Data Mining & Machine Learning

CS37300
Purdue University

Sep 22, 2023

Today's topics

- Logistic Regression
- Kernels and Kernel-SVM

Modeling the Noise: Conditional Probability

- Logistic Model:

$$P_{w,b}(Y = y | X = x) = \frac{1}{1 + e^{-y(w^\top x + b)}}$$

Training: Logistic Regression

- How do we find a good $\hat{w}$?
- **Maximum conditional likelihood** estimation
- For a data set S = $\{(x_1,y_1),\ldots,(x_n,y_n)\}$
- Define the conditional likelihood

Models the labels y_i as being conditionally independent given x_i

$$L_{Y|X}(w,b;S) = \prod_{i=1}^{n} P_{w,b}(Y = y_i | X = x_i)$$

- The maximum conditional likelihood estimator is

$$(\hat{w},\hat{b}) = \underset{w,b}{\operatorname{argmax}}\, L_{Y|X}(w,b;S)$$

- Equivalently (because it's easier to work with), conditional **log-likelihood**:

$$l_{Y|X}(w,b;S) = \ln(L_{Y|X}(w,b;S))$$

- and then

$$(\hat{w},\hat{b}) = \underset{w,b}{\operatorname{argmax}}\, l_{Y|X}(w,b;S)$$

Training: Logistic Regression

- Explicitly:

$$
\begin{aligned}
l_{Y|X}(w, b; S) &= \ln\left(\prod_{i=1}^{n} P_w(Y = y_i | X = x_i)\right) \\
&= \sum_{i=1}^{n} \ln(P_w(Y = y_i | X = x_i)) \\
&= \sum_{i=1}^{n} \ln\left(\frac{1}{1 + e^{-y_i(w^\top x_i + b)}}\right) \\
&= -\sum_{i=1}^{n} \ln\left(1 + e^{-y_i(w^\top x_i + b)}\right)
\end{aligned}
$$

- Unfortunately, there is no simple expression for the $\widehat{w}$ that maximizes this
- But $-\sum_{i=1}^{n} \ln\left(1 + e^{-y_i(w^\top x_i + b)}\right)$ is a concave function, which can be maximized using iterative numerical methods: e.g., gradient ascent or Newton's method.

Training: Logistic Regression

- Explicitly: denote $\sigma(x) = \frac{1}{1+e^{-x}}$

- Simple derivative: $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x))$

$$\nabla l_{Y|X}(w,b;S) = \nabla \sum_{i=1}^{n} \ln\big(\sigma\big(y_i(w^\top x_i + b)\big)\big)$$

$$= \sum_{i=1}^{n} \frac{1}{\sigma(y_i(w^\top x_i + b))} \sigma\big(y_i(w^\top x_i + b)\big) \left(1 - \sigma\big(y_i(w^\top x_i + b)\big)\right) y_i [x_i, 1]^\top$$

$$= \sum_{i=1}^{n} \left(1 - \sigma\big(y_i(w^\top x_i + b)\big)\right) y_i [x_i, 1]^\top$$

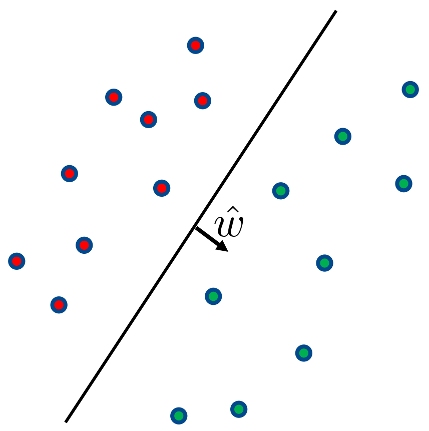
$$\text{gradient ascent: iterate} \quad (w,b) \leftarrow (w,b) + \epsilon \nabla l_{Y|X}(w,b;S)$$

Logistic Regression vs SVM

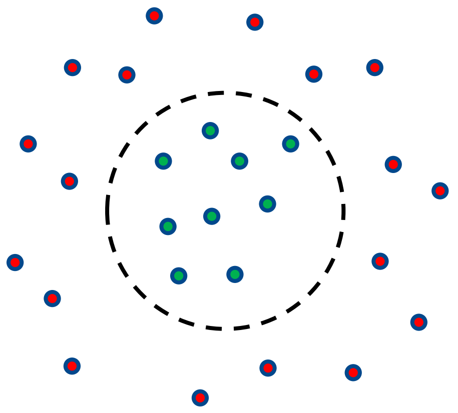
- For a given data set, which one should we use?
- If you think the data are (nearly) linearly separable, and with large margin, makes sense to use (Soft)-SVM
- If you think the cause of non-separability truly is label noise, makes sense to use logistic regression
- One nice thing about Logistic Regression is that it provides an estimated probability $P(y|x)$ for its predictions, rather than just a -1,1 prediction.
- Doesn't hurt to try them both and find out which is better on a validation set!

Nonlinear Decision Boundaries

- Linear separators are simple



- But there are many scenarios that are separable, only non-linearly

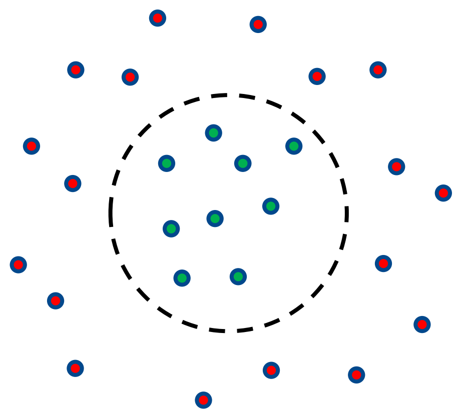


- We want to learn these with optimization based ML (SVM)

Learning Nonlinear Decision Boundaries

- We've already seen some nonlinear separators:
  - Decision Trees, KNN
- Question: How can we learn this example using SVM?

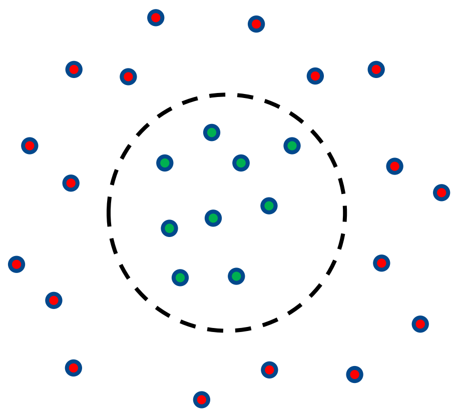
Example:



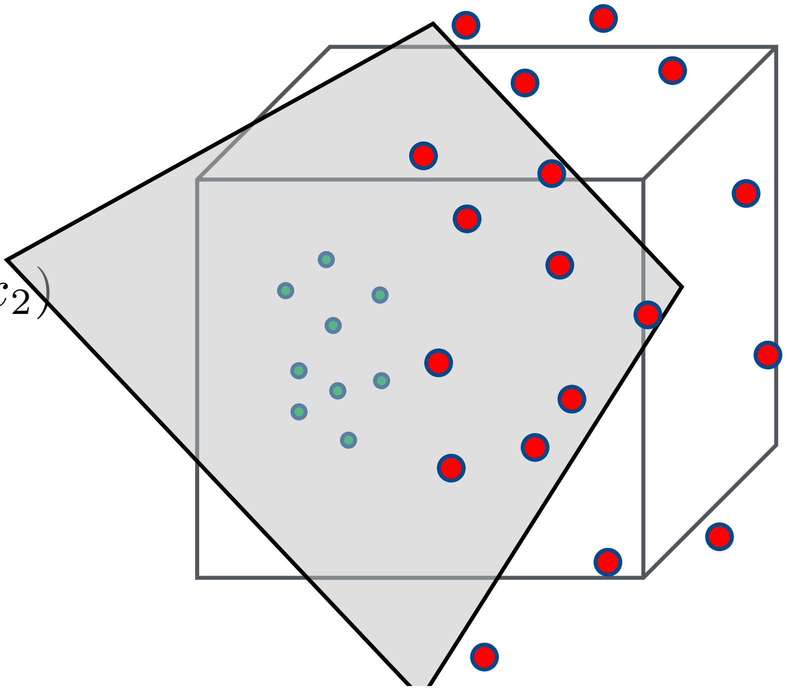
Learning Nonlinear Decision Boundaries

- We've already seen some nonlinear separators:
  - Decision Trees, KNN
- Question: How can we learn this example using SVM?
  - Map the examples into a higher-dimensional space and learn a linear separator in that space

Example:



Re-represent each $\underline{x} = (x_1, x_2)$ as 3-dimensional vector $\phi(\underline{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$



Representing Nonlinear Decision Boundaries

- General strategy: consider a function: $\phi(x) \in \mathbb{R}^N$
  - N may be large, or even infinite
- Then learn a linear separator in the ϕ space:

$$\hat{h}(x) = \text{sign}\big(w^\top \phi(x)\big)\,, \quad \text{where } w \in \mathbb{R}^N$$

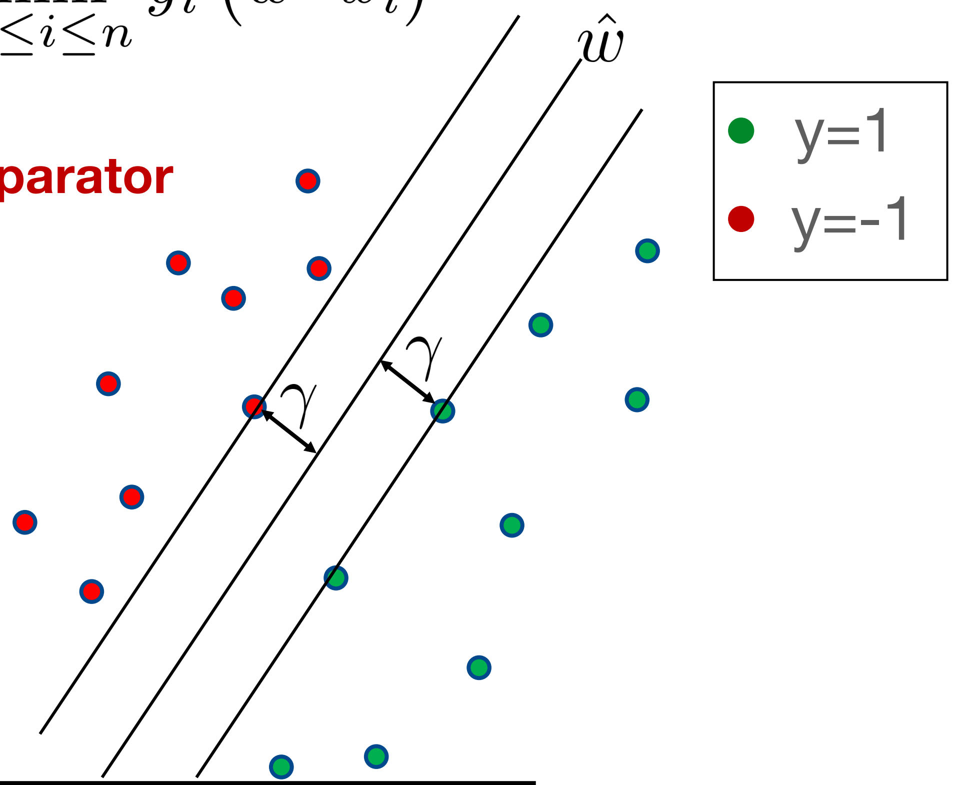
Refresher: Support Vector Machine (SVM)

$$\hat{w} = \underset{w:\|w\|=1}{\operatorname{argmax}} \min_{1\le i\le n} y_i \left(w^\top x_i\right)$$

- SVM: pick the **maximum margin separator**

geometric margin:

$$\gamma = \max_{w:\|w\|=1} \min_{1\le i\le n} y_i \left(w^\top x_i\right)$$



$$\boxed{\begin{array}{ll} \text{Minimize} & \|w\|^2 \\ \text{subject to} & y_i\left(w^\top x_i\right) \ge 1,\ \forall i:\ 1\le i\le n \end{array}}$$

Representing Nonlinear Decision Boundaries

- General strategy: consider a function: $\phi(x) \in \mathbb{R}^N$
  - N may be large, or even infinite
- Then learn a linear separator in the ϕ space:

$$\hat{h}(x) = \operatorname{sign}\left(w^\top \phi(x)\right), \quad \text{where } w \in \mathbb{R}^N$$

$$\hat{w} = \underset{w \in \mathbb{R}^N : \|w\| = 1}{\operatorname{argmax}} \; \min_{1 \le i \le n} y_i \left(w^\top \phi(x_i)\right)$$

$$\boxed{\begin{array}{ll} \text{Minimize} & \|w\|^2 \\ \text{subject to} & y_i \left(w^\top \phi(x_i)\right) \ge 1, \; \forall i : \; 1 \le i \le n \end{array}}$$

- Question: Why might this be a problem?

Representing Nonlinear Decision Boundaries

- General strategy: consider a function: $\phi(x) \in \mathbb{R}^N$
    - N may be large, or even infinite
- Then learn a linear separator in the ϕ space:

$$\hat{h}(x) = \operatorname{sign}\left(w^\top \phi(x)\right), \quad \text{where } w \in \mathbb{R}^N$$

- Two concerns:

1. Computation and memory required to represent $\phi(x)$, w
2. Dimension of linear separators on $\mathbb{R}^N$ is N.
   $\rightarrow$ Concerns about overfitting

Today's lecture is mostly about the first concern.
Solution: Kernels

The Dual Form of the SVM Optimization Problem

$$\hat{w} = \operatorname*{argmax}_{w:\|w\|=1} \min_{1\le i\le n} y_i \left(w^\top x_i\right)$$

- Recall: The SVM **classifier** is the unique solution to a **quadratic program**:

$$\boxed{\begin{array}{ll} \text{Minimize} & \|w\|^2 \\ \text{subject to} & y_i\left(w^\top x_i\right) \ge 1, \ \forall i: \ 1 \le i \le n \end{array}}$$

- We can re-express this in Lagrangian dual form:

$$\hat{w} = \textstyle\sum_{i=1}^n \alpha_i y_i x_i$$

where $\alpha_1, \ldots, \alpha_n$ are solutions to:

$$\boxed{\begin{array}{ll} \text{Maximize} & \displaystyle\sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n y_i y_j \alpha_i \alpha_j x_i^\top x_j \\ \text{subject to} & \alpha_i \ge 0, \ \forall i: \ 1 \le i \le n \\ & \displaystyle\sum_{i=1}^n \alpha_i y_i = 0 \end{array}}$$

The Dual Form of the SVM Optimization Problem

$$\hat{w} = \underset{w:\|w\|=1}{\operatorname{argmax}} \min_{1\le i\le n} y_i\left(w^\top x_i\right)$$

- Recall: The SVM **classifier** is the unique solution to a **quadratic program**:

$$\boxed{\begin{array}{ll} \text{Minimize} & \|w\|^2 \\ \text{subject to} & y_i\left(w^\top x_i\right) \ge 1,\ \forall i:\ 1\le i\le n \end{array}}$$

- We can re-express this in Lagrangian dual form:

$$\hat{w} = \textstyle\sum_{i=1}^{n} \alpha_i y_i x_i$$

where $\alpha_1, \ldots, \alpha_n$ are solutions to:

$$\boxed{\begin{array}{ll} \text{Maximize} & \displaystyle\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j x_i^\top x_j \\ \text{subject to} & \alpha_i \ge 0,\ \forall i:\ 1\le i\le n \\ & \displaystyle\sum_{i=1}^{n}\alpha_i y_i = 0 \end{array}}$$

Side note: the training points (x_i, y_i) with non-zero α_i values are called the **support vectors.**

SVM with high-dim mapping

- We could use a mapping ϕ to get a non-linear separator:

$$\hat{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{\phi(x_i)}$$

where $\alpha_1, \ldots, \alpha_n$ are solutions to:

$$\boxed{\begin{aligned} &\text{Maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{\phi(x_i)}^\top \boldsymbol{\phi(x_j)} \\ &\text{subject to} \quad \alpha_i \geq 0, \; \forall i : \; 1 \leq i \leq n \\ &\qquad\qquad\quad \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned}}$$

- Notice training only uses ϕ when computing inner product $\phi(x_i)^\top \phi(x_j)$

- Instead of starting by defining ϕ, we could start by defining a function $K(x_i, x_j)$ that computes an inner product, without computing ϕ: $K(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$

- Kernel: way to compute inner products without computing $\phi(x)$!

SVM with high-dim mapping

- We could use a mapping ϕ to get a non-linear separator:

$$\hat{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{\phi(x_i)}$$

where $\alpha_1, \ldots, \alpha_n$ are solutions to:

$$\boxed{\begin{aligned} &\text{Maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{\phi(x_i)^\top \phi(x_j)} \\ &\text{subject to} \quad \alpha_i \geq 0, \ \forall i : \ 1 \leq i \leq n \\ &\qquad\qquad\quad \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned}}$$

- Notice training only uses ϕ when computing inner product

$$\phi(x_i)^\top \phi(x_j)$$

- And the classifier also uses an inner product (no need to compute w)

$$h_{\hat{w}}(x) = \text{sign}\big(\hat{w}^\top \phi(x)\big) = \text{sign}\left(\left(\sum_{i=1}^{n} \alpha_i y_i \phi(x_i)\right)^\top \phi(x)\right) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \boldsymbol{\phi(x_i)^\top \phi(x)}\right)$$

- Replace all of these with

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

Kernel SVM

- Training time:

Solve for $\alpha_1, \ldots, \alpha_n$:

$$\boxed{\begin{array}{ll} \text{Maximize} & \displaystyle\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{K(x_i, x_j)} \\ \text{subject to} & \alpha_i \geq 0, \ \forall i : \ 1 \leq i \leq n \\ & \displaystyle\sum_{i=1}^{n} \alpha_i y_i = 0 \end{array}}$$

- Test time:

- Classify a new point x with

$$\hat{h}(x) = \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \boldsymbol{K(x_i, x)} \right)$$

Kernels

Example:

Quadratic kernel: $K(\underline{u}, \underline{v}) = \left(\underline{u}^\top \underline{v} + 1\right)^2$

For $\underline{x} \in \mathbb{R}^d$, implicitly computes an inner product in $\frac{1}{2}d(d-1) + 2d + 1$ dim

e.g., For $\underline{x} \in \mathbb{R}^2$, implicitly defined ϕ :

$$\phi(\underline{x}) = \left[1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2\right]^\top$$

Check: $\phi(\underline{u})^\top \phi(\underline{v})$

$$= \left[1, \sqrt{2}u_1, \sqrt{2}u_2, u_1^2, u_2^2, \sqrt{2}u_1u_2\right] \left[1, \sqrt{2}v_1, \sqrt{2}v_2, v_1^2, v_2^2, \sqrt{2}v_1v_2\right]^\top$$

$$= 1 + 2u_1v_1 + 2u_2v_2 + u_1^2v_1^2 + u_2^2v_2^2 + 2u_1u_2v_1v_2$$

$$= (1 + u_1v_1 + u_2v_2)^2 = \left(1 + \underline{u}^\top \underline{v}\right)^2 = K(\underline{u}, \underline{v})$$

The point is that we can compute this higher-dim inner product just by evaluating K(u,v): no need to compute ϕ

Kernels

- More Examples:
  - Polynomial kernel: $K(\underline{u}, \underline{v}) = \left(\underline{u}^\top \underline{v} + 1\right)^p$
    - Implicitly computes an inner product in $\sim d^p$ dimensions
  - Gaussian kernel:

$$K(\underline{u}, \underline{v}) = e^{-\frac{\|\underline{u} - \underline{v}\|^2}{2\sigma^2}}$$

    - Implicitly computes an inner product in **infinite** dimensions
    - Remark: This is the most popular kernel in practice

What are Legal Kernels?

- A kernel K(·,·) is a legal definition of an inner product
- Technically, this is called a **Mercer kernel**
  - A kernel should be a symmetric function: K(u,v)=K(v,u)
  - A kernel should also be positive semi-definite: namely,
    - For any set of data points $x_1,\ldots,x_n$
    - And for any values $a_1, \ldots, a_n \in \mathbb{R}$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K(x_i, x_j) \geq 0$$

- Mercer's Theorem: For any K satisfying the above, there exists some function ϕ such that

$$K(u,v) = \phi(u)^\top \phi(v)$$

- In other words: K is a valid kernel

Example: Soft-SVM with Gaussian Kernel

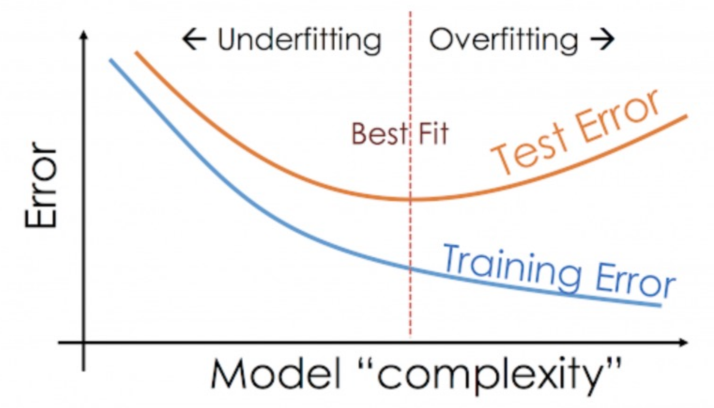
- A common choice is to use the Gaussian kernel

$$K(\underline{u}, \underline{v}) = e^{-\frac{\|\underline{u}-\underline{v}\|^2}{2\sigma^2}}$$

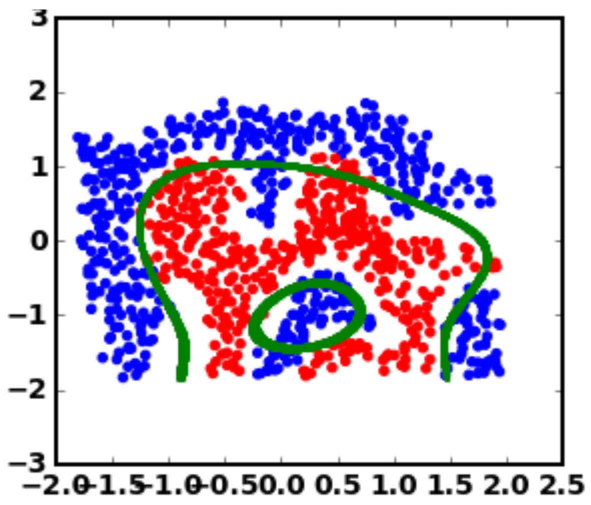
- σ called the **bandwidth**

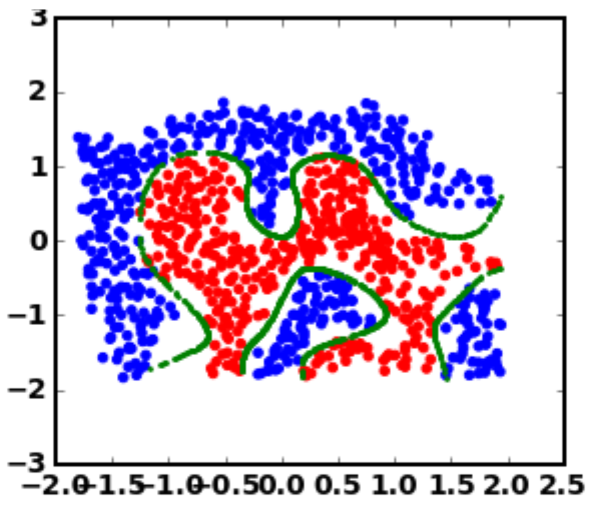
- It controls smoothness of the boundary

- Gives a notion of model complexity:



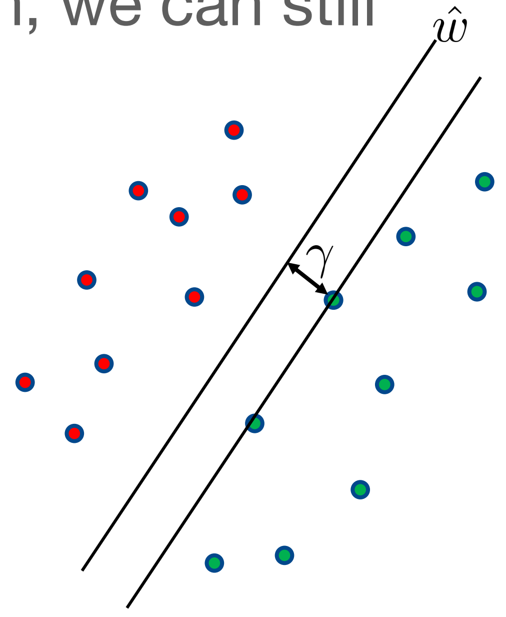
- large σ → simple boundaries, small σ → complex boundaries


large σ


small σ

What about overfitting?

- If we're using such a high-dimensional representation, won't the dimension be large? Doesn't this lead to overfitting?
- Margin: If we can find a solution with large margin, we can still avoid overfitting.
- This is true even with kernels

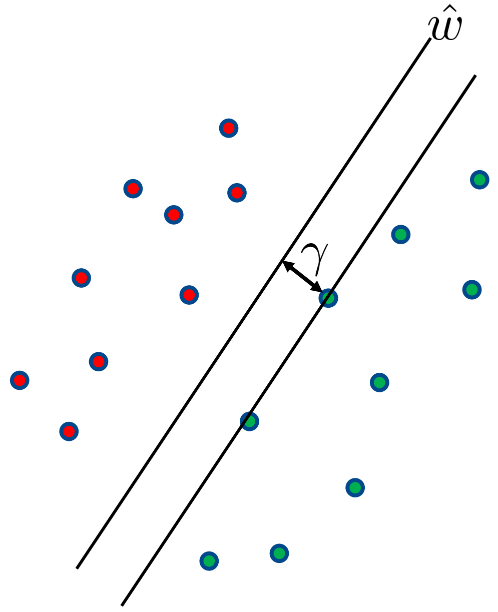


What about overfitting?

- Margin for kernel SVM:

geometric margin of classifier w:

$$\gamma = \min_{1 \le i \le n} y_i \left(\frac{w^\top}{\|w\|} \phi(x_i) \right)$$



Recall: The $\hat{w}$ solution of SVM primal problem satisfies $\|\hat{w}\| = \frac{1}{\gamma}$

For kernel SVM, this means

$$\frac{1}{\gamma^2} = \|\hat{w}\|^2 = \hat{w}^\top \hat{w} = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

Solve for γ to compute the margin

So we can also calculate the margin without computing ϕ, using K

Summary

- Kernel methods are a convenient family of algorithms
- They allow us to specify a non-linear representation for the classifier just by providing a kernel function, and the rest is automatic
- In this sense, they are “plug-and-play”: very easy to use
- We need to be careful that the implicit high-dimensional representation doesn’t lead to overfitting
- If the solution has large margin, it can avoid overfitting
- Recall that SVM is designed to maximize the margin, so it is well-suited to this type of guarantee