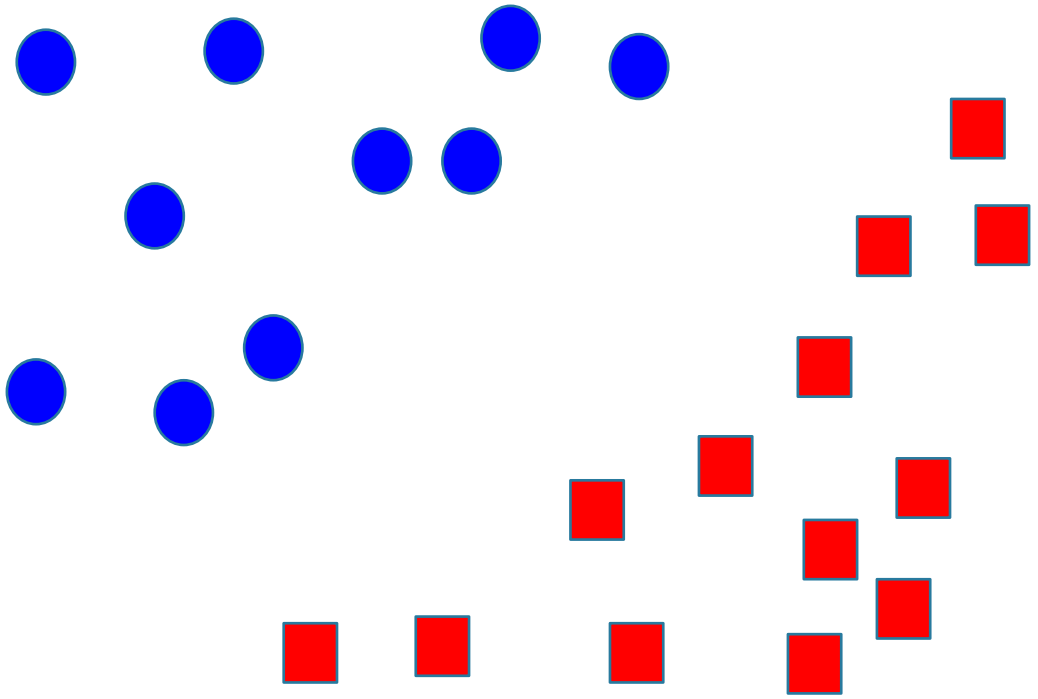
Machine Learning EN.601.475/675

DR. PHILIP GRAFF

Kernel Methods

OR… MAKING THE NON-LINEAR LINEAR

Linearly Separable



Linear Prediction Methods

- Linear Regression $\hat{y} = \mathbf{w}^T\mathbf{x} + b$
- Logistic Regression $\hat{y} = \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{x})}$
- Perceptron $\hat{y} = \text{sign}(\mathbf{w}^T\mathbf{x})$
- Support Vector Machines

SVM Solution (Dual formulation)

Select αs that maximize

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i^T \mathbf{x}_j)$$

such that $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$

This is quadratic programming problem can be solved using the same methods as for the primal representation

Result: predictions for new examples are given by

$$\mathbf{w}^T \mathbf{x} = \left[\sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \right]^T \mathbf{x} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i^T \mathbf{x}$$

One additional QP solver: Sequential Minimal Optimization (SMO)

- **Problem:** Select αs that maximize:

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i^T \mathbf{x}_j)$$

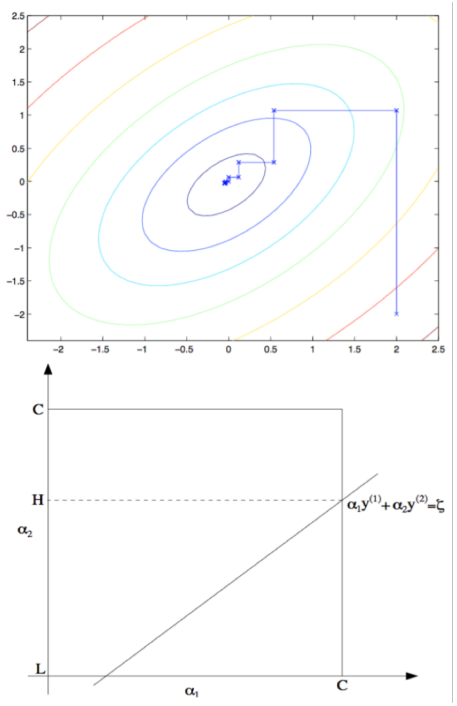
such that $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$

- **SMO Solution:**

Repeat till convergence {

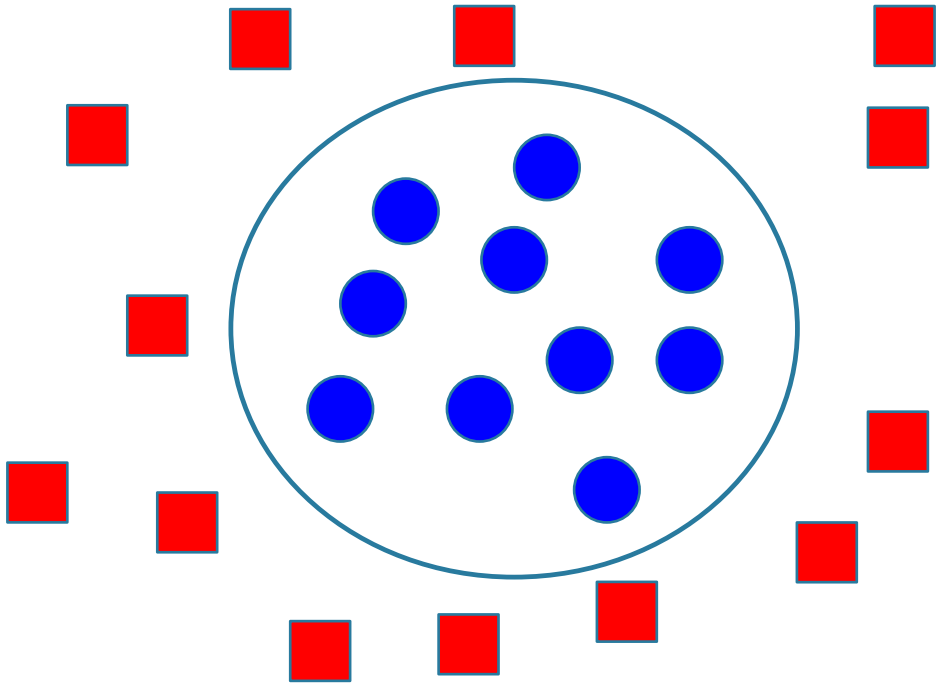
1. Select some pair α_i and α_j to update next (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum).
2. Reoptimize $W(\alpha)$ with respect to α_i and α_j, while holding all the other α_k's $(k \neq i, j)$ fixed.

}



Content/Figure credit: Ng

Not Linearly Separable Data



Slack Variables

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i \quad \text{s.t.} \quad (\mathbf{w}^T\mathbf{x}_i)y_i + \xi_i \geq 1, \forall i$$
$$\xi_i \geq 0, \forall i$$

We can always satisfy the margin using ξ

- We want these ξs to be small
- Trade off parameter C (similar to λ before)

ξs are called slack variables

- They cut the margin some "slack"

Review: Lingering Questions

- What would we do if we saw all of the data (batch)?
  - We'd pick the best separating hyperplane!
- Which separating hyperplane is the best?
  - The maximum margin separator
  - Use a quadratic regularizer on the weights
- What can we do for non-linear data?
  - It's not separable, use slack variables
  - Can we do better? 

Handling Non-Linear Data

Option 1: Add features by hand that make the data separable

- Requires feature engineering

Option 2: Learn a small number of additional features that will suffice

- We'll see this eventually

Option 3: Kernel trick

- Today

Basis / Feature Mapping Functions

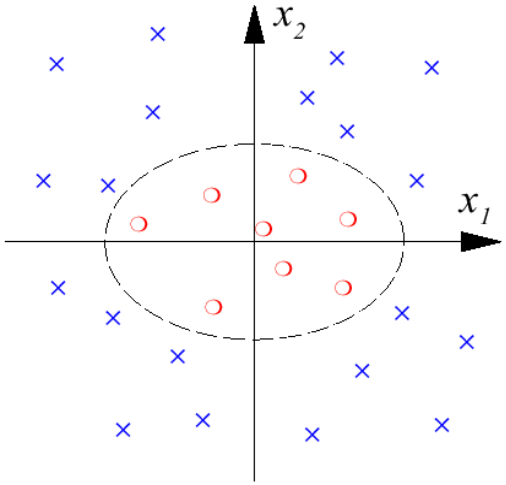
- We denote basis functions as $\phi(\mathbf{x})$ which could be nonlinear functions of the elements in **x**
- Terminology varies; often basis functions are called feature mapping functions in the context of kernels and that is what we will use here
- Can we use this concept to address data sets that are not linearly separable?

Feature Mapping Functions: Example

- Assume a two-dimensional input vector: $\mathbf{x} = [x_{(1)}, x_{(2)}]$
- Apply a feature mapping function: $\phi(\mathbf{x}) = (x_{(1)}^2, \sqrt{2}x_{(1)}x_{(2)}, x_{(2)}^2)$
- Why is this useful?
  - Results in an elliptical decision boundary! $\phi_{(1)}(\mathbf{x}) + 2\phi_{(2)}(\mathbf{x}) < 3$
  - Not linear in **x**, but linear in $\phi(\mathbf{x})$
  - Boundaries defined by linear combinations of second order terms are ellipses, parabolas, and hyperbolas in the original space

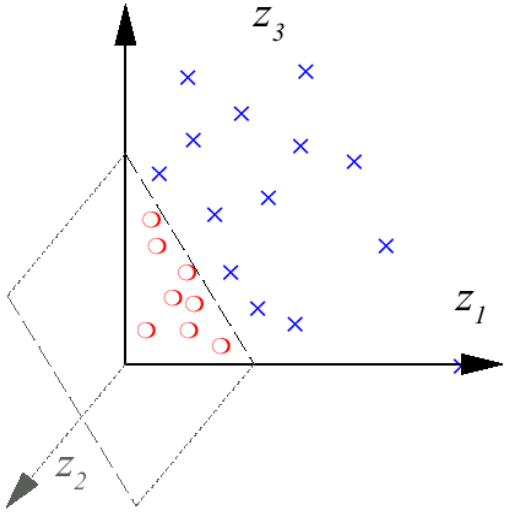
Geometric Interpretation

$$x_{(1)}^2 + 2x_{(2)}^2 < 3$$



Non-linear in **x**

$$\phi_{(1)}(\mathbf{x}) + 2\phi_{(2)}(\mathbf{x}) < 3$$



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

Why Feature Mapping Functions?

- Recall that to make something linearly separable I can just add a unique feature to every example
- Any dataset is linearly separable if we use enough dimensions
  - In an n-dimensional space, almost any set of up to n+1 labeled points is linearly separable!
- We can obtain linear separability by projecting data into higher dimensional spaces
  - Use smarter techniques to obtain generalizeable separability

Feature Functions + SVM

Replace **x** with a feature mapping function:

$$\text{argmin}_{\mathbf{w}} ||\mathbf{w}||^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^T \phi(\mathbf{x}_i)) \geq 1 \quad \forall\, i$$

Dot product now taken over a higher dimensional feature space

- If ϕ is quadratic, then the feature space is a quadratic space in terms of the inputs

Limitations

We still must learn **w**

- **w** will grow with the feature space
- e.g. quadratic kernel: $[\mathbf{x}] = 10^2 \rightarrow [\phi(\mathbf{x})] \sim O(10^4)$

Feature functions just increase the feature space in a non-linear way

Too expensive – is there another way?

Dual Approach with Feature Mapping

We want to select the αs that maximize

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j (\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j))$$

There is no **w**!

There is no modeling constraint that prevents us from making the dimension of $\phi(\mathbf{x})$ very large

The αs do not grow with the dimension of $\phi(\mathbf{x})$

Interesting…

Kernels in SVM

Let's replace $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ with a "kernel" function K

$$K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}')$$

We now have:

- Maximization Problem: $\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$
- New Prediction: $\mathbf{w}^T\mathbf{x} \Rightarrow \mathbf{w}^T\phi(\mathbf{x}) = \left(\sum_{i=1}^{N}\alpha_i y_i \phi(\mathbf{x}_i)^T\right)\phi(\mathbf{x})$

$$= \sum_{i=1}^{N}\alpha_i y_i \phi(\mathbf{x}_i)^T\phi(\mathbf{x}) = \sum_{i=1}^{N}\alpha_i y_i K(\mathbf{x}_i, \mathbf{x})$$

Why?

We have removed all dependencies in the SVM on the size of the feature space

- The feature space $\phi(\mathbf{x})$ appears only in the kernel

If the kernel function does the work, we can handle any feature space.

Kernels are often faster to compute than the set of basis functions

- More on this later…

Intuition About Over-Fitting

Wait a minute!

Assuming we project features then even using the simple projection shown so far, we'd have way too many features!

Didn't we learn that too many features means over-fitting?

Saved by the Dual

We aren't free to choose a parameter for each feature

w is a linear combination of the inputs

- We can only choose the parameters for α's
- There are only *N* α's, no matter how large our feature space projection

The input data **X** puts a constraint on our flexibility in high dimensional space

The Kernel Trick

1. Take a linear SVM
2. Substitute a non-linear kernel
3. Optimize objective in the dual
4. We get non-linear classification!

- Without
  - Over-fitting
  - Learning too many parameters
  - Computing a large feature space

Kernels: a closer look

A kernel is a scalar product between two high dimensional feature vectors.

A proposed kernel function can be written in the following form:

$$K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

We can define any mapping function and then compute the kernel

Example: Quadratic Kernel

Let's take the cross product of all features (quadratic)

Why is the quadratic a valid kernel?

It's just a scalar product of the two vectors

$$
\begin{aligned}
K(\mathbf{x}, \mathbf{x}') &= (\mathbf{x} \cdot \mathbf{x}')^2 \\
&= (x_{(1)} x'_{(1)} + x_{(2)} x'_{(2)})^2 \\
&= (x^2_{(1)} x'^2_{(1)} + x^2_{(2)} x'^2_{(2)} + 2 x_{(1)} x_{(2)} x'_{(1)} x'_{(2)}) \\
&= (x^2_{(1)}, x^2_{(2)}, \sqrt{2} x_{(1)} x_{(2)}) \cdot (x'^2_{(1)}, x'^2_{(2)}, \sqrt{2} x'_{(1)} x'_{(2)}) \\
&= \phi(\mathbf{x}) \cdot \phi(\mathbf{x}')
\end{aligned}
$$

$\phi(\mathbf{x})$ is the feature mapping function used for the ellipse example

This can be applied for arbitrary dimensions of **x**

Example: Polynomial Kernel

In fact, this can be applied with any exponent P

$$K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x} \cdot \mathbf{x}')^P$$

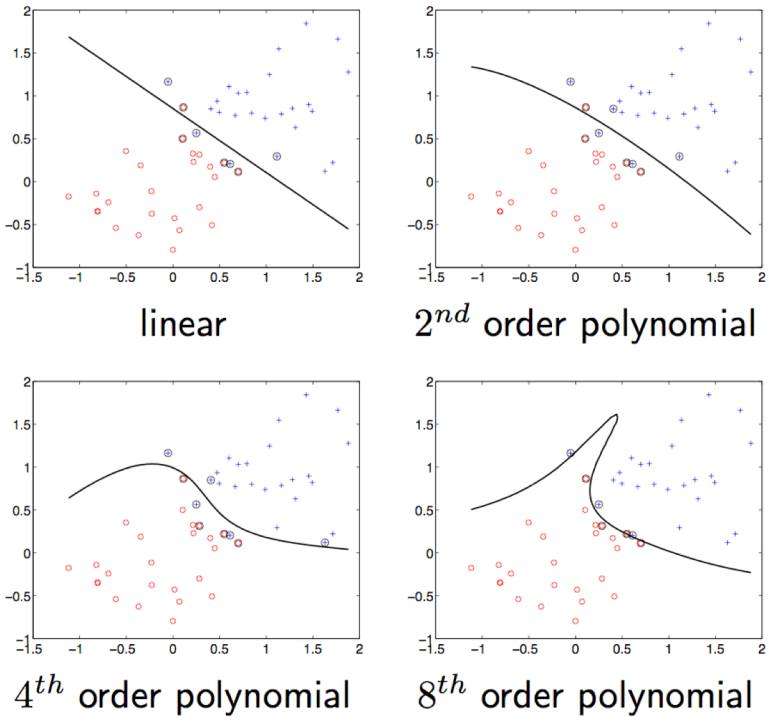
This is the polynomial kernel

- To get the feature vectors we would concatenate all elements up to the p^{th} order polynomial terms of the components of **x** (weighted appropriately)

The polynomial kernel in action: http://www.youtube.com/watch?v=3liCbRZPrZA

Polynomial Kernel

- In the given feature space the separators are non-linear
- In the high dimensional space they are linear
- Support vectors are circled



Decision Boundary

How does the kernel influence the decision boundary?

Recall that prediction is given by

$$\mathbf{w}^T\phi(\mathbf{x}) = \left(\sum_{i=1}^{N}\alpha_i y_i \phi(\mathbf{x}_i)^T\right)\phi(\mathbf{x}) = \sum_{i=1}^{N}\alpha_i y_i \phi(\mathbf{x}_i)^T\phi(\mathbf{x}) = \sum_{i=1}^{N}\alpha_i y_i K(\mathbf{x}_i, \mathbf{x})$$

The larger $K(\mathbf{x}, \mathbf{x}_i)$ is, the more $\mathbf{x}_i$ contributes to the decision for $\mathbf{x}$

- $\mathbf{x}$ receives a label based on those support vectors (examples with large α) with highest $K(\mathbf{x}, \mathbf{x}_i)$

Is the Kernel a Similarity Function?

Does that mean $K(\mathbf{x}, \mathbf{x}_i)$ is a similarity function?

- Give same label as most similar examples

Sort of…

- Recall: $\cos(\theta) = \frac{\mathbf{x} \cdot \mathbf{x}'}{||\mathbf{x}||||\mathbf{x}'||}$
- Therefore: $\mathbf{x} \cdot \mathbf{x}' = ||\mathbf{x}||||\mathbf{x}'|| \cos(\theta)$
- So: $\alpha K(\mathbf{x}, \mathbf{x}') = \alpha \phi(\mathbf{x}) \cdot \phi(\mathbf{x}') = \alpha ||\phi(\mathbf{x})||||\phi(\mathbf{x}')|| \cos(\theta)$

Is the Kernel a Similarity Function?

Note:

- $\phi(\mathbf{x})$: constant across all $\mathbf{x}'$ in the prediction
- $\alpha\phi(\mathbf{x}')$: α is scaled per $\mathbf{x}'$ so this just assigns importance
- θ: the angle between the vectors

When $\theta = 0$, this is 1 so larger values for more similar vectors

Mercer's Theorem

Suppose ***K*** is a valid kernel

Define the Kernel / Gram matrix for a set of *m* points as:

$$K_{ij} = K(x_i, x_j)$$

K must be symmetric:

$$K_{ij} = K(x_i, x_j) = \phi(x_i)^T \phi(x_j) = \phi(x_j)^T \phi(x_i) = K(x_j, x_i) = K_{ji}$$

$$i = 1, ..., m$$
$$j = 1, ..., m$$

Mercer's Theorem

$\phi_k(x)$: kth position of basis function vector

$$
\begin{aligned}
\mathbf{z}^T K \mathbf{z} &= \sum_i \sum_j z_{(i)} K_{ij} z_{(j)} \\
&= \sum_i \sum_j z_{(i)} \phi(x_{(i)})^T \phi(x_{(j)}) z_{(j)} \\
&= \sum_i \sum_j z_{(i)} \sum_k \phi_k(x_{(i)}) \phi_k(x_{(j)}) z_{(j)} \\
&= \sum_k \sum_i \sum_j z_{(i)} \phi_k(x_{(i)}) \phi_k(x_{(j)}) z_{(j)} \\
&= \sum_k \left(\sum_i z_{(i)} \phi_k(x_{(i)}) \right)^2 \\
&\geq 0
\end{aligned}
$$

Mercer's Theorem

Let $K : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ be given. Then for ***K*** to be a valid (Mercer) kernel, it is necessary and sufficient that for any finite data set, the corresponding kernel matrix is symmetric positive semi-definite.

Kernel Definitions

A kernel is

- A scalar product of two vectors in high dimensional space
- Defined by Mercer's theorem

How do we test a kernel without writing $\phi(\mathbf{x})$ explicitly?

Equivalent definition

- The Kernel/Gram matrix $\boldsymbol{K}$ should be positive semidefinite for all $\mathbf{x}$
- Kernel/Gram matrix $\boldsymbol{K}$: $\boldsymbol{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
- Positive semidefinite:

$$\mathbf{x}^T M \mathbf{x} \geq 0, \forall \mathbf{x} \in \mathbb{R}^m$$

Building Kernels

How do we build a kernel?

- Decide on a projection that is meaningful for data

How do we know something is valid?

- Show it's a scalar product (symmetric)
- Show positive semidefinite kernel matrix
- Best: compose new kernels from old kernels

Kernel Operations

Many operations over kernels yield new kernels:

$$K(\mathbf{x}, \mathbf{x}') = cK_1(\mathbf{x}, \mathbf{x}')$$
$$K(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})K_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$$
$$K(\mathbf{x}, \mathbf{x}') = \exp\left(K_1(\mathbf{x}, \mathbf{x}')\right)$$
$$K(\mathbf{x}, \mathbf{x}') = K_1(\mathbf{x}, \mathbf{x}') + K_2(\mathbf{x}, \mathbf{x}')$$
$$K(\mathbf{x}, \mathbf{x}') = K_1(\mathbf{x}, \mathbf{x}')K_2(\mathbf{x}, \mathbf{x}')$$

More examples in Murphy and Bishop!

Gaussian Kernel

Use a Gaussian to define a radial basis function (RBF) kernel

- Since this is not a probability drop the normalization

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

- Why is this a valid kernel?
- Expand the square:

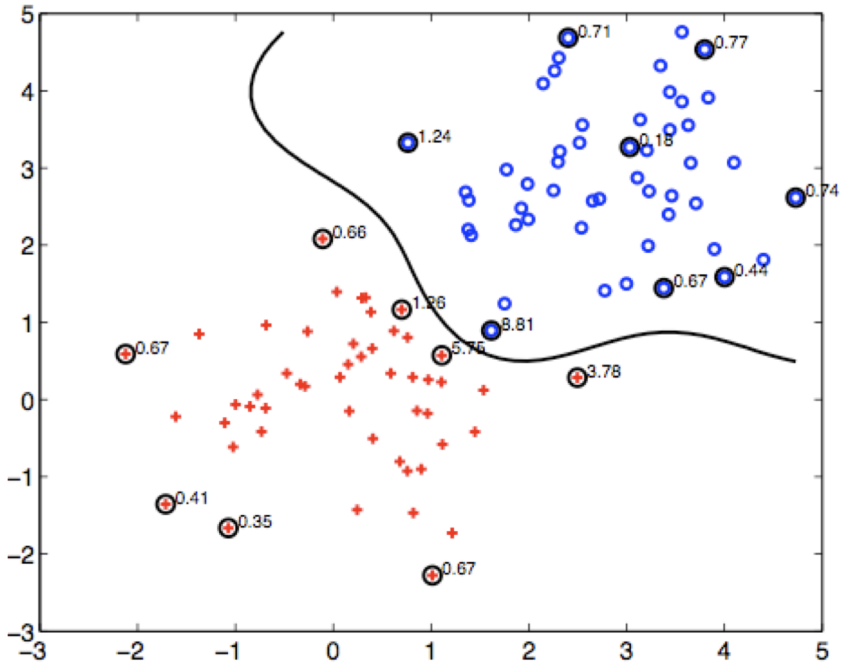
$$||\mathbf{x} - \mathbf{x}'||^2 = \mathbf{x}^T\mathbf{x} + \mathbf{x}'^T\mathbf{x}' - 2\mathbf{x}^T\mathbf{x}'$$

- Substitute:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-\mathbf{x}^T\mathbf{x}}{2\sigma^2}\right)\exp\left(\frac{-\mathbf{x}'^T\mathbf{x}'}{2\sigma^2}\right)\exp\left(\frac{2\mathbf{x}^T\mathbf{x}'}{2\sigma^2}\right)$$

$$K(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})K_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$$
$$K_1(\mathbf{x}, \mathbf{x}') = \exp\left(K_2(\mathbf{x}, \mathbf{x}')\right)$$

Radial Basis Function example



Picture by Tommi Jaakkola

Gaussian Kernel

Two-dimensional Gaussian magnitude

- σ determines the smoothness of the function
- Large σ means the support vector has greater influence area
  - Fewer support vectors needed to cover boundaries

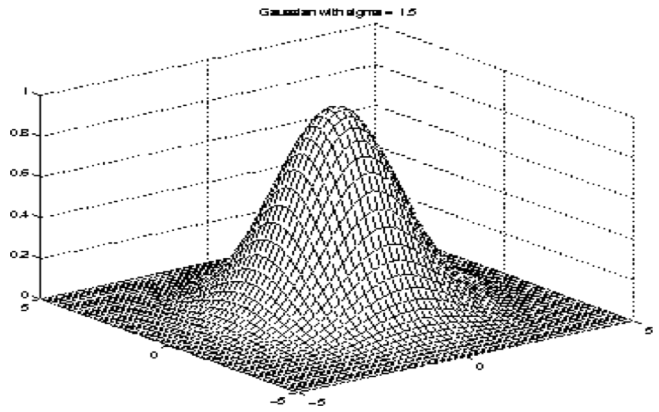
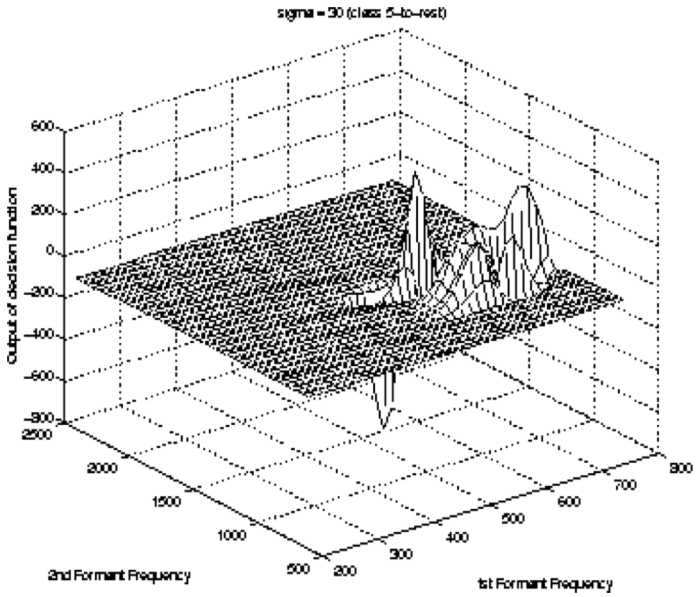
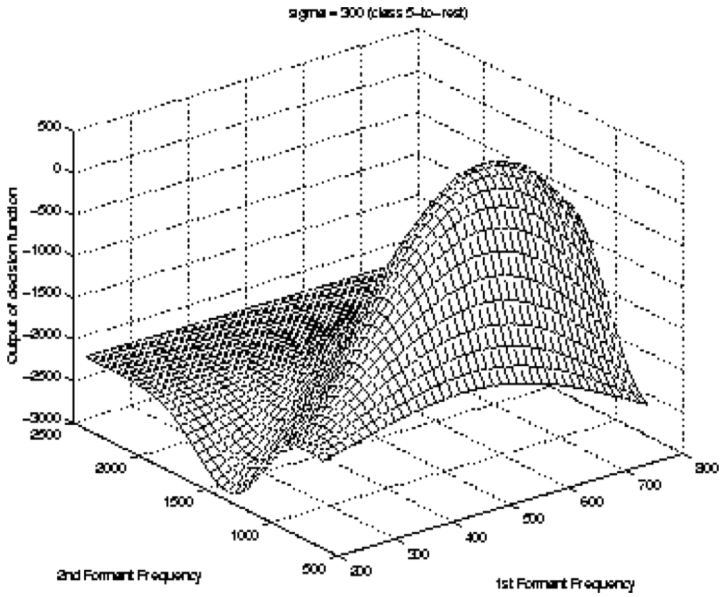


Figure credit: http://mi.eng.cam.ac.uk/~kkc21/thesis_main/node31.html

Decision Boundary



Smaller σ



Larger σ

Kernels for Objects

We've talked about kernels as operating over $\mathbf{x} \in \mathbb{R}^M$

However, we can define **x** as anything as long as we can compute $K(\mathbf{x}, \mathbf{x}')$

Kernels for

- Strings
- Trees/Graphs
- Images

Kernels for Strings

Represent a document as a feature vector

- Each feature corresponds to a word in the document
- Classify document based on the words

Even better: each feature corresponds to a sub-string in the document

- Include non-contiguous sub-strings
- Value of feature is dependent on frequency of where it appears

We are limited computationally by the size of the documents and the resolution of the substrings!

Kernels for Strings

String Subsequence Kernel

$$K(\mathbf{x}, \mathbf{x}') = \sum_{u \in \Sigma^d} \sum_{i:u=\mathbf{x}_{(i)}} \sum_{i:u=\mathbf{x}'_{(j)}} \lambda^{|i|+|j|}$$

- For all strings u of length d
- For all substrings of **x**
- For all substrings of **x'**
- λ to the power of the size of the combined lengths

- Computing features would take $O(|\Sigma|^d)$ time
- Can compute the kernel for this feature representation using dynamic programming

Lodhi, et al. 2002

Biology: Splice Site Recognition

Find the boundary between exons and introns in eukaryotes (complex organism)

- What part of DNA codes for genes?

Input is a sequence of DNA base pairs

Normally each feature indicates a substring of base pairs appearing in the sequence

Biology: Splice Site Recognition

Each possible substring of DNA is a new feature

Use the kernel approach as for strings

Problem for DNA: long substrings unlikely but still informative

Solution: a kernel from many weighted spectrum kernels

$$K_l(\mathbf{x}, \mathbf{x}') = \sum_{d=1}^{l} \beta_d K_d^{\text{spectrum}}(\mathbf{x}, \mathbf{x}')$$

Other Kernel Methods

EVERYTHING CAN BE NON-LINEAR…

Kernel Perceptron

There is a dual form for the perceptron:

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x})\right)$$

- Recall: α_i=1 if we made a mistake on round i

Replace the dot product with a kernel:

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right)$$

Kernel Linear Regression

We can define linear regression with quadratic regularization using a linear combination of **x**:

$$C(\mathbf{w}) = E_D(\mathbf{w}) + E_W(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i))^2 + \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\frac{\partial}{\partial \mathbf{w}}C(\mathbf{w}) = -\sum_{i=1}^{N}(y_i - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i))\boldsymbol{\phi}(\mathbf{x}_i) + \lambda\mathbf{w} = 0 \quad \Rightarrow \quad \mathbf{w} = \left(\lambda\mathbf{I} + \sum_{i=1}^{N}\boldsymbol{\phi}(\mathbf{x}_i)\boldsymbol{\phi}(\mathbf{x}_i)^T\right)^{-1}\left(\sum_{j=1}^{N}y_j\boldsymbol{\phi}(\mathbf{x}_j)\right)$$

So we can get a kernel version:

$$\mathbf{w} = (\lambda\mathbf{I}_d + \Phi\Phi^T)^{-1}\Phi\mathbf{y} = \Phi(\Phi^T\Phi + \lambda\mathbf{I}_n)^{-1}\mathbf{y} = \Phi(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}$$

$$\mathbf{w} = \sum_{i=1}^{N}\alpha_i\phi(\mathbf{x}_i)$$

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}$$

$$\begin{aligned}\hat{y} &= \mathbf{w}^T\phi(\mathbf{x}) \\ &= \mathbf{y}(\Phi^T\Phi + \lambda\mathbf{I})^{-1}\Phi^T\phi(\mathbf{x}) \\ &= \mathbf{y}(\mathbf{K} + \lambda\mathbf{I})^{-1}\boldsymbol{\kappa}(\mathbf{x})\end{aligned}$$

$$\kappa_i(\mathbf{x}) = K(\mathbf{x}_i, \mathbf{x})$$

Kernel Logistic Regression

We can do the same trick with logistic regression

Represent **w** in terms of **x** and α:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}_i)$$

Insert a kernel in place of a dot product in the model:

$$\Pr(y = 1|\mathbf{x}, \alpha) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i)\right)}$$

Derive new gradient descent rule on α

Summary

The good

- Arbitrarily high dimensionality
- Extensions to other data types
- Non-linearity in a parametric linear framework

The bad

- What is a good kernel?
  - Whole field on designing kernels, learning kernels
- Cannot handle large data
  - Kernel matrix grows quadratically in *N*

Next Time

DECISION TREES!