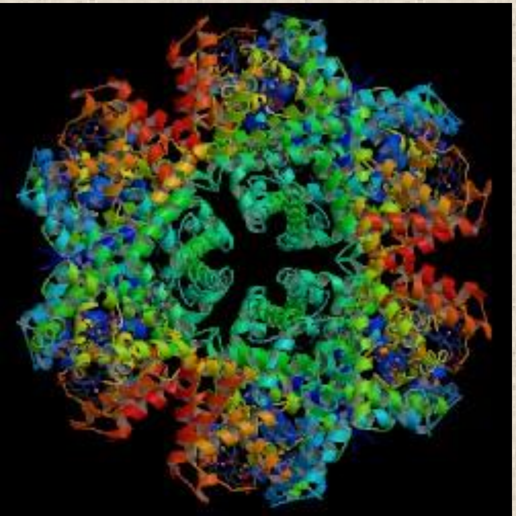
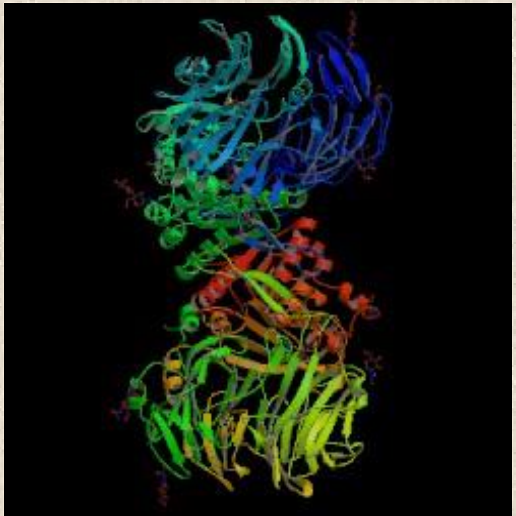
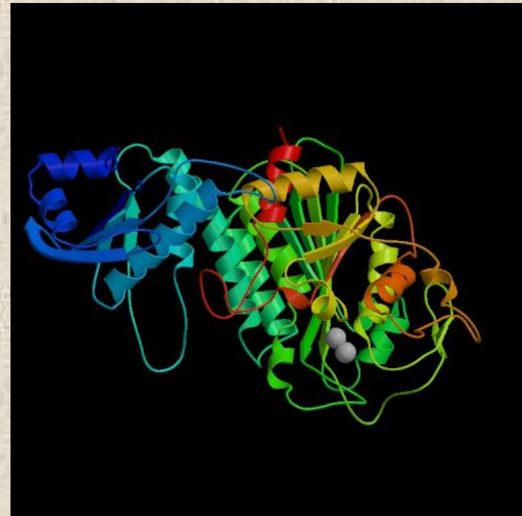
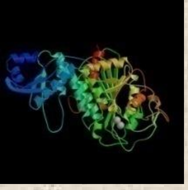
Understanding SVMs

Professor Ami M. Gates
Georgetown University



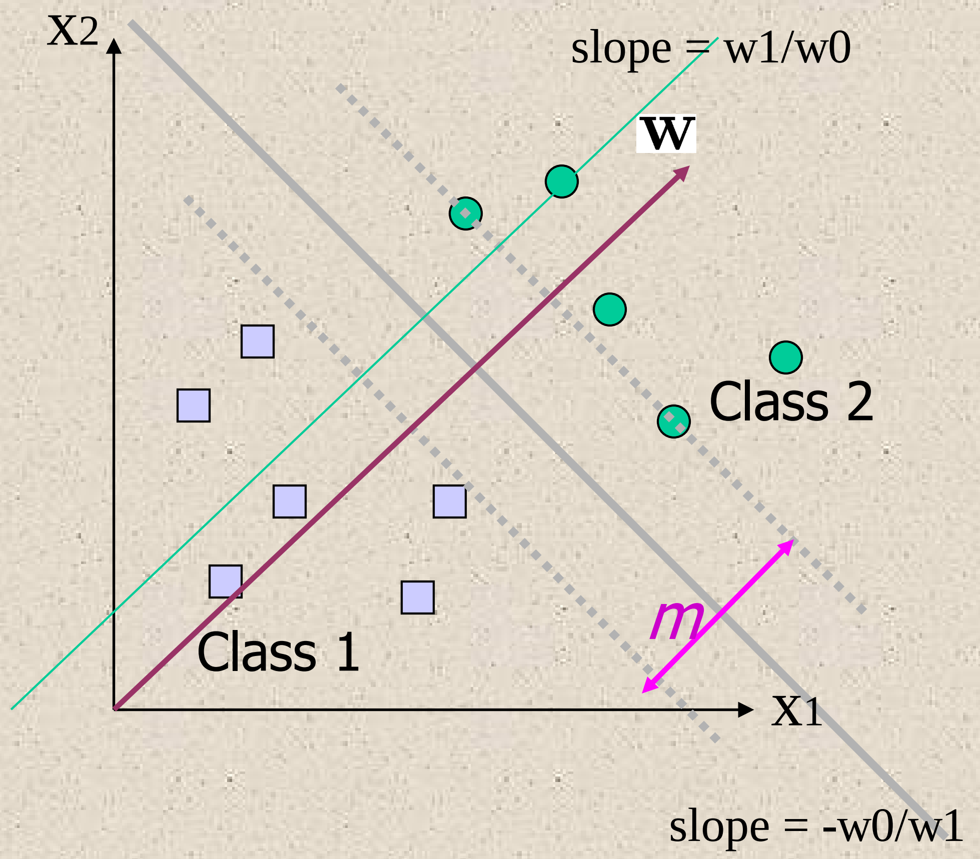






A Review of Vector Math

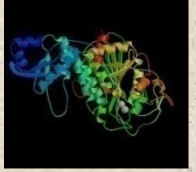
- Any line in dimension D can be represented as $\mathbf{w}^T\mathbf{x} + b = 0$.
- **w** is a vector of coefficients, **x** is the vector of variables, b is the translation (can be thought of in 2D as the y intercept).



Suppose we are in 2D. (the common Cartesian coordinate system).

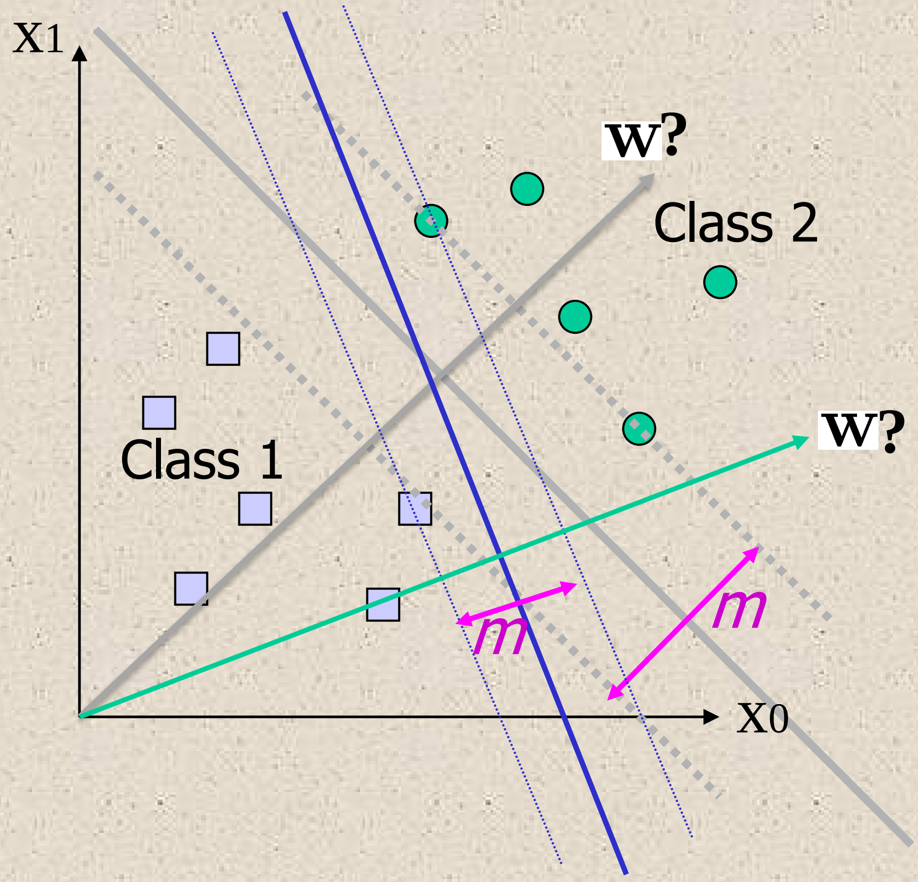
Then, a line can be written as
w0x0 + w1x1 + b = 0
x1 = - b/w1 - (w0/w1)(x0)
→ The “slope” is **–w0/w1**

Given any vector **w** (in 2D, [w0,w1]**,** from the origin, a line parallel to **w** has slope **w1/w0**
The line **perpendicular** to that vector has slope **–w0/w1**



A Review of Vector Math, cont.

- The goal of an SVM is to determine (via training) the “best” line in 2D (or hyperplane in higher D) that **separates** two classes.
- **There are an infinite number of possible lines (hyperplanes)** that may work.



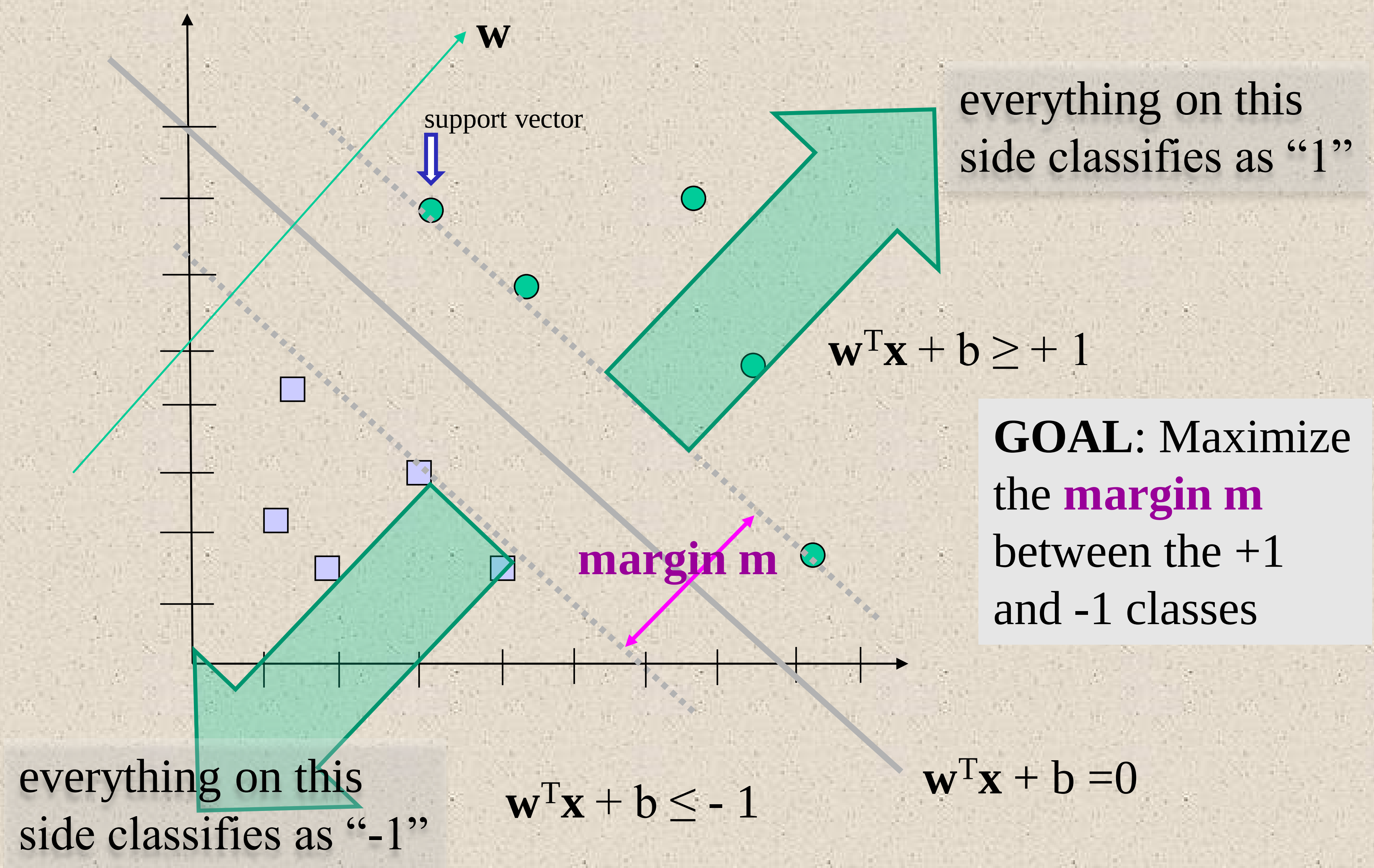
A general linear equation in two variables is:

w0x0 + w1x1 + b = 0

→ $w^T x + b = 0$

An SVM algorithm must determine the <u>weight vector w</u> that will result in a line (hyperplane) that <u>best separates the two classes</u>.

Setting Up the SVM Problem



GOAL: Maximize the **margin m** between the +1 and -1 classes

Important Concepts

- The **margin m** can be created such that

$w0x0 + w1x1 + b \geq 1$

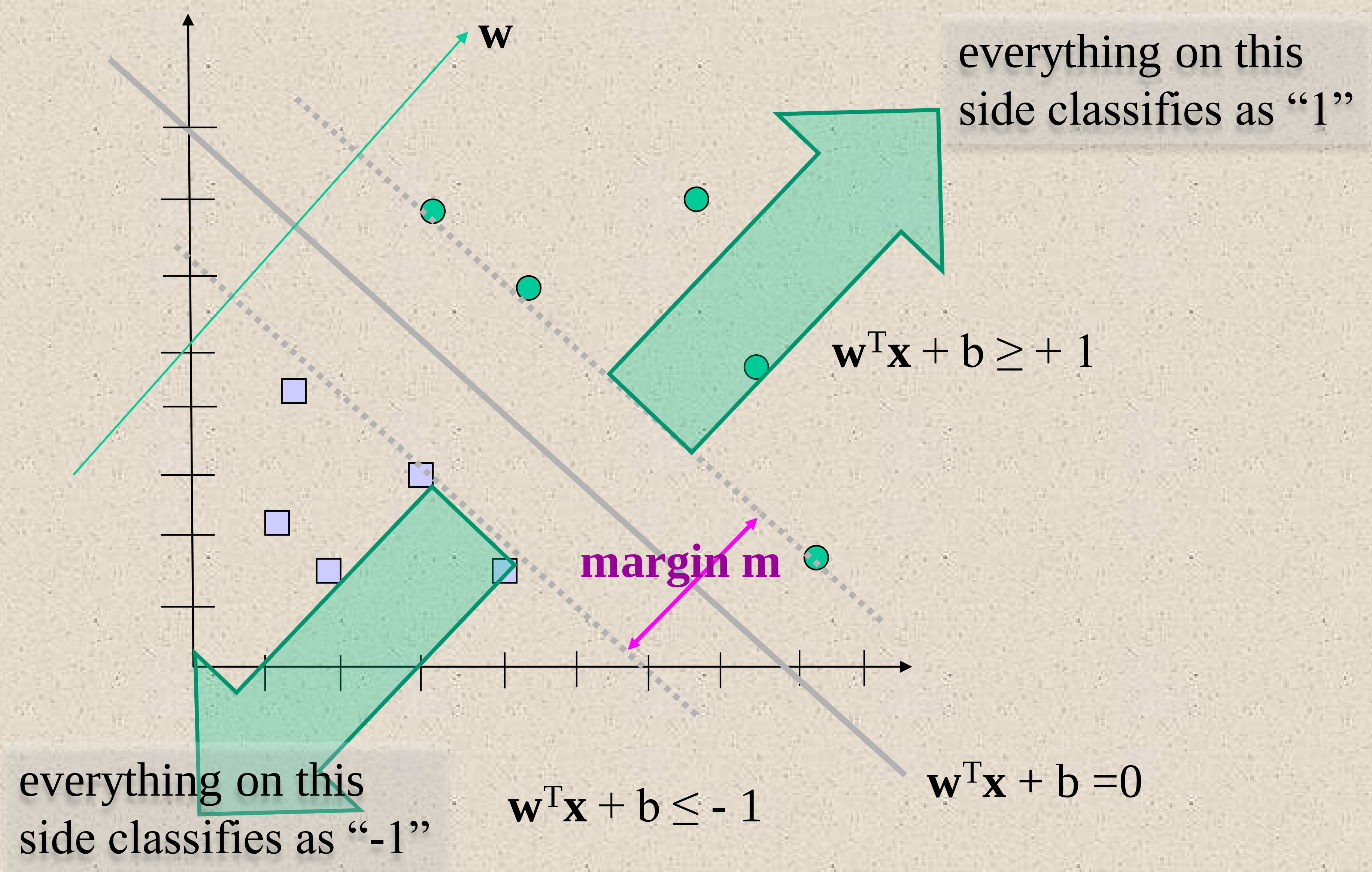
$\rightarrow \mathbf{w}^T\mathbf{x} + b \geq 1$

$w0x0 + w1x1 + b \leq -1$

$\rightarrow \mathbf{w}^T\mathbf{x} + b \leq -1$

- Remember that **w** will have to be determined (**using the training data**) so that the classes are “best” separated.
- The value of “b” is a constant that affects the location (but not slope) of the line (in 2D).
- These concepts are the same in higher D.

The **margin m** between the two classes should be maximized.



Key Math Concepts

1) Any line in 2D or hyperplane in higher D can be represented as:

$w_0x_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n + b = 0$

→ same as: $\mathbf{w}^T\mathbf{x} + b = 0$

2) This linear equation can be represented with vector **w**, with vector **x**, and with b.

If vector **w** is plotted, it will be perpendicular to the linear equation above.

The distance between any hyperplane $\mathbf{w}^T\mathbf{x} = \mathbf{k}$ and the origin is $k/||\mathbf{w}||$

If vector **w** is altered, the "slope" (in 2D) or "rotation" (in higher D) is affected. If "b" is altered the "y intercept (in 2D) or the "translation" (in high D) is affected.

3) We can use $\mathbf{w}^T\mathbf{x} + b = 0$ to try different lines. **To do this, we need w and b and a measure of "best".**

The SVM Margin m

Assume that in some dimension that we have two classes that can be linearly separated.

- Then, there must be a line (hyperplane) that separates the classes.
- This line is defined by $\mathbf{w}^T\mathbf{x} + b = 0$.
- This line can be contained within two parallel lines that create a **"margin"** between the two classes.

The two parallel lines can initially be defined as:

$\mathbf{w}^T\mathbf{x} + b = 1$ and $\mathbf{w}^T\mathbf{x} + b = -1$

(Note that b is a constant that can be altered as needed)

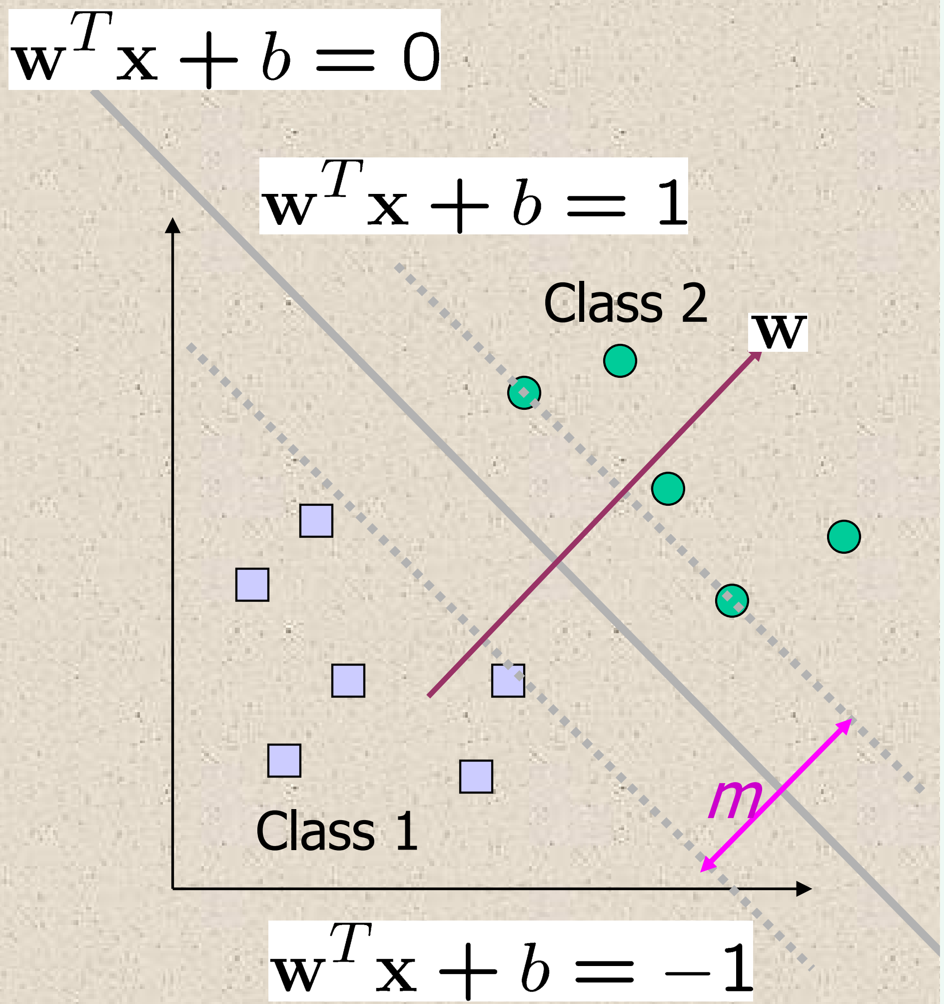
margin m:

Distance between two parallel lines (hyperplanes)

$\mathbf{w}^T\mathbf{x} + \mathbf{b} = 1$ and $\mathbf{w}^T\mathbf{x} + \mathbf{b} = -1$ is…

$|b - 1| - |b + 1| \,/\, ||w|| = \mathbf{2/||w||}$

Maximizing margin m will create the "best" separation between the Classes.



margin m is the **distance** between the two parallel boundary lines (or two parallel hyperplanes)

Distance between a line
w^{T}x + b= 0 (hyperplane)
and the origin is
b/||**w**||

Margin m:
Distance between two parallel lines (hyperplanes)
w^{T}x + b= 1 and w^{T}x + b= -1
|b -1| - |b + 1| / ||w|| = **2/||w||**

Maximizing margin m

To **maximize** the margin m = 2/||w||

→We need to **minimize** ||w|| which is same as min ½||w||^2

→which makes the math easier.

Note that ||w|| is sqrt(w0^2 + w1^2).

We can say that margin m =

2/ sqrt(w0^2 + w1^2)

as we do in scikit-learn example

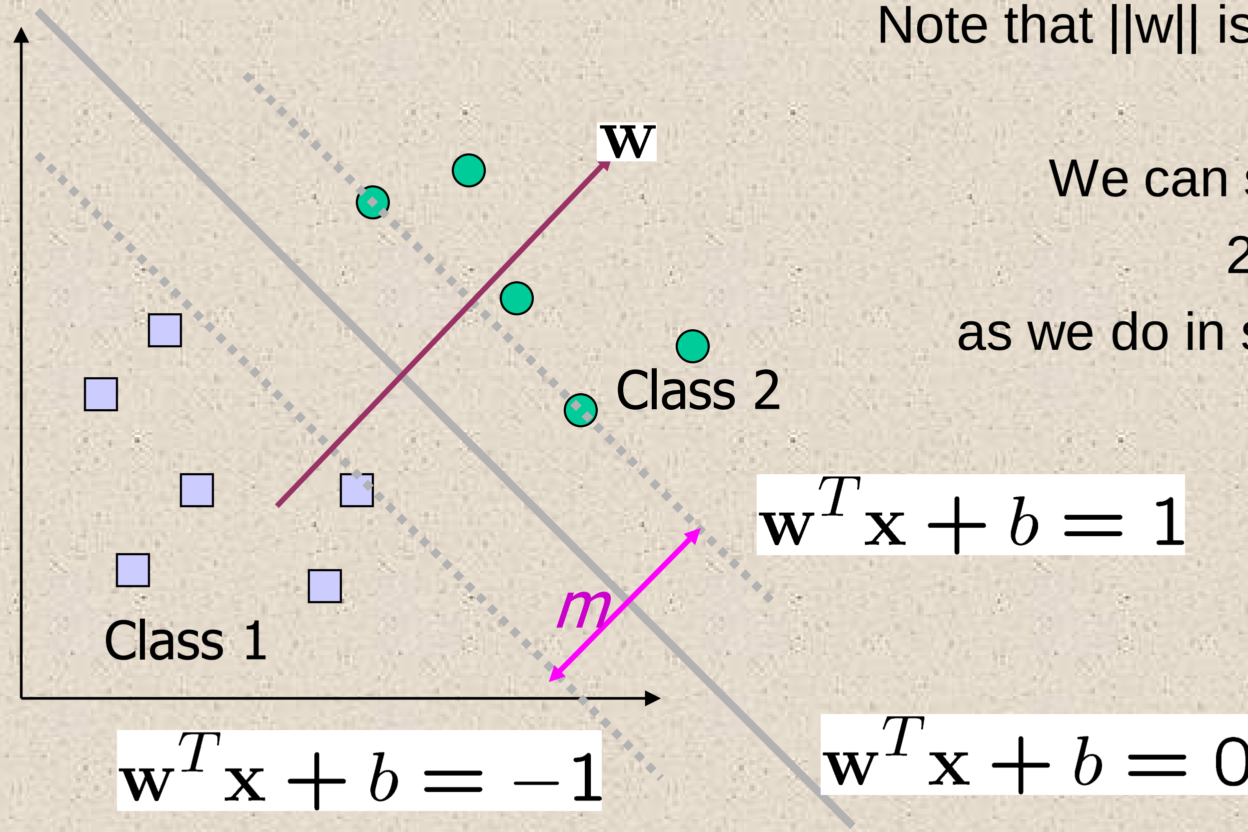


Image: Martin Law Michigan State university

The Optimization Problem

1) Find the **min** of $\|\mathbf{w}\|$ and **b**

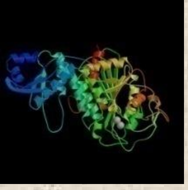
This is the same min $\frac{1}{2}\|\mathbf{w}\|^2$ and b

2) Such that this **constraint** holds:

$\mathbf{y}_i\,(\mathbf{w}^T\mathbf{x}_i + \mathbf{b}) \geq \mathbf{1}$ for all input vectors $\mathbf{x}_i$

Recall that $\mathbf{y}_i$ will be +1 or -1 (depending on the class)

This is a **quadratic** optimization problem.



A Soft Margin Hyperplane SVM

- If we minimize $\sum_i \xi_i$,

ξ_i can be computed by

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

ξ_i are "**slack variables**" in optimization

– Note that ξ_i=0 if there is no error for $\mathbf{x}_i$

ξ_i is an upper bound of the number of errors

- We want to **minimize: $\frac{1}{2}||\mathbf{w}||^2 + C \sum_i \xi_i$**
  - C : tradeoff parameter between error and margin
- The optimization problem becomes

$$\frac{1}{2}||\mathbf{w}||^2 + C \sum_i \xi_i$$

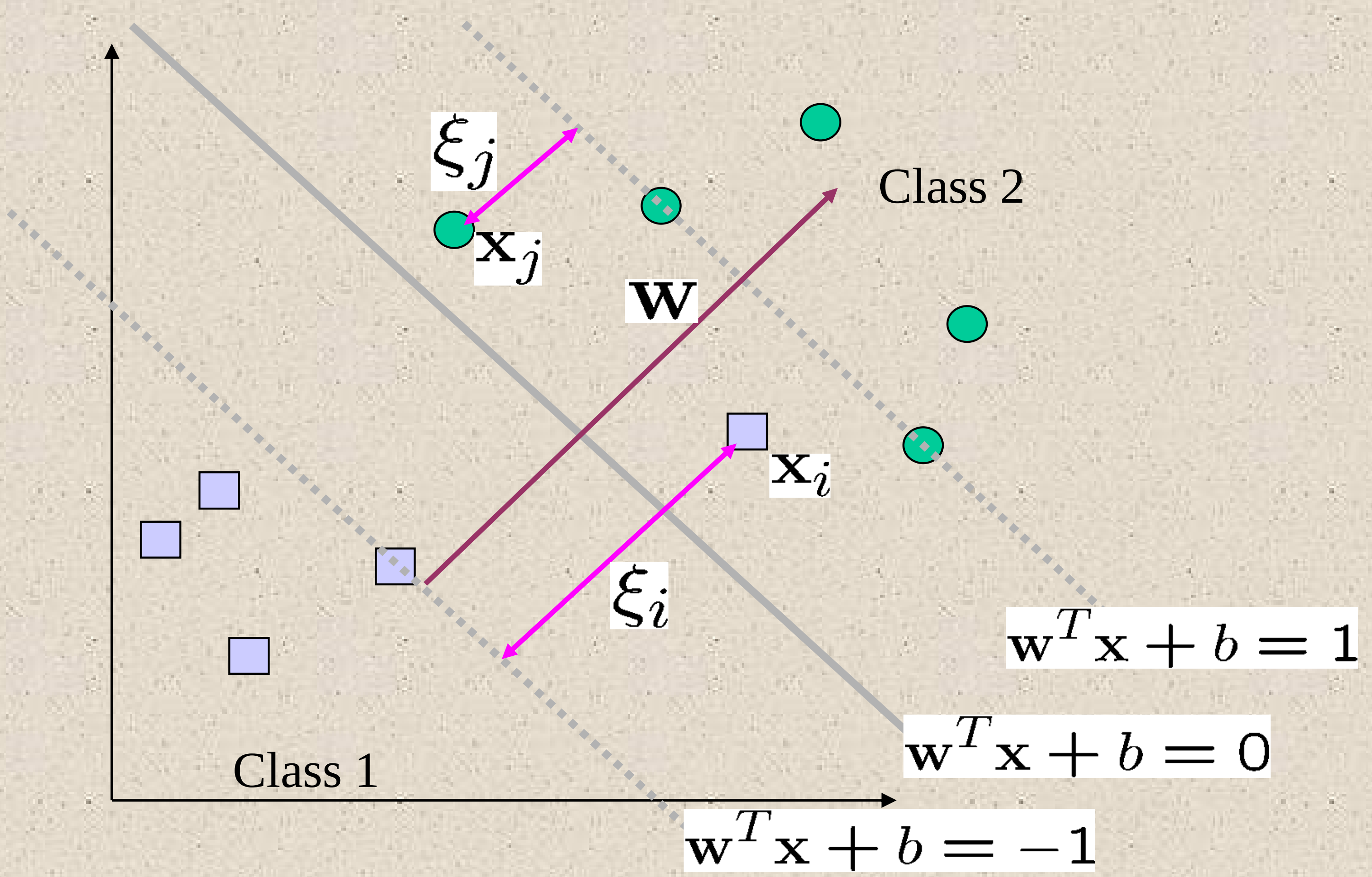
$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Image: Martin Law Michigan State university

Soft Margin Example

- Allow "error" ξ_i in classification

ξ_i approximates the number of misclassified samples



Constrained Optimization Quick Review

Suppose you want to minimize some function **f(x)** subject to the constraint of **g(x).**

Then, for any x to be a solution, the following must be true.

Note that the "α" is a Lagrange multiplier

$$\begin{cases} \frac{\partial}{\partial \mathbf{x}}\Big(f(\mathbf{x}) + \alpha g(\mathbf{x})\Big)\Big|_{\mathbf{x}=\mathbf{x}_0} = 0 \\ g(\mathbf{x}) = 0 \end{cases}$$

Using Lagrange in SVM Solving

Goal: Minimize $\frac{1}{2} ||w||^2$

subject to constraint:

$1 - y_i(w^T x_i + b) \leq 0$, $\quad\quad i = 1, 2..,n$

The **Lagrangian** L is

$L = \frac{1}{2} w^T w + \sum_{(i=1 \text{ to } n)} \alpha_i (1 - y_i(w^T x_i + b))$

Them, setting the gradient of L w.r.t. **w** and b to 0, gives:

$\mathbf{w} + \sum_{(i=1 \text{ to } n)} \alpha_i(- y_i) x_i = 0 \rightarrow$

$\mathbf{w} = \sum_{(i=1 \text{ to } n)} \alpha_i y_i x_i$ and $\sum_{(i=1 \text{ to } n)} \alpha_i y_i = 0$

Finally, if we substitute **w** into L we get (after a bit of math): $-1/2 \sum_{(i=1 \text{ to } n)} \sum_{(j=1 \text{ to } n)} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{(i=1 \text{ to } n)} \alpha_i$

The Dual Problem in SVM

The new objective function is called the **dual problem** in SVM math.

$W(\alpha) = -1/2 \sum_{(i=1 \text{ to } n)} \sum_{(j=1 \text{ to } n)} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{(i=1 \text{ to } n)} \alpha_i$

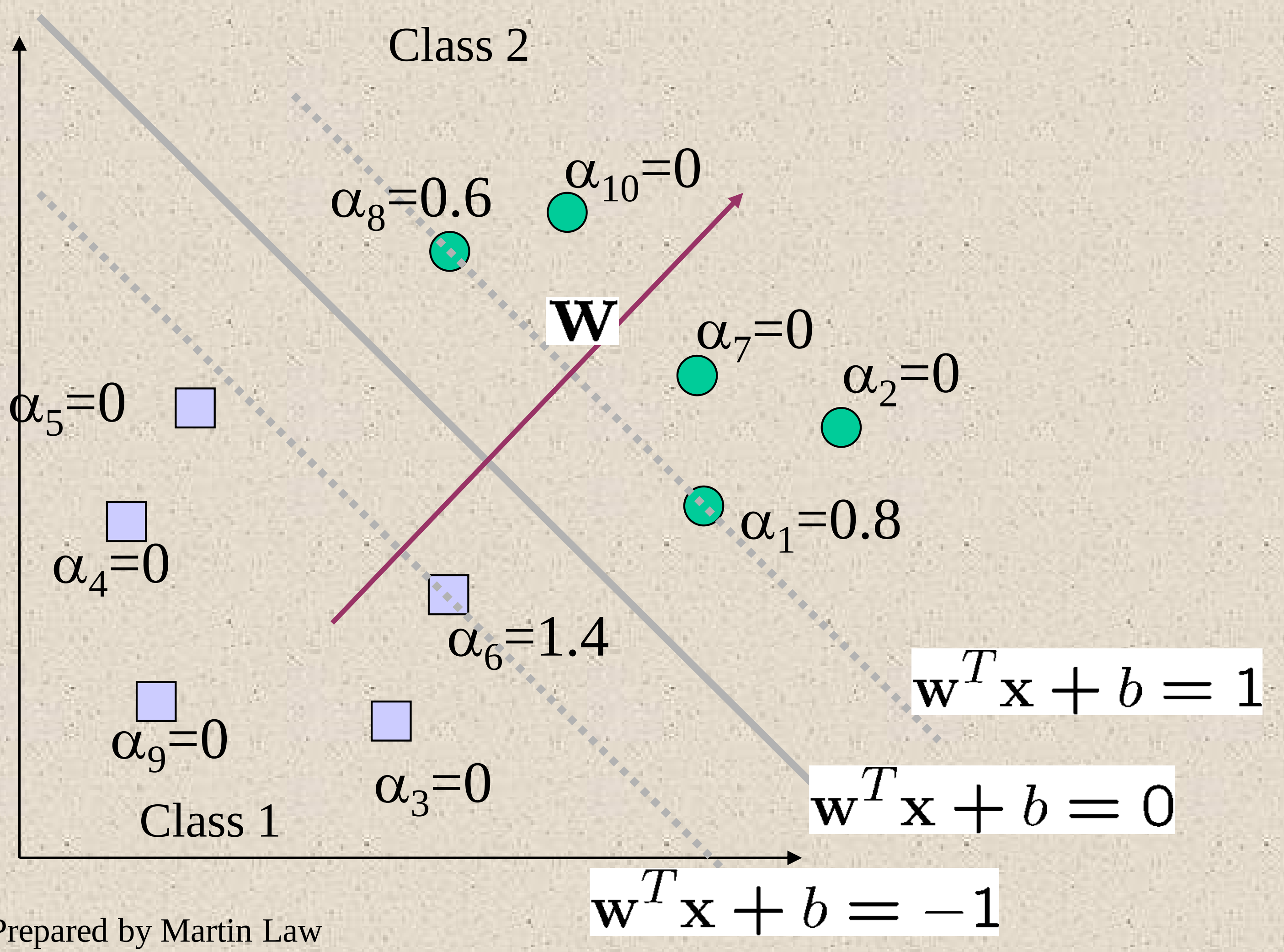
It is a function of alpha.

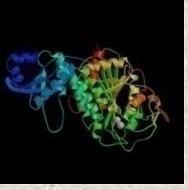
The goal is now to maximize $W(\alpha)$ subject to alphas all being positive and $\sum_{(i=1 \text{ to } n)} \alpha_i y_i = 0$

This is a **quadratic programming** problem with

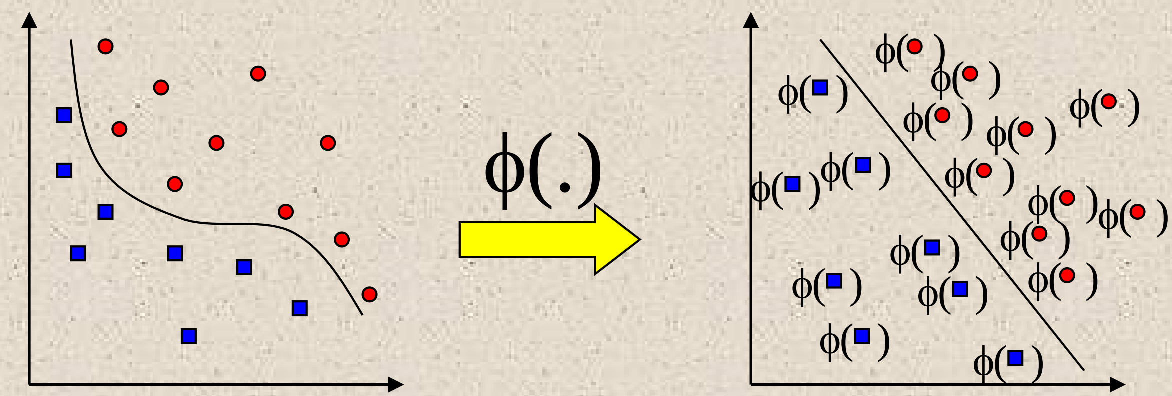
$\mathbf{w} = \sum_{(i=1 \text{ to } n)} \alpha_i y_i x_i$

A Visual Interpretation of α





SVM kernel Transformation: $\varphi(\mathbf{x})$



Input space

Feature space

Recall that the SVM optimization problem solved is:

$W(\alpha) = -1/2 \sum_{(i=1 \text{ to } n)} \sum_{(j=1 \text{ to } n)} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{(i=1 \text{ to } n)} \alpha_i$

Here, only the inner product of the x values is needed.

Therefore, if a kernel transformation is used, the updated optimization problem becomes

$W(\alpha) = -1/2 \sum_{(i=1 \text{ to } n)} \sum_{(j=1 \text{ to } n)} \alpha_i \alpha_j y_i y_j \, \varphi(x_i)^T \varphi(x_j) + \sum_{(i=1 \text{ to } n)} \alpha_i$

where kernel $k(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$

We only need the inner product of the feature space – not an explicit mapping.

Common Kernels

Polynomial with degree d

K(x, y) = (xTy + 1)d

Radial

$K(x,y) = \exp(-\|x - y\|^2 / 2\sigma^2))$

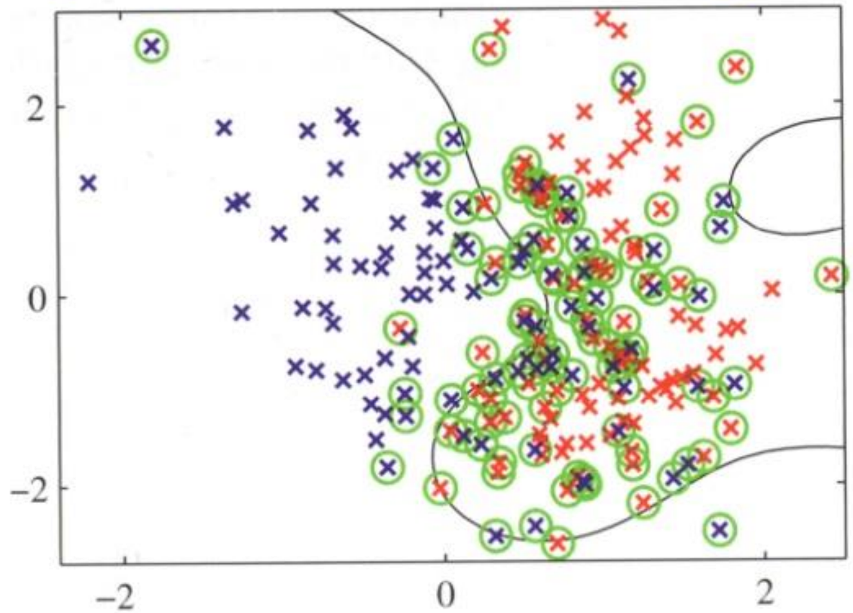
Sigmoid:

$K(x,y) = \tanh(cx^Ty + d)$

Gaussian:

$K(x, y) = \exp(-a\|x - y\|^2)$, $a>0$

SVM Soft Margin Decision Surface using Gaussian Kernel



[from Bishop, figure 7.4]

$$\hat{f}(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) + b$$

Circled points are the *support vectors*: training examples with non-zero α_l

Points plotted in original 2-D space.

Contour lines show constant $\hat{f}(\mathbf{x})$

$$\hat{f}(\mathbf{x}) = b + \sum_{l=1}^{M} \alpha_l \; y_l \; \kappa(\mathbf{x}, \mathbf{x}_l) = b + \sum_{l=1}^{M} \alpha_l \; y_l \exp(-\|\mathbf{x} - \mathbf{x}_l\|^2 / 2\sigma^2)$$

Compliments of: https://www.cs.cmu.edu/~tom/10701_sp11/slides/Kernels_SVM2_04_12_2011-ann.pdf

Example non-linear

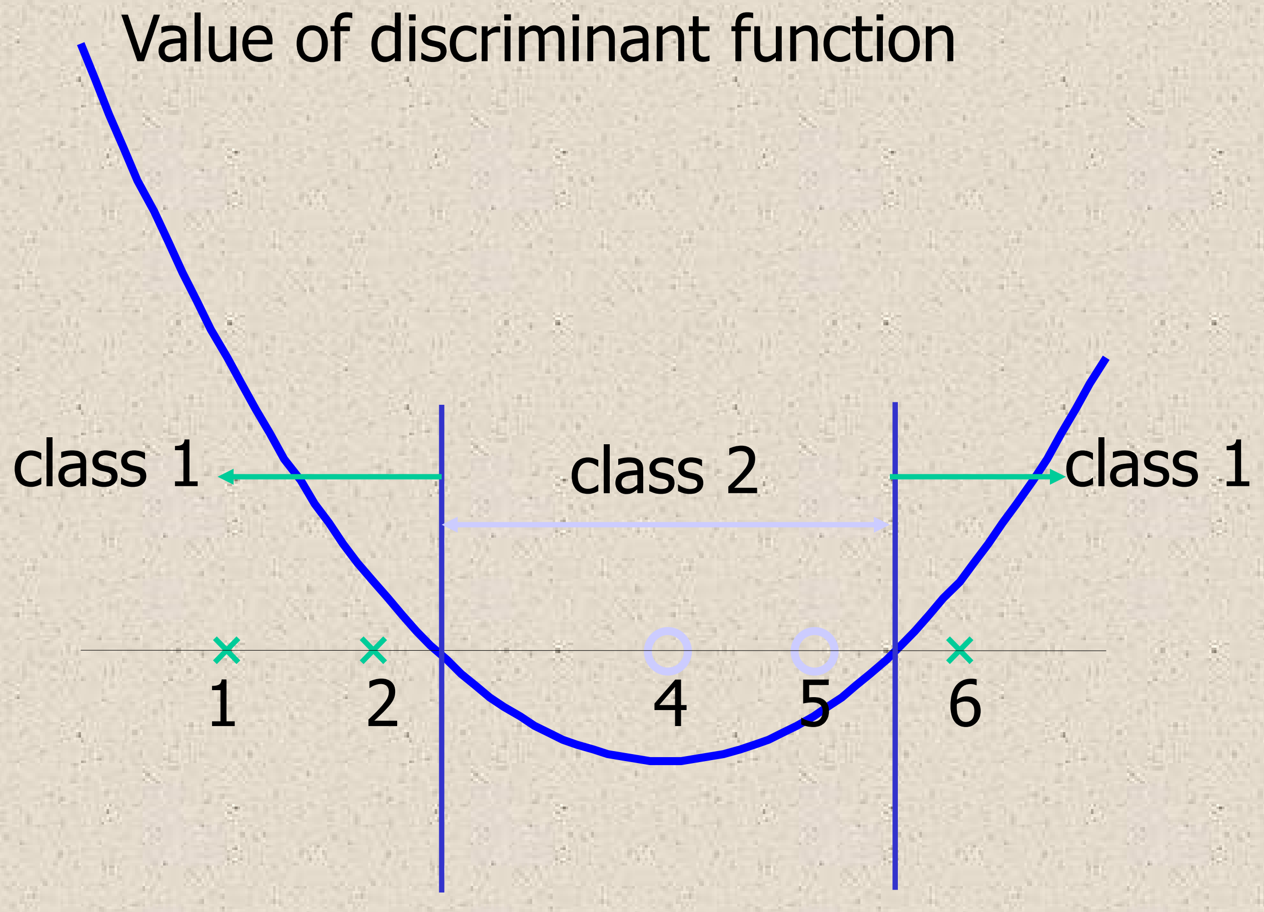


Image: Martin Law Michigan State university

Using Python 3 scikit-learn: SVM

http://scikit-learn.org/stable/modules/svm.html

SVM:

1) supervised learning method
2) for classification, regression, outlier detection

Advantages:

1) Effective in high D space
2) Effective when dimension > # samples
3) Uses "subset" of training data – just the "support vectors"
4) Allows or the use of kernel functions

Disadvantages:

1) If features >> samples → poor performance
2) No prob estimates – these must be "done by hand" using at least 5-fold cross validation

SVM Classification with scikit-learn: Example 1

```
from sklearn.svm import SVC
import numpy as np
#Train
#SVC takes in two arrays [n_samples, n_features]
#This holds the training samples
X = np.array([[-1, -1], [-2, -1], [-3, -4], [3,3], [1, 1], [2, 1]])
#The classes of the training data
y = np.array([1, 1, 1, 2, 2, 2])
#TRAIN
#Define the model with SVC
# Fit SVM with training data
clf = SVC(C=1, kernel="rbf", verbose=False)
clf.fit(X, y)
#PREDICT - perform classification
print(clf.predict([[-.8, -1]]))
print(clf.predict([[.8, 1]]))
print(clf.predict([[0, 0]]))
```

```
[1]
[2]
[1]
```

More scikit-learn Examples

```
from sklearn.svm import SVC
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt

iris = datasets.load_iris()
#Place first two columns into X
X=np.array(iris.data[:,:2])
#Place the targets/classes into y
y = np.array(iris.target)
#Set up the SVM - linear kernel
clf = SVC(C=1, kernel="linear")
clf.fit(X, y)
#Margin is 2/ ||w||
margin = 2 /
np.sqrt(np.sum(clf.coef_ ** 2))

#Predict a new data points
print(clf.predict([[6, 3]]))
```

```
# get the separating hyperplane
#The weights vector w
w = clf.coef_[0]
#The slope of the SVM sep line
a = -w[0] / w[1]
#Create a variable xx that are values
between 4 and 8
xx = np.linspace(4, 8)
## This is the y values for the main sep
line
yy = a * xx - (clf.intercept_[0]) / w[1]
# adding or subtracting ½ of the margin
yy_down = yy + .5*margin
yy_up = yy - .5*margin
```

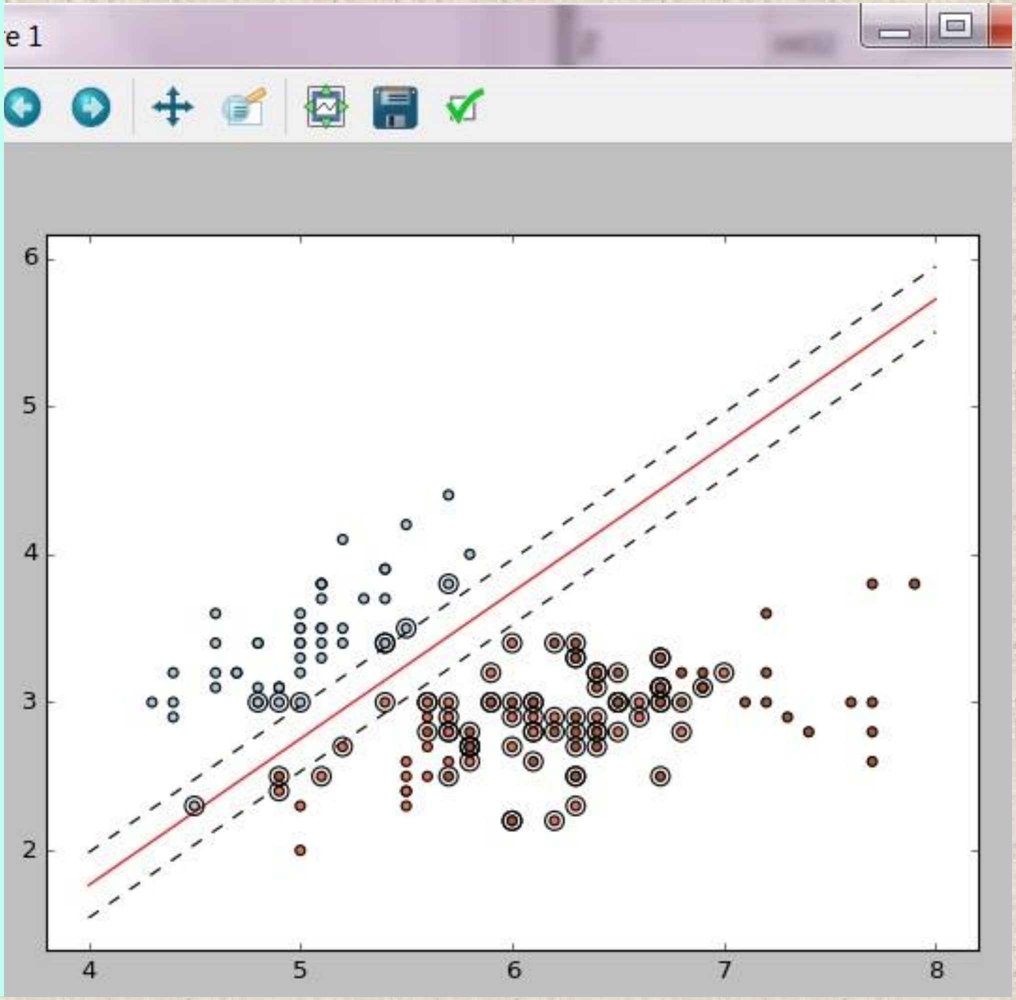
continued…

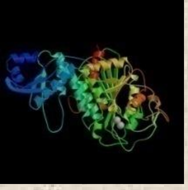
```
# plot the line, the points, and the
nearest vectors to the plane
plt.clf()
plt.plot(xx, yy, 'r-')
plt.plot(xx, yy_down, 'k--')
plt.plot(xx, yy_up, 'k--')

plt.scatter(clf.support_vectors_[:,
0], clf.support_vectors_[:, 1], s=80,
facecolors='none', zorder=5)

#cmap is the color map
plt.scatter(X[:, 0], X[:, 1], c=y,
zorder=5, cmap=plt.cm.Paired)

plt.axis('tight')
```





The SVM Math - FYI

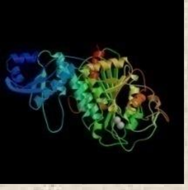
The **perpendicular distance** from a point to hyperplane $\mathbf{y(x)=0}$, where $\mathbf{y(x) = w^T\varphi(x)+b}$, is defined as $\mathbf{t_n(w^T \varphi(x_n)+b) / ||w||}$**,** since only correctly predicted points are of interest.

The **margin** is given by the perpendicular distance to the closest point x_n, and **w** and **b** can be **optimized to maximize** the distance.

The max margin solution can be found by solving:
$$\mathbf{arg\ max\ w, b\ \{ 1/||w||\ \ min\ n\ [t_n(w^T\varphi(x_n)+b)]\}}$$

Note: rescaling w and b by the same constant does not affect the distance between x_n and the decision boundary.
Therefore, set $\mathbf{t_n(w^T\varphi(x_n)+b) = 1}$ **for the closest point and** $\mathbf{t_n(w^T\varphi(x_n)+b) \geq 1}$ **for all other points.**

- Extra slides…



The SVM Math - FYI

$\mathbf{t_n(w^T\varphi(x_n)+b) \geq 1}$ is the canonical representation of the hyperplane.

To maximize the margin, **minimize** $\mathbf{w^Tw}$

Therefore, the **Quadratic Programming** problem can be stated as:

$$\textbf{arg min w, b} \;:\; \mathbf{w^Tw}$$

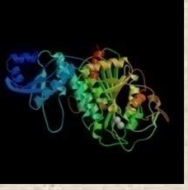
$$\text{such that } \mathbf{t_n(w^T\varphi(x_n)+b) \geq 1}$$

<u>Lagrangian to solve:</u>

$$\mathbf{L(w,b,a)=w^Tw - \Sigma_{n=1 \text{ to } N}\; a_n\{t_n(w^T\varphi(x_n)+b) - 1\}}$$

Take derivatives of L(w,b,a) w.r.t. w and b to get:

$$\mathbf{w = \Sigma_{n=1 \text{ to } N}\; a_nt_n\varphi(x_n) \quad \text{and} \quad \Sigma_{n=1 \text{ to } N}\; a_nt_n = 0}$$



The SVM: The kernel

Substituting:

$\mathbf{w = \Sigma_{n=1\ to\ N}\ \ a_n t_n \varphi(x_n)}$ **and** $\mathbf{\Sigma_{n=1\ to\ N}\ \ a_n t_n = 0}$

into

$\mathbf{L(w,b,a)=w^T w\ \ –\ \ \Sigma_{n=1\ to\ N}\ \ a_n\{t_n(w^T\varphi(x_n)+b)\ \ - 1\}}$

The Dual Representation:

$\mathbf{L'(a) = \Sigma_{n=1\ to\ N}\ \ a_n – ½\ \Sigma_{n=1\ to\ N}\ \Sigma_{m=1\ to\ N}\ a_n a_m t_n t_m \varphi(x_n)^T\ \varphi(x_m)}$

Kernel:

$\mathbf{K(x_{n,}x_m) = \varphi(x_n)^T\ \varphi(x_m)}$

NOTE: This portion of the math allows for the use of a kernel to represent the inner product of data points.

Kernels

Because the solution to the SVM optimization problem (the Dual Form) involves only **inner products**, the data values can be "transformed" to a different dimension using a kernel.

- **The Dual Representation:**

$$L'(a) = \Sigma_{n=1 \text{ to } N}\ a_n - \tfrac{1}{2}\, \Sigma_{n=1 \text{ to } N}\, \Sigma_{m=1 \text{ to } N}\, a_n a_m t_n t_m \varphi(x_n)^T \varphi(x_m)$$

Kernel: $K(x_n, x_m) = \varphi(x_n)^T \varphi(x_m)$

This can be thought of as a virtual transformation as the points themselves do not have to be projected.

This is the Gaussian Kernel for example.

$$K(x_i, x_j) = exp(-\gamma \parallel x_i - x_j \parallel^2), \gamma > 0,$$