Lab 6. SVM Revised

Intro to Machine Learning
Fall 2018, Innopolis University

Plan

- Soft-margin SVM
- Kernel trick for SVM
- Homework Explanation

Soft-margin SVM

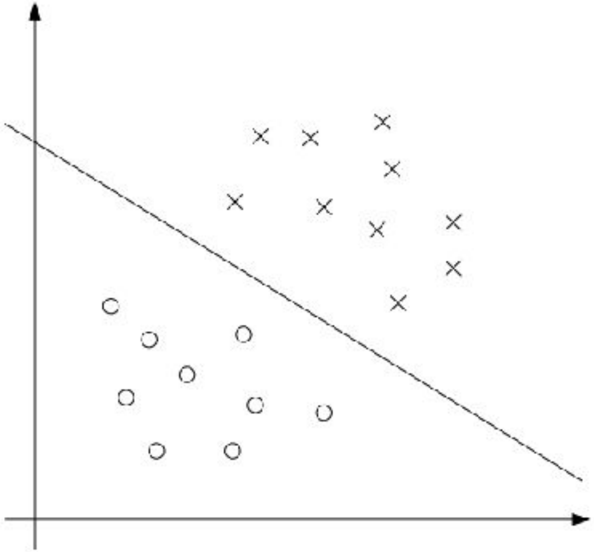
Why do we need it?

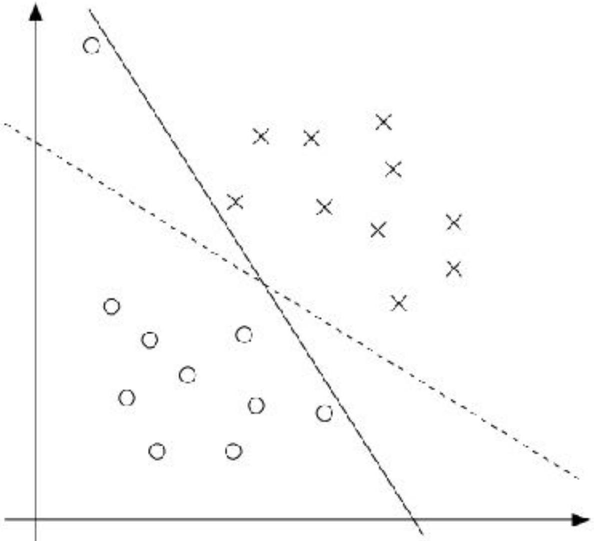
Soft-margin SVM

Why do we need it?

- Non-linearly separable examples
- Outliers?

Soft-margin SVM. Outliers





Soft-margin SVM

To make the algorithm work for non-linearly separable datasets as well as be less sensitive to outliers, we reformulate our optimization as follows:

$$
\begin{aligned}
\min_{w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\
\text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \\
& \xi_i \geq 0, \quad i = 1, \ldots, m.
\end{aligned}
$$

Soft-margin SVM

After forming the Lagrangian, setting the derivatives with respect to w and b to zero, substituting them back in, and simplifying, we obtain the following dual form of the problem:

$$
\begin{aligned}
\max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\
\text{s.t.} \quad & 0 \le \alpha_i \le C, \quad i = 1, \ldots, m \\
& \sum_{i=1}^{m} \alpha_i y^{(i)} = 0,
\end{aligned}
$$

Which is exactly the same as original problem except for alphas now are restricted

Soft-margin SVM. What is C responsible for?

$$\min_{w,b} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i$$

Soft-margin SVM. What is C responsible for?

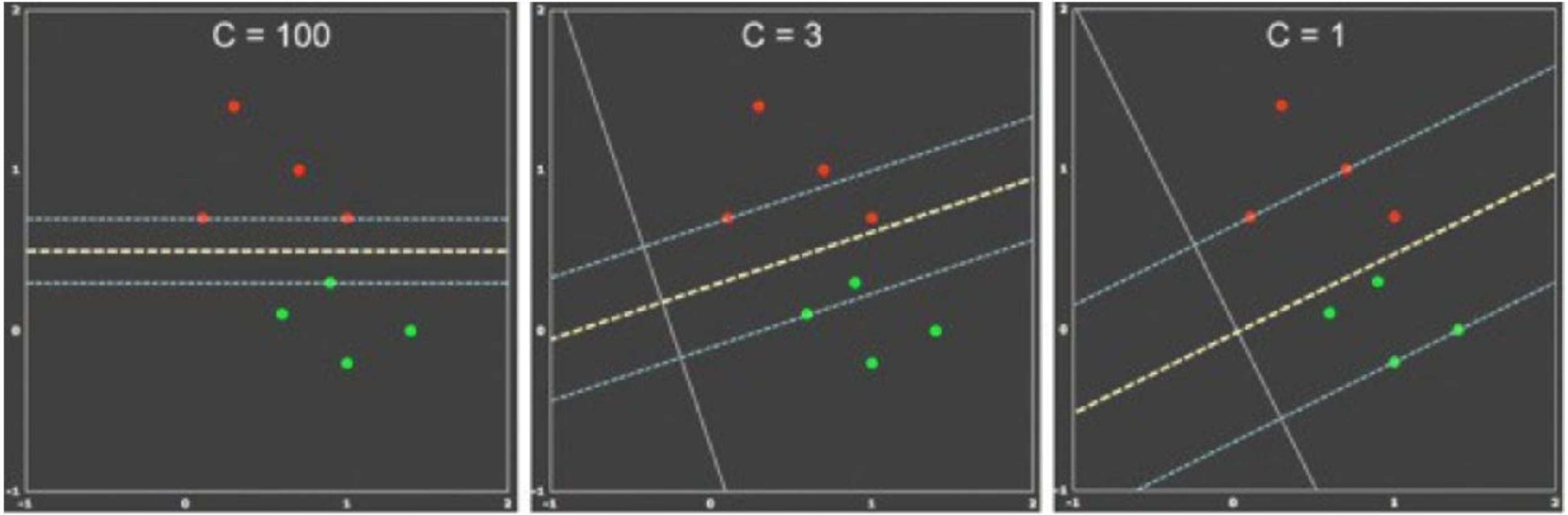
$$\min_{w,b} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i$$

The C parameter is responsible for compromise between margin width and number of misclassified examples.

Larger C => smaller-margin hyperplane, less misclassifications

Smaller C => larger-margin separating hyperplane, even if it misclassifies more points.

Soft-margin SVM. Choice of C



Change in margin with change in C

Taken from https://medium.com

Kernel trick

Kernel trick

Recall from lecture:

$$
\begin{aligned}
\underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\
\text{subject to} \quad & 0 \le \alpha_i \le C, \;\; \text{for any } i = 1, \ldots, m \\
& \sum_{i=1}^{m} \alpha_i y_i = 0
\end{aligned}
$$

$$
h(\mathbf{x}_i) = \operatorname{sign}\left(\sum_{j=1}^{S} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_i) + b \right)
$$

Kernel trick

Recall from lecture:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to} \quad 0 \le \alpha_i \le C, \ \text{for any } i = 1, \ldots, m$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0$$

$$\begin{aligned} w^T x + b &= \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}\right)^T x + b \\ &= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b. \end{aligned}$$

$$h(\mathbf{x}_i) = \operatorname{sign}\left(\sum_{j=1}^{S} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_i) + b\right)$$

Kernel trick

So, all we need is Kernel!

$$K(x, z) = \phi(x)^T \phi(z)$$

Kernel trick

Often, K(x, z) may be very inexpensive to calculate, even though φ(x) itself may be **very expensive** to calculate. Thereby, we can get SVMs to learn in the high dimensional feature space given by φ, but without ever having to explicitly find vectors φ(x).

$$K(x, z) = \phi(x)^T \phi(z)$$

Kernel trick. Example

$$K(x, z) = (x^T z)^2$$

Kernel trick. Example

$$K(x,z) = (x^T z)^2 = \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right)$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j$$

$$= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j)$$

Kernel trick. Example

$$
\begin{aligned}
K(x,z) = (x^T z)^2 \;&=\; \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right) \\
&=\; \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j \\
&=\; \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j)
\end{aligned}
$$

$$
\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}
$$

Kernel trick. Example

$$O(n^2)$$

$$K(x,z) = (x^T z)^2 = \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right)$$

$$O(n)$$

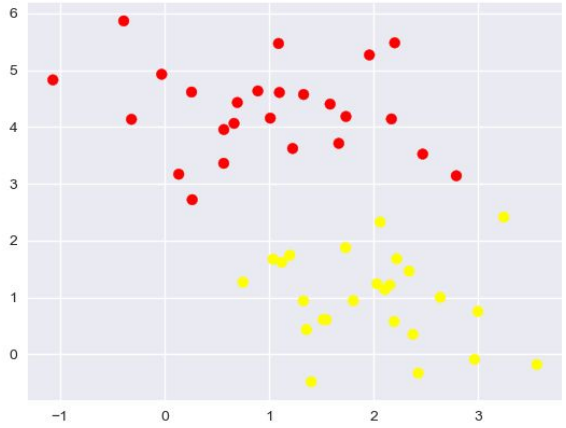
$$= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j$$

$$= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j)$$

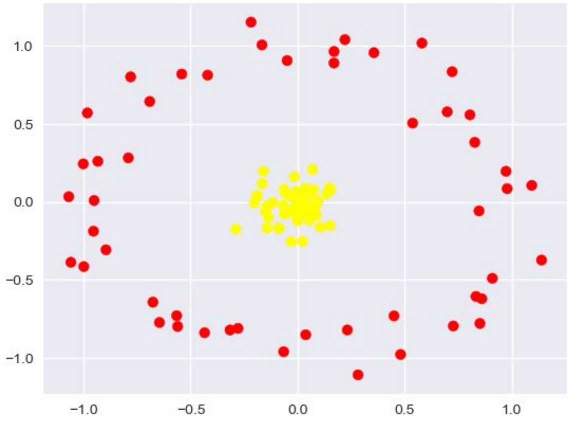
$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}$$

Kernel trick. Let’s see how it works in practice!

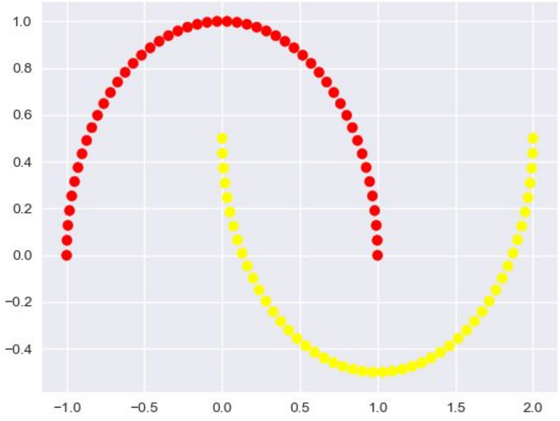
We have three toy sets of data:



Linearly separable



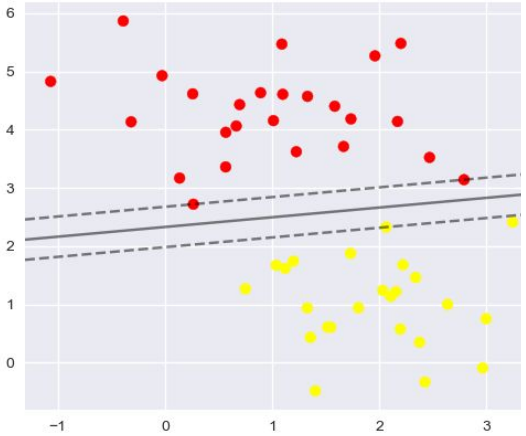
Concentric



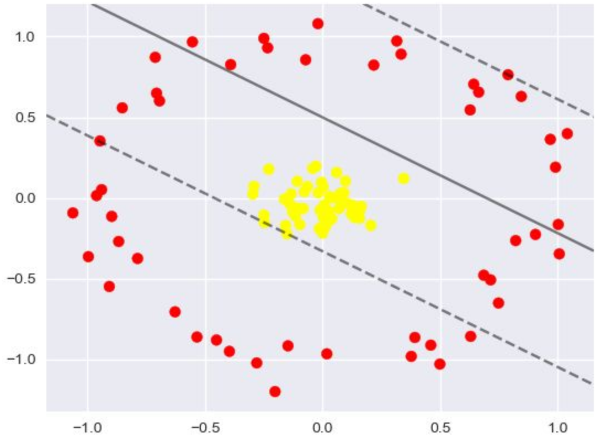
Moon-shaped

Kernel trick. Linear kernel

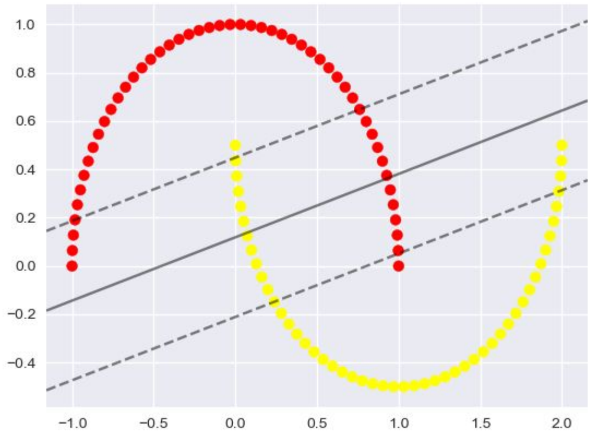
$$K(x, z) = x^T z$$



Linearly separable



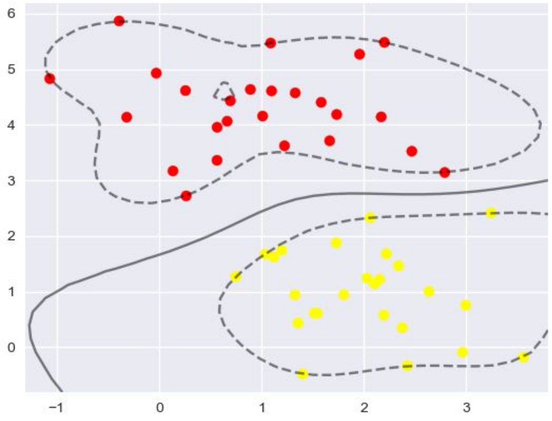
Concentric



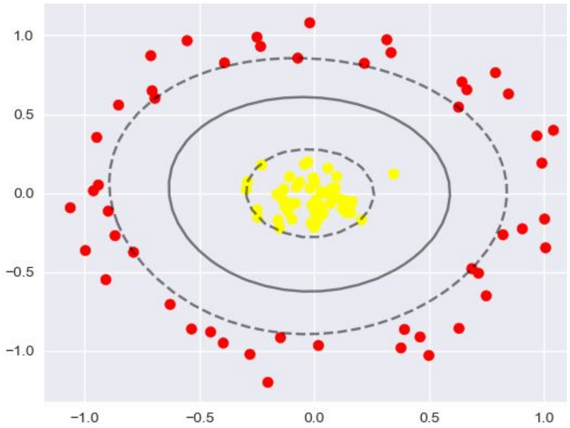
Moon-shaped

Kernel trick. RBF kernel

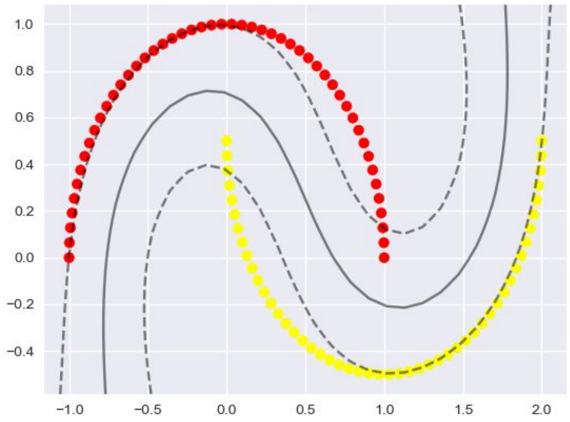
$$K(x, z) = \exp\left(-\frac{||x - z||^2}{2\sigma^2}\right)$$



Linearly separable



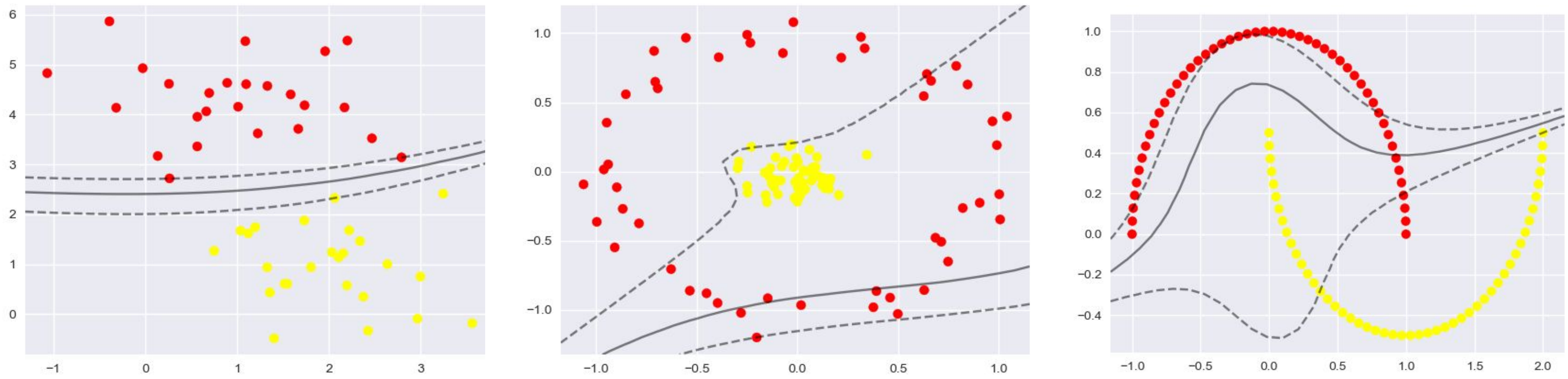
Concentric



Moon-shaped

Kernel trick. Polynomial kernel (degree = 2)

$$K(x, z) = (x^T z + c)^2$$



Linearly separable Concentric Moon-shaped

Cool! But how do we actually find alphas?

$$\begin{aligned}
\underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\
\text{subject to} \quad & 0 \le \alpha_i \le C, \ \text{ for any } i = 1, \ldots, m \\
& \sum_{i=1}^{m} \alpha_i y_i = 0
\end{aligned}$$

Cool! But how do we actually find alphas?

$$\begin{aligned}
\underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\
\text{subject to} \quad & 0 \le \alpha_i \le C, \ \text{for any } i = 1, \ldots, m \\
& \sum_{i=1}^{m} \alpha_i y_i = 0
\end{aligned}$$

It is a problem solvable by **Quadratic Programming** methods (optimizing a quadratic function of several variables subject to linear constraints)

QP methods are usually polynomial in time

There is a better solution for our particular problem - **Sequential Minimal Optimization** (SMO)

SMO

There is a better solution for our particular problem - **Sequential Minimal Optimization** (SMO)

Based on Coordinate ascent method:

Loop until convergence: {

For $i = 1, \ldots, m$, {

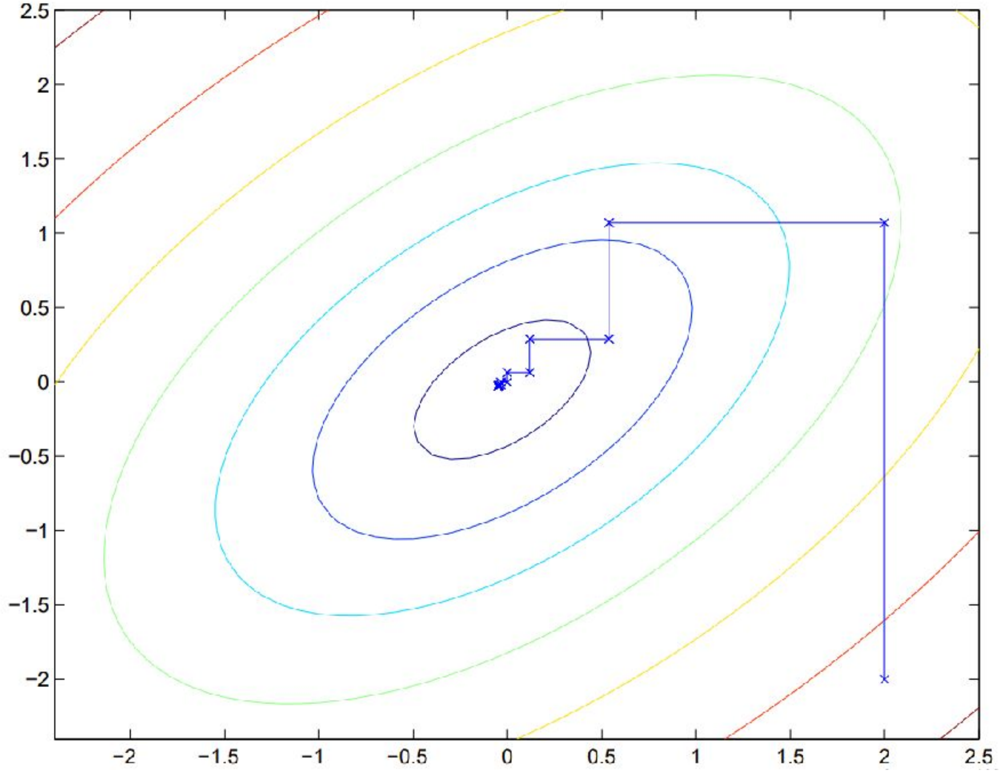
$$\alpha_i := \arg\max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$$

}

}

SMO

Coordinate ascend in action:



For SMO we are changing 2 alphas at each iteration, because of this constraint:

$$\sum_{i=1}^{m} \alpha_i y_i = 0$$

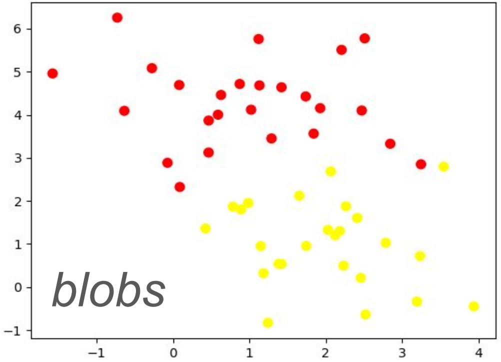
Highly recommended reading

http://cs229.stanford.edu/notes/cs229-notes3.pdf

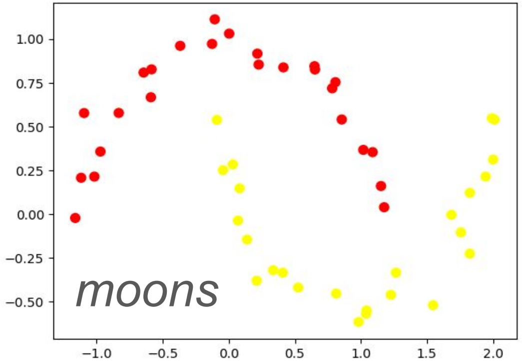
Refer to it when doing your homework, especially part 8 - *Regularization and the non-separable case*

Homework

You have 2 sets of data: *blobs*



and *moons*



You are asked to implement **soft-margin** SVM with 2 types of **kernel**:

- linear $K(x, z) = x^T z$
- polynomial $K(x, z) = (x^T z + c)^d$ in our case we choose $c = 1$, $d = 3$

In the template library QP solver is used to find Lagrangians (alphas) instead of SMO, and all necessary plotting is provided.

Your task is to define the function to optimize, 2 kernel types, define constraints, calculate w and b parameters given alphas, and make predictions.

At the end we want to see 4 pictures for each combination of data and kernels.

Homework (Continued)

The last one (moons with polynomial kernel) should look like follows:

This is what you can do with **kernels**!