Lecture 5

Support vector machine (SVM)

Support vector machine (SVM)

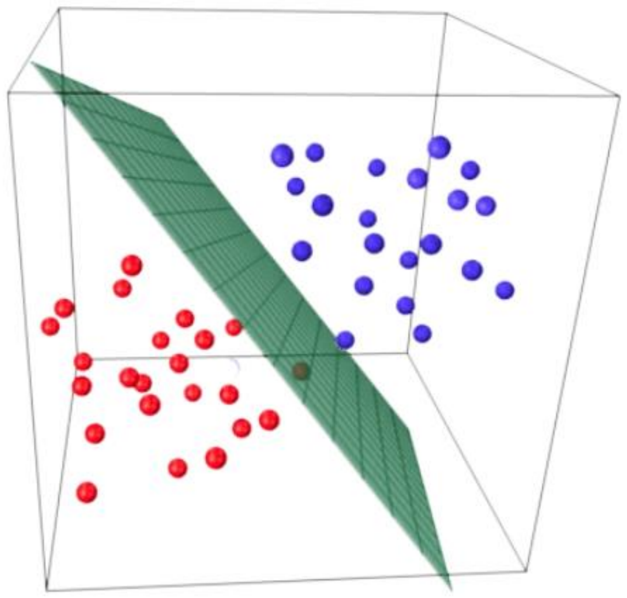
One of the most popular learning methods used to solve classification and regression problems.

The main idea of the method is **to construct a hyperplane that separates the sample objects in an optimal way**.

The algorithm operates under the assumption that the greater the distance (gap) between the separating hyperplane and the objects of the separable classes, the smaller the average classifier error will be.

Linear classification

The basic idea of a linear classifier is that the feature space can be divided by a hyperplane into two half-spaces, each of which predicts one of two values of the target class. If this can be done without errors, then the training set is called linearly separable.



Linear classification

Consider a binary classification problem in which $X = R^d$ – object space;

$Y = \{-1, +1\}$ – set of valid answers (target feature);

$X = \{(x_i, y_i)\}_{i=1}^{l}$ – training set;

Let’s call class $+1$ positive, and class -1 negative.

Here d – dimension of feature space, l – number of examples in the training set.

Linear classification

The linear classification model is defined as:

$$a(x) = sign(\langle w, x \rangle + b) = sign\left(\sum_{j=1}^{d} w_j x_j + b\right)$$

where $w \in R^d -$ vector of weights, $b \in R -$ bias, $sign(\,) -$"signum" function that returns the sign of its argument, $a(x) -$ classifier response for x.

Linear classification

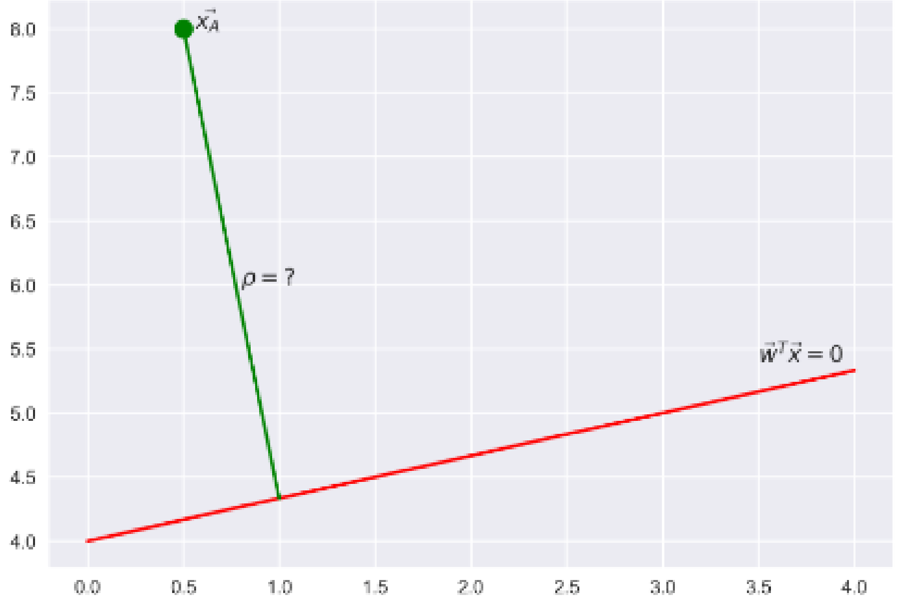
Geometrically linear classifier corresponds to a hyperplane with a normal vector w which is given by the equation $\langle w, x \rangle = 0$. Scalar product $\langle w, x \rangle$ is proportional to the distance from the hyperplane to the point and its sign shows on which side of the hyperplane the given point is located.

Linear classification

To be precise, the distance from a point with a radius vector x_A

to plane $\langle w, x\rangle = 0$:

$$\rho(x_A, \langle w, x\rangle = 0) = \frac{\langle w,x\rangle}{\|w\|}$$



Linear classification

Thus, the linear classifier divides the space into two parts using a hyperplane, and at the same time assigns one half-space to the positive class and the other to the negative class.

We can find the above formula for the distance from a point to a plane using the Lagrange method.

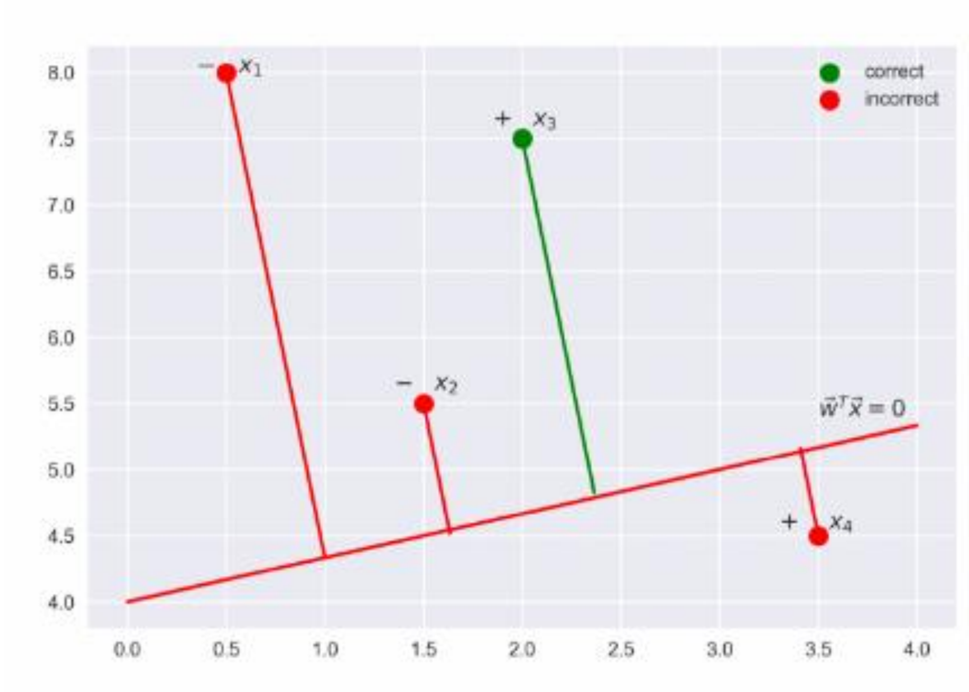
So, we want to find a point x on plane $\langle w, x\rangle = 0$ such that the distance between x and x_0 is minimal:

$$\begin{cases} \rho(x, x_0) \to \min\limits_x \\ \langle w, x\rangle = 0 \end{cases}$$

Classification margin

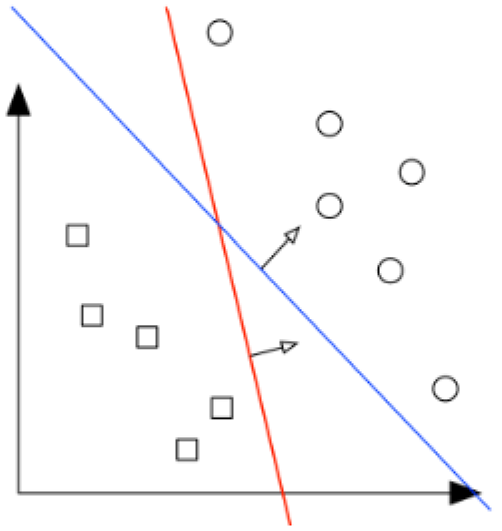
Classification margin for an object of training sample (x_i, y_i) looks like:

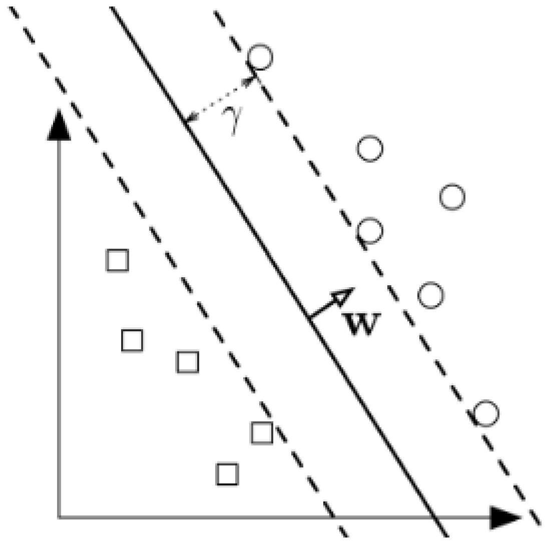
$$M(x_i, y_i, w) = y_i \cdot \langle w, x \rangle$$



Support Vector Machine (SVM)

Support Vector Machine (SVM) is based on the idea of maximizing the gap between classes.





SVM for linearly separable sampling

Let’s consider linear classifiers:

$$a(x) = sign(\langle w, x \rangle + b), w \in R^d, b \in R$$

We will assume that there are such parameters w_*, b_* that $a(x)$ does not make a single error on the training set.

In this case, the sample is said to be **linearly separable**.

SVM for linearly separable sampling

Multiply the parameters w, b by the same positive constant, the classifier will not change and we will get:

$$\min_{x \in X} |\langle w, x \rangle + b| = 1$$

The distance from a point with a radius vector x_0 to plane a:

$$\rho(x_0, a) = \frac{|\langle w, x \rangle + b|}{\|w\|}$$

Then the distance from the hyperplane to the nearest example from the training set is equal to:

$$\min_{x \in X} \frac{|\langle w, x \rangle + b|}{\|w\|} = \frac{1}{\|w\|} \min_{x \in X} |\langle w, x \rangle + b| = \frac{1}{\|w\|}$$

SVM for linearly separable sampling

$$\min_{x \in X} \frac{|\langle w, x\rangle + b|}{\|w\|} = \frac{1}{\|w\|} \min_{x \in X} |\langle w, x\rangle + b| = \frac{1}{\|w\|}$$

This value is also called the margin. It is a function of w.

The width of classifier is equal to $\frac{2}{\|w\|}$.

So, we need to build a classifier that perfectly separates the training sample and at the same time has a maximum margin.

SVM for linearly separable sampling

So, the corresponding optimization problem (hard margin support vector machine):

$$\begin{cases} \frac{1}{2}\|w\|^2 \to \min\limits_{w,b} \\ y_i(\langle w, x_i \rangle + b) \geq 1, i = 1, \ldots, l \end{cases}$$

The functional is strictly convex, and the constraints are linear.

SVM for linearly separable sampling

Let us now consider the general case when the sample cannot be perfectly divided by a hyperplane. This means that at least one of the restrictions in the previous problem will be violated.

Let's make these restrictions soft by introducing a penalty $\xi_i \geq 0$:

$$y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i, i = 1, \dots, l$$

If margin $0 \leq y_i(\langle w, x_i \rangle + b) < 1$ then the object is correctly classified, but has a non-zero penalty.

SVM for linearly separable sampling

So $\frac{1}{\|w\|}$ in this case it is called **soft margin**.

We need to maximize the gap and minimize the penalty for non-ideal sample division.

We will minimize the weighted sum of the two indicated quantities.

SVM for linearly separable sampling

An optimization problem corresponding to the support vector machine for a linearly inseparable sample (soft margin support vector machine):

$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i \to \min\limits_{w,b,\xi} \\ y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i, i = 1, \dots, l \\ \xi_i \geq 0, i = 1, \dots, l \end{cases}$$

The larger the parameter C here, the more tuned we will be to the training sample. This problem is also convex and has a unique solution.

Unconditional optimization problem

Let's rewrite the conditions of the problem:

$$\begin{cases} \xi_i \geq 1 - y_i(\langle w, x_i \rangle + b), i = 1, \dots, l \\ \xi_i \geq 0, i = 1, \dots, l \end{cases}$$

ξ_i should be as small as possible:

$$\xi_i = \max(0,1 - y_i(\langle w, x_i \rangle + b))$$

Let’s substitute it into the functional and get an unconditional optimization problem:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} \max\big(0,1 - y_i(\langle w, x_i \rangle + b)\big) \rightarrow \min_{w,b}$$

Solving the support vector machine problem

So, the support vector machine reduces to solving the optimization problem:

$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i \to \min\limits_{w,b,\xi} \\ y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i, i = 1, \dots, l \\ \xi_i \geq 0, i = 1, \dots, l \end{cases}$$

To solve such conditional optimization problems with conditions in the form of inequalities or equalities, the Lagrangian and the dual optimization problem are often used.

Optimization problems and the Kuhn-Tucker theorem

Consider the minimization problem:

$$\begin{cases} f_0(x) \to \min\limits_{x \in R^d} \\ f_i(x) \leq 0, i = 1, \dots, m \\ h_i(x) = 0, i = 1, \dots, p \end{cases}$$

Consider the function:

$$L(x, \lambda, v) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} v_i h_i(x)$$

where $\lambda_i \geq 0$

Dual function

The dual function for a problem is the function obtained by taking the minimum of the Lagrangian in x:

$$g(\lambda, v) = \inf_x L(x, \lambda, v)$$

This function gives a lower bound for the minimum in the original optimization problem.

Let $x_* -$ the solution of the problem, $x' -$ valid solution:

$$L(x', \lambda, v) = f_0(x') + \sum_{i=1}^{m} \lambda_i f_i(x') + \sum_{i=1}^{p} v_i h_i(x') \leq f_0(x')$$

Dual function

So:

$$\inf_x L(x, \lambda, v) \leq f_0(x')$$

The dual function gives a lower bound for the minimum:

$$g(\lambda, v) \leq f_0(x_*)$$

Dual function

The dual problem to the original one has the form:

$$\begin{cases} g(\lambda, v) \to \max\limits_{\lambda, v} \\ \lambda_i \geq 0, i = 1, \dots, m \end{cases}$$

Strong and weak duality

Let (λ^*, v^*) the solution to the dual problem. The value of the dual function in this always does not exceed the conditional minimum of the original problem:

$$g(\lambda^*, v^*) \le f_0(x_*)$$

This is **weak duality**. $f_0(x_*) - g(\lambda^*, v^*)$ is the gap between the solutions of the direct and dual problems.

If $f_0(x_*) = g(\lambda^*, v^*)$ then talk about **strong duality.**

Kuhn-Tucker conditions

Let x_* and (λ^*, v^*) be the solutions to the direct and dual problems.

We assume, that there is a strong duality:

$$f_0(x_*) = g(\lambda^*, v^*) = \inf_x \left(f_0(x) + \sum_{i=1}^{m} \lambda_i^* f_i(x) + \sum_{i=1}^{p} v_i^* h_i(x) \right)$$

$$\leq f_0(x_*) + \sum_{i=1}^{m} \lambda_i^* f_i(x_*) + \sum_{i=1}^{p} v_i^* h_i(x_*) \leq f_0(x_*)$$

Kuhn-Tucker conditions

From the last inequality:

$$\sum_{i=1}^{m} \lambda_i^* f_i(x_*) = 0$$

$\lambda_i^* f_i(x_*) = 0, i = 1, \ldots, m$

These conditions are called complementary **slackness conditions**. They say that the Lagrange multiplier under the i-th constraint can be non-zero only if the constraint is satisfied with equality (in which case it is said to be active).

Kuhn-Tucker conditions

We can write down the Kuhn-Tucker conditions (necessary extremum conditions):

$$\begin{cases} \nabla f_0(x) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x) + \sum_{i=1}^{p} v_i^* \nabla h_i(x) = 0 \\ f_i(x_*) \leq 0, i = 1, \dots, m \\ h_i(x_*) = 0, i = 1, \dots, p \\ \lambda_i^* \geq 0, i = 1, \dots, m \\ \lambda_i^* f_i(x_*) = 0, i = 1, \dots, m \end{cases}$$

Kuhn-Tucker conditions

Theorem. Let x_* be a solution to the original problem. Then there are vectors λ^* and v^* that the conditions (KKT) are met.

Solving the support vector machine problem

So, the support vector machine reduces to solving the optimization problem:

$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} \xi_i \to \min\limits_{w,b,\xi} \\ y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i, i = 1, \dots, l \\ \xi_i \geq 0, i = 1, \dots, l \end{cases}$$

Solving the support vector machine problem

Let us construct a dual problem to the support vector machine problem. Let's write the Lagrangian:

$$L(w, b, \xi, \lambda, \mu) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\lambda_i\,[y_i(\langle w, x_i\rangle + b) - 1 + \xi_i] - \sum_{i=1}^{l}\mu_i\,\xi_i$$

Solving the support vector machine problem

The Kuhn-Tucker conditions:

$$\nabla_w L = w - \sum_{i=1}^{l} \lambda_i y_i x_i = 0 \Rightarrow w = \sum_{i=1}^{l} \lambda_i y_i x_i$$

$$\nabla_b L = -\sum_{i=1}^{l} \lambda_i y_i = 0 \Rightarrow \sum_{i=1}^{l} \lambda_i y_i = 0$$

$$\nabla_{\xi_i} L = C - \lambda_i - \mu_i \Rightarrow \lambda_i + \mu_i = C$$

Solving the support vector machine problem

The Kuhn-Tucker conditions:

$$\lambda_i[y_i(\langle w, x_i\rangle + b) - 1 + \xi_i] = 0 \Rightarrow \lambda_i = 0 \; or \; y_i(\langle w, x_i\rangle + b) = 1 - \xi_i$$
$$\mu_i\xi_i = 0 \Rightarrow \mu_i = 0 \; or \; \xi_i = 0$$
$$\xi_i \geq 0, \lambda_i \geq 0, \mu_i \geq 0$$

Solving the support vector machine problem

Depending on the values of ξ_i and λ_i objects x_i are divided into three categories:

•$\xi_i = 0, \lambda_i = 0$. Such objects do not influence the decision and lie outside the dividing line. Objects in this category are called peripheral.

•$\xi_i = 0$, $0 < \lambda_i < C$. From condition $y_i(\langle w, x_i \rangle + b) = 1 - \xi_i$ we have $y_i(\langle w, x_i \rangle + b) = 1$. The object lies strictly on the border of the dividing strip.

•$\xi_i = 0, \lambda_i = C$. Such objects may lie inside the dividing strip or extend beyond it. If $0 < \xi_i < 1$ then the object is classified correctly, otherwise - incorrectly. Objects in this category are called support intruders.

Solving the support vector machine problem

So, the final classifier depends only on the objects lying on the border of the dividing strip and on the violating objects.

Let's construct a dual function. Let's substitute the expression $w = \sum_{i=1}^{l} \lambda_i y_i x_i$ into the Lagrangian and use the equations $\sum_{i=1}^{l} \lambda_i y_i = 0$ and $\lambda_i + \mu_i = C$:

Solving the support vector machine problem

$$L =$$

$$= \frac{1}{2}\left\|\sum_{i=1}^{l} \lambda_i y_i x_i\right\|^2 - \sum_{i,j=1}^{l} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle - b \sum_{i=1}^{l} \lambda_i y_i + \sum_{i=1}^{l} \lambda_i + \sum_{i=1}^{l} \xi_i (C - \lambda_i \mu_i)$$

$$= \sum_{i=1}^{l} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{l} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle$$

And conditions must be met: $\sum_{i=1}^{l} \lambda_i y_i = 0, \lambda_i + \mu_i = C$ and $\lambda_i, \mu_i \geq 0$

Solving the support vector machine problem

We arrive at the following dual problem:

$$\begin{cases} \sum_{i=1}^{l} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{l} \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle \rightarrow \max\limits_{\lambda} \\ 0 \leq \lambda_i \leq C, i = 1, \dots, l \\ \sum_{i=1}^{l} \lambda_i \, y_i = 0 \end{cases}$$

Kernel transition

The dual SVM problem depends only on scalar products of objects—individual feature descriptions are not included in it in any way.

This means that it is possible to make a kernel transition:

$$\begin{cases} \sum_{i=1}^{l} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{l} \lambda_i \lambda_j y_i y_j K(x_i, x_j) \to \max_{\lambda} \\ 0 \leq \lambda_i \leq C, i = 1, \ldots, l \\ \sum_{i=1}^{l} \lambda_i \, y_i = 0 \end{cases}$$

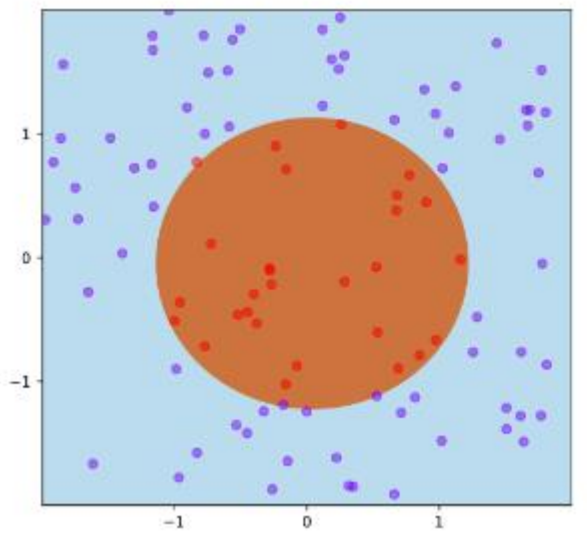
Kernel transition

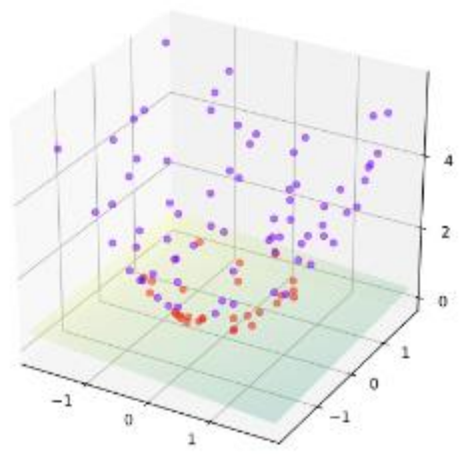
Using $w = \sum_{i=1}^{l} \lambda_i y_i x_i$ for classifier we receive:

$$a(x) = sign\left(\sum_{i=1}^{l} \lambda_i y_i \langle x_i, x \rangle + b\right)$$

Thus, the classifier measures the similarity of the new object to the objects from the training set by calculating the dot product between them. This expression also depends only on scalar products, so we can also go to the kernel in it.

Kernel transition





An example of separability in a new space

Pros and cons of SVM

Pros:

•well studied, there are important theoretical results;

•nicely formulated as an optimization problem;

•Linear SVM is fast and can work on very large samples;

•linear SVM is as well interpreted as other linear models;

•the solution depends only on scalar products of vectors, and the idea of the “kernel trick” is one of the most beautiful in the history of machine learning;

•nonlinear SVM is generalized to work with a wide variety of data types (sequences, graphs, etc.) due to specific kernels.

Pros and cons of SVM

Minuses:

•nonlinear SVM has high computational complexity and fundamentally does not scale well;

•nonlinear SVM is essentially uninterpretable;

•in classification problems, you often want to give the probability of assignment to a class, SVM cannot do this, and heuristics, as a rule, lead to poorly calibrated probabilities;

•kernel SVM is inferior to specific neural networks in many tasks, for example, in applications to graphs

Support Vector Machine in Python

Solving the problem of binary (when there are only two classes) classification:

•the algorithm is trained on objects from the training set for which the class labels are known in advance

•the already trained algorithm predicts the class label for each object from the delayed/test sample

Class labels can take values $Y = \{-1, +1\}$.

Object is a vector with n features $x = (x_1, x_2, \dots, x_n)$ in R^n.

When training, the algorithm must construct a **function** $F(x) = y$, where x is an object and y is a class mark.

Support Vector Machine in Python

The main goal of SVM as a classifier is to find the equation of the separating hyperplane $w_1x_1 + w_2x_2 + \cdots + w_nx_n + w_0 = 0$.

General view of the function: $F(x) = sign(w^T x - b)$

After adjusting the weights of the algorithm, all objects falling on one side of the constructed hyperplane will be predicted as the first class, and objects falling on the other side will be predicted as the second class.

Support Vector Machine in Python

We configure the separating hyperplane in such a way that class objects lie as far as possible from the separating hyperplane.

The algorithm maximizes the gap between the hyperplane and the class objects that are closest to it.

Support Vector Machine in Python

So, we need to maximize gap between two classes. Vector w is the normal vector to the separating hyperplane.

Distance between classes: $\frac{2}{\|w\|} \to max \Rightarrow \|w\| \to min$

$$\begin{cases} \frac{1}{2}\|w\|^2 \to \min\limits_{w,b} \\ y_i(\langle w, x_i\rangle + b) \geq 1, i = 1, \dots, l \end{cases}$$

We got a default SVM setting with a hard-margin SVM, when no object is allowed to fall into the separation strip

Support Vector Machine in Python

For linearly inseparable classes, we will allow the algorithm to make errors on training objects, but at the same time we will try to ensure that there are fewer errors:

$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i \to \min\limits_{w,b,\xi} \\ y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i, i = 1, \dots, l \\ \xi_i \geq 0, i = 1, \dots, l \end{cases}$$

Let’s substitute it into the functional and get an unconditional optimization problem:

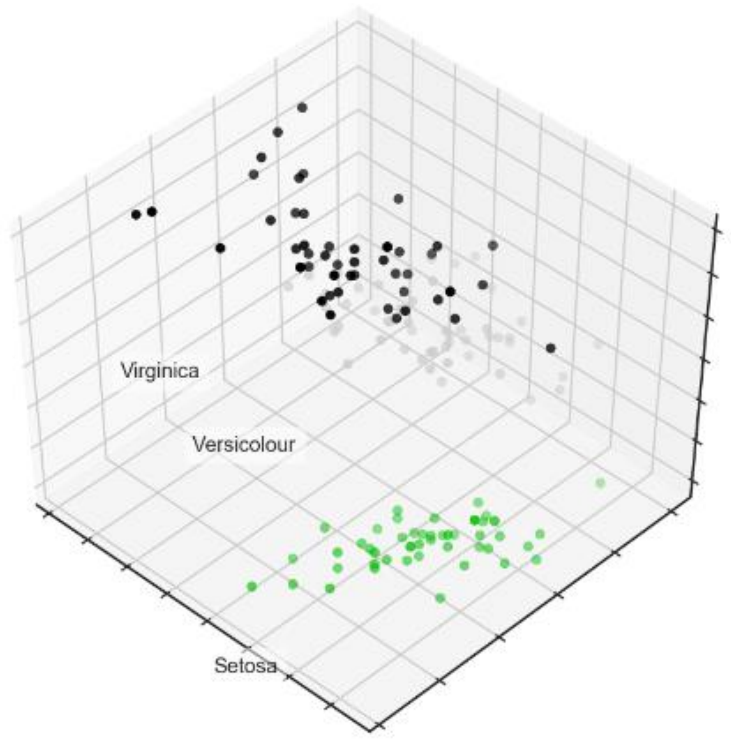
$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\max\big(0,1 - y_i(\langle w, x_i\rangle + b)\big) \to \min_{w,b}$$

Support Vector Machine in Python

```
pip install -U scikit-learn
```

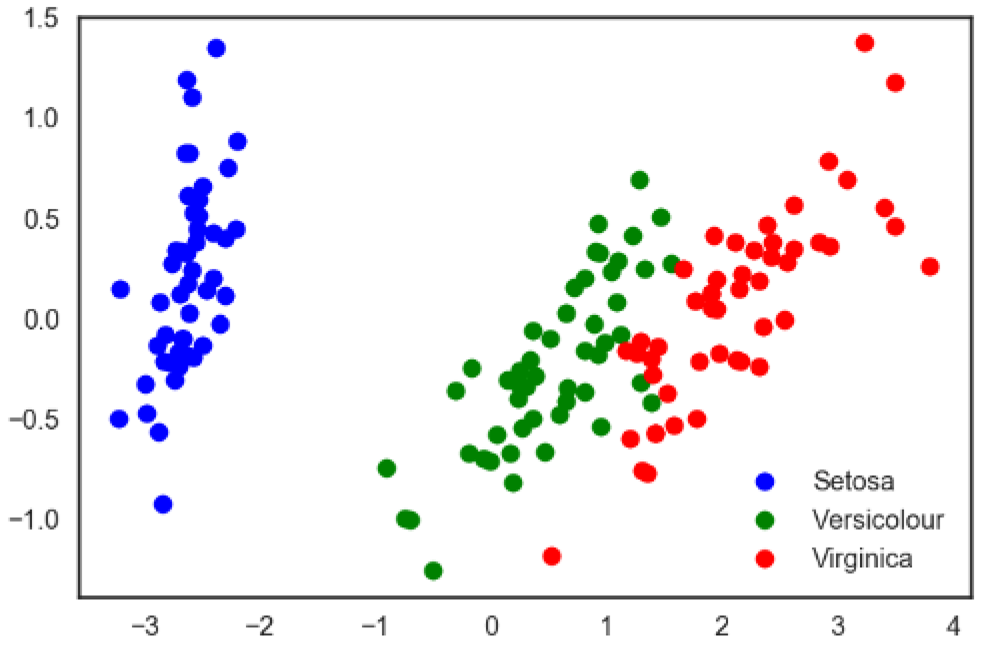
Support Vector Machine in Python

Let's start by loading all the necessary modules and spinning the usual dataset with irises following the example from the documentation of the scikit-learn package.



Support Vector Machine in Python

Now let's see how much PCA will improve the results for the model, which in this case will perform poorly at classification due to the fact that it does not have enough complexity to describe the data (2D):



```
pca = decomposition.PCA(n_components=2)
X_centered = X - X.mean(axis=0)
pca.fit(X_centered)
X_pca = pca.transform(X_centered)
```

```
Out: Accuracy: 0.88889
```

```
Out: Accuracy: 0.91111
```

Support Vector Machine in Python

Dataset “Iris”

```
# import some data to play with
iris = datasets.load_iris()

print('The data matrix:\n',iris['data'])
print('The classification target:\n',iris['target'])
print('The names of the dataset columns:\n',iris['feature_names'])
print('The names of target classes:\n',iris['target_names'])
print('The full description of the dataset:\n',iris['DESCR'])
print('The path to the location of the data:\n',iris['filename'])
```

Support Vector Machine in Python

Dataset “Iris”

- The data matrix
- The classification target
- The names of the dataset columns

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

- The names of target classes

```
['setosa' 'versicolor' 'virginica']
```

Support Vector Machine in Python

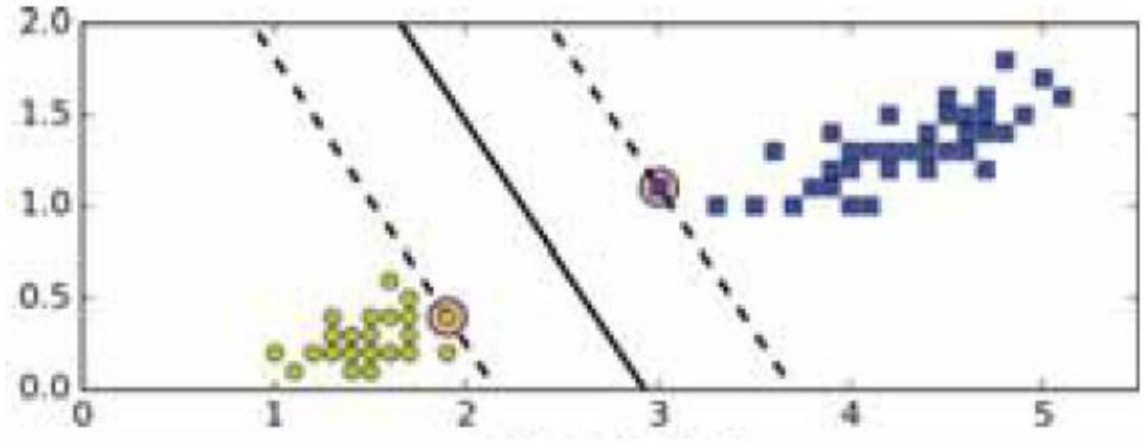
The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken from Fisher's paper. Note that it's the same as in R, but not as in the UCI Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Support Vector Machine in Python

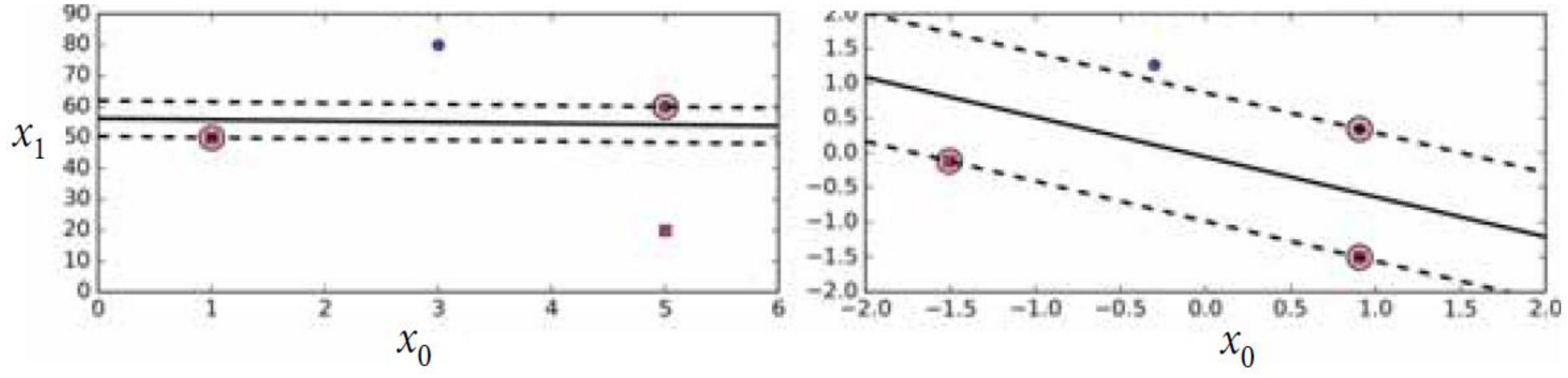
Large margin classification.

The line not only separates the two classes but is also located as far away as possible from the nearest training samples. The SVM classifier sets the widest possible band (represented by parallel dotted lines) between classes.



Support Vector Machine in Python

Adding additional training samples “outside the strip” will not affect the decision boundary at all: it is completely determined by the samples located at the edges of the strip (or “rests” on them). Such samples are called **support vectors**; in picture they are circled. SVM methods are sensitive to feature scales. After scaling the features (for example, using the **StandardScaler** class from Scikit-Learn), the decision boundary looks much better (in the graph on the right).



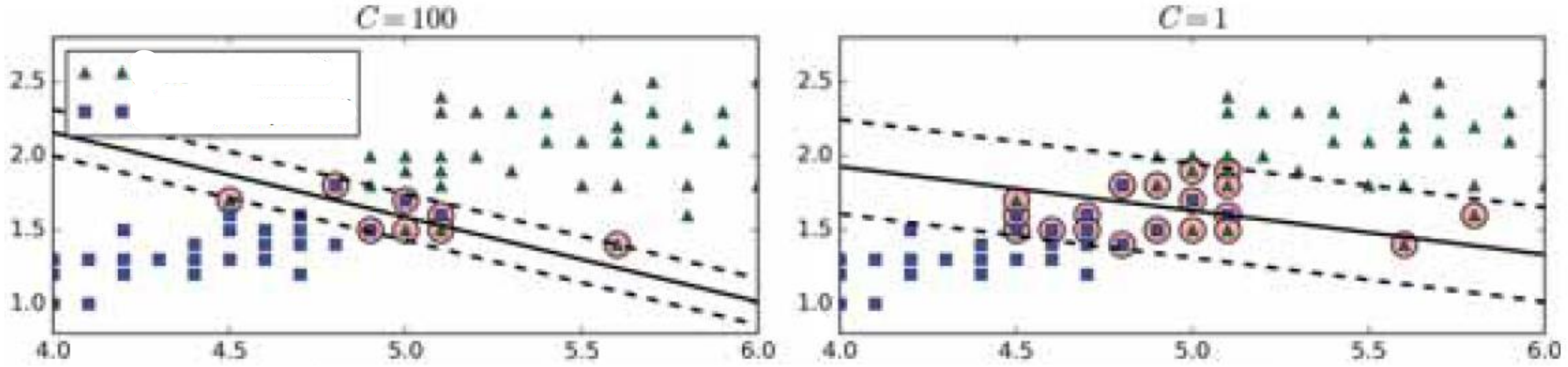
Support Vector Machine in Python

Hard margin classification. If we strictly fix that all samples are outside the band and on the right side, we get a hard margin classification. There are two main problems with rigid-gap classification. First, it only works if the data is linearly separable. Secondly, it is quite sensitive to emissions.

To avoid such problems, it is preferable to use a more flexible model. The goal is to find a good balance between keeping the strip as wide as possible and limiting the number of clearance violations. This is a **soft margin classification.**

Support Vector Machine in Python

In the SVM classes of the Scikit-Learn library, you can control said balance using the hyperparameter C: a smaller value of C leads to a wider band but more gap violations.

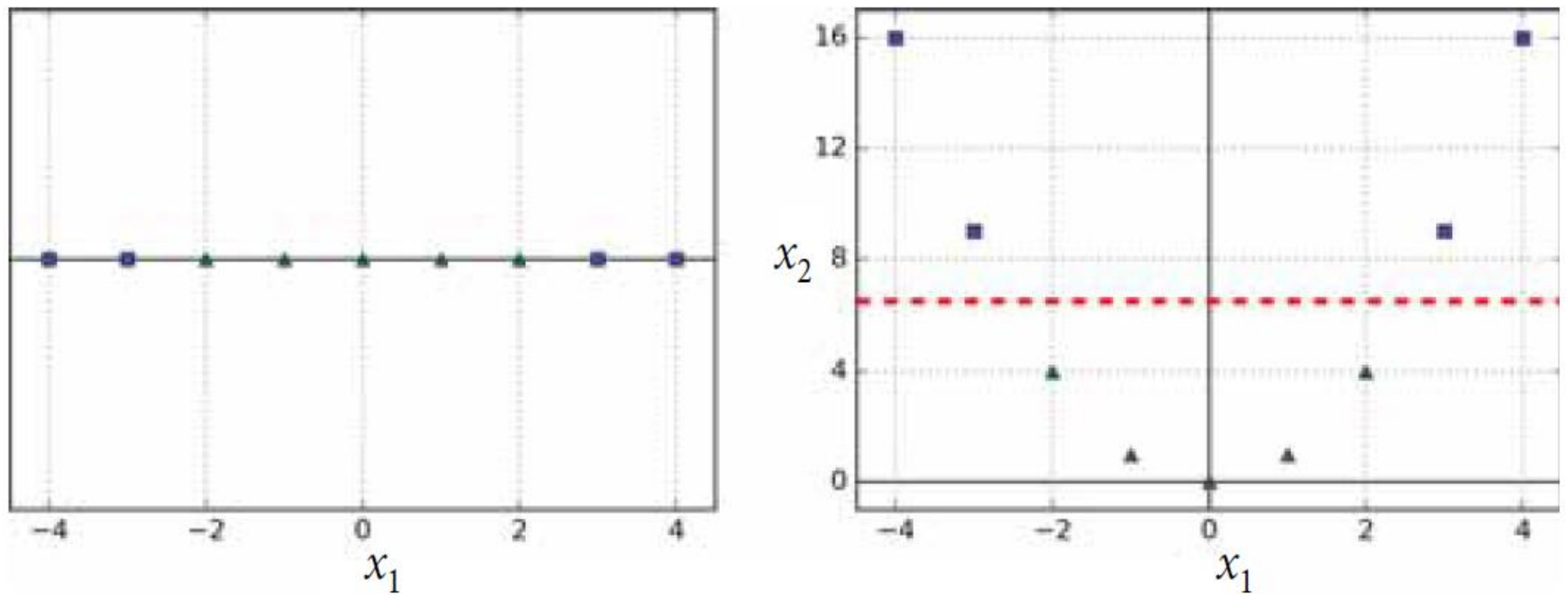


Support Vector Machine in Python

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

iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # длина лепестка, ширина лепестка
y = (iris["target"] == 2).astype(np.float64)  # ирис виргинский

svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("linear_svc", LinearSVC(C=1, loss="hinge")),
    ])

svm_clf.fit(X, y)
```

Support Vector Machine in Python

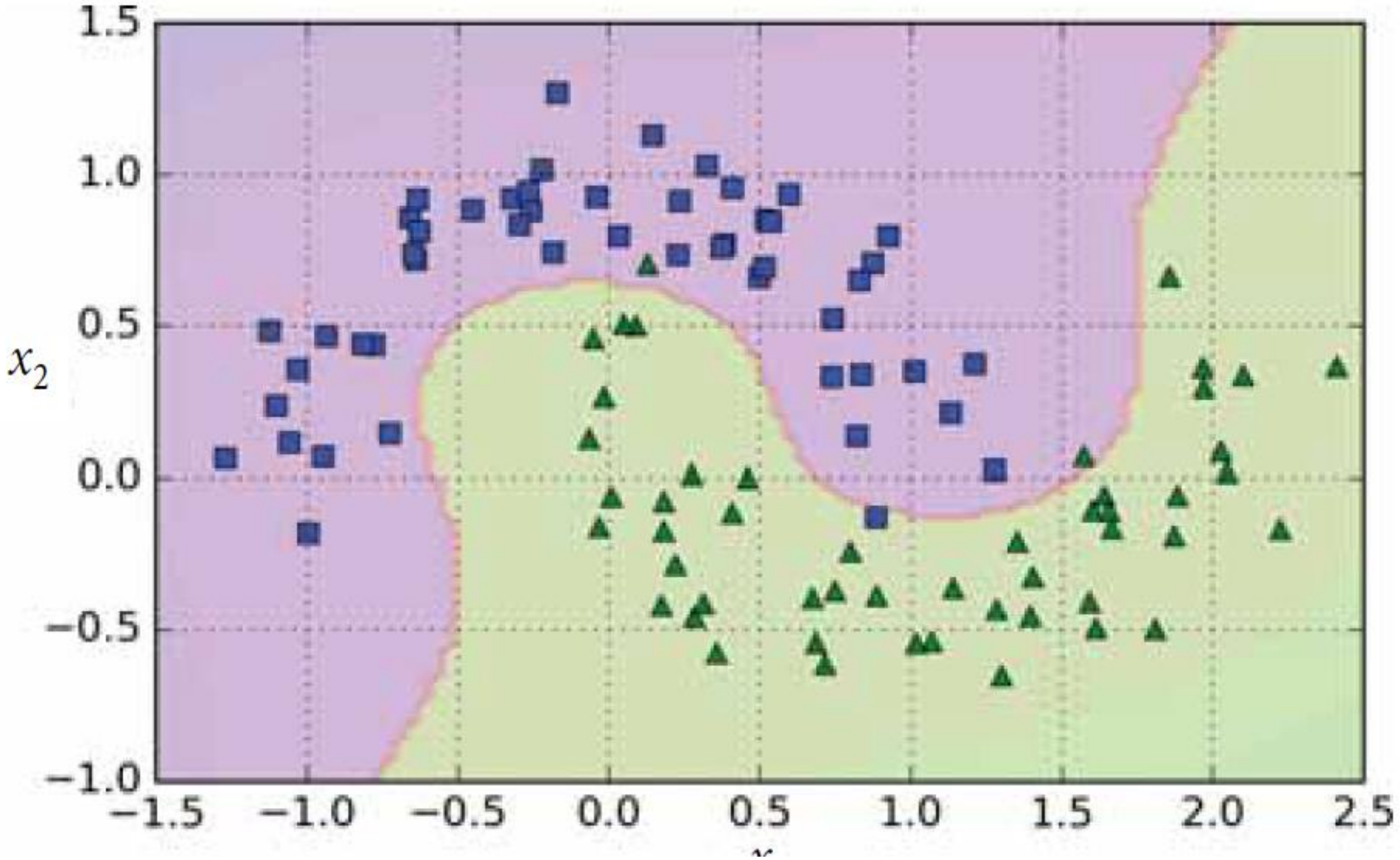
Although linear SVM classifiers are efficient and work surprisingly well in numerous cases, many datasets are far from being linearly separable.



Support Vector Machine in Python

```
from sklearn.datasets import make_moons
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"))
    ])

polynomial_svm_clf.fit(X, y)
```

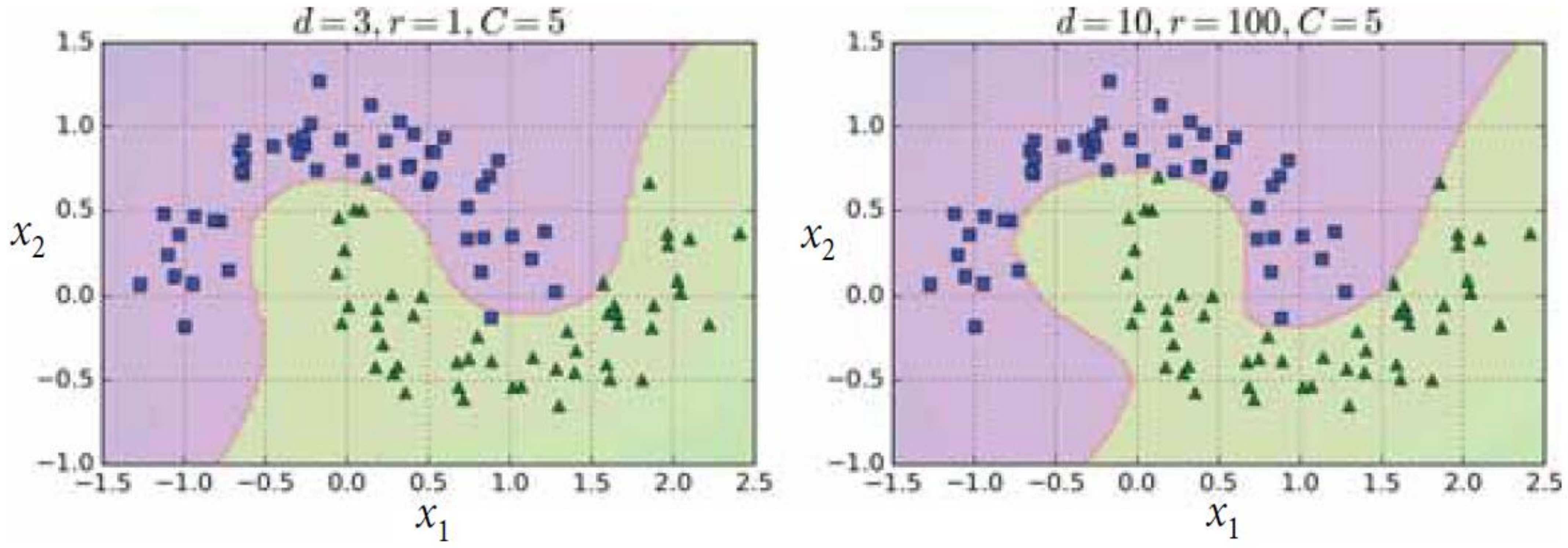
Support Vector Machine in Python



Various SVM parameters

```
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)
```

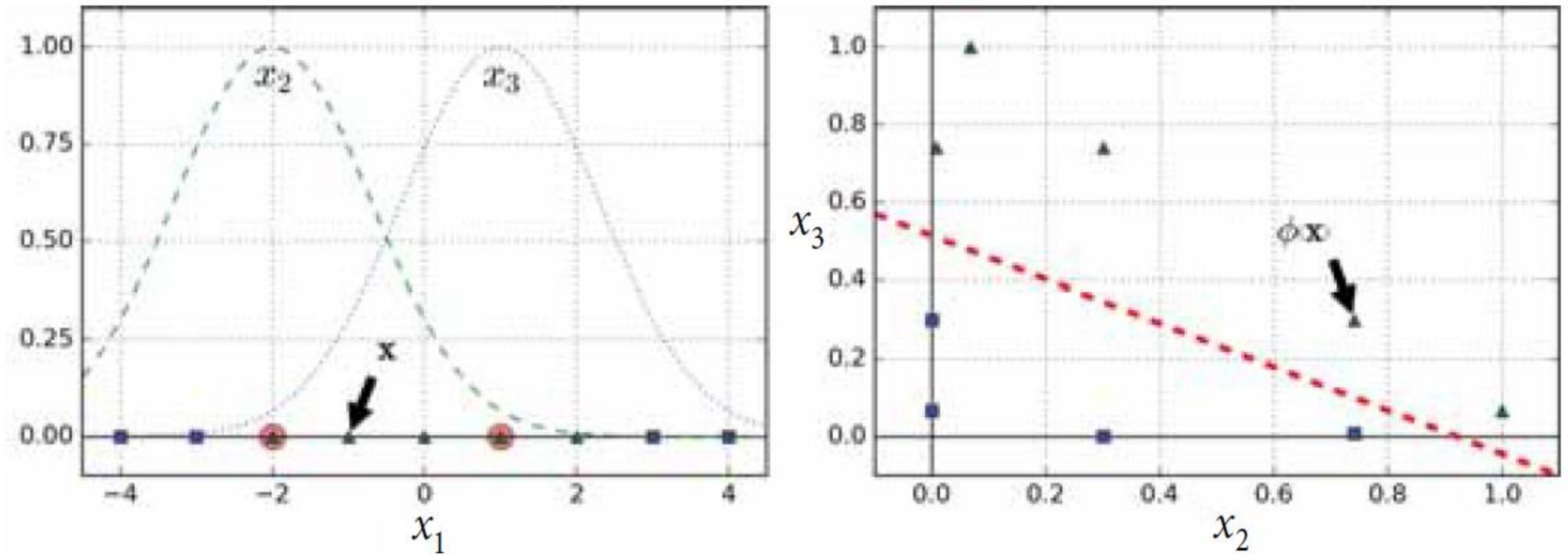
Various SVM parameters



Various SVM parameters

Similarity function. Another technique for solving nonlinear problems involves adding features calculated using a similarity function, which measures how much similarity each sample has to a particular landmark. For example, take the previously discussed one-dimensional data set and add two landmarks to it at $x_1 = -2$ and $x_2 = 1$. Then define the proximity function as a Gaussian Radial Basis Function (RBF) with $\gamma = 0.3$.

Various SVM parameters


x_2
x_3
$\mathbf{x}$
x_1
x_3
$\phi\,\mathbf{x}$
x_2