Beyond plugin methods

Plugin methods can be useful in practice, but they are also somewhat limited

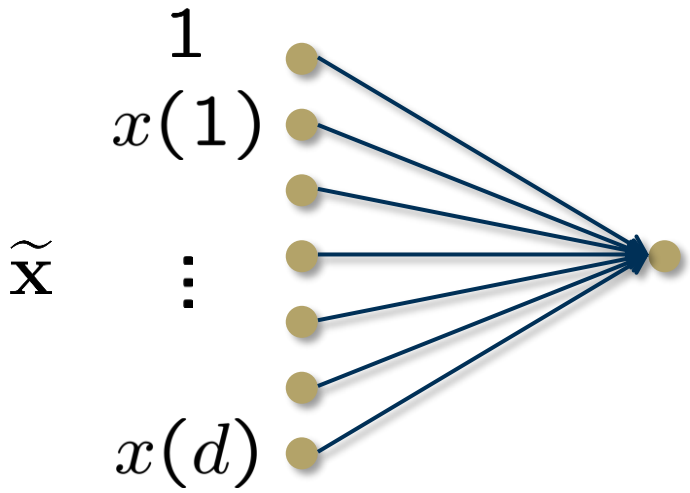
- There are always distributions where our assumptions are violated
- If our assumptions are wrong, the output is totally unpredictable
- Can be hard to verify whether our assumptions are right
- Require solving a more difficult problem as an intermediate step

For most of the remainder of this course will focus on (nonparametric) methods that avoid making such strong assumptions about the (unknown) process generating the data

Nonparametric linear classifiers

Suppose $K = 2$ and that $Y \in \{-1, +1\}$

By setting $\tilde{\mathbf{x}} = [1, x(1), \ldots, x(d)]^T$, we can reduce any linear classifier to comparing $\boldsymbol{\theta}^T \tilde{\mathbf{x}}$ to a threshold for some $\boldsymbol{\theta}$



$$y = \begin{cases} +1 & \text{if } \boldsymbol{\theta}^T \tilde{\mathbf{x}} > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$y = \text{sign}(\boldsymbol{\theta}^T \tilde{\mathbf{x}})$$

Single layer neural network

How to learn $\boldsymbol{\theta}$?

Perceptron Learning Algorithm (PLA)

Frank Rosenblatt (1957)

Given

- training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$
- a guess for $\boldsymbol{\theta}^0$

Pick any i (i.e., pick any of our observations), and update according to

$$\boldsymbol{\theta}^{j+1} = \begin{cases} \boldsymbol{\theta}^j + y_i \widetilde{\mathbf{x}}_i & \text{if } y_i \neq \text{sign}((\boldsymbol{\theta}^j)^T \widetilde{\mathbf{x}}_i) \\ \boldsymbol{\theta}^j & \text{otherwise} \end{cases}$$

Iterate

Simply repeat this process

$$\boldsymbol{\theta}^{j+1} = \begin{cases} \boldsymbol{\theta}^j + y_i \widetilde{\mathbf{x}}_i & \text{if } y_i \neq \text{sign}((\boldsymbol{\theta}^j)^T \widetilde{\mathbf{x}}_i) \\ \boldsymbol{\theta}^j & \text{otherwise} \end{cases}$$

That's it!

Why might this work? $(\boldsymbol{\theta}^{j+1})^T \widetilde{\mathbf{x}}_i = (\boldsymbol{\theta}^j + y_i \widetilde{\mathbf{x}}_i)^T \widetilde{\mathbf{x}}_i$

$= (\boldsymbol{\theta}^j)^T \widetilde{\mathbf{x}}_i + y_i \widetilde{\mathbf{x}}_i^T \widetilde{\mathbf{x}}_i$ ← pushes in the right direction

If $y_i = 1$ and $\text{sign}((\boldsymbol{\theta}^j)^T \widetilde{\mathbf{x}}_i) = -1$, then the update ***increases*** $(\boldsymbol{\theta}^{j+1})^T \widetilde{\mathbf{x}}_i$

If $y_i = -1$ and $\text{sign}((\boldsymbol{\theta}^j)^T \widetilde{\mathbf{x}}_i) = 1$, then the update ***decreases*** $(\boldsymbol{\theta}^{j+1})^T \widetilde{\mathbf{x}}_i$

Another perspective

The core iteration of the PLA is

$$\boldsymbol{\theta}^{j+1} = \begin{cases} \boldsymbol{\theta}^j + y_i \widetilde{\mathbf{x}}_i & \text{if } y_i \neq \text{sign}((\boldsymbol{\theta}^j)^T \widetilde{\mathbf{x}}_i) \\ \boldsymbol{\theta}^j & \text{otherwise} \end{cases}$$

We can also write this as

$$\boldsymbol{\theta}^{j+1} = \boldsymbol{\theta}^j + \frac{y_i - \text{sign}((\boldsymbol{\theta}^j)^T \widetilde{\mathbf{x}}_i)}{2} \cdot \widetilde{\mathbf{x}}_i$$

You will encounter an expression that looks a lot like this on the next homework…

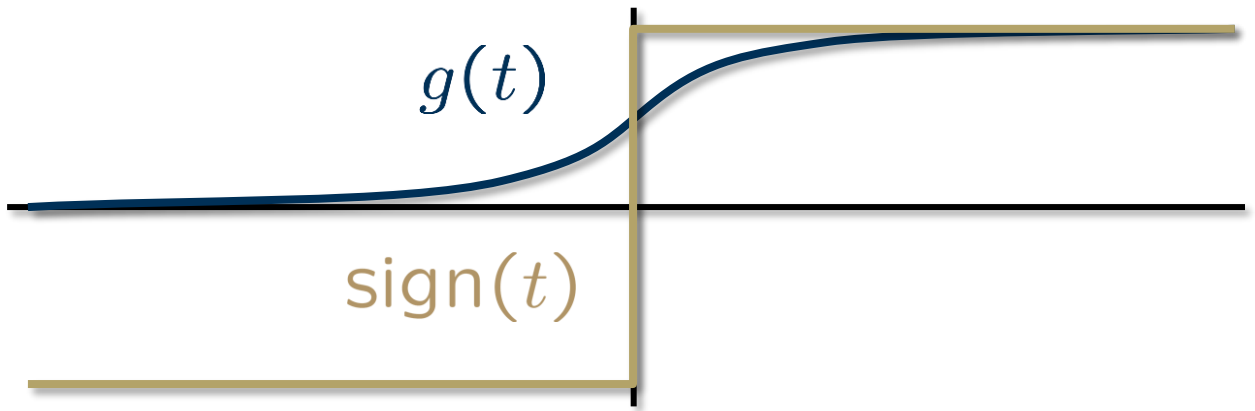
PLA as stochastic gradient descent

In the next homework, you will implement a ***stochastic gradient descent*** version of logistic regression, where each update is of the form

$$\boldsymbol{\theta}^{j+1} = \boldsymbol{\theta}^{j} + \alpha(y_i - g((\boldsymbol{\theta}^{j})^T \widetilde{\mathbf{x}}_i)) \cdot \widetilde{\mathbf{x}}_i$$

In contrast, the PLA consists of updates of the form

$$\boldsymbol{\theta}^{j+1} = \boldsymbol{\theta}^{j} + \frac{1}{2}(y_i - \text{sign}((\boldsymbol{\theta}^{j})^T \widetilde{\mathbf{x}}_i)) \cdot \widetilde{\mathbf{x}}_i$$



Provable guarantees for the PLA

We can view the PLA as (almost) stochastic gradient descent applied to a slightly different likelihood function than we considered in the context of logistic regression

However, we can also provide very simple proofs that, under certain (nonparametric) assumptions, the PLA will result in a good classifier

In particular, we will see that if the training data is ***linearly separable***, then the PLA will find a ***separating hyperplane*** in a finite number of iterations

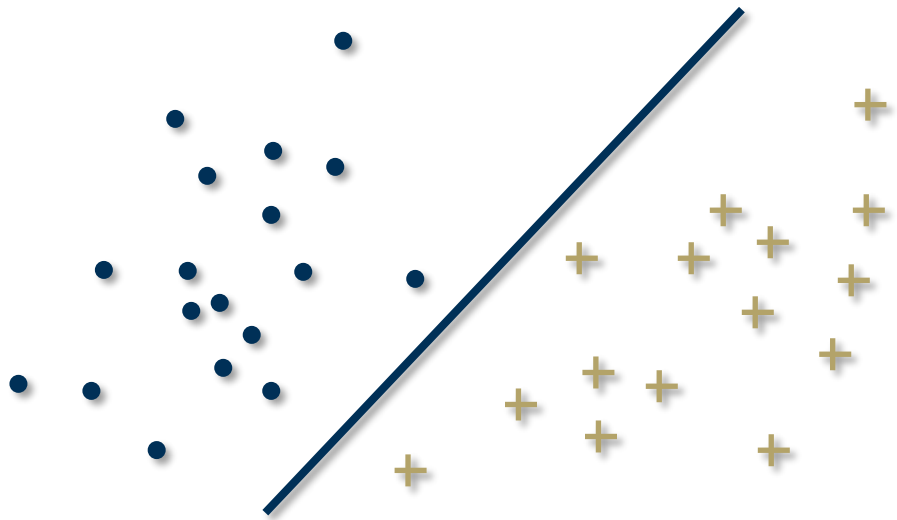
Linearly separable data sets

We say that a data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ is ***linearly separable*** if there exists $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that

$$y_i = \text{sign}(\mathbf{w}^T \mathbf{x}_i + b)$$

for $i = 1, \ldots, n$

We refer to $\{\mathbf{x} : \mathbf{w}^T \mathbf{x} + b = 0\}$ as a ***separating hyperplane***



How can we find a separating hyperplane?

One approach is to use the PLA

You will prove this on the next homework, but to see roughly how the proof works, we need to do a bit of geometry first

Let $\mathbf{w}, b$ define a hyperplane, and pick two points $\mathbf{x}, \mathbf{x}'$ on the hyperplane, then

$$\begin{aligned} 0 &= (\mathbf{w}^T\mathbf{x} + b) - (\mathbf{w}^T\mathbf{x}' + b) \\ &= \mathbf{w}^T(\mathbf{x} - \mathbf{x}') \end{aligned}$$

Hence, $\mathbf{w}$ is ***orthogonal*** to all vectors that are ***parallel*** to the hyperplane

Geometry of separating hyperplanes

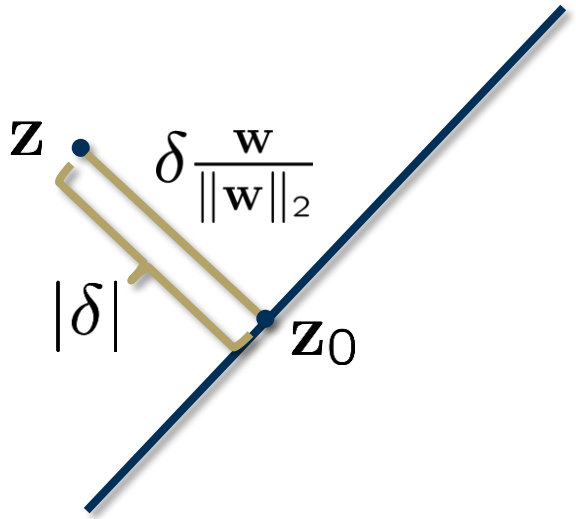
We call $\frac{\mathbf{w}}{\|\mathbf{w}\|_2}$ the ***normal vector*** to the hyperplane

It is unique up to its sign

Question

Let $\mathbf{z} \in \mathbb{R}^d$. How far is $\mathbf{z}$ from $\{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^T\mathbf{x} + b = 0\}$?

Write $\mathbf{z} = \mathbf{z}_0 + \delta\frac{\mathbf{w}}{\|\mathbf{w}\|_2}$ where $\mathbf{w}^T\mathbf{z}_0 + b = 0$ and $\delta \in \mathbb{R}$



Distance to the hyperplane

We can get a formula for the distance from $\mathbf{z}$ to $\{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^T\mathbf{x} + b = 0\}$ by observing that

$$\begin{aligned}\mathbf{w}^T\mathbf{z} + b &= \mathbf{w}^T\left(\mathbf{z}_0 + \delta\frac{\mathbf{w}}{\|\mathbf{w}\|_2}\right) + b \\ &= \mathbf{w}^T\mathbf{z}_0 + b + \delta\frac{\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|_2} \\ &= \delta\|\mathbf{w}\|_2\end{aligned}$$

$$|\delta| = \frac{|\mathbf{w}^T\mathbf{z} + b|}{\|\mathbf{w}\|_2}$$

PLA finds a separating hyperplane

Theorem

If a separating hyperplane exists, then the PLA will find one in finitely many iterations.

Let $\boldsymbol{\theta}^*$ define a separating hyperplane with $\|\mathbf{w}^*\|_2 = 1$. Let

$$\rho = \min_i |(\boldsymbol{\theta}^*)^T \widetilde{\mathbf{x}}_i| = \min_i |(\mathbf{w}^*)^T \mathbf{x}_i + b^*|$$

denote the distance from the hyperplane to the closest $\mathbf{x}_i$ in the training data, and let $R = \max_i \|\mathbf{x}_i\|_2$.

Suppose that we count iterations as the number of actual updates to $\boldsymbol{\theta}$, i.e., we assume that at each iteration we select an i such that $y_i \neq \text{sign}((\boldsymbol{\theta}^j)^T \widetilde{\mathbf{x}}_i)$ and update according to $\boldsymbol{\theta}^{j+1} = \boldsymbol{\theta}^j + y_i \widetilde{\mathbf{x}}_i$

PLA finds a separating hyperplane

Under these assumptions, you can (will!) show that if at iteration j there exists an $\mathbf{x}_i$ such that $y_i \neq \text{sign}((\boldsymbol{\theta}^j)^T \widetilde{\mathbf{x}}_i)$, then we necessarily have

$$j \leq \frac{(R^2+1)\|\boldsymbol{\theta}^*\|_2^2}{\rho^2}$$

Said differently, if

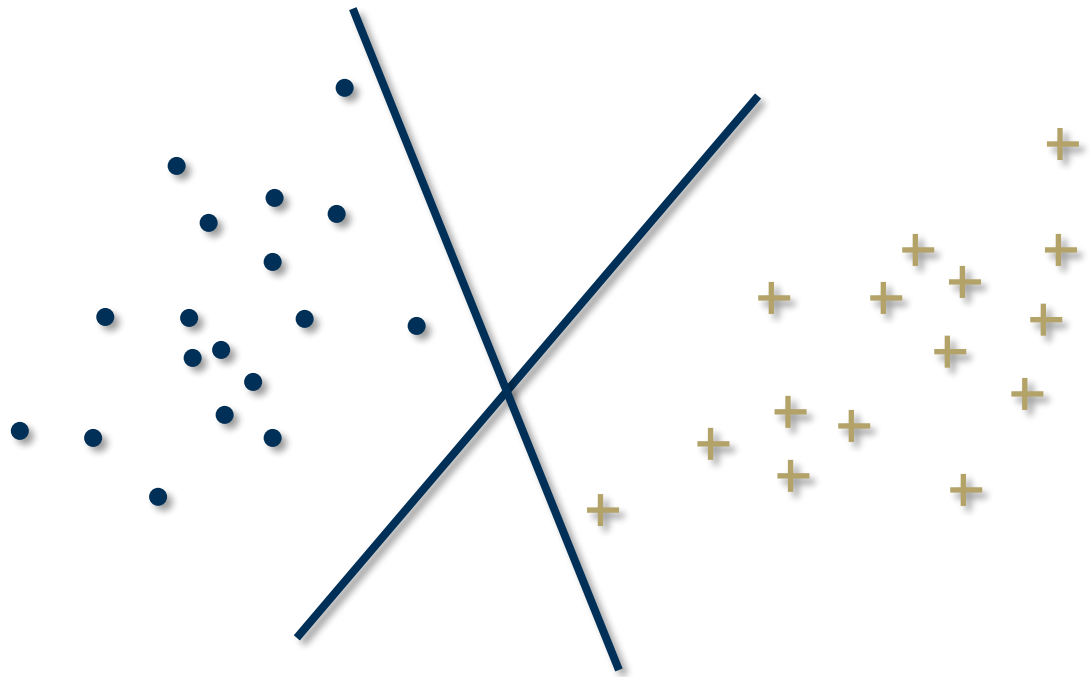
$$j > \frac{(R^2+1)\|\boldsymbol{\theta}^*\|_2^2}{\rho^2}$$

then we must have found a separating hyperplane

Drawbacks

- we don't know $\|\boldsymbol{\theta}^*\|_2^2/\rho^2$
- ***a*** separating hyperplane may not be the ***best*** hyperplane

Are all separating hyperplanes equal?



The maximum margin hyperplane

The ***margin*** ρ of a separating hyperplane is the distance from the hyperplane to the closest $\mathbf{x}_i$

$$\rho(\mathbf{w}, b) = \min_i \frac{|\mathbf{w}^T \mathbf{x}_i + b|}{\|\mathbf{w}\|_2}$$

The ***maximum margin*** or ***optimal*** separating hyperplane is the solution of

$$(\mathbf{w}^*, b^*) = \arg\max_{\mathbf{w}, b} \rho(\mathbf{w}, b)$$

Larger margin ➡ better generalization to new data

Canonical form

Our parameterization of a hyperplane as a normal vector $\mathbf{w}$ and an offset b is (sort of) overdetermined

- e.g., in $\mathbb{R}^2$ we are using three parameters to describe a line, which really only needs two parameters plus a sign…

Another way of seeing this is to realize that

$$\{\mathbf{x} : (\alpha\mathbf{w})^T\mathbf{x} + \alpha b = 0\}$$

describes the same hyperplane for all $\alpha \in \mathbb{R}$, and the same classifier for all $\alpha > 0$

It is often useful to remove this ambiguity by choosing a particular scaling. In the case of a separating hyperplane, we will say $(\mathbf{w}, b)$ are in ***canonical form*** if

- $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$ for ***all*** i
- $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$ for ***some*** i

Maximum margin revisited

If we restrict ourselves to hyperplanes in canonical form, then

$$\rho(\mathbf{w}, b) = \min_i \frac{|\mathbf{w}^T \mathbf{x}_i + b|}{\|\mathbf{w}\|_2} = \frac{1}{\|\mathbf{w}\|_2}$$

Thus we can express $(\mathbf{w}^*, b^*) = \arg\max_{\mathbf{w},b} \rho(\mathbf{w}, b)$ as

$$(\mathbf{w}^*, b^*) = \arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2$$

$$\text{s.t. } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \text{ for } i = 1, \ldots, n$$

Optimal separating hyperplanes

$$(\mathbf{w}^*, b^*) = \arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2$$

$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \text{ for } i = 1, \ldots, n$$

- This is an example of a ***constrained*** optimization program
  - in particular, ***a quadratic program***
- At the solution, there will always be at least some $\mathbf{x}_i$ such that $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$

  These are called ***support vectors***
- This optimization problem forms the core idea behind ***support vector machines***

What if our data is not linearly separable?

The plugin methods we described can naturally accommodate data that isn't perfectly linearly separable

How can we extend the notion of an optimal separating hyperplane to the case of data that is not separable?

The constraint that $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$ for every training sample can only be satisfied for separable data

Idea: Introduce slack variables $\xi_1, \ldots, \xi_n \geq 0$ that allow us to violate some of the constraints by changing them to

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$

How should we set $\xi_1, \ldots, \xi_n$?

Optimal soft-margin hyperplane

We would ideally like for most of the ξ_i to be zero

Note that if $\mathbf{x}_i$ is misclassified, then $\xi_i > 1$

Hence, our training error $\leq \frac{1}{n}\sum_{i=1}^{n}\xi_i$

The optimal ***soft-margin*** hyperplane is given by

$$(\mathbf{w}^*, b^*) = \arg\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, \ldots, n$$

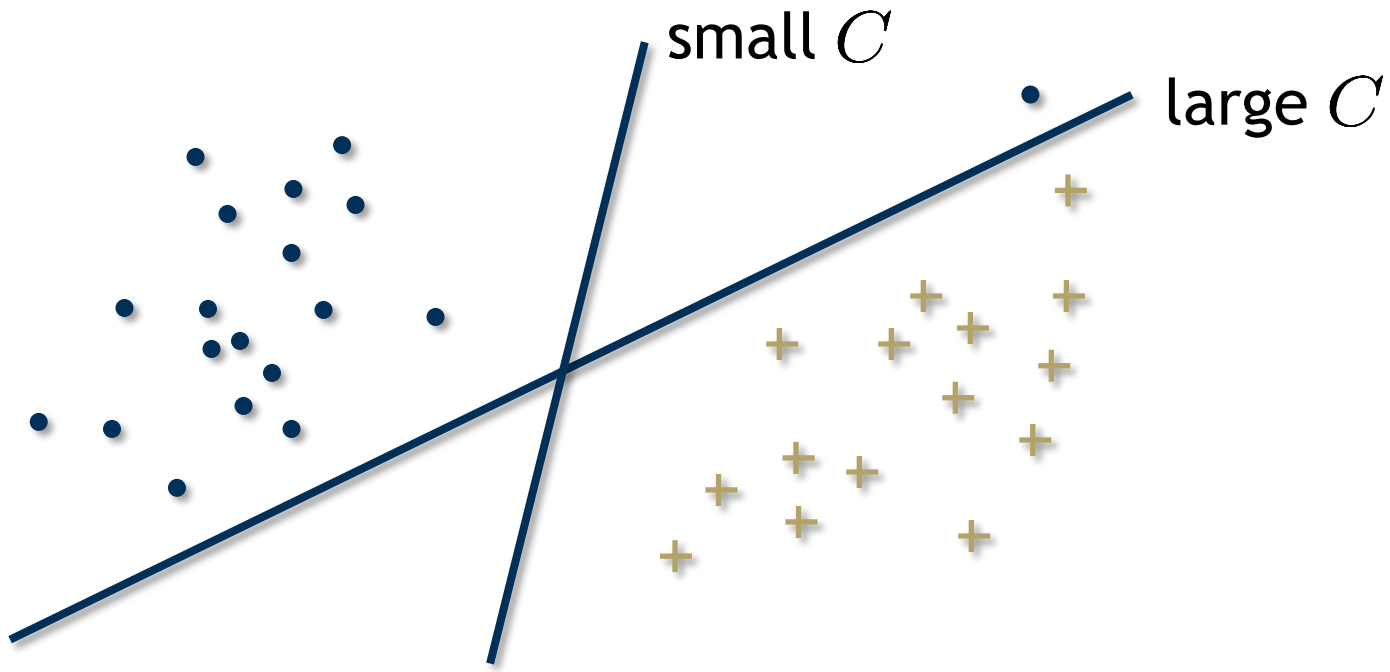
$$\xi_i \geq 0 \quad i = 1, \ldots, n$$

C is a "cost" parameter set by the user

Setting the cost parameter

C allows us to trade off between fitting the data and having a large "margin"

It also controls the influence of outliers



We will discuss strategies for setting C in more detail later on

Nonlinear feature maps

Sometimes linear classifiers are terrible!

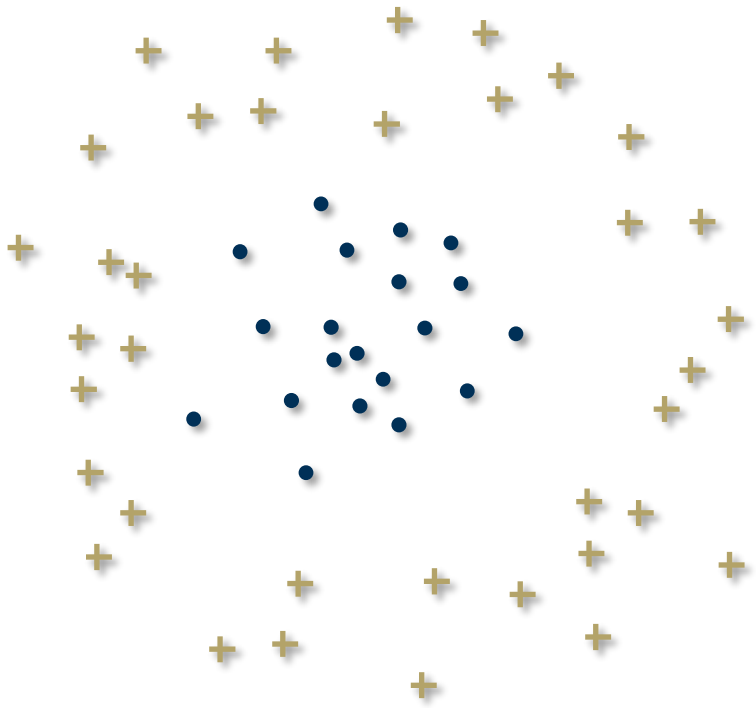
One way to create nonlinear estimators or classifiers is to first transform the data via a nonlinear feature map

$$\Phi : \mathbb{R}^d \to \mathbb{R}^{d'}$$

After applying Φ, we can then try applying a linear method to the transformed data

$$\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)$$

Example



This data set is not linearly separable

Consider the mapping

$$\Phi(\mathbf{x}) = \begin{bmatrix} 1 \\ x(1) \\ x(2) \\ x(1)x(2) \\ x(1)^2 \\ x(2)^2 \end{bmatrix}$$

The dataset ***is*** linearly separable after applying this feature map:

$$\mathbf{w} = [-\tau, 0, 0, 0, 1, 1]^T$$

Issues with nonlinear feature maps

Suppose we transform our data via

$$\mathbf{x} = \begin{bmatrix} x(1) \\ \vdots \\ x(d) \end{bmatrix} \xrightarrow{\Phi} \Phi(\mathbf{x}) = \begin{bmatrix} \Phi^{(1)}(x) \\ \\ \vdots \\ \\ \Phi^{(p)}(x) \end{bmatrix}$$

where $p \gg d$

- If $p \geq n$, then this can lead to problems with overfitting
  - you can ***always*** find a separating hyperplane
- When p is very large, there can be an increased ***computational*** burden

The “kernel trick”

Fortunately, there is a clever way to get around this computational challenge by exploiting two facts:

- Many machine learning algorithms only involve the data through ***inner products***
- For many interesting feature maps Φ, the function

$$k(\mathbf{x}, \mathbf{x}') := \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle = \Phi(\mathbf{x}')^T \Phi(\mathbf{x})$$

has a simple, closed form expression that can be evaluated ***without explicitly calculating*** $\Phi(\mathbf{x})$ and $\Phi(\mathbf{x}')$

$$\langle \cdot, \cdot \rangle = \begin{array}{c} \text{standard} \\ \text{dot product} \end{array}$$

Kernel-based classifiers

Algorithms we've seen that only need to compute inner products:

- nearest-neighbor classifiers

$$\|\mathbf{x} - \mathbf{x}_i\|_2^2 = \langle \mathbf{x} - \mathbf{x}_i, \mathbf{x} - \mathbf{x}_i \rangle$$

$$\begin{aligned} \|\Phi(\mathbf{x}) - \Phi(\mathbf{x}_i)\|_2^2 &= \langle \Phi(\mathbf{x}) - \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) - \Phi(\mathbf{x}_i) \rangle \\ &= k(\mathbf{x}, \mathbf{x}) - 2k(\mathbf{x}, \mathbf{x}_i) + k(\mathbf{x}_i, \mathbf{x}_i) \end{aligned}$$

Kernel-based classifiers

Algorithms we've seen that only need to compute inner products:

- maximum margin hyperplanes

$$(\mathbf{w}^*, b^*) = \arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2$$

$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \text{ for } i = 1, \ldots, n$$

it is a fact that the optimal $\mathbf{w}$ can be expressed as $\mathbf{w} = \sum_{i=1}^n \alpha_i \mathbf{x}_i$ for some choice of α_i

$$\|\mathbf{w}\|_2^2 = \left\|\sum_{i=1}^n \alpha_i \mathbf{x}_i\right\|_2^2 = \sum_{i,j=1}^n \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

$$\mathbf{w}^T\mathbf{x}_i = \left(\sum_{j=1}^n \alpha_j \mathbf{x}_j\right)^T \mathbf{x}_i = \sum_{j=1}^n \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

Kernel-based classifiers

Algorithms we've seen that only need to compute inner products:

- maximum margin hyperplanes

$$(\boldsymbol{\alpha}^*, b^*) = \arg\min_{\boldsymbol{\alpha}, b} \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

$$\text{s.t. } y_i \left(\sum_{j=1}^{n} \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + b \right) \geq 1$$

$$\text{for } i = 1, \ldots, n$$

Example: Quadratic kernel

$$
\begin{aligned}
(\mathbf{u}^T\mathbf{v})^2 &= \left(\sum_{i=1}^{d} u(i)v(i)\right)^2 \\
&= \left(\sum_{i=1}^{d} u(i)v(i)\right)\left(\sum_{j=1}^{d} u(j)v(j)\right) \\
&= \sum_{i=1}^{d}\sum_{j=1}^{d} u(i)v(i)u(j)v(j) \\
&= \sum_{i=1}^{d} u(i)^2 v(i)^2 + \sum_{i\neq j} u(i)u(j)\cdot v(i)v(j)
\end{aligned}
$$

Quadratic kernel

$$\sum_{i=1}^{d} u(i)^2 v(i)^2 + \sum_{i \neq j} u(i)u(j) \cdot v(i)v(j) \stackrel{?}{=} \langle \Phi(\mathbf{u}), \Phi(\mathbf{v}) \rangle$$

What is Φ and what is the dimension p of the corresponding feature space?

$$\Phi(\mathbf{u}) = \Big[u(1)^2, \ldots, u(d)^2, \ldots \\ \sqrt{2}u(1)u(2), \ldots, \sqrt{2}u(d-1)u(d)\Big]^T$$

$$p = d + \frac{d(d-1)}{2}$$

Nonhomogeneous quadratic kernel

$$k(\mathbf{u}, \mathbf{v}) = (\mathbf{u}^T\mathbf{v} + 1)^2 = (\mathbf{u}^T\mathbf{v})^2 + 2\mathbf{u}^T\mathbf{v} + 1$$

In this case, the corresponding $\Phi(\mathbf{u})$ is similar to the homogenous quadratic kernel, but it also replicates $\mathbf{u}$

$$\Phi(\mathbf{u}) = \Big[1, \sqrt{2}u(1), \ldots, \sqrt{2}u(d), u(1)^2, \ldots, u(d)^2, \ldots \\ \sqrt{2}u(1)u(2), \ldots, \sqrt{2}u(d-1)u(d)\Big]^T$$

Inner product kernel

Definition. An ***inner product kernel*** is a mapping

$$k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$

for which there exists an inner product space $\mathcal{H}$ and a mapping $\Phi : \mathbb{R}^d \to \mathcal{H}$ such that

$$k(\mathbf{u}, \mathbf{v}) = \langle \Phi(\mathbf{u}), \Phi(\mathbf{v}) \rangle_{\mathcal{H}}$$

for all $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$

Given a function $k(\mathbf{u}, \mathbf{v})$, how can we tell when it is an inner product kernel?

- Mercer's theorem
- Positive semidefinite property

Positive semidefinite kernels

We say that $k(\mathbf{u}, \mathbf{v})$ is a ***positive semidefinite*** kernel if

- k is symmetric
- for all n and all $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, the ***Gram matrix*** $\mathbf{K}$ defined by

$$K(i, j) = k(\mathbf{x}_i, \mathbf{x}_j)$$

  is positive semidefinite, i.e., $\mathbf{x}^T \mathbf{K} \mathbf{x} \geq 0$ for all $\mathbf{x}$

Theorem

k is an inner product kernel if and only if k is a positive semidefinite kernel

Proof: (Future homework)

Examples: Polynomial kernels

Homogeneous polynomial kernel

$$k(\mathbf{u}, \mathbf{v}) = (\mathbf{u}^T \mathbf{v})^m \qquad m = 1, 2, \ldots$$

Inhomogenous polynomial kernel

$$k(\mathbf{u}, \mathbf{v}) = (\mathbf{u}^T \mathbf{v} + c)^m \qquad m = 1, 2, \ldots$$
$$c > 0$$

Φ maps to the set of all monomials of degree $\leq m$

Examples: Gaussian/RBF kernels

Gaussian / Radial basis function (RBF) kernel:

$$k(\mathbf{u}, \mathbf{v}) = (2\pi\sigma^2)^{-d/2} \exp\left(-\frac{\|\mathbf{u} - \mathbf{v}\|_2^2}{2\sigma^2}\right)$$

$$k(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u} - \mathbf{v}\|_2^2}{2\sigma^2}\right)$$

One can show that k is a positive semidefinite kernel, but what is $\mathcal{H}$?

$\mathcal{H}$ is ***infinite dimensional***!

Kernels in action: SVMs

In order to more deeply appreciate

- why we can kernelize the maximum margin optimization problem
- how to actually solve this optimization problem with a practical algorithm

we will need to spend a bit of time learning about constrained optimization

This stuff is super useful, even outside of this context