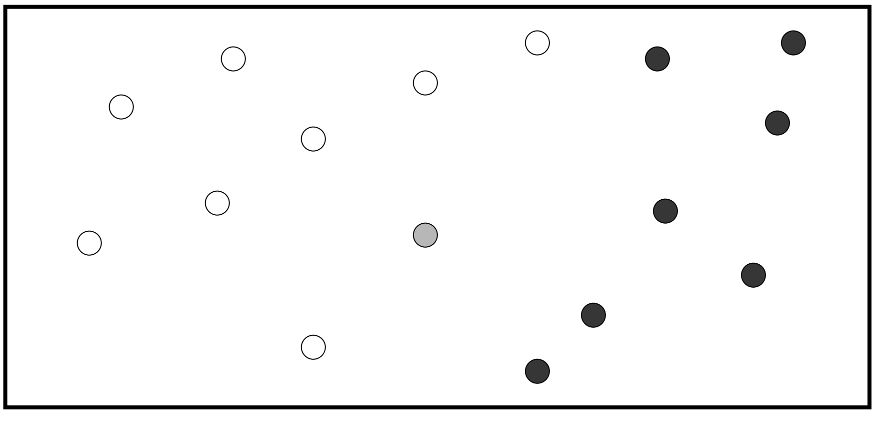
Announcements

- See Chapter 5 of Duda, Hart, and Stork.
- Tutorial by Burge linked to on web page.

Supervised Learning

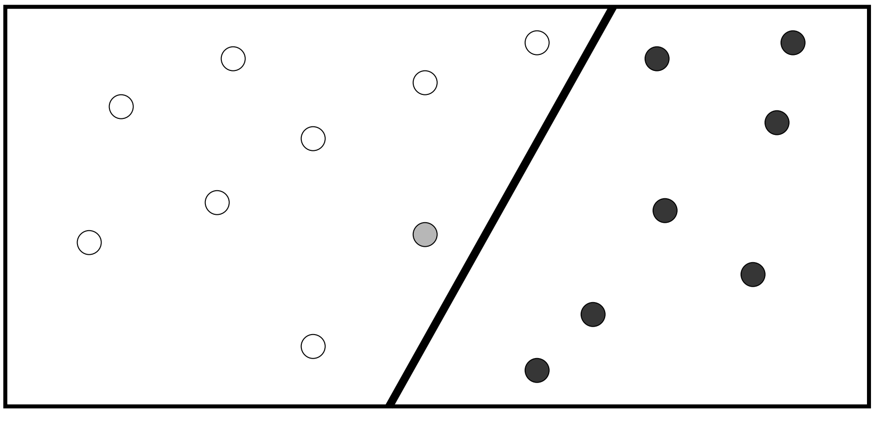
- Classification with labeled examples.
- Images → vectors in high-D space.



Supervised Learning

- Labeled examples called *training* set.
- Query examples called *test* set.
- Training and test set must come from same distribution.

Linearly Separable Classes



Linear Discriminants

- Images represented as vectors, $\mathbf{x}_1$, $\mathbf{x}_2$, ….
  - These could be pixels
  - But could also be *any* features
- Use these to find hyperplane defined by vector w and w_0.

$\mathbf{x}$ is on hyperplane: $\mathbf{w}^T\mathbf{x} + w_0 = 0$.

- Notation: $\mathbf{a}^\mathbf{T} = [w_0, w_1, \ldots]$. $y^T = [1, x_1, x_2, \ldots]$

So hyperplane is $\mathbf{a}^\mathbf{T}\mathbf{y}=0$.

- A query, *q,* is classified based on whether

$\mathbf{a}^T q > 0$ or $\mathbf{a}^T q < 0$.

Why linear discriminants?

- Optimal if classes are Gaussian with same covariances.
- Linear separators easier to find.
- Hyperplanes have few parameters, prevents overfitting.
  - Have low VC dimension.
- *Linear* may seem like a big limitation
  - But it’s not if the features are complex enough

Example

- XOR. An object is a 2D binary vector (x,y).
- Class is xor(x,y) (ie., (1,0) & (0,1)).
- Not linearly separable in (x,y) space.
- But is linearly separable in (x,y,x*x,y*y,x*y) space
  - x*x + y*y – 2*x*y > 0

Example: Naïve Bayes

- Assume all features are independent.
- Build optimal Bayesian classifier.
- For binary features, two classes, this produces a linear classifier.

$$P(\omega_1 \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid \omega_1)P(\omega_1)}{P(\mathbf{x})} = \frac{P(\mathbf{x} \mid \omega_1)P(\omega_1)}{P(\mathbf{x} \mid \omega_1)P(\omega_1) + P(\mathbf{x} \mid \omega_2)P(\omega_2)}$$

$$\text{Choose } \omega_1 \text{ when}: \frac{P(\mathbf{x} \mid \omega_1)P(\omega_1)}{P(\mathbf{x} \mid \omega_2)P(\omega_2)} > 1$$

$$\text{Independence} \rightarrow P(\mathbf{x} \mid \omega_j) = \prod_{i=1}^{d} P(x_i \mid \omega_j)$$

$$\text{Define}: p_i = P(x_i = 1 \mid \omega_1) \quad q_i = P(x_i = 1 \mid \omega_2)$$

$$\frac{P(\mathbf{x} \mid \omega_1)P(\omega_1)}{P(\mathbf{x} \mid \omega_2)P(\omega_2)} = \prod_{i=1}^{d} \left(\frac{p_i}{q_i}\right)^{x_i} \left(\frac{1-p_i}{1-q_i}\right)^{1-x_i} \frac{P(\omega_1)}{P(\omega_2)}$$

$$\text{Choose } \omega_1 : \sum_{\mathrm{i}=1}^{\mathrm{d}} \mathrm{x}_\mathrm{i} \ln \frac{p_i}{q_i} + (1-x_i) \ln \frac{1-p_i}{1-q_i} + \ln \frac{P(\omega_1)}{P(\omega_2)} > 0$$

Linearly Separable Classes

- For one set of classes, $\mathbf{a}^T\mathbf{y} > 0$. For other set: $\mathbf{a}^T\mathbf{y} < 0$.
- Notationally convenient if, for second class, make **y** negative.
- Then, finding a linear separator means finding **a** such that, for all *i,*

$\mathbf{a}^T\mathbf{y} > 0$.

- Note, this is a linear program.
  - Problem convex, so descent algorithms converge to global optimum.

Perceptron Algorithm

Perceptron Error Function $J_P(a) = \sum_{y \in Y} (-a^T y)$

Y is set of misclassified vectors.

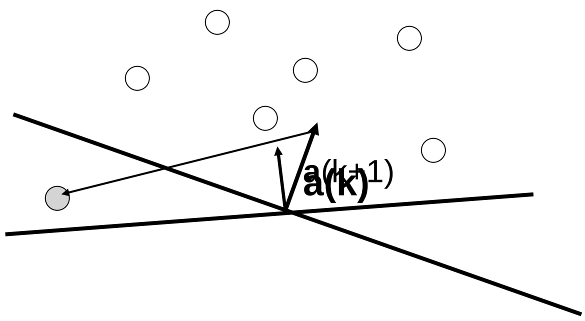
$\nabla J_P = \sum_{y \in Y} (-y)$ So update **a** by:

$$a(k+1) = a(k) + \eta \sum_{y \in Y} y$$

Simplify by cycling through **y** and whenever one is misclassified, update **a** ß **a** + c**y.**

This converges after finite # of steps.

Perceptron Intuition



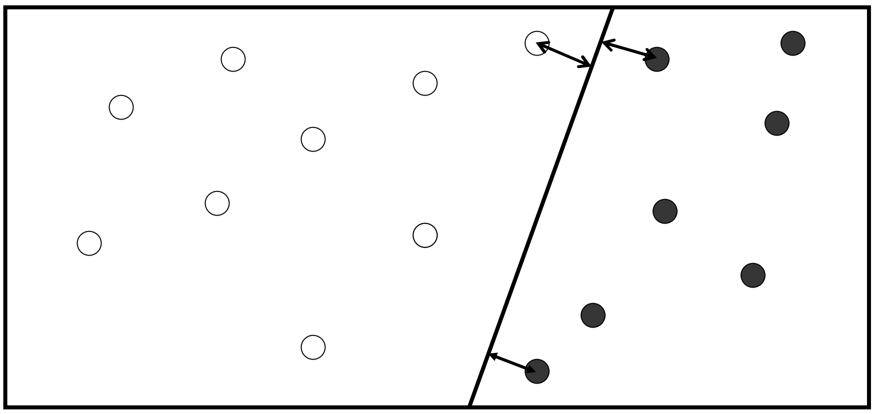
Support Vector Machines

- Find linear separator with maximum margin.
  - Some guarantees this generalizes well.
- Can work in high-dimensional space without overfitting.
  - Nonlinear map to high-dim. space, then find linear separator.
  - Special tricks allow efficiency in ridiculously high dimensional spaces.
- Can handle non-separable classes also.
  - Not as important if space very high-dimensional.

Maximum Margin

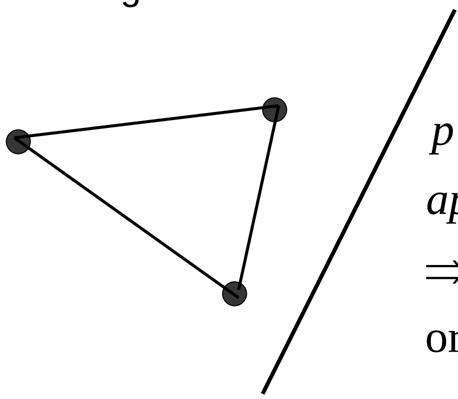
Maximize the minimum distance from hyperplane to points.

Points at this minimum distance are support vectors.



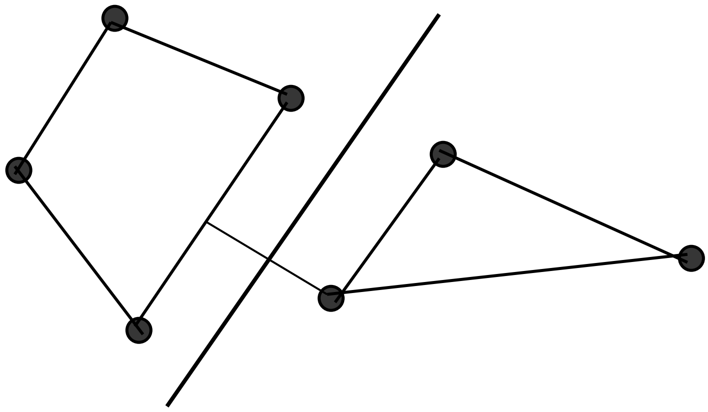
Geometric Intuitions

- Maximum margin between points -> Maximum margin between convex sets



$$p = kp_1 + (1-k)p_2 \quad 0 \le k \le 1$$
$$ap = kap_1 + (1-k)ap_2$$
$$\Rightarrow ap_1 \le ap \le ap_2$$
$$\text{or } ap_1 \ge ap \ge ap_2$$

This implies max margin hyperplane is orthogonal to vector connecting nearest points of convex sets, and halfway between.

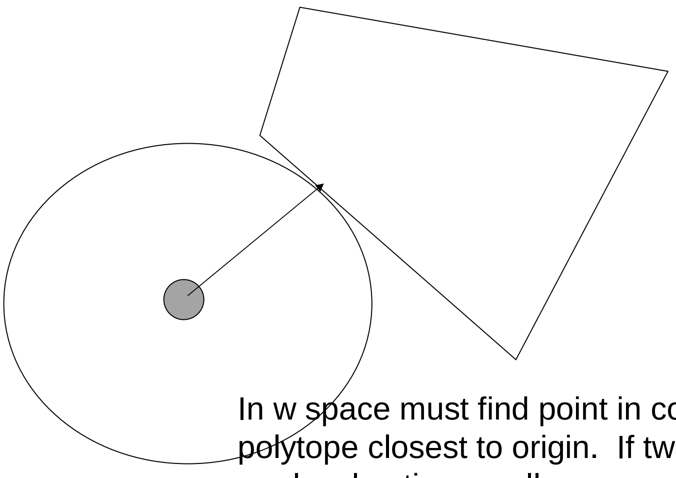


Why is max margin good?

- Intuitively best for noisy data.
- Guarantees good results in leave-one-out classification if #support vectors small.
  - If you leave out non-support vector, get same hyperplane, and correct classification
- Intuitively related to Fisher LDA.

Computation: Finding max margin classifier is convex

- Find w and b such that:
  - wx + b >= 1 for one class
  - wx + b <= 1 for other.
- Margin is 1/||w||.
- So minimize <w,w> subject to linear constraints.
- This is a convex optimization problem.



In w space must find point in convex polytope closest to origin. If two points are local optimum, all convex combinations of them are too, which include closer points.

Fast computation in high dimensional spaces

- w is linear combination of support vectors.
- To compute wx + b need inner products of x and support vectors.
- Use kernels in which inner products for high dimensional space computed in low dimensional space.

Example: monomial kernels

$$\langle x, z \rangle^2 = \left(\sum x_i z_i\right)^2 = \left(\sum x_i z_i\right)\left(\sum x_j z_j\right)$$
$$= \sum\sum x_i x_j z_i z_j = \sum (x_i x_j)(z_i z_j)$$

This is equivalent to the inner product of vectors:

$$\left(x_1 x_1, x_1 x_2, ..., x_n x_n\right)$$

SVM Summary

- Max margin is good, and efficiently computed
- Kernel method allows computatoins in ridiculously high dimensional spaces
- **Combination** is what's important.
  - Arbitrary linear separator won't generalize well.
  - Max margin can generalize in high d space.

Non-statistical learning

- There are a class of functions that could label the data.
- Our goal is to select the correct function, with as little information as possible.
- Don't think of data coming from a class described by probability distributions.
- Look at worst-case performance.
  - This is CS'ey approach.
  - In statistical model, worst case not meaningful.

On-Line Learning

- Let X be a set of objects (eg., vectors in a high-dimensional space).
- Let C be a class of possible classifying functions (eg., hyperplanes).
  - f in C: X-> {0,1}
  - One of these correctly classifies all data.
- The learner is asked to classify an item in X, then told the correct class.
- Eventually, learner determines correct f.
  - Measure number of mistakes made.
  - Worst case bound for learning strategy.

VC Dimension

- S, a subset of X, is *shattered* by C if, for any U, a subset of S, there exists f in C such that f is 1 on U and 0 on S-U.
- The *VC Dimension* of C is the size of the largest set shattered by C.

VC Dimension and worst case learning

- Any learning strategy makes at least VCdim(C) mistakes in the worst case.
  - If S is shattered by C
  - Then for any assignment of values to S, there is an f in C that makes this assignment.
  - So any set of choices the learner makes for S can be entirely wrong.
- Alternately, sets of C exist where no generalization possible based on C-1 examples.

VC Dimension and SVMs

- VC dimension depends on number of support vectors.
- Example: suppose 2 support vectors
  - For n points, at most n*n classes.
  - To shatter points, must have 2^n classes.
  - VC dimension < 5.
  - This does not depend on dimension of space.
- Similarly for k support vectors