CMSC 471

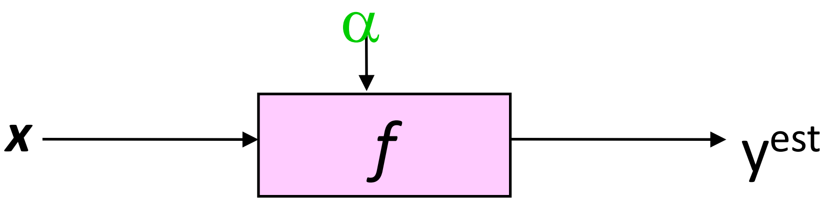
ML: Support Vector Machines (SVM)

Slide Courtesy: Tim Finin

Support Vector Machines

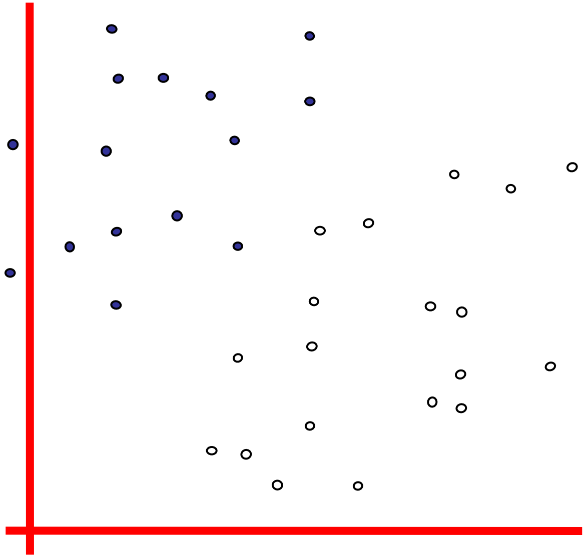
- Very popular ML technique
  - Became popular in the late 90s (Vapnik 1995; 1998)
  - Invented in the late 70s (Vapnik, 1979)
- Controls complexity and overfitting, so works well on a wide range of practical problems
- Can handle high dimensional vector spaces, which makes feature selection less critical
- Fast and memory efficient implementations, e.g., svm_light
- Not always best solution, especially for problems with small vector spaces

Linear Classifiers



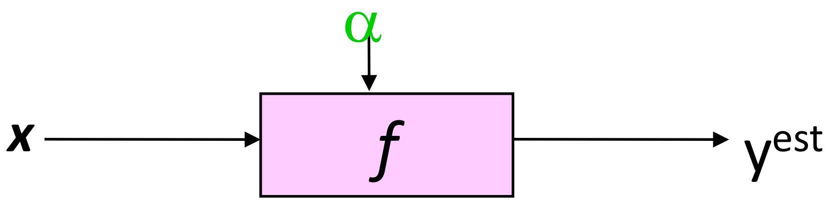
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

- ● denotes +1
- ○ denotes -1

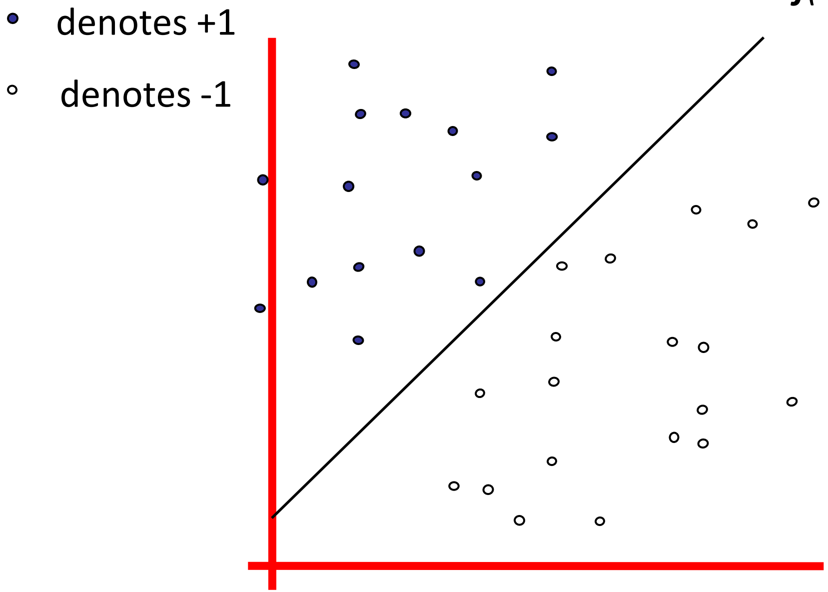


How would you classify this data?

Linear Classifiers

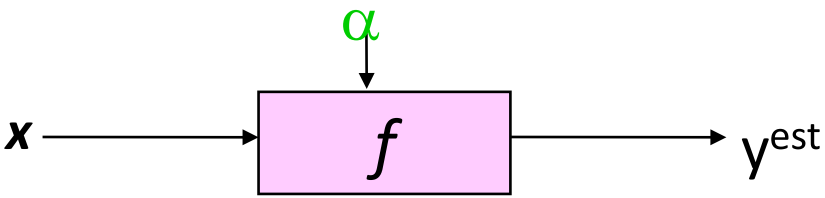


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

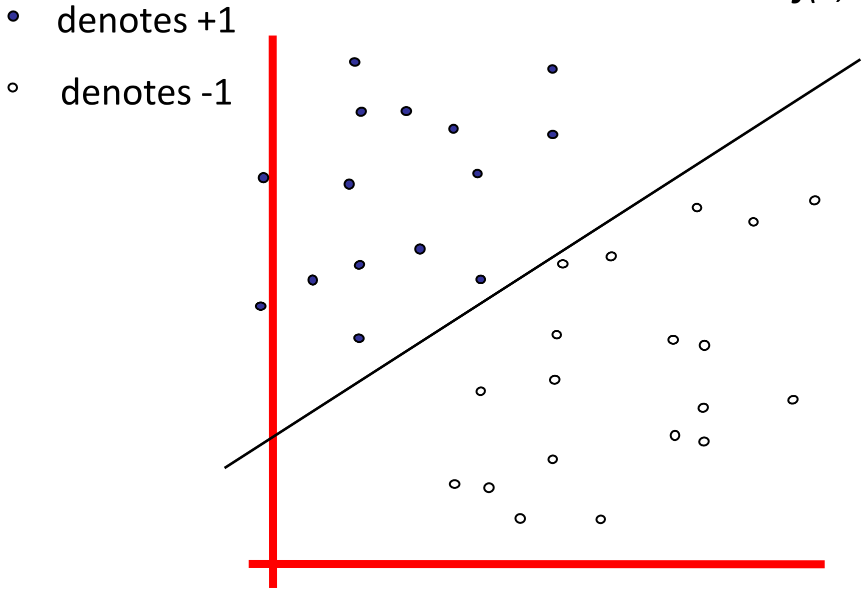


How would you classify this data?

Linear Classifiers

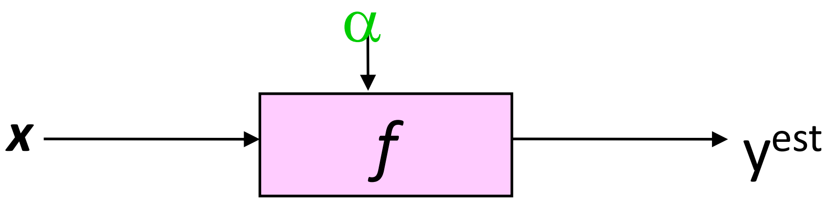


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

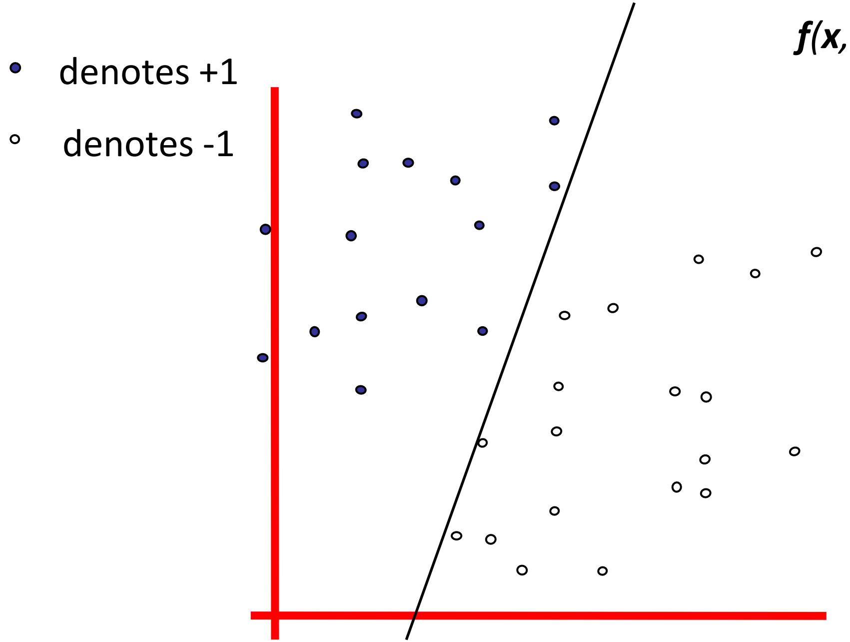


How would you classify this data?

Linear Classifiers

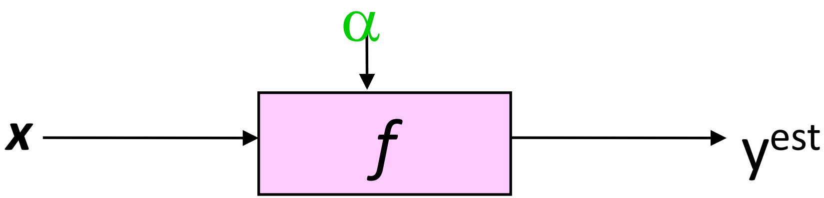


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

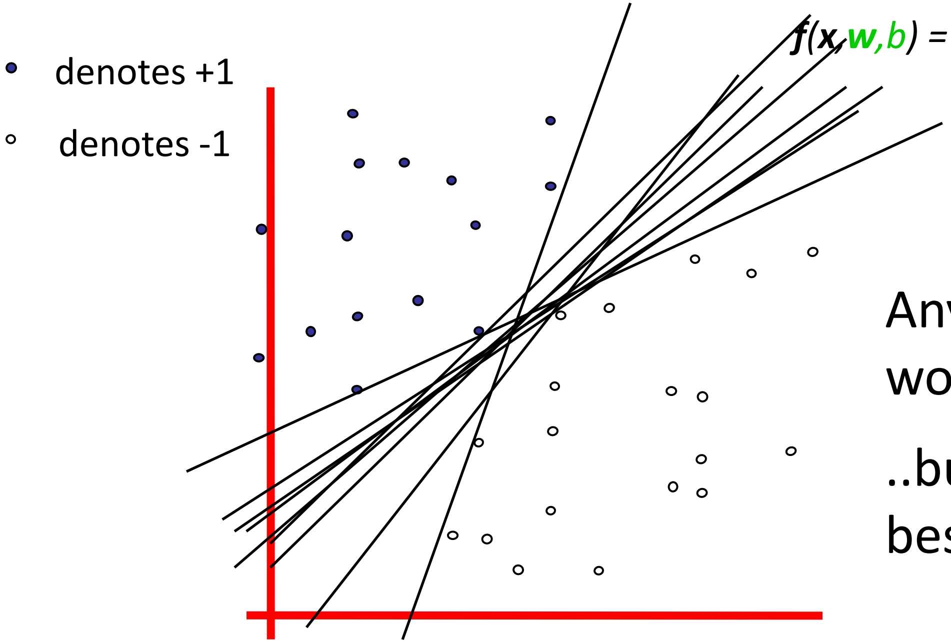


How would you classify this data?

Linear Classifiers



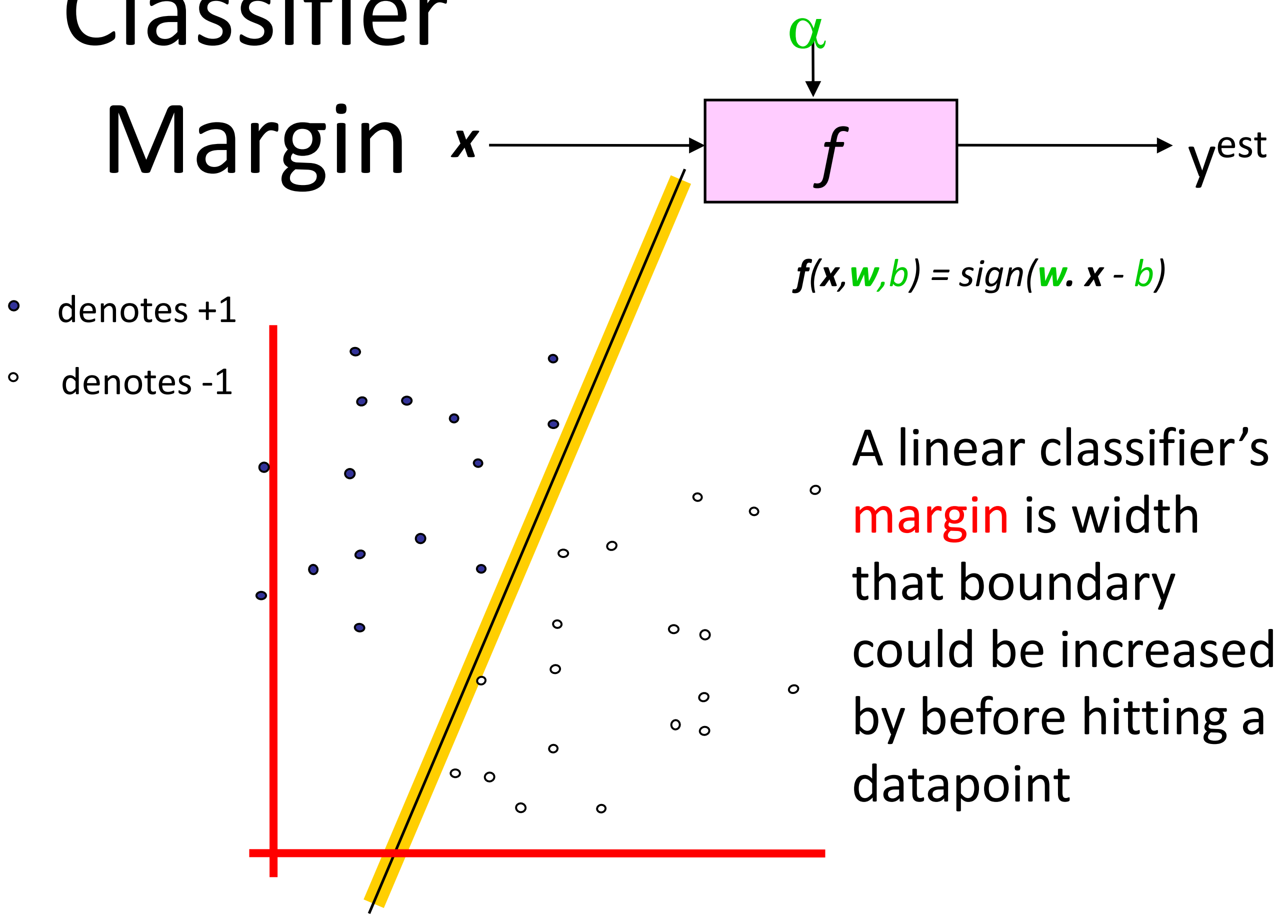
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$



Any of these would be fine..

..but which is best?

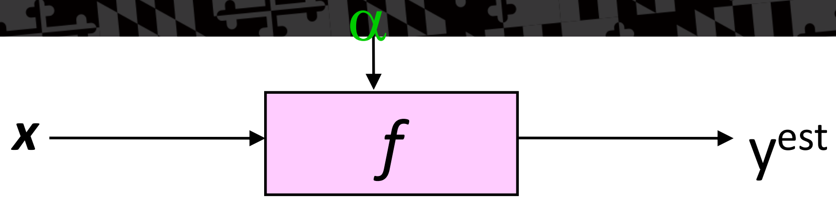
Classifier Margin



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w} \cdot \mathbf{x} - b)$

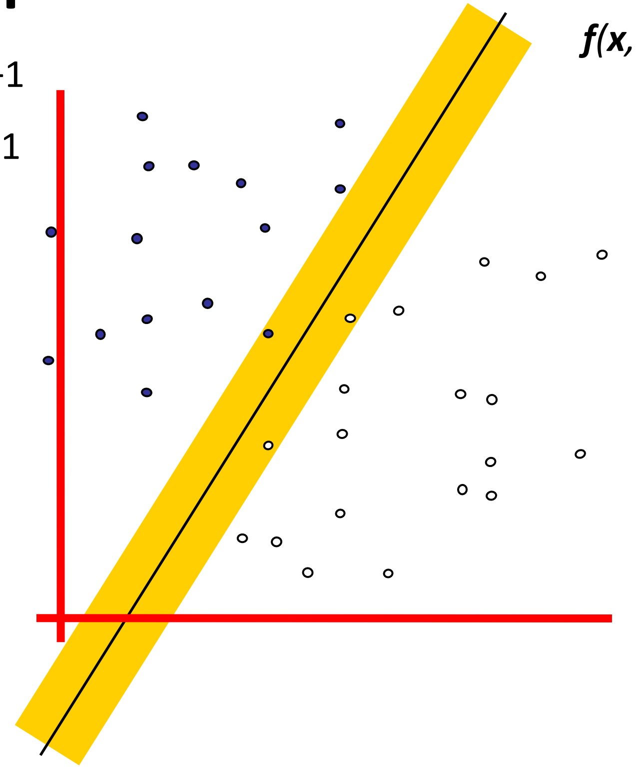
A linear classifier's margin is width that boundary could be increased by before hitting a datapoint

Maximum Margin



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

- denotes +1
- denotes -1

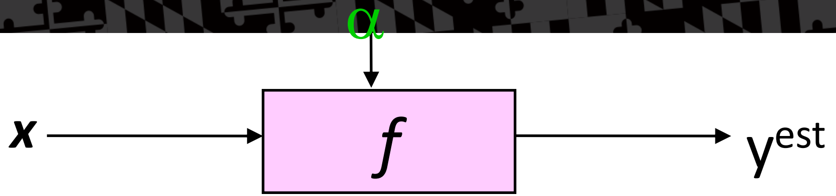


Maximum margin linear classifier is the linear classifier with the largest margin

The simplest kind of SVM, called an LSVM

Linear SVM

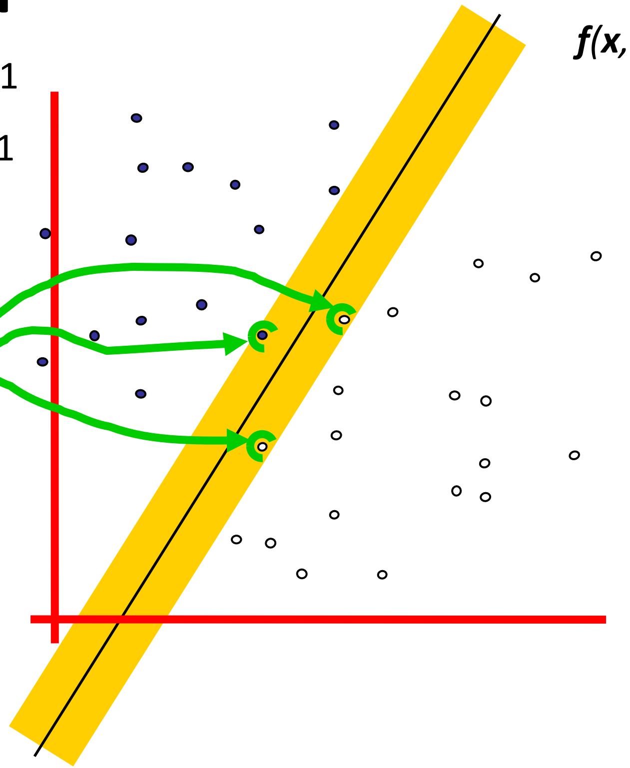
Maximum Margin



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}. \mathbf{x} - b)$

• denotes +1

° denotes -1



Support Vectors are the datapoints that margin pushes up against

Maximum margin linear classifier is the linear classifier with the largest margin

The simplest kind of SVM, called an LSVM

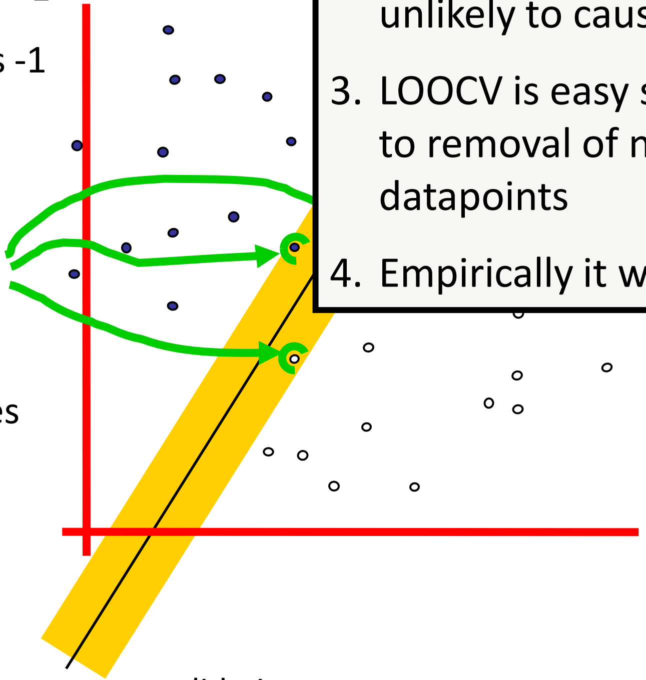
Linear SVM

Why Maximum Margin?

1. Intuitively this feels safest
2. Small errors in boundary location unlikely to cause misclassification
3. LOOCV is easy since model is immune to removal of non-support-vector datapoints
4. Empirically it works very very well

• denotes +1

∘ denotes -1



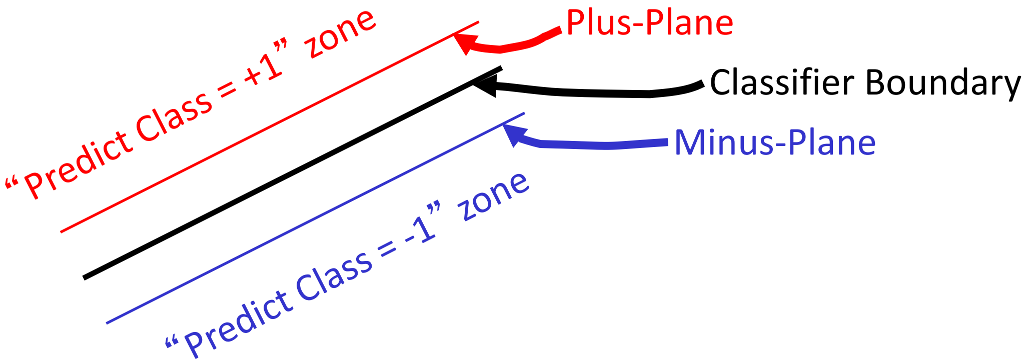
Support Vectors are those datapoints that the margin pushes up against

margin.

This is the simplest kind of SVM (Called an LSVM)

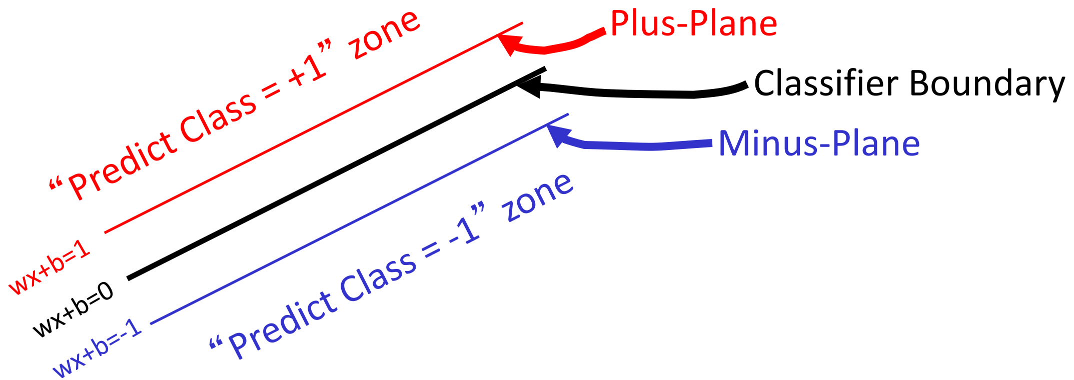
LOOCV = leave one out cross validation

Specifying a line and margin



- How do we represent this mathematically?
- ...in *m* input dimensions?

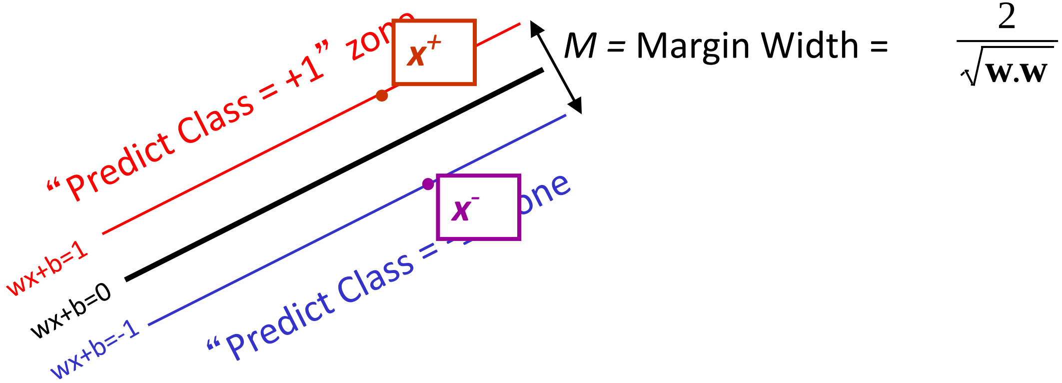
Specifying a line and margin



- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \,.\, \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \,.\, \mathbf{x} + b = -1 \}$

Classify as..	+1	if	$\mathbf{w} \,.\, \mathbf{x} + b >= 1$
	-1	if	$\mathbf{w} \,.\, \mathbf{x} + b <= -1$
	Universe explodes	if	$-1 < \mathbf{w} \,.\, \mathbf{x} + b < 1$

Learning the Maximum Margin Classifier



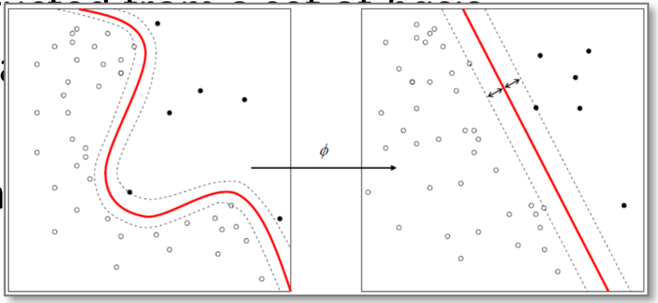
- Given a guess of ***w*** and *b* we can
  - Compute whether all data points in the correct half-planes
  - Compute the width of the margin
- Write a program to search the space of **w**s and *b*s to find widest margin matching all the datapoints.
- *How?* -- Gradient descent? Simulated Annealing? Matrix Inversion? EM? Newton's Method?

Learning SVMs

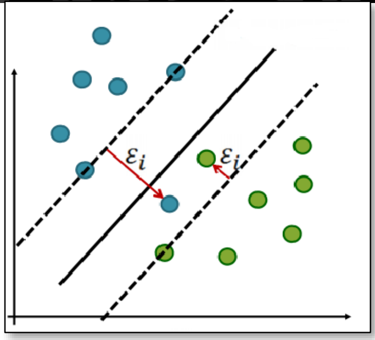
- Trick #1: Find points that would be closest to optimal separating plane ("support vectors") and work directly from those instances
- Trick #2: Represent as a **quadratic optimization problem**, and use quadratic programming techniques
- Trick #3 ("kernel trick"):
  - Instead of using raw features, represent data in a high-dimensional feature space constructed from a set of *basis functions* (e.g., polynomial and Gaussian combinations of the base features)
  - Find separating plane/SVM in that space
  - Voila: A nonlinear classifier!

Learning SVMs

- Trick #1: Find points that would be closest to optimal separating plane ("support vectors") and work directly from those instances
- Trick #2: Represent as a **quadratic optimization problem**, and use quadratic programming techniques
- Trick #3 ("kernel trick"):
  - Instead of using raw features, represent data in a high-dimensional feature space constr
    functions (e.g., polynomial and G
    the base features)
  - Find separating plane/SVM in tha
  - Voila: A nonlinear classifier!



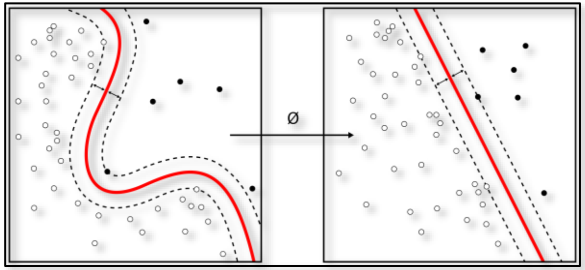
Soft margin classification



- What if data from two classes not linearly separable?
- Allow a fat decision margin to make a few mistakes
- Some points, outliers or noisy examples, are inside or on wrong side of the margin
- Each outlier incurs a cost based on distance to hyperplane

Kernel trick

- What if data from two classes not linearly separable?
- Project data onto a higher dimensional space where it becomes linearly separable
- Many SVMs can take an argument, a **kernel**, that does the transformation of the data
- Deciding what **kernel function** to use is done through experimentation



Kernel Trick example

Can’t separate the blue & red points with a line

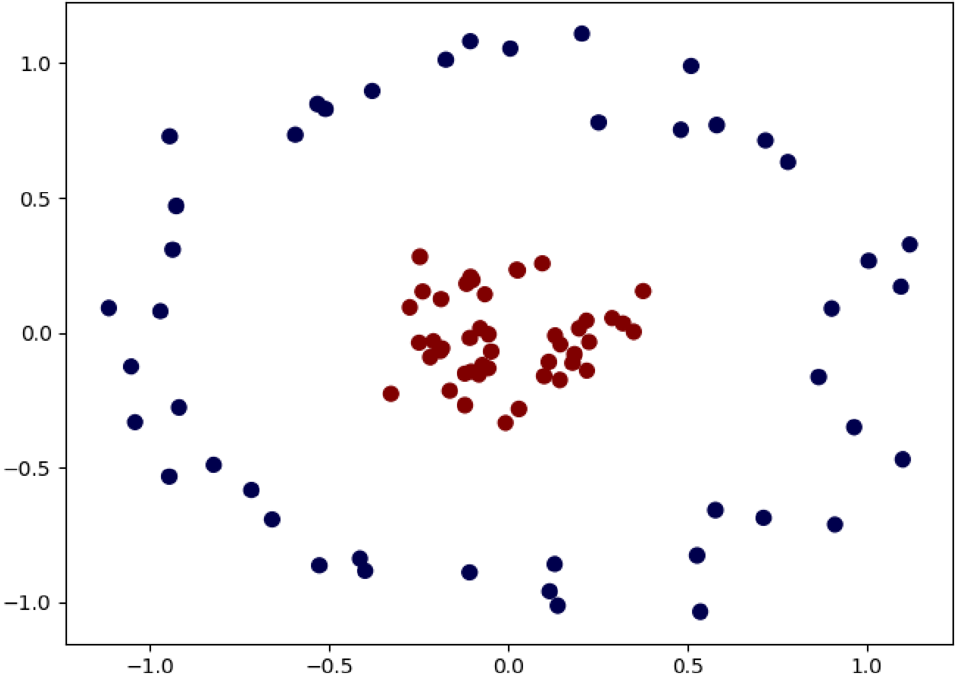
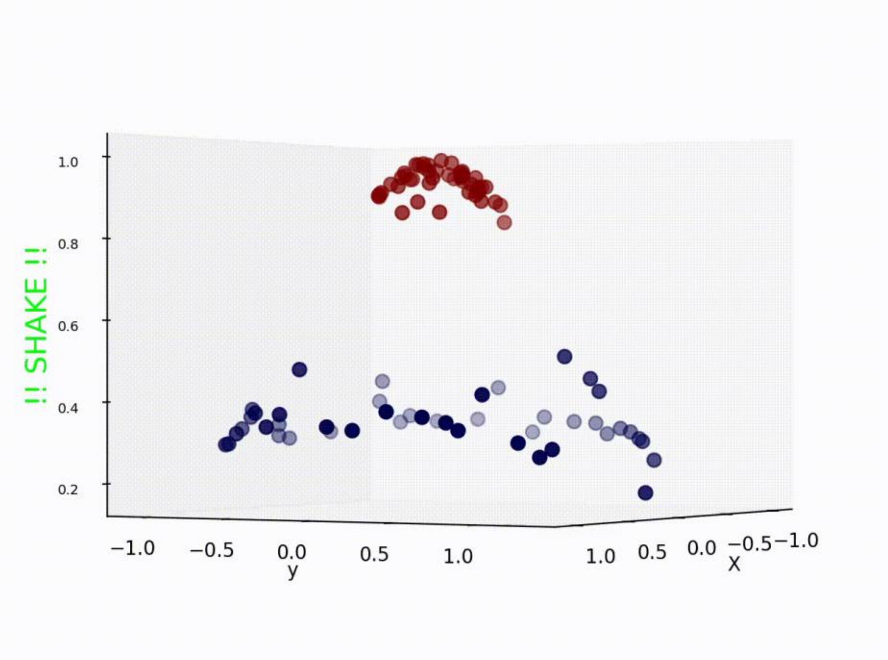


Figure from [S. Bhattacharyya]

Use a different kernel

- Applying a kernel can transform data to make it more nearly linearly separable
- E.g., use polar coordinates or map to three dimentions



SVM Performance

- SVMs can handle very large features spaces (e.g., 100K features)
- Relatively fast
- Anecdotally they work very well indeed
- Example: They are among the best-known classifier on a well-studied hand-written-character recognition benchmark

SVMs in scikit-learn

- Scikit-learn has three SVM classifiers: SVC, NuSVC, and LinearSVC
- Data can be either in **dense numpy** arrays or **sparse scipy** arrays
- All directly support multi-way classifica-tion, SVC and NuSCV using OVO and LinearSVC using OVA