Non-linear SVM

Lecture 6b

Last time

✓ Know **maximum margin** classification
✓ Derive **objective function** for Linear SVM
✓ Handle **soft-margin** classification with Hinge Loss

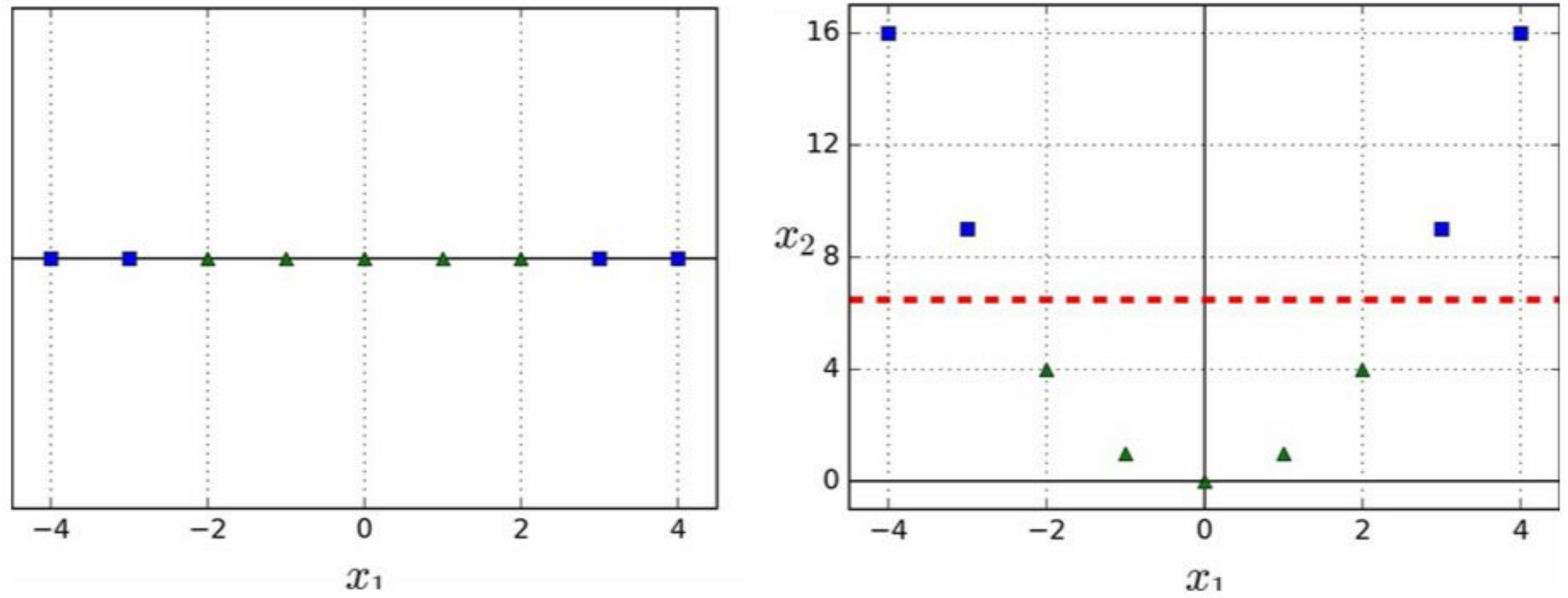
Today: Learning Objectives

1. Expand to **non-linear** case
2. Formulate the **dual problem**
3. Understand **the kernel trick**

1. Expand to non-linear Classification

Non-linear SVM Classification (more powerful!)

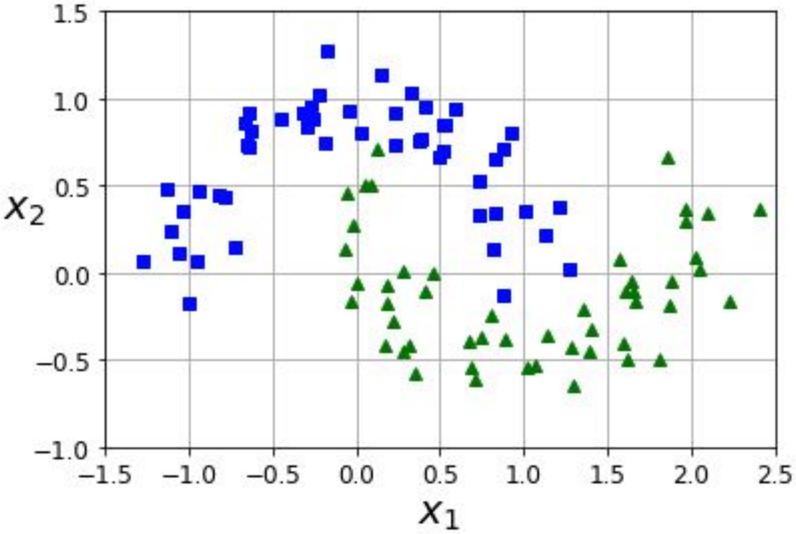
What should we do if the dataset is non-linear? Recall previously



Adding a polynomial feature ($x_2 = x_1^2$)

Code Demo on Moons dataset

```
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=100, noise=0.15, random_state=42)
```

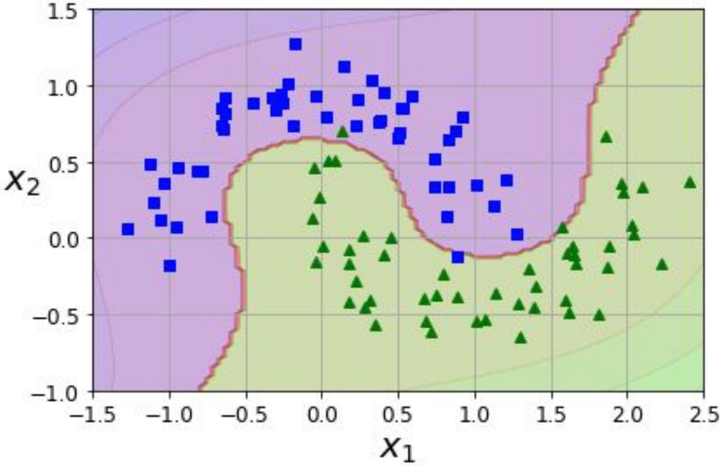


Code Demo

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

polynomial_svm_clf = Pipeline([
        ("poly_features", PolynomialFeatures(degree=3)),
        ("scaler", StandardScaler()),
        ("svm_clf", LinearSVC(C=10, loss="hinge", random_state=42))
    ])

polynomial_svm_clf.fit(X, y)
```



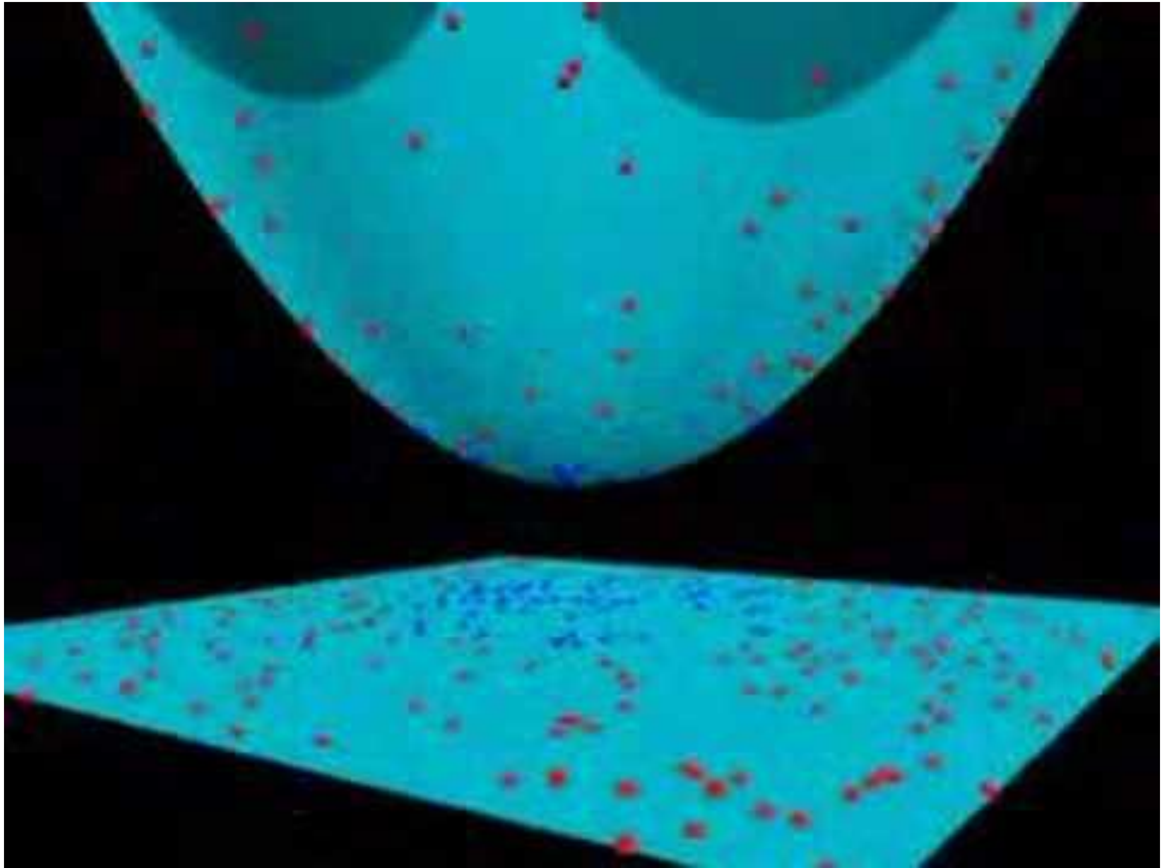
Limitations of Polynomial Features

Adding polynomial features can work great with all sorts of ML algorithms, but it has some limitations:

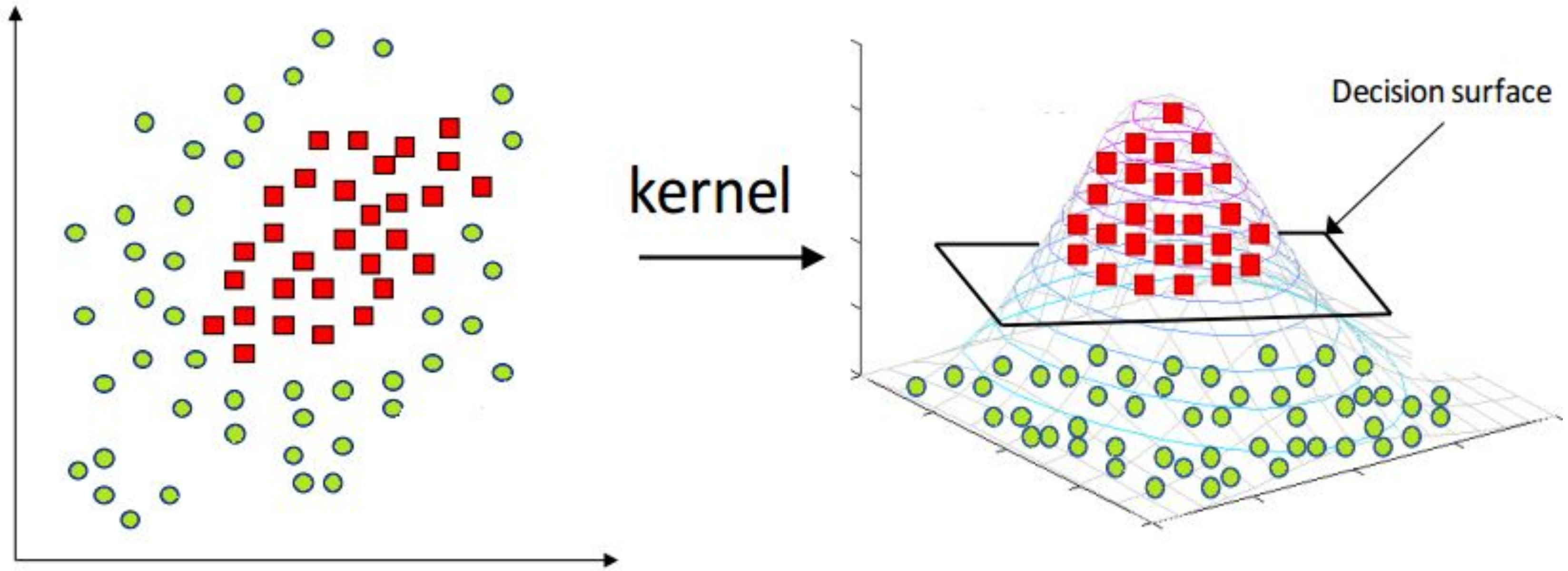
- **Low** polynomial degrees → cannot deal with highly complex datasets
- **High** polynomial degrees → creates huge number of features, slow and possibly overfit

For SVM, we can apply a **kernel trick**: getting the same result as adding many polynomial features *without actually having to add them*.

Transforming into higher dimension



Using a Kernel

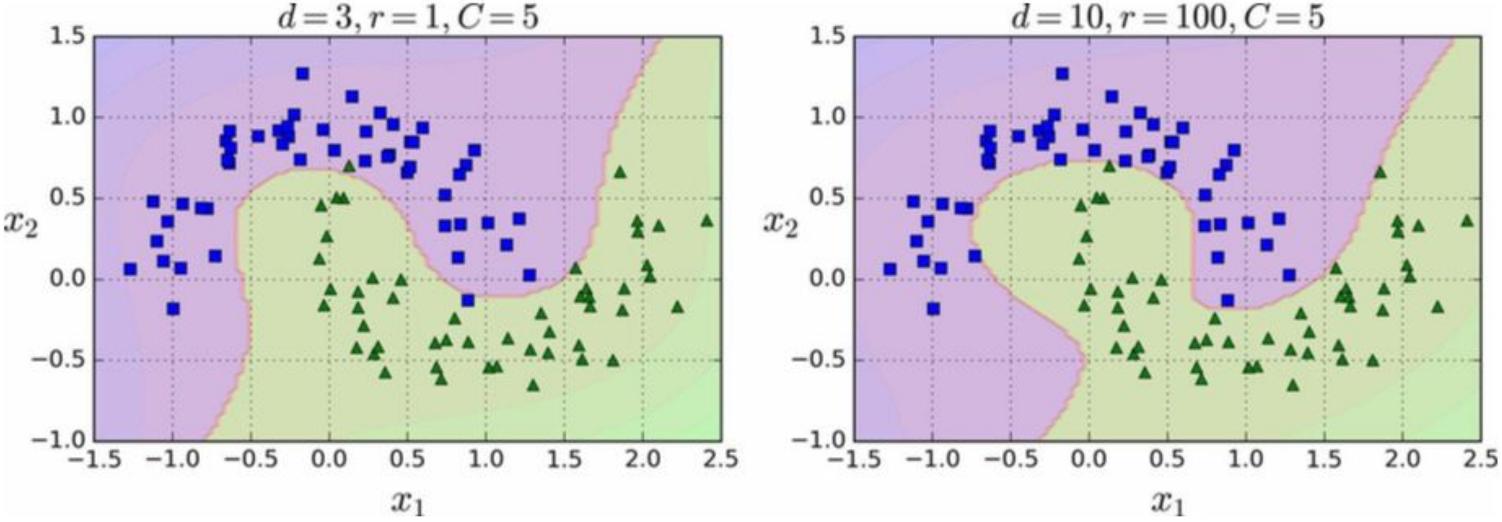


Code

Controls how much the model is influenced by high degree polynomial

```
from sklearn.svm import SVC

poly_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="poly", degree=3, coef0=1, C=5))
    ])
poly_kernel_svm_clf.fit(X, y)
```



Similarity Features

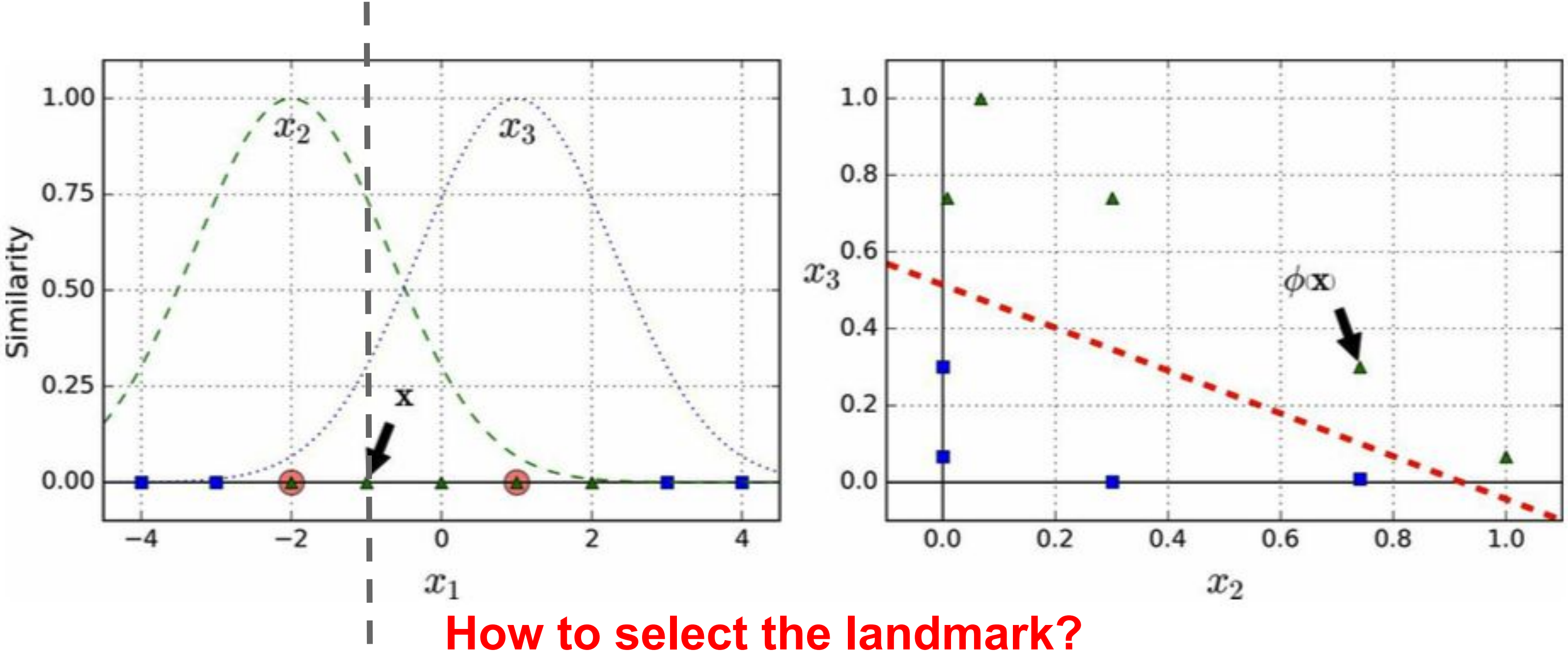
- Besides polynomial transform, another way to handle nonlinear problems is to add features computed using **a similarity function**.
- Similarity function measures how much each training sample resembles a particular **landmark** $\boldsymbol{\ell}$.
- Example: the Gaussian Radial Basis Function (RBF) is a bell-shaped function varying from 0 (far away from the landmark) to 1 (at the landmark)

$$\phi\gamma(\mathbf{x}, \ell) = \exp\left(-\gamma \|\, \mathbf{x} - \ell \,\|^2\right)$$

$$= \exp\left(-\frac{\|\mathbf{x} - l\|^2}{2\sigma^2}\right)$$

Similarity Features using the Gaussian RBF

$$\phi\gamma(\mathbf{x}, \ell) = \exp\left(-\gamma \| \mathbf{x} - \ell \|^2\right) \quad \text{with } \gamma = 0.3$$



How to select the landmark?

Any associated computational problem?

Where do get the landmarks?

Get the locations of all training examples themselves

m training examples → **m** landmarks → higher dimensionality if **m** $>>$ **n**

Disadvantage: training set with **m** examples and **n** features gets transformed into a training set with **m** examples and **m** features (assuming we drop the original features) → can be painfully **slow**!

The Gaussian RBF Kernel

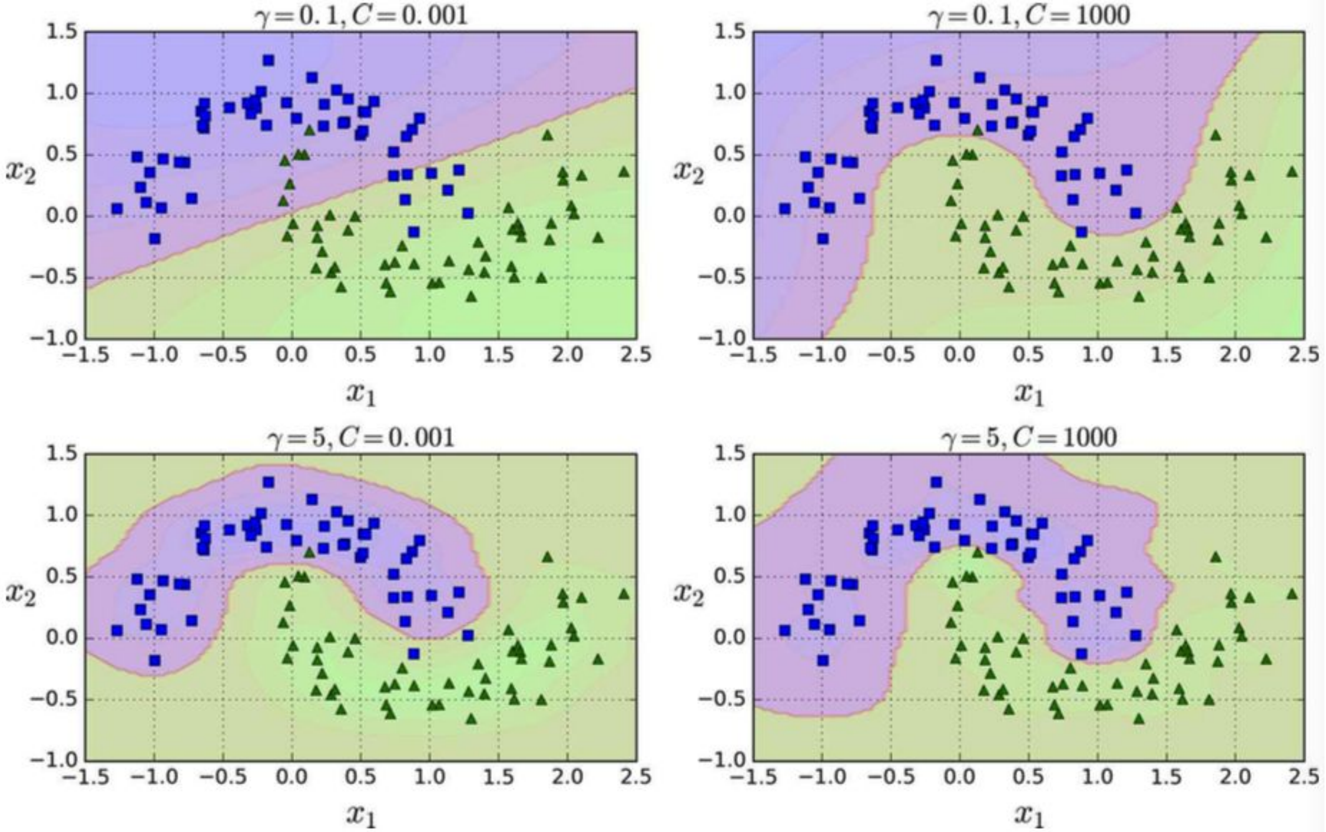
Just like polynomial features, similarity features can be useful, but it computationally expensive to compute all the additional features.

Fortunately, it can be **faster** with the kernel trick...

- **/gamma** (controls the spread of the Gaussian RBF)
- **C** (controls between maximize the margin and minimize the violation)

```
rbf_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="rbf", gamma=5, C=0.001))
    ])
rbf_kernel_svm_clf.fit(X, y)
```

SVM Classifiers using an RBF kernel



2. The Dual Problem

Introducing Lagrange Multipliers

Idea: Making a constrained optimization problem (ie. SVM) into an unconstrained one by subtracting the constraints from the objective function.

$$\min_{x,y} \boxed{x^2 + 2y}$$
$$\text{s.t.} \quad \boxed{3x + 2y + 1} = 0$$

Lagrange Multipliers

$$\mathcal{L}(x, y, \alpha) = \boxed{x^2 + 2y} - \alpha \boxed{(3x + 2y + 1)}$$

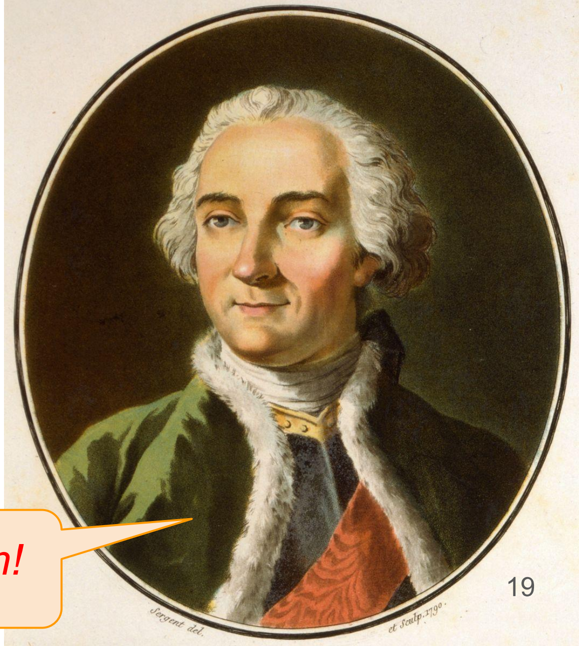
$$\min_{x,y} \max_{\alpha} \mathcal{L}(x, y, \alpha) = x^2 + 2y - \alpha(3x + 2y + 1)$$

Lagrange's Stationary Point

Joseph-Louis Lagrange show that if (x,y) is a solution to the constrained optimization problem, then there must exist an α such that (x,y,α) is a **stationary point** (which has all partial derivatives equal to zero)

$$\mathcal{L}(x,y,\alpha) = x^2 + 2y - \alpha(3x + 2y + 1)$$

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial x} = 2x - 3\alpha \doteq 0 \\ \dfrac{\partial \mathcal{L}}{\partial y} = 2 - 2\alpha \doteq 0 \\ \dfrac{\partial \mathcal{L}}{\partial \alpha} = -3x - 2y - 1 \doteq 0 \end{cases} \qquad \begin{cases} \hat{x} = \dfrac{3}{2} \\ \hat{y} = -\dfrac{11}{4} \\ \hat{\alpha} = 1 \end{cases}$$



This stationary point is also the solution of the original optimization!

KKT Multipliers

Lagrange method applies only to **equality constraints**

Fortunately, under the Karush-Kuhn-Tucker (KKT) multipliers ($\alpha \geq 0$), Lagrange method can be generalized to work for the hard-margin SVM problem:

$$\min_{\mathbf{w},b} \boxed{\frac{1}{2}\mathbf{w}^T\mathbf{w}}$$

$$\text{s.t.} \quad \boxed{y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1} \geq 0 \text{ for } i = 1, \ldots, m$$

$$\min_{\mathbf{w},b} \max_{\alpha \geq 0} \underbrace{\boxed{\frac{1}{2}\mathbf{w}^T\mathbf{w}} - \sum_{i=1}^{m} \alpha^{(i)} \boxed{\left(y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1\right)}}_{\mathcal{L}(\mathbf{w}, b, \alpha)}$$

KKT Conditions

$$\min_{\mathbf{w},b} \max_{\alpha \geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{m} \alpha^{(i)} \left(y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + b) - 1 \right)$$

The **optimized solution** is a stationary point that satisfies the KKT conditions:

$$y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + b) - 1 \geq 0$$

$$\alpha^{(i)} \geq 0 \text{ for } i = 1, \ldots, m$$

$\text{Either } \hat{\alpha}^{(i)} = 0$ or the i^{th} instance is a support vector (lies on the boundary)

Fortunately, the SVM optimization problem **happens to meet** the KKT conditions.

The Dual Problem

Express the SVM objective in a different but related problem called dual problem

PRIMAL: $$\min_{\mathbf{w},b} \max_{\alpha \geq 0} \mathcal{L}(\mathbf{w}, b, \alpha)$$

These two optimization problems are equivalent!

DUAL: $$\max_{\alpha \geq 0} \min_{\mathbf{w},b} \mathcal{L}(\mathbf{w}, b, \alpha)$$

We will focus on the **dual problem** going forward.

Formulate the Dual Problem

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{m} \alpha^{(i)} \left(y^{(i)} \left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) - 1\right)$$

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \mathbf{w}^T\mathbf{x}^{(i)} - b\sum_{i=1}^{m} \alpha^{(i)} y^{(i)} + \sum_{i=1}^{m} \alpha^{(i)}$$

To find $\mathbf{w}$ that minimizes $\mathcal{L}$, take partial derivatives w.r.t. $\mathbf{w}$, and set it to zero

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \doteq 0$$

$$\boxed{\Rightarrow \hat{\mathbf{w}} = \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \qquad (1)}$$

Similarly, take partial derivatives w.r.t. **b**, and set it to zero

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \doteq 0$$

$$\boxed{\Rightarrow \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} = 0 \qquad (2)}$$

$$\hat{\mathbf{w}} = \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \qquad (1)$$

$$\sum_{i=1}^{m} \alpha^{(i)} y^{(i)} = 0 \qquad (2)$$

Formulate the Dual Problem

$$\max_{\alpha \geq 0} \min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)} - b\sum_{i=1}^{m} \alpha^{(i)} y^{(i)} + \sum_{i=1}^{m} \alpha^{(i)}$$

Plug in (1) and (2)

$$\Rightarrow \max_{\alpha \geq 0} \frac{1}{2}\left(\sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}\right)^T \left(\sum_{j=1}^{m} \alpha^{(j)} y^{(j)} \mathbf{x}^{(j)}\right) - \left(\sum_{j=1}^{m} \alpha^{(j)} y^{(j)} \mathbf{x}^{(j)}\right)^T \left(\sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}\right) - b(0) + \sum_{i=1}^{m} \alpha^{(i)}$$

$$\Rightarrow \max_{\alpha \geq 0} -\frac{1}{2}\left(\sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}\right)^T \left(\sum_{j=1}^{m} \alpha^{(j)} y^{(j)} \mathbf{x}^{(j)}\right) + \sum_{i=1}^{m} \alpha^{(i)}$$

$$\Rightarrow \max_{\alpha \geq 0} -\frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} {\mathbf{x}^{(i)}}^T \mathbf{x}^{(j)} + \sum_{i=1}^{m} \alpha^{(i)}$$

$$\Rightarrow \max_{\alpha \geq 0} \boxed{\sum_{i=1}^{m}} \alpha^{(i)} - \frac{1}{2} \boxed{\sum_{i=1}^{m} \sum_{j=1}^{m}} \boxed{\alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)}} \boxed{{\mathbf{x}^{(i)}}^T \mathbf{x}^{(j)}}$$

Sum over all examples

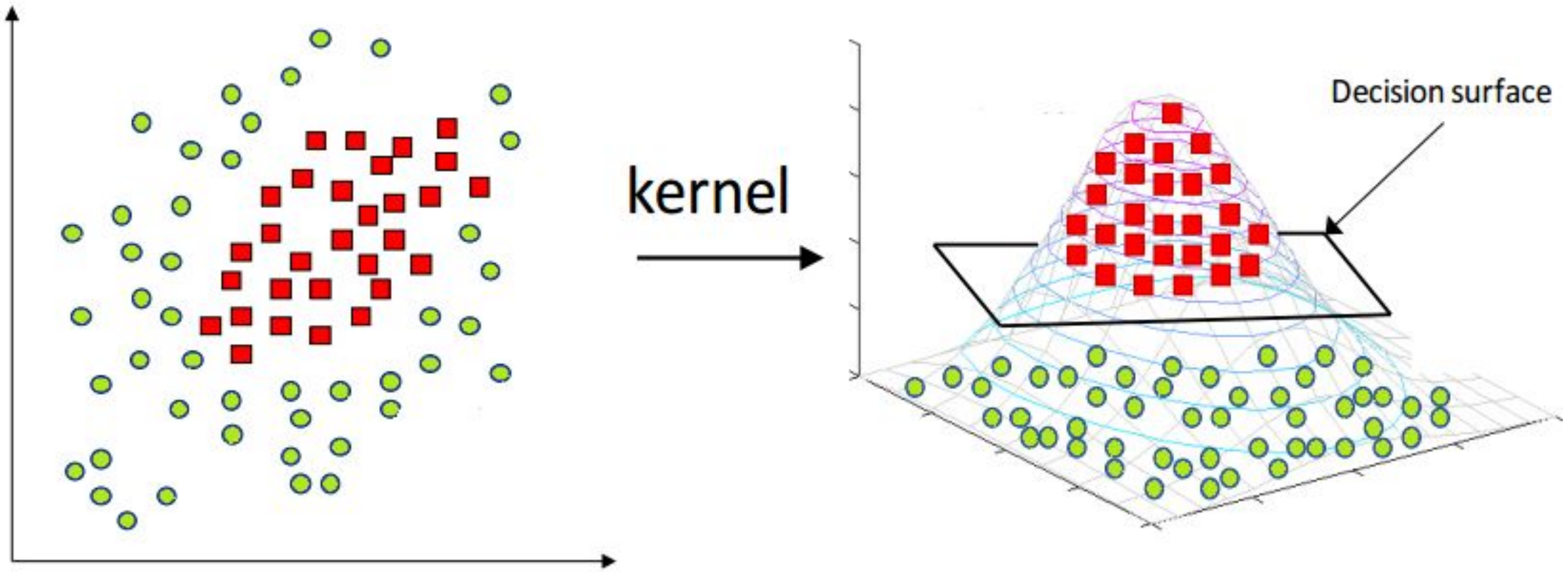
Scalars

Dot product

Note: This dual formulation only depends on **dot-products**
→ This is what makes the **kernel trick** possible!

3. The Kernel Trick

Review: Transform using a Kernel



A Kernel

Suppose we want to transform the training set into another feature space, we can use **a kernel** (mapping function):

$$\phi(\mathbf{x}) = \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1^2 \\ x_1 x_2 \\ x_2^2 \end{bmatrix}$$

Notice that we now have **three** new features (from the **two** original ones). In another word, we mapped this training examples from 2D into 3D.

What if we compute the dot product of the transformed vectors?

Dot product of Polynomial Transformation

d=1

$$\phi(u).\phi(v) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1v_1 + u_2v_2 = u.v$$

d=2

$$\phi(u).\phi(v) = \begin{pmatrix} u_1^2 \\ u_1u_2 \\ u_2u_1 \\ u_2^2 \end{pmatrix} \cdot \begin{pmatrix} v_1^2 \\ v_1v_2 \\ v_2v_1 \\ v_2^2 \end{pmatrix} \begin{aligned} &= u_1^2v_1^2 + 2u_1v_1u_2v_2 + u_2^2v_2^2 \\ &= (u_1v_1 + u_2v_2)^2 \\ &= (u.v)^2 \end{aligned}$$

For any *d (we will skip proof):*

$$\phi(u).\phi(v) = (u.v)^d$$

So, to transform data into high-D space and then take dot product **is the same as** to take the dot product of data and then exponent. Which is faster?

Polynomial Kernel: $\boxed{\phi(u).\phi(v) = (u.v)^d}$

How the kernel trick work

- If we apply the transformation to all training samples, the dual problem will contain **a dot product.**

$$\max_{\alpha \geq 0} \sum_{i=1}^{m} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \boxed{\phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})}$$

$$\mathbf{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

$$\max_{\alpha \geq 0} \sum_{i=1}^{m} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \boxed{(\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})^d}$$

- **So, we do NOT need to transform the samples at all!** This trick makes the process much more efficient (**O(n)** for high-dim dot product)

Common Kernels

- Polynomials of degree exactly *d*

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v})^d$$

- Polynomials of degree up to *d*

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v}+1)^d$$

- Gaussian kernels

$$K(\vec{u},\vec{v}) = \exp\left(-\frac{||\vec{u}-\vec{v}||_2^2}{2\sigma^2}\right)$$

- Sigmoid

$$K(\mathbf{u},\mathbf{v}) = \tanh(\eta\mathbf{u}\cdot\mathbf{v}+\nu)$$

- And many others: very active area of research!

SVM Modification Due to Kernel Trick

Without kernel (Linear Classification):

$$\max_{\alpha} \sum_{i=1}^{m} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)T} \mathbf{x}^{(j)}$$

$$\sum_i \alpha^{(i)} y^{(i)} = 0$$

$$\alpha^{(i)} \geq 0$$

With kernel (Nonlinear Classification)

$$\max_{\alpha} \sum_{i=1}^{m} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})$$

$$\sum_i \alpha^{(i)} y^{(i)} = 0$$

$$\alpha^{(i)} \geq 0$$

Solving for w and b

$$\max_{\alpha} \sum_{i=1}^{m} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} \mathbf{x}^{(i)}{}^{T} \mathbf{x}^{(j)}$$

$$\sum_i \alpha^{(i)} y^{(i)} = 0$$

$$\alpha^{(i)} \geq 0$$

Find α that maximizes this equation using a **QP solver**, then compute **w** and b:

$$\boxed{\hat{\mathbf{w}} = \sum_{i=1}^{m} \hat{\alpha}^{(i)} y^{(i)} \mathbf{x}^{(i)}}$$

from (1)

$$y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1 \text{ for } i = 1, \ldots, m$$

For a support vector k: $\hat{\alpha}^{(k)} > 0$

same +1, -1

$$y^{(k)} (\hat{\mathbf{w}}^T \mathbf{x}^{(k)} + b) = 1 \quad \Rightarrow \hat{\mathbf{w}}^T \mathbf{x}^{(k)} + b = y^{(k)}$$

$$\Rightarrow \hat{b} = y^{(k)} - \hat{\mathbf{w}}^T \mathbf{x}^{(k)}$$

For stable value, compute the mean: $\boxed{\hat{b} = \frac{1}{n_s} \sum_{k=1, \alpha^{(k)}>0}^{n_s} y^{(k)} - \hat{\mathbf{w}}^T \mathbf{x}^{(k)}}$

Predicting the labels

Without Kernel (same as Linear Classification)

$$\hat{y}^{(\text{test})} = \hat{\mathbf{w}}^T \mathbf{x}^{(\text{test})} + \hat{b}$$

$$= \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)T} \mathbf{x}^{(\text{test})} + \hat{b}$$

$$\hat{\alpha}^{(i)} > 0$$

$$\hat{\mathbf{w}} = \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}$$

$$\hat{b} = y^{(i)} - \hat{\mathbf{w}}^T \mathbf{x}^{(i)}$$

With Kernel (Nonlinear Classification)

$$\hat{y}^{(\text{test})} = \hat{\mathbf{w}}^T \phi(\mathbf{x}^{(\text{test})}) + \hat{b}$$

$$= \left(\sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \phi(\mathbf{x}^{(i)}) \right)^T \phi(\mathbf{x}^{(\text{test})}) + \hat{b}$$

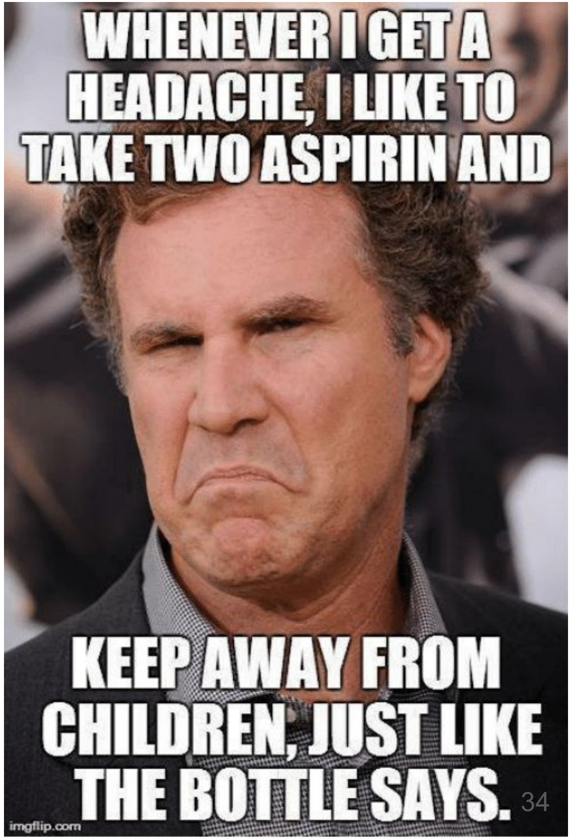
$$= \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \left(\phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(\text{test})}) \right) + \hat{b}$$

$$= \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x}^{(\text{test})}) + \hat{b}$$

$$\hat{\alpha}^{(i)} > 0$$

$$\hat{\mathbf{w}} = \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} \phi(\mathbf{x}^{(i)})$$

$$\hat{b} = y^{(i)} - \hat{\mathbf{w}}^T \phi(\mathbf{x}^{(i)})$$



Computational Complexity of SVM

Class	Time complexity	Out-of-core support	Scaling required	Kernel trick
`LinearSVC`	$O(m \times n)$	No	Yes	No
`SGDClassifier`	$O(m \times n)$	Yes	Yes	No
`SVC`	$O(m^2 \times n)$ to $O(m^3 \times n)$	No	Yes	Yes

For large-scale and non-linear problems, we might also consider neural network instead (coming later this semester)

Today: Learning Objectives

✓ Expand to non-linear case
✓ Formulate the dual problem
✓ Understand the kernel trick

Bonus content

Mercer Kernel vs. Smoothing Kernel

The kernels used in SVM are different from the ones used in Locally Weighted / Kernel Regression.

The kernels in SVM must satisfy a few mathematical conditions called **Mercer's condition**:

1. It is symmetric $K(a,b) = K(b,a)$
2. There exists a function ϕ to map a, b into another space: $K(a,b) = \phi(a) \cdot \phi(b)$

We do NOT need to transform the samples at all! The kernel trick makes the process much more efficient (**O(n)** for high-dim dot product)

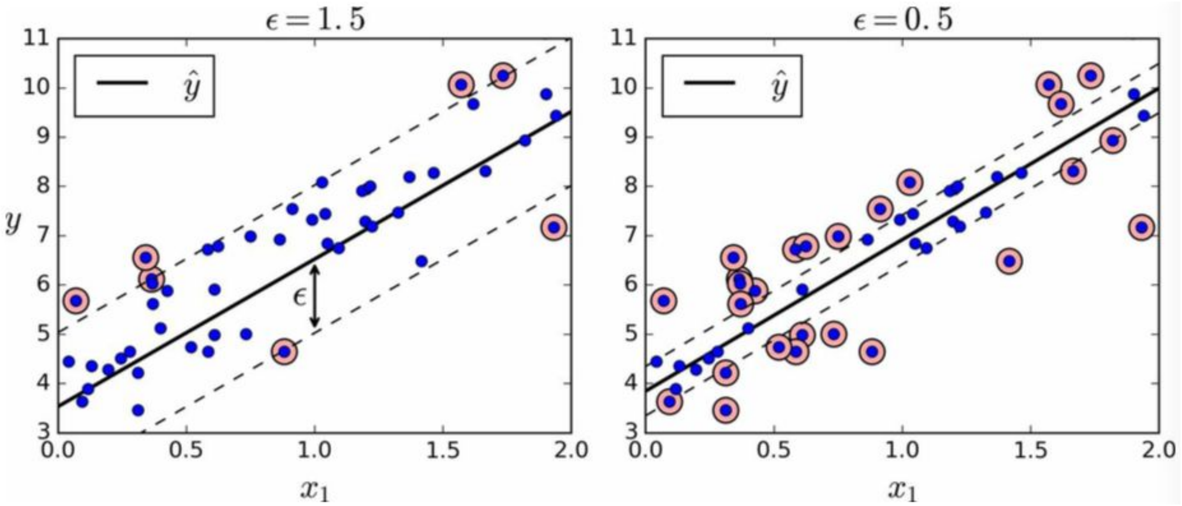
SVM Regression

- SVM also supports linear and non-linear regression
- The trick is to **reverse** the objective: SVM Regression tries to fit as many samples as possible on the street while limiting margin violations (samples that are **off** the street)
- Width of the street is controlled by hyperparameter /epsilon

SVR Code

```
from sklearn.svm import LinearSVR

svm_reg = LinearSVR(epsilon=1.5, random_state=42)
svm_reg.fit(X, y)
```



Kernelized SVR for non-linear case

```
from sklearn.svm import SVR

svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svm_poly_reg.fit(X, y)
```