Speeding Up Kernel Methods, and Intro to Unsupervised Learning

Piyush Rai

Introduction to Machine Learning (CS771A)

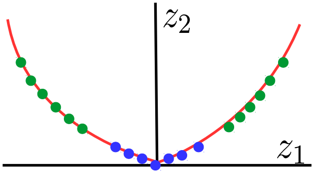
September 11, 2018

Recap: Nonlinear Mappings

- Idea: Use a nonlinear mapping $\phi : \mathbb{R}^D \to \mathbb{R}^M$ to map original data to a high-dim space, e.g.,

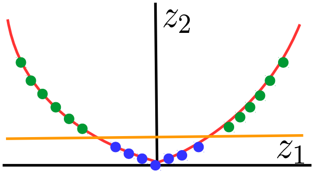
Recap: Nonlinear Mappings

- Idea: Use a nonlinear mapping $\phi : \mathbb{R}^D \to \mathbb{R}^M$ to map original data to a high-dim space, e.g.,



Recap: Nonlinear Mappings

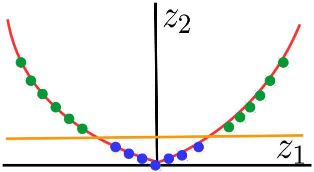
- Idea: Use a nonlinear mapping $\phi : \mathbb{R}^D \to \mathbb{R}^M$ to map original data to a high-dim space, e.g.,



- Learn a linear model in the new space using the mapped inputs $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_N)$

Recap: Nonlinear Mappings

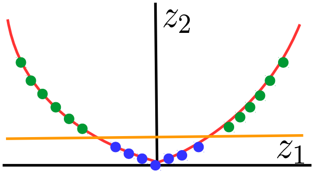
- Idea: Use a nonlinear mapping $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$ to map original data to a high-dim space, e.g.,



- Learn a linear model in the new space using the mapped inputs $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_N)$
- Equivalent to learning a nonlinear model on the original data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$

Recap: Nonlinear Mappings

- Idea: Use a nonlinear mapping $\phi : \mathbb{R}^D \to \mathbb{R}^M$ to map original data to a high-dim space, e.g.,

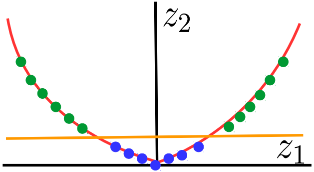


- Learn a linear model in the new space using the mapped inputs $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_N)$
- Equivalent to learning a nonlinear model on the original data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- The mappings can be explicitly defined, or implicitly defined via a kernel function k, s.t.

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$$

Recap: Nonlinear Mappings

- Idea: Use a nonlinear mapping $\phi : \mathbb{R}^D \to \mathbb{R}^M$ to map original data to a high-dim space, e.g.,



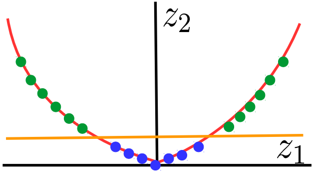
- Learn a linear model in the new space using the mapped inputs $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_N)$
- Equivalent to learning a nonlinear model on the original data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- The mappings can be explicitly defined, or implicitly defined via a kernel function k, s.t.

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$$

- Benefit of using kernels: Don't need to explicitly compute the mappings (M can be very large)

Recap: Nonlinear Mappings

- Idea: Use a <u>nonlinear</u> mapping $\phi : \mathbb{R}^D \to \mathbb{R}^M$ to map original data to a high-dim space, e.g.,



- Learn a linear model in the new space using the mapped inputs $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_N)$
- Equivalent to learning a nonlinear model on the original data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- The mappings can be explicitly defined, or implicitly defined via a kernel function k, s.t.

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$$

- Benefit of using kernels: Don't need to explicitly compute the mappings (M can be very large)
- Many ML algos only have data appearing as inner products. Can kernelize such algos

Recap: Nonlinear Mappings and Kernels

- A kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$ defines inner-product similarity between two inputs
- This is a Euclidean similarity in ϕ space but a "nonlinear" similarity in original space

Recap: Nonlinear Mappings and Kernels

- A kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$ defines inner-product similarity between two inputs
- This is a Euclidean similarity in ϕ space but a "nonlinear" similarity in original space
- Some popular examples of kernel functions

$$
\begin{aligned}
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= \boldsymbol{x}_n^\top \boldsymbol{x}_m \quad \text{(Linear kernel)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= (1 + \boldsymbol{x}_n^\top \boldsymbol{x}_m)^2 \quad \text{(Quadratic kernel)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= (1 + \boldsymbol{x}_n^\top \boldsymbol{x}_m)^d \quad \text{(Polynomial kernel of degree } d\text{)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= \exp[-\gamma ||\boldsymbol{x}_n - \boldsymbol{x}_m||^2] \quad \text{(RBF/Gaussian kernel)}
\end{aligned}
$$

Recap: Nonlinear Mappings and Kernels

- A kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$ defines inner-product similarity between two inputs
- This is a Euclidean similarity in ϕ space but a "nonlinear" similarity in original space
- Some popular examples of kernel functions

$$
\begin{aligned}
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= \boldsymbol{x}_n^\top \boldsymbol{x}_m \quad \text{(Linear kernel)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= (1 + \boldsymbol{x}_n^\top \boldsymbol{x}_m)^2 \quad \text{(Quadratic kernel)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= (1 + \boldsymbol{x}_n^\top \boldsymbol{x}_m)^d \quad \text{(Polynomial kernel of degree } d\text{)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= \exp[-\gamma ||\boldsymbol{x}_n - \boldsymbol{x}_m||^2] \quad \text{(RBF/Gaussian kernel)}
\end{aligned}
$$

- Each of these kernels have an associated feature mapping $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$

Recap: Nonlinear Mappings and Kernels

- A kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$ defines inner-product similarity between two inputs
- This is a Euclidean similarity in ϕ space but a "nonlinear" similarity in original space
- Some popular examples of kernel functions

$$
\begin{aligned}
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= \boldsymbol{x}_n^\top \boldsymbol{x}_m \quad \text{(Linear kernel)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= (1 + \boldsymbol{x}_n^\top \boldsymbol{x}_m)^2 \quad \text{(Quadratic kernel)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= (1 + \boldsymbol{x}_n^\top \boldsymbol{x}_m)^d \quad \text{(Polynomial kernel of degree } d\text{)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= \exp[-\gamma ||\boldsymbol{x}_n - \boldsymbol{x}_m||^2] \quad \text{(RBF/Gaussian kernel)}
\end{aligned}
$$

- Each of these kernels have an associated feature mapping $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$, e.g.,
  - Quadratic kernel: $\phi(\boldsymbol{x}) = [1, \sqrt{2}x_1, \ldots, \sqrt{2}x_1 x_2, \ldots, x_1^2, \ldots]$

Recap: Nonlinear Mappings and Kernels

- A kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$ defines inner-product similarity between two inputs
- This is a Euclidean similarity in ϕ space but a "nonlinear" similarity in original space
- Some popular examples of kernel functions

$$
\begin{aligned}
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= \boldsymbol{x}_n^\top \boldsymbol{x}_m \quad \text{(Linear kernel)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= (1 + \boldsymbol{x}_n^\top \boldsymbol{x}_m)^2 \quad \text{(Quadratic kernel)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= (1 + \boldsymbol{x}_n^\top \boldsymbol{x}_m)^d \quad \text{(Polynomial kernel of degree } d\text{)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= \exp[-\gamma ||\boldsymbol{x}_n - \boldsymbol{x}_m||^2] \quad \text{(RBF/Gaussian kernel)}
\end{aligned}
$$

- Each of these kernels have an associated feature mapping $\phi : \mathbb{R}^D \to \mathbb{R}^M$, e.g.,
  - Quadratic kernel: $\phi(\boldsymbol{x}) = [1, \sqrt{2}x_1, \ldots, \sqrt{2}x_1x_2, \ldots, x_1^2, \ldots]$. Here $M = O(D + D^2)$ dimensional

Recap: Nonlinear Mappings and Kernels

- A kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$ defines inner-product similarity between two inputs
- This is a Euclidean similarity in ϕ space but a "nonlinear" similarity in original space
- Some popular examples of kernel functions

$$
\begin{aligned}
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= \boldsymbol{x}_n^\top \boldsymbol{x}_m \quad \text{(Linear kernel)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= (1 + \boldsymbol{x}_n^\top \boldsymbol{x}_m)^2 \quad \text{(Quadratic kernel)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= (1 + \boldsymbol{x}_n^\top \boldsymbol{x}_m)^d \quad \text{(Polynomial kernel of degree } d\text{)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= \exp[-\gamma ||\boldsymbol{x}_n - \boldsymbol{x}_m||^2] \quad \text{(RBF/Gaussian kernel)}
\end{aligned}
$$

- Each of these kernels have an associated feature mapping $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$, e.g.,
  - Quadratic kernel: $\phi(\boldsymbol{x}) = [1, \sqrt{2}x_1, \ldots, \sqrt{2}x_1 x_2, \ldots, x_1^2, \ldots]$. Here $M = O(D + D^2)$ dimensional
  - RBF kernel: $\phi(\boldsymbol{x})$ is infinite dimensional (saw in the last class). Here $M = \infty$

Recap: Nonlinear Mappings and Kernels

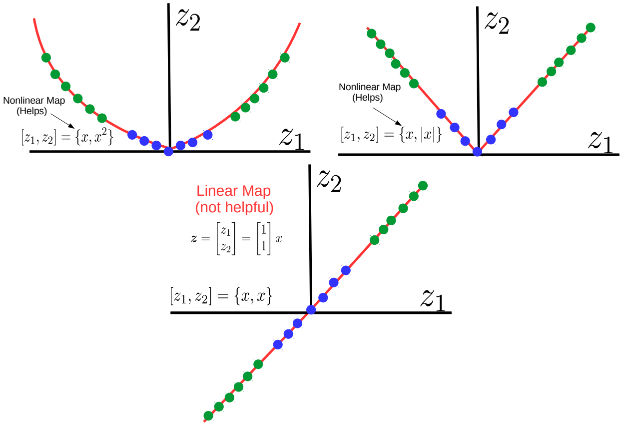
- A kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$ defines inner-product similarity between two inputs
- This is a Euclidean similarity in ϕ space but a "nonlinear" similarity in original space
- Some popular examples of kernel functions

$$
\begin{aligned}
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= \boldsymbol{x}_n^\top \boldsymbol{x}_m \quad \text{(Linear kernel)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= (1 + \boldsymbol{x}_n^\top \boldsymbol{x}_m)^2 \quad \text{(Quadratic kernel)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= (1 + \boldsymbol{x}_n^\top \boldsymbol{x}_m)^d \quad \text{(Polynomial kernel of degree } d\text{)} \\
k(\boldsymbol{x}_n, \boldsymbol{x}_m) &= \exp[-\gamma ||\boldsymbol{x}_n - \boldsymbol{x}_m||^2] \quad \text{(RBF/Gaussian kernel)}
\end{aligned}
$$

- Each of these kernels have an associated feature mapping $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$, e.g.,
  - Quadratic kernel: $\phi(\boldsymbol{x}) = [1, \sqrt{2}x_1, \ldots, \sqrt{2}x_1x_2, \ldots, x_1^2, \ldots]$. Here $M = O(D + D^2)$ dimensional
  - RBF kernel: $\phi(\boldsymbol{x})$ is infinite dimensional (saw in the last class). Here $M = \infty$
- Again, remember that when using kernels, we don't have to compute ϕ explicitly

Recap: Nonlinear Mappings and Kernels

- Not every high-dim mapping is helpful. The mapping ϕ must be nonlinear



Kernel Methods can be Slow

Training phase can be slow (if N is very large)

$$\text{Soft-Margin SVM:} \quad \max_{\boldsymbol{\alpha} \leq C} \; \mathcal{L}_D(\alpha) = \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top \underbrace{\mathbf{G}}_{\text{N x N size}}\boldsymbol{\alpha}$$

Kernel Methods can be Slow

Training phase can be slow (if N is very large)

Soft-Margin SVM: $$\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{G}\boldsymbol{\alpha}$$

N x N size

Storing the learned model may be expensive

$$\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \phi(\boldsymbol{x}_n)$$

Possibly very high-dimensional

Kernel Methods can be Slow

Training phase can be slow (if N is very large)

Soft-Margin SVM: $\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top \underbrace{\mathbf{G}}_{\text{N x N size}}\boldsymbol{\alpha}$

Storing the learned model may be expensive

$$\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \underbrace{\phi(\boldsymbol{x}_n)}_{\text{Possibly very high-dimensional}}$$

Testing (prediction) phase can be slow (scales in N or at least the number of support vectors)

$$y = \text{sign}(\boldsymbol{w}^\top \phi(\boldsymbol{x})) = \text{sign}(\sum_{n=1}^{N} \alpha_n y_n \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x})) = \text{sign}(\sum_{n=1}^{N} \alpha_n y_n k(\boldsymbol{x}_n, \boldsymbol{x}))$$

Kernel Methods can be Slow

Training phase can be slow (if N is very large)

Dual form of Ridge Regression: $\boldsymbol{w} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N)^{-1} \boldsymbol{y} = \mathbf{X}^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n$

Kernelized Ridge Regression: $\boldsymbol{w} = \phi(\mathbf{X})^\top (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \boldsymbol{y} = \phi(\mathbf{X})^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n \phi(\boldsymbol{x}_n)$

N x N size (also need to invert it)

Storing the learned model may be expensive

$$\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n \phi(\boldsymbol{x}_n)$$

Possibly very high-dimensional

Testing (prediction) phase can be slow (scales in N)

$$y = \boldsymbol{w}^\top \phi(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_n \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

Speeding Up Kernel Methods

- Kernel methods are slow at training and test time

Speeding Up Kernel Methods

- Kernel methods are slow at training and test time
- Would have been nice if we could easily compute the mapping $\phi(\boldsymbol{x})$ associated with kernel k

Speeding Up Kernel Methods

- Kernel methods are slow at training and test time
- Would have been nice if we could easily compute the mapping $\phi(\boldsymbol{x})$ associated with kernel k
- Then we could apply linear models directly on $\phi(\boldsymbol{x})$ without having to kernelize

Speeding Up Kernel Methods

- Kernel methods are slow at training and test time
- Would have been nice if we could easily compute the mapping $\phi(\boldsymbol{x})$ associated with kernel k
- Then we could apply linear models directly on $\phi(\boldsymbol{x})$ without having to kernelize
- But this is in general not possible since $\phi(\boldsymbol{x})$ is very high dimensional

Speeding Up Kernel Methods

- Kernel methods are slow at training and test time
- Would have been nice if we could easily compute the mapping $\phi(\boldsymbol{x})$ associated with kernel k
- Then we could apply linear models directly on $\phi(\boldsymbol{x})$ without having to kernelize
- But this is in general not possible since $\phi(\boldsymbol{x})$ is very high dimensional
- Instead of a high-dim $\phi(\boldsymbol{x})$, can we get a good set of low-dim features $\psi(\boldsymbol{x}) \in \mathbb{R}^L$ using the kernel?

Speeding Up Kernel Methods

- Kernel methods are slow at training and test time
- Would have been nice if we could easily compute the mapping $\phi(\boldsymbol{x})$ associated with kernel k
- Then we could apply linear models directly on $\phi(\boldsymbol{x})$ without having to kernelize
- But this is in general not possible since $\phi(\boldsymbol{x})$ is very high dimensional
- Instead of a high-dim $\phi(\boldsymbol{x})$, can we get a good set of low-dim features $\psi(\boldsymbol{x}) \in \mathbb{R}^L$ using the kernel?
- If $\psi(\boldsymbol{x})$ is a good approximation of $\phi(\boldsymbol{x})$, then we can just use $\psi(\boldsymbol{x})$ in a linear model

Speeding Up Kernel Methods

- Kernel methods are slow at training and test time
- Would have been nice if we could easily compute the mapping $\phi(\boldsymbol{x})$ associated with kernel k
- Then we could apply linear models directly on $\phi(\boldsymbol{x})$ without having to kernelize
- But this is in general not possible since $\phi(\boldsymbol{x})$ is very high dimensional
- Instead of a high-dim $\phi(\boldsymbol{x})$, can we get a good set of low-dim features $\psi(\boldsymbol{x}) \in \mathbb{R}^L$ using the kernel?
- If $\psi(\boldsymbol{x})$ is a good approximation of $\phi(\boldsymbol{x})$, then we can just use $\psi(\boldsymbol{x})$ in a linear model

$$\text{“Goodness” Criterion:} \quad \psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m) \approx \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$$

Speeding Up Kernel Methods

- Kernel methods are slow at training and test time
- Would have been nice if we could easily compute the mapping $\phi(\boldsymbol{x})$ associated with kernel k
- Then we could apply linear models directly on $\phi(\boldsymbol{x})$ without having to kernelize
- But this is in general not possible since $\phi(\boldsymbol{x})$ is very high dimensional
- Instead of a high-dim $\phi(\boldsymbol{x})$, can we get a good set of low-dim features $\psi(\boldsymbol{x}) \in \mathbb{R}^L$ using the kernel?
- If $\psi(\boldsymbol{x})$ is a good approximation of $\phi(\boldsymbol{x})$, then we can just use $\psi(\boldsymbol{x})$ in a linear model

$$\text{“Goodness” Criterion:} \quad \psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m) \approx \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$$

i.e., we want $\psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m) \approx k(\boldsymbol{x}_n, \boldsymbol{x}_m)$

Speeding Up Kernel Methods

- Kernel methods are slow at training and test time
- Would have been nice if we could easily compute the mapping $\phi(\boldsymbol{x})$ associated with kernel k
- Then we could apply linear models directly on $\phi(\boldsymbol{x})$ without having to kernelize
- But this is in general not possible since $\phi(\boldsymbol{x})$ is very high dimensional
- Instead of a high-dim $\phi(\boldsymbol{x})$, can we get a good set of low-dim features $\psi(\boldsymbol{x}) \in \mathbb{R}^L$ using the kernel?
- If $\psi(\boldsymbol{x})$ is a good approximation of $\phi(\boldsymbol{x})$, then we can just use $\psi(\boldsymbol{x})$ in a linear model

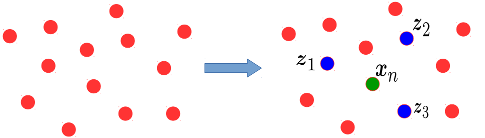
$$\text{"Goodness" Criterion:} \quad \psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m) \approx \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$$

  i.e., we want $\psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m) \approx k(\boldsymbol{x}_n, \boldsymbol{x}_m)$

- We will see two popular approaches: Landmarks and Random Features

Using Kernels to "Extract" Good Features: Landmarks

- Suppose we choose a small set of L "landmark" inputs $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L$ in the training data



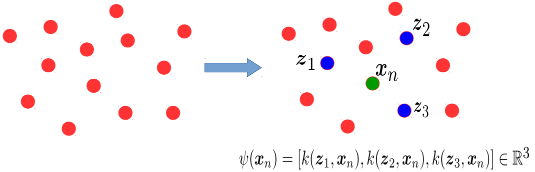
$\psi(\boldsymbol{x}_n) = [k(\boldsymbol{z}_1, \boldsymbol{x}_n), k(\boldsymbol{z}_2, \boldsymbol{x}_n), k(\boldsymbol{z}_3, \boldsymbol{x}_n)] \in \mathbb{R}^3$

- For each input $\boldsymbol{x}_n$, using a kernel k, define an L-dimensional feature vector as follows

$$\psi(\boldsymbol{x}_n) = [k(\boldsymbol{z}_1, \boldsymbol{x}_n), k(\boldsymbol{z}_2, \boldsymbol{x}_n), \ldots, k(\boldsymbol{z}_L, \boldsymbol{x}_n)]$$

Using Kernels to "Extract" Good Features: Landmarks

- Suppose we choose a small set of L "landmark" inputs $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L$ in the training data



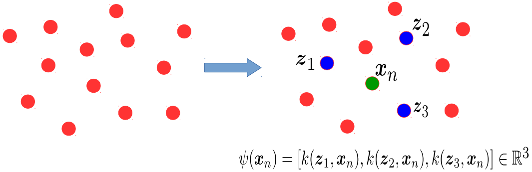
- For each input $\boldsymbol{x}_n$, using a kernel k, define an L-dimensional feature vector as follows

$$\psi(\boldsymbol{x}_n) = [k(\boldsymbol{z}_1, \boldsymbol{x}_n), k(\boldsymbol{z}_2, \boldsymbol{x}_n), \ldots, k(\boldsymbol{z}_L, \boldsymbol{x}_n)]$$

- $\psi(\boldsymbol{x}_n) \in \mathbb{R}^L$ is such that $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) \approx \psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m)$

Using Kernels to "Extract" Good Features: Landmarks

- Suppose we choose a small set of L "landmark" inputs $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L$ in the training data



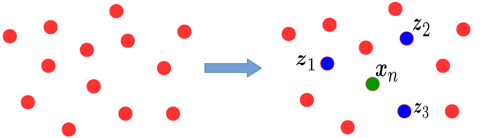
- For each input $\boldsymbol{x}_n$, using a kernel k, define an L-dimensional feature vector as follows

$$\psi(\boldsymbol{x}_n) = [k(\boldsymbol{z}_1, \boldsymbol{x}_n), k(\boldsymbol{z}_2, \boldsymbol{x}_n), \ldots, k(\boldsymbol{z}_L, \boldsymbol{x}_n)]$$

- $\psi(\boldsymbol{x}_n) \in \mathbb{R}^L$ is such that $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) \approx \psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m)$
- Can now apply a linear model on the ψ representation (L-dimensional now) of the inputs

Using Kernels to "Extract" Good Features: Landmarks

- Suppose we choose a small set of L "landmark" inputs $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L$ in the training data



$$\psi(\boldsymbol{x}_n) = [k(\boldsymbol{z}_1, \boldsymbol{x}_n), k(\boldsymbol{z}_2, \boldsymbol{x}_n), k(\boldsymbol{z}_3, \boldsymbol{x}_n)] \in \mathbb{R}^3$$

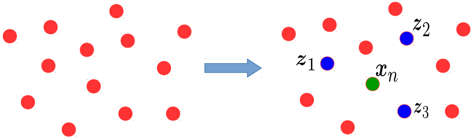
- For each input $\boldsymbol{x}_n$, using a kernel k, define an L-dimensional feature vector as follows

$$\psi(\boldsymbol{x}_n) = [k(\boldsymbol{z}_1, \boldsymbol{x}_n), k(\boldsymbol{z}_2, \boldsymbol{x}_n), \ldots, k(\boldsymbol{z}_L, \boldsymbol{x}_n)]$$

- $\psi(\boldsymbol{x}_n) \in \mathbb{R}^L$ is such that $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) \approx \psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m)$
- Can now apply a linear model on the ψ representation (L-dimensional now) of the inputs
- This will be fast both at training as well as test time if L is small

Using Kernels to "Extract" Good Features: Landmarks

- Suppose we choose a small set of L "landmark" inputs $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L$ in the training data



$\psi(\boldsymbol{x}_n) = [k(\boldsymbol{z}_1, \boldsymbol{x}_n), k(\boldsymbol{z}_2, \boldsymbol{x}_n), k(\boldsymbol{z}_3, \boldsymbol{x}_n)] \in \mathbb{R}^3$

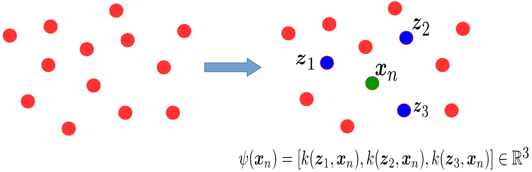
- For each input $\boldsymbol{x}_n$, using a kernel k, define an L-dimensional feature vector as follows

$$\psi(\boldsymbol{x}_n) = [k(\boldsymbol{z}_1, \boldsymbol{x}_n), k(\boldsymbol{z}_2, \boldsymbol{x}_n), \ldots, k(\boldsymbol{z}_L, \boldsymbol{x}_n)]$$

- $\psi(\boldsymbol{x}_n) \in \mathbb{R}^L$ is such that $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) \approx \psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m)$
- Can now apply a linear model on the ψ representation (L-dimensional now) of the inputs
- This will be fast both at training as well as test time if L is small
- No need to kernelize the linear model and work with kernels (but still reap their benefits :-))

Using Kernels to "Extract" Good Features: Landmarks

- Suppose we choose a small set of L "landmark" inputs $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L$ in the training data



- For each input $\boldsymbol{x}_n$, using a kernel k, define an L-dimensional feature vector as follows

$$\psi(\boldsymbol{x}_n) = [k(\boldsymbol{z}_1, \boldsymbol{x}_n), k(\boldsymbol{z}_2, \boldsymbol{x}_n), \ldots, k(\boldsymbol{z}_L, \boldsymbol{x}_n)]$$

- $\psi(\boldsymbol{x}_n) \in \mathbb{R}^L$ is such that $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) \approx \psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m)$
- Can now apply a linear model on the ψ representation (L-dimensional now) of the inputs
- This will be fast both at training as well as test time if L is small
- No need to kernelize the linear model and work with kernels (but still reap their benefits :-))
- Note: The landmarks need not be actual inputs. Can even be learned from data.

Using Kernels to “Extract” Good Features: Random Features

- Many kernel functions can be written as[†]

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w} \sim p(\boldsymbol{w})}[t_{\boldsymbol{w}}(\boldsymbol{x}_n) t_{\boldsymbol{w}}(\boldsymbol{x}_m)]$$

[†] Random Features for Large-Scale Kernel Machines (Ben and Retch, NIPS 2007. Note: This paper actually won the test-of-time award at NIPS 2017)

Using Kernels to "Extract" Good Features: Random Features

- Many kernel functions can be written as[†]

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w} \sim p(\boldsymbol{w})}[t_{\boldsymbol{w}}(\boldsymbol{x}_n) t_{\boldsymbol{w}}(\boldsymbol{x}_m)]$$

where $t_{\boldsymbol{w}}(.)$ is a scalar-valued function with parameters $\boldsymbol{w} \in \mathbb{R}^D$ from some distribution $p(\boldsymbol{w})$

[†] Random Features for Large-Scale Kernel Machines (Ben and Retch, NIPS 2007. Note: This paper actually won the test-of-time award at NIPS 2017)

Using Kernels to "Extract" Good Features: Random Features

- Many kernel functions can be written as[†]

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w} \sim p(\boldsymbol{w})}[t_{\boldsymbol{w}}(\boldsymbol{x}_n) t_{\boldsymbol{w}}(\boldsymbol{x}_m)]$$

  where $t_{\boldsymbol{w}}(.)$ is a scalar-valued function with parameters $\boldsymbol{w} \in \mathbb{R}^D$ from some distribution $p(\boldsymbol{w})$

- Example: For the RBF kernel, $t_{\boldsymbol{w}}(.)$ is cosine function and $p(\boldsymbol{w})$ is zero mean Gausssian

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w} \sim p(\boldsymbol{w})}[\cos(\boldsymbol{w}^\top \boldsymbol{x}_n) \cos(\boldsymbol{w}^\top \boldsymbol{x}_m)]$$

[†] Random Features for Large-Scale Kernel Machines (Ben and Retch, NIPS 2007. Note: This paper actually won the test-of-time award at NIPS 2017)

Using Kernels to "Extract" Good Features: Random Features

- Many kernel functions can be written as[†]

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w}\sim p(\boldsymbol{w})}[t_{\boldsymbol{w}}(\boldsymbol{x}_n) t_{\boldsymbol{w}}(\boldsymbol{x}_m)]$$

  where $t_{\boldsymbol{w}}(.)$ is a scalar-valued function with parameters $\boldsymbol{w} \in \mathbb{R}^D$ from some distribution $p(\boldsymbol{w})$

- Example: For the RBF kernel, $t_{\boldsymbol{w}}(.)$ is cosine function and $p(\boldsymbol{w})$ is zero mean Gausssian

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w}\sim p(\boldsymbol{w})}[\cos(\boldsymbol{w}^\top \boldsymbol{x}_n) \cos(\boldsymbol{w}^\top \boldsymbol{x}_m)]$$

- Given $\boldsymbol{w}_1 \ldots, \boldsymbol{w}_L$ drawn from $p(\boldsymbol{w})$, using Monte-Carlo approximation of expectation above

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) \approx \frac{1}{L}\sum_{\ell=1}^{L} \cos(\boldsymbol{w}_\ell^\top \boldsymbol{x}_n) \cos(\boldsymbol{w}_\ell^\top \boldsymbol{x}_m)$$

[†] Random Features for Large-Scale Kernel Machines (Ben and Retch, NIPS 2007. Note: This paper actually won the test-of-time award at NIPS 2017)

Using Kernels to "Extract" Good Features: Random Features

- Many kernel functions can be written as[†]

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w} \sim p(\boldsymbol{w})}[t_{\boldsymbol{w}}(\boldsymbol{x}_n) t_{\boldsymbol{w}}(\boldsymbol{x}_m)]$$

  where $t_{\boldsymbol{w}}(.)$ is a scalar-valued function with parameters $\boldsymbol{w} \in \mathbb{R}^D$ from some distribution $p(\boldsymbol{w})$

- Example: For the RBF kernel, $t_{\boldsymbol{w}}(.)$ is cosine function and $p(\boldsymbol{w})$ is zero mean Gausssian

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w} \sim p(\boldsymbol{w})}[\cos(\boldsymbol{w}^\top \boldsymbol{x}_n) \cos(\boldsymbol{w}^\top \boldsymbol{x}_m)]$$

- Given $\boldsymbol{w}_1 \ldots, \boldsymbol{w}_L$ drawn from $p(\boldsymbol{w})$, using Monte-Carlo approximation of expectation above

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) \approx \frac{1}{L} \sum_{\ell=1}^{L} \cos(\boldsymbol{w}_\ell^\top \boldsymbol{x}_n) \cos(\boldsymbol{w}_\ell^\top \boldsymbol{x}_m) = \psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m)$$

[†] Random Features for Large-Scale Kernel Machines (Ben and Retch, NIPS 2007. Note: This paper actually won the test-of-time award at NIPS 2017)

Using Kernels to “Extract” Good Features: Random Features

- Many kernel functions can be written as[†]

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w}\sim p(\boldsymbol{w})}[t_{\boldsymbol{w}}(\boldsymbol{x}_n) t_{\boldsymbol{w}}(\boldsymbol{x}_m)]$$

where $t_{\boldsymbol{w}}(.)$ is a scalar-valued function with parameters $\boldsymbol{w} \in \mathbb{R}^D$ from some distribution $p(\boldsymbol{w})$

- Example: For the RBF kernel, $t_{\boldsymbol{w}}(.)$ is cosine function and $p(\boldsymbol{w})$ is zero mean Gausssian

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w}\sim p(\boldsymbol{w})}[\cos(\boldsymbol{w}^\top \boldsymbol{x}_n) \cos(\boldsymbol{w}^\top \boldsymbol{x}_m)]$$

- Given $\boldsymbol{w}_1 \ldots, \boldsymbol{w}_L$ drawn from $p(\boldsymbol{w})$, using Monte-Carlo approximation of expectation above

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) \approx \frac{1}{L}\sum_{\ell=1}^{L} \cos(\boldsymbol{w}_\ell^\top \boldsymbol{x}_n) \cos(\boldsymbol{w}_\ell^\top \boldsymbol{x}_m) = \psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m)$$

where $\psi(\boldsymbol{x}_n) = \frac{1}{\sqrt{L}}[\cos(\boldsymbol{w}_1^\top \boldsymbol{x}_n), \ldots, \cos(\boldsymbol{w}_L^\top \boldsymbol{x}_n)]$ is an L-dim. feature vector (L needs to be set)

[†] Random Features for Large-Scale Kernel Machines (Ben and Retch, NIPS 2007. Note: This paper actually won the test-of-time award at NIPS 2017)

Using Kernels to "Extract" Good Features: Random Features

- Many kernel functions can be written as[†]

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w} \sim p(\boldsymbol{w})}[t_{\boldsymbol{w}}(\boldsymbol{x}_n) t_{\boldsymbol{w}}(\boldsymbol{x}_m)]$$

  where $t_{\boldsymbol{w}}(.)$ is a scalar-valued function with parameters $\boldsymbol{w} \in \mathbb{R}^D$ from some distribution $p(\boldsymbol{w})$

- Example: For the RBF kernel, $t_{\boldsymbol{w}}(.)$ is cosine function and $p(\boldsymbol{w})$ is zero mean Gausssian

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w} \sim p(\boldsymbol{w})}[\cos(\boldsymbol{w}^\top \boldsymbol{x}_n) \cos(\boldsymbol{w}^\top \boldsymbol{x}_m)]$$

- Given $\boldsymbol{w}_1 \ldots, \boldsymbol{w}_L$ drawn from $p(\boldsymbol{w})$, using Monte-Carlo approximation of expectation above

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) \approx \frac{1}{L} \sum_{\ell=1}^{L} \cos(\boldsymbol{w}_\ell^\top \boldsymbol{x}_n) \cos(\boldsymbol{w}_\ell^\top \boldsymbol{x}_m) = \psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m)$$

  where $\psi(\boldsymbol{x}_n) = \frac{1}{\sqrt{L}}[\cos(\boldsymbol{w}_1^\top \boldsymbol{x}_n), \ldots, \cos(\boldsymbol{w}_L^\top \boldsymbol{x}_n)]$ is an L-dim. feature vector (L needs to be set)

- Can apply a linear model on this L-dim representation of the data (no need to kernelize)

[†] Random Features for Large-Scale Kernel Machines (Ben and Retch, NIPS 2007. Note: This paper actually won the test-of-time award at NIPS 2017)

Using Kernels to "Extract" Good Features: Random Features

- Many kernel functions can be written as[†]

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w} \sim p(\boldsymbol{w})}[t_{\boldsymbol{w}}(\boldsymbol{x}_n) t_{\boldsymbol{w}}(\boldsymbol{x}_m)]$$

  where $t_{\boldsymbol{w}}(.)$ is a scalar-valued function with parameters $\boldsymbol{w} \in \mathbb{R}^D$ from some distribution $p(\boldsymbol{w})$

- Example: For the RBF kernel, $t_{\boldsymbol{w}}(.)$ is cosine function and $p(\boldsymbol{w})$ is zero mean Gausssian

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \mathbb{E}_{\boldsymbol{w} \sim p(\boldsymbol{w})}[\cos(\boldsymbol{w}^\top \boldsymbol{x}_n) \cos(\boldsymbol{w}^\top \boldsymbol{x}_m)]$$

- Given $\boldsymbol{w}_1 \ldots, \boldsymbol{w}_L$ drawn from $p(\boldsymbol{w})$, using Monte-Carlo approximation of expectation above

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) \approx \frac{1}{L} \sum_{\ell=1}^{L} \cos(\boldsymbol{w}_\ell^\top \boldsymbol{x}_n) \cos(\boldsymbol{w}_\ell^\top \boldsymbol{x}_m) = \psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m)$$

  where $\psi(\boldsymbol{x}_n) = \frac{1}{\sqrt{L}}[\cos(\boldsymbol{w}_1^\top \boldsymbol{x}_n), \ldots, \cos(\boldsymbol{w}_L^\top \boldsymbol{x}_n)]$ is an L-dim. feature vector (L needs to be set)

- Can apply a linear model on this L-dim representation of the data (no need to kernelize)
- Such techniques exist for several kernels (RBF, polynomial, etc)

[†] Random Features for Large-Scale Kernel Machines (Ben and Retch, NIPS 2007. Note: This paper actually won the test-of-time award at NIPS 2017)

Other Techniques for Speeding Up Kernel Methods

- Reducing the number of support vectors (for SVM based models), For example,
  - Learn the kernelized SV. Identify the support vectors.
  - Cluster the support vectors
  - Pick one SV from each cluster, retrain SVM using the chosen SVs

Other Techniques for Speeding Up Kernel Methods

- Reducing the number of support vectors (for SVM based models), For example,
  - Learn the kernelized SV. Identify the support vectors.
  - Cluster the support vectors
  - Pick one SV from each cluster, retrain SVM using the chosen SVs
- Low-rank approximations of kernel matrix (Nyström approximation)

Kernel Methods: Some Final Comments

- Sometimes, even linear models can be trained via kernelization (but with linear kernel)

Kernel Methods: Some Final Comments

- Sometimes, even linear models can be trained via kernelization (but with linear kernel)
- Benefit? Well, this may be beneficial sometimes due to computational reasons

Kernel Methods: Some Final Comments

- Sometimes, even linear models can be trained via kernelization (but with linear kernel)
- Benefit? Well, this may be beneficial sometimes due to computational reasons
- For example, ridge regression requires solving

$$\boldsymbol{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \boldsymbol{y}$$

.. where we learn $\boldsymbol{w}$ by inverting a $D \times D$ matrix

Kernel Methods: Some Final Comments

- Sometimes, even linear models can be trained via kernelization (but with linear kernel)
- Benefit? Well, this may be beneficial sometimes due to computational reasons
- For example, ridge regression requires solving

$$\boldsymbol{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \boldsymbol{y}$$

.. where we learn $\boldsymbol{w}$ by inverting a $D \times D$ matrix

- Instead, the dual version of Ridge Regression, as we saw earlier, requires solving

$$\boldsymbol{w} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N)^{-1} \boldsymbol{y}$$

Kernel Methods: Some Final Comments

- Sometimes, even linear models can be trained via kernelization (but with linear kernel)
- Benefit? Well, this may be beneficial sometimes due to computational reasons
- For example, ridge regression requires solving

$$\boldsymbol{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \boldsymbol{y}$$

.. where we learn $\boldsymbol{w}$ by inverting a $D \times D$ matrix

- Instead, the dual version of Ridge Regression, as we saw earlier, requires solving

$$\boldsymbol{w} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N)^{-1} \boldsymbol{y} = \mathbf{X}^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n$$

.. where we learn $\boldsymbol{w}$ in terms of $\boldsymbol{\alpha}$ by inverting $N \times N$ matrix

Kernel Methods: Some Final Comments

- Sometimes, even linear models can be trained via kernelization (but with linear kernel)
- Benefit? Well, this may be beneficial sometimes due to computational reasons
- For example, ridge regression requires solving

$$\boldsymbol{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \boldsymbol{y}$$

.. where we learn $\boldsymbol{w}$ by inverting a $D \times D$ matrix

- Instead, the dual version of Ridge Regression, as we saw earlier, requires solving

$$\boldsymbol{w} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N)^{-1} \boldsymbol{y} = \mathbf{X}^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n$$

.. where we learn $\boldsymbol{w}$ in terms of $\boldsymbol{\alpha}$ by inverting $N \times N$ matrix

- Even when working with linear model, if $D > N$, the latter way may be preferable

Kernel Methods: Some Final Comments

- Sometimes, even linear models can be trained via kernelization (but with linear kernel)
- Benefit? Well, this may be beneficial sometimes due to computational reasons
- For example, ridge regression requires solving

$$\boldsymbol{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \boldsymbol{y}$$

.. where we learn $\boldsymbol{w}$ by inverting a $D \times D$ matrix

- Instead, the dual version of Ridge Regression, as we saw earlier, requires solving

$$\boldsymbol{w} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N)^{-1} \boldsymbol{y} = \mathbf{X}^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n$$

.. where we learn $\boldsymbol{w}$ in terms of $\boldsymbol{\alpha}$ by inverting $N \times N$ matrix

- Even when working with linear model, if $D > N$, the latter way may be preferable
- Similar considerations apply to other kernelizable models too (e.g., SVM)

Kernel Methods: Some Final Comments

- Sometimes, even linear models can be trained via kernelization (but with linear kernel)
- Benefit? Well, this may be beneficial sometimes due to computational reasons
- For example, ridge regression requires solving

$$\boldsymbol{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \boldsymbol{y}$$

.. where we learn $\boldsymbol{w}$ by inverting a $D \times D$ matrix

- Instead, the dual version of Ridge Regression, as we saw earlier, requires solving

$$\boldsymbol{w} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N)^{-1} \boldsymbol{y} = \mathbf{X}^\top \boldsymbol{\alpha} = \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n$$

.. where we learn $\boldsymbol{w}$ in terms of $\boldsymbol{\alpha}$ by inverting $N \times N$ matrix

- Even when working with linear model, if $D > N$, the latter way may be preferable
- Similar considerations apply to other kernelizable models too (e.g., SVM)
- If linear model is what you want, still makes sense to look at the relative values of N and D to decide whether to go for the dual (kernelized) formulation of the problem with a linear kernel

Kernel Methods: Some Final Comments

- Kernel methods give us good features to make learning easier
- However, these features are pre-defined (due to the choice of kernel)
- Example: Consider the quadratic kernel applied to input $\boldsymbol{x} = [x_1, x_2]$

$$\phi(\boldsymbol{x}) = [1, \sqrt{2}x_1, x_1^2, \sqrt{2}x_1x_2, x_2, x_2^2, \sqrt{2}x_2] \quad \text{(fixed definition for } \phi\text{)}$$

Kernel Methods: Some Final Comments

- Kernel methods give us good features to make learning easier
- However, these features are pre-defined (due to the choice of kernel)
- Example: Consider the quadratic kernel applied to input $\boldsymbol{x} = [x_1, x_2]$

$$\phi(\boldsymbol{x}) = [1, \sqrt{2}x_1, x_1^2, \sqrt{2}x_1x_2, x_2, x_2^2, \sqrt{2}x_2] \quad \text{(fixed definition for } \phi\text{)}$$

- Another alternative is to learn good features from data

Kernel Methods: Some Final Comments

- Kernel methods give us good features to make learning easier
- However, these features are pre-defined (due to the choice of kernel)
- Example: Consider the quadratic kernel applied to input $\boldsymbol{x} = [x_1, x_2]$

$$\phi(\boldsymbol{x}) = [1, \sqrt{2}x_1, x_1^2, \sqrt{2}x_1x_2, x_2, x_2^2, \sqrt{2}x_2] \quad \text{(fixed definition for } \phi\text{)}$$

- Another alternative is to learn good features from data
- We will revisit this when we talk about deep neural networks

Unsupervised Learning

Unsupervised Learning

- Roughly speaking, it is about learning interesting structures in the data (unsupervisedly!)
- There is no supervision (no labels/responses), only inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$

Unsupervised Learning

- Roughly speaking, it is about learning interesting structures in the data (unsupervisedly!)
- There is no supervision (no labels/responses), only inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- Some examples of unsupervised learning

Unsupervised Learning

- Roughly speaking, it is about learning interesting structures in the data (unsupervisedly!)
- There is no supervision (no labels/responses), only inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- Some examples of unsupervised learning
  - Clustering: Grouping similar inputs together (and dissimilar ones far apart)

Unsupervised Learning

- Roughly speaking, it is about learning interesting structures in the data (unsupervisedly!)
- There is no supervision (no labels/responses), only inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- Some examples of unsupervised learning
  - Clustering: Grouping similar inputs together (and dissimilar ones far apart)
  - Dimensionality Reduction: Reducing the data dimensionality

Unsupervised Learning

- Roughly speaking, it is about learning interesting structures in the data (unsupervisedly!)
- There is no supervision (no labels/responses), only inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- Some examples of unsupervised learning
  - Clustering: Grouping similar inputs together (and dissimilar ones far apart)
  - Dimensionality Reduction: Reducing the data dimensionality
  - Estimating the probability density of data (which distribution "generated" the data)

Unsupervised Learning

- Roughly speaking, it is about learning interesting structures in the data (unsupervisedly!)
- There is no supervision (no labels/responses), only inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- Some examples of unsupervised learning
  - Clustering: Grouping similar inputs together (and dissimilar ones far apart)
  - Dimensionality Reduction: Reducing the data dimensionality
  - Estimating the probability density of data (which distribution "generated" the data)
- Most unsupervised learning algos can also be seen as learning a new representation of data

Unsupervised Learning

- Roughly speaking, it is about learning interesting structures in the data (unsupervisedly!)
- There is no supervision (no labels/responses), only inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- Some examples of unsupervised learning
  - Clustering: Grouping similar inputs together (and dissimilar ones far apart)
  - Dimensionality Reduction: Reducing the data dimensionality
  - Estimating the probability density of data (which distribution "generated" the data)
- Most unsupervised learning algos can also be seen as learning a new representation of data
  - Typically a compressed representation, e.g., clustering can be used to get a one-hot representation

K = 6 clusters

z_n $\boxed{0 \mid 0 \mid 1 \mid 0 \mid 0 \mid 0}$ $\longleftarrow$ x_n belongs to cluster 3

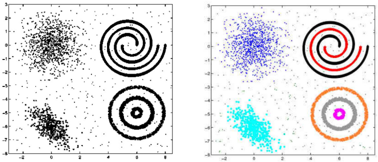
A one-hot (quantized) rep.

Clustering

- Given: N unlabeled examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; no. of desired partitions K
- Goal: Group the examples into K "homogeneous" partitions

Clustering

- Given: N unlabeled examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; no. of desired partitions K
- Goal: Group the examples into K "homogeneous" partitions

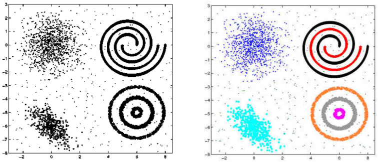


(a) Input data (b) Desired clustering

Picture courtesy: "Data Clustering: 50 Years Beyond K-Means", A.K. Jain (2008)

Clustering

- Given: N unlabeled examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; no. of desired partitions K
- Goal: Group the examples into K "homogeneous" partitions



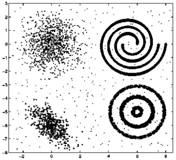
(a) Input data (b) Desired clustering

Picture courtesy: "Data Clustering: 50 Years Beyond K-Means", A.K. Jain (2008)

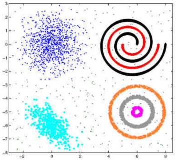
- Loosely speaking, it is classification without ground truth labels

Clustering

- Given: N unlabeled examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; no. of desired partitions K
- Goal: Group the examples into K "homogeneous" partitions



(a) Input data



(b) Desired clustering

Picture courtesy: "Data Clustering: 50 Years Beyond K-Means", A.K. Jain (2008)

- Loosely speaking, it is classification without ground truth labels
- A good clustering is one that achieves:
  - High within-cluster similarity
  - Low inter-cluster similarity

Similarity can be Subjective

- Clustering only looks at similarities, no labels are given
- Without labels, similarity can be hard to define

Similarity can be Subjective

- Clustering only looks at similarities, no labels are given
- Without labels, similarity can be hard to define

- Thus using the right distance/similarity is very important in clustering

Similarity can be Subjective

- Clustering only looks at similarities, no labels are given
- Without labels, similarity can be hard to define

- Thus using the right distance/similarity is very important in clustering
- Also important to define/ask: "Clustering based on what"?

Similarity can be Subjective

- Clustering only looks at similarities, no labels are given
- Without labels, similarity can be hard to define

- Thus using the right distance/similarity is very important in clustering
- Also important to define/ask: "Clustering based on what"?

Picture courtesy: `http://www.guy-sports.com/humor/videos/powerpoint_presentation_dogs.htm`

Clustering: Some Examples

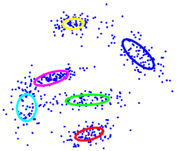
- Document/Image/Webpage Clustering
- Image Segmentation (clustering pixels)

- Clustering web-search results
- Clustering (people) nodes in (social) networks/graphs
- .. and many more..

Picture courtesy: `http://people.cs.uchicago.edu/~pff/segment/`

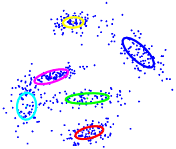
Types of Clustering

1. **Flat or Partitional clustering**
   - Partitions are independent of each other

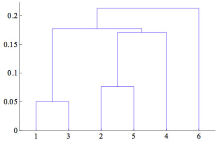


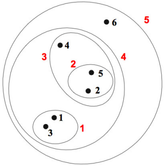
Types of Clustering

1. **Flat or Partitional clustering**
   - Partitions are independent of each other



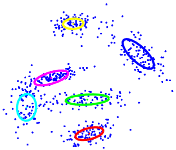
2. **Hierarchical clustering**
   - Partitions can be visualized using a tree structure (a dendrogram)



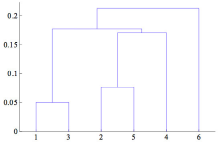


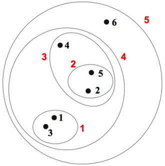
Types of Clustering

1. **Flat or Partitional clustering**
   - Partitions are independent of each other



2. **Hierarchical clustering**
   - Partitions can be visualized using a tree structure (a dendrogram)





   - Possible to view partitions at different levels of granularities by "cutting" the tree at some level

Flat Clustering: K-means algorithm (Lloyd, 1957)

- **Input:** N examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; $\boldsymbol{x}_n \in \mathbb{R}^D$; the number of partitions K
- **Desired Output:** Cluster assignments of these N examples and K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$

Flat Clustering: K-means algorithm (Lloyd, 1957)

- **Input:** N examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; $\boldsymbol{x}_n \in \mathbb{R}^D$; the number of partitions K
- **Desired Output:** Cluster assignments of these N examples and K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$
- **Initialize:** K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, each $\boldsymbol{\mu}_k \in \mathbb{R}^D$

Flat Clustering: K-means algorithm (Lloyd, 1957)

- **Input:** N examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; $\boldsymbol{x}_n \in \mathbb{R}^D$; the number of partitions K
- **Desired Output:** Cluster assignments of these N examples and K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$
- **Initialize:** K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, each $\boldsymbol{\mu}_k \in \mathbb{R}^D$
  - Usually initialized randomly, but good initialization is crucial; many smarter initialization heuristics exist (e.g., K-means++, Arthur & Vassilvitskii, 2007)

Flat Clustering: K-means algorithm (Lloyd, 1957)

- **Input:** N examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; $\boldsymbol{x}_n \in \mathbb{R}^D$; the number of partitions K
- **Desired Output:** Cluster assignments of these N examples and K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$
- **Initialize:** K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, each $\boldsymbol{\mu}_k \in \mathbb{R}^D$
    - Usually initialized randomly, but good initialization is crucial; many smarter initialization heuristics exist (e.g., K-means++, Arthur & Vassilvitskii, 2007)
- **Iterate:**

Flat Clustering: K-means algorithm (Lloyd, 1957)

- **Input:** N examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; $\boldsymbol{x}_n \in \mathbb{R}^D$; the number of partitions K
- **Desired Output:** Cluster assignments of these N examples and K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$
- **Initialize:** K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, each $\boldsymbol{\mu}_k \in \mathbb{R}^D$
  - Usually initialized randomly, but good initialization is crucial; many smarter initialization heuristics exist (e.g., K-means++, Arthur & Vassilvitskii, 2007)
- **Iterate:**
  - (Re)-Assign each example $\boldsymbol{x}_n$ to its closest cluster center (based on the smallest Euclidean distance)

$$\boxed{\mathcal{C}_k = \{n : \quad k = \arg\min_k ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2\}}$$

  ($\mathcal{C}_k$ is the set of examples assigned to cluster k with center $\boldsymbol{\mu}_k$)

Flat Clustering: K-means algorithm (Lloyd, 1957)

- **Input:** N examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; $\boldsymbol{x}_n \in \mathbb{R}^D$; the number of partitions K
- **Desired Output:** Cluster assignments of these N examples and K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$
- **Initialize:** K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, each $\boldsymbol{\mu}_k \in \mathbb{R}^D$
  - Usually initialized randomly, but good initialization is crucial; many smarter initialization heuristics exist (e.g., K-means++, Arthur & Vassilvitskii, 2007)
- **Iterate:**
  - (Re)-Assign each example $\boldsymbol{x}_n$ to its closest cluster center (based on the smallest Euclidean distance)

$$\boxed{\mathcal{C}_k = \{n : \quad k = \arg\min_k ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2\}}$$

    ($\mathcal{C}_k$ is the set of examples assigned to cluster k with center $\boldsymbol{\mu}_k$)
  - Update the cluster means

$$\boxed{\mu_k = \text{mean}(\mathcal{C}_k) = \frac{1}{|\mathcal{C}_k|} \sum_{n \in \mathcal{C}_k} \boldsymbol{x}_n}$$

Flat Clustering: K-means algorithm (Lloyd, 1957)

- **Input:** N examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; $\boldsymbol{x}_n \in \mathbb{R}^D$; the number of partitions K
- **Desired Output:** Cluster assignments of these N examples and K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$
- **Initialize:** K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, each $\boldsymbol{\mu}_k \in \mathbb{R}^D$
  - Usually initialized randomly, but good initialization is crucial; many smarter initialization heuristics exist (e.g., K-means++, Arthur & Vassilvitskii, 2007)
- **Iterate:**
  - (Re)-Assign each example $\boldsymbol{x}_n$ to its closest cluster center (based on the smallest Euclidean distance)

$$\boxed{\mathcal{C}_k = \{n : \quad k = \arg\min_k ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2\}}$$

    ($\mathcal{C}_k$ is the set of examples assigned to cluster k with center $\boldsymbol{\mu}_k$)

  - Update the cluster means

$$\boxed{\mu_k = \text{mean}(\mathcal{C}_k) = \frac{1}{|\mathcal{C}_k|} \sum_{n \in \mathcal{C}_k} \boldsymbol{x}_n}$$

  - Repeat while not converged

Flat Clustering: K-means algorithm (Lloyd, 1957)

- **Input:** N examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; $\boldsymbol{x}_n \in \mathbb{R}^D$; the number of partitions K
- **Desired Output:** Cluster assignments of these N examples and K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$
- **Initialize:** K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, each $\boldsymbol{\mu}_k \in \mathbb{R}^D$
  - Usually initialized randomly, but good initialization is crucial; many smarter initialization heuristics exist (e.g., K-means++, Arthur & Vassilvitskii, 2007)
- **Iterate:**
  - (Re)-Assign each example $\boldsymbol{x}_n$ to its closest cluster center (based on the smallest Euclidean distance)

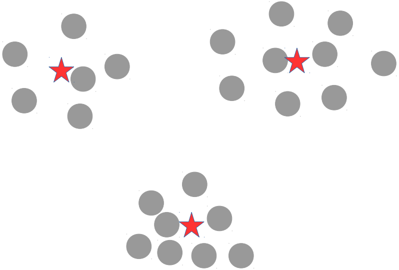
$$\boxed{\mathcal{C}_k = \{n : \quad k = \arg\min_k ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2\}}$$

    ($\mathcal{C}_k$ is the set of examples assigned to cluster k with center $\boldsymbol{\mu}_k$)
  - Update the cluster means

$$\boxed{\mu_k = \text{mean}(\mathcal{C}_k) = \frac{1}{|\mathcal{C}_k|} \sum_{n \in \mathcal{C}_k} \boldsymbol{x}_n}$$

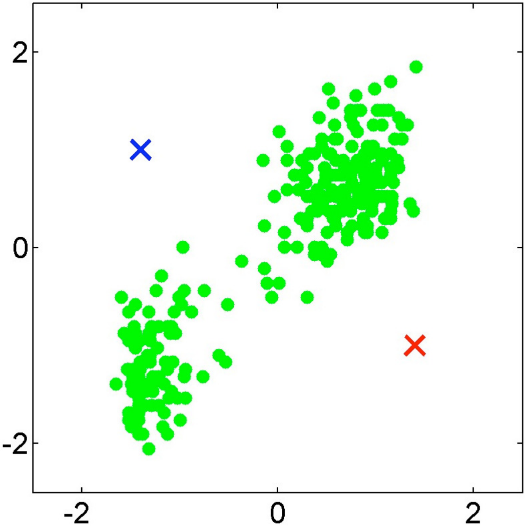
  - Repeat while not converged
- Stop when cluster means or the "loss" does not change by much

K-means = Prototype Classification (with unknown labels)

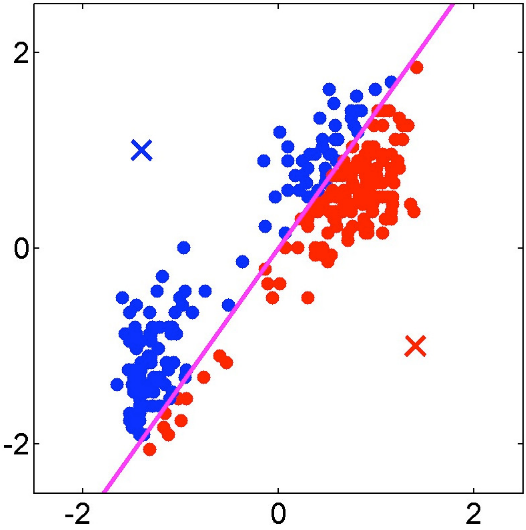


- **Guess the means**
- **Predict the labels**
- **Recompute the means**
- **Repeat**

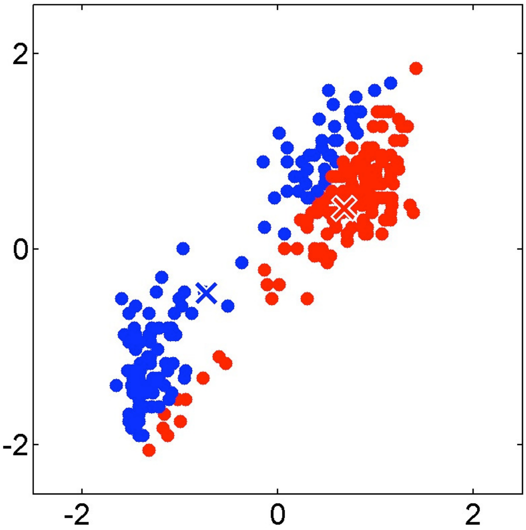
K-means: Initialization (assume $K = 2$)



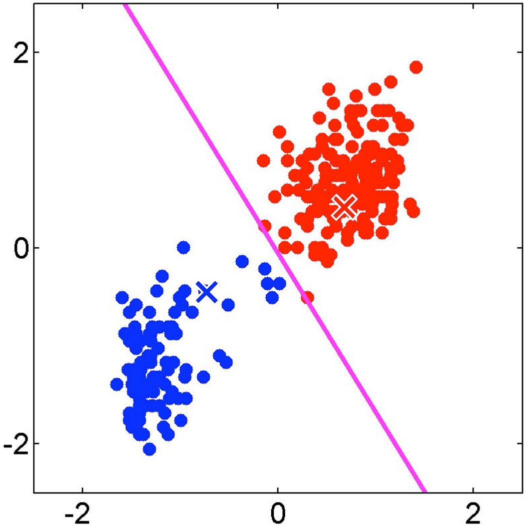
K-means iteration 1: Assigning points



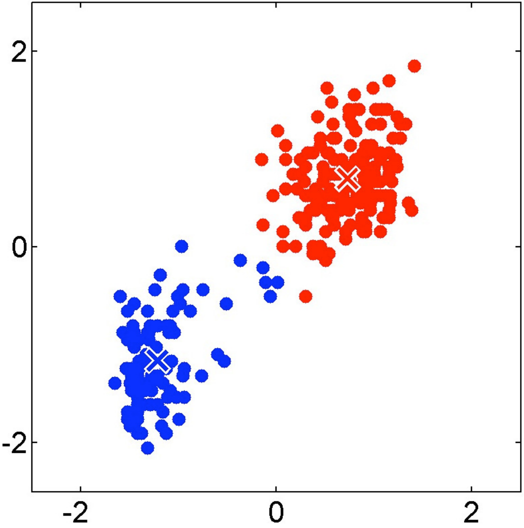
K-means iteration 1: Recomputing the centers



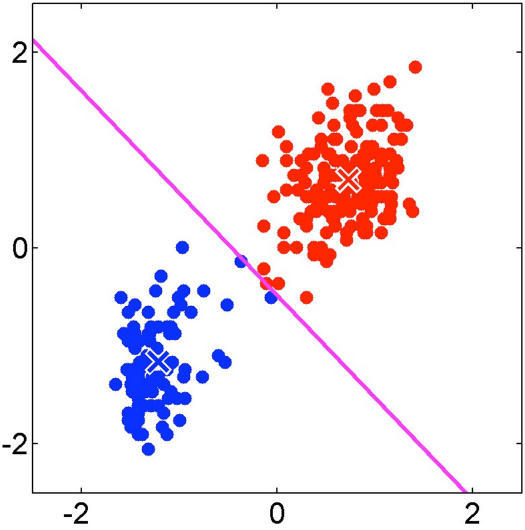
K-means iteration 2: Assigning points



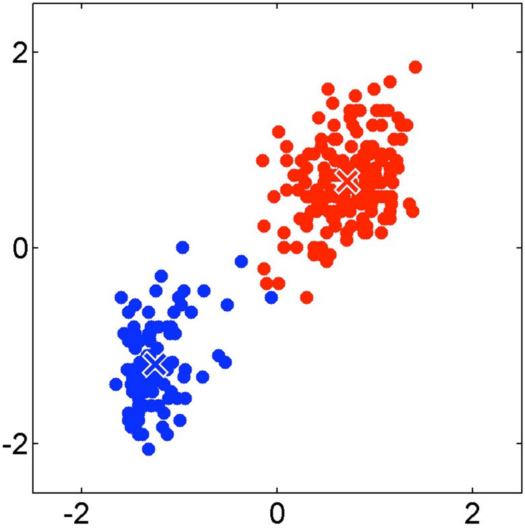
K-means iteration 2: Recomputing the centers



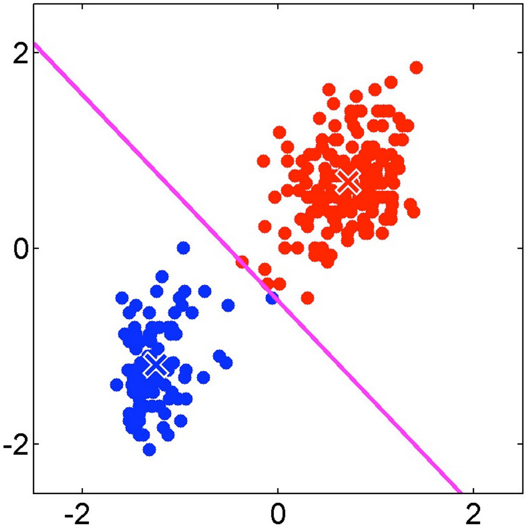
K-means iteration 3: Assigning points



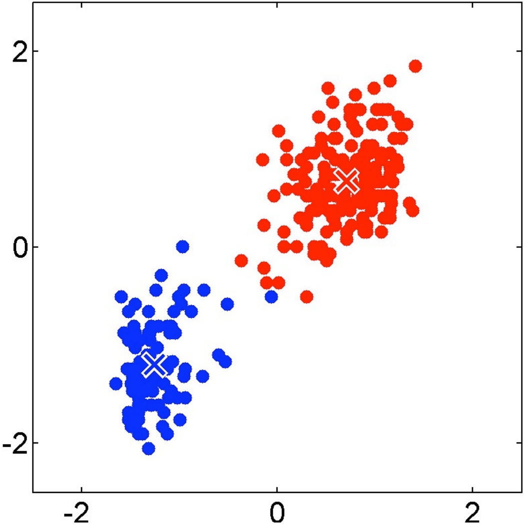
K-means iteration 3: Recomputing the centers



K-means iteration 4: Assigning points



K-means iteration 4: Recomputing the centers



What Loss Function is K-means Optimizing?

What Loss Function is K-means Optimizing?

- Let $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ be the K cluster centroids (means)
- Let $z_{nk} \in \{0, 1\}$ be s.t. $z_{nk} = 1$ if $\boldsymbol{x}_n$ belongs to cluster k, and 0 otherwise

What Loss Function is K-means Optimizing?

- Let $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ be the K cluster centroids (means)
- Let $z_{nk} \in \{0, 1\}$ be s.t. $z_{nk} = 1$ if $\boldsymbol{x}_n$ belongs to cluster k, and 0 otherwise
  - Note: $\boldsymbol{z}_n = [z_{n1} \; z_{n2} \; \ldots \; z_{nK}]$ represents a length K one-hot encoding of $\boldsymbol{x}_n$

What Loss Function is K-means Optimizing?

- Let $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ be the K cluster centroids (means)
- Let $z_{nk} \in \{0, 1\}$ be s.t. $z_{nk} = 1$ if $\boldsymbol{x}_n$ belongs to cluster k, and 0 otherwise
  - Note: $\boldsymbol{z}_n = [z_{n1} \; z_{n2} \; \ldots \; z_{nK}]$ represents a length K one-hot encoding of $\boldsymbol{x}_n$
- Define the distortion or "loss" for the cluster assignment of $\boldsymbol{x}_n$

$$\ell(\boldsymbol{\mu}, \boldsymbol{x}_n, \boldsymbol{z}_n) = \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$$

What Loss Function is K-means Optimizing?

- Let $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ be the K cluster centroids (means)
- Let $z_{nk} \in \{0, 1\}$ be s.t. $z_{nk} = 1$ if $\boldsymbol{x}_n$ belongs to cluster k, and 0 otherwise
  - Note: $\boldsymbol{z}_n = [z_{n1} \; z_{n2} \; \ldots \; z_{nK}]$ represents a length K one-hot encoding of $\boldsymbol{x}_n$
- Define the distortion or "loss" for the cluster assignment of $\boldsymbol{x}_n$

$$\ell(\boldsymbol{\mu}, \boldsymbol{x}_n, \boldsymbol{z}_n) = \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$$

- Total distortion over all points defines the K-means "loss function"

$$L(\boldsymbol{\mu}, \mathbf{X}, \mathbf{Z}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \mu_k||^2$$

What Loss Function is K-means Optimizing?

- Let $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ be the K cluster centroids (means)
- Let $z_{nk} \in \{0, 1\}$ be s.t. $z_{nk} = 1$ if $\boldsymbol{x}_n$ belongs to cluster k, and 0 otherwise
  - Note: $\boldsymbol{z}_n = [z_{n1} \; z_{n2} \; \ldots \; z_{nK}]$ represents a length K one-hot encoding of $\boldsymbol{x}_n$
- Define the distortion or "loss" for the cluster assignment of $\boldsymbol{x}_n$

$$\ell(\boldsymbol{\mu}, \boldsymbol{x}_n, \boldsymbol{z}_n) = \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$$

- Total distortion over all points defines the K-means "loss function"

$$L(\boldsymbol{\mu}, \mathbf{X}, \mathbf{Z}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \mu_k||^2 = \underbrace{||\mathbf{X} - \mathbf{Z}\boldsymbol{\mu}||_F^2}_{\text{matrix factorization view}}$$

where $\mathbf{Z}$ is $N \times K$ (row n is $\boldsymbol{z}_n$) and $\boldsymbol{\mu}$ is $K \times D$ (row k is $\boldsymbol{\mu}_k$)

What Loss Function is K-means Optimizing?

- Let $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ be the K cluster centroids (means)
- Let $z_{nk} \in \{0, 1\}$ be s.t. $z_{nk} = 1$ if $\boldsymbol{x}_n$ belongs to cluster k, and 0 otherwise
  - Note: $\boldsymbol{z}_n = [z_{n1} \; z_{n2} \; \ldots \; z_{nK}]$ represents a length K one-hot encoding of $\boldsymbol{x}_n$
- Define the distortion or "loss" for the cluster assignment of $\boldsymbol{x}_n$

$$\ell(\boldsymbol{\mu}, \boldsymbol{x}_n, \boldsymbol{z}_n) = \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$$

- Total distortion over all points defines the K-means "loss function"

$$L(\boldsymbol{\mu}, \mathbf{X}, \mathbf{Z}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2 = \underbrace{||\mathbf{X} - \mathbf{Z}\boldsymbol{\mu}||_F^2}_{\text{matrix factorization view}}$$

  where $\mathbf{Z}$ is $N \times K$ (row n is $\boldsymbol{z}_n$) and $\boldsymbol{\mu}$ is $K \times D$ (row k is $\boldsymbol{\mu}_k$)

- The K-means **problem** is to minimize this objective w.r.t. $\boldsymbol{\mu}$ and $\mathbf{Z}$

What Loss Function is K-means Optimizing?

- Let $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ be the K cluster centroids (means)
- Let $z_{nk} \in \{0, 1\}$ be s.t. $z_{nk} = 1$ if $\boldsymbol{x}_n$ belongs to cluster k, and 0 otherwise
  - Note: $\boldsymbol{z}_n = [z_{n1}\ z_{n2}\ \ldots\ z_{nK}]$ represents a length K one-hot encoding of $\boldsymbol{x}_n$
- Define the distortion or "loss" for the cluster assignment of $\boldsymbol{x}_n$

$$\ell(\boldsymbol{\mu}, \boldsymbol{x}_n, \boldsymbol{z}_n) = \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$$

- Total distortion over all points defines the K-means "loss function"

$$L(\mu, \mathbf{X}, \mathbf{Z}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \mu_k||^2 = \underbrace{||\mathbf{X} - \mathbf{Z}\boldsymbol{\mu}||_F^2}_{\text{matrix factorization view}}$$

where $\mathbf{Z}$ is $N \times K$ (row n is $\boldsymbol{z}_n$) and $\boldsymbol{\mu}$ is $K \times D$ (row k is $\boldsymbol{\mu}_k$)

- The K-means **problem** is to minimize this objective w.r.t. $\boldsymbol{\mu}$ and $\mathbf{Z}$
  - Alternating optimization would give the K-means (Lloyd's) algorithm we saw earlier!

Next Class: Clustering (Contd.)