COMPSCI 689
Lecture 7: Kernel Representations and Kernelized Linear Regression

Brendan O'Connor

College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).

Outline

Review

- We now have methods for learning basic linear models for both regression and classification.

Review

- We now have methods for learning basic linear models for both regression and classification.
- **Question:** What can we do when the data we are trying to model have non-linear relationships between inputs and outputs?

Outline

Basis Function Expansion

- A simple solution to the linearity problem is to apply a set of functions $\phi_1,...,\phi_K$ to the raw feature vector $\mathbf{x}$ to map it in to a new feature space:

$$\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), ..., \phi_K(\mathbf{x})]$$

- This is called a *basis function expansion* since $K > D$ in general. This requires that we know the functions $\phi_1,...,\phi_K$ that we want to apply in advance. (*Feature engineering*: manually experimenting to develop ϕs)
- Then define a linear model in this new feature space: $\theta \in \mathbb{R}^K$.
- In the regression setting, we obtain the model $\hat{y} = \phi(\mathbf{x})\theta$.
- In the classification setting, we obtain the model $\hat{y} = \text{sign}(\phi(\mathbf{x})\theta)$.

Basis Function Expansion Examples

- **Univariate Functions:** We can set $\phi_k(\mathbf{x})$ to any univariate function of a single x_d to obtain mappings like $\phi(\mathbf{x}) = [x_1, x_2, \sin(x_1), \exp(x_2)]$, etc.

Basis Function Expansion Examples

- **Univariate Functions:** We can set $\phi_k(\mathbf{x})$ to any univariate function of a single x_d to obtain mappings like $\phi(\mathbf{x}) = [x_1, x_2, \sin(x_1), \exp(x_2)]$, etc.
- **Degree 2 Polynomial Basis:** We include all single features x_d, their squares x_d^2, and all products of two distinct features $x_d x_{d'}$.

Basis Function Expansion Examples

- **Univariate Functions:** We can set $\phi_k(\mathbf{x})$ to any univariate function of a single x_d to obtain mappings like $\phi(\mathbf{x}) = [x_1, x_2, \sin(x_1), \exp(x_2)]$, etc.
- **Degree 2 Polynomial Basis:** We include all single features x_d, their squares x_d^2, and all products of two distinct features $x_d x_{d'}$.
- **Degree B Polynomial Basis:** We include all single features x_d, and all unique products of between 2 and B features.

Basis Function Expansion Examples

- **Univariate Functions:** We can set $\phi_k(\mathbf{x})$ to any univariate function of a single x_d to obtain mappings like $\phi(\mathbf{x}) = [x_1, x_2, \sin(x_1), \exp(x_2)]$, etc.
- **Degree 2 Polynomial Basis:** We include all single features x_d, their squares x_d^2, and all products of two distinct features $x_d x_{d'}$.
- **Degree B Polynomial Basis:** We include all single features x_d, and all unique products of between 2 and B features.
- Don't forget that basis expansion still requires a bias term in the model!

Basis Function Expansion Examples

- **Univariate Functions:** We can set $\phi_k(\mathbf{x})$ to any univariate function of a single x_d to obtain mappings like $\phi(\mathbf{x}) = [x_1, x_2, \sin(x_1), \exp(x_2)]$, etc.
- **Degree 2 Polynomial Basis:** We include all single features x_d, their squares x_d^2, and all products of two distinct features $x_d x_{d'}$.
- **Degree *B* Polynomial Basis:** We include all single features x_d, and all unique products of between 2 and B features.
- Don't forget that basis expansion still requires a bias term in the model!
- A key question is how complex should we let the basis expanded model be?

Roadmap: non-linear functions in machine learning

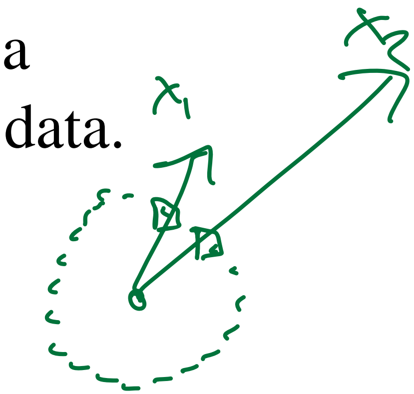
- (previous) Basis function expansions

$$\mathbf{x} \Rightarrow \phi(\mathbf{x})$$

- (today) Kernel methods: basis function implicit in example-to-example similarity functions
- (next) Methods that learn basis functions
  - neural networks (powerful, customizable, but fiddly)
  - ensemble decision trees (highly automatic, but not customizable)

Kernel functions

- A *kernel function* $K(\mathbf{x_1}, \mathbf{x_2}) : \mathcal{X} \times \mathcal{X} \to \cancel{\mathbb{R}}\ \mathbb{R}^+$ measures the similarity between two data instances. Similarity can be used for many forms of dat analysis; *kernel methods* in ML use a predefined kernel function to make predictions on new data. Examples:



- Linear kernel

$$K(x_1, x_2) = x_1^T x_2$$

- Cosine similarity

$$K(x_1, x_2) = \frac{x_1^T x_2}{\|x_1\|_2 \, \|x_2\|_2} = \left(\frac{x_1}{\|x_1\|_2}\right)^T \frac{x_2}{\|x_2\|_2}$$

- Polynomial kernel

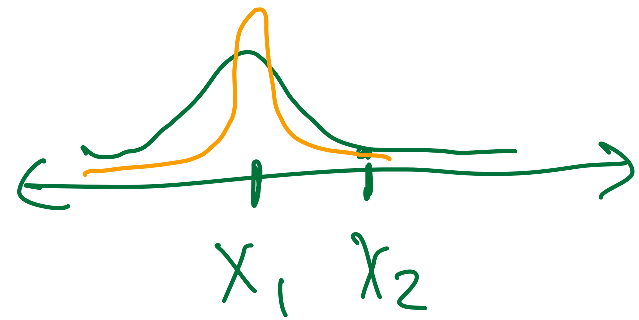
$$K(x_1, x_2) = (1 + \underbrace{\gamma}_{\text{Hyperparam}} x_1^T x_2)^B$$

Kernel functions examples, cont'd

- Radial Basis Function (RBF; a.k.a. Exponential a.k.a. Gaussian kernel)

$$K(x_1, x_2) = e^{-\gamma \|x_1 - x_2\|^2}$$

sensitivity hyperparam aka Bandwidth



- String kernels (more generally: kernels can be nice for crazy structured objects!)

ABCBAPBCDA . . .

BCBA a RRT . . .

Kernel methods

- Non-parametric models: do supervised learning and make predictions, but explicitly manipulate parameters (contrast linear/log reg)
  don't

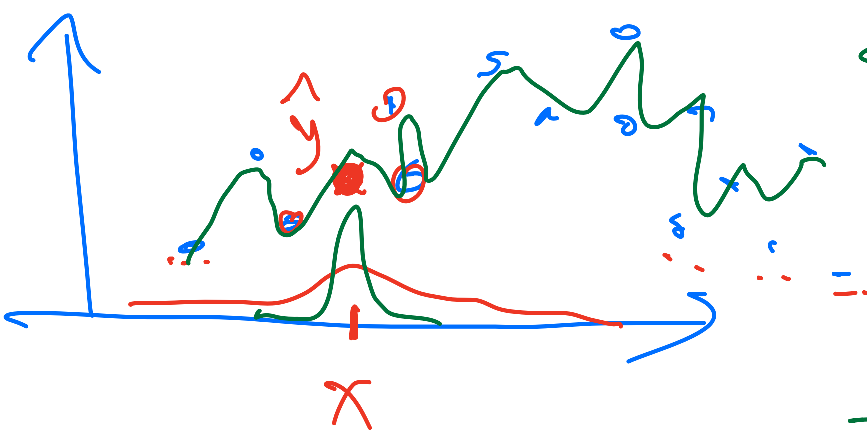
Kernel methods

- Non-parametric models: do supervised learning and make predictions, but explicitly manipulate parameters (contrast linear/log reg)
- Two classes of kernel methods: (1) Direct prediction (no training-time optimization; not in MLPP Ch. 14), (2) “Kernel trick” (with training-time optimization: MLPP Ch. 14). Non-param. statistics emphasizes (1) but machine learning emphasizes (2); be careful with terminology!

Kernel regression

With user-supplied kernel function K and labeled training set $\mathcal{D}^{\text{tr}}$, predict on new data point x based on the kernel-weighted-average of points in the training set:

$$\hat{y} = f(x; \mathcal{D}^{tr}) = \frac{\sum_{x',y' \in \mathcal{D}^{tr}} K(x, x')\, y'}{\sum_{x',y' \in \mathcal{D}^{tr}} K(x, x')}$$



- Bias variance tradeoff? → Kernel function hyperparams e.g. bandwidth γ: $e^{-\gamma \|x_1 - x_2\|^2}$
- Computational requirements? K fn. complexity

Duality between kernels and basis function representations

- Some kernel functions can be proven equivalent to an inner product of some basis function expansion of the two data points.

Duality between kernels and basis function representations

- Some kernel functions can be proven equivalent to an inner product of some basis function expansion of the two data points.
- Example: Polynomial kernel

$D = 2$

$$k(x, x') = (1 + x^T x')^2$$

$$= (1 + x_1 x_1' + x_2 x_2')^2$$

$$= 1 + 2 x_1 x_1' + 2 x_2 x_2' + (x_1 x_1')^2 + (x_2 x_2')^2 + 2 x_1 x_1' x_2 x_2'$$

$$= \phi(x)^T \phi(x')$$

where $\phi(x) = [1, \sqrt{2} x_1, \sqrt{2} x_2, x_1^2, x_2^2, \sqrt{2} x_1 x_2]$

“Kernel trick”

- Many parametric machine learning models can be “kernelized” by doing

"Kernel trick"

- Many parametric machine learning models can be "kernelized" by doing

"Kernel trick"

- Many parametric machine learning models can be "kernelized" by doing
- 1. Rewrite the model in terms of inner products between pairs of examples

"Kernel trick"

- Many parametric machine learning models can be "kernelized" by doing
- 1. Rewrite the model in terms of inner products between pairs of examples
  2. Swap in a kernel function for the inner product: equivalent to moving from linear kernel to the new kernel

$$\cdots\; x^T x' \;\cdots$$

$$\Downarrow$$

$$K(x, x')$$

"Kernel trick"

- Many parametric machine learning models can be "kernelized" by doing
- 1. Rewrite the model in terms of inner products between pairs of examples
  2. Swap in a kernel function for the inner product: equivalent to moving from linear kernel to the new kernel
- For better or worse, this is called "the kernel trick."

Kernelized linear regression

- Let’s apply to kernel trick to L2-regularized linear regression (“ridge regression”)

Kernelized linear regression

- Let's apply to kernel trick to L2-regularized linear regression ("ridge regression")
- Kind-of review: Ridge regression objective admits a closed-form solution similar to OLS.

$$J(w) = (Y - Xw)^T (Y - Xw) + \lambda \|w\|^2$$

$$\Rightarrow w = (X^T X + \lambda I_D)^{-1} X^T Y$$

Kernelized linear regression

- Let's apply to kernel trick to L2-regularized linear regression ("ridge regression")
- Kind-of review: Ridge regression objective admits a closed-form solution similar to OLS.
- New terminology: we'll now call this the "primal problem."

~~Solve for W~~

Kernelized linear regression: Dual problem

- Apply matrix inversion lemma(s) to rewrite solution:

Primal: $w = (X^T X + \lambda I)^{-1} X^T Y = (\sum_i x_i x_i^T + \lambda I)^{-1} X^T Y$

$$w = X^T \underbrace{(XX^T + \lambda I_N)}_{N \times N}^{-1} Y$$

$[k(x_i, x_j)]_{i,j \in 1..N}$

$$\alpha = (K + \lambda I_N)^{-1} Y$$

$$w = X^T \alpha = \sum_i \alpha_i \vec{x}_i$$

Kernelized linear regression: Dual problem

- Apply matrix inversion lemma(s) to rewrite solution:

$N \times N$ (inversion $= O(N^3)$)

- Define dual variables to rewrite as weighted sum of instances:

$$\alpha = (K + \lambda I)^{-1} y$$

$$w = X^T \alpha$$

$$\hat{f}(x) = w^T x = \sum_i \alpha_i x_i^T x = \sum_i \alpha_i K(x, x_i)$$

Computational cost

- Training time cost and storage cost: compare primal vs dual