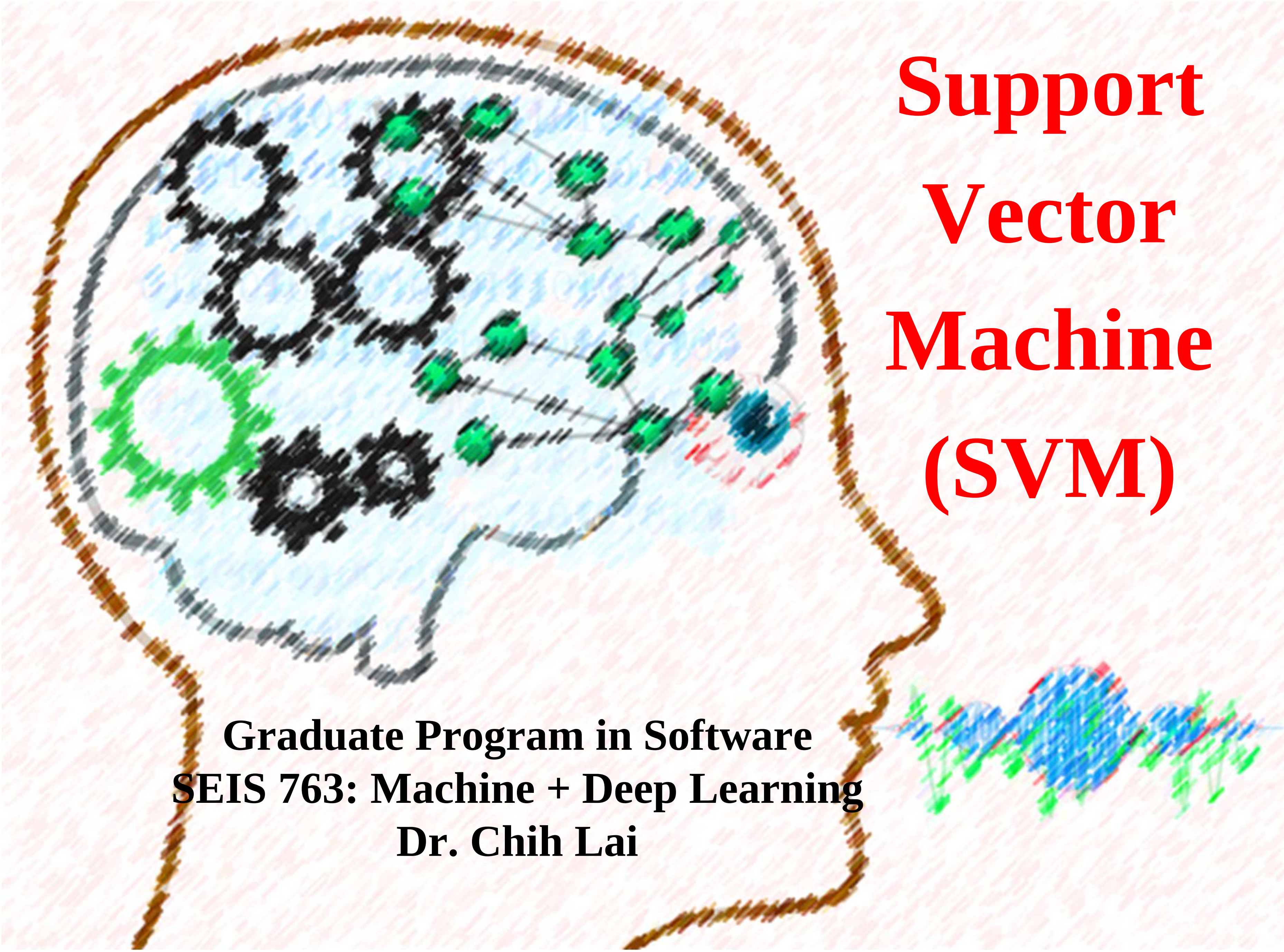


Support Vector Machine (SVM)

Graduate Program in Software
SEIS 763: Machine + Deep Learning
Dr. Chih Lai

Matlab SVM-Related Functions and References

- Matlab SVM: **http://www.mathworks.com/help/stats/support-vector-machines-svm.html#responsive_offcanvas**

- Matlab SVM class
  - http://www.mathworks.com/help/stats/support-vector-machine-classification.html
  - http://www.mathworks.com/help/stats/classificationsvm-class.html

- Matlab Functions:
  - **fitcsvm**(X, Y, 'KernelFunction', 'rbf', 'Crossval', 'on', 'Standardize', true);
    - http://www.mathworks.com/help/stats/fitcsvm.html
    - http://www.mathworks.com/help/stats/classificationsvm-class.html
    - **Mutip-class SVM ➔ http://www.mathworks.com/help/stats/fitcecoc.html**
  - [**label**, **score**] = **predict**(SVMModel, newX);
    - http://www.mathworks.com/help/stats/compactclassificationsvm.predict.html
  - crossval(), kFoldLoss(), kfoldPredict()

Python sklearn SVM References

- Support Vector Machine
  - **http://scikit-learn.org/stable/modules/svm.html**

- **sklearn.svm.SVC**
  - **http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html**
  - sklearn.svm.LinearSVC **http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html**

- Plot different SVM classifiers in the iris dataset
  - **http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html**

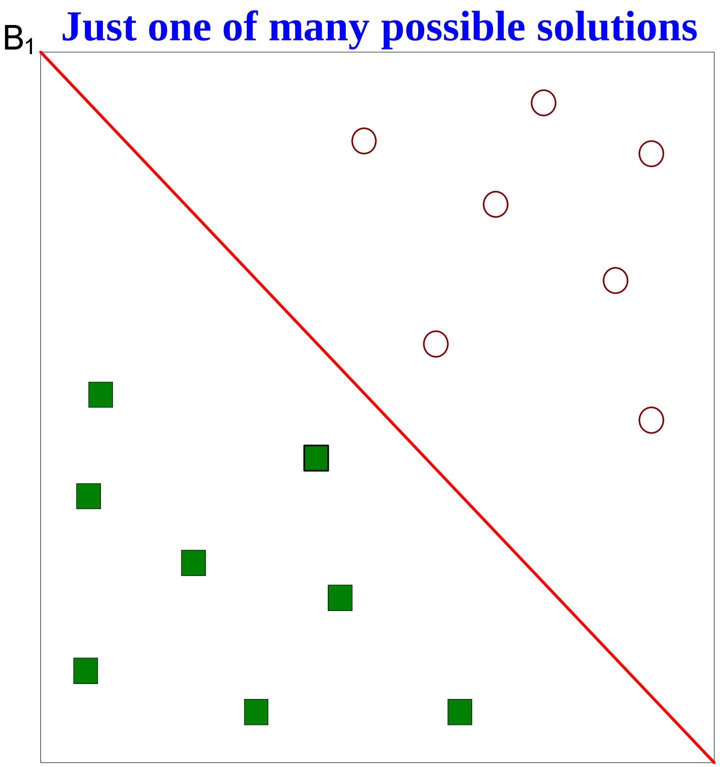
- One-class SVM
  - http://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html#sklearn.svm.OneClassSVM

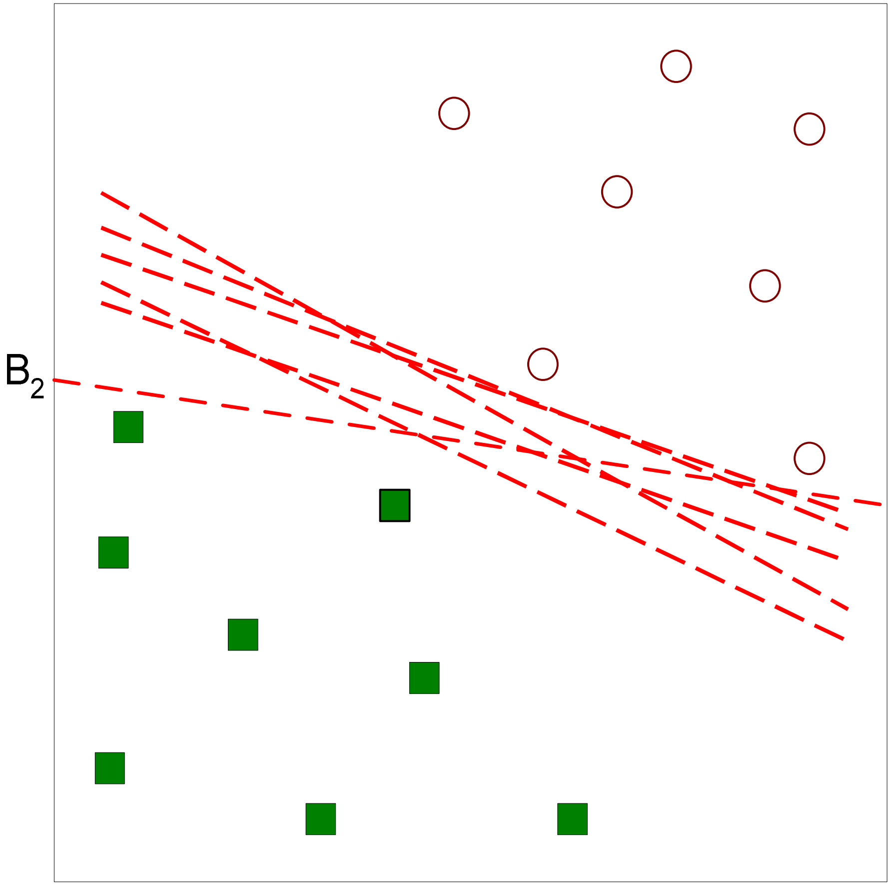
Outline

- SVM Basic Idea, Large Margin Classifier, Support Vectors.
- SVM Model
  - Model Complexity and SVM Vector Length.
  - SVM Decision Boundary and SVM Margin.
  - SVM Hinge Loss Function vs. Logistic Regression.
- SVM Regularization, Box Constraint, SVM Support Vector and α.
- SVM Multiclasses.

Support Vector Machines

- Find a decision boundary to separate data of different classes.
  - Optimal hyperplane for linearly separable patterns.
  - In 2-D a hyperplane is a line.
  - Non-linearly separable data?



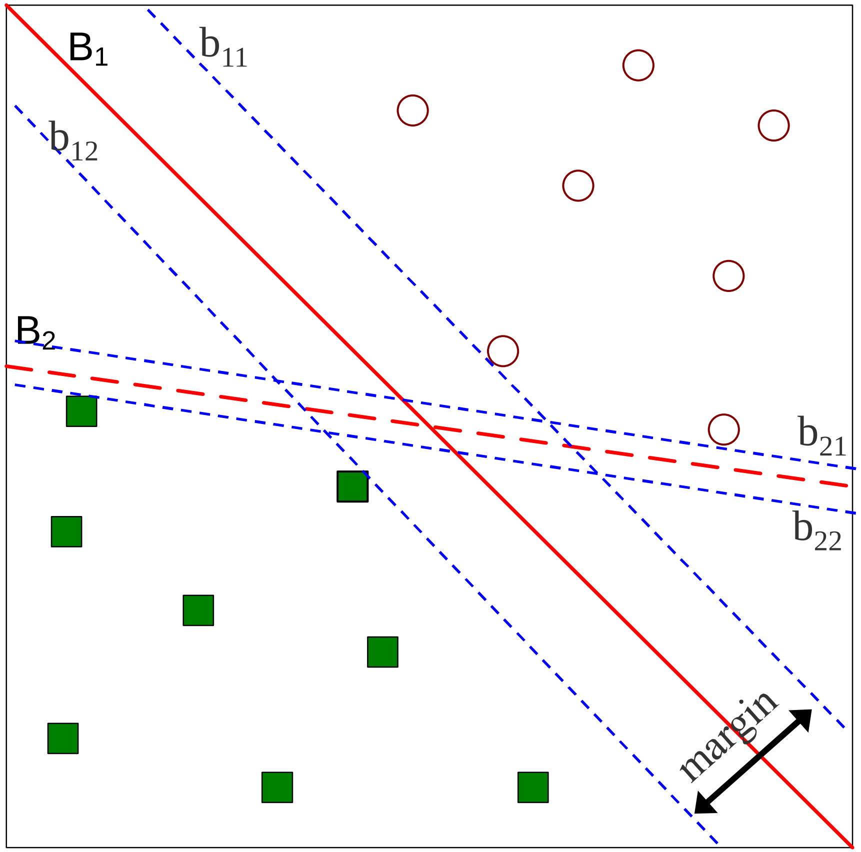


SVM = Large Margin Classifier

- Find the maximum-margin hyperplane that divides points of different classes.
  - Maximize hyperplane margin (to be safe). B1 is better than B2.

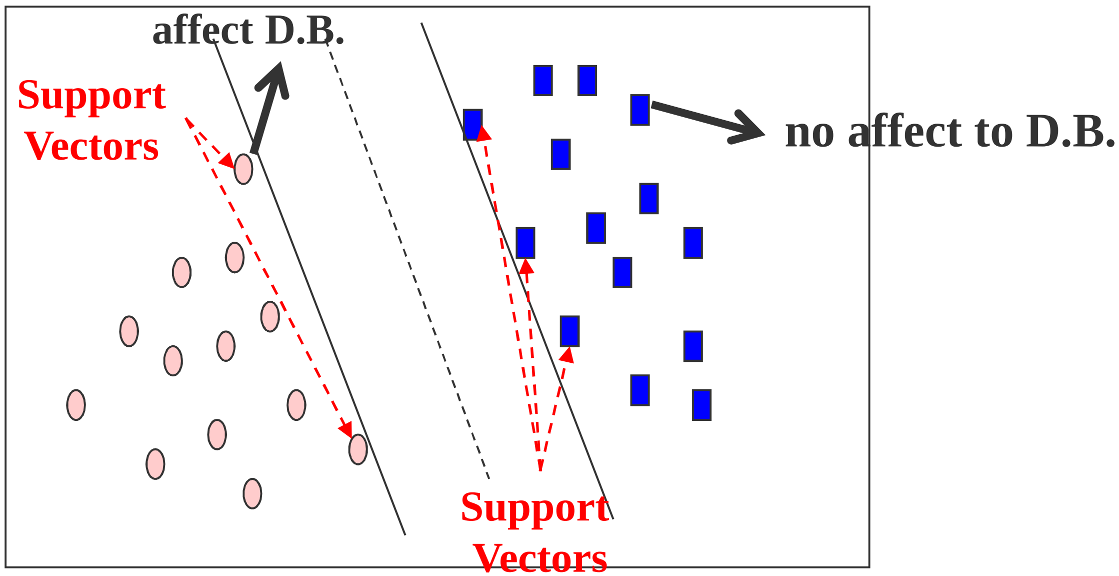
- SVM is usually referred to as ***large margin classifier***.

- But, SVM is **more than** just a large margin classifier.
  - Outliers.
  - ∞-dimension.

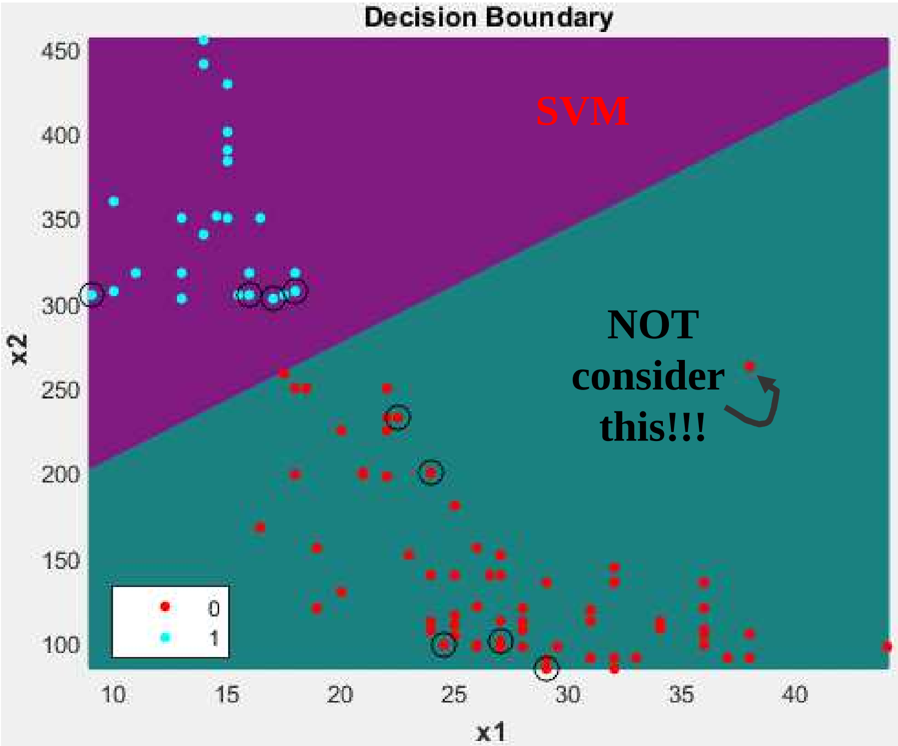


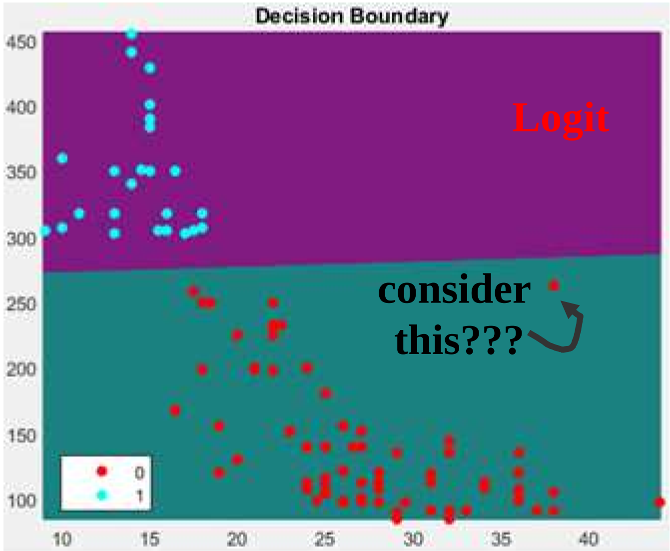
Support Vectors

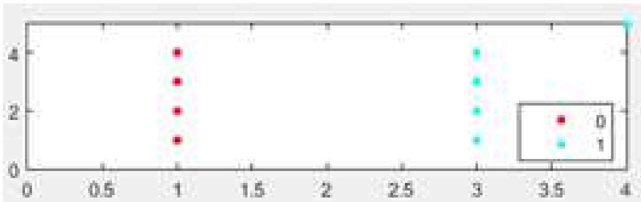
- Support vectors are data points that lie ***closest to*** decision surface.
  - Support vectors are training subset that would **change** hyperplane if moved.
  - Support vectors are the **critical** elements of the training set, most difficult to classify.
  - **Support vectors must be numeric!!** **Converting discrete to numeric???**
- **ONLY** "difficult points" have influence on SVM.
  - DTs, Naïve Bayes, LR, Logit will be influenced by **ALL** points.

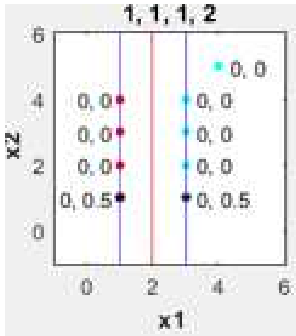


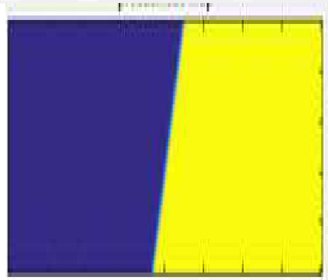
MPG + Displacement ➔ Cylinders, Outlier impact?!



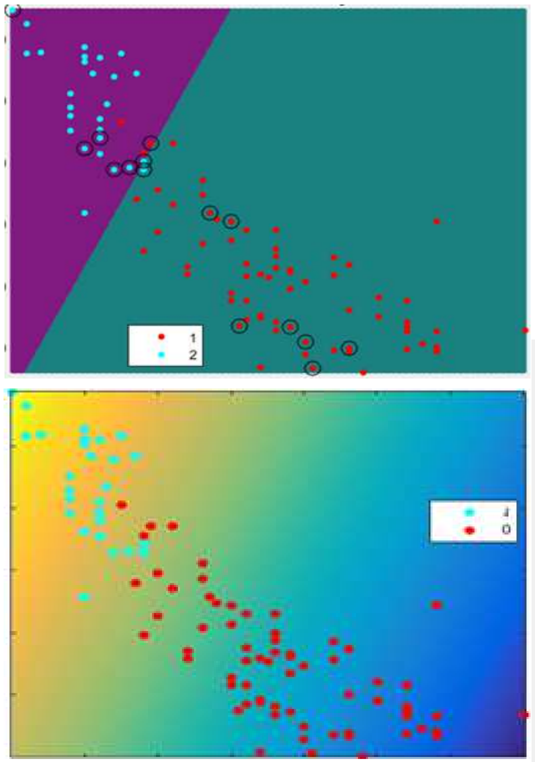


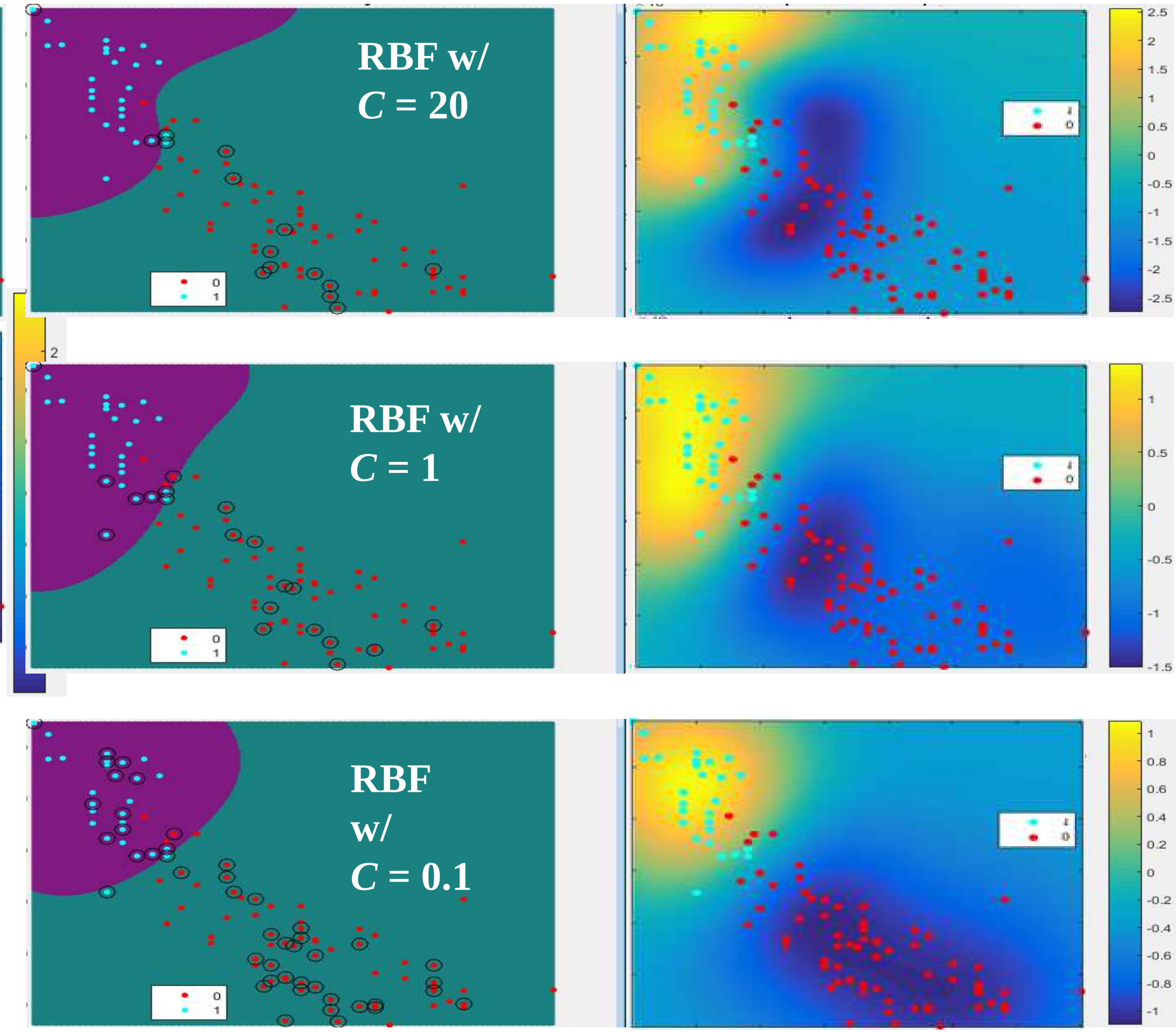






SVM Linear / RBF Kernel + Regularization

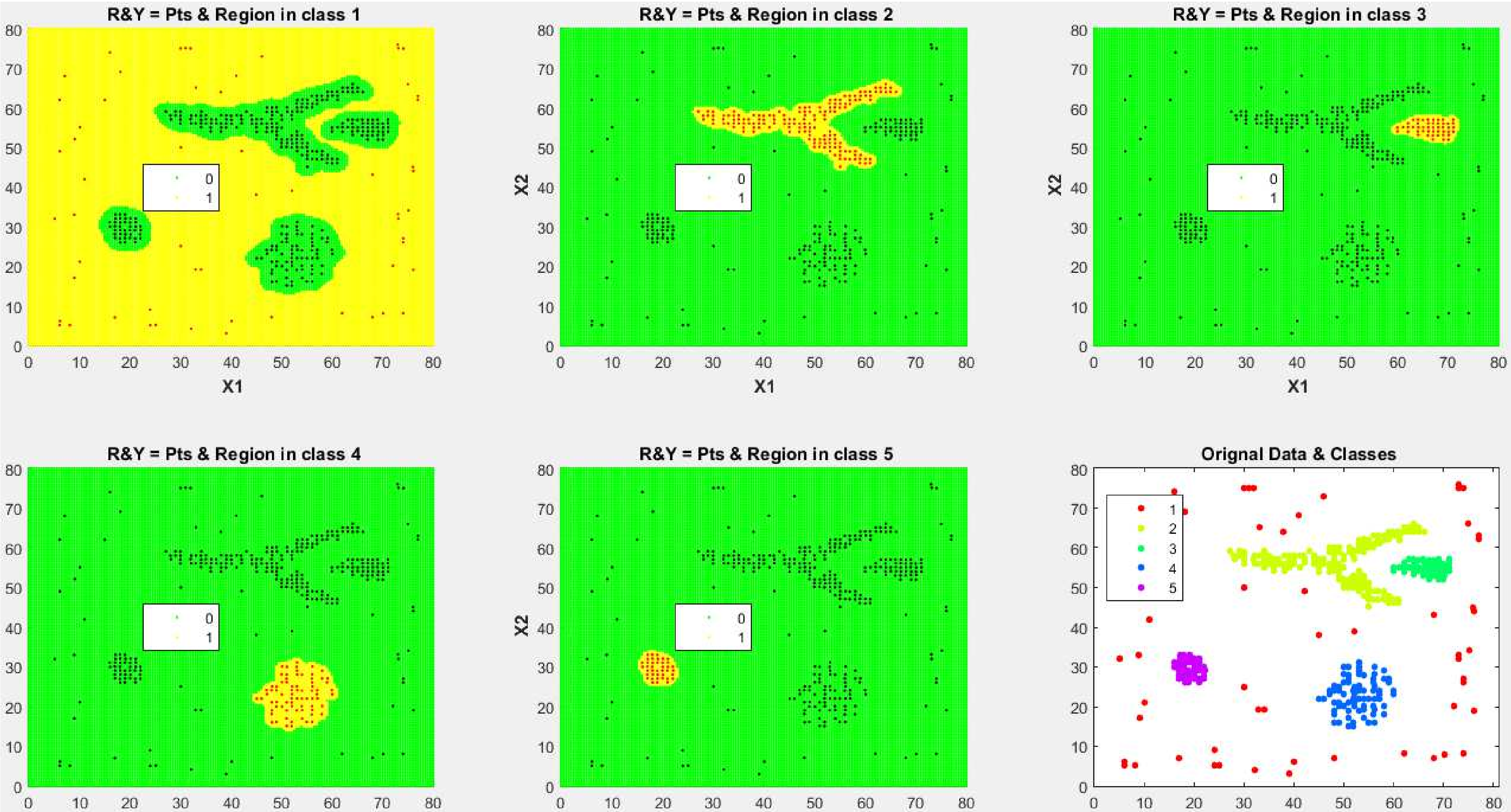




- Predicting vehicle cylinders using MPG + weight.

Powerful SVM Kernal with ∞-Dimensionality

- Build SVM in ∞-dimension



Outline

- SVM Basic Idea, Large Margin Classifier, Support Vectors.

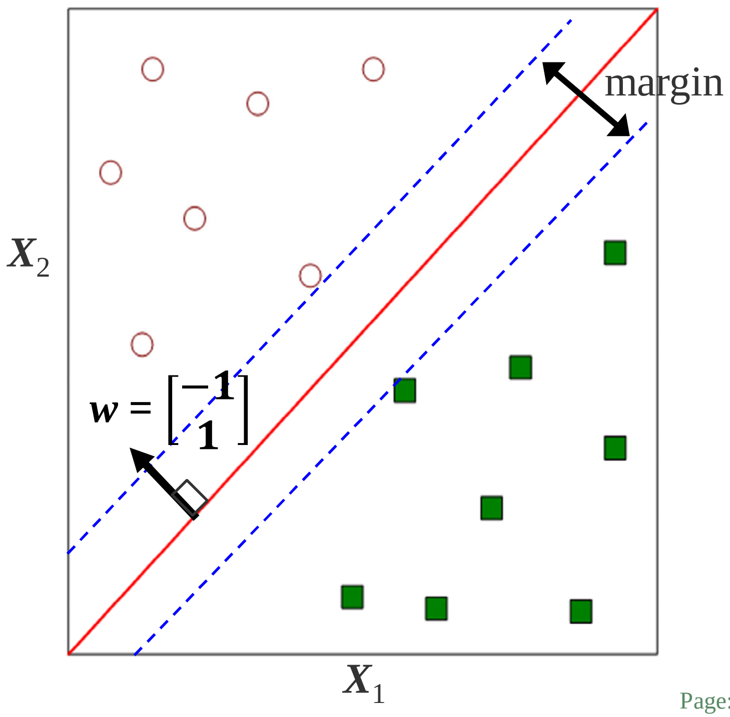
- SVM Model
  - Model Complexity and SVM Vector Length.
  - SVM Decision Boundary and SVM Margin.
  - SVM Hinge Loss Function vs Logistic Regression.

- SVM Regularization, Box Constraint, SVM Support Vector and α.

- SVM Multiclasses.

SVM Model in Plain English

- SVM model learns parameters $\boldsymbol{w}$ (i.e. θ) and $\boldsymbol{b}$ (θ_0) just like other ML methods.
- SVM model learns parameters $\boldsymbol{w}$ (i.e. θ) such that…
  1) It is perpendicular to the class boundary, and at the same time…
  2) It creates the largest possible margin to separate data of 2 different classes.
- So finding $\boldsymbol{w}$ (i.e. θ) **=** finding class boundary.
- Magnitude of $\boldsymbol{w}$ = model complexity
  - Just like other ML methods…
  - One way to define model complexity $\boldsymbol{L}_2$
  - $\boldsymbol{L}_2 = \sqrt[2]{w_1^2 + w_2^2 + \cdots} = ||\boldsymbol{w}||$ = vector length.
  - Turn out Margin = $\frac{\mathbf{2}}{||\boldsymbol{w}||}$

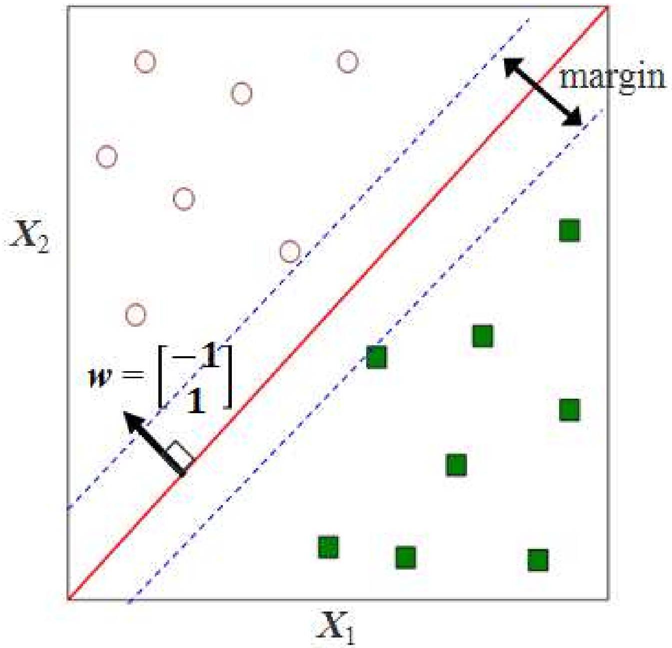


See Appendix for math reasons.

||*w*|| = Length of A Vector *w*

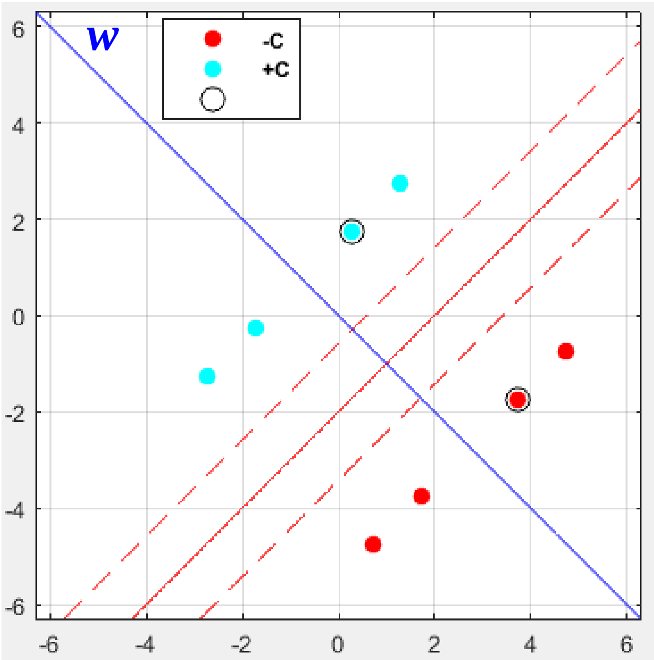
- The length of vector ***w*** = ||***w***|| = $\sqrt{w_1^2 + w_2^2 + \cdots} = \sqrt{w^T w} = \sqrt{\sum_j^d w_j^2}$
  - **Length of vector *w*** **(i.e. θ)** decide the **margin** = $\frac{2}{||w||}$

    See Appendix for math reasons.
  - **Length of vector *w* = complexity** of the SVM model!!!
    - w = [-1 1]; ||w|| = sqrt(w(1)^2 + w(2)^2) = **norm**(w, 2) = sqrt(sum(w.^2)) = 1.414
    - w = [-2 2]; ||w|| = sqrt(w(1)^2 + w(2)^2) = **norm**(w, 2) = sqrt(sum(w.^2)) = 2.8284



$$\text{Margin} = \frac{2}{||w||} = \frac{2}{\sqrt{w^T w}} \qquad ||w|| = \sqrt{w_1^2 + w_2^2 + \cdots}$$

- $||w||$ decide the **margin** of an SVM model, $||w||$ = **model complexity**



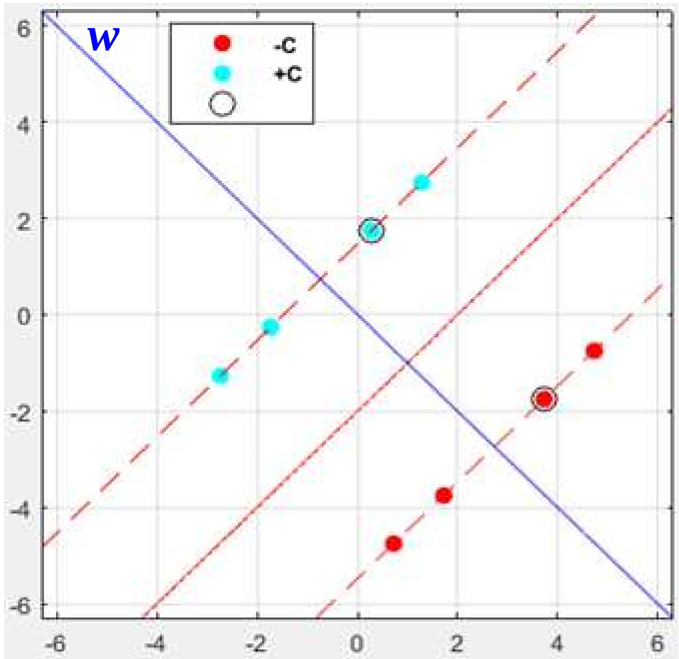
w = -0.70711 0.70711 (magnitude↑)

||w|| = 1, M = 2

w = [-0.70711; 0.70711 1.4142];

w_len = **norm**(w), % sqrt((w' * w))

margin = 2 / w_len



w = -0.28868 0.28868 (magnitude↓)

||w|| = 0.40825, M = 4.899

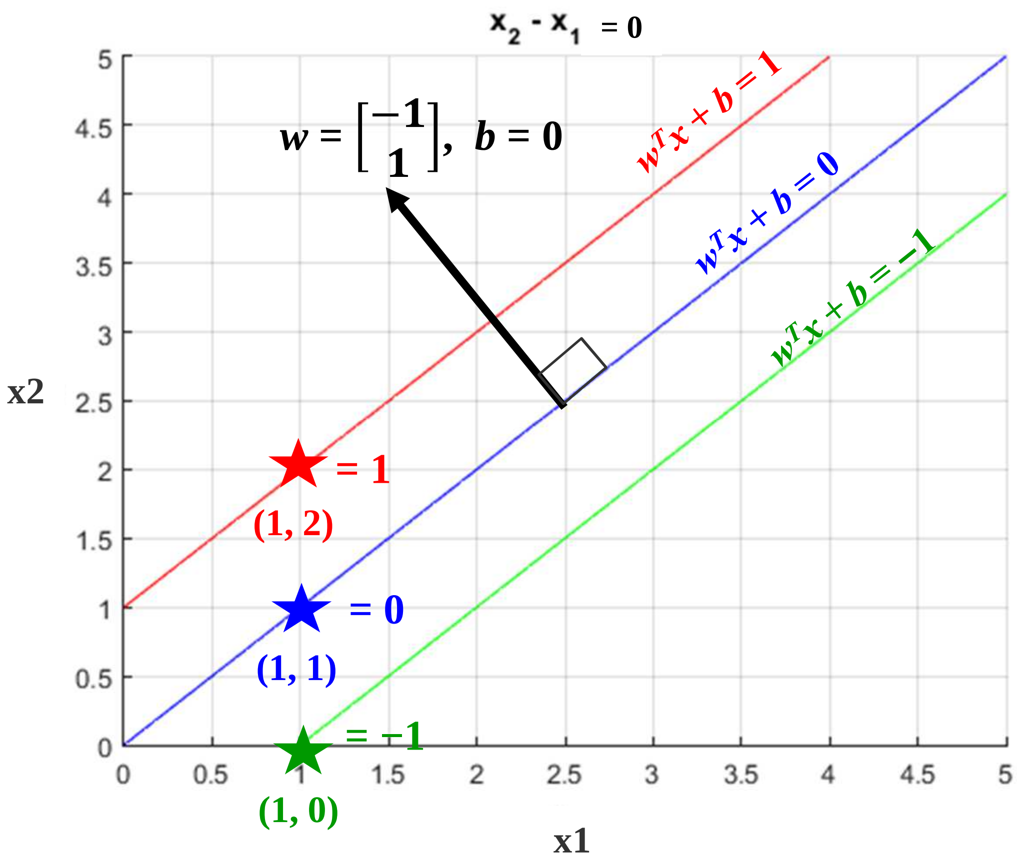
w = [-0.28868; 0.28868 0.57735];

w_len = **norm**(w), % sqrt((w' * w))

margin = 2 / w_len

Hyperplane (Decision Boundary)

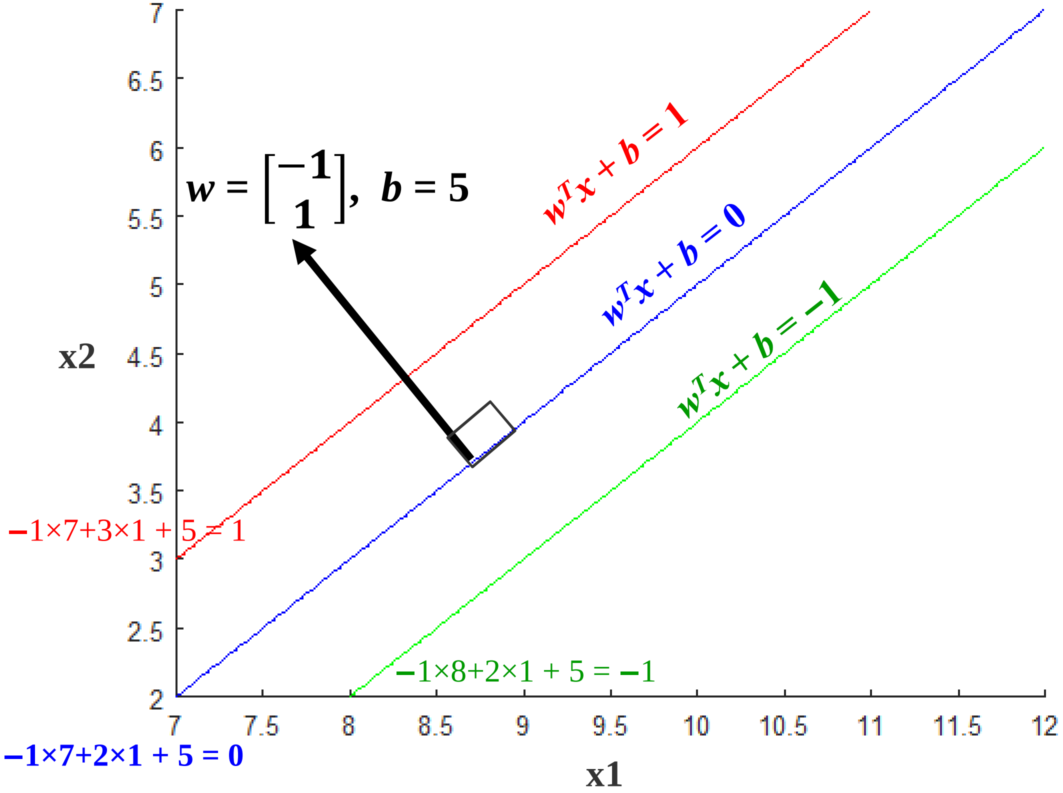
- A hyperplane is a set of points X satisfying $\boldsymbol{w^T X + b = c}$ or $\boldsymbol{w^T X = c}$
  - Vector $\boldsymbol{w}$ is **perpendicular** to the hyperplane (D.B.)
  - $\begin{bmatrix} -1 \\ 1 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \boldsymbol{b} = -x_1 + x_2 + 0$ with $\boldsymbol{b = 0}$ and $\boldsymbol{c} = 0$ for the blue line.



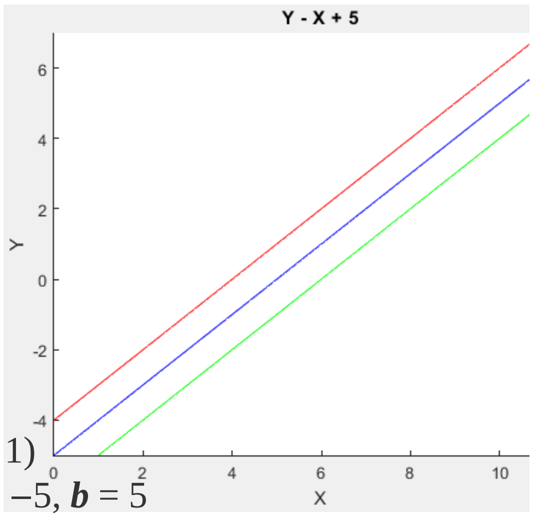

```
syms x1 x2
hold on;
h1 = ezplot(-x1+x2, [0 5 0 5]);     % B
h2 = ezplot(-x1+x2-1, [0 5 0 5]);   % R
h3 = ezplot(-x1+x2+1, [0 5 0 5]);   % G
hold off, grid on
set(h1,'color',[0 0 1])
set(h2,'color',[1 0 0])
set(h3,'color',[0 1 0])
```

Hyperplane– 2

- A hyperplane is a set of points X satisfying $\boldsymbol{w^T X} + \boldsymbol{b} = \boldsymbol{c}$ or $\boldsymbol{w^T X} = \boldsymbol{c}$
  - Vector $\boldsymbol{w}$ is **perpendicular** to the hyperplane.
  - $\begin{bmatrix} -1 \\ 1 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \boldsymbol{b} = -x_1 + x_2 + 5$ with $\boldsymbol{b} = 5$ and $\boldsymbol{c} = 0$ for the blue line



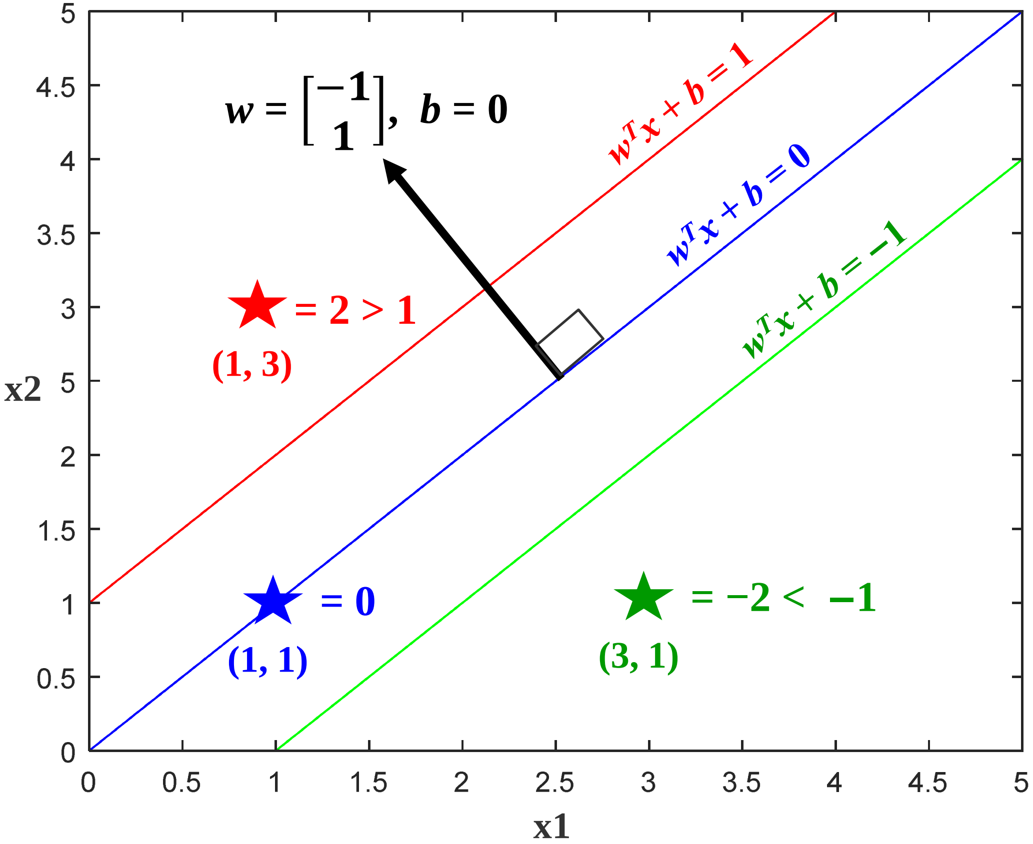

```
syms X Y
hold on;
h1 = ezplot(-X+Y+5, [0 12 -5 7]);  % B
h2 = ezplot(-X+Y+4, [0 12 -5 7]);  % R
h3 = ezplot(-X+Y+6, [0 12 -5 7]);  % G
hold off, grid on
set(h1,'color',[0 0 1]), set(h2,'color',[1 0 0])
set(h3,'color',[0 1 0])
```



$\boldsymbol{w} = (-1, 1)$
$\boldsymbol{x} = 0, \boldsymbol{y} = -5, \boldsymbol{b} = 5$

> 1 or < -1

- A hyperplane is a set of points X satisfying $\boldsymbol{w^T X + b = c}$ or $\boldsymbol{w^T X = c}$
  - Vector $\boldsymbol{w}$ is **perpendicular** to the hyperplane.
  - $\begin{bmatrix} -1 \\ 1 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \boldsymbol{b} = -x_1 + x_2 + 0$ with $\boldsymbol{b} = 0$ and $\boldsymbol{c} = 0$ for the blue line.

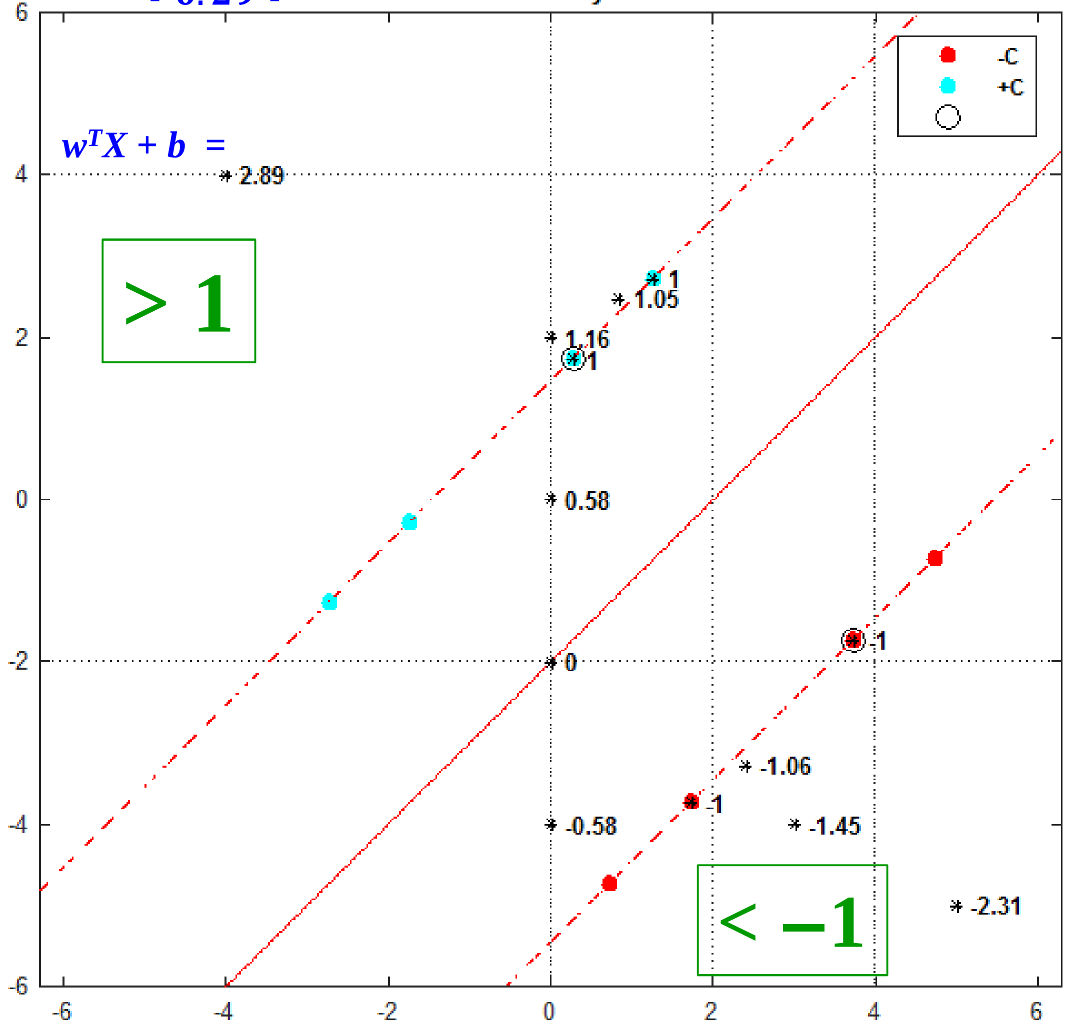



```
syms X Y
hold on;
h1 = ezplot(-X+Y, [0 5 0 5]);
h2 = ezplot(-X+Y-1, [0 5 0 5]);
h3 = ezplot(-X+Y+1, [0 5 0 5]);
hold off,     grid on
set(h1,'color',[0 0 1])
set(h2,'color',[1 0 0])
set(h3,'color',[0 1 0])
```

> 1 or < -1

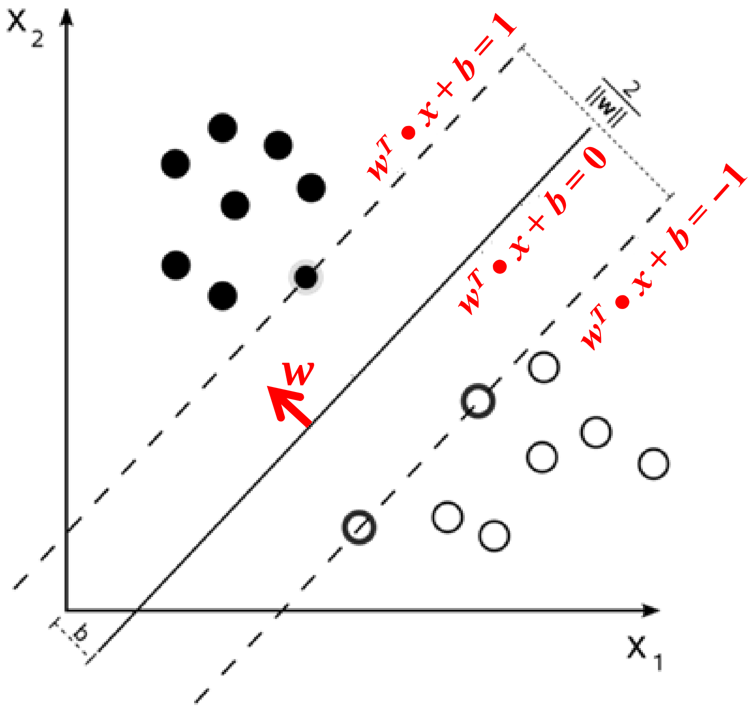
- $w^T X + b$

$w = \begin{bmatrix} -0.29 \\ 0.29 \end{bmatrix}$ **$||w|| = 0.408$, M = 4.899**



SVM Definition

- A hyperplane is a set of points X satisfying $\boldsymbol{w^T X + b = c}$ or $\boldsymbol{w^T X = c}$
  - Vector $\boldsymbol{w}$ is **perpendicular** to the hyperplane.
  - Margin = $\frac{2}{||w||}$ magnitude of $\boldsymbol{w}$ = **model complexity**
  - Maximize the margin by choosing right $\boldsymbol{w}$ and $\boldsymbol{b}$, w.r.t. $\boldsymbol{X}$
    - Parallel hyperplanes are as far apart as possible, but still separate data



http://en.wikipedia.org/wiki/Support_vector_machine

Matlab SVM-Related Functions and References

- Matlab SVM: **http://www.mathworks.com/help/stats/support-vector-machines-svm.html#responsive_offcanvas**

- Matlab SVM class
  - http://www.mathworks.com/help/stats/support-vector-machine-classification.html
  - http://www.mathworks.com/help/stats/classificationsvm-class.html

- Matlab Functions:
  - **fitcsvm**(X, Y, **'KernelFunction', 'rbf',** 'Crossval', 'on', 'Standardize', true);
    - http://www.mathworks.com/help/stats/fitcsvm.html
    - http://www.mathworks.com/help/stats/classificationsvm-class.html

    $w^T X + b$

  - [**label**, **score**] = **predict**(SVMModel, newX);
    - http://www.mathworks.com/help/stats/compactclassificationsvm.predict.html

  - crossval(), kFoldLoss(), kfoldPredict()

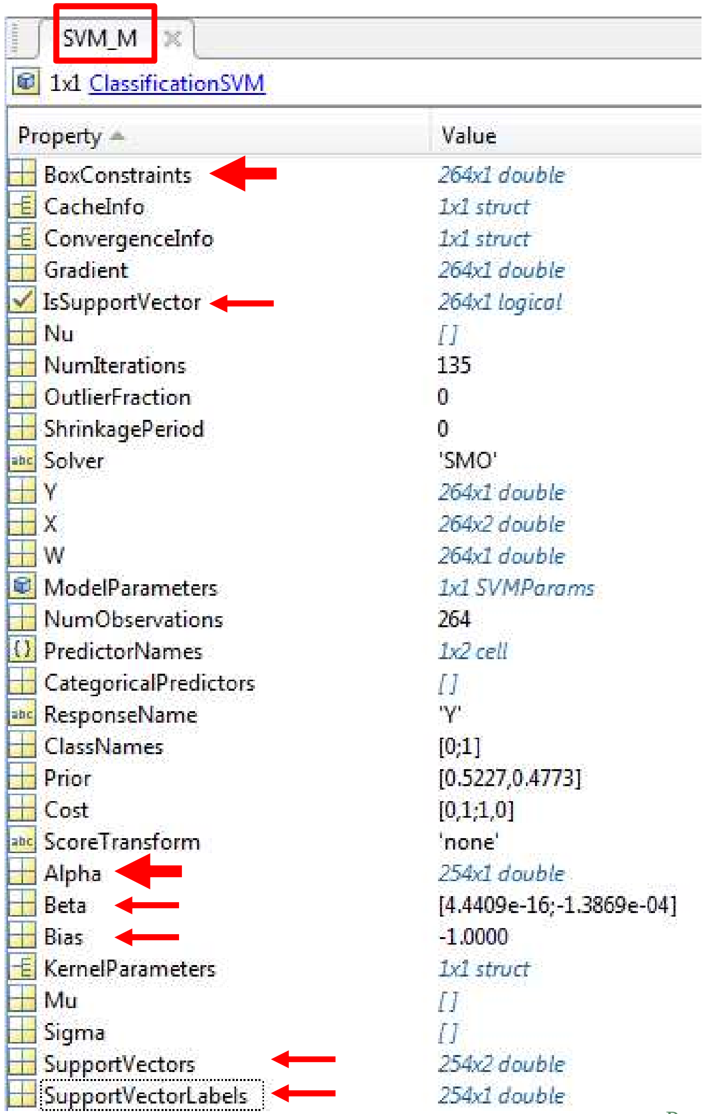
Detailed Information Returned from Matlab SVM

- Matlab **SVM** class
  - http://www.mathworks.com/help/stats/support-vector-machine-classification.html
  - http://www.mathworks.com/help/stats/classificationsvm-class.html

- **.Beta** **.Bias** **(i.e. $[\theta_{1..N} \quad \theta_0]$ $[w_{1..N} \quad w_0]$)**
- **.IsSupportVector**
- **.SupportVectorLabels**
- **.SupportVectors**
- **.BoxConstraints** **(regularizer C)**
- **.Alpha**

- $\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{X} + \boldsymbol{b} = \boldsymbol{c}$
- Margin $= \frac{\mathbf{2}}{||\boldsymbol{w}||}$
- $min_{w,b,\alpha} \frac{1}{2}||\boldsymbol{w}||^2 - \boldsymbol{C} \times \sum_{i=1}^{m}[\boldsymbol{\alpha_i}(1 - y^{(i)}(\boldsymbol{w}x^{(i)} + \boldsymbol{b}))]$

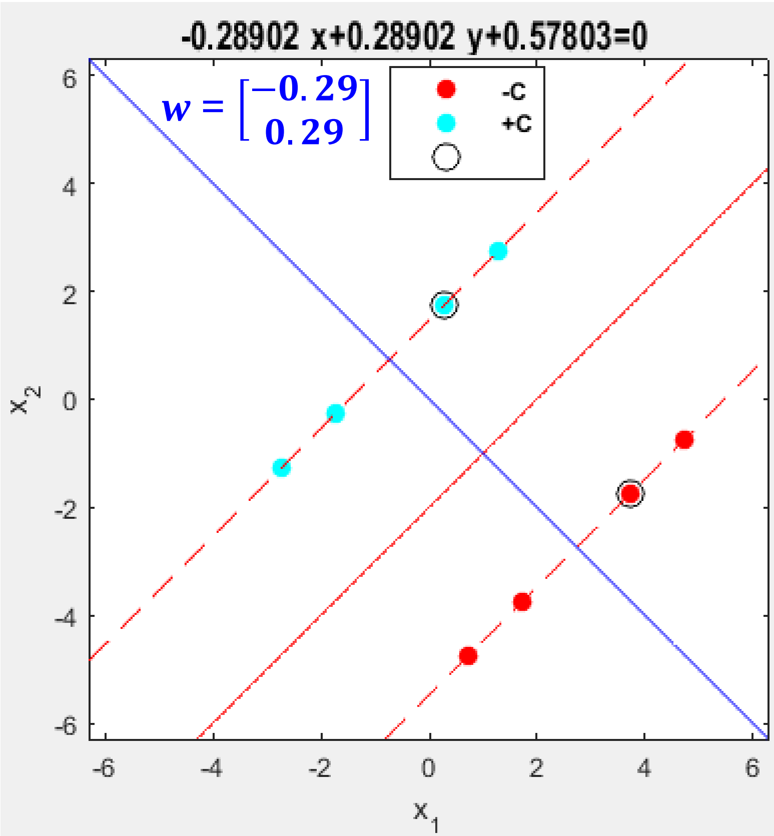
- **.W**
- **.OutlierFraction**



Matlab SVM Simple Example

3.7321	-1.7321	-1
4.7321	-0.7321	-1
1.7321	-3.7321	-1
0.7321	-4.7321	-1
0.2679	1.7321	1
1.2679	2.7321	1
-1.7321	-0.2679	1
-2.7321	-1.2679	1

$||w|| = 0.408$, M = 4.899

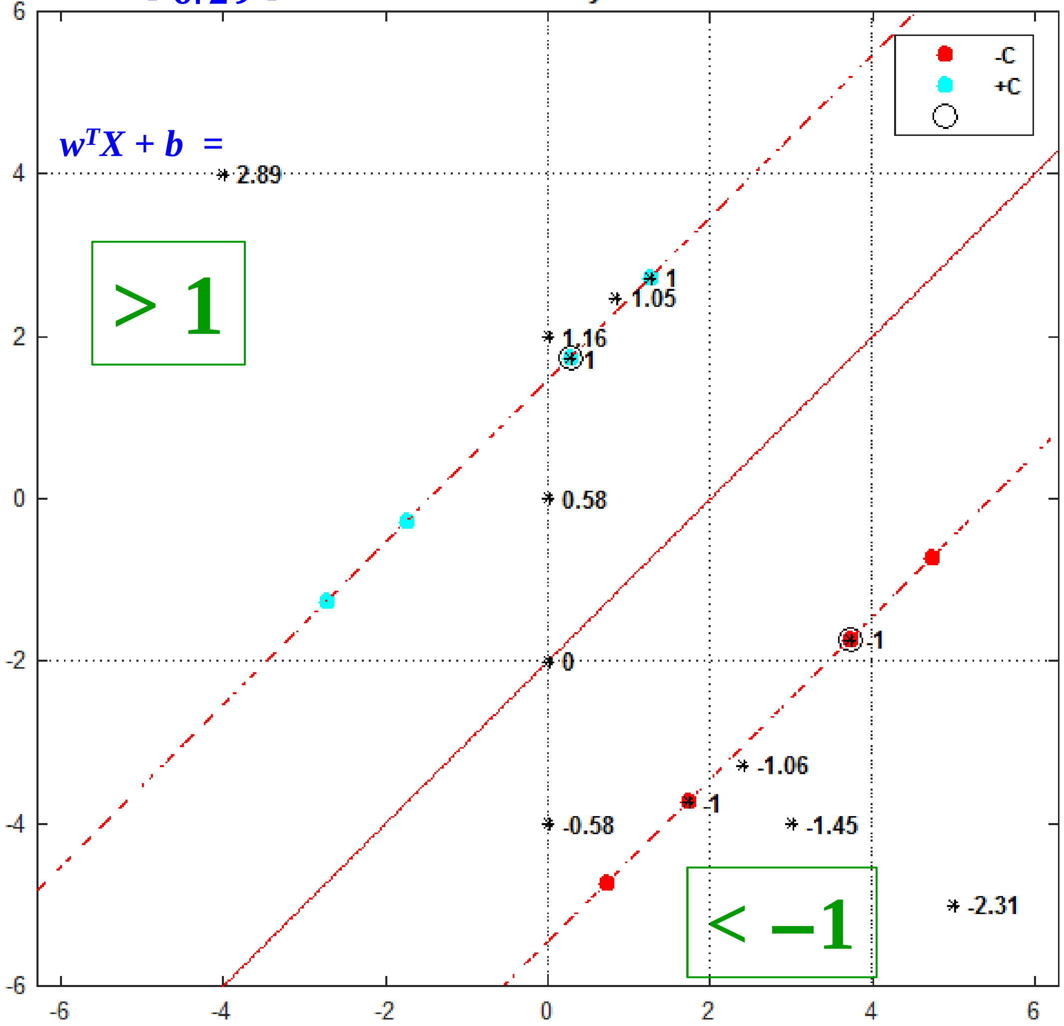



```
X = [3.73, 4.73, 1.73, 0.73, 0.27, 1.27, -1.73, -2.73; -1.73, -0.73, -3.73, -4.73, 1.73, 2.73, -0.27, -1.27]';
Y = [-1; -1; -1; -1; 1; 1; 1; 1];

SVM = fitcsvm(X, Y)

sv = SVM.SupportVectors;

theta = SVM.Beta;        theta0 = SVM.Bias;          % w

Sep = null(theta');     % dot(theta, Sep) ≈ 0

gscatter(X(:,1),X(:,2), Y, '', '', 25)

hold on, plot(sv(:,1), sv(:,2), 'ko'),    % ← plot SVs

fstr1 = [num2str(theta(1)) '*x+'  num2str(theta(2)) ...
                '*y+' num2str(theta0) '=0']

h1 = ezplot(fstr1);     hold off          % plot separation


% worry this later.
w_len = norm(theta);      margin = 2 / w_len
```

>1 or <–1 using Matlab predict() Function

- $w^T X + b$ = **predict**(**SVM_Model**, **newData**)

$w = \begin{bmatrix} -0.29 \\ 0.29 \end{bmatrix}$ $||w|| = 0.408$, $M = 4.899$



X = [3.73, 4.73, 1.73, 0.73, 0.27, 1.27, -1.73, -2.73; -1.73, -0.73, -3.73, ...
-4.73, 1.73, 2.73, -0.27, -1.27]';
Y = [-1; -1; -1; -1; 1; 1; 1; 1];

SVM = **fitcsvm**(X, Y)

sv = SVM.**SupportVectors**;

theta = SVM.**Beta**; theta0 = SVM.**Bias**;

newX = [-4 4; 0 2; 3 -4; 3.73 -1.73; 1.73 -3.7300; ...
1.27 2.73; 0.27 1.73; 0.85 2.48; 2.4 -3.27; 5 -5];

theta' * newX' + theta0 % equal to next

[label, **score**] = **predict**(SVM, newX)

clf.**predict**(X) clf.**decision_function**(X)

from sklearn.svm import SVC

clf = SVC()

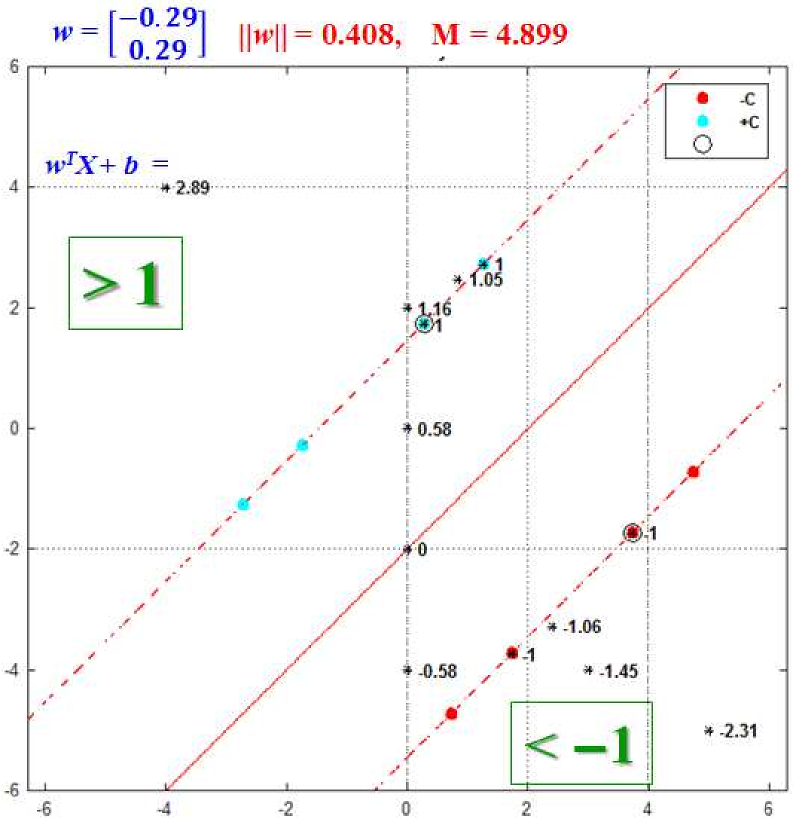
clf.fit(X, y) clf.**support_vectors_**

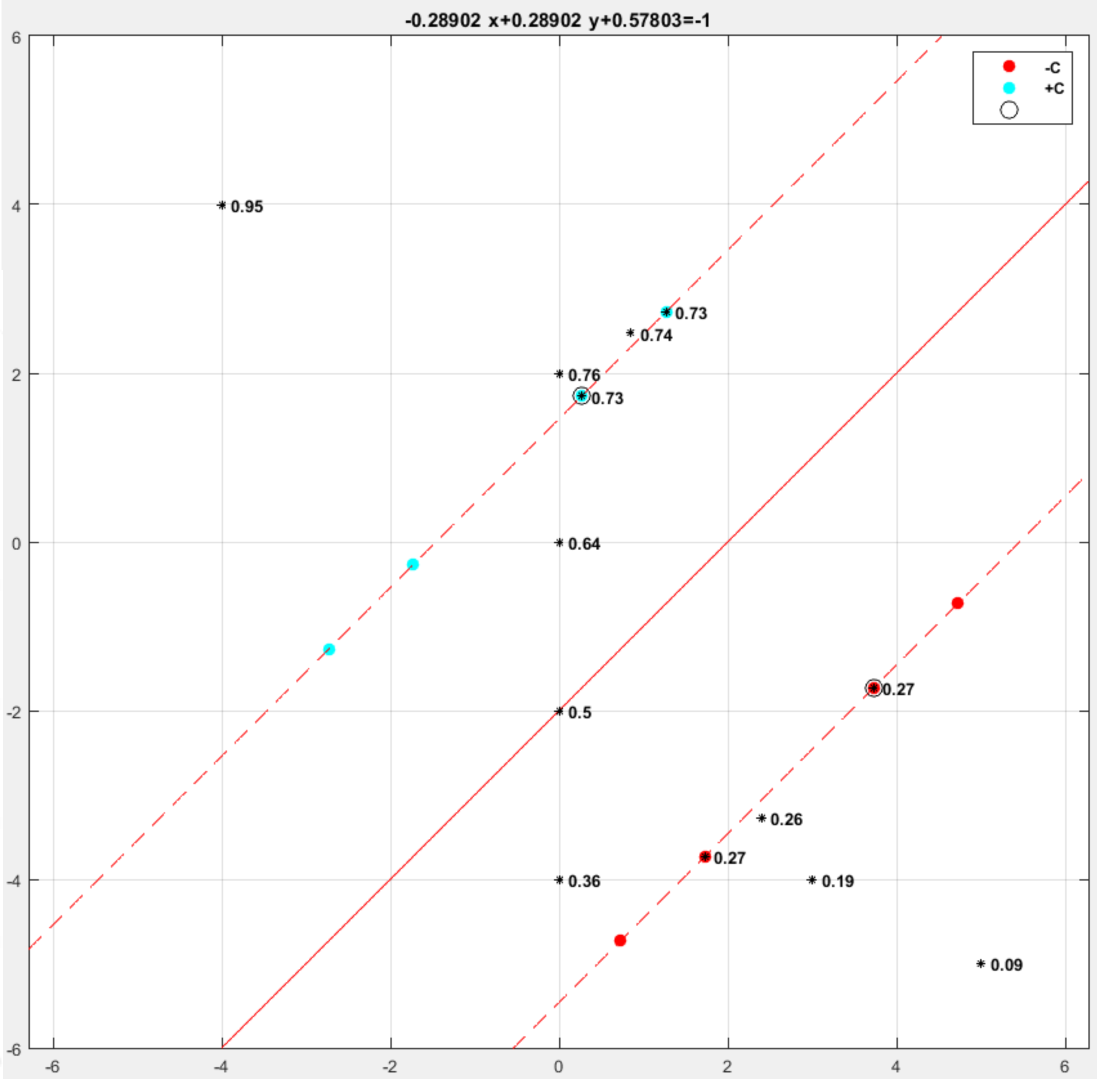
Probability Calculation

- How do we do that?
  - Utilizing what we have learned so far…

SVM = fitcsvm(X, Y, <u>**'ScoreTransform', 'logit')**</u>

1./(1+exp(-(SVM.Beta'*X'+SVM.Bias)))

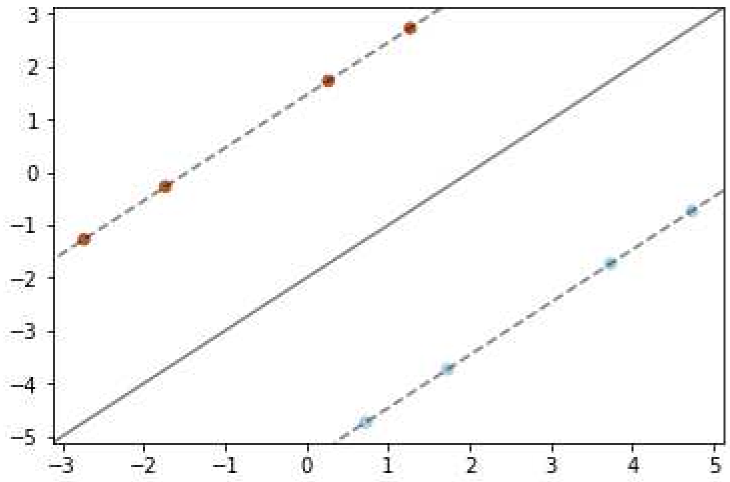




sklearn.svm.SVC, Build Model

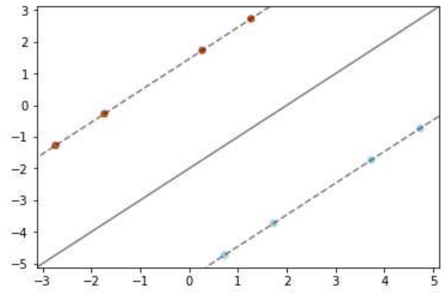
- Don’t use “*svm.LinearSVC*”, use **svm.SVC**(**kernel = 'linear'**) instead.
  - It doesn’t return support-vector information.
  - http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.decision_function

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
X1 =  [3.73,  4.73,  1.73,  0.73,  0.27,  1.27,  -1.73, -2.73]
X2 = [-1.73,  -0.73,  -3.73,  -4.73,  1.73,  2.73,  -0.27,  -1.27]
X = np.column_stack(  (X1, X2)  )
y = np.array([-1, -1, -1, -1, 1, 1, 1, 1])

# fit the model, don't regularize for illustration purposes
clf = svm.SVC(kernel = 'linear', C = 1000) # ←
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)
# plot the decision function
ax = plt.gca();  xlim = ax.get_xlim();  ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)
```




```
# plot decision boundary and margins
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
        linestyles=['--', '-', '--'])
# plot support vectors
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
        linewidth=1, facecolors='none’)
plt.show()
```

sklearn.svm.SVC, Obtain Results




```
print(clf.n_support_)  # number of support vectors in EACH class
print(clf.support_)      # Indices of support vectors
print(clf.support_vectors_)  # Support vectors
print(clf.coef_)          # coefficients in "primary" form
print(clf.dual_coef_)  # coefficients in "dual" form
```

```
[1 1]
[3 7]
[[ 0.73 -4.73]
 [-2.73 -1.27]]
[[-0.28901735  0.28901735]]
[[-0.08353102  0.08353102]]
```

```
print(clf.predict(X), '\n')                # predicted classes
print(clf.score(X, y), '\n')               # mean accuracy
print(clf.decision_function(X), '\n')  # Distance of X to hyperplane
```

```
[-1 -1 -1 -1  1  1  1  1]

1.0

[-1.00000001 -1.00000001 -1.00000001 -1.00000001  1.00000002  1.00000002
  1.00000002  1.00000002]
```

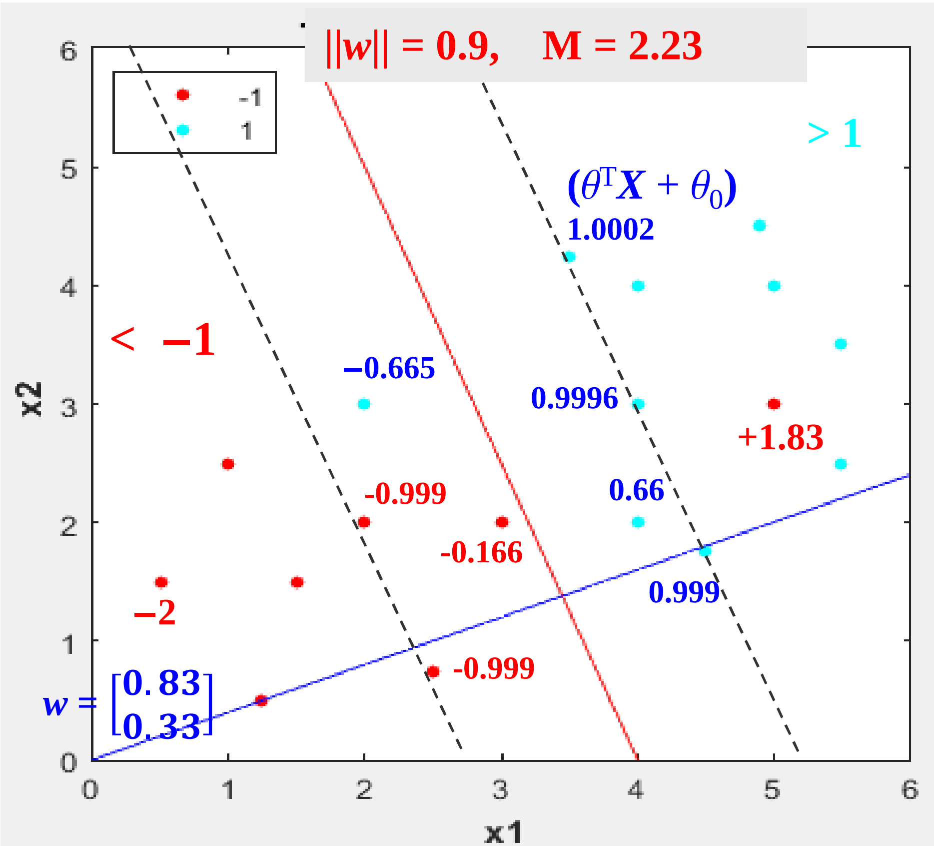
```
clf = svm.SVC(kernel='linear', C=1000, probability = True)
print(clf.predict_proba(X), '\n')       # predicted probability
```

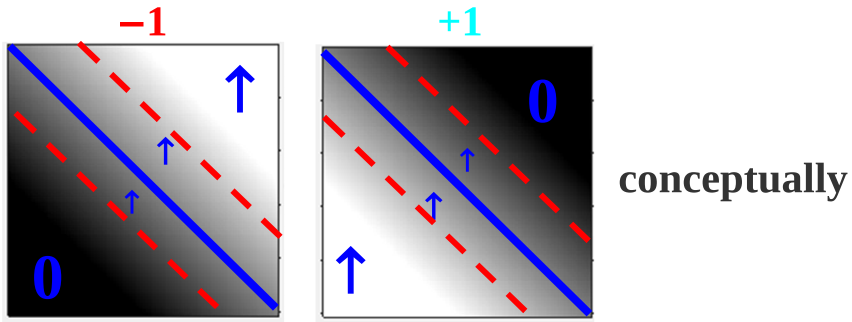
```
[ 0.833333    0.166667   ]
[ 0.16658152  0.83341848]
```

- Similar example at: http://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane.html

> 1 or < –1 and Cost (i.e. Error) Function?

- How to impose penalty (i.e. error) to the points that are wrongly classified?
  - Especially for those VERY VERY wrongly classified?
  - But, **NO** penalty to points that are correctly classified.





$$[max(\,0, (1 - y_i \times w^T x_i\,)\ ]$$

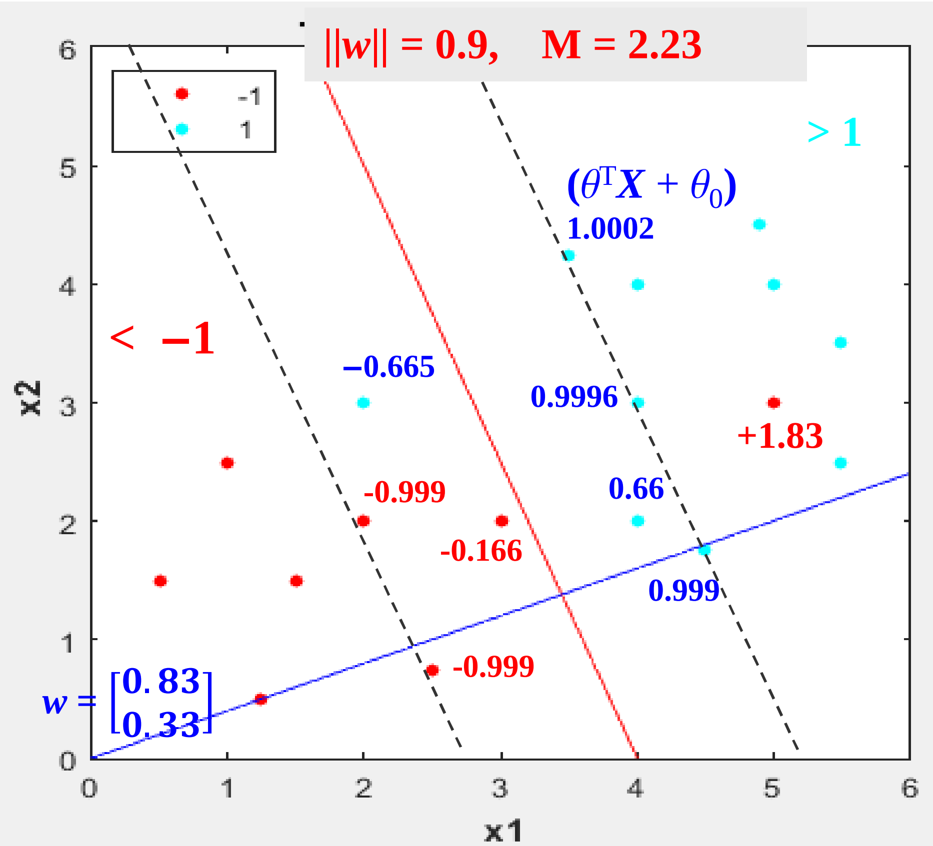
max(0, (1 – (–1) × **1.83**)) = max(0, (1 – (–1.83)) =
max (0, 1 + 1.83) = 2.83

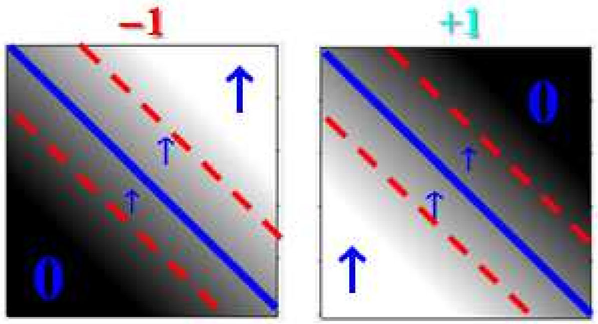
max(0, (1 – (–1) × **–2**)) = max(0, (1 – 2) =
max (0, –1) = 0

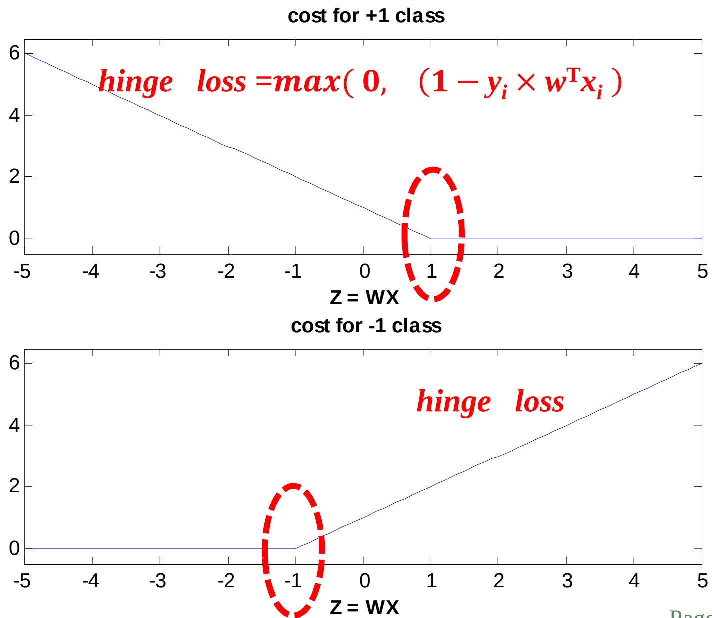
max(0, (1 – 1 × **–0.665**)) = max(0, (1 + 0.665) =
max (0, 1.665) = 1.665

Hinge Loss = $[max(0, (1 - y_i \times w^T x_i)\ )]$

- ***Hinge loss*** ➔ error↑ as pts move **opposite** from the correct class boundary.
  - Need **right** margin $\frac{2}{||w||}$ by choosing **right** w to minimize error.
    - Moving points far away from f(x) = **±1**.

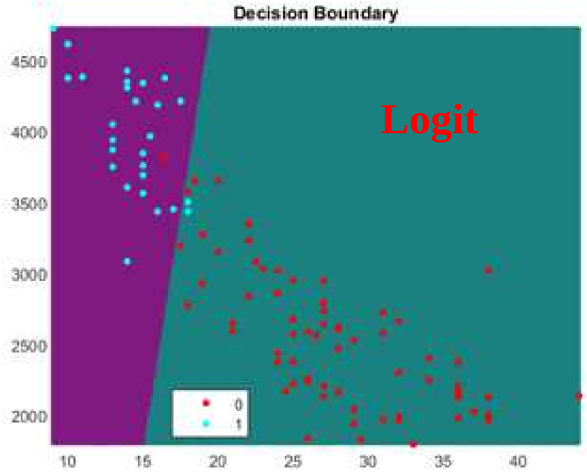


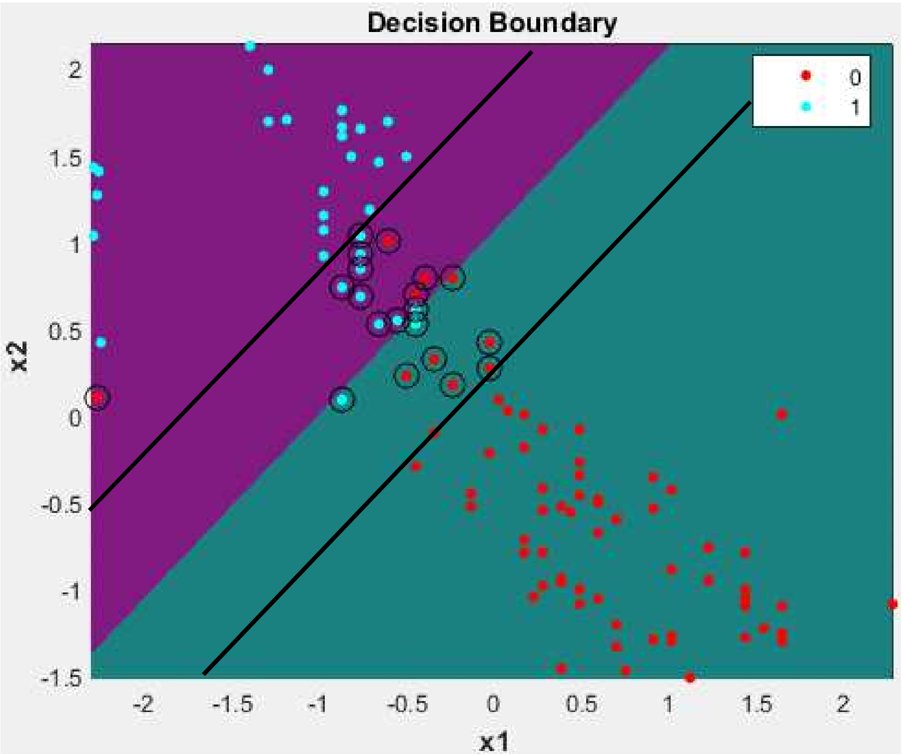


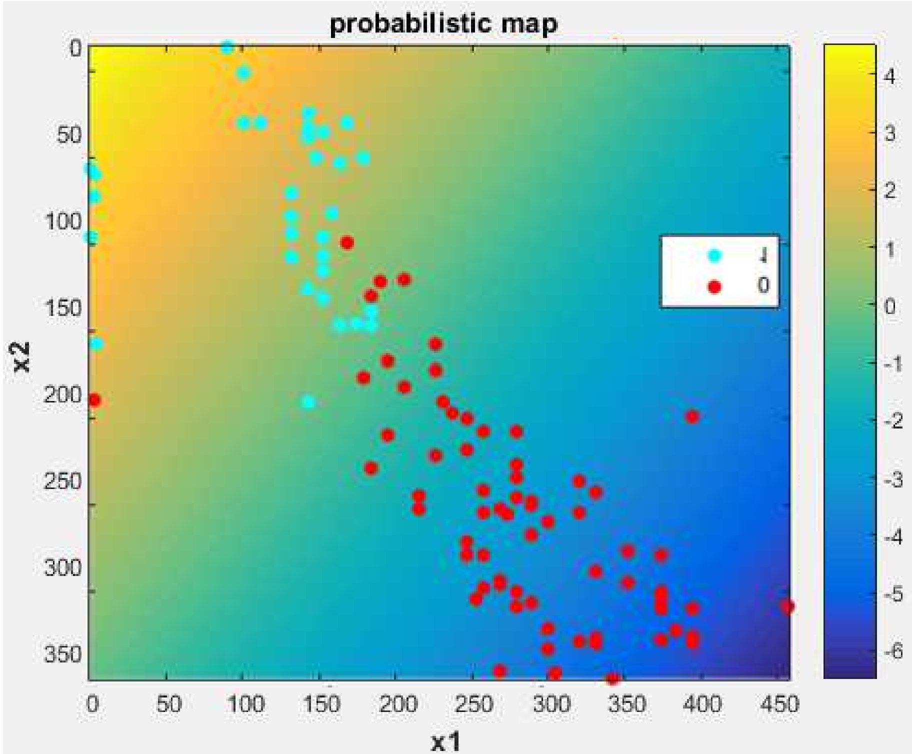


SVM, MPG + Weight ➔ # Cylinders

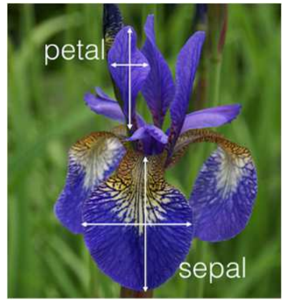
- $w^T X + b$ = **predict(SVM_Model, newData)**
  - [**labels, score**] = **predict**(svm_mdl, X);
- Margin = $\frac{2}{||w||}$
  - $||w||$ = magnitude of w = **model complexity**







SVM Prediction– Iris Example (using var 3 and 4)



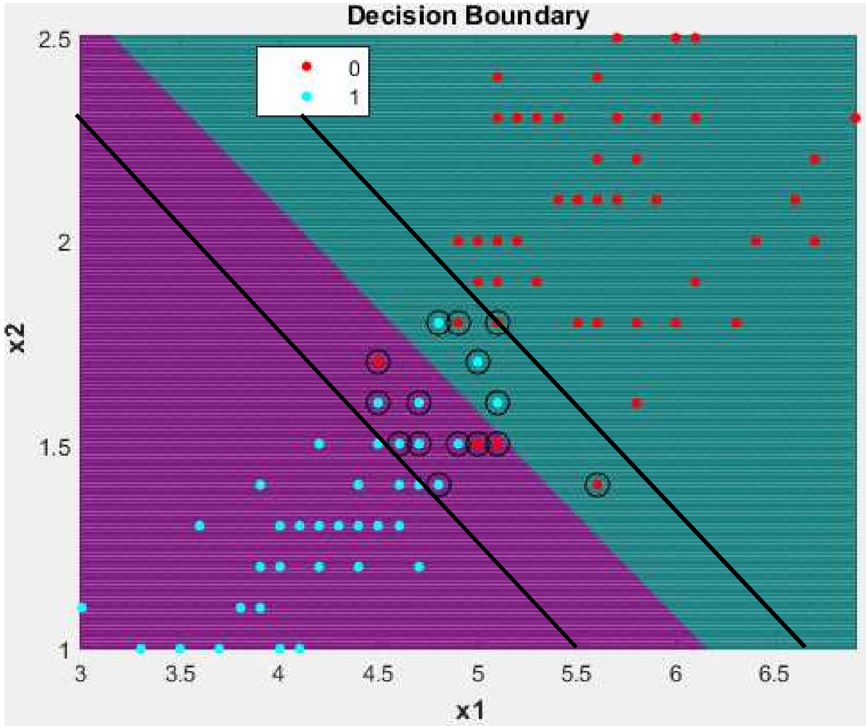
http://sebastianraschka.com/Articles/2014_python_lda.html

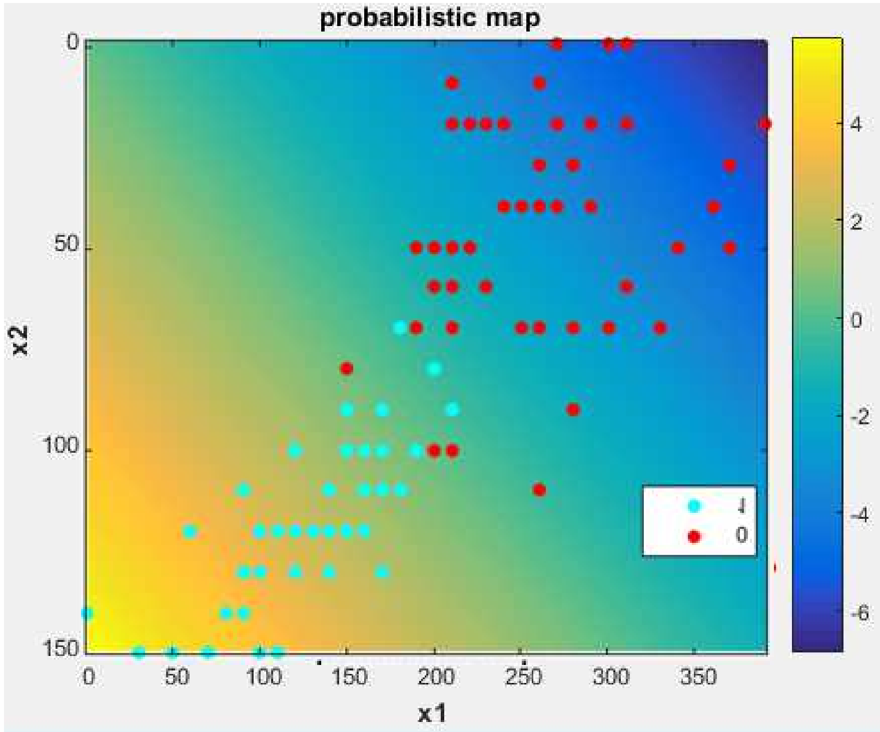
```
load fisheriris
X = meas(51:end, 3:4);
Y = double(strcmp('versicolor',species(51:end)));     % create BINARY class

svm_mdl = fitcsvm(X, Y, 'Standardize', true)
[labels score] = predict(svm_mdl , X);
CFM = confusionmat(Y,  labels)
accuracy = sum(diag(CFM))/sum(CFM(:)),
```

CFM

47	3
3	47

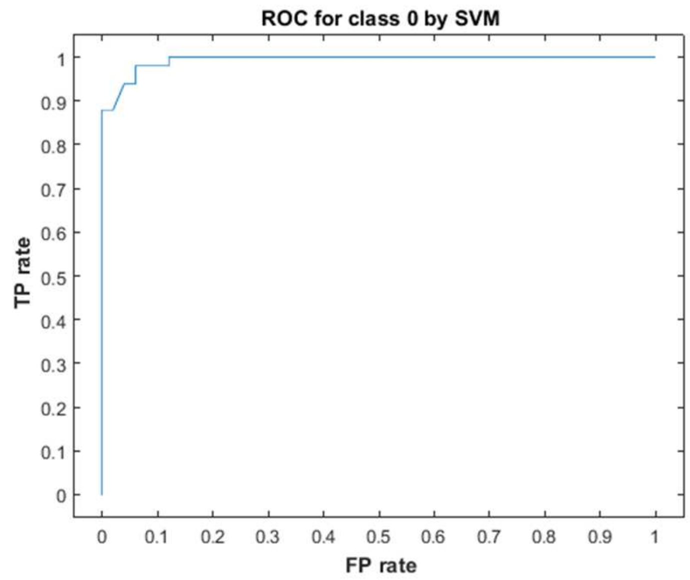
0.9400

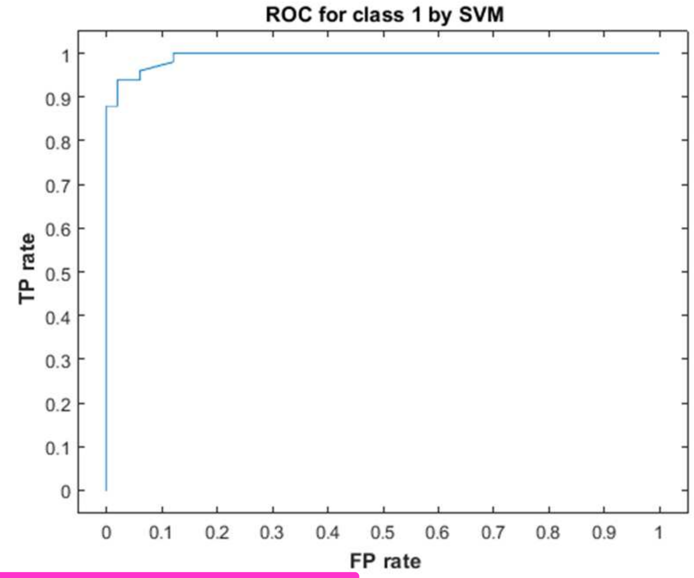




SVM Iris Prediction and ROC Curves

```
load fisheriris
X = meas(51:end, 3:4);
Y = double(strcmp('versicolor',species(51:end)));     % BINARY class

svm_mdl = fitcsvm(X, Y, 'Standardize', true)
[PredictedClasses, scores] = predict(svm_mdl , X);
CFM = confusionmat(Y,  PredictedClasses)
accuracy = sum(diag(CFM))/sum(CFM(:)),

[xpos, ypos, T, AUC0] = perfcurve(Y, scores(:, 1), 0);
figure,     plot(xpos, ypos)
xlim([-0.05 1.05]), ylim([-0.05 1.05])
xlabel('\bf FP rate'),  ylabel('\bf TP rate')
title('\bf ROC for class 0 by SVM')

[xpos, ypos, T, AUC1] = perfcurve(Y, scores(:, 2), 1);
figure,     plot(xpos, ypos)
xlim([-0.05 1.05]), ylim([-0.05 1.05])
xlabel('\bf FP rate'),  ylabel('\bf TP rate')
title('\bf ROC for class 1 by SVM')
```





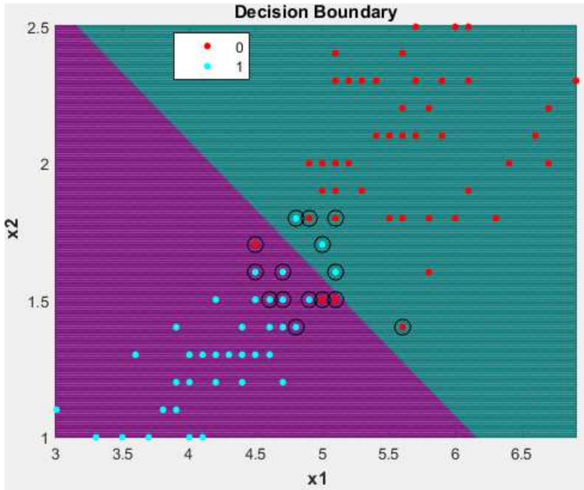
sklearn.**metrics**.**roc_curve**(y, scores, pos_label)

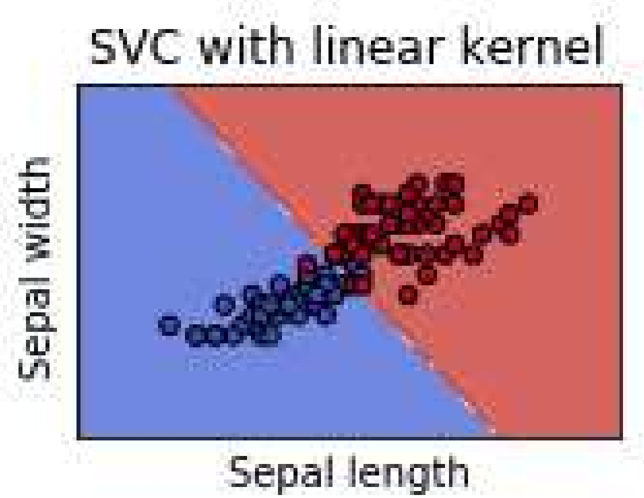
sklearn.**metrics**.**roc_auc_score**

sklearn, SVM Prediction– Iris Example (using var 3 and 4)

- **http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html**

```
# import some data to play with
iris = datasets.load_iris()
# Take the last two features.
X = iris.data[50:, 2:4]
y = iris.target[50:]
print(y.shape, y.size)

C = 1.0        # SVM regularization parameter
clf = svm.SVC(kernel = 'linear', C = C)
clf.fit(X, y)
```

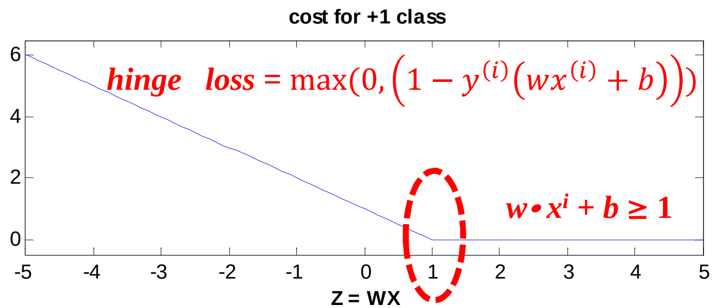


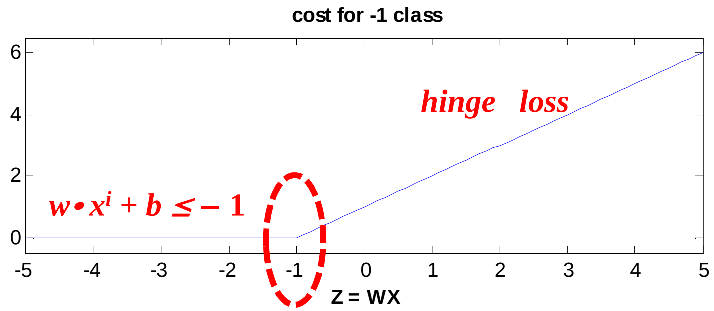


Hinge Loss and Logit

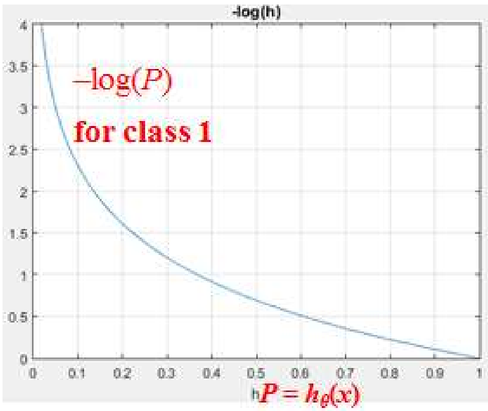
- If y = class 1, need $\theta^T x \geq \mathbf{1}$, not just $\theta^T x \geq \mathbf{0}$.
- If y = class −1, need $\theta^T x \leq \mathbf{-1}$, not just $\theta^T x \leq \mathbf{0}$.
- SVM has **<u>less</u>** influence from outliers.

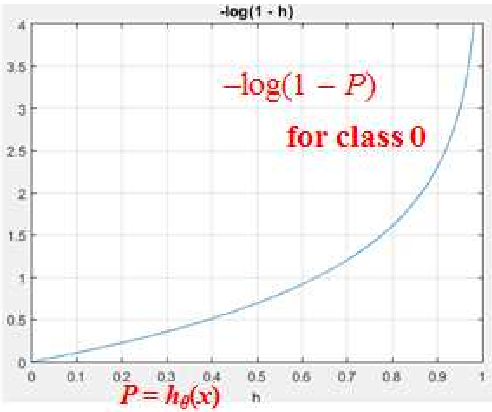
- Logistic Regression???

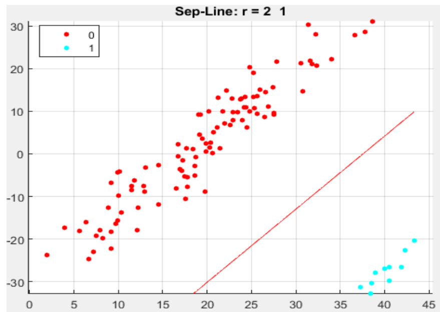


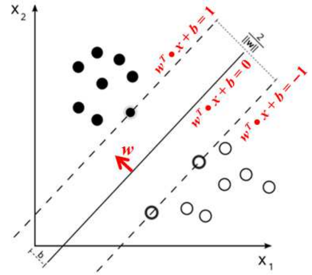


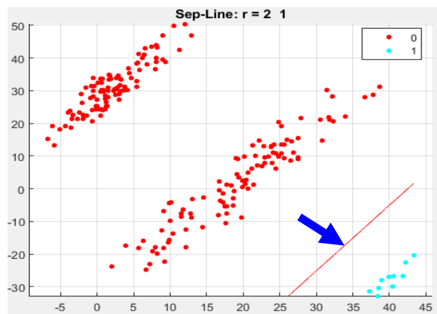
syms z, ezplot(-log(1/(1+exp(1)^(-z))), [-6 6])

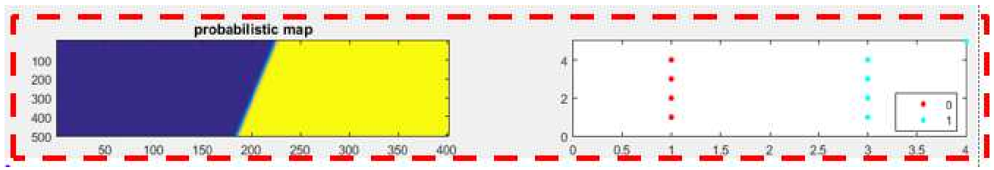




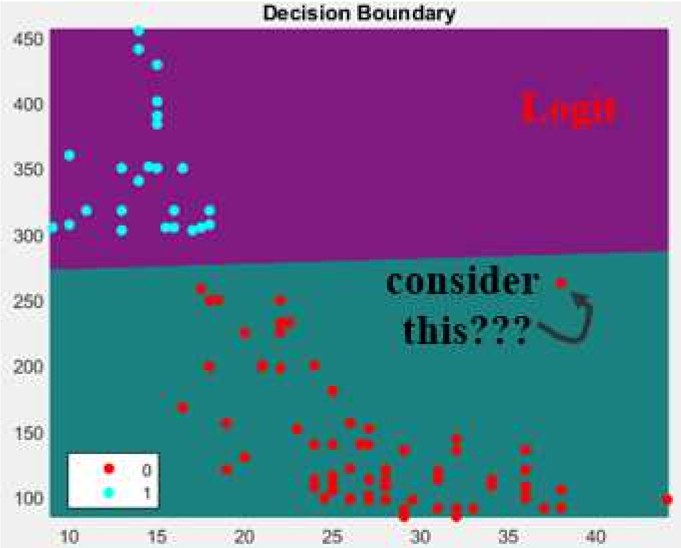






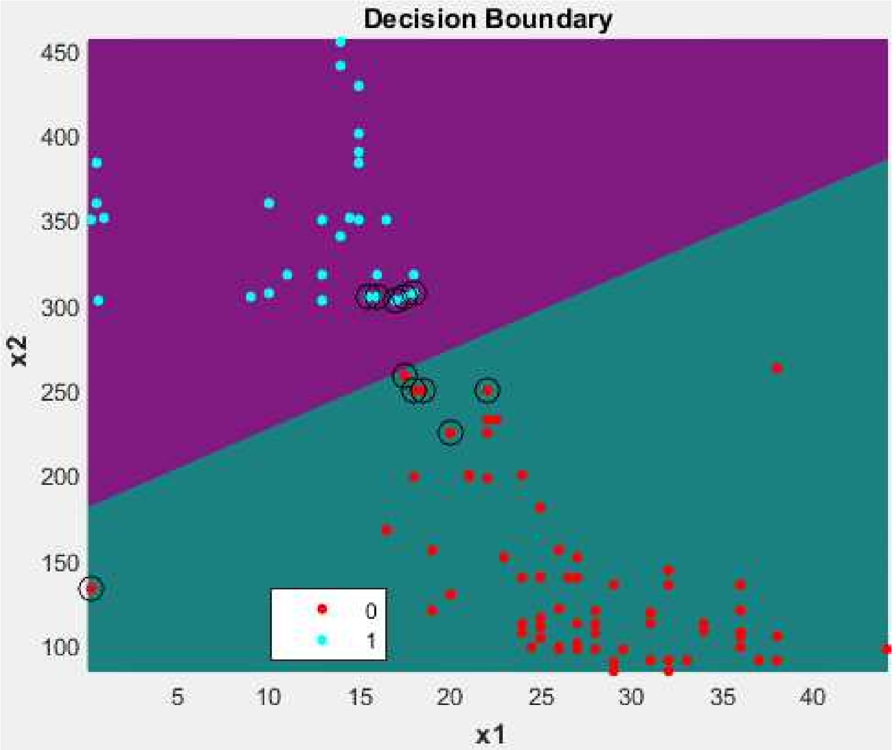


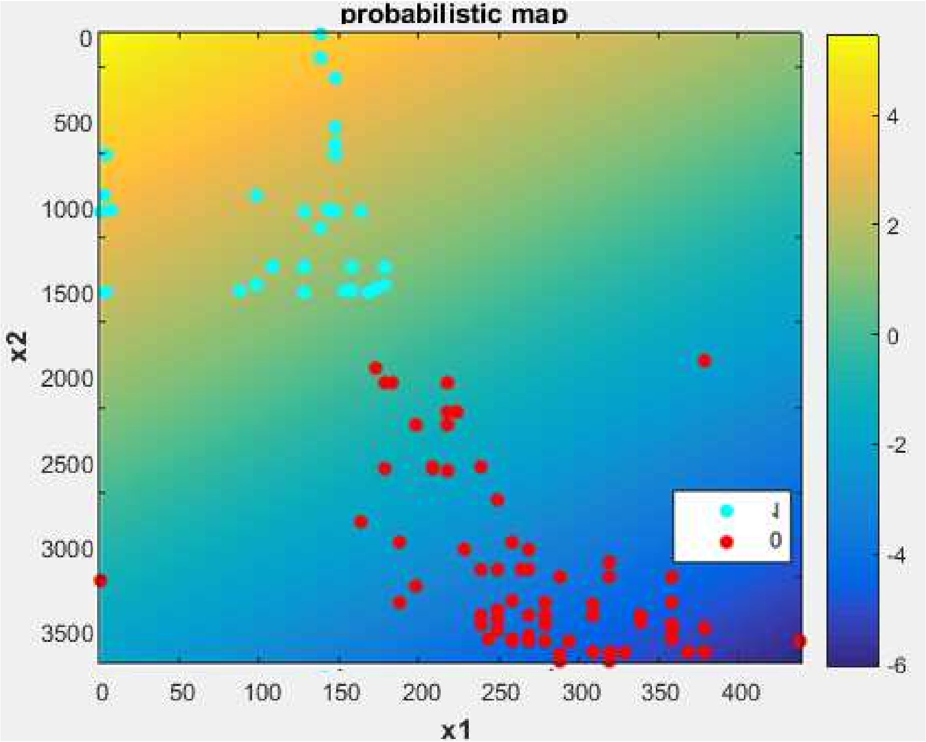
SVM, **MPG + Displacement**



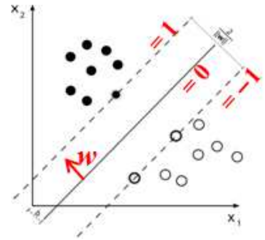
CFM = 0.95

68	0
5	27

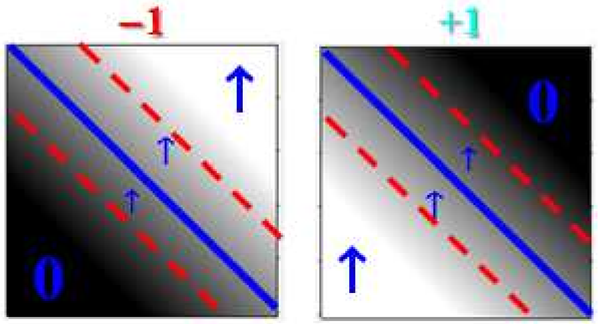




Summary– SVM Objectives



- To place data of each class on the correct side, SVM needs to…
  - **MAX** ***margin*** $= \frac{2}{||w||}$ = **MIN** $||\boldsymbol{w}||^2$ or $\sum_j^d \boldsymbol{w}_j^2$.



- Moe precisely, our objectives are:
  - **Minimize** $||w||^2$ such that…

1. $\boldsymbol{w^T x_i} + b \geq 1$ for record $\boldsymbol{i}$ with $y_i = +1$ **AND**

2. $\boldsymbol{w^T x_i} + b \leq -1$ for record $\boldsymbol{i}$ with $y_i = -1$.

- Combine above "AND" statements➔ **minimize** $[\boldsymbol{max}(\boldsymbol{0}, (1 - y_i(w^T x_i + b))]$
  - If $y_i = +1$ & $w^T x_i \approx \infty$, ➔ $\max(0,\ 1 - y_i \times w^T x_i) = \max(0, -\infty) = 0$. What if $w^T x_i =$ **0.5**??
  - If $y_i = -1$ & $w^T x_i \approx -\infty$, ➔ $\max(0,\ 1 - y_i \times w^T x_i) = \max(0, -\infty) = 0$. What if $w^T x_i =$ **−0.5**??

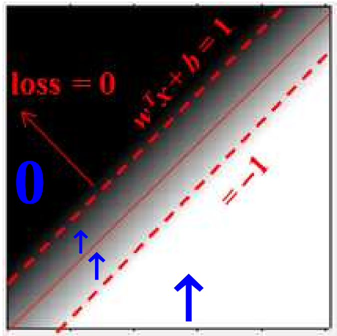
- Combine above condition **1 & 2** together as:
  - $min_w\ [\sum_{i=1}^{m}[\boldsymbol{max}(\boldsymbol{0},\ (\boldsymbol{1} - \boldsymbol{y_i} \times \boldsymbol{w^T x_i})\ ] = min_w\ [\boldsymbol{E}]$.

Combine Together… SVM Cost Function

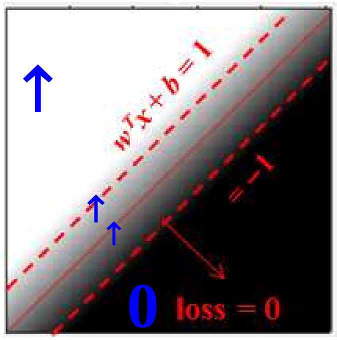
- MAX margin = $\frac{2}{||w||}$ or **minimize** $||\boldsymbol{w}||^2$.
- **minimize** $[\boldsymbol{max}(\,\boldsymbol{0},\ \ (\boldsymbol{1} - \boldsymbol{y_i} \times \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i}\,)\ \ ].$

$$min_w \sum_{i=1}^{m}[\boldsymbol{max}(\boldsymbol{0},\ \ (1 - \boldsymbol{y_i} \times \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i})\ ] + \frac{1}{2}||\boldsymbol{w}||^2$$
$$= min_w[\boldsymbol{E} + \boldsymbol{L}]$$

- $min_w \sum_{i=1}^{m}[\boldsymbol{max}(\boldsymbol{0},\ \ (1 - \boldsymbol{y_i} \times \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i})\ ] = min_w[\boldsymbol{E}].$
  - If $y_i = +1$
    - and $w^{\mathrm{T}}x_i \approx \infty$, ➔ $\max(0,\ 1 - y_i \times w^{\mathrm{T}}x_i) = \max(0, -\infty) = 0$.
    - and $w^{\mathrm{T}}x_i = 1$, ➔ $\max(0,\ 1 - y_i \times w^{\mathrm{T}}x_i) = \max(0, 0) = 0$.
    - and $w^{\mathrm{T}}x_i = 0.5$, ➔ $\max(0,\ 1 - y_i \times w^{\mathrm{T}}x_i) = \max(0, 0.5) = 0.5$.
    - and $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i} = -0.5$, ➔ $\max(0,\ 1 - y_i \times w^{\mathrm{T}}x_i) = \max(0, 1.5) = 1.5$.
    - and $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i} = -2$, ➔ $\max(0,\ 1 - y_i \times w^{\mathrm{T}}x_i) = \max(0, 3) = 3$.



  - If $y_i = -1$
    - and $w^{\mathrm{T}}x_i \approx -\infty$, ➔ $\max(0,\ 1 - y_i \times w^{\mathrm{T}}x_i) = \max(0, -\infty) = 0$.
    - and$w^{\mathrm{T}}x_i = -1$, ➔ $\max(0,\ 1 - y_i \times w^{\mathrm{T}}x_i) = \max(0, 0) = 0$.
    - and $w^{\mathrm{T}}x_i = -0.5$, ➔ $\max(0,\ 1 - y_i \times w^{\mathrm{T}}x_i) = \max(0, 0.5) = 0.5$.
    - and $w^{\mathrm{T}}x_i = +0.5$, ➔ $\max(0,\ 1 - y_i \times w^{\mathrm{T}}x_i) = \max(0, 1.5) = 1.5$.
    - and $w^{\mathrm{T}}x_i = +2$, ➔ $\max(0,\ 1 - y_i \times w^{\mathrm{T}}x_i) = \max(0, 3) = 3$.



Outline

- SVM Basic Idea, Large Margin Classifier, Support Vectors.

- SVM Model
  - Model Complexity and SVM Vector Length.
  - SVM Decision Boundary and SVM Margin.
  - SVM Hinge Loss Function vs Logistic Regression.

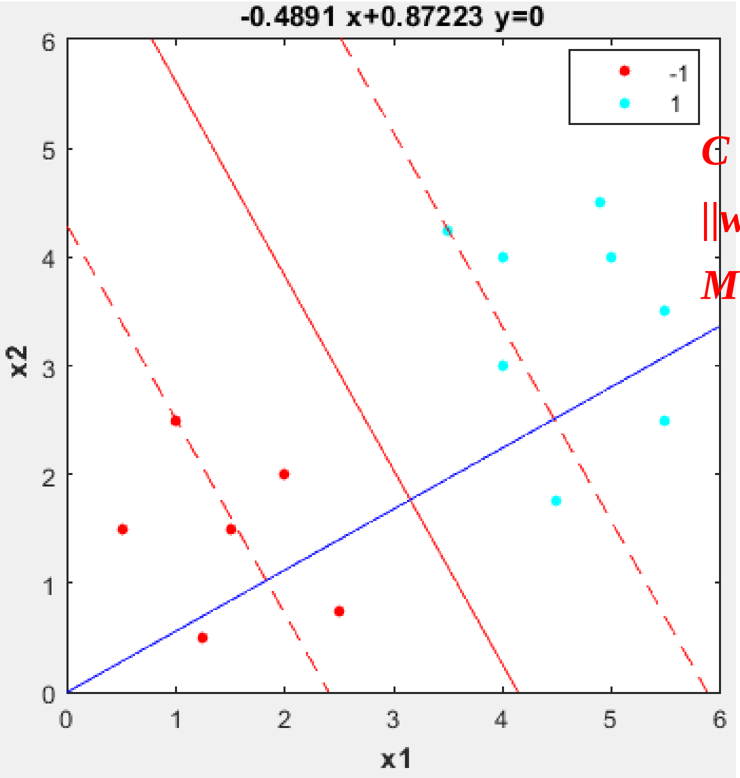
- SVM Regularization, Box Constraint, SVM Support Vector and α.

- SVM Multiclasses.

Regularization Factor C (*Box Constraint*)

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

- $min_w \; C \times \sum_{i=1}^{m}[max(0, \;\; (1 - y_i \times w^T x_i) \;]+\frac{1}{2}||w||^2 = min_w[CE + L].$
  - $C\uparrow \rightarrow$ E$\downarrow \rightarrow$ M$\downarrow \rightarrow$ Regularization$\downarrow \rightarrow$ #SV$\downarrow \rightarrow$ $w\uparrow$ (L$\uparrow$) (bias$\downarrow$, variance$\uparrow$)
  - $C\downarrow \rightarrow$ E$\uparrow \rightarrow$ M$\uparrow \rightarrow$ Regularization$\uparrow \rightarrow$ #SV$\uparrow \rightarrow$ $w\downarrow$ (L$\downarrow$) (bias$\uparrow$, variance$\downarrow$)
    - SVM_Model = **fitcsvm**(X, Y, **'BoxConstraint'**, C)
    - clf = **svm.SVC(kernel = 'linear'**, **C = 100**)

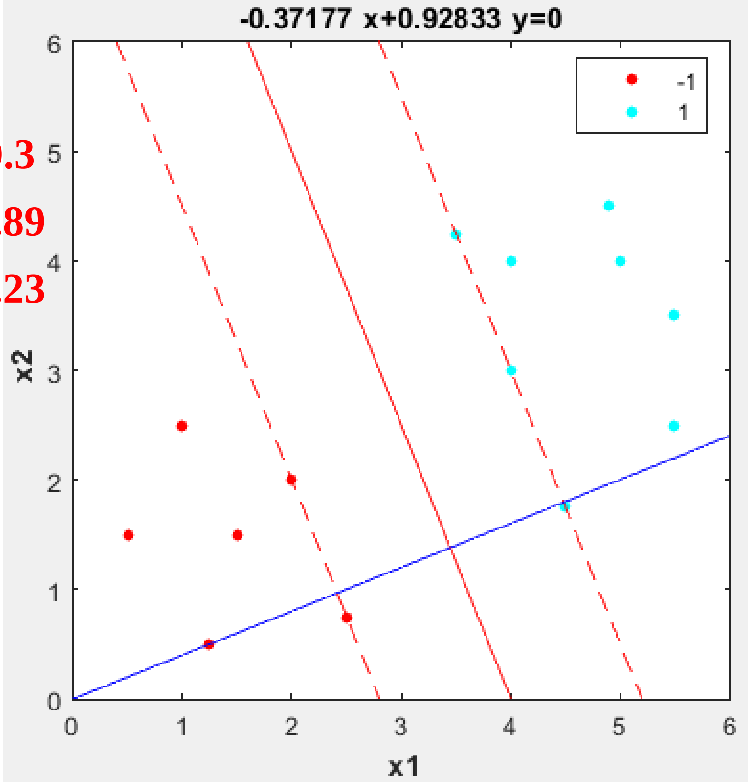


$C = 0.1\downarrow$ $\uparrow C = 0.3$

$||w|| = 0.66$ $||w|| = 0.89$

$M = 3.04\uparrow$ $\downarrow M = 2.23$

LR regularization
min(**E** + $\lambda\theta$)

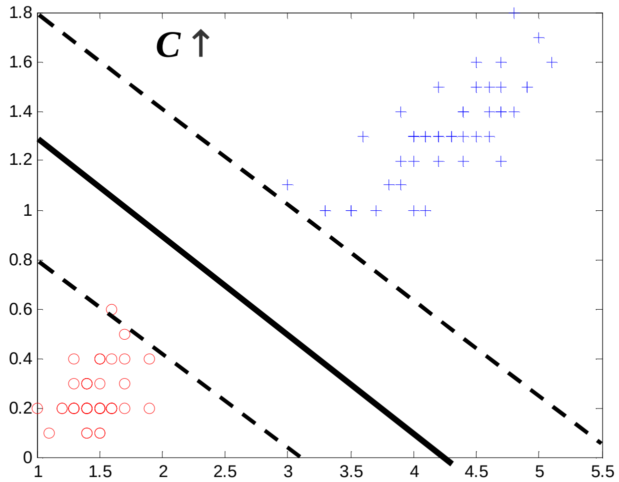


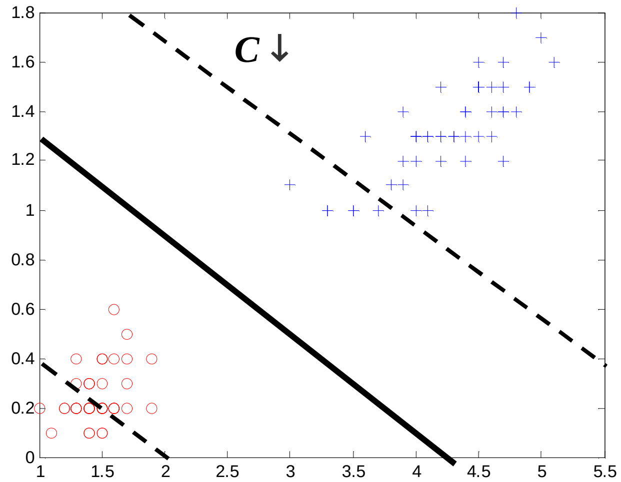
SVM Regularization

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

- $min_w\ \boldsymbol{C} \times \sum_{i=1}^{m}[\boldsymbol{max}(\mathbf{0},\ \ (1 - \boldsymbol{y}_i \times \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i)\ ] + \frac{1}{2}||\boldsymbol{w}||^{\mathbf{2}} = min_w[\boldsymbol{CE} + \boldsymbol{L}].$
- Need larger margin $\mathbf{M} = \frac{2}{||\boldsymbol{w}||}$ by choosing smaller parameter ***w*** to minimize ***L***.
- Need smaller ***E*** (error) by moving points far away from ***f***(*x*) = **±1**.

- ***C***↑ ➔ need ***E***↓ ➔ move boundary far from pts ➔ smaller margin ➔ ***L***↑.
  - **More complex model** w/ larger parameters ***w***.
- ***C***↓ ➔ ***E*** less important ➔ larger margin more error ➔ ***L***↓.
  - **Simpler model** w/ smaller parameters ***w***.





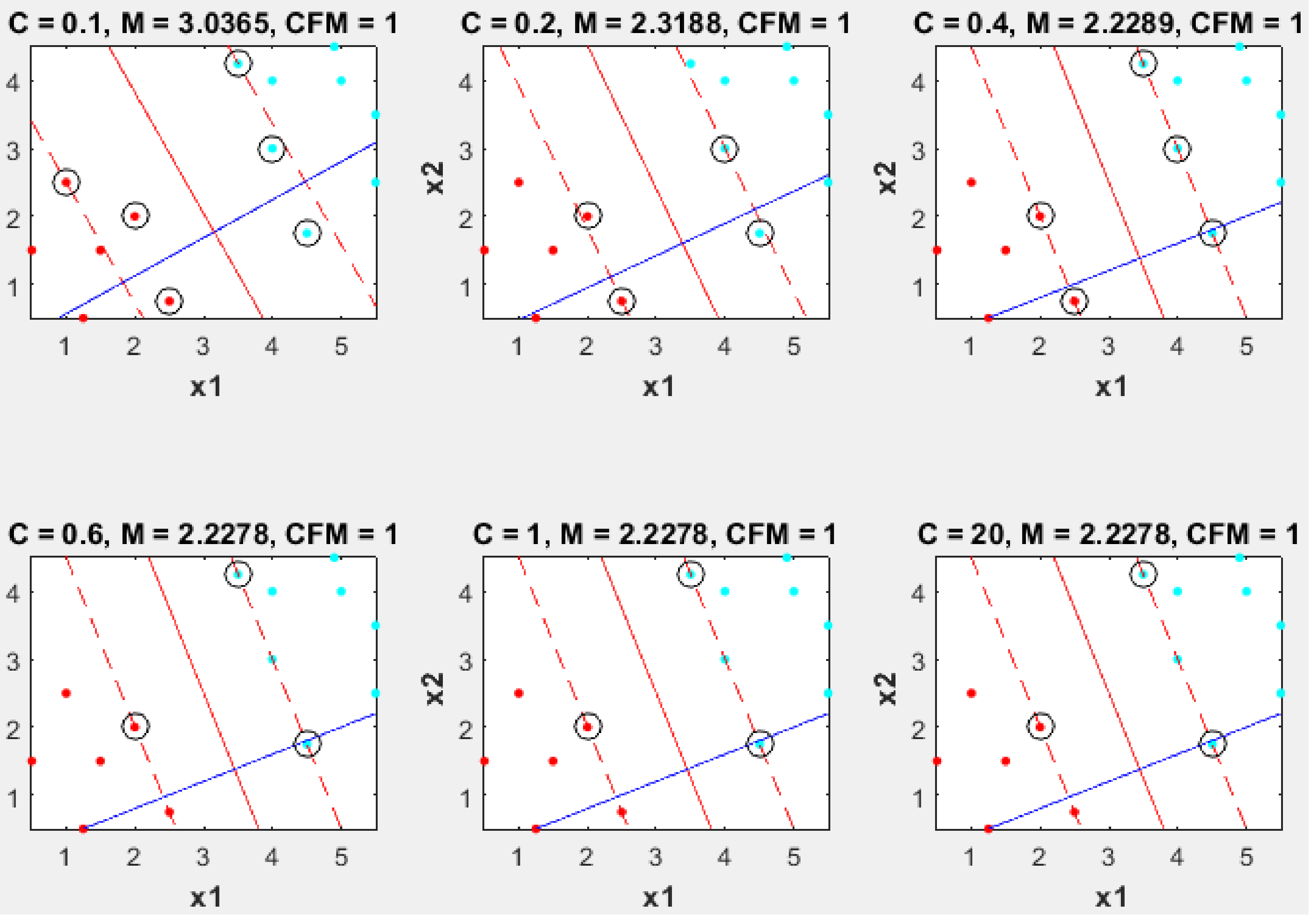
LR regularization
min(**E** + *λ**θ***)

Different C with Linearly Separable Data

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

- $min_w\ \boldsymbol{C} \times \sum_{i=1}^{m}[\boldsymbol{max}(\mathbf{0},\ \ (1 - \boldsymbol{y_i} \times \mathbf{w}^{\mathrm{T}}\boldsymbol{x_i})\ ] + \frac{1}{2}||\boldsymbol{w}||^2 \qquad = min_w[\boldsymbol{CE} + \boldsymbol{L}].$
  - $C\uparrow$ ➔ $w\uparrow$ ➔ Regularization↓ ➔ M↓ ➔ Err↓ ➔ #SV↓
  - $C\downarrow$ ➔ $w\downarrow$ ➔ Regularization↑ ➔ M↑ ➔ Err↑ ➔ #SV↑

Count # of SVs!!!



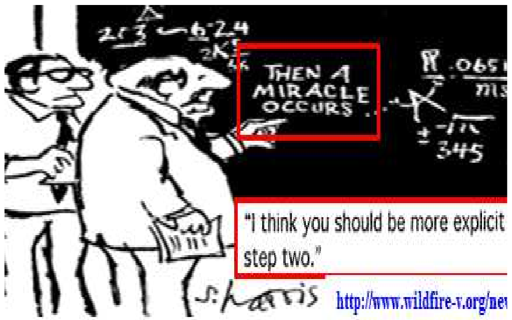
LR regularization
min($\mathbf{E} + \lambda\boldsymbol{\theta}$)

Solving Cost Function, Important Parameters, and **Miracles**

- Objective function:
  - $min_w\ \boldsymbol{C} \times \sum_{i=1}^{m}[\boldsymbol{max}(\boldsymbol{0},\ \ (1 - \boldsymbol{y_i} \times \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i})\ ] + \frac{1}{2}||\boldsymbol{w}||^{\mathbf{2}} = min_w[\boldsymbol{CE} + \boldsymbol{L}].$

- How? By solving w/ **Lagrange multiplier $\boldsymbol{\alpha_i}$** for every point $\boldsymbol{i}$:
  - **Min** $\boldsymbol{L_P} = \frac{1}{2}||w||^2 - \sum_{i=1}^{m}[\boldsymbol{\alpha_i}(1 - \boldsymbol{y_i} \times \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i}))]$ **(Primary form)** ← **sklearn**
    - Parameters include $\boldsymbol{w}$**,** and **multiplier** α**.**
  - $\frac{\partial L_P}{\partial \boldsymbol{w}} = 0$ ➔ $\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i x_i$ $\frac{\partial L_P}{\partial \boldsymbol{b}} = 0$ ➔ $\sum_{i=1}^{m} \alpha_i y_i = 0$ **[F1]**
  - Karush-Kuhn-Tucker (***KKT***) conditions: $\alpha_i \geq \mathbf{0}$ and $(1 - \boldsymbol{y_i} \times \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i}) = 0$
    - **$\alpha_i > 0$ only if x_i is critical. Also, $C \geq \alpha_i \geq 0$.**

DM Tran, p262



  - Substitute **[F1]** to primary form we get dual form as:
  - **Min** $\boldsymbol{L_D} = \frac{1}{2}\sum_{i,j} \boldsymbol{\alpha_i \alpha_j y_i y_j x_i x_j} - \sum_{i=1}^{m} \alpha_i$ **(Dual form)**
    - Parameters include ONLY multiplier α.
    - Decision boundary: $\boldsymbol{wX} + \boldsymbol{b} = 0$ ➔ $(\sum_{i=1}^{m} \alpha_i y_i x_i \mathrm{X}) + \boldsymbol{b} = 0.$

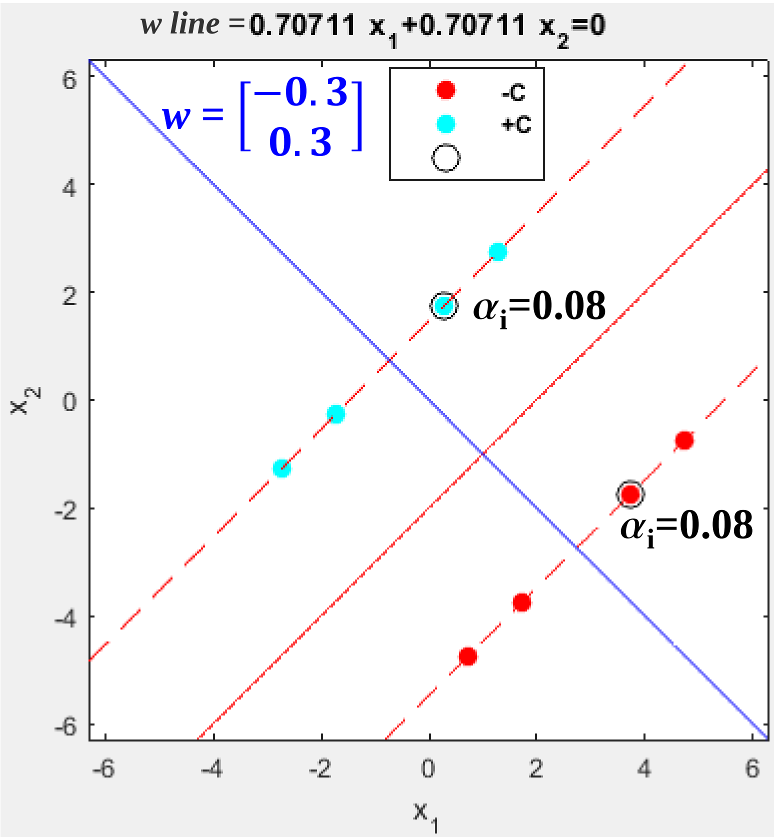
Matlab: score
Python: decision_function

Matlab SVM Simple Example with α_i

3.7321	-1.7321	-1
4.7321	-0.7321	-1
1.7321	-3.7321	-1
0.7321	-4.7321	-1
0.2679	1.7321	1
1.2679	2.7321	1
-1.7321	-0.2679	1
-2.7321	-1.2679	1

- $\alpha_i \uparrow$ ➔ ***criticalness*** to decision boundary.

$||w|| = 0.408$, M = 4.899



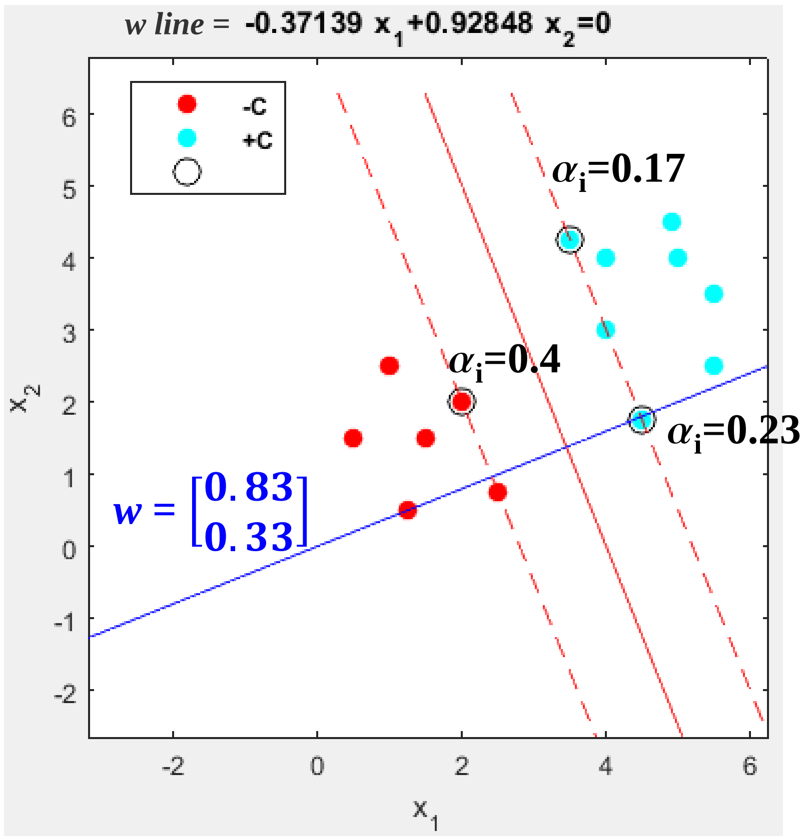
X = [3.73, 4.73, 1.73, 0.73, 0.27, 1.27, -1.73, -2.73; -1.73, -0.73, -3.73, -4.73, 1.73, 2.73, -0.27, -1.27]';
Y = [-1; -1; -1; -1; 1; 1; 1; 1];

```
SVM_M = fitcsvm(X, Y)

sv = SVM_M.SupportVectors;

theta = SVM_M.Beta;
theta0 = SVM_M.Bias;

Sep = null(theta');     % dot(theta, Sep) close to 0

w_len = norm(theta);     margin = 2 / w_len

SVM_M.IsSupportVector
SVM_M.SupportVectorLabels
SVM_M.Alpha
```

Matlab SVM Example w/ α_i ←→ **Python dual_coef_**

- α_i↑ ➔ ***criticalness*** to decision boundary.

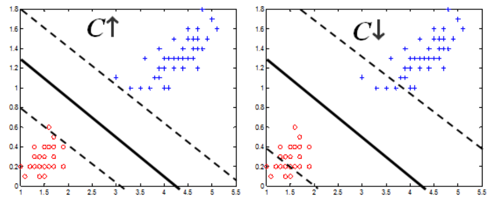
$||w||$ = 0.90, M = 2.23




```
X = [3.5 4  4  4.5  4.9  5  5.5  5.5  0.5  1  1.3   1.5  2  2.5
4.3  3   4   1.8  4.5   4  2.5  3.5  1.5  2.5  0.5  1.5   2  0.8]';
Y = [1   1   1   1   1  1   1   1  -1   -1  -1   -1  -1  -1 ]';

SVM_M = fitcsvm(X, Y)

sv = SVM_M.SupportVectors;

theta = SVM_M.Beta;
theta0 = SVM_M.Bias;

Sep = null(theta');     % dot(theta, Sep) close to 0

w_len = norm(theta);
margin = 2 / w_len

SVM_M.IsSupportVector
SVM_M.SupportVectorLabels
SVM_M.Alpha
```

Python code on slide 25, 26

Parameters, Relationships $min_{w,b}[\boldsymbol{CE} + \boldsymbol{L}]$

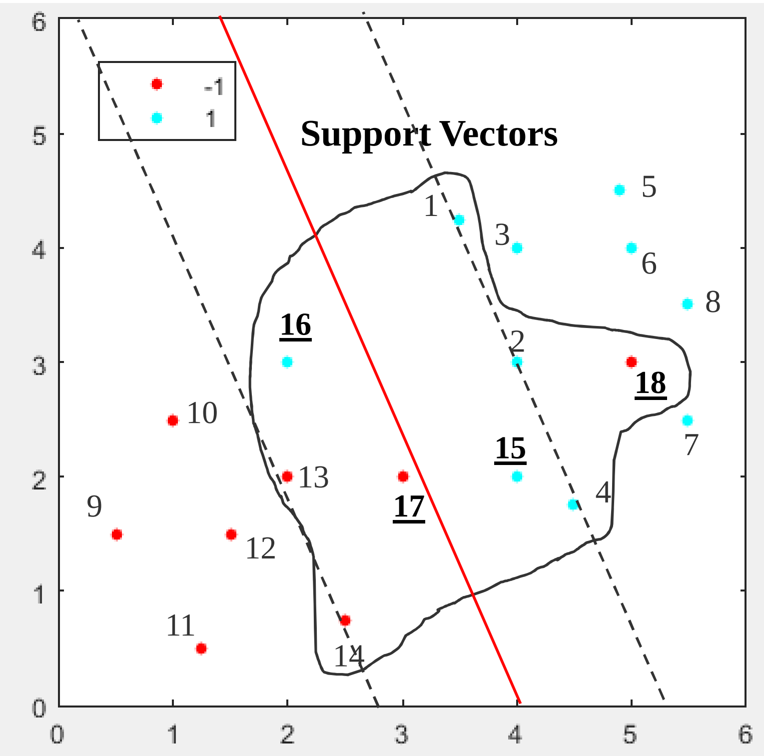


- Penalty parameter (box constraint) $\boldsymbol{C} > 0$.
  - C↑ ➔ w↑ ➔ Regularization↓ ➔ M↓ ➔ Err↓ ➔ #SV↓ ➔ training time↑
    - Matlab function: fitcsvm(X, Y, 'Standardize', 1, 'BoxConstraint', **C**) Sparsity in $\boldsymbol{w}$ (or θ)

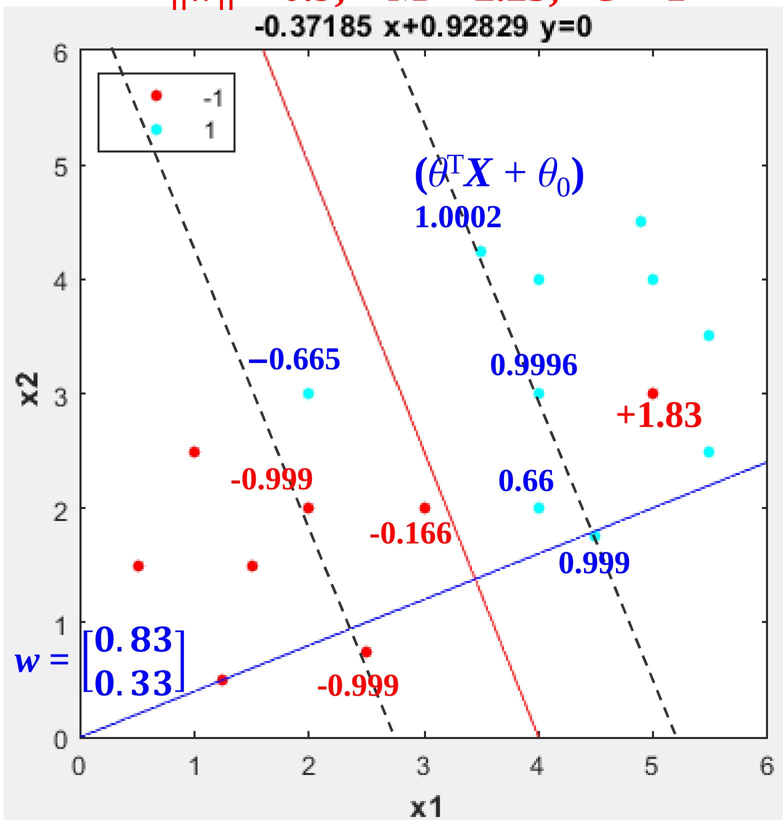
- With **additional constraints** $\boldsymbol{C \geq \alpha_i \geq 0}$. ($\alpha_i = 0$ ➔ non-support vector)
  - α_i↑ ➔ ***criticalness*** to decision boundary (???).

C = 1

	α_i
1	0.0423
2	0.1936
4	1
13	1
14	0.2359
15	1
16	1
17	1
18	1



||w|| = 0.9, M = 2.23, C = 1

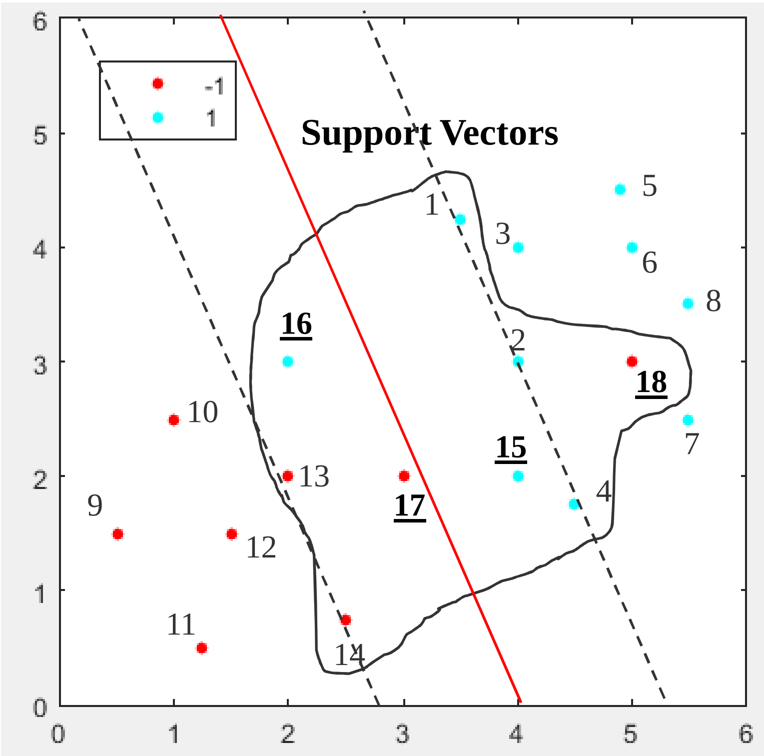


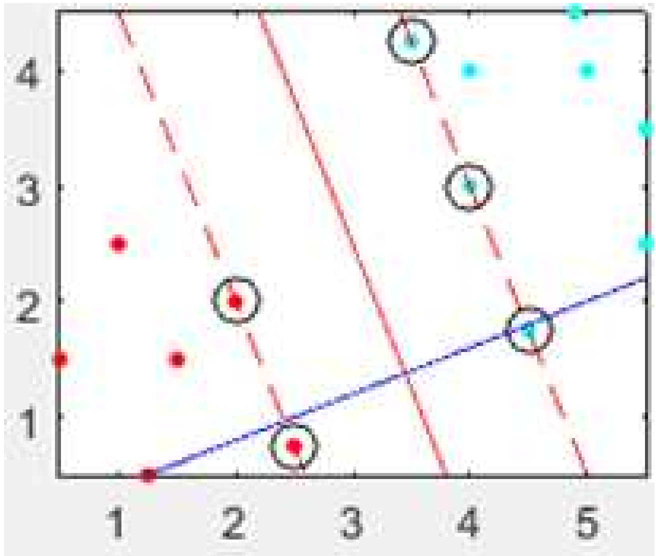
Hard Margin, Soft Margin, Support Vectors

- Extending the definition of support vectors…

***C* = 1**

	α_i
1	0.0423
2	0.1936
4	1
13	1
14	0.2359
15	1
16	1
17	1
18	1

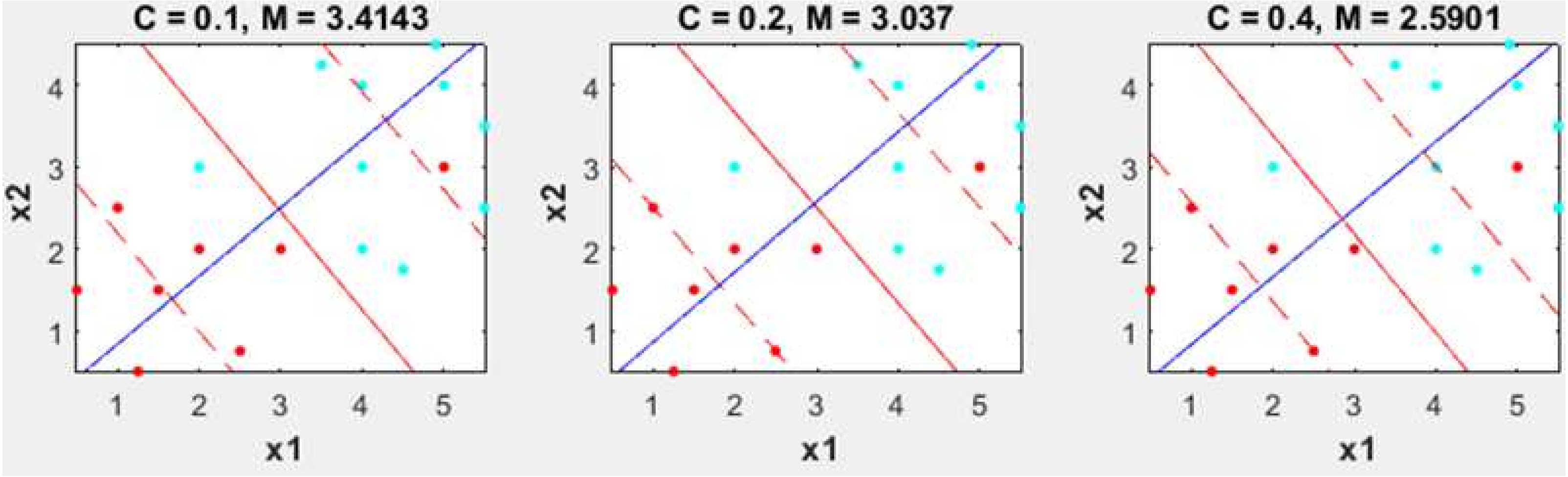


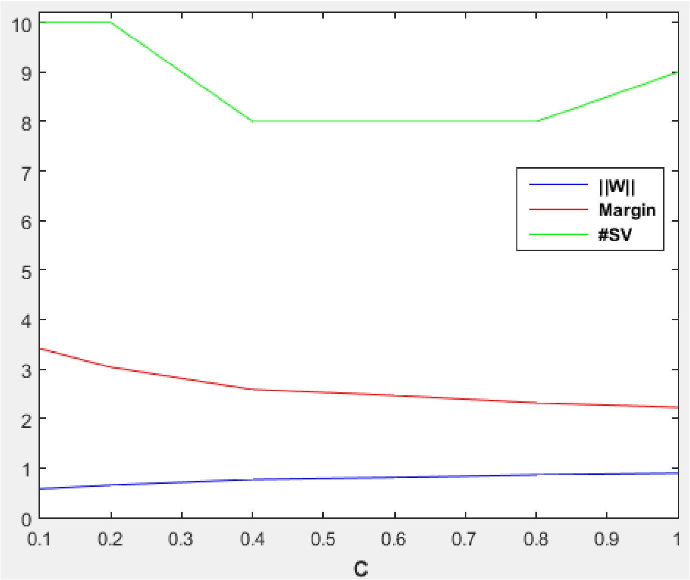


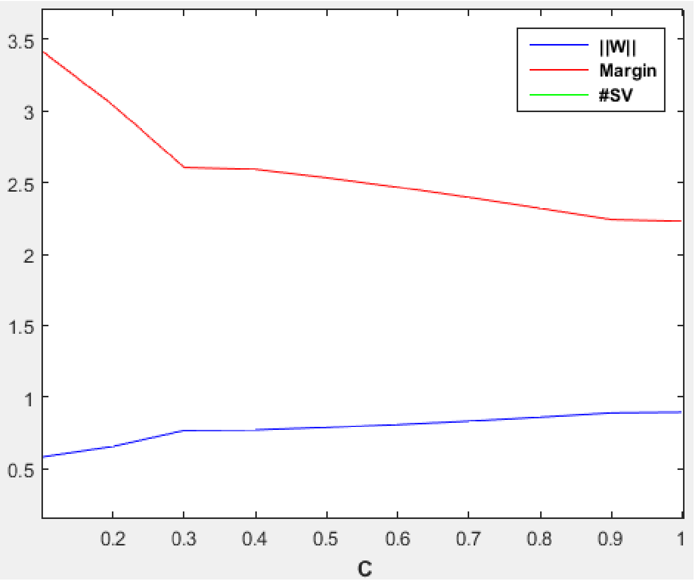
How *C* Impacts ||*W*||, Margin, and # of SVs $\quad min_{w,b}[\boldsymbol{CE} + \boldsymbol{L}]$

- fitcsvm(X, Y, 'Standardize', 1, 'BoxConstraint', **C**) **C↑ ➔ w↑ ➔ Regularization↓ ➔ M↓ ➔ Err↓ ➔ #SV↓**

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



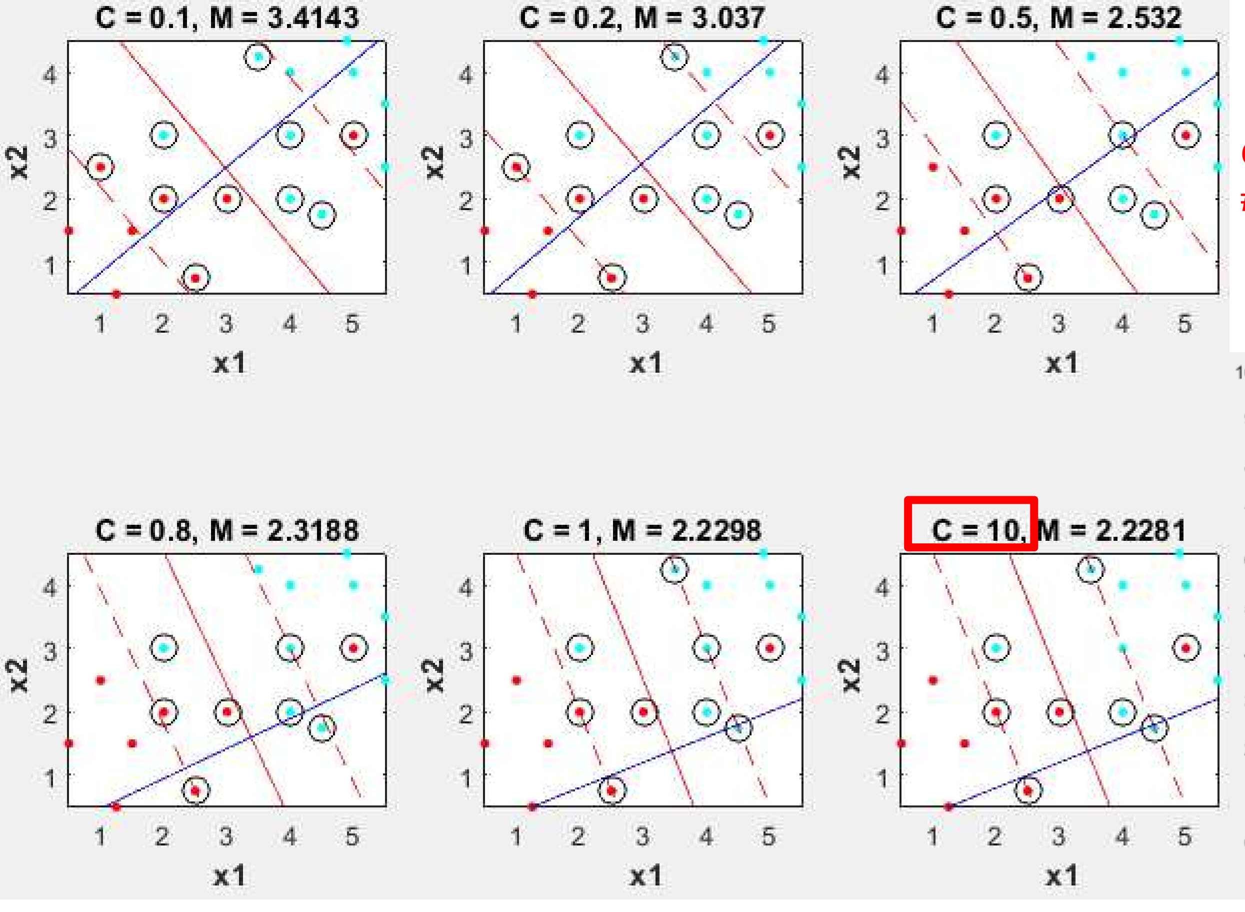




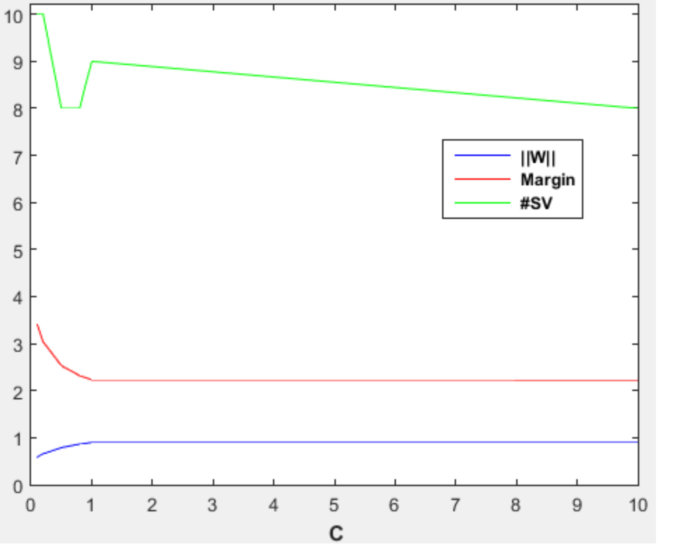
$C\uparrow$ ➔ $w\uparrow$ ➔ Regularization↓ ➔ M↓ ➔ Err↓ ➔ #SV↓

- Notice the orientation of the decision boundary is also changing.
  - $C\uparrow$ ➔ $w\uparrow$ ➔ Regularization↓ ➔ M↓ ➔ Err↓ ➔ #SV↓
  - $C\downarrow$ ➔ $w\downarrow$ ➔ Regularization↑ ➔ M↑ ➔ Err↑ ➔ #SV↑

$$min_{w,b}[\boldsymbol{CE} + \boldsymbol{L}]$$

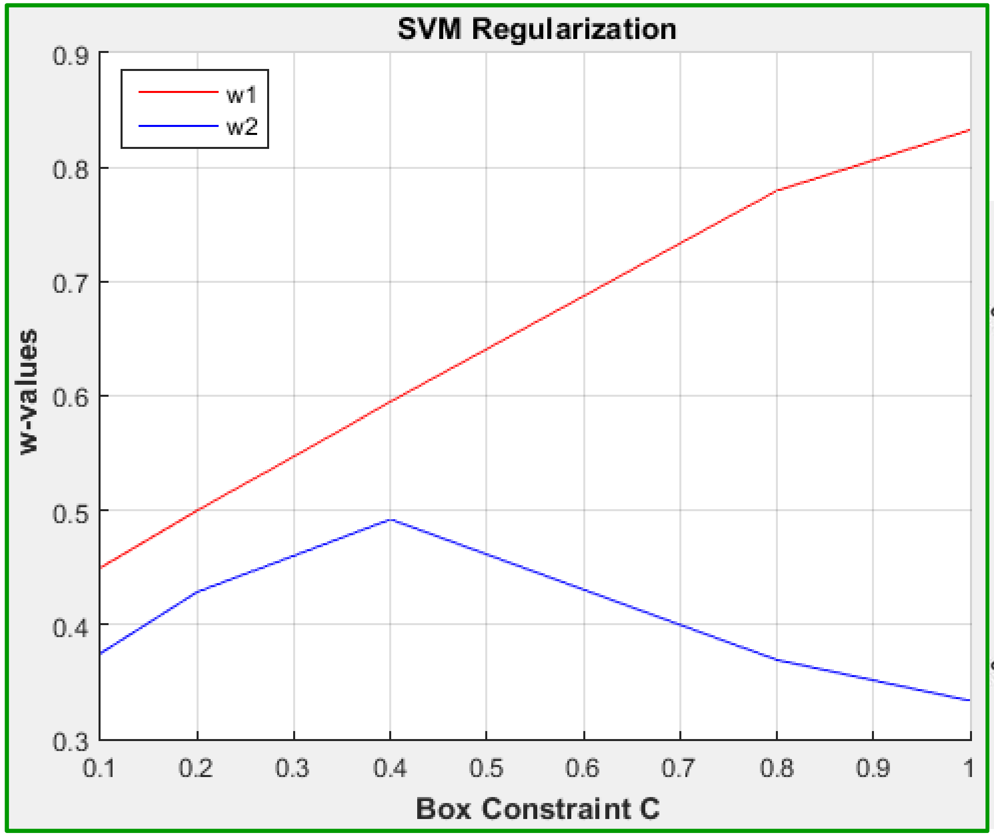


Count
# of SVs!!!

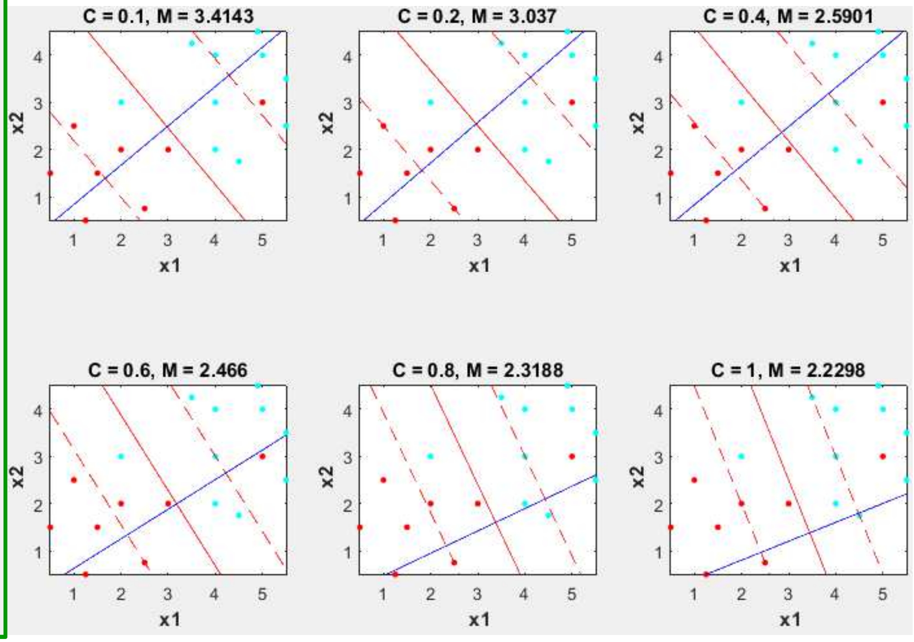


Visualizing SVM Regularization

- SVM Regularization is controlled by the Box Constraint C.
  - $C\uparrow \Rightarrow w\uparrow \Rightarrow$ Regularization$\downarrow \Rightarrow$ M$\downarrow \Rightarrow$ Err$\downarrow \Rightarrow$ #SV$\downarrow$
  - $C\downarrow \Rightarrow w\downarrow \Rightarrow$ Regularization$\uparrow \Rightarrow$ M$\uparrow \Rightarrow$ Err$\uparrow \Rightarrow$ #SV$\uparrow$
  - More like ridge regularization in which ***w***-values closer to 0s, but $\neq$ 0s.

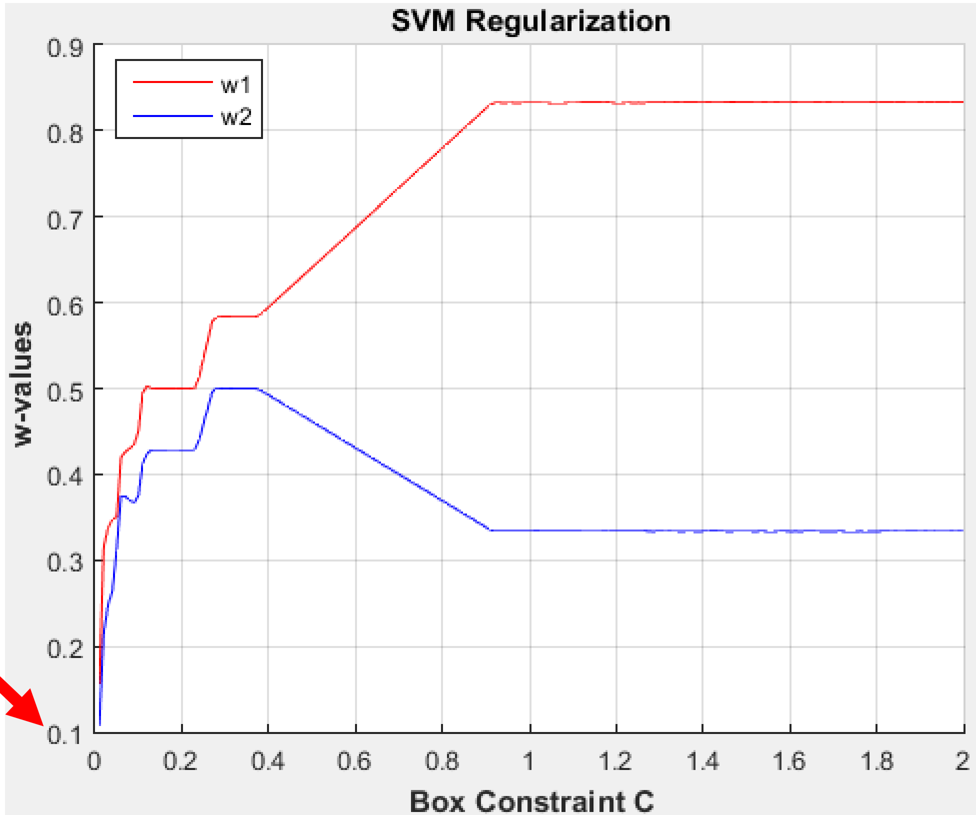


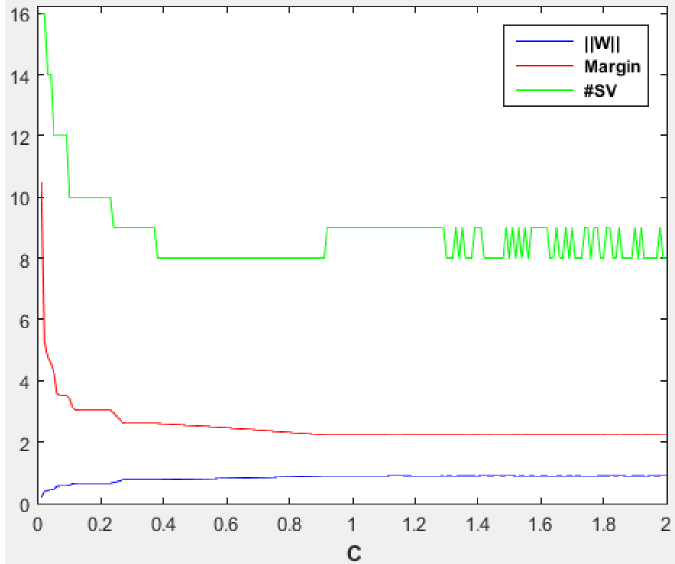
$$min_{w,b}[\boldsymbol{C}\boldsymbol{E} + \boldsymbol{L}]$$

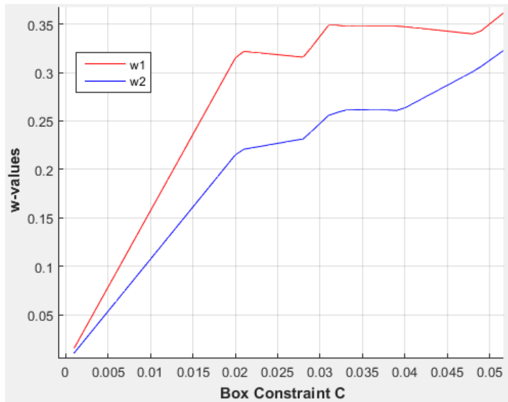


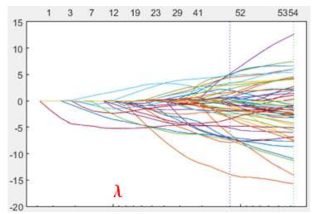
Visualizing SVM Regularization with More *C*-Values

- SVM Regularization is controlled by the Box Constraint C. $min_{w,b}[\boldsymbol{C}\boldsymbol{E} + \boldsymbol{L}]$
  - C↑ ➔ w↑ ➔ Regularization↓ ➔ M↓ ➔ Err↓ ➔ #SV↓
  - C↓ ➔ w↓ ➔ Regularization↑ ➔ M↑ ➔ Err↑ ➔ #SV↑
  - More like **[ridge]** regularization in which ***w***-values closer to 0s, but ≠ 0s. **(see y-axis)**

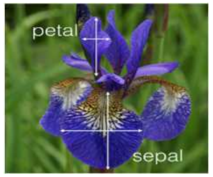








SVM + Regularization: Iris Example (using var 3 and 4)



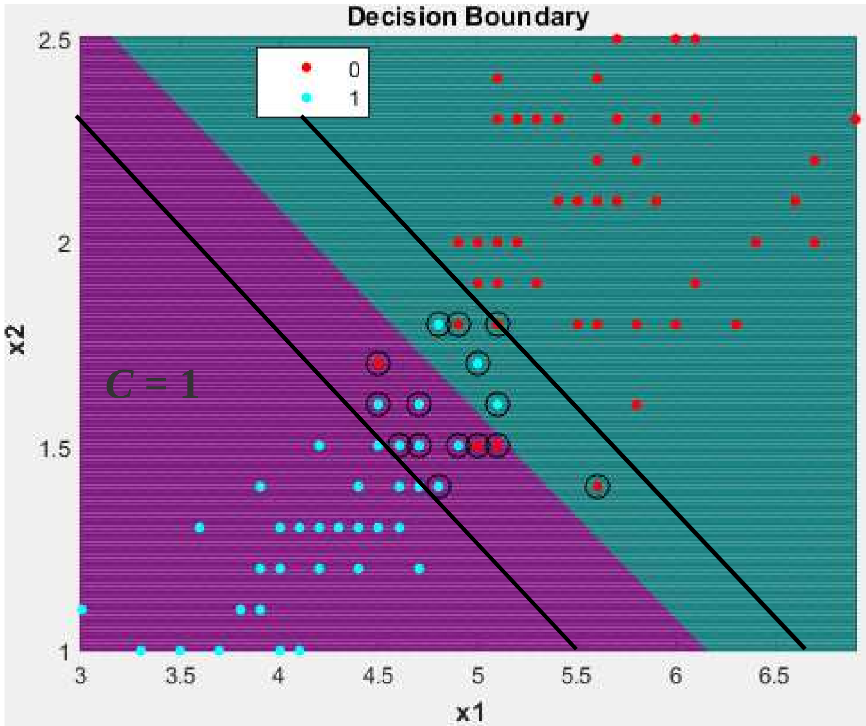
http://sebastianraschka.com/Articles/2014_python_lda.html

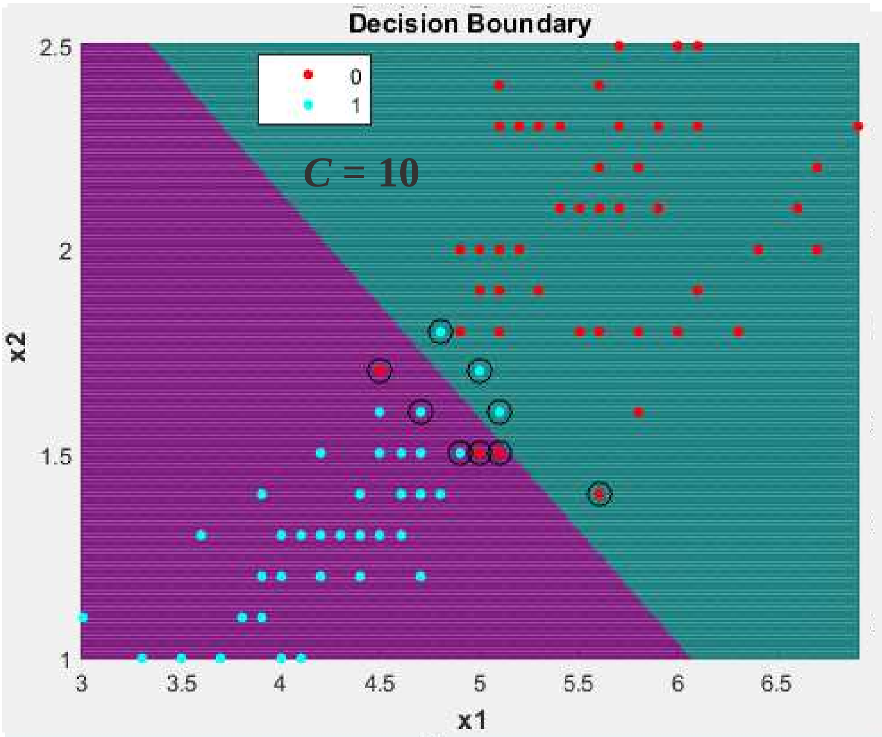
- Box constraint $C > 0$.
  - $C\uparrow \Rightarrow w\uparrow \Rightarrow$ **Regularization↓ ➔ M↓ ➔ Err↓ ➔ #SV↓ ➔ training time ↑**

```
load fisheriris
X = meas(51:end, 3:4);
Y = strcmp('versicolor',species(51:end));     % create BINARY class
svm_mdl = fitcsvm(X, Y, 'Standardize', true, 'BoxConstraint', 2)
[labels score] = predict(svm_mdl , X);
```

$$min_{w,b}[CE + L]$$

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

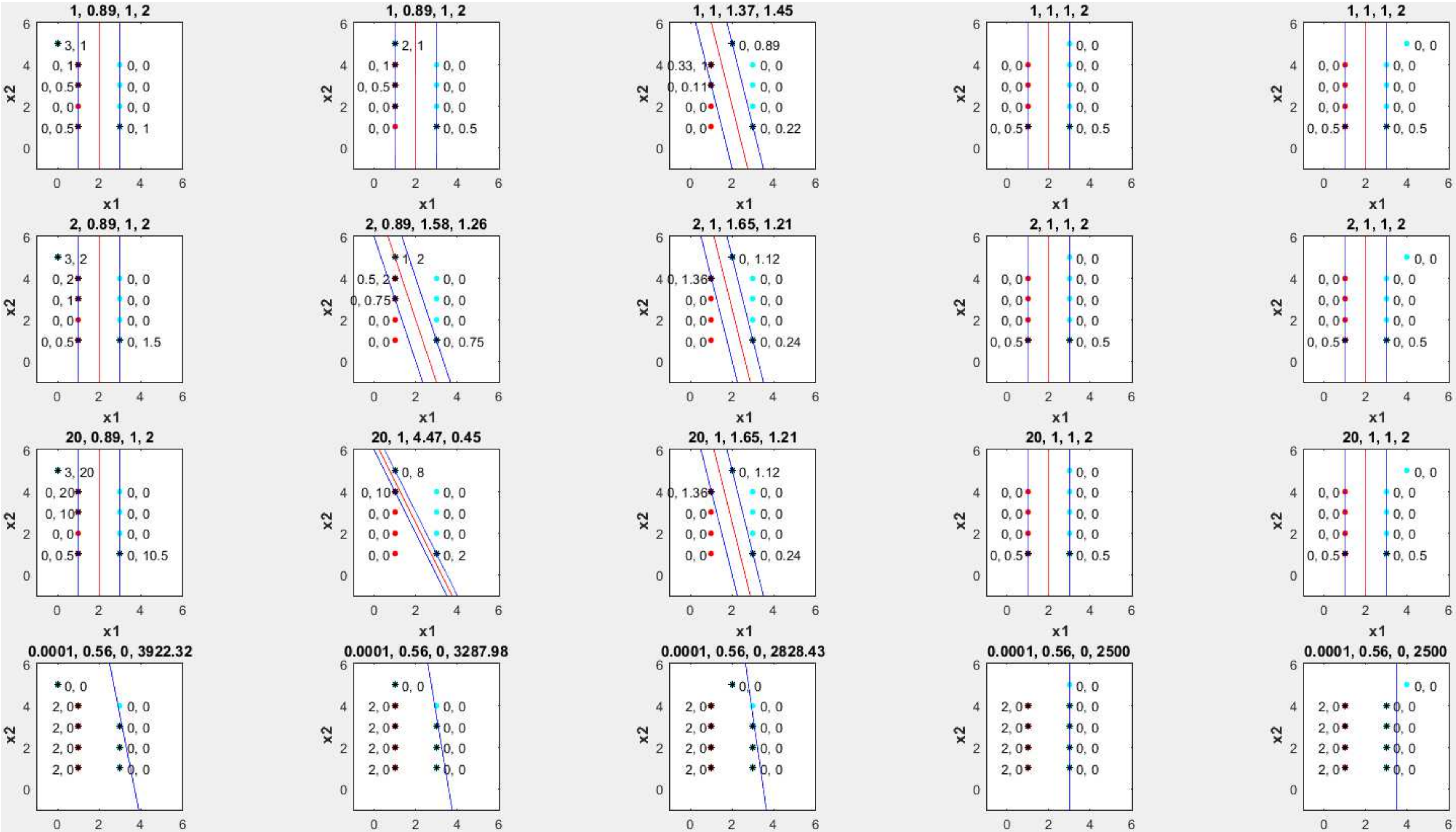




Moving One Point (with Y = 1) Across Decision Boundary

- Title = [C, Accuracy, ||w||, Margin].
- Each point = [~~slack~~, α].

$$min_{w,b}[CE + L], \quad C \geq \alpha_i \geq 0$$



Speed Performance

MDL-01 2018S, Bank Marketing

Terrence White, Ronald E Twite
Leela Sowjanya Chippada, Nathan Adams
Ahmad K Lubnani, Mowlid Abdillahi

Training						
KernelScale	.1	**1**	10	1	1	1
BoxConstraint	1	**1**	1	.1	10	100
SVM	444 sec	**33 sec**	5 sec	7 sec	234 sec	454 sec
SVM with lasso	353 sec	**14 sec**	3 sec	5 sec	99 sec	349 sec

Market Basket Analysis, DM-01, 18S

Nikhil Dama, Semere Tekleab, Deepika Ravikumar
Hughbert Kumwesiga, Ashenafi Darihun, Neo Ce Tao

Model	Decision Tree	Random Forest	Naive Bayes	KNN	Logistic Regression	MLP Classifier	Gradient Boosting Classifier
Train Time	**10.85282135**	**27.07999134**	**1.410721064**	**177.724442**	**51.38662076**	**110.8316162**	**283.5447896**
ROC-AUC Score	0.811	0.813	0.8	0.788	0.809	0.813	0.814
F1	0.734	0.735	0.649	0.719	0.729	0.734	0.737
Accuracy	0.736	0.737	0.704	0.72	0.733	0.737	0.737
Precision	0.738	0.738	0.794	0.72	0.737	0.74	0.737
Recall	0.73	0.732	0.549	0.718	0.722	0.729	0.738
Log Loss	9.115	9.089	10.234	9.669	9.235	9.092	9.072
Confusion Matrix	Decision Tree	Random Forest	Naive Bayes	KNN	Logistic Regression	MLP Classifier	Gradient Boosting Classifier
TN, FP	[[123183 42868]	[[123102 42949]	[[142512 23539]	[[119954 46097]	[[123438 42613]	[[123562 42489]	[[122404 43647]
FN, TP	[44624 120852]]	[44292 121184]]	[74690 90786]]	[46710 118766]]	[46032 119444]]	[44782 120694]]	[43433 122043]]

Remember to Perform ***k***-Fold Cross Validation

- crossval()

- kFoldLoss()

- kfoldPredict()

After Build SVM with Cross-Validation, How To Do Prediction??

SVMModel = fitcsvm(Predictor, Interest, 'KernelFunction', 'rbf', **'Crossval', 'on'**, 'Standardize', true);
[label, score] = predict(SVMModel, X);

% however, an error occurs as ➔

```
Undefined function 'predict' for input arguments of type
'classreg.learning.partition.ClassificationPartitionedModel'.

Error in a6 (line 10)
[label, score] = predict(SVMModel, X);
```

%% WHY??

Since you built a cross-validation (by default 10 folds). You have 10 models.

If you want to predict new data, you have to specify which model you want to use. For example:

[label, score] = predict(SVMModel.Trained**{3,1}**, X);

Or you can use your 10 cross-validation models to evaluate the prediction of your **"Non-Training Data"** by calling

[label, score] = **kfoldPredict**(SVMModel);

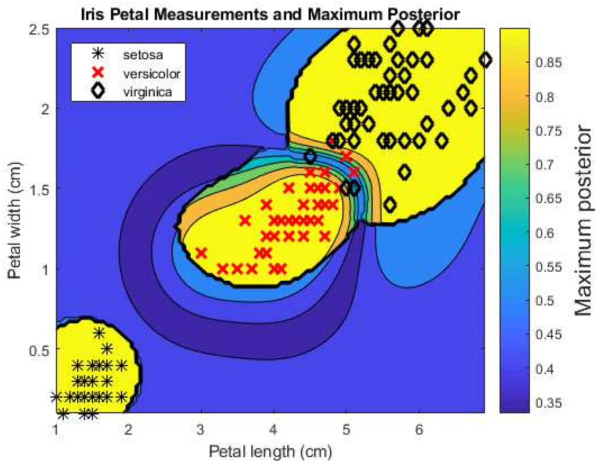
```
# http://scikit-learn.org/stable/modules/cross_validation.html
from sklearn.model_selection import cross_val_score
clf = svm.SVC(kernel='linear', C=1)
scores = cross_val_score(clf, iris.data, iris.target, cv=5)
```

Multiple Classes

- CVMdl = **fitcecoc**(x, y, 'CrossVal', 'on');

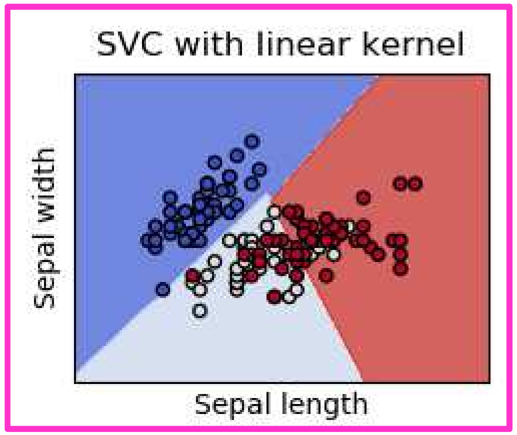
- CompactSVMModel = CVMdl.Trained{1};

- **http://www.mathworks.com/help/stats/fitcecoc.html**
  - **One vs. one**

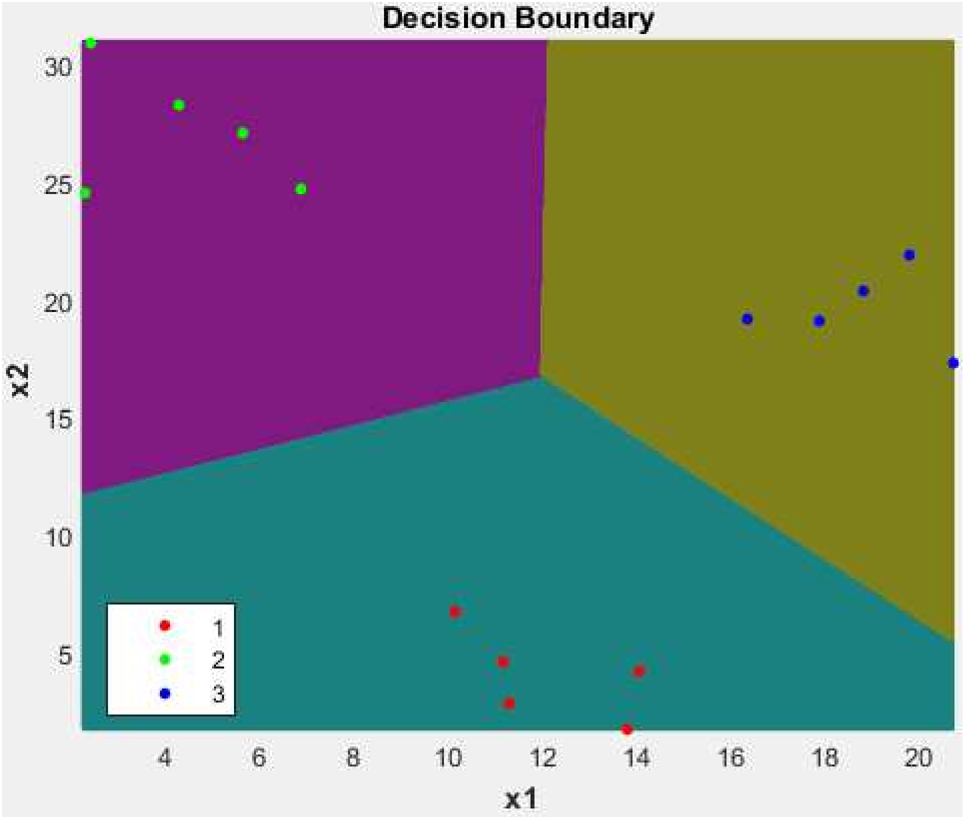


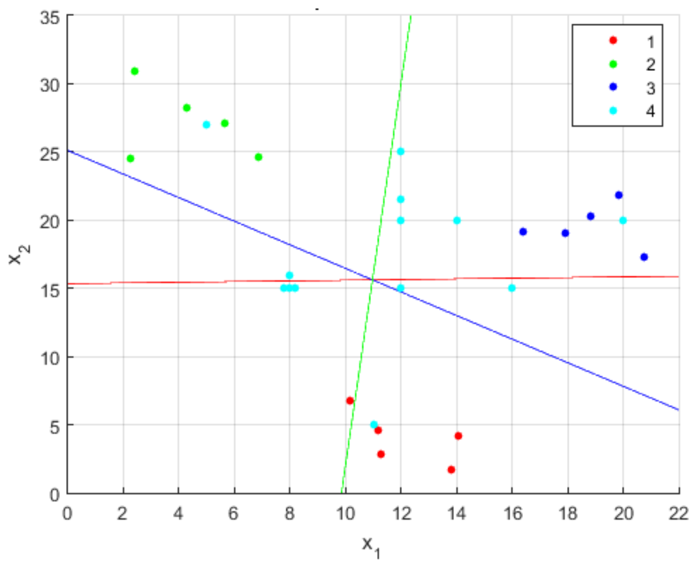
- **clf = svm.SVC(decision_function_shape = 'ovo')** **# one vs. one**

- **clf = svm.SVC(decision_function_shape = 'ovr')** **# one vs. reset**



SVM, Multiple Classes





Test Data

x1	x2	Class	
11	5	1	1
5	27	2	2
20	20	3	3
14	20	3	3
12	15	3	**1**
12	20	3	3
12	21.5000	3	3
16	15	3	3
12	25	3	3
7.8000	15	1	1
8	15	1	1
8.2000	15	1	1
8	16	2	2

Linear and Quadratic

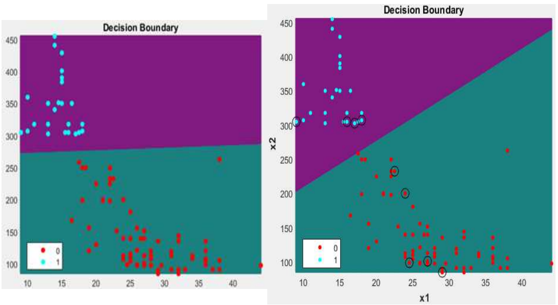
- Matlab function.
  - fitcsvm(X, Y, 'Standardize', 1, 'BoxConstraint', 0.5, 'KernelFunction', 'polynomial', 'PolynomialOrder', 3)

- NOT going to spend too much time on this.
  - Why?
  - Going to **∞–dimension** can find more effective separation.

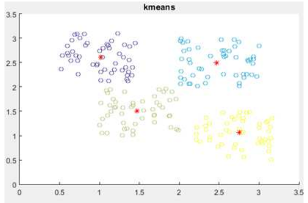
Linear SVM Summary

- SVMs produces large margin separating hyperplane, and efficient in high-D.
  - Use ***regularization*** to find the lowest-normed vector (margin) that separates data.
  - Hard-margin SVM will find a hyperplane that separates ALL data (if possible).
  - Soft-margin SVMs (generally preferred) do better when there's noise data.

- SVM maximizes the **margin** between points closest to the boundary.
  - SVMs only consider points near the margin (support vectors).
  - In SVM, **only few points** near the decision boundary really make a difference.
  - Because of this, SVM is slightly more robust.

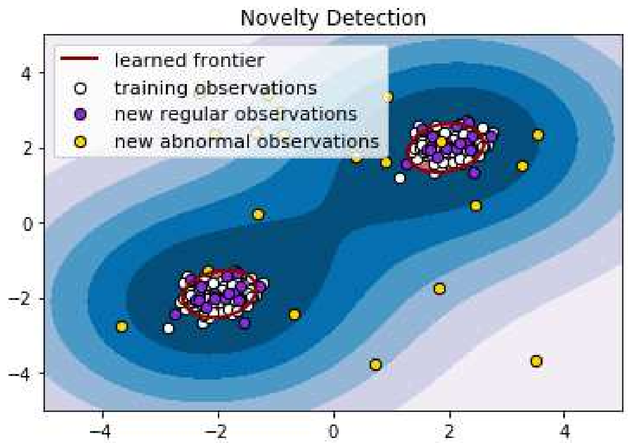


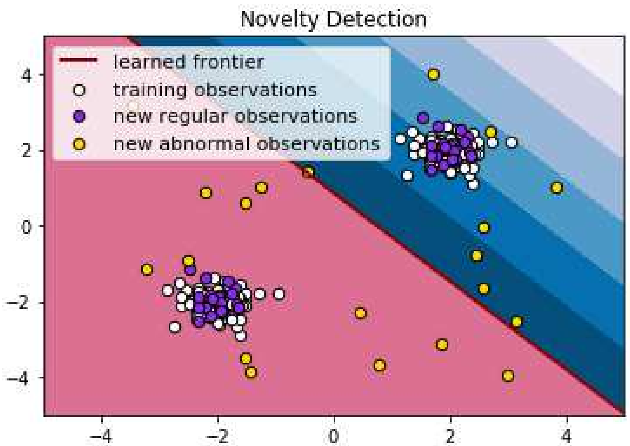
- Logistic regression finds a model to maximize the **likelihood** of the training data.
  - Logistic regression considers **ALL** the points in the data set.
  - i.e. **EVERY** point has a certain influence on the final model.

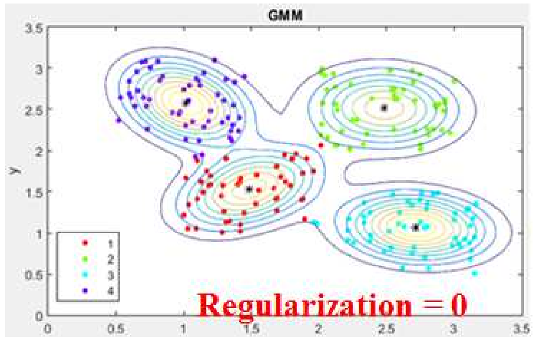


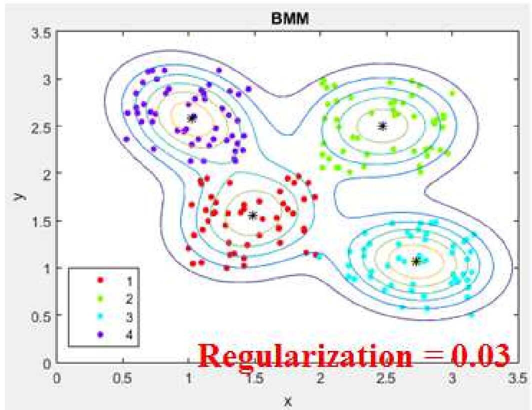
SVM One Class Classification

- **Matlab fitcsvm( )** **or** **Sklearn svm.OneClassSVM( )**
  - An unsupervised method to learn a decision function for novelty detection.
  - Classify new data as how similar or how different to the training set.
  - **Like clustering???** Compare either to ***k***-means or ***GMM*** (Gaussian Mixture Model).
  - http://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html#sklearn.svm.OneClassSVM
  - http://scikit-learn.org/stable/auto_examples/covariance/plot_outlier_detection.html#sphx-glr-auto-examples-covariance-plot-outlier-detection-py
  - http://scikit-learn.org/stable/auto_examples/svm/plot_oneclass.html







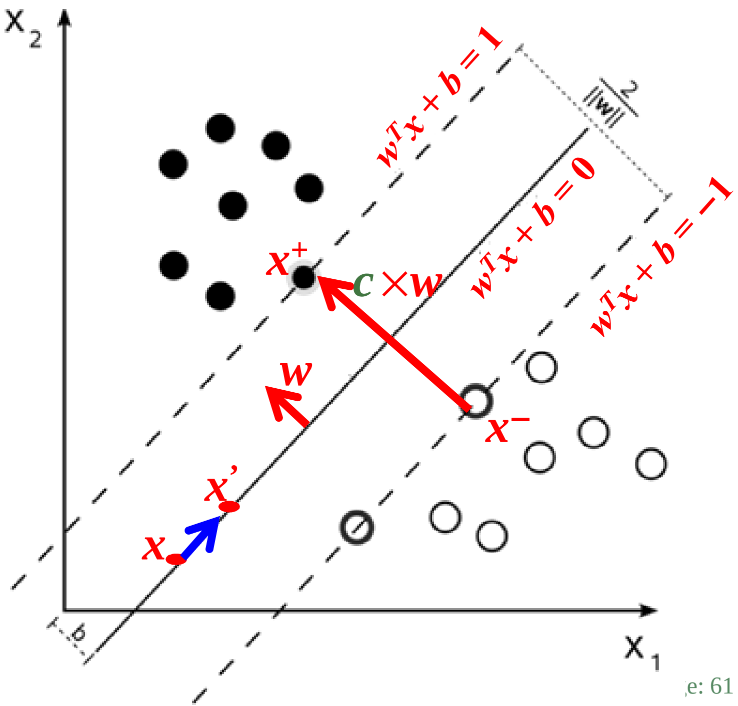



```
# kernel must be one of 'linear', 'poly', 'rbf', 'sigmoid'
clf = svm.OneClassSVM(nu=0.1, kernel="rbf", gamma=0.1)
clf.fit(X_train);        y_pred_test = clf.predict(X_test)
```

Appendix

SVM and Its Margin = $\frac{\mathbf{2}}{||\boldsymbol{w}||} = \frac{2}{\sqrt{w^T w}}$

- If we know vector ***w*** and 3 hyperplanes, then few facts…
  - The length of vector ***w*** = $||w|| = \sqrt{w^T w} = \sqrt{w_1^2 + w_2^2} = \sqrt{\sum_j^d w_j^2}$.   or $||w||^2 = \mathbf{w^T} \times \boldsymbol{w}$
  - $x^+ = x^- + mw$ ➔ $x^+ - x^- = cw$.
  - $w^T x + b = 0$ and $w^T x' + b = 0$ ➔ $w^T(x' - x) + b = 0$.   (perpendicular)
  - $wx^+ + b = 1$ since x^+ is on $w^T x + b = 1$. $wx^- + b = -1$ since x^- is on $w^T x + b = -1$.
  - $w^T x^+ + b = 1$ ➔ $w^T(x^- + cw) + b = 1$ ➔ $c||w||^2 + \boldsymbol{w^T x^- + b} = 1$ ➔ $c||w||^2 - 1 = 1$ ➔
  - ➔ $c||w||^2 = 2$ ➔ $\boldsymbol{c} = \frac{2}{||w||^2}$.



- Margin = $||x^+ - x^-|| = ||cw|| = \boldsymbol{c}||w|| = \frac{2}{||w||^2}||w|| = \ldots$

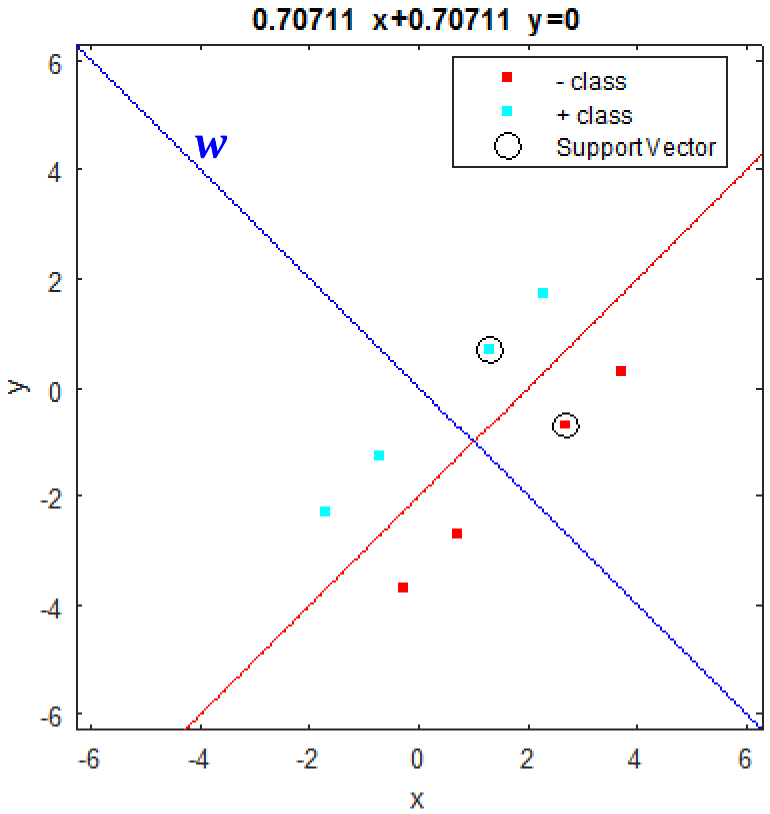
  $= \frac{\mathbf{2}}{||\boldsymbol{w}||} = \frac{2}{\sqrt{w^T w}}$

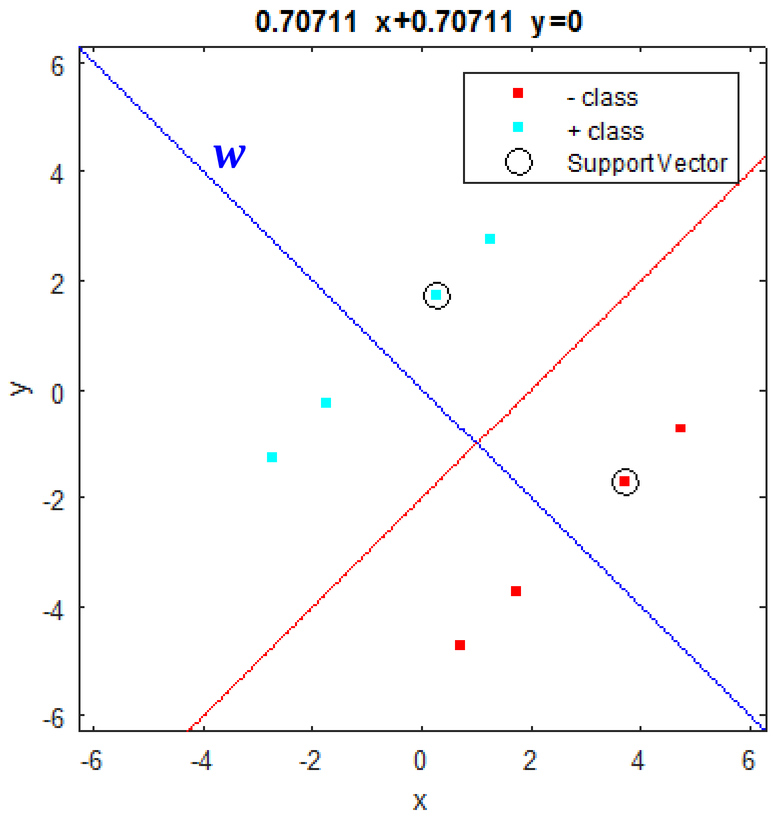
- Margin is defined as **2 over the length of *w***.

- How do we find ***w*** (i.e. θ) to maximize the margin?

SVM Margin Example

$$\text{Margin} = \frac{2}{||w||} = \frac{2}{\sqrt{w^T w}}$$





w = -0.70711 0.70711 (magnitude↑)

||w|| = 1, M = 2

w = -0.28868 0.28868 (magnitude↓)

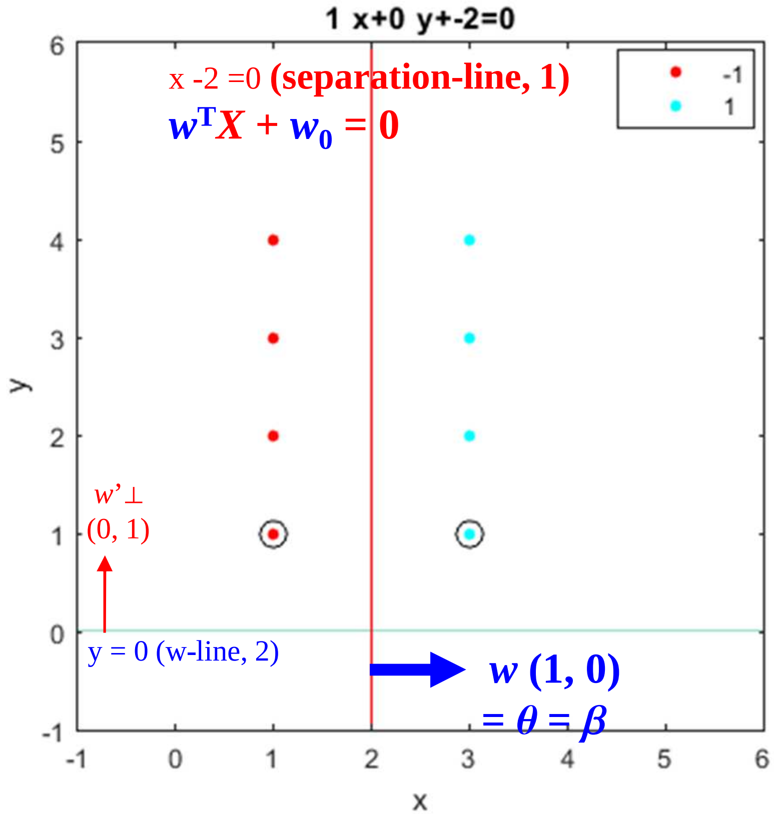
||w|| = 0.40825, M = 4.899

```
w = [-0.70711; 0.70711];
w_len = norm(w),        % sqrt((w' * w))
margin = 2 / w_len
```

```
w = [-0.28868; 0.28868];
w_len = norm(w),        % sqrt((w' * w))
margin = 2 / w_len
```

$\boldsymbol{\theta}^T \boldsymbol{X} + \boldsymbol{\theta}_0 = \mathbf{0}$ Example

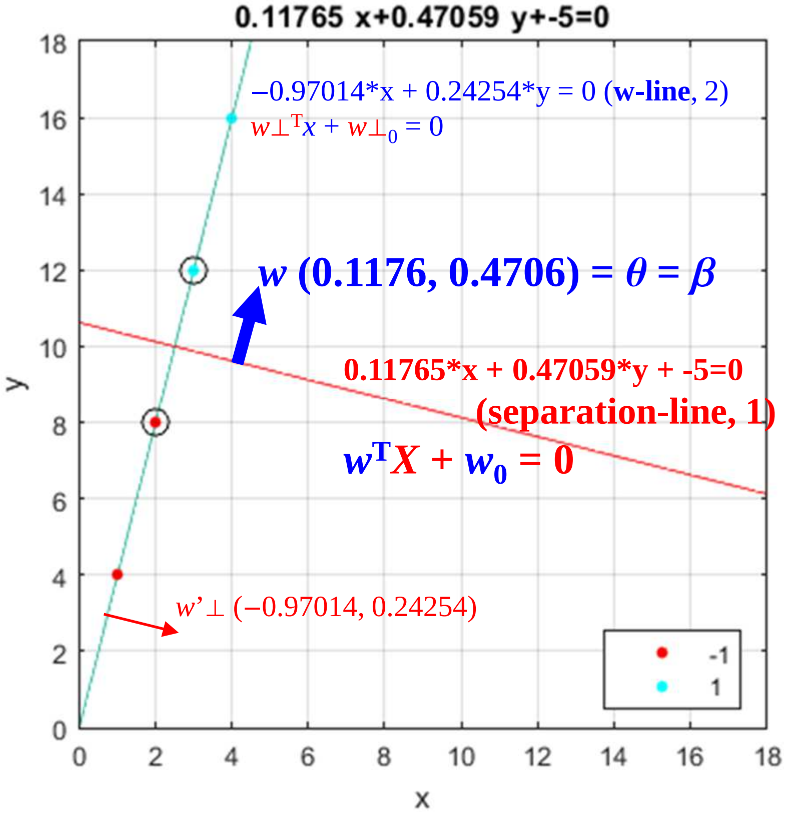
```
X = [1 1; 1 2; 1 3; 1 4; 3 1; 3 2; 3 3; 3 4;];          Y = [ones(4, 1) * -1; ones(4, 1)];
SVM_M = fitcsvm(X, Y, 'BoxConstraint', 1)

sv = SVM_M.SupportVectors;
theta = SVM_M.Beta;                theta0 = SVM_M.Bias;
Sep = null(theta');                %  perpendicular to theta

fstr1 = [num2str(theta(1)) '*x+'  num2str(theta(2)) '*y+' num2str(theta0) '=0']
fstr2 = [num2str(Sep(1)) '*x+'  num2str(Sep(2)) '*y'  '=0']

ezplot(fstr2, [-1 6 -1 6]), hold on,
gscatter(X(:,1),X(:,2),Y),
h1=ezplot(fstr1, [-1 6 -1 6]); axis equal,
set(h1, 'color', [1 0 0]),
plot(sv(:,1), sv(:,2), 'ko', 'MarkerSize', 10),   hold off,

theta'*[2; 5]+theta0         % w^T X + w_0 = 0
```



$\theta^T X + \theta_0 = 0$ vs. $\theta^T X = \hat{y}$ Example

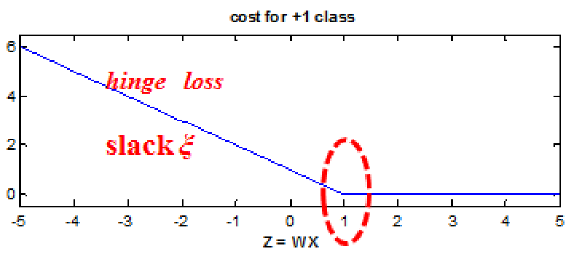
```
X=[1 4; 2 8; 3 12; 4 16];                Y = [-1; -1; 1; 1];
SVM_M = fitcsvm(X, Y, 'BoxConstraint', 1)

sv = SVM_M.SupportVectors;
theta = SVM_M.Beta;                      theta0 = SVM_M.Bias;
Sep = null(theta');                      %  perpendicular to theta

fstr1 = [num2str(theta(1)) '*x+'  num2str(theta(2)) ...
              '*y+' num2str(theta0) '=0']
fstr2 = [num2str(Sep(1)) '*x+'  num2str(Sep(2)) '*y'  '=0']

ezplot(fstr2, [0 18 0 18]), hold on,
gscatter(X(:,1),X(:,2),Y),
h1=ezplot(fstr1, [0 18 0 18]); axis equal, grid on,
set(h1, 'color', [1 0 0]),
plot(sv(:,1), sv(:,2), 'ko', 'MarkerSize', 10),   hold off,

% Next, try to compute y when x = 10. We obtained y = 8.125
(10*theta(1)-5)/theta(2)                 % y = 8.125 when x = 10.
theta'*[10; 8.125]+theta0                % w^T X + w_0 = 0
```

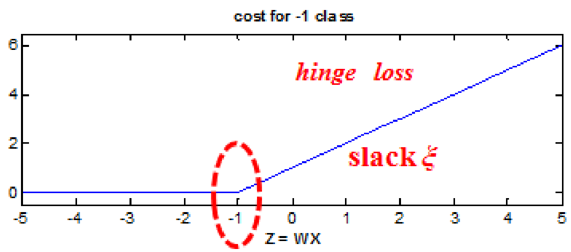
- In LR, θ **fix** to data.
  - θ = 4 in the figure. **(cyan line)**
- In SVM, θ **separate** data.
  - θ = (-5, 0.1176, 0.4706) **(red line)**



Soft Margin SVM with ***Slack*** ξ /xi/

- ***Slack var*** $\xi_j = 0$ for data <u>NOT</u> cross the boundary of its class, otherwise $\xi_j > 0$.
  - ***slack***$_i$ = $\max(0, (1 - y^{(i)}(wx^{(i)} + b))$ = max(0, ***dist_to_bound***) **$\geq$ 0**
    - ***dist_to_bound*** = 1 – ***y*** × **negLoss(:, 2)'** "**negLoss**" from **predict(…)** = $(\theta^T X + \theta_0)$.





X=[1 4; 2 8; 3 12; 4 16]; Y = [-1; -1; 1; 1];

SVM = **fitcsvm**(X, Y)

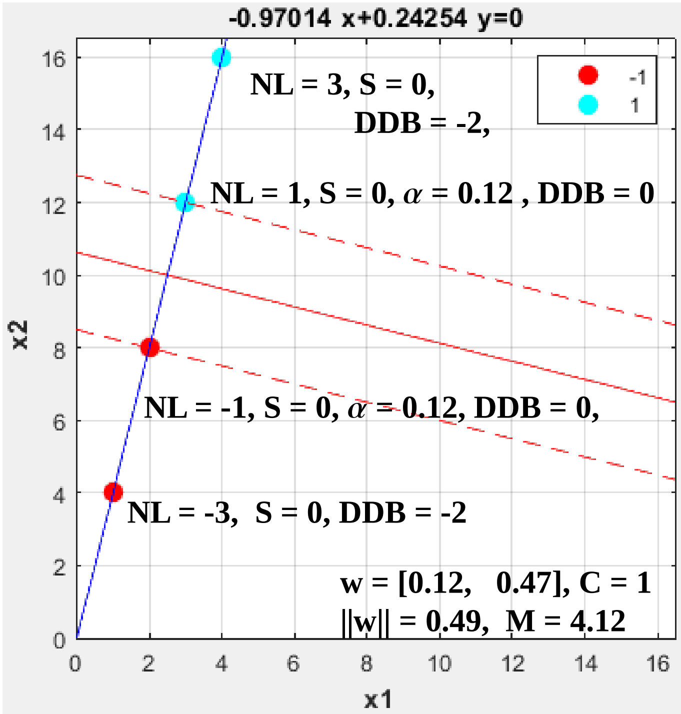
theta = SVM.**Beta**; theta0 = SVM.**Bias**;

w_len = **norm**(theta); **margin** = 2 / w_len

[labels **negLoss** yHatScores] = **predict**(SVM, X);

dist2DB = 1-Y'.* **negLoss**(:, 2)' **% = 1 – $y(\theta^T X + \theta_0)$**

slack = max(zeros(1, length(dist2DB)), dist2DB);

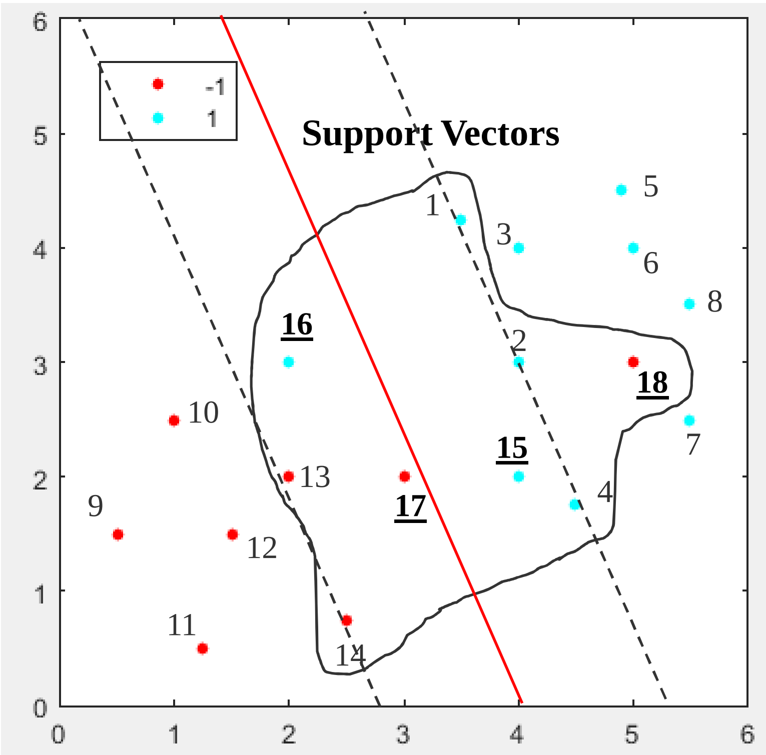


Example for Non-Zero Slack ξ /xi/

- $slack_i = max(\,0,\ \ (1 - y_i \times w^T x_i\,) = \max(0,\ dist\_to\_bound) \geq\ 0$
  - $dist\_to\_bound = 1 - y \times$ **negLoss(:, 2)'** "**negLoss**" from **predict(…) =** $w^T X$.
  - $\xi_j = 0$ if data is on right side of its class.
  - otherwise $\xi_j > 0$.

```
SVM = fitcsvm(X, Y)
[labels negLoss yHatScores] = predict(SVM, X);
```

	α_i
1	0.0423
2	0.1936
4	1
13	1
14	0.2359
15	1
16	1
17	1
18	1



$\|w\| = 0.9$, $M = 2.23$, $C = 1$