CS 335: Kernel Trick

Dan Sheldon

Big Picture: So Far

- Cost function paradigm for supervised machine learning
  - Input $\mathbf{x}$
  - Output $\mathbf{y}$
  - Find $h_{\boldsymbol{\theta}}(\mathbf{x})$ such that $h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \approx y^{(i)}$
  - Cost function $J(\boldsymbol{\theta})$
  - Regularization to avoid overfitting
- Everything so far has been based on **linear models** $h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\boldsymbol{\theta}^T\mathbf{x})$

Kernel-Trick Motivation

- But what we really want are flexible non-linear classifers!
  - How can we get this with linear methods?
  - Kernel trick!
- Wait... feature expansions already allow non-linear learning...

$$(x_1, x_2) \mapsto (1, x_1, x_2, x_1^2, x_2^2, x_1x_2)$$

How to apply feature mappings? Do they really give a flexible class of non-linear models? How many features? And which ones?

- We would like something more "automatic"
- We don't want to expand our datasets to many times their original size
- **Kernel trick**: non-linear feature expansions in implicit way way
  - Computationally efficient
  - Don't actually do expansion

Kernel Trick Starting Point

Assumption (*): $\boldsymbol{\theta} = \sum_{i=1}^m \alpha_i \mathbf{x}^{(i)}$ for some $\alpha_1, \ldots, \alpha_m$

- $\boldsymbol{\theta}$ in span of feature vectors
- We'll discuss later how to find $\alpha_1, \ldots, \alpha_m$

Linear Regression

Assumption (*): $\boldsymbol{\theta} = \sum_{i=1}^m \alpha_i \mathbf{x}^{(i)}$.

What does linear regression hypothesis look like?

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T\mathbf{x} = \Big(\sum_{i=1}^m \alpha_i \mathbf{x}^{(i)}\Big)^T \mathbf{x} = \sum_{i=1}^m \alpha_i (\mathbf{x}^{(i)})^T \mathbf{x} = \sum_{i=1}^m \alpha_i K(\mathbf{x}^{(i)}, \mathbf{x})$$

- $K(\mathbf{x}, \mathbf{z}) := \mathbf{x}^T\mathbf{z}$ is the "kernel function" (just dot product for now)
- Predictions only depend on training data through kernel function! (dot products)

Linear Regression

Assumption (*): $\boldsymbol{\theta} = \sum_{i=1}^m \alpha_i \mathbf{x}^{(i)}$. Then $h_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{i=1}^m \alpha_i K(\mathbf{x}^{(i)}, x)$.

What does linear regression cost function look like?

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{k=1}^m \left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(k)}) - y^{(k)}\right)^2$$
$$= \frac{1}{2}\sum_{k=1}^m \left(\sum_{i=1}^m \alpha_i K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) - y^{(k)}\right)^2 := J(\boldsymbol{\alpha})$$

- Cost function only depends on training data through kernel function! (dot products)
- How to find $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$? Minimize $J(\boldsymbol{\alpha})$. (More later)
- Note: (*) only needs to hold for $\boldsymbol{\theta}$ that minimizes $J(\boldsymbol{\theta})$

Takeaway

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{i=1}^m \alpha_i K(\mathbf{x}^{(i)}, \mathbf{x})$$
$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{k=1}^m \left(\sum_{i=1}^m \alpha_i K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) - y^{(k)}\right)^2$$

Thought experiment: I hold feature vectors in a box. You can ask me only for dot products. Can you still train model? Make predictions? Yes!

Concrete Example: "Kernelized" Linear Regression

- **Observation:** can rewrite linear regression as a *different* linear regression model:

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_i \alpha_i K(\mathbf{x}^{(i)}, \mathbf{x}) = \boldsymbol{\alpha}^T k(\mathbf{x})$$

$$\boldsymbol{\alpha}^T = \begin{bmatrix}\alpha_1 & \ldots & \alpha_m\end{bmatrix}, \quad k(\mathbf{x}) = \begin{bmatrix} K(\mathbf{x}^{(1)}, \mathbf{x}) \\ K(\mathbf{x}^{(2)}, \mathbf{x}) \\ \vdots \\ K(\mathbf{x}^{(m)}, \mathbf{x}) \end{bmatrix}$$

- Map $\mathbf{x}$ to new "feature vector" $k(\mathbf{x})$ (= kernel evaluation between $\mathbf{x}$ and each training feature vector.
- What happens to original data matrix X under this mapping? (Recall: ith row of X is ith feature vector $\mathbf{x}^{(i)}$.)

- We get a new "data matrix" K, whose ith row is holds dot products between $\mathbf{x}^{(i)}$ and each *other* training point:

$$K_{ij} = K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- This is called the kernel matrix of a training set
- Our reasoning so far says you can learn an equivalent linear model using the kernel matrix in place of the original data matrix.
- Demo
- Note: this equivalance is only exact without regularization. In practice: use a different optimization method to find $\boldsymbol{\alpha}$ to minimize $J(\boldsymbol{\alpha})$

Linear Models

Same reasoning applies more generally to any linear model of this form:

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\boldsymbol{\theta}^T \mathbf{x})$$
$$J(\boldsymbol{\theta}) = \sum_{k=1}^m \text{cost}(\boldsymbol{\theta}^T \mathbf{x}^{(k)}, y^{(k)})$$

- Does this include logistic regression? Yes.
- Substitute $\boldsymbol{\theta} = \sum_{i=1}^m \alpha_i \mathbf{x}^{(i)}$ and observe that cost function and hypothesis only depend on training data through dot products.
- Can fit model by substituting kernel matrix for data matrix.

Kernel Trick

- This doesn't seem that special...
- Real trick: fancy non-linear feature expansions in a computationally efficient way
- Suppose we want to do feature expansion before learning

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \phi(\mathbf{x}), \quad \phi : \mathbb{R}^n \to \mathbb{R}^p$$

- To solve the learning problem and make predictions, we only need to be able to compute $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$. This is called the kernel corresponding to ϕ.

Example: Polynomial Kernel

Important trick: we can often compute kernel without actually doing the expansion

$$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2$$

Claim: this is the kernel corresponding to $\phi(\mathbf{x}) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_2 x_1 \\ x_2 x_2 \end{bmatrix}$

Exercise: verify this on board

More Polynomial Kernels

Claim: these two are equivalent

$$\phi(\mathbf{x}) = \begin{bmatrix} 1 \\ \sqrt{2} x_1 \\ \sqrt{2} x_2 \\ x_1 x_1 \\ x_1 x_2 \\ x_2 x_1 \\ x_2 x_2 \end{bmatrix} \qquad K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z} + 1)^2$$

- Complexity of computing $\phi(\mathbf{x})^T \phi(\mathbf{z})$?
- Complexity of computing $\mathbf{x}^T \mathbf{z}$?
- Complexity of computing $(\mathbf{x}^T \mathbf{z} + 1)^2$?

Polynomial Kernel: Significance

- Compute $\phi(\mathbf{x})^T \phi(\mathbf{z})$: $O(n^2)$
- Compute $\mathbf{x}^T \mathbf{z}$: $O(n)$
- Compute $(\mathbf{x}^T \mathbf{z} + 1)^2$: $O(n)$

If using *kernel trick*, can implement a non-linear feature expansion at no additional cost

Even More Polynomial Kernels

$$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z} + 1)^d$$

Corresponds to ϕ that takes all products of up to d original features

$O(n)$ time to compute kernel instead of $O(n^d)$

Gaussian Kernel

$$K(\mathbf{x}, \mathbf{z}) = \exp(-\gamma ||\mathbf{x} - \mathbf{z}||^2)$$

- Highly flexible, non-linear kernel
- Corresponds to infinite dimensional ϕ (cannot implement feature mapping, but can still use kernel)
- Demos
  - Gaussian kernel intuition: similarity function
  - Linear regression

A Word on Regularization

- Suppose we want to combine feature expansion with regularization

$$J(\boldsymbol{\theta}) = \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 + \sum_{k=1}^{m} \text{cost}\Big(\boldsymbol{\theta}^T \phi(\mathbf{x}^{(k)}), y^{(k)}\Big)$$

- Assume $\boldsymbol{\theta} = \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}^{(i)})$. Then regularization term becomes

$$\|\boldsymbol{\theta}\|^2 = \boldsymbol{\theta}^T \boldsymbol{\theta} = \boldsymbol{\alpha}^T K \boldsymbol{\alpha}$$

  (derivation next slide)

- This is *not* the same as penalizing $\|\boldsymbol{\alpha}\|^2$
  - Tip: use regularization with kernelized linear models
  - Tip: Use a custom optimizer for to minimize $J(\boldsymbol{\alpha})$

A Word on Regularization

Derivation of regularization term:

$$\begin{aligned}\boldsymbol{\theta}^T\boldsymbol{\theta} &= \left(\sum_{i=1}^{m}\alpha_i\phi(\mathbf{x}^{(i)})\right)^T\left(\sum_{j=1}^{m}\alpha_i\phi(\mathbf{x}^{(j)})\right)\\ &= \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j\phi(\mathbf{x}^{(i)})^T\phi(\mathbf{x}^{(j)})\\ &= \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j K(\mathbf{x}^{(i)},\mathbf{x}^{(j)})\\ &= \boldsymbol{\alpha}^T K\boldsymbol{\alpha}\end{aligned}$$

Practical Tips

- Use support vector machines (SVMs) for kernelized classification
  - Like logistic regression, with *slightly* different loss function. (Derivation based on geometric principles, but end point the same.)
  - More efficient than logistic regression when used with kernels (many α_i values are **zero**)
- Use kernel ridge regression or support vector regression for kernelized regression
- Use Gaussian kernels
- Use regularization with kernels
- How to select λ and γ? Cross-validation!

Demos

- Kernel logistic regression
- SVM loss
- SVM classification