EE445 Mod2-Lec4: Kernel Regression

What is Kernel Regression?

- We have been talking about supervised ML in the context of data fitting with least squares
- Cost function paradigm for supervised machine learning
  - Features x
  - Output/response y
  - Goal: Find $f(x)$ such that $f(x^{(i)}) \approx y^{(i)}$
  - objective/cost function $F(\theta) = \|A\theta - y\|^2$

Kernel Motivation

- But what we really want are flexible non-linear classifers/predictors!
- We can get this via a linear model using the kernel trick
- Note that feature maps are already all non-linear

$$x \mapsto 1, x, x^2, \dots$$

- Yet, we want something a little more automatic that implicitly captures nonlinearities without expanding out data to many times the original size
- Kernels give us this

Kernel Trick: Starting Point

- [A2]:

$$\theta = \sum_{i=1}^{m} \alpha_i x^{(i)} \quad \text{for some} \quad \alpha_1, \ldots, \alpha_m \in \mathbb{R}$$

  i.e., θ is in the span of the feature vectors
- We will see shortly how to find these α_i's

Kernel Trick: Linear Regression

- [A2]: $\theta = \sum_{i=1}^{m} \alpha_i x^{(i)}$ for some $\alpha_1, \ldots, \alpha_m \in \mathbb{R}$

- **Kernel function**: $K(x, z) = x^\top z$
- Predictions only depend on training data through kernel function which is just a dot product.

Linear Regression: Objective Function

- [A2]: $\theta = \sum_{i=1}^{m} \alpha_i x^{(i)}$ for some $\alpha_1, \ldots, \alpha_m \in \mathbb{R}$
- The predictor has the form

$$f(x) = \sum_{i=1}^{m} \alpha_i K(x^{(i)}, x)$$

- The objective function has the form

- Objective function only depends on training data through kernel function which is just dot products
- Choose α by minimizing $F(\alpha)$

Kernel Trick: Take-Aways

- Predictor and objective only depend on training data through the kernel which is itself just dot products
- Hence, if we only have the ability to do dot product operations, then we can still suprisingly train a model (i.e., find a prediction of y)

Kernelized Linear Regression

- Rewrite linear regression as a different linear regression model:

$$f(x) = \sum_{i=1}^{m} \alpha_i K(x^{(i)}, x) = \alpha^\top k(x)$$

where

$$\alpha^\top = \begin{bmatrix} \alpha_1 & \cdots & \alpha_m \end{bmatrix} \quad \text{and} \quad k(x) = \begin{bmatrix} K(x^{(1)}, x) \\ \vdots \\ K(x^{(m)}, x) \end{bmatrix}$$

- i.e., we map x to a new "feature vector" $k(x)$ (= kernel evaluation between x and each training feature vector).

What happens to original data matrix X under this mapping?

- Recall: i–th row of X is i–th feature vector $x^{(i)}$
- Kernel Matrix: new "data matrix" K such that the i–th row contains dot products between $x^{(i)}$ and every other training point:

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

- Sometimes this is called the Kernel Trick.
- Take-Away: you can learn an equivalent linear model using the kernel matrix in place of the original data matrix.
- this equivalence is only exact without regularization (I will talk about this shortly)

Nonlinear Feature Maps

- Suppose we want to do feature mappings before learning such as

$$f(x) = \theta^\top \phi(x), \quad \phi : \mathbb{R}^n \to \mathbb{R}^p$$

- **Kernel corresponding to ϕ:** To solve the learning problem and make predictions, we only need to be able to compute

$$K(x, z) = \phi(x)^\top \phi(z)$$

Examples: Polynomial Kernel

Note: we can often compute kernel without actually doing the expansion

- Consider $K(x, z) = (x^\top z)^2$
- **What is $\phi : \mathbb{R}^2 \to \mathbb{R}^4$?**

$$\phi(x) = (x_1^2, x_1 x_2, x_2 x_1, x_2^2)$$

- Check:

Examples: Polynomial Kernel

Note: computational complexity is lower

- Consider $K(x,z) = (x^\top z + 1)^2$
- **What is $\phi : \mathbb{R}^2 \to \mathbb{R}^4$?**

$$\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_1x_2, x_2x_1, x_2^2)$$

- complexity of $\phi(x)^\top \phi(z)$: $O(n^2)$
- complexity of $x^\top z$: $O(n)$
- complexity of $(x^\top z + 1)^2$: $O(n)$
- If using kernel trick, can implement a non-linear feature expansion at no additional cost
- **More general**: $K(x,z) = (x^\top z + 1)^d$
  - complexity of computing corresponding features with ϕ: $O(n^d)$
  - complexity of computing K: $O(n)$

Example: Gaussian Kernel

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

Some observations:

- non-linear kernel with a lot of flexibility
- corresponds to an infinite dimensional ϕ—i.e., cannot implement the corresponding feature mapping ϕ.

In Practice: Regularization

- We often introduce a regularization term in practice:

$$F(\theta) = \sum_{k=1}^{m} (\theta^\top \phi(x^{(k)}) - y^{(k)})^2 + \frac{\lambda}{2}\|\theta\|^2$$

- **why?**: Regularization improves the conditioning of the problem and reduces the variance of the estimates.
- Taking derivatives and setting them to zero we have

Deriving the α-dependent regularization term

Recall that we converted $F(\theta)$ to a cost in terms of α. We will do the same thing for the regularized cost.

- [A2]: $\theta = \sum_{i=1}^{m} \alpha_i x^{(i)}$ for some $\alpha_1, \ldots, \alpha_m \in \mathbb{R}$

Kernelized Regression Regularized Cost (Ridge Regression)

$$F(\alpha) = \frac{1}{2}\|K\alpha - y\|^2 + \frac{\lambda}{2}\alpha^\top K\alpha$$

- Choose λ via cross validation!