Aim:

The tutorial is intended to give a broad introduction to the kernel approach to pattern analysis. This will cover:

- Why linear pattern functions?
- Why kernel approach?
- How to plug and play with the different components of a kernel-based pattern analysis system?
What won’t be included:

- Other approaches to Pattern Analysis
- Complete History
- Bayesian view of kernel methods
- More recent developments
OVERALL STRUCTURE

Part 1: Introduction to the Kernel methods approach.

Part 2: Projections and subspaces in the feature space.

Part 3: Other learning algorithms with the example of Support Vector Machines.

Part 4: Kernel design strategies.
PART 1 STRUCTURE

• Introduction to pattern analysis and brief history
• Kernel methods approach
• Worked example of kernel Ridge Regression
• Properties of kernels.
Pattern Analysis

• Data can exhibit regularities that may or may not be immediately apparent
  – exact patterns – eg motions of planets
  – complex patterns – eg genes in DNA
  – probabilistic patterns – eg market research

• Detecting patterns makes it possible to understand and/or exploit the regularities to make predictions

• Pattern analysis is the study of automatic detection of patterns in data
Defining patterns

• Exact patterns: non-trivial function $f$ such that

$$f(x) = 0$$

• Approximate patterns: $f$ such that

$$f(x) \approx 0$$

• Statistical patterns: $f$ such that

$$\mathbb{E}_x[f(x)] \approx 0$$
Pattern analysis algorithms

We would like algorithms to be:

- Computationally efficient – running time polynomial in the size of the data – often needs to be of a low degree

- Robust – able to handle noisy data, eg examples misclassified, noisy sensors or outputs only to a certain accuracy

- Statistical stability – able to distinguish between chance patterns and those characteristic of the underlying source of the data
Brief Historical Perspective

- Machine learning using neural like structures first considered seriously in 1960s with such systems as the Perceptron
  - Linear patterns
  - Simple learning algorithm
  - shown to be limited in complexity

- Resurrection of ideas in more powerful multi-layer perceptrons in 1980s
  - networks of perceptrons with continuous activation functions
  - very slow learning
  - limited statistical analysis
Kernel methods

Kernel methods (re)introduced in 1990s with Support Vector Machines

- Linear functions but in high dimensional spaces equivalent to non-linear functions in the input space
- Statistical analysis showing large margin can overcome curse of dimensionality
- Extensions rapidly introduced for many other tasks other than classification
Kernel methods approach

- Data embedded into a Euclidean feature space
- Linear relations are sought among the images of the data
- Algorithms implemented so that only require inner products between vectors
- Embedding designed so that inner products of images of two points can be computed directly by an efficient ‘short-cut’ known as the kernel.
The function $\phi$ embeds the data into a feature space where the non-linear pattern now appears linear. The kernel computes inner products in the feature space directly from the inputs.

$$\kappa(x, z) = \langle \phi(x), \phi(z) \rangle$$
Worked example: Ridge Regression

Consider the problem of finding a homogeneous real-valued linear function

\[ g(x) = \langle w, x \rangle = x'w = \sum_{i=1}^{n} w_i x_i, \]

that best interpolates a given training set

\[ S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \]

of points \( x_i \) from \( X \subseteq \mathbb{R}^n \) with corresponding labels \( y_i \) in \( Y \subseteq \mathbb{R} \).
Possible pattern function

- Measures discrepancy between function output and correct output – squared to ensure always positive:

\[ f_g((x, y)) = (g(x) - y)^2 \]

Note that the pattern function \( f_g \) is not itself a linear function, but a simple functional of the linear functions \( g \).

- We introduce notation: matrix \( X \) has rows the \( m \) examples of \( S \). Hence we can write

\[ \xi = y - Xw \]

for the vector of differences between \( g(x_i) \) and \( y_i \).

Optimising the choice of $g$

Need to ensure flexibility of $g$ is controlled – controlling the norm of $w$ proves effective:

$$\min_w \mathcal{L}_\lambda(w, S) = \min_w \lambda \|w\|^2 + \|\xi\|^2,$$

where we can compute

$$\|\xi\|^2 = \langle y - Xw, y - Xw \rangle = y'y - 2w'X'y + w'X'Xw$$

Setting derivative of $\mathcal{L}_\lambda(w, S)$ equal to 0 gives

$$X'Xw + \lambda w = (X'X + \lambda I_n) w = X'y$$
Primal solution

The term primal is used for the explicit representation in the feature space:

- We get the primal solution weight vector:

  \[ w = (X'X + \lambda I_n)^{-1} X'y \]

- and regression function

  \[ g(x) = x'w = x' (X'X + \lambda I_n)^{-1} X'y \]
**Dual solution**

A dual solution expresses the weight vector as a linear combination of the training examples:

\[ X'Xw + \lambda w = X'y \quad \text{implies} \]

\[ w = \frac{1}{\lambda} (X'y - X'Xw) = X'\frac{1}{\lambda} (y - Xw) = X'\alpha, \]

where

\[ \alpha = \frac{1}{\lambda} (y - Xw) \quad (1) \]

or equivalently

\[ w = \sum_{i=1}^{m} \alpha_i x_i \]

The vector \( \alpha \) is the dual solution.
Dual solution

Substituting $\mathbf{w} = \mathbf{X}'\alpha$ into equation (1) we obtain:

$$\lambda\alpha = y - \mathbf{XX}'\alpha$$

implying

$$(\mathbf{XX}' + \lambda\mathbf{I}_m)\alpha = y$$

This means the dual solution can be computed as:

$$\alpha = (\mathbf{XX}' + \lambda\mathbf{I}_m)^{-1}y$$

with the regression function

$$g(x) = \mathbf{x}'\mathbf{w} = \mathbf{x}'\mathbf{X}'\alpha = \left\langle \mathbf{x}, \sum_{i=1}^{m} \alpha_i \mathbf{x}_i \right\rangle = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$
Key ingredients of dual solution

**Step 1:** Compute

\[ \alpha = (K + \lambda I_m)^{-1} y \]

where \( K = XX' \) that is \( K_{ij} = \langle x_i, x_j \rangle \)

**Step 2:** Evaluate on new point \( x \) by

\[ g(x) = \sum_{i=1}^{m} \alpha_i \langle x, x_i \rangle \]

**Important observation:** Both steps only involve inner products between input data points
Applying the ‘kernel trick’

Since the computation only involves inner products, we can substitute for all occurrences of $\langle \cdot, \cdot \rangle$ a kernel function $\kappa$ that computes:

$$\kappa(x, z) = \langle \phi(x), \phi(z) \rangle$$

and we obtain an algorithm for ridge regression in the feature space $F$ defined by the mapping

$$\phi : x \mapsto \phi(x) \in F$$

Note if $\phi$ is the identity this has no effect.
A simple kernel example

The simplest non-trivial kernel function is the quadratic kernel:

\[ \kappa(x, z) = \langle x, z \rangle^2 \]

involving just one extra operation. But surprisingly this kernel function now corresponds to a complex feature mapping:

\[
\kappa(x, z) = (x'z)^2 = z'(xx')z
= \langle \text{vec}(zz'), \text{vec}(xx') \rangle
\]

where \( \text{vec}(A) \) stacks the columns of the matrix \( A \) on top of each other. Hence, \( \kappa \) corresponds to the feature mapping

\[
\phi : x \mapsto \text{vec}(xx')
\]
Implications of the kernel trick

- Consider for example computing a regression function over 1000 images represented by pixel vectors – say $32 \times 32 = 1024$ pixels.

- By using the quadratic kernel we implement the regression function in a $1,000,000$ dimensional space

- but actually using less computation for the learning phase than we did in the original space – inverting a $1000 \times 1000$ matrix instead of a $1024 \times 1024$ matrix.
Implications of kernel algorithms

- Can perform linear regression in very high-dimensional (even infinite dimensional) spaces efficiently.

- This is equivalent to performing non-linear regression in the original input space: for example quadratic kernel leads to solution of the form

\[ g(x) = \sum_{i=1}^{m} \alpha_i \langle x_i, x \rangle^2 \]

that is a quadratic polynomial function of the components of the input vector \( x \).

- Using these high-dimensional spaces must surely come with a health warning, what about the curse of dimensionality?
Defining kernels

- Natural to consider defining kernels for your data. Clearly, kernel must be symmetric and satisfy

\[ \kappa(x, x) > 0 \]

- BUT not every function satisfying these conditions is a kernel.

- Commonly used kernel is Gaussian kernel:

\[ \kappa(x, z) = \exp \left( -\frac{||x - z||}{2\sigma^2} \right) \]

corresponds to infinite dimensional feature space.
Means and distances

Suppose we are given a kernel function:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

and a training set $S$, what can we estimate?

- Consider some vector

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

we have

$$\|\mathbf{w}\|^2 = \left\langle \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i), \sum_{j=1}^{m} \alpha_j \phi(\mathbf{x}_j) \right\rangle = \sum_{i,j=1}^{m} \alpha_i \alpha_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$$
Means and distances

- Hence, we can normalise data in the feature space:
  \[ \phi(x) \mapsto \hat{\phi}(x) = \frac{\phi(x)}{\|\phi(x)\|} \]
  since we can compute the corresponding kernel \( \hat{\kappa} \) by

\[
\hat{\kappa}(x, z) = \left\langle \frac{\phi(x)}{\|\phi(x)\|}, \frac{\phi(z)}{\|\phi(z)\|} \right\rangle = \frac{\kappa(x, z)}{\sqrt{\kappa(x, x) \kappa(z, z)}}
\]
Means and distances

- Given two vectors:

\[ w_a = \sum_{i=1}^{m} \alpha_i \phi(x_i) \quad \text{and} \quad w_b = \sum_{i=1}^{m} \beta_i \phi(x_i) \]

we have

\[ w_a - w_b = \sum_{i=1}^{m} (\alpha_i - \beta_i) \phi(x_i) \]

so we can compute the distance between \( w_a \) and \( w_b \) as

\[ d(w_a, w_b) = \| w_a - w_b \| \]
Means and distances

- For example the norm of the mean of a sample is given by

\[ \| \phi_S \| = \left\| \frac{1}{m} \sum_{i=1}^{m} \phi(x_i) \right\| = \frac{1}{m} \sqrt{j'Kj} \]

where \( j \) is the all ones vector.

- Hence, expected squared distance to the mean of a sample is:

\[ \mathbb{E}[\| \phi(x) - \phi_S \|^2] = \frac{1}{m} \sum_{i=1}^{m} \kappa(x_i, x_i) - \langle \phi_S, \phi_S \rangle \]

\[ = \frac{1}{m} \text{tr}(K) - \frac{1}{m^2} j'Kj \]
Means and distances

- Consider centering the sample, i.e. moving the origin to the sample mean: this will result in

$$\|\phi_S\|^2 = \frac{1}{m^2} j'Kj = 0$$

in the new coordinate system, while the lhs of previous equation is unchanged by centering. Hence, centering minimises the trace.

- Centering is achieved by transformation:

$$\phi(x) \mapsto \hat{\phi}(x) = \phi(x) - \phi_S$$
Means and distances

• What is effect on kernel and kernel matrix?

\[ \hat{\kappa}(x, z) = \langle \hat{\phi}(x), \hat{\phi}(z) \rangle \]

\[ = \kappa(x, z) - \frac{1}{m} \sum_{i=1}^{m} (\kappa(x, x_i) + \kappa(z, x_i)) + \frac{1}{m^2} j'Kj \]

• Hence we can implement the centering of a kernel matrix by

\[ \hat{K} = K - \frac{1}{m} (jj'K + Kjj') + \frac{1}{m^2} (j'Kj)jj' \]
Simple novelty detection

• Consider putting a ball round the centre of mass $\phi_S$ of radius sufficient to contain all the data:

$$\|\phi(x) - \phi_S\| > \max_{1 \leq i \leq m} \|\phi(x_i) - \phi_S\|$$

• Give a kernel expression for this quantity.
OVERALL STRUCTURE

Part 1: Introduction to the Kernel methods approach.

Part 2: Projections and subspaces in the feature space.

Part 3: Other learning algorithms with the example of Support Vector Machines.

Part 4: Kernel design strategies.
Part 2 structure

• Simple classification algorithm

• Fisher discriminant analysis.

• Principal components analysis.
Simple classification algorithm

- Consider finding the centres of mass of positive and negative examples and classifying a test point by measuring which is closest

\[
h(x) = \text{sgn} \left( \| \phi(x) - \phi_{S-} \|^2 - \| \phi(x) - \phi_{S+} \|^2 \right)
\]

- we can express as a function of kernel evaluations

\[
h(x) = \text{sgn} \left( \frac{1}{m_+} \sum_{i=1}^{m_+} \kappa(x, x_i) - \frac{1}{m_-} \sum_{i=m_++1}^{m} \kappa(x, x_i) - b \right),
\]

where

\[
b = \frac{1}{2m_+^2} \sum_{i,j=1}^{m_+} \kappa(x_i, x_j) - \frac{1}{2m_-^2} \sum_{i,j=m_++1}^{m} \kappa(x_i, x_j)
\]
Simple classification algorithm

• equivalent to dividing the space with a hyperplane perpendicular to the line half way between the two centres with vector given by

\[ w = \frac{1}{m^+} \sum_{i=1}^{m^+} \phi(x_i) - \frac{1}{m^-} \sum_{i=m^++1}^{m} \phi(x_i) \]

• Function is the difference in likelihood of the Parzen window density estimators for positive and negative examples

• We will see some examples of the performance of this algorithm in Part 3.
Variance of projections

• Consider projections of the datapoints $\phi(x_i)$ onto a unit vector direction $v$ in the feature space: average is given by

$$\mu_v = \hat{E} [\|P_v(\phi(x))\|] = \hat{E} [v'\phi(x)] = v'\phi_S$$

of course this is 0 if the data has been centred.

• average squared is given by

$$\hat{E} [\|P_v(\phi(x))\|^2] = \hat{E} [v'\phi(x)\phi(x)'v] = \frac{1}{m}v'X'Xv$$
Variance of projections

• Now suppose \( \mathbf{v} \) has the dual representation \( \mathbf{v} = \mathbf{X}'\mathbf{\alpha} \). Average is given by

\[
\mu_{\mathbf{v}} = \frac{1}{m} \mathbf{\alpha}'\mathbf{X}\mathbf{x}'\mathbf{j} = \frac{1}{m} \mathbf{\alpha}'\mathbf{K}\mathbf{j}
\]

• Average squared is given by

\[
\frac{1}{m} \mathbf{v}'\mathbf{X}'\mathbf{X}\mathbf{v} = \frac{1}{m} \mathbf{\alpha}'\mathbf{XX}'\mathbf{X}'\mathbf{\alpha} = \frac{1}{m} \mathbf{\alpha}'\mathbf{K}^2\mathbf{\alpha}
\]

• Hence, variance in direction \( \mathbf{v} \) is given by

\[
\sigma_v^2 = \frac{1}{m} \mathbf{\alpha}^2\mathbf{K}^2\mathbf{\alpha} - \frac{1}{m^2}(\mathbf{\alpha}'\mathbf{K}\mathbf{j})^2
\]
Fisher discriminant

• The Fisher discriminant is a thresholded linear classifier:

\[ f(x) = \text{sgn}(\langle w, \phi(x) \rangle + b) \]

where \( w \) is chosen to maximise the quotient:

\[ J(w) = \frac{(\mu_w^+ - \mu_w^-)^2}{(\sigma_w^+)^2 + (\sigma_w^-)^2} \]

• As with Ridge regression is makes sense to regularise if we are working in high-dimensional kernel spaces, so maximise

\[ J(w) = \frac{(\mu_w^+ - \mu_w^-)^2}{(\sigma_w^+)^2 + (\sigma_w^-)^2 + \lambda \| w \|^2} \]
Fisher discriminant

- Using the results we now have we can substitute dual expressions for all of these quantities and solve using lagrange multipliers.

- The resulting classifier has dual variables

\[ \alpha = (B K + \lambda I)^{-1} y \]

where \( B = D - C \) with

\[ C_{ij} = \begin{cases} 
2m^-/(mm^+) & \text{if } y_i = 1 = y_j \\
2m^+/(mm^-) & \text{if } y_i = -1 = y_j \\
0 & \text{otherwise}
\end{cases} \]
and

\[ D = \begin{cases} 
2m^-/m & \text{if } i = j \text{ and } y_i = 1 \\
2m^+/m & \text{if } i = j \text{ and } y_i = -1 \\
0 & \text{otherwise}
\end{cases} \]

and \( b = 0.5\alpha K_t \) with

\[ t_i = \begin{cases} 
1/m^+ & \text{if } y_i = 1 \\
1/m^- & \text{if } y_i = -1 \\
0 & \text{otherwise}
\end{cases} \]

giving a decision function

\[
f(x) = \text{sgn} \left( \sum_{i=1}^{m} \alpha_i \kappa(x_i, x) - b \right)
\]
Overview of remainder of tutorial

- Plug and play aspects of kernel methods:
  
  Data → kernel → preprocessing → pattern analysis

- Part 2: preprocessing: for example normalisation, projection into subspaces, kernel PCA, kernel CCA, etc.

- Part 3: pattern analysis: support vector machines, novelty detection, support vector regression.

- Part 4: kernel design: properties of kernels, kernels for text, string kernels.
Preprocessing

- Corresponds to feature selection, or learning the feature space

- Note that in kernel methods the feature space is only determined up to orthogonal transformations (change of basis):

\[
\hat{\phi}(x) = U\phi(x)
\]

for some orthogonal transformation \( U \) (\( U'U = I = UU' \)), then

\[
\hat{\kappa}(x, z) = \langle U\phi(x), U\phi(z) \rangle = \phi(x)'U'U\phi(z) = \phi(x)'\phi(z) = \kappa(x, z)
\]

- so feature selection is equivalent to subspace projection in kernel defined feature spaces
Subspace methods

• **Principal components analysis**: choose directions to maximise variance in the training data

• **Canonical correlation analysis**: choose directions to maximise correlations between two different views of the same objects

• **Gram-Schmidt**: greedily choose directions according to largest residual norms

• **Partial least squares**: greedily choose directions with maximal covariance with the target (will not cover this)

In all cases we need kernel versions in order to apply these methods in high-dimensional kernel defined feature spaces
Principal Components Analysis

- PCA is a subspace method – that is it involves projecting the data into a lower dimensional space.

- Subspace is chosen to ensure maximal variance of the projections:

  \[ w = \text{argmax}_{w: \|w\|=1} w'X'Xw \]

- This is equivalent to maximising the Raleigh quotient:

  \[ \frac{w'X'Xw}{w'w} \]
Principal Components Analysis

- We can optimise using Lagrange multipliers in order to remove the contraints:

\[ L(w, \lambda) = w'X'Xw - \lambda w'w \]

Taking derivatives wrt \( w \) and setting equal to 0 gives:

\[ X'Xw = \lambda w \]

implying \( w \) is an eigenvalue of \( X'X \).

- Note that

\[ \lambda = w'X'Xw = \sum_{i=1}^{m} \langle w, x_i \rangle^2 \]
Principal Components Analysis

- So principal components analysis performs an eigenvalue decomposition of $X'X$ and projects into the space spanned by the first $k$ eigenvectors

- Captures a total of

$$\sum_{i=1}^{k} \lambda_i$$

of the overall variance:

$$\sum_{i=1}^{m} \|x_i\|^2 = \sum_{i=1}^{n} \lambda_i = \text{tr}(K)$$
Kernel PCA

- We would like to find a dual representation of the principal eigenvectors and hence of the projection function.

- Suppose that \( w, \lambda \neq 0 \) is an eigenvector/eigenvalue pair for \( X'X \), then \( Xw, \lambda \) is for \( XX' \):

\[
(XX')Xw = X(X'X)w = \lambda Xw
\]

- and vice versa \( \alpha, \lambda \rightarrow X'\alpha, \lambda \)

\[
(X'X)X'\alpha = X'(XX')\alpha = \lambda X'\alpha
\]

- Note that we get back to where we started if we do it twice.
Kernel PCA

• Hence, 1-1 correspondence between eigenvectors corresponding to non-zero eigenvalues, but note that if $\|\alpha\| = 1$

$$\|X'\alpha\|^2 = \alpha'XX'\alpha = \alpha'K\alpha = \lambda$$

so if $\alpha^i, \lambda_i, i = 1, \ldots, k$ are first $k$ eigenvectors/values of $K$

$$\frac{1}{\sqrt{\lambda_i}}\alpha^i$$

are dual representations of first $k$ eigenvectors $w^1, \ldots, w^k$ of $X'X$ with same eigenvalues.

• Computing projections:

$$\langle w^i, \phi(x) \rangle = \frac{1}{\sqrt{\lambda_i}}\langle X'\alpha^i, \phi(x) \rangle = \frac{1}{\sqrt{\lambda_i}} \sum_{j=1}^{m} \alpha^i_j \kappa(x_i, x)$$
OVERALL STRUCTURE

Part 1: Introduction to the Kernel methods approach.

Part 2: Projections and subspaces in the feature space.

Part 3: Other learning algorithms with the example of Support Vector Machines.

Part 4: Kernel design strategies.
Part 3 structure

- Perceptron algorithm
- Generalisation of SVMs
- Support Vector Machine Optimisation
- Novelty detection
Kernel algorithms

• Have already seen three kernel based algorithms:
  – Ridge regression
  – Fisher discriminant
  – Simple novelty detector

• Key properties that enable an algorithm to be kernelised:
  – Must reduce to estimating a linear function in the feature space
  – Weight vector must be in the span of the training examples
  – Algorithm to find dual coefficients only involves inner products of training data

• Very simple example: perceptron algorithm
Perceptron algorithm

- initialise $w \leftarrow 0$

- repeat if for some example: $y_i \langle w, \phi(x_i) \rangle \leq 0$

  $$w \leftarrow w + y_i \phi(x_i)$$

- Clearly dual version:
  - initialise $\alpha_i \leftarrow 0$ for all $i$
  - update: $\alpha_i \leftarrow \alpha_i + y_i$
  - Note can evaluate as as for ridge regression:

  $$\langle w, \phi(x) \rangle = \sum_i \alpha_i \kappa(x_i, x)$$

- Dual version by Aizerman et al. (1964) but tends to overfit
Margin Perceptron algorithm

- Margin version if replace test by

\[ y_i \langle w, \phi(x_i) \rangle \leq \tau \]

- Set \( \tau = 1 \) – in 1964 one parameter away from a kernel classification algorithm able to generalise in high dimensions
Support Vector Machines (SVM)

• SVM seeks linear function in a feature space defined implicitly via a kernel $\kappa$:

$$\kappa(x, z) = \langle \phi(x), \phi(z) \rangle$$

that optimises a bound on the generalisation.

• Several bounds on the performance of SVMs exist all use the margin to give an empirically defined complexity: data-dependent structural risk minimisation

• Tightest is the PAC-Bayes bound
Margins in SVMs

• Critical to the bound will be the margin of the classifier

\[ \gamma(x, y) = yg(x) = y(\langle w, \phi(x) \rangle + b) : \]

positive if correctly classified, and measures distance from the separating hyperplane when the weight vector is normalised.

• The margin of a linear function \( g \) is

\[ \gamma(g) = \min_i \gamma(x_i, y_i) \]

though this is frequently increased to allow some ‘margin errors’.
Margins in SVMs
Form of the SVM bound

- If we define the inverse of the KL by

\[ KL^{-1}(q, A) = \max \{ p : KL(q\|p) \leq A \} \]

then have with probability at least \(1 - \delta\) for all \(\mu\)

\[ Pr(\langle w, \phi(x) \rangle \neq y) \leq 2 \min_{\mu} KL^{-1}\left( \mathbb{E}_m[\tilde{F}(\mu \gamma(x, y))], \frac{\mu^2/2 + \ln \frac{m+1}{\delta}}{m} \right) \]

where \(\tilde{F}(t) = 1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{t} e^{-x^2/2} dx\).
Slack variable conversion

Bound and slack variable used in optimisation
Gives SVM Optimisation

- Primal form:

\[
\begin{align*}
\min_{\mathbf{w}, \xi_i} & \ [\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \xi_i] \\
\text{s.t.} & \ y_i \mathbf{w}^T \phi(x_i) \geq 1 - \xi_i \quad i = 1, \ldots, m \\
& \quad \xi_i \geq 0 \quad i = 1, \ldots, m
\end{align*}
\]

- Dual form:

\[
\begin{align*}
\max_{\alpha} \left[ \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \kappa(x_i, x_j) \right] \\
\text{s.t.} & \ 0 \leq \alpha_i \leq C \quad i = 1, \ldots, m
\end{align*}
\]

where \( \kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \) and \( \langle \mathbf{w}, \phi(x) \rangle = \sum_{i=1}^{m} \alpha_i y_i \kappa(x_i, x) \).
Dual form of the SVM problem

Decision boundary and $\gamma$ margin for 1-norm svm with a gaussian kernel:
Novelty detection

We can also motivate novelty detection by a similar analysis as that for SVM: consider a hypersphere centred at $c$ of radius $r$ and the function $g$:

$$
g(x) = \begin{cases} 
0, & \text{if } \|c - \phi(x)\| \leq r; \\
(\|c - \phi(x)\|^2 - r^2)/\gamma, & \text{if } r^2 \leq \|c - \phi(x)\|^2 \leq r^2 + \gamma; \\
1, & \text{otherwise}.
\end{cases}
$$

with probability at least $1 - \delta$

$$
\mathbb{E}[g(x)] \leq \hat{\mathbb{E}}[g(x)] + \frac{6R^2}{\gamma\sqrt{m}} + 3\sqrt{\frac{\ln(2/\delta)}{2m}}
$$

Note that tension is between creating a tight bound and defining a small sphere.
Novelty detection

Let

$$\xi_i = (\|c - \phi(x)\|^2 - r^2)_+$$

so that

$$\hat{E}[g(x)] \leq \frac{1}{\gamma m} \|\xi\|_1$$

Treating $\gamma$ as fixed we minimise the bound by minimising $\|\xi\|_1$ and $r$:

$$\min_{c, r, \xi} \quad r^2 + C \|\xi\|_1$$
subject to

$$\|\phi(x_i) - c\|^2 \leq r^2 + \xi_i$$
$$\xi_i \geq 0, \quad i = 1, \ldots, m$$
Novelty detection

Dual optimisation maximise

\[ W(\alpha) = \sum_{i=1}^{m} \alpha_i \kappa(x_i, x_i) - \sum_{i,j=1}^{m} \alpha_i \alpha_j \kappa(x_i, x_j) \]

subject to \( \sum_{i=1}^{m} \alpha_i = 1 \) and \( 0 \leq \alpha_i \leq C, i = 1, \ldots, m. \)

with final novelty test being:

\[ f(\cdot) = \mathcal{H} \left[ \kappa(\cdot, \cdot) - 2 \sum_{i=1}^{m} \alpha_i^* \kappa(x_i, \cdot) + D \right] \]

where

\[ D = \sum_{i,j=1}^{m} \alpha_i^* \alpha_j^* \kappa(x_i, x_j) - (r^*)^2 - \gamma \]
Novelty detection
OVERALL STRUCTURE

Day 1: Introduction to the Kernel methods approach.

Day 2: Projections and subspaces in the feature space.

Day 3: Other learning algorithms with the example of Support Vector Machines.

Day 4: Kernel design strategies.
Part 4 structure

- Kernel design strategies.
- Kernels for text and string kernels.
- Kernels for other structures.
- Kernels from generative models.
Kernel functions

- Already seen some properties of kernels:
  - symmetric:

\[
\kappa(x, z) = \langle \phi(x), \phi(z) \rangle = \langle \phi(z), \phi(x) \rangle = \kappa(z, x)
\]

- kernel matrices psd:

\[
u'Ku = \sum_{i,j=1}^{m} u_i u_j \langle \phi(x_i), \phi(x_j) \rangle
\]

\[
= \left\langle \sum_{i=1}^{m} u_i \phi(x_i), \sum_{j=1}^{m} u_j \phi(x_j) \right\rangle
\]

\[
= \left\| \sum_{i=1}^{m} u_i \phi(x_i) \right\|^2 \geq 0
\]
Kernel functions

• These two properties are all that is required for a kernel function to be valid: symmetric and every kernel matrix is psd.

• Note that this is equivalent to all eigenvalues non-negative – recall that eigenvalues of the kernel matrix measured the sum of the squares of the projections onto the eigenvector.

• If we have uncountable domains should also have continuity, though there are exceptions to this as well.
Kernel functions

Proof outline:

• Define feature space as class of functions:

\[ \mathcal{F} = \left\{ \sum_{i=1}^{m} \alpha_i \kappa(x_i, \cdot) : m \in \mathbb{N}, x_i \in X, \alpha_i \in \mathbb{R}, i = 1, \ldots, m \right\} \]

• Linear space

• embedding given by

\[ x \mapsto \kappa(x, \cdot) \]
Kernel functions

- inner product between

\[ f(x) = \sum_{i=1}^{m} \alpha_i \kappa(x_i, x) \quad \text{and} \quad g(x) = \sum_{i=1}^{n} \beta_i \kappa(z_i, x) \]

defined as

\[ \langle f, g \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_i \beta_j \kappa(x_i, z_j) = \sum_{i=1}^{m} \alpha_i g(x_i) = \sum_{j=1}^{n} \beta_j f(z_j), \]

- well-defined

- \( \langle f, f \rangle \geq 0 \) by psd property.
Kernel functions

- so-called reproducing property:

\[ \langle f, \phi(x) \rangle = \langle f, \kappa(x, \cdot) \rangle = f(x) \]

- implies that inner product corresponds to function evaluation – learning a function corresponds to learning a point being the weight vector corresponding to that function:

\[ \langle w_f, \phi(x) \rangle = f(x) \]
Kernel constructions

For $\kappa_1, \kappa_2$ valid kernels, $\phi$ any feature map, $\mathbf{B}$ psd matrix, $a \geq 0$ and $f$ any real valued function, the following are valid kernels:

- $\kappa(x, z) = \kappa_1(x, z) + \kappa_2(x, z),$
- $\kappa(x, z) = a\kappa_1(x, z),$
- $\kappa(x, z) = \kappa_1(x, z)\kappa_2(x, z),$
- $\kappa(x, z) = f(x)f(z),$
- $\kappa(x, z) = \kappa_1(\phi(x), \phi(z)),$
- $\kappa(x, z) = x^T\mathbf{B}z.$
Kernel constructions

Following are also valid kernels:

- \( \kappa(x, z) = p(\kappa_1(x, z)) \), for \( p \) any polynomial with positive coefficients.

- \( \kappa(x, z) = \exp(\kappa_1(x, z)) \),

- \( \kappa(x, z) = \exp(-\|x - z\|^2/(2\sigma^2)) \).

Proof of third: normalise the second kernel:

\[
\frac{\exp(\langle x, z \rangle / \sigma^2)}{\sqrt[\psi^2]{\exp(\|x\|^2 / \sigma^2)}} = \exp \left( \frac{\langle x, z \rangle}{\sigma^2} - \frac{\langle x, x \rangle}{2\sigma^2} - \frac{\langle z, z \rangle}{2\sigma^2} \right) = \exp \left( -\frac{\|x - z\|^2}{2\sigma^2} \right).
\]
Subcomponents kernel

For the kernel $\langle x, z \rangle^s$ the features can be indexed by sequences

$$i = (i_1, \ldots, i_n), \sum_{j=1}^{n} i_j = s$$

where

$$\phi_i(x) = x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$$

A similar kernel can be defined in which all subsets of features occur:

$$\phi : x \mapsto (\phi_A(x))_{A \subseteq \{1, \ldots, n\}}$$

where

$$\phi_A(x) = \prod_{i \in A} x_i$$
Subcomponents kernel

So we have

\[ \kappa_{\subseteq}(x, y) = \langle \phi(x), \phi(y) \rangle \]

\[ = \sum_{A \subseteq \{1, \ldots, n\}} \phi_A(x) \phi_A(y) \]

\[ = \sum_{A \subseteq \{1, \ldots, n\}} \prod_{i \in A} x_i y_i = \prod_{i=1}^{n}(1 + x_i y_i) \]

Can represent computation with a graph:

Each path in the graph corresponds to a feature.
Graph kernels

Can also represent polynomial kernel

\[ \kappa(x, y) = \left( \langle x, y \rangle + R \right)^d = (x_1y_1 + x_2y_2 + \cdots + x_ny_n + R)^d \]

with a graph:
Graph kernels

The ANOVA kernel is represented by the graph:

Chicago/TTI Summer School, June 2009
ANOVA kernels

Features are all the combinations of exactly $d$ distinct features, while computation is given by recursion:

$$
\kappa_0^m(x, z) = 1, \text{ if } m \geq 0,
$$

$$
\kappa_s^m(x, z) = 0, \text{ if } m < s,
$$

$$
\kappa_s^m(x, z) = (x_m z_m) \kappa_{s-1}^{m-1}(x, z) + \kappa_{s-1}^{m-1}(x, z)
$$

While the resulting kernel is given by

$$
\kappa_d^n(x, z)
$$

in the bottom right corner of the graph.
General Graph kernels

- Defined over directed acyclic graphs (DAGs)

- Number vertices $1, \ldots, s$ compatible with edge directions, i.e. $u_i \rightarrow u_j \implies i < j$.

- Compute using dynamic programming table DP

  - Initialise $DP(1) = 1$;

  - For $i = 2, \ldots, s$ compute

    $$DP(i) = \sum_{j \rightarrow i} \kappa_{(u_j \rightarrow u_i)} (x, z) \cdot DP(j)$$

- Result given at output node $s$: $\kappa(x, z) = DP(s)$.
Kernels for text

- The simplest representation for text is the kernel given by the feature map known as the vector space model

\[ \phi : d \mapsto \phi(d) = (\text{tf}(t_1, d), \text{tf}(t_2, d), \ldots, \text{tf}(t_N, d))' \]

where \( t_1, t_2, \ldots, t_N \) are the terms occurring in the corpus and \( \text{tf}(t, d) \) measures the frequency of term \( t \) in document \( d \).

- Usually use the notation \( \mathbf{D} \) for the document term matrix (cf. \( \mathbf{X} \) from previous notation).
Kernels for text

• Kernel matrix is given by

\[ K = DD' \]

wrt kernel

\[ \kappa(d_1, d_2) = \sum_{j=1}^{N} \text{tf}(t_j, d_1) \text{tf}(t_j, d_2) \]

• despite high-dimensionality kernel function can be computed efficiently by using a linked list representation.
Semantics for text

- The standard representation does not take into account the importance or relationship between words.

- Main methods do this by introducing a ‘semantic’ mapping $S$:

  $$\hat{\kappa}(d_1, d_2) = \phi(d_1)^T S S' \phi(d_2)$$
Semantics for text

- Simplest is diagonal matrix giving term weightings (known as inverse document frequency – tfidf):

\[ w(t) = \ln \frac{m}{df(t)} \]

- Hence kernel becomes:

\[ \kappa(d_1, d_2) = \sum_{j=1}^{N} w(t_j)^2 tf(t_j, d_1) tf(t_j, d_2) \]
Semantics for text

• In general would also like to include semantic links between terms with off-diagonal elements, eg stemming, query expansion, wordnet.

• More generally can use co-occurrence of words in documents:

\[ S = D' \]

so

\[ (SS')_{ij} = \sum_d tf(i, d)tf(j, d) \]
Semantics for text

- Information retrieval technique known as latent semantic indexing uses SVD decomposition:

\[ D' = U \Sigma V' \]

so that

\[ d \mapsto U_k \phi(d) \]

which is equivalent to performing kernel PCA to give latent semantic kernels:

\[ \tilde{\kappa}(d_1, d_2) = \phi(d_1)'U_k U_k' \phi(d_2) \]
String kernels

- Consider the feature map given by

\[ \phi^p_u(s) = |\{(v_1, v_2) : s = v_1uv_2\}| \]

for \( u \in \Sigma^p \) with associated kernel

\[ \kappa_p(s, t) = \sum_{u \in \Sigma^p} \phi^p_u(s) \phi^p_u(t) \]
String kernels

- Consider the following two sequences:

\[ s = \text{"statistics"} \]
\[ t = \text{"computation"} \]

The two strings contain the following substrings of length 3:

- "sta", "tat", "ati", "tis",
- "ist", "sti", "tic", "ics",
- "com", "omp", "mpu", "put",
- "uta", "tat", "ati", "tio", "ion"

and they have in common the substrings "tat" and "ati", so their inner product would be \( \kappa_3(s, t) = 2. \)
Trie based p-spectrum kernels

- Computation organised into a trie with nodes indexed by substrings – root node by empty string \( \epsilon \).

- Create lists of substrings at root node:

\[
L_s(\epsilon) = \{(s(i : i + p - 1), 0) : i = 1, |s| - p + 1\}
\]

Similarly for \( t \).

- Recursively through the tree: if \( L_s(v) \) and \( L_t(v) \) both not empty:
  - for each \((u, i) \in L_*(v)\) add \((u, i + 1)\) to list \( L_*(vu_{i+1}) \)

- At depth \( p \) increment global variable \( \text{kern} \) initialised to 0 by \(|L_s(v)||L_t(v)|\).
Gap weighted string kernels

- Can create kernels whose features are all substrings of length \( p \) with the feature weighted according to all occurrences of the substring as a subsequence:

<table>
<thead>
<tr>
<th>( \phi )</th>
<th>ca</th>
<th>ct</th>
<th>at</th>
<th>ba</th>
<th>bt</th>
<th>cr</th>
<th>ar</th>
<th>br</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>( \lambda^2 )</td>
<td>( \lambda^3 )</td>
<td>( \lambda^2 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>car</td>
<td>( \lambda^2 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \lambda^3 )</td>
<td>( \lambda^2 )</td>
<td>0</td>
</tr>
<tr>
<td>bat</td>
<td>0</td>
<td>0</td>
<td>( \lambda^2 )</td>
<td>( \lambda^2 )</td>
<td>( \lambda^3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bar</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \lambda^2 )</td>
<td>0</td>
<td>0</td>
<td>( \lambda^2 )</td>
<td>( \lambda^3 )</td>
</tr>
</tbody>
</table>

- This can be evaluated using a dynamic programming computation over arrays indexed by the two strings.
Tree kernels

- We can consider a feature mapping for trees defined by

\[ \phi : T \mapsto (\phi_S(T))_{S \in I} \]

where \( I \) is a set of all subtrees and \( \phi_S(T) \) counts the number of co-rooted subtrees isomorphic to the tree \( S \).

- The computation can again be performed efficiently by working up from the leaves of the tree integrating the results from the children at each internal node.

- Similarly we can compute the inner product in the feature space given by all subtrees of the given tree not necessarily co-rooted.
Probabilistic model kernels

- There are two types of kernels that can be defined based on probabilistic models of the data.

- The most natural is to consider a class of models index by a model class $M$: we can then define the similarity as

$$\kappa(x, z) = \sum_{m \in M} P(x|m)P(z|m)P_M(m)$$

also known as the marginalisation kernel.

- For the case of Hidden Markov Models this can be again be computed by a dynamic programming technique.
Probabilistic model kernels

- Pair HMMs generate pairs of symbols and under mild assumptions can also be shown to give rise to kernels that can be efficiently evaluated.

- Similarly hidden tree generating models of data, again using a recursion that works upwards from the leaves.
Fisher kernels

Fisher kernels are an alternative way of defining kernels based on probabilistic models.

• We assume the model is parametrised according to some parameters: consider the simple example of a 1-dim Gaussian distribution parametrised by \( \mu \) and \( \sigma \):

\[
M = \left\{ P(x|\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left( -\frac{(x-\mu)^2}{2\sigma^2} \right) : \theta = (\mu, \sigma) \in \mathbb{R}^2 \right\}.
\]

• The Fisher score vector is the derivative of the log likelihood of an input \( x \) wrt the parameters:

\[
\log \mathcal{L}_{(\mu,\sigma)} (x) = -\frac{(x-\mu)^2}{2\sigma^2} - \frac{1}{2} \log (2\pi\sigma).
\]
Fisher kernels

• Hence the score vector is given by:

\[ g(\theta^0, x) = \left( \frac{x - \mu_0}{\sigma_0^2}, \frac{(x - \mu_0)^2}{\sigma_0^3} - \frac{1}{2\sigma_0} \right). \]

• Taking \( \mu_0 = 0 \) and \( \sigma_0 = 1 \) the feature embedding is given by:
Fisher kernels

Chicago/TTI Summer School, June 2009
Fisher kernels

Can compute Fisher kernels for various models including

- ones closely related to string kernels
- mixtures of Gaussians
- Hidden Markov Models
Conclusions

Kernel methods provide a general purpose toolkit for pattern analysis

- kernels define flexible interface to the data enabling the user to encode prior knowledge into a measure of similarity between two items – with the proviso that it must satisfy the psd property.

- composition and subspace methods provide tools to enhance the representation: normalisation, centering, kernel PCA, kernel Gram-Schmidt, kernel CCA, etc.

- algorithms well-founded in statistical learning theory enable efficient and effective exploitation of the high-dimensional representations to enable good off-training performance.
Where to find out more

**Web Sites:**  www.support-vector.net (SV Machines)

www.kernel-methods.net (kernel methods)

www.kernel-machines.net (kernel Machines)

www.pascal-network.org

**References**


