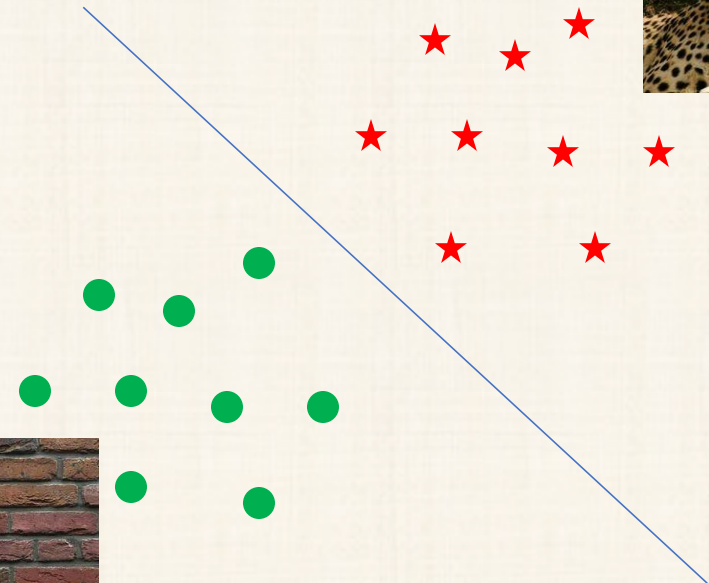


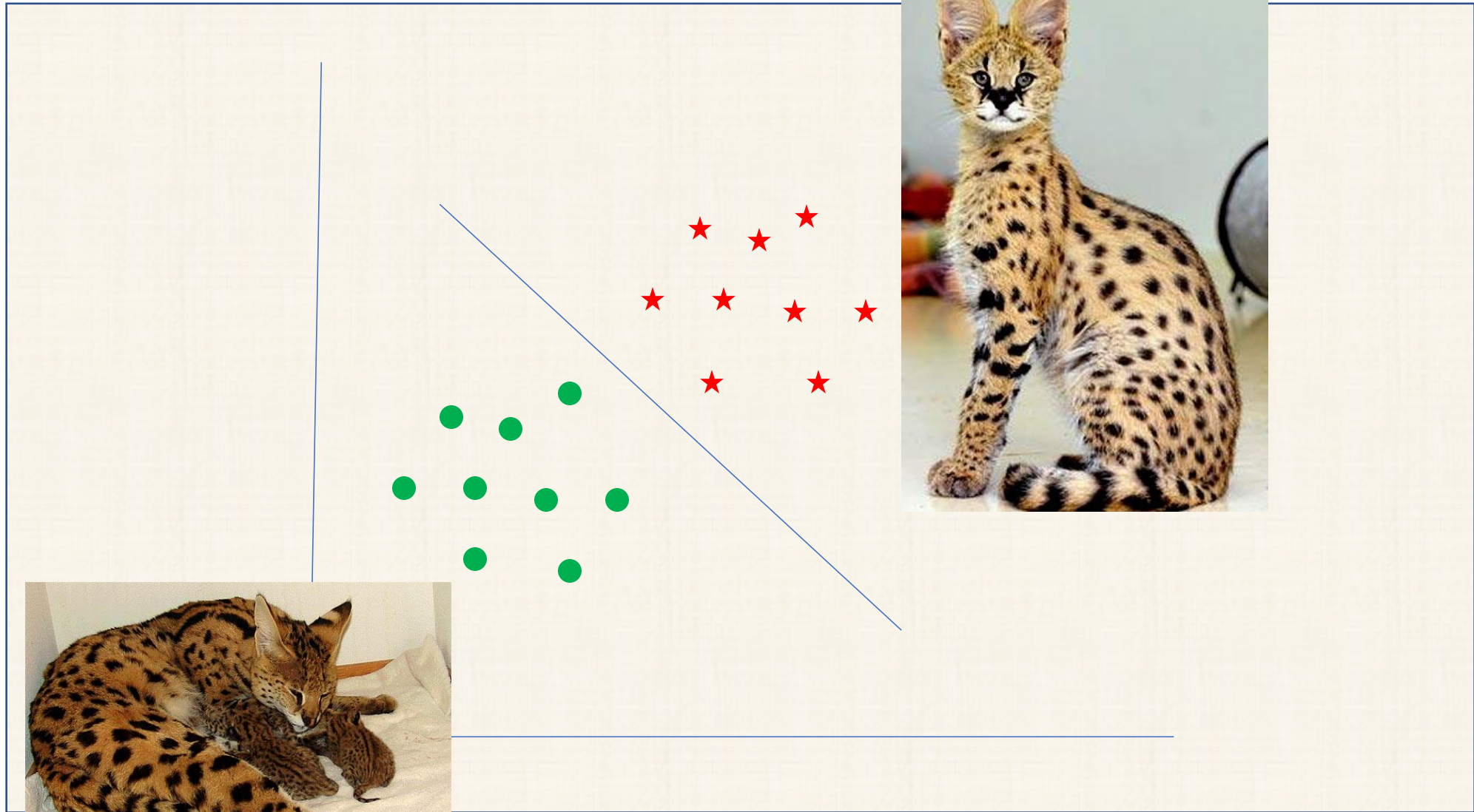
Support Vector Machines

Contents

- Motivation
- Linear SVM
 - Theory
 - Margin Maximization
 - Loss Function
 - Objective Function
 - Optimization
- Slack variables
- Support Vectors
- Code Components and Outline
- Build SVM from scratch - Code
- SVM using SKLearn – Code
- Solve Iris Petal problem using SVM - Code
- Kernel SVM
- Explore Kernels - Code
- One-Class SVM
- Multi-class SVM
- SVM Parameters
- Spam detection using SVM - Code







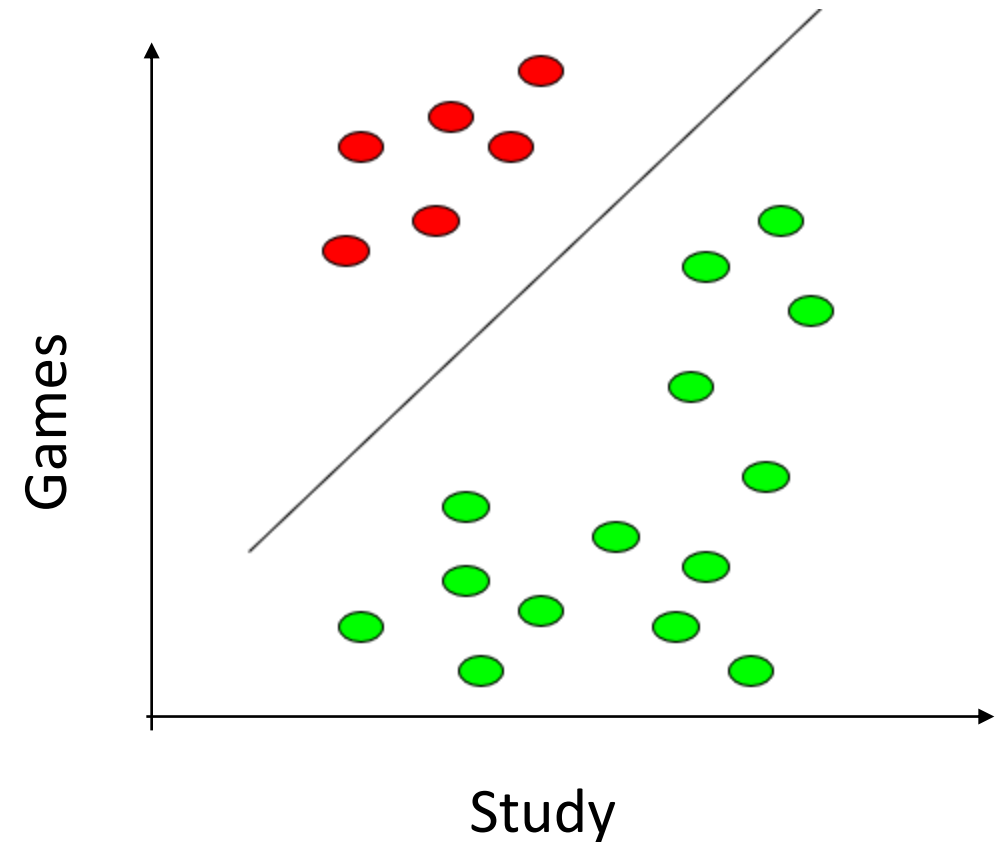
Illustration

Who will pass the exam ?

Two questions are asked to a set of subjects,

- How many hours studied
- How many hours played

● = spend time playing
● = spend time studying



Definitions

Define the hyperplane H such that:

$$\mathbf{x}_i \bullet \mathbf{w} + b \geq +1 \text{ when } y_i = +1$$

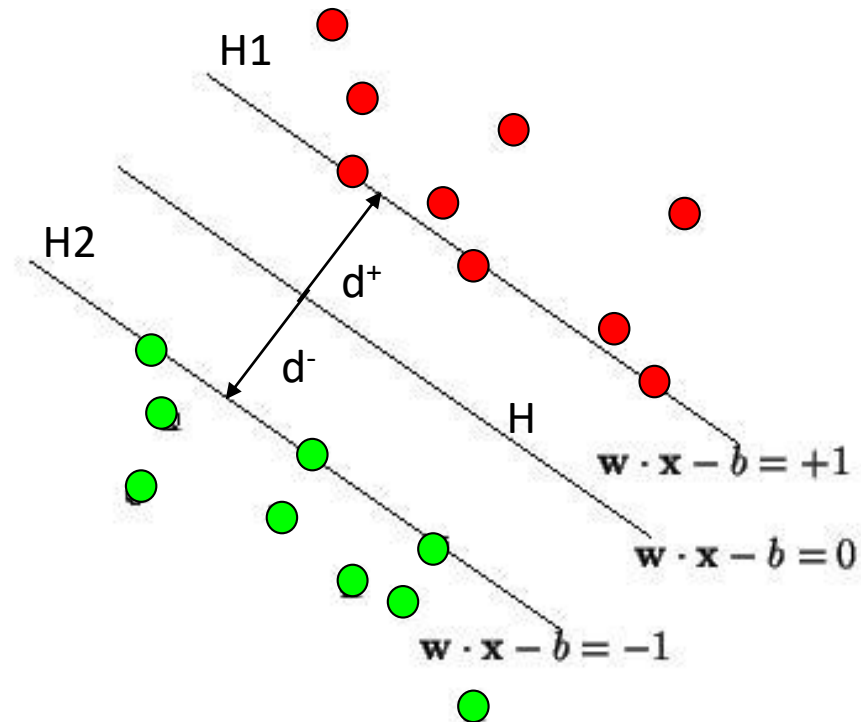
$$\mathbf{x}_i \bullet \mathbf{w} + b \leq -1 \text{ when } y_i = -1$$

H_1 and H_2 are the planes:

$$H_1: \mathbf{x}_i \bullet \mathbf{w} + b = +1$$

$$H_2: \mathbf{x}_i \bullet \mathbf{w} + b = -1$$

The points on the planes H_1 and H_2 are the Support Vectors



d^+ = the shortest distance to the closest positive point

d^- = the shortest distance to the closest negative point

The margin of a separating hyperplane is $d^+ + d^-$.

Maximizing the margin

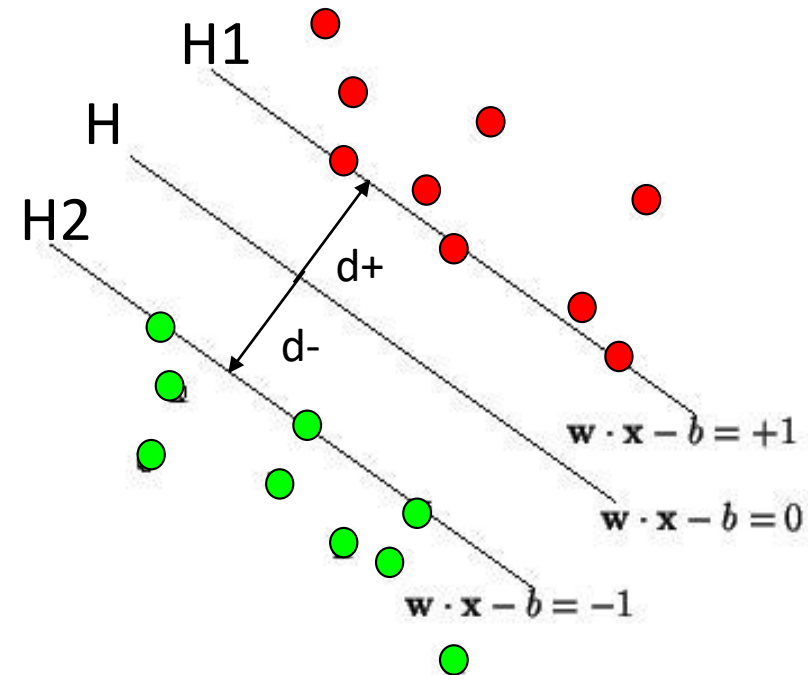
We want a classifier with as big margin as possible.

Recall the distance from a point (x_0, y_0) to a line:
 $Ax + By + c = 0$ is $|Ax_0 + By_0 + c| / \sqrt{A^2 + B^2}$

The distance between H and H1 is:

$$|\mathbf{w} \cdot \mathbf{x} + b| / \|\mathbf{w}\| = 1 / \|\mathbf{w}\|$$

The distance between H1 and H2 is: $2 / \|\mathbf{w}\|$



In order to maximize the margin, we need to minimize $\|\mathbf{w}\|$. With the condition that there are no datapoints between H1 and H2:

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 \text{ when } y_i = +1$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 \text{ when } y_i = -1$$

} Can be combined into $y_i(\mathbf{x}_i \cdot \mathbf{w}) \geq 1$

Constrained Optimization Problem

Minimize $\| \mathbf{w} \| = \langle \mathbf{w} \cdot \mathbf{w} \rangle$ subject to $y_i (\langle \mathbf{x}_i \cdot \mathbf{w} \rangle + b) \geq 1$ for all i

Lagrangian method : maximize $\inf_{\mathbf{w}} L(\mathbf{w}, b, \alpha)$, where

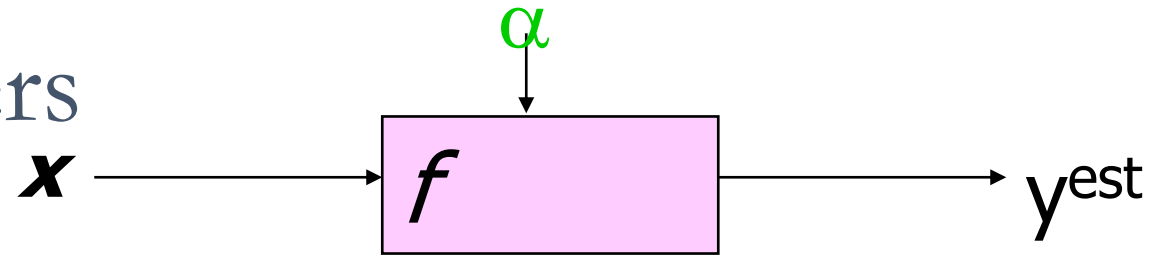
$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \| \mathbf{w} \|^2 - \sum_i \alpha_i [(y_i (\langle \mathbf{x}_i \cdot \mathbf{w} \rangle + b) - 1)]$$

At the extremum, the partial derivative of L with respect both \mathbf{w} and b must be 0. Taking the derivatives, setting them to 0, substituting back into L , and simplifying yields:

$$\text{Maximize } \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$$

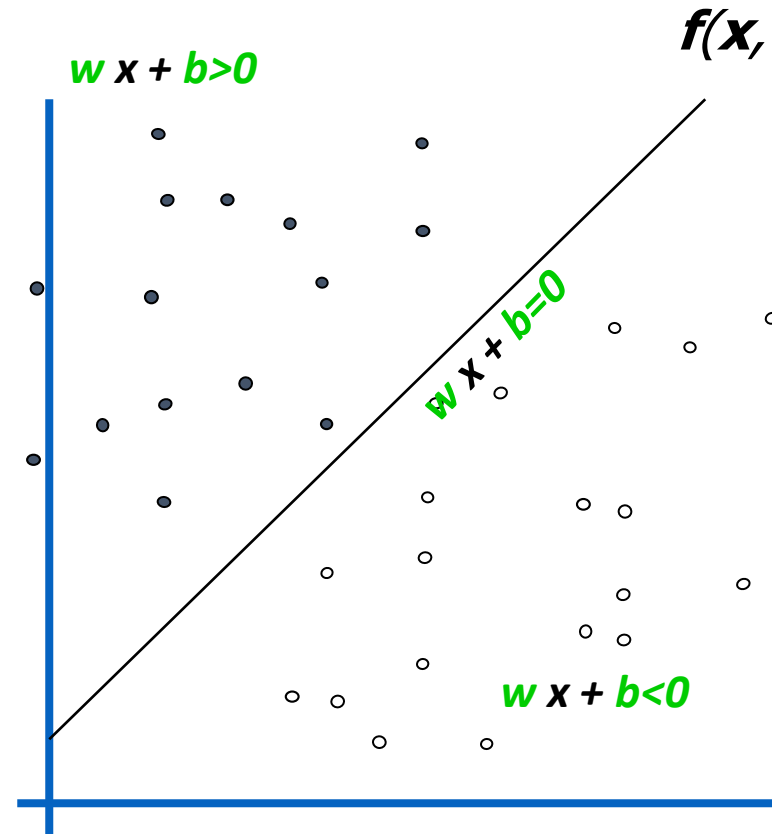
$$\text{subject to } \sum_i y_i \alpha_i = 0 \text{ and } \alpha_i \geq 0$$

Linear Classifiers



• denotes +1

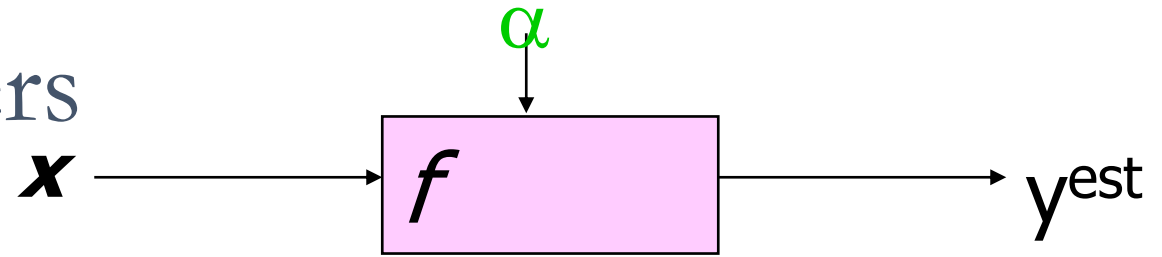
◦ denotes -1



$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w}\mathbf{x} + b)$$

How would you
classify this data?

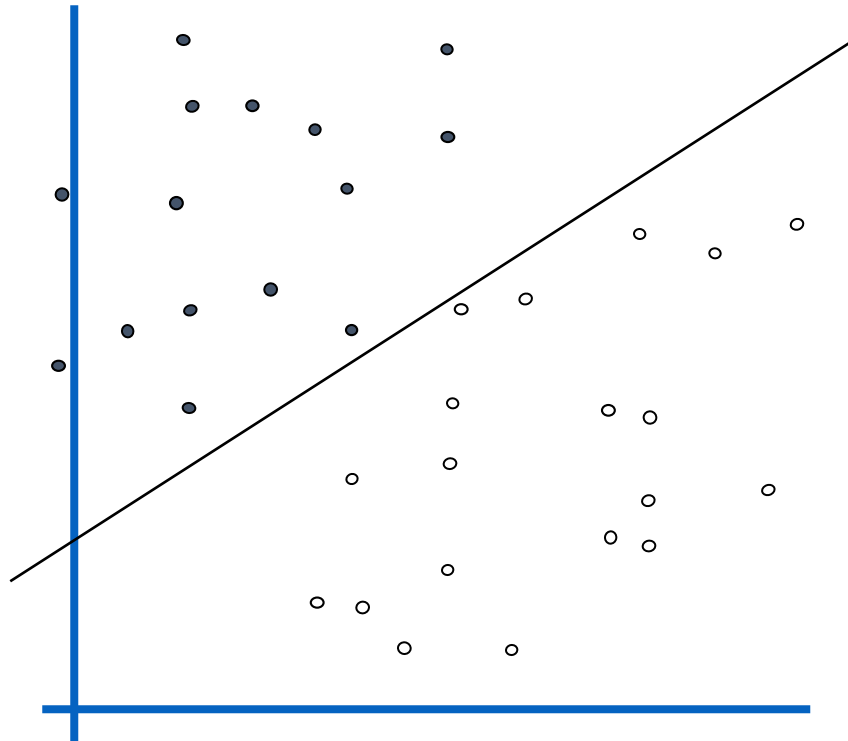
Linear Classifiers



$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

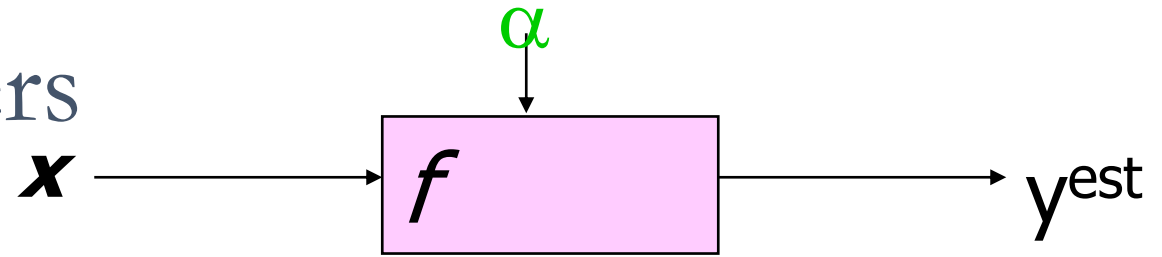
• denotes +1

○ denotes -1



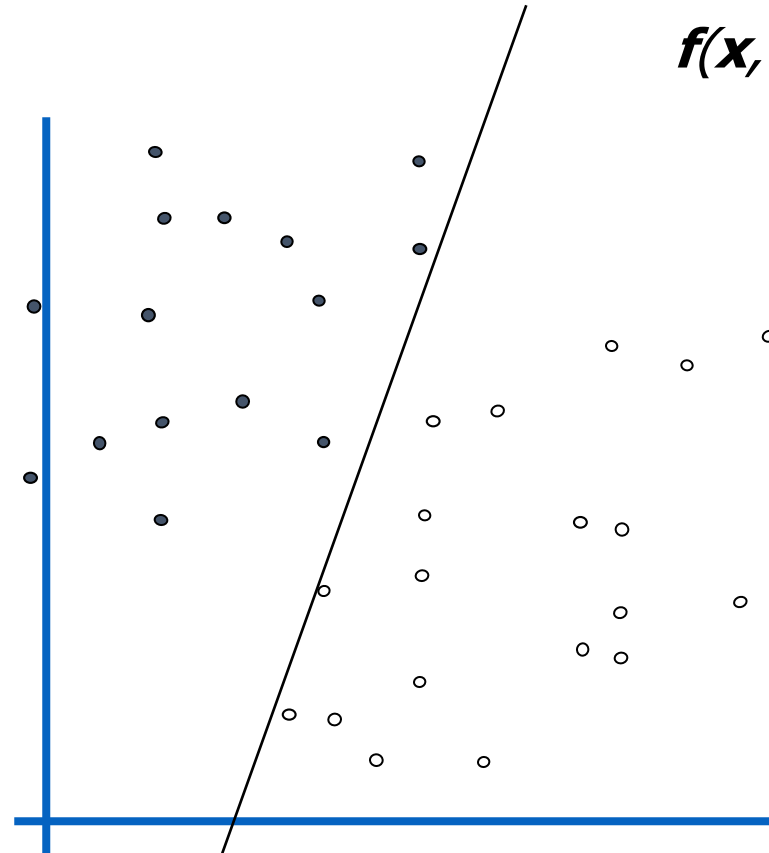
How would you
classify this data?

Linear Classifiers



• denotes +1

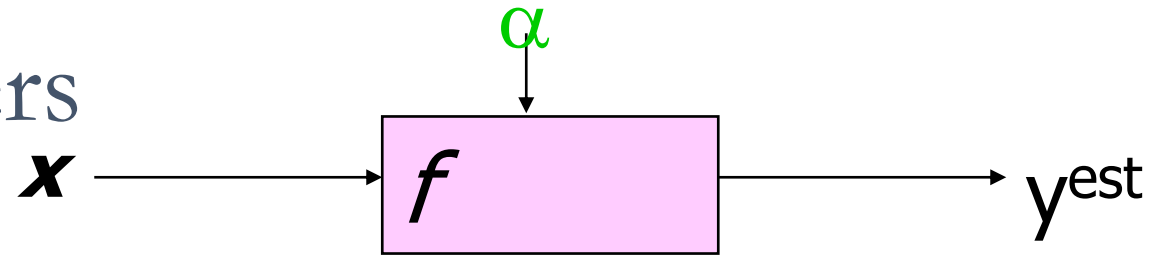
○ denotes -1



$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

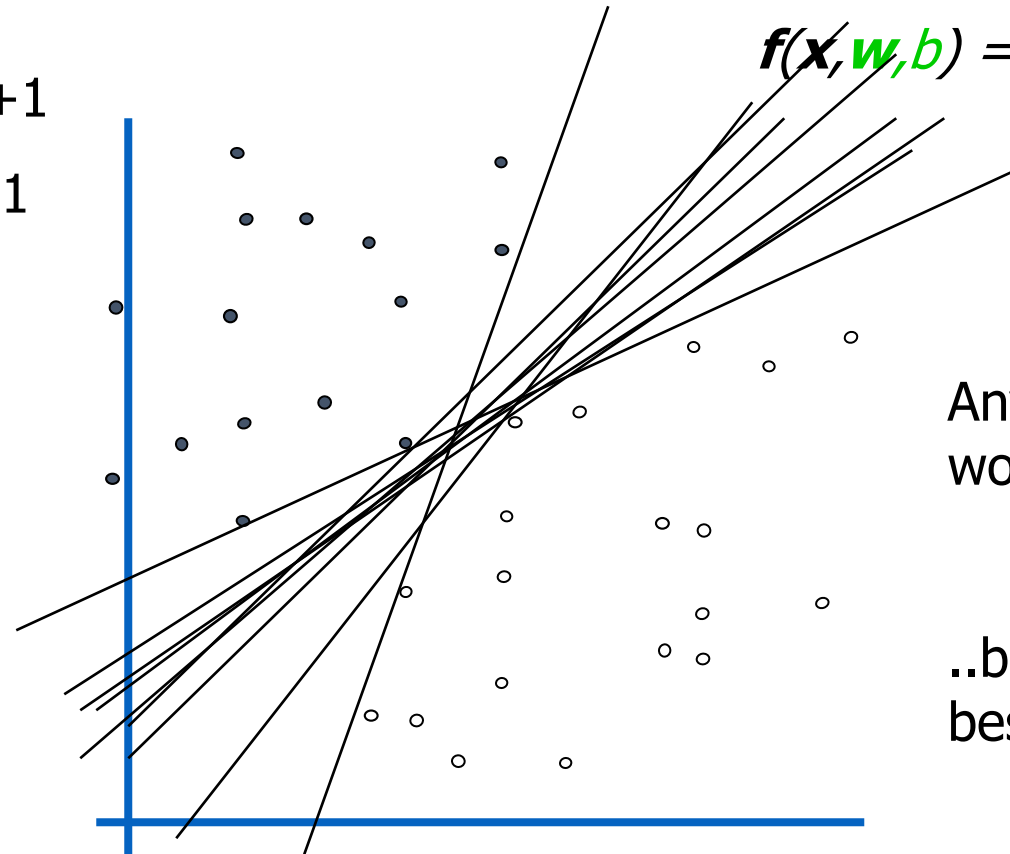
How would you
classify this data?

Linear Classifiers



• denotes +1

○ denotes -1

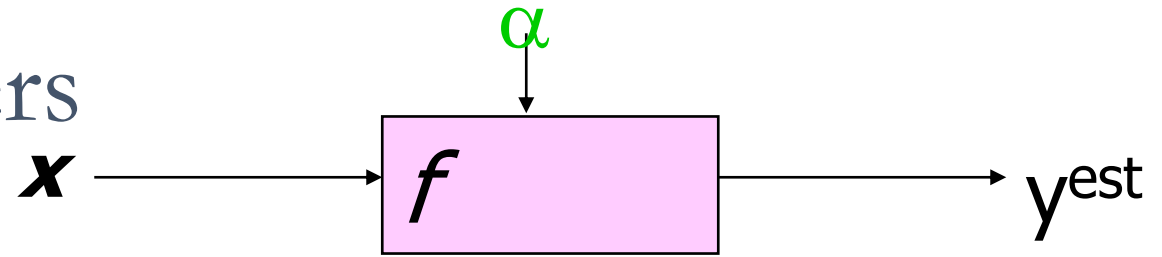


$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

Any of these
would be fine..

..but which is
best?

Linear Classifiers

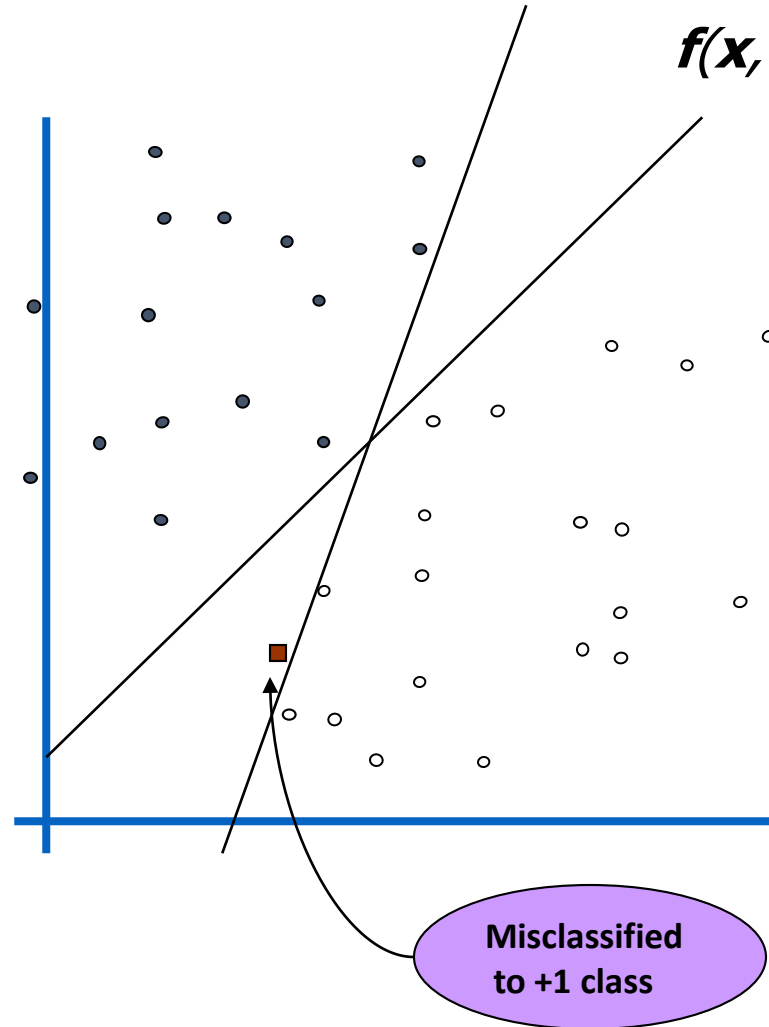


• denotes +1

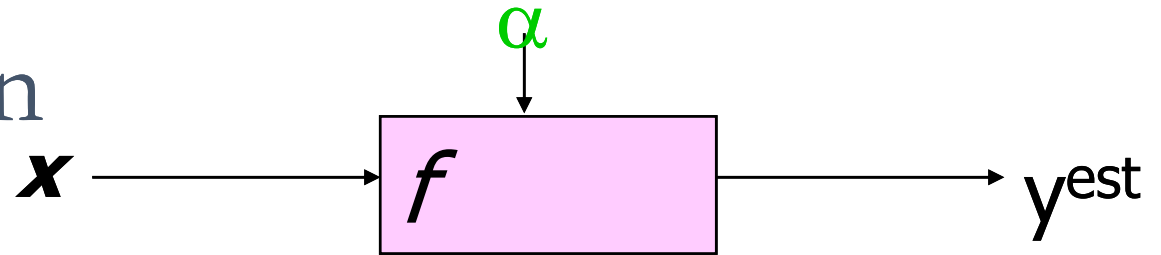
○ denotes -1

$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

How would you
classify this data?



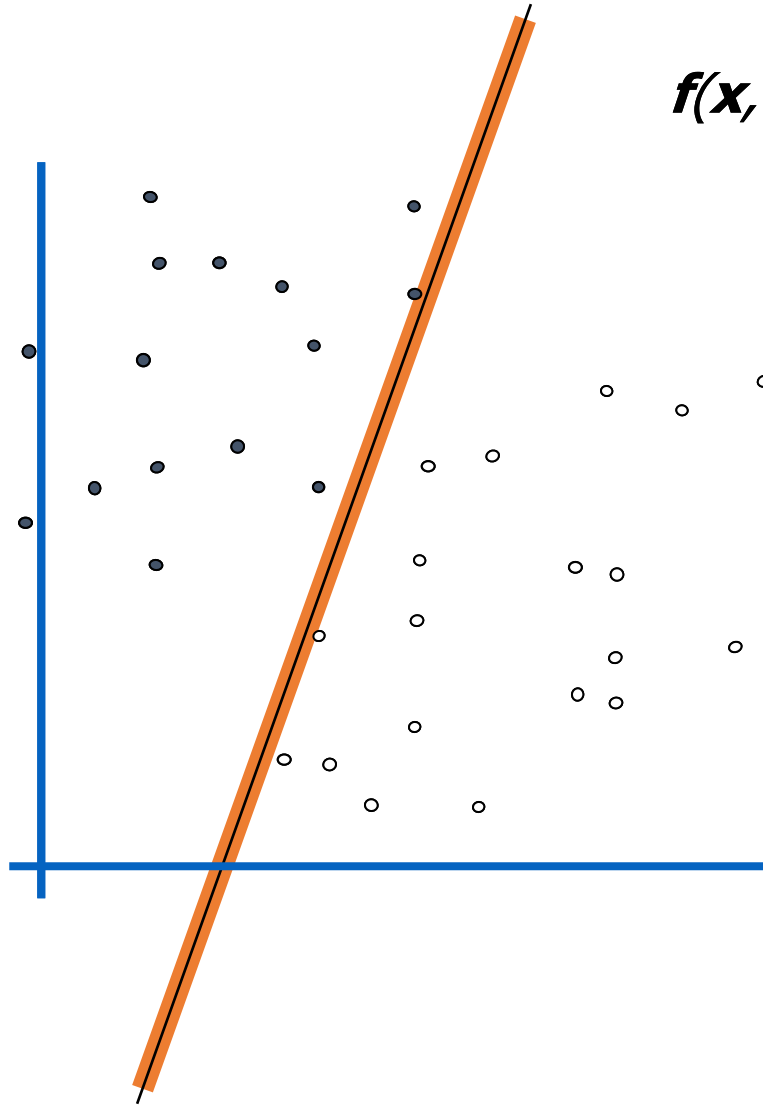
Classifier Margin



$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

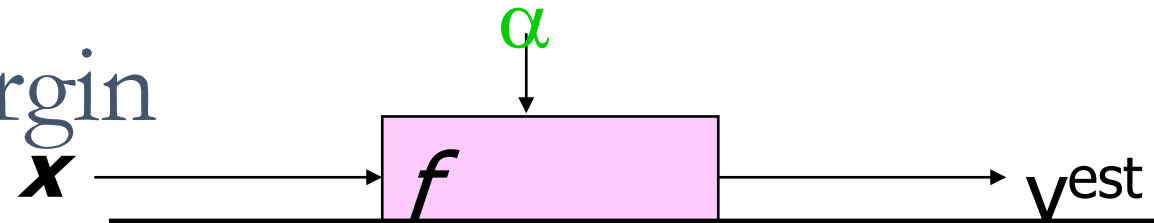
• denotes +1

○ denotes -1



Define the **margin** of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

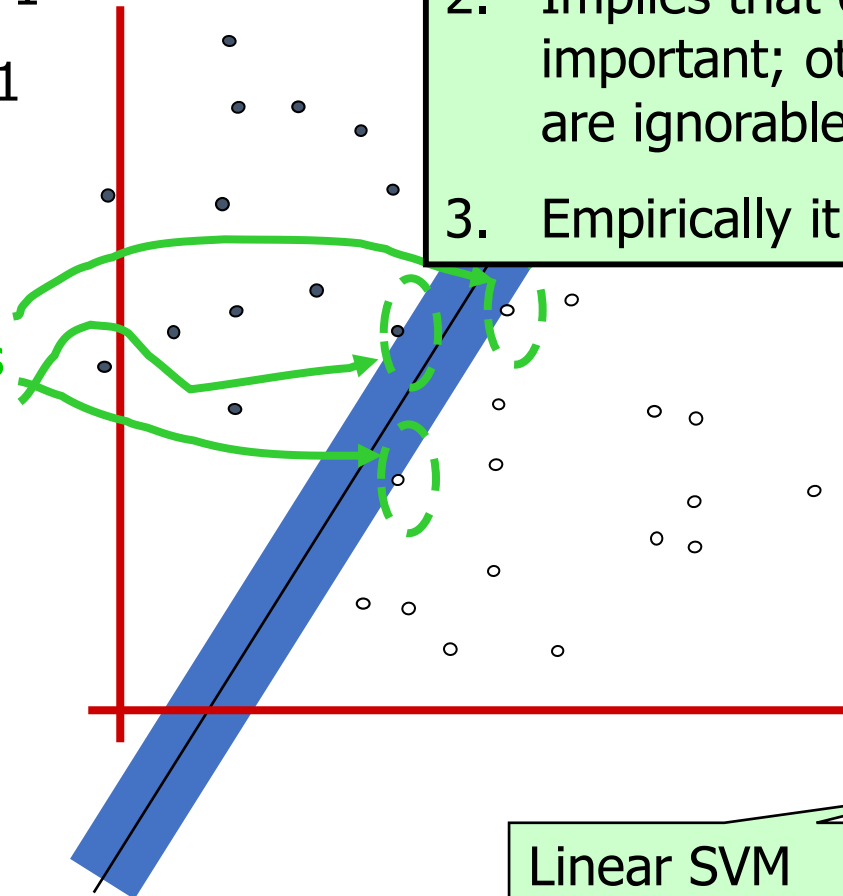
Maximum Margin



1. Maximizing the margin is good according to intuition and PAC theory
2. Implies that only support vectors are important; other training examples are ignorable.
3. Empirically it works very very well.

• denotes +1
○ denotes -1

Support Vectors
are those datapoints that the margin pushes up against

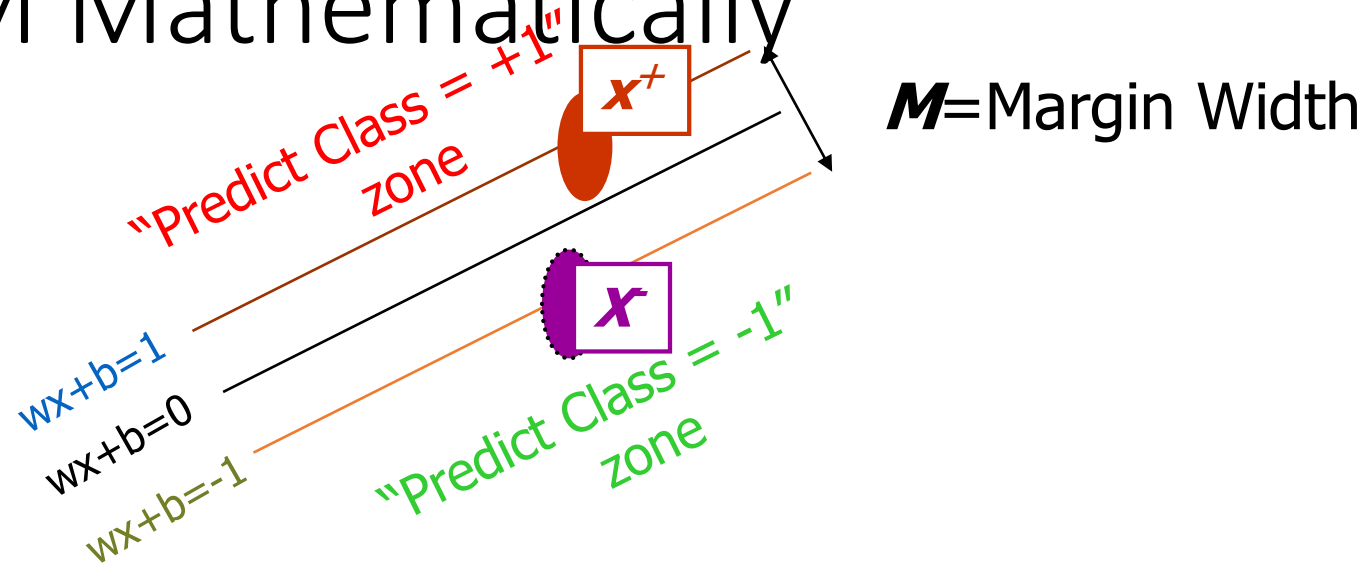


linear classifier
with the, um,
maximum margin.

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

Linear SVM

Linear SVM Mathematically



What we know:

- $w \cdot x^+ + b = +1$
- $w \cdot x^- + b = -1$
- $w \cdot (x^+ - x^-) = 2$

$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

Linear SVM Mathematically

- **Goal: 1) Correctly classify all training data**

$$\begin{aligned}
 wx_i + b &\geq 1 && \text{if } y_i = +1 \\
 wx_i + b &\leq -1 && \text{if } y_i = -1 \\
 y_i(wx_i + b) &\geq 1 && \text{for all } i
 \end{aligned}$$

2) Maximize the Margin

same as minimize

$$M = \frac{2}{|w|}$$

$$\frac{1}{2} w^t w$$

- **We can formulate a Quadratic Optimization Problem and solve for w and b**

$$\begin{aligned} \text{Minimize} \quad & \Phi(w) = \frac{1}{2} w^t w \\ \text{subject to} \quad & y_i (w x_i + b) \geq 1 \quad \forall i \end{aligned}$$

Solving the Optimization Problem

Find \mathbf{w} and b such that

$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w}$ is minimized;

and for all $\{(\mathbf{x}_i, y_i)\}$: $y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

- **Need to optimize a *quadratic* function subject to *linear* constraints.**
- **Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them.**
- **The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:**

Find $\alpha_1 \dots \alpha_N$ such that

$Q(\boldsymbol{\alpha}) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and

(1) $\sum \alpha_i y_i = 0$

(2) $\alpha_i \geq 0$ for all α_i

The Optimization Problem Solution

- The solution has the form:

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i \quad b = y_k - \mathbf{w}^T \mathbf{x}_k \text{ for any } \mathbf{x}_k \text{ such that } \alpha_k \neq 0$$

- Each non-zero α_i indicates that corresponding \mathbf{x}_i is a support vector.
- Then the classifying function will have the form:

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

- Notice that it relies on an *inner product* between the test point \mathbf{x} and the support vectors \mathbf{x}_i – we will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x}_i^T \mathbf{x}_j$ between all pairs of training points.

Components of an SVM

Fit ()

Predict ()

SVMObjective ()

SVMDecisionBoundary ()

Score()

Slack Variables

What to do if there is no separating plane

- Use a much bigger set of features.
 - This looks as if it would make the computation hopelessly slow, but in the next part of the lecture we will see how to use the “kernel” trick to make the computation fast even with huge numbers of features.
- Extend the definition of maximum margin to allow non-separating planes.
 - This can be done by using “slack” variables

Introducing slack variables

- Slack variables are constrained to be non-negative. When they are greater than zero they allow us to cheat by putting the plane closer to the datapoint than the margin. So we need to minimize the amount of cheating. This means we have to pick a value for lambda (this sounds familiar!)

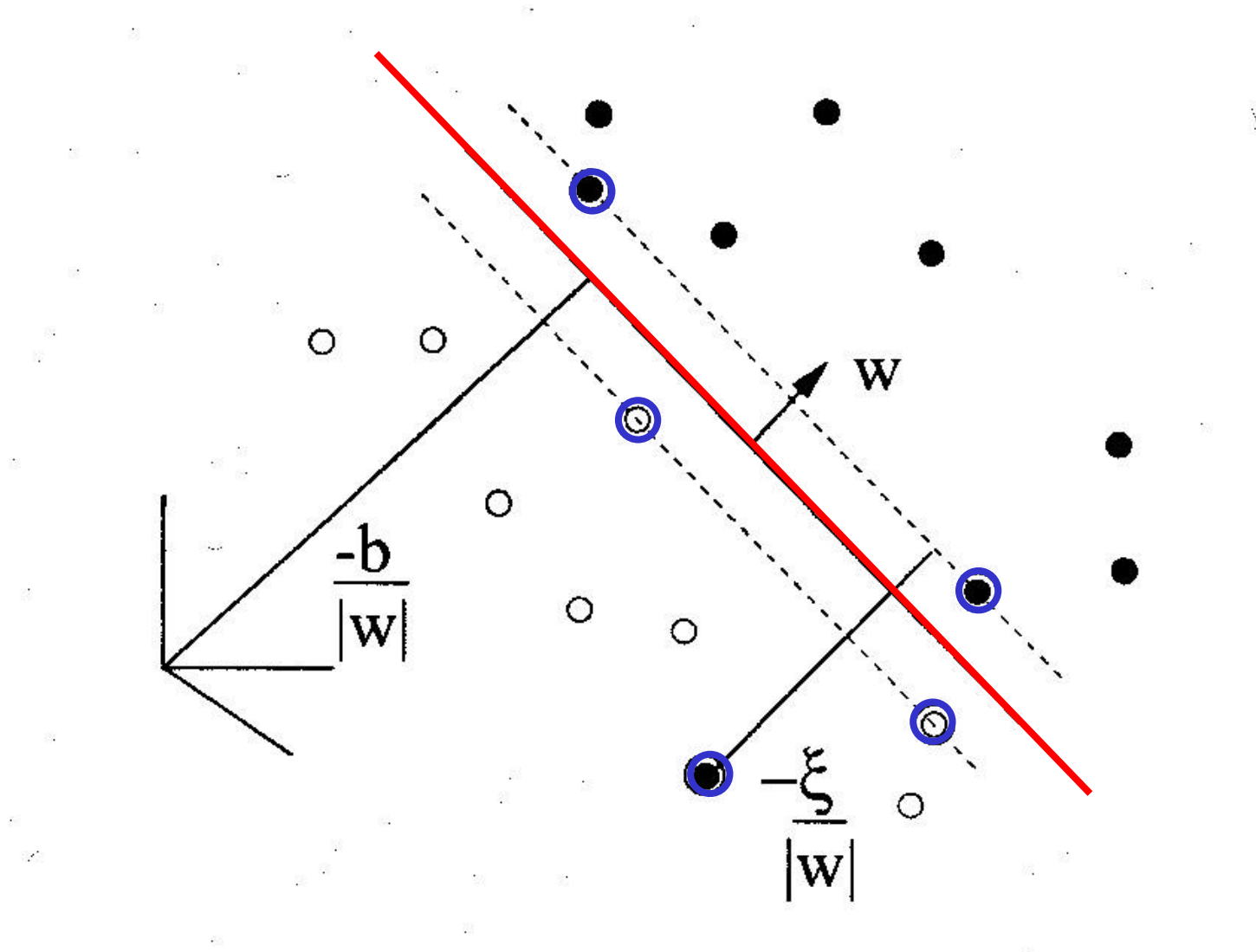
$$\mathbf{w} \cdot \mathbf{x}^c + b \geq +1 - \xi^c \quad \text{for positive cases}$$

$$\mathbf{w} \cdot \mathbf{x}^c + b \leq -1 + \xi^c \quad \text{for negative cases}$$

$$\text{with } \xi^c \geq 0 \quad \text{for all } c$$

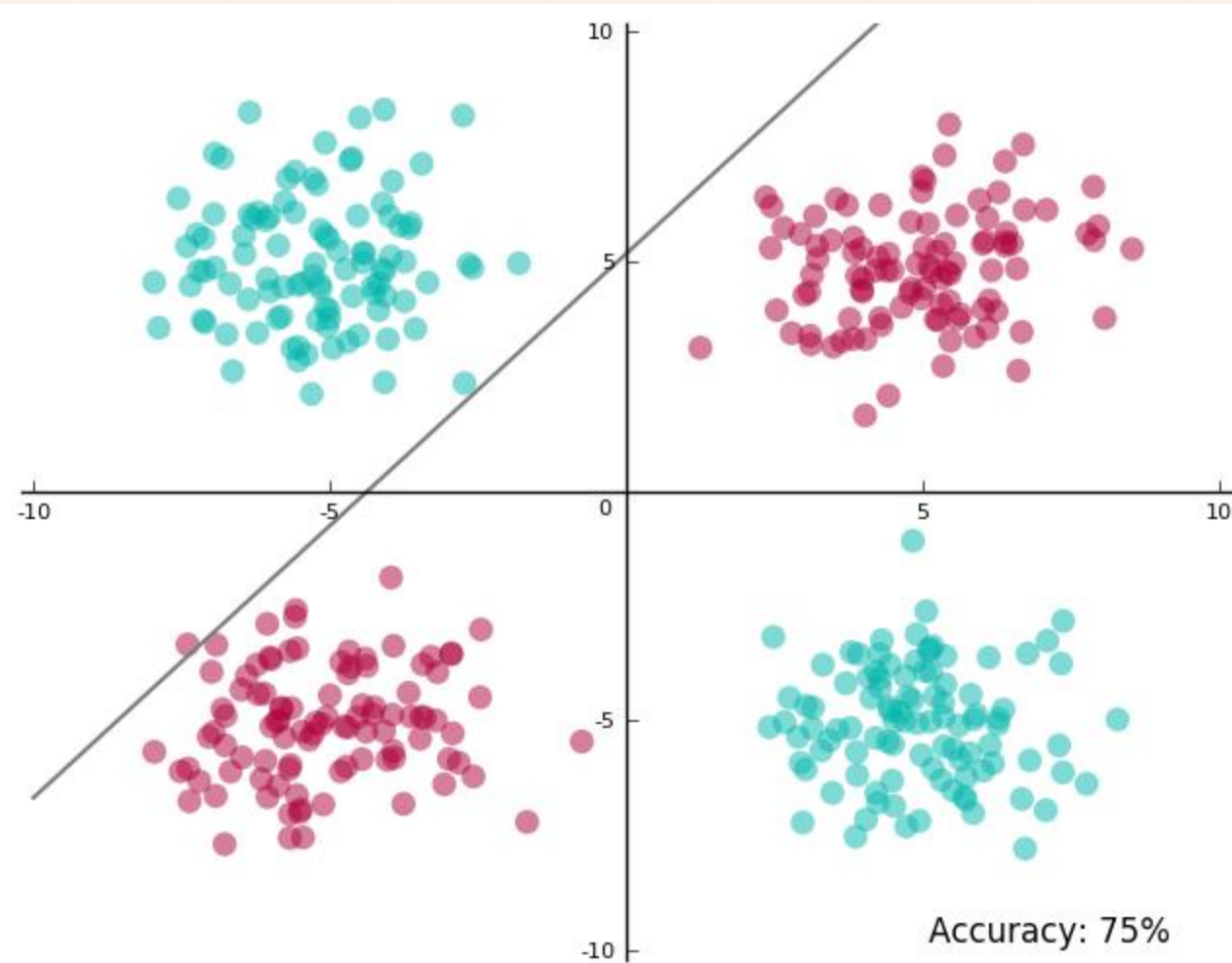
$$\text{and } \frac{\|\mathbf{w}\|^2}{2} + \lambda \sum_c \xi^c \quad \text{as small as possible}$$

A picture of the best plane with a slack variable



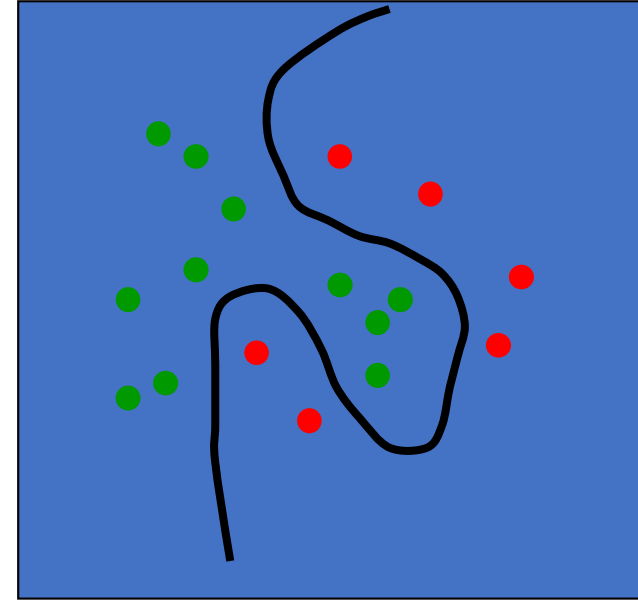
Kernels

Non-linearly Separable Data



How to make a plane curved

- Fitting hyperplanes as separators is mathematically easy.
 - The mathematics is linear.
- By replacing the raw input variables with a much larger set of features we get a nice property:
 - A planar separator in the high-dimensional space of feature vectors is a curved separator in the low dimensional space of the raw input variables.



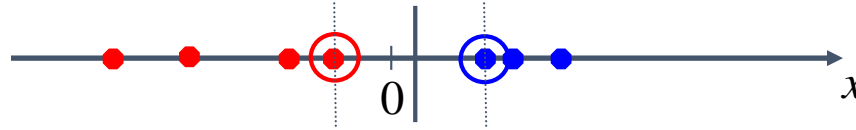
A planar separator in a 20-D feature space projected back to the original 2-D space

A potential problem and a magic solution

- If we map the input vectors into a **very** high-dimensional feature space, surely the task of finding the maximum-margin separator becomes computationally intractable?
 - The mathematics is all linear, which is good, but the vectors have a huge number of components.
 - So taking the scalar product of two vectors is very expensive.
- The way to keep things tractable is to use **“the kernel trick”**
- The kernel trick makes your brain hurt when you first learn about it, but its actually very simple.

Another Example

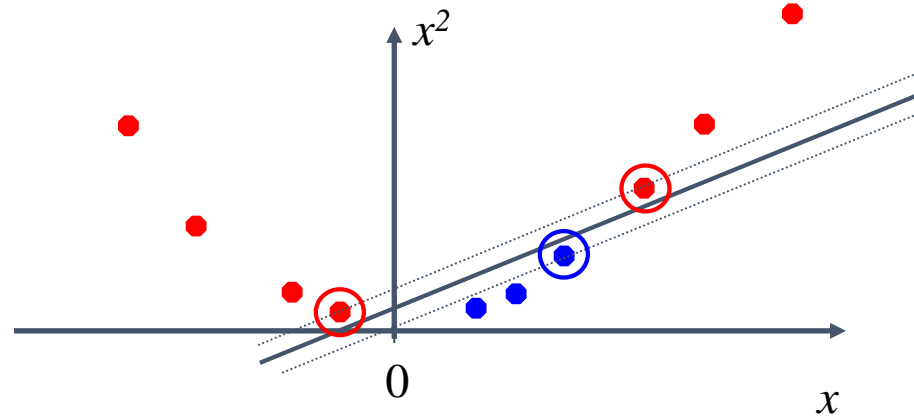
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?



- How about... mapping data to a higher-dimensional space:



What the kernel trick achieves

- All of the computations that we need to do to find the maximum-margin separator can be expressed in terms of scalar products between pairs of datapoints (in the high-dimensional feature space).
- These scalar products are the only part of the computation that depends on the dimensionality of the high-dimensional space.
 - So if we had a fast way to do the scalar products we would not have to pay a price for solving the learning problem in the high-D space.
- The kernel trick is just a magic way of doing scalar products a whole lot faster than is usually possible.
 - It relies on choosing a way of mapping to the high-dimensional feature space that allows fast scalar products.

The kernel trick

- For many mappings from a low-D space to a high-D space, there is a simple operation on two vectors in the low-D space that can be used to compute the scalar product of their two images in the high-D space.

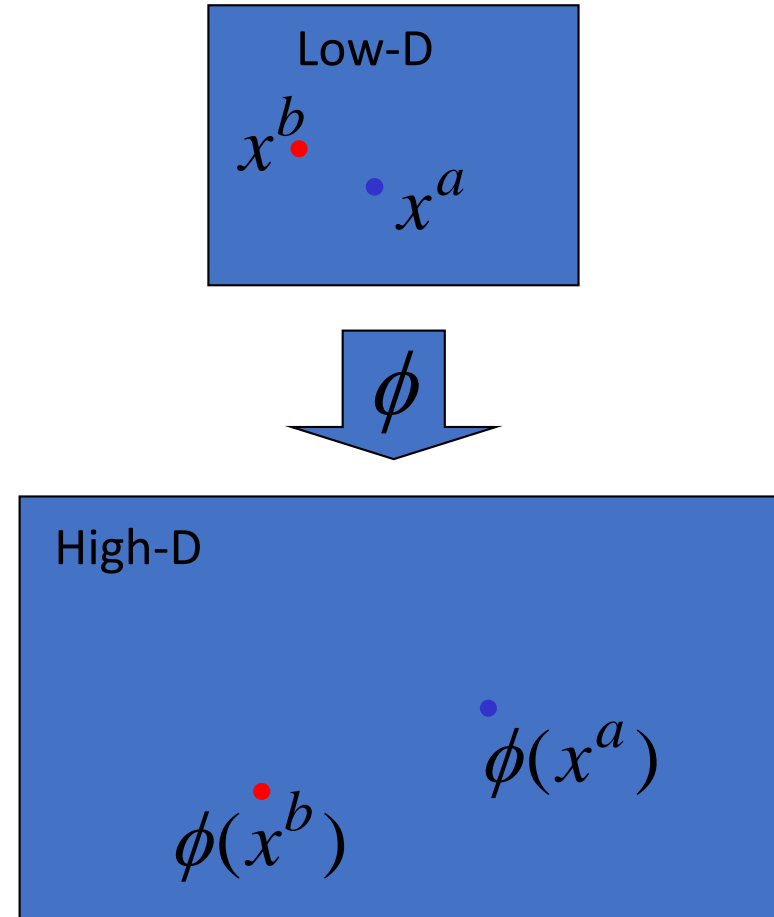
$$K(x^a, x^b) = \phi(x^a) \cdot \phi(x^b)$$



Letting the
kernel do
the work



doing the scalar
product in the
obvious way



Dealing with the test data

- If we choose a mapping to a high-D space for which the kernel trick works, we do not have to pay a computational price for the high-dimensionality when we find the best hyper-plane.
 - We cannot express the hyperplane by using its normal vector in the high-dimensional space because this vector would have a huge number of components.
 - Luckily, we can express it in terms of the support vectors.
- But what about the test data. We cannot compute the scalar product $\mathbf{w} \cdot \phi(\mathbf{x})$ because its in the high-D space.

$$\mathbf{w} \cdot \phi(\mathbf{x})$$

Dealing with the test data

- We need to decide which side of the separating hyperplane a test point lies on and this requires us to compute a scalar product.
- We can express this scalar product as a weighted average of scalar products with the stored support vectors
 - This could still be slow if there are a lot of support vectors .

The classification rule

- The final classification rule is quite simple:

$$bias + \sum_{s \in SV} w_s K(x^{test}, x^s) > 0$$

↑
The set of
support vectors

- All the cleverness goes into selecting the support vectors that maximize the margin and computing the weight to use on each support vector.
- We also need to choose a good kernel function and we may need to choose a lambda for dealing with non-separable cases.

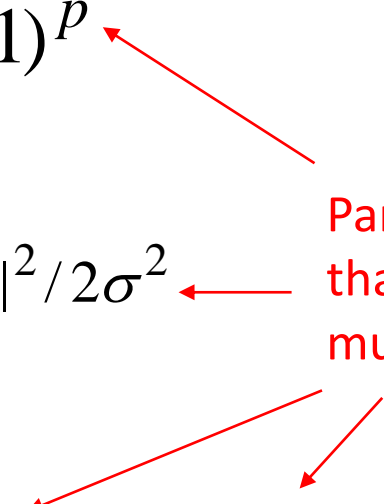
Some commonly used kernels

Polynomial: $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p$

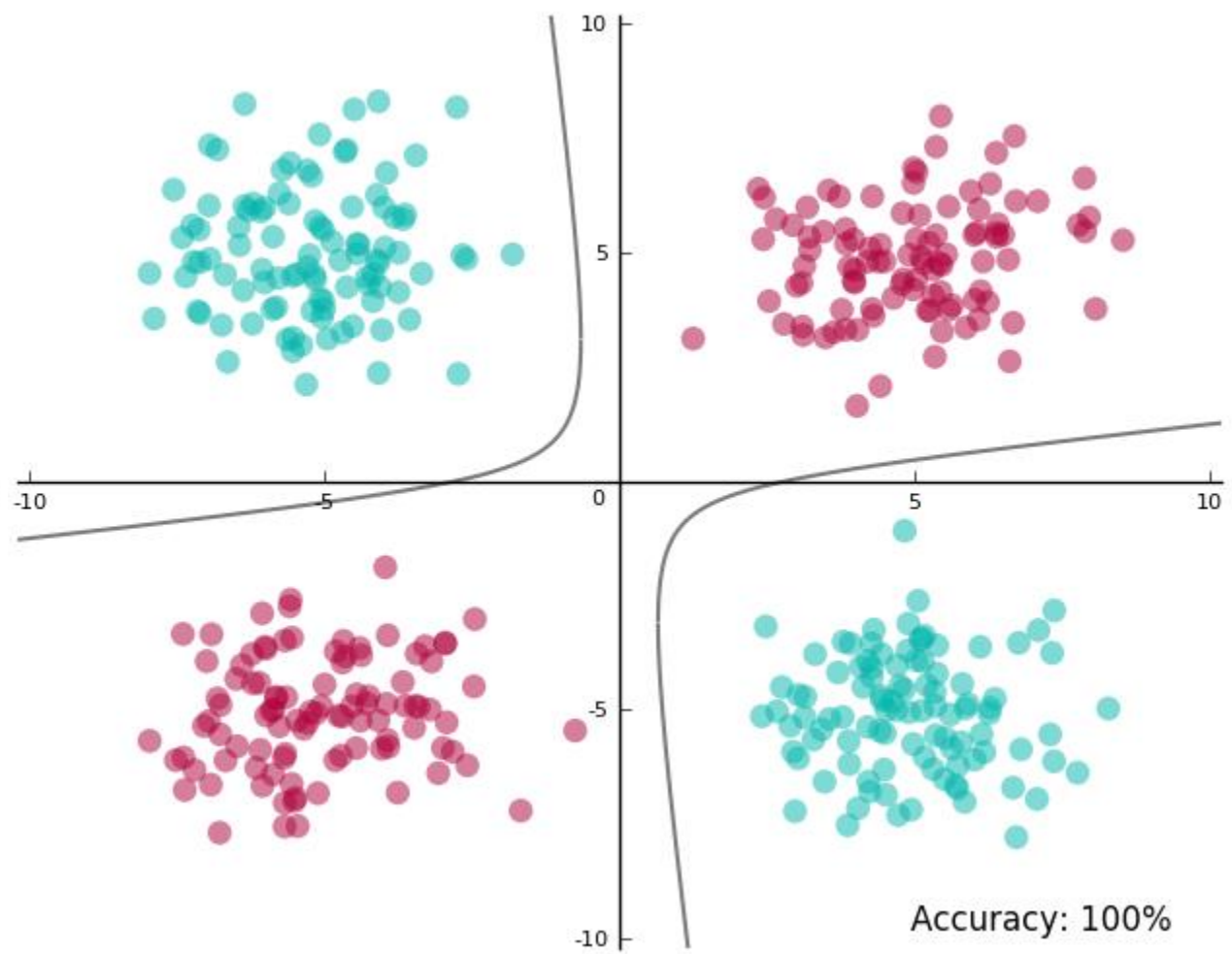
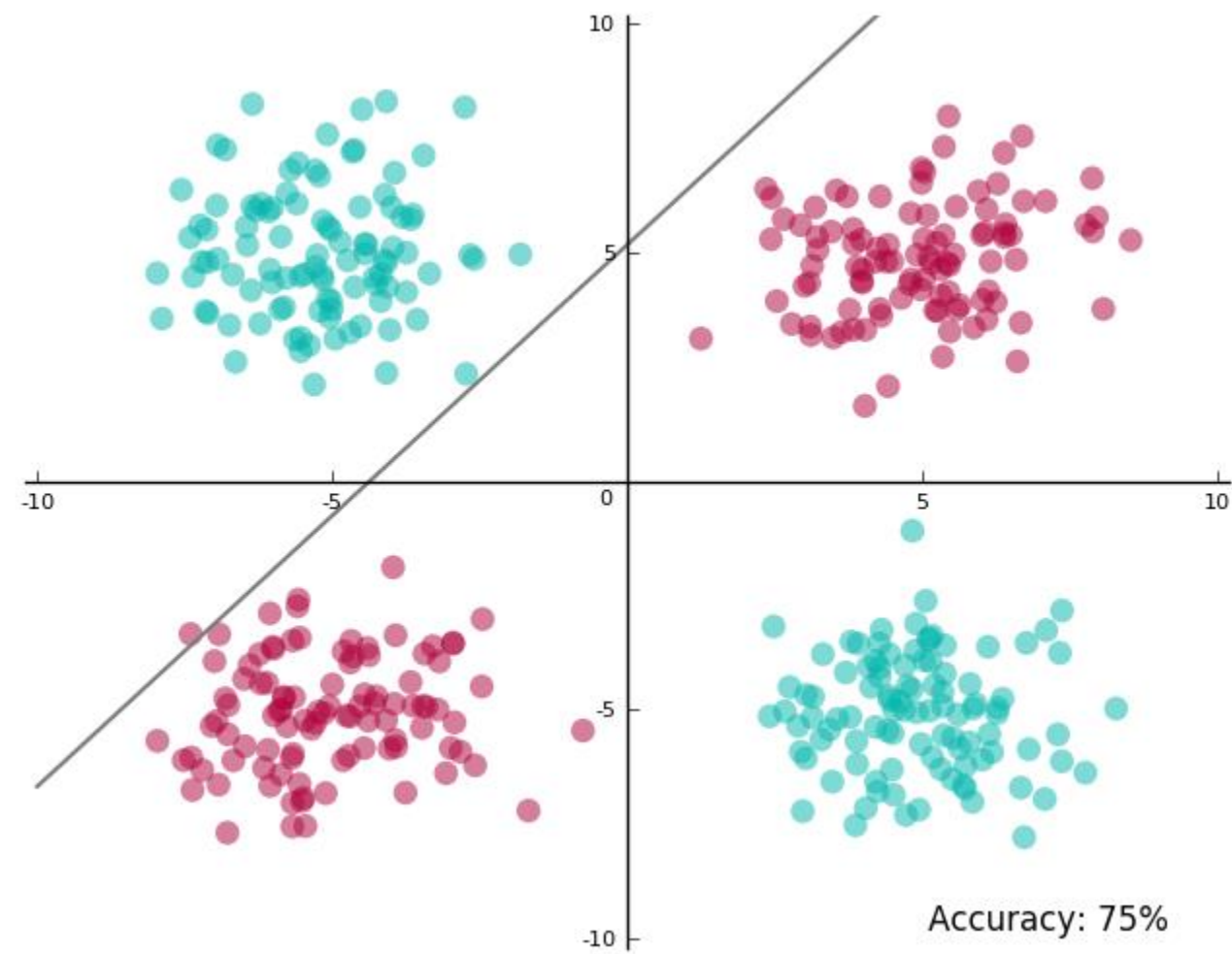
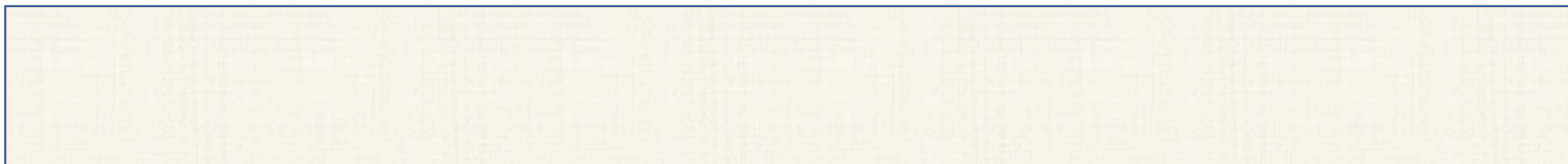
Gaussian radial
basis function $K(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x} - \mathbf{y}\|^2 / 2\sigma^2}$

Neural net: $K(\mathbf{x}, \mathbf{y}) = \tanh(k \mathbf{x} \cdot \mathbf{y} - \delta)$

Parameters
that the user
must choose



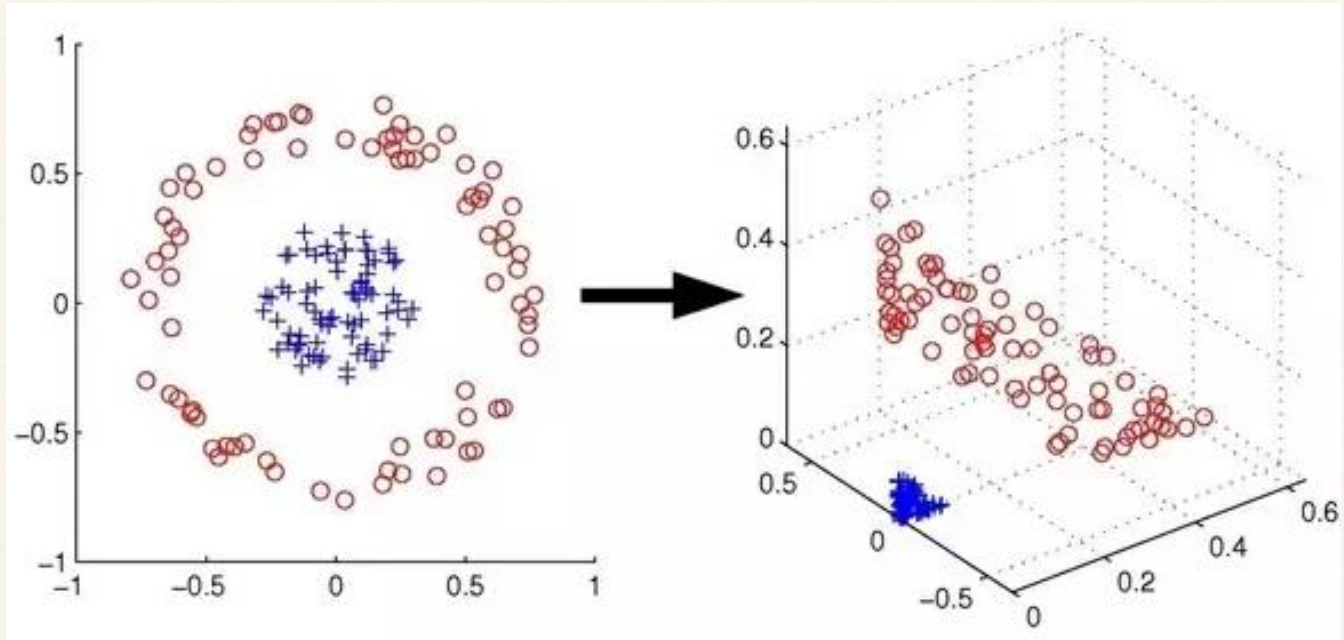
For the neural network kernel, there is one “hidden unit” per support vector, so the process of fitting the maximum margin hyperplane decides how many hidden units to use. Also, it may violate Mercer’s condition.



- How to decide what space to project the data onto?
- Cover's theorem!
- Data is more *likely* to be linearly separable when projected onto higher dimensions
- In practice, we try out a few high-dimensional projections to see what works
- In fact, we can project data onto *infinite* dimensions and that often works pretty well

- SVMs use Kernels to do these projections – and these are pretty fast and less computationally complex
- Kernels also help in projecting data to infinite dimensions in a much easier way
- Imagine if you had to project the data yourself, how do you represent or store infinite dimensions!
- Lets learn more about Kernels...

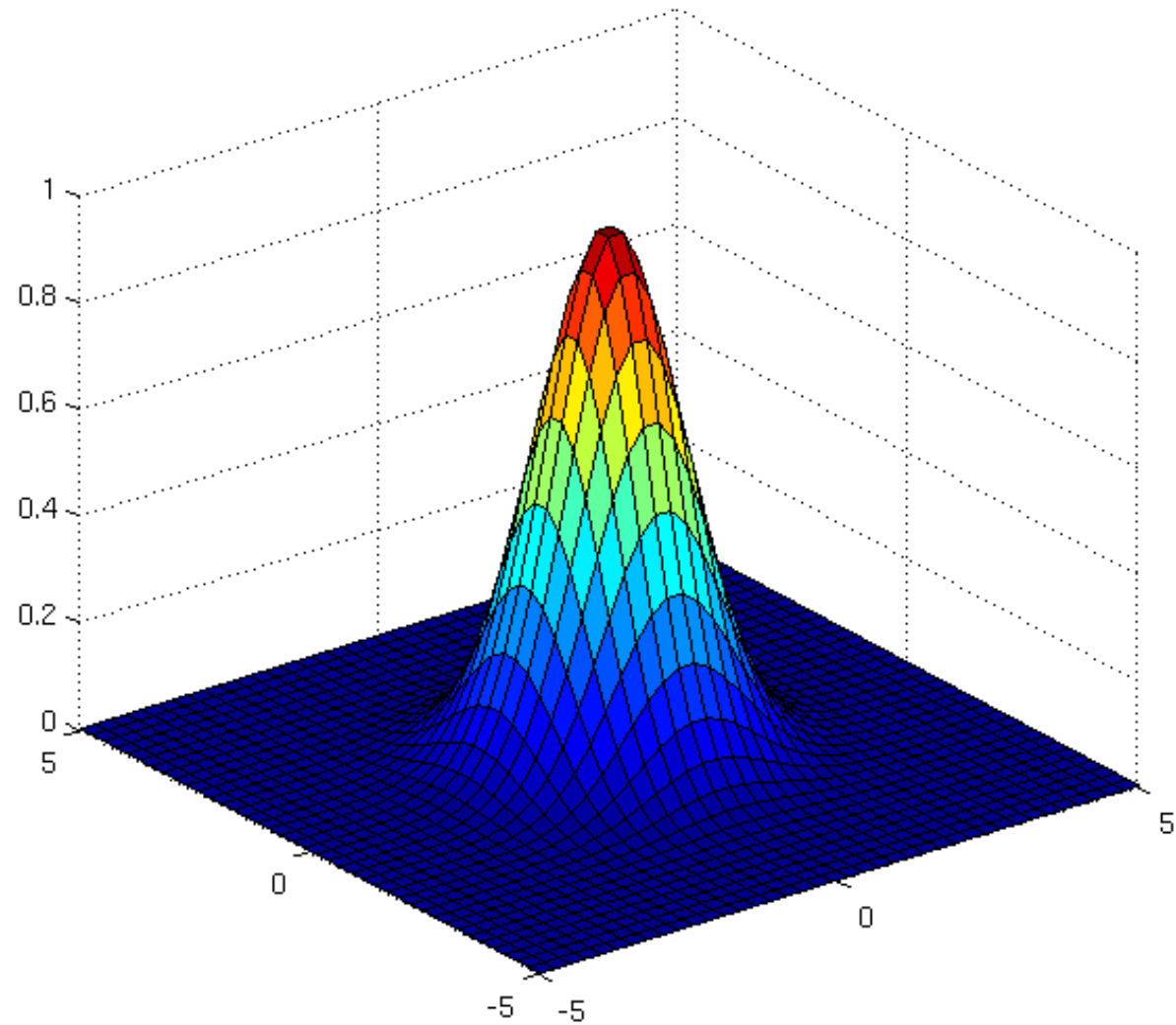
- Lets consider another example:



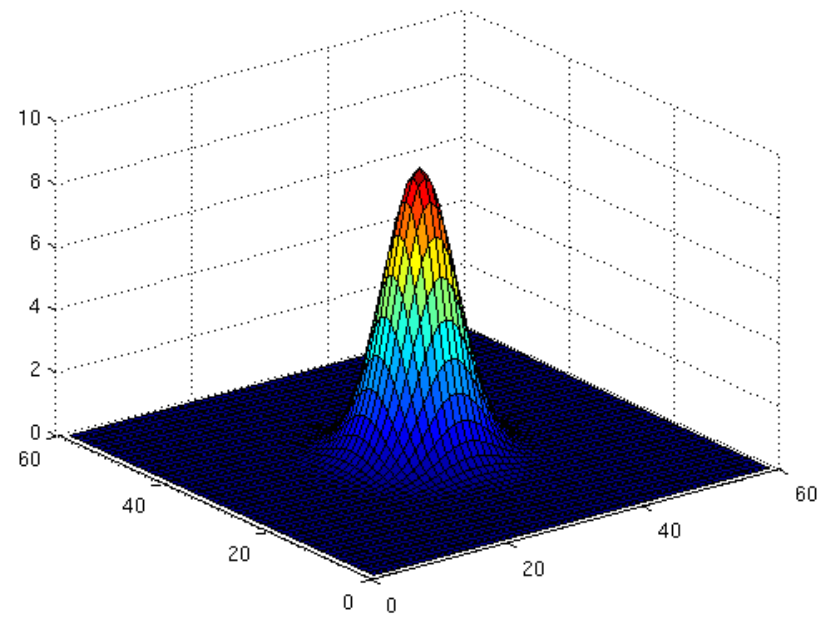
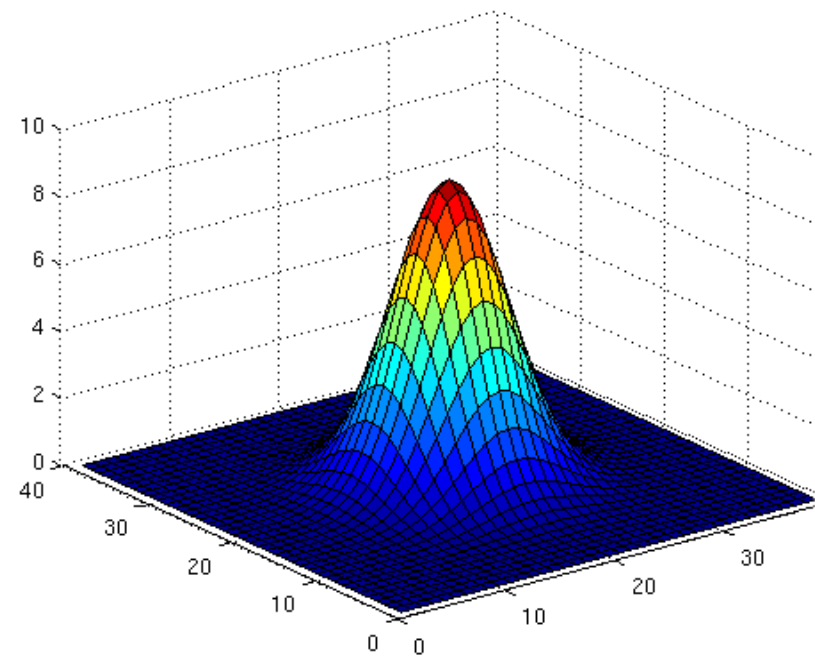
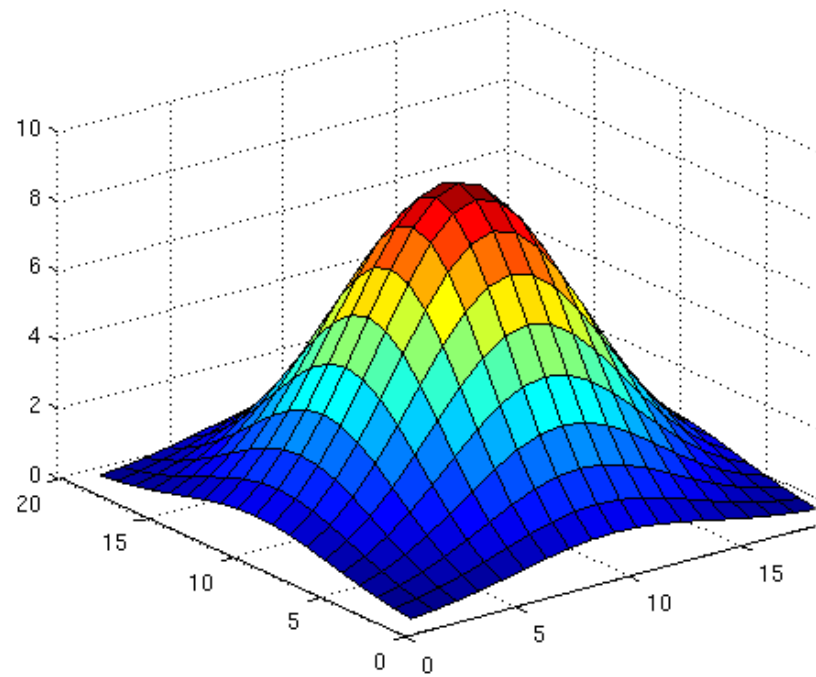
Types of Kernels

- Lets learn more about Gaussian Kernels...

$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x} - \vec{l}^i\|^2}{2\sigma^2}}$$



$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x} - \vec{l}^i\|^2}{2\sigma^2}}$$



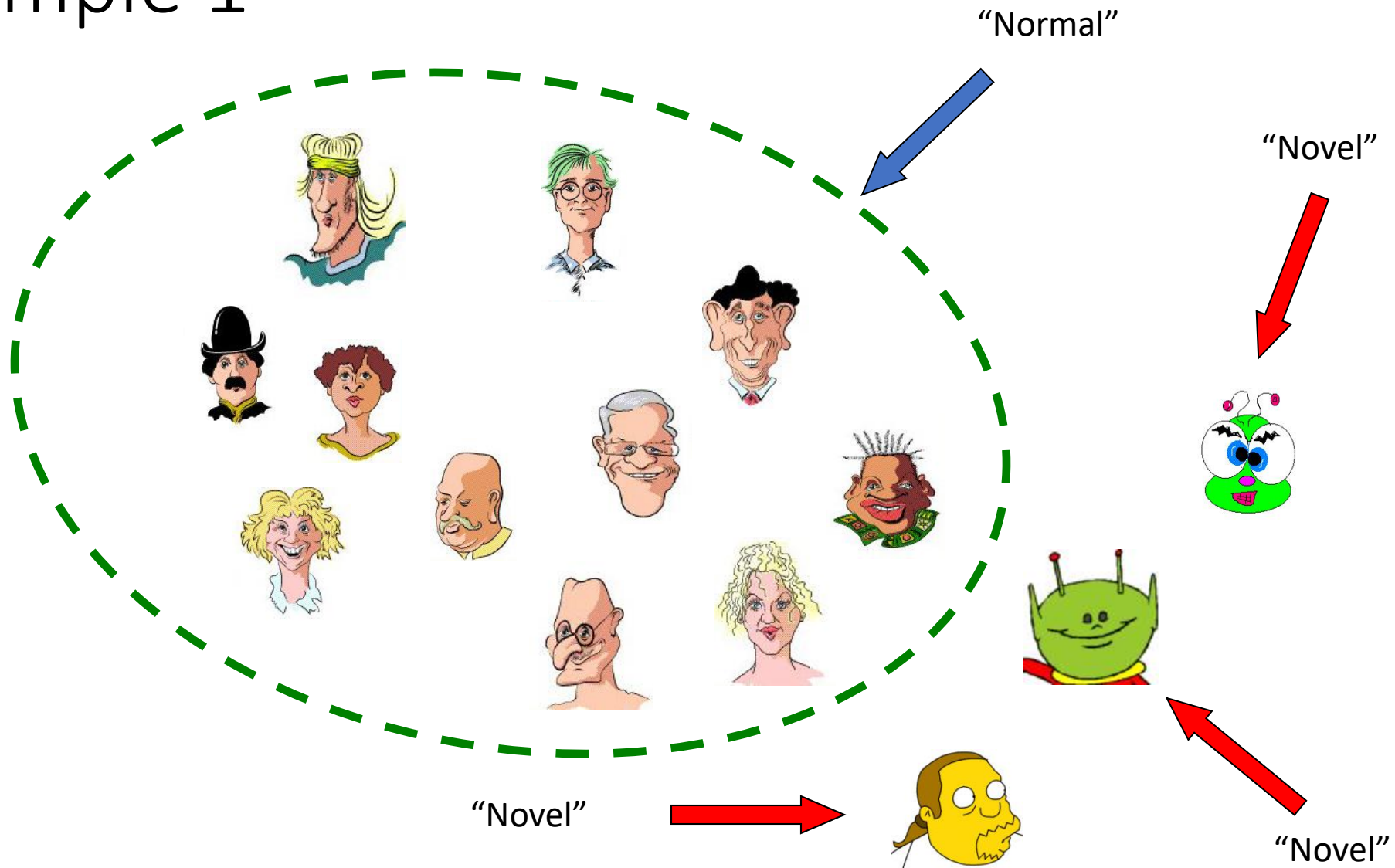
One Class Support Vector Machine (OCSVM) Algorithm

- Maps input data into a high dimensional feature space
- Iteratively finds the maximal margin in the hyperplane which best separates the training data from the origin
- An unsupervised learning problem (data unlabeled)
- About the identification of new or unknown data or signal that a machine learning system is not aware of during training

Applications:-

- Outlier/Anomaly detection
- Fraud/Spam detection
- Novelty detection

Example 1



So what's seems to be the problem?

It's a 2-Class problem
"Normal v. Not Normal"

Wrong!

The Problem is

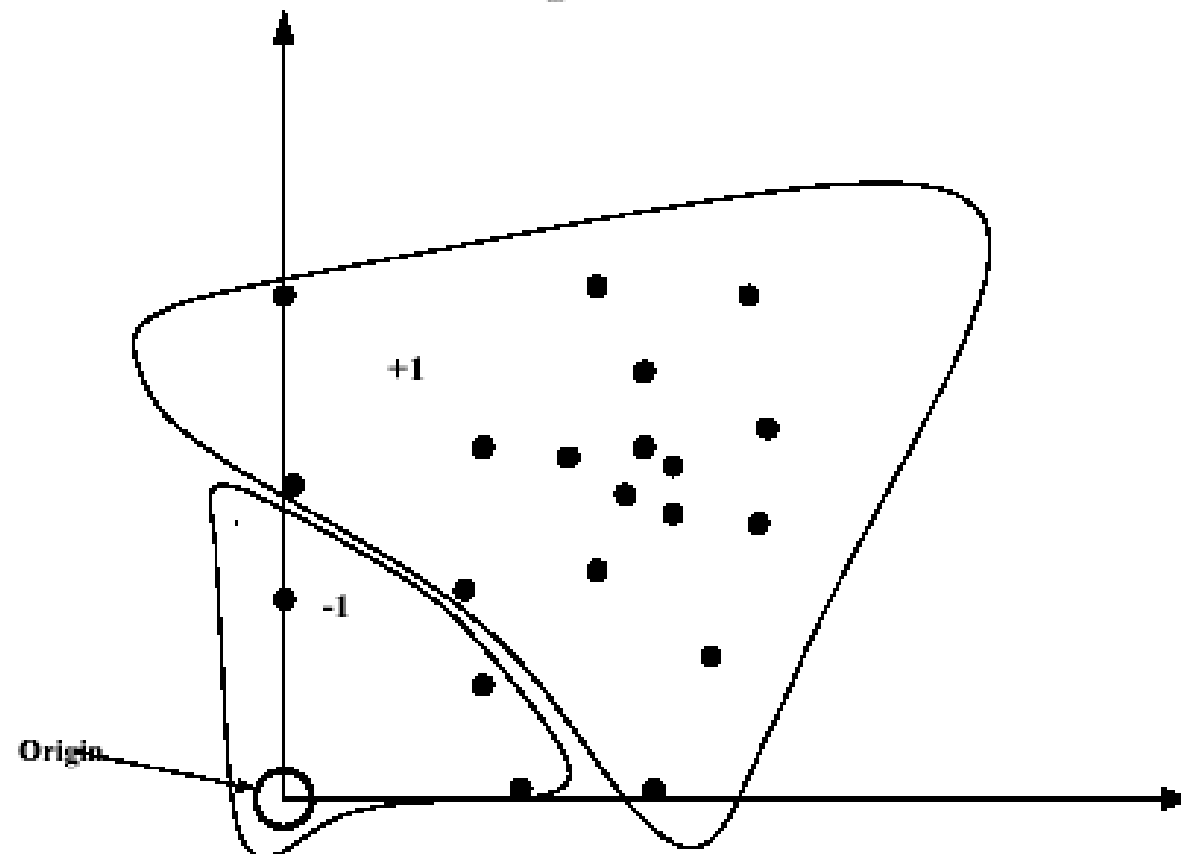
That “All positive examples are alike but each negative example is negative in its own way”.

Example 2

- Suppose we want to build a classifier that recognizes web pages about “pickup sticks”.
- How can we collect a training data?
 - We can surf the web and pretty easily assemble a sample to be our collection of **positive examples**.
- What about **negative examples** ?
 - The negative examples are... the rest of the web. That is \sim (“pickup sticks web page”)
- So the **negative examples** come from an unknown # of negative classes.

OCSVM

Figure 1: One-class SVM



One-Class SVM Classifier. The origin is the only original member of the second class.

Multi-class SVM

Combining binary classifiers

- One-vs-all
- All-vs-all
- Error correcting codes

Binary to multiclass

- Can we use a binary classifier to construct a multiclass classifier?
 - Decompose the prediction into multiple binary decisions
- How to decompose?
 - One-vs-all
 - All-vs-all
 - Error correcting codes

General setting

- Input $\mathbf{x} \in \mathbb{R}^n$
 - The inputs are represented by their feature vectors
- Output $\mathbf{y} \in \{1, 2, \dots, K\}$
 - These classes represent domain-specific labels
- **Learning:** Given a dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$
 - Need to specify a learning algorithm that takes D to construct a function that can predict \mathbf{y} given \mathbf{x}
 - Goal: find a predictor that does well on the training data and has low generalization error
- **Prediction/Inference:** Given an example \mathbf{x} and the learned function (often also called the model)
 - Using the learned function, compute the class label for \mathbf{x}

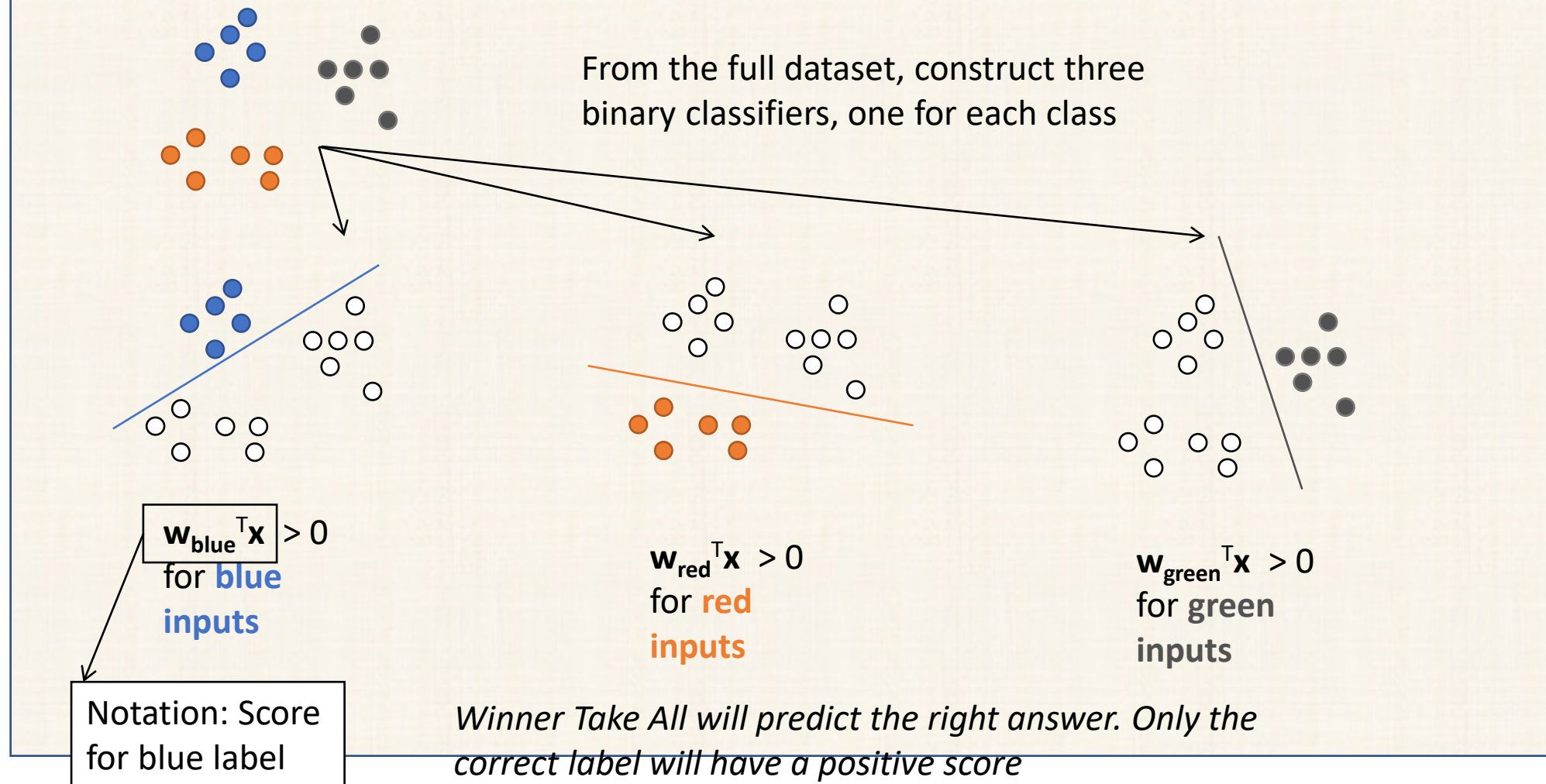
1. One-vs-all classification

- **Assumption:** Each class individually separable from ***all*** the others
- **Learning:** Given a dataset $D = \{<\mathbf{x}_i, \mathbf{y}_i>\}$,
Note: $\mathbf{x}_i \in \mathbb{R}^n$, $\mathbf{y}_i \in \{1, 2, \dots, K\}$
 - Decompose into K binary classification tasks
 - For class k , construct a binary classification task as:
 - Positive examples: Elements of D with label k
 - Negative examples: All other elements of D
 - Train K binary classifiers $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K$ using any learning algorithm we have seen
- **Prediction:** “*Winner Takes All*”

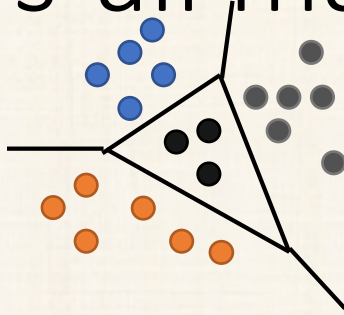
$$\operatorname{argmax}_i \mathbf{w}_i^T \mathbf{x}$$

Question: What is the dimensionality of each \mathbf{w}_i ?

Visualizing One-vs-all

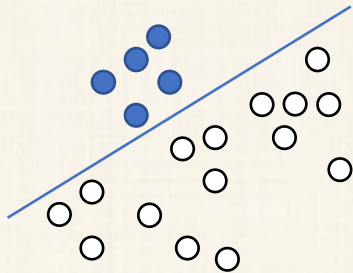


One-vs-all may not always work

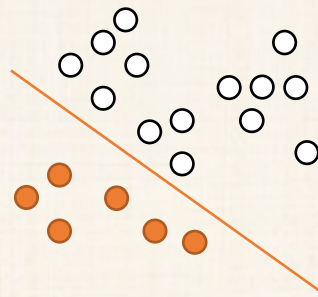


Black points are not separable with a single binary classifier

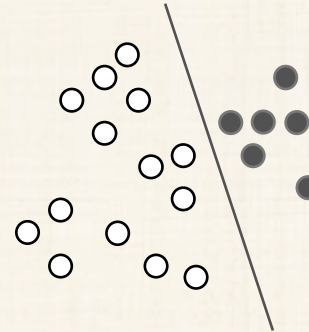
The decomposition will not work for these cases!



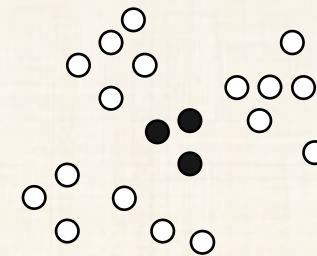
$\mathbf{w}_{\text{blue}}^T \mathbf{x} > 0$
for **blue**
inputs



$\mathbf{w}_{\text{red}}^T \mathbf{x} > 0$
for **red**
inputs



$\mathbf{w}_{\text{green}}^T \mathbf{x} > 0$
for **green**
inputs



???

One-vs-all classification: Summary

- Easy to learn
 - Use any binary classifier learning algorithm
- Problems
 - No theoretical justification
 - Calibration issues
 - We are comparing scores produced by K classifiers trained independently. No reason for the scores to be in the same numerical range!
 - Might not always work
 - Yet, works fairly well in many cases, especially if the underlying binary classifiers are tuned, regularized

Side note about Winner Take All prediction

- If the final prediction is winner take all, is a bias feature useful?
 - Recall bias feature is a constant feature for all examples
 - Winner take all:

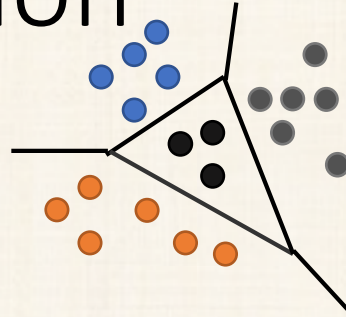
$$\operatorname{argmax}_i \mathbf{w}_i^T \mathbf{x}$$

- Answer: No
 - The bias adds a constant to all the scores
 - Will not change the prediction

2. All-vs-all classification Sometimes called one-vs-one

- **Assumption:** Every pair of classes is separable
- **Learning:** Given a dataset $D = \{\langle \mathbf{x}_i, \mathbf{y}_i \rangle\}$,
Note: $\mathbf{x}_i \in \mathcal{X}$, $\mathbf{y}_i \in \{1, 2, \dots, K\}$
 - For every pair of labels (j, k) , create a binary classifier with:
 - Positive examples: All examples with label j
 - Negative examples: All examples with label k
 - Train $\frac{K(K-1)}{2}$ classifiers in all
- **Prediction:** More complex, each label get $K-1$ votes
 - How to combine the votes? Many methods
 - Majority: Pick the label with maximum votes
 - Organize a tournament between the labels

All-vs-all classification



- Every pair of labels is linearly separable here
 - When a pair of labels is considered, all others are ignored
- Problems
 1. $O(K^2)$ weight vectors to train and store
 2. Size of training set for a pair of labels could be very small, leading to overfitting
 3. Prediction is often ad-hoc and might be unstable
Eg: What if two classes get the same number of votes? For a tournament, what is the sequence in which the labels compete?

3. Error correcting output codes (ECOC)

- Each binary classifier provides one bit of information
- With K labels, we only need $\log_2 K$ bits
 - One-vs-all uses K bits (one per classifier)
 - All-vs-all uses $O(K^2)$ bits
- Can we get by with $O(\log K)$ classifiers?
 - **Yes!** Encode each label as a binary string
 - Or alternatively, if we do train more than $O(\log K)$ classifiers, can we use the redundancy to improve classification accuracy?

Using $\log_2 K$ classifiers

#	Code		
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

8 classes, code-length = 3

- **Learning:**

- Represent each label by a bit string
- Train one binary classifier for each bit

- **Prediction:**

- Use the predictions from all the classifiers to create a $\log_2 N$ bit string that uniquely decides the output

- **What could go wrong here?**

- Even if one of the classifiers makes a mistake, final prediction is wrong!

Error correcting output code

#	Code				
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	1
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

8 classes, code-length = 5

Answer: Use redundancy

Learning:

- Assign a binary string with each label
 - Could be random
 - Length of the code word $L \geq \log_2 K$ is a parameter
- Train one binary classifier for each bit
 - Effectively, split the data into random dichotomies
 - We need only $\log_2 K$ bits
 - Additional bits act as an error correcting code
- One-vs-all is a special case.
 - How?

How to predict?

- Prediction

- Run all L binary classifiers on the example
- Gives us a predicted bit string of length L
- Output = label whose code word is “closest” to the prediction
- Closest defined using Hamming distance
 - Longer code length is better, better error-correction

- Example

- Suppose the binary classifiers here predict 11010
- The closest label to this is 6, with code word 11000

#	Code				
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	1
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

8 classes, code-length = 5

Error correcting codes: Discussion

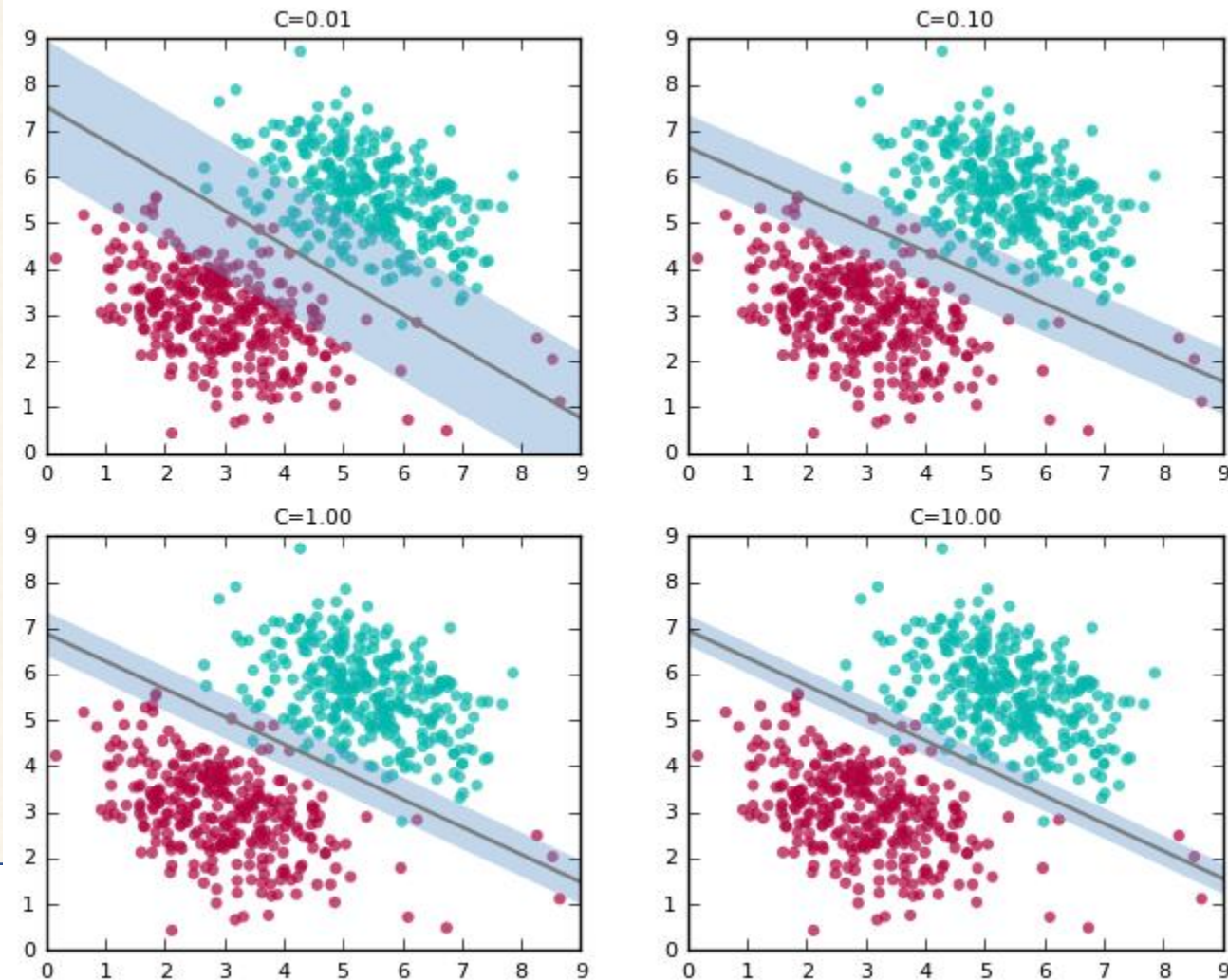
- Assumes that columns are independent
 - Otherwise, ineffective encoding
- Strong theoretical results that depend on code length
 - If minimal Hamming distance between two rows is d , then the prediction can correct up to $(d-1)/2$ errors in the binary predictions
- Code assignment could be random, or designed for the dataset/task
- One-vs-all and all-vs-all are special cases
 - All-vs-all needs a ternary code (not binary)

Decomposition methods: Summary

- **General idea**
 - Decompose the multiclass problem into many binary problems
 - We know how to train binary classifiers
 - Prediction depends on the decomposition
 - Constructs the multiclass label from the output of the binary classifiers
- Learning optimizes *local correctness*
 - Each binary classifier does not need to be globally correct
 - That is, the classifiers do not need to agree with each other
 - The learning algorithm is not even aware of the prediction procedure!
- Poor decomposition gives poor performance
 - Difficult local problems, can be “unnatural”
 - Eg. For ECOC, why should the binary problems be separable?

Questions?

SVM Parameters – Penalty Parameter (C)



- Note how the line “tilts” as we increase the value of C
- At high values, it tries to accommodate the labels of most of the red points present at the bottom right of the plots
- This is probably not what we want for test data. The first plot with $C=0.01$ seems to capture the general trend better, although it suffers from a lower accuracy on the training data compared to higher values for C .