Machine Learning I

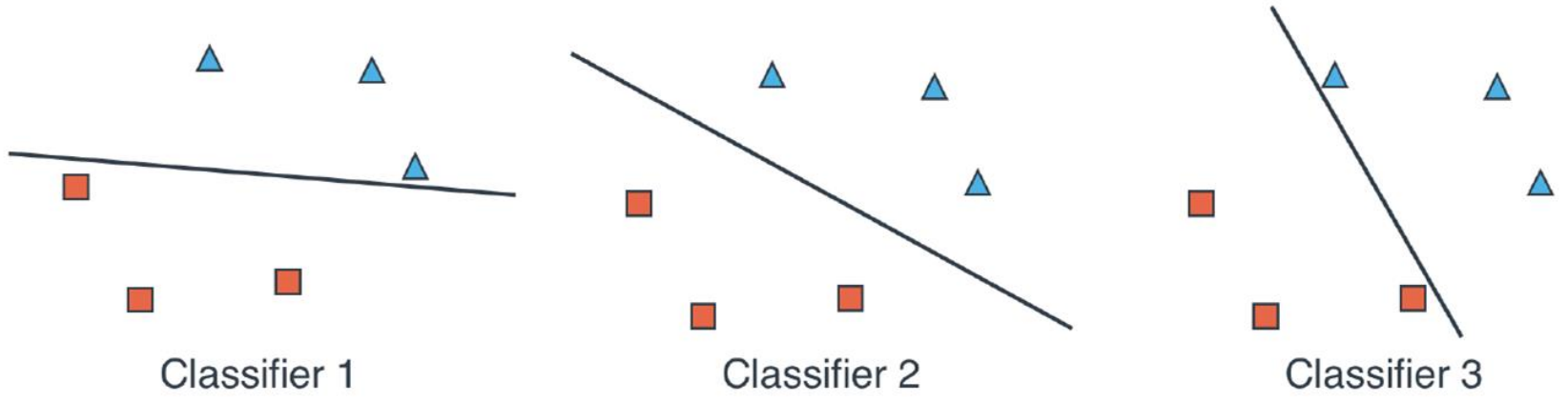
Mohamed Hussien

Support Vector Machine (SVM)

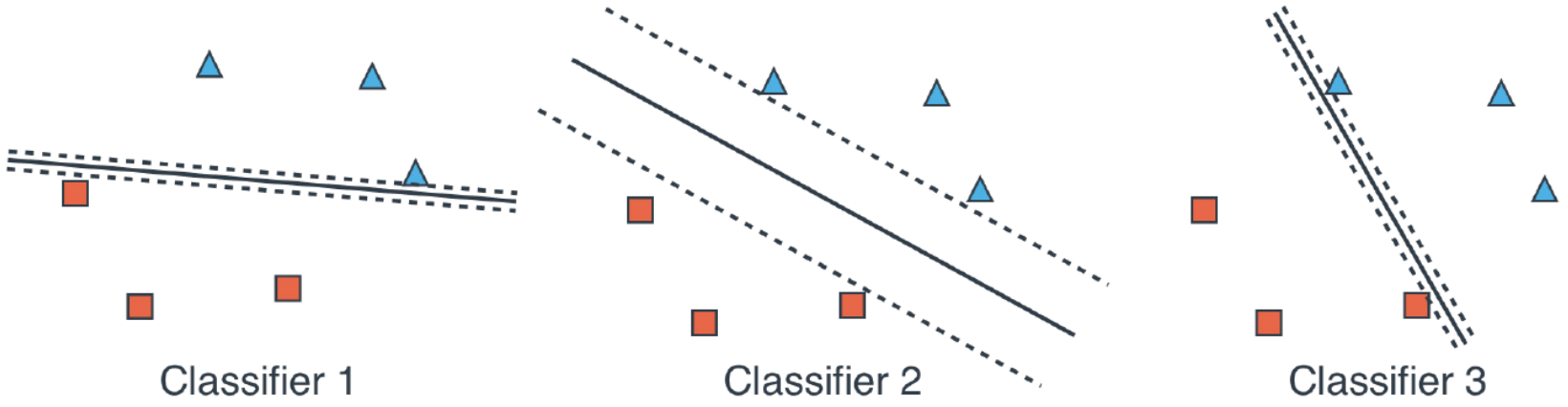
Lecture Overview

- Problem Definition
- SVM Error Function
- Classification Error
- Distance Error
- Linear SVM coding
- Polynomial Kernel
- Polynomial Kernel Coding
- RBF Kernel
- RBF Kernel Coding
- Hinge Loss
- Support Vector Regression (SVR)

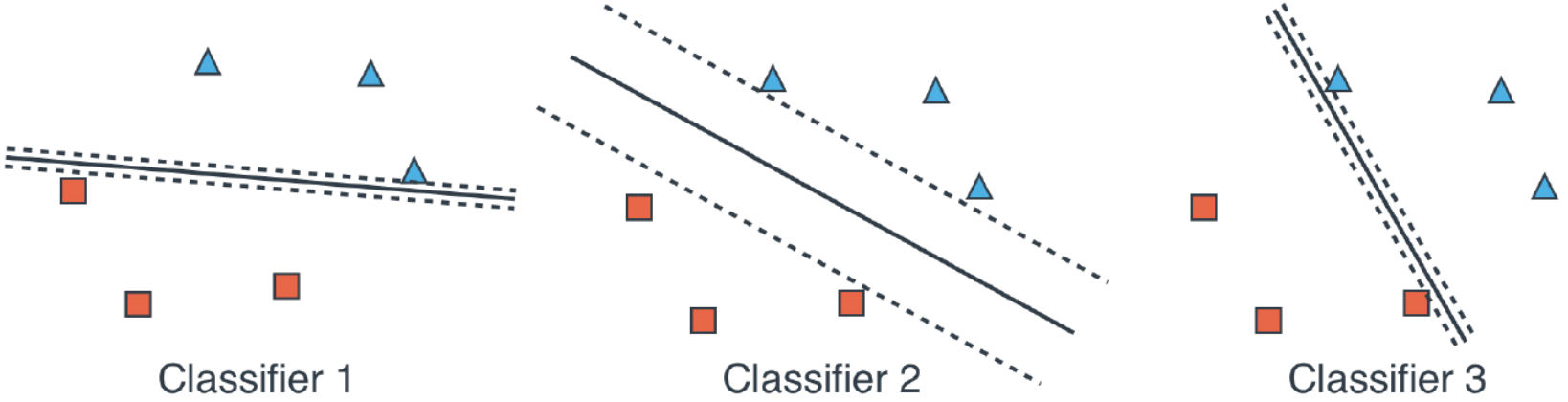
Which is the best classifier?



Which is the best classifier?



Which is the best classifier?



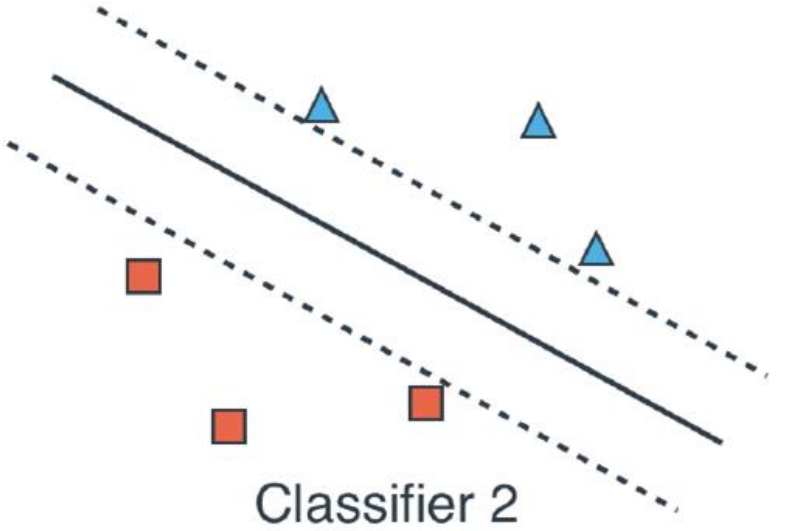
All three lines separate the dataset well. Since as you can see, the first and the third line are very close to some of the points, while the second line is **not close to any points**.

How to build a classifier like that? **Here comes the SVM**

Lecture Overview

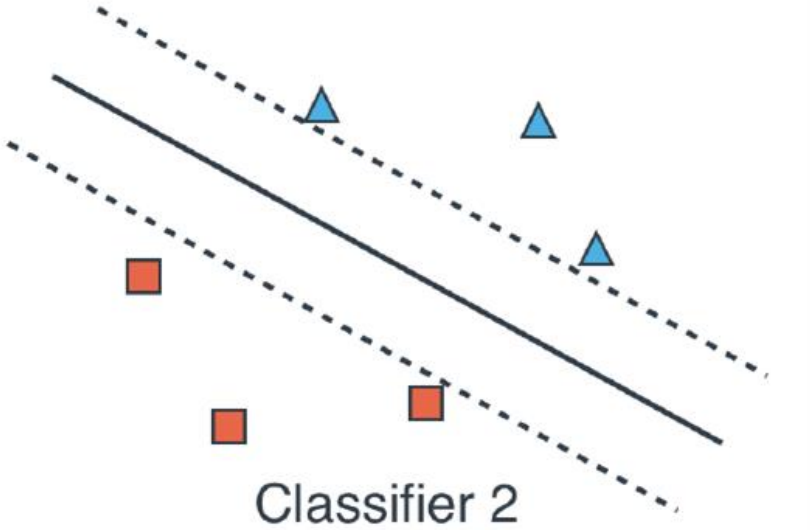
- **Problem Definition**
- **SVM Error Function**
- **Classification Error**
- **Distance Error**
- **Linear SVM coding**
- **Polynomial Kernel**
- **Polynomial Kernel Coding**
- **RBF Kernel**
- **RBF Kernel Coding**
- **Hinge Loss**
- **Support Vector Regression (SVR)**

SVM Error Function



SVM Error Function

What do we need from our model?



SVM Error Function

What do we need from our model?

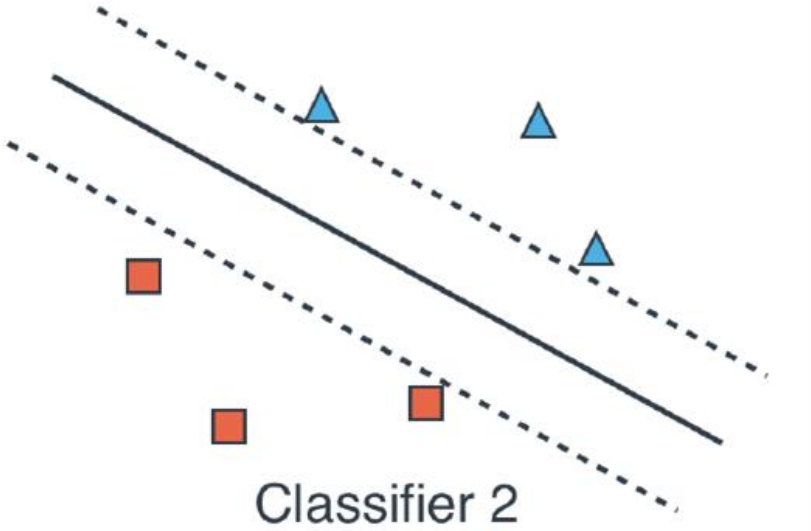
Error Function

=

Classification Error

+

Distance Error



Lecture Overview

- Problem Definition
- SVM Error Function
- Classification Error
- Distance Error
- Linear SVM coding
- Polynomial Kernel
- Polynomial Kernel Coding
- RBF Kernel
- RBF Kernel Coding
- Hinge Loss
- Support Vector Regression (SVR)

Classification Error Function

Classification Error Function

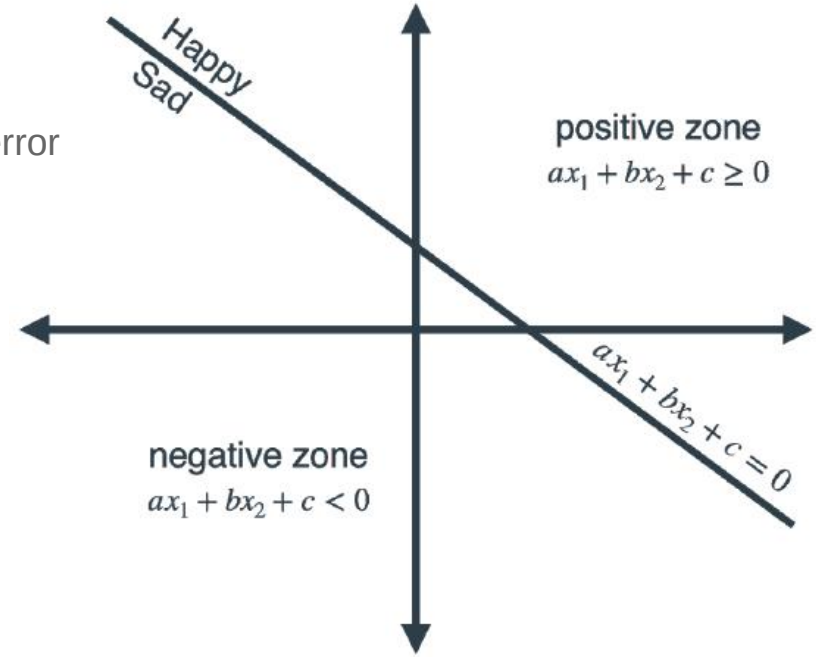
Back to logistic regression..

- We used a line to separate the points
- Our goal was to decrease this line cross entropy error

Classification Error Function

Back to logistic regression..

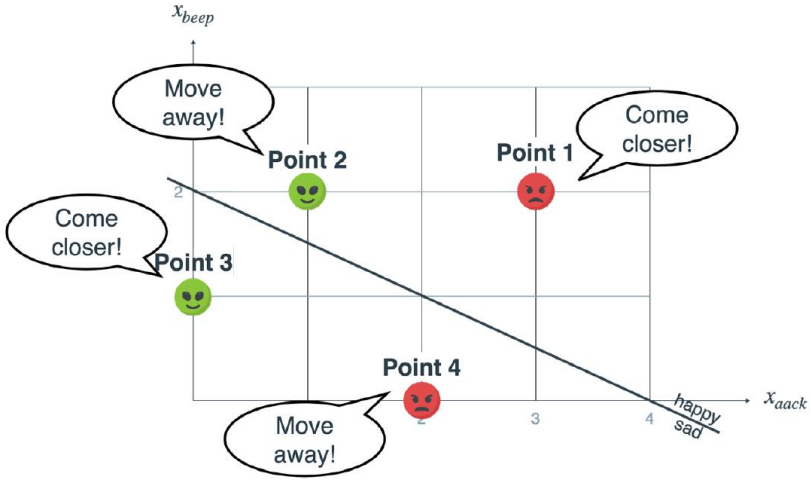
- We used a line to separate the points
- Our goal was to decrease this line cross entropy error

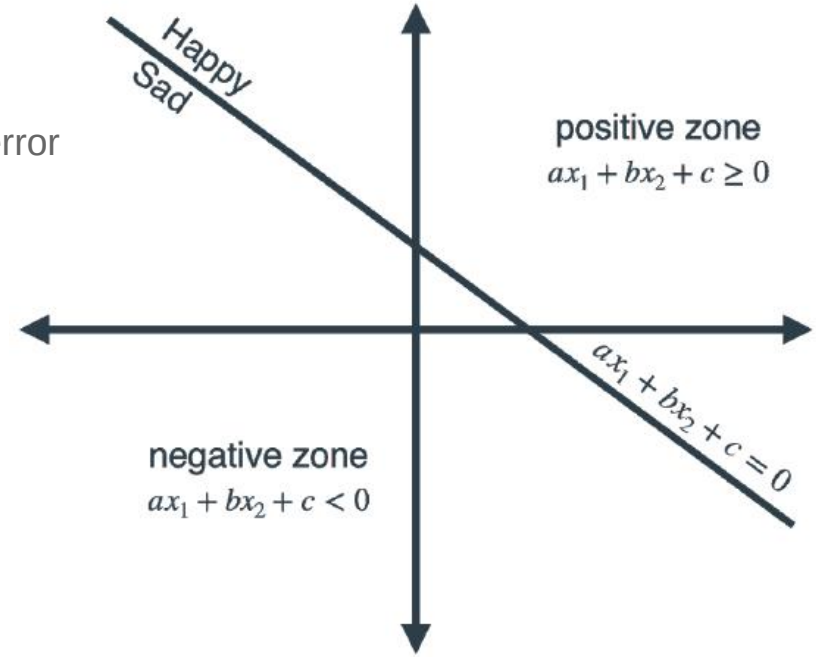


Classification Error Function

Back to logistic regression..

- We used a line to separate the points
- Our goal was to decrease this line cross entropy error





Classification Error Function

Classification Error Function

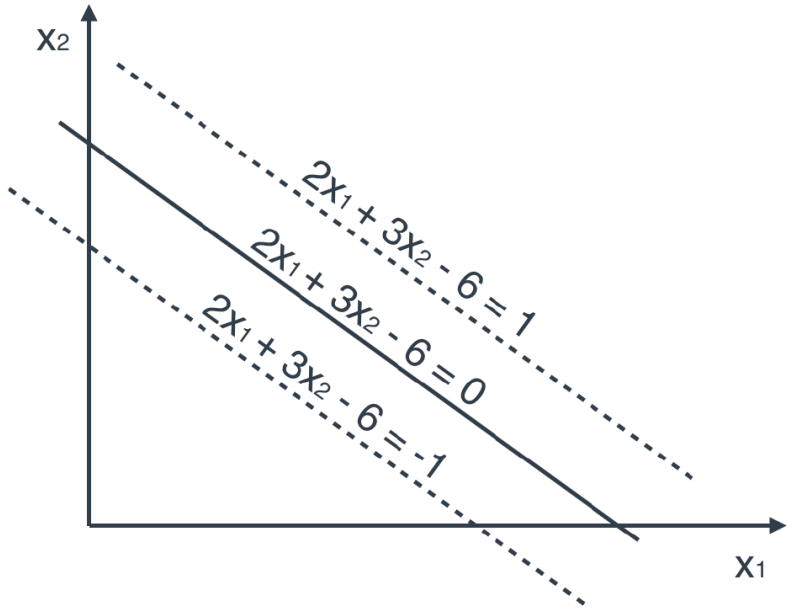
- In SVM, Instead of using one line, we use two parallel lines shifted by bias.
  $L+: W_1X_1 + W_2X_2 + b = 1$
  $L-: W_1X_1 + W_2X_2 + b = -1$

- Example:
  $L: 2X_1 + 3X_2 + 6 = 0$
  $L+: 2X_1 + 3X_2 + 6 = 1$
  $L-: 2X_1 + 3X_2 + 6 = -1$

Classification Error Function

- In SVM, Instead of using one line, we use two parallel lines shifted by bias.
  L+: $W_1X_1 + W_2X_2 + b = 1$
  L-: $W_1X_1 + W_2X_2 + b = -1$

- Example:
  L: $2X_1 + 3X_2 + 6 = 0$
  L+: $2X_1 + 3X_2 + 6 = 1$
  L-: $2X_1 + 3X_2 + 6 = -1$



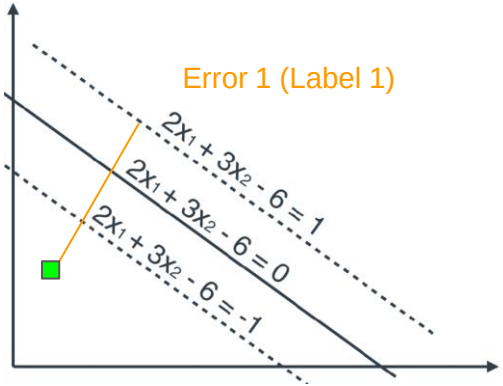
Classification Error Function

Classification Error Function

The goal of this classifier is to have **the two lines separate the points as best as possible**.

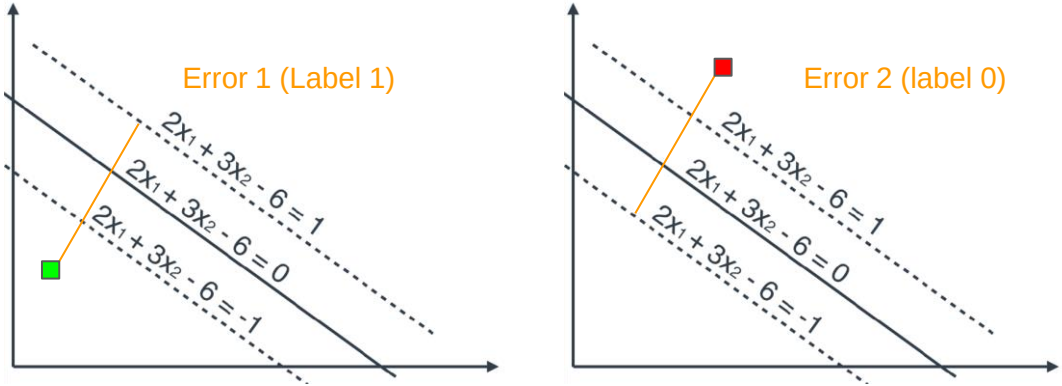
Classification Error Function

The goal of this classifier is to have **the two lines separate the points as best as possible**.



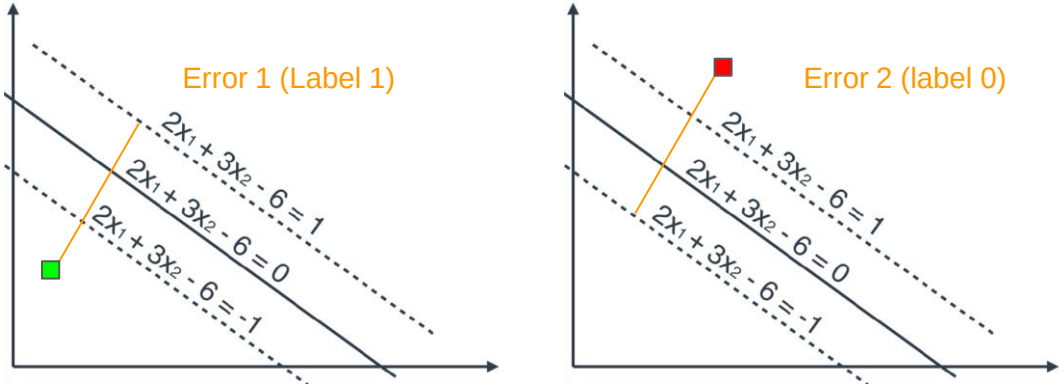
Classification Error Function

The goal of this classifier is to have **the two lines separate the points as best as possible**.



Classification Error Function

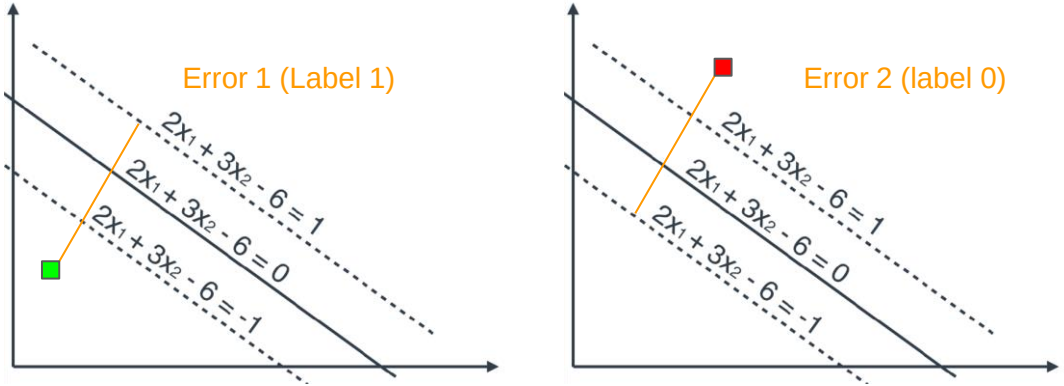
The goal of this classifier is to have **the two lines separate the points as best as possible**.



Error = Error 1 + Error 2

Classification Error Function

The goal of this classifier is to have **the two lines separate the points as best as possible**.



Error = Error 1 + Error 2

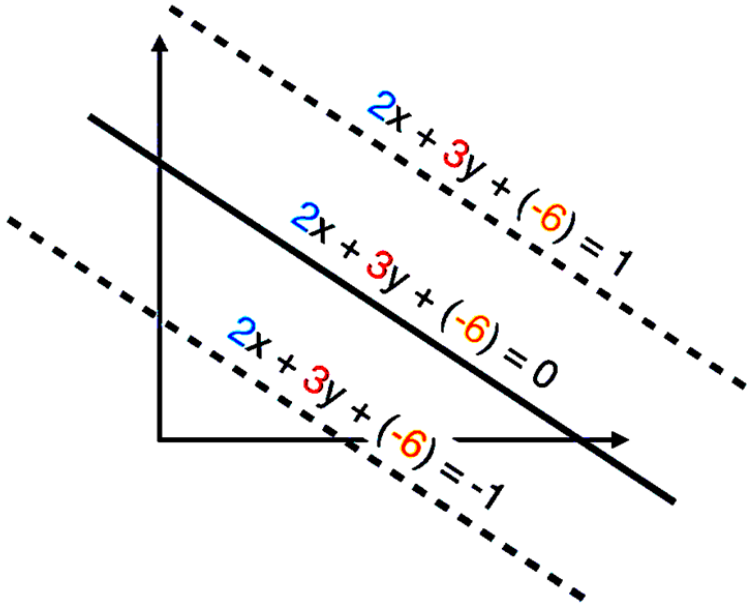
Notice that any point between the two lines is always misclassified by one of the lines

Why we need a distance error?

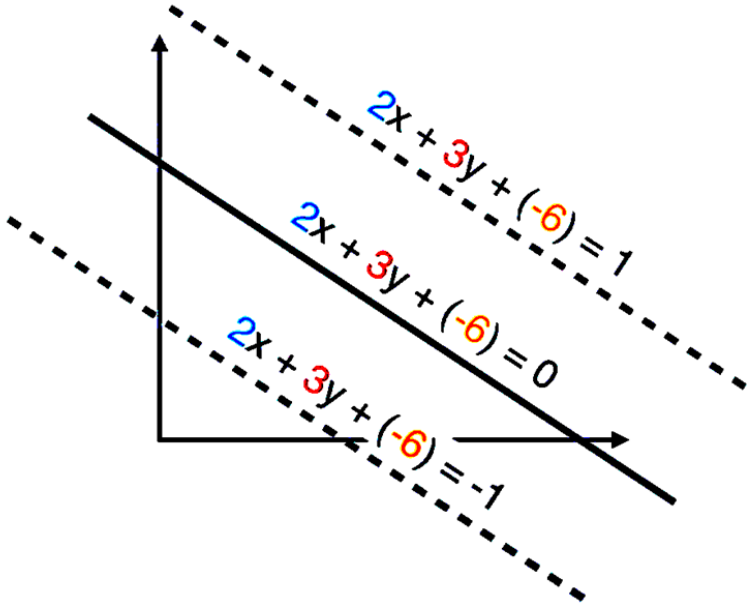
Lecture Overview

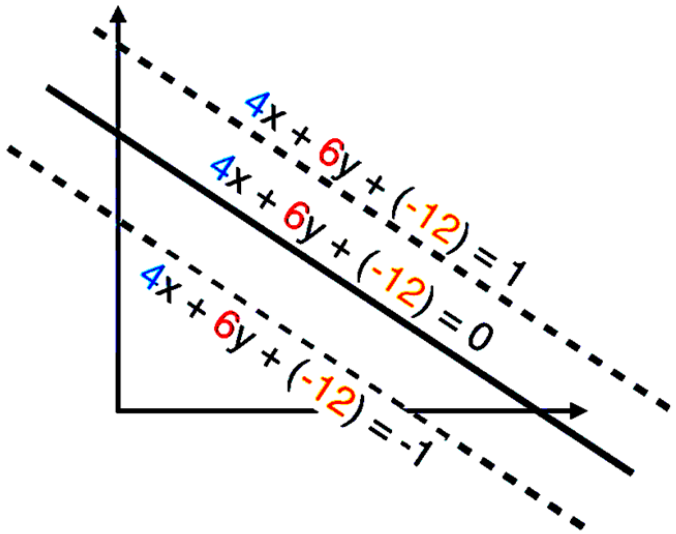
- Problem Definition
- SVM Error Function
- Classification Error
- Distance Error
- Linear SVM coding
- Polynomial Kernel
- Polynomial Kernel Coding
- RBF Kernel
- RBF Kernel Coding
- Hinge Loss
- Support Vector Regression (SVR)

Distance Error Function

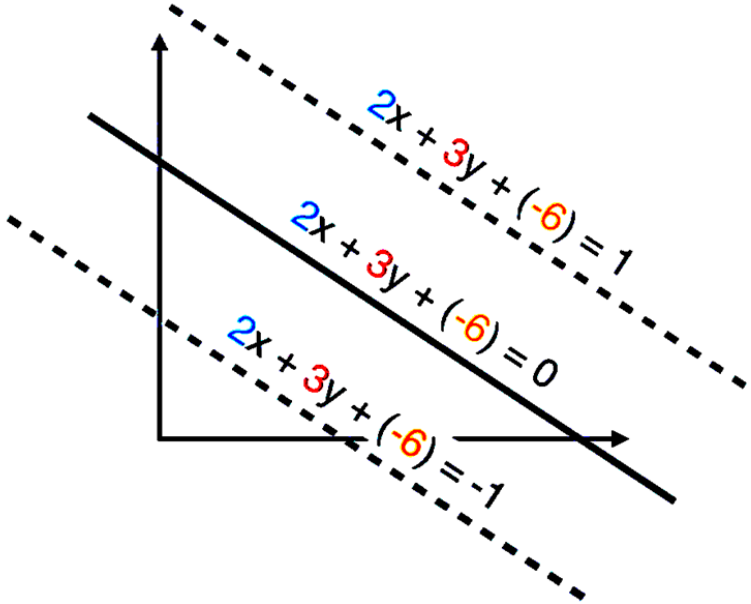


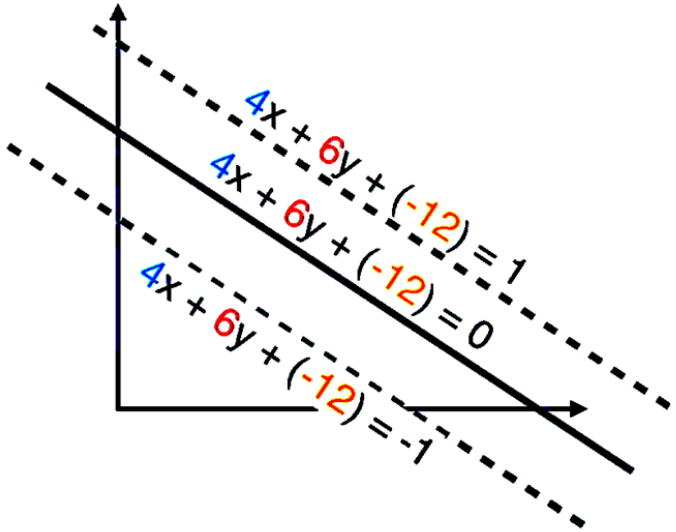
Distance Error Function





Distance Error Function





Changing weight changes the distance between the two lines, so we need an error to reduce the distance

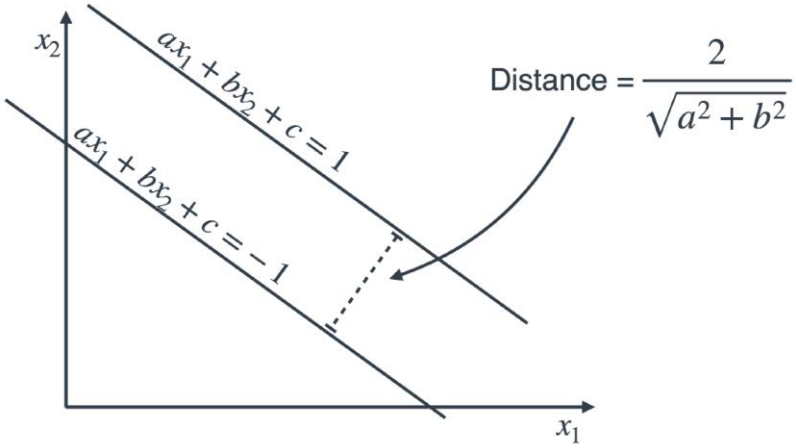
Distance Error Function

Distance Error Function

Distance between two parallel lines $= \dfrac{|c_2 - c_1|}{\sqrt{a^2 + b^2}}$

Distance Error Function

Distance between two parallel lines $= \dfrac{|c_2 - c_1|}{\sqrt{a^2 + b^2}}$



Distance Error Function

What formula can we use as a distance error?

Distance error function is a^2+b^2

If a^2+b^2 is small, distance is large => **good classifier**

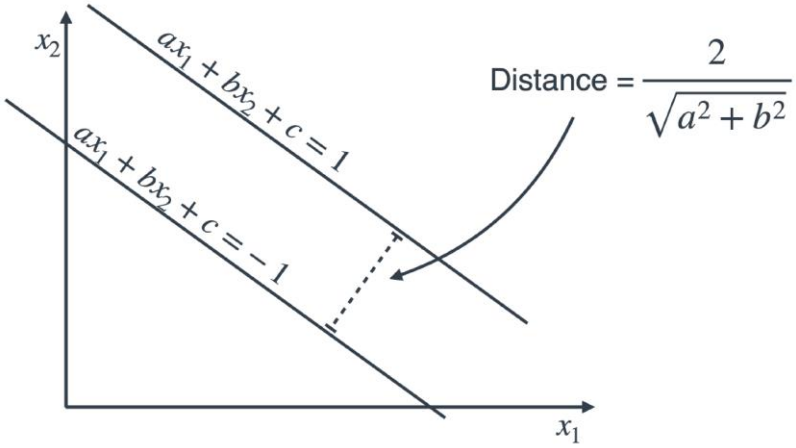
If a^2+b^2 is large, distance is small => **bad classifier**

Remember you of something?

L2 Norm regularization term

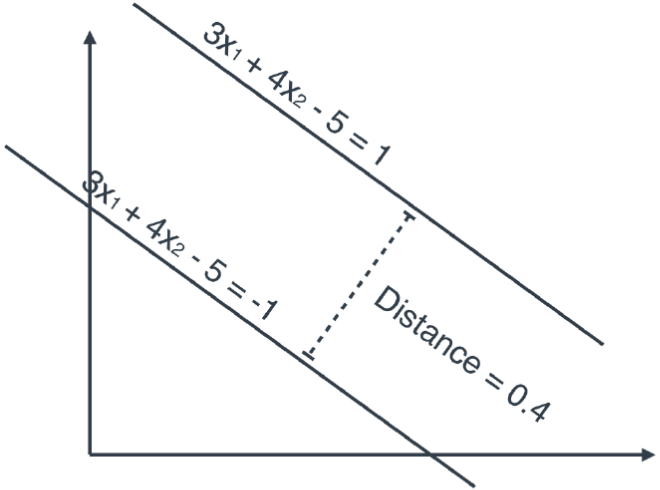
Weight update due to distance error term:
a'= a - 2a
b'= b - 2b

Distance between two parallel lines $= \frac{|c_2 - c_1|}{\sqrt{a^2 + b^2}}$

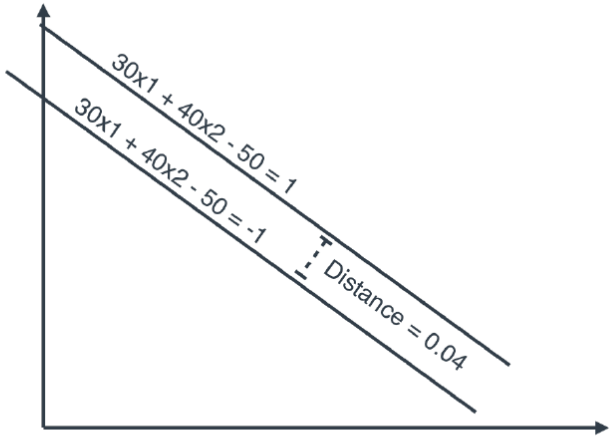


Distance Error Function

As weights become near to zero, as distance increased.



Error function = 25
Good classifier



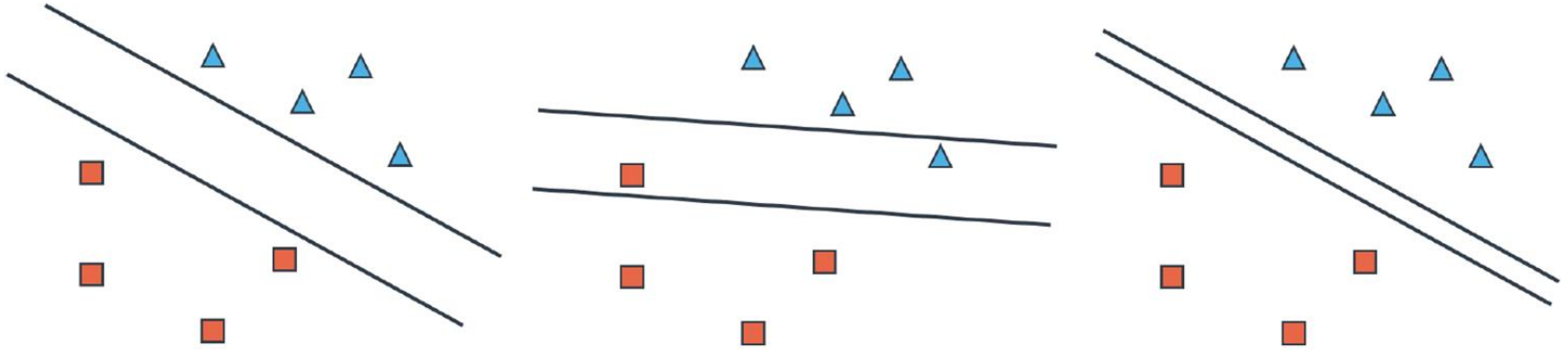
Error function = 2500
Bad classifier

SVM Error Function [Adding the two errors]

Error = Classification Error + Distance Error

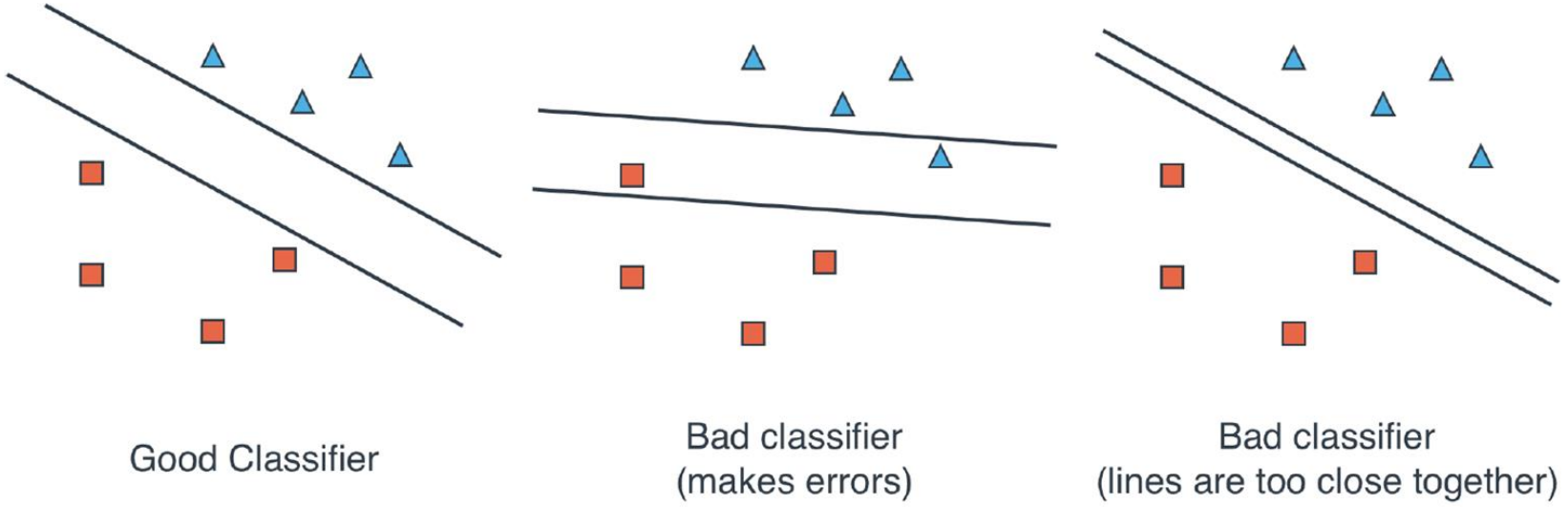
SVM Error Function [Adding the two errors]

Error = Classification Error + Distance Error



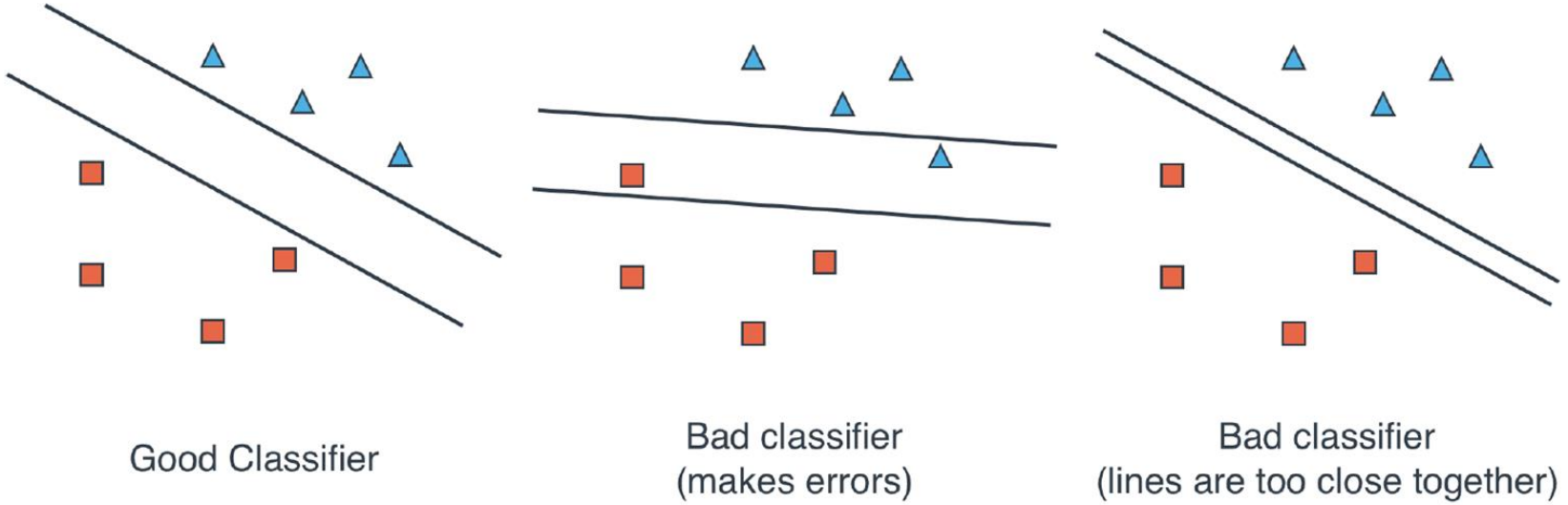
SVM Error Function [Adding the two errors]

Error = Classification Error + Distance Error



SVM Error Function [Adding the two errors]

Error = Classification Error + Distance Error



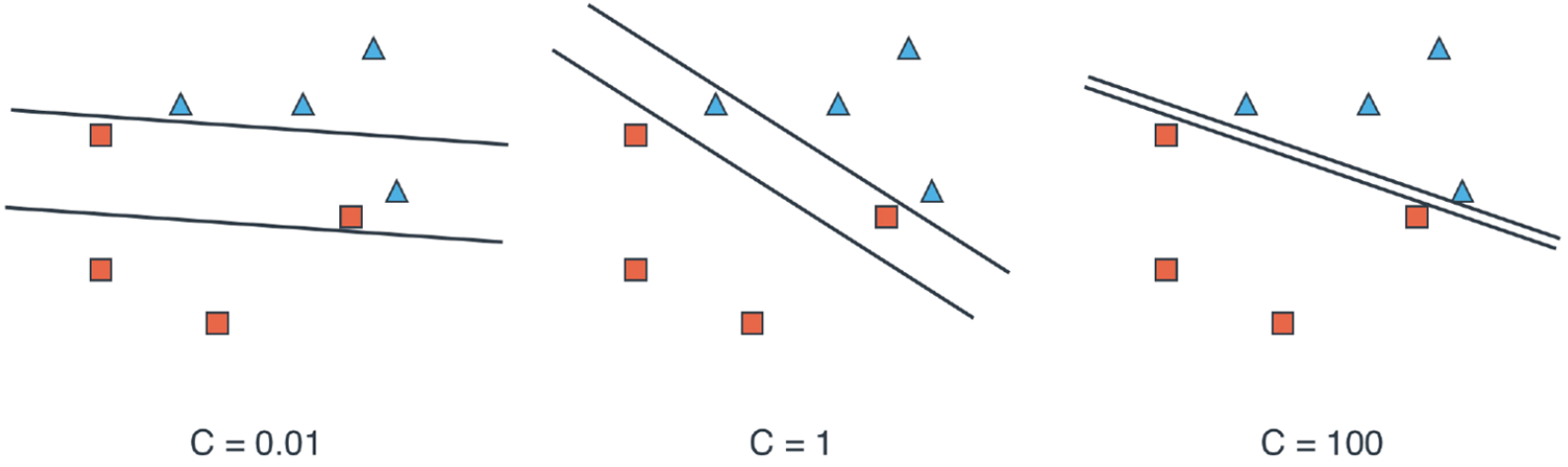
What is more important, classification error or distance error? And how much?

SVM Error Function [**C parameter**]

Error formula = C * (Classification Error) + (Distance Error)

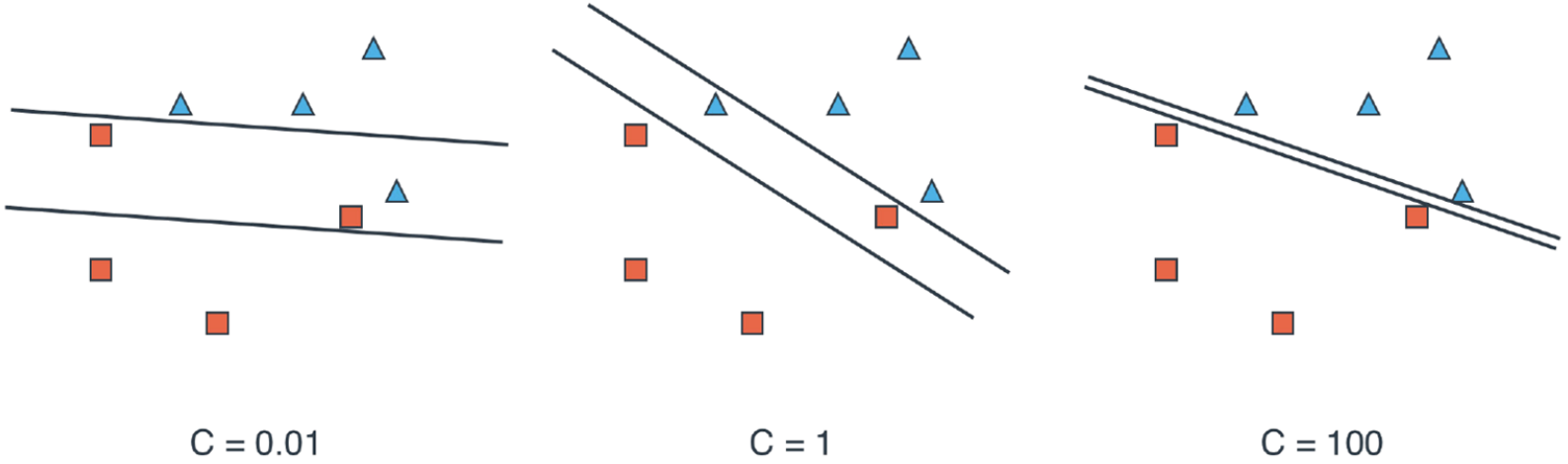
SVM Error Function [**C parameter**]

Error formula = C * (Classification Error) + (Distance Error)



SVM Error Function [C parameter]

Error formula = C * (Classification Error) + (Distance Error)



How C parameter affects overfitting?

C parameter ↑ , overfitting ↑

C=1/λ (λ:regularization parameter)

Lecture Overview

- Problem Definition
- SVM Error Function
- Classification Error
- Distance Error
- Linear SVM coding
- Polynomial Kernel
- Polynomial Kernel Coding
- RBF Kernel
- RBF Kernel Coding
- Hinge Loss
- Support Vector Regression (SVR)

Linear SVM Coding

Use colab to open this github notebook:

“s7s/machine_learning_1/support_vector_machines/SVM_coding.ipynb”

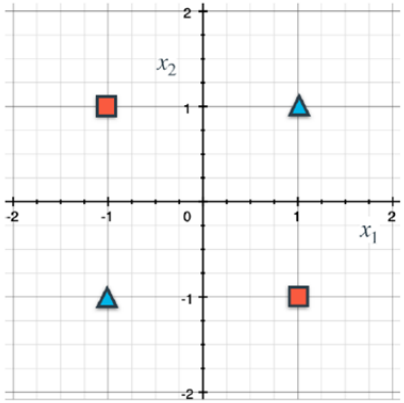
SKlearn [Linear SVM]

```
# C = 0.01
svm_c_001 = SVC(kernel='linear', C=0.01)
svm_c_001.fit(features, labels)
print("C = 0.01")
print("Accuracy:", svm_c_001.score(features, labels))
```

Lecture Overview

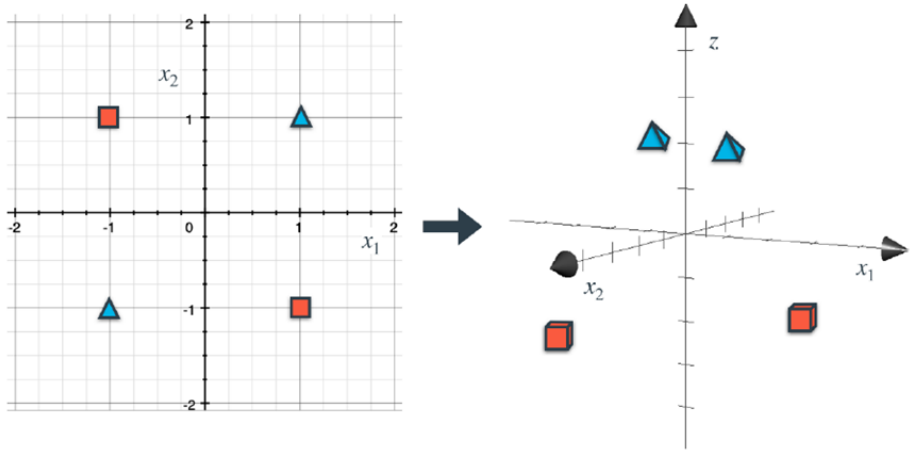
- Problem Definition
- SVM Error Function
- Classification Error
- Distance Error
- Linear SVM coding
- Polynomial Kernel
- Polynomial Kernel Coding
- RBF Kernel
- RBF Kernel Coding
- Hinge Loss
- Support Vector Regression (SVR)

The Kernel Trick



Not separable
by a line

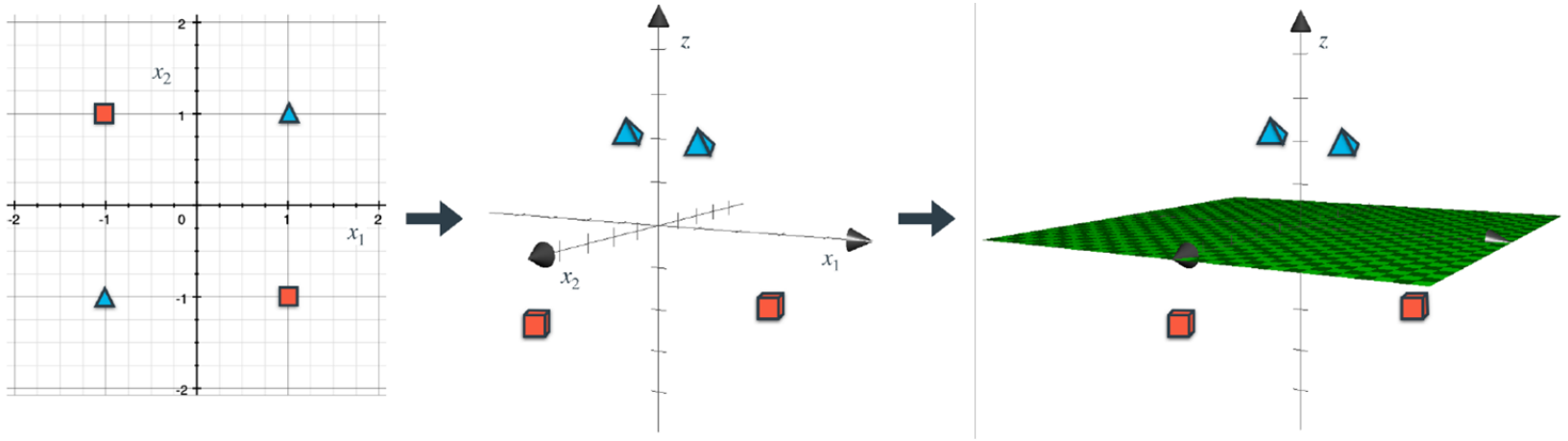
The Kernel Trick



Not separable
by a line

Bring triangles up
Bring squares down

The Kernel Trick



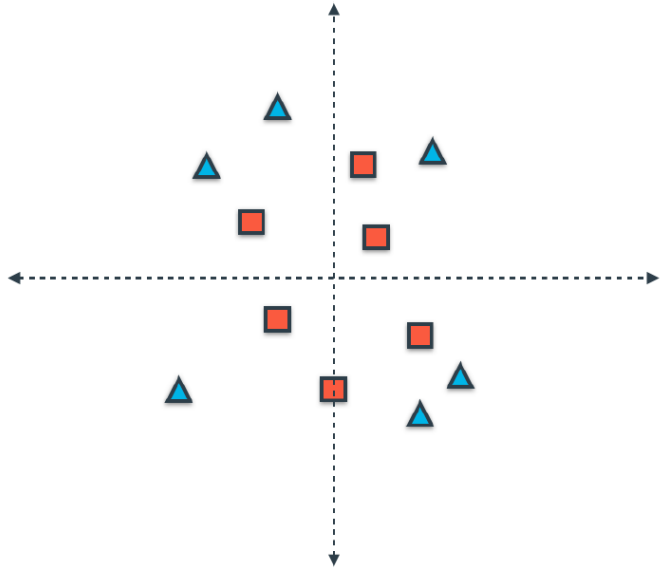
Not separable
by a line

Bring triangles up
Bring squares down

Now separable
by a plane

The Kernel Trick [polynomial equations (circles, parabolas,etc.)]

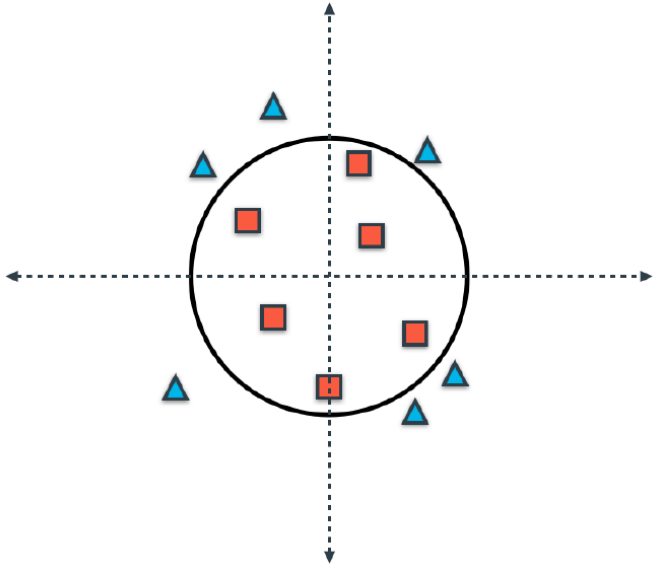
x_1	x_2	y
0.3	0.3	0
0.2	0.8	0
-0.6	0.4	0
0.6	-0.4	0
-0.4	-0.3	0
0	-0.8	0
-0.4	1.2	1
0.9	-0.7	1
-1.1	-0.8	1
0.7	0.9	1
-0.9	0.8	1
0.6	-1	1



We need a circle with this formula: $x_1^2 + x_2^2 = 1$

The Kernel Trick [polynomial equations (circles, parabolas,etc.)]

x_1	x_2	y
0.3	0.3	0
0.2	0.8	0
-0.6	0.4	0
0.6	-0.4	0
-0.4	-0.3	0
0	-0.8	0
-0.4	1.2	1
0.9	-0.7	1
-1.1	-0.8	1
0.7	0.9	1
-0.9	0.8	1
0.6	-1	1



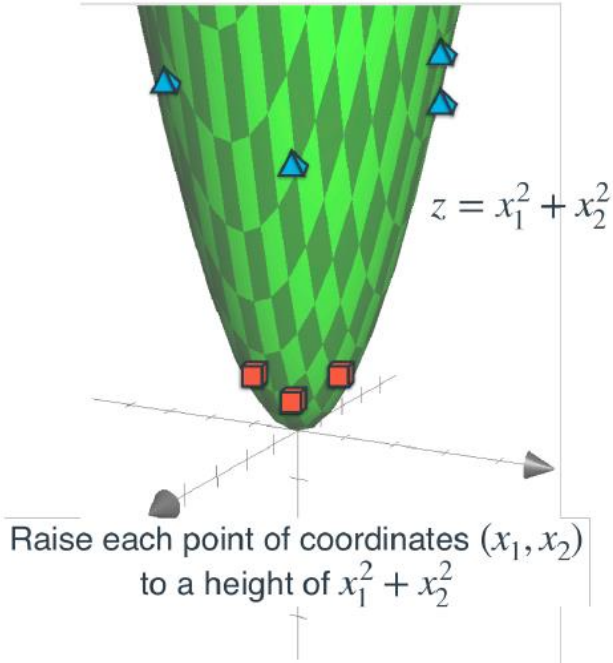
We need a circle with this formula: $x_1^2 + x_2^2 = 1$

The Kernel Trick [polynomial equations (circles, parabolas,etc.)]

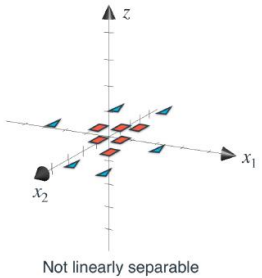
x_1	x_2	$x_1^2 + x_2^2$	y
0.3	0.3	0.18	0
0.2	0.8	0.68	0
-0.6	0.4	0.52	0
0.6	-0.4	0.52	0
-0.4	-0.3	0.25	0
0	-0.8	0.64	0
-0.4	1.2	1.6	1
0.9	-0.7	1.3	1
-1.1	-0.8	1.85	1
0.7	0.9	1.3	1
-0.9	0.8	1.45	1
0.6	-1	1.36	1

The Kernel Trick [polynomial equations (circles, parabolas,etc.)]

x_1	x_2	$x_1^2 + x_2^2$	y
0.3	0.3	0.18	0
0.2	0.8	0.68	0
-0.6	0.4	0.52	0
0.6	-0.4	0.52	0
-0.4	-0.3	0.25	0
0	-0.8	0.64	0
-0.4	1.2	1.6	1
0.9	-0.7	1.3	1
-1.1	-0.8	1.85	1
0.7	0.9	1.3	1
-0.9	0.8	1.45	1
0.6	-1	1.36	1

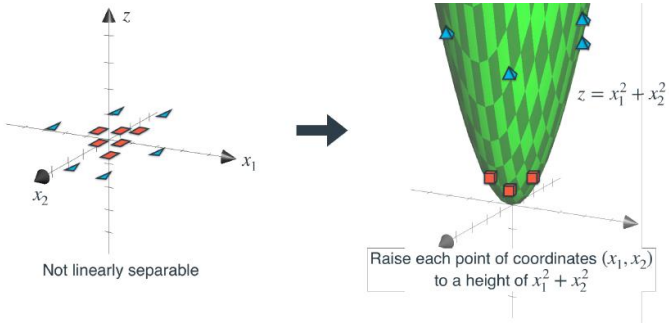


The Kernel Trick [polynomial equations (circles, parabolas,etc.)]

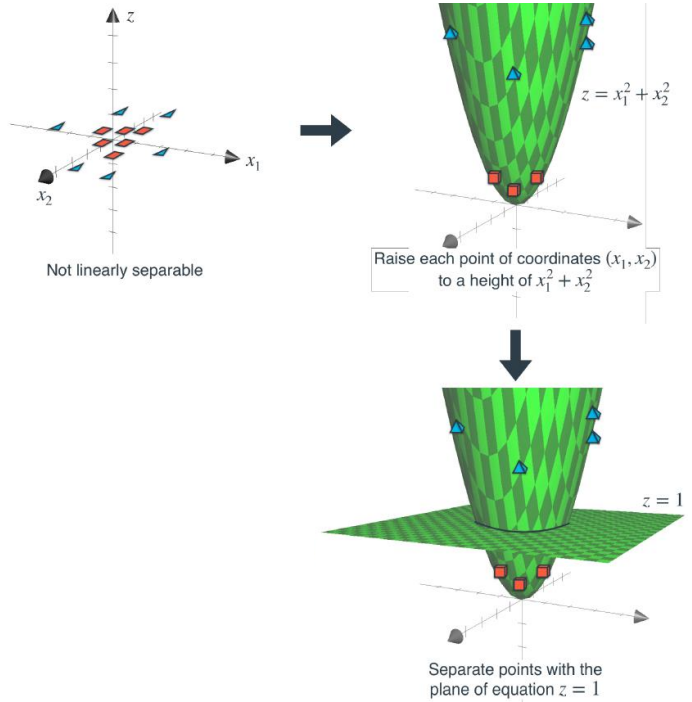


Not linearly separable

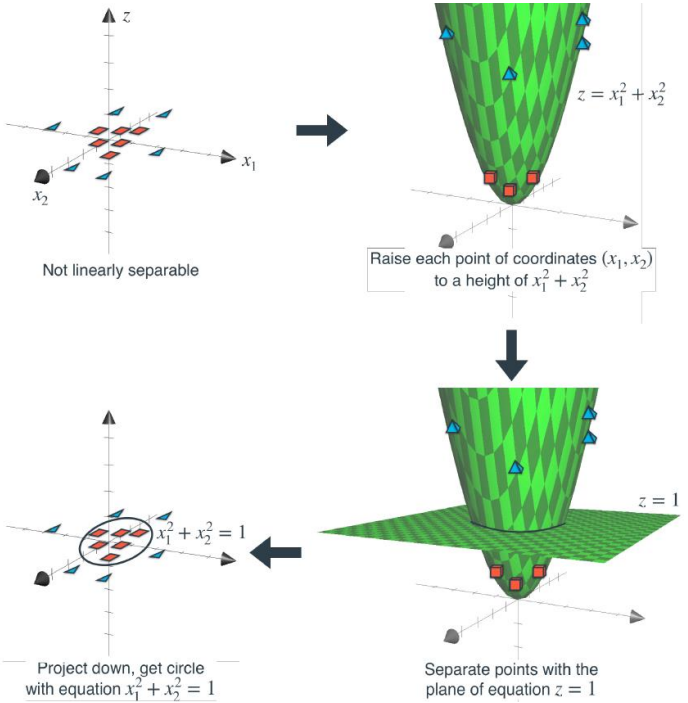
The Kernel Trick [polynomial equations (circles, parabolas,etc.)]



The Kernel Trick [polynomial equations (circles, parabolas,etc.)]



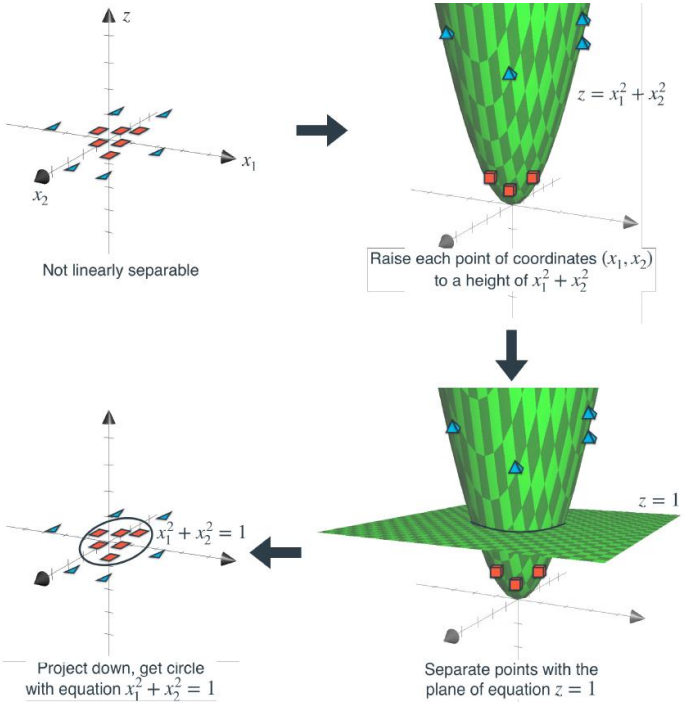
The Kernel Trick [polynomial equations (circles, parabolas,etc.)]



The Kernel Trick [polynomial equations (circles, parabolas,etc.)]

How did we find this expression in real example?

- we may not have the luxury to look at a plot and eyeball an expression that will help us out.
- We consider all the possible monomials of degree 2 (quadratic).
- These are the following three monomials:
  x_1^2 , x_2^2, $x_1 x_2$



Lecture Overview

- Problem Definition
- SVM Error Function
- Classification Error
- Distance Error
- Linear SVM coding
- Polynomial Kernel
- Polynomial Kernel Coding
- RBF Kernel
- RBF Kernel Coding
- Hinge Loss
- Support Vector Regression (SVR)

Poly SVM Coding

Use colab to open this github notebook:

“s7s/machine_learning_1/support_vector_machines/SVM_coding.ipynb”

SKlearn [Poly SVM]

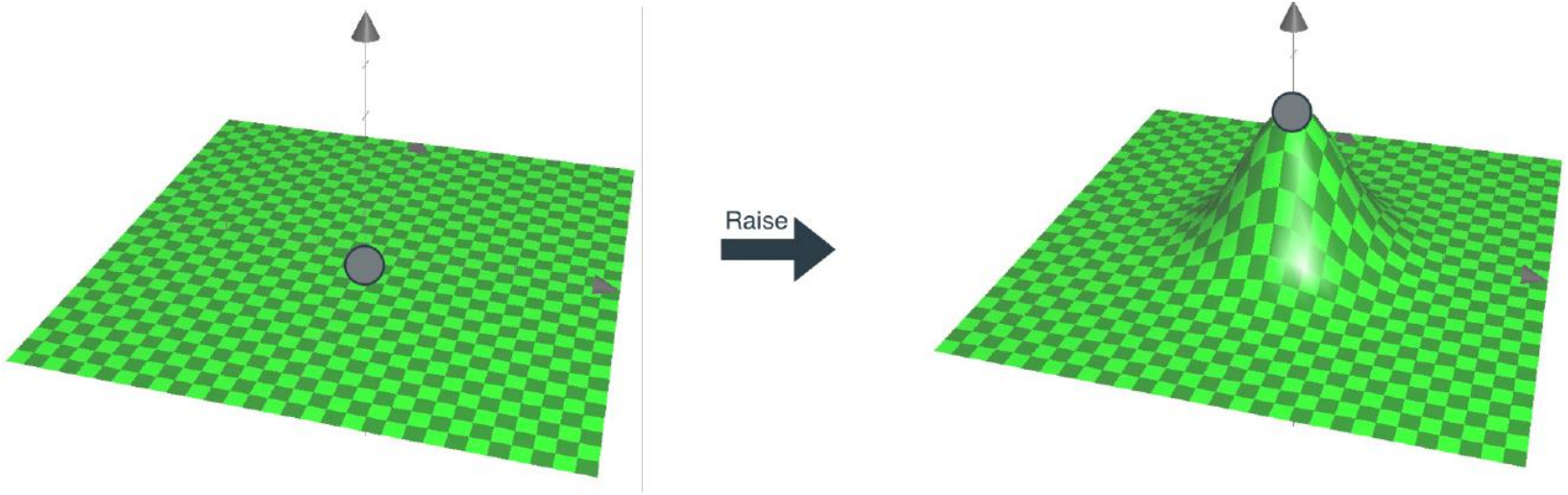
```
svm_degree_2 = SVC(kernel='poly', degree=2)
svm_degree_2.fit(features, labels)
print("Polynomial kernel of degree = 2")
print("Accuracy:", svm_degree_2.score(features, labels))
```

Lecture Overview

- Problem Definition
- SVM Error Function
- Classification Error
- Distance Error
- Linear SVM coding
- Polynomial Kernel
- Polynomial Kernel Coding
- RBF Kernel
- RBF Kernel Coding
- Hinge Loss
- Support Vector Regression (SVR)

The Kernel Trick [Radial Basis Function (rbf)]

Imagine if you had a point in the plane, and the plane was like a blanket. Then you pinch the blanket at that point, and raise it. This is how a radial basis function looks like.
We pinch each class points in a direction.

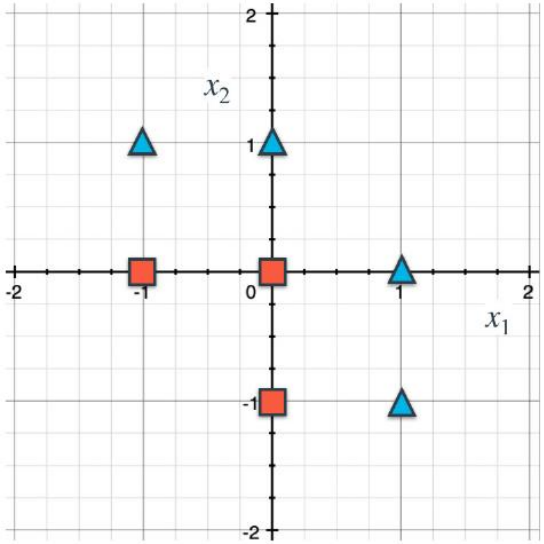


The Kernel Trick [Radial Basis Function (rbf)]

We applied the rbf function to generate new features, so we can classify data using a linear hyperplane.

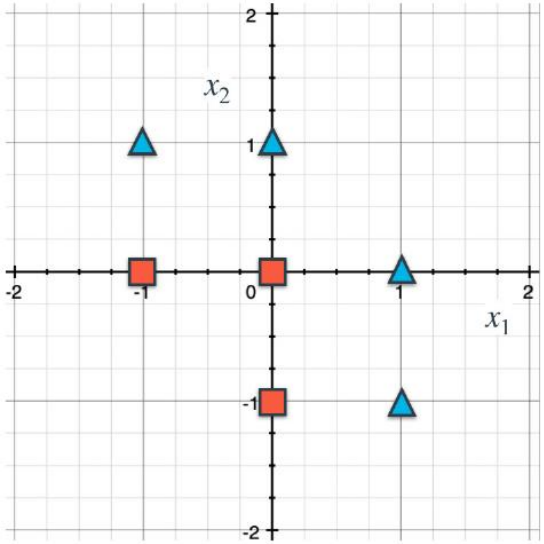
The Kernel Trick [Radial Basis Function (rbf)]

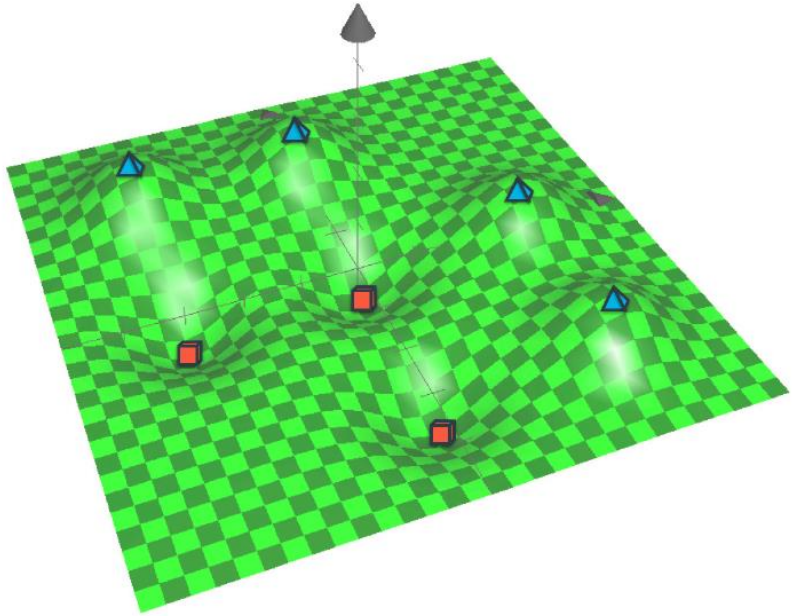
We applied the rbf function to generate new features, so we can classify data using a linear hyperplane.



The Kernel Trick [Radial Basis Function (rbf)]

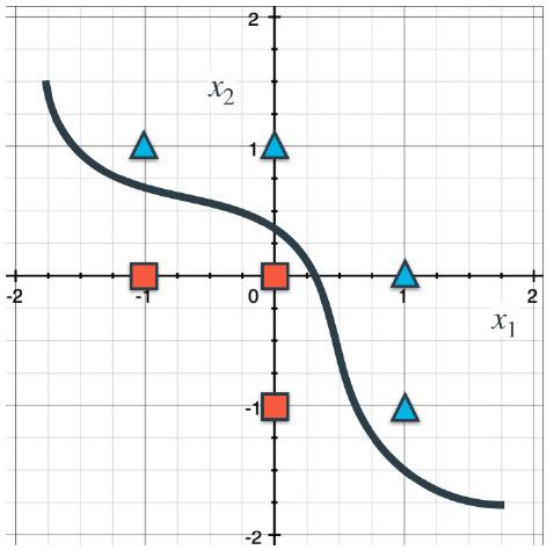
We applied the rbf function to generate new features, so we can classify data using a linear hyperplane.

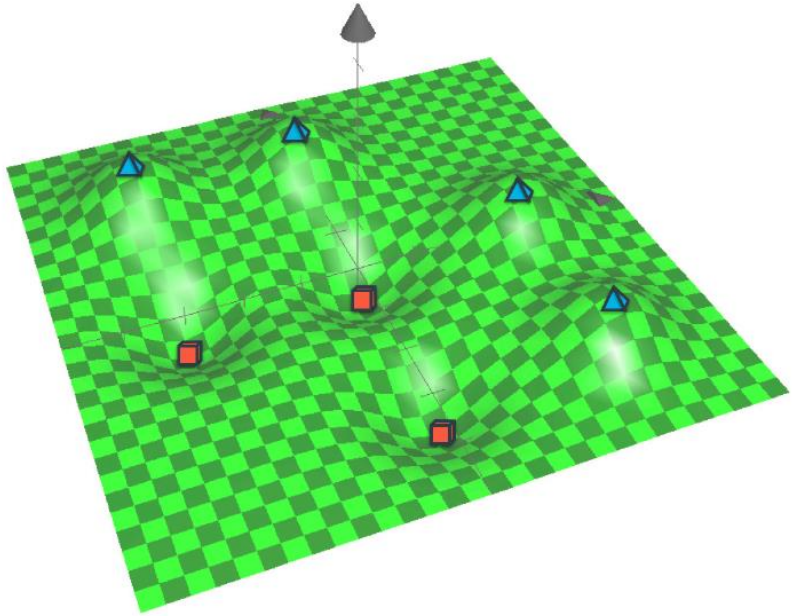




The Kernel Trick [Radial Basis Function (rbf)]

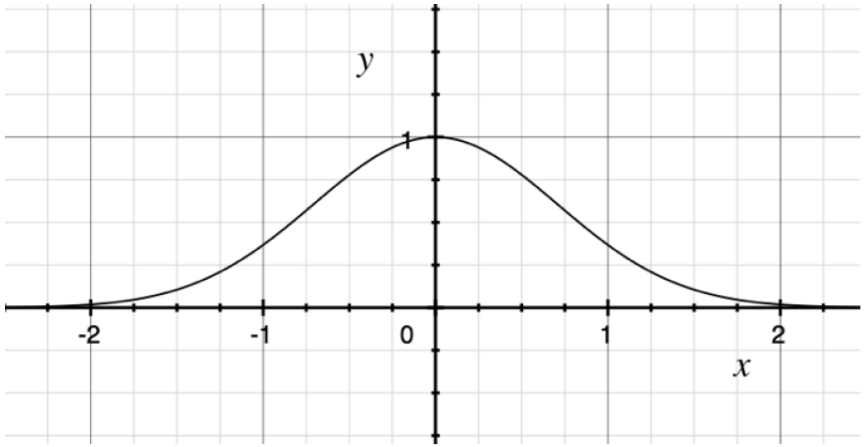
We applied the rbf function to generate new features, so we can classify data using a linear hyperplane.





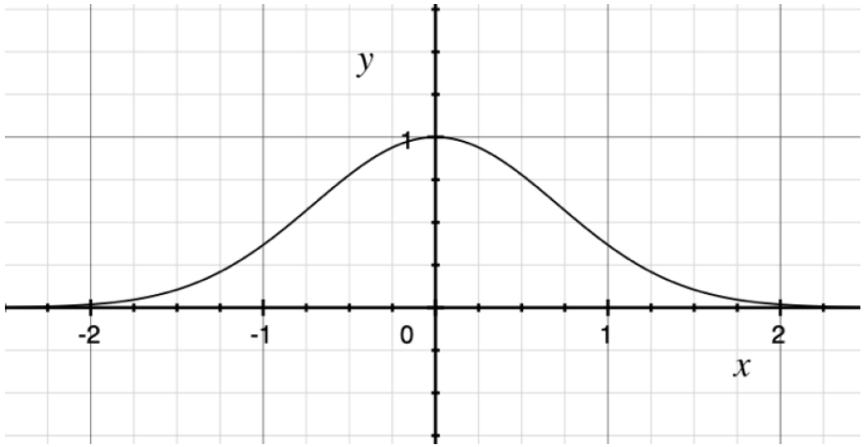
The Kernel Trick [Radial Basis Function (rbf)]

The Kernel Trick [Radial Basis Function (rbf)]



The Kernel Trick [Radial Basis Function (rbf)]

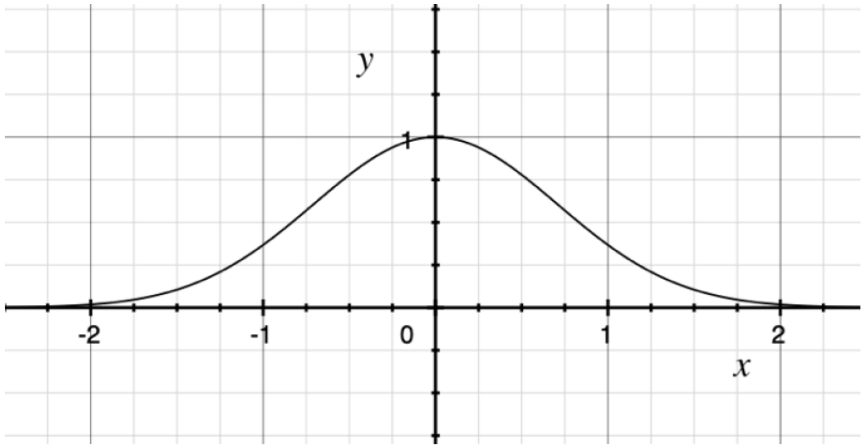
The radial basis function has the formula $y = e^{-x^2}$



The Kernel Trick [Radial Basis Function (rbf)]

The radial basis function has the formula $y = e^{-x^2}$

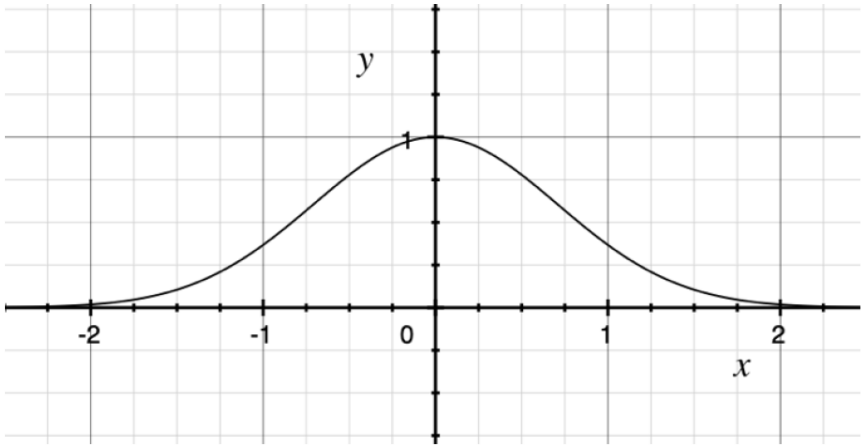
If bump not at zero point the formula $y = e^{-(x-p)^2}$

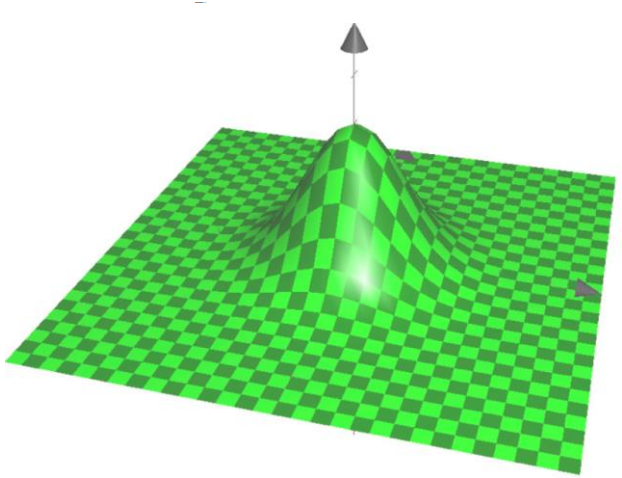


The Kernel Trick [Radial Basis Function (rbf)]

The radial basis function has the formula $y = e^{-x^2}$

If bump not at zero point the formula $y = e^{-(x-p)^2}$

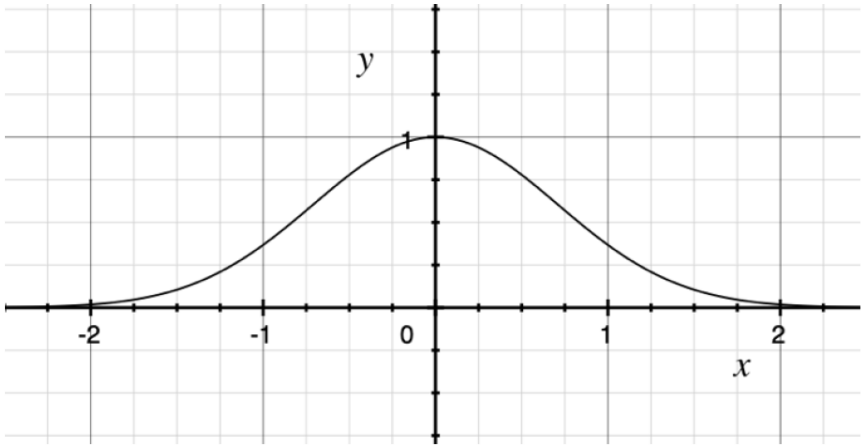


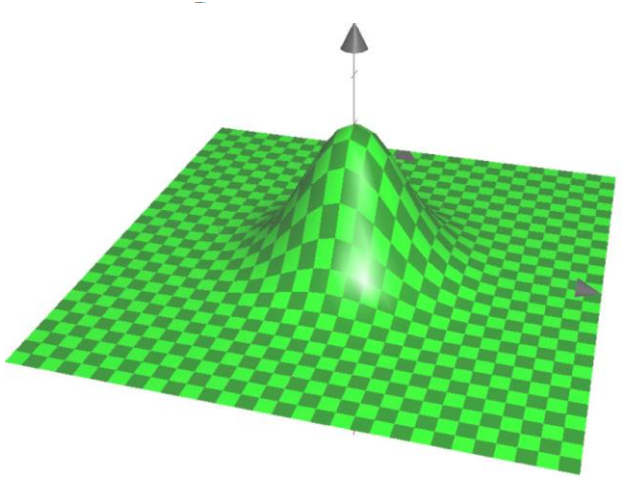


The Kernel Trick [Radial Basis Function (rbf)]

The radial basis function has the formula $y = e^{-x^2}$, and for two variables $z = e^{-(x^2+y^2)}$

If bump not at zero point the formula $y = e^{-(x-p)^2}$

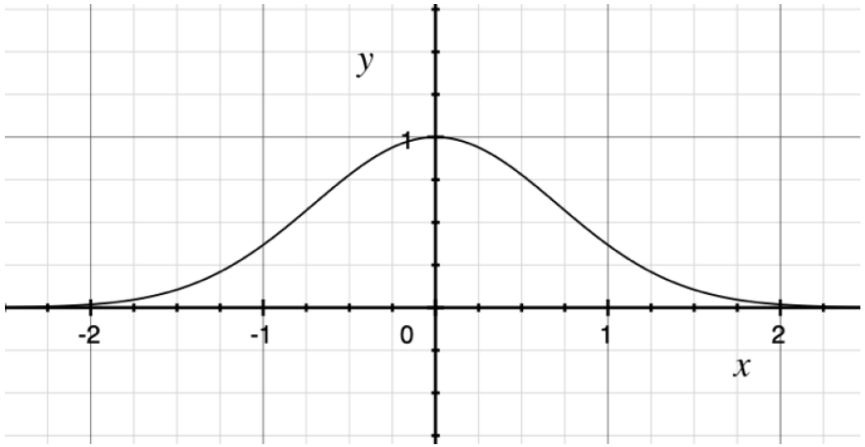


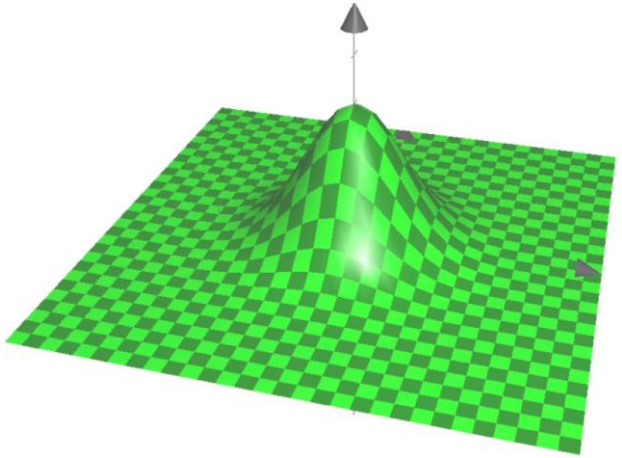


The Kernel Trick [Radial Basis Function (rbf)]

The radial basis function has the formula $y = e^{-x^2}$, and for two variables $z = e^{-(x^2+y^2)}$

If bump not at zero point the formula $y = e^{-(x-p)^2}$, and for two variables $y = e^{-[(x-p)^2+(y-q)^2]}$

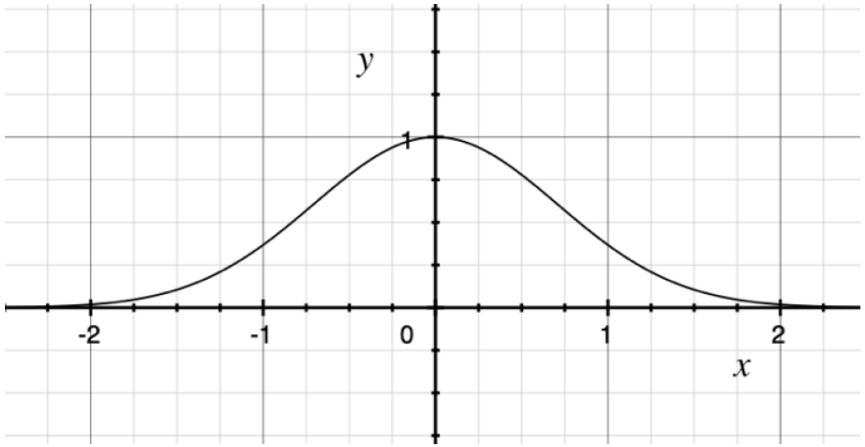


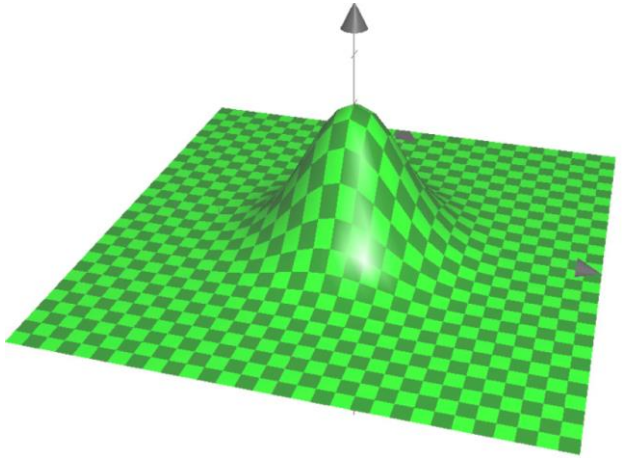


The Kernel Trick [Radial Basis Function (rbf)]

The radial basis function has the formula $y = e^{-x^2}$, and for two variables $z = e^{-(x^2+y^2)}$

If bump not at zero point the formula $y = e^{-(x-p)^2}$, and for two variables $y = e^{-[(x-p)^2+(y-q)^2]}$





How this function is used to classify points?

The Kernel Trick [Radial Basis Function (rbf)]

Radial basis function is used as a similarity function.

The Kernel Trick [Radial Basis Function (rbf)]

Radial basis function is used as a similarity function.

$$y = e^{-(x-p)^2}$$

The Kernel Trick [Radial Basis Function (rbf)]

Radial basis function is used as a similarity function.

Distance

$$y = e^{-(x-p)^2}$$

The Kernel Trick [Radial Basis Function (rbf)]

Radial basis function is used as a similarity function.

Distance

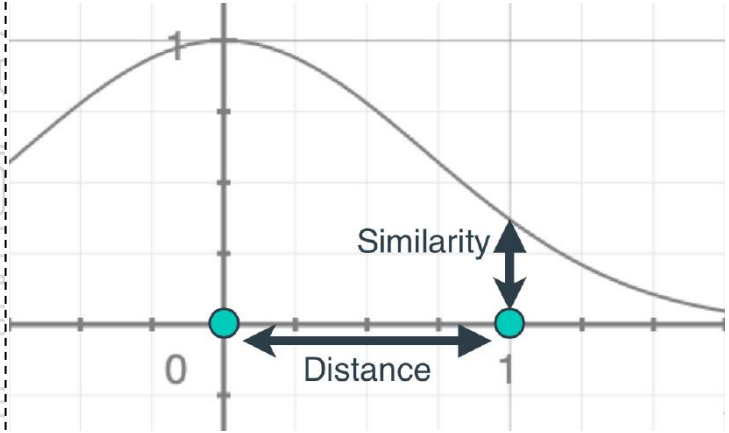
Similarity $y = e^{-(x-p)^2}$

The Kernel Trick [Radial Basis Function (rbf)]

Radial basis function is used as a similarity function.

Distance

Similarity $y = e^{-(x-p)^2}$

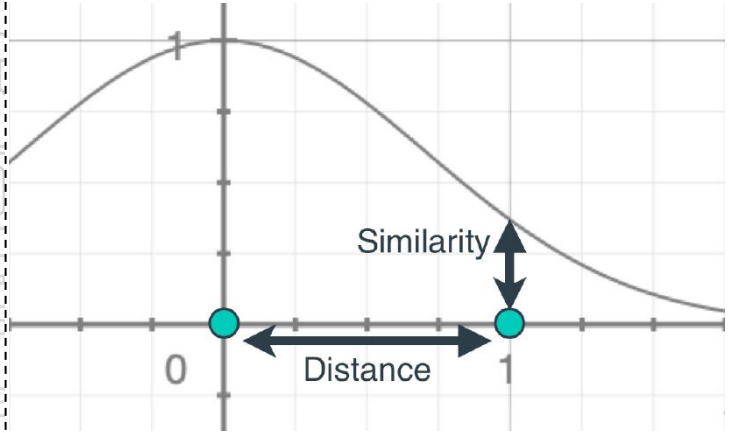


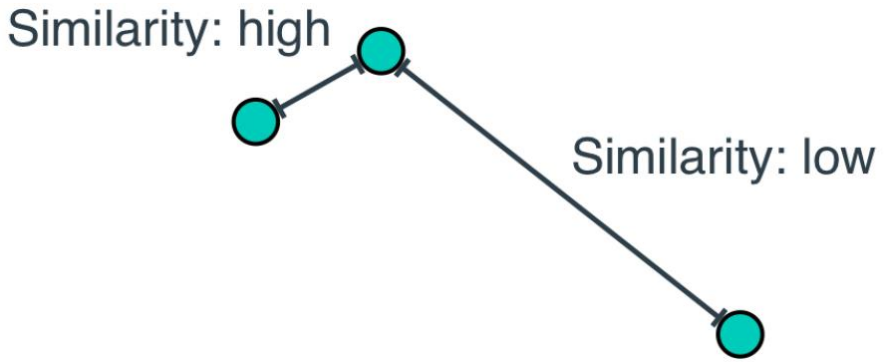
The Kernel Trick [Radial Basis Function (rbf)]

Radial basis function is used as a similarity function.

Distance

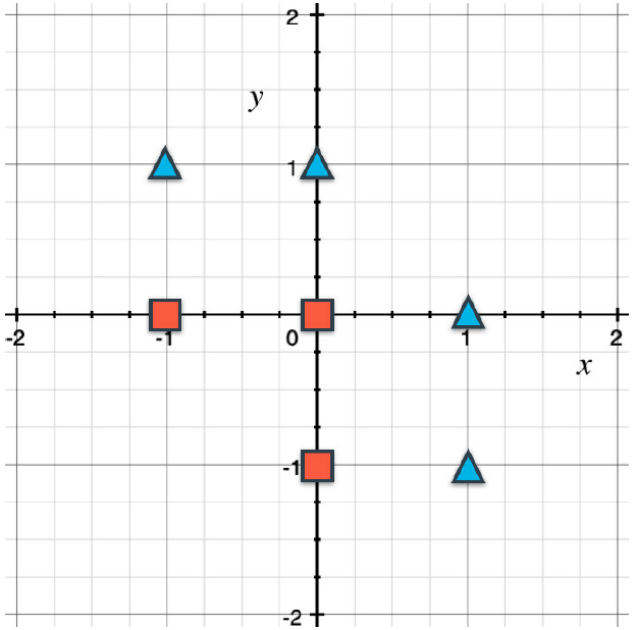
Similarity $y = e^{-(x-p)^2}$





The Kernel Trick [Radial Basis Function (rbf)][Training]

Point	x_1	x_2	y
1	0	0	0
2	-1	0	0
3	0	-1	0
4	0	1	1
5	1	0	1
6	-1	1	1
7	1	-1	1



The Kernel Trick [Radial Basis Function (rbf)][Training]

Point	x_1	x_2	y
1	0	0	0
2	-1	0	0
3	0	-1	0
4	0	1	1
5	1	0	1
6	-1	1	1
7	1	-1	1

The Kernel Trick [Radial Basis Function (rbf)][Training]

Point	x_1	x_2	Sim 1
1	0	0	1
2	-1	0	0.135
3	0	-1	0.135
4	0	1	0.135
5	1	0	0.135
6	-1	1	0.368
7	1	-1	0.368

y
0
0
0
1
1
1
1

The Kernel Trick [Radial Basis Function (rbf)][Training]

Point	x_1	x_2	Sim 1	Sim 2	Sim 3	Sim 4	Sim 5	Sim 6	Sim 7	y
1	0	0	1	0.135	0.135	0.135	0.135	0.368	0.368	0
2	-1	0	0.135	1	0.368	0.368	0.018	0.135	0.018	0
3	0	-1	0.135	0.368	1	0.018	0.135	0.007	0.368	0
4	0	1	0.135	0.368	0.018	1	0.135	0.368	0.007	1
5	1	0	0.135	0.018	0.135	0.135	1	0.007	0.368	1
6	-1	1	0.368	0.135	0.007	0.368	0.007	1	0	1
7	1	-1	0.368	0.018	0.368	0.007	0.368	0	1	1

The Kernel Trick [Radial Basis Function (rbf)][Training]

Point	x_1	x_2	Sim 1	Sim 2	Sim 3	Sim 4	Sim 5	Sim 6	Sim 7	y
1	0	0	1	0.135	0.135	0.135	0.135	0.368	0.368	0
2	-1	0	0.135	1	0.368	0.368	0.018	0.135	0.018	0
3	0	-1	0.135	0.368	1	0.018	0.135	0.007	0.368	0
4	0	1	0.135	0.368	0.018	1	0.135	0.368	0.007	1
5	1	0	0.135	0.018	0.135	0.135	1	0.007	0.368	1
6	-1	1	0.368	0.135	0.007	0.368	0.007	1	0	1
7	1	-1	0.368	0.018	0.368	0.007	0.368	0	1	1

Many linear models can classify this data, one of them is:

$W_1 = 0$
$W_2 = 0$
$V_1 = -1$
$V_2 = -1$
$V_3 = -1$
$V_4 = 1$
$V_5 = 1$
$V_6 = 1$
$V_7 = 1$
$b = 0$

Why this classifier works?

The Kernel Trick [Radial Basis Function (rbf)][Training]

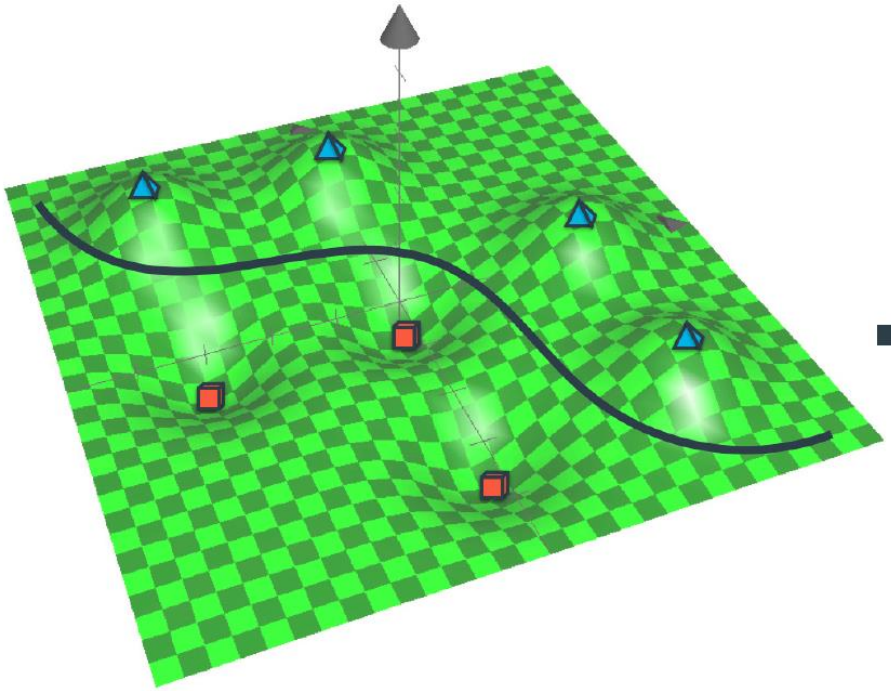
Point	x_1	x_2	Sim 1	Sim 2	Sim 3	Sim 4	Sim 5	Sim 6	Sim 7	y
1	0	0	1	0.135	0.135	0.135	0.135	0.368	0.368	0
2	-1	0	0.135	1	0.368	0.368	0.018	0.135	0.018	0
3	0	-1	0.135	0.368	1	0.018	0.135	0.007	0.368	0
4	0	1	0.135	0.368	0.018	1	0.135	0.368	0.007	1
5	1	0	0.135	0.018	0.135	0.135	1	0.007	0.368	1
6	-1	1	0.368	0.135	0.007	0.368	0.007	1	0	1
7	1	-1	0.368	0.018	0.368	0.007	0.368	0	1	1

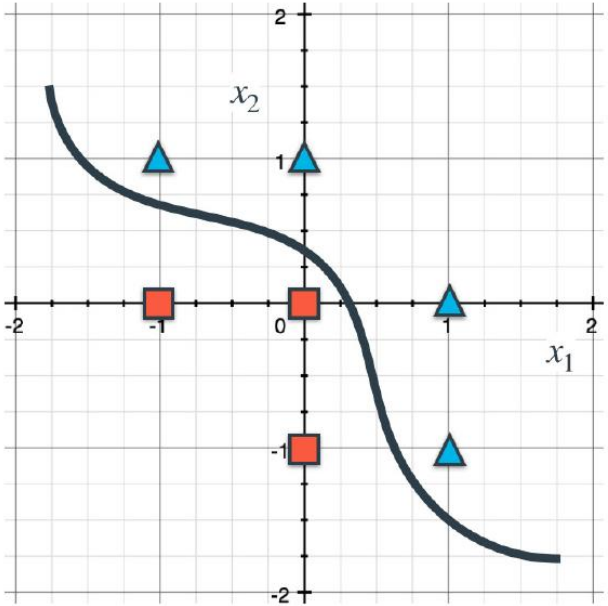
Many linear models can classify this data, one of them is:

$W_1 = 0$
$W_2 = 0$
$V_1 = -1$
$V_2 = -1$
$V_3 = -1$
$V_4 = 1$
$V_5 = 1$
$V_6 = 1$
$V_7 = 1$
$b = 0$

Why this classifier works?

The Kernel Trick [Radial Basis Function (rbf)][**Training**]





The Kernel Trick [Radial Basis Function][**Gamma parameter**]

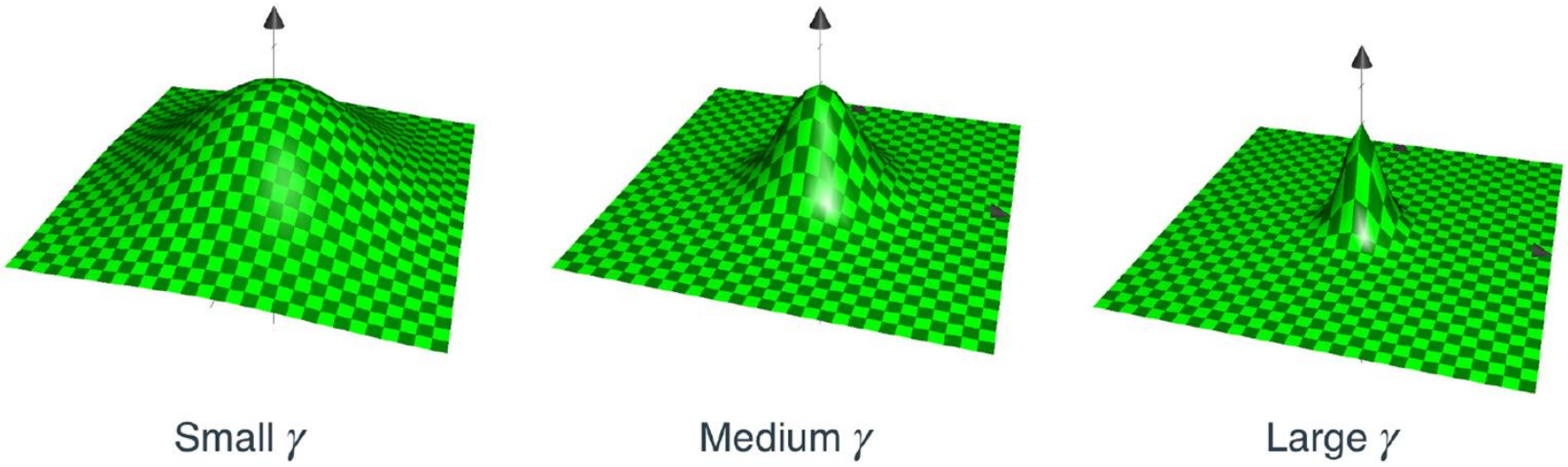
Gamma is a hyperparameter that tunes the wideness of the rbf kernel

$$y = e^{-\gamma[(x_1-p_1)^2+\cdots+(x_n-p_n)^2]}$$

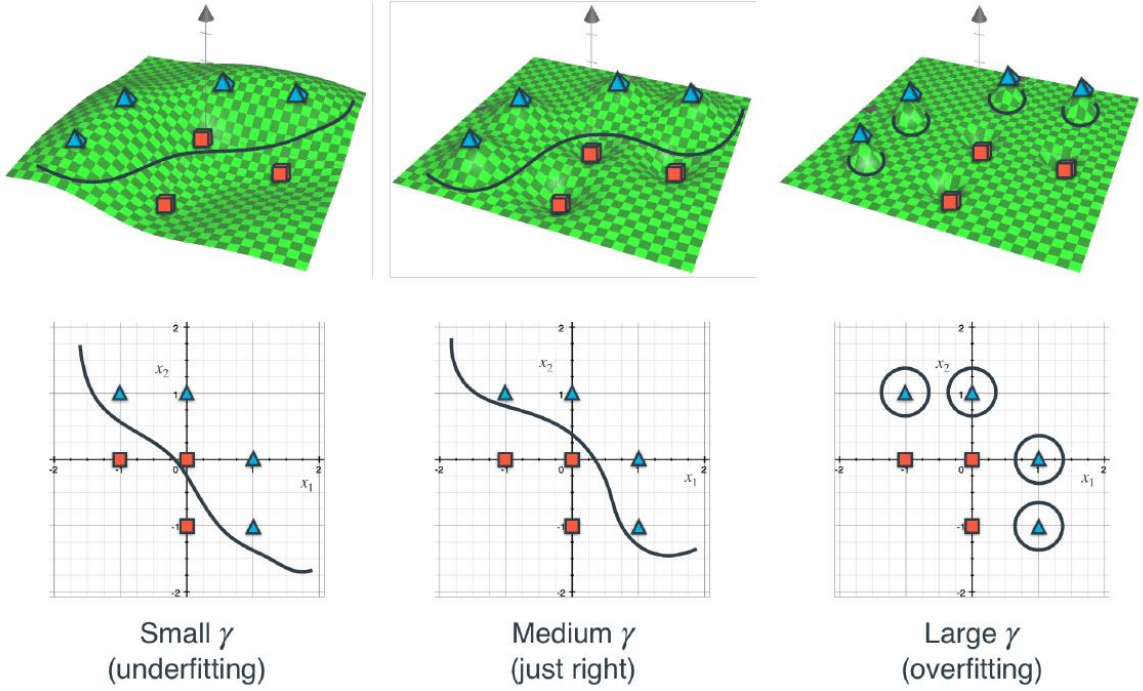
The Kernel Trick [Radial Basis Function][**Gamma parameter**]

Gamma is a hyperparameter that tunes the wideness of the rbf kernel

$$y = e^{-\gamma[(x_1-p_1)^2+\cdots+(x_n-p_n)^2]}$$



The Kernel Trick [Radial Basis Function][**Gamma parameter**]



Lecture Overview

- Problem Definition
- SVM Error Function
- Classification Error
- Distance Error
- Linear SVM coding
- Polynomial Kernel
- Polynomial Kernel Coding
- RBF Kernel
- RBF Kernel Coding
- Hinge Loss
- Support Vector Regression (SVR)

RBF SVM Coding

Use colab to open this github notebook:

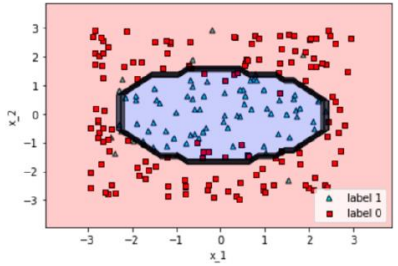
“s7s/machine_learning_1/support_vector_machines/SVM_coding.ipynb”

SKlearn [RBF SVM]

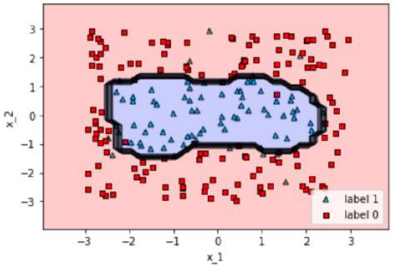
```
svm_gamma_01 = SVC(kernel='rbf', gamma=0.1)
svm_gamma_01.fit(features, labels)
print("Gamma = 0.1")
print("Accuracy:", svm_gamma_01.score(features, labels))
```

The Kernel Trick [Radial Basis Function][**Gamma parameter**]

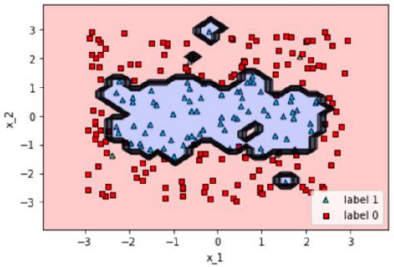
Gamma = 0.1
Accuracy: 0.8772727272727273



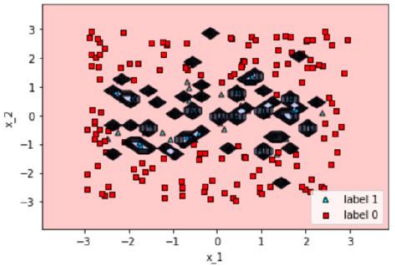
Gamma = 1
Accuracy: 0.9045454545454545



Gamma = 10
Accuracy: 0.9636363636363636



Gamma = 100
Accuracy: 0.990909090909091



Lecture Overview

- Problem Definition
- SVM Error Function
- Classification Error
- Distance Error
- Linear SVM coding
- Polynomial Kernel
- Polynomial Kernel Coding
- RBF Kernel
- RBF Kernel Coding
- Hinge Loss
- Support Vector Regression (SVR)

The Log Loss (Cross Entropy Loss)

The Log Loss (Cross Entropy Loss)

If y=1

Loss = -ln(σ(z))

If y=0

Loss = -ln(1 - σ(z))

The Log Loss (Cross Entropy Loss)

If y=1 $z = ax1 + bx2 + c$

Loss = -ln(σ(z))

If y=0

Loss = -ln(1 - σ(z))

The Log Loss (Cross Entropy Loss)

If y=1 $z = ax1 + bx2 + c$ **If y=0**

Loss = $-\ln(\sigma(z)) = -\ln(1/(1+e^{-z}))$ **Loss = $-\ln(1 - \sigma(z))$**

The Log Loss (Cross Entropy Loss)

If y=1 $z = ax1 + bx2 + c$

Loss = $-\ln(\sigma(z)) = -\ln(1/(1+e^{-z}))$

If y=0

Loss = $-\ln(1 - \sigma(z)) = -\ln(1-(1/(1+e^{-z})))$

The Log Loss (Cross Entropy Loss)

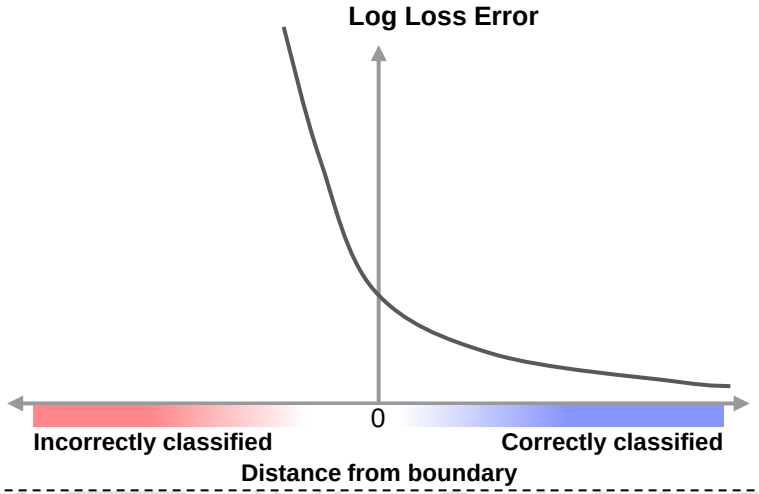
If y=1 $z = ax1 + bx2 + c$

Loss = $-\ln(\sigma(z)) = -\ln(1/(1+e^{-z}))$

The Log Loss (Cross Entropy Loss)

If y=1 $z = ax1 + bx2 + c$

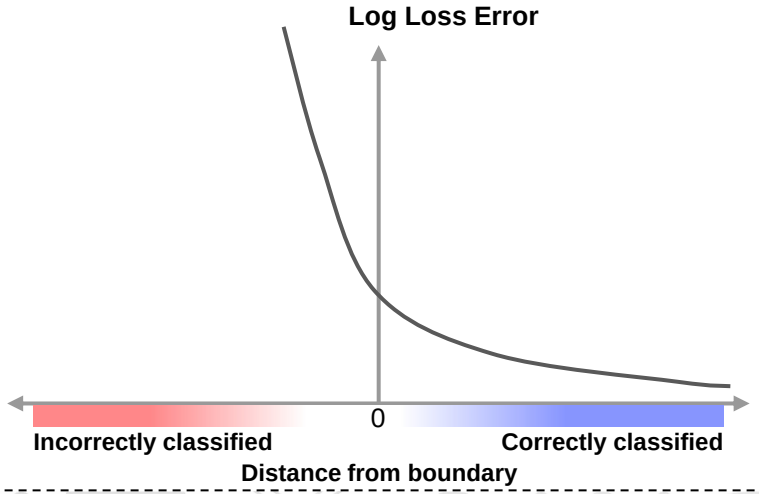
Loss = $-\ln(\sigma(z)) = -\ln(1/(1+e^{-z}))$

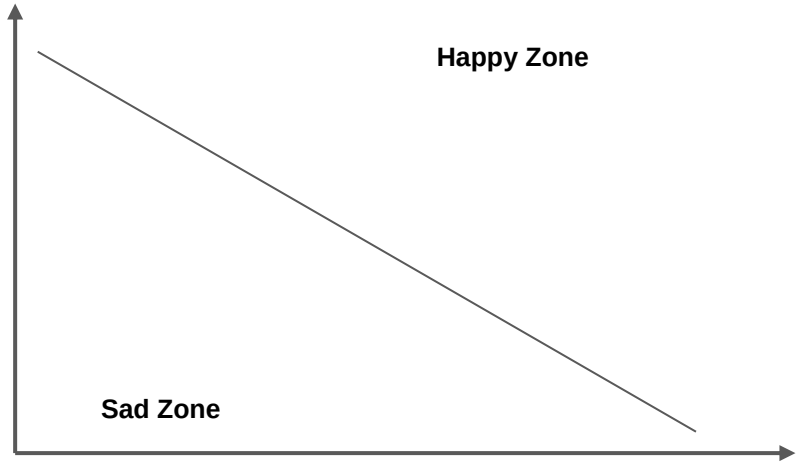


The Log Loss (Cross Entropy Loss)

If y=1 $z = ax1 + bx2 + c$

Loss = $-\ln(\sigma(z)) = -\ln(1/(1+e^{-z}))$

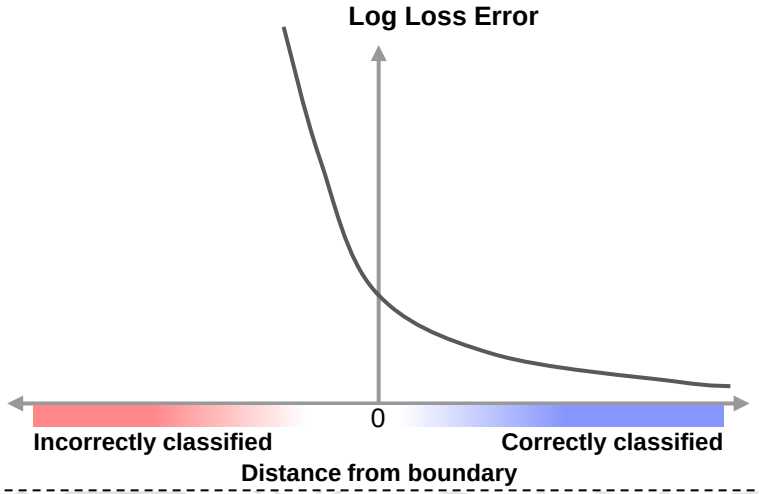


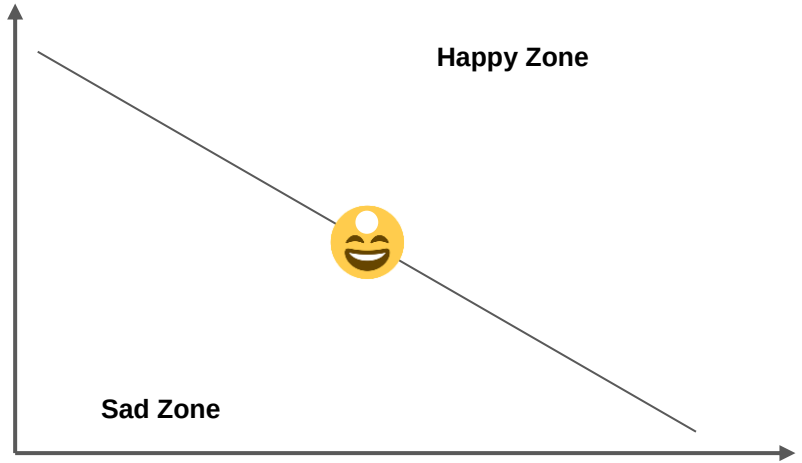


The Log Loss (Cross Entropy Loss)

If y=1 $z = ax1 + bx2 + c$

Loss = $-\ln(\sigma(z)) = -\ln(1/(1+e^{-z}))$

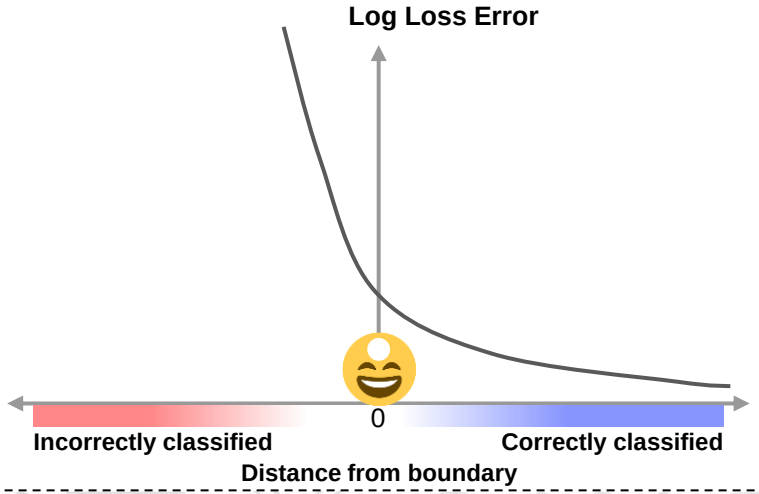


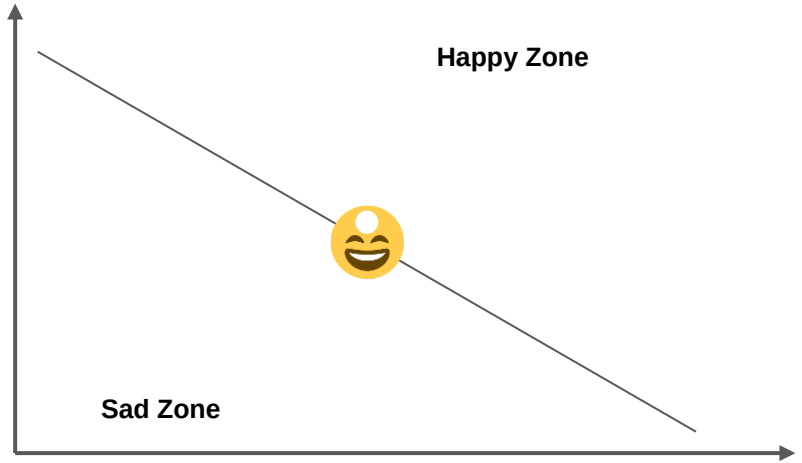


The Log Loss (Cross Entropy Loss)

If y=1 $z = ax1 + bx2 + c$

Loss = $-\ln(\sigma(z)) = -\ln(1/(1+e^{-z}))$

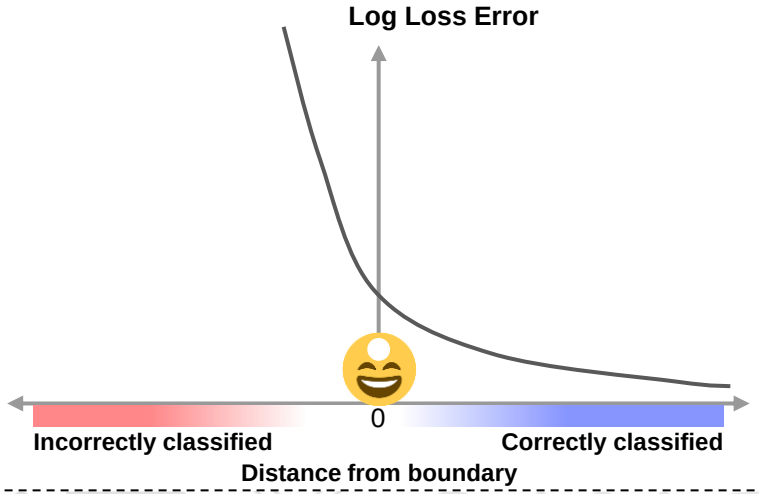


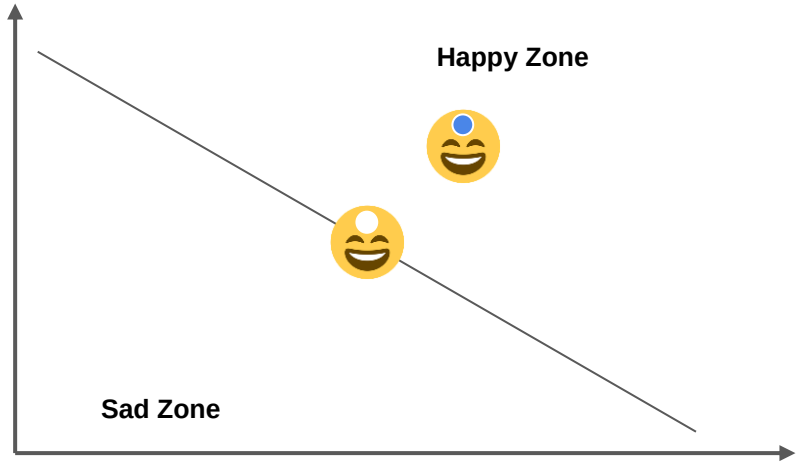


The Log Loss (Cross Entropy Loss)

If y=1 $z = ax1 + bx2 + c$

Loss = $-\ln(\sigma(z)) = -\ln(1/(1+e^{-z}))$

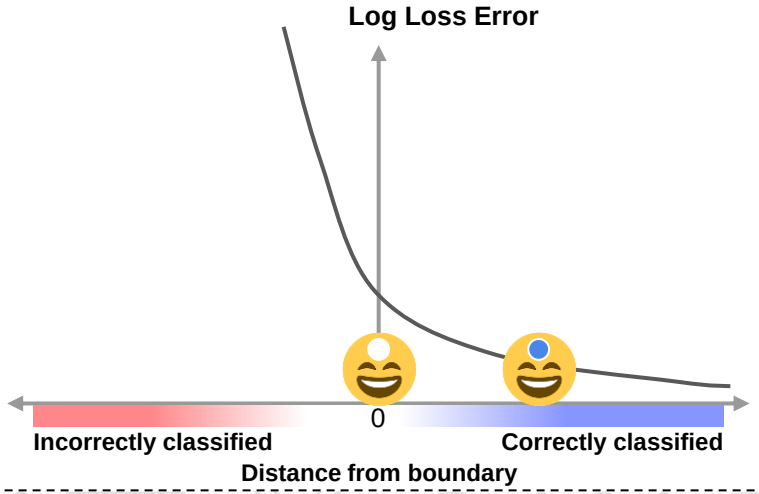


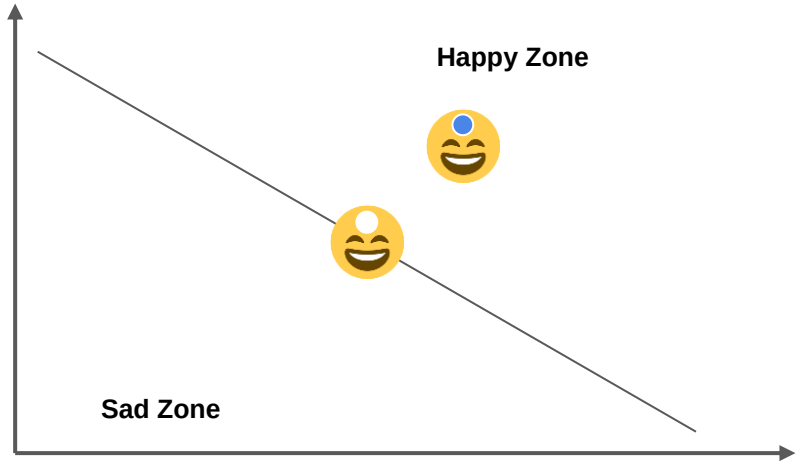


The Log Loss (Cross Entropy Loss)

If y=1 $z = ax1 + bx2 + c$

Loss = $-\ln(\sigma(z)) = -\ln(1/(1+e^{-z}))$

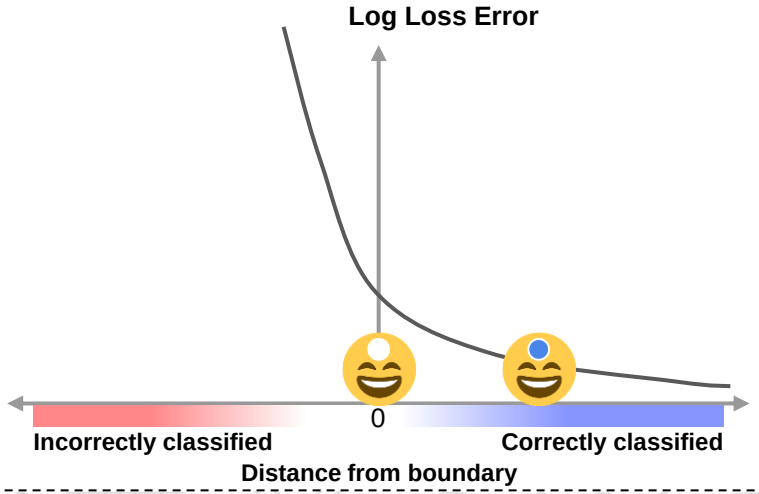


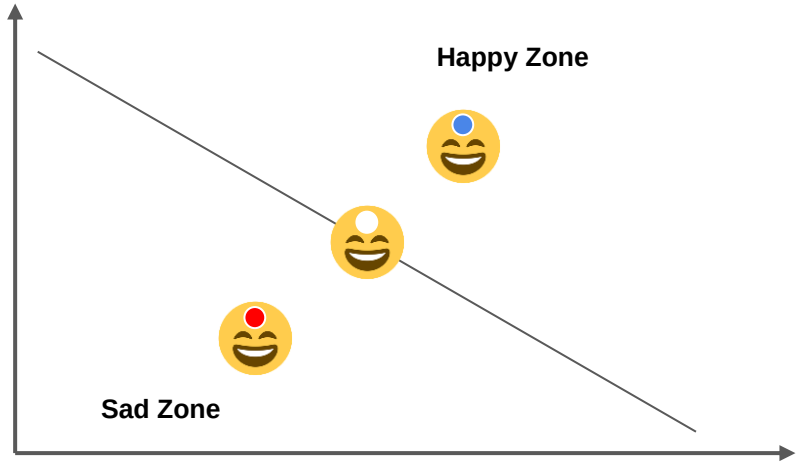


The Log Loss (Cross Entropy Loss)

If y=1 $z = ax1 + bx2 + c$

Loss = $-\ln(\sigma(z)) = -\ln(1/(1+e^{-z}))$

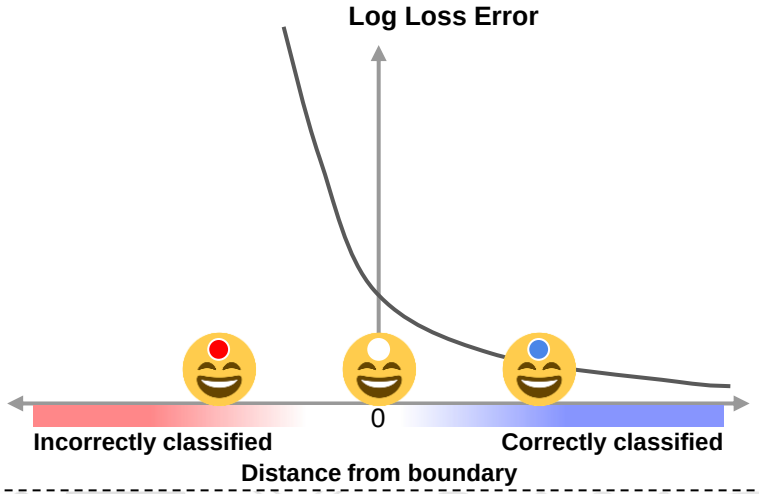


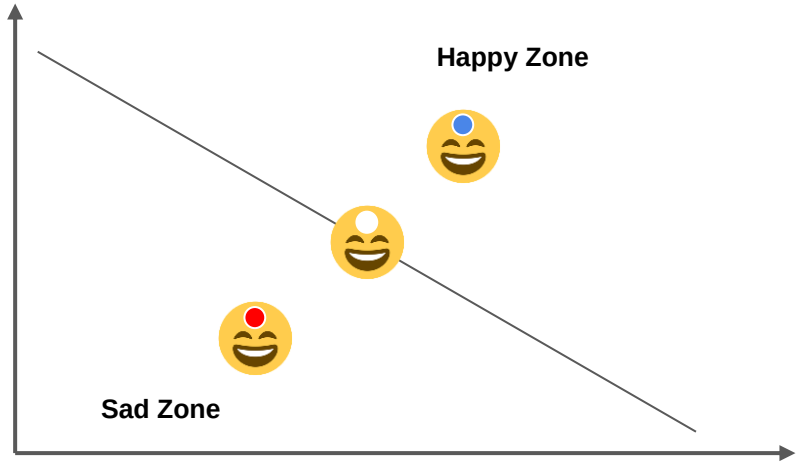


The Log Loss (Cross Entropy Loss)

If y=1 $z = ax1 + bx2 + c$

Loss = $-\ln(\sigma(z)) = -\ln(1/(1+e^{-z}))$





The Log Loss (Cross Entropy Loss)

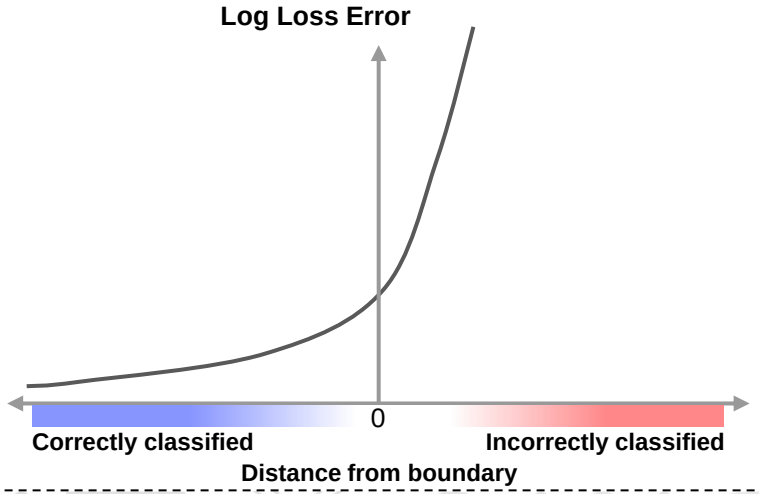
If y=0

$$\textbf{Loss} = -\ln(1 - \sigma(z)) = -\ln\left(1-\left(1/\left(1+e^{-z}\right)\right)\right)$$

The Log Loss (Cross Entropy Loss)

If y=0

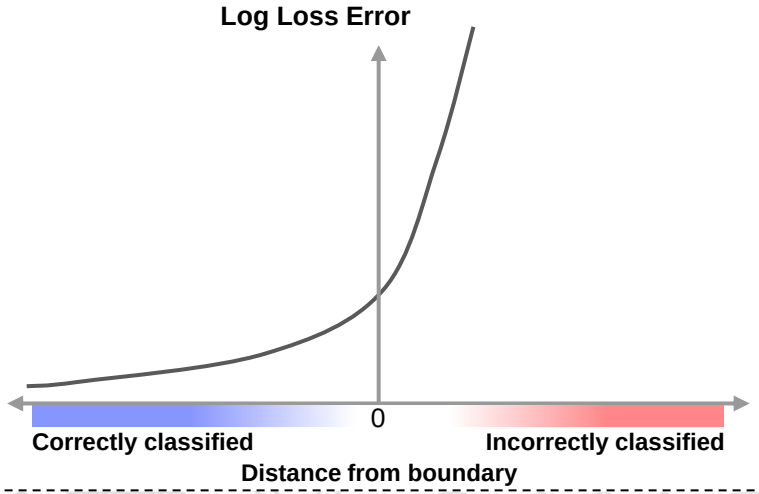
Loss = $-\ln(1 - \sigma(z)) = -\ln(1-(1/(1+e^{-z})))$

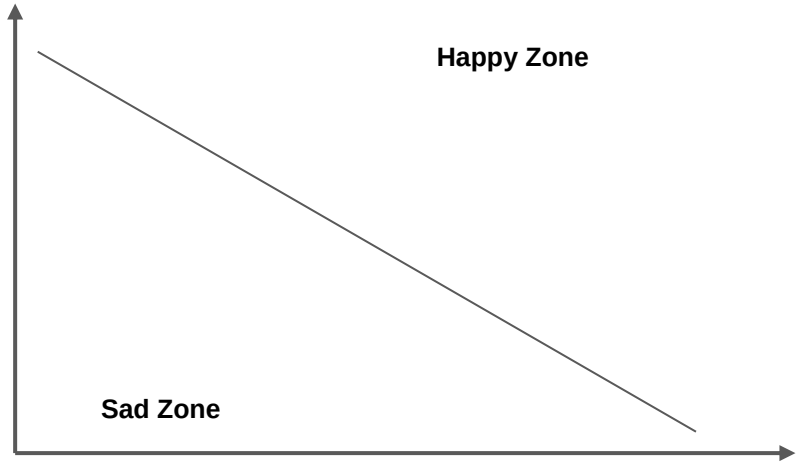


The Log Loss (Cross Entropy Loss)

If y=0

Loss = $-\ln(1 - \sigma(z)) = -\ln(1-(1/(1+e^{-z})))$

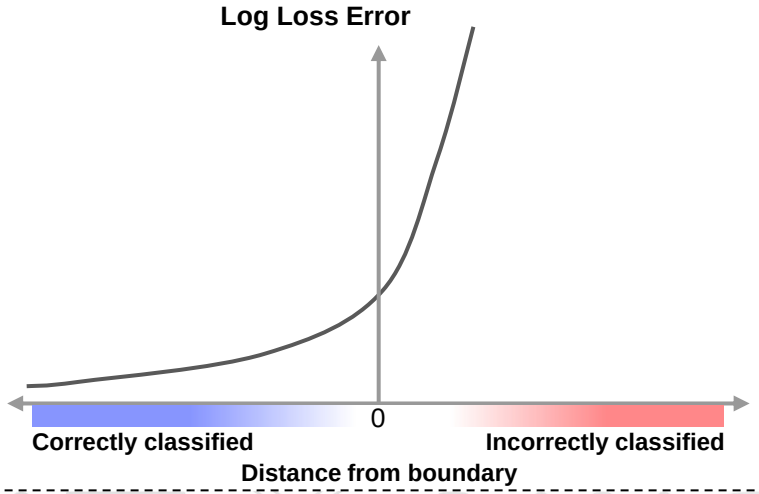


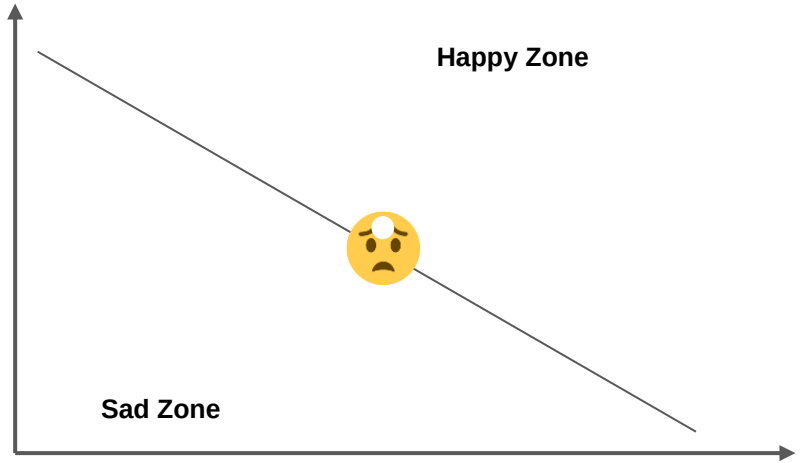


The Log Loss (Cross Entropy Loss)

If y=0

$$\text{Loss} = -\ln(1 - \sigma(z)) = -\ln(1-(1/(1+e^{-z})))$$

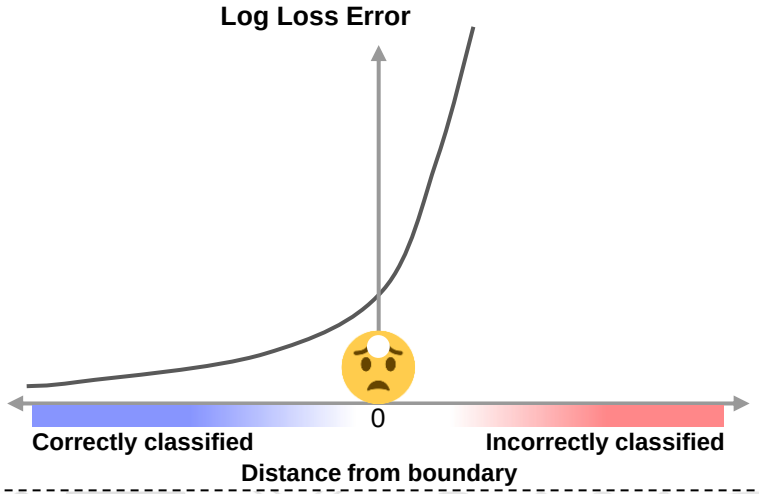


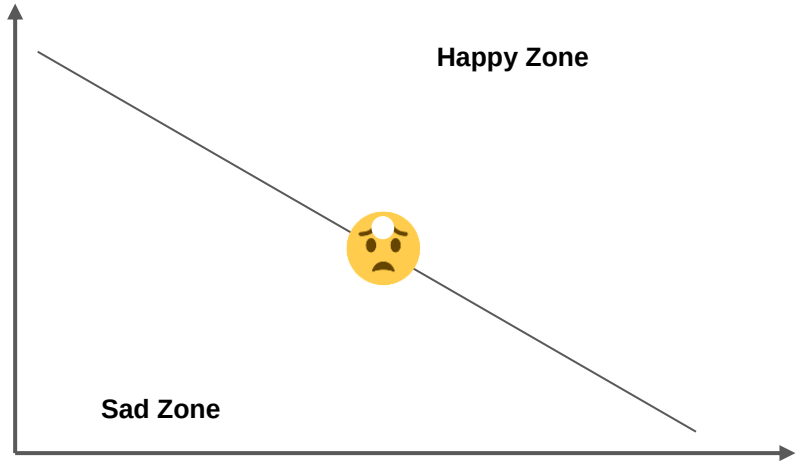


The Log Loss (Cross Entropy Loss)

If y=0

$$\text{Loss} = -\ln(1 - \sigma(z)) = -\ln(1-(1/(1+e^{-z})))$$

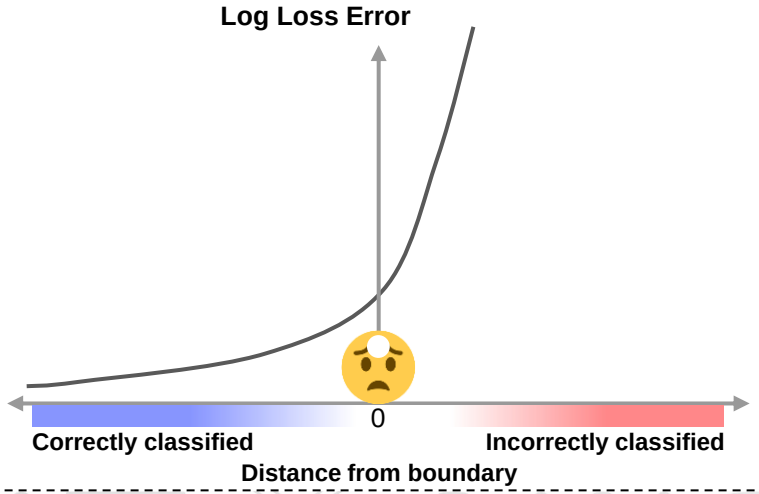


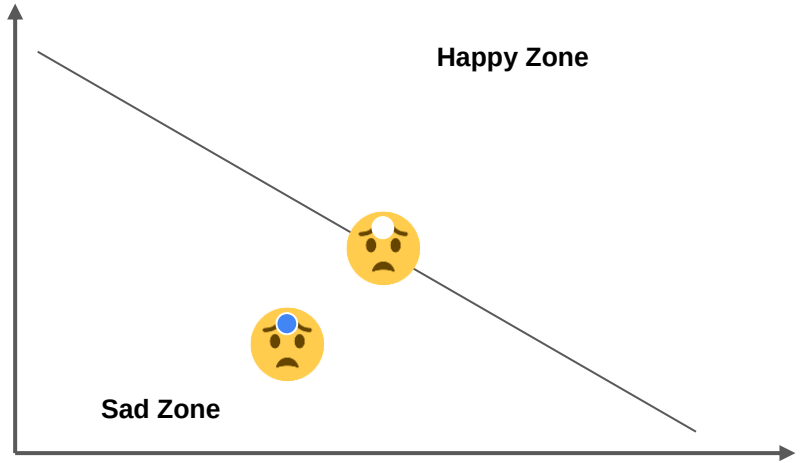


The Log Loss (Cross Entropy Loss)

If y=0

$$\text{Loss} = -\ln(1 - \sigma(z)) = -\ln(1-(1/(1+e^{-z})))$$

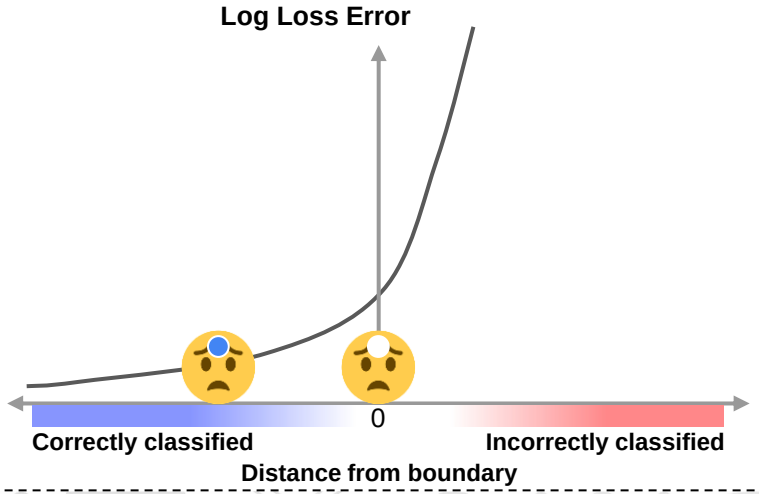


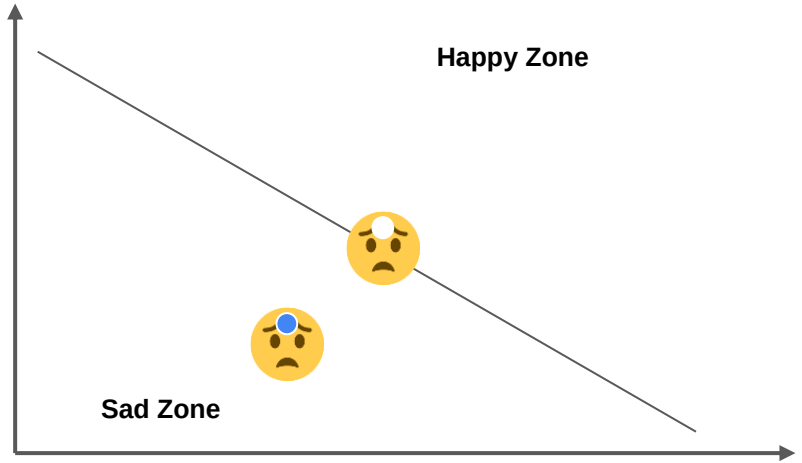


The Log Loss (Cross Entropy Loss)

If y=0

$$\text{Loss} = -\ln(1 - \sigma(z)) = -\ln(1-(1/(1+e^{-z})))$$

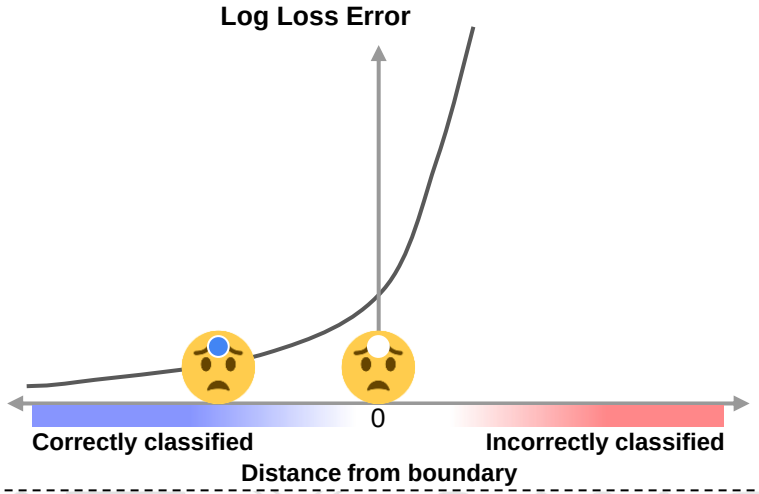


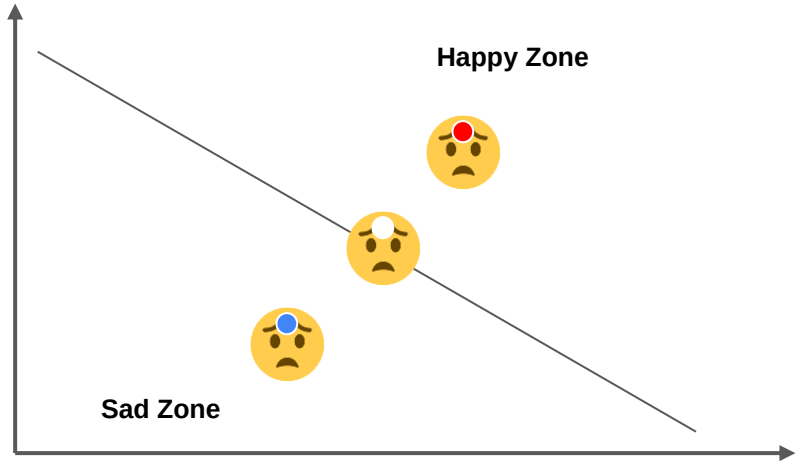


The Log Loss (Cross Entropy Loss)

If y=0

$$\text{Loss} = -\ln(1 - \sigma(z)) = -\ln(1-(1/(1+e^{-z})))$$

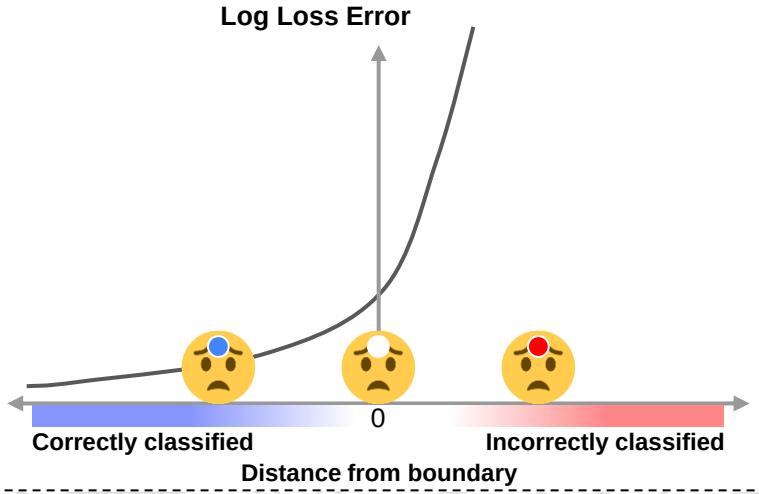


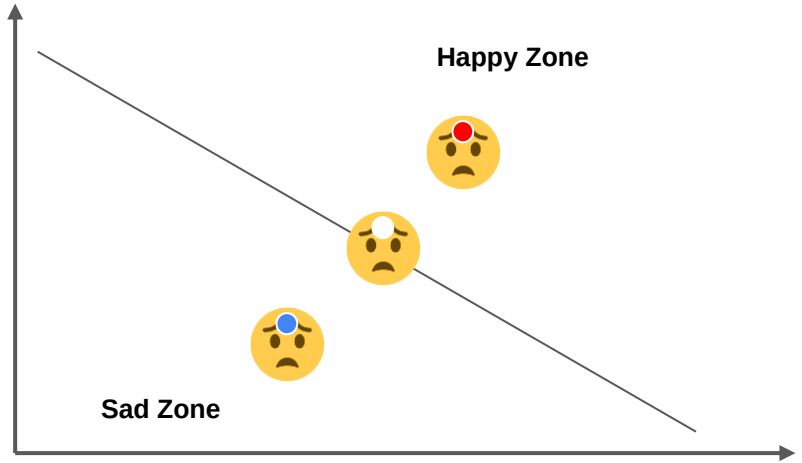


The Log Loss (Cross Entropy Loss)

If y=0

$$\text{Loss} = -\ln(1 - \sigma(z)) = -\ln(1-(1/(1+e^{-z})))$$

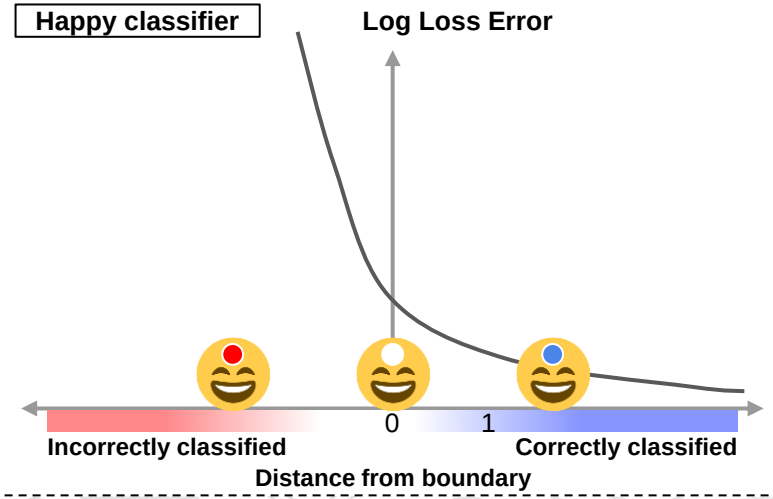




The Log Loss (Cross Entropy Loss)

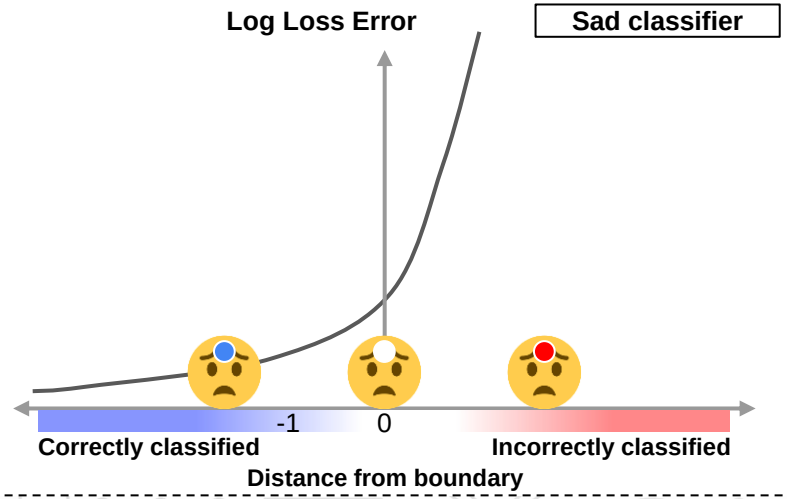
If y=1 $z = ax1 + bx2 + c$

Loss = $-\ln(\sigma(z)) = -\ln(1/(1+e^{-z}))$

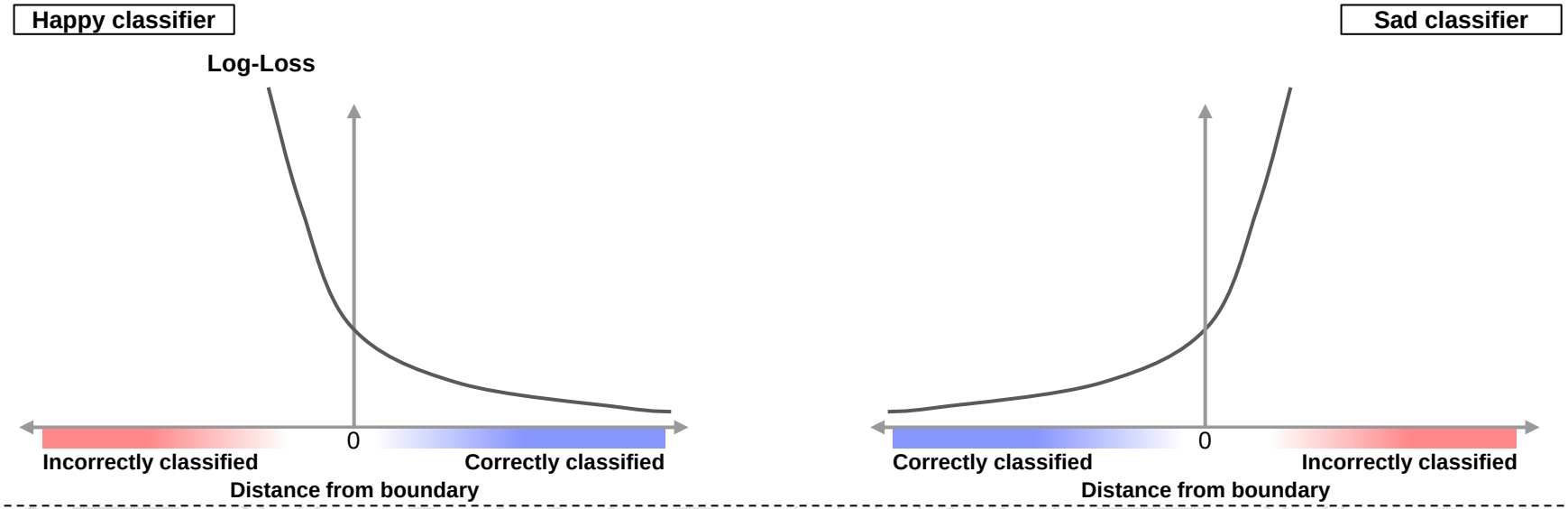


If y=0

Loss = $-\ln(1 - \sigma(z)) = -\ln(1-(1/(1+e^{-z})))$

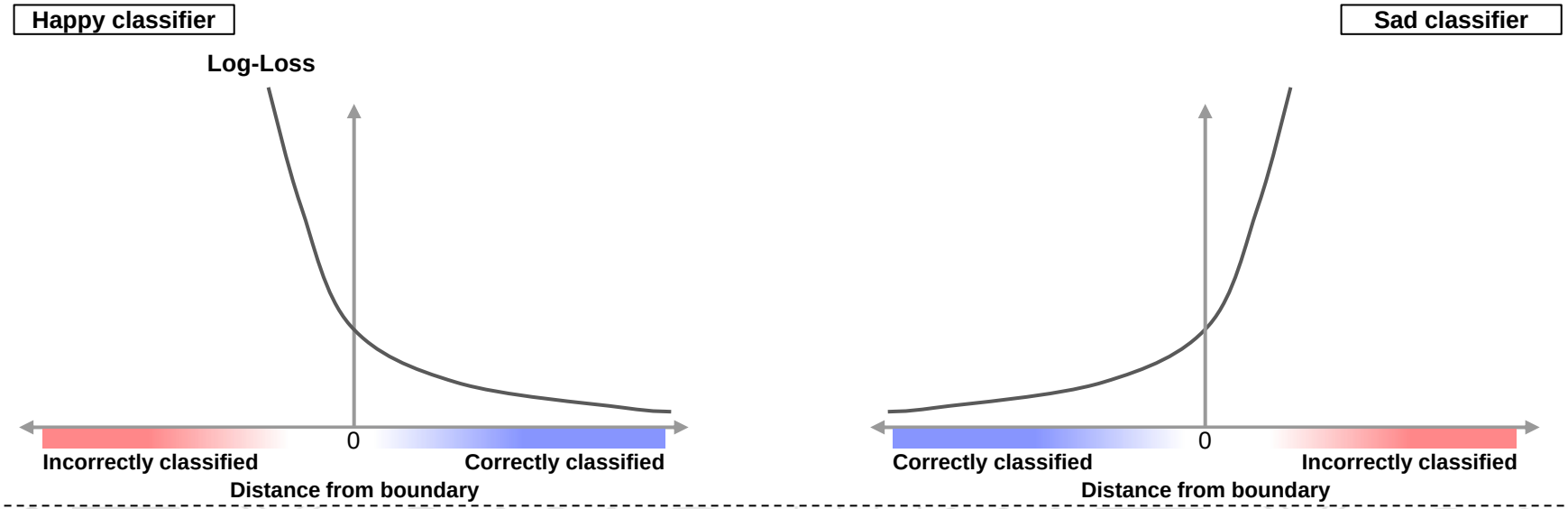


Hinge Loss


Happy classifier
Log-Loss
0
Incorrectly classified
Correctly classified
Distance from boundary
Sad classifier
0
Correctly classified
Incorrectly classified
Distance from boundary

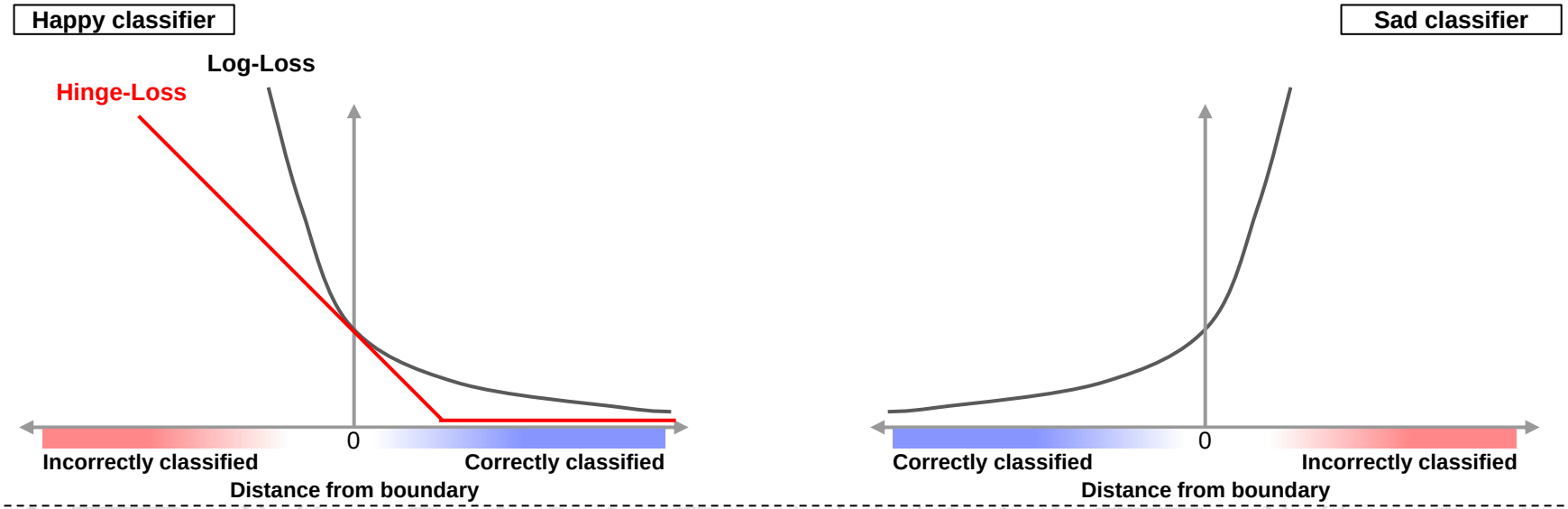
Hinge Loss

SVM uses hinge-loss instead of log-loss for the classification error term.



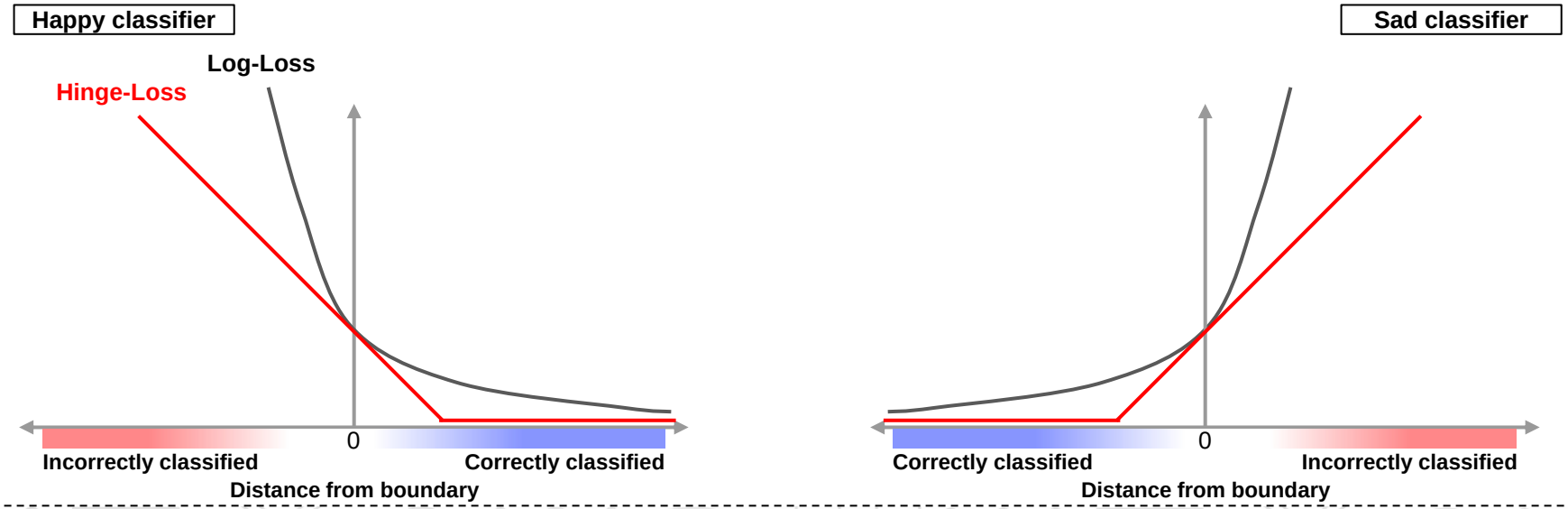
Hinge Loss

SVM uses hinge-loss instead of log-loss for the classification error term.



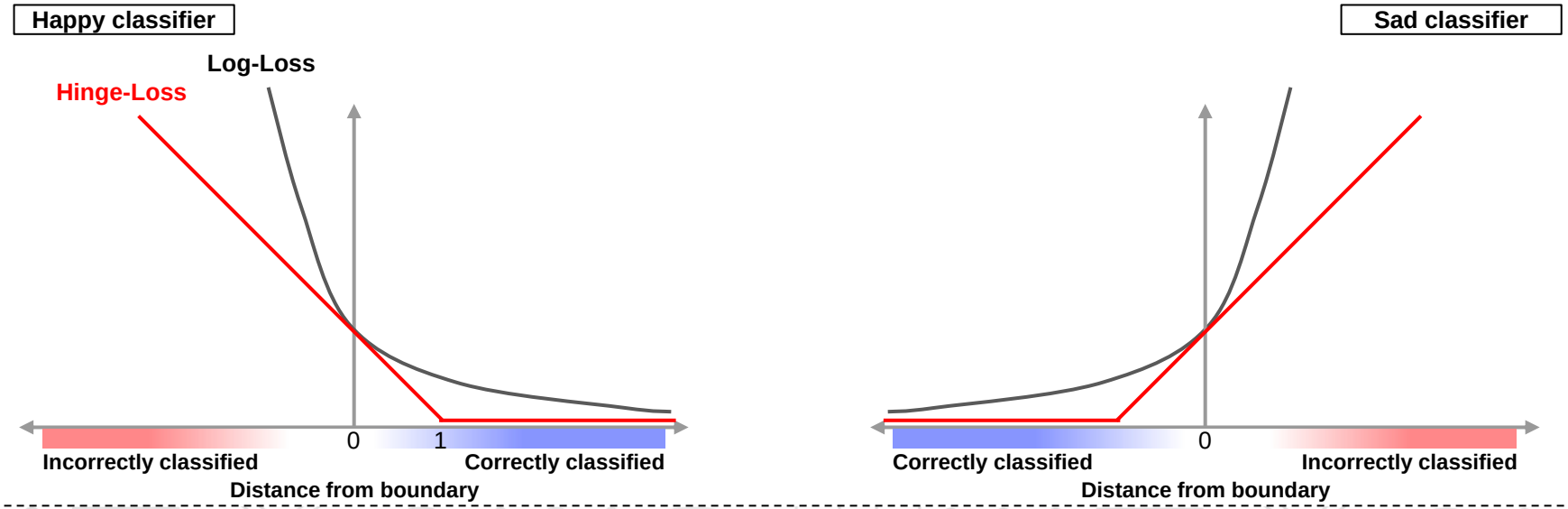
Hinge Loss

SVM uses hinge-loss instead of log-loss for the classification error term.



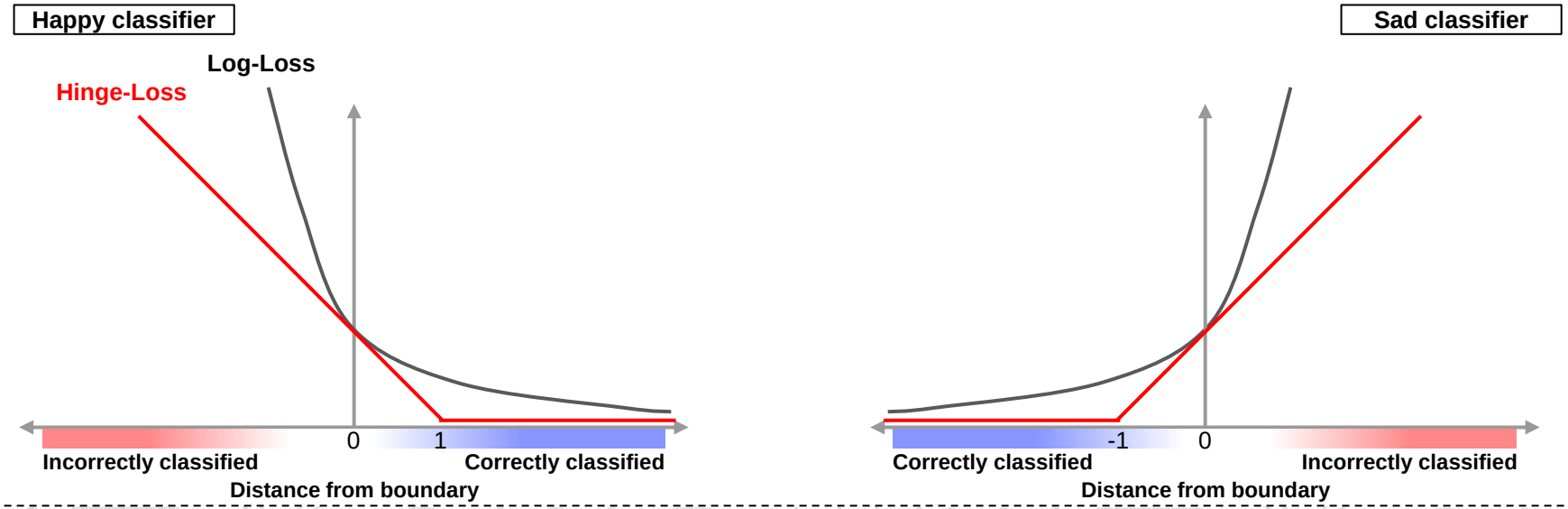
Hinge Loss

SVM uses hinge-loss instead of log-loss for the classification error term.



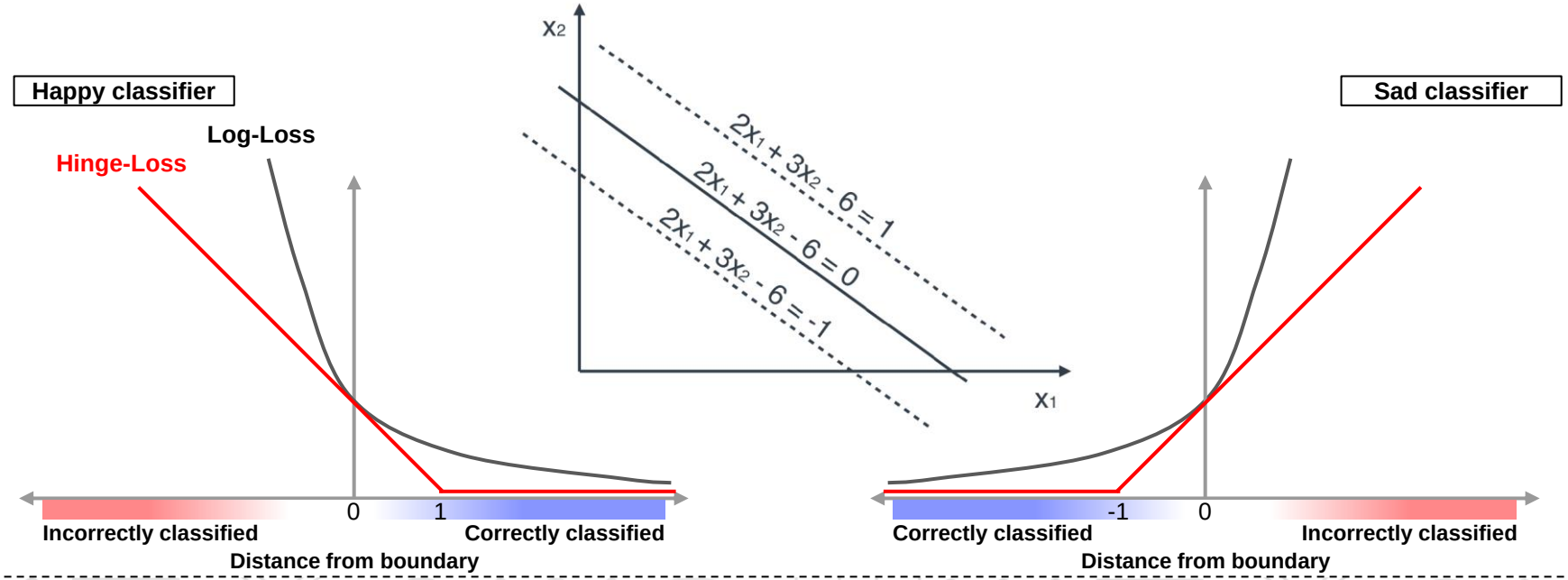
Hinge Loss

SVM uses hinge-loss instead of log-loss for the classification error term.



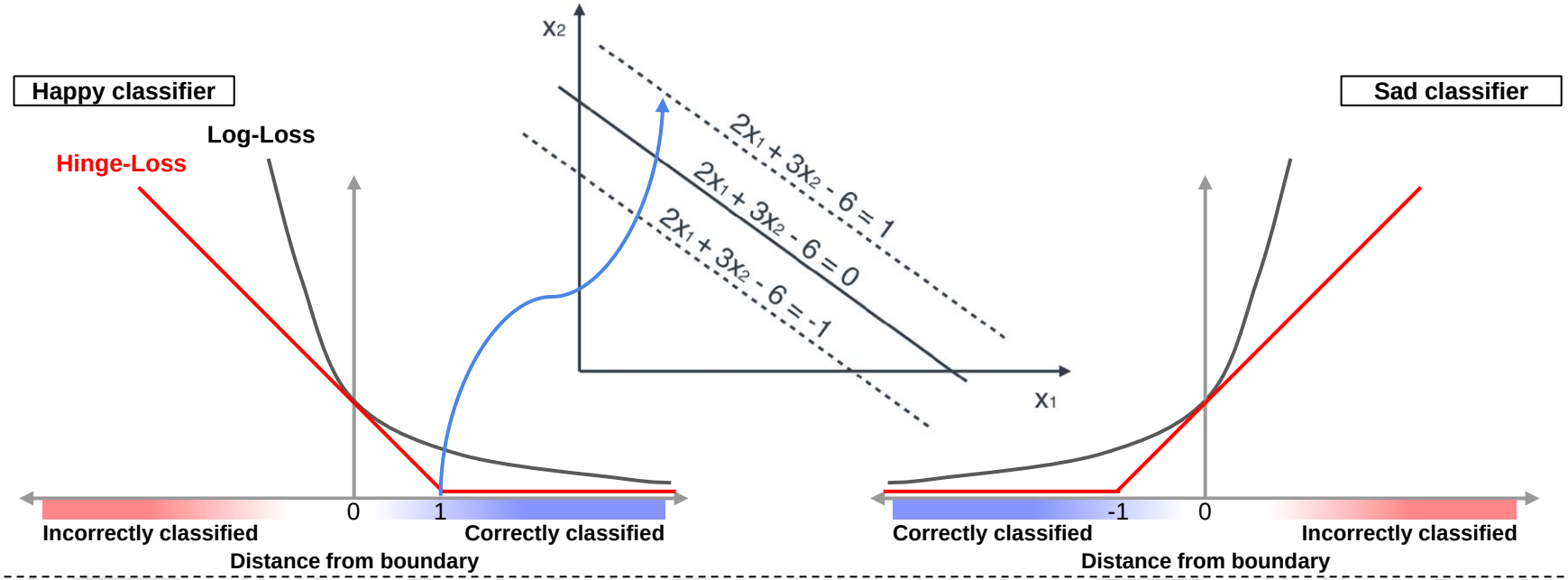
Hinge Loss

SVM uses hinge-loss instead of log-loss for the classification error term.



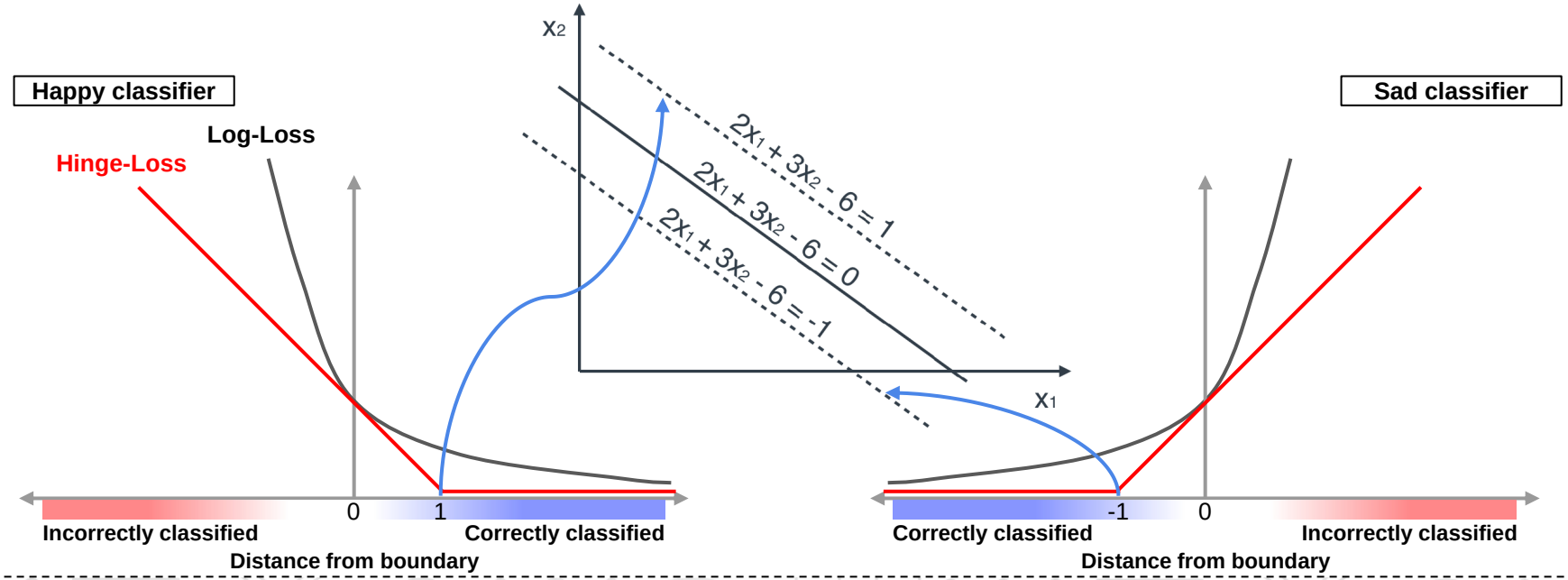
Hinge Loss

SVM uses hinge-loss instead of log-loss for the classification error term.



Hinge Loss

SVM uses hinge-loss instead of log-loss for the classification error term.



Hinge Loss VS Log Loss

Hinge Loss VS Log Loss

Log loss is **better for probability estimation**.
Hinge loss is **better for determine margins** (so it's used in SVM).

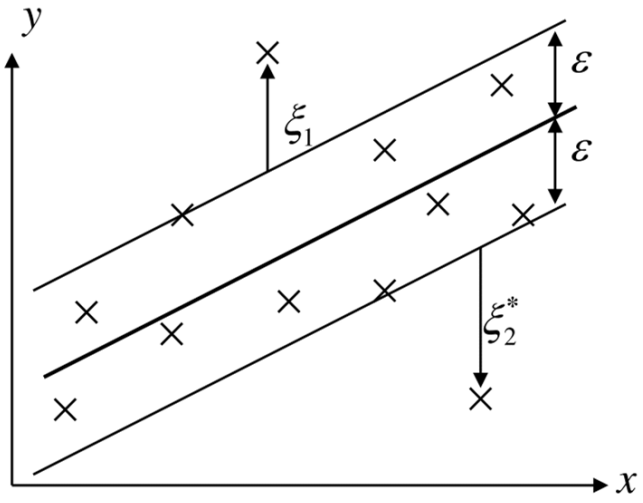
Log loss is **differentiable over the entire domain**, so it could be minimized using **gradient descent**.
Hinge loss is **not differentiable over the entire domain**, so we use **sub-gradient** descent instead of gradient descent to minimize it.

Lecture Overview

- Problem Definition
- SVM Error Function
- Classification Error
- Distance Error
- Linear SVM coding
- Polynomial Kernel
- Polynomial Kernel Coding
- RBF Kernel
- RBF Kernel Coding
- Hinge Loss
- Support Vector Regression (SVR)

Support Vector Regressor (SVR)

Support Vector Regressor (SVR)



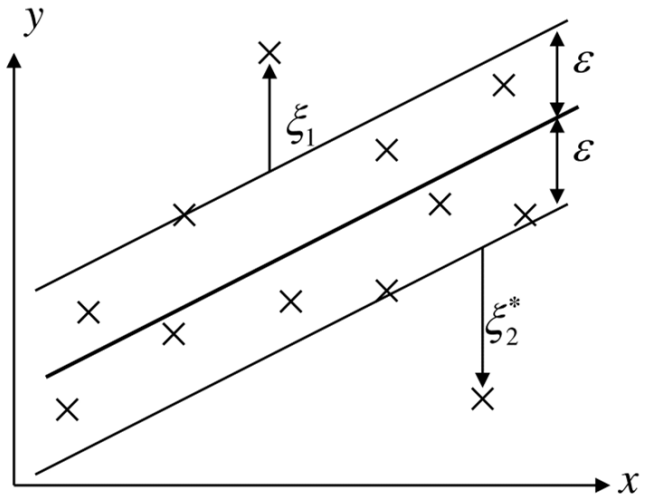
Support Vector Regressor (SVR)

ε: vertical distance margin hyper parameter. (predefined distance)

The goal is to include more points as possible within the **ε** tube.

The error of the points within the tube = zero
So the SVR seeks to include as much points there

With this way, we ensure to cover the overfitting problem.



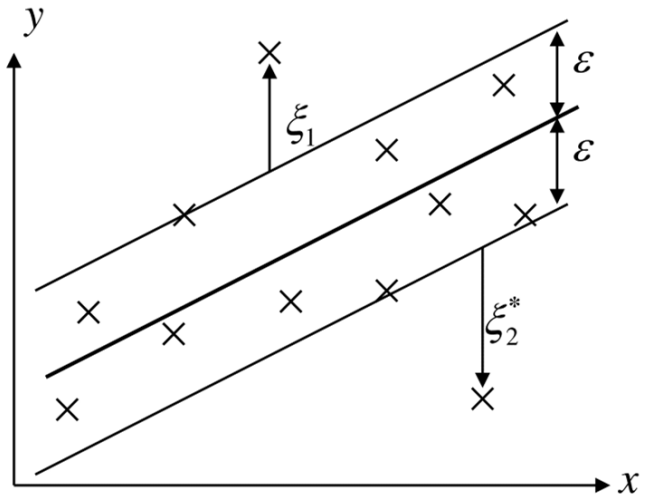
Support Vector Regressor (SVR)

ε: vertical distance margin hyper parameter. (predefined distance)

The goal is to include more points as possible within the **ε** tube.

The error of the points within the tube = zero
So the SVR seeks to include as much points there

With this way, we ensure to cover the overfitting problem.



Error to be minimized

$$\frac{1}{2}\| \mathbf{w} \|^2 + C \sum_{i=1}^{m} (\xi_i + \xi_i^*)$$

SVR Coding

Use colab to open this github notebook:

“s7s/machine_learning_1/support_vector_machines/SVM_coding.ipynb”

Feedback