

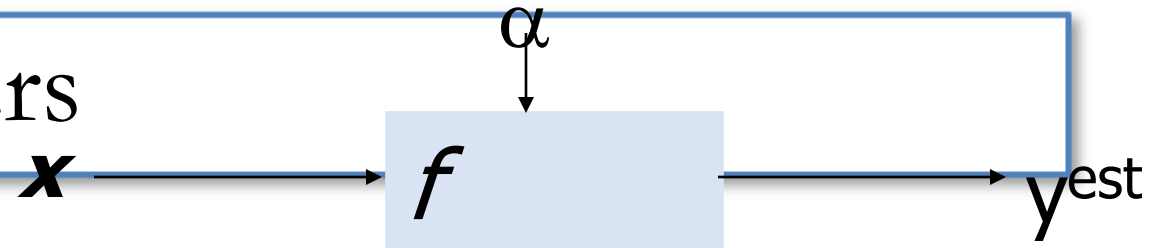
COSC 3337 : Data Science I



N. Rizk

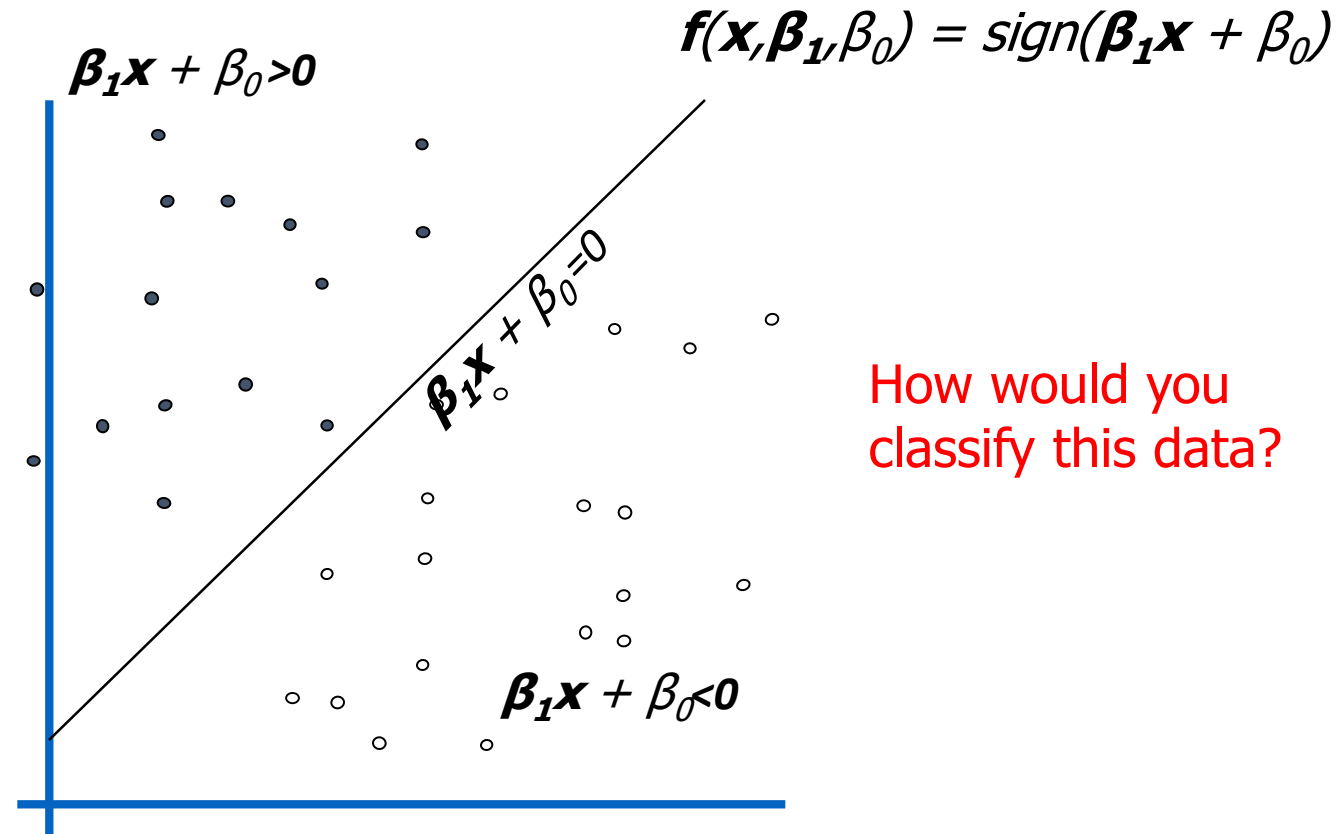
College of Natural and Applied Sciences
Department of Computer Science
University of Houston

Linear Classifiers



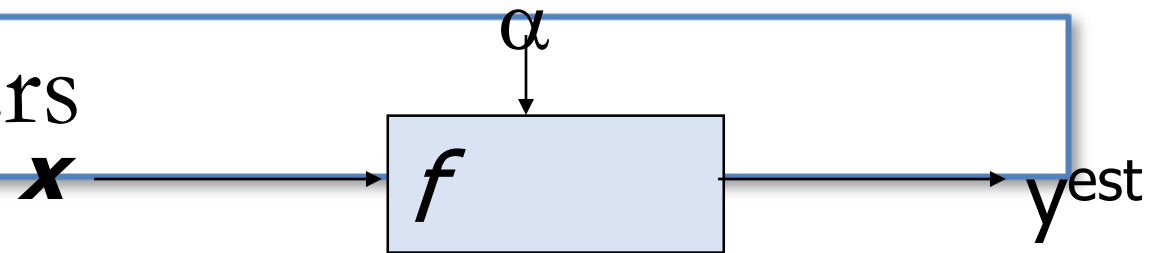
• denotes +1

◦ denotes -1



How would you
classify this data?

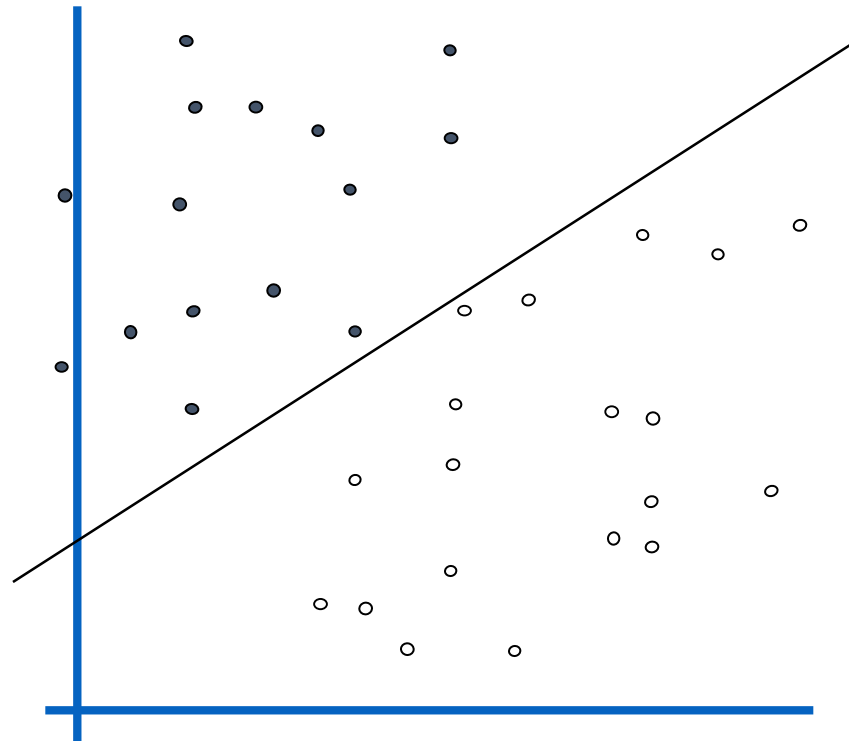
Linear Classifiers



$$f(\mathbf{x}, \boldsymbol{\beta}_1, \beta_0) = \text{sign}(\boldsymbol{\beta}_1 \mathbf{x} + \beta_0)$$

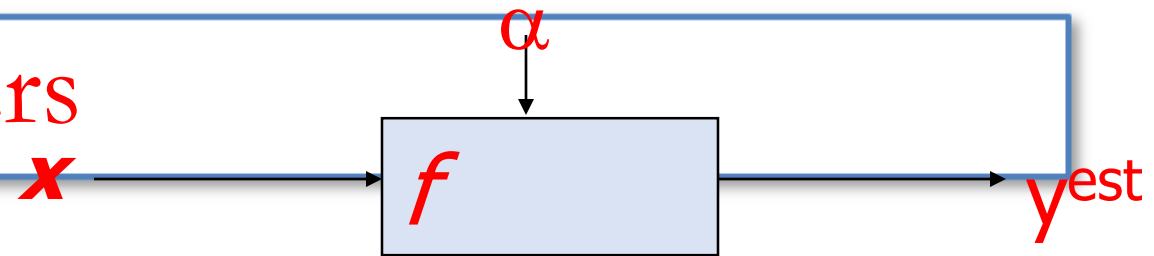
• denotes +1

○ denotes -1

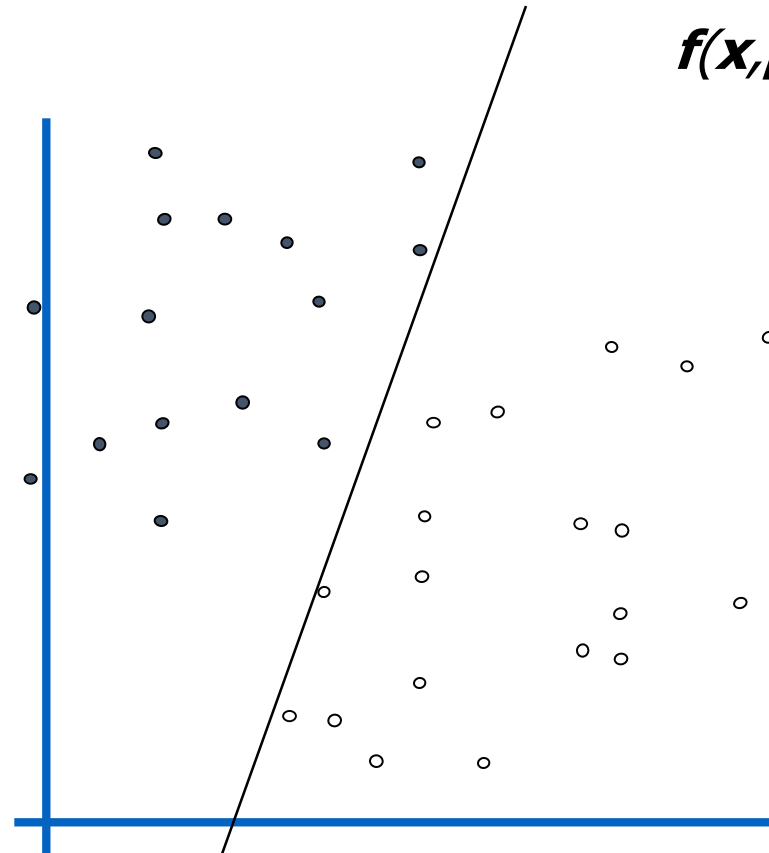


How would you
classify this data?

Linear Classifiers



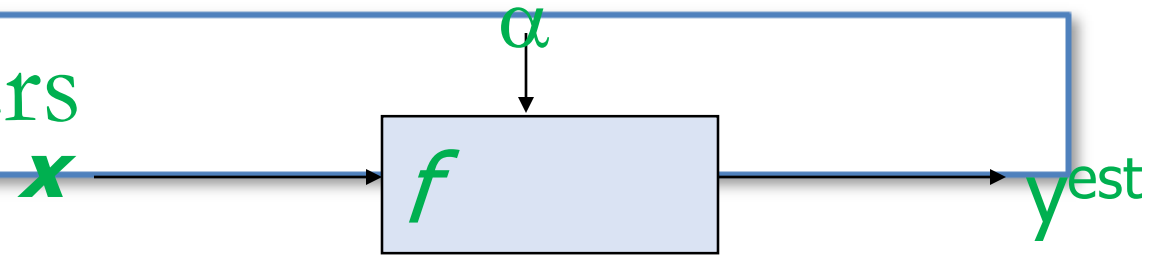
- denotes +1
- denotes -1



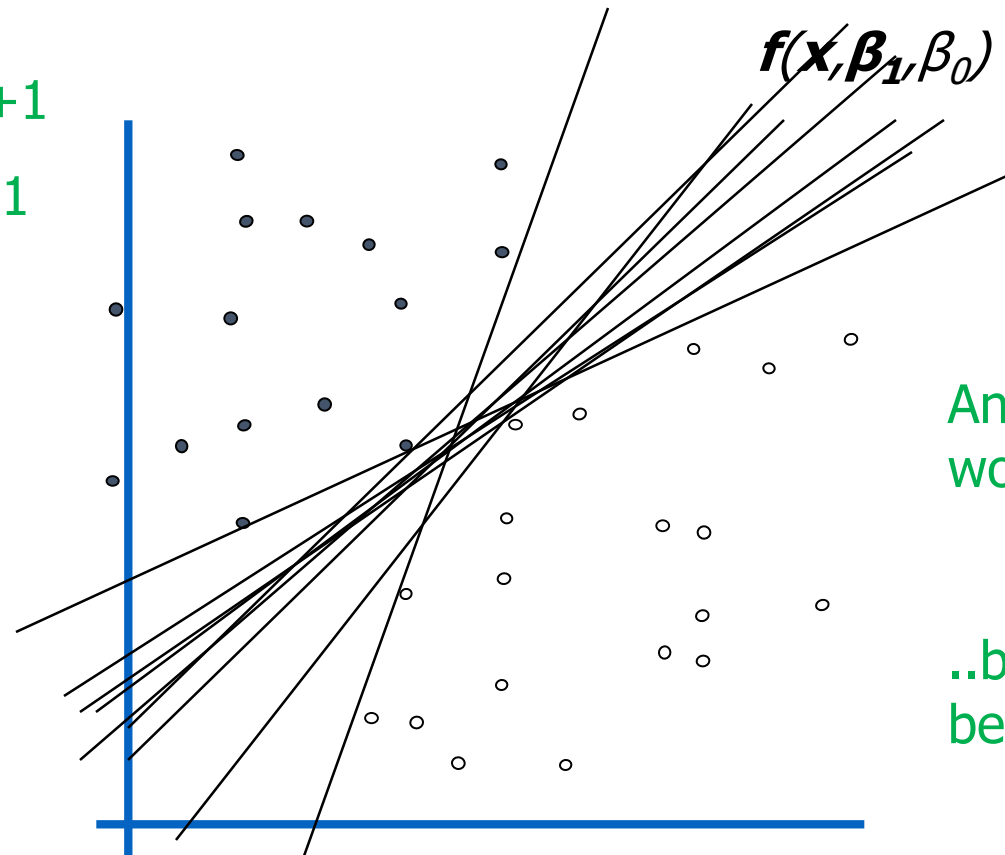
$$f(\mathbf{x}, \boldsymbol{\beta}_1, \beta_0) = \text{sign}(\boldsymbol{\beta}_1 \mathbf{x} + \beta_0)$$

How would you
classify this data?

Linear Classifiers



- denotes +1
- denotes -1

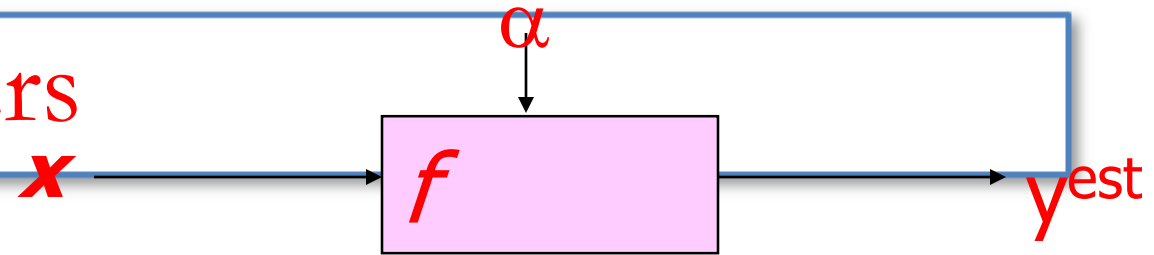


$$f(\mathbf{x}, \boldsymbol{\beta}_1, \beta_0) = \text{sign}(\boldsymbol{\beta}_1 \mathbf{x} + \beta_0)$$

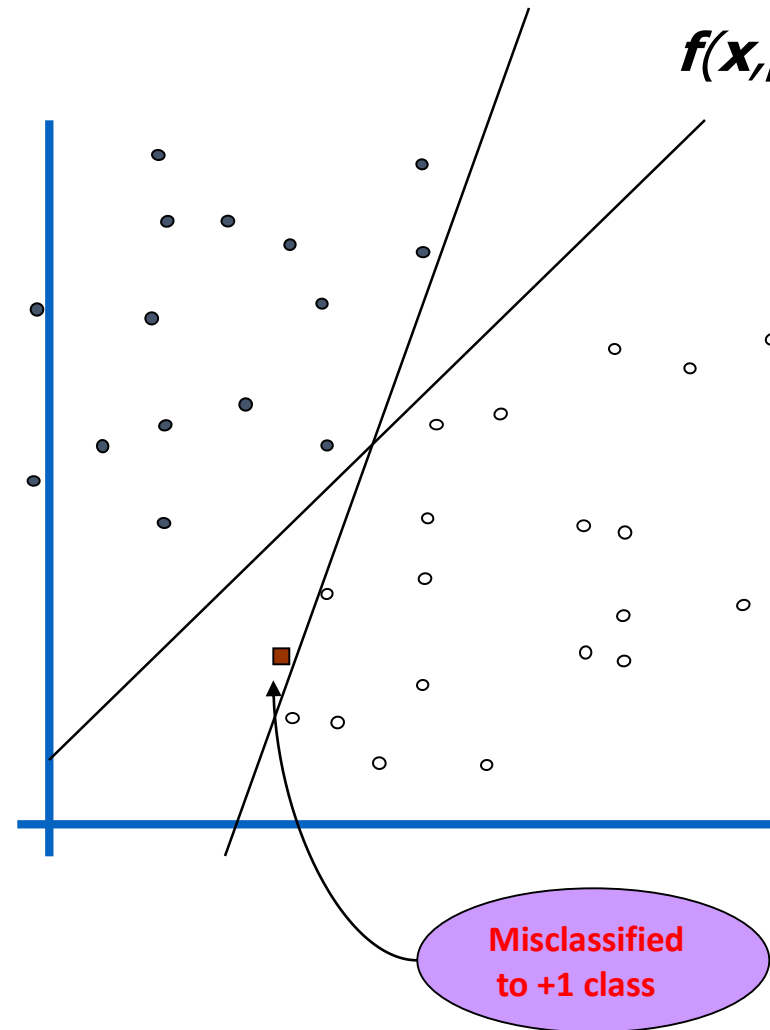
Any of these
would be fine..

..but which is
best?

Linear Classifiers



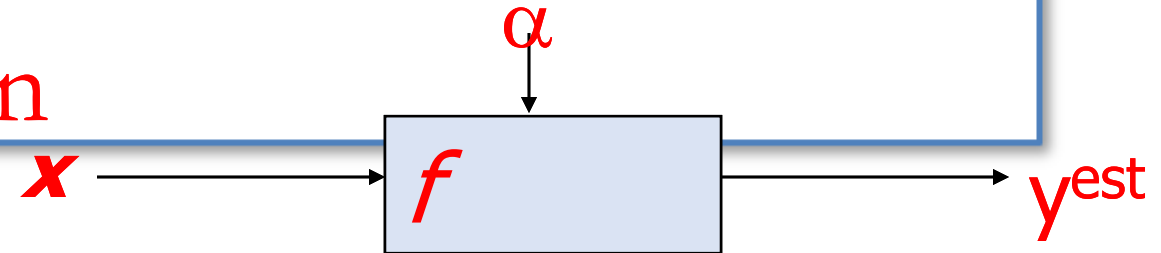
- denotes +1
- denotes -1



$$f(\mathbf{x}, \boldsymbol{\beta}_1, \beta_0) = \text{sign}(\boldsymbol{\beta}_1 \mathbf{x} + \beta_0)$$

How would you classify this data?

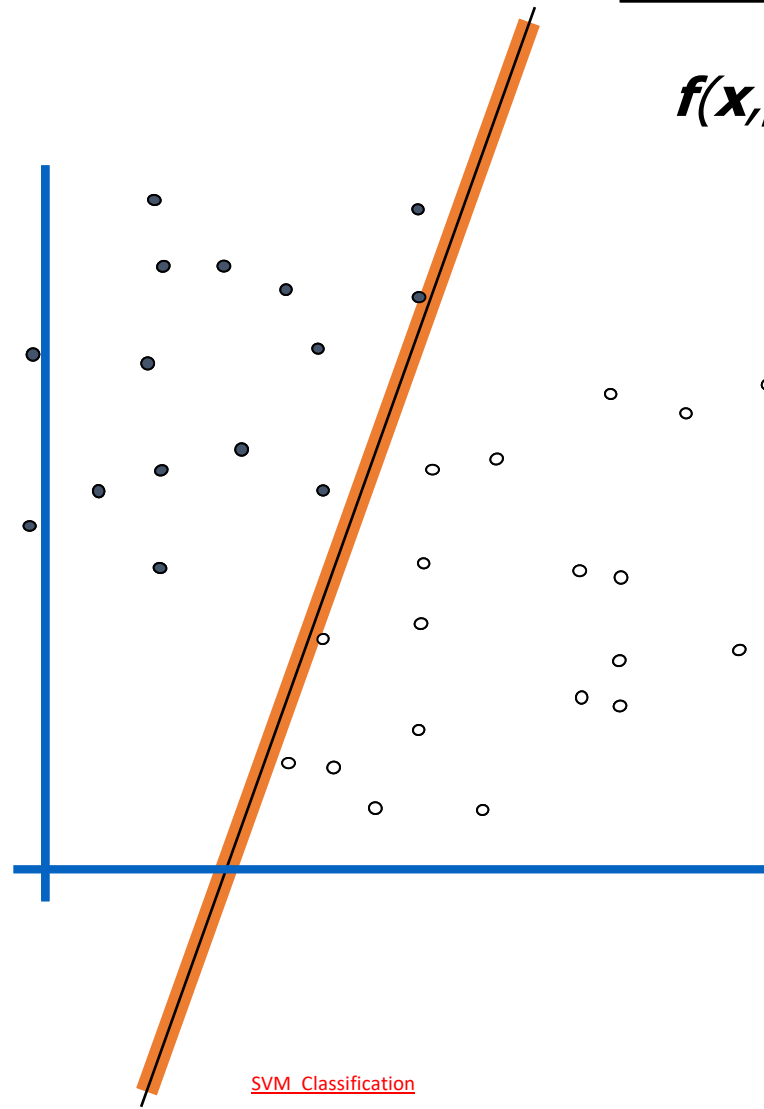
Classifier Margin



$$f(\mathbf{x}, \boldsymbol{\beta}_1, \beta_0) = \text{sign}(\boldsymbol{\beta}_1 \mathbf{x} + \beta_0)$$

• denotes +1

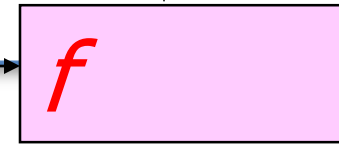
○ denotes -1



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

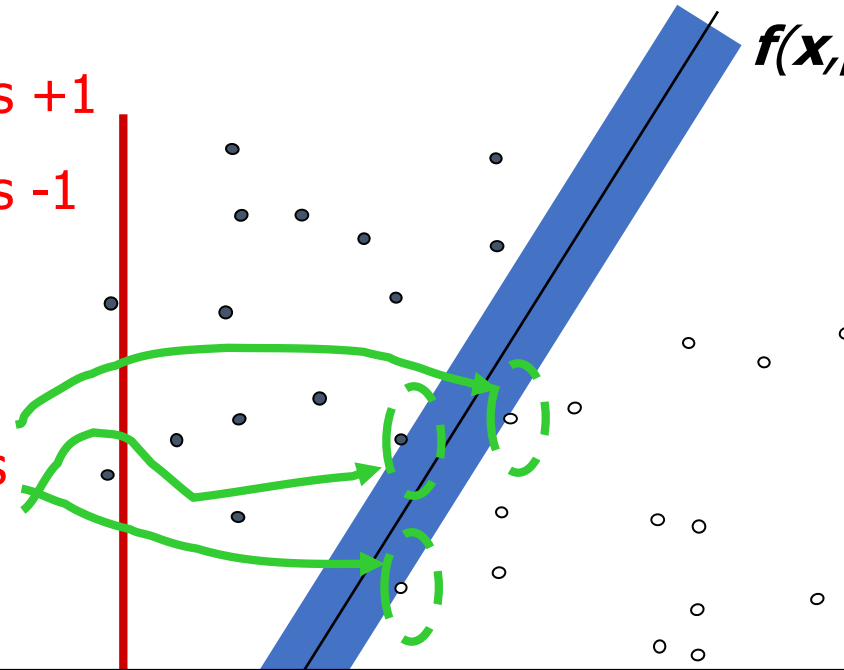
Maximum Margin

Linear SVM



- denotes +1
- denotes -1

Support Vectors are those datapoints that the margin pushes up against



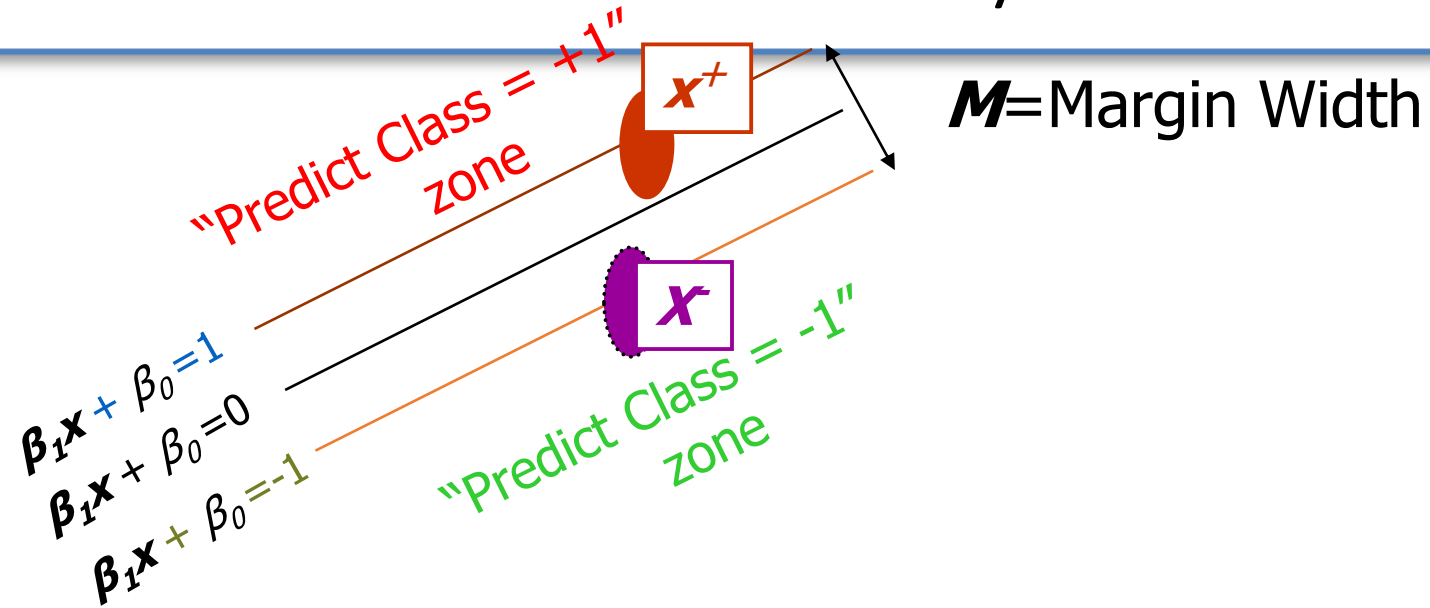
$$f(\mathbf{x}, \boldsymbol{\beta}_1, \beta_0) = \text{sign}(\boldsymbol{\beta}_1 \mathbf{x} + \beta_0)$$

The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

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

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 well.

Linear SVM Mathematically



What we know:

- $\beta_1 x^+ + \beta_0 = +1$
- $\beta_1 x^- + \beta_0 = -1$
- $\beta_1 \cdot (x^+ - x^-) = 2$

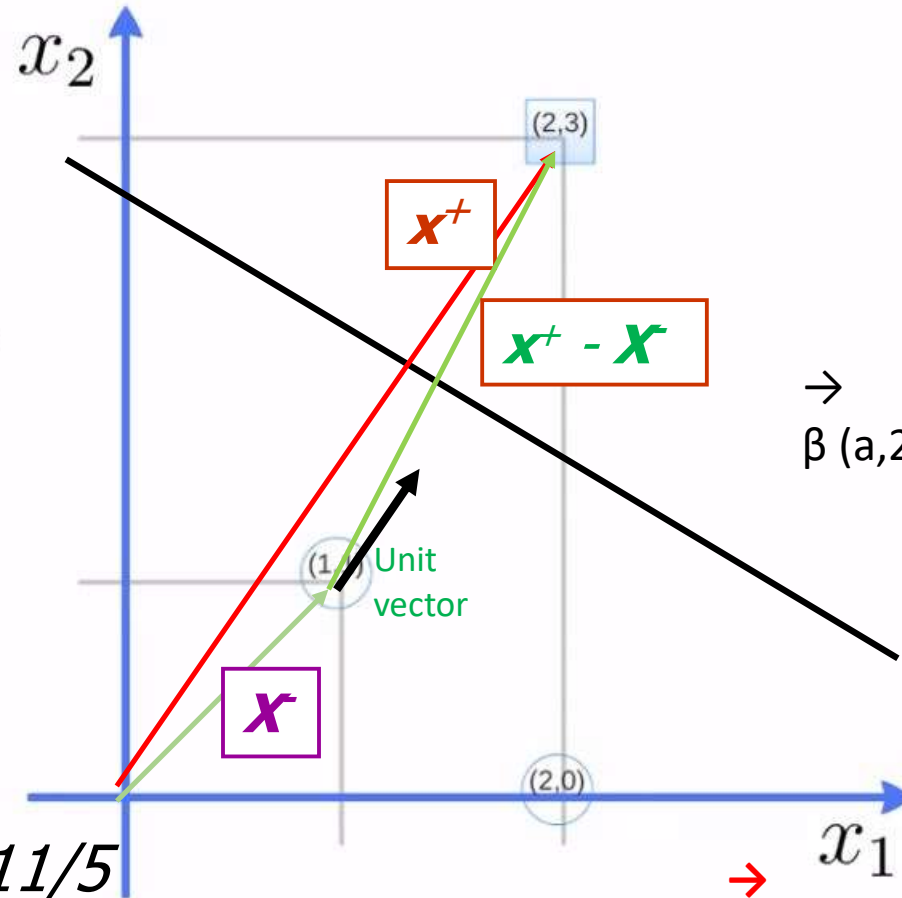
$$M = ((x^+ - x^-) \cdot \beta) / |\beta|$$

$$= 2 / |\beta|$$

$$\text{Width} = (\mathbf{x}^+ - \mathbf{x}^-) \cdot \boldsymbol{\beta} / \|\boldsymbol{\beta}\|$$

Linear SVM

Example



Beta divided by magnitude to get the unit vector

$$\rightarrow \beta(a, 2a) = (2, 3) - (1, 1)$$

$$g(1,1) = -1 = a + 2a + \beta_0 = -1$$

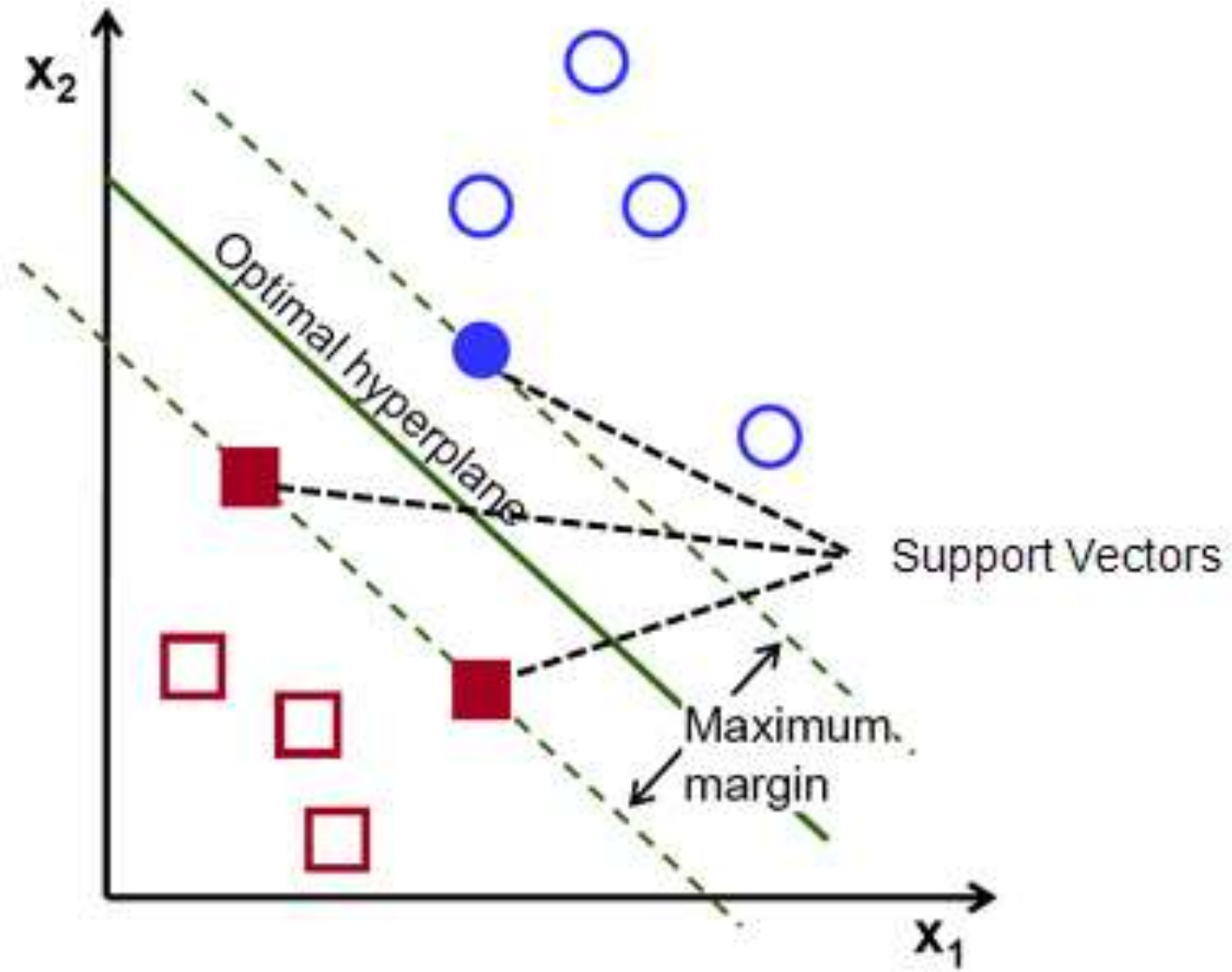
$$g(2,3) = 1 = 2a + 6a + \beta_0 = 1$$

$$\rightarrow a = 2/5; \beta_0 = 11/5$$

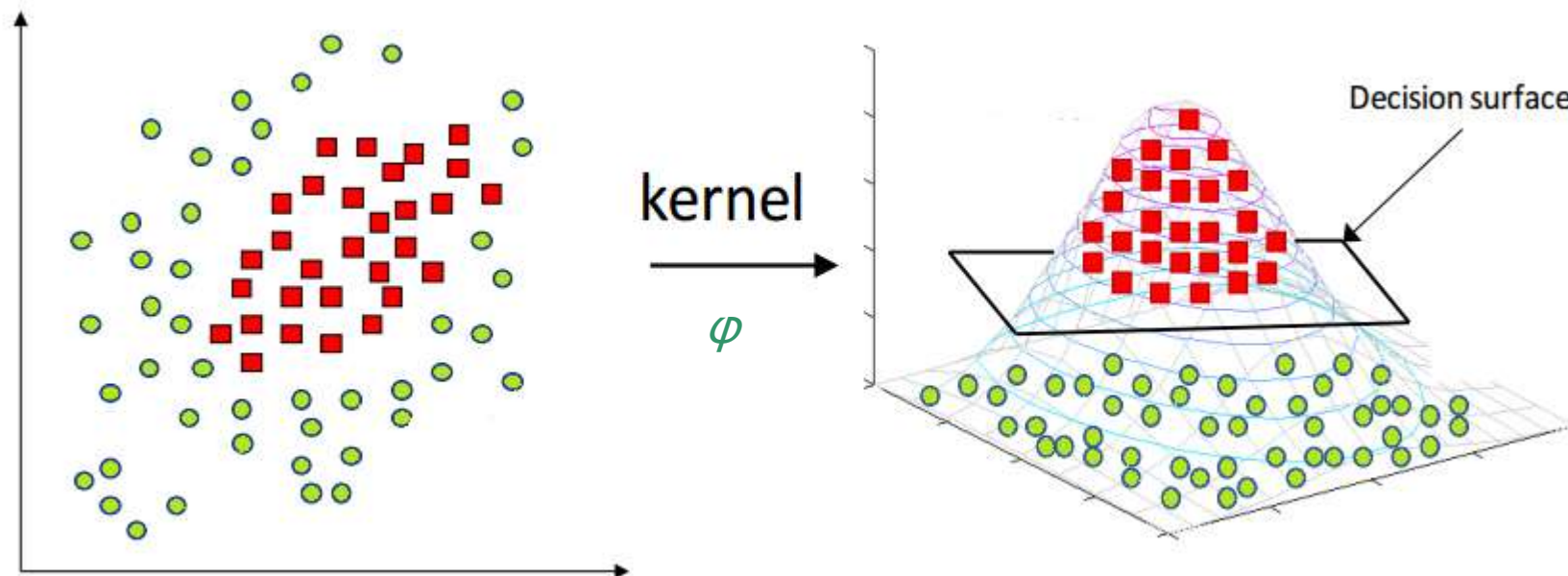
$$\rightarrow g(x) = 2/5 x_1 + 4/5 x_2 - 11/5$$

$$\rightarrow g(x) = x_1 + 2 x_2 - 5.5$$

$$\rightarrow \beta(2/5, 4/5) = (2, 3) - (1, 1)$$



Kernel? φ 2D \rightarrow any dimension

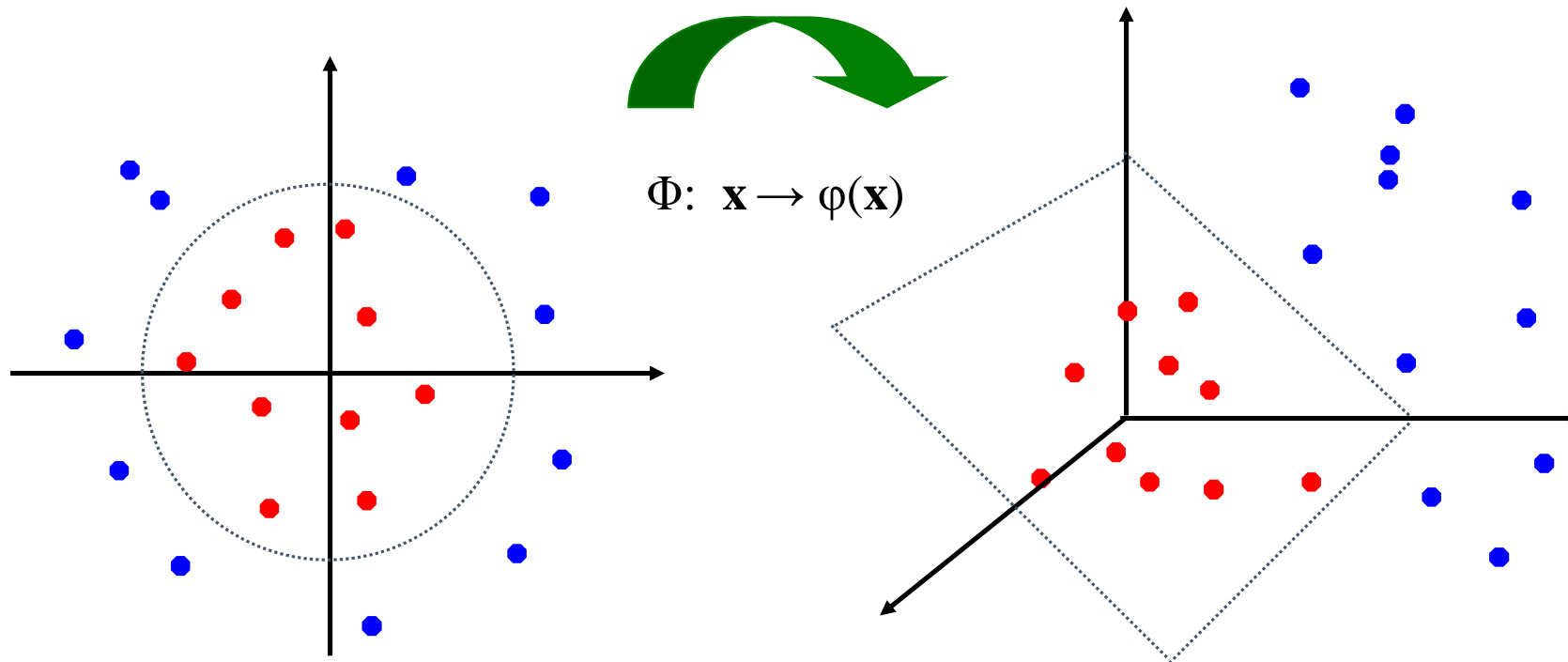


Function to map the data (transform) from 2-dimensional space to 3-dimensional space, we will be able to find a decision surface that clearly divides between different classes. **Find the boundary, and make the classification.**

Non-linear SVMs: Feature spaces



- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



Kernel allows us to operate in the original feature space without computing the coordinates of the data in a higher dimensional space.



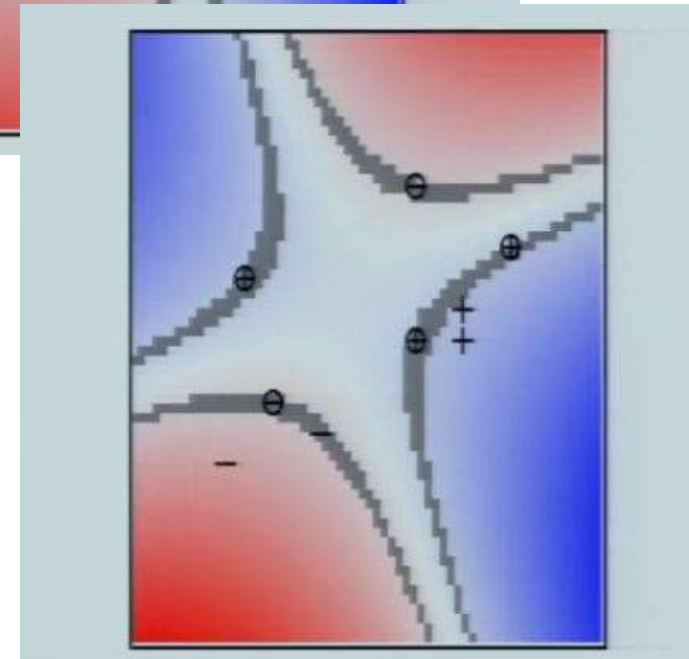
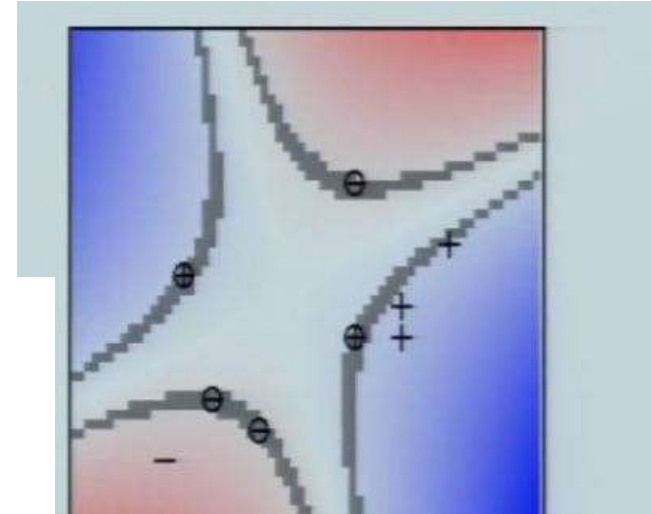
Need to optimize a kernel function

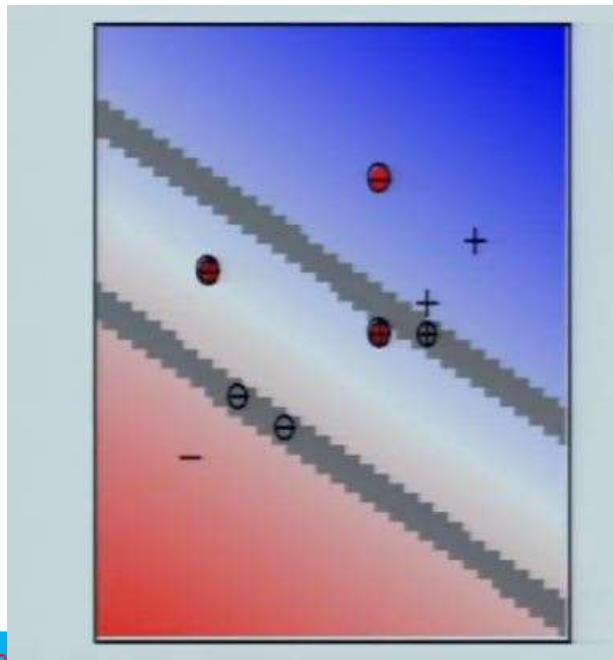
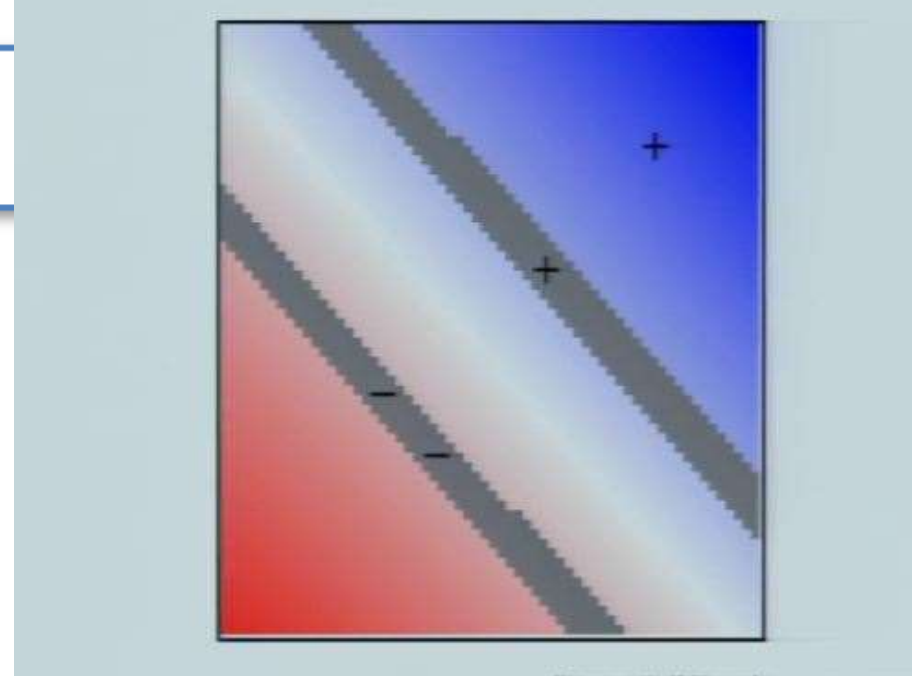
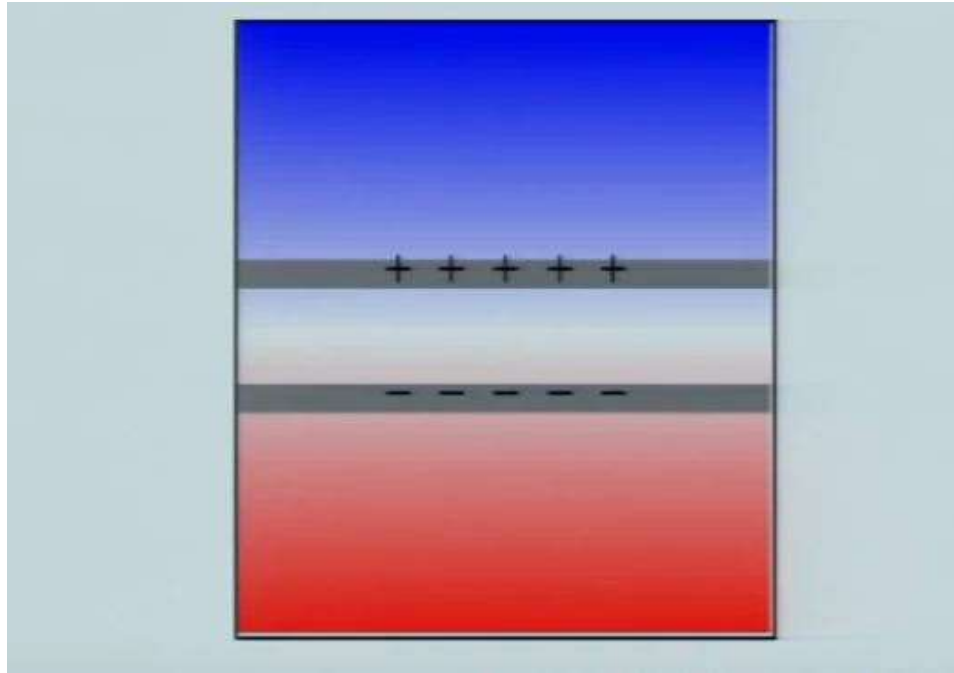
$$k(\mathbf{x}_i, \mathbf{y}_i) = \varphi(\mathbf{x}_i) \bullet \varphi(\mathbf{y}_i)$$

A kernel choice can be

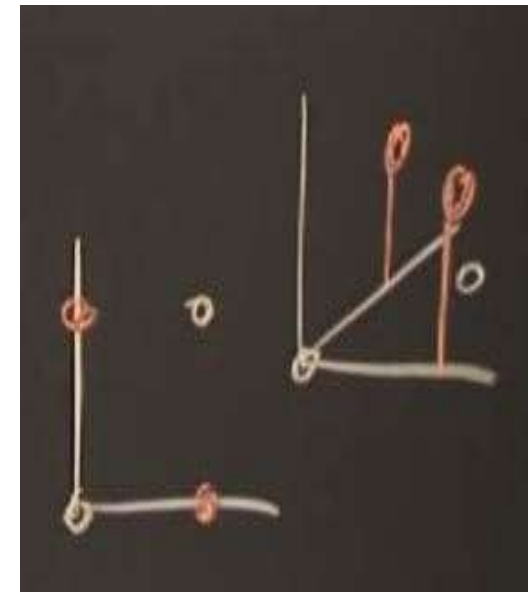
$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$$

$$k(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x} - \mathbf{y}\|^2}, \gamma > 0$$





Points are not linearly separable →
Need of transformation
in another space



Instead of.....

Transform to 9th dimension

$$\phi(\mathbf{x}) = (x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2)^T$$

$$\phi(\mathbf{y}) = (y_1^2, y_1y_2, y_1y_3, y_2y_1, y_2^2, y_2y_3, y_3y_1, y_3y_2, y_3^2)^T$$

$$\phi(\mathbf{x})^T \phi(\mathbf{y}) = \sum_{i,j=1}^3 x_i x_j y_i y_j$$

Kernel's Role

instead of doing the complicated computations in the 9-dimensional space, we reach the same result within the 3-dimensional space by calculating the dot product of \mathbf{x} - transpose and \mathbf{y}

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= (\mathbf{x}^T \mathbf{y})^2 \\ &= (x_1y_1 + x_2y_2 + x_3y_3)^2 \\ &= \sum_{i,j=1}^3 x_i x_j y_i y_j \end{aligned}$$

Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- Polynomial of power p : $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$

- Gaussian (radial-basis function network):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

- Sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$

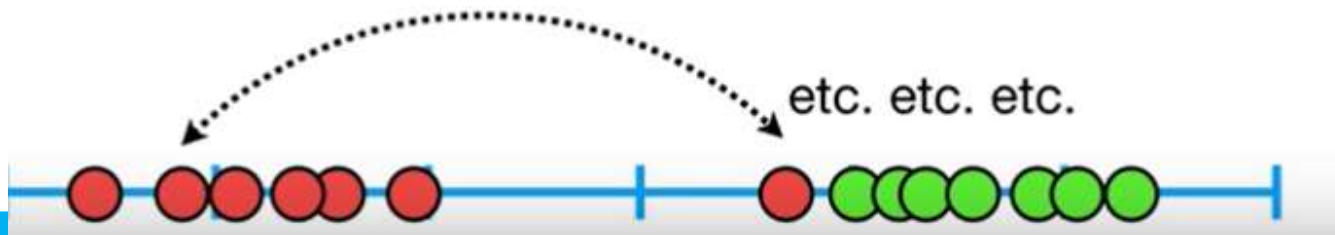
The polynomial kernel

The polynomial kernel: With n original features and d degrees of polynomial, the polynomial kernel yields n^d expanded features.

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$$

When $d = 1$, the **Polynomial Kernel** computes the relationships between each pair of observations in **1-Dimension**...

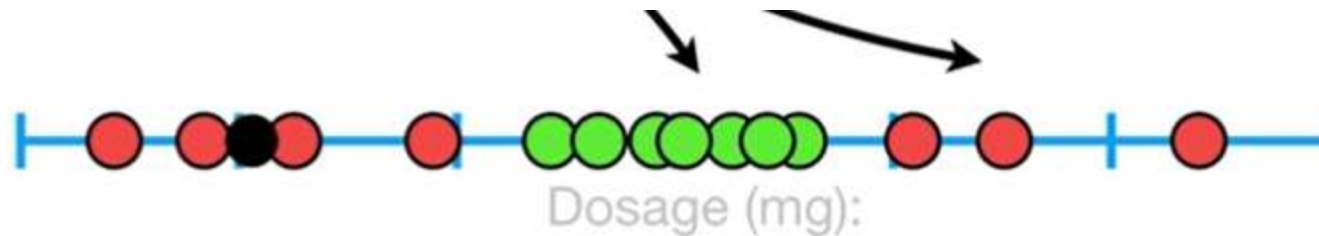
These relationships are used to find the support vector classifiers
Good value of d can be found using cross validation



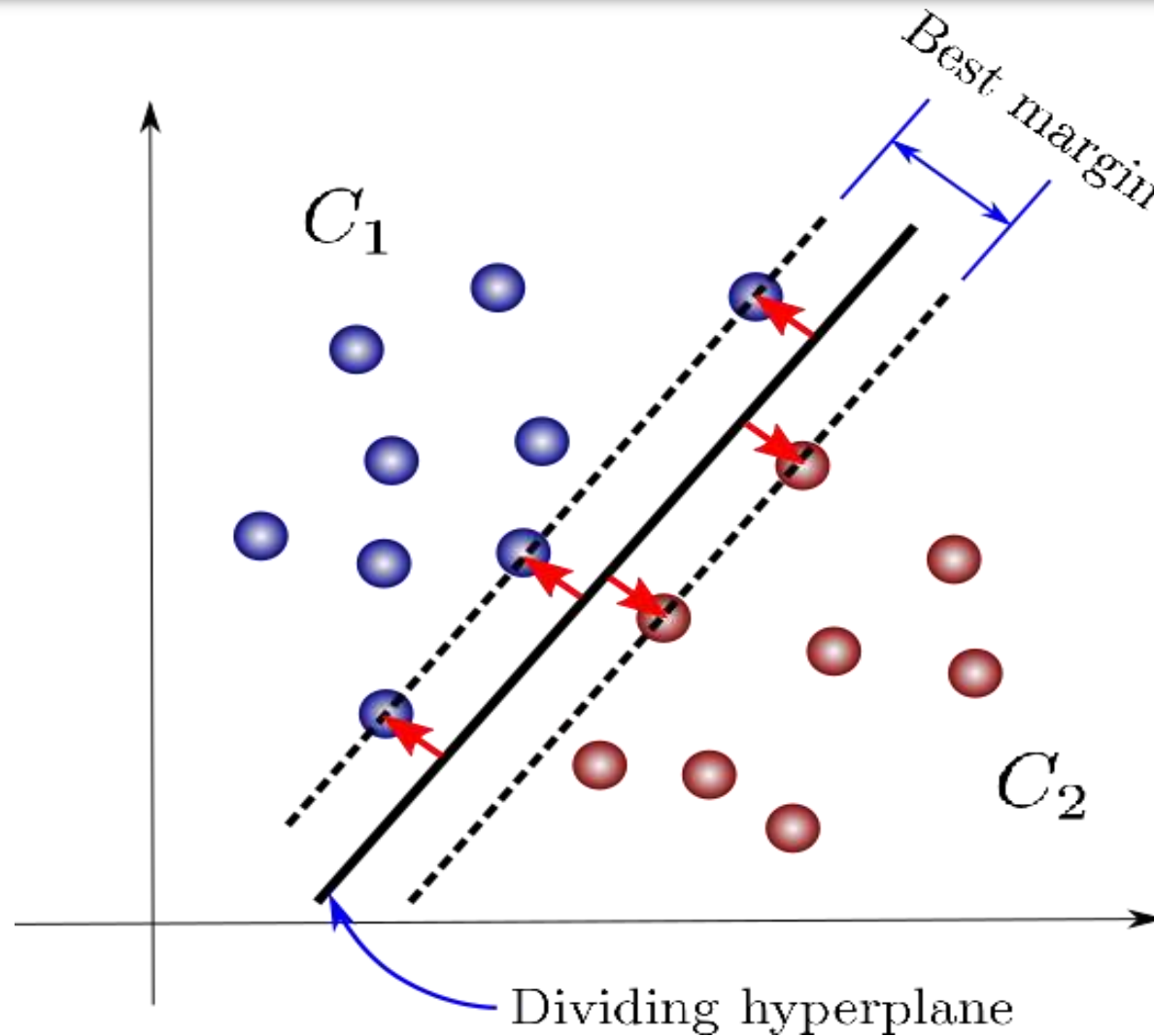
Gaussian Kernel RBF

The Radial Basis Function (RBF) kernel is also called the Gaussian kernel: The γ parameter defines how much influence a single training example has. It behaves like KNN the closer the more influence on the classification

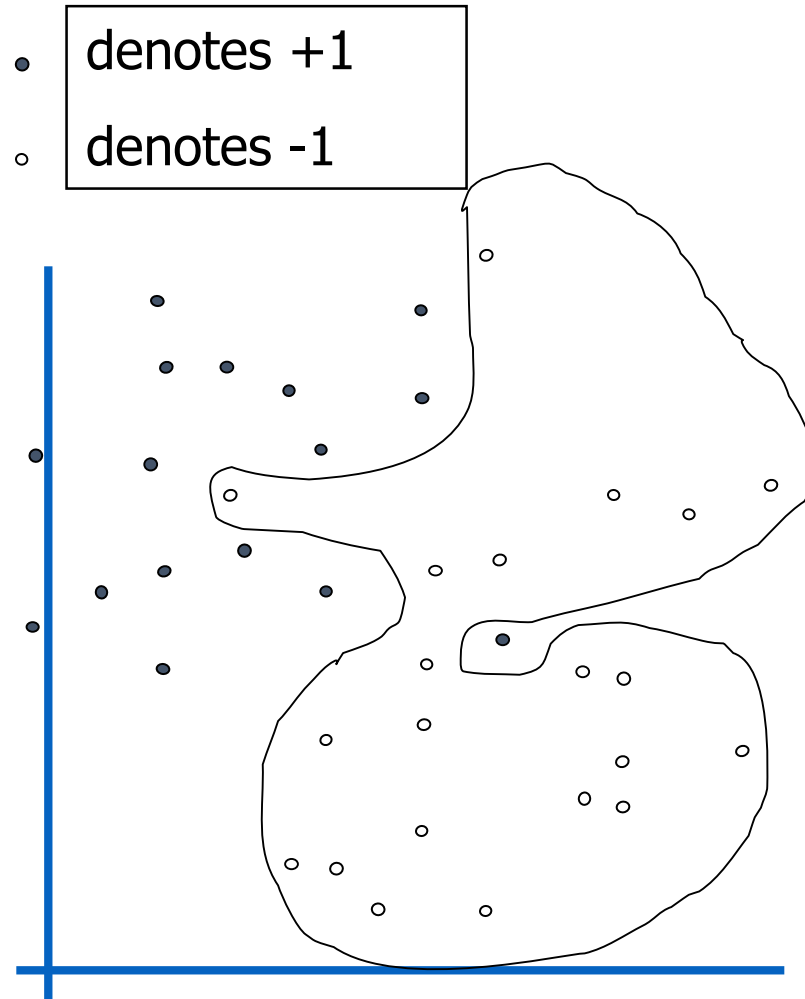
$$k(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x} - \mathbf{y}\|^2}, \gamma > 0$$



Hard Margin vs Soft vs Best Margin



Hard Margin ...Dataset with noise

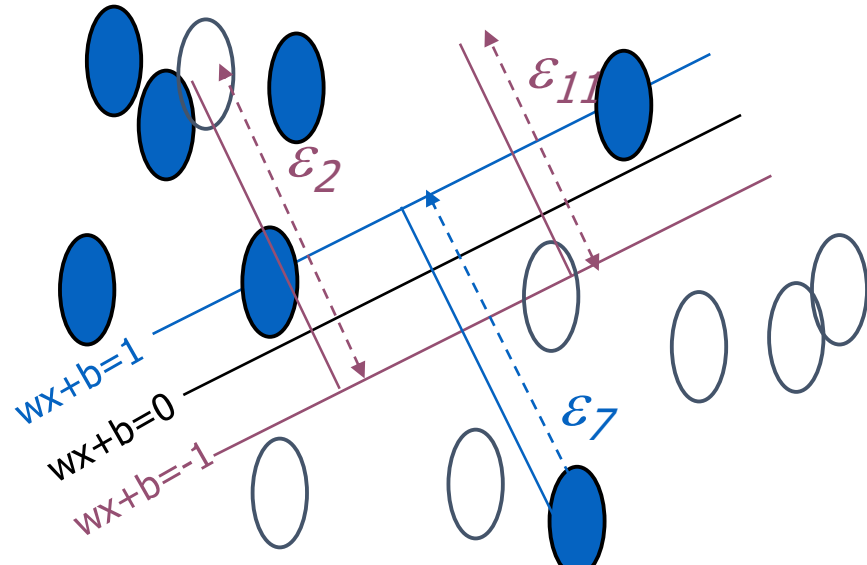


- **Hard Margin:** So far we require all data points be classified correctly
 - **No training error**
- **What if the training set is noisy?**
 - **Solution 1: use very powerful kernels**

OVERFITTING!

Soft Margin Classification

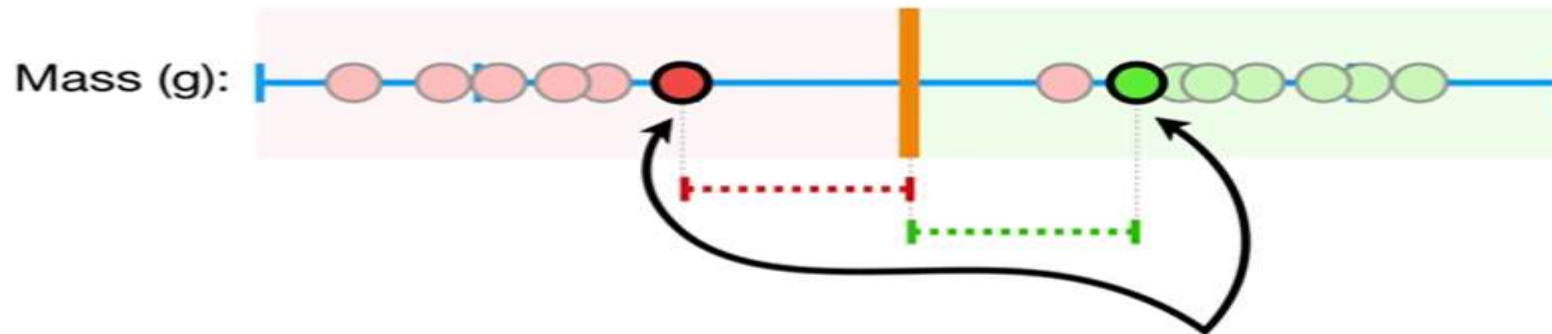
Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples.



What should our quadratic optimization criterion be?

Minimize

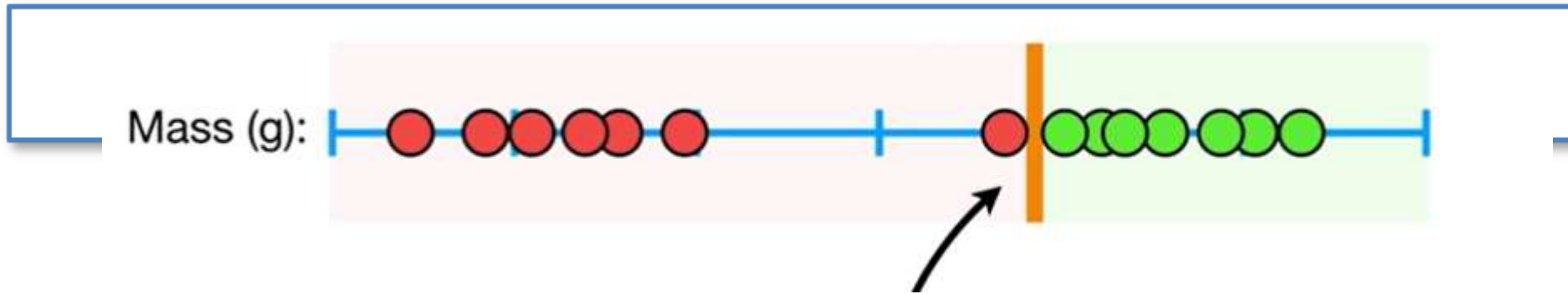
$$\frac{1}{2} \mathbf{w} \cdot \mathbf{w} + C \sum_{k=1}^R \varepsilon_k$$



For example, if we put the threshold halfway between these two observations...



Choosing a threshold that allows misclassifications is an example of the **Bias/Variance Tradeoff** that plagues all of machine learning.

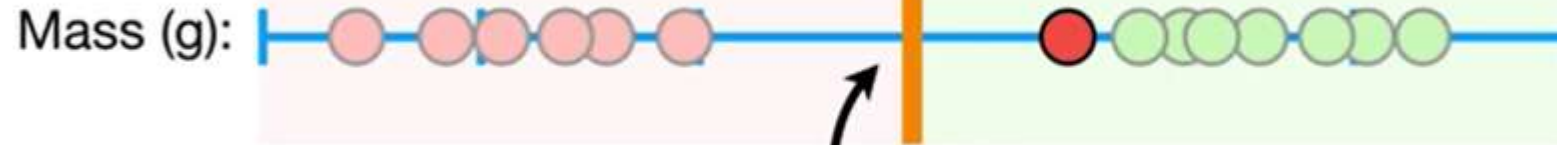


Threshold very sensitive to training data=>low bias

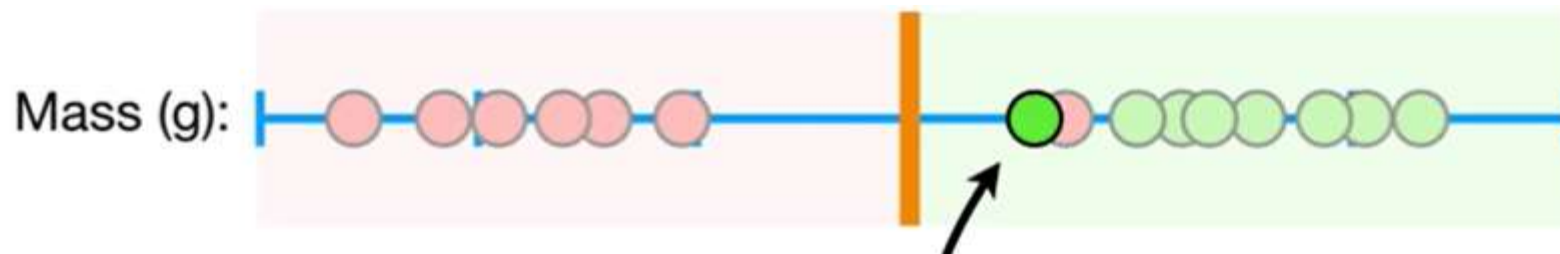


Performed poorly when we get new data=>high variance

In contrast if we select a threshold



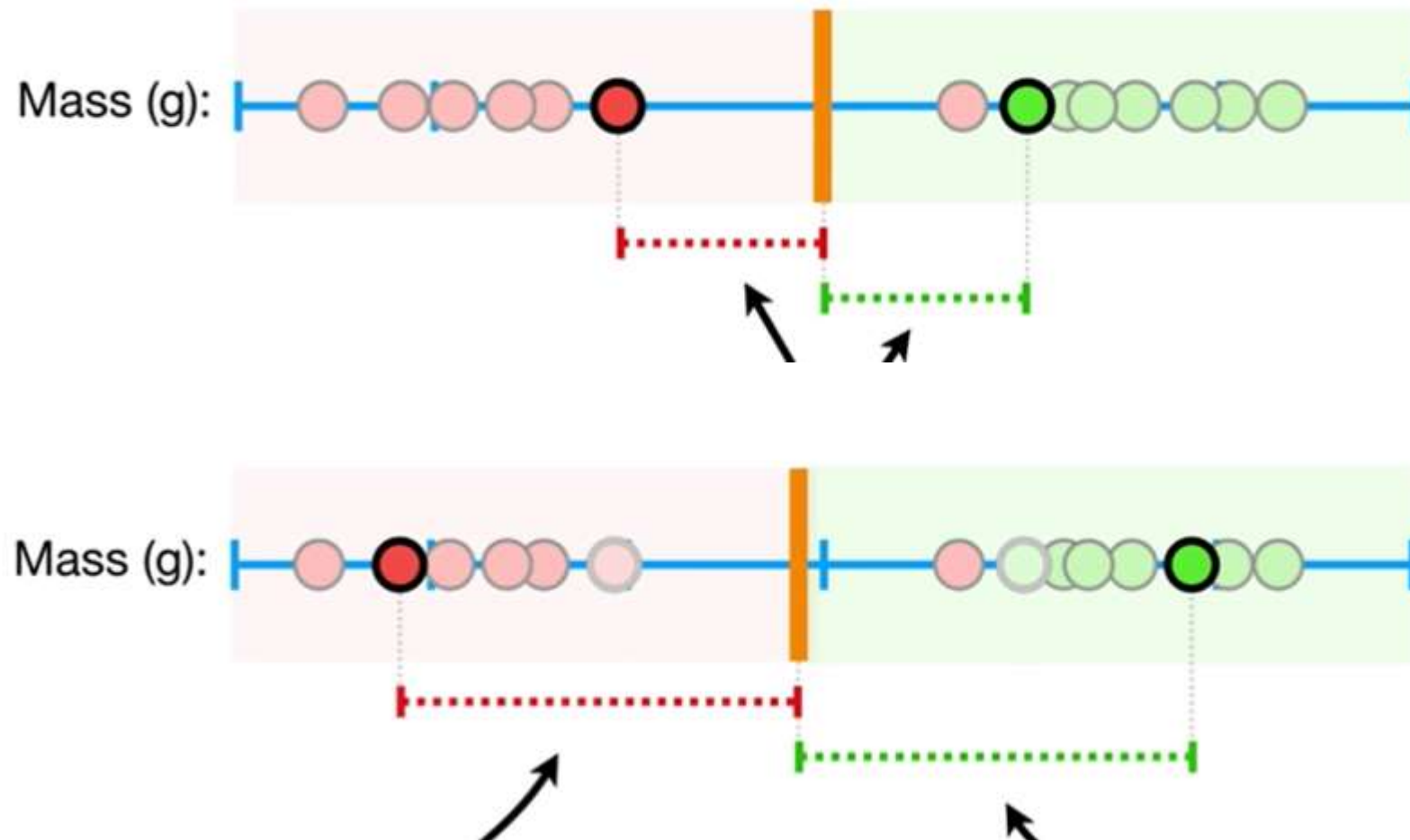
Threshold less sensitive to training data and allow misclassifications => High bias



Performed better when we get new data => low variance

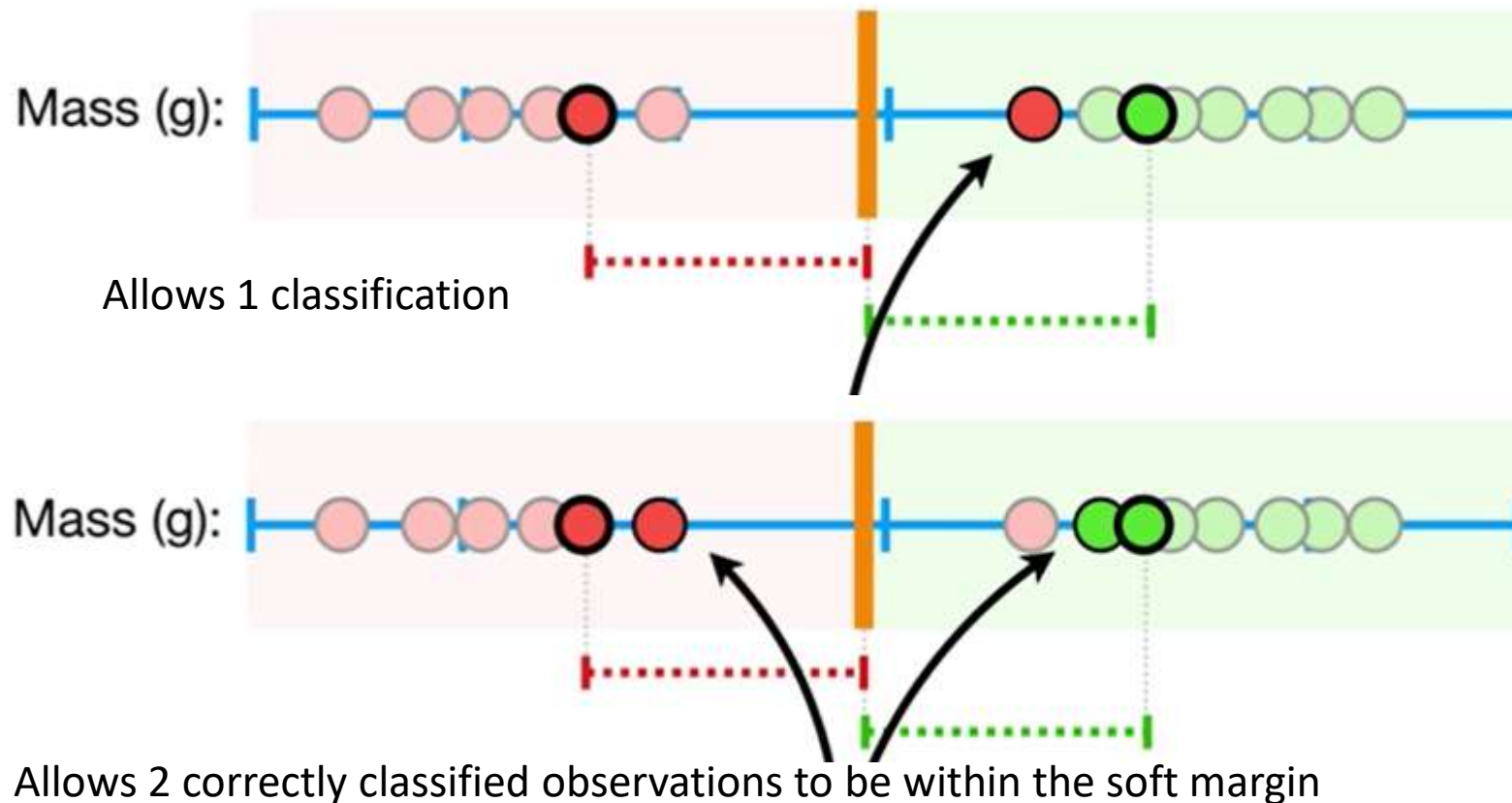
When we allow misclassifications the distance is called **SOFT MARGIN**

How do we find the best soft margin?

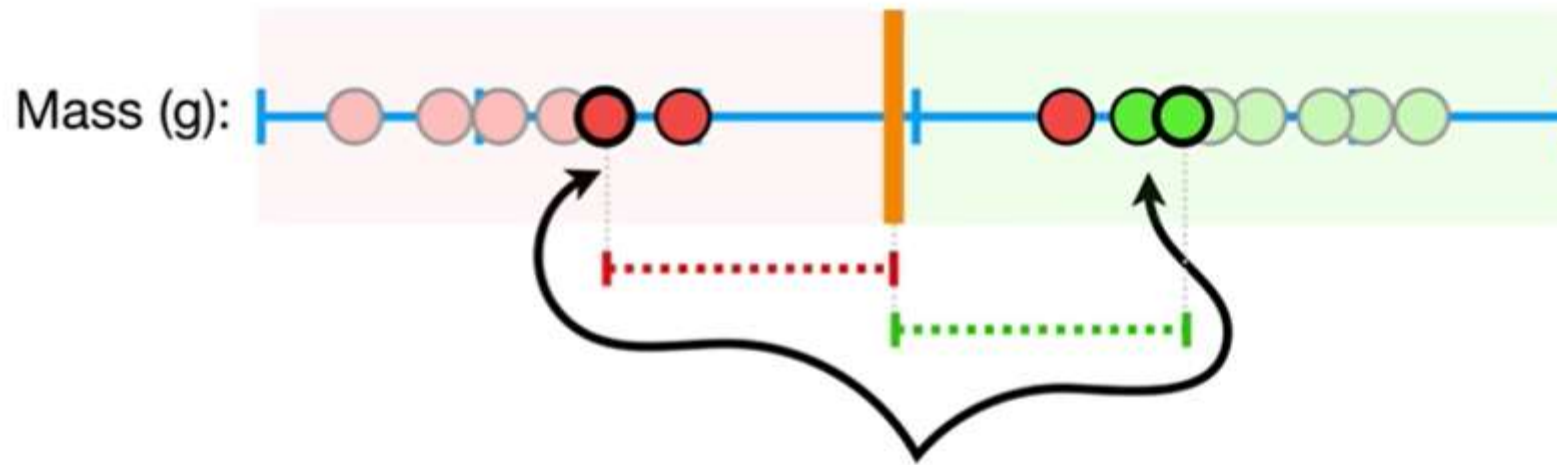


Cross validation

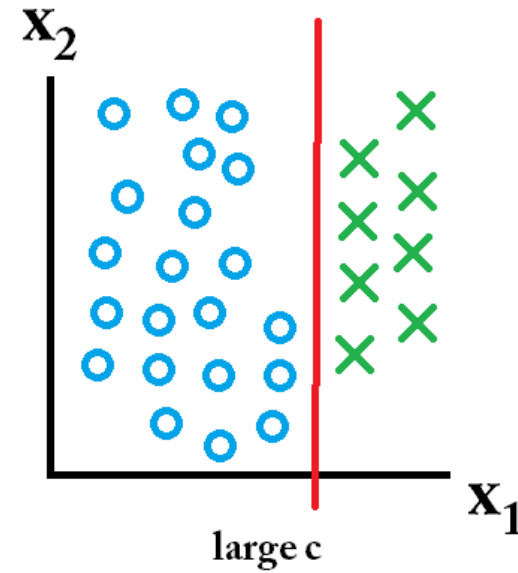
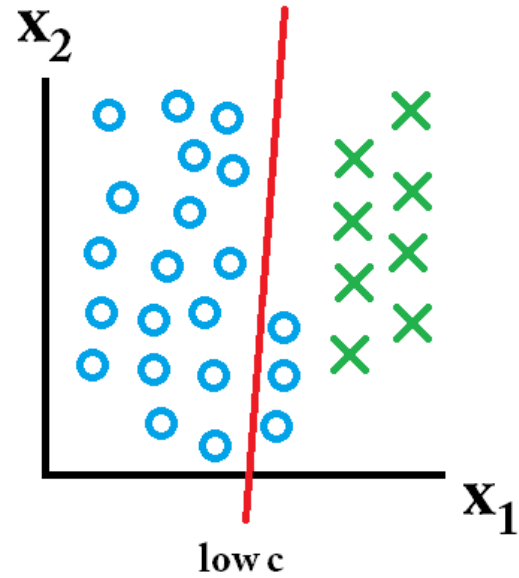
To find out the best threshold by determining how many misclassifications and observations to be allowed in the soft margin to get the best classifications



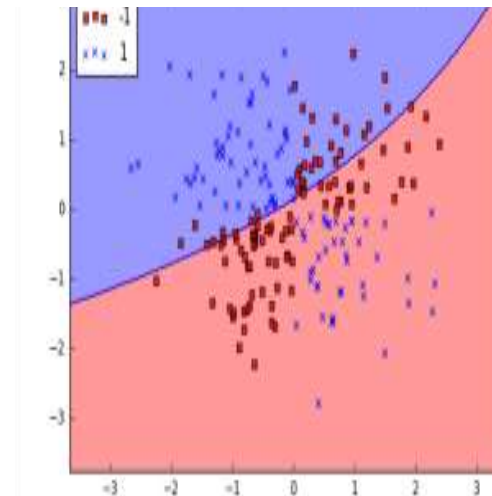
The correctly classified observations on the edge and within the soft margin are called **support vector machine**.



C vs Gamma

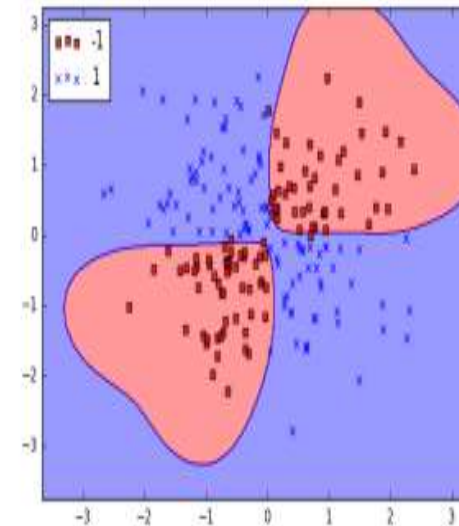


SVM((C regularization (low ==> wide margin)
gamma how far the influence of a single training
example reaches (low values ==> 'far'))

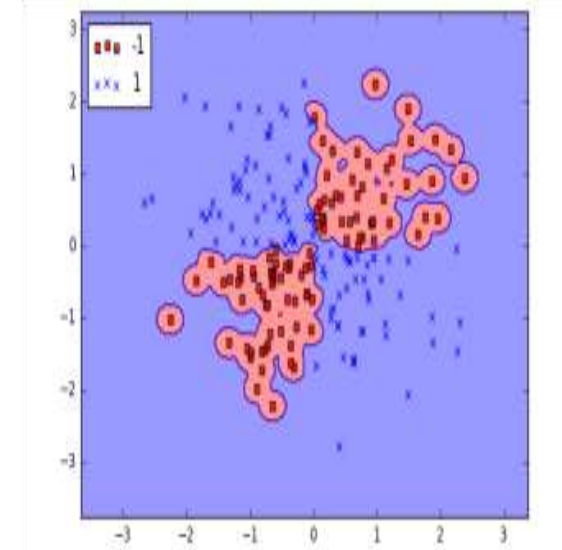


$\gamma=0.01$

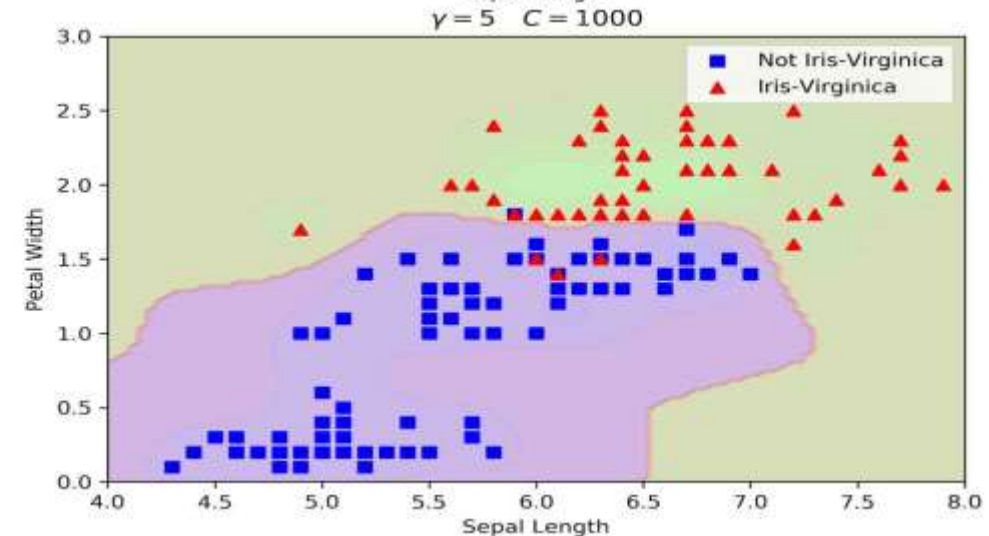
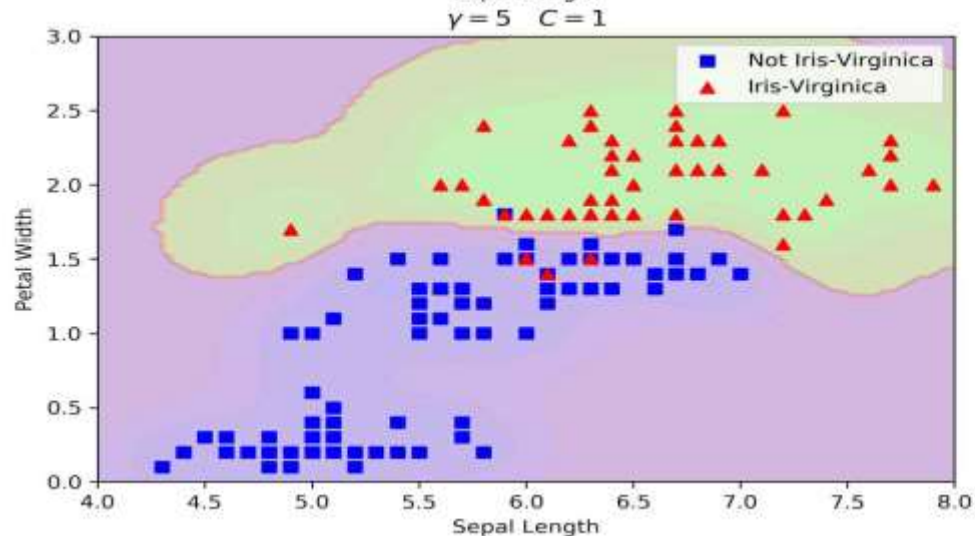
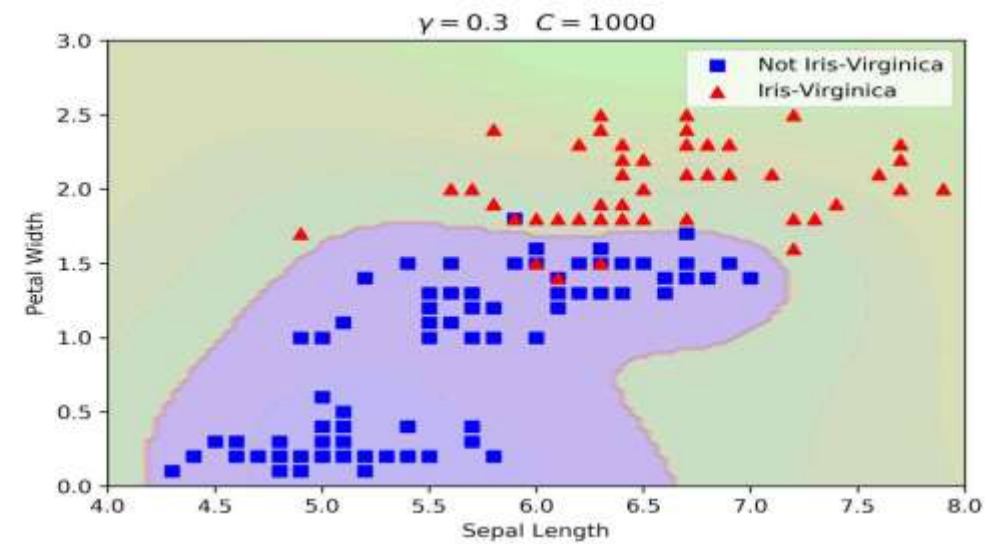
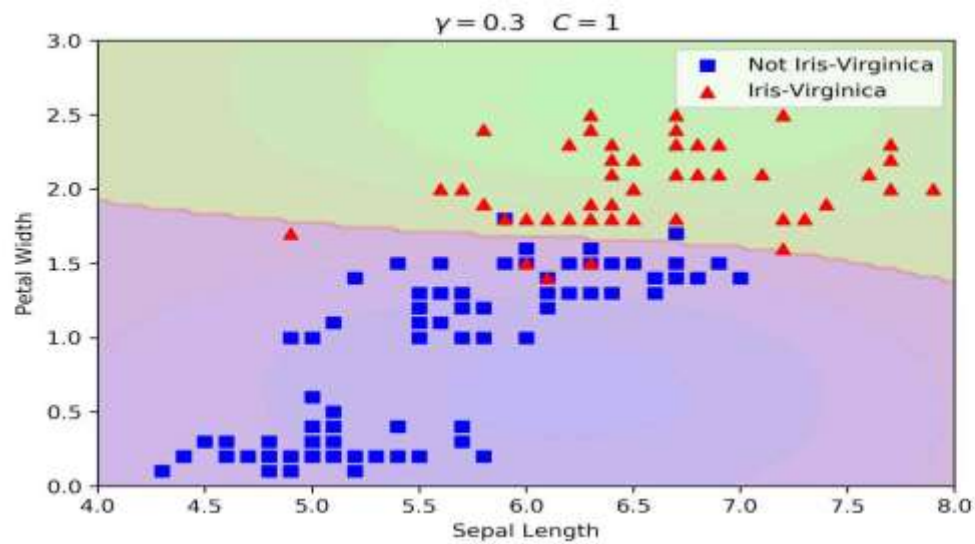
[SVM Classification](#)



$\gamma=1$



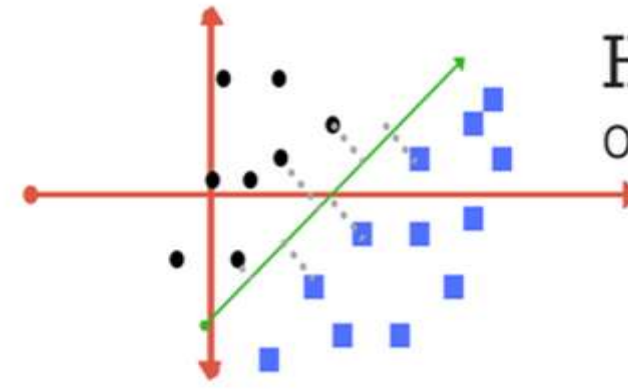
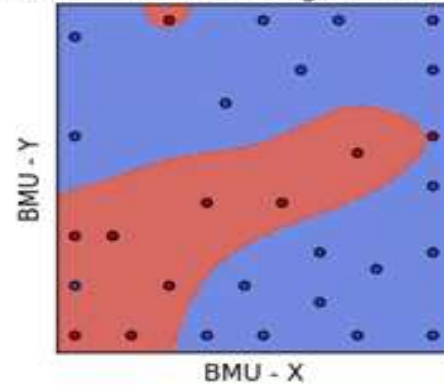
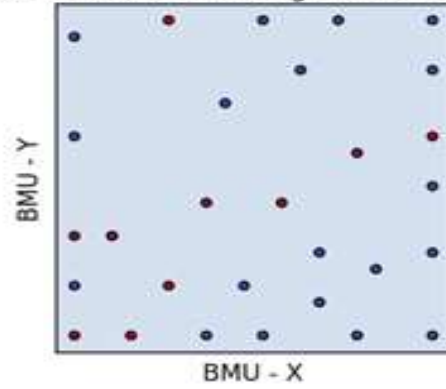
$\gamma=100$



When C is very large, the soft-margin SVM is equivalent to hard-margin SVM

When C is very small, misclassification in the training data at the expense of small norm

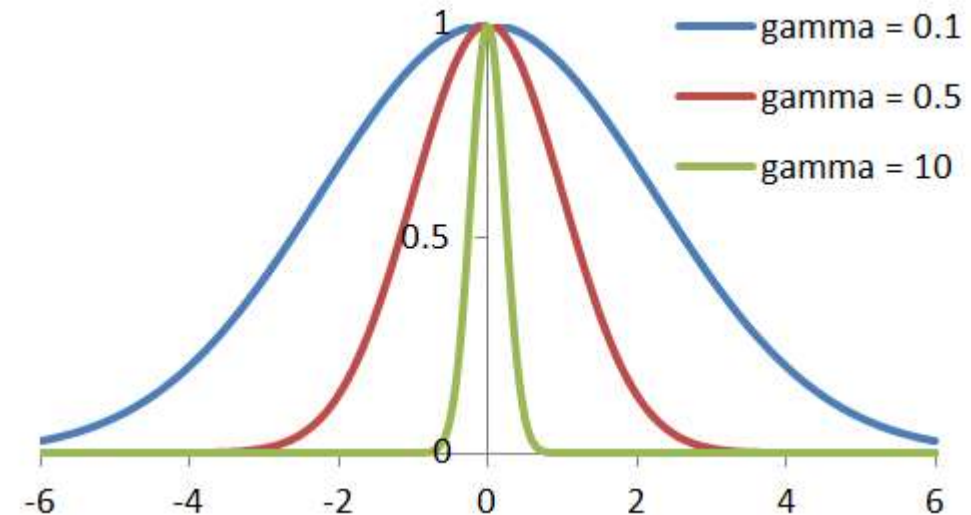
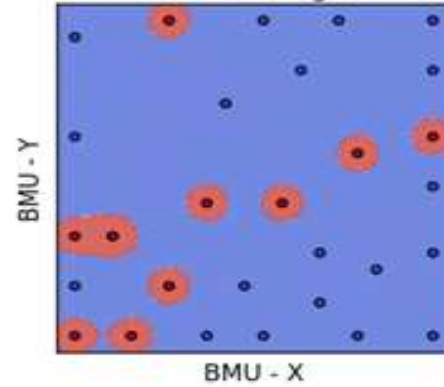
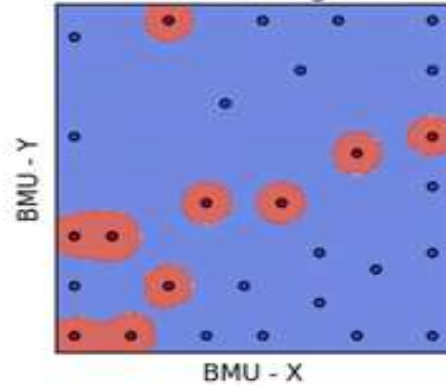
SVC with RBF kernel gamma = 0.00 SVC with RBF kernel gamma = 0.05





High Gamma
Only nearby points are considered.



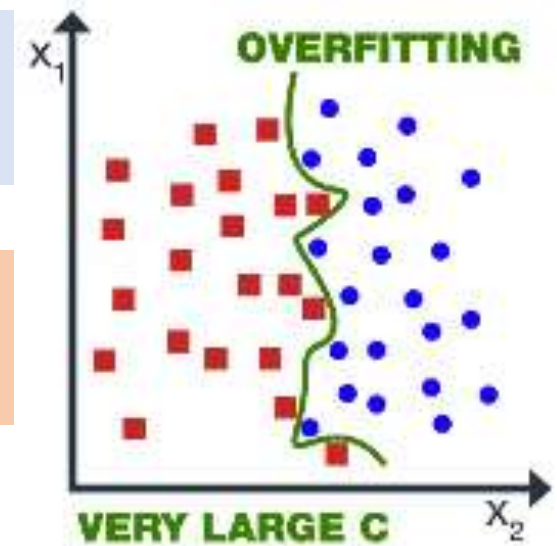
SVC with RBF kernel gamma = 0.5 SVC with RBF kernel gamma = 0.7



	Large Gamma	Small Gamma	Large C	Small C
Variance	Low	High	High	Low
Bias	High 	Low	Low 	High

High value of **Gamma** leads to more accuracy but **biased** results and vice-versa.

A large value of Cost parameter (**C**) indicates poor accuracy but low **bias** and vice-versa.



Properties of SVM

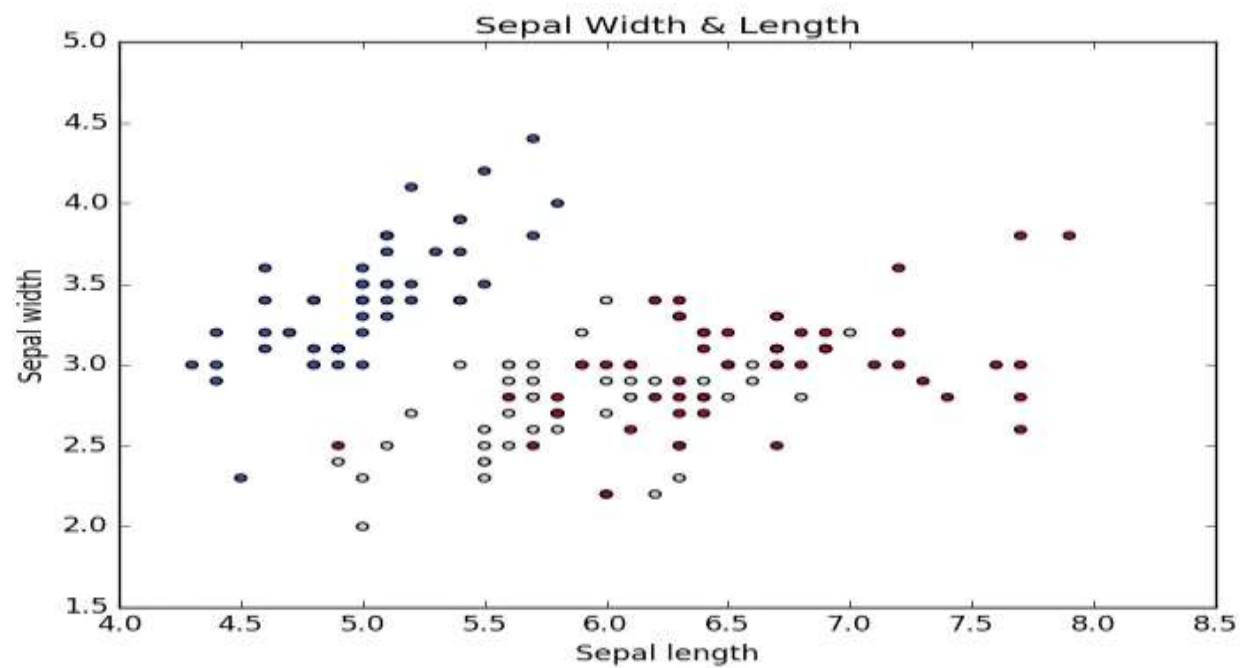


- **Flexibility in choosing a similarity function**
- Sparseness of solution when dealing with large data sets
 - only support vectors are used to specify the separating hyperplane
- **Ability to handle large feature spaces**
 - complexity does not depend on the dimensionality of the feature space
- **Overfitting can be controlled by soft margin approach**
- Nice math property: a simple convex optimization problem which is guaranteed to converge to a single global solution
- **Feature Selection**

Some Issues



- **Choice of kernel**
 - **Gaussian or polynomial kernel** is default
 - if ineffective, more elaborate kernels are needed
 - domain experts can give assistance in formulating appropriate similarity measures
- **Choice of kernel parameters**
 - e.g. σ in Gaussian kernel
 - **σ is the distance between closest points with different classifications**
 - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.
- **Optimization criterion – Hard margin v.s. Soft margin**
 - a lengthy series of experiments in which various parameters are tested



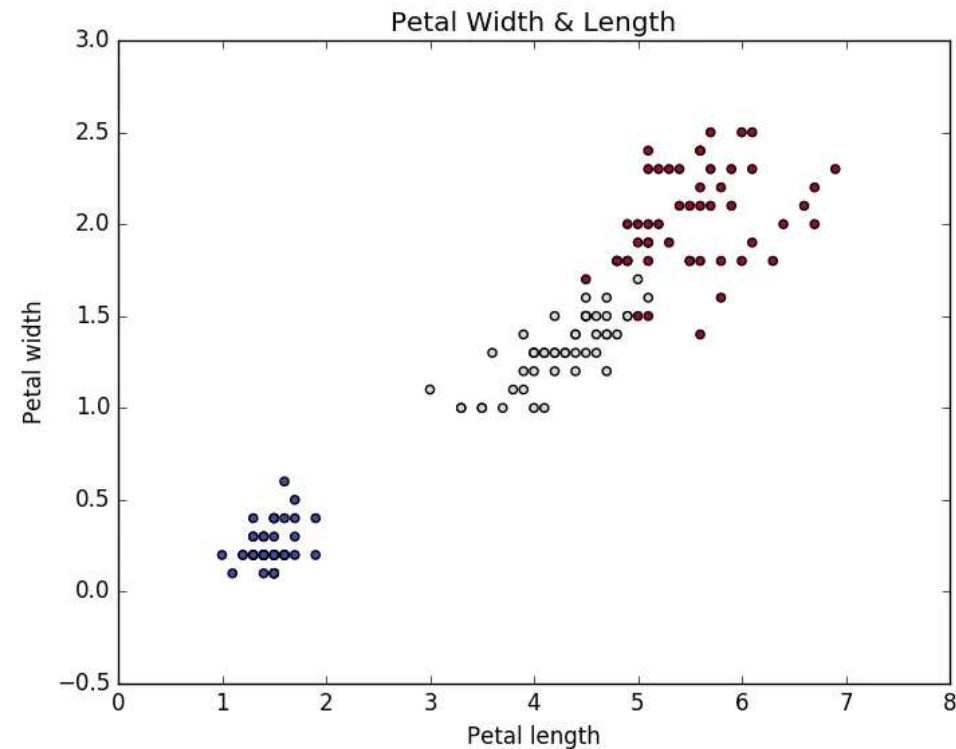
```
print iris.target .. , iris_dataset[ target ]
```

```
def visuvalize_sepal_data():  
    iris = datasets.load_iris()  
    X = iris.data[:, :2] # we only take the first two features.  
    y = iris.target  
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)  
    plt.xlabel('Sepal length')  
    plt.ylabel('Sepal width')  
    plt.title('Sepal Width & Length')  
    plt.show()  
visuvalize_sepal_data()
```

et
e svm



```
def visuvalize_petal_data():
    iris = datasets.load_iris()
    X = iris.data[:, 2:] # we only take the last two features.
    y = iris.target
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    plt.xlabel('Petal length')
    plt.ylabel('Petal width')
    plt.title('Petal Width & Length')
    plt.show()
visuvalize_petal_data()
```



```
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the Sepal two features.
y = iris.target
C = 1.0 # SVM regularization parameter
# SVC with linear kernel
svc = svm.SVC(kernel='linear', C=C).fit(X, y)
# LinearSVC (linear kernel)
lin_svc = svm.LinearSVC(C=C).fit(X, y)
# SVC with RBF kernel
rbf_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C).fit(X, y)
# SVC with polynomial (degree 3) kernel
poly_svc = svm.SVC(kernel='poly', degree=3, C=C).fit(X, y)
```



`h = .02 # step size in the mesh`

`# create a mesh to plot in`

`x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1`

`y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1`

`xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))`

`# title for the plots`

`titles = ['SVC with linear kernel',`

`'LinearSVC (linear kernel)',`

`'SVC with RBF kernel',`

`'SVC with polynomial (degree 3) kernel']`

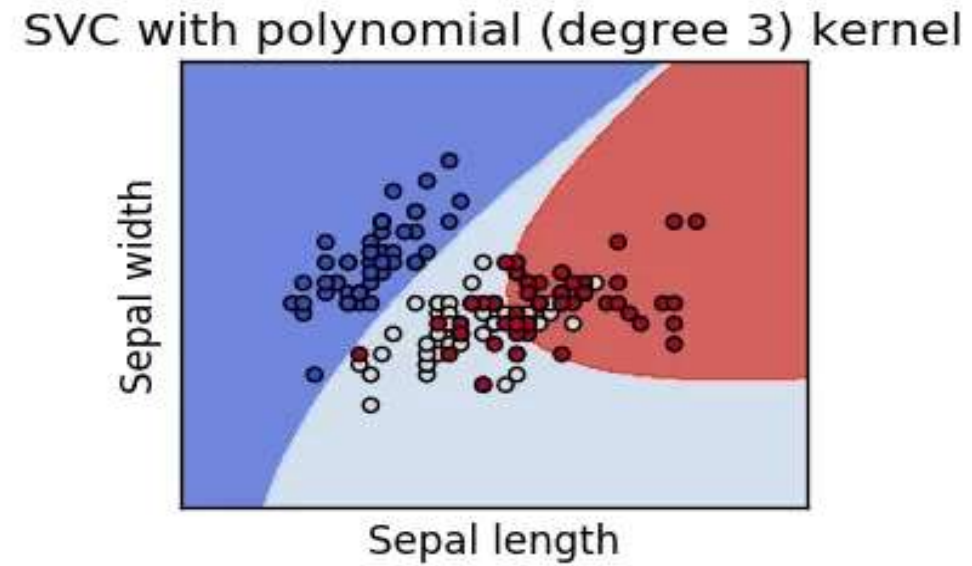
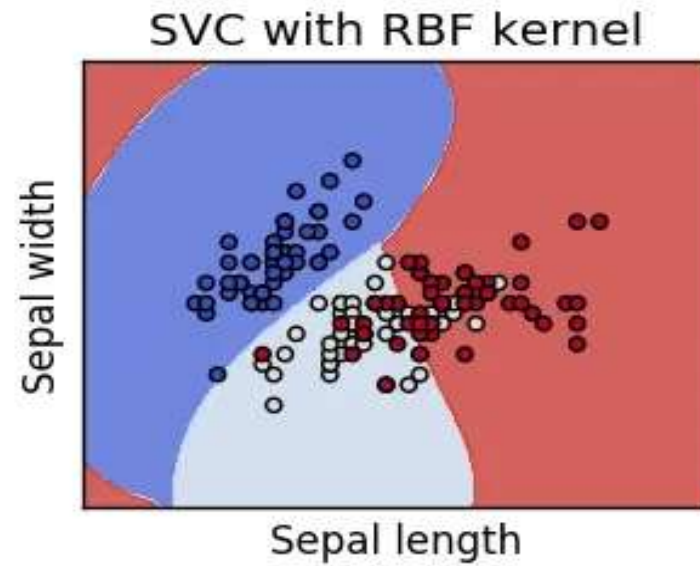
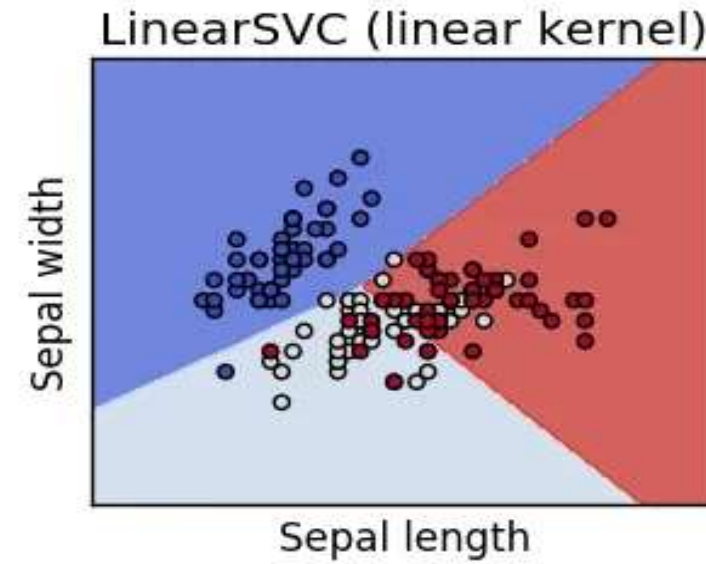
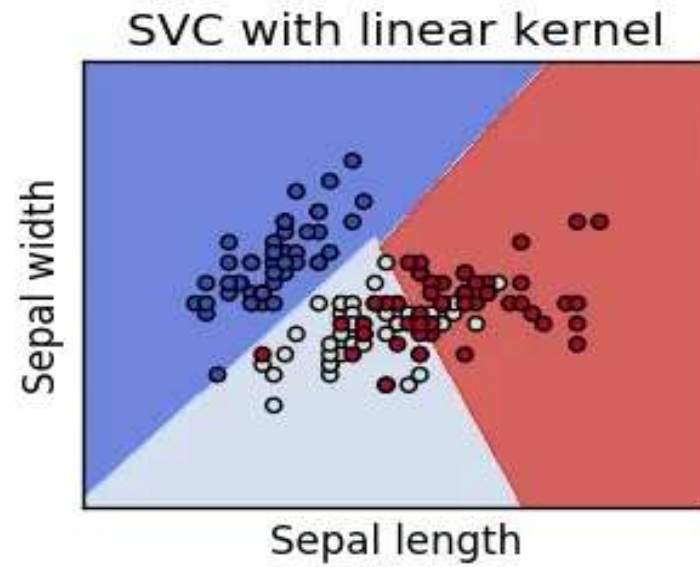
```
for i, clf in enumerate((svc, lin_svc, rbf_svc, poly_svc)):
    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    plt.subplot(2, 2, i + 1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(titles[i])

plt.show()
```




```
iris = datasets.load_iris()
X = iris.data[:, 2:] # we only take the last two features.
#loading the Petal features into X variable
y = iris.target
C = 1.0 # SVM regularization parameter

# SVC with linear kernel
svc = svm.SVC(kernel='linear', C=C).fit(X, y)
# LinearSVC (linear kernel)
lin_svc = svm.LinearSVC(C=C).fit(X, y)
# SVC with RBF kernel
rbf_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C).fit(X, y)
# SVC with polynomial (degree 3) kernel
poly_svc = svm.SVC(kernel='poly', degree=3, C=C).fit(X, y)
```



```
h = .02 # step size in the mesh
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
```

```
# title for the plots
titles = ['SVC with linear kernel',
          'LinearSVC (linear kernel)',
          'SVC with RBF kernel',
          'SVC with polynomial (degree 3) kernel']
```

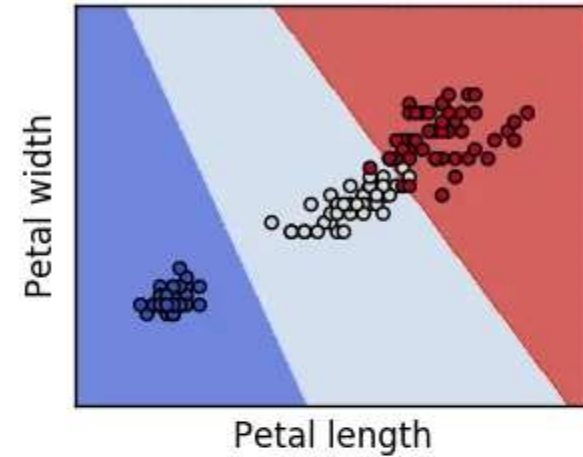
```
for i, clf in enumerate((svc, lin_svc, rbf_svc, poly_svc)):
    # Plot the decision boundary. For that, we will assign
    # a color to each point in the mesh [x_min, x_max][y_min, y_max].
    plt.subplot(2, 2, i + 1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

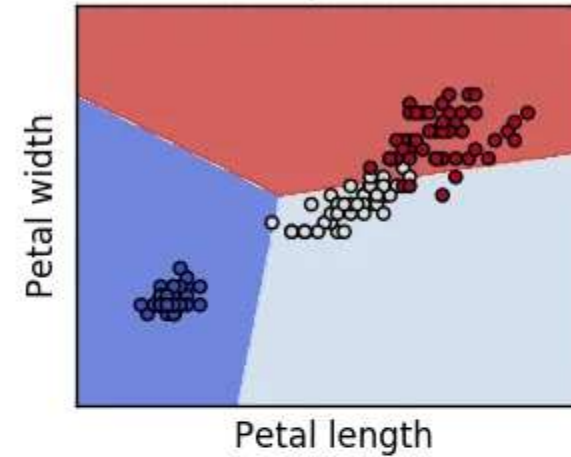
    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    plt.xlabel('Petal length')
    plt.ylabel('Petal width')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(titles[i])
plt.show()
```

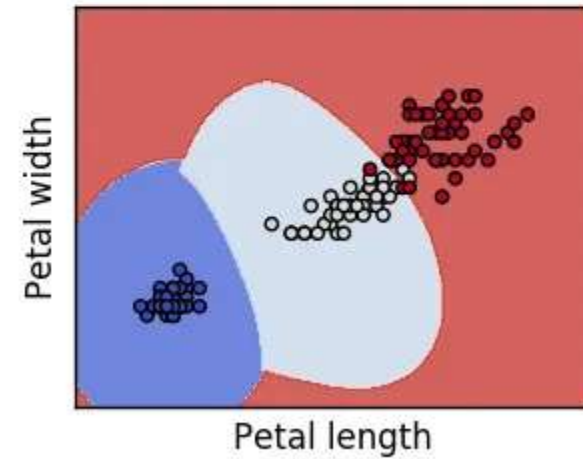
SVC with linear kernel



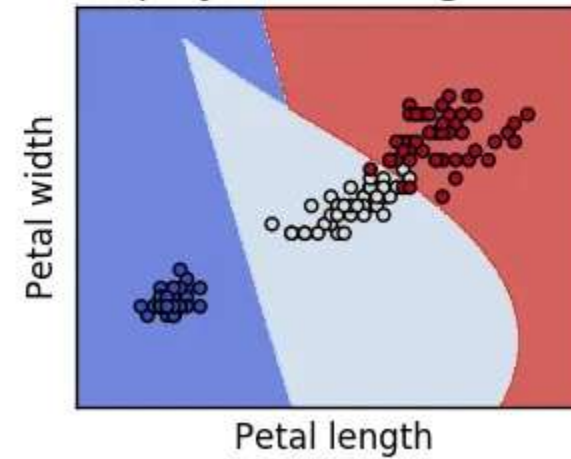
LinearSVC (linear kernel)



SVC with RBF kernel



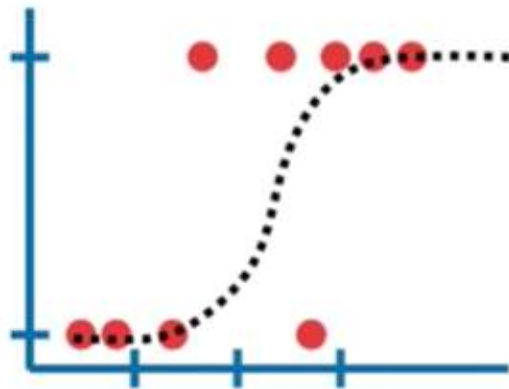
SVC with polynomial (degree 3) kernel



Which one is better?

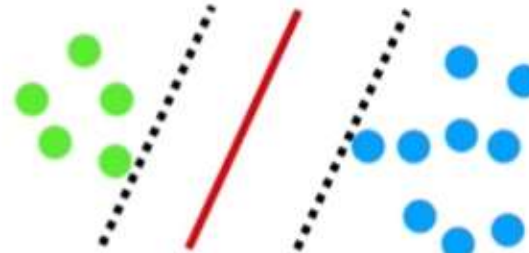


Logistic Regression



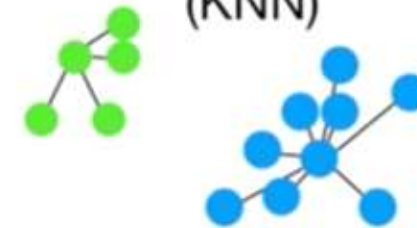
Correct	Incorrect
16	8

Support Vector machines (SVM)



Correct	Incorrect
18	6

K-nearest neighbors (KNN)



Correct	Incorrect
10	12

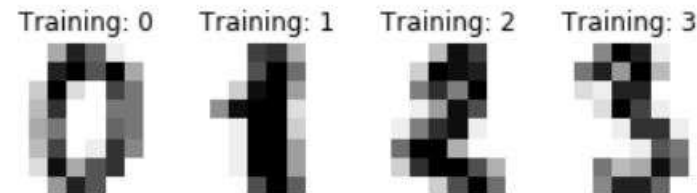
Additional example(optional)



```
In [1]: import matplotlib.pyplot as plt
        from sklearn import datasets
        from sklearn import svm
        from sklearn import datasets, svm, metrics
        iris = datasets.load_iris()
        digits = datasets.load_digits()
```

```
In [2]: # The digits dataset
        digits = datasets.load_digits()
```

```
In [3]: #images` attribute of the dataset 8x8
        images_and_labels = list(zip(digits.images, digits.target))
        for index, (image, label) in enumerate(images_and_labels[:4]):
            plt.subplot(2, 4, index + 1)
            plt.axis('off')
            plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
            plt.title('Training: %i' % label)
```



In [4]: *#flatten the image, to turn the data in a (samples, feature) matrix:*

```
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))
```

In [5]: classifier = svm.SVC(gamma=0.001)

In [6]: *# We Learn the digits on the first half of the digits*
 classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])

Out[6]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
 decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
 max_iter=-1, probability=False, random_state=None, shrinking=True,
 tol=0.001, verbose=False)

In [7]: *# Now predict the value of the digit on the second half:*
 expected = digits.target[n_samples // 2:]
 predicted = classifier.predict(data[n_samples // 2:])

In [8]:

```
print("Classification report for classifier %s:\n%s\n"
      % (classifier, metrics.classification_report(expected, predicted)))
print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected, predicted))
```

```
Classification report for classifier SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
max_iter=-1, probability=False, random_state=None, shrinking=True,
tol=0.001, verbose=False):
```

	precision	recall	f1-score	support
0	1.00	0.99	0.99	88
1	0.99	0.97	0.98	91
2	0.99	0.99	0.99	86
3	0.98	0.87	0.92	91
4	0.99	0.96	0.97	92
5	0.95	0.97	0.96	91
6	0.99	0.99	0.99	91
7	0.96	0.99	0.97	89
8	0.94	1.00	0.97	88
9	0.93	0.98	0.95	92
avg / total	0.97	0.97	0.97	899

Confusion matrix:

```
[[87  0  0  0  1  0  0  0  0  0]
 [ 0 88  1  0  0  0  0  0  1  1]
 [ 0  0 85  1  0  0  0  0  0  0]
 [ 0  0  0 79  0  3  0  4  5  0]
 [ 0  0  0  0 88  0  0  0  0  4]
 [ 0  0  0  0  0 88  1  0  0  2]
 [ 0  1  0  0  0  0 90  0  0  0]
 [ 0  0  0  0  0  1  0 88  0  0]
 [ 0  0  0  0  0  0  0  0 88  0]
 [ 0  0  0  1  0  1  0  0  0 90]]
```

SVM Classification




```
In [9]: images_and_predictions = list(zip(digits.images[n_samples // 2:], predicted))
        for index, (image, prediction) in enumerate(images_and_predictions[:4]):
            plt.subplot(2, 4, index + 5)
            plt.axis('off')
            plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
            plt.title('Prediction: %i' % prediction)

        plt.show()
```

