Applied Machine Learning

Kernel Methods

Computer Science, Fall 2022

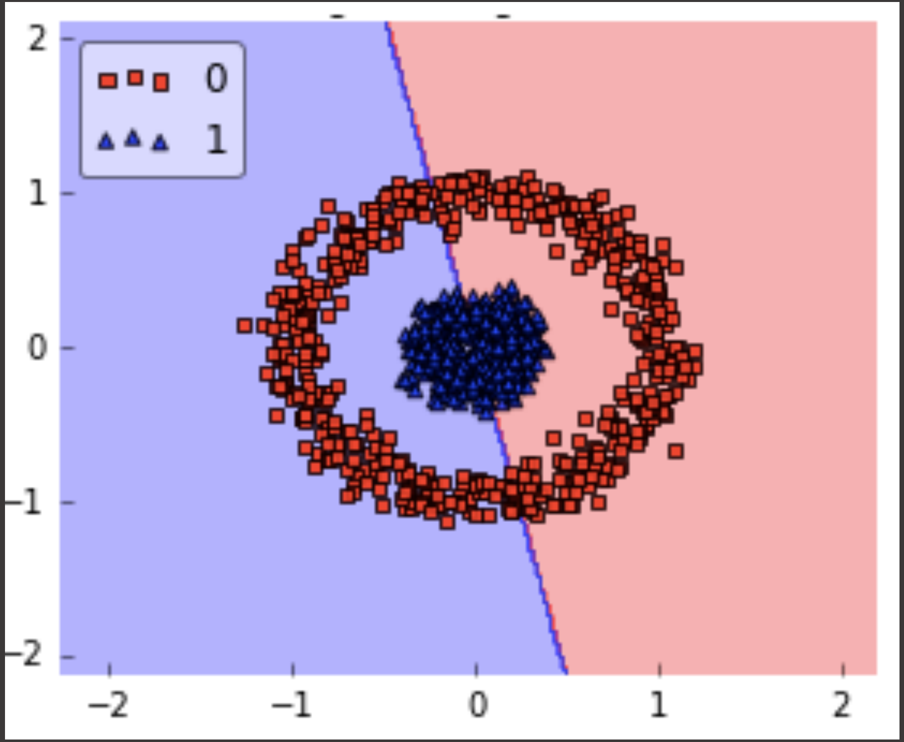
Instructor: Xuhong Zhang

Kernel Method

- Not matter hard SVM or soft SVM, we assume linear separation (w/o compromise).

- **Question**: What if the surface is **Non-Linear**?

Kernel Method



1, Can we use a linear classifier to classify the data?

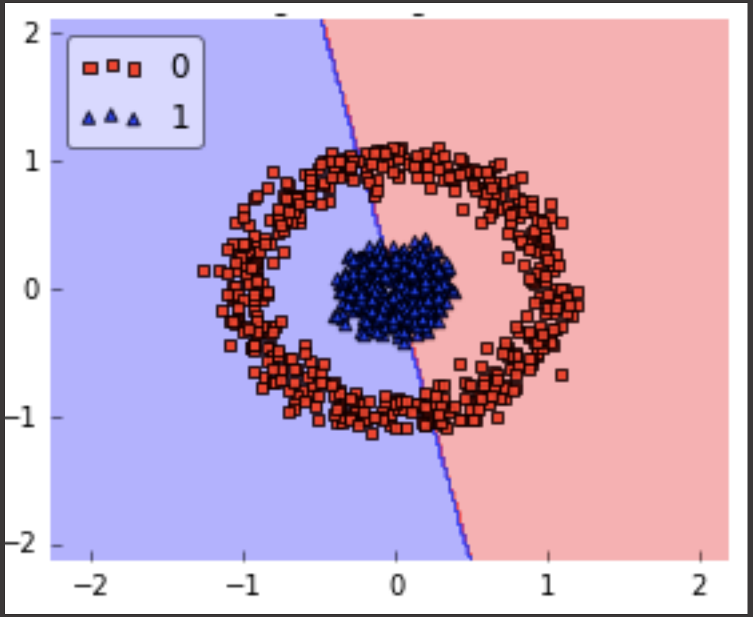
2, Can I fine-tune a linear classifier to classify the data?

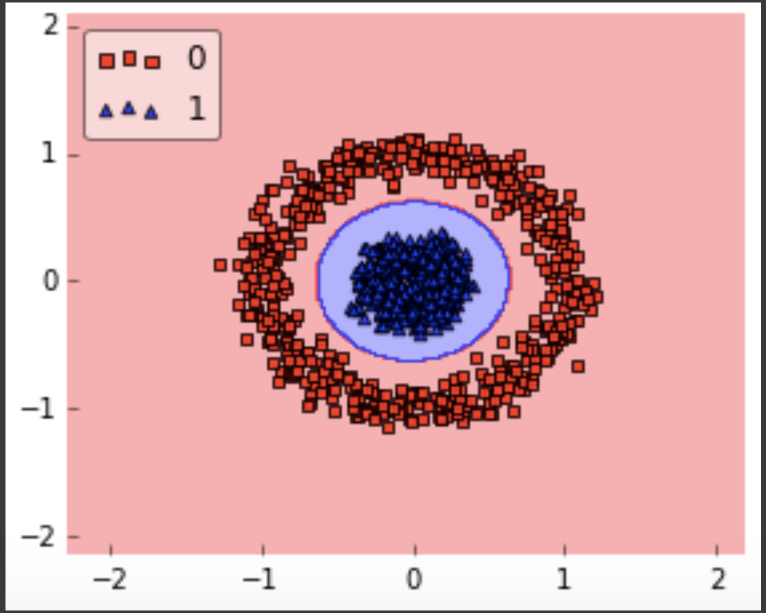
3, What if I just want to use a linear classifier?

Kernel Method

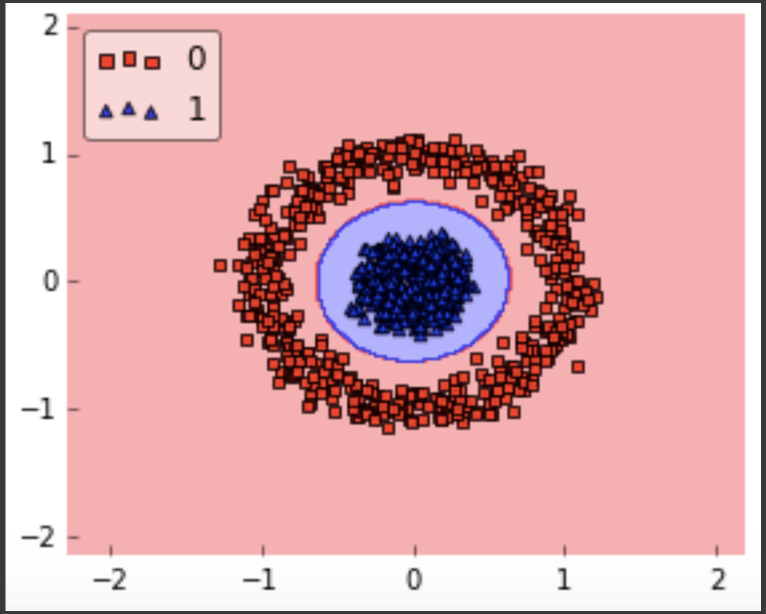
- The motivation for kernel method:

Making the Non-Linear Linear

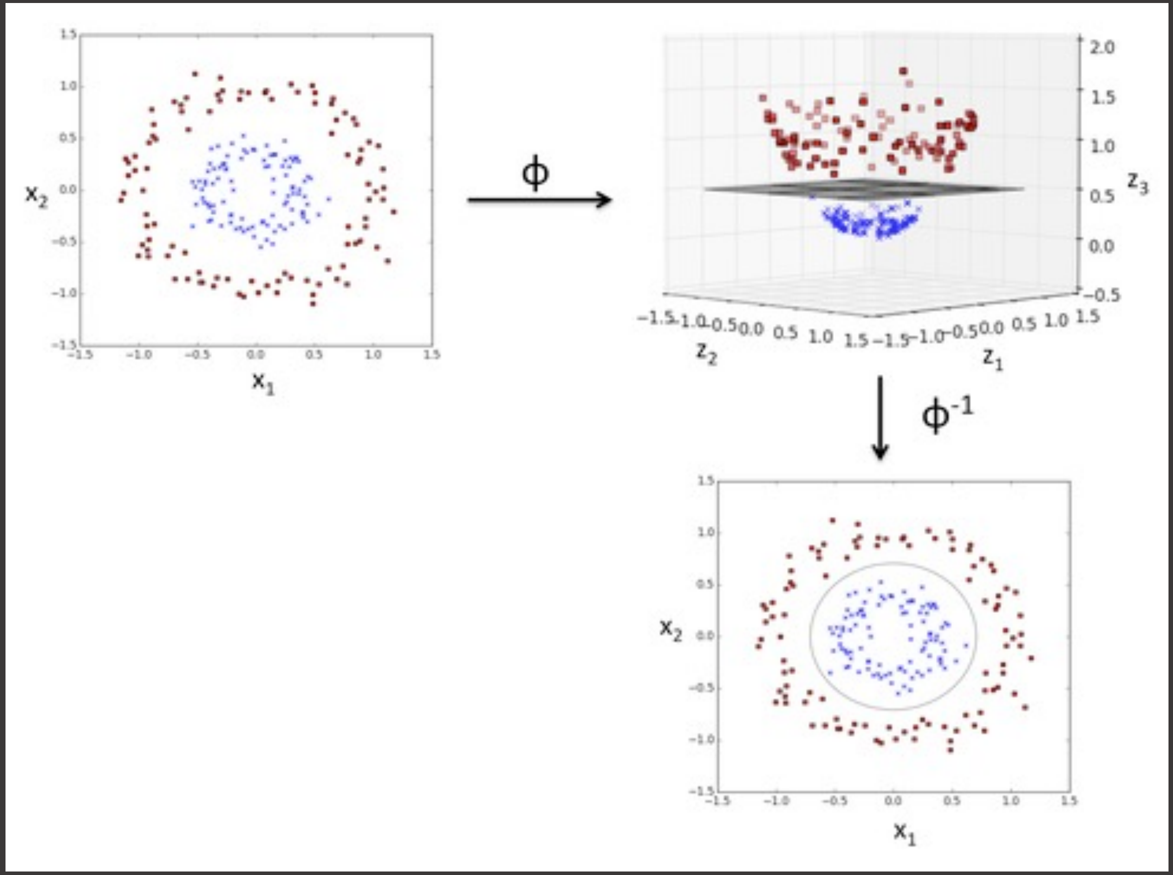




Kernel Method



The RBF Kernel



Question: In order to get this decision boundary, what new features I can create to make this data separatable by this circular decision boundary?

Kernel Method

https://www.youtube.com/watch?v=3liCbRZPrZA

Kernel Method

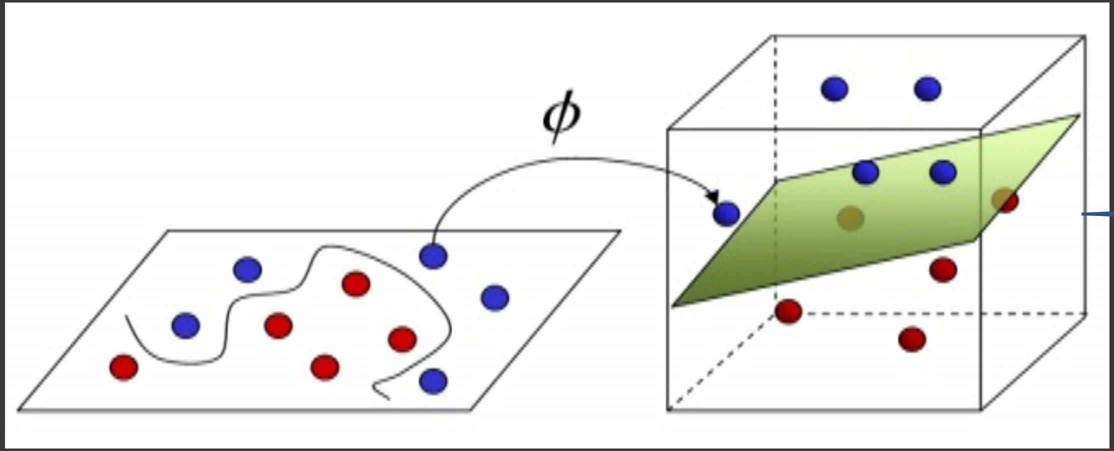
- Motivation: we want to add/create new features so that the data is separatable by trivial linear classifier.
- Kernel method:

Making the Non-Linear Linear

Kernel Method

- **The key** : Mapping into a new Feature Space

$$\Phi: \chi \mapsto \hat{\chi} = \Phi(x)$$



Input Space   Feature Space

Recall what we talked about high dimensional space

- For example, with $x_i \in \mathbb{R}^2$

We can construct

$$\Phi([x_{i1}, x_{i2}]) = [x_{i1}, x_{i2}, x_{i1}x_{i2}, x_{i1}^2, x_i^2]$$

Image from http://web.engr.oregonstate.edu/ ~afern/classes/cs534/

Handcrafted Feature Expansion

- By applying basis function (feature transformations) on the input feature vectors, we can make linear classifiers non-linear.
- For a data vector $x \in \mathbb{R}^d$, we apply the transformation $x \rightarrow \phi(x)$ where $\phi(x) \in \mathbb{R}^D$ (usually $D \gg d$)
- New question: not every problem is like the one we just saw and we don't know what features we need to add/create….

Handcrafted Feature Expansion

- Solution: I just add all the features we can think of...
- Example:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \qquad \phi(x) = \begin{pmatrix} x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \dots x_d \end{pmatrix}$$

$\phi(x)$ is very expressive and allows for complicated non-linear decision boundaries, but the dimensionality is extremely high.

Question: what is the dimension of $\phi(x)$

Kernel Method

- In practice, we will apply two tricks to apply kernel methods.
  - The way to add all the interaction terms will cause memory issue and also prohibitively expensive for computation.
  - First trick: We have a linear classifier, and it turns out to be, we can express the linear classifier with the expression of inner product, e.g., $K_{ij} = x_i^T x_j$. We will take the square loss as an example.
  - If we only access our data with the inner product, we can pre-compute and store them.

Gradient Descent for Squared Loss

- For a linear model with the format of $w^T x_j$, claim that the w can be expressed as a linear combination of $x, \sum_{i=1}^{n} \alpha_i x_i$.

- If so, then the format $w^T x_j = \sum_{i=1}^{n} \alpha_i \underbrace{\boxed{x_i^T x_j}}_{K_{ij}}$

- If everything can be expressed in the inner-product, then the inner-product can be pre-computed and stored. Save computational time. No matter how high dimension the data is, it is tolerable.

Gradient Descent for Squared Loss

Some observation:

the gradient is a linear combination of input samples.

- Look at the squared loss:

$$l(w) = \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

- The gradient descent rule, with step-size/learning-rate $s > 0$, w is updated as:

γ_i: function of x_i, y_i
And it is a scaler.

$$w_{t+1} \leftarrow w_t - s\left(\frac{\partial l}{\partial w}\right), \text{where } \frac{\partial l}{\partial w} = \sum_{i=1}^{n} \boxed{2(w^T x_i - y_i)} x_i = \sum_{i=1}^{n} \gamma_i x_i$$

Proof

- Claim: w is a linear combination of all input vectors.

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

- Proof :
  1. Since the loss is convex, the final solution is independent of the initialization, and we can initialize w^0 to be whatever we want. So we pick $w^0 = [0, \dots, 0]^T$ and here we have $\alpha_1 = \cdots = \alpha_n = 0$.

Proof

- Proof :
  2. We re-write the gradient updates entirely in terms of updating the α_i coefficients

$$w_1 = w_0 - s\sum_{i=1}^{n} 2(w_0^T x_i - y_i)x_i = \sum_{i=1}^{n} \alpha_i^0 x_i - s\sum_{i=1}^{n} \gamma_i^0\, x_i = \sum_{i=1}^{n} \alpha_i^1 x_i \quad \text{(with } \alpha_i^1 = \alpha_i^0 - s\gamma_i^0 \text{)}$$

$$w_2 = w_1 - s\sum_{i=1}^{n} 2(w_1^T x_i - y_i)x_i = \sum_{i=1}^{n} \alpha_i^1 x_i - s\sum_{i=1}^{n} \gamma_i^1\, x_i = \sum_{i=1}^{n} \alpha_i^2 x_i \quad \text{(with } \alpha_i^2 = \alpha_i^1 - s\gamma_i^1 \text{)}$$

$$w_3 = w_2 - s\sum_{i=1}^{n} 2(w_2^T x_i - y_i)x_i = \sum_{i=1}^{n} \alpha_i^2 x_i - s\sum_{i=1}^{n} \gamma_i^2\, x_i = \sum_{i=1}^{n} \alpha_i^3 x_i \quad \text{(with } \alpha_i^3 = \alpha_i^2 - s\gamma_i^2 \text{)}$$

……

Induction proof

$$w_t = w_{t-1} - s\sum_{i=1}^{n} 2(w_{t-1}^T x_i - y_i)x_i = \boxed{\sum_{i=1}^{n} \alpha_i^{t-1} x_i - s\sum_{i=1}^{n} \gamma_i^{t-1}\, x_i} = \sum_{i=1}^{n} \alpha_i^{t} x_i \quad \text{(with } \alpha_i^t = \alpha_i^{t-1} - s\gamma_i^{t-1} \text{)}$$

Proof

- Put everything together:
  1. w is trivially a linear combination of the training vectors for w_0 (base case).
  2. If we apply the inductive hypothesis for w_t it follows for w_{t+1}.
  3. The update-rule for α_i^t is thus

$$\alpha_i^t = \alpha_i^{t-1} - s\gamma_i^{t-1}, \text{ and we have } \alpha_i^t = -s\sum_{r=0}^{t-1}\gamma_i^r$$

We only need to keep track of the n coefficients $\alpha_1, \dots, \alpha_n$, and they are independent of the dimensionality of the data ; only scale with n ; no need to compute w!

Good news: we can perform the entire gradient descent update rule without ever expressing w explicitly.

Loss function

In addition, we can also express the inner-product of w with any input x_i purely in terms of inner-products between training inputs.

$$w^T x_j = \sum_{i=1}^{n} \alpha_i x_i^T x_j$$

Thus, if we come back to the squared loss function, $l(w)$, we can rewrite it in terms of inner-product between training inputs:

$$l(w) = \sum_{i=1}^{n} (w^T x_i - y_i)^2 \quad \Longleftrightarrow \quad l(\alpha) = \sum_{i=1}^{n} (\sum_{j=1}^{n} \alpha_j x_j^T x_i - y_i)^2$$

Benefit

During the test-time we also only need these coefficients to make a prediction on a test-point x_t, and can write the entire classifier in terms of inner-products between the test point and training points:

$$h(x_t) = w^T x_t = \sum_{i=1}^{n} \alpha_i x_i^T x_t$$

The only information we ever need in order to learn a hyper-plane classifier with the squared-loss is inner-products between all pairs of data vectors.

Inner-Product

- Now we get the idea that we can express our linear model using inner products. Remember this is the first trick.

- Remember that we want to add/create new features for the model. With the inner-product, $x_i^T x_j$, we have $\phi(x_i)^T \phi(x_j)$, and we just adopt

$$\phi(x) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \ldots x_d \end{pmatrix}$$

Inner-Product

Now the question is, we need to compute the $\phi(x)^T\phi(z)$:

Kernel Function

$$\phi(x) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1x_2 \\ \vdots \\ x_{d-1}x_d \\ \vdots \\ x_1x_2\dots x_d \end{pmatrix}$$

$$\phi(x_i)^T\phi(x_j) = \boxed{k(x_i, x_j)}$$

The inner product $\emptyset(x)^T\emptyset(z)$ can be formulated as:

$$\phi(x)^T\phi(z) = \underbrace{1\cdot 1 + x_1z_1 + x_2z_2 + \cdots + x_1x_2z_1z_2 + \cdots + x_1 \ldots x_d z_1 \ldots z_d}_{\text{The sum of } 2^{2d} \text{ terms}} = \underbrace{\prod_{k=1}^{d}(1 + x_k z_k)}_{\text{The product of } d \text{ terms}}$$

Inner-Product

- So with a finite training set of n samples, inner products are often pre-compute
and stored in a Kernel Matrix

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

- If we store the matrix K, we only need to do simple inner-product look-ups
- Now instead of compute for w (which has the same dimensions with $\phi(x_i)$)
we only need to compute α low-dimensional computations throughout the gradient descent algorithm. For prediction on the test data x_t

$$h(x_t) = \sum_{j=1}^{n} \alpha_j k(x_j, x_t)$$

Summary

- For kernel methods:
  - Re-express the model with inner-product
  - Replace inner-product terms with a kernel function
  - Compute the kernel term values and store them (basically, this learns interaction patterns among the training data)
  - Derive the new coefficients (e.g. α not w)
  - When testing, the prediction is based on the interaction of the training data and the new test data

We capture the interactions between data points in a very high dimensional space, and we never explicitly compute an instance in that space.