Support Vector Machines and Kernel Methods

Geoff Gordon

ggordon@cs.cmu.edu

June 15, 2004

Support vector machines

The SVM is a machine learning algorithm which

- solves classification problems
- uses a flexible representation of the class boundaries
- implements automatic complexity control to reduce overfitting
- has a single global minimum which can be found in polynomial time

It is popular because

- it can be easy to use
- it often has good generalization performance
- the same algorithm solves a variety of problems with little tuning

SVM concepts

Perceptrons

Convex programming and duality

Using maximum margin to control complexity

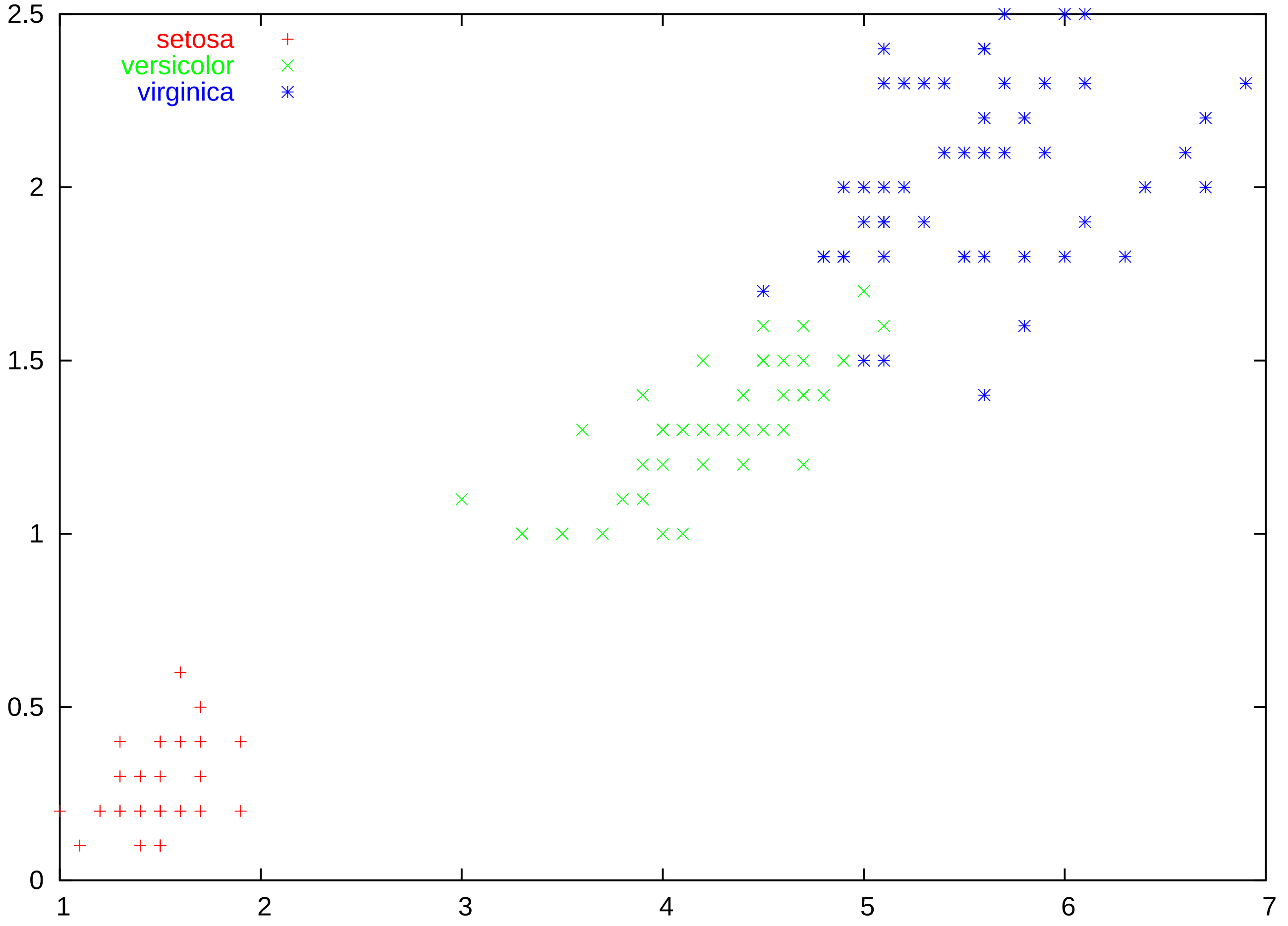
Representing nonlinear boundaries with feature expansion

The “kernel trick” for efficient optimization

Outline

- Classification problems
- Perceptrons and convex programs
- From perceptrons to SVMs
- Advanced topics

Classifi cation example—Fisher's irises



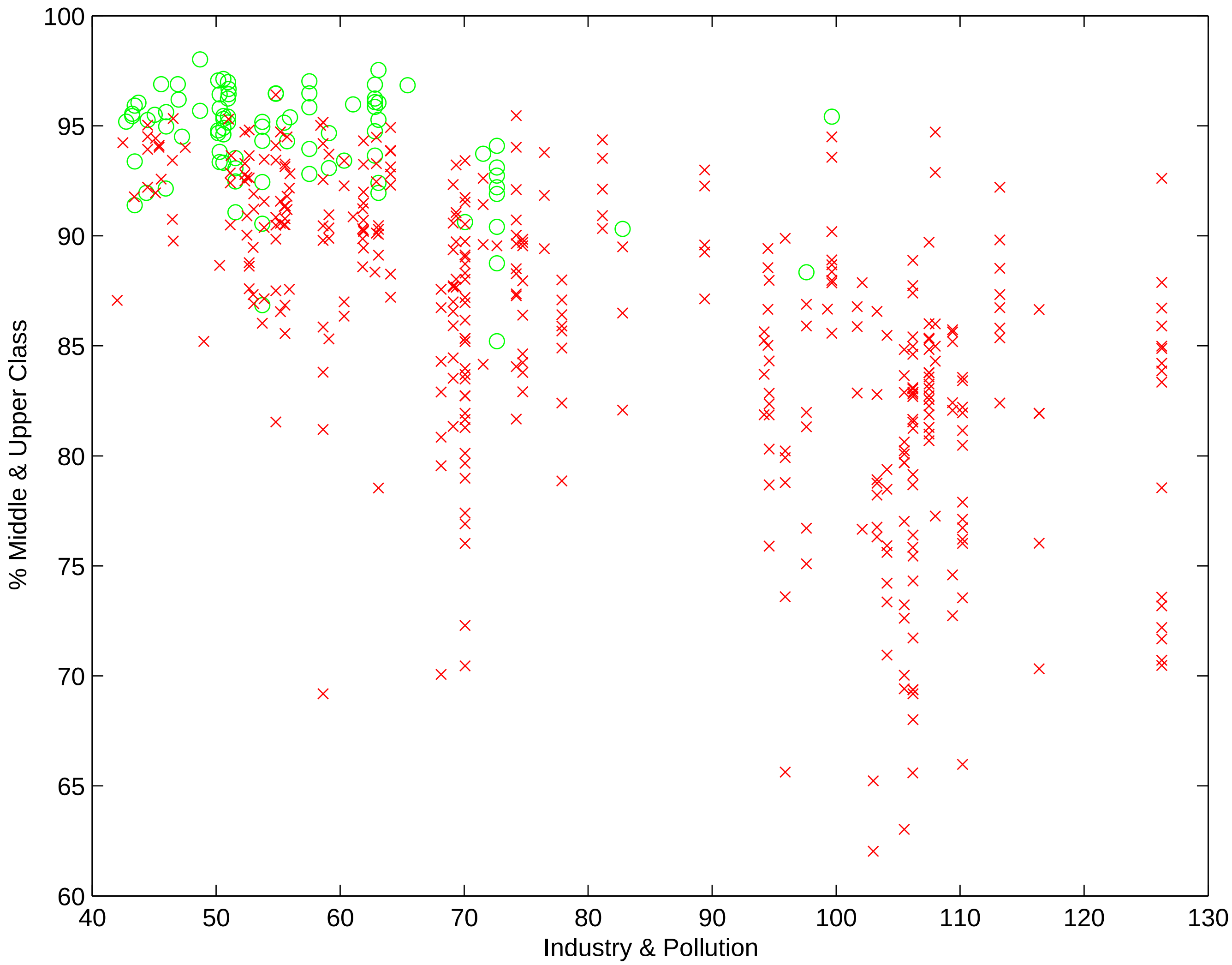
Iris data

Three species of iris

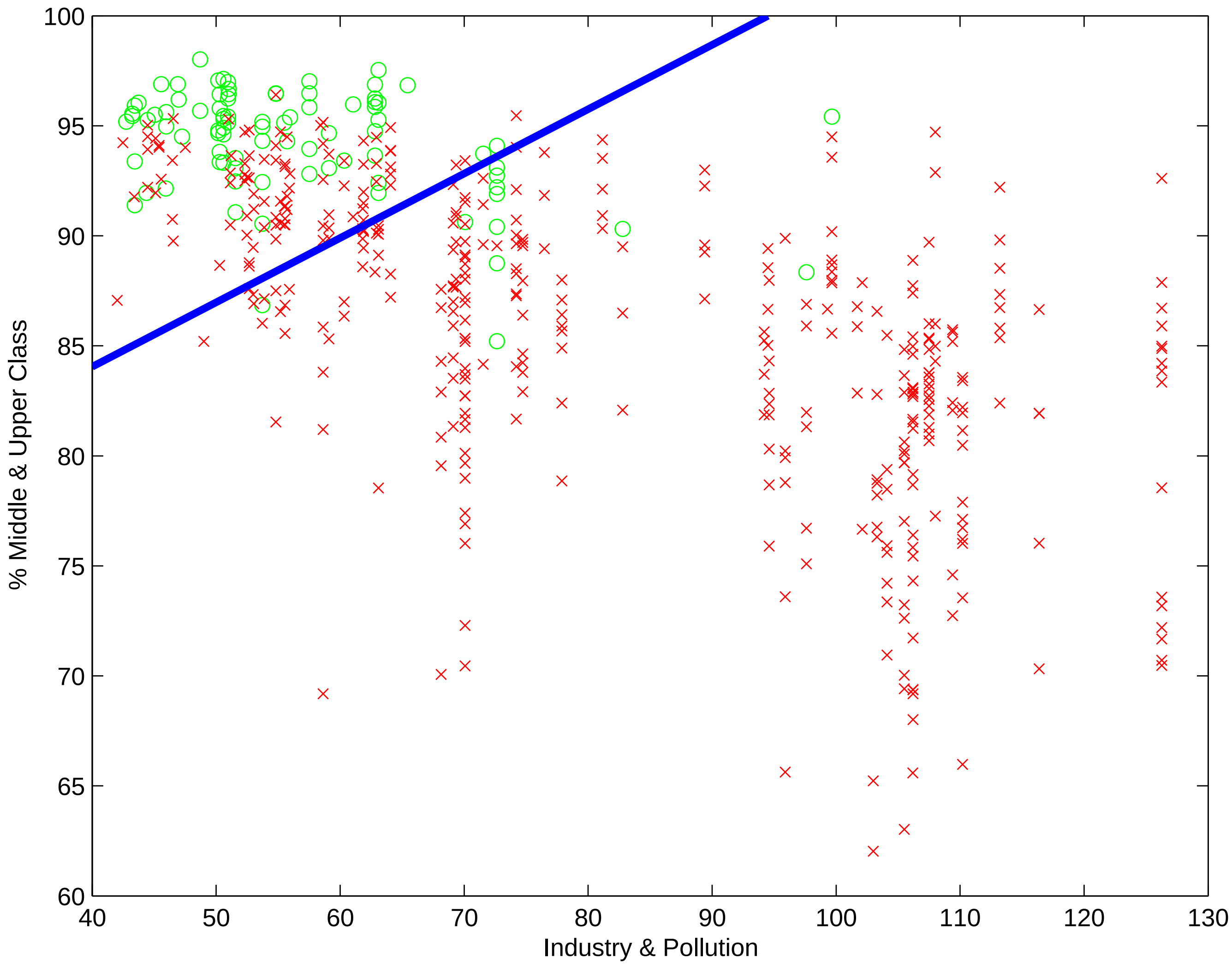
Measurements of petal length, width

Iris setosa is linearly separable from *I. versicolor* and *I. virginica*

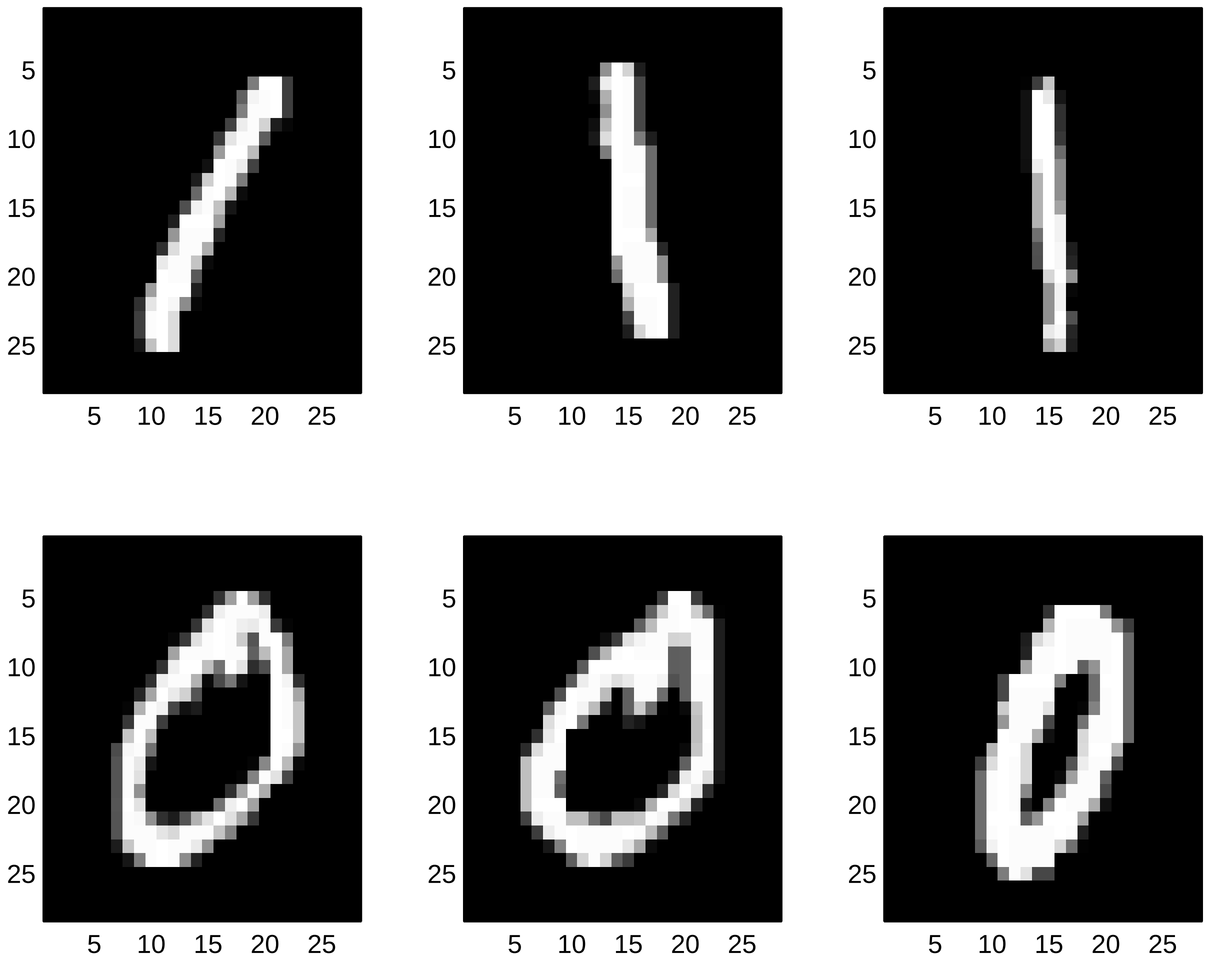
Example—Boston housing data



A good linear classifi er

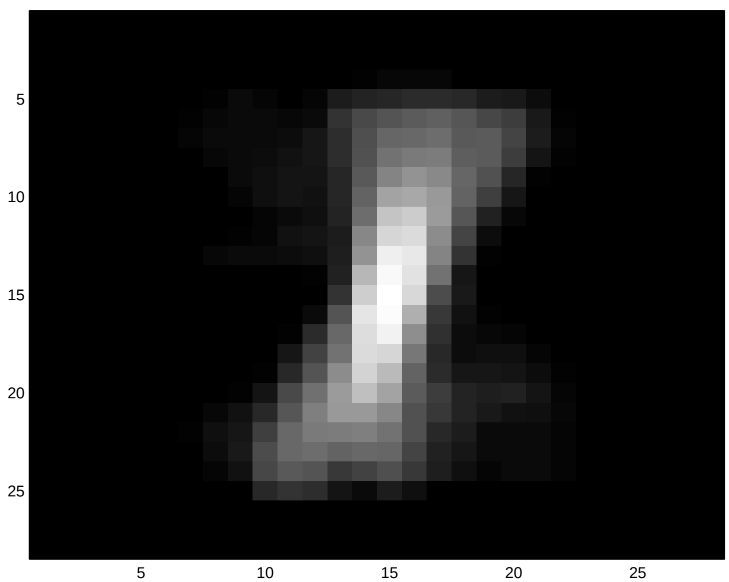


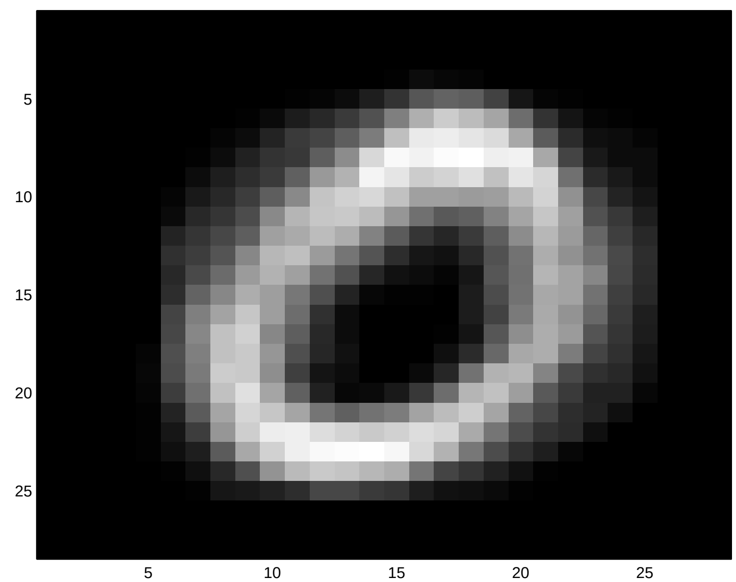
Example—NIST digits



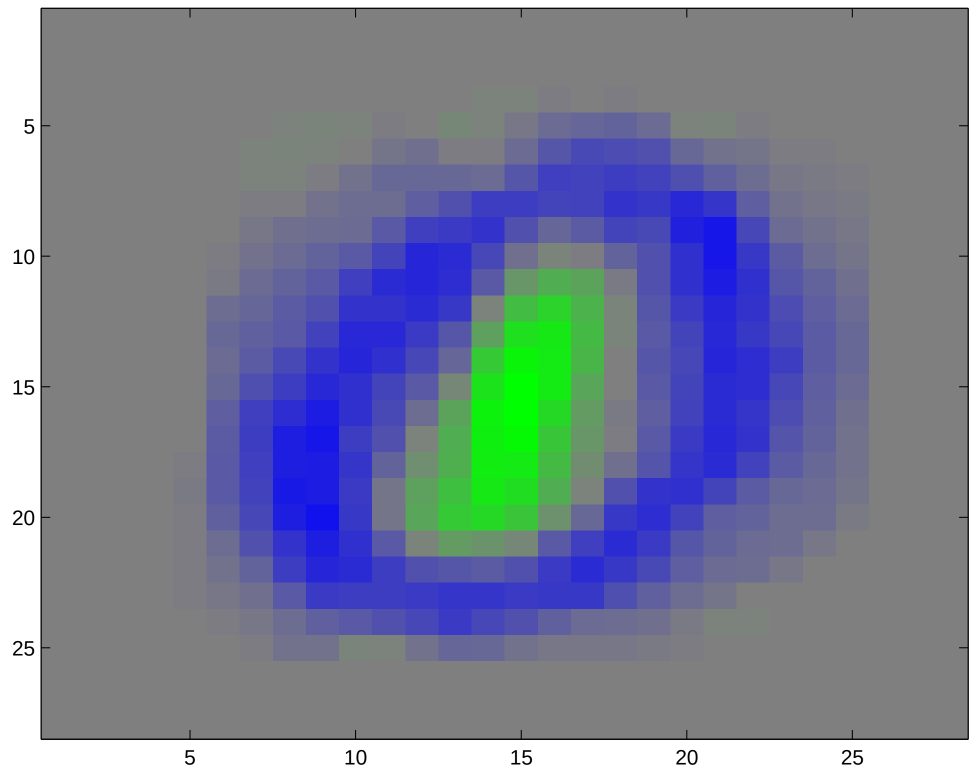
$$28 \times 28 = 784 \text{ features in } [0, 1]$$

Class means

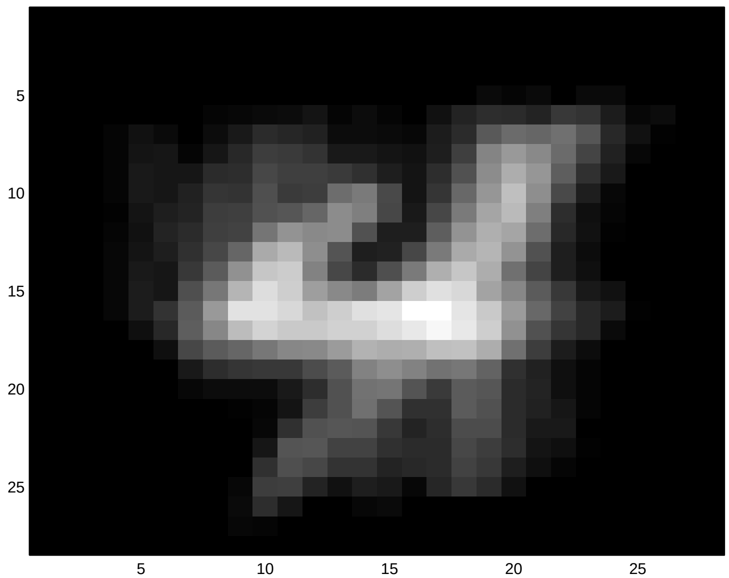


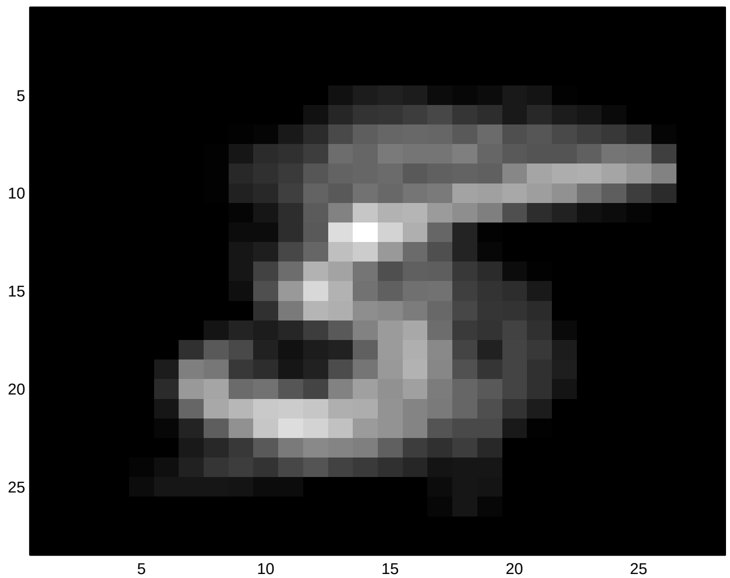


A linear separator

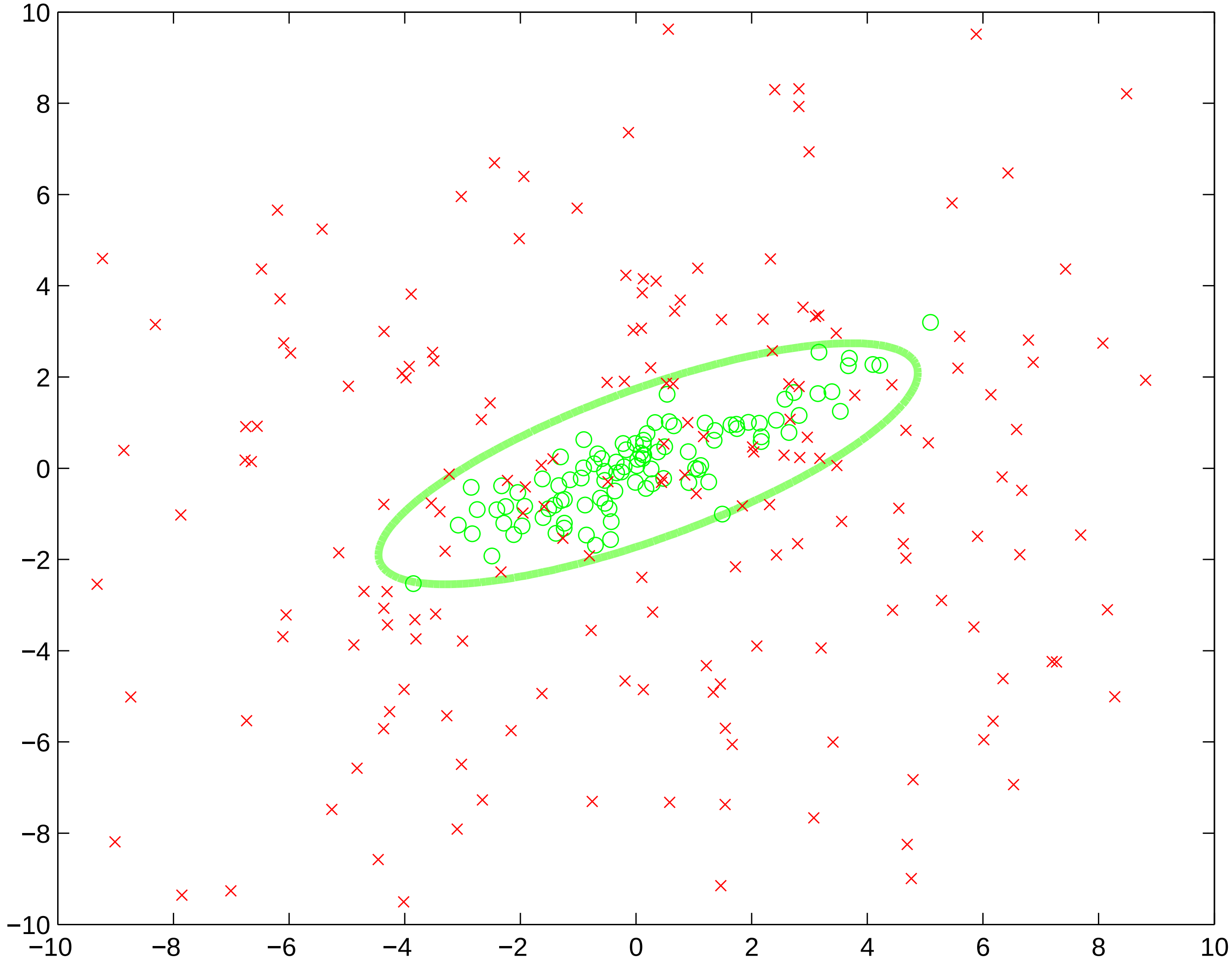


Sometimes a nonlinear classifier is better

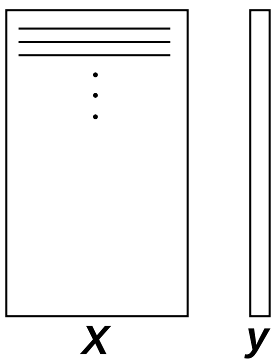


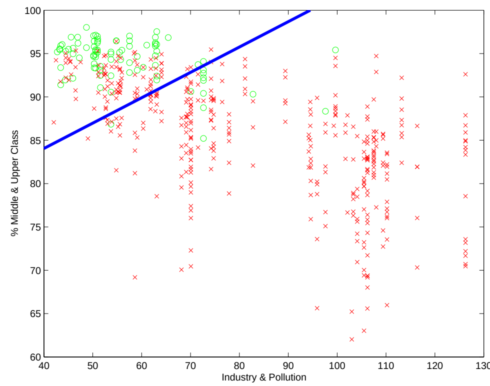


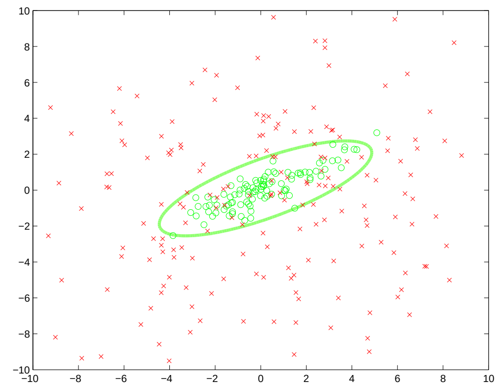
Sometimes a nonlinear classifi er is better



Classification problem







Data points $X = [\mathbf{x}_1; \mathbf{x}_2; \mathbf{x}_3; \ldots]$ with $\mathbf{x}_i \in \mathbb{R}^n$

Labels $\mathbf{y} = [y_1; y_2; y_3; \ldots]$ with $y_i \in \{-1, 1\}$

Solution is a subset of $\mathbb{R}^n$, the “classifier”

Often represented as a test $f(\mathbf{x}, \text{learnable parameters}) \geq 0$

Define: decision surface, linear separator, linearly separable

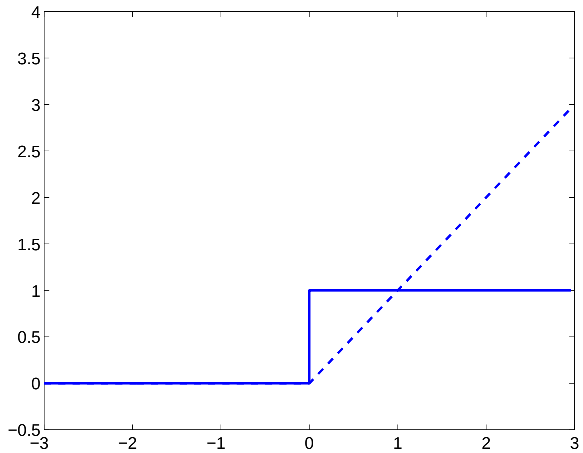
What is goal?

Classify new data with fewest possible mistakes

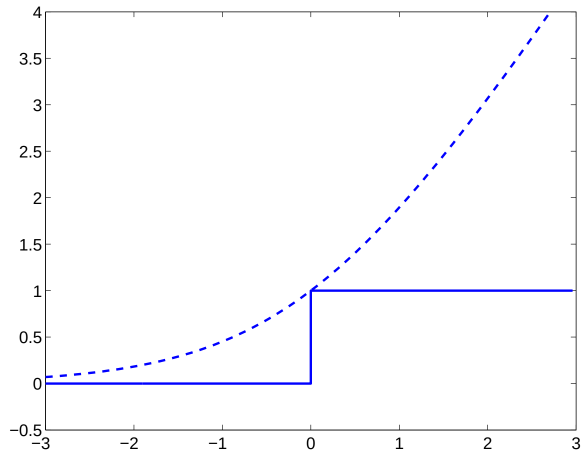
Proxy: minimize some function on training data

$$\min_{\mathbf{w}} \sum_i l(y_i f(\mathbf{x}_i; \mathbf{w})) + l_0(\mathbf{w})$$

That's $l(f(\mathbf{x}))$ for +ve examples, $l(-f(x))$ for -ve



piecewise linear loss



logistic loss

Getting fancy

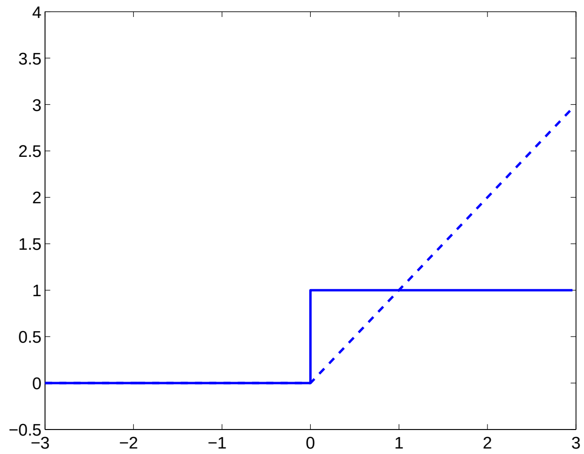
Text? Hyperlinks? Relational database records?

- difficult to featurize w/ reasonable number of features
- but what if we could handle large or infinite feature sets?

Outline

- Classification problems
- Perceptrons and convex programs
- From perceptrons to SVMs
- Advanced topics

Perceptrons



Weight vector $\mathbf{w}$, bias c

Classification rule: sign$(f(\mathbf{x}))$ where $f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + c$

Penalty for mispredicting: $l(yf(\mathbf{x})) = [yf(\mathbf{x})]_-$

This penalty is convex in $\mathbf{w}$, so all minima are global

Note: unit-length $\mathbf{x}$ vectors

Training perceptrons

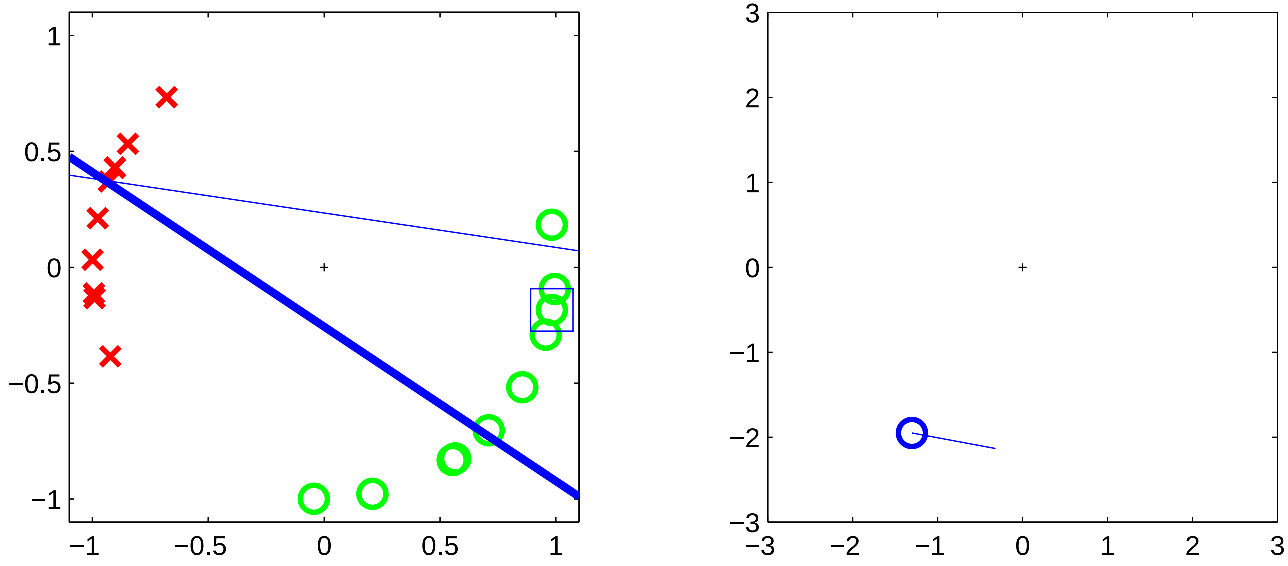
Perceptron learning rule: on mistake,

$$\begin{aligned} \mathbf{w} &\text{ += } y\mathbf{x} \\ c &\text{ += } y \end{aligned}$$

That is, gradient descent on $l(yf(\mathbf{x}))$, since

$$\nabla_w \left[y(\mathbf{x} \cdot \mathbf{w} + c) \right]_- = \begin{cases} -y\mathbf{x} & \text{if } y(\mathbf{x} \cdot \mathbf{w} + c) \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron demo



Perceptrons as linear inequalities

Linear inequalities (for separable case):

$$y(\mathbf{x} \cdot \mathbf{w} + c) > 0$$

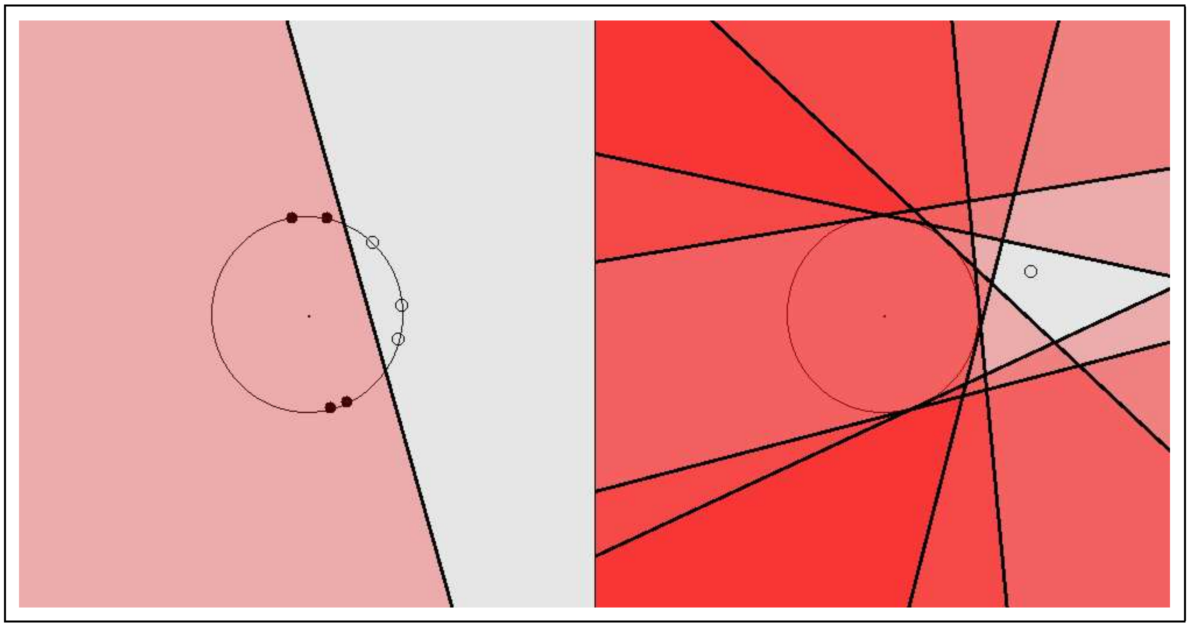
That's

$$\begin{array}{ll} \mathbf{x} \cdot \mathbf{w} + c > 0 & \text{for positive examples} \\ \mathbf{x} \cdot \mathbf{w} + c < 0 & \text{for negative examples} \end{array}$$

i.e., ensure correct classification of training examples

Note $>$ not $\geq$

Version space



$$\mathbf{x} \cdot \mathbf{w} + c = 0$$

As a fn of $\mathbf{x}$: hyperplane w/ normal $\mathbf{w}$ at distance $c/\|\mathbf{w}\|$ from origin

As a fn of $\mathbf{w}$: hyperplane w/ normal $\mathbf{x}$ at distance $c/\|\mathbf{x}\|$ from origin

Convex programs

Convex program:

$$\min\ f(\mathbf{x}) \text{ subject to}$$

$$g_i(\mathbf{x}) \le 0 \qquad i = 1 \ldots m$$

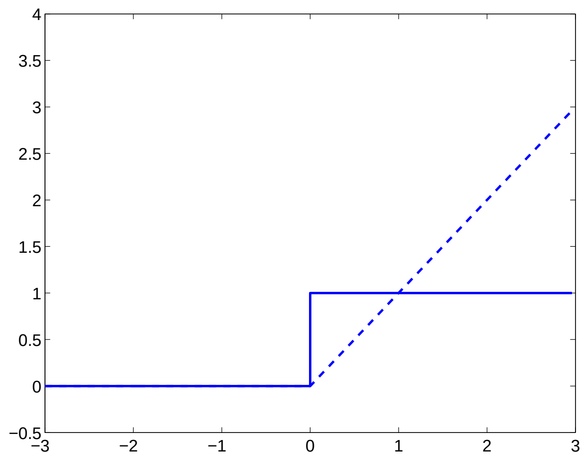
where f and g_i are convex functions

Perceptron is almost a convex program ($>$ vs. $\ge$)

Trick: write

$$y(\mathbf{x} \cdot \mathbf{w} + c) \ge 1$$

Slack variables



If not linearly separable, add slack variable $s \geq 0$

$$y(\mathbf{x} \cdot \mathbf{w} + c) + s \geq 1$$

Then $\sum_i s_i$ is total amount by which constraints are violated

So try to make $\sum_i s_i$ as small as possible

Perceptron as convex program

The final convex program for the perceptron is:

$$\min \ \sum_i s_i \ \text{ subject to}$$

$$(y_i \mathbf{x}_i) \cdot \mathbf{w} + y_i c + s_i \geq 1$$

$$s_i \geq 0$$

We will try to understand this program using convex duality

Duality

To every convex program corresponds a dual

Solving original (primal) is equivalent to solving dual

Dual often provides insight

Can derive dual by using Lagrange multipliers to eliminate constraints

Lagrange Multipliers

Way to phrase constrained optimization problem as a game

$\max_{\mathbf{x}} f(\mathbf{x})$ subject to $g(\mathbf{x}) \geq 0$

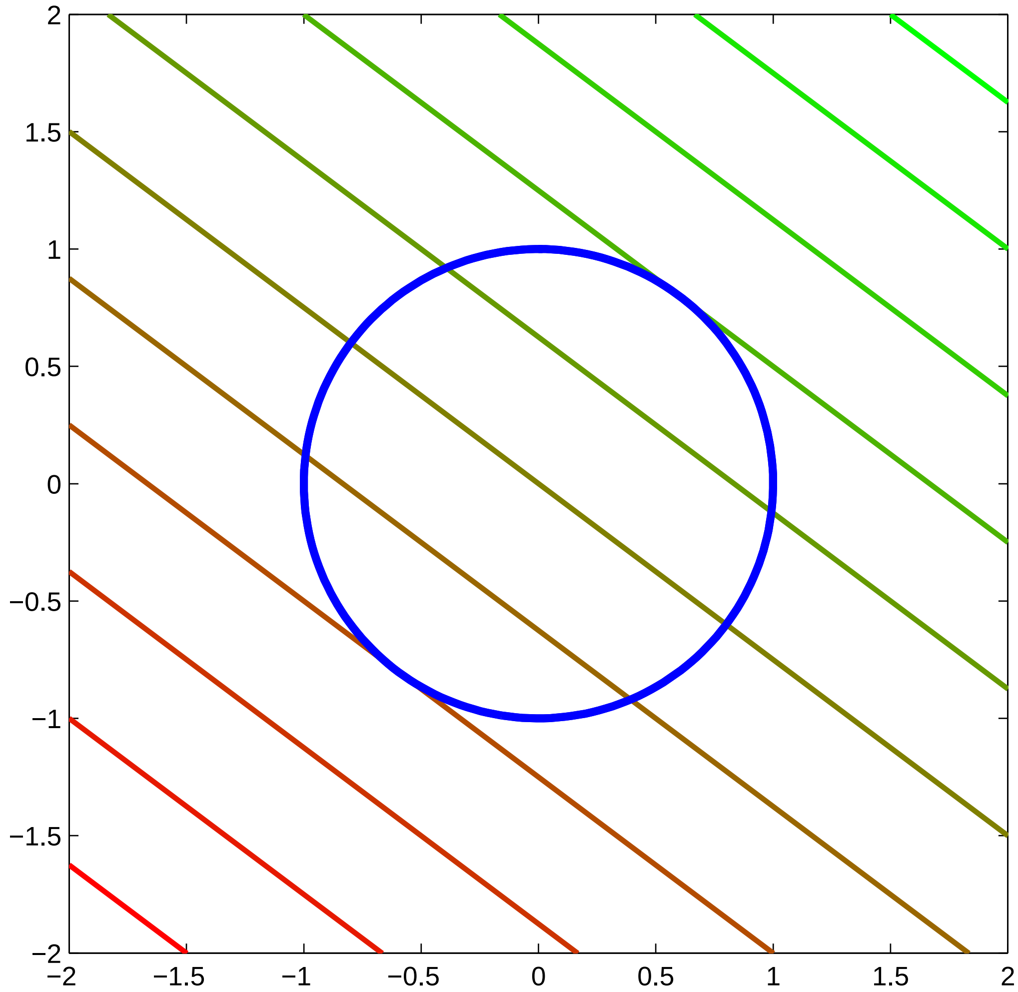
(assume f, g are convex downward)

$\max_{\mathbf{x}} \min_{a \geq 0} \quad f(\mathbf{x}) + a g(\mathbf{x})$

If $\mathbf{x}$ plays $g(\mathbf{x}) < 0$, then a wins: playing big numbers makes payoff approach $-\infty$

If $\mathbf{x}$ plays $g(\mathbf{x}) \geq 0$, then a must play 0

Lagrange Multipliers: the picture



Lagrange Multipliers: the caption

Problem: maximize

$$f(x,y) = 6x + 8y$$

subject to

$$g(x,y) = x^2 + y^2 - 1 \geq 0$$

Using a Lagrange multiplier a,

$$\max_{xy} \min_{a \geq 0} \quad f(x,y) + a g(x,y)$$

At optimum,

$$0 = \nabla f(x,y) + a \nabla g(x,y) = \begin{pmatrix} 6 \\ 8 \end{pmatrix} + 2a \begin{pmatrix} x \\ y \end{pmatrix}$$

Duality for the perceptron

Notation: $\mathbf{z}_i = y_i \mathbf{x}_i$ and $Z = [\mathbf{z}_1; \mathbf{z}_2; \ldots]$, so that:

$$\min_{s,\mathbf{w},c} \ \textstyle\sum_i s_i \ \text{ subject to}$$

$$Z\mathbf{w} + c\mathbf{y} + \mathbf{s} \geq \mathbf{1}$$

$$\mathbf{s} \geq 0$$

Using a Lagrange multiplier vector $\mathbf{a} \geq 0$,

$$\min_{s,\mathbf{w},c} \ \max_{\mathbf{a}} \ \textstyle\sum_i s_i - \mathbf{a}^\top (Z\mathbf{w} + c\mathbf{y} + \mathbf{s} - \mathbf{1})$$

subject to $\mathbf{s} \geq 0 \ \ \mathbf{a} \geq 0$

Duality cont'd

From last slide:

$$\min_{s,\mathbf{w},c} \ \max_{\mathbf{a}} \ \sum_i s_i - \mathbf{a}^\top (Z\mathbf{w} + c\mathbf{y} + \mathbf{s} - \mathbf{1})$$

subject to $\mathbf{s} \geq 0 \ \ \mathbf{a} \geq 0$

Minimize wrt $\mathbf{w}, c$ explicitly by setting gradient to 0:

$$0 = \mathbf{a}^\top Z$$
$$0 = \mathbf{a}^\top \mathbf{y}$$

Minimizing wrt s yields inequality:

$$0 \leq \mathbf{1} - \mathbf{a}$$

Duality cont'd

Final form of dual program for perceptron:

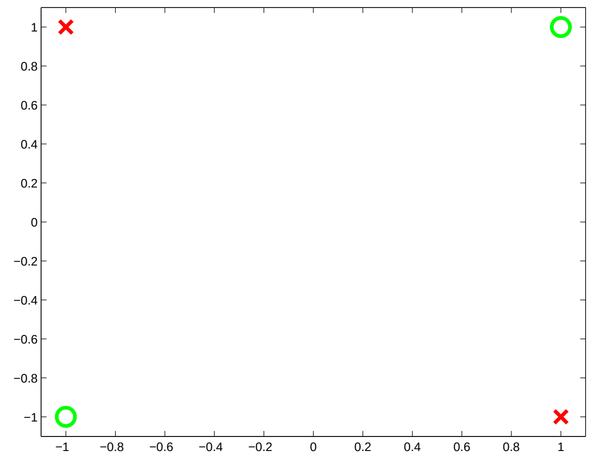
$$\max_{\mathbf{a}} \ \mathbf{1}^\top \mathbf{a} \ \text{ subject to}$$

$$0 = \mathbf{a}^\top Z$$

$$0 = \mathbf{a}^\top \mathbf{y}$$

$$0 \leq \mathbf{a} \leq 1$$

Problems with perceptrons



Vulnerable to overfitting when many input features

Not very expressive (XOR)

Outline

- Classification problems
- Perceptrons and convex programs
- From perceptrons to SVMs
- Advanced topics

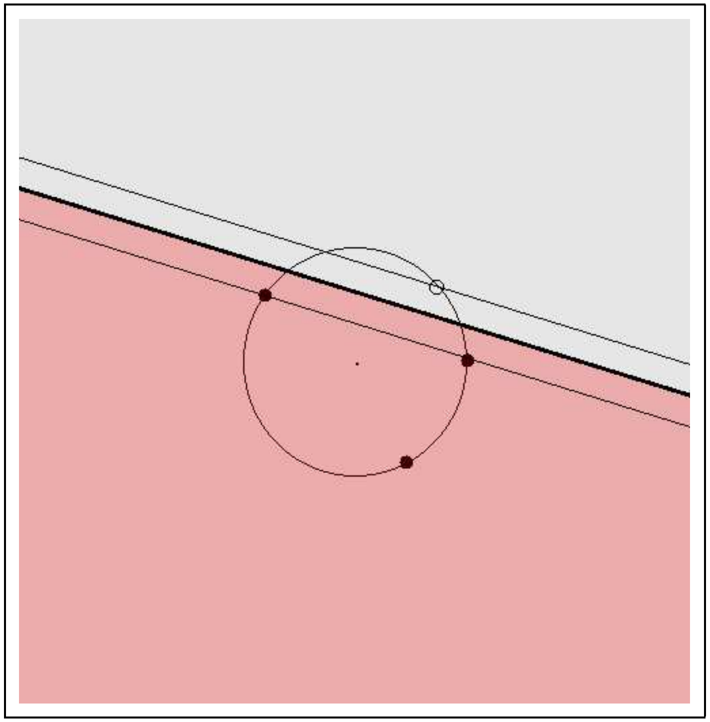
Modernizing the perceptron

Three extensions:

- Margins
- Feature expansion
- Kernel trick

Result is called a Support Vector Machine (reason given below)

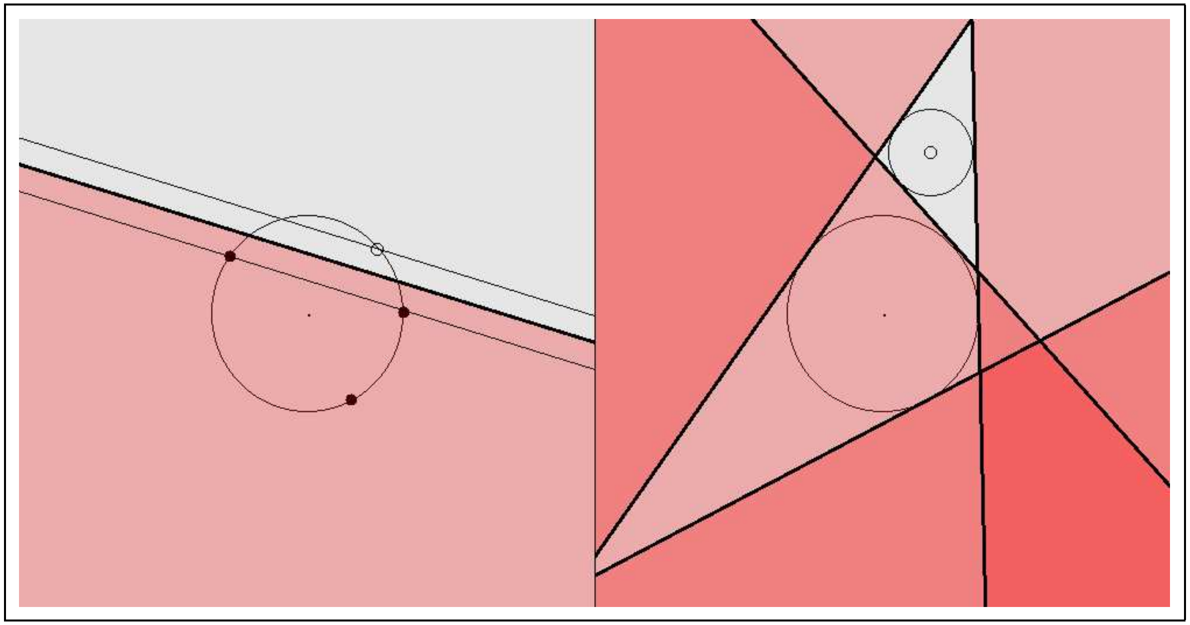
Margins



Margin is the signed $\perp$ distance from an example to the decision boundary

+ve margin points are correctly classified, -ve margin means error

SVMs are maximum margin



Maximize minimum distance from data to separator

“Ball center” of version space (caveats)

Other centers: analytic center, center of mass, Bayes point

Note: if not linearly separable, must trade margin vs. errors

Why do margins help?

If our hypothesis is near the boundary of decision space, we don’t necessarily learn much from our mistakes

If we’re far away from any boundary, a mistake has to eliminate a large volume from version space

Why margins help, explanation 2

Occam's razor: "simple" classifiers are likely to do better in practice

Why? There are fewer simple classifiers than complicated ones, so we are less likely to be able to fool ourselves by finding a really good fit by accident.

What does "simple" mean? Anything, as long as you tell me before you see the data.

Explanation 2 cont'd

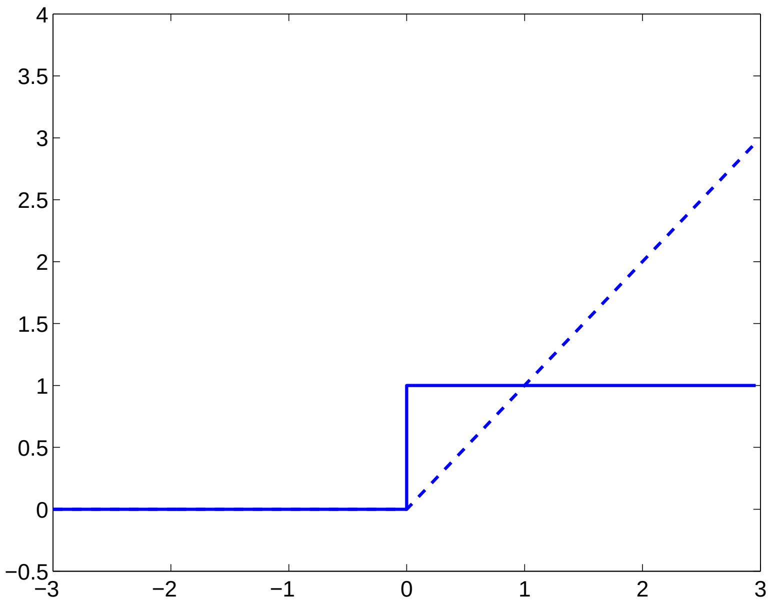
"Simple" can mean:

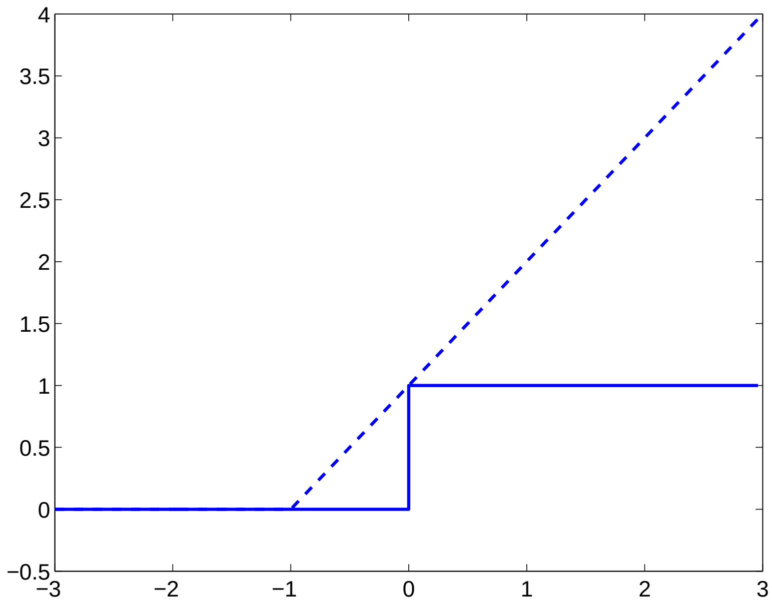
- Low-dimensional
- Large margin
- Short description length

For this lecture we are interested in large margins and compact descriptions

By contrast, many classical complexity control methods (AIC, BIC) rely on low dimensionality alone

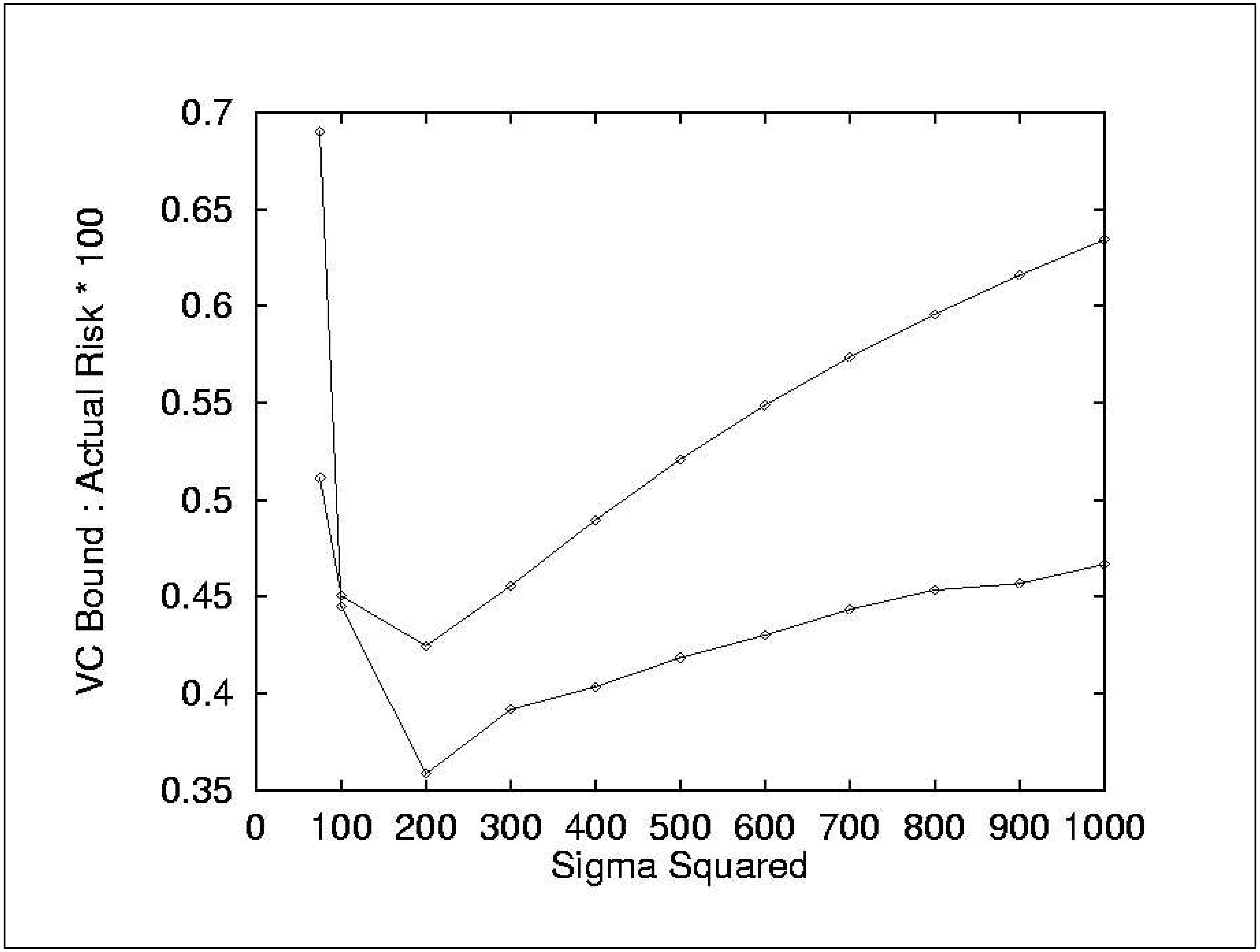
Why margins help, explanation 3





Margin loss is an upper bound on number of mistakes

Why margins help, explanation 4



Optimizing the margin

Most common method: convex quadratic program

Efficient algorithms exist (essentially the same as some interior point LP algorithms)

Because QP is strictly* convex, *unique* global optimum

Next few slides derive the QP. Notation:

- Assume w.l.o.g. $\|\mathbf{x}_i\|_2 = 1$
- Ignore slack variables for now (i.e., assume linearly separable)

*if you ignore the intercept term

Optimizing the margin, cont'd

Margin M is $\perp$ distance to decision surface: for pos example,

$$\begin{aligned}(\mathbf{x} - M\mathbf{w}/\|\mathbf{w}\|) \cdot \mathbf{w} + c &= 0 \\ \mathbf{x} \cdot \mathbf{w} + c &= M\mathbf{w} \cdot \mathbf{w}/\|\mathbf{w}\| = M\|\mathbf{w}\|\end{aligned}$$

SVM maximizes $M > 0$ such that all margins are $\geq M$:

$\max_{M>0,\mathbf{w},c}\ M$ subject to

$(y_i \mathbf{x}_i) \cdot \mathbf{w} + y_i c \geq M\|\mathbf{w}\|$

Notation: $\mathbf{z}_i = y_i \mathbf{x}_i$ and $Z = [\mathbf{z}_1; \mathbf{z}_2; \ldots]$, so that:

$$Z\mathbf{w} + \mathbf{y}c \geq M\|\mathbf{w}\|$$

Note $\lambda w, \lambda c$ is a solution if w, c is

Optimizing the margin, cont'd

Divide by $M\|\mathbf{w}\|$ to get $(Z\mathbf{w} + \mathbf{y}c)/M\|\mathbf{w}\| \geq 1$

Define $\mathbf{v} = \frac{\mathbf{w}}{M\|\mathbf{w}\|}$ and $d = \frac{c}{M\|\mathbf{w}\|}$, so that $\|\mathbf{v}\| = \frac{\|\mathbf{w}\|}{M\|\mathbf{w}\|} = \frac{1}{M}$

$\max_{\mathbf{v},d}\ 1/\|\mathbf{v}\|$ subject to
$Z\mathbf{v} + \mathbf{y}d \geq 1$

Maximizing $1/\|\mathbf{v}\|$ is minimizing $\|\mathbf{v}\|$ is minimizing $\|\mathbf{v}\|^2$

$\min_{\mathbf{v},d}\ \|\mathbf{v}\|^2$ subject to
$Z\mathbf{v} + \mathbf{y}d \geq 1$

Add slack variables to handle non-separable case:

$\min_{\mathbf{s}\geq 0,\mathbf{v},d}\ \|\mathbf{v}\|^2 + C\sum_i s_i$ subject to
$Z\mathbf{v} + \mathbf{y}d + \mathbf{s} \geq 1$

Modernizing the perceptron

Three extensions:

- Margins
- Feature expansion
- Kernel trick

Feature expansion

Given an example $\mathbf{x} = [a\ b\ ...]$

Could add new features like a^2, ab, a^7b^3, $\sin(b)$, ...

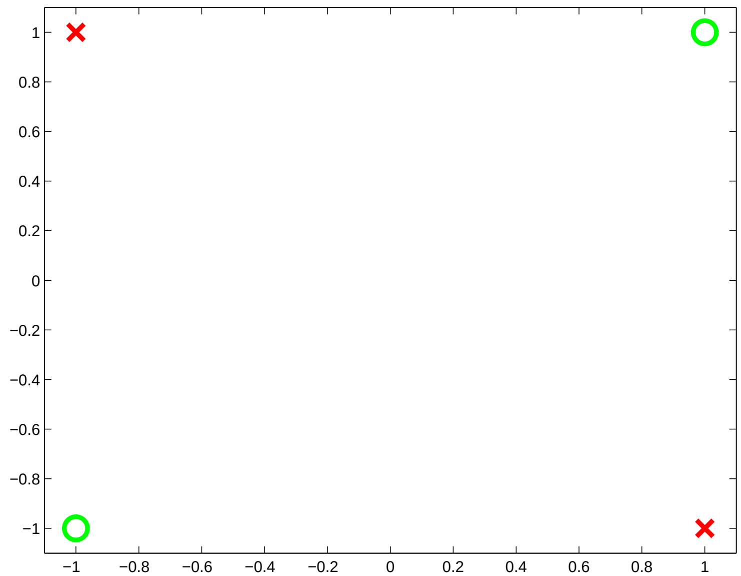
Same optimization as before, but with longer $\mathbf{x}$ vectors and so longer $\mathbf{w}$ vector

Classifier: "is $3a + 2b + a^2 + 3ab - a^7b^3 + 4\sin(b) \geq 2.6$?"

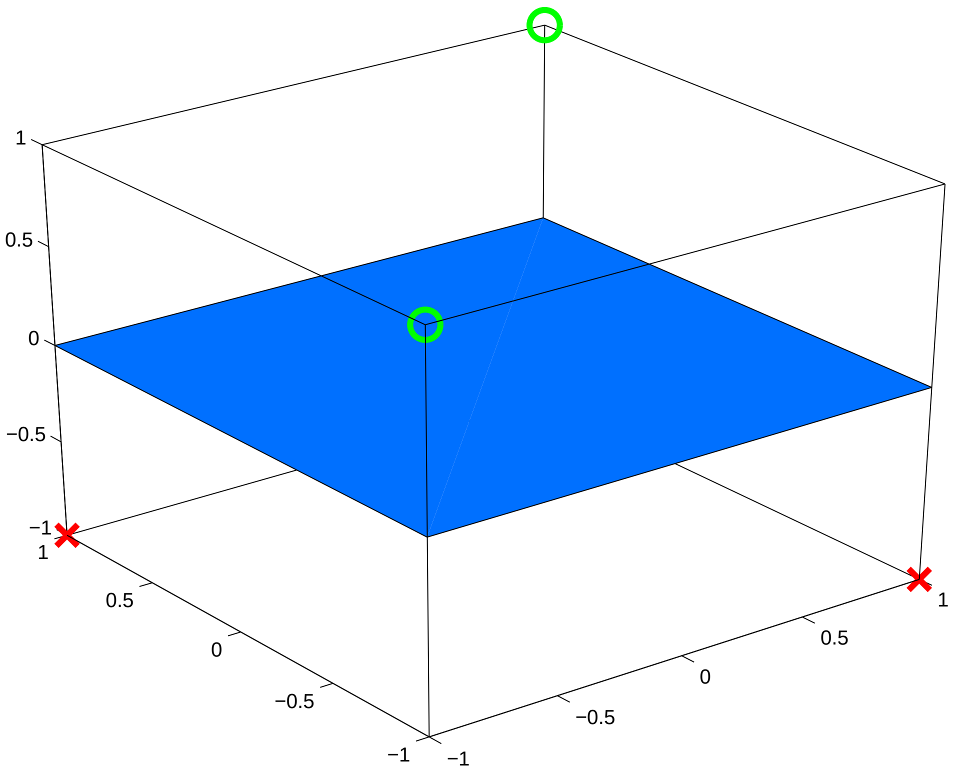
This classifier is nonlinear in original features, but linear in expanded feature space

We have replaced $\mathbf{x}$ by $\phi(\mathbf{x})$ for some nonlinear ϕ, so decision boundary is nonlinear surface $\mathbf{w} \cdot \phi(\mathbf{x}) + c = 0$

Feature expansion example



x, y



x, y, xy

Some popular feature sets

Polynomials of degree k

$$1, a, a^2, b, b^2, ab$$

Neural nets (sigmoids)

$$\tanh(3a + 2b - 1), \tanh(5a - 4), \ldots$$

RBFs of radius σ

$$\exp\left(-\frac{1}{\sigma^2}((a - a_0)^2 + (b - b_0)^2)\right)$$

Feature expansion problems

Feature expansion techniques yield *lots* of features

E.g. polynomial of degree k on n original features yields $O(n^k)$ expanded features

E.g. RBFs yield infinitely many expanded features!

Inefficient (`for i = 1 to infinity do ...`)

Overfitting (VC-dimension argument)

How to fi x feature expansion

We have already shown we can handle the overfitting problem: even if we have lots of parameters, large margins make simple classifiers

“All” that’s left is efficiency

Solution: kernel trick

Modernizing the perceptron

Three extensions:

- Margins
- Feature expansion
- Kernel trick

Kernel trick

Way to make optimization efficient when there are lots of features

Compute one Lagrange multiplier per training example instead of one weight per feature (part I)

Use kernel function to avoid representing w ever (part II)

Will mean we can handle infinitely many features!

Kernel trick, part I

$\min_{\mathbf{w},c} \quad |\mathbf{w}|^2/2 \quad \text{subject to} \quad Z\mathbf{w} + \mathbf{y}c \geq 1$

$\min_{\mathbf{w},c} \max_{\mathbf{a}\geq 0} \quad \mathbf{w}^\top \mathbf{w}/2 + \mathbf{a} \cdot (1 - Z\mathbf{w} - \mathbf{y}c)$

Minimize wrt $\mathbf{w}, c$ by setting derivatives to 0

$0 = \mathbf{w} - Z^\top \mathbf{a} \qquad 0 = \mathbf{a} \cdot \mathbf{y}$

Substitute back in for $\mathbf{w}, c$

$\max_{\mathbf{a}\geq 0} \quad \mathbf{a} \cdot 1 - \mathbf{a}^\top Z Z^\top \mathbf{a}/2 \quad \text{subject to} \quad \mathbf{a} \cdot \mathbf{y} = 0$

Note: to allow slacks, add an upper bound $\mathbf{a} \leq C$

What did we just do?

$\max_{0 \leq \mathbf{a} \leq C} \quad \mathbf{a} \cdot 1 - \mathbf{a}^\top Z Z^\top \mathbf{a}/2$ subject to $\mathbf{a} \cdot \mathbf{y} = 0$

Now we have a QP in $\mathbf{a}$ instead of $\mathbf{w}, c$

Once we solve for $\mathbf{a}$, we can find $\mathbf{w} = Z^\top \mathbf{a}$ to use for classification

We also need c which we can get from complementarity:

$$y_i \mathbf{x}_i \cdot \mathbf{w} + y_i c = 1 \qquad \Leftrightarrow \qquad a_i > 0$$

or as dual variable for $\mathbf{a} \cdot \mathbf{y} = 0$

Representation of $\mathbf{w}$

Optimal $\mathbf{w} = Z^\top \mathbf{a}$ is a linear combination of rows of Z

I.e., $\mathbf{w}$ is a linear combination of (signed) training examples

I.e., $\mathbf{w}$ has a finite representation even if there are infinitely many features

Support vectors

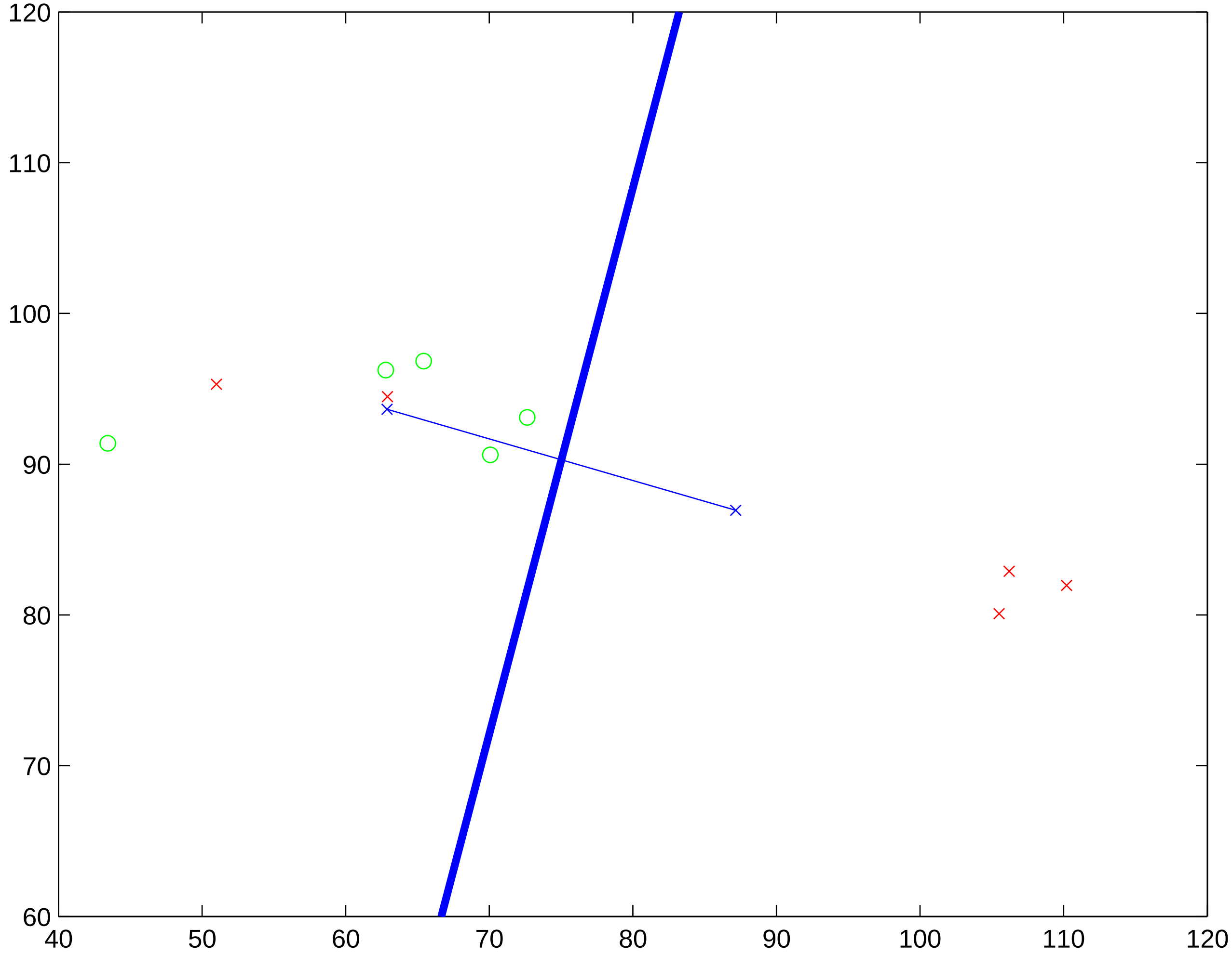
Examples with $a_i > 0$ are called support vectors

"Support vector machine" = learning algorithm ("machine") based on support vectors

Often many fewer than number of training examples ($\mathbf{a}$ is sparse)

This is the "short description" of an SVM mentioned above

Intuition for support vectors



Partway through optimization

Suppose we have 5 positive support vectors and 5 negative, all with equal weights

Best $\mathbf{w}$ so far is $Z^\top \mathbf{a}$: diff between mean of +ve SVs, mean of -ve

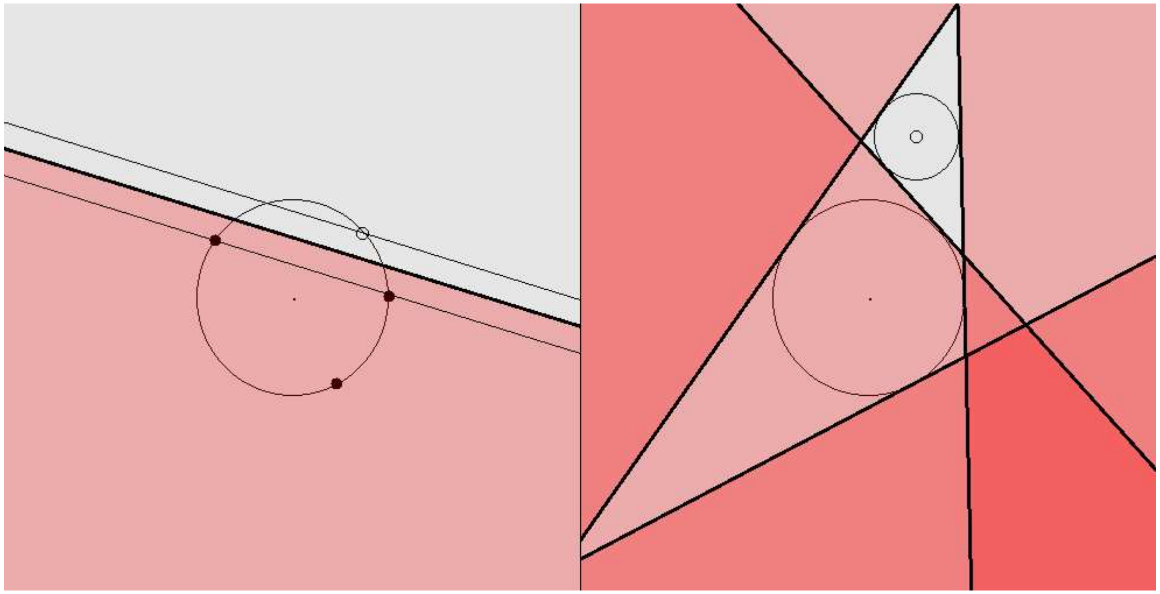
Averaging $\mathbf{x}_i \cdot \mathbf{w} + c = y_i$ for all SVs yields

$$c = -\bar{\mathbf{x}} \cdot \mathbf{w}$$

That is,

- Compute mean of +ve SVs, mean of -ve SVs
- Draw the line between means, and its perpendicular bisector
- This bisector is current classifier

At end of optimization



Gradient wrt a_i is $1 - y_i(\mathbf{x}_i \cdot \mathbf{w} + c)$

Increase a_i if (scaled) margin < 1, decrease if margin > 1

Stable iff ($a_i = 0$ AND margin ≥ 1) OR margin $= 1$

How to avoid writing down weights

Suppose number of features is really big or even infinite?

Can't write down X, so how do we solve the QP?

Can't even write down $\mathbf{w}$, so how do we classify new examples?

Solving the QP

$\max_{0 \leq \mathbf{a} \leq C} \quad \mathbf{a} \cdot 1 - \mathbf{a}^\top Z Z^\top \mathbf{a}/2 \quad$ subject to $\quad \mathbf{a} \cdot \mathbf{y} = 0$

Write $G = ZZ^\top$ (called Gram matrix)

That is, $G_{ij} = y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$

$\max_{0 \leq \mathbf{a} \leq C} \quad \mathbf{a} \cdot 1 - \mathbf{a}^\top G \mathbf{a}/2 \quad$ subject to $\quad \mathbf{a} \cdot \mathbf{y} = 0$

With m training examples, G is $m \times m$ —(often) small enough for us to solve the QP even if we can't write down X

Can we compute G directly, without computing X?

Kernel trick, part II

Yes, we can compute G directly—sometimes!

Recall that $\mathbf{x}_i$ was the result of applying a nonlinear feature expansion function ϕ to some shorter vector (say $\mathbf{q}_i$)

Define $K(\mathbf{q}_i, \mathbf{q}_j) = \phi(\mathbf{q}_i) \cdot \phi(\mathbf{q}_j)$

Mercer kernels

K is called a (Mercer) kernel function

Satisfies Mercer condition: $K(\mathbf{q}, \mathbf{q}') \geq 0$

Mercer condition for a function K is analogous to nonnegative definiteness for a matrix

In many cases there is a simple expression for K even if there isn't one for ϕ

In fact, it sometimes happens that we know K without knowing ϕ

Example kernels

Polynomial (typical component of ϕ might be $17q_1^2q_2^3q_4$)

$$K(\mathbf{q}, \mathbf{q}') = (1 + \mathbf{q} \cdot \mathbf{q}')^k$$

Sigmoid (typical component $\mathtt{tanh}(q_1 + 3q_2)$)

$$K(\mathbf{q}, \mathbf{q}') = \mathtt{tanh}(a\mathbf{q} \cdot \mathbf{q}' + b)$$

Gaussian RBF (typical component $\exp(-\frac{1}{2}(q_1 - 5)^2)$)

$$K(\mathbf{q}, \mathbf{q}') = \exp(-\|\mathbf{q} - \mathbf{q}'\|^2/\sigma^2)$$

Detail: polynomial kernel

Suppose $\mathbf{x} = \begin{pmatrix} 1 \\ \sqrt{2}q \\ q^2 \end{pmatrix}$

Then $\mathbf{x}' \cdot \mathbf{x} = 1 + 2qq' + q^2(q')^2$

From previous slide,

$$K(q, q') = (1 + qq')^2 = 1 + 2qq' + q^2(q')^2$$

Dot product + addition + exponentiation vs. $O(n^k)$ terms

The new decision rule

Recall original decision rule: sign$(\mathbf{x} \cdot \mathbf{w} + c)$

Use representation in terms of support vectors:

$$\text{sign}(\mathbf{x} \cdot Z^{\top} \mathbf{a} + c) = \text{sign}\left(\sum_i \mathbf{x} \cdot \mathbf{x}_i y_i a_i + c\right) = \text{sign}\left(\sum_i K(\mathbf{q}, \mathbf{q}_i) y_i a_i + c\right)$$

Since there are usually not too many support vectors, this is a reasonably fast calculation

Summary of SVM algorithm

Training:

- Compute Gram matrix $G_{ij} = y_i y_j K(\mathbf{q}_i, \mathbf{q}_j)$
- Solve QP to get $\mathbf{a}$
- Compute intercept c by using complementarity or duality

Classification:

- Compute $k_i = K(\mathbf{q}, \mathbf{q}_i)$ for support vectors $\mathbf{q}_i$
- Compute $f = c + \sum_i a_i k_i y_i$
- Test sign(f)

Outline

- Classification problems
- Perceptrons and convex programs
- From perceptrons to SVMs
- Advanced topics

Advanced kernels

All problems so far: each example is a list of numbers

What about text, relational DBs, . . . ?

Insight: $K(x, y)$ can be defined when x and y are not fixed length

Examples:

- String kernels
- Path kernels
- Tree kernels
- Graph kernels

String kernels

Pick $\lambda \in (0, 1)$

$\texttt{cat} \mapsto \texttt{c, a, t,}\ \lambda\ \texttt{ca,}\ \lambda\ \texttt{at,}\ \lambda^2\ \texttt{ct,}\ \lambda^2\ \texttt{cat}$

Strings are similar if they share lots of nearly-contiguous substrings

Works for words in phrases too: “man bites dog” similar to “man bites hot dog,” less similar to “dog bites man”

There is an efficient dynamic-programming algorithm to evaluate this kernel (Lodhi et al, 2002)

Combining kernels

Suppose $K(x, y)$ and $K'(x, y)$ are kernels

Then so are

- $K + K'$
- αK for $\alpha > 0$

Given a set of kernels $K_1, K_2, \ldots$, can search for best

$$K = \alpha_1 K_1 + \alpha_2 K_2 + \ldots$$

using cross-validation, etc.

“Kernel X”

Kernel trick isn’t limited to SVDs

Works whenever we can express an algorithm using only sums, dot products of training examples

Examples:

- kernel Fisher discriminant
- kernel logistic regression
- kernel linear and ridge regression
- kernel SVD or PCA
- 1-class learning / density estimation

Summary

Perceptrons are a simple, popular way to learn a classifier

They suffer from inefficient use of data, overfitting, and lack of expressiveness

SVMs fix these problems using margins and feature expansion

In order to make feature expansion computationally feasible, we need the kernel trick

Kernel trick avoids writing out high-dimensional feature vectors by use of Lagrange multipliers and representer theorem

SVMs are popular classifiers because they usually achieve good error rates and can handle unusual types of data

References

`http://www.cs.cmu.edu/~ggordon/SVMs`
these slides, together with code

`http://svm.research.bell-labs.com/SVMdoc.html`
Burges's SVM tutorial

`http://citeseer.nj.nec.com/burges98tutorial.html`
Burges's paper "A Tutorial on Support Vector Machines for Pattern Recognition" (1998)

References

Huma Lodhi, Craig Saunders, Nello Cristianini, John Shawe-Taylor, Chris Watkins. “Text Classification using String Kernels.” 2002.

`http://www.cs.rhbnc.ac.uk/research/compint/areas/comp_learn/sv/pub/slide1.ps`
Slides by Stitson & Weston

`http://oregonstate.edu/dept/math/CalculusQuestStudyGuides/vcalc/lagrang/lagrang.html`
Lagrange Multipliers

`http://svm.research.bell-labs.com/SVT/SVMsvt.html`
on-line SVM applet