CS 383 – Machine Learning

Kernels

Slides adapted from material created by E. Alpaydin
Prof. Mordohai, Prof. Greenstadt, Pattern Classification (2nd Ed.),
Pattern Recognition and Machine Learning

Objectives

- Kernels

Kernels

- For many algorithms, the only time we actually use the samples themselves is when we compute some distance or similarity between two samples
- A function that takes two samples and returns a similarity is called a *kernel function*, $\kappa(X_i, X_j)$
- There are many kernel functions out there, one of the most popular is the *cosine similarity kernel:*

$$\kappa(X_i, X_j) = X_i X_j^T$$

Kernels

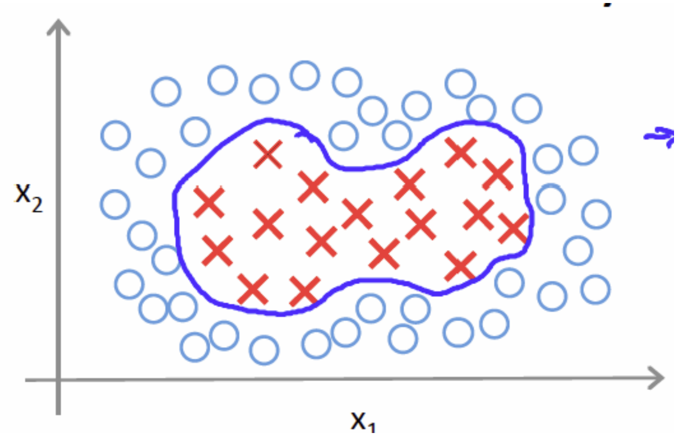
- Here are some other common kernels:
  - Linear/Cosine: $\kappa(X_i, X_j) = X_i X_j^T$
  - Polynomial kernel: $\kappa(X_i, X_j) = \left(X_i X_j^T + 1\right)^p$
  - Gaussian Radial Basis kernel (RBF):

$$\kappa(X_i, X_j) = e^{-\frac{1}{2\sigma^2}\left|\left|X_i - X_j\right|\right|^2}$$

- If feature vectors are histogram
  - Histogram intersection: $\kappa(X_i, X_j) = \sum_{k=1}^{N} \min(X_{i,k}, X_{j,k})$
  - Hellinger kernel: $\kappa(X_i, X_j) = \sum_{k=1}^{N} \sqrt{X_{i,k} X_{j,k}}$

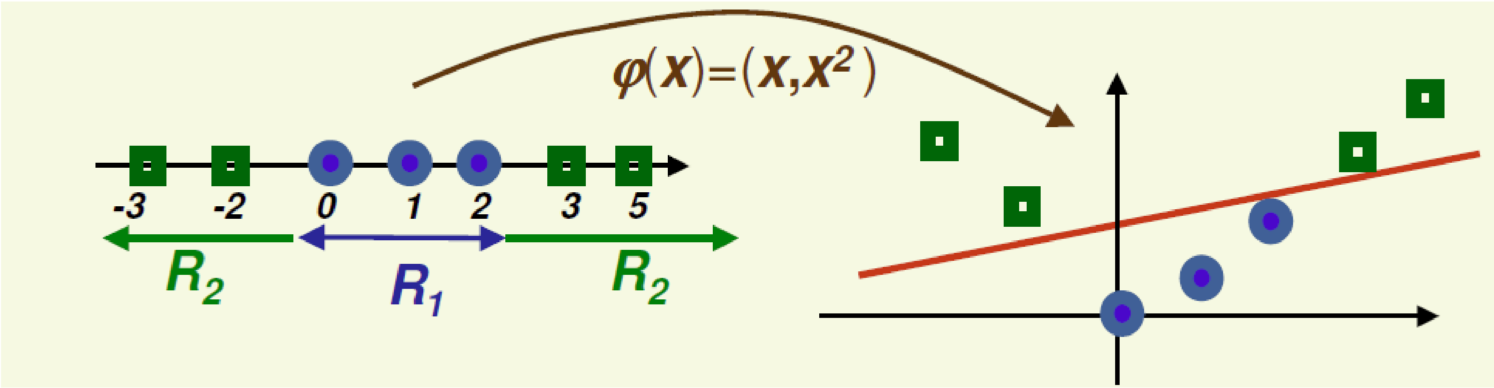
Kernel Trick

- Kernels can also provide an additional benefit.
- Sometimes we may want to go to a *higher* feature space.
- Why?
  - Because we have a linear classifier and the data is not directly linearly separable.



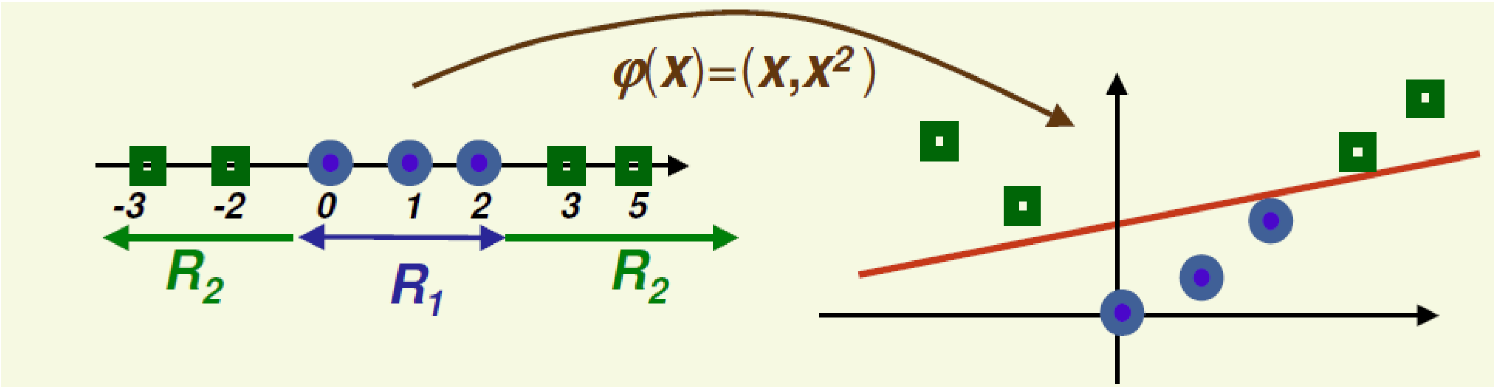
Kernel Trick

- One solution would be to map our current space to another separable space
- Then project data x to **higher** dimension using mapping function $\phi(x)$



Non-Linear Mapping

- Of course what's the issue with lifting data to higher dimensional space?
  - Overfitting
  - Computational power
- Maybe we could then project the data back down.. (using PCA or something else)



Kernel Trick

- The "kernel trick" provides a way to avoid performing operations in high dimensional space explicitly (as if we had more/new features)
  - But the computation done in lower space produces the same value as if done in higher space
- We want a way to express $\kappa(X_i, X_j)$ as a cosine similarity in higher space:

$$\kappa(X_i, X_j) = \phi(X_i)\phi(X_j)^T$$

Kernel Trick

- Let's look at the polynomial kernel of degree two, $\kappa(X_i, X_j) = (X_i X_j^T + 1)^2$ as applied to observations with one feature:

$$x = [x_1]$$

- Therefore $\kappa(X_i, X_j) = (X_i X_j^T + 1)^2 = (X_{i,1} X_{j,1} + 1)^2$

$$= (X_{i,1} X_{j,1})^2 + 2(X_{i,1} X_{j,1}) + 1$$
$$= X_{i,1}^2 X_{j,1}^2 + 2X_{i,1} X_{j,1} + 1$$

- Can we write this as the product of two things, each just dependent on its own observation?
  - I.e $\kappa(X_i, X_j) = \phi(X_i)\phi(X_j)^T$?

Kernel Trick

- $\kappa(X_i, X_j) = [X_{i,1}^2 , \sqrt{2}X_{i,1}, 1][X_{j,1}^2, \sqrt{2}X_{j,1}, 1]$
- Therefore

$$\phi(x) = [x_1^2, \sqrt{2}x_1, 1]$$

- Using the polynomial kernel of degree two on observations with a single feature is equivalent to compute the cosine similarity on observations in 3D space.

Choice of Kernel

- Constraints on kernels
  - $\kappa(X_i, X_j)$should correspond to $\phi(X_i)\phi(X_j)^T$ in a higher dimensional space
  - If $\kappa$and $\kappa'$are kernels then $a\kappa + b\kappa'$ is also a kernel (linear combination)
- Intuitively the kernel should measure the similarity between X_i and X_j
  - After all, the dot (inner) product measures similarity of unit vectors
  - Therefore the kernel choice may be application, problem, and or data specific

Choice of Kernel

- So which should we use?
- Again depends on the feature themselves
  - Type
  - Value
  - Quantity
- If we have tons of features then we probably don't even need to go to higher dimensional space
  - So linear is suffice
  - Consider "ton" to be more features than observations

Kernel Usage

- We’ve already seen one place where we could use kernels
  - K-Means
- In the next few weeks we’ll look at a few more that are “kernelizable”
  - K-Nearest Neighbors (KNNs)
  - Support Vector Machines (SVMs)
  - Logistic Regression