MACH

Support Vector Machines

Support Vector Machines (SVM)

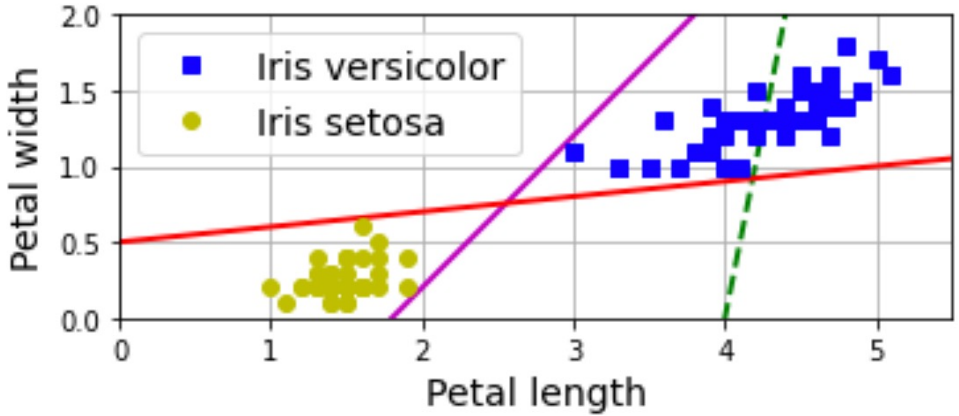
- A **powerful and versatile** machine learning model capable of performing:
  - Linear and non-linear classification
  - Regression
  - Novelty detection
- SVMs shine with **small to medium-sized nonlinear** datasets (i.e., hundreds to thousands of instances), especially for classification tasks.
- **Don’t scale very well to very large datasets**
- In this lesson:
  - The core concepts of SVMs
  - How to use them,
  - How they work.

Linear SVM Classification

When the two classes can be separated easily with a straight line

The fundamental idea

- The two classes are **linearly separable**.
- The decision boundaries of three possible models:
- Dash green line: so bad !
- Solid lines: **work perfectly on the training set**
- But decision boundaries come so close to the instances that these models **will probably not perform as well on new instances.**

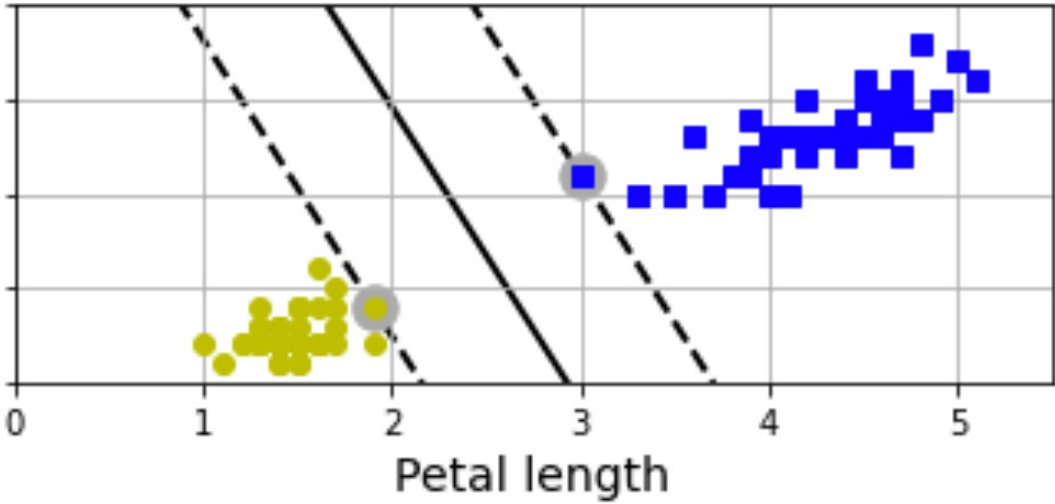


An idea about the separation line ?

EPITA

Vast Margins

- Solid line ⇔ the decision boundary of an SVM classifier
- Not only separates the two classes but
  - Also **stay as far away from the closest training instances as possible**.
- SVM classifier as fitting the widest possible band between the classes.
- ➔ **Large margin classification**.

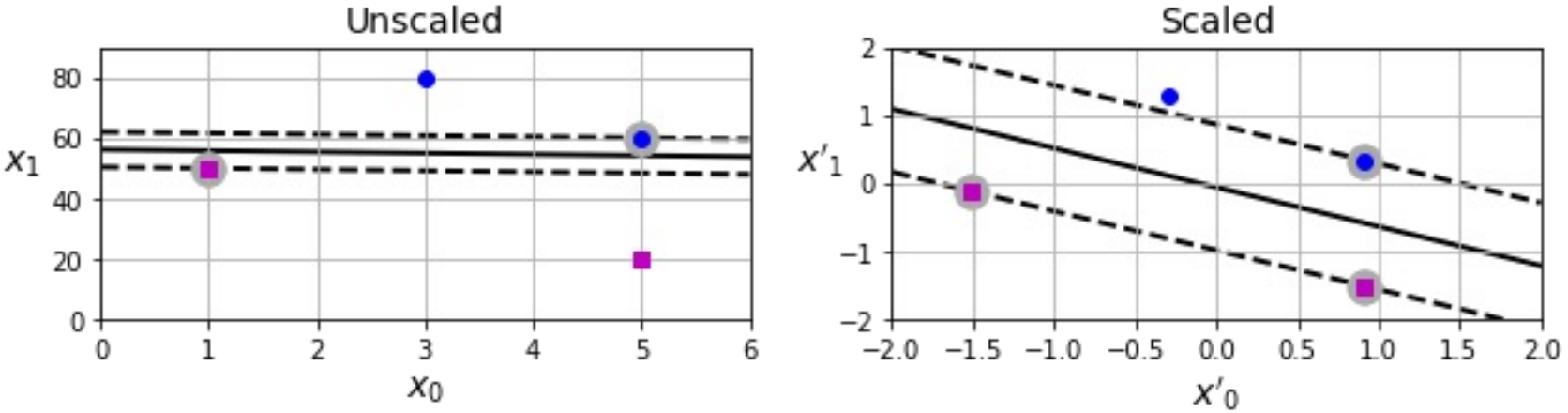


- Notice that adding more training instances “off de the street” will not affect the decision boundary at all.
- It is **fully determined (or « supported ») by the instances located on the edge** of the street.

➔ These instances are called **the support vectors** (circled)

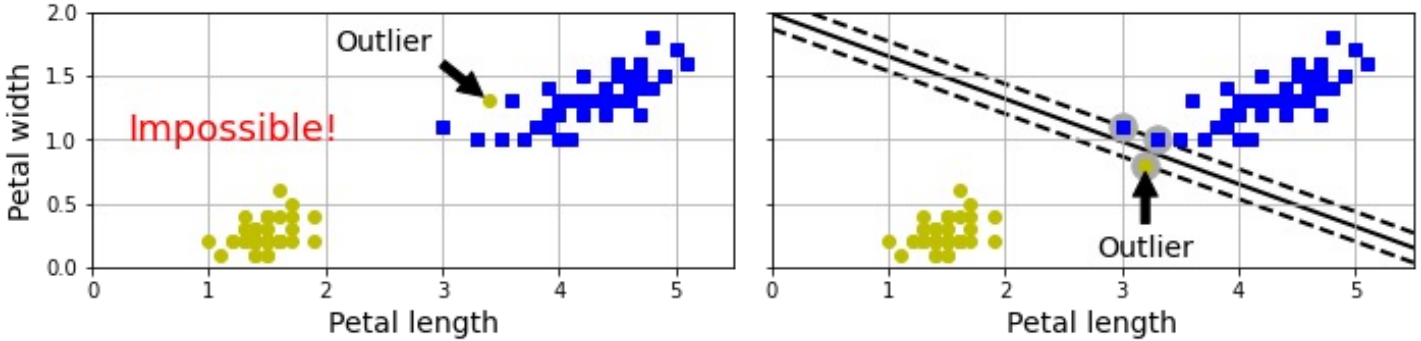
EPITA

SVM are sensitive to the feature scales



- Largest street closed to the horizontal on non-scaled data.
- After feature scaling: decision boundary looks much better.

Hard Margin sensitivity to outliers



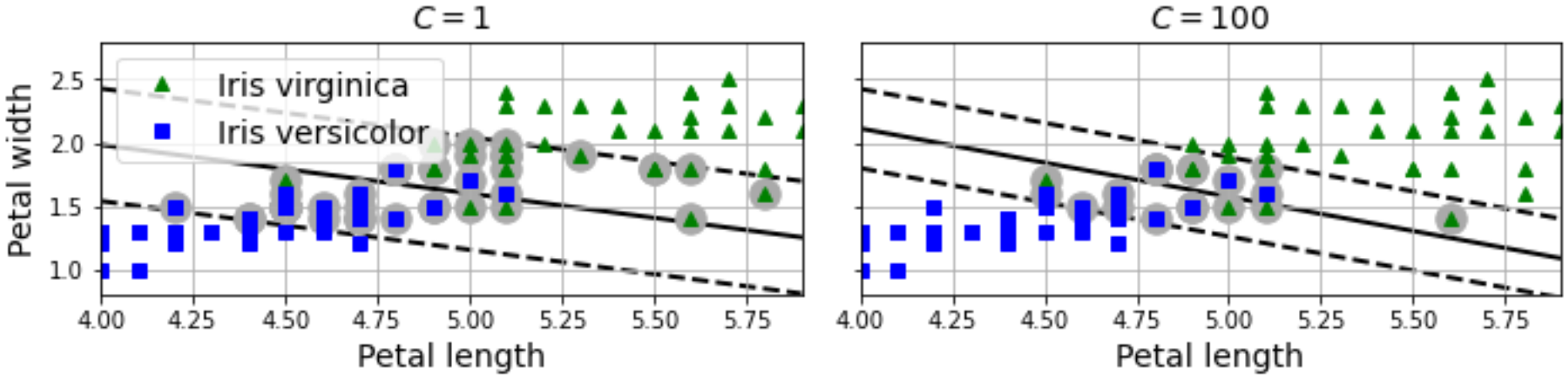
- Hard margin classification: **all instances must be off the street**
  - **Only work with linearly separable data**
  - **Sensitive to outliers ➔** that makes not possible good generalization.

➔ Need of more flexible models

- Good balance between **keeping the street as largest as possible** and **limiting violations** (instances in the middle or event on the wrong side)

➔ **Soft margin classification**

Regularization hyperparameter C



- Reducing **C** makes the street larger but leads also to more margin violations
- Reducing **C** ⇔ less risk of overfitting
- Reducing **C** to much: risk of underfitting

EPITA

Scikit-Learn code

- Unlike Logistic Regression, LinearSVC doesn't have a **predict_proba()** method to estimate the class probability.
- SVC instead with **probability** hyperparameter set to **True** ➔ the model will fit an extra-model to map the SVM decision function scores to estimated probabilities.
- **Requires 5-fold CV**
- It will **slow down training** considerably
- After that: **predict_proba()** and **predict_log_proba()** will be available.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

iris = load_iris(as_frame=True)
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = (iris.target == 2)  # Iris virginica

svm_clf = make_pipeline(StandardScaler(),
                        LinearSVC(C=1, random_state=42))
svm_clf.fit(X, y)
```

```
1 X_new = [[5.5, 1.7], [5.0, 1.5]]
2 svm_clf.predict(X_new)
```

```
array([ True, False])
```

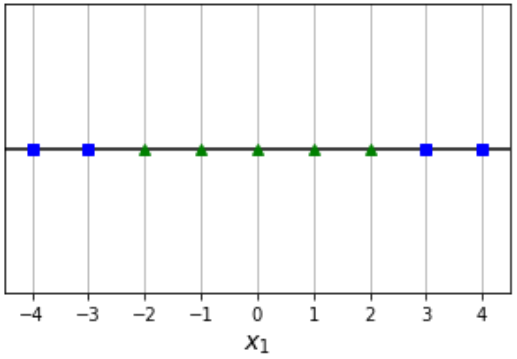
```
1 svm_clf.decision_function(X_new)
```

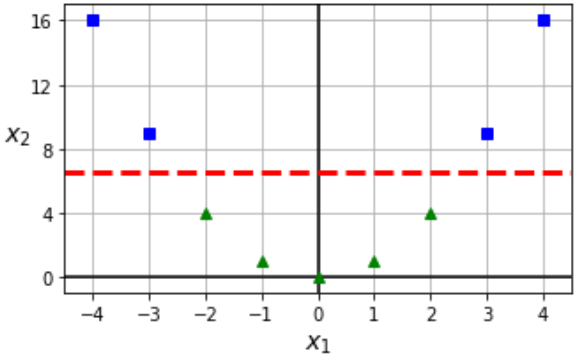
```
array([ 0.66163411, -0.22036063])
```

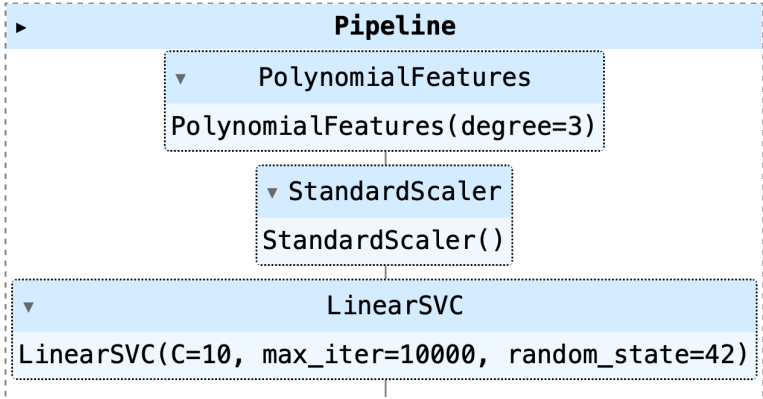
Non Linear SVM Classification

Although linearSVM are efficient and often work surprisingly well, many datasets are not even close to being linearly separable.

Adding polynomial features to handle nonlinear datasets

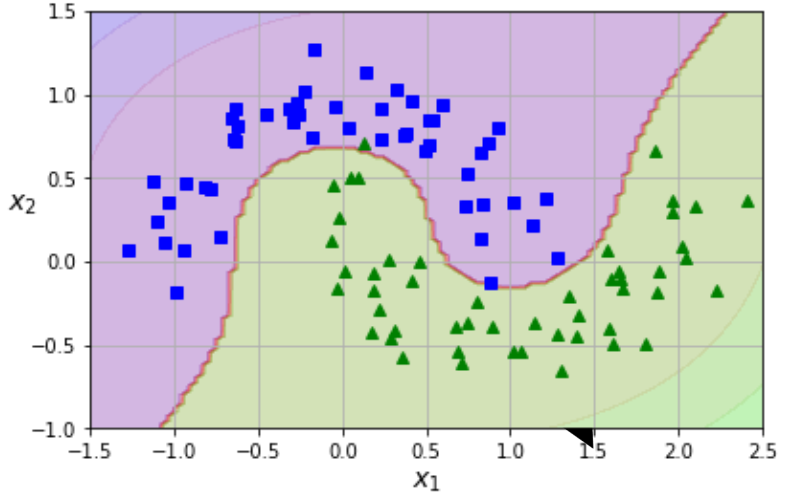






- **Simple** to implement but
- A **low polynomial** degree **cannot deal with very complex datasets**
- A **high polynomial degree** leads to a **huge number of features** making the model **too slow**.

➔ Polynomial kernel

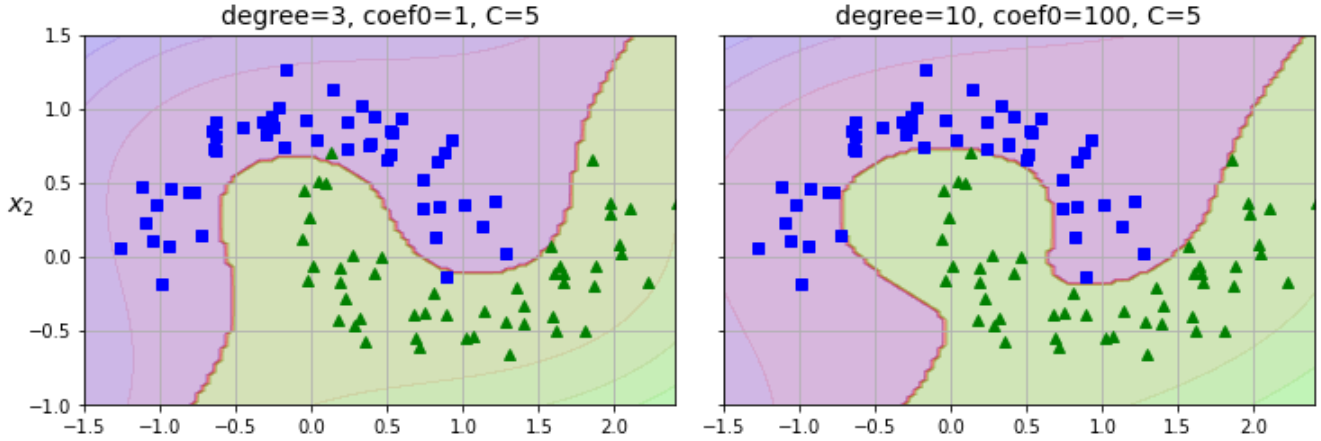


Polynomial Kernel

- When using SVMs, you can apply an almost miraculous mathematical technique called **the kernel trick** (explained later)
  - Makes it possible to get the same result as if you had added many polynomial features, even with a very high degree, without actually having to add them.
  - ➔ There is not combinatorial explosion of the number of features.
  - Trick implemented by the SVC class

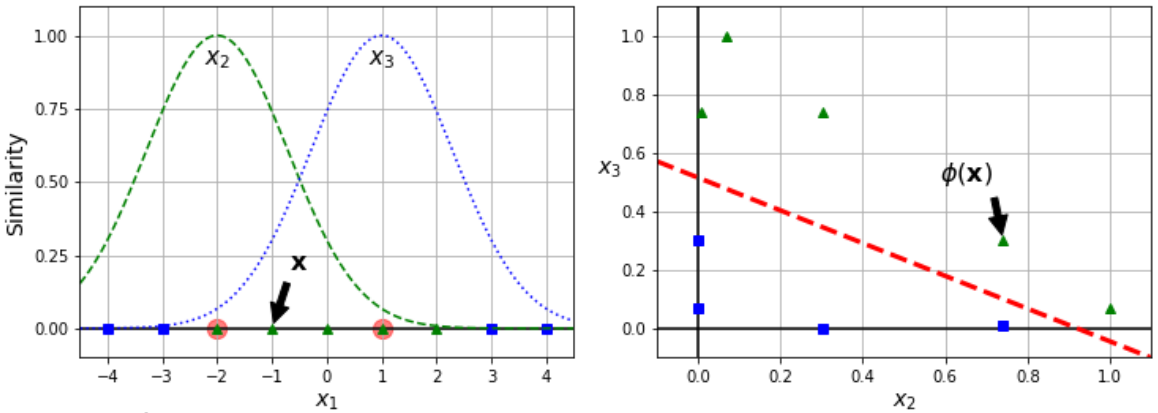
```
from sklearn.svm import SVC
poly_kernel_svm_clf = make_pipeline(StandardScaler(),SVC(kernel="poly", degree=3, coef0=1, C=5))
poly_kernel_svm_clf.fit(X, y)
```

Hyperparameter **coef0** controls how much the model is influenced by high-degree terms versus low-degree terms.



Similarity features

- **Add features** computed using a **similarity function**, which *measures how much each instances ressembles* a **particular landmark**.
- **Gaussian RBF** similarity function with $\gamma = 0.3$
- X=-1 ➔ $x_2 = \exp(-0.3 \times 1^2) \approx 0.74$; $x_3 = \exp(-0.3 \times 2^2) \approx 0.30$



How to select landmarks?

➔ Simplest approach is to create a landmark at the location of each instance

➔ m features

➔ m*m matrix (with all the problems it may raise). ➔ Similar result obtained with kernel tricks

https://en.wikipedia.org/wiki/Radial_basis_function_kernel

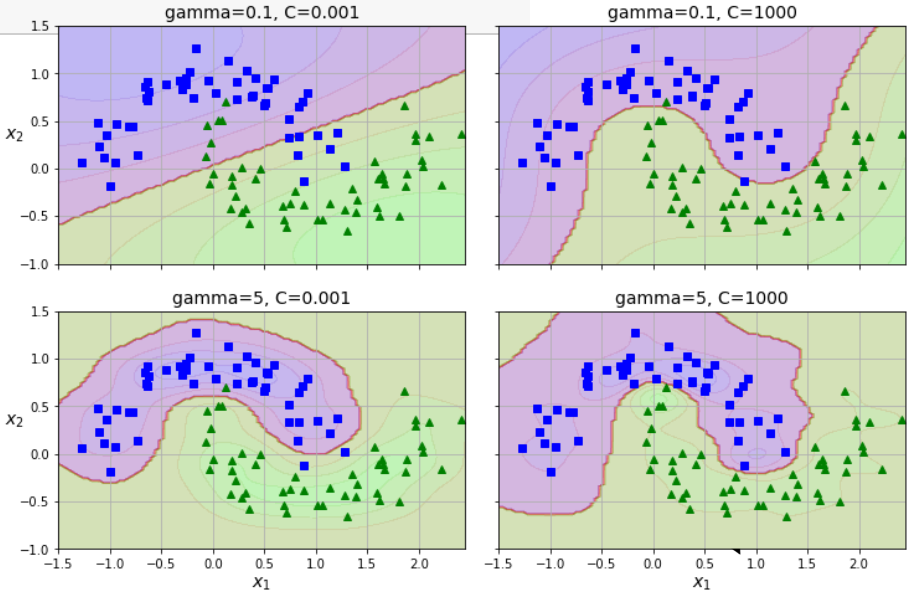
Gaussian RBF Kernel

- Kernel trick makes it possible ti obtain similar results as the similarity features without actually computing them.
- SVC Class with Gaussian RBF Kernel

```
rbf_kernel_svm_clf = make_pipeline(StandardScaler(),
                                   SVC(kernel="rbf", gamma=5, C=0.001))
rbf_kernel_svm_clf.fit(X, y)
```

Different values of hyperparameter **gamma** and **C**:

- **Increasing** gamma makes **the bell-shaped curve narrower** ➔ each instance's range of influence is **smaller**: decision boundary ends up being more **irregular**.
- Small gamma ➔ **bell-shapped curve wider** ➔ instances have a **larger range** of influence ➔ decision boundary **smoother**
- Gamma acts as a regularization hyperparameter: if the model overfits reduce gamma (similar to **C**)



Other kernels

- Used much more rarely:
  - Kernels specialized for specific data structures
    - String kernels for documents or DNA sequences (string subsequence kernel or kernel based on Levenshtein distance).
- With so many kernels to choose, how can you decide which one to use?
  - Try linear kernel first (LinearSVC is much faster than SVC(kernel= ''linear'' especially if the training set is very large)
  - If training set not too large, also try kernelized SVMs
    - Starting with GaussianRBF kernel which often works really well.
    - Then if you have spare time and computing power:
      - Experiment with a few other kernel using hyperparameter search
      - If there are kernels specialized for your training set data structure, make sure to give them a try too.

SVM Classes and Computational Complexity

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

- LinearSVC may take longer if you require very high precision (***ε***, hyperparameter **tol**)
- SVC is best for small or mediym size dataset
  - Scales well with the number of features

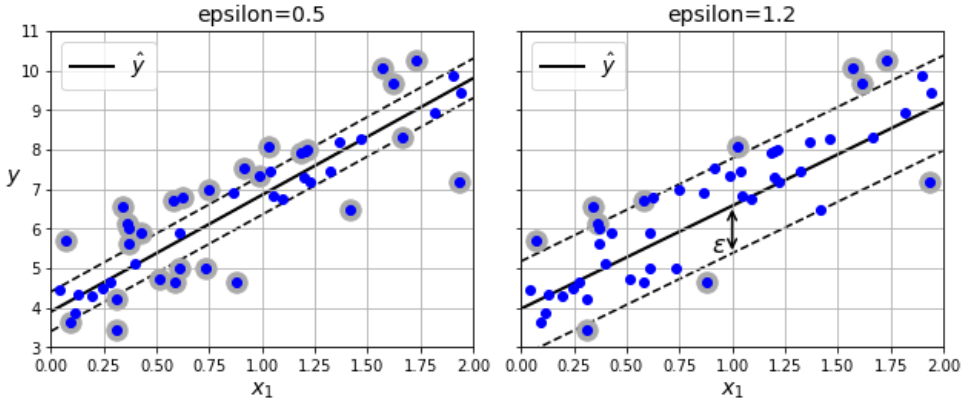
SVM Regression

Let's see how the SVM algorithms can also be used for linear and non linear regression ...

An idea?

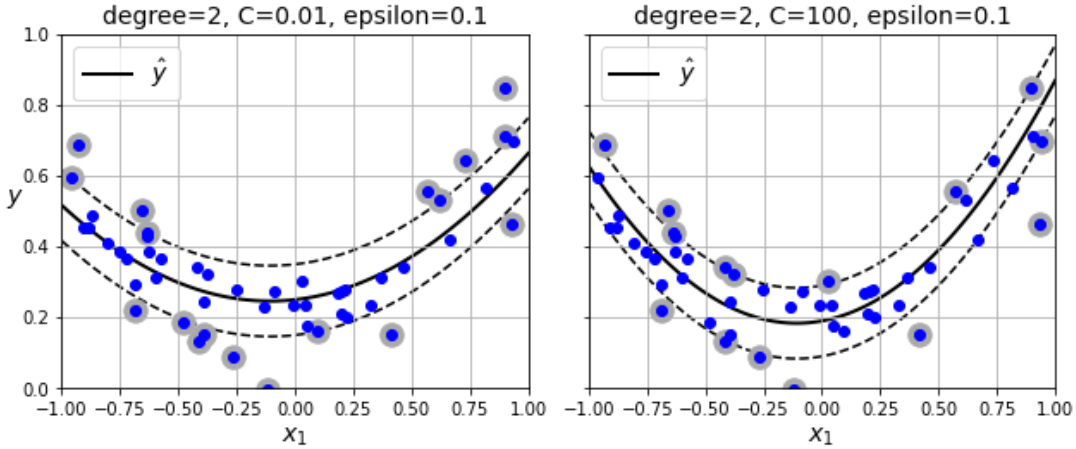
SVM Regression

- To use SVMs for regression instead of classification: tweak the objective.
  - Instead of trying to find the « *largest possible street* » between two classes while limiting margin violations, **SVM regression tries to fit as many instances as possible on the street while limiting margin violations (instance off the street)**.
- The width of the street is controlled by a hyperparameter ε.
  - **Reducing ε increases** the **number** of **support vectors**, which **regularize** the model.
  - If more training instances within the margin, it will not affect the model's predictions ➔model ε-insensitive.




```
svm_reg = make_pipeline(StandardScaler(),
                        LinearSVR(epsilon=0.5, random_state=42))
svm_reg.fit(X, y)
```

Kernelized SVM model for nonlinear regression tasks




```
svm_poly_reg = make_pipeline(StandardScaler(),
                             SVR(kernel="poly", degree=2, C=0.01, epsilon=0.1))
svm_poly_reg.fit(X, y)
```

- SVR Class is the regression equivalent to SVC class.
- LinearSVR class scales linearly with the size of the training set, while the SVR class gets much too slow when the training set grows very large (just like the SVC class)

EPITA

Under the hood of Linear SVM Classifiers

Explaining how SVMs make predictions and how their training algorithms work.

Linear SVM prediction

- A linear SVM Classifier predicts the class of a new instance **x** by first computing the **decision function** $\boldsymbol{\theta}^\top \boldsymbol{x} = \theta_0 x_0 + \cdots + \theta_n x_n$, where x_0 is the bias feature (always equal to 1).
  - If the result is **positive**, then the predicted class $\hat{y}$ is the **positive class** (1à
  - Otherwise, it is the **negative class** (0)

Convention

- Up to now: all the model parameters in one vector $\boldsymbol{\theta}$ including the bias term θ_0 and the input feature weights θ_1 to θ_n
  - This requires adding a bias input $x_0 = 1$ to all instances.
- Another very common convention: separate the bias term b (equal to θ_0) and the feature weights vector **w** (containing θ_1 to θ_n)
  - In this case, no bias feature needs to be added to the input feature vectors
  - The linear SVM's decision function is equal to $\boldsymbol{w}^\top \boldsymbol{x} + b = w_1\, x_1 + \cdots + w_n x_n + b$

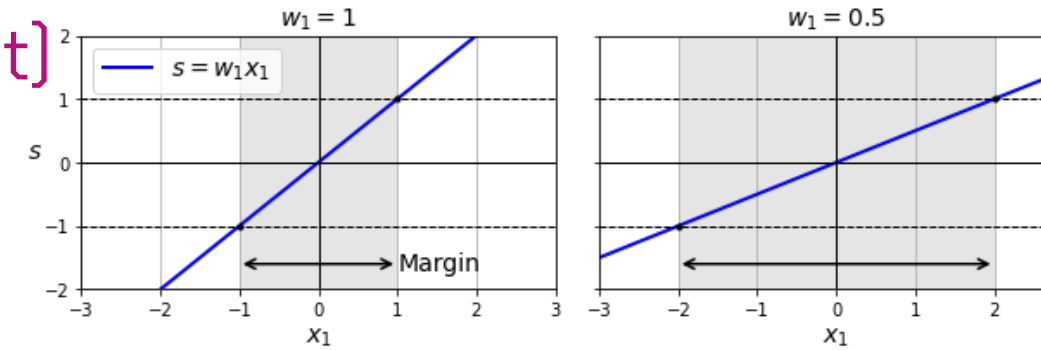
➔ We will use this notation!

Making predictions with linear SVM is quite straightforward. How about training?

How about training?

- This requires finding the weights vector **w** and the bias term *b* that make the margin as wide as possible while limiting the number of margin violations.

Width of the margin (street)



- To make the margin larger, make **w** smaller.
- Let's define the borders of the street as the points where the decision function is equal to-1 or +1.
  - (left) $w_1 = 1$ ➔ margin's size is 2.
  - (right) $w_1 = 0.5$ ➔ margin's size is 4

➔ We need to keep **w** as smallest as possible.

- Bias term b has no influence on the size of the margin.
  - Tweaking it just shifts the margin around, without affecting its size.

Avoid margin violations: Hard Margin linear SVM objective

- We need the decision function to be:
  - Greater than 1 for positive training instances
  - Lower than -1 for negative training instances
- If we define $t^{(i)} = -1$ when $y^{(i)} = 0$ **and** $t^{(i)} = 1$ when $y^{(i)} = 1$ **then**
  - We can write this constraint as $t^{(i)}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b\right) \geq 1$ for all instances
- We can therefore express the hard margin linear SVM classifier objective as the constrained optimization problem:
- $\text{Minimize}_{\boldsymbol{w},b} \quad \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w}$ subject to $t^{(i)}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b\right) \geq 1$ for all instances
  - We are minimizing $\frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w}$ which is equal to ½ $||\mathbf{w}||^2$ rather than minimizing $||\mathbf{w}||$ (the norm of **w**).
  - ½ $||\mathbf{w}||^2$ has a nice simple derivative while $||\mathbf{w}||$ is not diffentiable at **w**=0
  - Optimization algorithms work much better on differentiable functions.

EPITA

Soft margin objective

- We need to introduce a slack variable (zeta) $\zeta^{(i)} \geq 0$ for each instance.
- $\zeta^{(i)}$ measures how much the ith instance is allowed to violate the margin.
- We now have two conflicting objectives:
  - Make the slack variables as small as possible to reduce the margin violations.
  - Make $\frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w}$ as small as possible to increase the margin.
- This is where the **C** hyperparameter comes in:
  - Allow to define the trade-off between these two objectives.

➔ Constrained optimization problem:

$$\text{Minimize}_{\boldsymbol{w},b,\zeta} \ \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} + C\sum_{i=1}^{m}\zeta^{(i)} \text{ subject to } t^{(i)}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \text{ and } \zeta^{(i)} \geq 0 \text{ for i=1,2, ..., m}$$

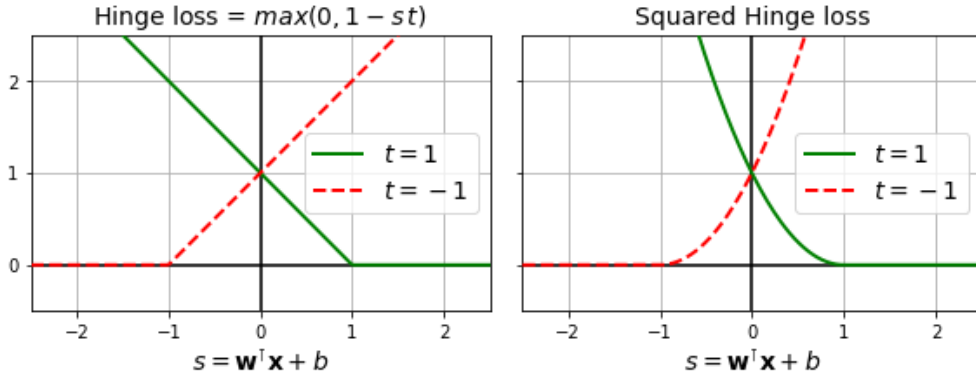
Solving the pbs

- Hard margin and soft margin problems are **both convex quadratic optimization problems with linear constraints**.
  ➔ known as **Quadratic Programming (QP) problems**.
- Off-the-shelf solvers are available to solve QP problems
- QP Solvers is one way to train an SVM
- Another: use a gradient descent to minimize the *hinge loss* or the *squared hinge loss*.

https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

Gradient Descent

- Given an instance x of the positive class (with t=1), the loss is 0 if the output s of the decision function ($s = \boldsymbol{w}^\top \boldsymbol{x} + b$) is greater than equal to 1.
  - **This happens when the instance is off the street and on the positive side.**
- Given an instance of the negative class (with t=-1), the loss is 0 $s \leq -1$
  - **This happens when the instance is off the street and on the negative side.**



The further away an instance is from the correct side of the margin, the higher the loss.

➔ It grows linearly for Hinge loss and quadratically for Squared Hinge loss.

➔ Squared Hinge Loss more sensitive to outlier but if the dataset is cleaned it tends to converge faster.

By default, LinearSVC uses the squared Hinge loss while SGDClassifier uses the Hinge loss.

EPITA

The Dual Problem

- Given a constrained optimization problem, known as the **primal problem,** it is possible to express a different but closely related problem, called its **dual problem**.
- The solution to the **dual problem gives a lower bound** to the solution of the primal problem, but **under some conditions it can have the same solution as the primal problem**.
- SVM problem happens to meet these conditions (objective function is convex and the inequality constraints are continuously differentiable and convex functions)
  - ➔ Can choose to solve the primal or the dual problem: both have the same solution.
- The dual form of the linear SVM objective:

$$\text{Minimize}_{\boldsymbol{\alpha}} \ \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} t^{(i)} t^{(j)} \boldsymbol{x}^{(i)\top} \boldsymbol{x}^{(j)} \ - \sum_{i=1}^{m} \alpha^{(i)}$$

$$\text{subject to } \alpha^{(i)} \geq 0 \text{ for all i} = 1,2, \ldots, m \text{ and } \sum_{i=1}^{m} \alpha^{(i)} t^{(i)} = 0$$

EPITA

- Once one finds the vector $\hat{\boldsymbol{\alpha}}$ that minimizes the equation (using a QP solver), one can compute $\hat{\boldsymbol{w}}$ and $\hat{b}$ that minimize the primal problem using the following equation:
  - $\hat{\boldsymbol{w}} = \sum_{i=1}^{m} \widehat{\alpha^{(i)}} t^{(i)} \boldsymbol{x}^{(i)}$
  - $\hat{b} = \frac{1}{n_s} \sum_{i=1}^{m} \left(t^{(i)} - \hat{\boldsymbol{w}}^{\top} \boldsymbol{x}^{(i)} \right)$
  - $\hat{\alpha}^{(i)} > 0$;
  - n_s is the number of support vectors.
- The **dual problem is faster to solve than the primal one when the number of training instances is smaller than the number of features**.
- More importantly, the dual problem makes the kernel tricks possible while the primal problem does not.

Kernelized SVMs

- Suppose you want to apply a second-degree polynomial transformation to a two dimensional training set then train a linear SVM classifier on the transformed training set.
- The polynomial mapping function to be applied is:
- Notice that we have now a 3D vector instead of 2D.
- Dot product between two transformed vectors:

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

$$\phi(\mathbf{a})^\top \phi(\mathbf{b}) = \begin{pmatrix} {a_1}^2 \\ \sqrt{2}\, a_1 a_2 \\ {a_2}^2 \end{pmatrix}^\top \begin{pmatrix} {b_1}^2 \\ \sqrt{2}\, b_1 b_2 \\ {b_2}^2 \end{pmatrix} = {a_1}^2 {b_1}^2 + 2 a_1 b_1 a_2 b_2 + {a_2}^2 {b_2}^2$$

$$= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^\top \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = (\mathbf{a}^\top \mathbf{b})^2$$

Key insight: we do not need to apply the transformation in the dual problem

➔ Much more computationnaly efficient

EPITA

Kernels

- Function $K(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{a}^\top \boldsymbol{b})$ is a second-degree polynomial kernel
- In machine learning: a kernel is a function capable of computing the dot product $\phi(\boldsymbol{a})^\top \phi(\boldsymbol{b})$ based only on the original vectors **a** and **b**, without having to compute the tranformation ϕ.
- The common kernels:

$$\text{Linear:} \quad K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^\top \mathbf{b}$$

$$\text{Polynomial:} \quad K(\mathbf{a}, \mathbf{b}) = (\gamma \mathbf{a}^\top \mathbf{b} + r)^d$$

$$\text{Gaussian RBF:} \quad K(\mathbf{a}, \mathbf{b}) = \exp\left(-\gamma \| \mathbf{a} - \mathbf{b} \|^2\right)$$

$$\text{Sigmoid:} \quad K(\mathbf{a}, \mathbf{b}) = \tanh\left(\gamma \mathbf{a}^\top \mathbf{b} + r\right)$$

EPITA

Mercer's Theorem

- If a function K(**a**,**b**) respects a few mathematical conditions called Mercer's conditions (e.g., K must be continuous and symmetric in its argument so that K(**a**,**b**) = K(**b**,**a**), etc.) then there exists a function ϕ that maps **a** and **b** into another space (possibly with much higher dimension) such that $\mathrm{K}(\mathbf{a}, \mathbf{b}) = \phi(\boldsymbol{a})^{\top}\phi(\boldsymbol{b})$.

➔ You can use K as a kernel because you know ϕ exists even you don't know what ϕ is.

➔ RBF Kernel: ϕ maps each training instance to an infinite dimensional space, so nice to not actually perform the mapping.

➔ Some frequently used kernels (sigmoid kernel) do not respect all of Mercer's conditions, but they generally work well in practice.

Making prediction with kernelized SVM

$$\hat{\boldsymbol{w}} = \sum_{i=1}^{m} \widehat{\alpha^{(i)}} t^{(i)} x^{(i)}$$
$$\hat{b} = \frac{1}{n_s} \sum_{i=1}^{m} \left(t^{(i)} - \hat{\boldsymbol{w}}^{\top} \boldsymbol{x}^{(i)} \right)$$

- You know how to go from dual solution to primal one for linear SVM Classifier.
- But if you apply the kernel trick, you end up with equations that include $\phi\left(x^{(i)}\right)$!
- In fact $\hat{\boldsymbol{w}}$ must have the same number of dimensions as $\phi\left(x^{(i)}\right)$ which may be huge even infinite ➔ you cannot compute it!
- You can make prediction without knowing $\hat{\boldsymbol{w}}$
  - Plug the formula for $\hat{\boldsymbol{w}}$ into the decision function for a new instance $\mathbf{x}^{(n)}$, and we get an equation with only dot product between input vectors.

$$h_{\hat{\mathbf{w}}, \hat{b}}\left(\phi\left(\mathbf{x}^{(n)}\right)\right) = \hat{\mathbf{w}}^{\top}\phi\left(\mathbf{x}^{(n)}\right) + \hat{b} = \left(\sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \phi\left(\mathbf{x}^{(i)}\right)\right)^{\top} \phi\left(\mathbf{x}^{(n)}\right) + \hat{b}$$

$$= \sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \left(\phi\left(\mathbf{x}^{(i)}\right)^{\top} \phi\left(\mathbf{x}^{(n)}\right)\right) + \hat{b}$$

$$= \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \hat{\alpha}^{(i)} t^{(i)} K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(n)}\right) + \hat{b}$$

Since $\alpha^{(i)} \neq 0$ only for support vectors ➔ **compute the dot product of the new input with only the support vectors, not all the training instances.**

Computing the bias term $\hat{b}$

- Same tricks

$$\hat{b} = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \widehat{\mathbf{w}}^{\mathsf{T}} \phi\left(\mathbf{x}^{(i)}\right) \right) = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \left(\sum_{j=1}^{m} \hat{\alpha}^{(j)} t^{(j)} \phi\left(\mathbf{x}^{(j)}\right) \right)^{\mathsf{T}} \phi\left(\mathbf{x}^{(i)}\right) \right)$$

$$= \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \sum_{\substack{j=1 \\ \hat{\alpha}^{(j)} > 0}}^{m} \hat{\alpha}^{(j)} t^{(j)} K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) \right)$$

EPITA

Summary

- Fundamental idea behind SVM
- First « dating » with kernels
- Hyperparameters
- ...

Merci !

Exercises

- See on moodle


EPITA
ÉCOLE D'INGENIEURS EN INFORMATIQUE