



Support Vector Machines

CS 445/545



SVM merch
available in the
CS department
after lecture



Support Vector Machines

CS 445/545

Introduction

- Previously we considered *Perceptrons* based on the McCulloch and Pitts neuron model.
- We identified a method for updating weights, PLA, and noticed that the Perceptron is rather limited in that it could only identify straight line classifiers.
- This mean that it could not learn to distinguish classes, for instance, with **2D XOR function**.



- However, we saw that it was in fact possible to modify the problem so that the perceptron could solve the problem.

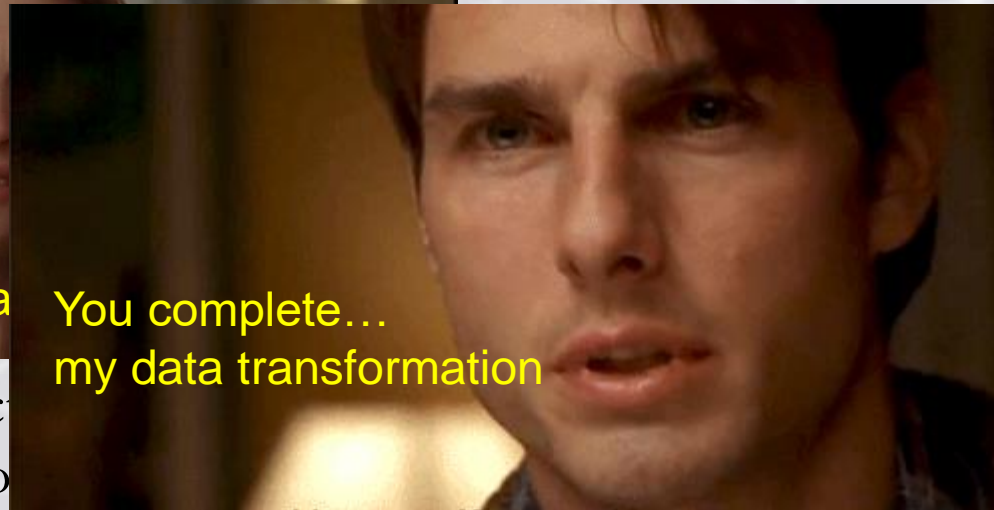
Q: How was this done?

Introduction

- Previously we considered *Perceptrons* based on the McCulloch and Pitts neuron model.
- We identified a method for updating weights, PLA, and noticed that the Perceptron is rather limited in that it could only identify straight line classifiers.
- This mean that it could not learn to distinguish classes, for instance, with **2D XOR function**.



You ha



You complete...

my data transformation

- However, we saw that it was in fact the perceptron could solve the problem

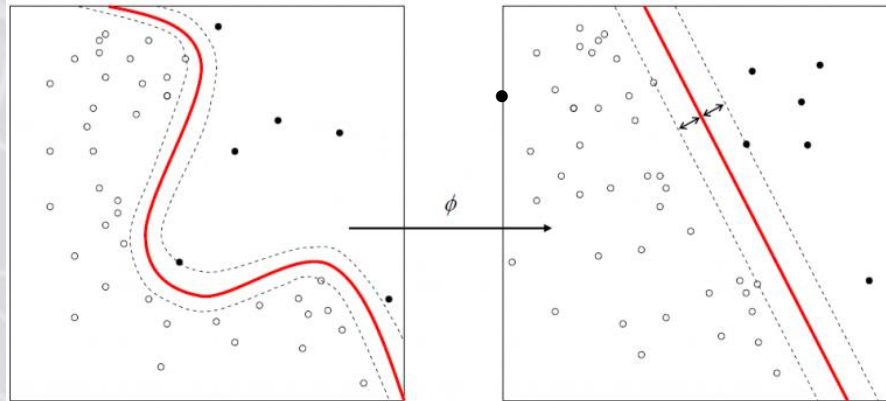
Q: How was this done? One way: **transform the data!**

Introduction

- Here we will modify the data by **changing its representation** (a powerful idea in ML and data science). In principle, it is always possible to transform any given data set so that the classes within it are *linearly separable*.
- We introduce *kernel functions* to achieve this transformation.
- The key idea is to ascertain which dimensions to “use”; generally, *kernel methods* comprise a class of algorithms that do just this.

Introduction

- We focus here on support vector machines (SVMs) for classification.
- Given a set of training examples, each one belonging to a specific category, an **SVM** training algorithm creates a model that separates the categories and that can later be used to decide the category of new set of data.



- The training component consists in finding the best separating plane (**with maximal margin**) based on specific vector called **support vector**. If the decision is not feasible in the initial description space, you can increase space dimension using a **kernel** function and subsequently identify a hyperplane that serves as your decision surface.

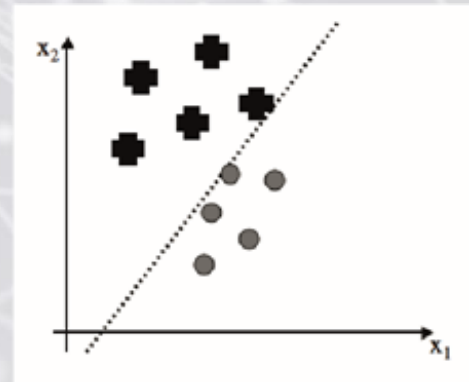
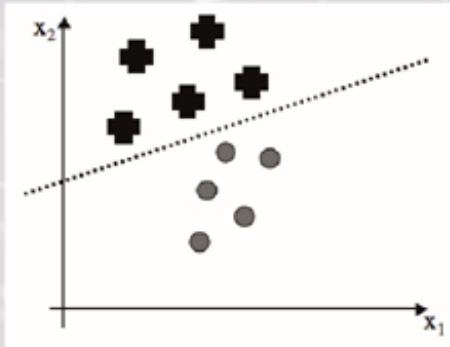
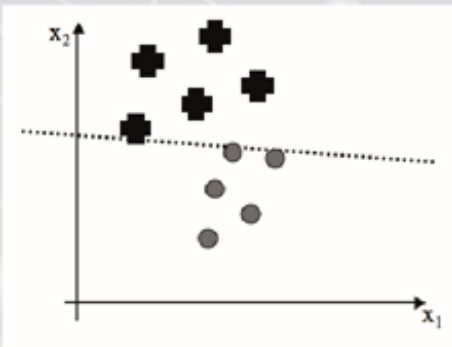
Introduction

- SVMs are one of the more popular algorithms in modern machine learning (although they have decreased in prominence more recently, with the advent of deep NNs).
- Vapnik introduced them in 1992, at which time they became wildly popular in ML due to their impressive classification performance on reasonably-sized datasets.
- In principle, SVMs do not work well on extremely large datasets, since the **computations don't scale well** with the number of training examples.
- In addition to incorporating kernel functions, the SVM algorithm also reformulates the classification problem

Introduction

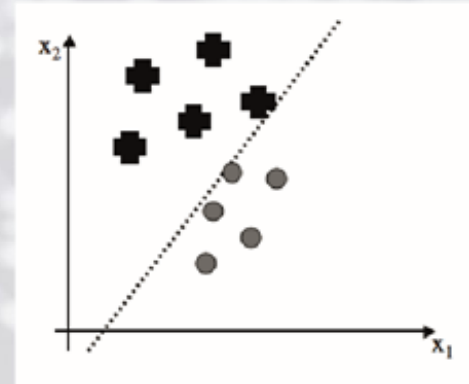
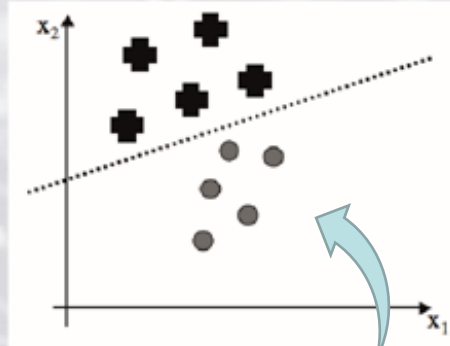
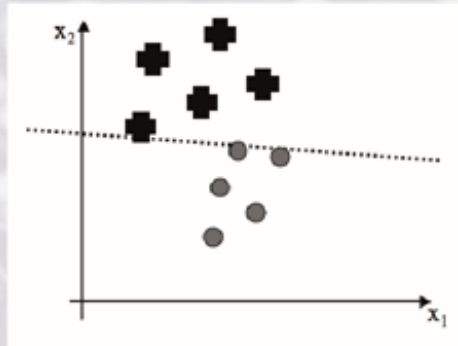
- In addition to incorporating kernel functions, the SVM algorithm also reformulates the classification problem in a way that allows us to discriminate between a good classifier and a bad one.
- Consider, for example, the following decision boundaries for identical sets of data.

Q: Which is the “best” classifier?



Introduction

Q: Which is the “best” classifier?

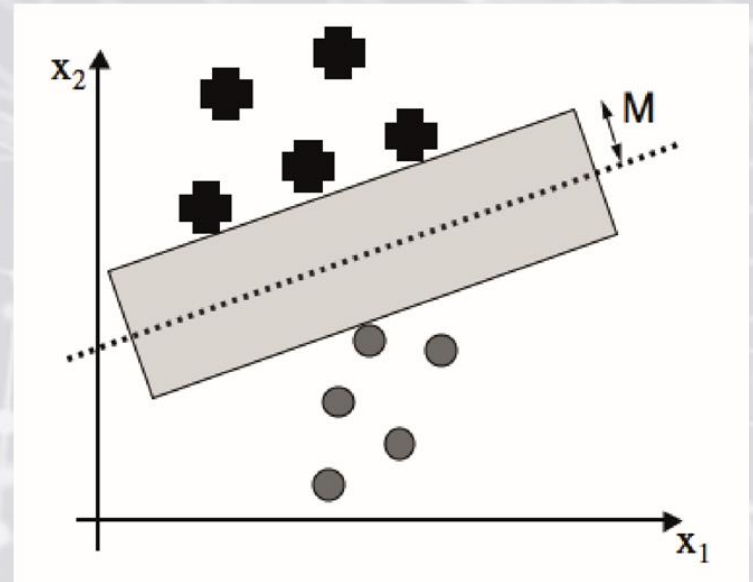


A: The classifier that **generalizes best**.

How do we quantify the notion of an optimal line?

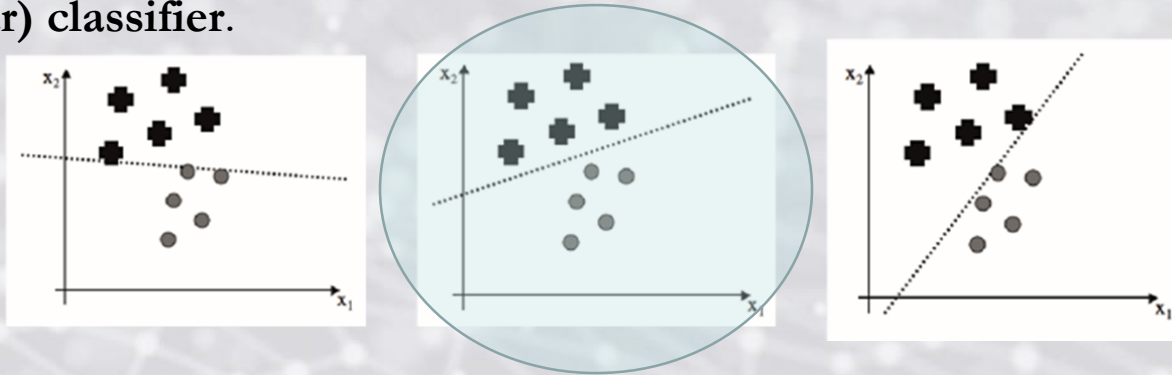
Introduction

- We can measure the distance that we have to travel away from the decision line (in a direction perpendicular to the line) before we encounter a data point.
- Imagine a region (“no man’s land”) around the line that we designate as representing data points that are “too close” to the line to be accurately classified; render this region as a cylinder in the “hypothesis space.”
- How large can we make the radius of this cylinder until classification becomes ambiguous (i.e. we capture points from different classes)?
- Call this maximum radius M , the **margin** of the decision surface.



Introduction

- The classifier in the middle has the largest margin of the three; we say it is the **maximum margin (linear) classifier**.



- The data points in each class that lie closest to the classification line are called the **support vectors** (of the classifier).
- Using the argument that the best classifier is the one that goes through the middle of “no man’s land”, we can make two arguments:
 - (1) The **margin should be as large as possible**.
 - (2) The **support vectors are the most useful data points** (they exist on the threshold of what we can correctly classify).
- (*) *Note:* SVMs allow us to discard all of the training data except for the support vectors! This is very useful for compression/data storage constraints.

Aside #1: VC Dimension

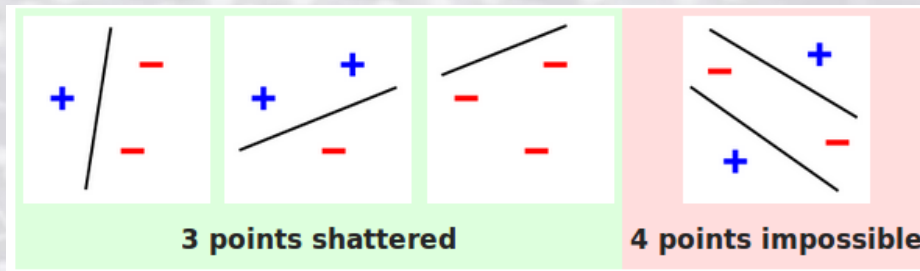


V. Vapnik

- *VC dimension* is an important and deep concept in ML.
- VC dimension is a measure of the **capacity** (i.e. *complexity*, *expressive power*, *richness*, or *flexibility*) of a space of functions that can be learned by a classification algorithm.
- A classification model f with some parameter θ is said to shatter a set of data points (x_1, \dots, x_n) if, for all assignments of labels to those points, there exists a θ such that the model f makes **no errors** when evaluating that set of data points.
- **VC dimension** (of a model f) := the maximum number of points that can be arranged so that f shatters them.

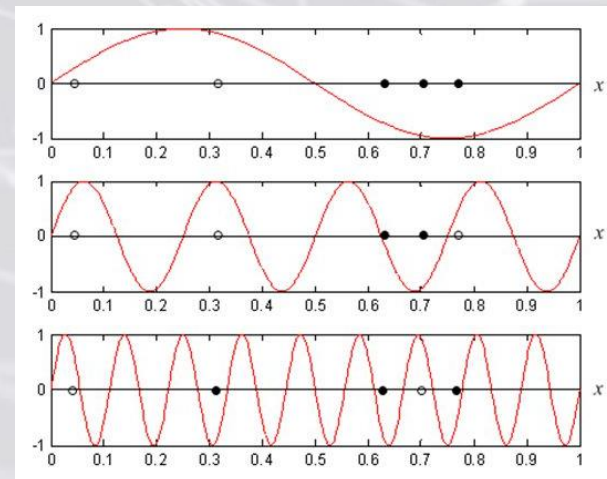
Aside #1: VC Dimension

- Example: If f is a constant classifier, its VC dimension is **zero**, since it cannot shatter even a single point.
- Example: If f is a straight line classifier (in 2-D). There exist sets of 3 points that can indeed be shattered using the model (any 3 points that are non-collinear can be shattered). However, no set of 4 points can be shattered. Thus the **VC dimension of a straight line is 3**.



Note we only show 3 of the $2^3=8$ possible binary labelings for 3 points.

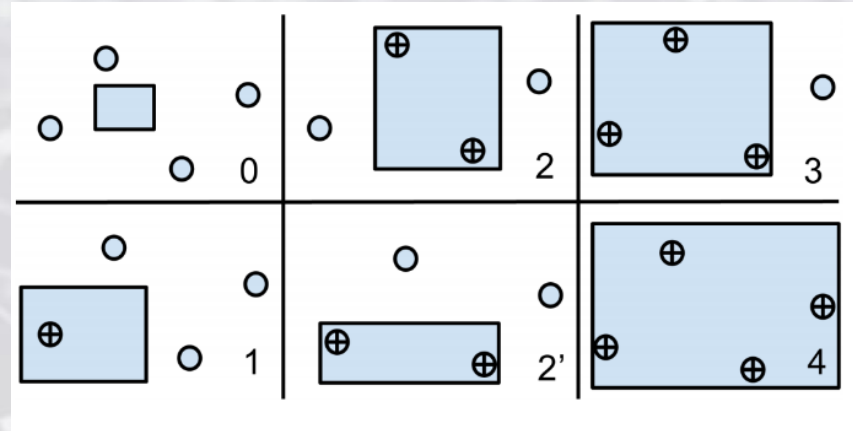
- Example: f is a single-parameter “sine classifier”, i.e. for a certain parameter θ , the classifier f_θ returns 1 if the input number x is larger than $\sin(\theta x)$ and 0 otherwise.
- The VC dimension of f is **infinite**, since it can shatter the set: $\{2^{-m} \mid m \in \mathbb{N}\}$ for any positive m .



(*) Note the last example shows, importantly, that VC dimension is not directly related to the number of model parameters!

Aside #1: VC Dimension

- Another example: the VC dimension of a *rectangle* in \mathbf{R}^2 (where the rectangle encompasses all data of a particular class).

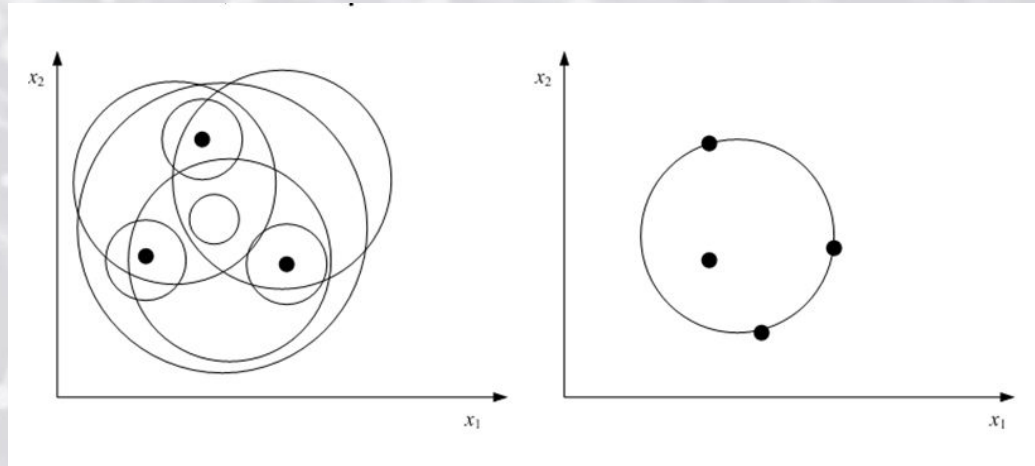


- The diagram shows that **a rectangle shatters at least 4 points in the plane**; for 5 points there exists a counter-example (try it).
- In general, the **VC dimension of a hyperplane in \mathbf{R}^d is $d+1$** .
(e.g. in \mathbf{R}^2 we previously showed a hyperplane has VC dimension = 3).

Aside #1: VC Dimension

- Another example: the VC dimension of a *spherical indicator function* in \mathbf{R}^2 (where the sphere encompasses all data of a particular class).

$$f(\mathbf{x}, \mathbf{c}, r) = I\left((\mathbf{x} - \mathbf{c})^2 \leq r^2\right)$$

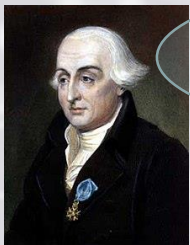


- Three points in \mathbf{R}^2 can be shattered, but four cannot; thus **VC dim(f) = 3** in \mathbf{R}^2 (useful for nearest neighbors classifiers, radial basis functions).

Aside #2: Optimization Paradigms

- Two of the most common optimization paradigms in ML include:

(1) Optimization with equality constraints.

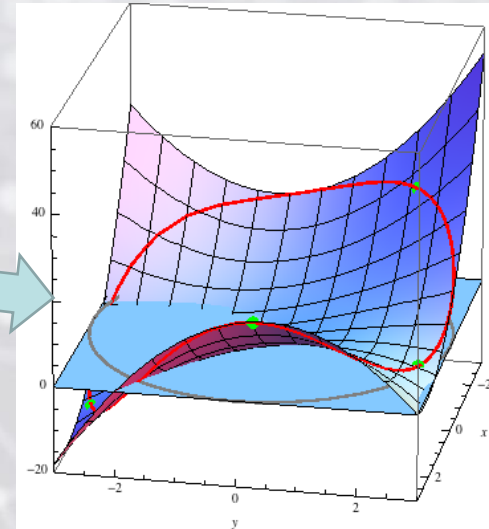


Lagrange

$$\begin{aligned} &\text{maximize } f(x, y, z) \\ &\text{subject to } g(x, y, z) = 0, \quad h(x, y, z) = 0 \end{aligned}$$

$$\nabla f(x, y, z) = \lambda \nabla g(x, y, z) + \mu \nabla h(x, y, z)$$

Method of Lagrange Multipliers

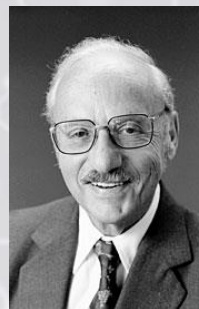
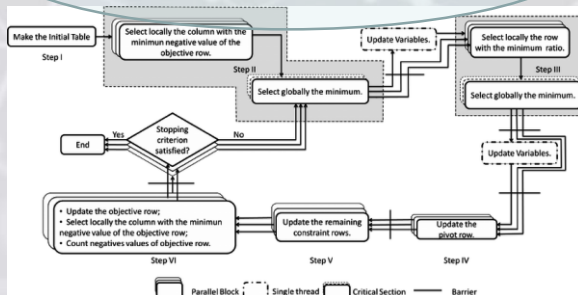


(2) Optimization with inequality constraints.



von Neumann

$$\begin{aligned} &\text{maximize } c^T x \\ &\text{subject to } Ax \leq b \text{ and } x \geq 0 \end{aligned}$$

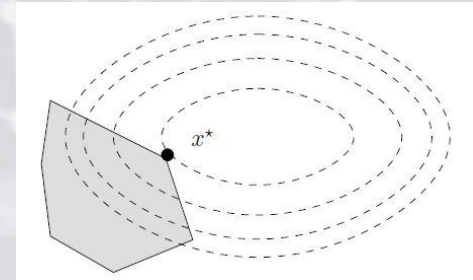
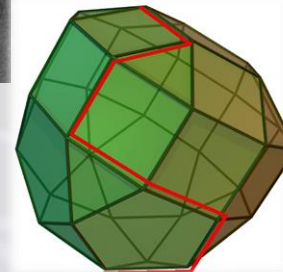


Dantzig

Linear Programming: Simplex Method

$$\begin{aligned} &\text{maximize } \frac{1}{2} x^T Q x + c^T x \\ &\text{subject to } Ax \leq b \end{aligned}$$

Quadratic Programming



Notation

- Assume a binary classification problem.

- Instances are represented by vector $\mathbf{x} \in \mathbb{R}^n$.

- Training examples: $\mathbf{x} = (x_1, x_2, \dots, x_n)$

$$S = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_m, t_m) \mid (\mathbf{x}_k, t_k) \in \mathbb{R}^n \times \{+1, -1\}\}$$

- Hypothesis: A function $h: \mathbb{R}^n \rightarrow \{+1, -1\}$.

$$h(\mathbf{x}) = h(x_1, x_2, \dots, x_n) \in \{+1, -1\}$$

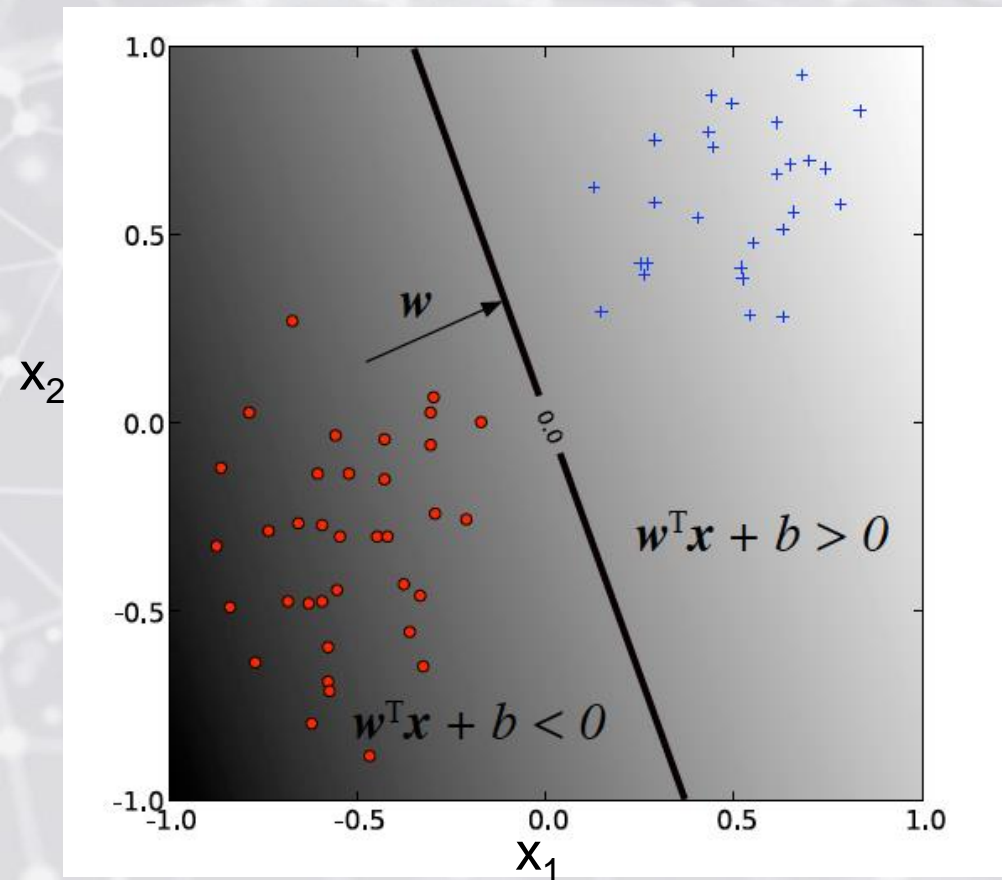
- Here, assume positive and negative instances are to be separated by the hyperplane

$$\mathbf{w}^T \mathbf{x} + b = 0$$

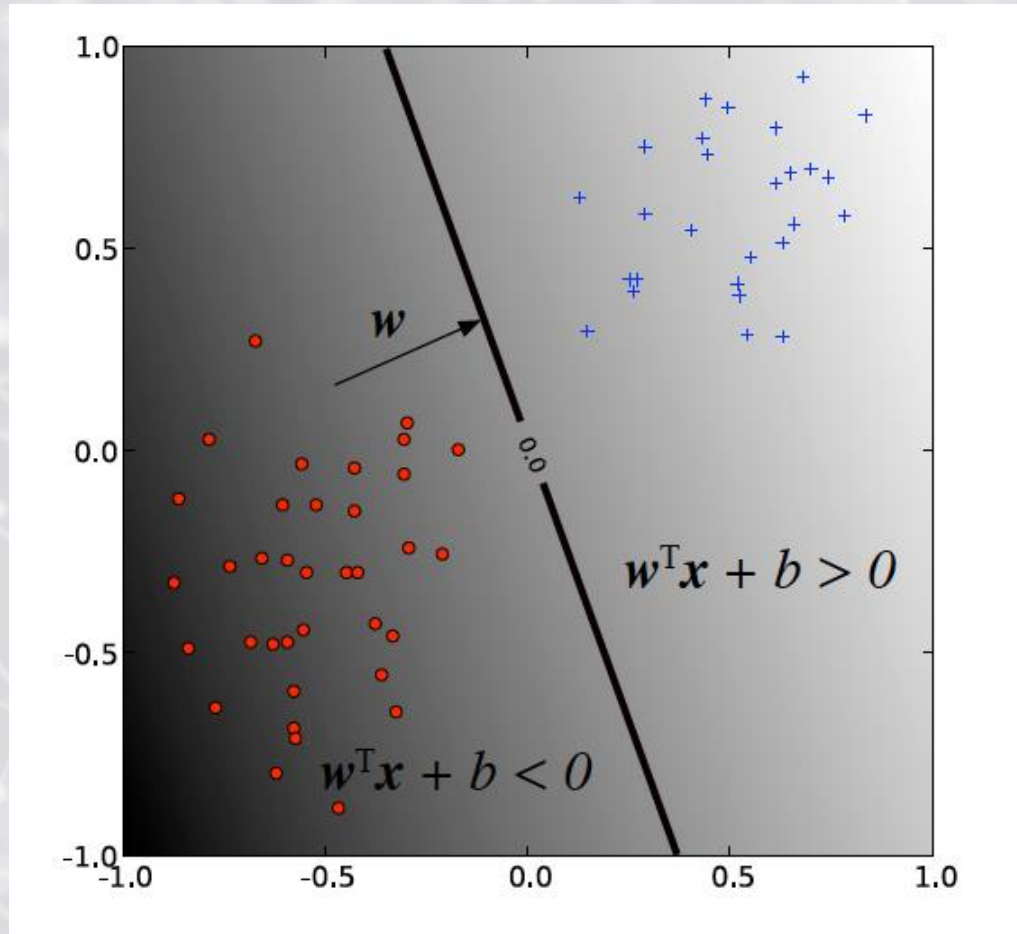
where b is the bias.

Equation of line:

$$\begin{aligned}\mathbf{w}^T \mathbf{x} + b &= \mathbf{w}^T \mathbf{x} + b \\ &= w_1 x_1 + w_2 x_2 + b = 0\end{aligned}$$



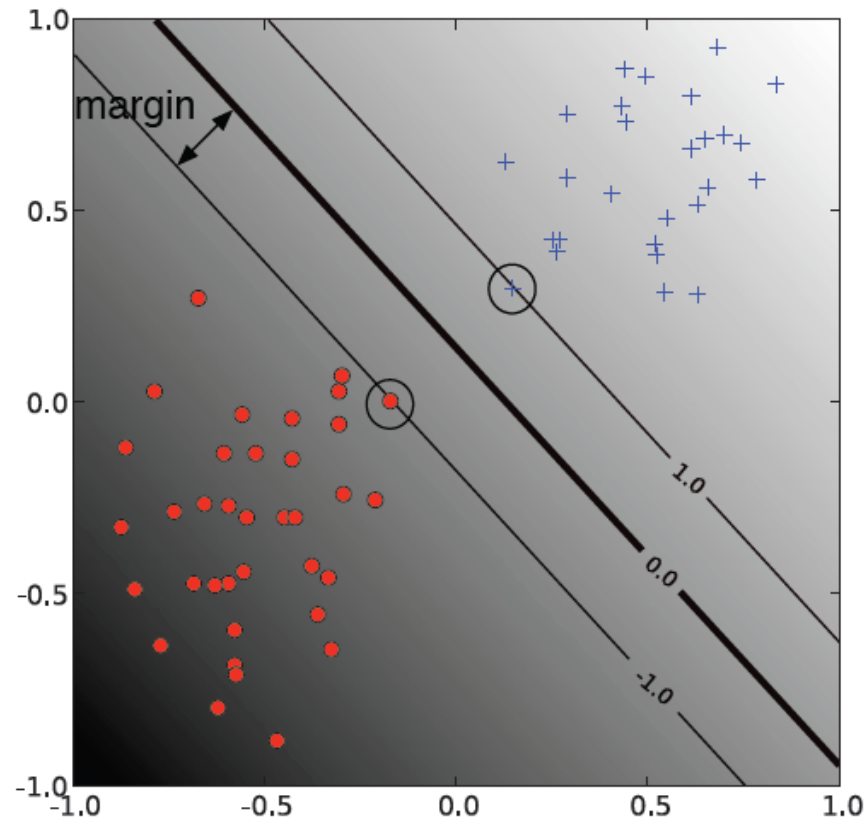
$$\mathbf{w}^\top \mathbf{x} + b = 0$$



- **Intuition:** the best hyperplane (for future generalization) will “maximally” separate the examples

Definition of Margin (with respect to a hyperplane):

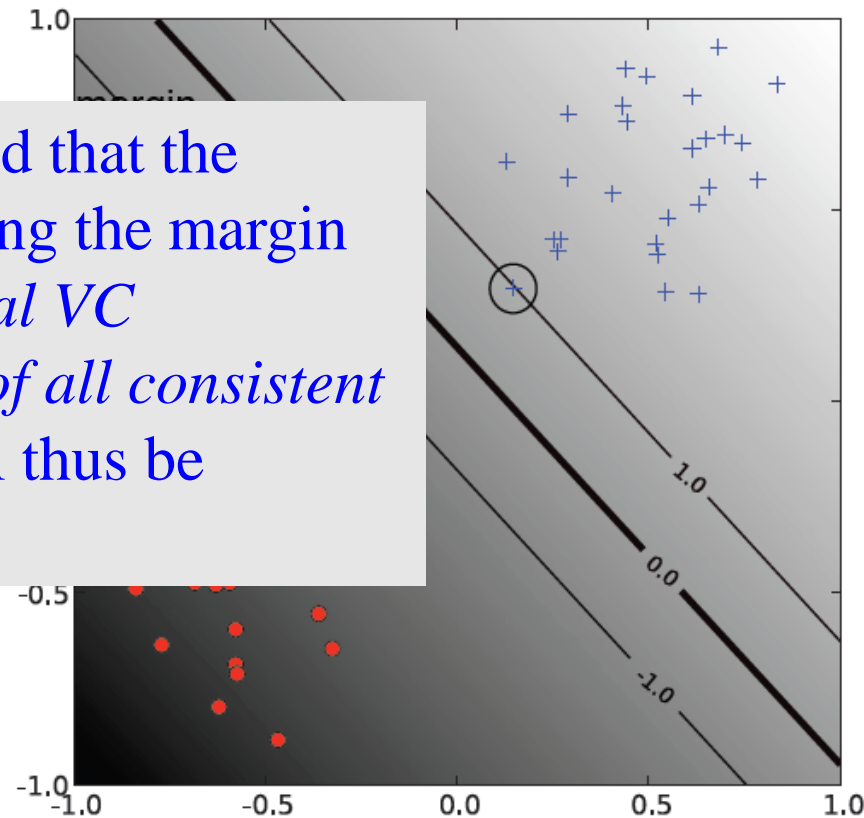
Distance from separating hyperplane to nearest positive (or negative) instance.



Definition of Margin (with respect to a hyperplane):

Distance from separating hyperplane to nearest positive (or negative) instance.

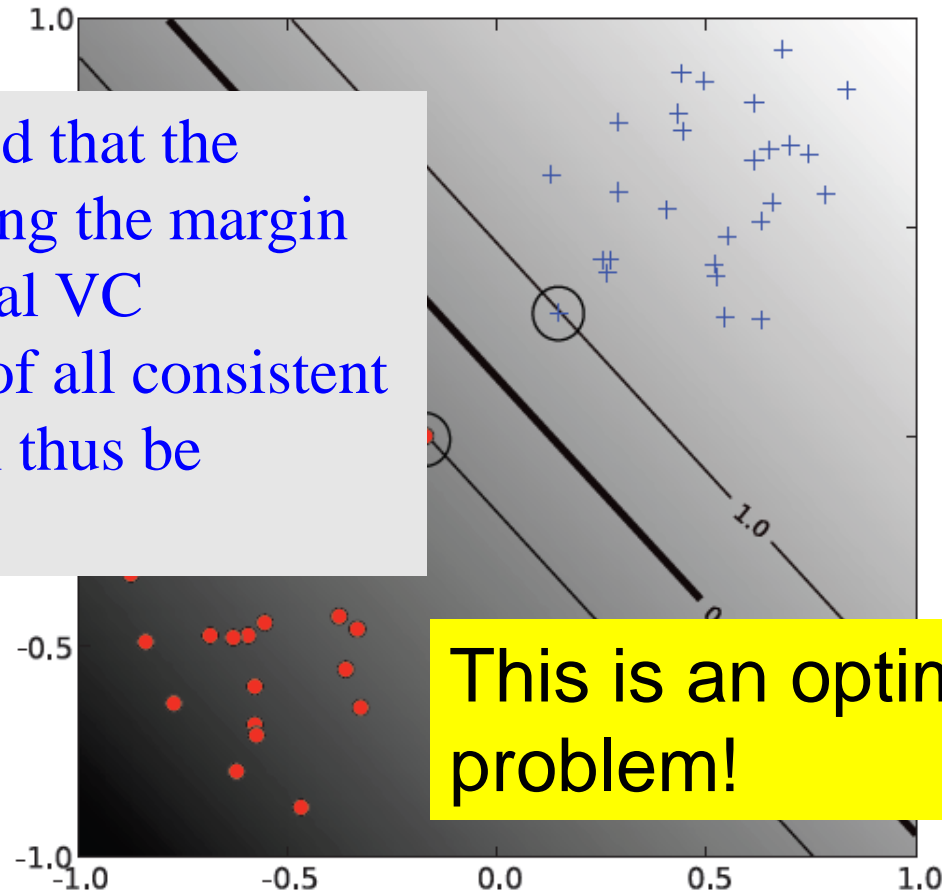
Vapnik (1979) showed that the hyperplane maximizing the margin of S will have *minimal VC dimension in the set of all consistent hyperplanes*, and will thus be optimal.



Definition of Margin (with respect to a hyperplane):

Distance from separating hyperplane to nearest positive (or negative) instance.

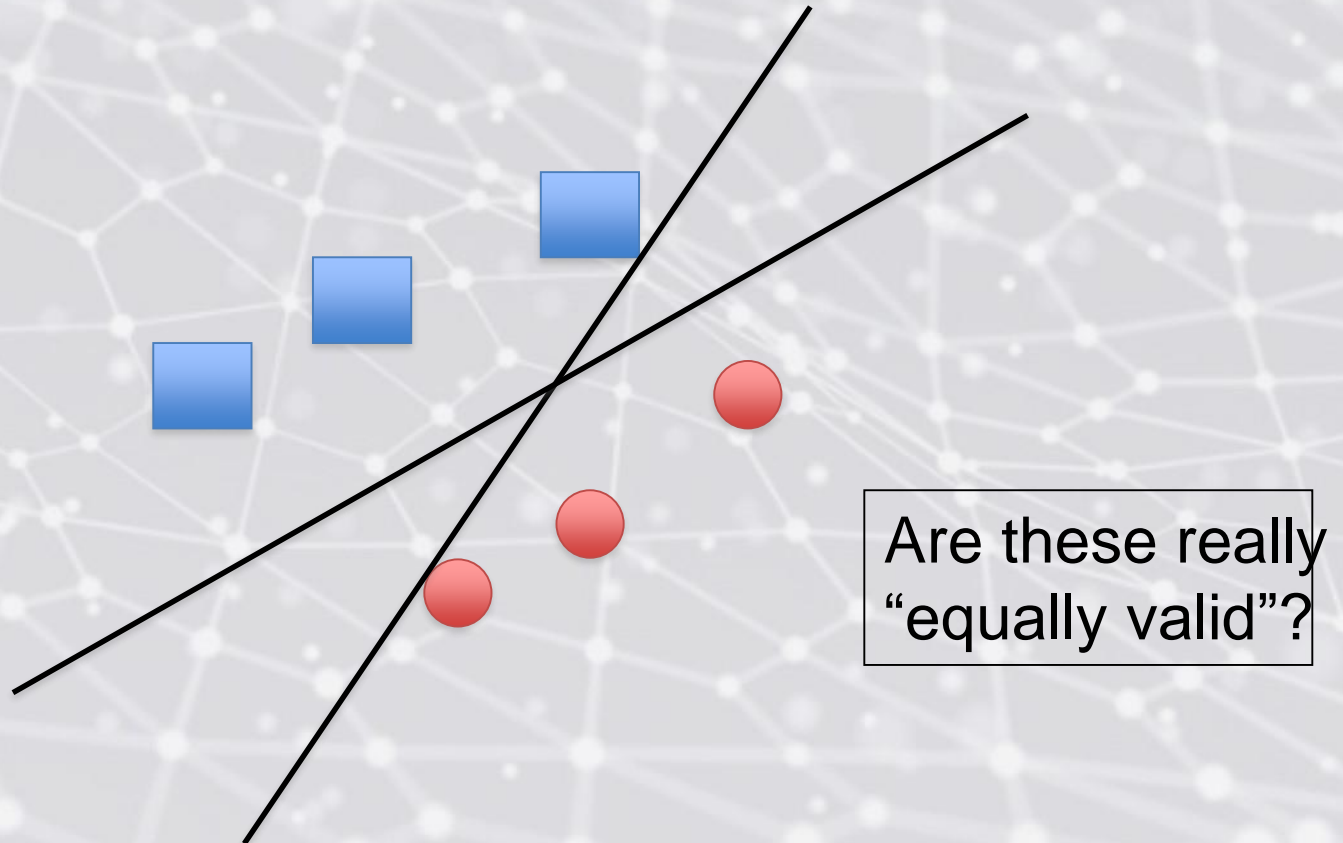
Vapnik (1979) showed that the hyperplane maximizing the margin of S will have minimal VC dimension in the set of all consistent hyperplanes, and will thus be optimal.



This is an optimization problem!

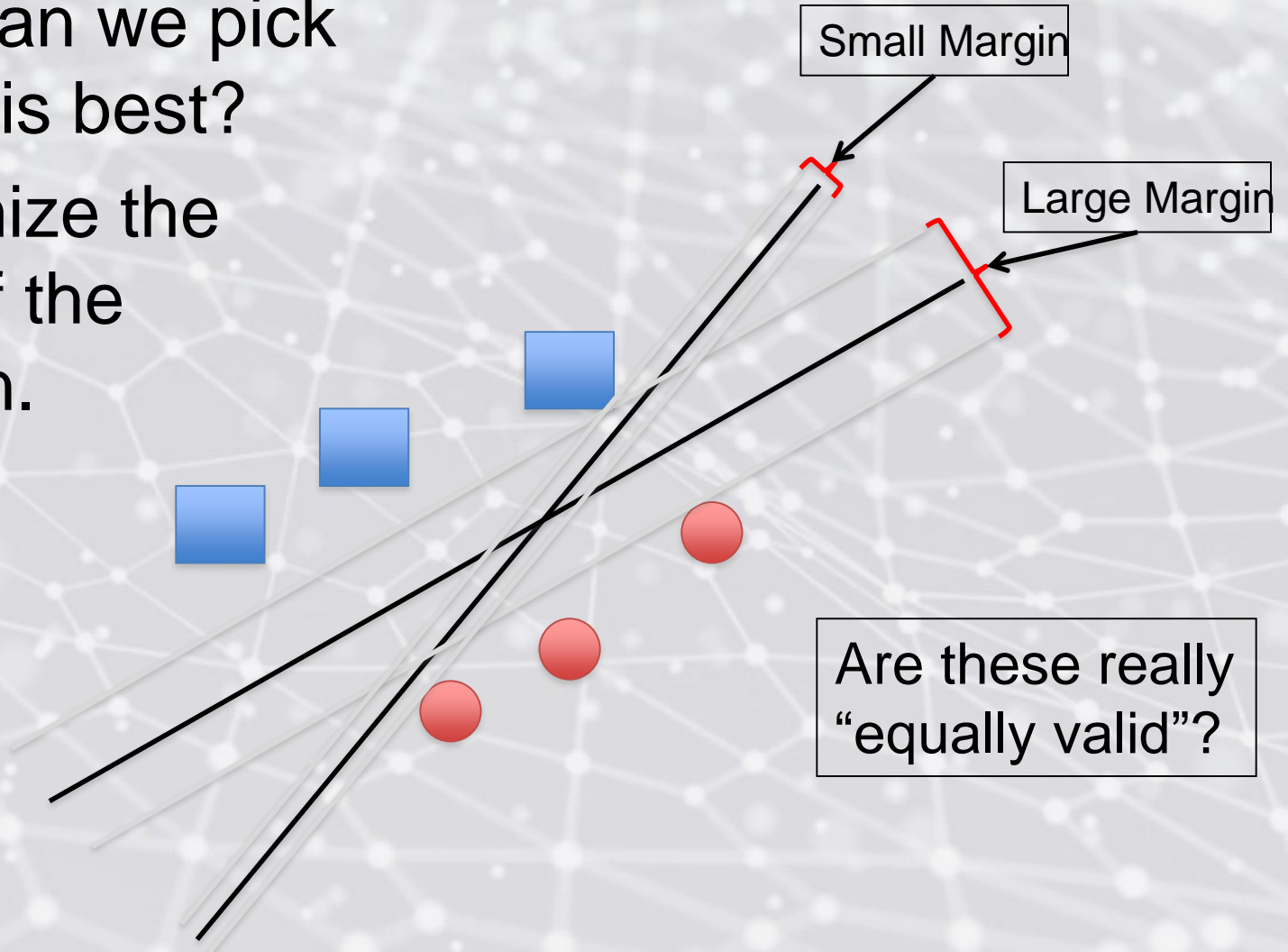
Maximum Margin

- Linear Classifiers can lead to many equally valid choices for the decision boundary



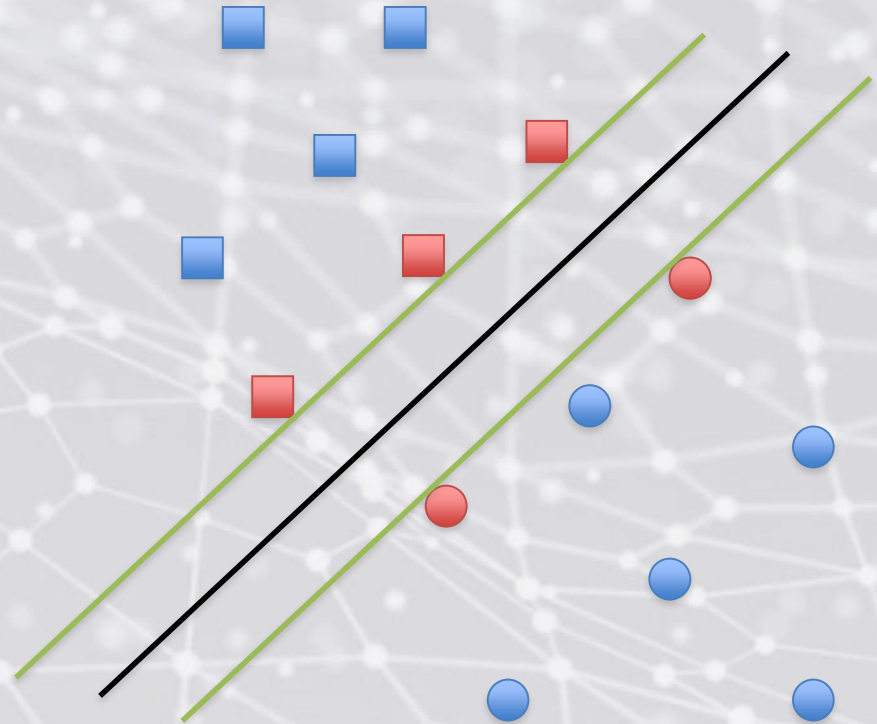
Max Margin

- How can we pick which is best?
- Maximize the size of the margin.



Support Vectors

- Support Vectors are those input points (vectors) closest to the decision boundary
- 1. They are vectors
- 2. They “support” the decision hyperplane

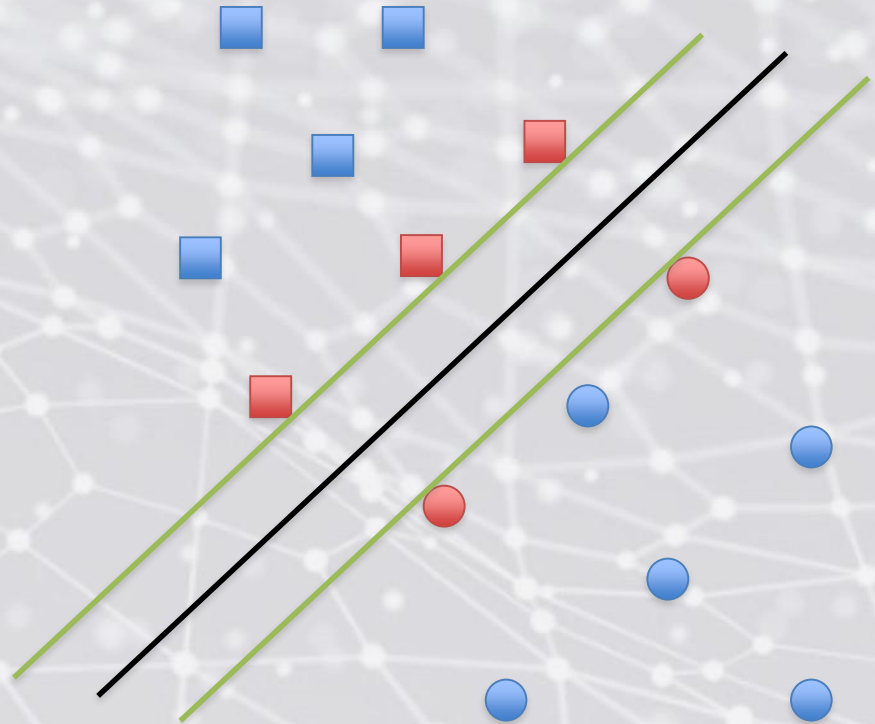


Support Vectors

- Define this as a decision problem
- The decision hyperplane:

$$\vec{w}^T \vec{x} + b = 0$$

- No fancy math, just the equation of a hyperplane.

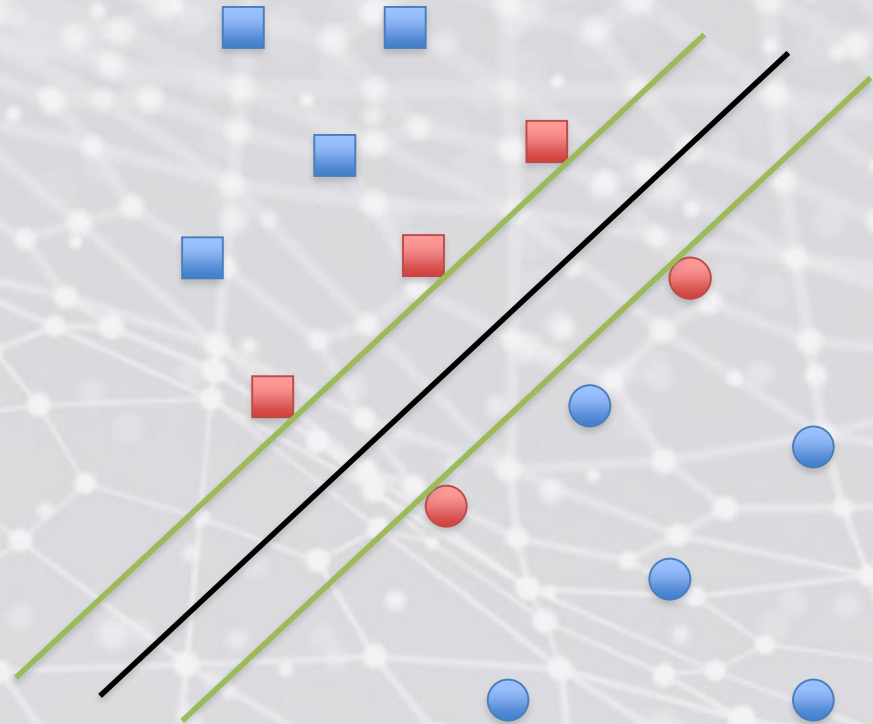


Support Vectors

\vec{x}_i are the data

$\vec{t}_i \in \{-1, +1\}$ are the labels

- **Aside: Why do some classifiers use $t_i \in \{0, 1\}$ or $t_i \in \{-1, +1\}$**
 - Simplicity of the math and interpretation.
 - For probability density function estimation 0,1 has a clear correlate.
 - For classification, a decision boundary of 0 is more easily interpretable than .5.



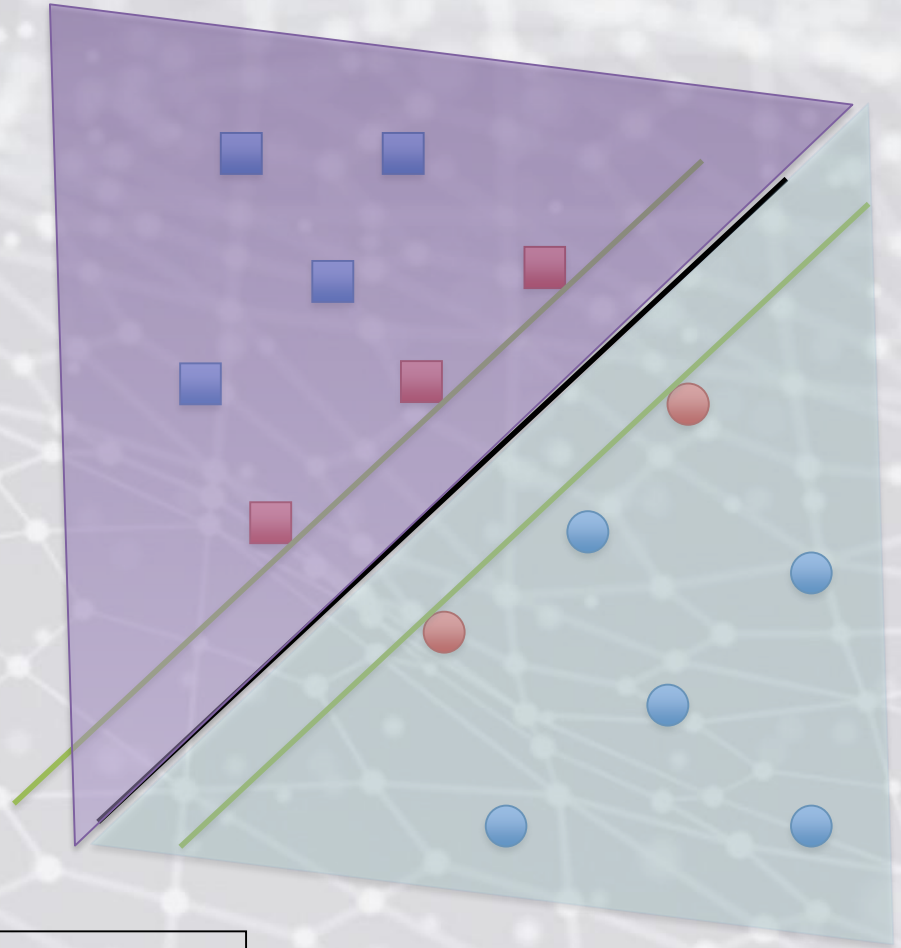
Support Vectors

- Define this as a decision problem
- The decision hyperplane:

$$\vec{w}^T \vec{x} + b = 0$$

- Decision Function:

$$D(\vec{x}_i) = \text{sign}(\vec{w}^T \vec{x}_i + b)$$



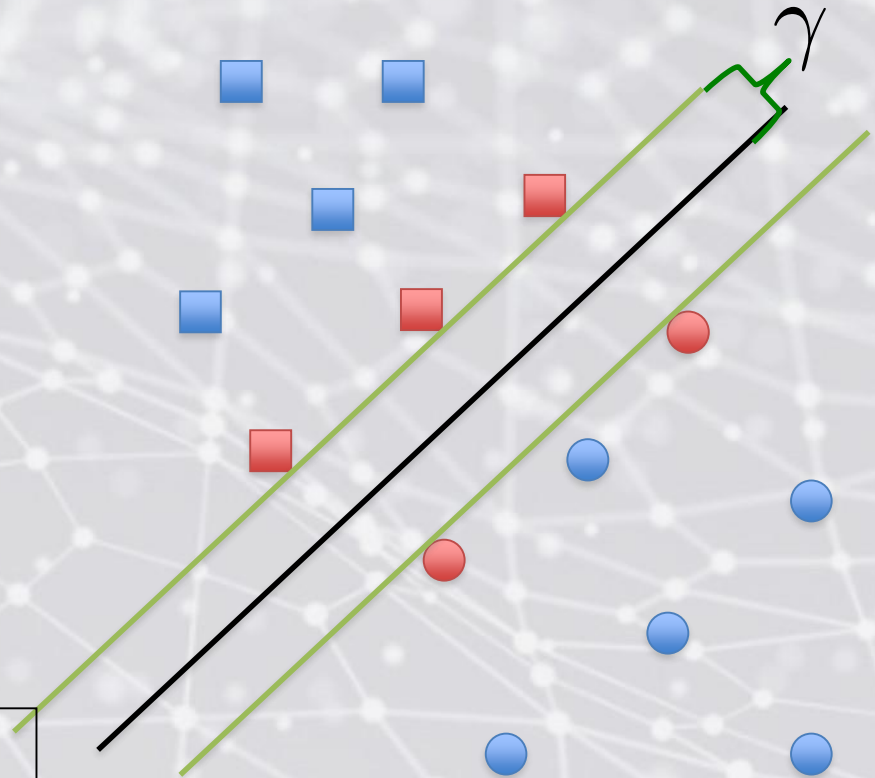
Support Vectors

- Define this as a decision problem
- The decision hyperplane:

$$\vec{w}^T \vec{x} + b = 0$$

- Margin hyperplanes:

$$\begin{aligned}\vec{w}^T \vec{x} + b &= \gamma \\ \vec{w}^T \vec{x} + b &= -\gamma\end{aligned}$$



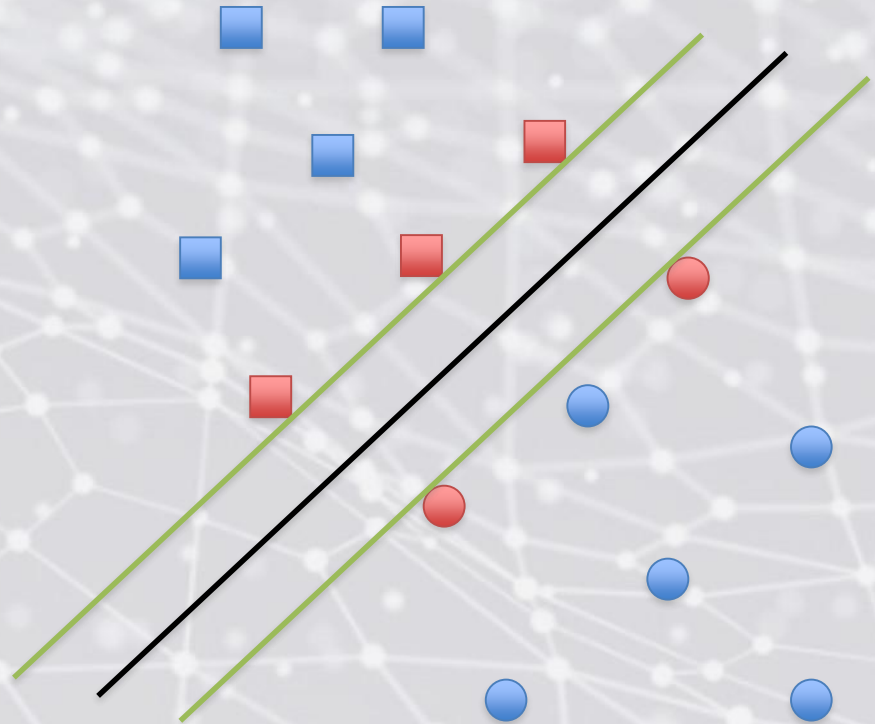
Support Vectors

- The decision hyperplane:

$$\vec{w}^T \vec{x} + b = 0$$

- Scale invariance

$$c\vec{w}^T \vec{x} + cb = 0$$



Support Vectors

- The decision hyperplane:

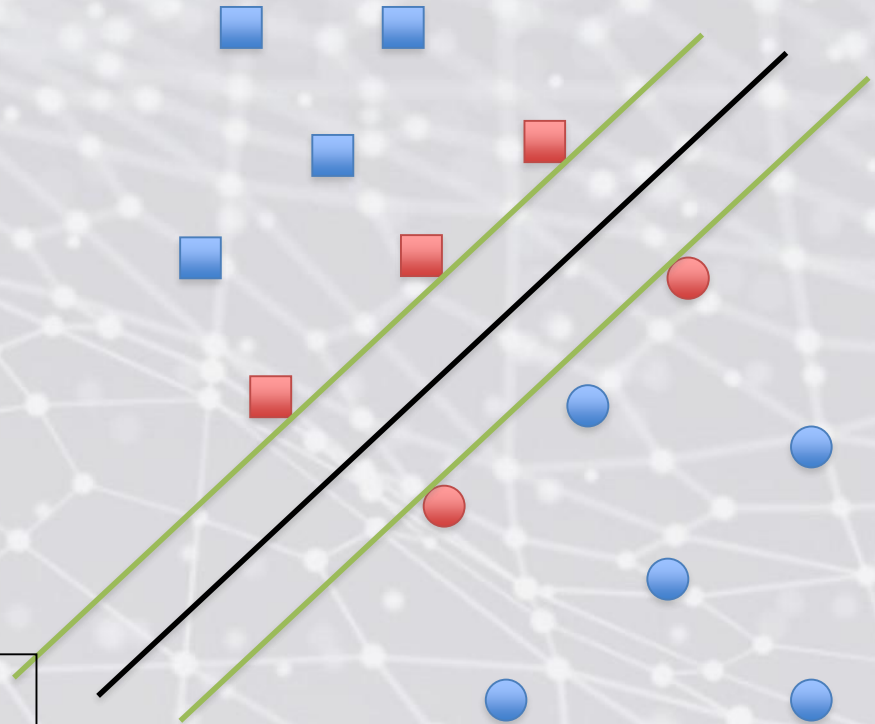
$$\vec{w}^T \vec{x} + b = 0$$

- Scale invariance

$$c\vec{w}^T \vec{x} + cb = 0$$

$$\vec{w}^T \vec{x} + b = \gamma$$

$$\vec{w}^T \vec{x} + b = -\gamma$$



Support Vectors

- The decision hyperplane:

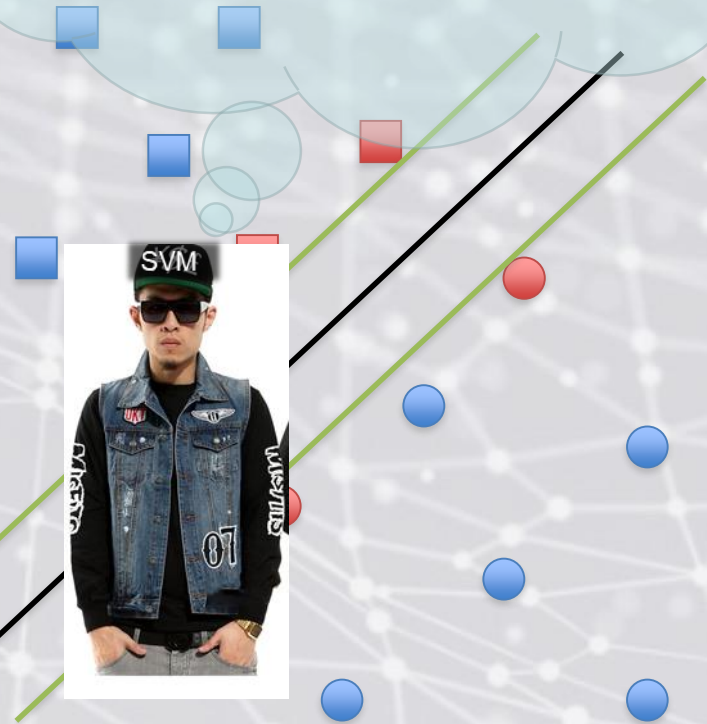
$$\vec{w}^T \vec{x} + b = 0$$

- Scale invariance

$$c\vec{w}^T \vec{x} + cb = 0$$

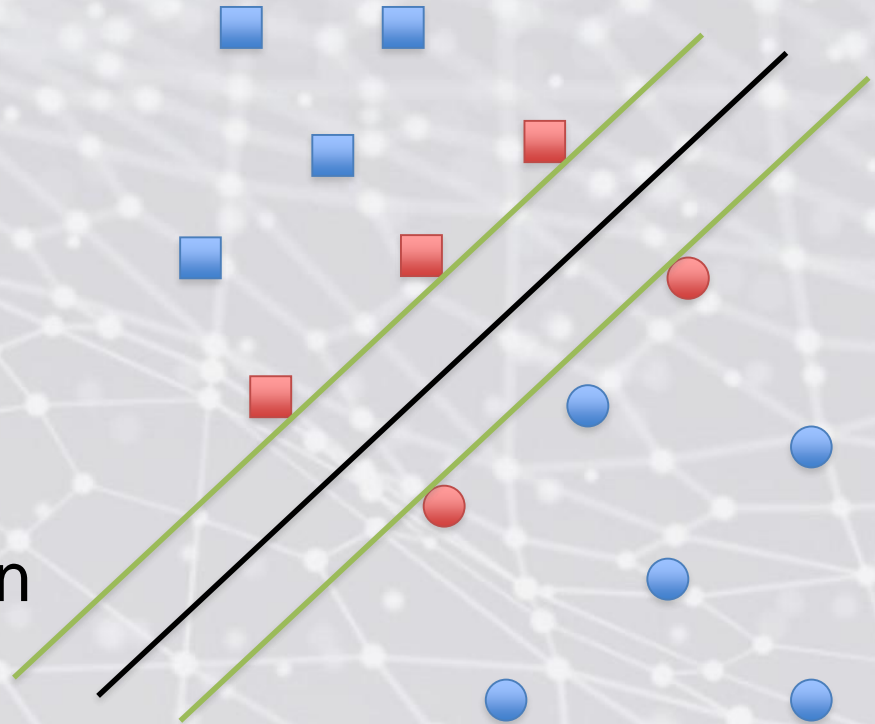
$$\begin{aligned} \vec{w}^{*T} \vec{x} + b^* &= 1 \\ \vec{w}^{*T} \vec{x} + b^* &= -1 \end{aligned}$$

This scaling does not change the decision hyperplane, or the support vector hyperplanes. But we will eliminate a variable from the optimization



What are we optimizing?

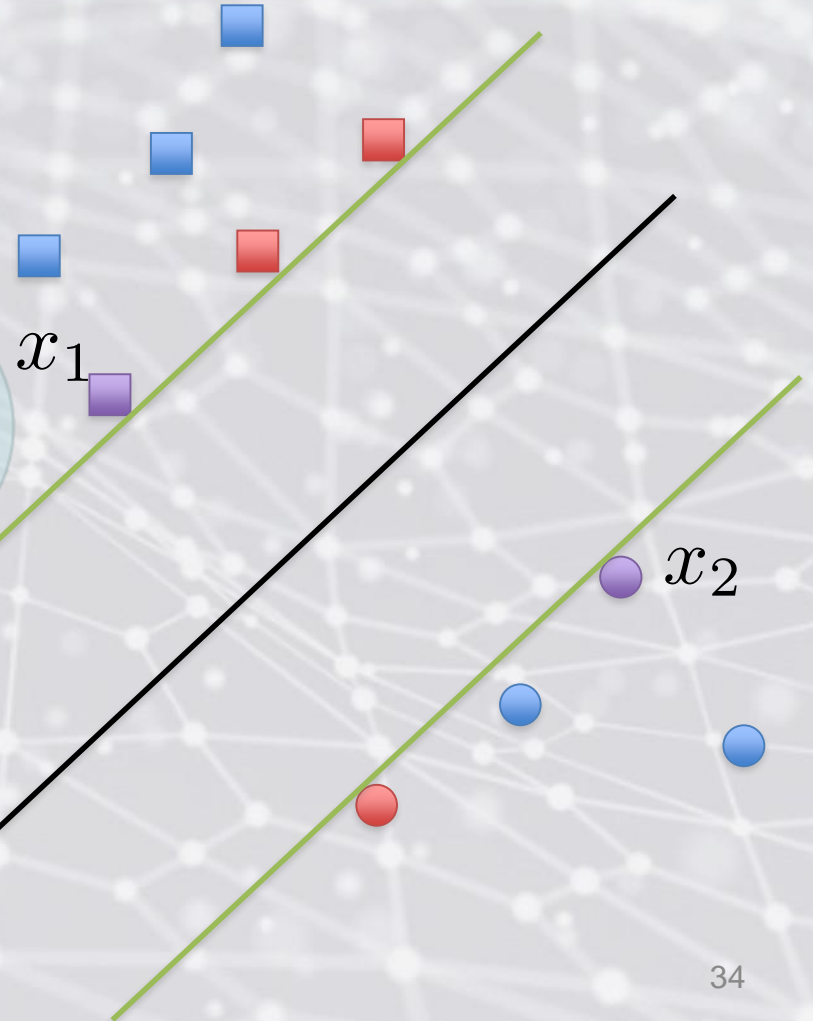
- We will represent the size of the margin in terms of w .
- This will allow us to simultaneously
 - Identify a decision boundary
 - Maximize the margin



How do we represent the size of the margin in terms of w ?

1. There must at least one point that lies on each support hyperplanes

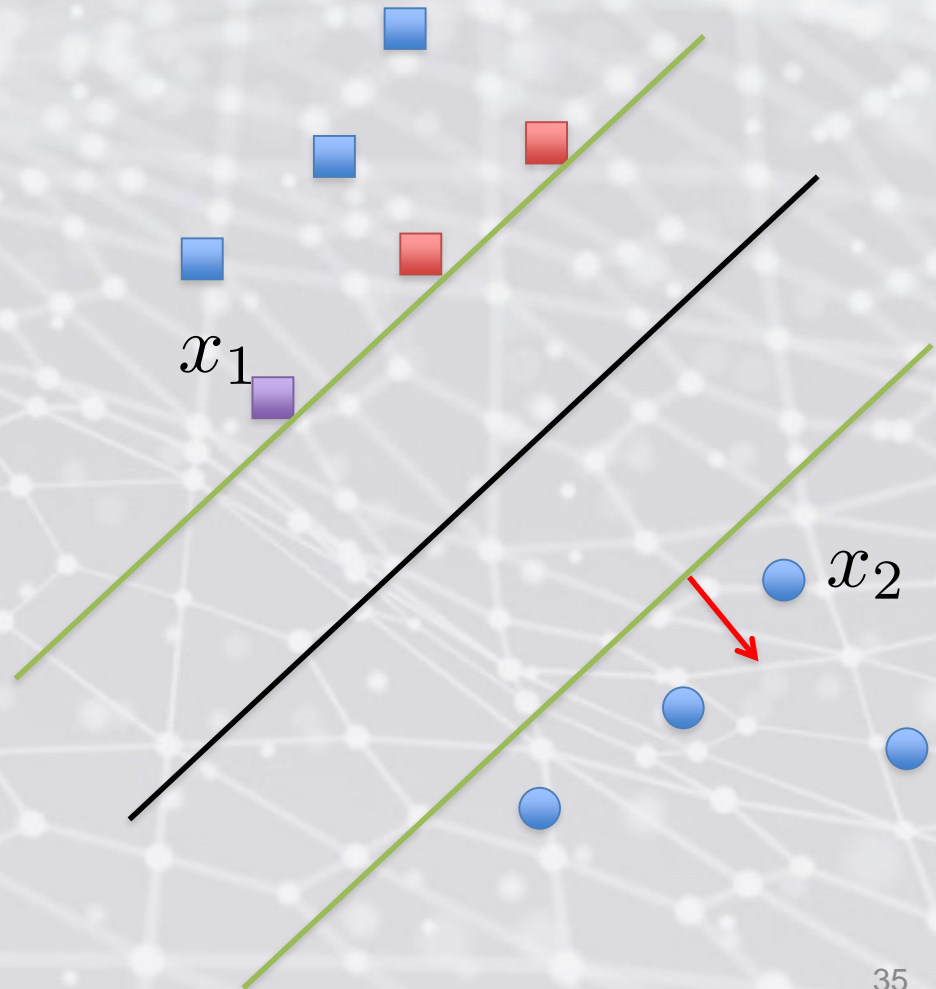
Proof outline: If not, we could define a larger margin support hyperplane that *does* touch the nearest point(s).



How do we represent the size of the margin in terms of w ?

1. There must at least one point that lies on each support hyperplanes

Proof outline: If not, we could define a larger margin support hyperplane that *does* touch the nearest point(s).



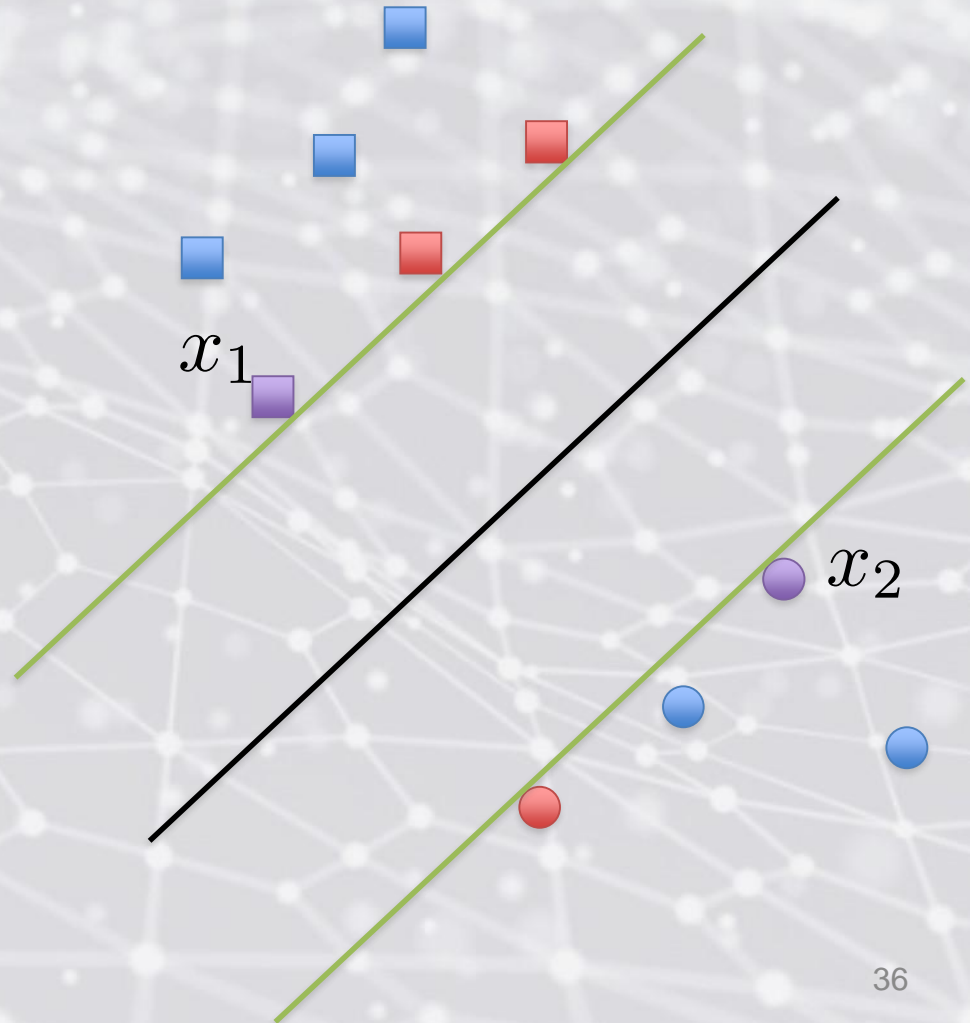
How do we represent the size of the margin in terms of w ?

1. There must at least one point that lies on each support hyperplanes
2. Thus:

$$\begin{aligned} w^T x_1 + b &= 1 \\ w^T x_2 + b &= -1 \end{aligned}$$

3. And:

$$w^T (x_1 - x_2) = 2$$



How do we represent the size of the margin in terms of w ?

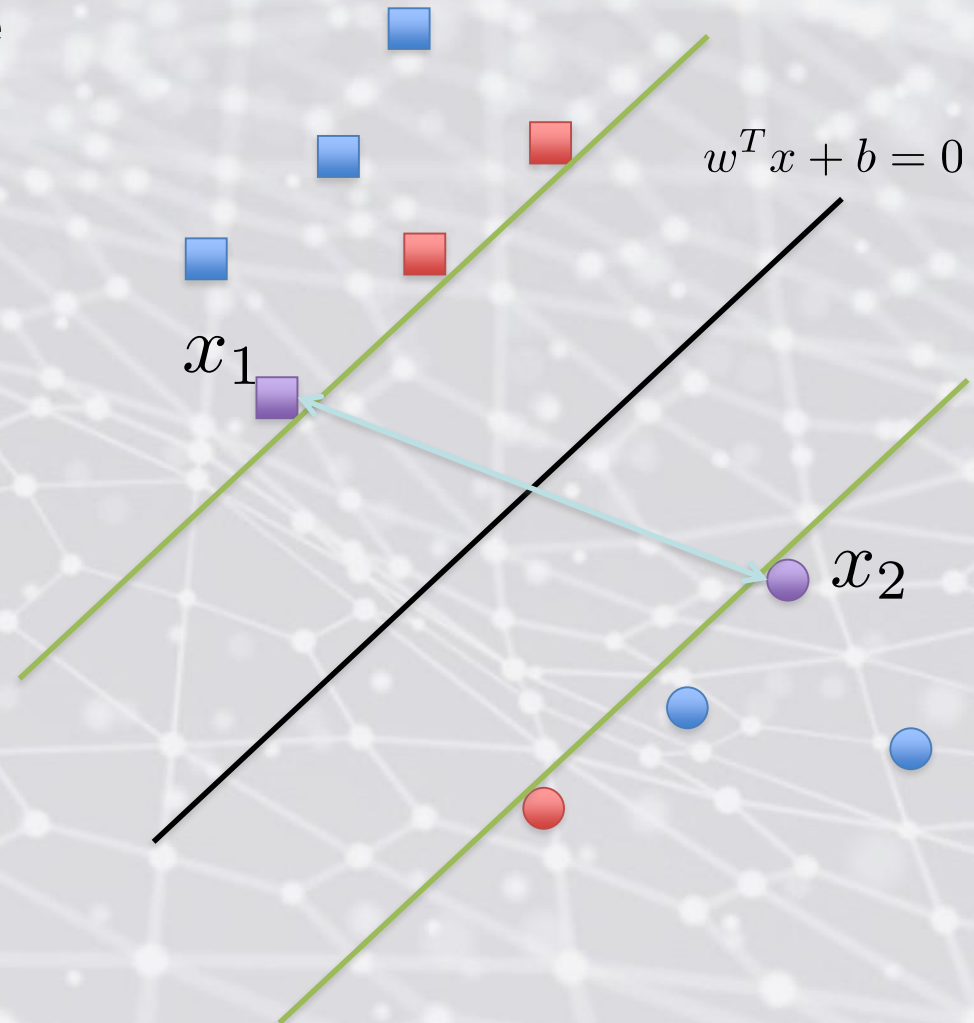
1. There must at least one point that lies on each support hyperplanes
2. Thus:

$$\begin{array}{rcl} w^T x_1 + b & = & 1 \\ w^T x_2 + b & = & -1 \end{array}$$

3. And:

$$w^T (x_1 - x_2) = 2$$

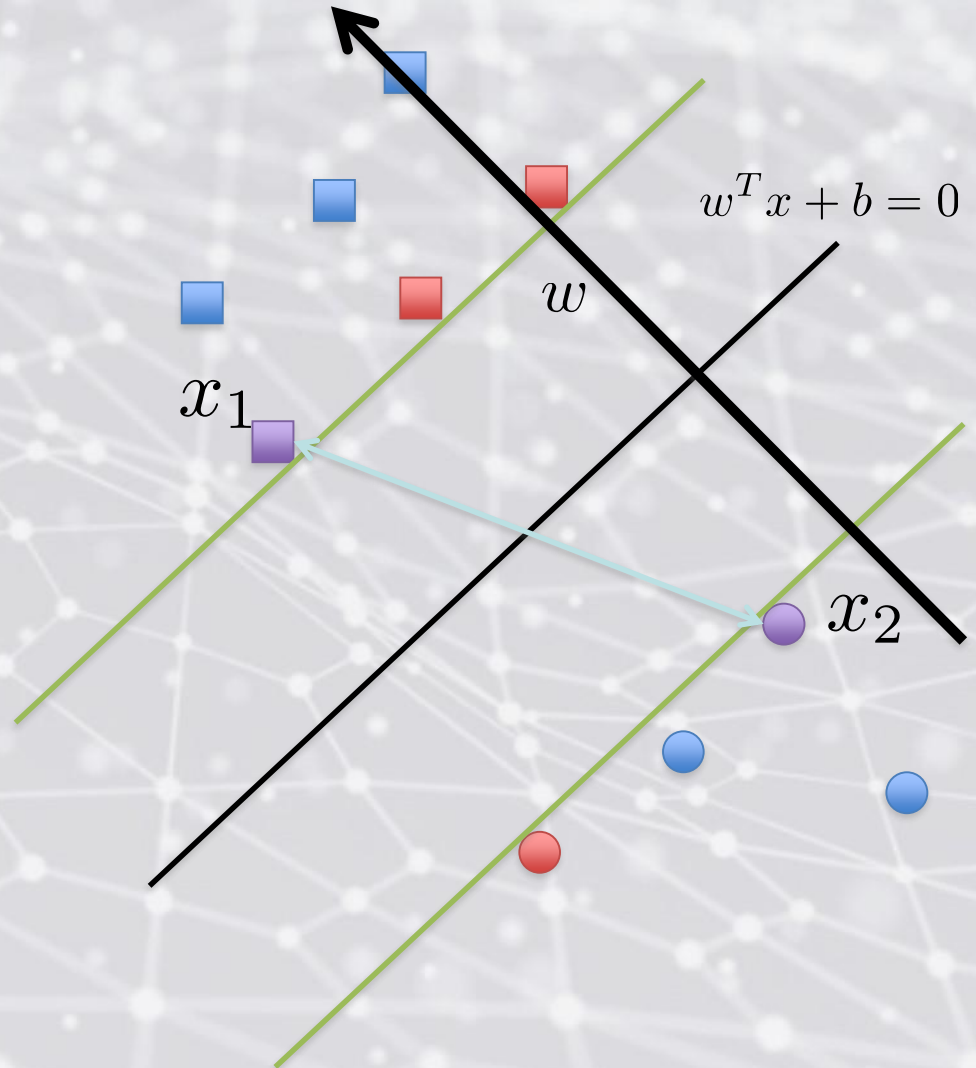
Why?



How do we represent the size of the margin in terms of w ?

- The vector w is perpendicular to the decision hyperplane
 - If the dot product of two vectors equals zero, the two vectors are perpendicular.

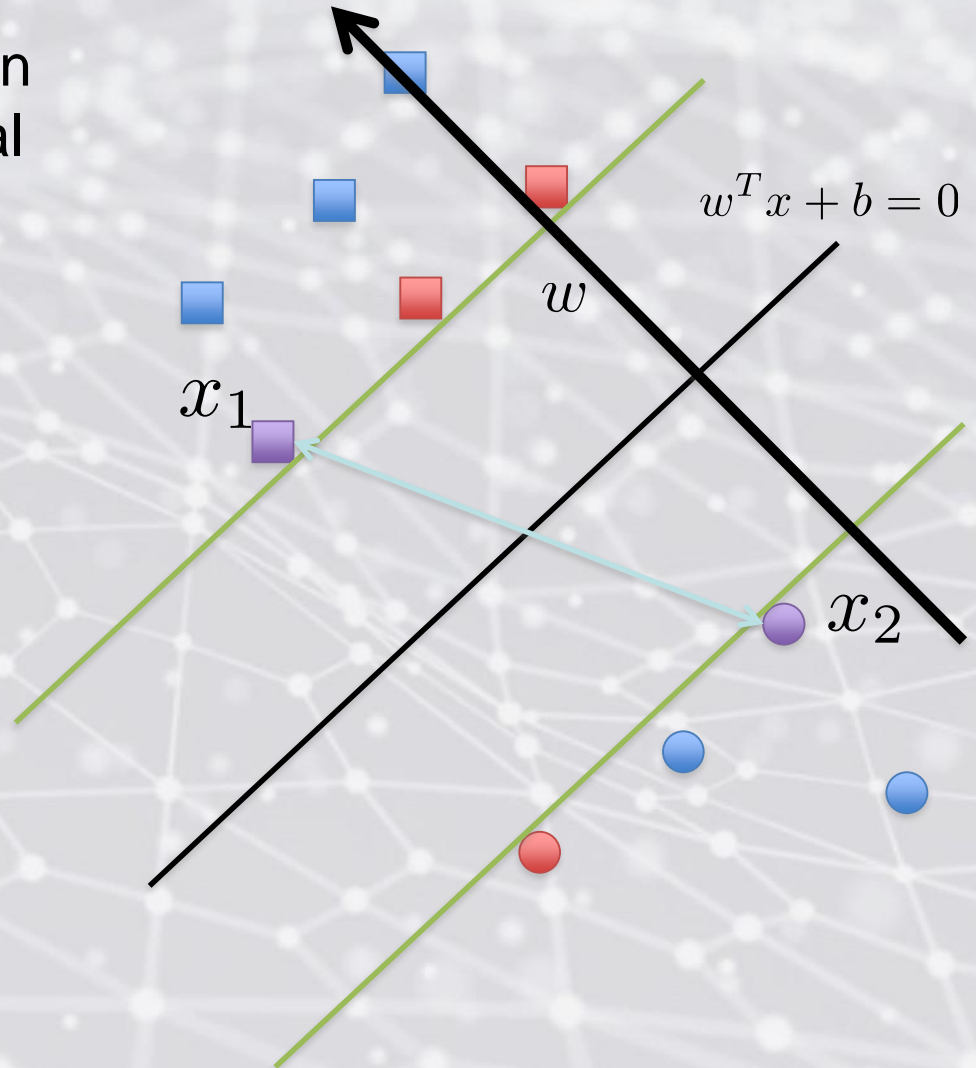
$$w^T (x_1 - x_2) = 2$$



How do we represent the size of the margin in terms of w ?

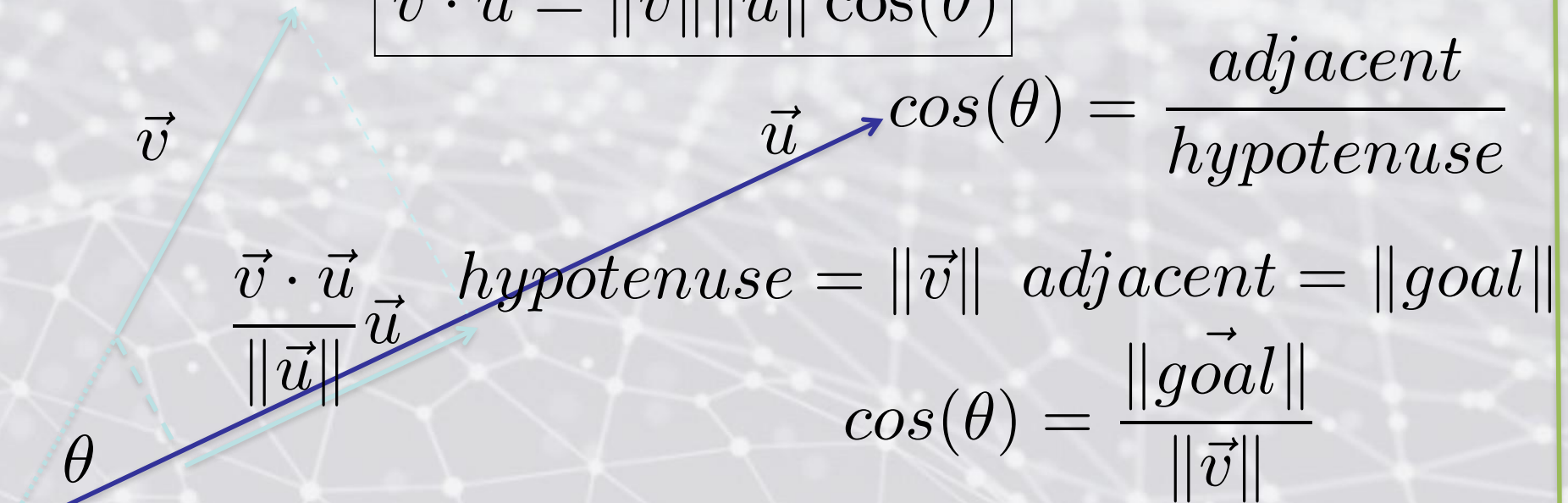
- The **margin** is the projection of $x_1 - x_2$ onto w , the normal of the hyperplane.

$$w^T (x_1 - x_2) = 2$$



Aside: Vector Projection

$$\vec{v} \cdot \vec{u} = \|\vec{v}\| \|\vec{u}\| \cos(\theta)$$



$$\frac{\vec{v} \cdot \vec{u}}{\|\vec{u}\|} \vec{u} \quad \text{hypotenuse} = \|\vec{v}\| \quad \text{adjacent} = \|\vec{goal}\|$$

$$\cos(\theta) = \frac{\|\vec{goal}\|}{\|\vec{v}\|}$$

$$\frac{\vec{v} \cdot \vec{u}}{\|\vec{u}\|} = \|\vec{goal}\|$$

$$\frac{\|\vec{v}\| \|\vec{u}\|}{\|\vec{v}\| \|\vec{u}\|} \cos(\theta) = \frac{\|\vec{goal}\|}{\|\vec{v}\|}$$

$$\frac{\vec{v} \cdot \vec{u}}{\|\vec{v}\| \|\vec{u}\|} = \frac{\|\vec{goal}\|}{\|\vec{v}\|}$$

How do we represent the size of the margin in terms of w ?

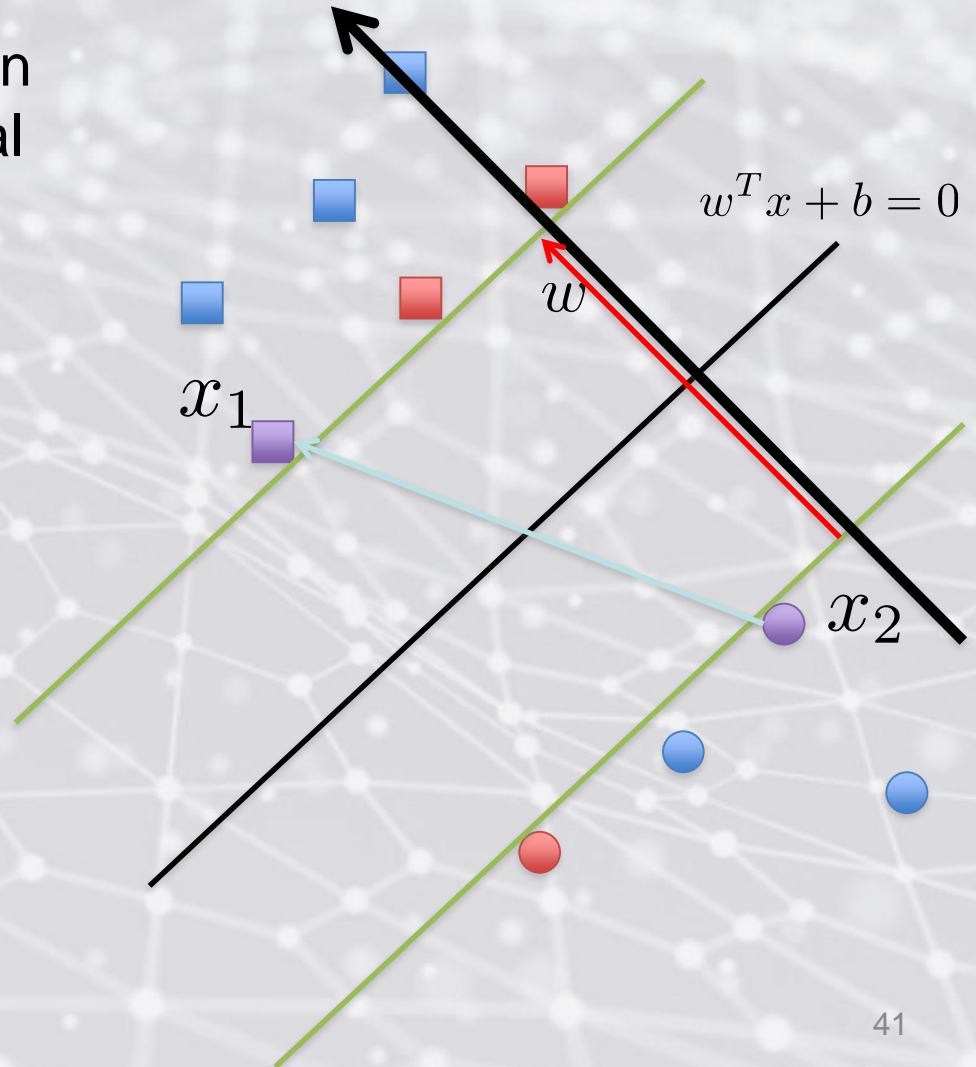
- The **margin** is the projection of $x_1 - x_2$ onto w , the normal of the hyperplane.

$$w^T (x_1 - x_2) = 2$$

Projection: $\frac{\vec{v} \cdot \vec{u}}{\|\vec{u}\|} \vec{u}$

$$\frac{\vec{w}^T (x_1 - x_2)}{\|\vec{w}\|} \vec{w}$$

Size of the Margin: $\frac{2}{\|\vec{w}\|}$



Maximizing the margin

- Goal: maximize the margin:
- This is equivalent to finding:

$$\max \frac{2}{\|\vec{w}\|}$$



$$\min \|\vec{w}\|$$

Linear Separability of the data
by the decision boundary

$$\text{where } t_i(\vec{w}^T x_i + b) \geq 1$$

$$\vec{w}^T x_i + b \geq 1 \text{ if } t_i = 1$$

$$\vec{w}^T x_i + b \leq -1 \text{ if } t_i = -1$$

Minimizing $\|\mathbf{w}\|$

The canonical SVM optimization problem is formalized as:

$$\text{minimize } \left(\frac{1}{2} \|\mathbf{w}\|^2 \right) \text{ subject to: } t_k (\mathbf{w} \cdot \mathbf{x}_k + b) \geq 1, \quad k = 1, \dots, m$$

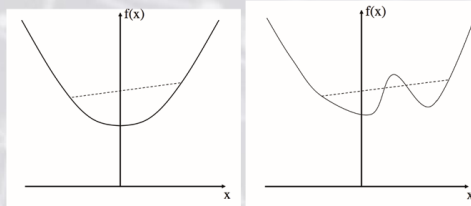
- Note that Gradient Descent would be ill-suited for this problem (enforcing the constraints is difficult).
- Instead, the method of **Quadratic Programming (QP)** is more efficient.

Minimizing $\|\mathbf{w}\|$

The canonical SVM optimization problem is formalized as:

$$\text{minimize } \left(\frac{1}{2} \|\mathbf{w}\|^2 \right) \text{ subject to: } t_k (\mathbf{w} \cdot \mathbf{x}_k + b) \geq 1, \quad k = 1, \dots, m$$

- Because this problem is quadratic, it is necessarily *convex*! Thus it admits of a unique solution (minimum).



- We won't worry about all of the details here, but we'd like to reformulate this problem using *Lagrange multipliers* at which point it can then be presented to a *quadratic solver* (in general the solution yielded is rendered in polynomial time).

Dual Representation

- It turns out (as Neumann first observed) that \mathbf{w} can be expressed as a linear combination of the training examples:

$$\mathbf{w} = \sum_{\mathbf{x}_k \in S} a_k \mathbf{x}_k$$

where $a_k \geq 0$ only if \mathbf{x}_k is a support vector

- The results of the SVM training algorithm (involving solving a quadratic programming problem) are the α_k and the bias b .

SVM Classification

- After solving the “dual problem” (i.e. obtaining the α_k and the bias b), we classify a new example \mathbf{x} as follows:

$$h(\mathbf{x}) = \text{sgn} \left(\sum_{k=1}^M \alpha_k (\mathbf{x} \cdot \mathbf{x}_k) + b \right)$$

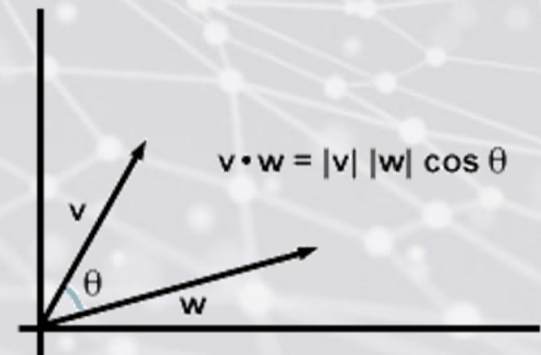
- Where $|S| = M$. (S is the training set)

SVM Classification

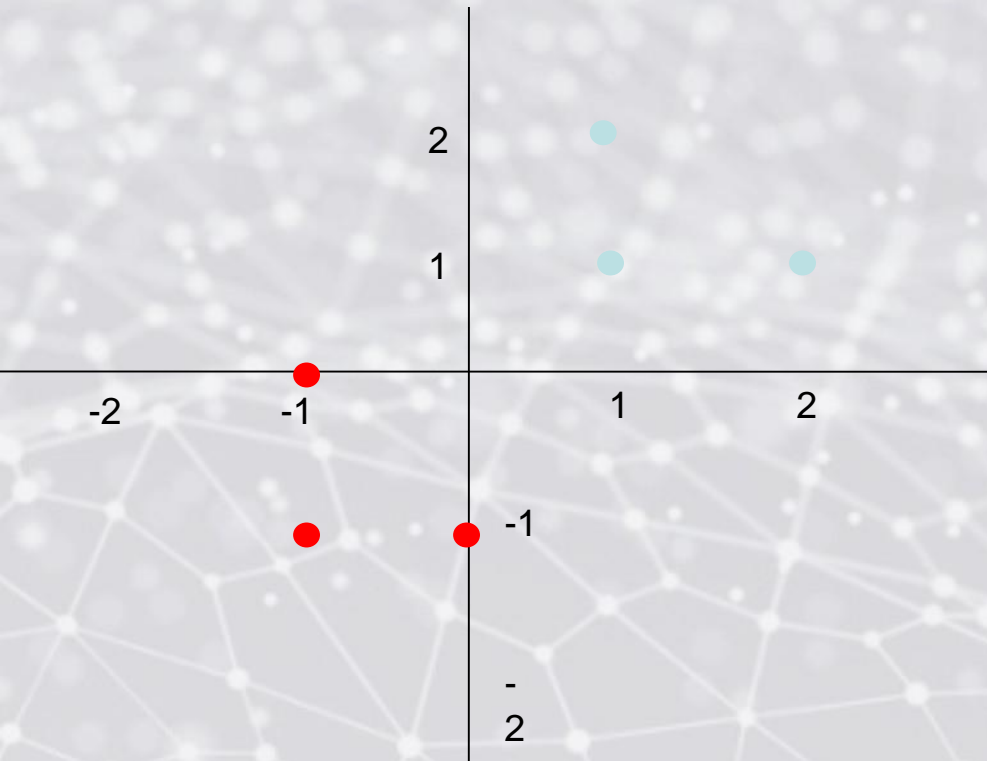
- After solving the “dual problem” (i.e. obtaining the α_k and the bias b), we classify a new example \mathbf{x} as follows:

$$h(\mathbf{x}) = \text{sgn} \left(\sum_{k=1}^M \alpha_k (\mathbf{x} \cdot \mathbf{x}_k) + b \right)$$

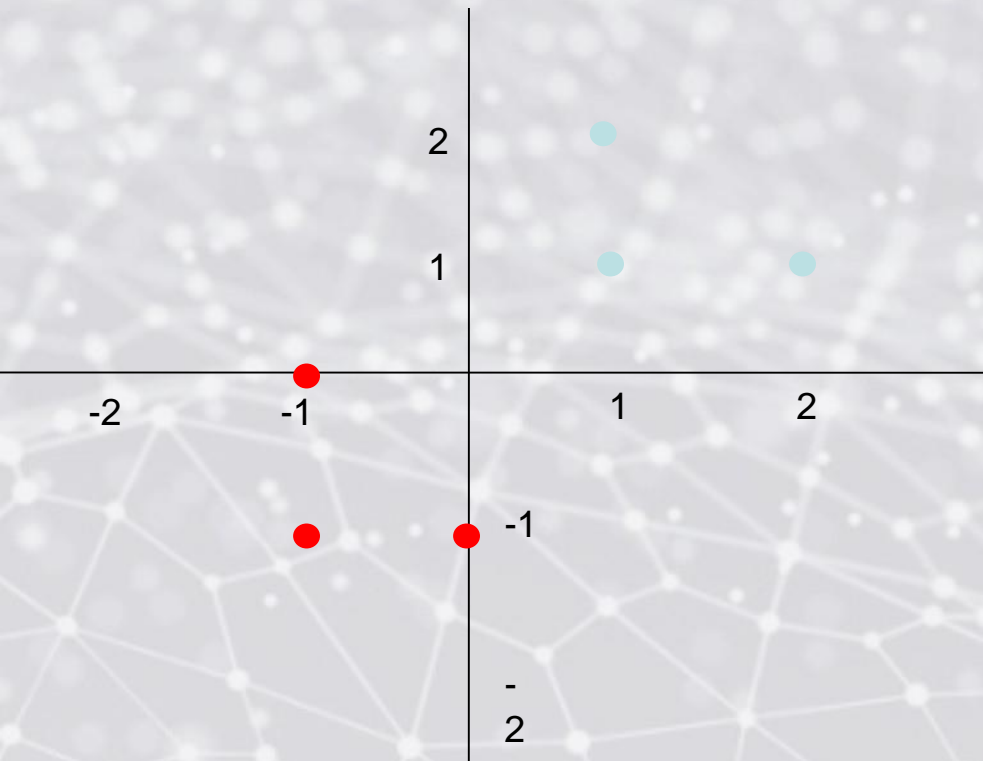
- where $|S| = M$. (S is the training set)
- Recall that the dot product furnishes a “similarity” measure.



Example



Example



Input to SVM optimizer:

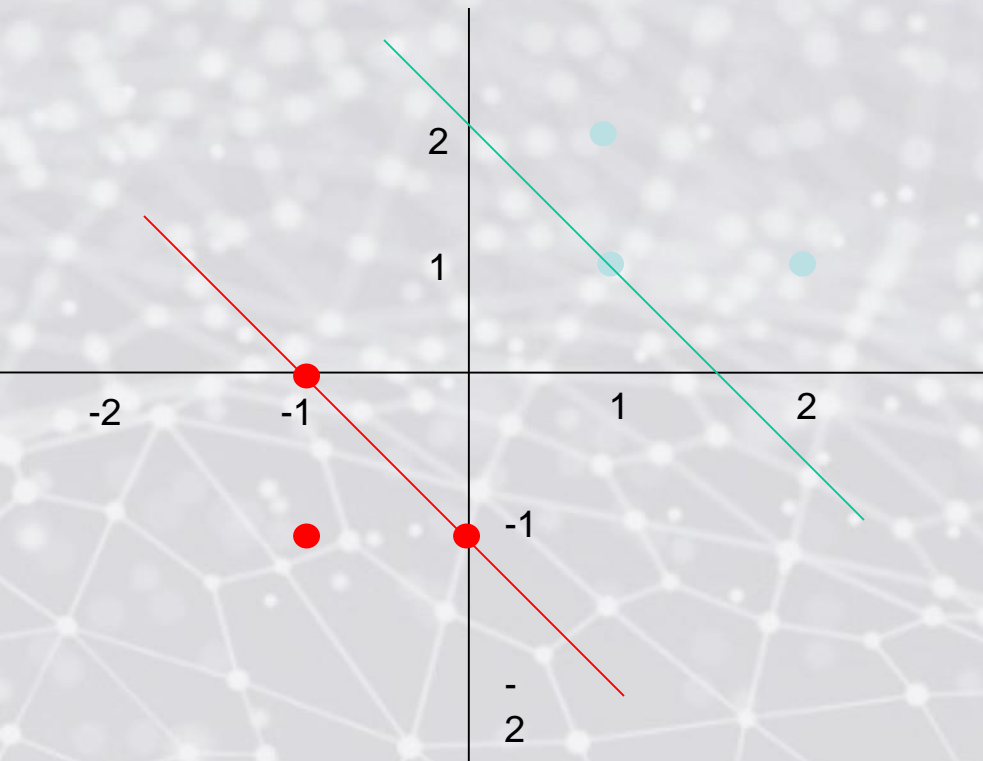
x_1	x_2	class
1	1	1
1	2	1
2	1	1
-1	0	-1
0	-1	-1
-1	-1	-1

Output from SVM optimizer:

Support vector	α
$(-1, 0)$	-.208
$(1, 1)$.416
$(0, -1)$	-.208

$$b = -.376$$

Example



Input to SVM optimizer:

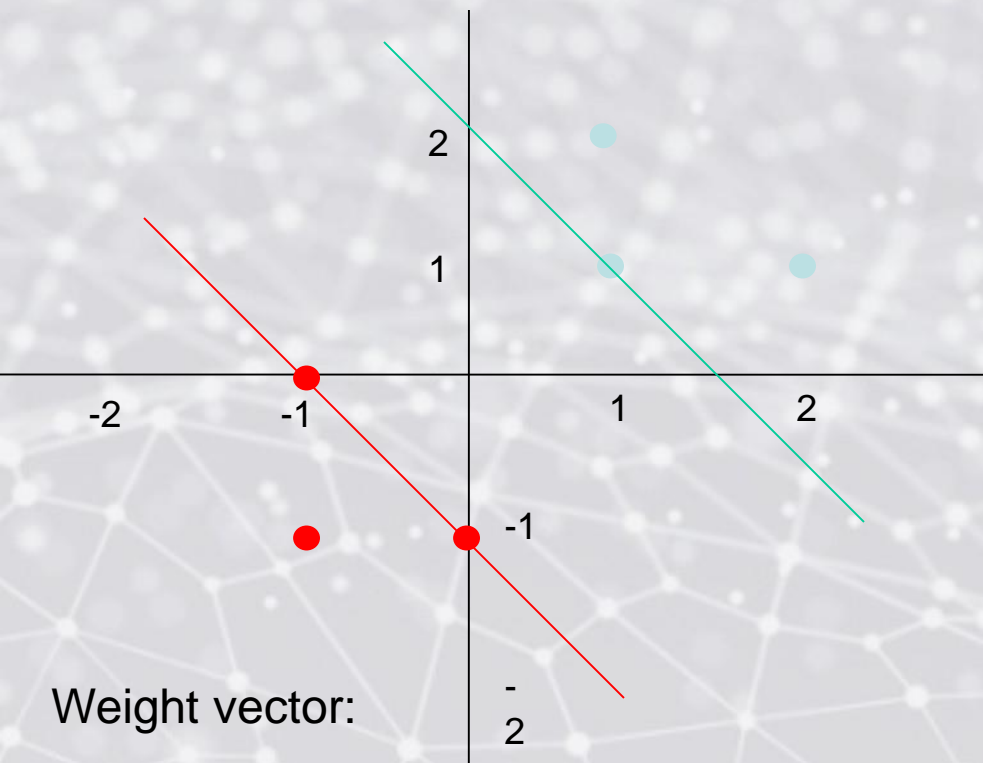
x_1	x_2	class
1	1	1
1	2	1
2	1	1
-1	0	-1
0	-1	-1
-1	-1	-1

Output from SVM optimizer:

Support vector	α
$(-1, 0)$	-.208
$(1, 1)$.416
$(0, -1)$	-.208

$$b = -.376$$

Example



Weight vector:

$$\mathbf{w} = \sum_{k \in \text{training examples}} a_k \mathbf{x}_k$$

$$= -.208(-1, 0) + .416(1, 1) - .208(0, -1)$$

$$= (.624, .624)$$

Input to SVM optimizer:

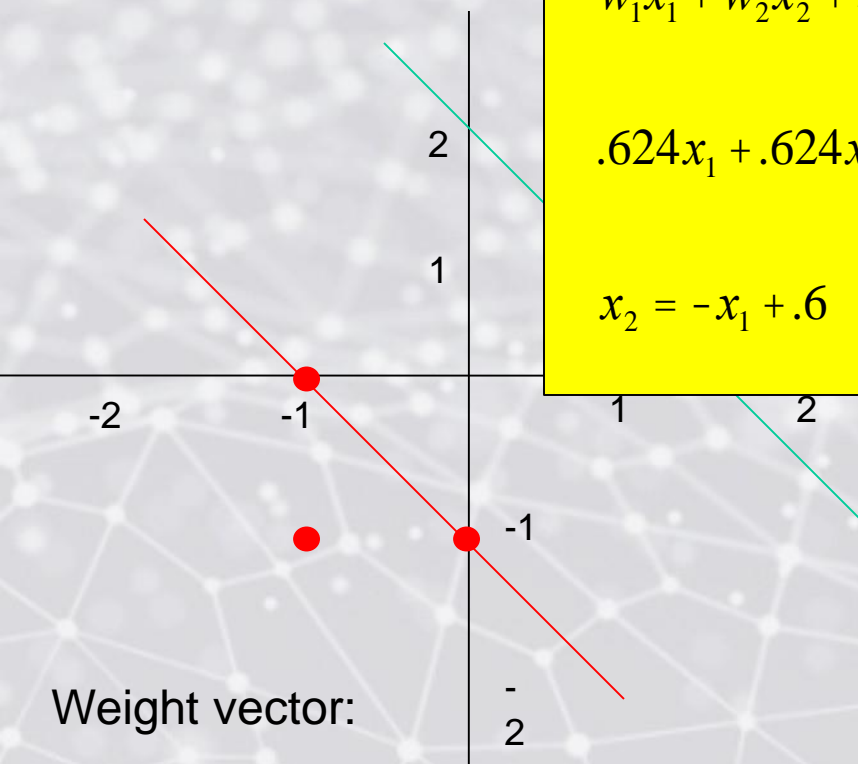
x_1	x_2	class
1	1	1
1	2	1
2	1	1
-1	0	-1
0	-1	-1
-1	-1	-1

Output from SVM optimizer:

Support vector	α
$(-1, 0)$	$-.208$
$(1, 1)$	$.416$
$(0, -1)$	$-.208$

$$b = -.376$$

Example



Weight vector:

$$\mathbf{w} = \sum_{k \in \{\text{training examples}\}} \alpha_k \mathbf{x}_k$$

$$= -.208(-1, 0) + .416(1, 1) - .208(0, -1)$$

$$= (.624, .624)$$

Separation line:

$$w_1x_1 + w_2x_2 + b = 0$$

$$.624x_1 + .624x_2 - .376 = 0$$

$$x_2 = -x_1 + .6$$

Input to SVM optimizer:

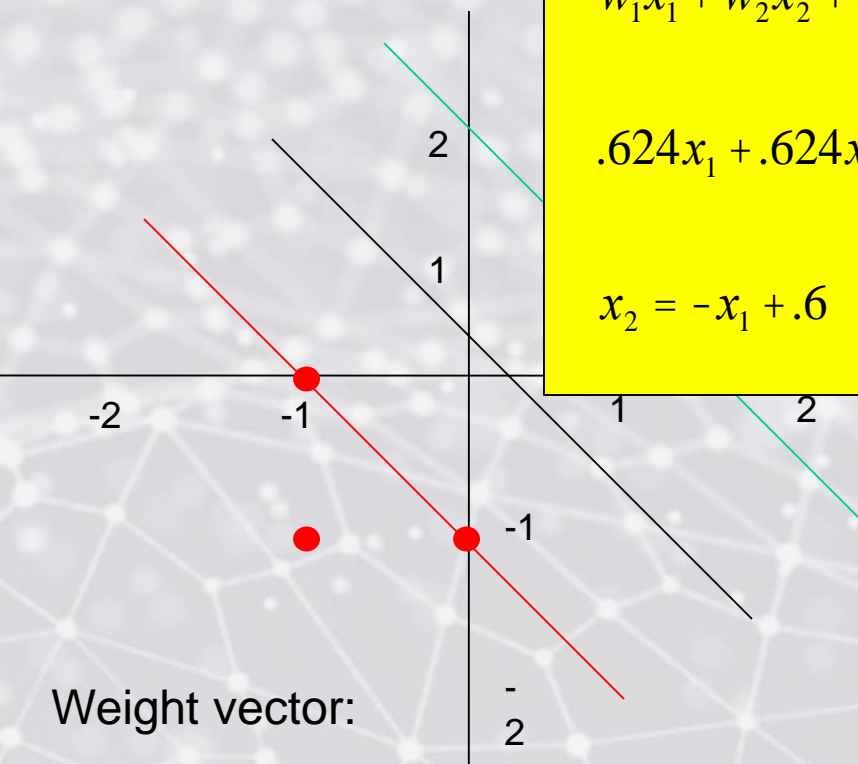
x_1	x_2	class
1	1	1
1	2	1
2	1	1
-1	0	-1
0	-1	-1
-1	-1	-1

Output from SVM optimizer:

Support vector	α
$(-1, 0)$	$-.208$
$(1, 1)$	$.416$
$(0, -1)$	$-.208$

$$b = -.376$$

Example



Weight vector:

$$\mathbf{w} = \sum_{k \in \text{training examples}} \alpha_k \mathbf{x}_k$$

$$= -.208(-1, 0) + .416(1, 1) - .208(0, -1)$$

$$= (.624, .624)$$

Separation line:

$$w_1x_1 + w_2x_2 + b = 0$$

$$.624x_1 + .624x_2 - .376 = 0$$

$$x_2 = -x_1 + .6$$

Input to SVM optimizer:

x_1	x_2	class
1	1	1
1	2	1
2	1	1
-1	0	-1
0	-1	-1
-1	-1	-1

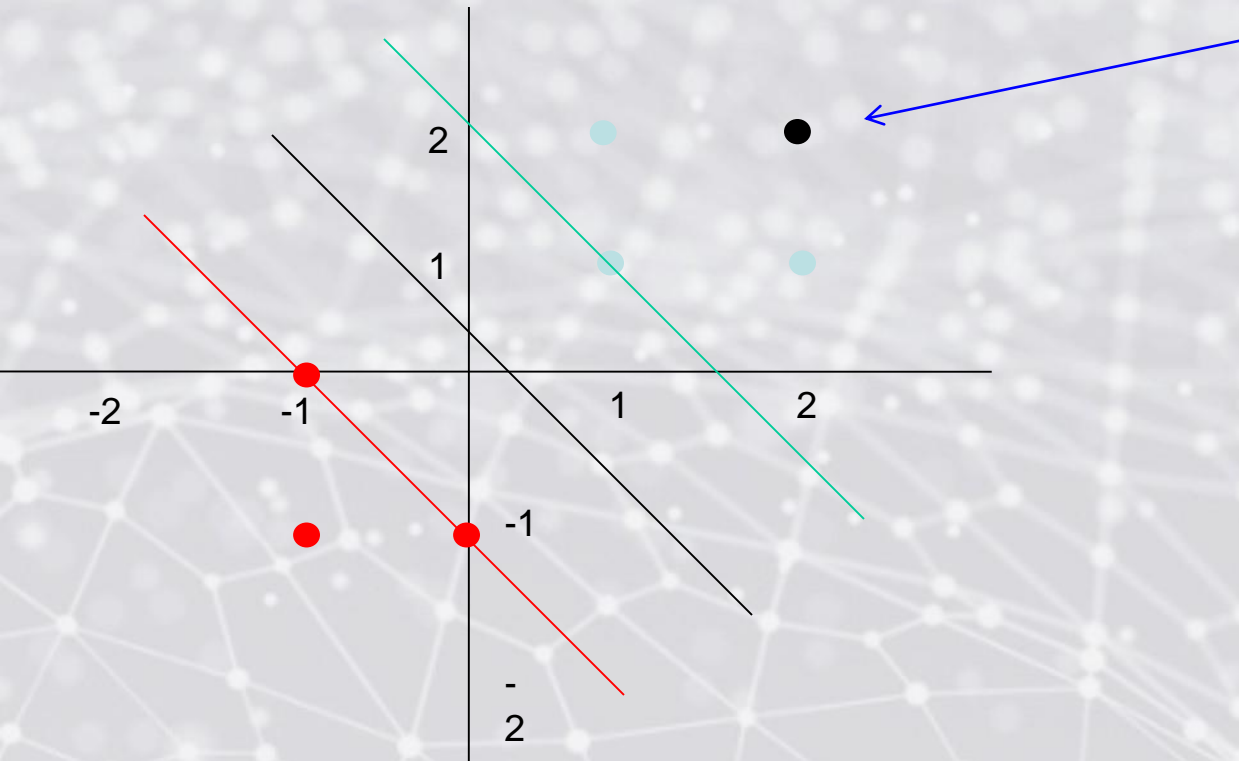
Output from SVM optimizer:

Support vector	α
$(-1, 0)$	$-.208$
$(1, 1)$	$.416$
$(0, -1)$	$-.208$

$$b = -.376$$

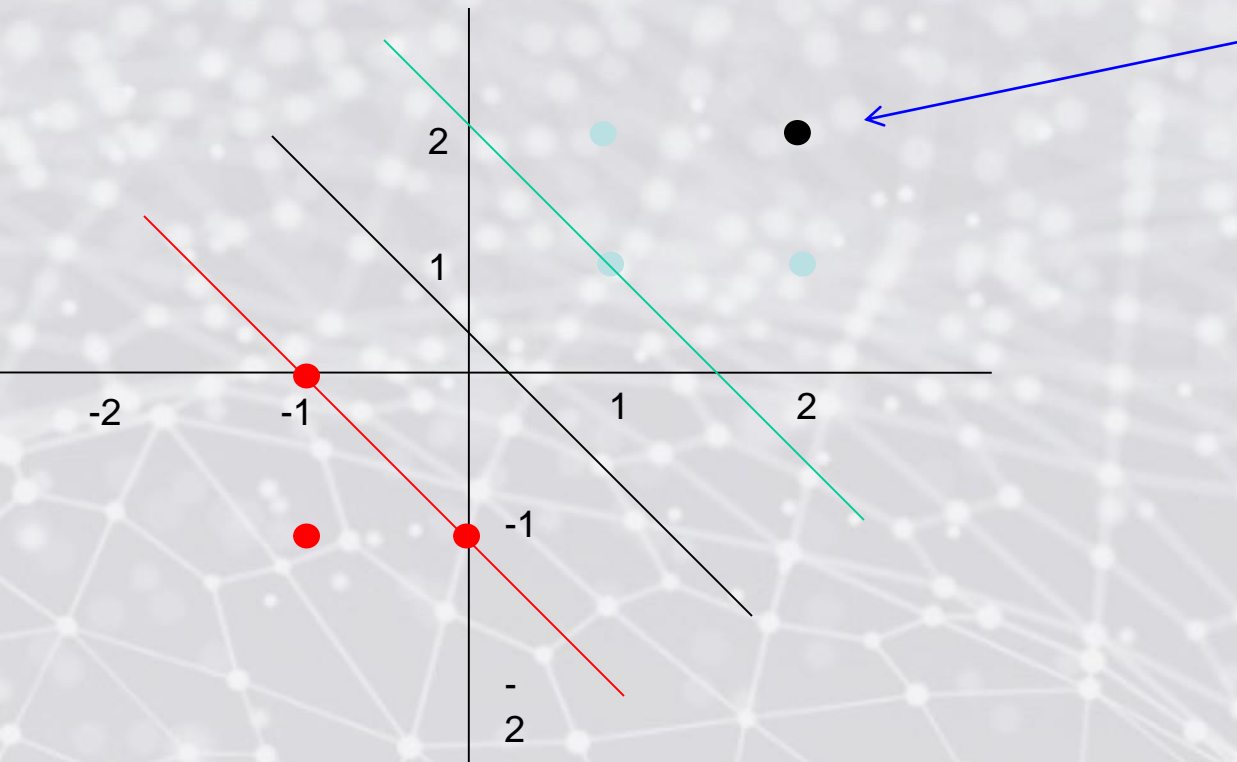
Example

Classifying a new point:



Example

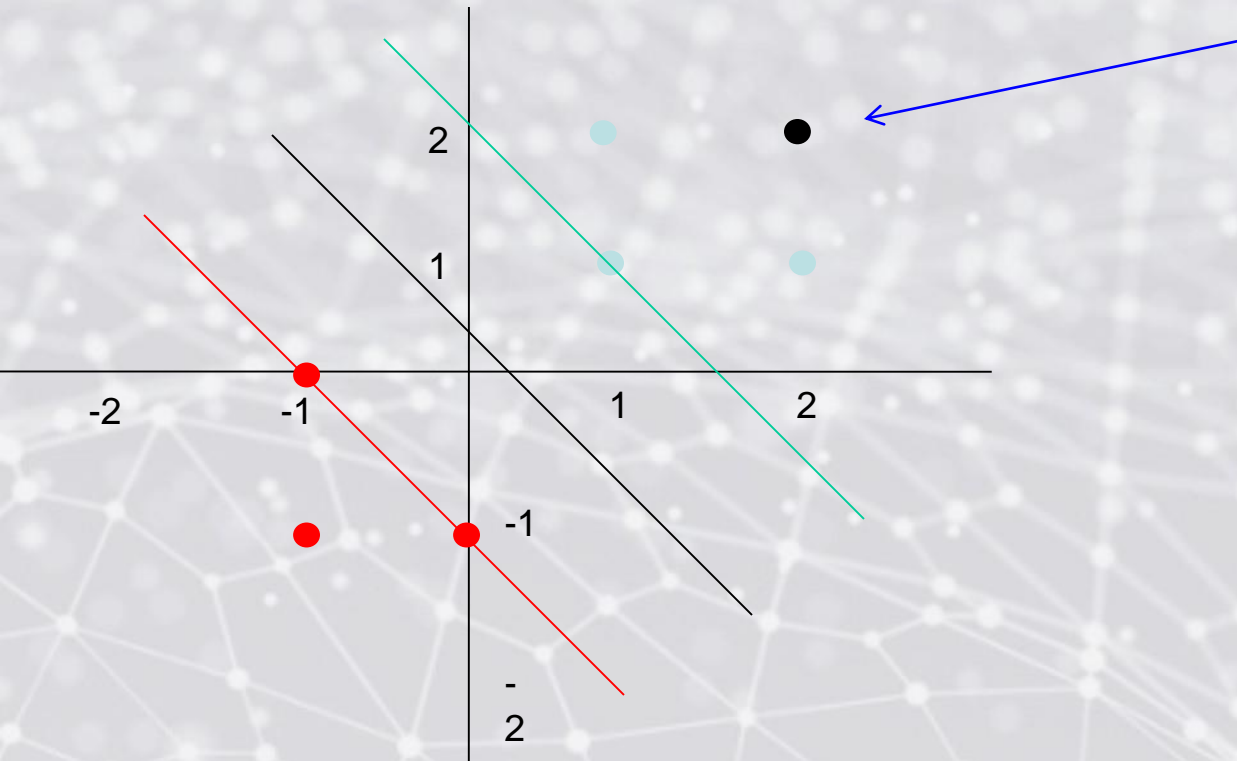
Classifying a new point:



$$h((2,2)) = \text{sgn} \left(\left(\sum_{k=1}^m \alpha_k (\mathbf{x}_k \cdot \mathbf{x}) \right) + b \right), \quad \text{where } \text{sgn}(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases}$$

Example

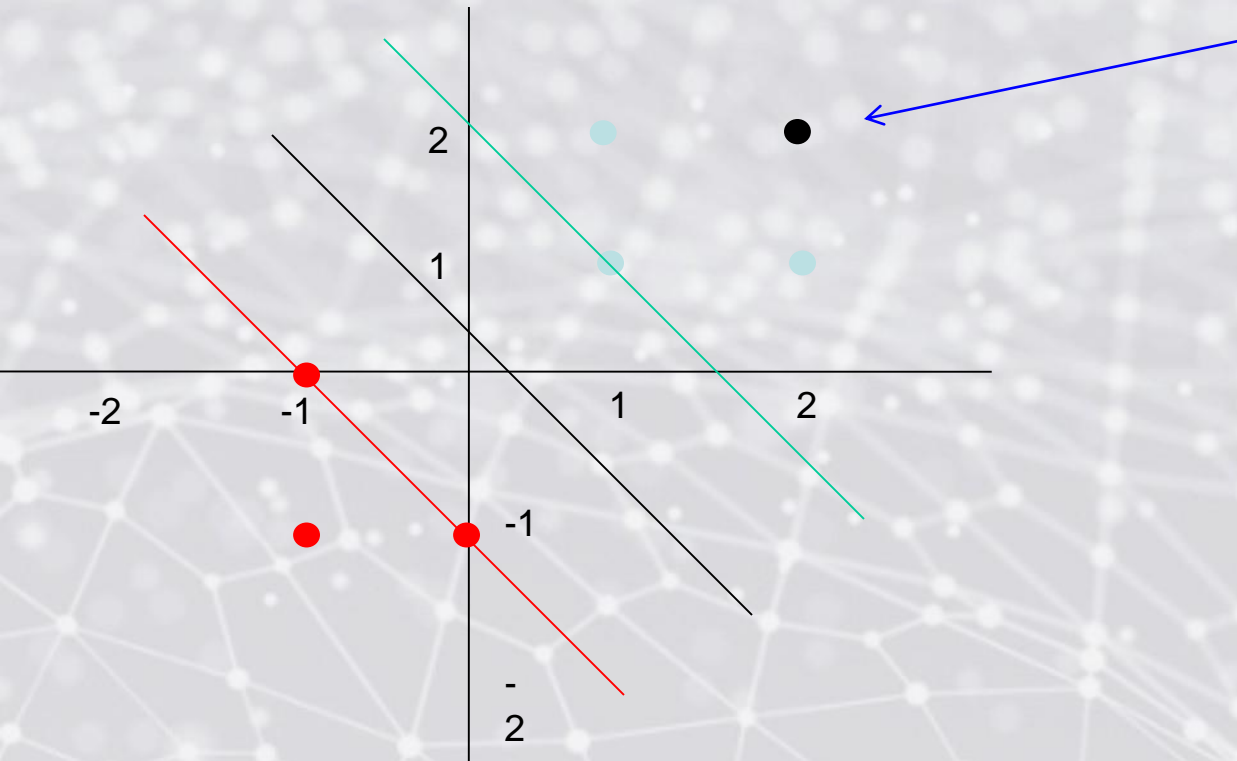
Classifying a new point:



$$h((2,2)) = \text{sgn}\left(\left(\sum_{k=1}^m \alpha_k (\mathbf{x}_k \cdot \mathbf{x})\right) + b\right), \quad \text{where } \text{sgn}(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases}$$
$$= \text{sgn}\left(-.208[(-1,0) \cdot (2,2)] + .416[(1,1) \cdot (2,2)] - .208[(0,-1) \cdot (2,2)] - .376\right)$$

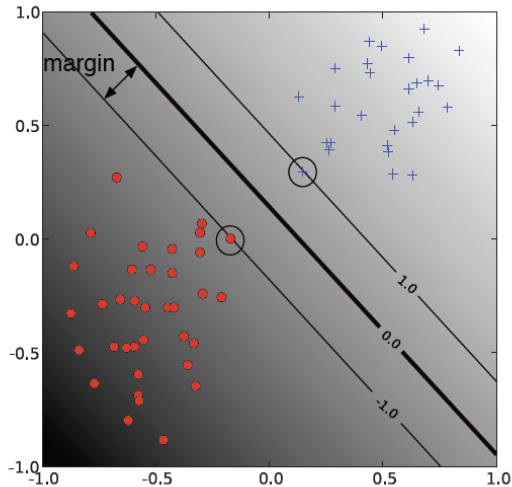
Example

Classifying a new point:



$$h((2,2)) = \text{sgn}\left(\left(\sum_{k=1}^m \alpha_k (\mathbf{x}_k \cdot \mathbf{x})\right) + b\right), \quad \text{where } \text{sgn}(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases}$$
$$= \text{sgn}\left(-.208[(-1,0) \cdot (2,2)] + .416[(1,1) \cdot (2,2)] - .208[(0,-1) \cdot (2,2)] - .376\right)$$
$$= \text{sgn}(.416 + 1.664 + .416 - .376) = +1$$

SVM summary



- Equation of line: $w_1x_1 + w_2x_2 + b = 0$

- Define margin using:

$\mathbf{x}_k \times \mathbf{w} + b \geq +1$ for positive instances ($t_k = +1$)

$\mathbf{x}_k \times \mathbf{w} + b \leq -1$ for negative instances ($t_k = -1$)

- Margin distance: $\frac{1}{\|\mathbf{w}\|}$

- To maximize the margin, we minimize $\|\mathbf{w}\|$ subject to the constraint that positive examples fall on one side of the margin, and negative examples on the other side:

$$t_k (\mathbf{w} \times \mathbf{x}_k + b) \geq 1, \quad k = 1, \dots, m$$

where $t_k \in \{-1, +1\}$

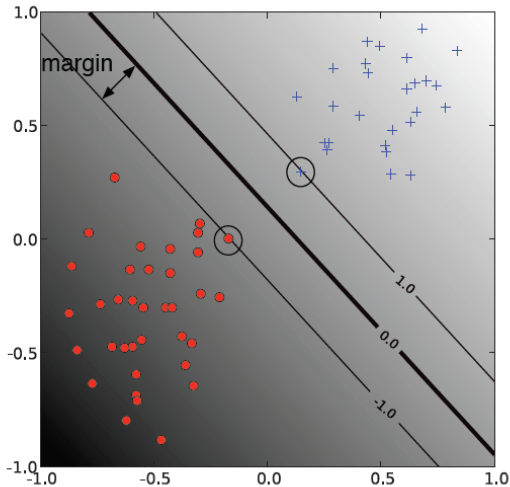
- We can relax this constraint using “slack variables”

SVM summary

- To do the optimization, we use the dual formulation:

$$\mathbf{w} = \sum_{k \in \hat{K}} a_k \mathbf{x}_k$$

The results of the optimization “black box” are $\{a_k\}$ and b .



SVM review

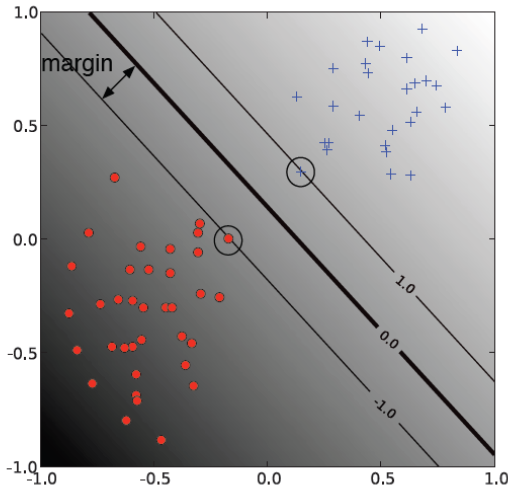
- Once the optimization is done, we can classify a new example \mathbf{x} as follows:

$$h(\mathbf{x}) = \text{class}(\mathbf{x}) = \text{sgn}(\mathbf{w} \times \mathbf{x} + b)$$

$$= \text{sgn}\left(\sum_{k=1}^m a_k \mathbf{x}_k \times \mathbf{x} + b\right)$$

$$= \text{sgn}\left(\sum_{k=1}^m a_k (\mathbf{x}_k \times \mathbf{x}) + b\right)$$

That is, classification is done entirely through a linear combination of dot products with training examples. This is a “kernel” method.



Example:

Watson used tree kernels and SVMs to classify question types for Jeopardy! questions

From Moschitti et al., 2011

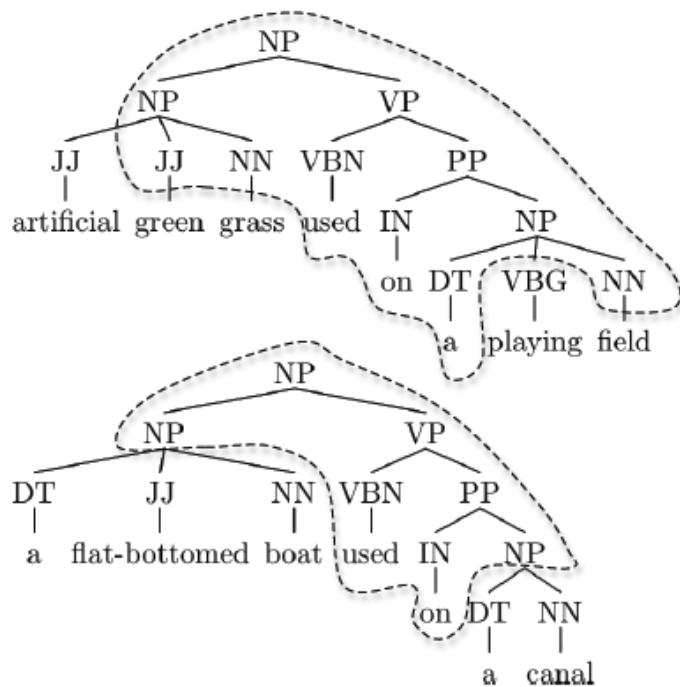
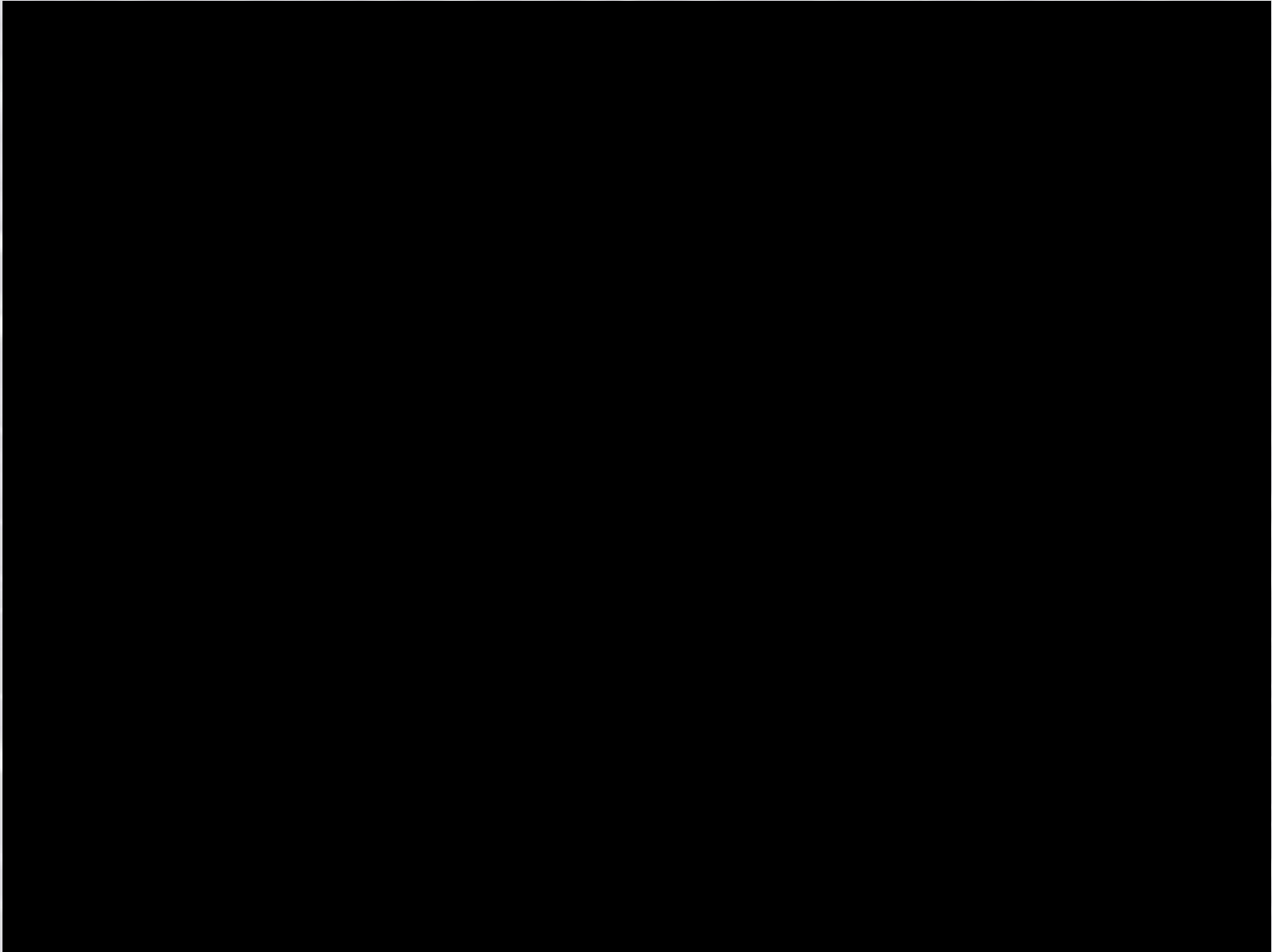


Figure 5: Similarity according to PTK and STK

Kernel Space	Prec.	Rec.	F1
WSK+CSK	70.00	57.19	62.95
PTK-CT+CSK	69.43	60.13	64.45
PTK-CT+WSK+CSK	68.59	62.09	65.18
CSK+RBC	47.80	74.51	58.23
PTK-CT+CSK+RBC	59.33	74.84	65.79
BOW+CSK+RBC	60.65	73.53	66.47
PTK-CT+WSK+CSK+RBC	67.66	66.99	67.32
PTK-CT+PASS+CSK+RBC	62.46	71.24	66.56
WSK+CSK+RBC	69.26	66.99	68.11
ALL	61.42	67.65	64.38

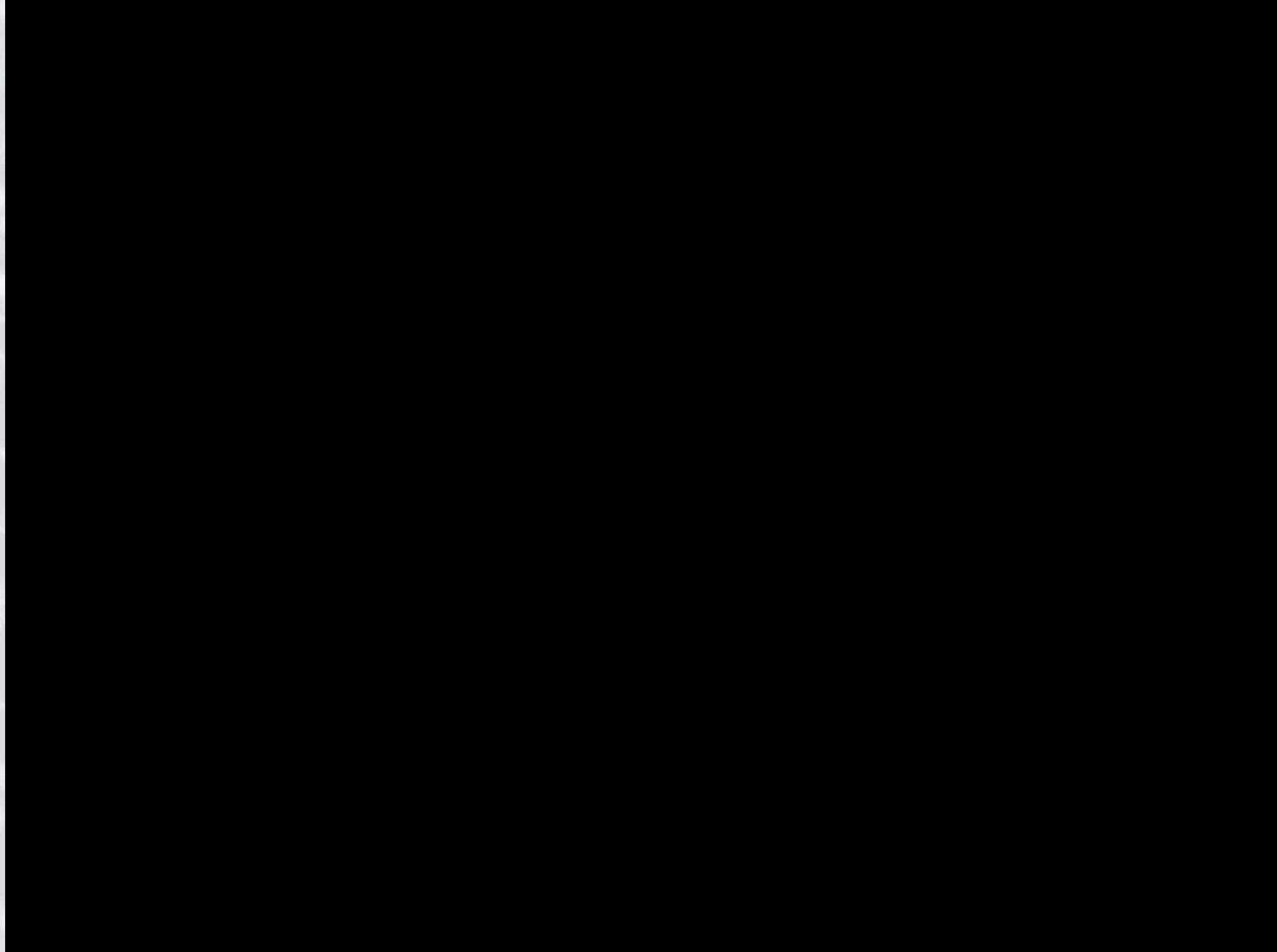
Table 2: Performance of Kernel Combinations using leave-one-out cross-validation.

IBM Watson



<https://www.youtube.com/watch?v=DywO4zksfXw>

IBM Watson: Jeopardy



<https://www.youtube.com/watch?v=P18EdAKuC1U>

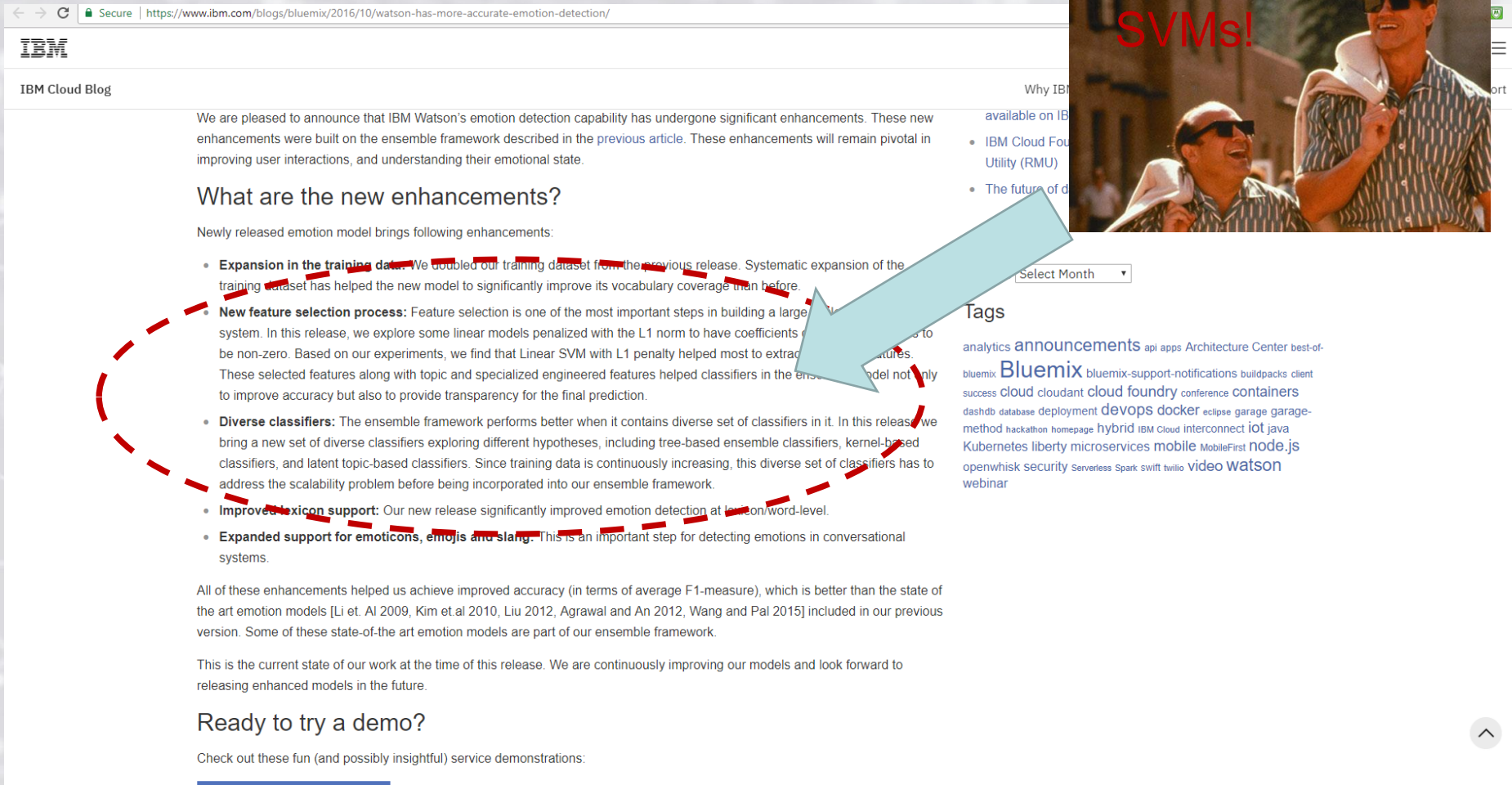
IBM Watson (2016)



- “**Emotion detection**” has been a central piece of recent research in attempts to make AI systems more compassionate.
- IBM Watson recently released *textual emotion detection* (with new enhancements) as a new functionality with Watson.
- Some of these enhancements include: (1) a new feature selection process using “**linear SVMs with L1 penalty**” And (2) new “**kernel-based classifiers.**”

IBM Watson (2016)

<https://www.ibm.com/blogs/bluemix/2016/10/watson-has-more-accurate-emotion-detection/>



← → ↻ Secure | <https://www.ibm.com/blogs/bluemix/2016/10/watson-has-more-accurate-emotion-detection/>

IBM

IBM Cloud Blog

We are pleased to announce that IBM Watson's emotion detection capability has undergone significant enhancements. These new enhancements were built on the ensemble framework described in the [previous article](#). These enhancements will remain pivotal in improving user interactions, and understanding their emotional state.

What are the new enhancements?

Newly released emotion model brings following enhancements:

- **Expansion in the training data:** We doubled our training dataset from the previous release. Systematic expansion of the training dataset has helped the new model to significantly improve its vocabulary coverage than before.
- **New feature selection process:** Feature selection is one of the most important steps in building a large machine learning system. In this release, we explore some linear models penalized with the L1 norm to have coefficients that are non-zero. Based on our experiments, we find that Linear SVM with L1 penalty helped most to extract features. These selected features along with topic and specialized engineered features helped classifiers in the ensemble model not only to improve accuracy but also to provide transparency for the final prediction.
- **Diverse classifiers:** The ensemble framework performs better when it contains diverse set of classifiers in it. In this release, we bring a new set of diverse classifiers exploring different hypotheses, including tree-based ensemble classifiers, kernel-based classifiers, and latent topic-based classifiers. Since training data is continuously increasing, this diverse set of classifiers has to address the scalability problem before being incorporated into our ensemble framework.
- **Improved lexicon support:** Our new release significantly improved emotion detection at lexicon/word-level.
- **Expanded support for emoticons, emojis and slang:** This is an important step for detecting emotions in conversational systems.

All of these enhancements helped us achieve improved accuracy (in terms of average F1-measure), which is better than the state of the art emotion models [Li et. al 2009, Kim et.al 2010, Liu 2012, Agrawal and An 2012, Wang and Pal 2015] included in our previous version. Some of these state-of-the art emotion models are part of our ensemble framework.

This is the current state of our work at the time of this release. We are continuously improving our models and look forward to releasing enhanced models in the future.

Ready to try a demo?

Check out these fun (and possibly insightful) service demonstrations:


Why IBM Watson is available on IBM Cloud Foundry

- IBM Cloud Foundry Utility (RMU)
- The future of data

Select Month ▾

Tags

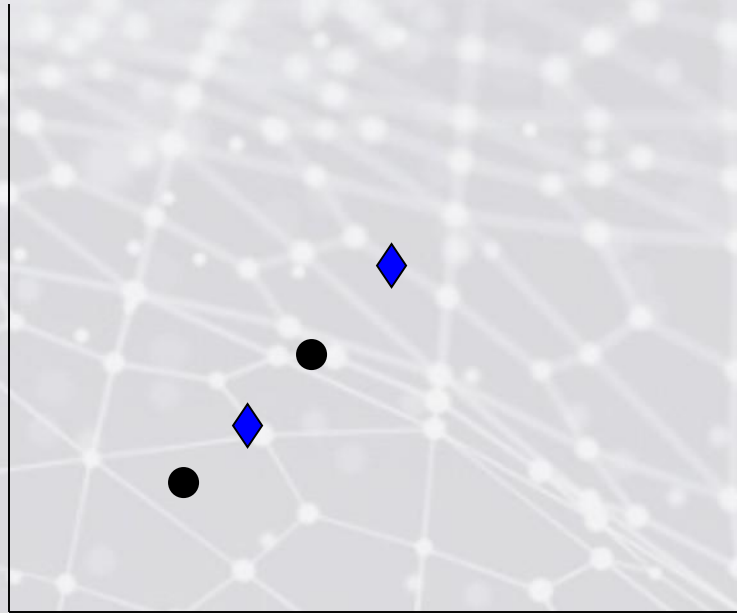
analytics announcements api apps Architecture Center best-of-breed bluemix Bluemix bluemix-support-notifications buildpacks client success cloud cloudant cloud foundry conference containers dashdb database deployment devops docker eclipse garage garage-method hackathon homepage hybrid IBM Cloud interconnect iot java Kubernetes liberty microservices mobile MobileFirst node.js openwhisk security Serverless Spark swift twilio video watson webinar



Demo: <https://natural-language-understanding-demo.ng.bluemix.net/>

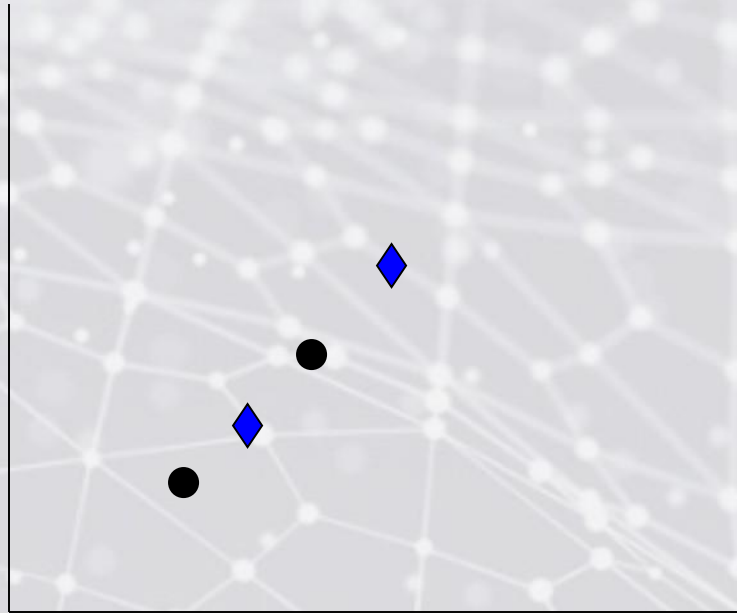
Non-linearly separable training examples

- What if the training examples are not linearly separable?

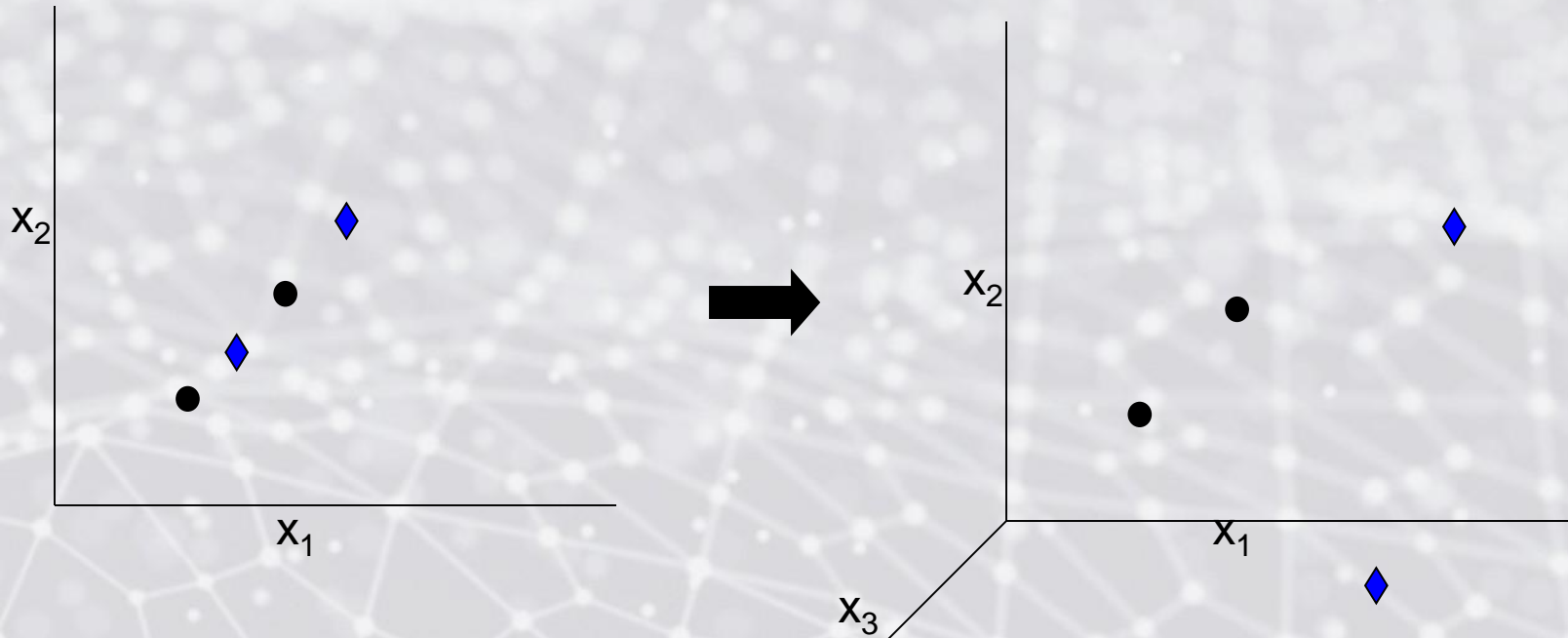


Non-linearly separable training examples

- What if the training examples are not linearly separable?



- Use old trick: Find a function that maps points to a higher dimensional space (“feature space”) in which they are linearly separable, and do the classification in that higher-dimensional space.



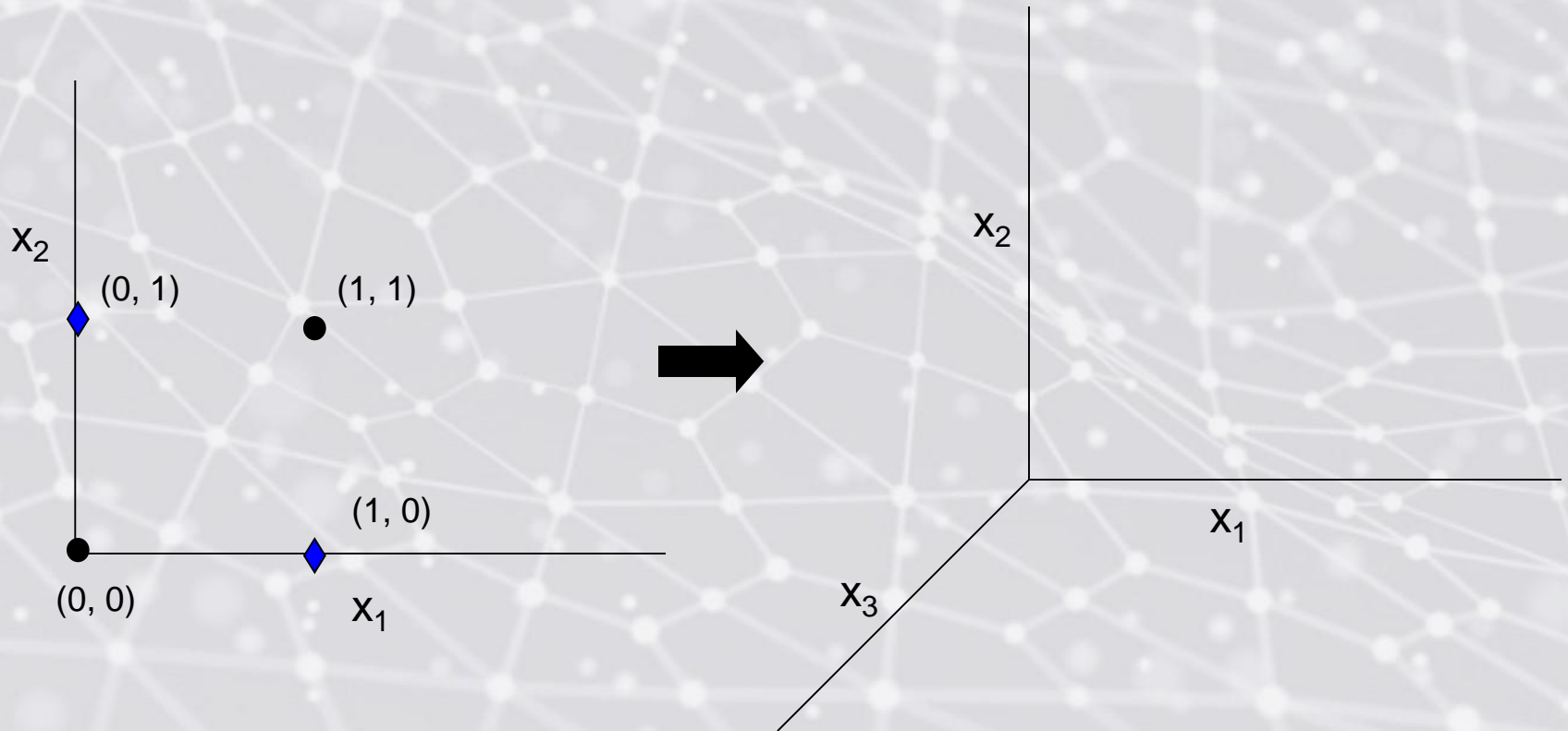
Need to find a function Φ that will perform such a mapping:

$$\Phi: \mathcal{R}^n \rightarrow F$$

Then can find hyperplane in higher dimensional feature space F , and do classification using that hyperplane in higher dimensional space.

Challenge

Find a 3-dimensional feature space in which XOR is linearly separable.



- **Problem:**

- Recall that classification of instance \mathbf{x} is expressed in terms of dot products of \mathbf{x} and support vectors.

$$\text{Class}(\mathbf{x}) = \text{sgn} \left(\sum_{k \in \{\text{training examples}\}} a_k (\mathbf{x} \times \mathbf{x}_k) + b \right)$$

- The quadratic programming problem of finding the support vectors and coefficients also depends only on dot products between training examples.

- So if each \mathbf{x}_k is replaced by $\Phi(\mathbf{x}_k)$ in these procedures, we will have to calculate $\Phi(\mathbf{x}_k)$ for each k as well as calculate a lot of dot products, $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_k)$
- But in general, if the feature space F is high dimensional, $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ will be expensive to compute.

- **Second trick (the “kernel trick”):**

- Suppose that there were some magic function,

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

such that K is cheap to compute even though $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ is expensive to compute.

- Then we wouldn’t need to compute the dot product directly; we’d just need to compute K during both the training and testing phases.
- The good news is: such K functions exist! They are called “kernel functions”, and come from the theory of integral operators.

Example: Polynomial kernel:

Suppose $\mathbf{x} = (x_1, x_2)$ and $\mathbf{z} = (z_1, z_2)$.

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \times \mathbf{z})^2$$

$$\text{Let } \Phi(\mathbf{x}) = (x_1^2, \sqrt{2} \cdot x_1 x_2, x_2^2).$$

Then :

$$\begin{aligned} \Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) &= \left(\begin{pmatrix} x_1^2 \\ \sqrt{2} \cdot x_1 x_2 \\ x_2^2 \end{pmatrix} \cdot \begin{pmatrix} z_1^2 \\ \sqrt{2} \cdot z_1 z_2 \\ z_2^2 \end{pmatrix} \right) \\ &= \left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \right)^2 = (\mathbf{x} \cdot \mathbf{z})^2 = k(\mathbf{x}, \mathbf{z}) \end{aligned}$$

Example: Polynomial kernel:

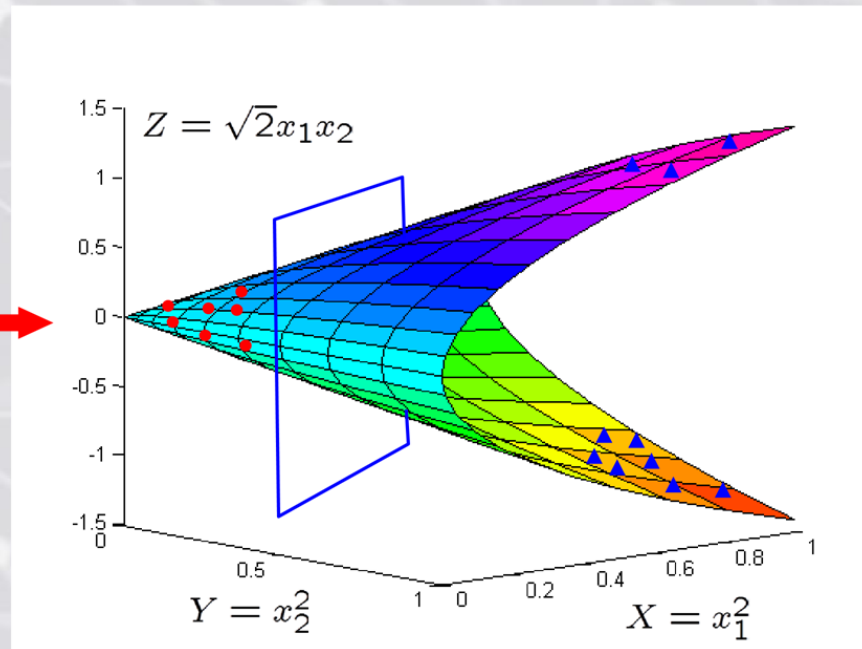
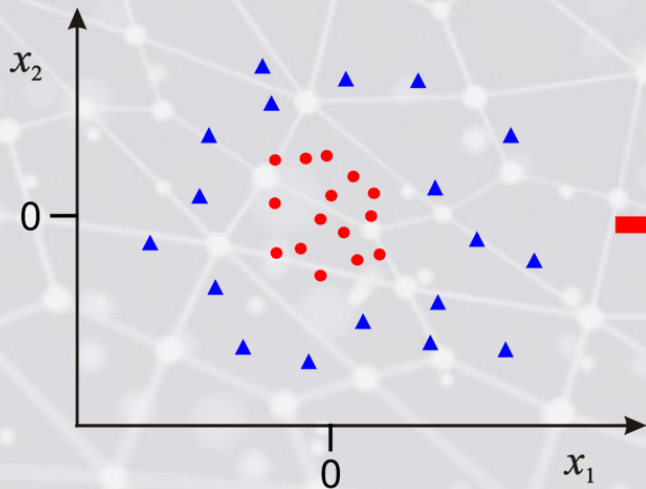
$$\text{Let } \Phi(\mathbf{x}) = (x_1^2, \sqrt{2} \cdot x_1 x_2, x_2^2).$$

Suppose $\mathbf{x} = (x_1, x_2)$ and $\mathbf{z} = (z_1, z_2)$.

Then:

$$\begin{aligned} \Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) &= \left(\begin{pmatrix} x_1^2 \\ \sqrt{2} \cdot x_1 x_2 \\ x_2^2 \end{pmatrix} \cdot \begin{pmatrix} z_1^2 \\ \sqrt{2} \cdot z_1 z_2 \\ z_2^2 \end{pmatrix} \right) \\ &= \left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \right)^2 = (\mathbf{x} \cdot \mathbf{z})^2 = k(\mathbf{x}, \mathbf{z}) \end{aligned}$$

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$



Most commonly used kernels

- Linear

$$K(\mathbf{x}, \mathbf{x}_i) = \mathbf{x} \cdot \mathbf{x}_i$$

- Polynomial

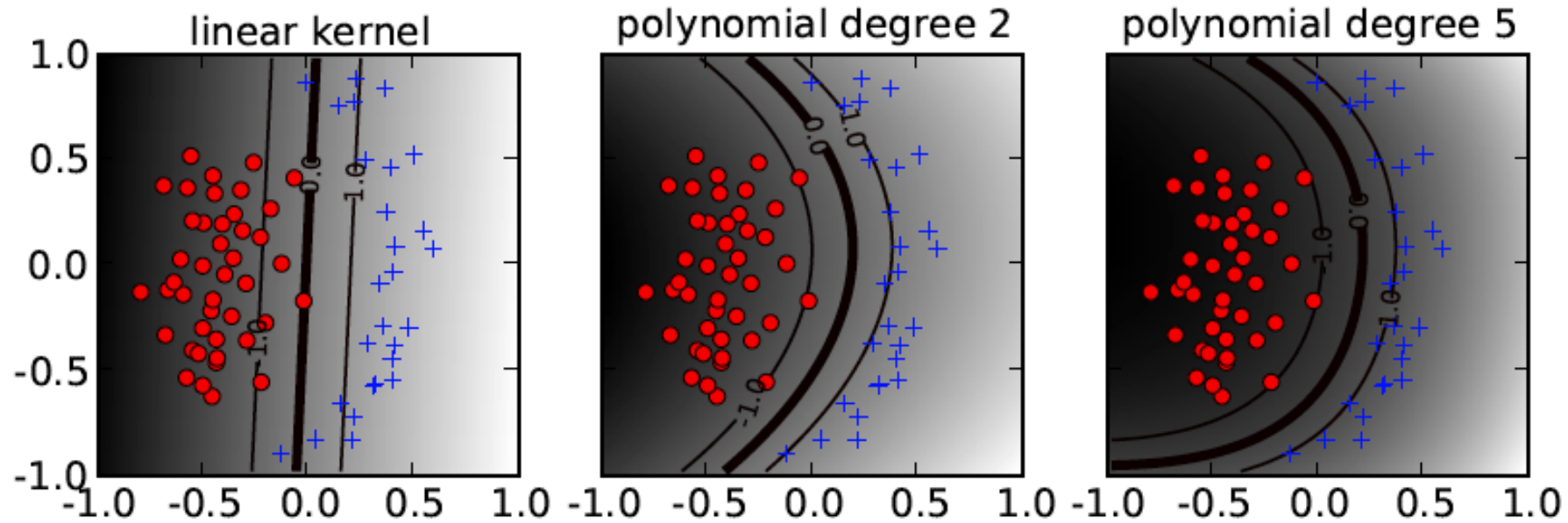
$$K(\mathbf{x}, \mathbf{x}_i) = [(\mathbf{x} \cdot \mathbf{x}_i) + 1]^d$$

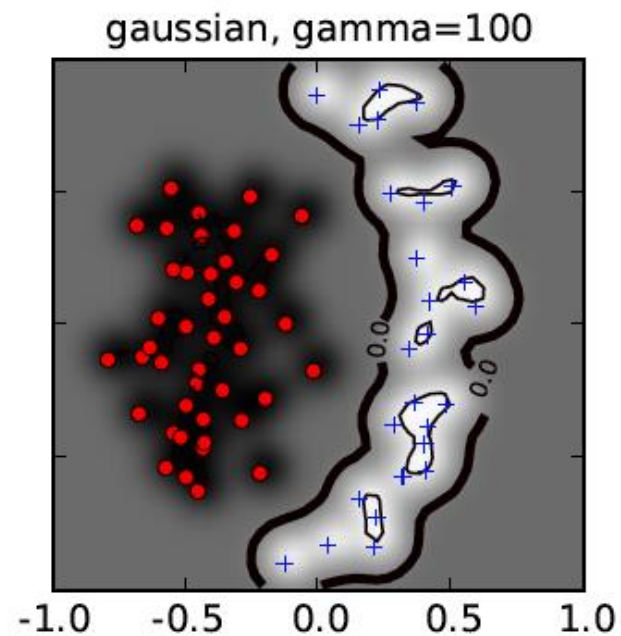
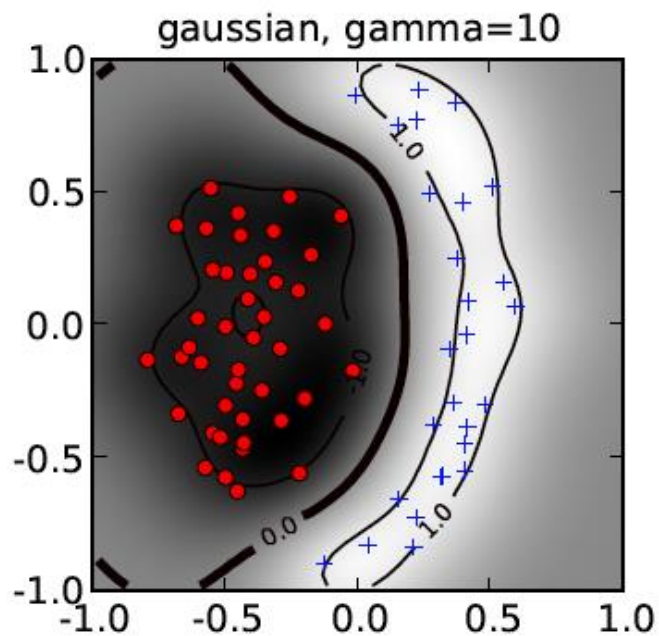
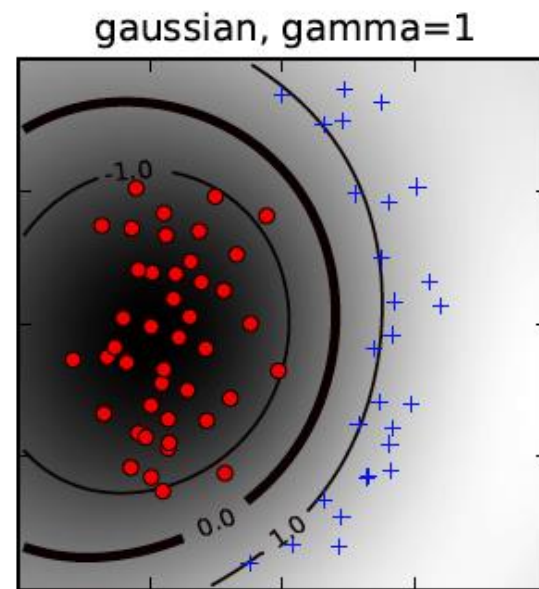
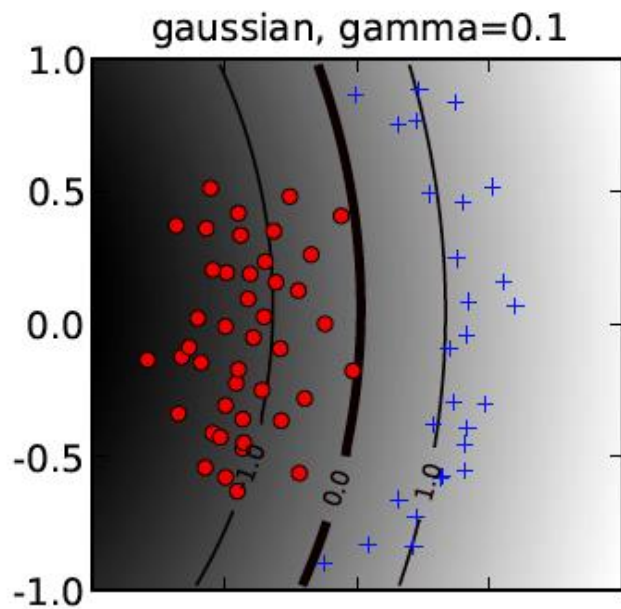
- Gaussian (or “radial basis function”)

$$K(\mathbf{x}, \mathbf{x}_i) = e^{-\gamma|\mathbf{x}-\mathbf{x}_i|^2}$$

- Sigmoid

$$K(\mathbf{x}, \mathbf{x}_i) = \tanh(a\mathbf{x} \cdot \mathbf{x}_i + b)$$





More on Kernels

- So far we've seen kernels that map instances in \mathbb{R}^n to instances in \mathbb{R}^z where $z > n$.
- One way to create a kernel: Figure out appropriate feature space $\Phi(\mathbf{x})$, and find kernel function k which defines inner product on that space.
- More practically, we usually don't know appropriate feature space $\Phi(\mathbf{x})$.
- What people do in practice is either:
 1. Use one of the “classic” kernels (e.g., polynomial),
 - or 2. Define their own function that is appropriate for their task, and show that it qualifies as a kernel.

How to define your own kernel

- Given training data $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$
- Algorithm for SVM learning uses *kernel matrix* (also called ***Gram matrix***):

$$\mathbf{K}_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j), \text{ for } i, j = 1, \dots, n$$

- We can choose some function K , and compute the kernel matrix \mathbf{K} using the training data.
- We just have to guarantee that our kernel defines an inner product on some feature space.
- Not as hard as it sounds.

What counts as a kernel?

- **Mercer's Theorem**: If the kernel matrix **K** is “positive definite”, it defines a kernel, that is, it defines an inner product in some feature space.
- We don't even have to know what that feature space is! It can have a huge number of dimensions.

- Recall:

▣ A $n \times n$ real matrix **M** is positive semi-definite

$$\text{if } \underline{z}^T M \underline{z} \geq 0 \quad \forall \underline{z}$$

Inner Product - Definition

Definition: An **inner product** on a vector space V is a function that to each pair of vectors \mathbf{u} and \mathbf{v} in V , associates a real number $\langle \mathbf{u}, \mathbf{v} \rangle$ and satisfies the following axioms for all $\mathbf{u}, \mathbf{v}, \mathbf{w}$ in V and all scalars c :

1. $\langle \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{v}, \mathbf{u} \rangle$
2. $\langle \mathbf{u} + \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle + \langle \mathbf{v}, \mathbf{w} \rangle$
3. $\langle c\mathbf{u}, \mathbf{v} \rangle = c\langle \mathbf{u}, \mathbf{v} \rangle$
4. $\langle \mathbf{u}, \mathbf{u} \rangle \geq 0$ & $\langle \mathbf{u}, \mathbf{u} \rangle = 0$ iff $\mathbf{u} = \mathbf{0}$

Summary of “kernel trick” (or kernel method)

- Kernel methods can be thought of as *instance-based learners*: rather than learning some fixed set of parameters corresponding to the features of their inputs, they instead “remember” the i th training example (\mathbf{x}_i, y_i) and learn it for a corresponding weight α_i .
- For generalization in the classification step, a new datum \mathbf{x} is compared (via the “similarity” function \mathbf{k}) to each training input \mathbf{x}_i :

$$\text{class}(\mathbf{x}) = \text{sgn} \left(\left(\sum_{k \in \text{support vectors}} a_k k(\mathbf{x}, \mathbf{x}_k) \right) + b \right)$$

- The *kernel trick* avoids the explicit mapping needed to get linear algorithms to render a **non-linear decision boundary**.
- The key idea is that $\mathbf{k}(\mathbf{x}, \mathbf{x}_i)$ is expressed as an inner product in another space V .
- By appropriately defining the kernel (e.g. using **Mercer’s theorem**) we are guaranteed that V is an **inner product space** (aside: an inner product space naturally induces a norm on a vector space, in which case we get “intuitive” geometric properties per Euclidean spaces).
- **The upshot for SVMs**: we get the rich representative qualities of non-linear decision boundaries without having to compute Φ explicitly.

Structural Kernels

- In domains such as natural language processing and bioinformatics, the similarities we want to capture are often structural (e.g., parse trees, formal grammars).
- An important area of kernel methods is defining *structural kernels* that capture this similarity (e.g., sequence alignment kernels, tree kernels, graph kernels, etc.)

- Design criteria - we want kernels to be:
 - **valid** – Satisfy Mercer condition of positive semi-definiteness.
 - **good** – embody the “true similarity” between objects
 - **appropriate** – generalize well
 - **efficient** – the computation of $K(\mathbf{x}, \mathbf{x}')$ is feasible

Summary of SVM algorithm

Given training set

$$S = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_m, t_m) \mid (\mathbf{x}_k, t_k) \in \mathbb{R}^n \times \{+1, -1\}\}$$

1. Choose a kernel function $k(\mathbf{x}, \mathbf{z})$.
2. Apply optimization procedure (using the kernel function K) to find support vectors \mathbf{x}_k , coefficients α_k , and bias b .
3. Given a new instance, \mathbf{x} , find the classification of \mathbf{x} by computing

$$\text{class}(\mathbf{x}) = \text{sgn} \left(\left(\sum_{k \in \text{support vectors}} a_k k(\mathbf{x}, \mathbf{x}_k) \right) + b \right)$$

Summary of SVM algorithm

Computational complexity for SVM: This can be difficult to compute in general.

Basic Idea: (1) We need to perform $\mathbf{O}(m^2n)$ primitives for the dot product/kernel (m is the number of data points and n is the dimensionality); (2) unfortunately, to solve the QP problem this requires inversion of the kernel matrix which is in general $\mathbf{O}(m^3)$.

$$\text{class}(\mathbf{x}) = \text{sgn} \left(\left(\sum_{k \in \text{support vectors}} a_k k(\mathbf{x}, \mathbf{x}_k) \right) + b \right)$$

How to define your own kernel

- Given training data $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m)$
- Algorithm for SVM learning uses *kernel matrix* (also called *Gram matrix*):

$$\mathbf{K}_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j), \text{ for } i, j = 1, \dots, m$$

$\mathbf{K} =$

$K(\mathbf{x}^1, \mathbf{x}^1)$	$K(\mathbf{x}^1, \mathbf{x}^2)$	$K(\mathbf{x}^1, \mathbf{x}^3)$...	$K(\mathbf{x}^1, \mathbf{x}^m)$
$K(\mathbf{x}^2, \mathbf{x}^1)$	$K(\mathbf{x}^2, \mathbf{x}^2)$	$K(\mathbf{x}^2, \mathbf{x}^3)$		$K(\mathbf{x}^2, \mathbf{x}^m)$
...
$K(\mathbf{x}^m, \mathbf{x}^1)$	$K(\mathbf{x}^m, \mathbf{x}^2)$	$K(\mathbf{x}^m, \mathbf{x}^3)$...	$K(\mathbf{x}^m, \mathbf{x}^m)$

How to define your own kernel

- We can choose some function K , and compute the kernel matrix K using the training data.
- We just have to guarantee that our kernel defines an inner product on some feature space.
- Mercer's Theorem: : If K is “positive semi-definite”, it defines a kernel, that is, it defines an inner product in some feature space.
- We don't even have to know what that feature space is!
- K is positive semi-definite if all the eigenvalues of K are positive.

Example of Simple “Custom” Kernel

Similarity between DNA Sequences:



E.g., (four bases: Adenine, Guanine, Cytosine and Thymine).

$s_1 = \text{GAATGTCCTTTCTCTAAGTCCTAAG}$

$s_2 = \text{GGAGACTTACAGGAAAGAGATTTCG}$

Define “**Hamming Distance Kernel**”:

$\text{hamming}(s_1, s_2) = \text{number of sites where strings match}$

Kernel matrix for *hamming* kernel

Suppose training set is

$s_1 = \text{GAATGTCCTTTCTCTAAGTCCTAAG}$

$s_2 = \text{GGAGACTTACAGGAAAGAGATTTCG}$

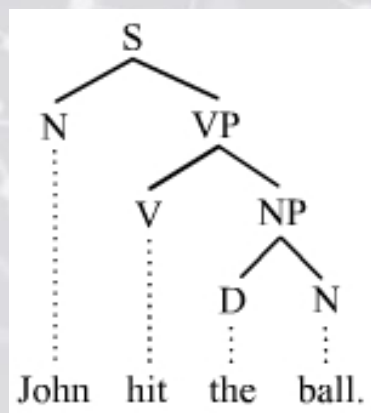
$s_3 = \text{GGAAACTTTCGGGAGAGAGTTTCG}$

What is the Kernel matrix **K**?

K	s_1	s_2	s_3
s_1			
s_2			
s_3			

Tree Kernels

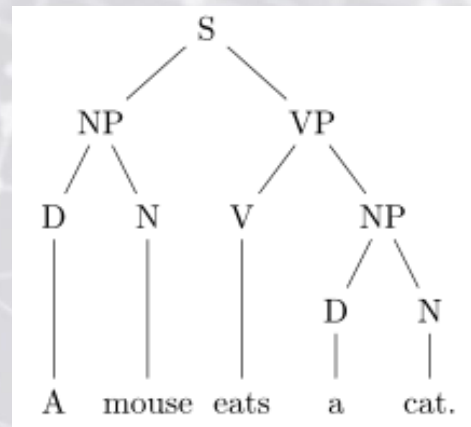
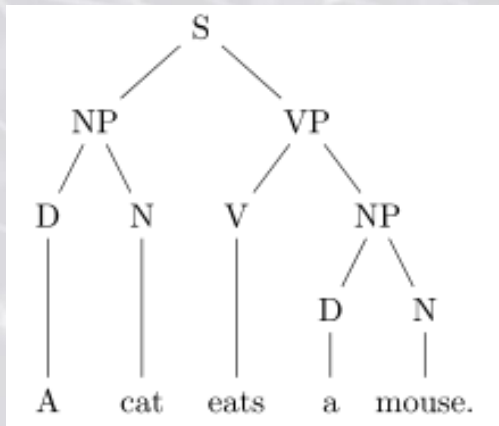
- In NLP *Tree Kernels* are commonly used for sentence classification.
- Beginning with a *parse tree* for a sentence we can compute the “similarity” of two sentences (i.e. two trees).
- Well-designed kernels enable us to compute the similarity in an efficient way – without explicitly computing (potentially) large dimensional feature vectors for these trees.



S: sentence
NP: noun-phrase
VP: verb-phrase
V: verb
N: noun
D: determiner (e.g. an article)

Tree Kernels

- A subtree is defined as a node and all of its children (note: terminals are not considered subtrees).
- A subtree kernel simply counts the number of common subtrees between two given trees.
- Consider the sentences: “*A cat eats a mouse*” vs. “*A mouse eats a cat.*”



[NP [D [a]] [N [cat]]],
[NP [D [a]] [N [mouse]]],
[N [mouse]],
[N [cat]],
[V [eats]],
[D [a]] (counted twice as it appears twice)



$K_{\text{subtree}}(s_1, s_2) = 7$ in this case because the sentences generate **seven common subtrees**.

- Other possible kernels include subset kernels (has higher granularity than subtree kernel). In this case, $K_{\text{subset}}(s_1, s_2) = 54$.

Hard- vs. soft- margin SVMs

Hard-margin SVMs

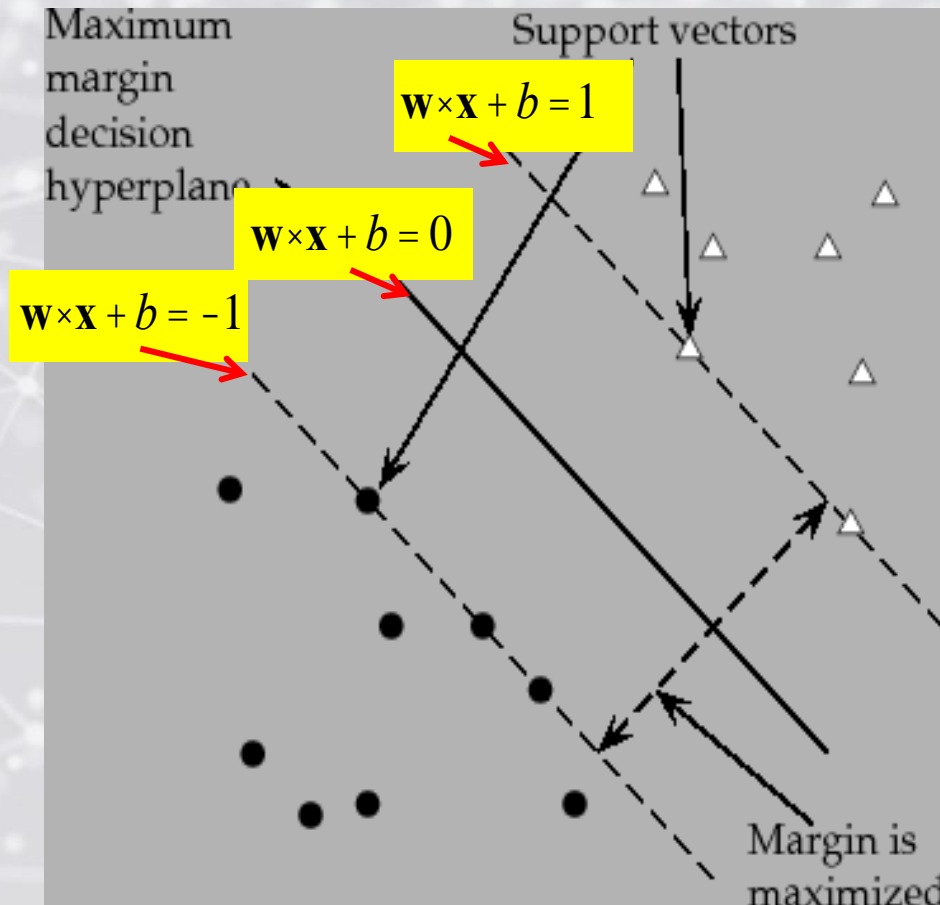
Find w and b by doing the following minimization:

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

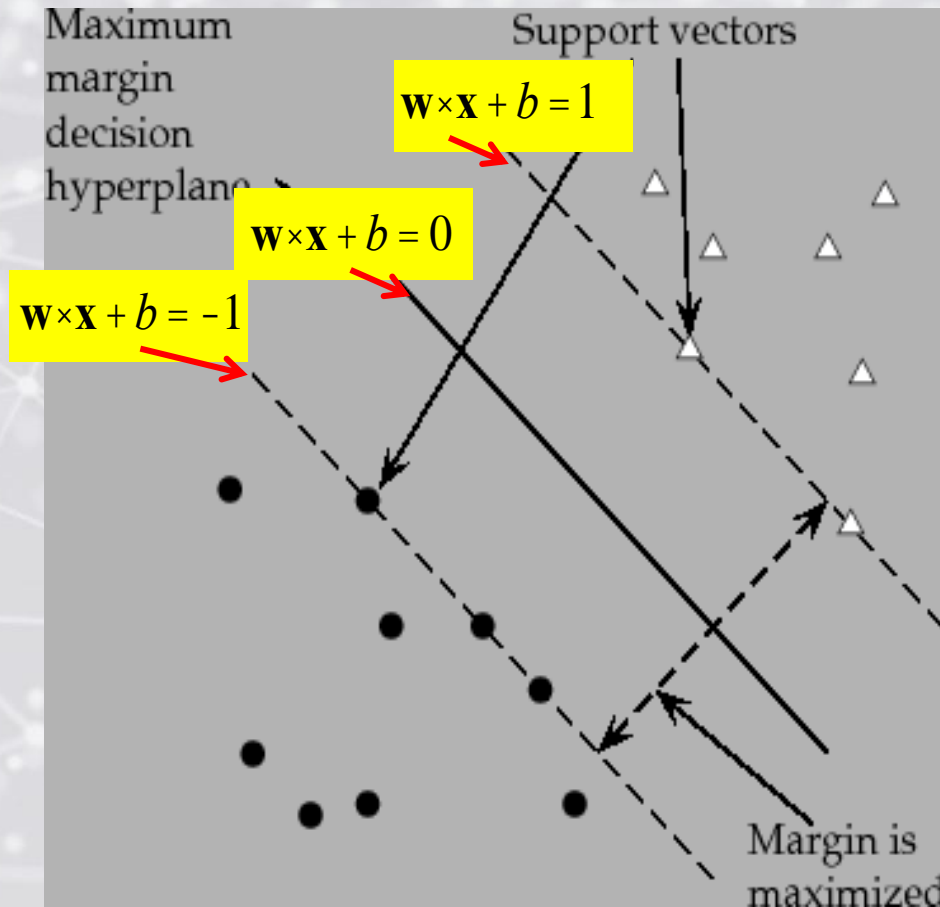
subject to:

$$t_k (w \cdot x_k + b) \geq 1, \quad k = 1, \dots, m$$

$$(t_k \in \{-1, +1\})$$

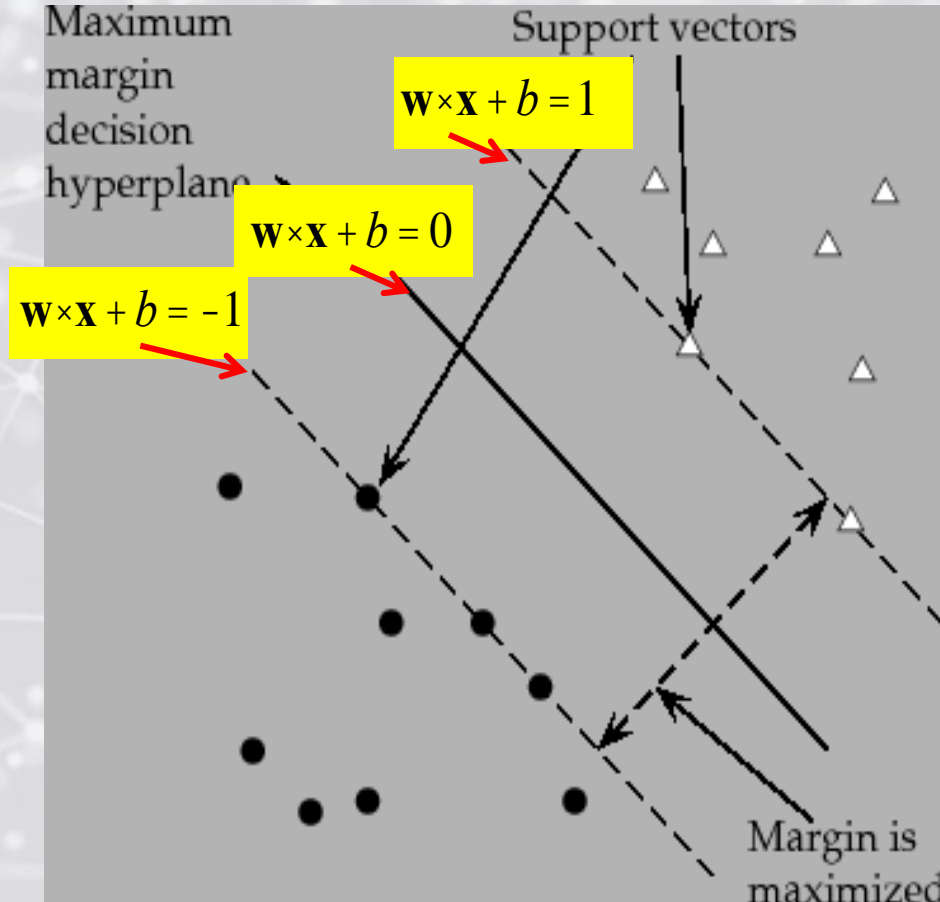


Extend to **soft-margin** SVMs



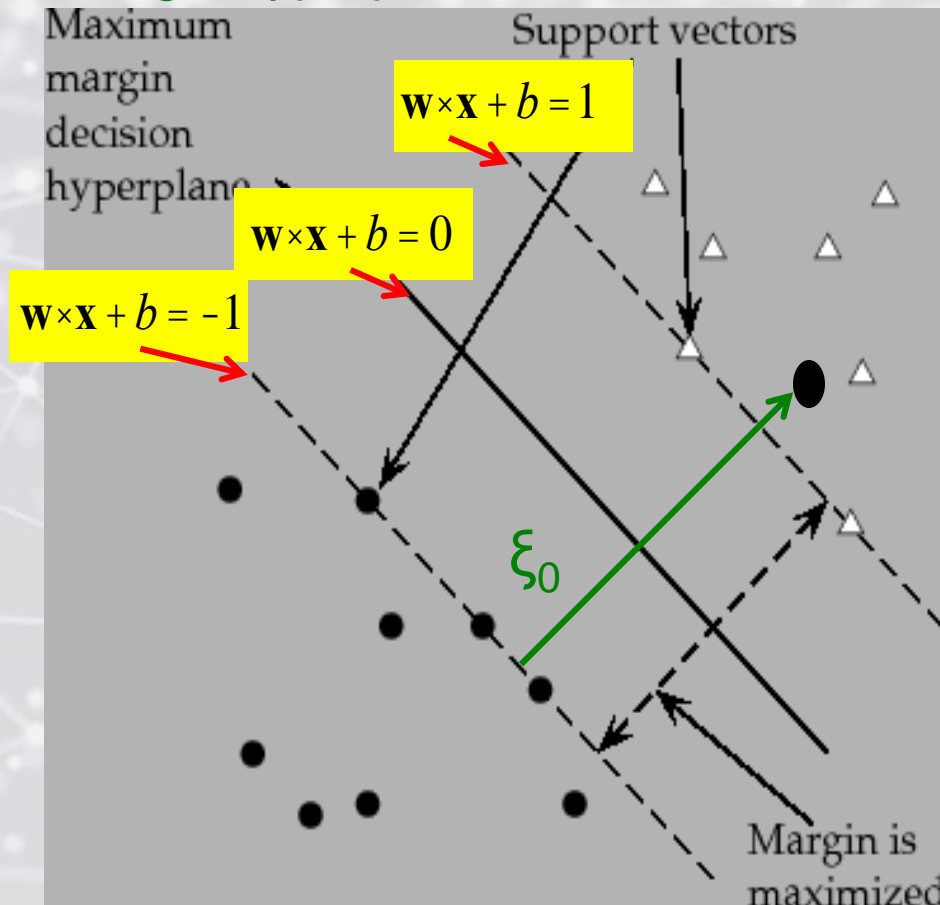
Extend to **soft-margin** SVMs

Allow some instances to be misclassified, or fall within margins, but penalize them by distance to margin hyperplane.



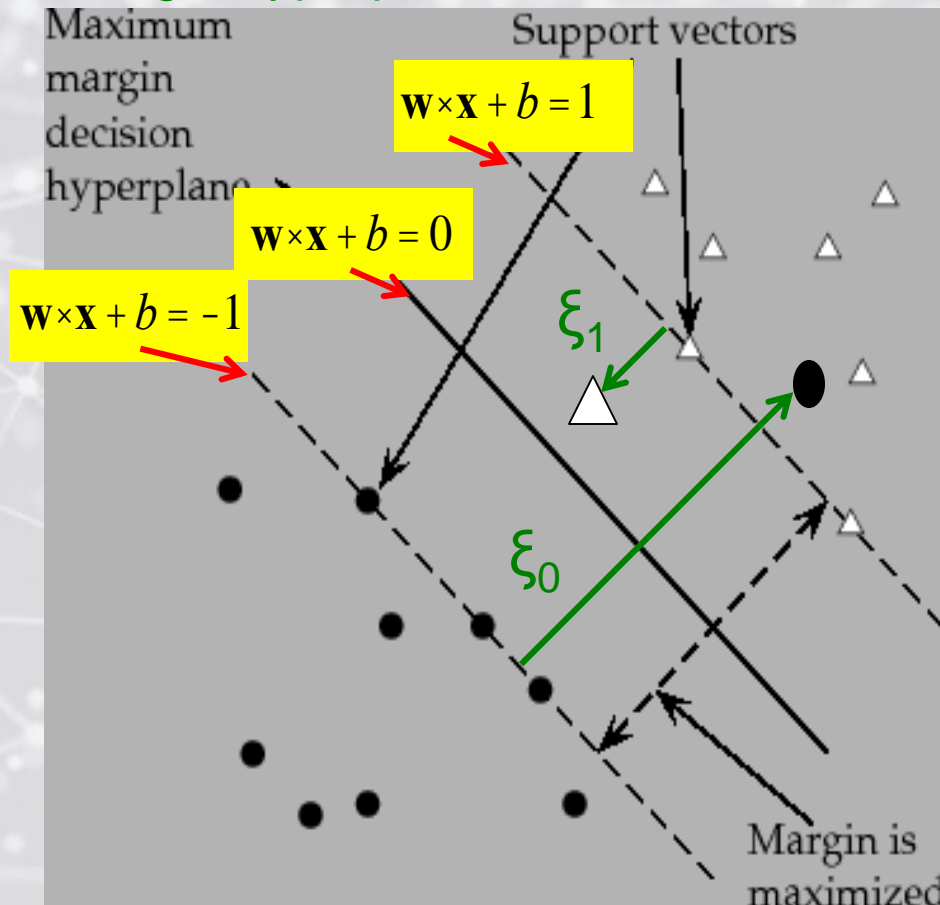
Extend to **soft-margin** SVMs

Allow some instances to be misclassified, or fall within margins, but penalize them by distance to margin hyperplane.



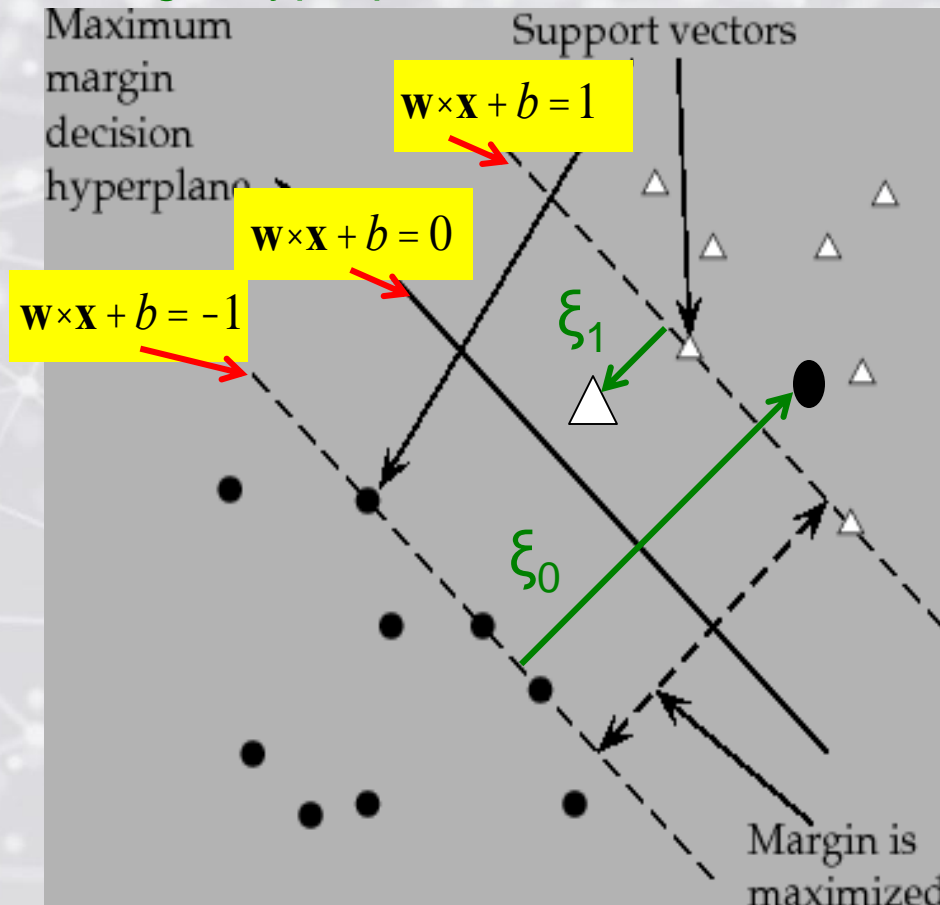
Extend to **soft-margin** SVMs

Allow some instances to be misclassified, or fall within margins, but penalize them by distance to margin hyperplane.



Extend to **soft-margin** SVMs

Allow some instances to be misclassified, or fall within margins, but penalize them by distance to margin hyperplane.

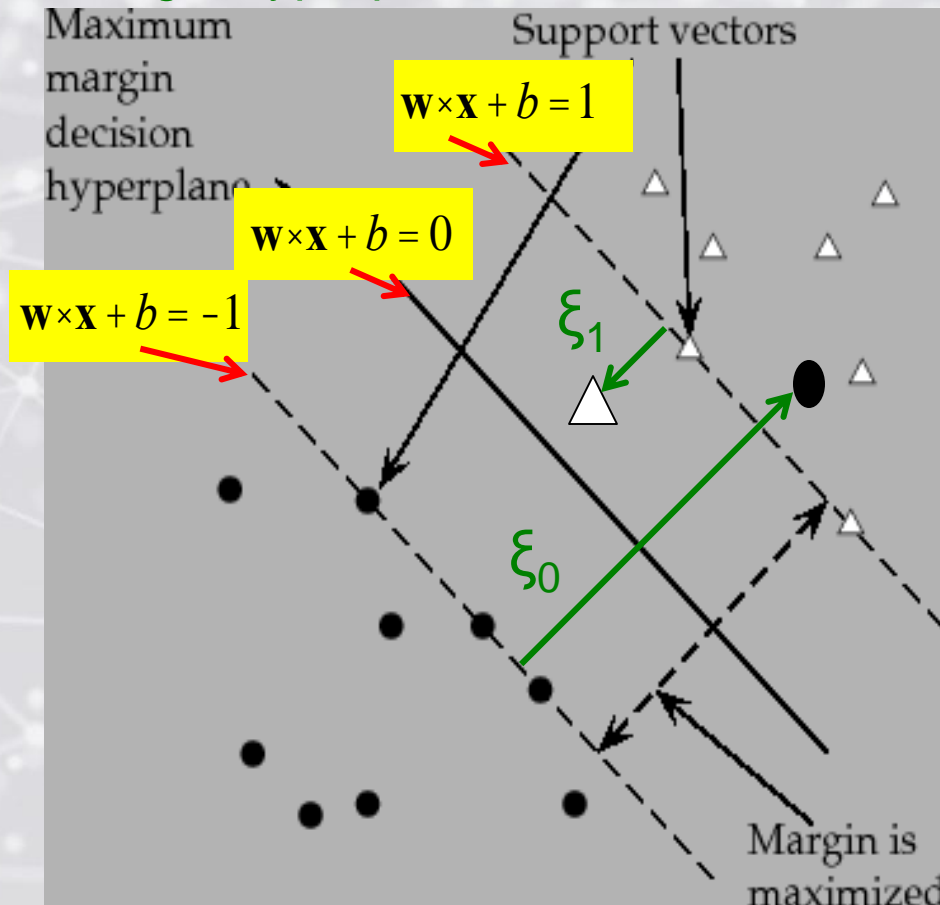


ξ_k are called slack variables.

$\xi_k > 0$ only if x_k is misclassified or inside margin

Extend to **soft-margin** SVMs

Allow some instances to be misclassified, or fall within margins, but penalize them by distance to margin hyperplane.



Revised optimization problem:

Find \mathbf{w} and b by doing the following minimization:

$$\min_{\mathbf{w}, b} \left(\frac{1}{2} \|\mathbf{w}\|^2 \right) + C \sum_k \chi_k$$

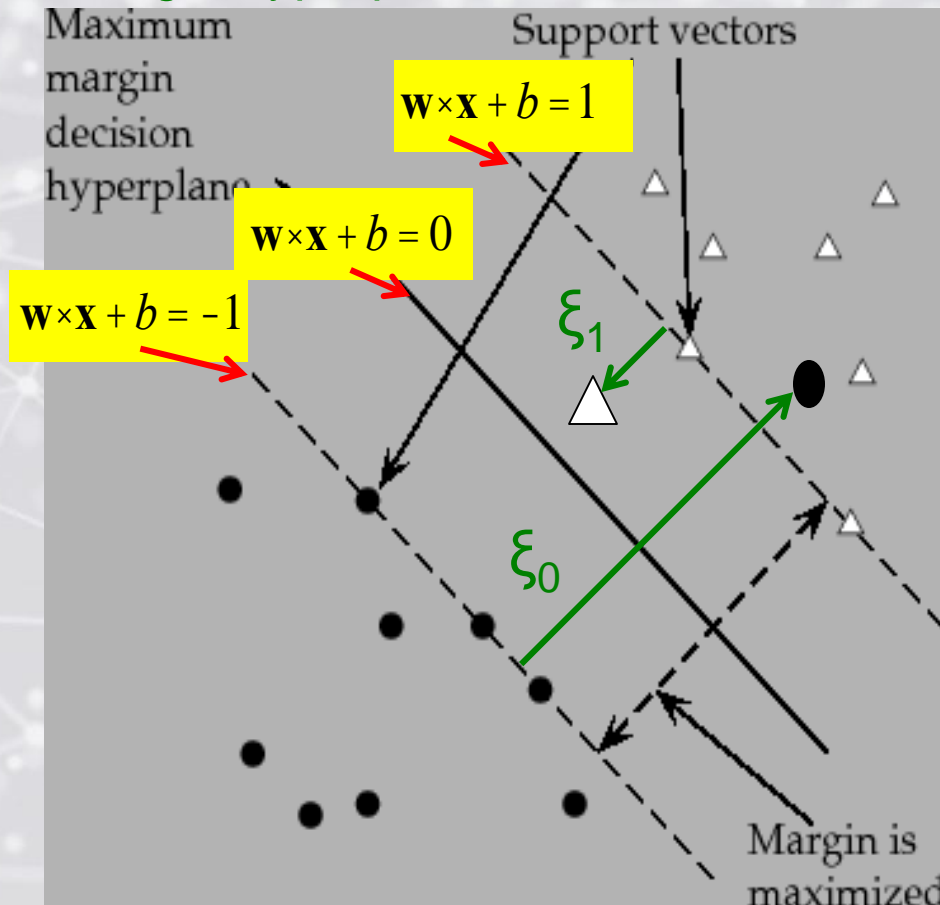
subject to:

$$t_k (\mathbf{w} \cdot \mathbf{x}_k + b) \geq 1 - \chi_k, \quad k = 1, \dots, m$$

$$(t_k \in \{-1, +1\})$$

Extend to **soft-margin** SVMs

Allow some instances to be misclassified, or fall within margins, but penalize them by distance to margin hyperplane.



<http://nlp.stanford.edu/IR-book/html/htmledition/img1260.png>

Revised optimization problem:

Find \mathbf{w} and b by doing the following minimization:

$$\min_{\mathbf{w}, b} \left(\frac{1}{2} \|\mathbf{w}\|^2 \right) + C \sum_k \chi_k$$

subject to:

$$t_k (\mathbf{w} \cdot \mathbf{x}_k + b) \geq 1 - \chi_k, \quad k = 1, \dots, m$$

$$(t_k \in \{-1, +1\})$$

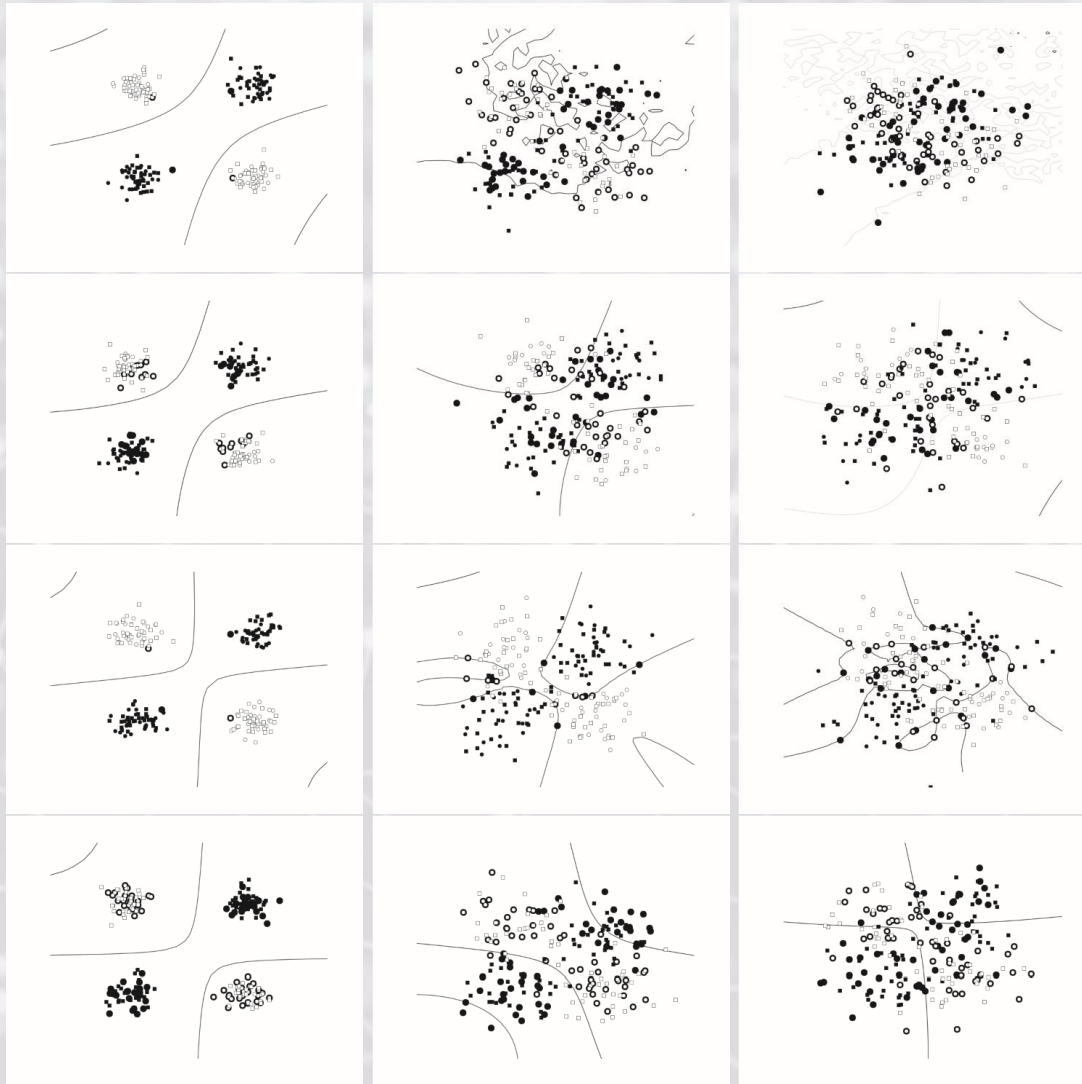
Optimization tries to keep ξ_k 's to zero while maximizing margin.

C is parameter that trades off margin width with misclassifications

Why use soft-margin SVMs?

- Always can be optimized (unlike hard-margin SVMs)
- More robust to outliers, noise
- **However:** Have to set C parameter

Soft Margin SVM Classification



$$\min_{\mathbf{w}, b} \left(\frac{1}{2} \|\mathbf{w}\|^2 \right) + C \sum_k \chi_k$$

subject to:

$$t_k (\mathbf{w} \cdot \mathbf{x}_k + b) \geq 1 - \chi_k, \quad k = 1, \dots, m$$

$$(t_k \in \{-1, +1\})$$

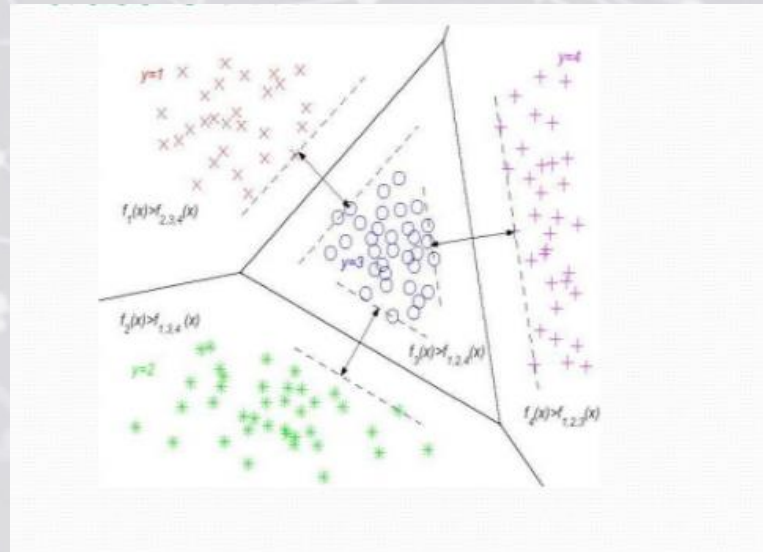
The effects of different kernels when learning a version of XOR with progressively more overlap (*left to right*) between the classes. *Top row: polynomial kernel of degree 3 with no slack variables, second row: polynomial of degree 3 with $C = 0.1$, third row: RBF kernel, no slack variables, bottom row: RBF kernel with $C = 0.1$.* The support vectors are highlighted, and the decision boundary is drawn for each case.

Multi-Class SVM Classification

- Previously we dealt with two-class classification for SVMs.
- Unfortunately, the theoretical work we used to derive the SVM algorithm only works for two classes.
- How then do we apply SVMs to the general N -class problem?

Multi-Class SVM Classification

- Simple answer: train an SVM that learns to classify one class from all other classes (i.e., “**one vs. all classification**”).
- So for N -classes we have **N SVMs**. For classification we merely choose the classifier that makes the strongest prediction.



Data Standardization

In general, we need to do data standardization for SVMs to avoid imbalance among feature scales:

Let μ_i denote the mean value of feature i in the training data, and σ_i denote the corresponding standard deviation. For each training example \mathbf{x} , replace each x_i as follows:

$$x'_i = \frac{x_i - \mu_i}{\sigma_i}$$

Scale the test data in the same way, using the μ_i and σ_i values computed from the training data, **not** the test data.

SVMs & Feature Selection

Goal: Select a subset of d features ($d < n$) in order to maximize classification performance with fewer features.

Types of Feature Selection Methods

- Filter
- Wrapper
- Embedded

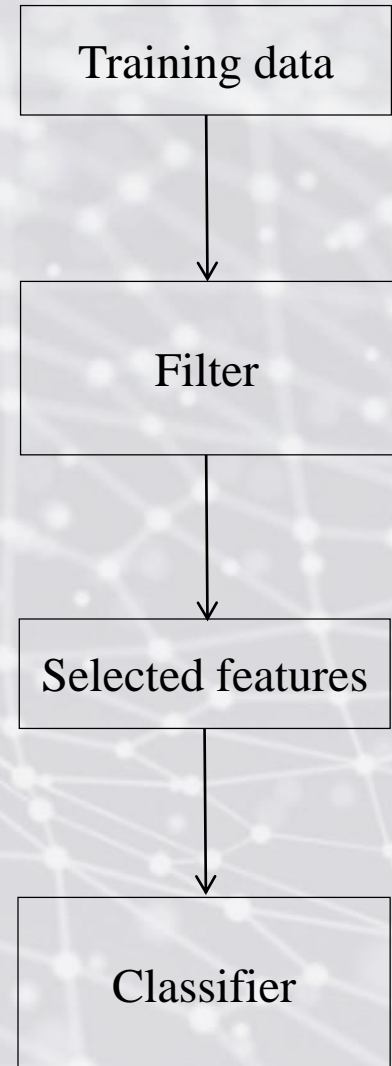
Filter Methods

Independent of the classification algorithm.

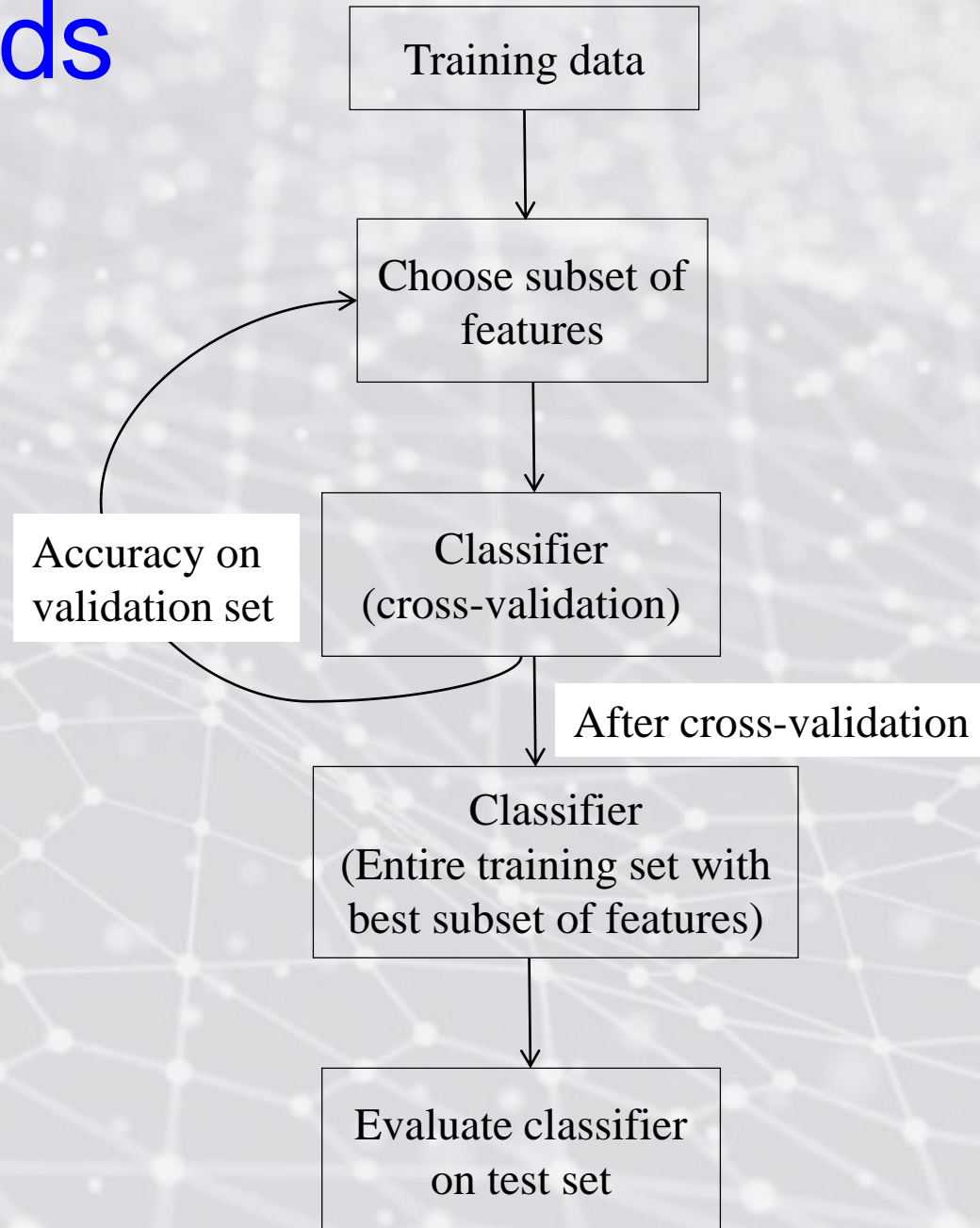
Apply a filtering function to the features before applying classifier.

Examples of filtering functions:

- Information gain of individual features
- Statistical variance of individual features
- Statistical correlation among features



Wrapper Methods



Filter Methods

Pros: Fast

Cons: Chosen filter might not be relevant for a specific kind of classifier.

Doesn't take into account interactions among features

Often hard to know how many features to select.

Filter Methods

Pros: Fast

Cons: Chosen filter might not be relevant for a specific kind of classifier.

Doesn't take into account interactions among features

Often hard to know how many features to select.

Wrapper Methods

Pros: Features are evaluated in context of classification

Wrapper method selects number of features to use

Cons: Slow

Filter Methods

Pros: Fast

Cons: Chosen filter might not be relevant for a specific kind of classifier.

Doesn't take into account interactions among features

Often hard to know how many features to select.

Intermediate method,
often used with SVMs:

Wrapper Methods

Pros: Features are evaluated in context of classification

Wrapper method selects number of features to use

Cons: Slow

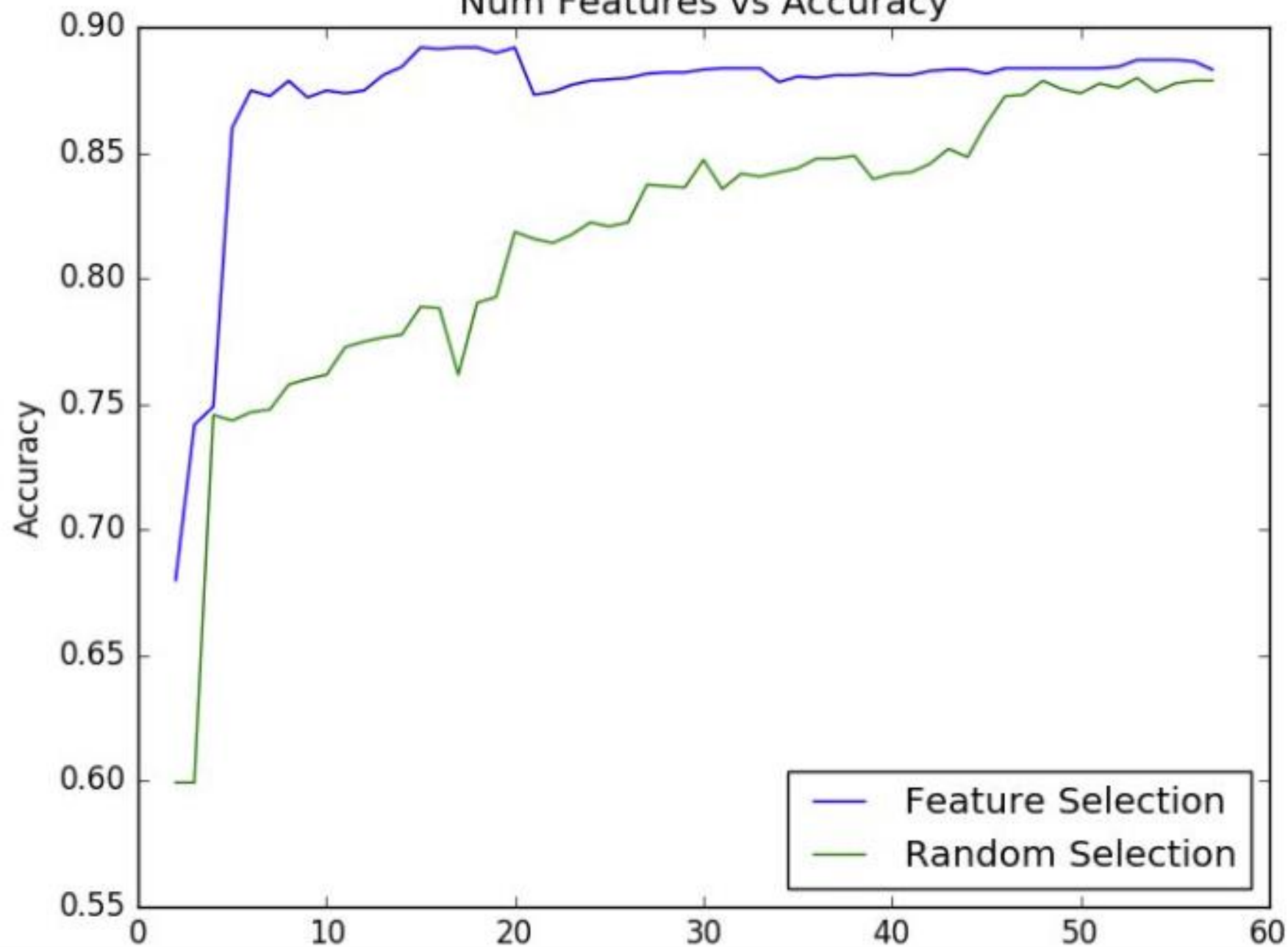
Train SVM using all features

Select features f_i with highest $|w_i|$

Retrain SVM with selected features

Test SVM on test set

Num Features vs Accuracy



Embedded Methods

Feature selection is intrinsic part of any learning algorithm

One example:

L_1 SVM: Instead of minimizing $\|\mathbf{w}\|$ (the “L2 norm”) we minimize the L_1 norm of the weight vector:

$$\|\mathbf{w}\|_1 = \sum_{i=1}^n |w_i|$$

Result is that most of the weights **go to zero**, leaving a small subset of the weights. (as mentioned previously: this is tantamount to putting a zero mean “prior” on w – in a Bayesian framework).

Cf., Field of “sparse coding.”

SVM for feature selection for Spam Detection Task

Feature	Weight
word_freq_hp	-19.63523611
char_freq_!	19.07223054
word_freq_george	-17.17445462
word_freq_free	16.84957797
word_freq_remove	15.97374434

Table 1: Top 5 features with highest weight magnitude from SVM. The sign of the weight corresponds to the feature correlating with “not spam” for negative values and “spam” for positive values.