ColumbiaX: Machine Learning
Lecture 9

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

LOGISTIC REGRESSION

Linear classifiers

Given: Data $(x_1, y_1), \ldots, (x_n, y_n)$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$

A **linear classifier** takes a vector $w \in \mathbb{R}^d$ and scalar $w_0 \in \mathbb{R}$ and predicts

$$y_i = f(x_i; w, w_0) = \text{sign}(x_i^T w + w_0).$$

We discussed two methods last time:

- Least squares: Sensitive to outliers
- Perceptron: Convergence issues, assumes linear separability

Can we combine the separating hyperplane idea with probability to fix this?

Linear discriminant analysis

We saw an example of a linear classification rule using a Bayes classifier.

For the model $y \sim \text{Bern}(\pi)$ and $x \,|\, y \sim N(\mu_y, \Sigma)$, declare $y = 1$ given x if

$$\ln \frac{p(x|y=1)p(y=1)}{p(x|y=0)p(y=0)} > 0\,.$$

In this case, the *log odds* is equal to

$$\begin{aligned}\ln \frac{p(x|y=1)p(y=1)}{p(x|y=0)p(y=0)} \quad &= \quad \underbrace{\ln \frac{\pi_1}{\pi_0} - \frac{1}{2}(\mu_1 + \mu_0)^T \Sigma^{-1}(\mu_1 - \mu_0)}_{\text{a constant } w_0} \\ &\quad + x^T \underbrace{\Sigma^{-1}(\mu_1 - \mu_0)}_{\text{a vector } w}\end{aligned}$$

Log odds and Bayes classification

Original formulation

Recall that originally we wanted to declare $y = 1$ given x if

$$\ln \frac{p(y=1|x)}{p(y=0|x)} > 0$$

We didn't have a way to define $p(y|x)$, so we used Bayes rule:

- Use $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$ and let the $p(x)$ cancel each other in the fraction
- Define $p(y)$ to be a Bernoulli distribution (coin flip distribution)
- Define $p(x|y)$ however we want (e.g., a single Gaussian)

Now, we want to directly define $p(y|x)$. We'll use the log odds to do this.

LOG ODDS AND BAYES CLASSIFICATION

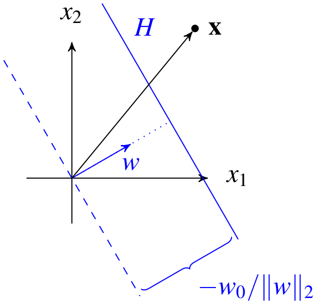
Log odds and hyperplanes

Classifying x based on the log odds

$$L = \ln \frac{p(y=+1|x)}{p(y=-1|x)},$$

we notice that

1. $L \gg 0$: more confident $y = +1$,
2. $L \ll 0$: more confident $y = -1$,
3. $L = 0$: can go either way



The linear function $x^T w + w_0$ captures these three objectives:

- The distance of x to a hyperplane H defined by (w, w_0) is $|\frac{x^T w}{\|w\|_2} + \frac{w_0}{\|w\|_2}|$.
- The sign of the function captures which side x is on.
- As x moves away/towards H, we become more/less confident.

Log odds and hyperplanes

Logistic link function

We can directly plug in the hyperplane representation for the log odds:

$$\ln \frac{p(y=+1|x)}{p(y=-1|x)} = x^T w + w_0$$

Question: What is different from the previous Bayes classifier?

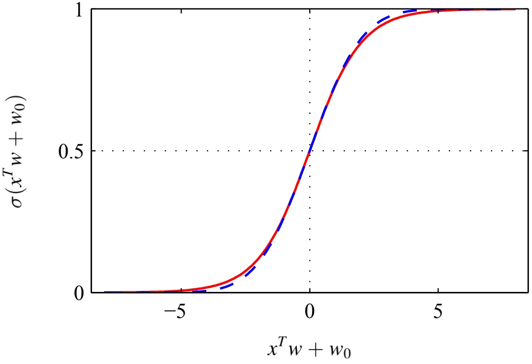
Answer: There was a formula for calculating w and w_0 based on the prior model and data x. Now, we put no restrictions on these values.

Setting $p(y=-1|x) = 1 - p(y=+1|x)$, solve for $p(y=+1|x)$ to find

$$p(y=+1|x) = \frac{\exp\{x^T w + w_0\}}{1+\exp\{x^T w + w_0\}} = \sigma(x^T w + w_0).$$

- This is called the *sigmoid function*.
- We have chosen $x^T w + w_0$ as the *link function* for the log odds.

Logistic sigmoid function



- Red line: Sigmoid function $\sigma(x^T w + w_0)$, which maps x to $p(y = +1|x)$.
- The function $\sigma(\cdot)$ captures our desire to be more confident as we move away from the separating hyperplane, defined by the x-axis.
- (Blue dashed line: On a later slide.)

Logistic regression

As with regression, absorb the offset: $w \leftarrow \begin{bmatrix} w_0 \\ w \end{bmatrix}$ and $x \leftarrow \begin{bmatrix} 1 \\ x \end{bmatrix}$.

Definition

Let $(x_1, y_1), \ldots, (x_n, y_n)$ be a set of binary labeled data with $y \in \{-1, +1\}$. *Logistic regression* models each y_i as independently generated, with

$$P(y_i = +1 | x_i, w) = \sigma(x_i^T w), \quad \sigma(x_i; w) = \frac{e^{x_i^T w}}{1 + e^{x_i^T w}}.$$

Discriminative vs Generative classifiers

- This is a *discriminative* classifier because x is not directly modeled.
- Bayes classifiers are known as *generative* because x is modeled.

Discriminative: $p(y|x)$ Generative: $p(x|y)p(y)$.

Logistic regression likelihood

Data likelihood

Define $\sigma_i(w) = \sigma(x_i^T w)$. The joint likelihood of $y_1, \dots, y_n$ is

$$\begin{aligned} p(y_1, \dots, y_n | x_1, \dots, x_n, w) &= \prod_{i=1}^{n} p(y_i | x_i, w) \\ &= \prod_{i=1}^{n} \sigma_i(w)^{\mathbb{1}(y_i=+1)} \left(1 - \sigma_i(w)\right)^{\mathbb{1}(y_i=-1)} \end{aligned}$$

- Notice that each x_i modifies the probability of success for its y_i.
- Predicting new data is the same:
  - If $x^T w > 0$, then $\sigma(x^T w) > 1/2$ and predict $y = +1$, and vice versa.
  - We now get a confidence in our prediction via the probability $\sigma(x^T w)$.

Logistic regression and maximum likelihood

More notation changes

Use the following fact to condense the notation:

$$\underbrace{\frac{e^{y_i x_i^T w}}{1+e^{y_i x_i^T w}}}_{\sigma_i(y_i \cdot w)} = \Bigg(\underbrace{\frac{e^{x_i^T w}}{1+e^{x_i^T w}}}_{\sigma_i(w)}\Bigg)^{\mathbb{1}(y_i=+1)} \Bigg(\underbrace{1 - \frac{e^{x_i^T w}}{1+e^{x_i^T w}}}_{1-\sigma_i(w)}\Bigg)^{\mathbb{1}(y_i=-1)}$$

therefore, the data likelihood can be written compactly as

$$p(y_1, \ldots, y_n | x_1, \ldots, x_n, w) = \prod_{i=1}^{n} \sigma_i(y_i \cdot w)$$

We want to maximize this over w.

Logistic regression and maximum likelihood

Maximum likelihood

The maximum likelihood solution for w can be written

$$\begin{aligned} w_{\text{ML}} &= \arg\max_{w} \sum_{i=1}^{n} \ln \sigma_i(y_i \cdot w) \\ &= \arg\max_{w} \mathcal{L} \end{aligned}$$

As with the Perceptron, we can't directly set $\nabla_w \mathcal{L} = 0$, so we need an iterative algorithm. At step t, we can update

$$w^{(t+1)} = w^{(t)} + \eta \nabla_w \mathcal{L}, \qquad \nabla_w \mathcal{L} = \sum_{i=1}^{n} \left(1 - \sigma_i(y_i \cdot w)\right) y_i x_i.$$

We will see that this results in an algorithm similar to the Perceptron.

Logistic regression algorithm (steepest ascent)

```
Input: Training data $(x_1, y_i), \ldots, (x_n, y_n)$ and step size $\eta > 0$
  1. Set $w^{(1)} = \vec{0}$
  2. For step $t = 1, 2, \ldots$ do
       • Update $w^{(t+1)} = w^{(t)} + \eta \sum_{i=1}^{n} (1 - \sigma_i(y_i \cdot w))\, y_i x_i$
```

Perceptron: Search for misclassified (x_i, y_i), update $w^{(t+1)} = w^{(t)} + \eta y_i x_i$.

Logistic regression: Something similar except we sum over all data.

- Recall that $\sigma_i(y_i \cdot w)$ picks out the probability assigned to the observed y_i.
- Therefore $1 - \sigma_i(y_i \cdot w)$ is the probability assigned to the *wrong* value.
- Perceptron is "all-or-nothing." Either it's correctly or incorrectly classified.
- Logistic regression has a probability "fudge-factor."

BAYESIAN LOGISTIC REGRESSION

Problem: If a hyperplane can separate all training data, then $\|w_{\text{ML}}\|_2 \to \infty$. This drives $\sigma_i(y_i \cdot w) \to 1$ for each (x_i, y_i).

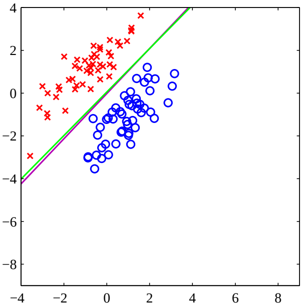
Even for nearly separable data it might get a few very wrong in order to be more confident about the rest. This is a case of "over-fitting."

A solution: Regularize w with $\lambda w^T w$:

$$w_{\text{MAP}} = \arg\max_w \sum_{i=1}^n \ln \sigma_i(y_i \cdot w) - \lambda w^T w$$

We've seen how this corresponds to a Gaussian prior distribution on w.

How about the posterior $p(w|x, y)$?



LAPLACE APPROXIMATION

Bayesian logistic regression

Posterior calculation

Define the prior distribution on w to be $w \sim N(0, \lambda^{-1}I)$. The posterior is

$$p(w|x,y) = \frac{p(w)\prod_{i=1}^{n} \sigma_i(y_i \cdot w)}{\int p(w)\prod_{i=1}^{n} \sigma_i(y_i \cdot w)\, dw}$$

This is not a "standard" distribution and we can't calculate the denominator.

Therefore we can't actually say what $p(w|x,y)$ is.

Can we approximate $p(w|x,y)$?

Laplace approximation

One strategy

Pick a distribution to approximate $p(w|x,y)$. We will say

$$p(w|x,y) \approx \text{Normal}(\mu, \Sigma).$$

Now we need a method for setting μ and Σ.

Laplace approximations

Using a condensed notation, notice from Bayes rule that

$$p(w|x,y) = \frac{e^{\ln p(y,w|x)}}{\int e^{\ln p(y,w|x)} dw}.$$

We will approximate $\ln p(y,w|x)$ in the numerator and denominator.

Laplace approximation

Let's define $f(w) = \ln p(y, w|x)$.

Taylor expansions

We can approximate $f(w)$ with a **second order Taylor expansion**.

Recall that $w \in \mathbb{R}^{d+1}$. For any point $z \in \mathbb{R}^{d+1}$,

$$f(w) \ \approx \ f(z) + (w-z)^T \nabla f(z) + \frac{1}{2}(w-z)^T \left(\nabla^2 f(z)\right)(w-z)$$

The notation $\nabla f(z)$ is short for $\nabla_w f(w)|_z$, and similarly for the matrix of second derivatives. We just need to pick z.

The Laplace approximation defines $z = w_{\text{MAP}}$.

Laplace approximation (solving)

Recall $f(w) = \ln p(y, w|x)$ and $z = w_{\text{MAP}}$. From Bayes rule and the Laplace approximation we now have

$$
\begin{aligned}
p(w|x,y) &= \frac{\mathrm{e}^{f(w)}}{\int \mathrm{e}^{f(w)} dw} \\
&\approx \frac{\mathrm{e}^{f(z)+(w-z)^T\nabla f(z)+\frac{1}{2}(w-z)^T\left(\nabla^2 f(z)\right)(w-z)}}{\int \mathrm{e}^{f(z)+(w-z)^T\nabla f(z)+\frac{1}{2}(w-z)^T(\nabla^2 f(z))(w-z)} dw}
\end{aligned}
$$

This can be simplified in two ways,

1. The term $\mathrm{e}^{f(w_{\text{MAP}})}$ in the numerator and denominator can be viewed as a constant since it doesn't vary in w. It therefore cancels out.
2. By definition of how we find w_{MAP}, the vector $\nabla_w \ln p(y, w|x)|w_{\text{MAP}} = 0$.

Laplace approximation (solving)

We're therefore left with the approximation

$$p(w|x,y) \quad \approx \quad \frac{e^{-\frac{1}{2}(w-w_{\text{MAP}})^T\left(-\nabla^2 \ln p(y,w_{\text{MAP}}|x)\right)(w-w_{\text{MAP}})}}{\int e^{-\frac{1}{2}(w-w_{\text{MAP}})^T(-\nabla^2 \ln p(y,w_{\text{MAP}}|x))(w-w_{\text{MAP}})}dw}$$

The solution comes by observing that this is a multivariate normal,

$$p(w|x,y) \approx \text{Normal}(\mu, \Sigma),$$

where

$$\mu = w_{\text{MAP}}, \quad \Sigma = \left(-\nabla^2 \ln p(y, w_{\text{MAP}}|x)\right)^{-1}$$

We can take the second derivative (Hessian) of the log joint likelihood to find

$$\nabla^2 \ln p(y, w_{\text{MAP}}|x) = -\lambda I - \sum_{i=1}^{n} \sigma(y_i \cdot x_i^T w_{\text{MAP}}) \left(1 - \sigma(y_i \cdot x_i^T w_{\text{MAP}})\right) x_i x_i^T$$

Bayesian logistic regression

Laplace approximation for logistic regression

Given labeled data $(x_1, y_1), \ldots, (x_n, y_n)$ and the model

$$P(y_i|x_i, w) = \sigma(y_i x_i^T w), \quad w \sim N(0, \lambda^{-1} I), \qquad \sigma(y_i x_i^T w) = \frac{e^{y_i x_i^T w}}{1 + e^{y_i x_i^T w}}$$

1. Find: $w_{\text{MAP}} = \arg\max_w \sum_{i=1}^n \ln \sigma(y_i x_i^T w_{\text{MAP}}) - \frac{\lambda}{2} w^T w$
2. Set: $-\Sigma^{-1} = -\lambda I - \sum_{i=1}^n \sigma(y_i x_i^T w_{\text{MAP}}) \left(1 - \sigma(y_i x_i^T w_{\text{MAP}})\right) x_i x_i^T$
3. Approximate: $p(w|x, y) = N\left(w_{\text{MAP}}, \Sigma\right)$.

ColumbiaX: Machine Learning
Lecture 10

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.

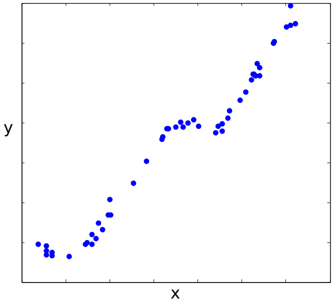
Problem: A linear model on the original feature space $x \in \mathbb{R}^d$ doesn't work.

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

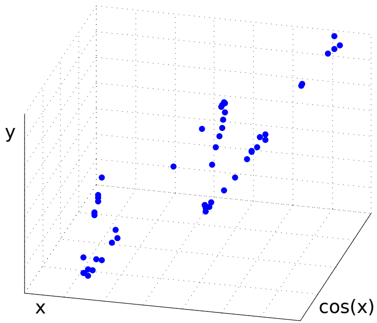
Examples

- 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.

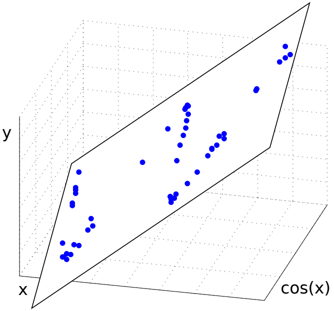
Left: Original $x \in \mathbb{R}$ and response y.

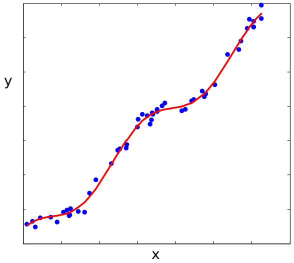
Right: x mapped to $\mathbb{R}^2$ using $\phi(x) = (x, \cos x)^T$.

MAPPING EXAMPLE FOR REGRESSION

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

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

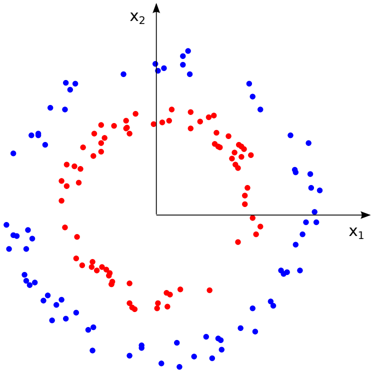




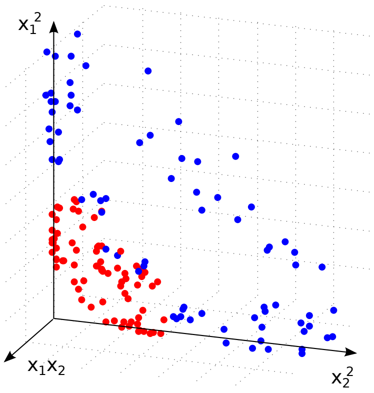
Left: Learn (w_0, w_1, w_2) to approximate data on the left with a plane.

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.

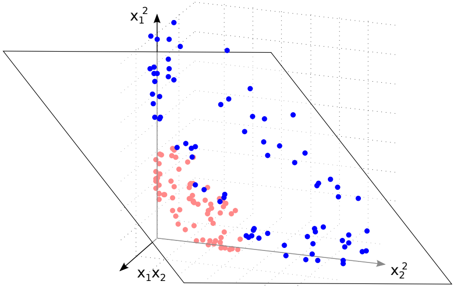
Left: Original data in $\mathbb{R}^2$.

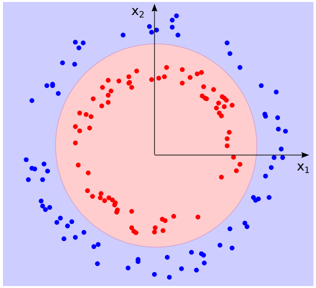
Right: Data mapped to $\mathbb{R}^3$ using $\phi(x) = (x_1^2, x_1 x_2, x_2^2)^T$.

Mapping example for classification

Using the mapping $\phi(x) = (x_1^2,\, x_1x_2,\, x_2^2)^T$, learn a linear classifier

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





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 $\mathbb{R}^2$.

FEATURE EXPANSIONS AND DOT PRODUCTS

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 $i = 1, \ldots, n$ observations. We saw that the Perceptron constructs the hyperplane from data,

$$w = \sum_{i \in \mathcal{M}} y_i x_i,$$

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:

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

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

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

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: For any set of n data 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*.

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 $\phi : \mathbb{R}^d \to \mathbb{R}^D$ 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)

By far 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

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

- $K(x, x')$ is like a Gaussian on x with x' as the mean (or vice versa).
- $\phi_t(x)$ can be thought of as a function of t with parameter x.

KERNELS

Another kernel

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

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

In fact, we can show: $K(x,x') = (1 + x^T x')^b$, for $b > 0$ 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):

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

KERNELIZED PERCEPTRON

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.

KERNELIZED 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:

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

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

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

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

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

- 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.

Kernel regression

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

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 $K(x_i, x_j)$.

GAUSSIAN PROCESSES

Definition

- Let $f(x) \in \mathbb{R}$ and $x \in \mathbb{R}^d$.
- 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

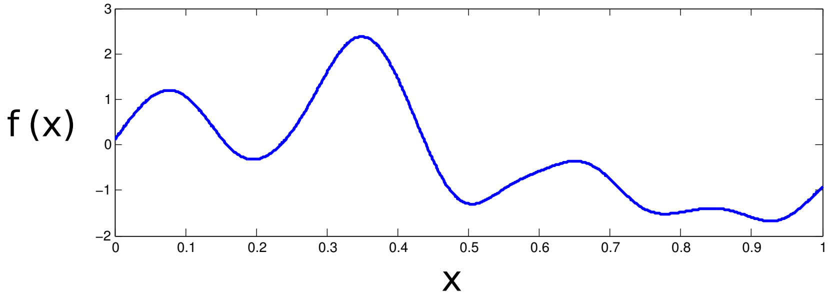
$$y \,|\, f \sim N(f, \sigma^2 I), \quad f \sim N(0, K) \quad \Leftrightarrow \quad y \sim N(0, \sigma^2 I + K)$$

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

Comments:

- ▶ We combined the previous λ^{-1} with K (for notation only).
- ▶ 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.”

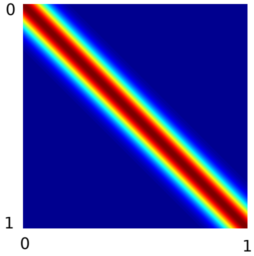
GAUSSIAN PROCESSES



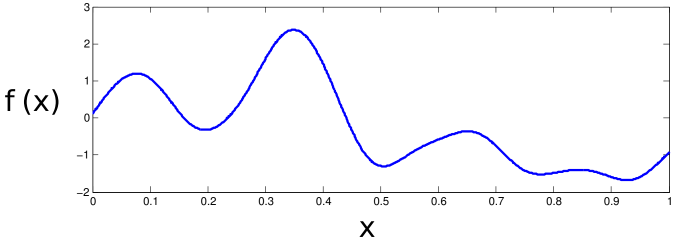
Above: A Gaussian process $f(x)$ generated using

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

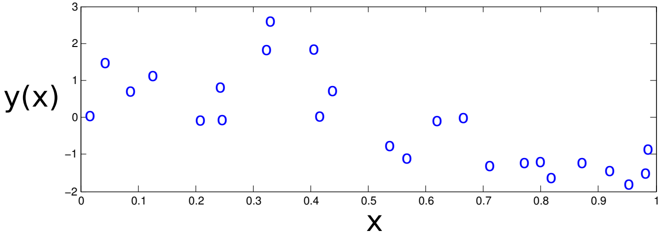
Right: The covariance of $f(x)$ defined by K.



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, the joint distribution is

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

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

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

The since the infinite Gaussian process is only evaluated at a finite set of points, we can use this fact.

PREDICTIONS WITH GAUSSIAN PROCESSES

Predictive distribution of $y(x)$

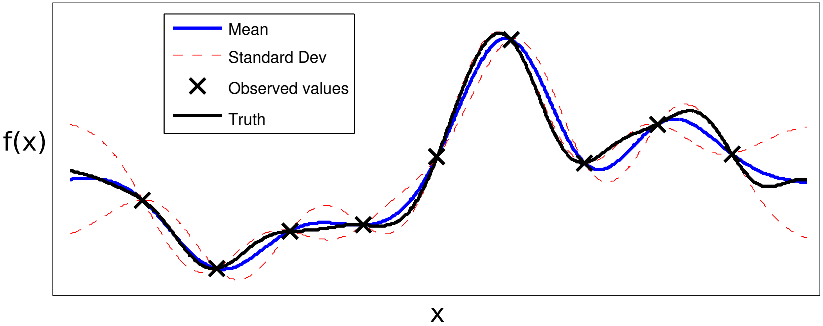
Given measured data $\mathcal{D}_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, the distribution of $y(x)$ can be calculated at any *new* x to make predictions.

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

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

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 a previous slide.
- The distribution on $y(x)$ adds variance σ^2 (*very* small above) point-wise.