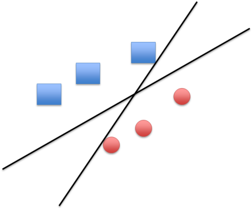
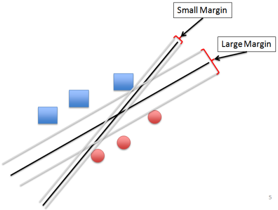
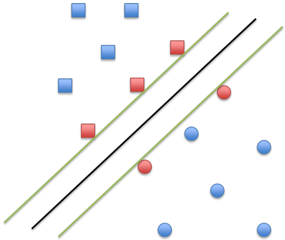
Kernel methods



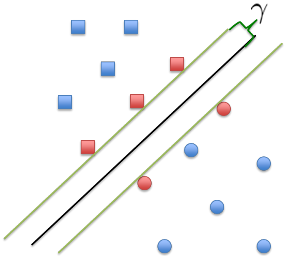
- Perceptron (and other linear classifiers) can lead to many equally valid choices for the decision boundary
- Are these really equally valid ?
- How can we pick which is best?



- Perceptron (and other linear classifiers) can lead to many equally valid choices for the decision boundary
- Are these really equally valid ?
- How can we pick which is best?
  $\rightarrow$ Maximize the size of the margin



- Support Vectors are those input points (vectors) closest to the decision boundary
- decision problem: $w^T x + b = 0$

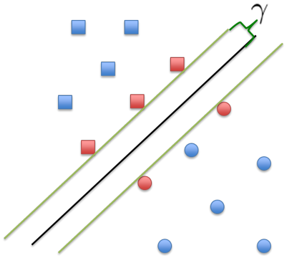


Notations

- x_i: data ; $y_i \in \{-1, 1\}$: labels
- decision hyperplane: $w^T x + b = 0$
- decision function : $f(x) = Sign(w^T x + b)$
- Margin hyperplanes: $w^T x + b = \pm\gamma$
- Scale invariance: $\lambda w^T x + \lambda b = 0$.

Scaling

This scaling does not change the decision hyperplane, or the support vector hyperplanes. $\Rightarrow$ Margin hyperplanes: $w^T x + b = \pm 1$



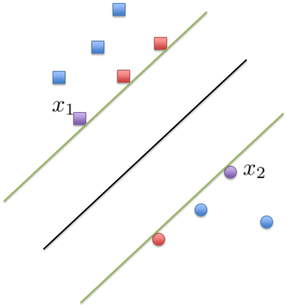
NOTATIONS

- x_i: data ; $y_i \in \{-1, 1\}$: labels
- decision hyperplane: $w^T x + b = 0$
- decision function : $f(x) = Sign(w^T x + b)$
- Margin hyperplanes: $w^T x + b = \pm\gamma$
- Scale invariance: $\lambda w^T x + \lambda b = 0$.

SCALING

This scaling does not change the decision hyperplane, or the support vector hyperplanes. $\Rightarrow$ Margin hyperplanes: $w^T x + b = \pm 1$

What are we optimizing ?



Size of the margin

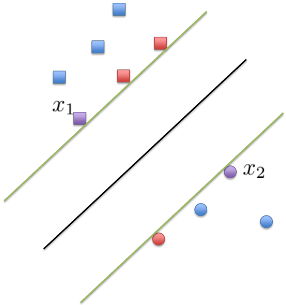
represented in terms of w.

1. identification of a decision boundary
2. maximization of the margin

Relation margin $\leftrightarrow w$

At least one point that lies on each support hyperplanes. $w^T x_1 + b = 1$ and $w^T x_2 + b = -1$ $\Rightarrow w^T(x_1 - x_2) = 2$

WHAT ARE WE OPTIMIZING ?



SIZE OF THE MARGIN

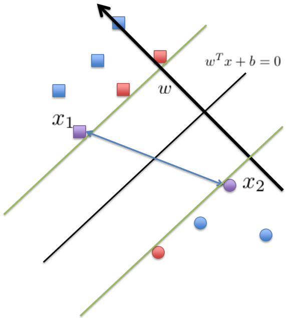
represented in terms of w.

1. identification of a decision boundary
2. maximization of the margin

RELATION MARGIN $\leftrightarrow w$

At least one point that lies on each support hyperplanes. $w^T x_1 + b = 1$ and $w^T x_2 + b = -1$
$\Rightarrow w^T(x_1 - x_2) = 2$

What are we optimizing ?



$w^T(x_1 - x_2) = 2$

1. w: orthogonal to the decision hyperplane
2. margin: projection of $x_1 - x_2$ onto w,

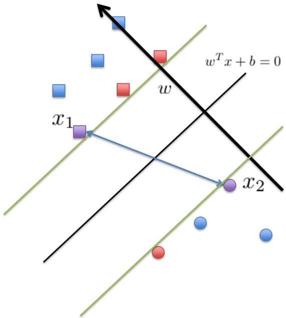
Projection

$w^T(x_1 - x_2) = 2$

Projection: $\frac{w^T(x_1 - x_2)}{||w||} w$

Size of the margin: $\frac{2}{||w||}$

What are we optimizing ?



$w^T(x_1 - x_2) = 2$

1. w: orthogonal to the decision hyperplane
2. margin: projection of $x_1 - x_2$ onto w,

Projection

$w^T(x_1 - x_2) = 2$

Projection: $\frac{w^T(x_1 - x_2)}{||w||} w$

Size of the margin: $\frac{2}{||w||}$

Maximizing the margin

Maximization

$Max \frac{2}{\|w\|}$
subject to $\forall i \quad y_i(w^T x_i + b) \geq 1$

Minimization

$Min \|w\|$
subject to $\forall i \quad y_i(w^T x_i + b) \geq 1$

Lagrangian relaxation

$$L(w,b) = \frac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i \left[y_i(w^T x_i + b) - 1 \right]$$

Maximizing the margin

Maximization

$Max \frac{2}{\|w\|}$
subject to $\forall i \quad y_i(w^T x_i + b) \geq 1$

Minimization

$Min\|w\|$
subject to $\forall i \quad y_i(w^T x_i + b) \geq 1$

Lagrangian relaxation

$$L(w,b) = \frac{1}{2}w^T w - \sum_{i=1}^{N} \alpha_i \left[y_i(w^T x_i + b) - 1 \right]$$

Maximizing the margin

Maximization

$Max \frac{2}{\|w\|}$
subject to $\forall i \quad y_i(w^T x_i + b) \geq 1$

Minimization

$Min\|w\|$
subject to $\forall i \quad y_i(w^T x_i + b) \geq 1$

Lagrangian relaxation

$$L(w, b) = \frac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i \left[y_i(w^T x_i + b) - 1 \right]$$

Max Margin Loss Function

Primal problem

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0$$

DUAL PROBLEM

DUAL PROBLEM

Now have to find α_i: substitute back to the loss function

$$L(w,b) = \frac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i \left[y_i (w^T x_i + b) - 1 \right]$$

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i$$

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

where $\alpha_i \geq 0$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$

Dual formulation of the error

Primal problem

Optimize this quadratic program to identify the lagrange multipliers and thus the weights

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

where $\alpha_i \geq 0$

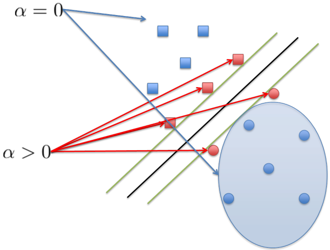
Support Vector Expansion

$$
\begin{aligned}
f(x) &= Sign(w^T x + b) \\
&= Sign\left(\left[\sum_{i=1}^{N} \alpha_i y_i x_i\right]^T x + b\right) \\
&= Sign\left(\left[\sum_{i=1}^{N} \alpha_i y_i x_i^T x\right] + b\right)
\end{aligned}
$$

- When α_i is non-zero then x_i is a support vector
- When α_i is zero x_i is not a support vector

Remark: $w = \sum_{i=1}^{N} \alpha_i y_i x_i$ Independent of the dimension of x_i

Kuhn-Tucker Conditions



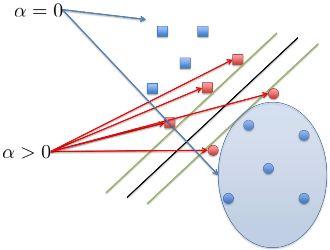
At the optimal solution

$$\alpha_i(1 - y_i(w^T x_i + b)) = 0$$

If $\alpha_i \neq 0$: $y_i(w^T x_i + b) = 1$

$\Rightarrow$ Only points on the decision boundary contribute to the solution.

KUHN-TUCKER CONDITIONS



AT THE OPTIMAL SOLUTION

$$\alpha_i(1 - y_i(w^T x_i + b)) = 0$$

If $\alpha_i \neq 0 : y_i(w^T x_i + b) = 1$

$\Rightarrow$ Only points on the decision boundary contribute to the solution.

Interpretability of SVM parameters

- α_i large $\Rightarrow$ the associated data point is quite important.
- It's either an outlier, or incredibly important

But this only gives us the best solution for linearly separable data sets

```
from sklearn.svm import LinearSVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
X, y = ...
svm = make_pipeline(StandardScaler(),LinearSVC(random_state=0, tol=1e-5))
svm.fit(X, y)
```

Learning Theory bases of SVMs

Bounds

Theoretical bounds on testing error:

→ The upper bound doesn't depend on the dim of the space
→ The lower bound is maximized by maximizing the margin associated with the decision boundary

Properties of SVM

→ Good generalization capability
→ Decision boundary is based on the data in the form of the support vectors → easy to interpret
→ Principled bounds on testing error from Learning Theory (VC dimension)

Learning Theory bases of SVMs

Bounds

Theoretical bounds on testing error:

- → The upper bound doesn't depend on the dim of the space
- → The lower bound is maximized by maximizing the margin associated with the decision boundary

Properties of SVM

- → Good generalization capability
- → Decision boundary is based on the data in the form of the support vectors → easy to interpret
- → Principled bounds on testing error from Learning Theory (VC dimension)

Soft margin classification

Outliers

- There can be outliers on the other side of the decision boundary, or leading to a small margin.
- ⇒ Introduce a penalty term to the constraint function

New function

$$Min \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i$$

s.c.

$$\begin{aligned} y_i(w^T x_i + b) &\geq 1 - \xi_i \\ \xi_i &\geq 0 \end{aligned}$$

Soft margin classification

Outliers

- There can be outliers on the other side of the decision boundary, or leading to a small margin.
- ⇒ Introduce a penalty term to the constraint function

New function

$$Min \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i$$

s.c.

$$\begin{aligned} y_i(w^T x_i + b) &\geq 1 - \xi_i \\ \xi_i &\geq 0 \end{aligned}$$

Lagrangian

$$L(w,b) = \frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left[y_i(w^T x_i + b) + \xi_i - 1\right]$$

Soft margin classification

New function

$Min \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i$

s.c.

$$\begin{aligned} y_i(w^T x_i + b) &\geq 1 - \xi_i \\ \xi_i &\geq 0 \end{aligned}$$

$$L(w,b) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i \left[y_i(w^T x_i + b) + \xi_i - 1 \right]$$

Soft margin classification

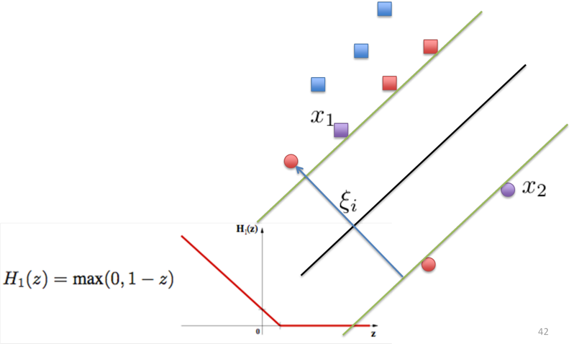
Still quadratic programming

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j$$

where $0 \leq \alpha_i \leq C$

$\sum_{i=1}^{N} \alpha_i y_i = 0$

Soft margin Example



H_1: hinge loss

```
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
X, y = ...
svm = make_pipeline(StandardScaler(),SVC(random_state=0, tol=1e-5))
svm.fit(X, y)
```

Multiclass

So far, binary classification problem. What about multiclass ?

Key ideas

- Decompose into multiple binary classification problems
- Make a final decision based on these binary classifiers

Two strategies

- One versus all
- One versus one

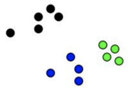
One vs all

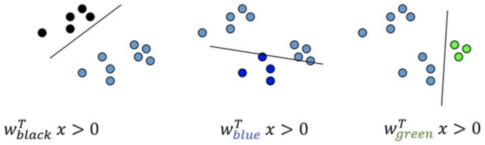
For C classes, decompose into C binary classification problems
For problem c:

- positive examples: x_i such that $y_i = c$
- negative examples: all others elements.

Inference: Winner takes all: for an example x

$$\hat{c} = arg \max_{c \in [\![1,c]\!]} w_c^T x$$



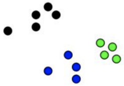


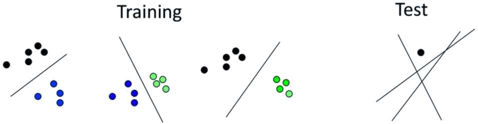
One vs one

For C classes, decompose into $C(C+1)/2$ binary classification problems
For each class pair (k,l):

- positive examples: x_i such that $y_i = k$
- negative examples: x_i such that $y_i = l$.

Inference: Majority.





One vs all

- Assumption: each class individually separable from the others
- No theoretical justification
- Easy to implement
- Unbalanced classes
- Works well in practice

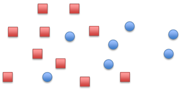
One vs one

- If class sizes are small, possible overfitting
- Need large memory to store models

These methods are applicable to any binary classifier.

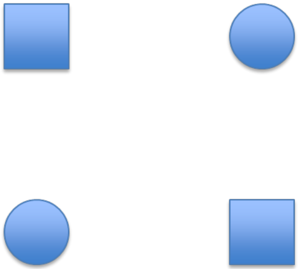
```
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsOneClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
X, y = ...
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, shuffle=True, random_state=0)
ovo = OneVsOneClassifier(LinearSVC(random_state=0)).fit(X_train, y_train)
ovo.predict(X_test)

ova = OneVsRestClassifier(LinearSVC(random_state=0)).fit(X_train, y_train)
ova.predict(X_test)
```

SVM: non linearly separable case



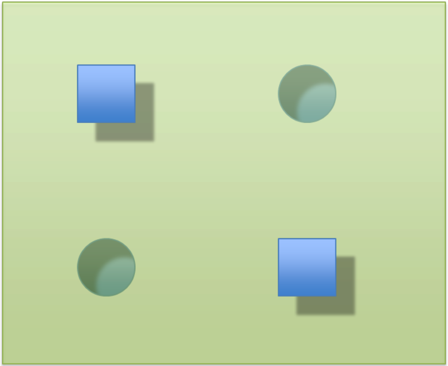
→ So far, support vector machines can only handle linearly separable data
→ But most data isn't.
→ We already see how to deal with this problem: soft margin
→ Now: another solution...

SVM: NON LINEARLY SEPARABLE CASE



→ Points that are not linearly separable in 2 dimension ..

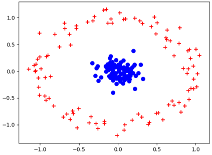
SVM: non linearly separable case



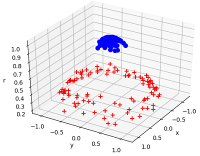
→ Points that are not linearly separable in 2 dimension, might be linearly separable in 3.

SVM: NON LINEARLY SEPARABLE CASE

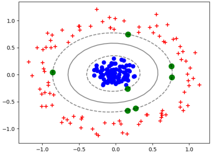
Another example



(a) Data.



(b) 3D Mapping (RBF).



(c) Classification.

Basis of Kernel Methods

Recall...

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \quad w = \sum_{i=1}^{N} \alpha_i y_i x_i$$

And then...

- → The decision process doesn't depend on the dimensionality of the data.
- → We can map to a higher dimensionality of the data space.
- → data points only appear within a dot product.
- → The error is based on the dot product of data points, not the data points themselves.

Basis of Kernel Methods

Recall...

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \; w = \sum_{i=1}^{N} \alpha_i y_i x_i$$

And then...

→ The decision process doesn't depend on the dimensionality of the data.
→ We can map to a higher dimensionality of the data space.
→ data points only appear within a dot product.
→ The error is based on the dot product of data points, not the data points themselves.

Basis of Kernel Methods

And so...

How to add dimensionality to the data in order to make it linearly separable ?

- Extreme case: construct a dimension for each data point ⇒ overfitting
- Mapping: $x_i^T x_j \leftrightarrow \phi(x_i)^T \phi(x_j)$

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j)$$

Why Dual formulation ?

Untractable example

$$\phi(x_0, x_1) = (x_0^2, x_0 x_1, x_1 x_0, x_1^2)$$

applied to a 20x30 image of 600 pixels $\approx$ 180000 dimensions ! Would be computationally infeasible to work in this space

Dual problem

- α_i: dual variables
- Since any component orthogonal to the space spanned by the training data has no effect, general result that weight vectors have dual representation: the representer theorem.
- can reformulate algorithms to learn dual variables rather than weight vector directly

Why Dual formulation ?

Untractable example

$$\phi(x_0, x_1) = (x_0^2, x_0x_1, x_1x_0, x_1^2)$$

applied to a 20x30 image of 600 pixels $\approx$ 180000 dimensions ! Would be computationally infeasible to work in this space

Dual problem

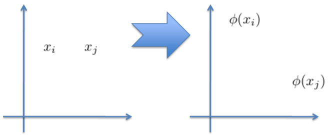
- α_i: dual variables
- Since any component orthogonal to the space spanned by the training data has no effect, general result that weight vectors have dual representation: the representer theorem.
- can reformulate algorithms to learn dual variables rather than weight vector directly

Kernel

Kernels

1. We can represent this dot product as a Kernel (Kernel Function, Kernel Matrix)
2. Finite (if large) dimensionality of $K(x_i, x_j)$ unrelated to dimensionality of x

Remember the dual



Kernels are a mapping

$$x_i^T x_j \leftrightarrow \phi(x_i)^T \phi(x_j)$$

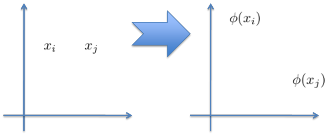
$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

Kernel

Kernels

1. We can represent this dot product as a Kernel (Kernel Function, Kernel Matrix)
2. Finite (if large) dimensionality of $K(x_i, x_j)$ unrelated to dimensionality of x

Remember the dual



Kernels are a mapping

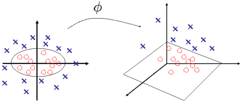
$$x_i^T x_j \leftrightarrow \phi(x_i)^T \phi(x_j)$$

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

Kernel

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

First example



with

$$\phi(y) = (y_0^2, \sqrt{2} y_0 y_1, y_1^2)$$

$$\begin{aligned} K(x,z) &= (x^T z)^2 \\ &= (x_0 z_0 + x_1 z_1)^2 \\ &= x_0^2 z_0^2 + 2 x_0 z_0 x_1 z_1 + x_1^2 z_1^2 \\ &= (x_0^2, \sqrt{2} x_0 x_1, x_1^2)^T (z_0^2, \sqrt{2} z_0 z_1, z_1^2) \\ &= \phi(x)^T \phi(z) \end{aligned}$$

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

Second example

$\phi : x \in X \rightarrow \phi(x) \in \mathcal{F}$
$(x, y) \mapsto (x_0^2, x_0 x_1, x_1 x_0, x_1^2)$
Linear equation in $\mathcal{F}$ $ax_0^2 + bx_1^2 = c \rightarrow$ ellipse (non linear shape) in X

Capacity of feature spaces

The capacity is proportional to the dimension

Theorem

Given $m+1$ examples in general position in a m-dimensional space, every possible classification can be generated with a thresholded linear function

Extension: Cover's theorem

- Capacity may easily become too large and lead to over-fitting: being able to realise every classifier means unlikely to generalise well
- Computational costs involved in dealing with large vectors

Kernel

Kernels

- In general: don't need to know the form of ϕ.
- Just specifying the kernel function is sufficient.
- A good kernel: Computing K_{ij} is cheaper than $\phi(x_i)$

Valid Kernels

- Symmetric
- Must be decomposable into ϕ functions
- Harder to show.
  - Gram matrix is positive semi-definite
  - Positive entries are definitely positive semi-definite.
  - Negative entries may still be positive semi-definite

$$x^T K x \geq 0$$

Kernel

Kernels

- In general: don't need to know the form of ϕ.
- Just specifying the kernel function is sufficient.
- A good kernel: Computing K_{ij} is cheaper than $\phi(x_i)$

Valid Kernels

- Symmetric
- Must be decomposable into ϕ functions
- Harder to show.
  - Gram matrix is positive semi-definite
  - Positive entries are definitely positive semi-definite.
  - Negative entries may still be positive semi-definite

$$x^T K x \geq 0$$

Kernel

Examples

K, K' Kernel $\Rightarrow \quad cK, K+K', K.K', exp(K)...$
Examples: Polynomial kernels, RBF, String kernels, graph kernels
Note: a SVM model using a sigmoid kernel function is equivalent to a two-layer, perceptron neural network.

Incorporating Kernels in SVMs

$$
\begin{aligned}
W(\alpha) &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) \\
&= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j)
\end{aligned}
$$

- optimize the α_i and b w.r.t. K
- decision function $f(x) = sign\left[\sum_{i=1}^{N} \alpha_i y_i K(x, x_i) + b\right]$

Examples

Polynomial Kernels

$K(x,z) = (x^T z + \theta)^d \quad \theta \geq 0$

- dot product: polynomial power of the original dot product.
- c large $\Rightarrow$ focus on linear terms
- c small $\Rightarrow$ focus on higher order terms
- Very fast to calculate

RBF

$K(x,z) = e^{\frac{\|x-z\|^2}{2\sigma^2}}$

- dot product: related to the distance in space between the two points.
- Placing a bump on each point

```
from sklearn.svm import SVC
X, y = ...
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, shuffle=True, random_state=0)
rbf=SVC(kernel="rbf", gamma=0.1, C=100).fit(X_train, y_train)
poly = SVC(kernel="poly", degree=4, coef0=0.5, C=5).fit(X_train, y_train)
```

Examples

Polynomial Kernels

$K(x,z) = (x^T z + \theta)^d \quad \theta \geq 0$

- dot product: polynomial power of the original dot product.
- c large $\Rightarrow$ focus on linear terms
- c small $\Rightarrow$ focus on higher order terms
- Very fast to calculate

RBF

$K(x,z) = e^{\frac{\|x-z\|^2}{2\sigma^2}}$

- dot product: related to the distance in space between the two points.
- Placing a bump on each point

```
from sklearn.svm import SVC
X, y = ...
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, shuffle=True, random_state=0)
rbf=SVC(kernel="rbf", gamma=0.1, C=100).fit(X_train, y_train)
poly = SVC(kernel="poly", degree=4, coef0=0.5, C=5).fit(X_train, y_train)
```

Examples

String kernels

Not a gaussian, but still a legitimate Kernel

- $K(s, s')$ = difference in length,count of different letters, minimum edit distance
- allow for infinite dimensional inputs
- don't need to manually encode the input

Examples

Graph Kernels

- Define the kernel function based on graph properties
- must be computable in poly-time (paths, spanning trees, cycles, bag of paths...)
- Possible incorporation of knowledge about the input without direct feature extraction

To conclude: kernel Trick

To conclude (Kernel trick) : a kernel can be applied where a dot product is used in an optimization:

- Kernel PCA
- Kernel perceptron
- unsupervised clustering (similarity $\approx$ distance $\leftrightarrow$ dot product)