DD2421 Machine Learning

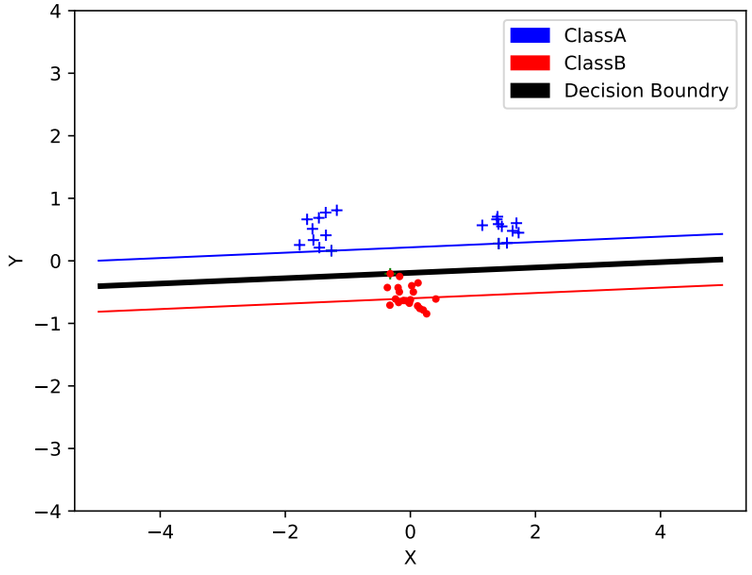
Lab 2: Support Vector Machines

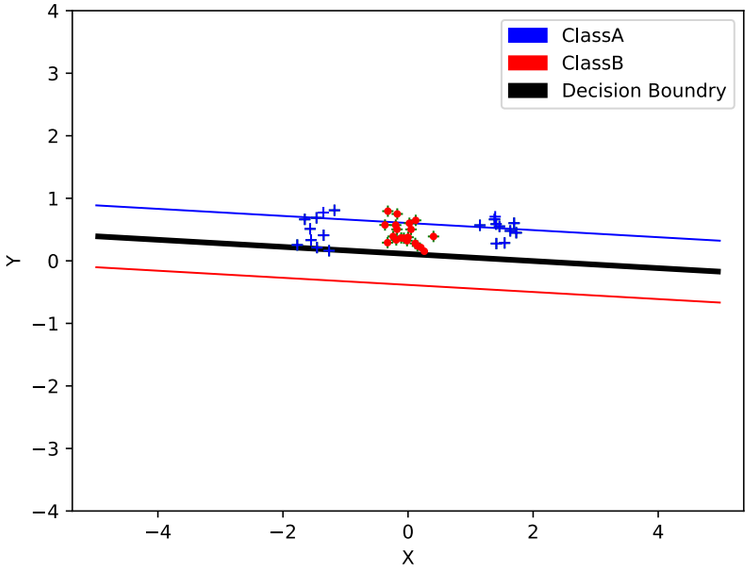
Daiki Shirafuji
daikish@kth.se
Takuya Nishi
nishi@kth.se

Move the clusters around and change their sizes to make it easier or harder for the classifier to find a decent boundary. Pay attention to when the optimizer (minimize function) is not able to find a solution at all.

Move the cluster (Class B) and train SVM with linear function and C=10. (deviation=0.2)

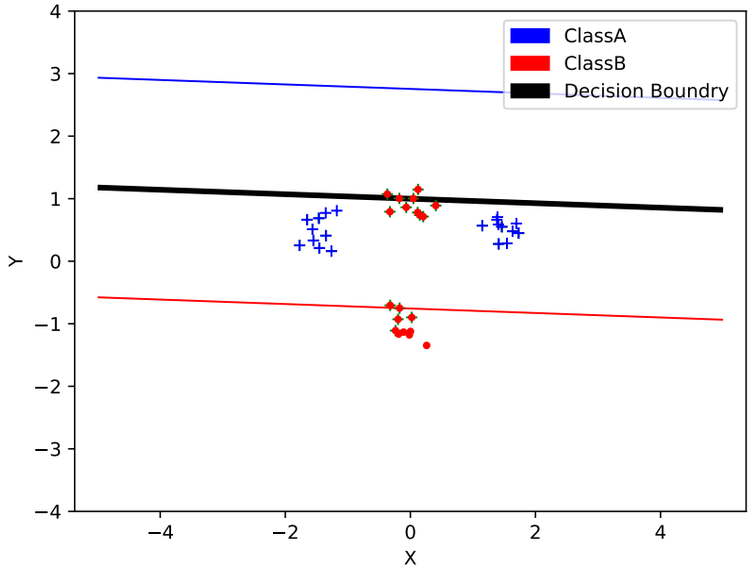
B: [0.0, -0.5]

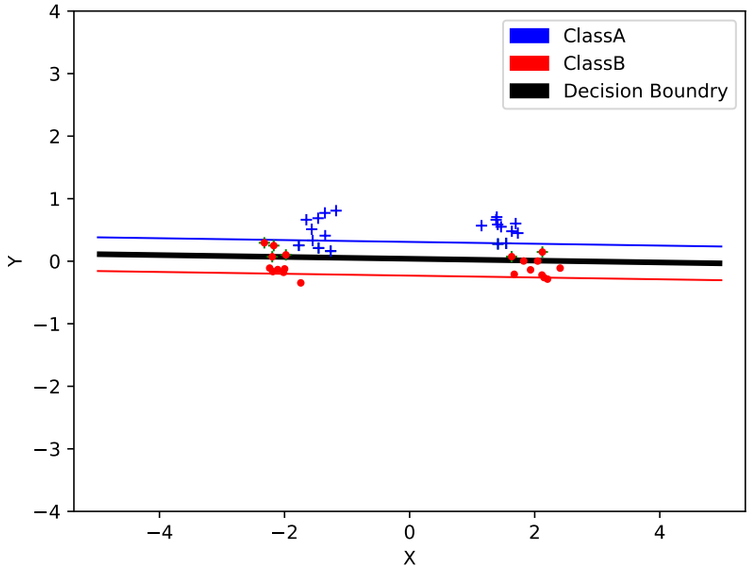




B: [0.0, 0.5]

B: [0.0, 1.0] and [0.0, -1.0]



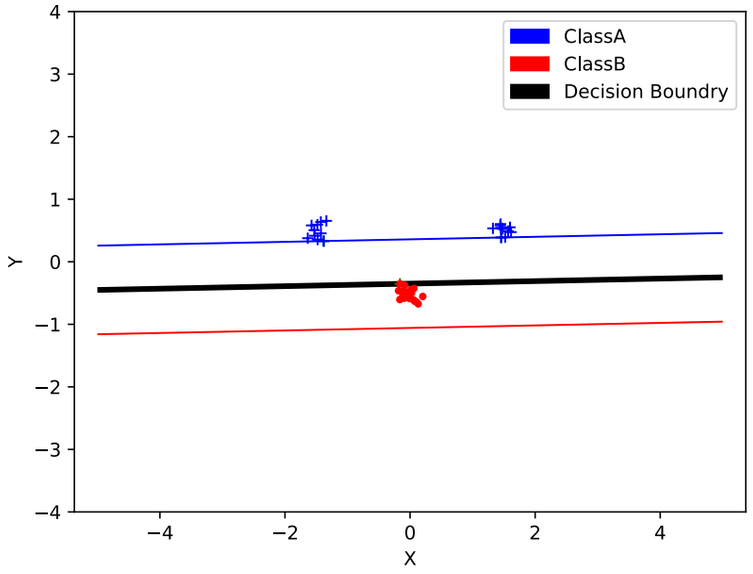


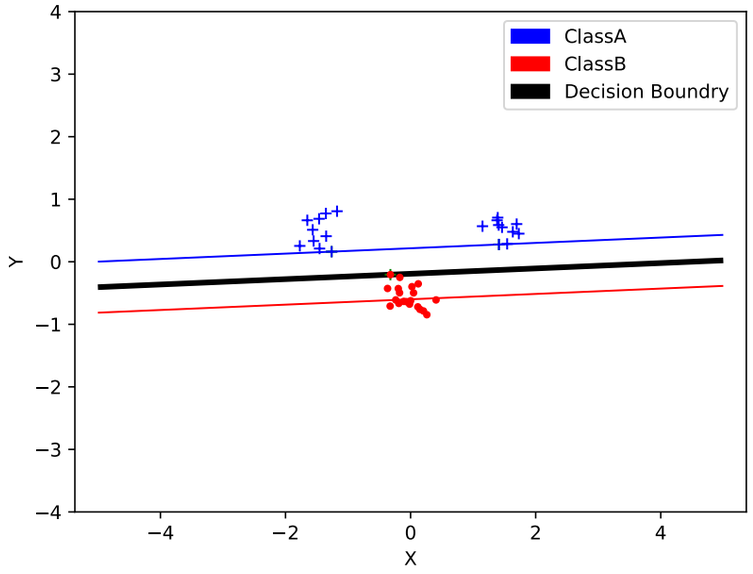
B: [2.0, 0.0] and [-2.0, 0.0]

Move the clusters around and **change their sizes** to make it easier or harder for the classifier to find a decent boundary. Pay attention to when the optimizer (minimize function) is not able to find a solution at all.

Change the size (deviation = 0.1, 0.2, 0.3, 0.4) and train SVM with linear function and C=10.

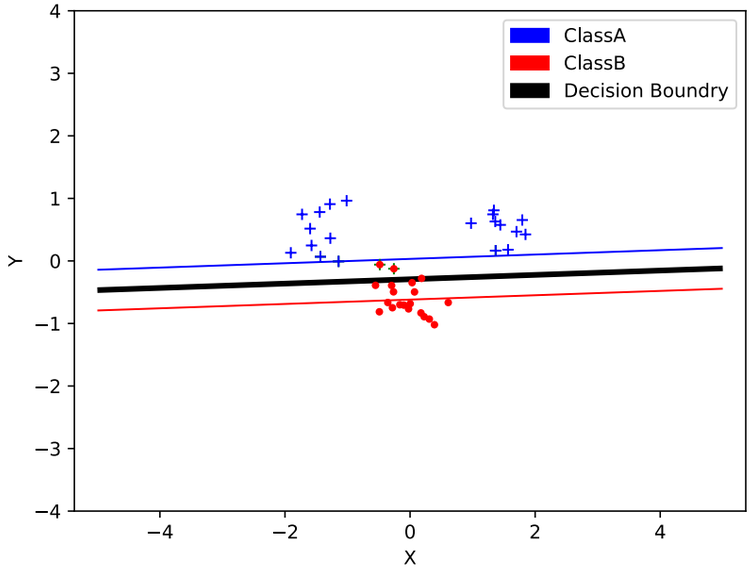
deviation: 0.1

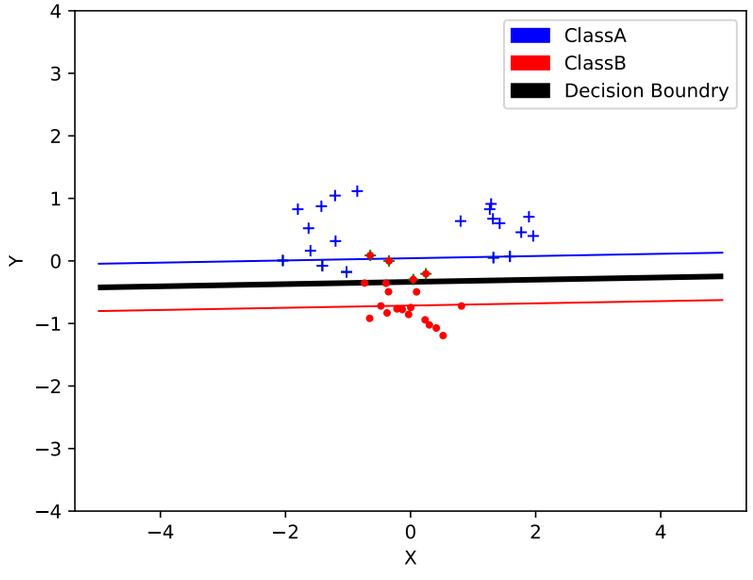




deviation: 0.2

deviation: 0.3





deviation: 0.4

Move the clusters around and change their sizes to make it easier or harder for the classifier to find a decent boundary. Pay attention to **when the optimizer (minimize function) is not able to find a solution at all.**

- Whether SVMs with the linear function can correctly divide classes depends on a dataset characteristic. If a dataset can be divided into different classes with a straight line, it is possible to find a solution.

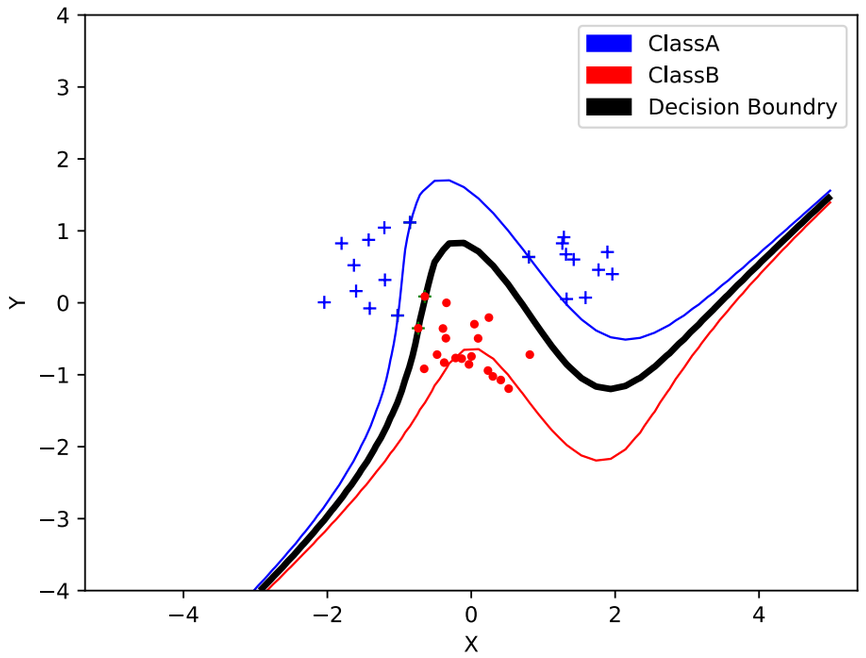
- However, in the case that straight lines cannot separate the dataset into classes, e.g. the deviation is large or different classes are located nearly, it is not efficient to use linear kernel function.

Implement the two non-linear kernels.
You should be able to classify very hard data sets with these.

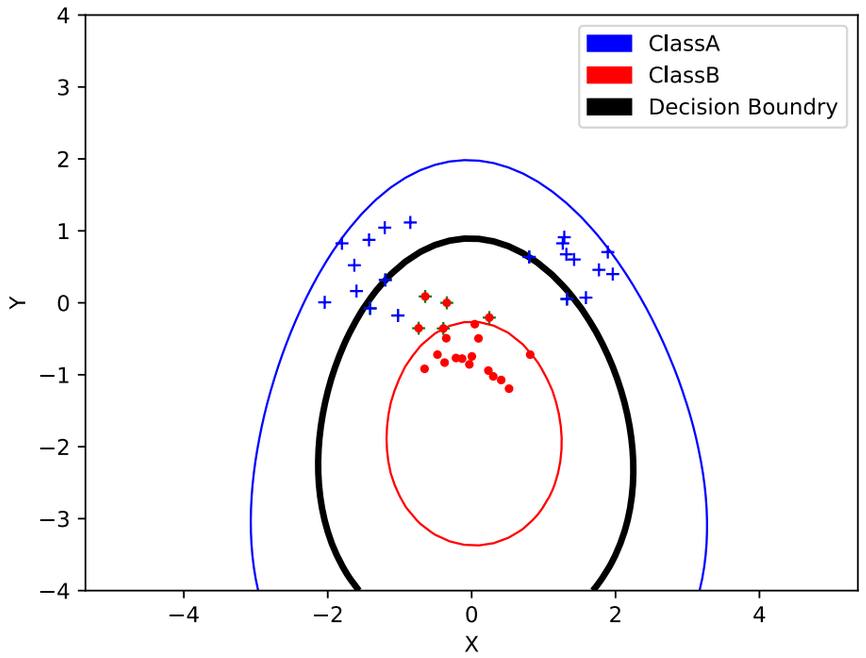
```python
# Implement Kernel Functions
def kernel(x, y, k_type):
    # Linear
    if (k_type == 'linear'):
        return np.dot(x, y) + 1
    # Polynomials ::: There is 1 para, p
    elif (k_type == 'poly'):
        return np.power((np.dot(x, y) + 1), polynomial)
    # RBF ::: The parameter sigma is used to control the smoothness of the boundary.
    elif (k_type == 'rbf'):
        difference = np.subtract(x, y)
        return math.exp((-np.dot(difference,difference))/(2*sigma*sigma))
else:
    print("ERROR in kearnel name!!!")
```

Kernel Functions: Linear, **Polynomials**, and **RBF**
The example is shown below with deviation=0.4
(parameters: C=10, p=3 (poly), sigma=3 (rbf)).

Polynomials



RBF

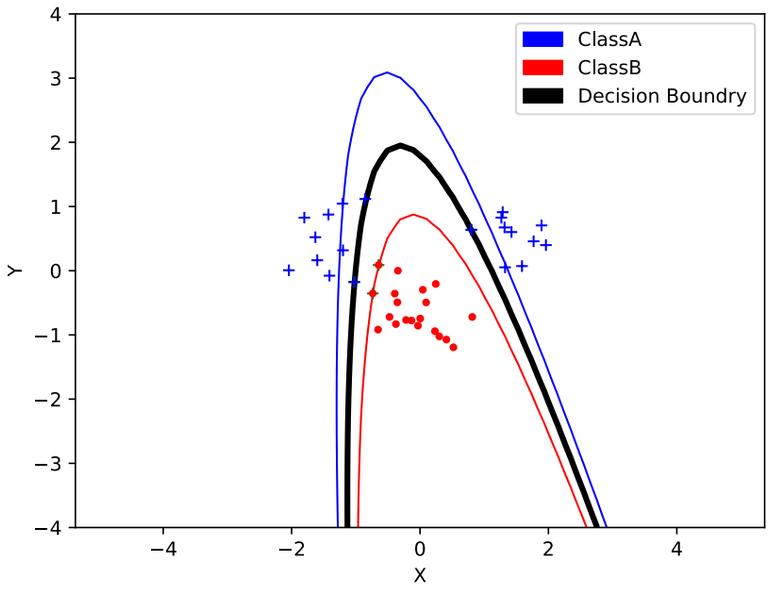


The non-linear kernels have parameters; explore how they influence the decision boundary. Reason about this in terms of the bias-variance trade-off.

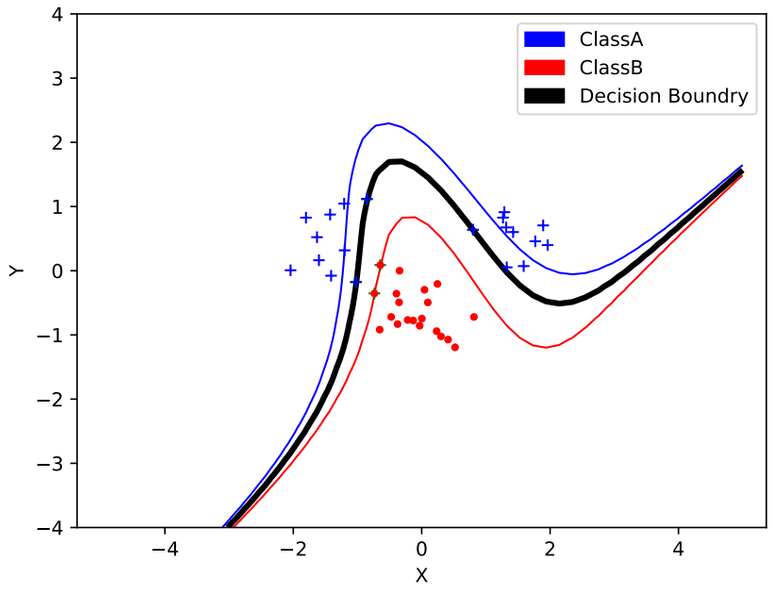
Polynomial Kernel (p: degree of a polynomial)

As p increases, its bias become high and its variance become low (Overfitting).
Otherwise, its bias become low and its variance become high.

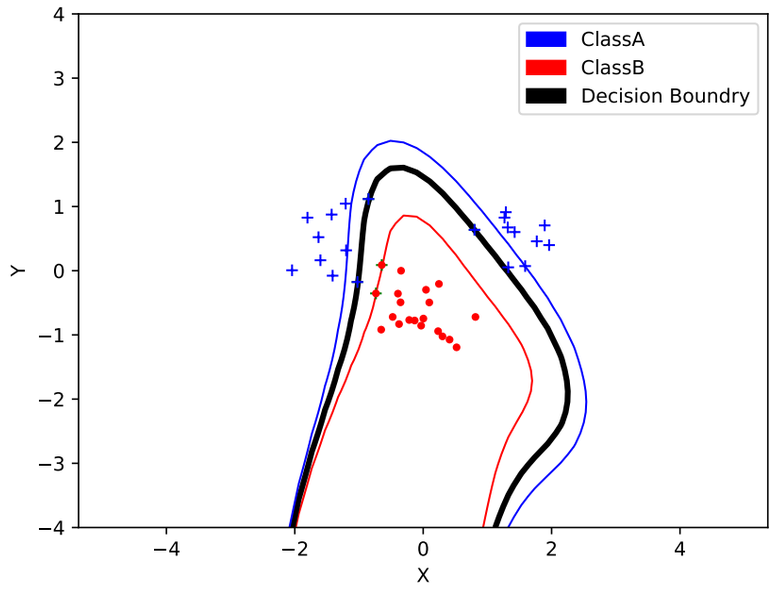
Polynomial (p=2, 3, 4, deviation=0.4)



p = 2



p = 3



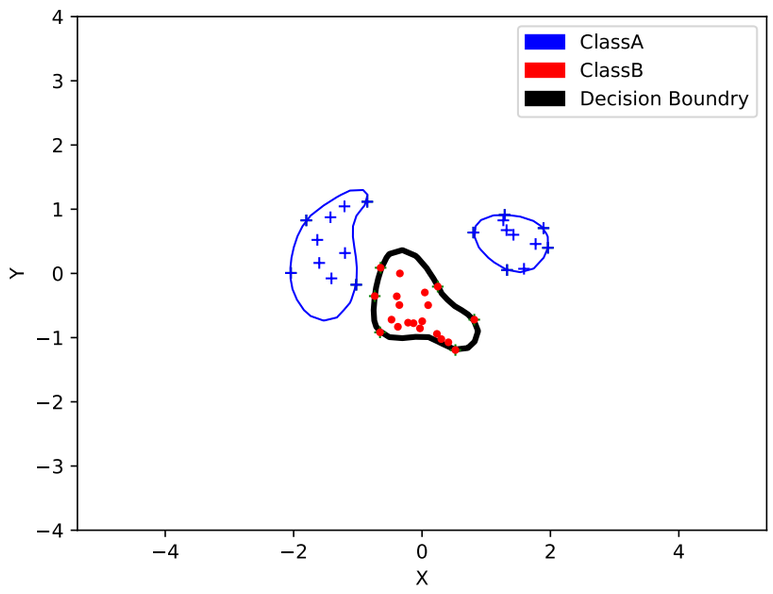
p = 4

The non-linear kernels have parameters; explore how they influence the decision boundary. Reason about this in terms of the bias-variance trade-off.

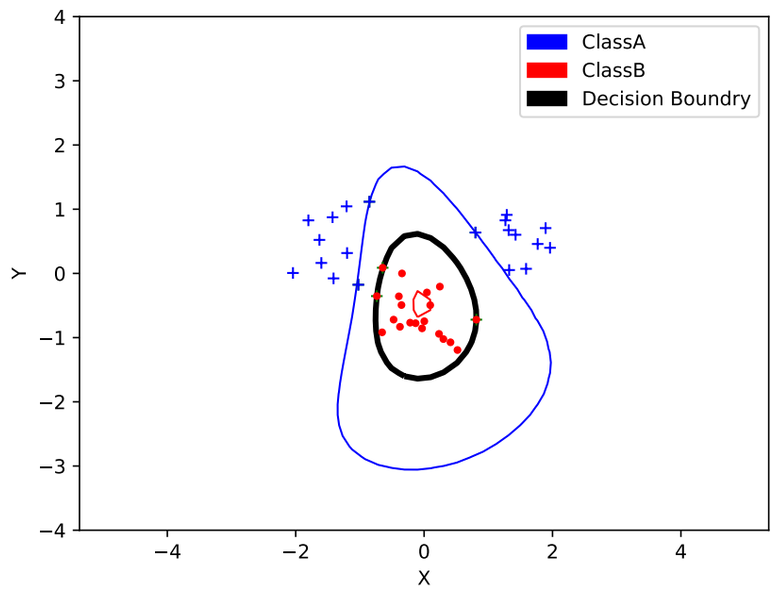
RBF (σ: control the smoothness of the boundary)

High σ makes a decision boundary roughly, and low σ makes it smoothly.
i.e. High σ means high variance and low bias
Low σ means low variance and high bias (Overfitting)

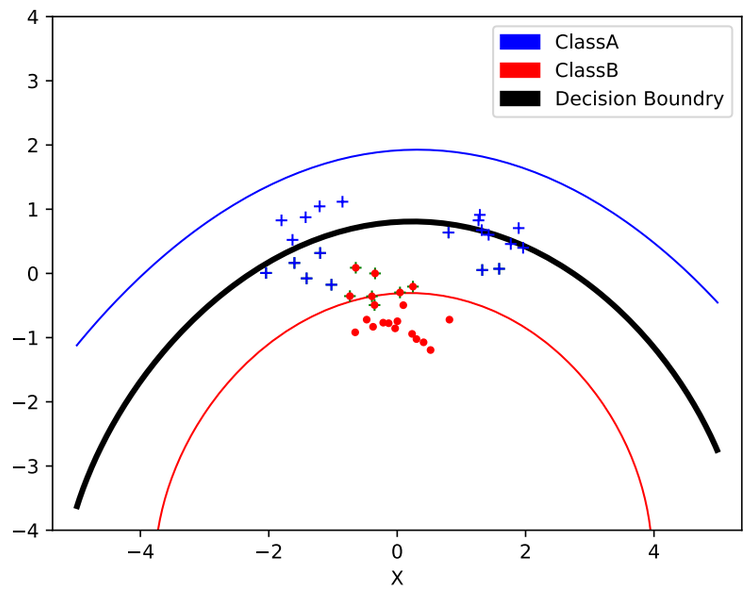
RBF (σ=2, 3, 4, deviation=0.4)



σ = 0.5



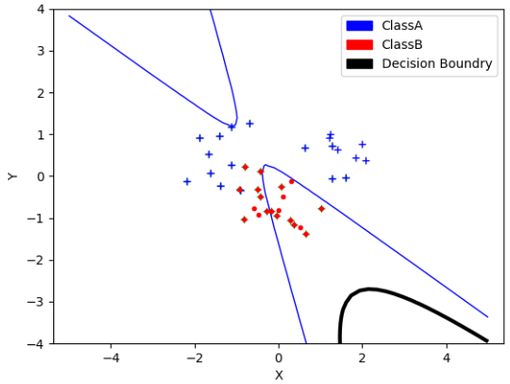
σ = 1



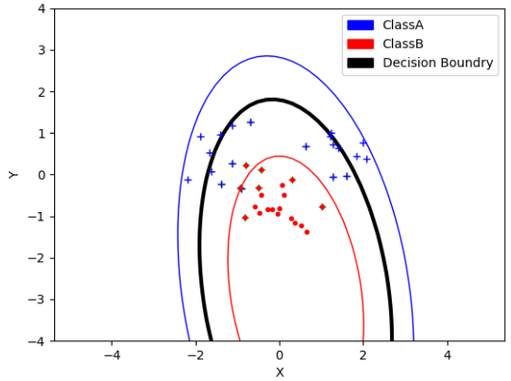
σ = 5

Explore the role of the slack parameter C.
What happens for very large/small values?

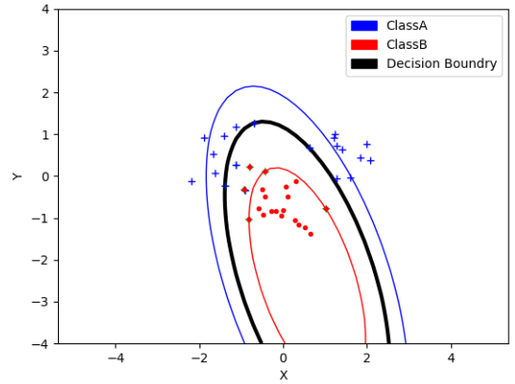
The role of the slack parameter C is **to control slack values.** C can **give more penalty to ξ as C increases**.
So, we can say:
The larger the value of C is, the smaller the margin become. Otherwise, the margin become larger.



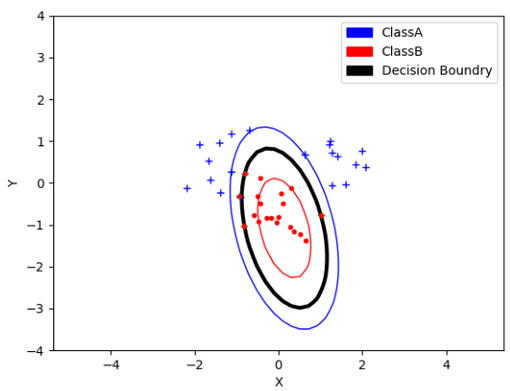
C=0.01



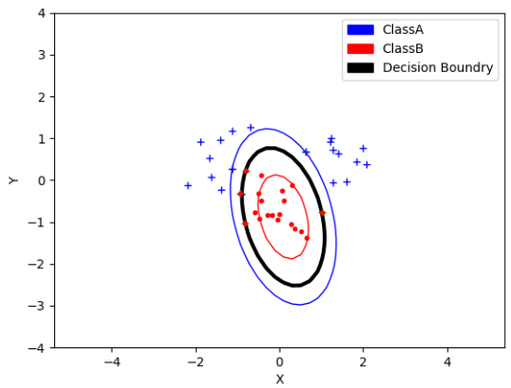
C=0.1



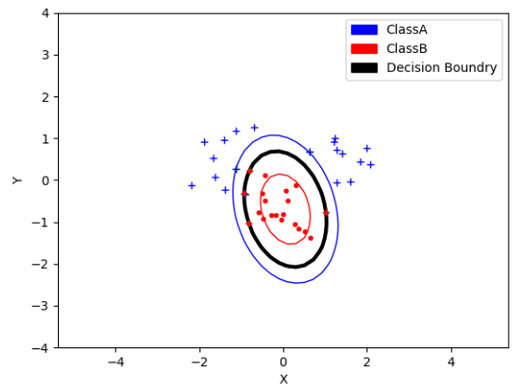
C=1



C=10



C=100

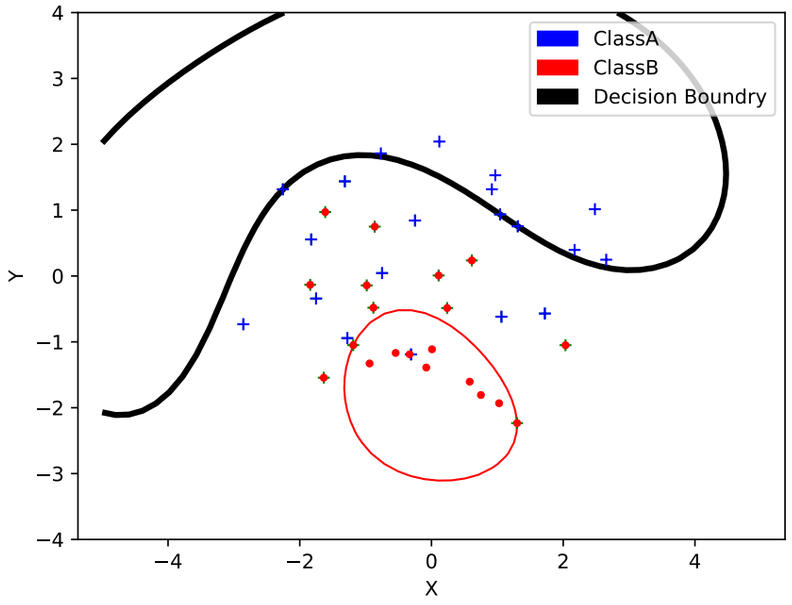


C=1000

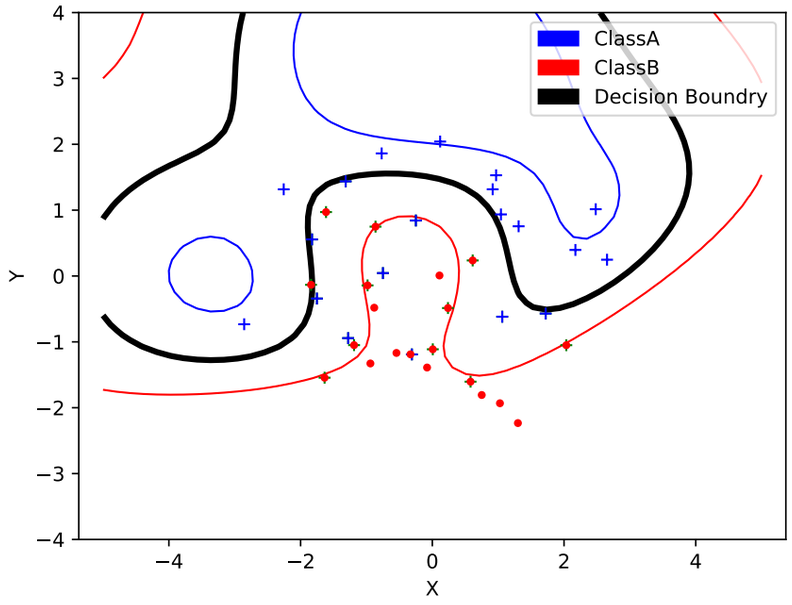
Imagine that you are given data that is not easily separable. When should you opt for **more slack rather than going for a more complex model (kernel)** and vice versa?

- When different classes are so mixed, and model cannot classify these classes correctly even with complex model, we have to make the slack increased.

RBF (C=1, deviation: 1.0)



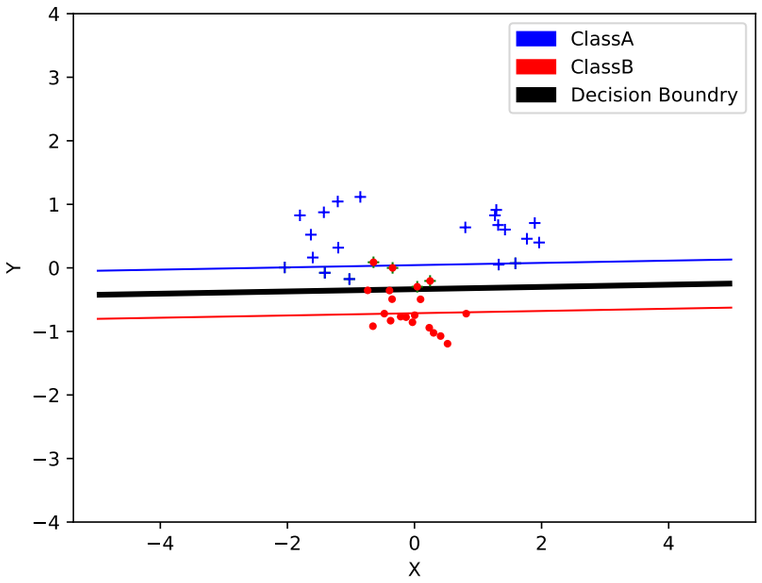
RBF (C=100, deviation: 1.0)



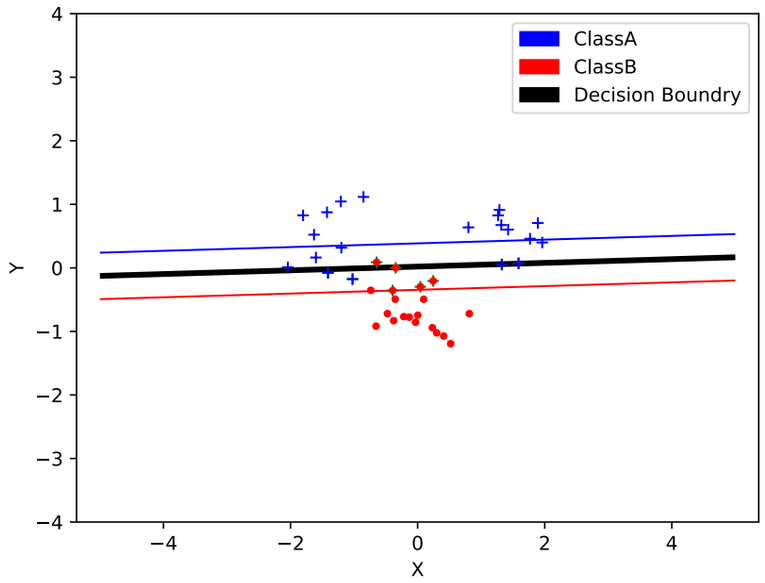
Imagine that you are given data that is not easily separable. When should you opt for more slack rather than going for a more complex model (kernel) and **vice versa**?

- When different classes are not so mixed, but cannot classified with not complex model, we have to use more complex model rather than make the slack increased.

Linear
(deviation: 0.4, C=10)



Linear
(deviation: 0.4, C=1000)



Polynomial
(p=3, deviation=0.4, C=10)