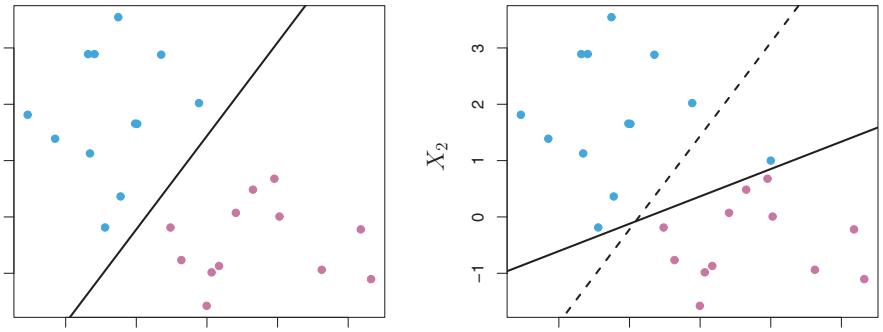
4 Soft-Margin Support Vector Machines; Features

SOFT-MARGIN SUPPORT VECTOR MACHINES (SVMs)

Solves 2 problems:
- Hard-margin SVMs fail if data not linearly separable.
- " " " sensitive to outliers.



sensitive.pdf (ISL, Figure 9.5) [Example where one outlier moves the decision boundary a lot.]

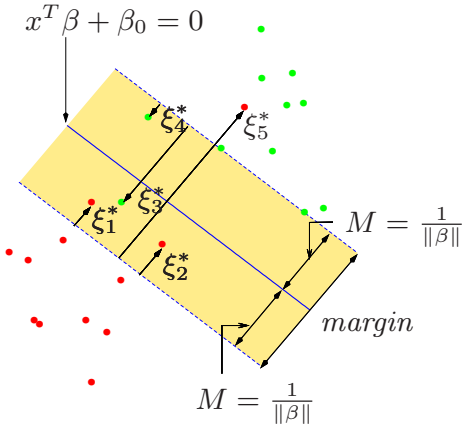
Idea: Allow some points to violate the margin, with slack variables.
Modified constraint for point i:

$$y_i(X_i \cdot w + \alpha) \geq 1 - \xi_i$$

[Observe that the only difference between these constraints and the hard margin constraints we saw last lecture is the extra slack term ξ_i.]
[We also impose new constraints, that the slack variables are never negative.]

$$\xi_i \geq 0$$

[This inequality ensures that all sample points that *don't* violate the margin are treated the same; they all have $\xi_i = 0$. Point i has nonzero ξ_i if and only if it violates the margin.]



slack.pdf (ESL, Figure 12.1) [A margin where some points have slack. For each violating point, the slack distance is $\xi_i^* = \xi_i/|w|$.]

To prevent abuse of slack, we add a loss term to objective fn.

Optimization problem:

Find w, α, and ξ_i that minimize $|w|^2 + C \sum_{i=1}^{n} \xi_i$
subject to $y_i(X_i \cdot w + \alpha) \geq 1 - \xi_i$ for all $i \in [1, n]$
$\xi_i \geq 0$ for all $i \in [1, n]$

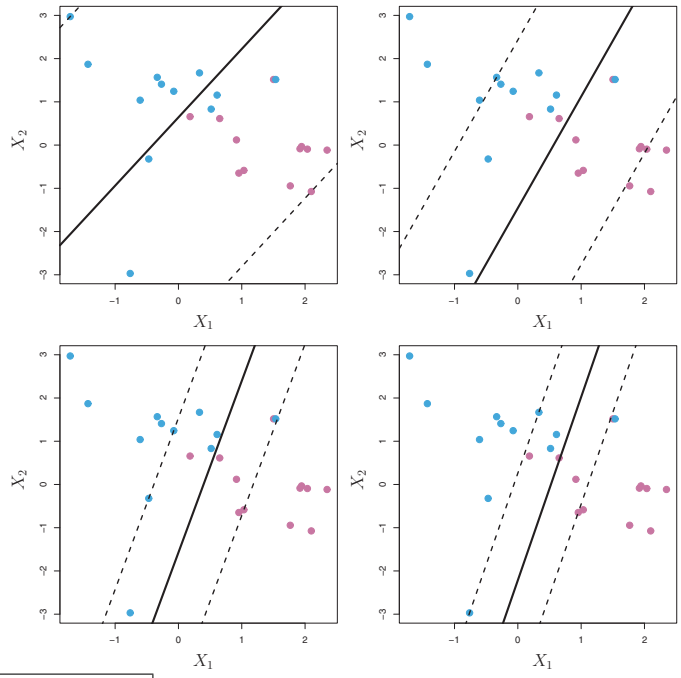
. . . a quadratic program in $d + n + 1$ dimensions and $2n$ constraints.
[It's a quadratic program because its objective function is quadratic and its constraints are linear inequalities.]

$C > 0$ is a scalar regularization hyperparameter that trades off:

	small C	big C
desire	maximize margin $1/\|w\|$	keep most slack variables zero or small
danger	underfitting (misclassifies much training data)	overfitting (awesome training, awful test)
outliers	less sensitive	very sensitive
boundary	more "flat"	more sinuous

[The last row only applies to nonlinear decision boundaries, which we'll discuss next. Obviously, a linear decision boundary can't be "sinuous."]

Use validation to choose C.



svmC.pdf (ISL, Figure 9.7) [Examples of how the slab varies with C. Smallest C at upper left; largest C at lower right.]

[One way to think about slack is to pretend that slack is money we can spend to buy permission for a sample point to violate the margin. The further a point penetrates the margin, the bigger the fine you have to pay. We want to make the margin as big as possible, but we also want to spend as little money as possible. If the regularization parameter C is small, it means we're willing to spend lots of money on violations so we can get a bigger margin. If C is big, it means we're cheap and we won't pay much for violations, even though we'll suffer a narrower margin . If C is infinite, we're back to a hard-margin SVM.]

FEATURES

Q: How to do nonlinear decision boundaries?

A: Make nonlinear features that lift points into a higher-dimensional space.
High-d linear classifier → low-d nonlinear classifier.

[Features work with all classifiers—not only linear classifiers like perceptrons and SVMs, but also classifiers that are not linear.]

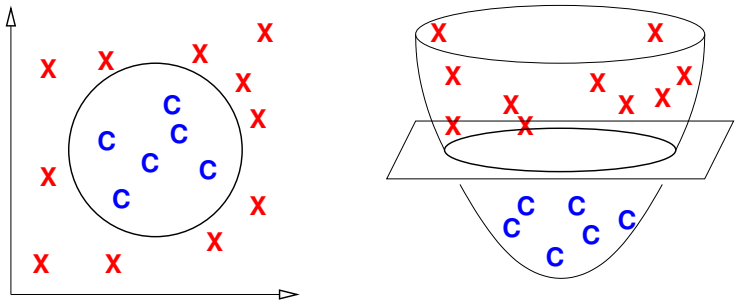
Example 1: The parabolic lifting map

$$\Phi(x) : \mathbb{R}^d \to \mathbb{R}^{d+1}$$

$$\Phi(x) = \begin{bmatrix} x \\ |x|^2 \end{bmatrix} \quad \leftarrow \text{lifts } x \text{ onto paraboloid } x_{d+1} = |x|^2$$

[We've added one new feature, $|x|^2$. Even though the new feature is just a function of other input features, it gives our linear classifier more power.]

Find a linear classifier in Φ-space.
It induces a sphere classifier in x-space.



[Draw this by hand. circledec.pdf]

Theorem: $\Phi(X_1), \ldots, \Phi(X_n)$ are linearly separable iff $X_1, \ldots, X_n$ are separable by a hypersphere.
(Possibly a degenerate hypersphere = hyperplane.)

Proof: Consider hypersphere in $\mathbb{R}^d$ w/center c & radius ρ. Points inside:

$$|x - c|^2 < \rho^2$$
$$|x|^2 - 2c \cdot x + |c|^2 < \rho^2$$
$$\underbrace{[-2c^\top \quad 1]}_{\text{normal vector}} \underbrace{\begin{bmatrix} x \\ |x|^2 \end{bmatrix}}_{\Phi(x)} < \rho^2 - |c|^2$$

Hence points inside sphere → same side of hyperplane in Φ-space.
(Reverse implication works too.)

[Hyperspheres include hyperplanes as a special, degenerate case. A hyperplane is essentially a hypersphere with infinite radius. So hypersphere decision boundaries can do everything hyperplane decision boundaries can do, plus a lot more. With the parabolic lifting map, if you pick a hyperplane in Φ-space that is vertical, you get a hyperplane in x-space.]

Example 2: Axis-aligned ellipsoid/hyperboloid decision boundaries

[Draw examples of axis-aligned ellipses & hyperbola.]

In 3D, these have the formula

$$Ax_1^2 + Bx_2^2 + Cx_3^2 + Dx_1 + Ex_2 + Fx_3 = -G$$

[Here, the capital letters are scalars, not matrices.]

$$\Phi(x) : \mathbb{R}^d \to \mathbb{R}^{2d}$$

$$\Phi(x) = [x_1^2 \quad \dots \quad x_d^2 \quad x_1 \quad \dots \quad x_d]^\top$$

Hyperplane is $\underbrace{(A, B, C, D, E, F)}_{w} \cdot \Phi(x) = -G$

[We've turned d input features into $2d$ features for our linear classifier. If the points are separable by an axis-aligned ellipsoid or hyperboloid, per the formula above, then the points lifted to Φ-space are separable by a hyperplane whose normal vector is (A, B, C, D, E, F).]

Example 3: Ellipsoid/hyperboloid

[Draw example of non-axis-aligned ellipse.]

General formula: [for an ellipsoid or hyperboloid]

$$Ax_1^2 + Bx_2^2 + Cx_3^2 + Dx_1x_2 + Ex_2x_3 + Fx_3x_1 + Gx_1 + Hx_2 + Ix_3 = -J$$

$$\Phi(x) : \mathbb{R}^d \to \mathbb{R}^{(d^2+3d)/2}$$

[Now, our decision function can be any degree-2 polynomial.]

Isosurface defined by this equation is called a quadric. [In the special case of two dimensions, it's also known as a conic section. So our decision boundary can be an arbitrary conic section.]

[You'll notice that there is a quadratic blowup in the number of features, because every *pair* of input features creates a new feature in Φ-space. If the dimension is large, these feature vectors are getting huge, and that's going to impose a serious computational cost. But it might be worth it to find good classifiers for data that aren't linearly separable.]

Example 4: Decision fn is degree-p polynomial

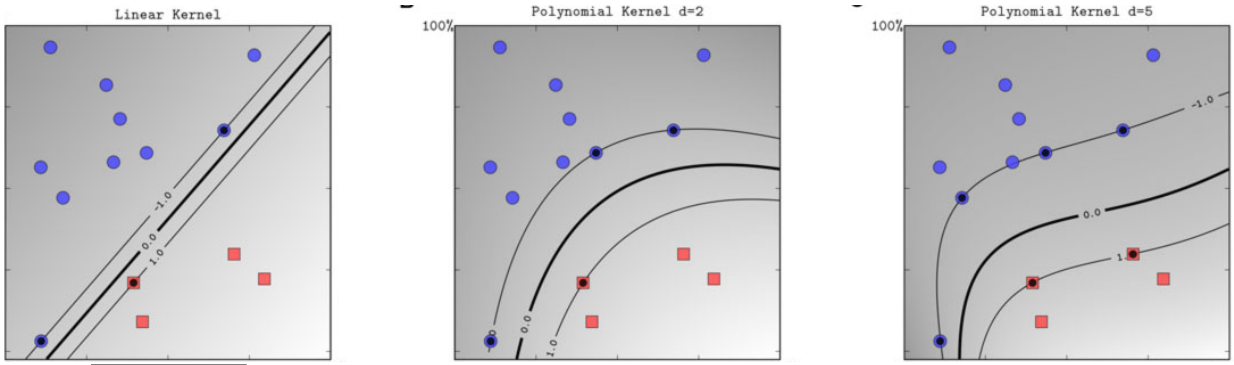
E.g. a cubic in $\mathbb{R}^2$:

$$\Phi(x) = [x_1^3 \quad x_1^2 x_2 \quad x_1 x_2^2 \quad x_2^3 \quad x_1^2 \quad x_1 x_2 \quad x_2^2 \quad x_1 \quad x_2]^\top$$

$$\Phi(x) : \mathbb{R}^d \to \mathbb{R}^{O(d^p)}$$

[Now we're really blowing up the number of features! If you have, say, 100 features per sample point and you want to use degree-4 decision functions, then each lifted feature vector has a length of roughly 4 million, and your learning algorithm will take approximately forever to run.]

[However, later in the semester we will learn an extremely clever trick that allows us to work with these huge feature vectors very quickly, without ever computing them. It's called "kernelization" or "the kernel trick." So even though it appears now that working with degree-4 polynomials is computationally infeasible, it can actually be done quickly.]

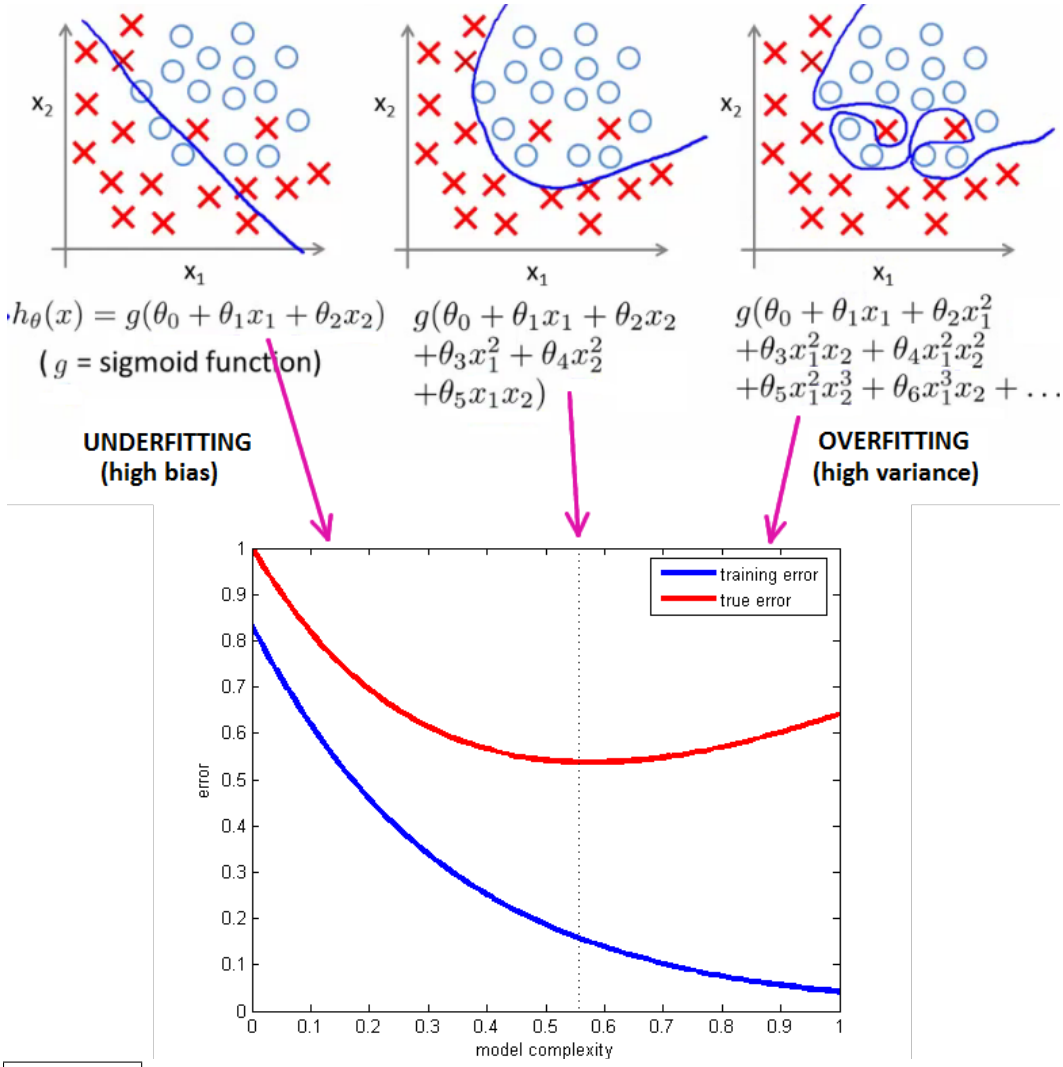


degree5.pdf [SVMs with degree 1/2/5 decision functions. Observe that the margin tends to get wider as the degree increases.]

[Increasing the degree like this accomplishes two things.

- First, the data might become linearly separable when you lift them to a high enough degree, even if the original data are not linearly separable.
- Second, raising the degree can increase the margin, so you might get a more robust separator.

However, if you raise the degree too high, you will overfit the data.]



overfit.pdf [Training vs. test error for degree 1/2/5 decision functions. (Artist's conception; these aren't actual calculations, just hand-drawn guesses. Please send me email if you know where to find figures like this with actual data.) In this example, a degree-2 decision gives the smallest test error.]

[Sometimes you should search for the ideal degree—not too small, not too big. It's a balancing act between underfitting and overfitting. The degree is an example of a *hyperparameter* that can be optimized by validation.]

[If you're using both polynomial features and a soft-margin SVM, now you have two hyperparameters: the degree and C. Generally, the optimal C will be different for every polynomial degree, so when you change the degree, you have to run validation again to find the best C for that degree.]

[So far I've talked only about polynomial features. But features can get much more complicated than polynomials, and they can be tailored to fit a specific problem. Let's consider a type of feature you might use if you wanted to implement, say, a handwriting recognition algorithm.]

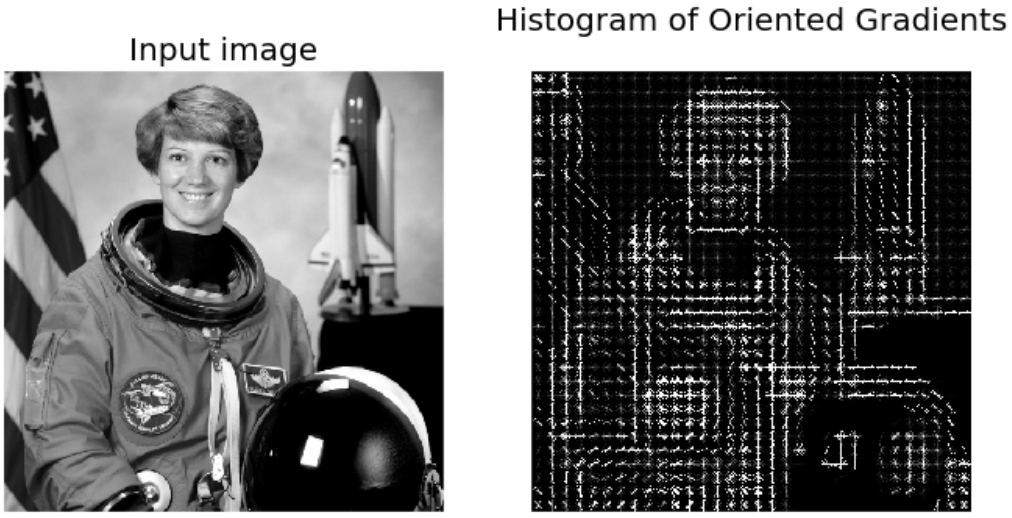
Example 5: Edge detection

Edge detector: algorithm for approximating grayscale/color gradients in image, e.g.

- tap filter
- Sobel filter
- oriented Gaussian derivative filter
  [images are discrete, not continuous fields, so approximation of gradients is necessary.]

[See "Image Derivatives" on Wikipedia.]

Collect line orientations in local histograms (each having 12 orientation bins per region); use histograms as features (*instead* of raw pixels).



orientgrad.png [Image histograms.]

Paper: Maji & Malik, 2009.

[If you want to, optionally, use these features in future homeworks and try to win the Kaggle competition, this paper is a good online resource.]

[When they use a linear SVM on the raw pixels, Maji & Malik get an error rate of 15.38% on the test set. When they use a linear SVM on the histogram features, the error rate goes down to 2.64%.]

[Many applications can be improved by designing application-specific features. There's no limit but your own creativity and ability to discern the structure hidden in your application.]