ELEN 4903: Machine Learning
Week 5, Lecture 2, 2/14/2018

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

Feature expansions

Feature expansions

Feature expansions (also called **basis expansions**) are names given to a technique we've already discussed and made use of.

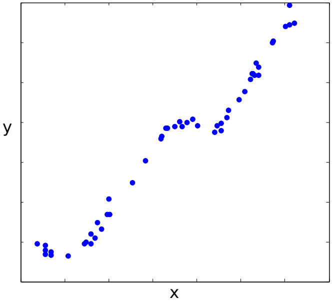
Problem: A linear model on the original feature space $x \in \mathbb{R}^d$ doesn't work.

Solution: Map the features to a higher dimensional space $\phi(x) \in \mathbb{R}^D$, where $D > d$, and do linear modeling there.

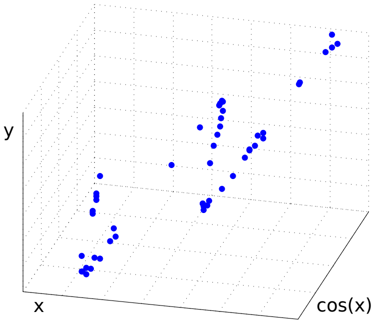
Examples

- For polynomial regression on $\mathbb{R}$, we let $\phi(x) = (x, x^2, \dots, x^p)$.
- For jump discontinuities, $\phi(x) = (x,\ \mathbb{1}\{x < a\})$.

Mapping example for regression



(a) Data for linear regression



(b) Same data mapped to higher dimension

High-dimensional maps can transform the data so output is linear in inputs.

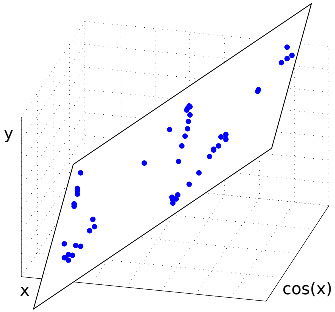
Left: Original $x \in \mathbb{R}$ and response y.

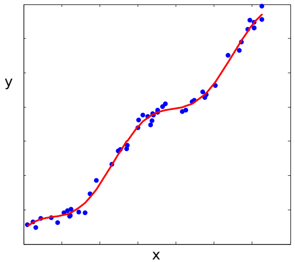
Right: x mapped to $\mathbb{R}^2$ using $\phi(x) = (x, \cos x)^T$.

Mapping example for regression

Using the mapping $\phi(x) = (x,\ \cos x)^T$, learn the linear regression model

$$\begin{aligned} y &\approx w_0 + \phi(x)^T w \\ &\approx w_0 + w_1 x + w_2 \cos x. \end{aligned}$$

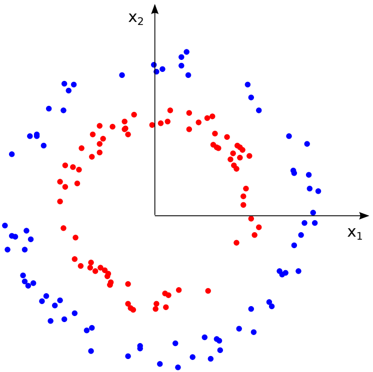




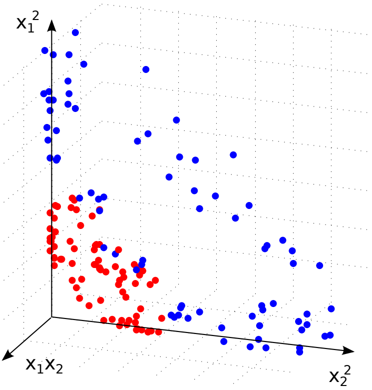
Left: Learn (w_0, w_1, w_2) to approximate data on the left with a plane.

Right: For each point x, map to $\phi(x)$ and predict y. Plot as a function of x.

Mapping example for classification



(e) Data for binary classification



(f) Same data mapped to higher dimension

High-dimensional maps can transform data so it becomes linearly separable.

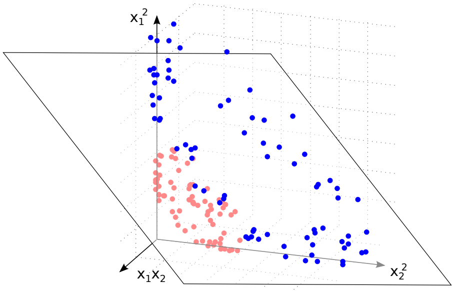
Left: Original data in $\mathbb{R}^2$.

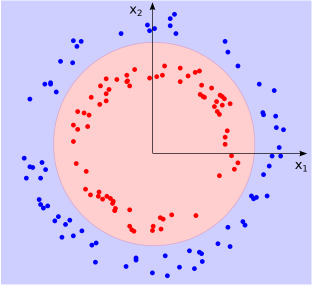
Right: Data mapped to $\mathbb{R}^3$ using $\phi(x) = (x_1^2,\, x_1 x_2,\, x_2^2)^T$.

Mapping example for classification

Using the mapping $\phi(x) = (x_1^2,\, x_1 x_2,\, x_2^2)^T$, learn a linear classifier

$$\begin{aligned} y &= \operatorname{sign}(w_0 + \phi(x)^T w) \\ &= \operatorname{sign}(w_0 + w_1 x_1^2 + w_2 x_1 x_2 + w_3 x_2^2). \end{aligned}$$





Left: Learn (w_0, w_1, w_2, w_3) to linearly separate classes with hyperplane.

Right: For each point x, map to $\phi(x)$ and classify. Color decision regions in $\mathbb{R}^2$.

What expansion should I use?

This is not obvious. The illustrations required knowledge about the data that we likely won't have (especially if it's in high dimensions).

One approach is to use the "kitchen sink": If you can think of it, then use it. Select the useful features with an ℓ_1 penalty

$$w_{\ell_1} = \arg\min_w \sum_{i=1}^n f(y_i, \phi(x_i), w) \ + \ \lambda \|w\|_1.$$

We know that this will find a sparse subset of the dimensions of $\phi(x)$ to use.

Often we only need to work with dot products $\phi(x_i)^T\phi(x_j) \equiv K(x_i, x_j)$. This is called a **kernel** and can produce some interesting results.

KERNELS

PERCEPTRON (SOME MOTIVATION)

Perceptron classifier

Let $x_i \in \mathbb{R}^{d+1}$ and $y_i \in \{-1, +1\}$ for $i = 1, \dots, n$ observations. We saw that the Perceptron constructs the hyperplane from data,

$$w = \textstyle\sum_{i \in \mathcal{M}} y_i x_i, \qquad \text{(assume } \eta = 1 \text{ and } \mathcal{M} \text{ has no duplicates)}$$

where $\mathcal{M}$ is the sequentially constructed set of misclassified examples.

Predicting new data

We also discussed how we can predict the label y_0 for a new observation x_0:

$$y_0 = \text{sign}(x_0^T w) = \text{sign}\left(\textstyle\sum_{i \in \mathcal{M}} y_i x_0^T x_i\right)$$

We've taken feature expansions for granted, but we can explicitly write it as

$$y_0 = \text{sign}(\phi(x_0)^T w) = \text{sign}\left(\textstyle\sum_{i \in \mathcal{M}} y_i \phi(x_0)^T \phi(x_i)\right)$$

We can represent the decision using dot products between data points.

KERNELS

Kernel definition

A kernel $K(\cdot,\cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a symmetric function defined as follows:

Definition: If for any n points $x_1, \dots, x_n \in \mathbb{R}^d$, the $n \times n$ matrix K, where $K_{ij} = K(x_i, x_j)$, is *positive semidefinite*, then $K(\cdot,\cdot)$ is a "kernel."

Intuitively, this means K satisfies the properties of a covariance matrix.

Mercer's theorem

If the function $K(\cdot,\cdot)$ satisfies the above properties, then there exists a mapping $\phi : \mathbb{R}^d \to \mathbb{R}^D$ (D can equal ∞) such that

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j).$$

If we first define $\phi(\cdot)$ and then K, this is obvious. However, sometimes we first define $K(\cdot,\cdot)$ and avoid ever using $\phi(\cdot)$.

Gaussian kernel (radial basis function)

The most popular kernel is the Gaussian kernel, also called the radial basis function (RBF),

$$K(x, x') = a \exp\left\{-\frac{1}{b}\|x - x'\|^2\right\}.$$

- This is a good, general-purpose kernel that usually works well.
- It takes into account proximity in $\mathbb{R}^d$. Things close together in space have larger value (as defined by kernel width b).

In this case, the the mapping $\phi(x)$ that produces the RBF kernel is *infinite dimensional* (it's a continuous function instead of a vector). Therefore

$$K(x, x') = \int \phi_t(x)\phi_t(x')\, dt.$$

- $\phi_t(x)$ can be thought of as a function of t with parameter x that also has a Gaussian form.

Another kernel

Map : $\phi(x) = (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \ldots, \sqrt{2}x_i x_j, \ldots)$

Kernel : $\phi(x)^T \phi(x') = K(x, x') = (1 + x^T x')^2$

In fact, we can show $K(x, x') = (1 + x^T x')^c$, for $c > 0$ is a kernel as well.

Kernel arithmetic

Certain functions of kernels can produce new kernels.

Let K_1 and K_2 be any two kernels, then constructing K in the following ways produces a new kernel (among many other ways):

$$\begin{aligned} K(x, x') &= K_1(x, x') K_2(x, x') \\ K(x, x') &= K_1(x, x') + K_2(x, x') \\ K(x, x') &= \exp\{K_1(x, x')\} \end{aligned}$$

Returning to the Perceptron

We write the feature-expanded decision as

$$\begin{aligned} y_0 &= \text{sign}\left(\textstyle\sum_{i\in\mathcal{M}} y_i \phi(x_0)^T \phi(x_i)\right) \\ &= \text{sign}\left(\textstyle\sum_{i\in\mathcal{M}} y_i K(x_0, x_i)\right) \end{aligned}$$

We can pick the kernel we want to use. Let's pick the RBF (set $a = 1$). Then

$$y_0 = \text{sign}\left(\textstyle\sum_{i\in\mathcal{M}} y_i \, e^{-\frac{1}{b}\|x_0 - x_i\|^2}\right)$$

Notice that we never actually need to calculate $\phi(x)$.

What is this doing?

- Notice $0 < K(x_0, x_i) \leq 1$, with bigger values when x_0 is closer to x_i.
- This is like a "soft voting" among the data picked by Perceptron.

Learning the kernelized Perceptron

Recall: Given a current vector $w^{(t)} = \sum_{i \in \mathcal{M}_t} y_i x_i$, we update it as follows,

1. Find a new x' such that $y' \neq \text{sign}(x'^T w^{(t)})$
2. Add the index of x' to $\mathcal{M}$ and set $w^{(t+1)} = \sum_{i \in \mathcal{M}_{t+1}} y_i x_i$

Again we only need dot products, meaning these steps are equivalent to

1. Find a new x' such that $y' \neq \text{sign}(\sum_{i \in \mathcal{M}_t} y_i K(x', x_i))$
2. Add the index of x' to $\mathcal{M}$ *but don't bother calculating* $w^{(t+1)}$

The trick is to realize that we never need to work with $\phi(x)$.

- We don't need $\phi(x)$ to do Step 1 above.
- We don't need $\phi(x)$ to classify new data (previous slide).
- We only ever need to calculate $K(x, x')$ between two points.

KERNEL k-NN

An extension

We can generalize kernelized Perceptron to *soft* k-NN with a simple change. Instead of summing over misclassified data $\mathcal{M}$, sum over *all* the data:

$$y_0 = \text{sign}\left(\sum_{i=1}^{n} y_i \, \text{e}^{-\frac{1}{b}\|x_0 - x_i\|^2}\right).$$

Next, notice the *decision* doesn't change if we divide by a positive constant.

$$\text{Let}: \; Z = \sum_{j=1}^{n} \text{e}^{-\frac{1}{b}\|x_0 - x_j\|^2}$$

$$\text{Construct}: \; \text{Vector } p(x_0), \text{ where } p_i(x_0) = \frac{1}{Z}\text{e}^{-\frac{1}{b}\|x_0 - x_i\|^2}$$

$$\text{Declare}: \; y_0 = \text{sign}\Big(\sum_{i=1}^{n} y_i p_i(x_0)\Big)$$

- We let all data vote for the label based on a "confidence score" $p(x_0)$.
- Set b so that most $p_i(x_0) \approx 0$ to only focus on neighborhood around x_0.

Nadaraya-Watson model

The developments are almost limitless.

Here's a regression example almost identical to the kernelized k-NN:

Before: $y \in \{-1, +1\}$
Now: $y \in \mathbb{R}$

Using the RBF kernel, for a new (x_0, y_0) predict

$$y_0 = \sum_{i=1}^{n} y_i \frac{K(x_0, x_i)}{\sum_{j=1}^{n} K(x_0, x_j)}.$$

What is this doing?
We're taking a locally weighted average of all y_i for which x_i is close to x_0 (as decided by the kernel width). *Gaussian processes* are another option. . .

GAUSSIAN PROCESSES (FOR REGRESSION)

Kernelized Bayesian linear regression

Regression setup: For n observations, with response vector $y \in \mathbb{R}^n$ and their feature matrix X, we define the likelihood and prior

$$y \sim N(Xw, \sigma^2 I), \quad w \sim N(0, \lambda^{-1} I).$$

Marginalizing: What if we integrate out w? We can solve this,

$$p(y|X) = \int p(y|X,w)p(w)dw \; = \; N(0, \sigma^2 I + \lambda^{-1} XX^T).$$

Kernelization: Notice that $(XX^T)_{ij} = x_i^T x_j$. Replace each x with $\phi(x)$ after which we can say $[\phi(X)\phi(X)^T]_{ij} = K(x_i, x_j)$. We can define K directly, so

$$p(y|X) = \int p(y|X,w)p(w)dw = N(0, \sigma^2 I + \lambda^{-1} K).$$

This is called a *Gaussian process*. We never use w or $\phi(x)$, but just $K(x_i, x_j)$.

Definition

- Let $f(x) \in \mathbb{R}$ and $x \in \mathcal{X}$.
- Define the *kernel* $K(x, x')$ between two points x and x'.
- Then $f(x)$ is a *Gaussian process* and $y(x)$ the noise-added process if for n observed pairs $(x_1, y_1), \ldots, (x_n, y_n)$, where $x \in \mathcal{X}$ and $y \in \mathbb{R}$,

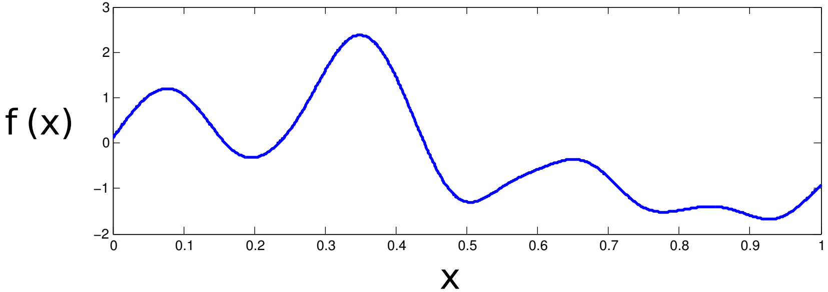
$$y \,|\, f \sim N(f, \sigma^2 I), \quad f \sim N(0, K) \quad \Longleftrightarrow \quad y \sim N(0, \sigma^2 I + K)$$

where $y = (y_1, \ldots, y_n)^T$ and K is $n \times n$ with $K_{ij} = K(x_i, x_j)$.

Comments:

- We assume $\lambda = 1$ to reduce notation.
- Typical breakdown: $f(x)$ is the GP and $y(x)$ equals $f(x)$ plus i.i.d. noise.
- The kernel is what keeps this from being "just a Gaussian."

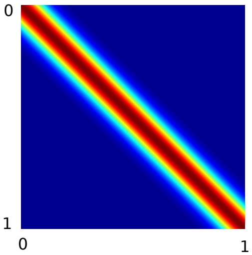
GAUSSIAN PROCESSES



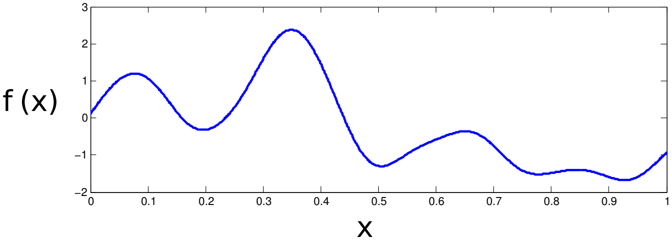
Above: A Gaussian process $f(x)$ ($x \in \mathbb{R}$) using

$$K(x_i, x_j) = \exp\left\{-\frac{\|x_i - x_j\|^2}{b}\right\}.$$

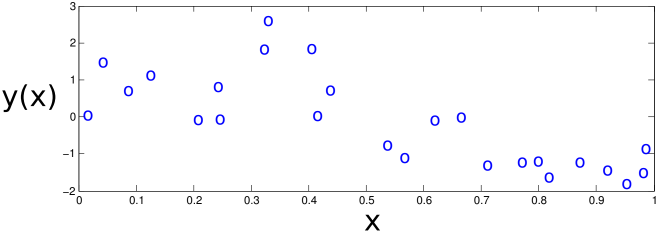
Right: The covariance of $f(x)$ defined by K.



GAUSSIAN PROCESSES



Top: Unobserved underlying function,
Bottom: Noisy observed data sampled from this function



PREDICTIONS WITH GAUSSIAN VECTORS

Bayesian linear regression

Imagine we have n observation pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ and want to predict y_0 given x_0. Integrating out w and setting $\lambda = 1$, the joint distribution is

$$\begin{bmatrix} y_0 \\ y \end{bmatrix} \sim \text{Normal}\left(\mathbf{0}\,,\ \sigma^2 I + \begin{bmatrix} x_0^T x_0 & (Xx_0)^T \\ Xx_0 & XX^T \end{bmatrix}\right)$$

We want to predict y_0 given $\mathcal{D}$ and x_0. Calculations can show that

$$\begin{aligned} y_0 | \mathcal{D}, x_0 &\sim \text{Normal}(\mu_0, \sigma_0^2) \\ \mu_0 &= (Xx_0)^T (\sigma^2 I + XX^T)^{-1} y \\ \sigma_0^2 &= \sigma^2 + x_0^T x_0 - (Xx_0)^T (\sigma^2 I + XX^T)^{-1} (Xx_0) \end{aligned}$$

Since the Gaussian process is only ever evaluated at a finite set of points, this is useful. Notice all the dot products between x_i. These can become kernels.

Predictive distribution of $y(x)$

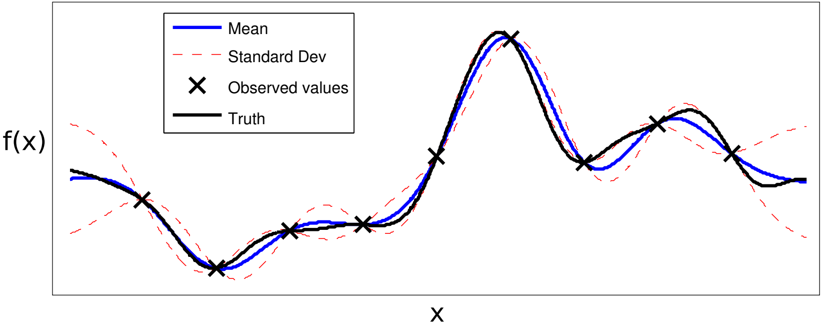
Given measured data $\mathcal{D}_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, the distribution of $y(x)$ can be calculated at any *new* x to make predictions.

Let $K(x, \mathcal{D}_n) = [K(x, x_1), \ldots, K(x, x_n)]$ and K_n be the $n \times n$ kernel matrix restricted points in $\mathcal{D}_n$. Then we can show

$$\begin{aligned} y(x)|\mathcal{D}_n &\sim N\left(\mu(x), \Sigma(x)\right), \\ \mu(x) &= K(x, \mathcal{D}_n)(\sigma^2 I + K_n)^{-1} y, \\ \Sigma(x) &= \sigma^2 + K(x, x) - K(x, \mathcal{D}_n)(\sigma^2 I + K_n)^{-1} K(x, \mathcal{D}_n)^T \end{aligned}$$

For the posterior of $f(x)$ instead of $y(x)$, just remove σ^2.

GAUSSIAN PROCESSES POSTERIOR



What does the posterior distribution of $f(x)$ look like?

- We have data marked by an $\times$.
- These values pin down the function $f(x)$ nearby
- We get a mean and variance for every possible x from previous slide.
- The distribution on $y(x)$ adds variance σ^2 (*very* small above) point-wise.