EEN1083
Data analysis and machine learning I

Ali Intizar

DCU
Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

Support Vector Machines

Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

Outline

Hard-margin linear SVM

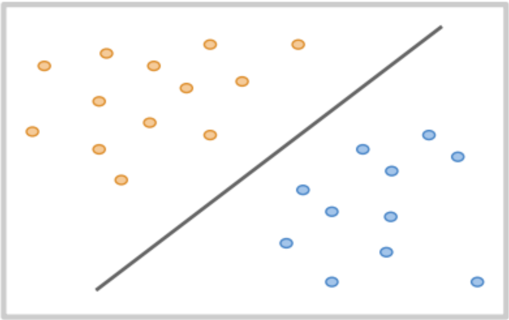
Soft-margin linear SVM

Kernel SVMs

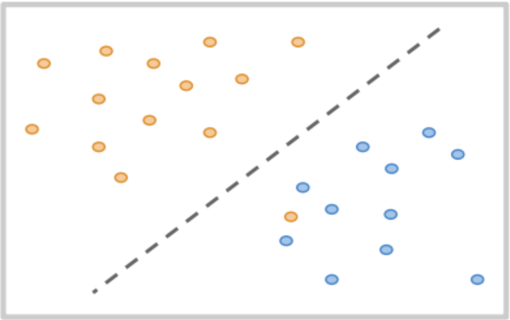
Tips for use

Hard-margin linear SVM

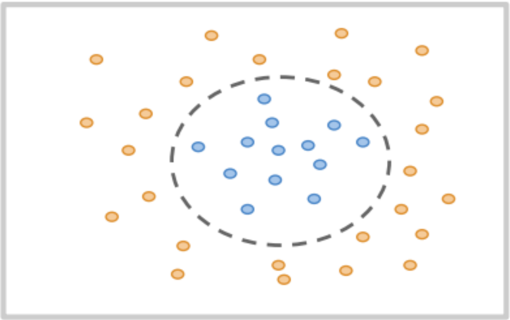
Three problems



Linearly separable



Almost linearly separable



Not linearly separable

The binary classification setup

Training set:

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{N} \qquad \mathbf{x}_i \in \mathrm{R}^D, \quad y_i \in \{-1, +1\}$$

Linear prediction function:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

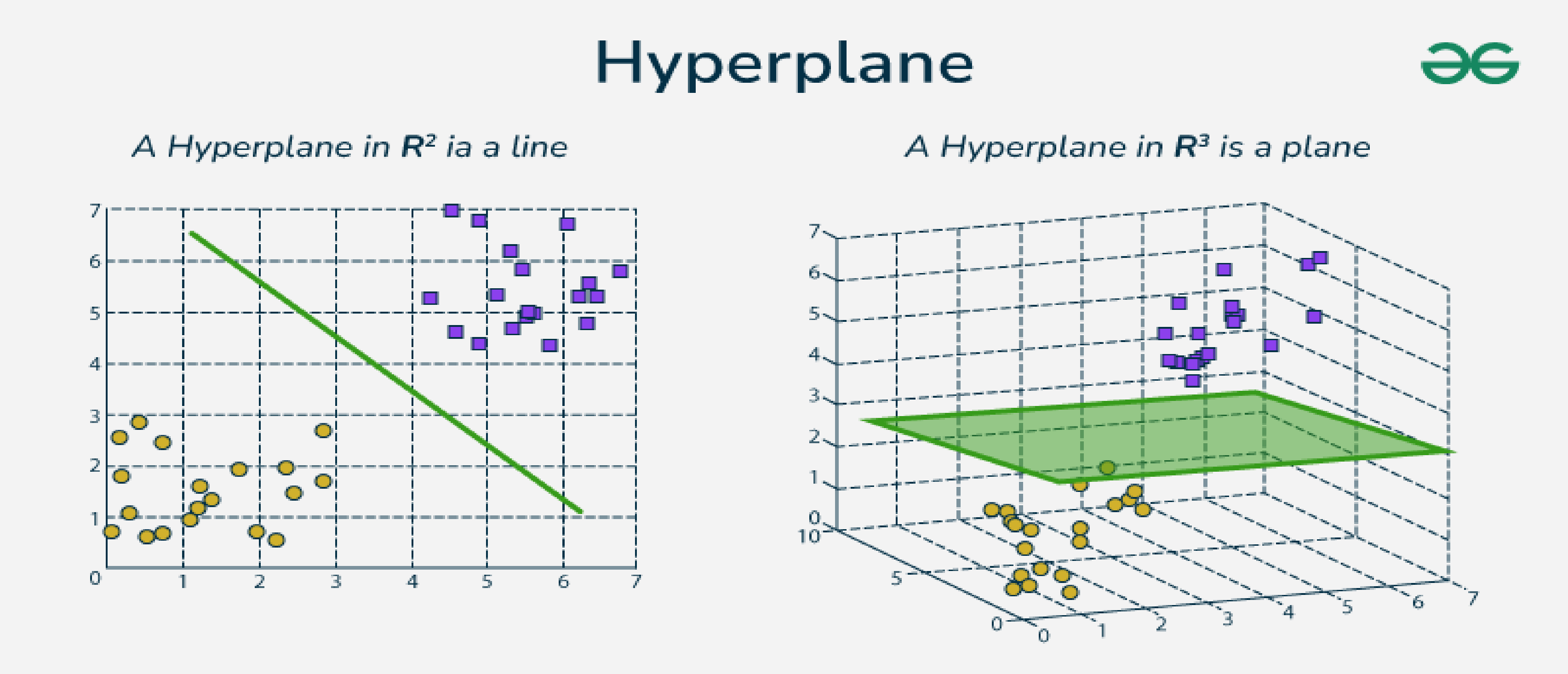
And we would like that:

$$f(\mathbf{x}_i) > 0 \quad \text{if} \quad y_i = +1$$
$$f(\mathbf{x}_i) < 0 \quad \text{if} \quad y_i = -1$$

Which we can also write as:

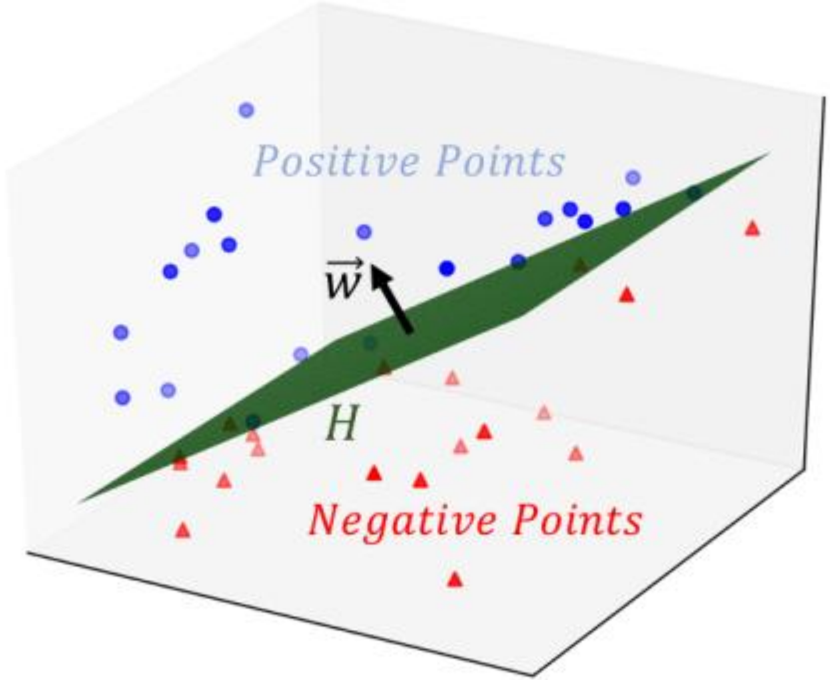
$$y_i f(\mathbf{x}_i) > 0$$
$$y_i(\mathbf{w}^T \mathbf{x}_i + b) > 0$$

Hyperplane geometry

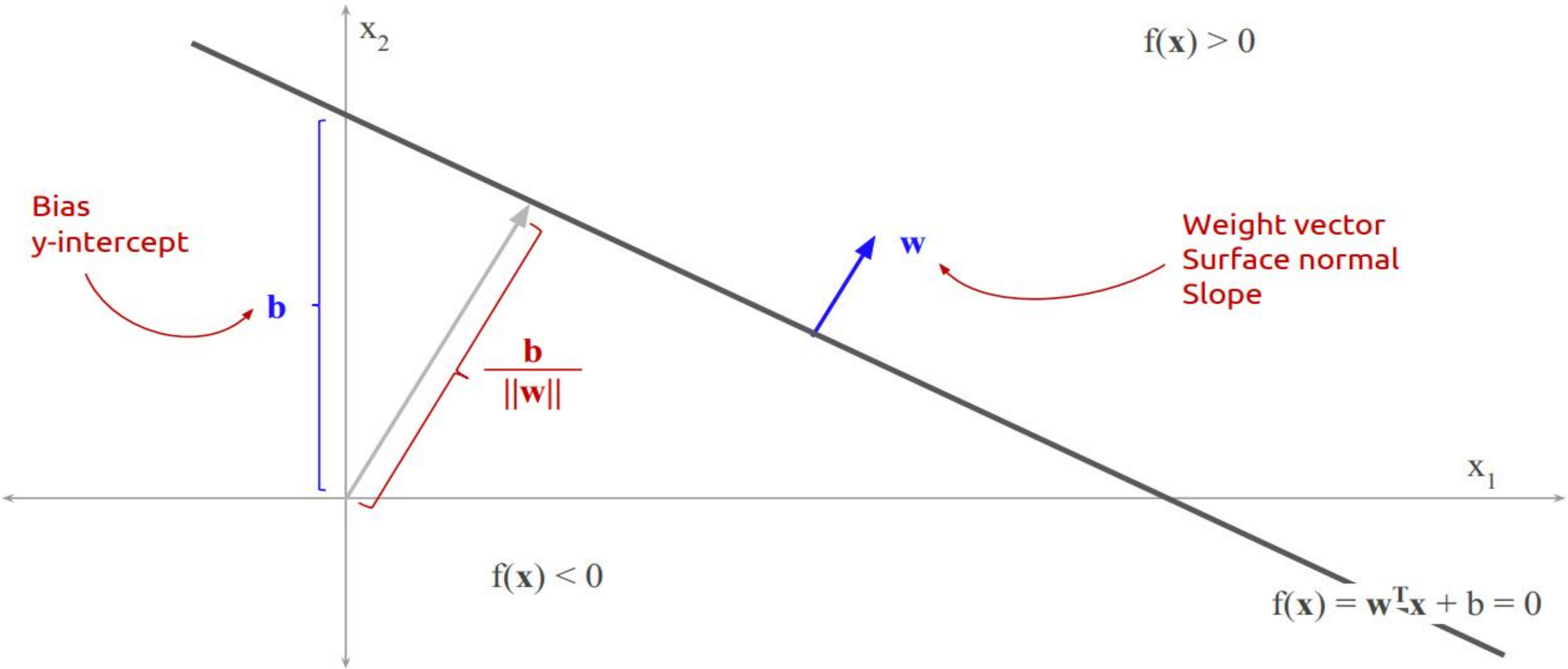


https://www.geeksforgeeks.org/hyperplane-subspace-and-halfspace/

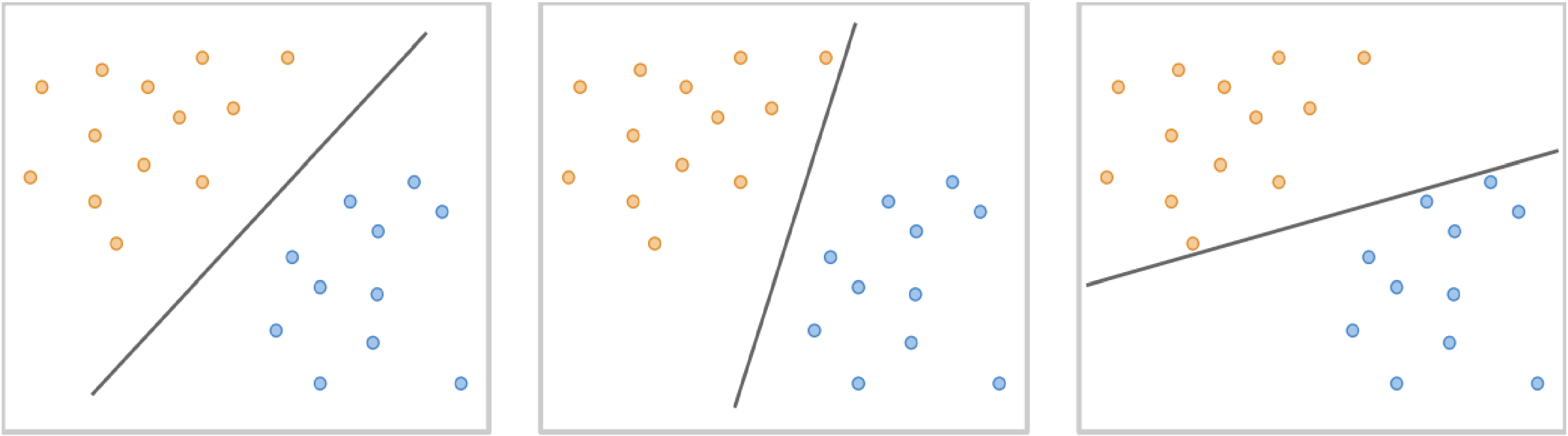
Hyperplane geometry



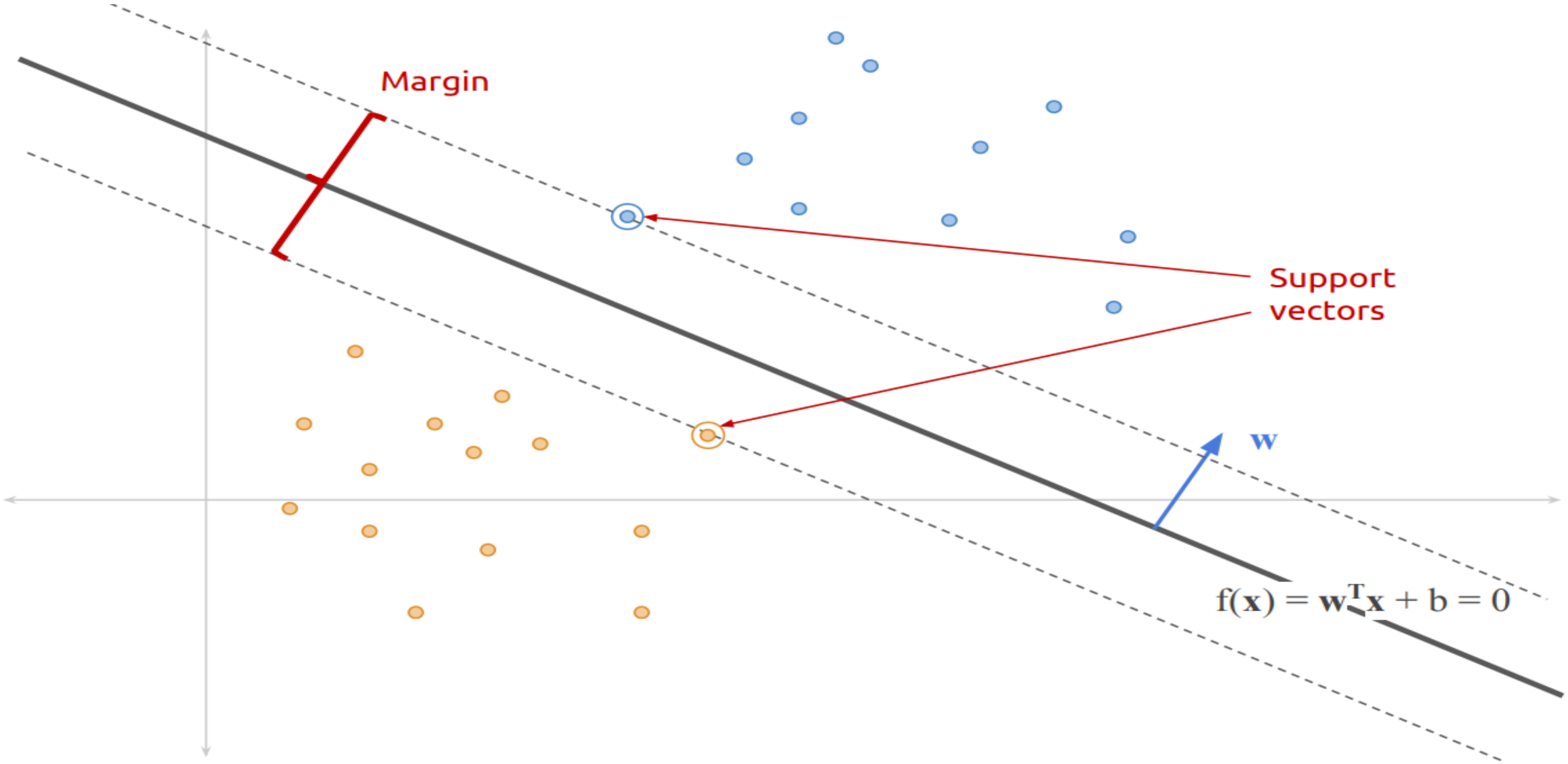
Hyperplane geometry



Which separating hyperplane is the best?



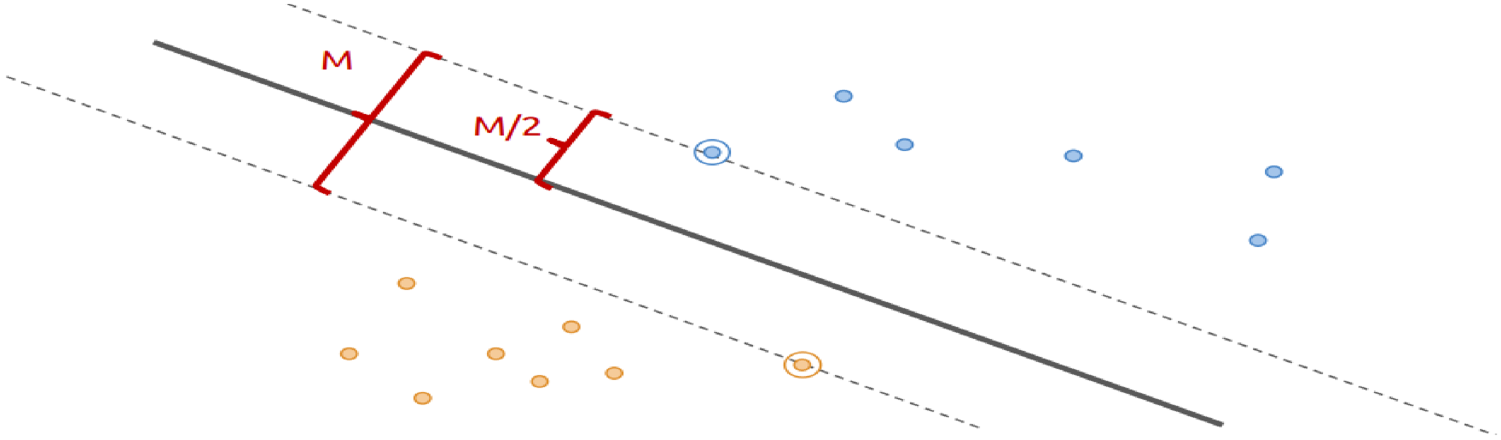
Max-margin principle



Max-margin principle

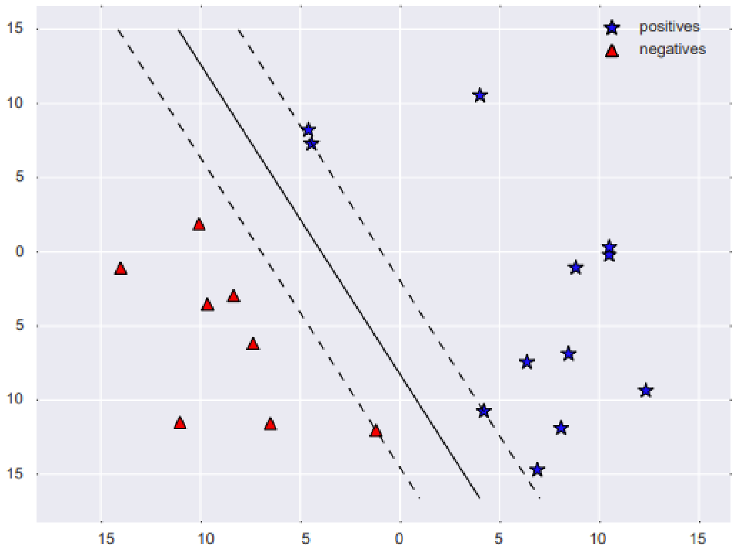
$$\underset{\mathbf{w},b}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \frac{y_i}{\|\mathbf{w}\|}(\mathbf{w}^T\mathbf{x}_i + b) \geq \frac{M}{2}, \ i = 1, \ldots, N.$$



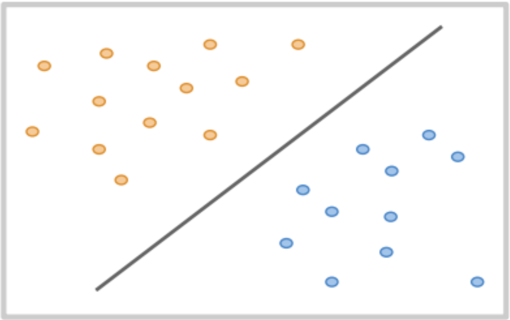
Python code to solve hard margin SVM with Quadratic Optimizer

```
class HardMarginSVM(object):
    def fit(self, X, y):
        n, m = X.shape
        mat = cvxopt.matrix

        # Formulate QP
        P = mat(np.eye(m+1))
        P[-1,-1] = 0
        q = mat(0.0, (m+1, 1))
        G = mat(np.c_[-y[:,np.newaxis] * X, -y])
        h = mat(-1.0, (n, 1))

        # Solve QP
        solution = cvxopt.solvers.qp(P, q, G, h)

        # Make sure we have the optimal solution
        if solution['status'] != 'optimal':
            raise ValueError('infeasible')

        # Assign model parameters w, b
        x = np.array(solution['x'])
        self.w = x[:-1].ravel()
        self.b = x[-1][0]
```

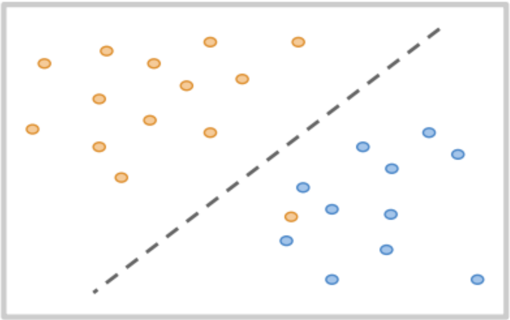


Soft-margin linear SVM

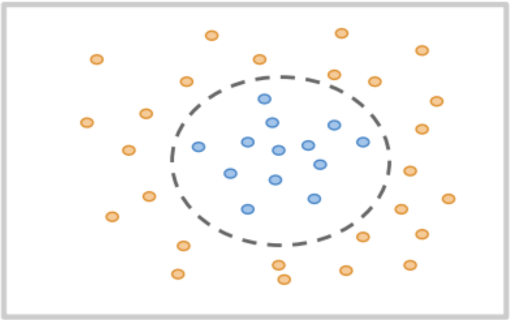
Three problems



Linearly separable

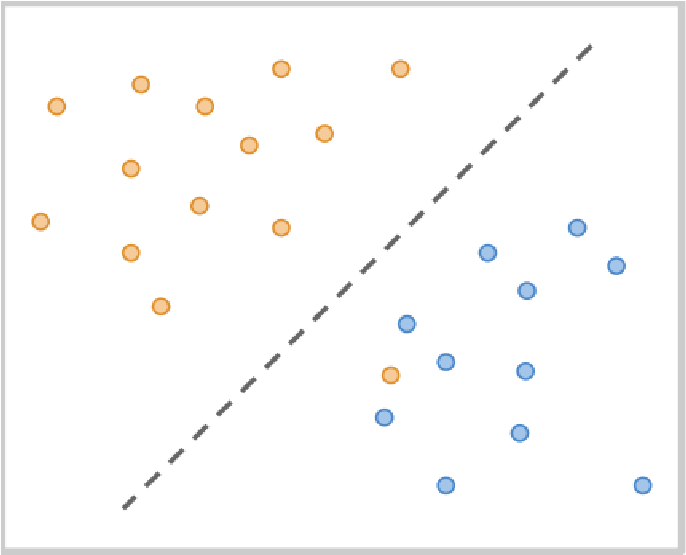


Almost linearly separable

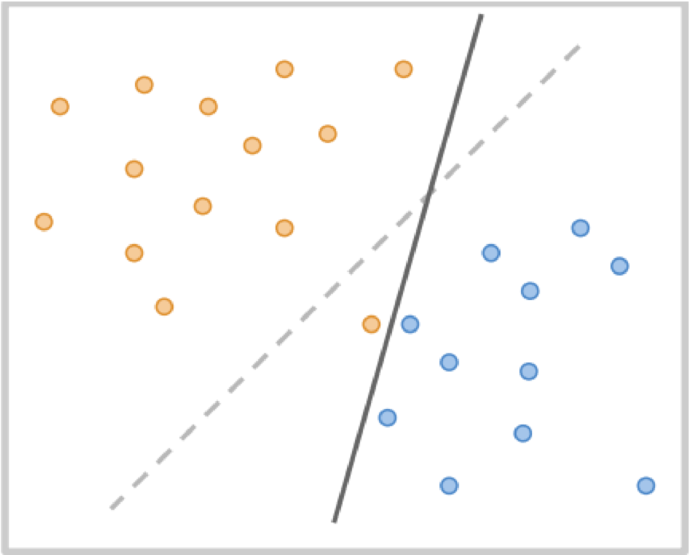


Not linearly separable

Motivating examples



Almost linearly separable



Linearly separable

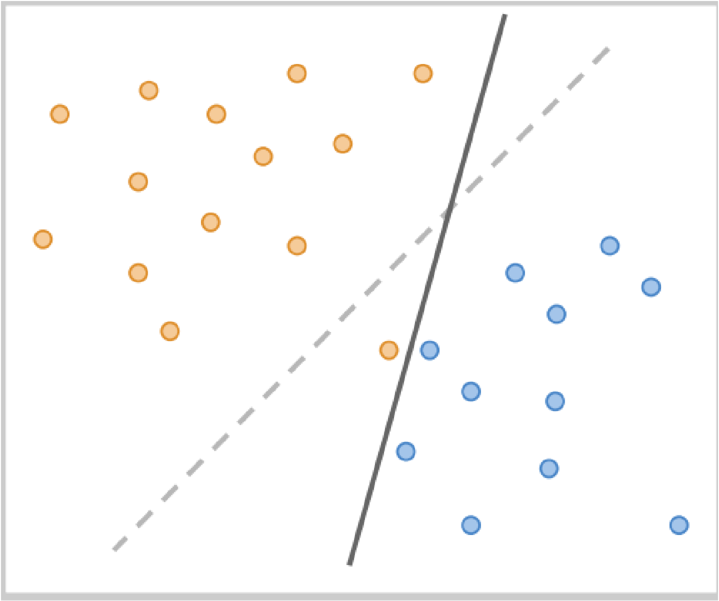
Motivating examples

Even when linearly separable, max margin solution can have a very narrow margin.

We could get a much larger margin if we ignored the constraint on the one outlier.

Trade off between satisfying constraints and maximizing the margin.

How do we allow the classifier to violate constraints? **Turn hard constraints into soft ones**.



Linearly separable

Objective

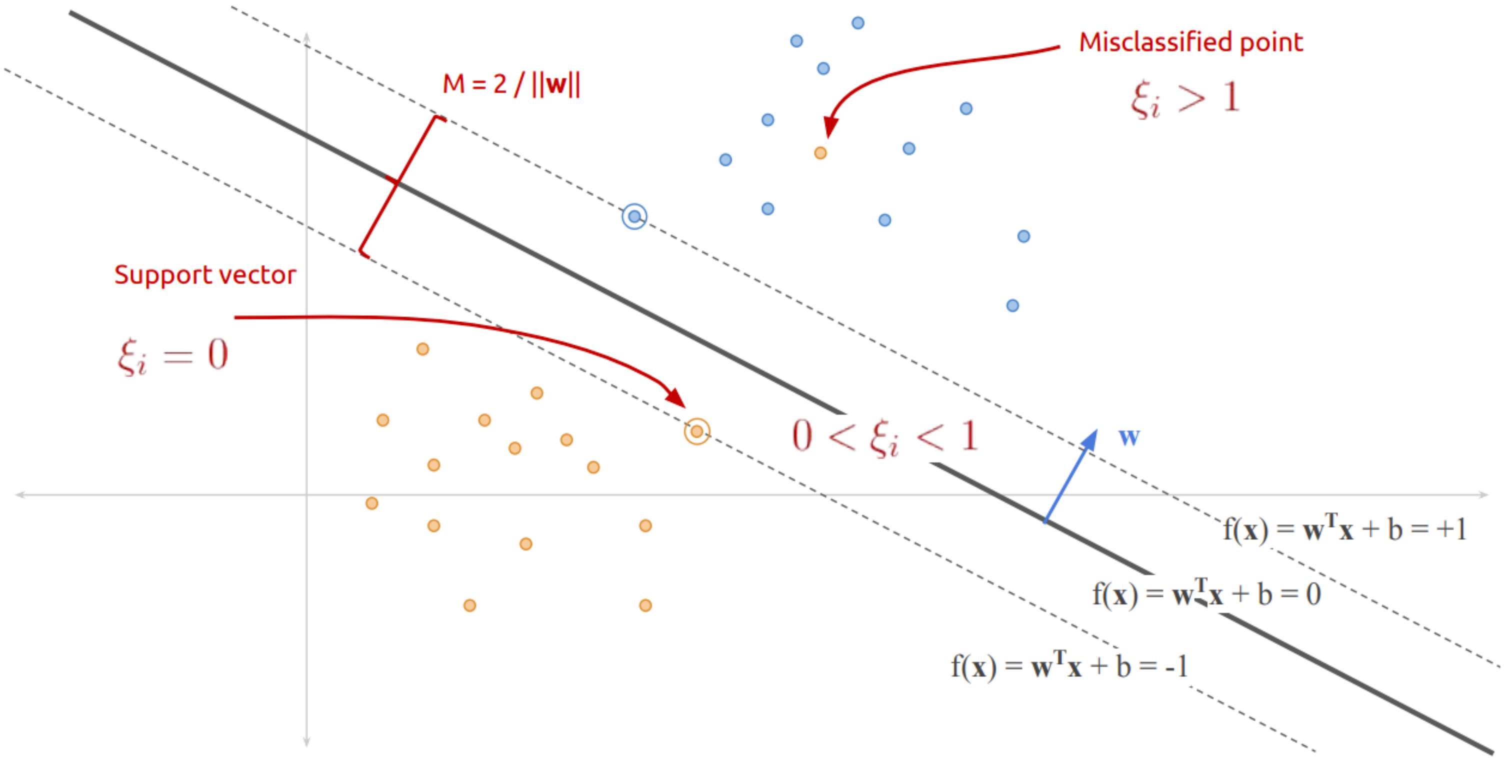
Hard-margin SVM:

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \end{aligned}$$

Soft-margin SVM:

$$\begin{aligned} \underset{\mathbf{w},b,\xi}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned}$$

Objective



Misclassified point
$\xi_i > 1$
M = 2 / ||**w**||
Support vector
$\xi_i = 0$
$0 < \xi_i < 1$
w
$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = +1$
$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0$
$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = -1$

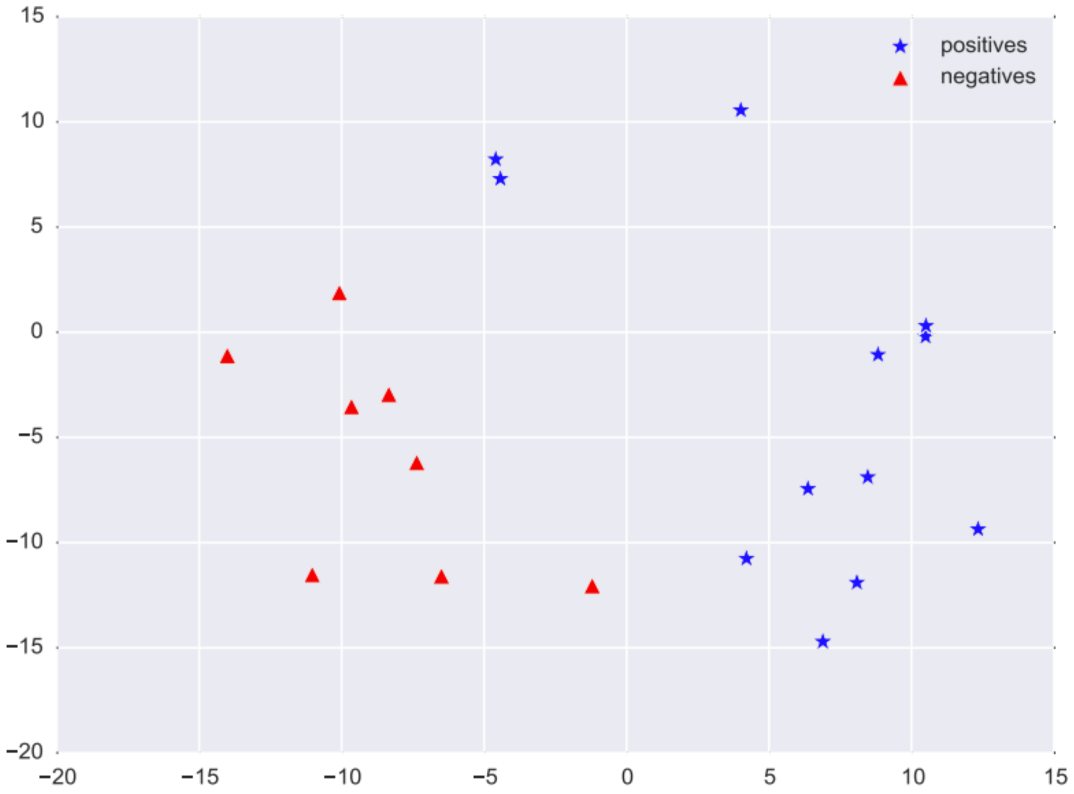
Soft-margin linear SVM

$$\begin{aligned} \underset{\mathbf{w},b,\xi}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned}$$

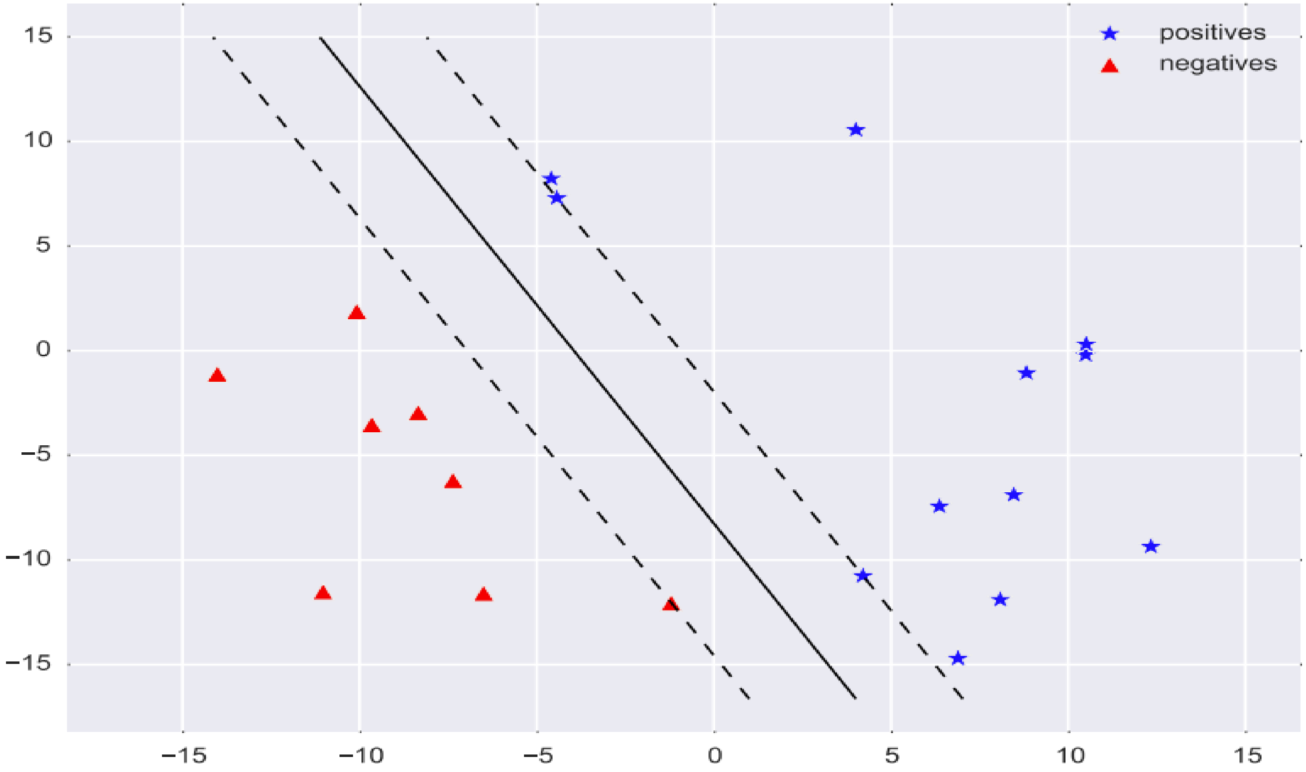
C is a **regularization** hyperparameter:

- Large C: high penalty for violation constraints $\rightarrow$ narrower margin.
- Small C: low penalty for violating constraints $\rightarrow$ larger margin.
- $C \rightarrow \infty$ produces hard-margin SVM.

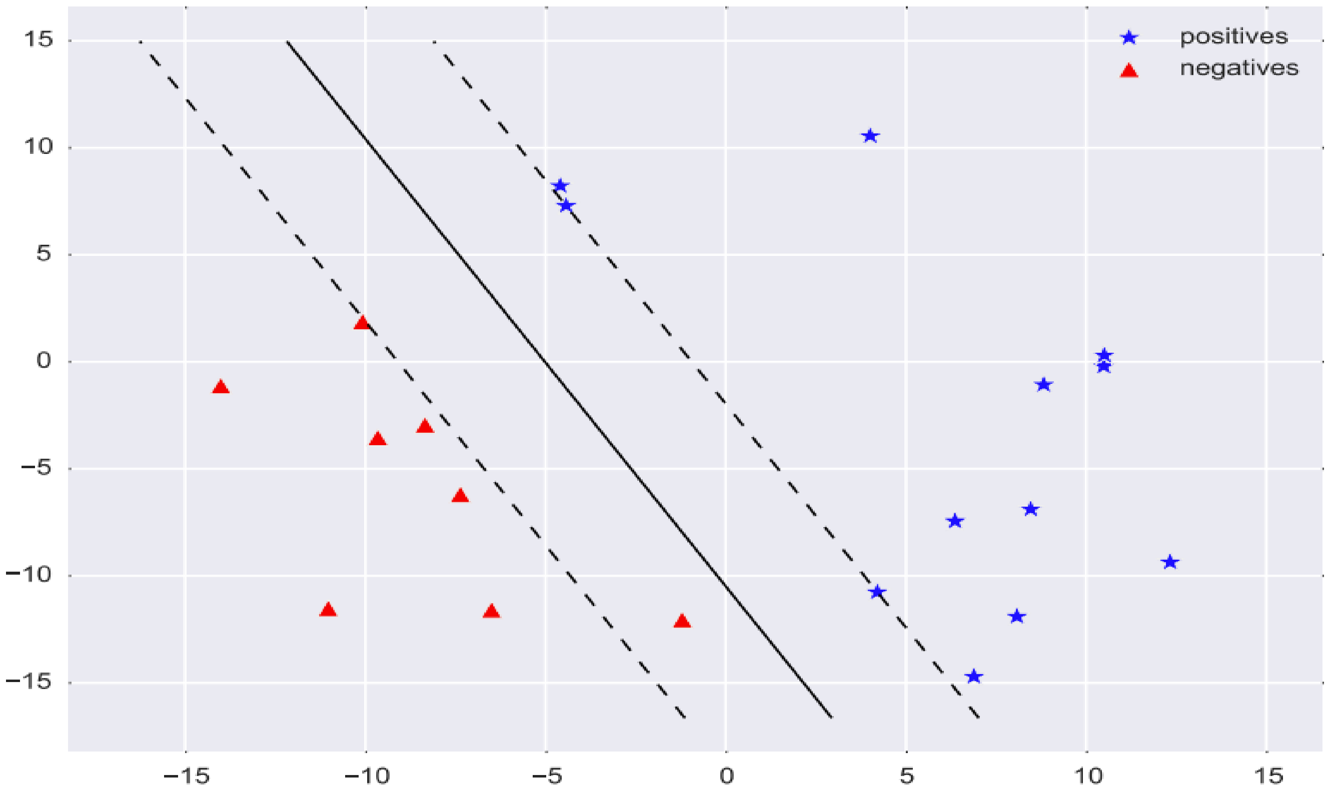
Examples: a separable dataset



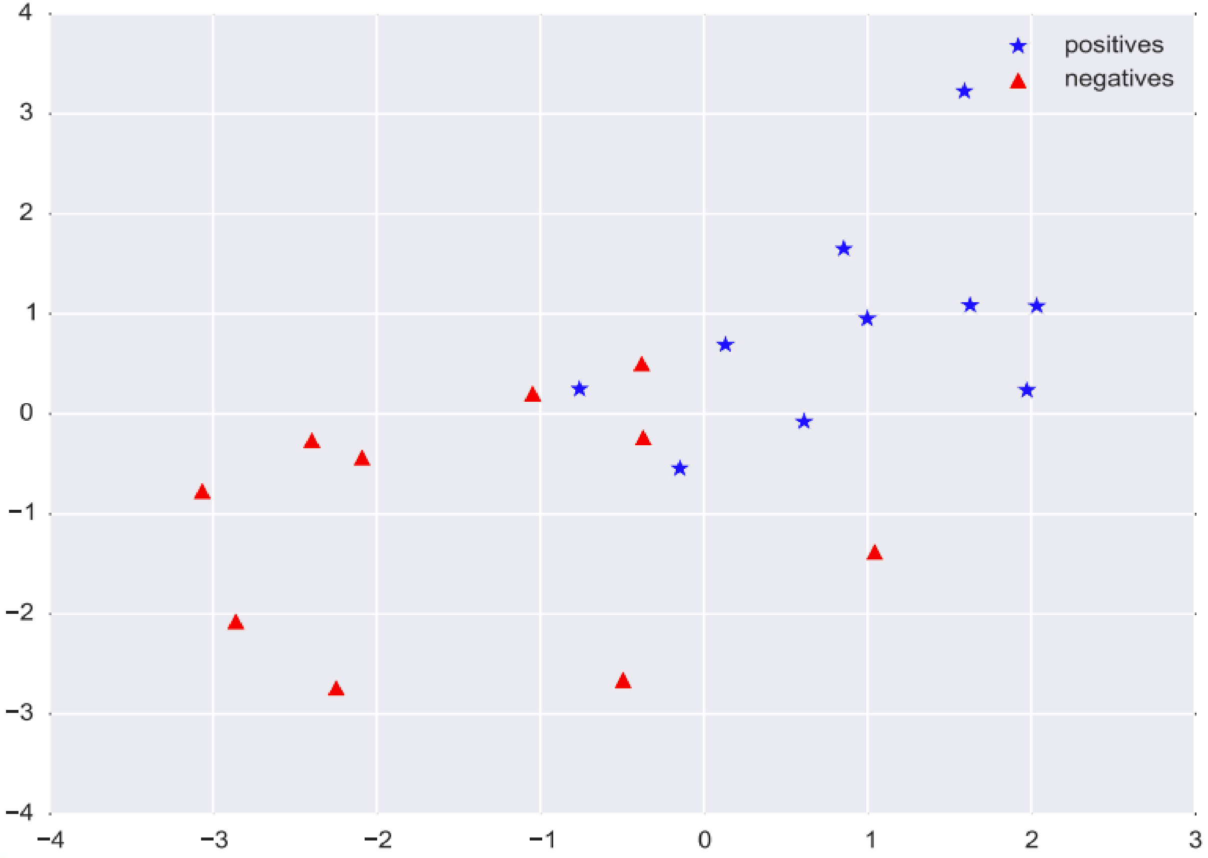
Examples: soft-margin SVM with $c = 1$



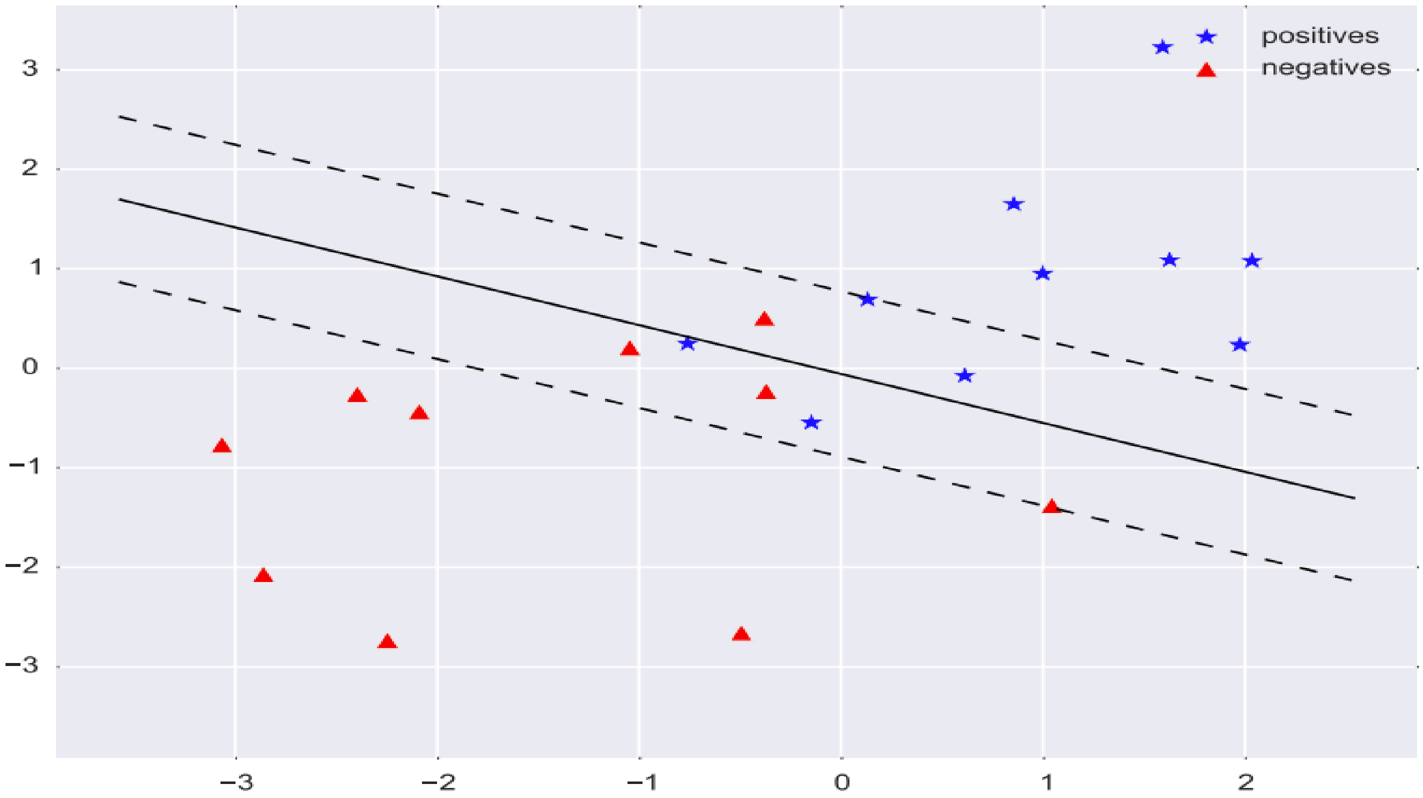
Examples: soft-margin SVM with c = 0.01



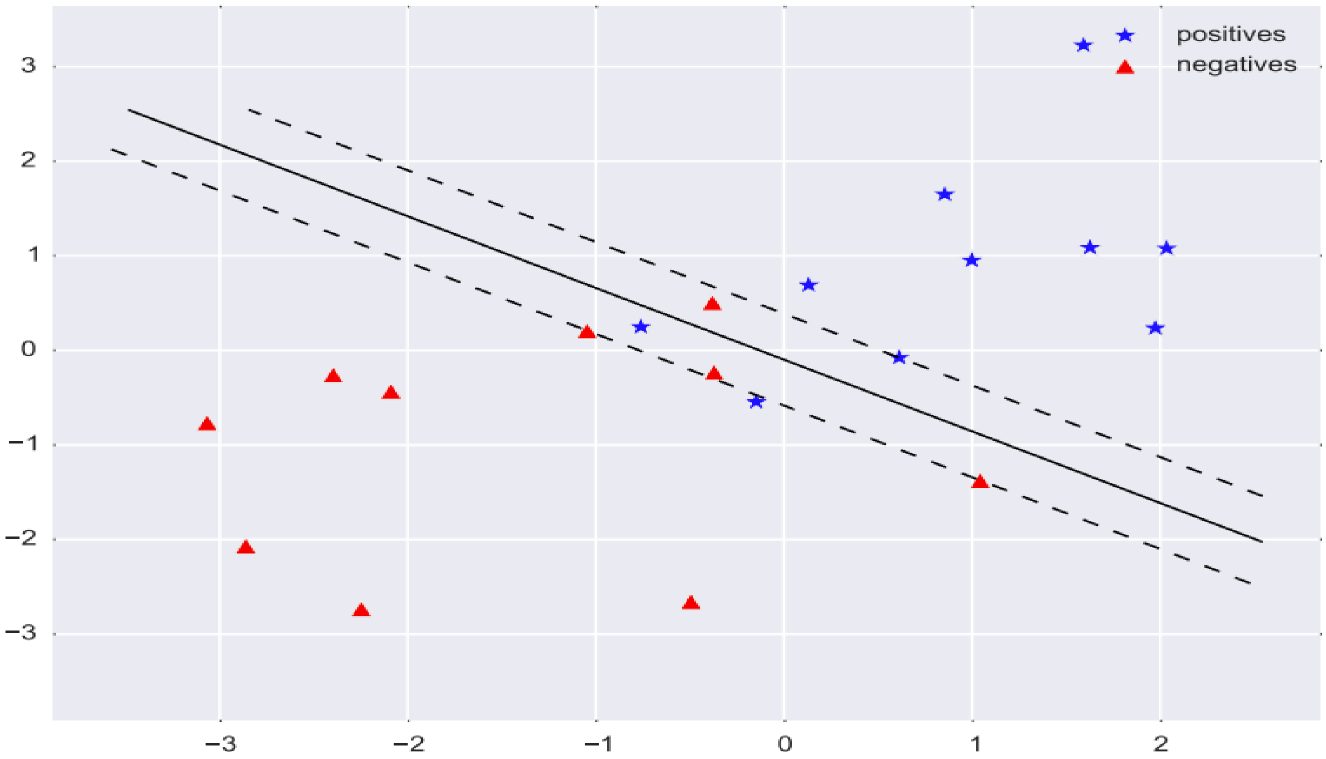
Examples: a non-separable dataset



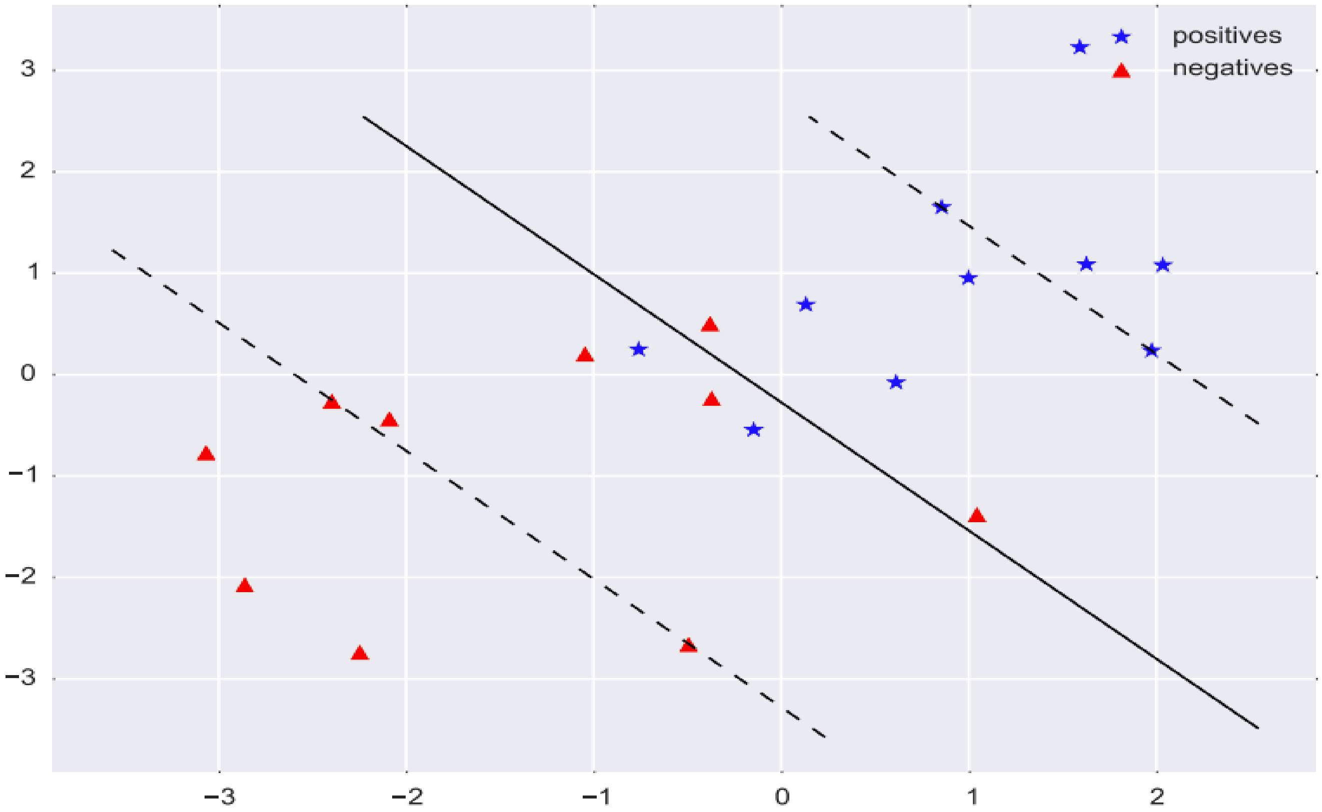
Examples: soft-margin SVM with $c = 1$



Examples: soft-margin SVM with $c = 100$



Examples: soft-margin SVM with c = 0.1



Soft-margin linear SVM in Python

- Available in scikit-learn[4]: sklearn.svm.LinearSVC
- Wrapper around LIBLINEAR[5] (very fast).
- Multi-class support: one-vs-rest (OVR)

```
from sklearn import svm

clf = svm.LinearSVC(C=1.0)
clf.fit(X_train, y_train)

# evaluate accuracy on val set
print(clf.score(X_val, y_val))

# predict on test set
y_pred = clf.predict(X_test)
```

[4] http://scikit-learn.org
[5] http://www.csie.ntu.edu.tw/~cjlin/liblinear/

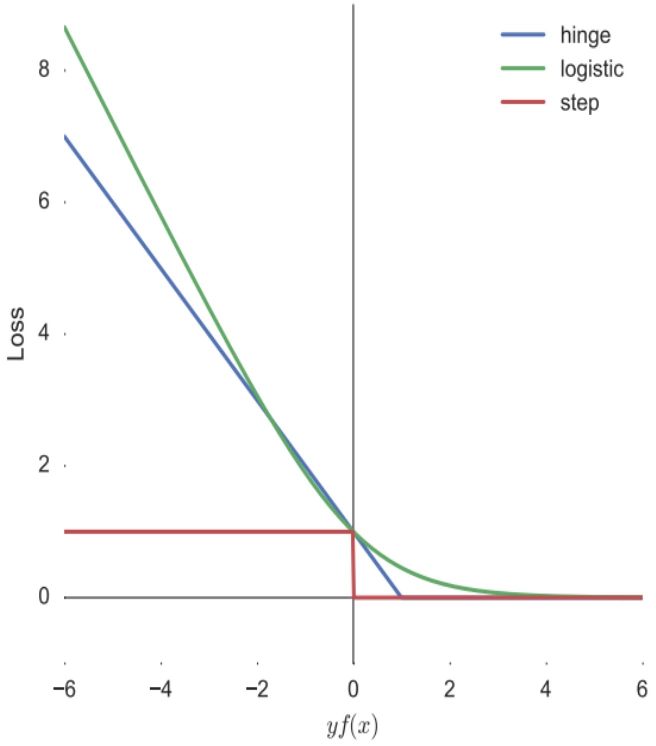
Comparison of linear SVM and logistic regression

Linear SVM uses the hinge loss:

- No penalty for examples on correct side of margin
- Linear penalty for examples that violate margin
- Robust to outliers
- Must use L_2 to maximize margin.

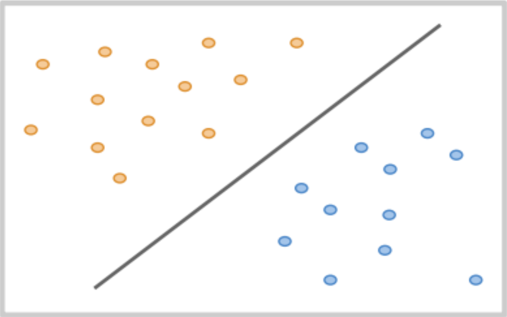
Logistic regression uses logistic loss:

- Penalty for correct examples
- Less robust than hinge to outliers
- Can use L_2 but not necessary
- Naturally produces probabilities

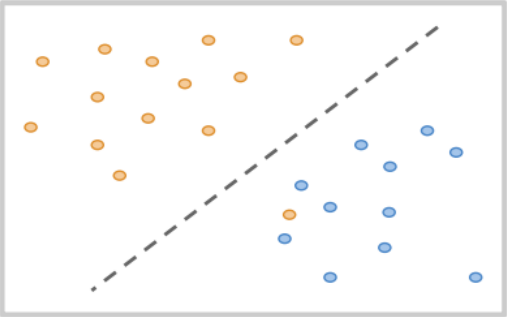


Kernel SVMs

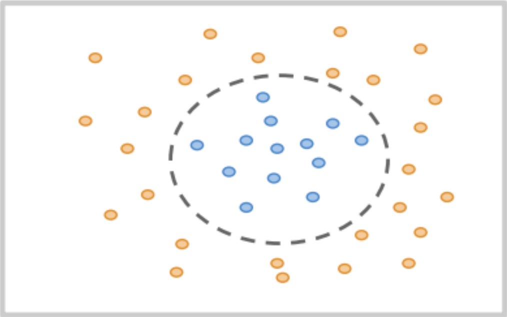
Three problems



Linearly separable



Almost linearly separable



Not linearly separable

Mapping functions

Linear SVM can only produce linear decision boundaries.

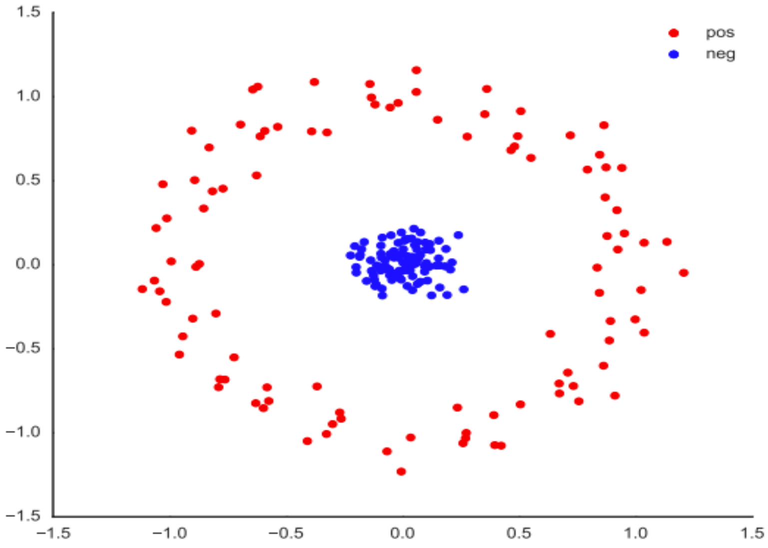
Many interesting classification problems are **not** linearly separable.

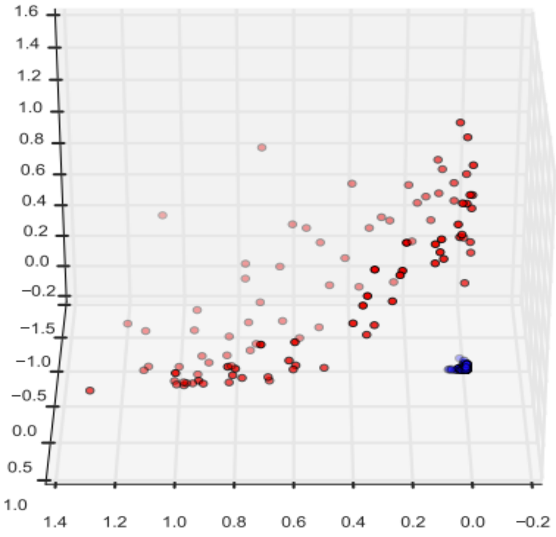
Although data may not be linearly separable in the original feature space, it may be separable in another feature space that we can compute from the original feature space via a mapping function $\varphi(\mathbf{x})$.

For example, for $\mathbf{x} \in \mathrm{R}^2$, we could define $\varphi : \mathrm{R}^2 \rightarrow \mathrm{R}^3$ as:

$$\phi(\mathbf{x}) = \begin{bmatrix} x_1^2 & \sqrt{2}x_1x_2 & x_2^2 \end{bmatrix}$$

Mapping functions





$$\phi(\mathbf{x}) = \begin{bmatrix} x_1^2 & \sqrt{2}x_1x_2 & x_2^2 \end{bmatrix}^T$$

Mapping functions

The decision boundary, while linear in the new space, corresponds to a non-linear boundary when back projected to the original space.

These mapping functions are sometimes called *feature extraction algorithms*.

Many non-linear ML problems can be solved using a good feature extraction algorithm.

But...

- Designing good feature extractors is hard!
- The more features we use, the more parameters we need to fit.
- Large number of features means we need a lot of data to avoid overfitting.

Kernel SVMs

Kernel SVMs offer a way to avoid needing to define an explicit feature mapping function.

Instead of the mapping function, we need to define a *kernel function* $K(\mathbf{x}, \mathbf{z})$ that compares **how similar** the examples $\mathbf{x}$ and $\mathbf{z}$ are.

This formulation allows us to implicitly work in very high dimensional feature spaces without ever actually computing the explicit projections $\varphi(\mathbf{x})$.

To do this, we'll need to reformulate the SVM objective function (loss) and decision function so that they only use inner products between training examples $\mathbf{x}_i^T \mathbf{x}_j$. We can then replace these inner products with kernel evaluations.

Kernels

Kernel design intuition:

- $K(\mathbf{x},\mathbf{z})$ should be large for similar values;
- $K(\mathbf{x},\mathbf{z})$ should be small for different values.

Common kernels:

Linear	$K(\mathbf{x},\mathbf{z}) = \mathbf{x}^T\mathbf{z}$	Same as linear SVM
Polynomial	$K(\mathbf{x},\mathbf{z}) = (1 + \mathbf{x}^T\mathbf{z})^d$	All polynomials up to degree d
Gaussian	$K(\mathbf{x},\mathbf{z}) = \exp(-\gamma \lVert \mathbf{x}-\mathbf{z} \rVert^2)$	Infinite dimensional feature spaces.

- Gaussian kernel also known as Radial Basis Function (**RBF**)

So, what has the kernel trick gotten us?

- No need to define feature mapping function, only the kernel.
- Number of parameters doesn’t depend on dimension of features.
- Reduce overfitting (number of parameters $\leq$ number of data points).
- Can work in very high dim (even infinite) feature spaces.

But...

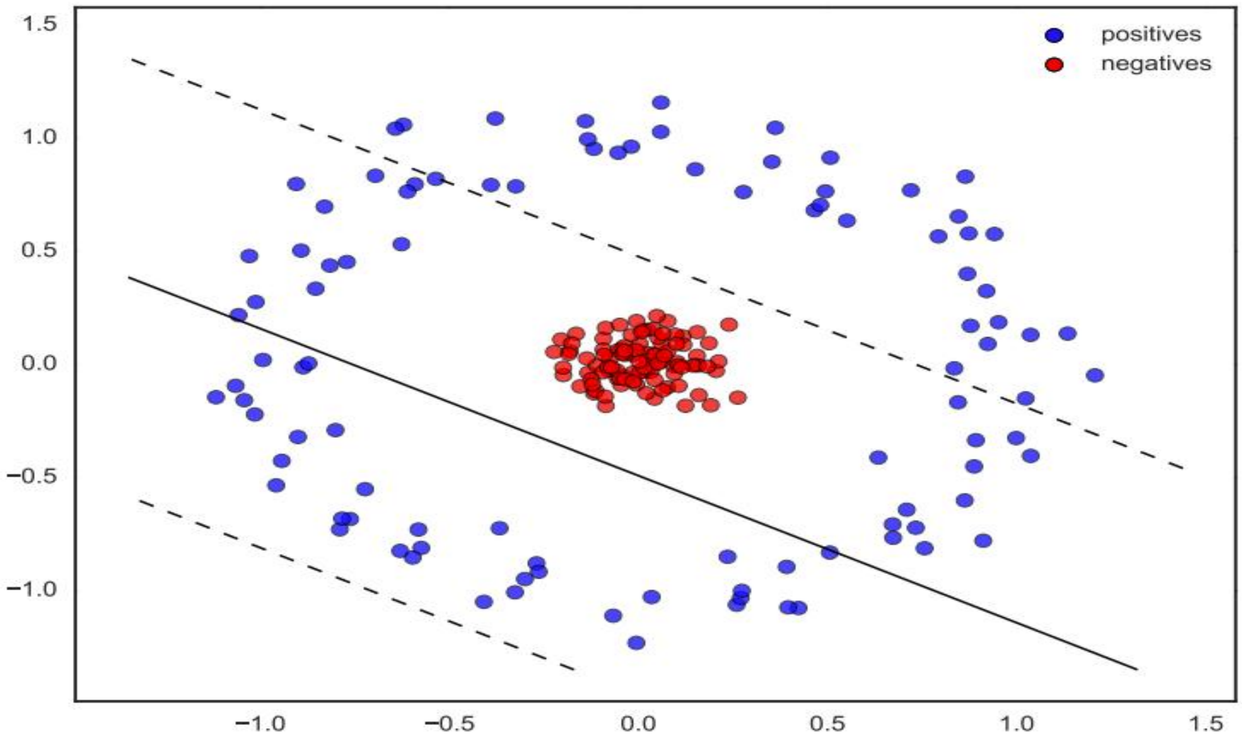
- We still need to design appropriate kernel functions. This is difficult.
- It is slower to solve the kernel SVM than the linear SVM when we have many data points (big N).
- Sometimes it’s more efficient to do an explicit feature mapping and solve the linear SVM.

Kernel SVM in Python

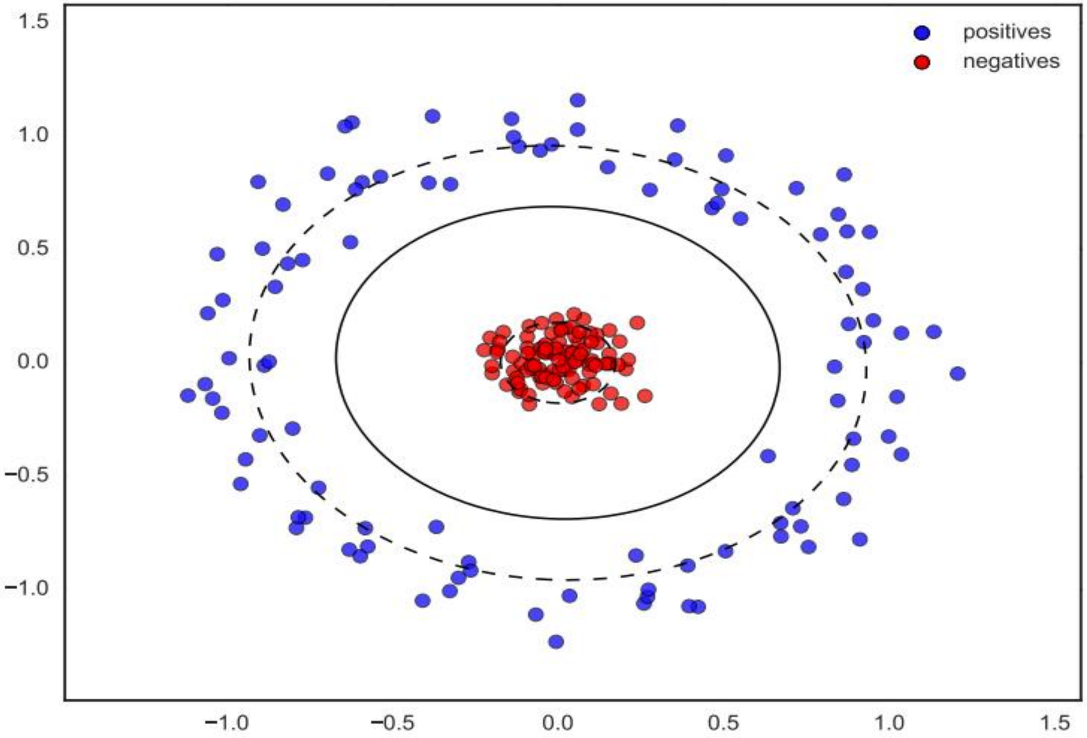
Can use sklearn.svm.SVC

```
import numpy as np
from sklearn import svm

# generate some data
X= np.random.randn(300, 2)
Y= np.logical_xor(X[:, 0] > 0, X[:, 1] > 0)

# fit the model
clf = svm.SVC(kernel='rbf')
clf.fit(X, Y)
```

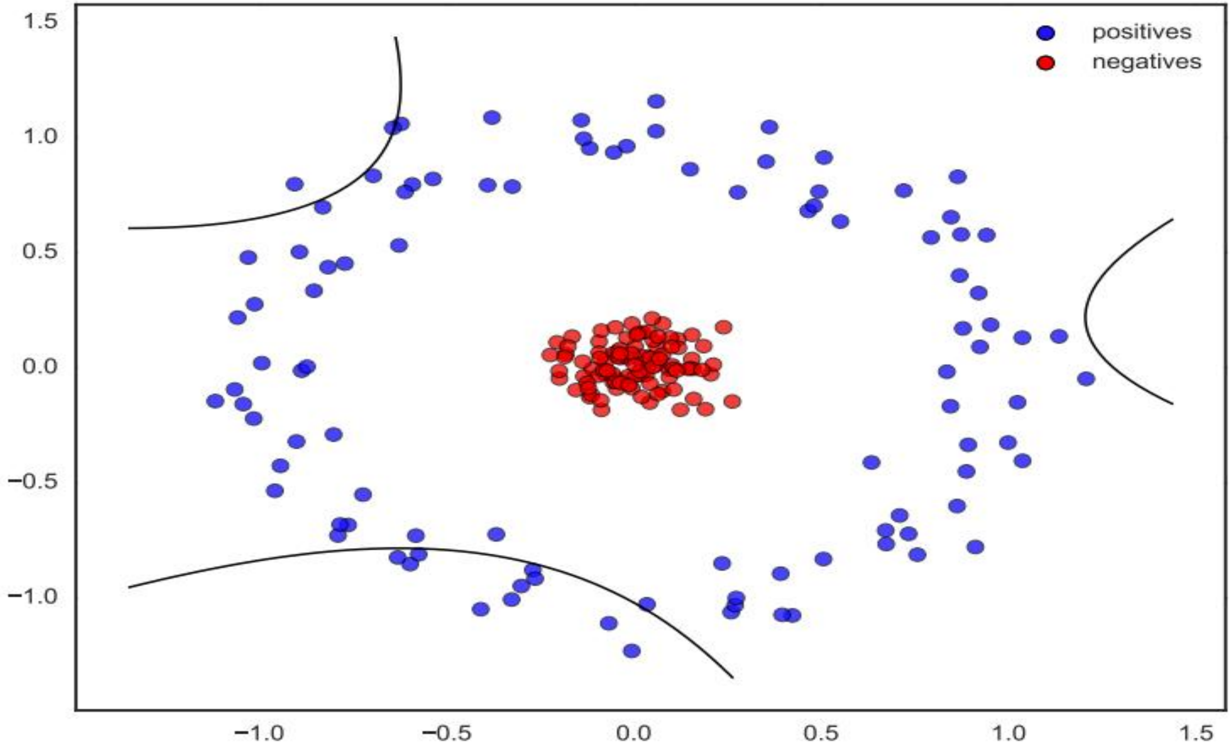
Example:Soft margin linear SVM with C=1.0



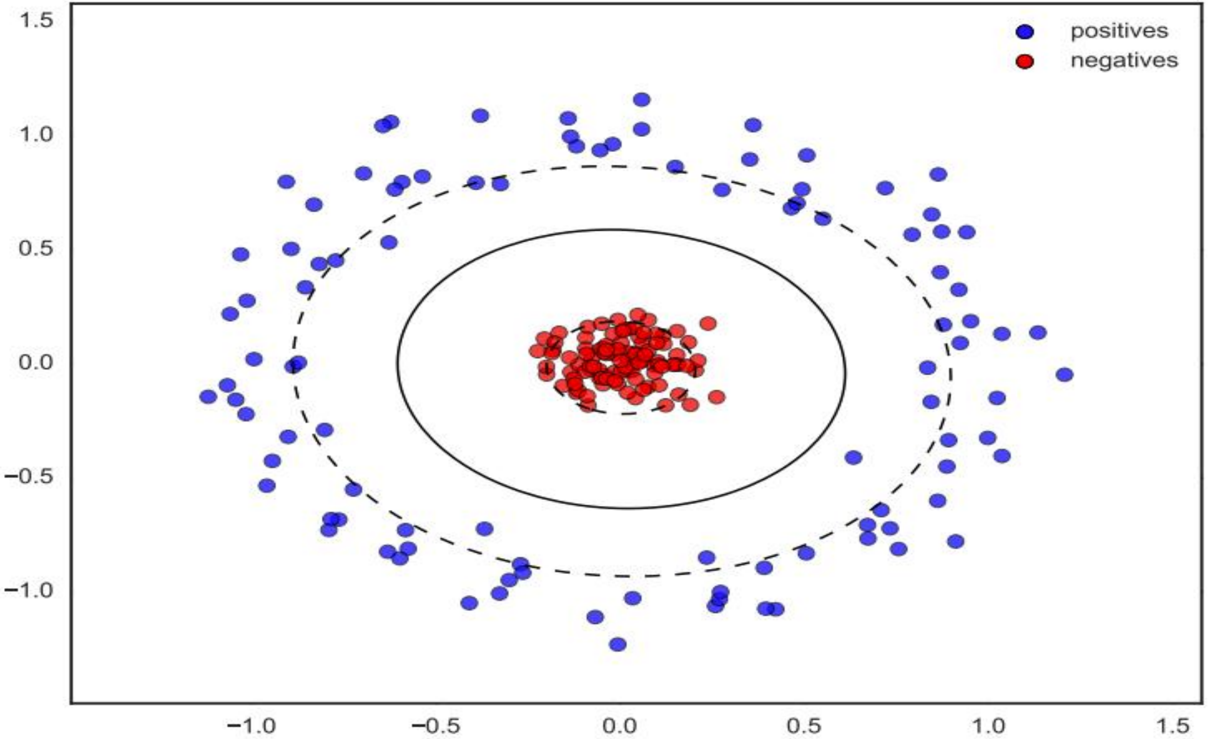
Example: SVM (C=1) with degree-2 polynomial kernel



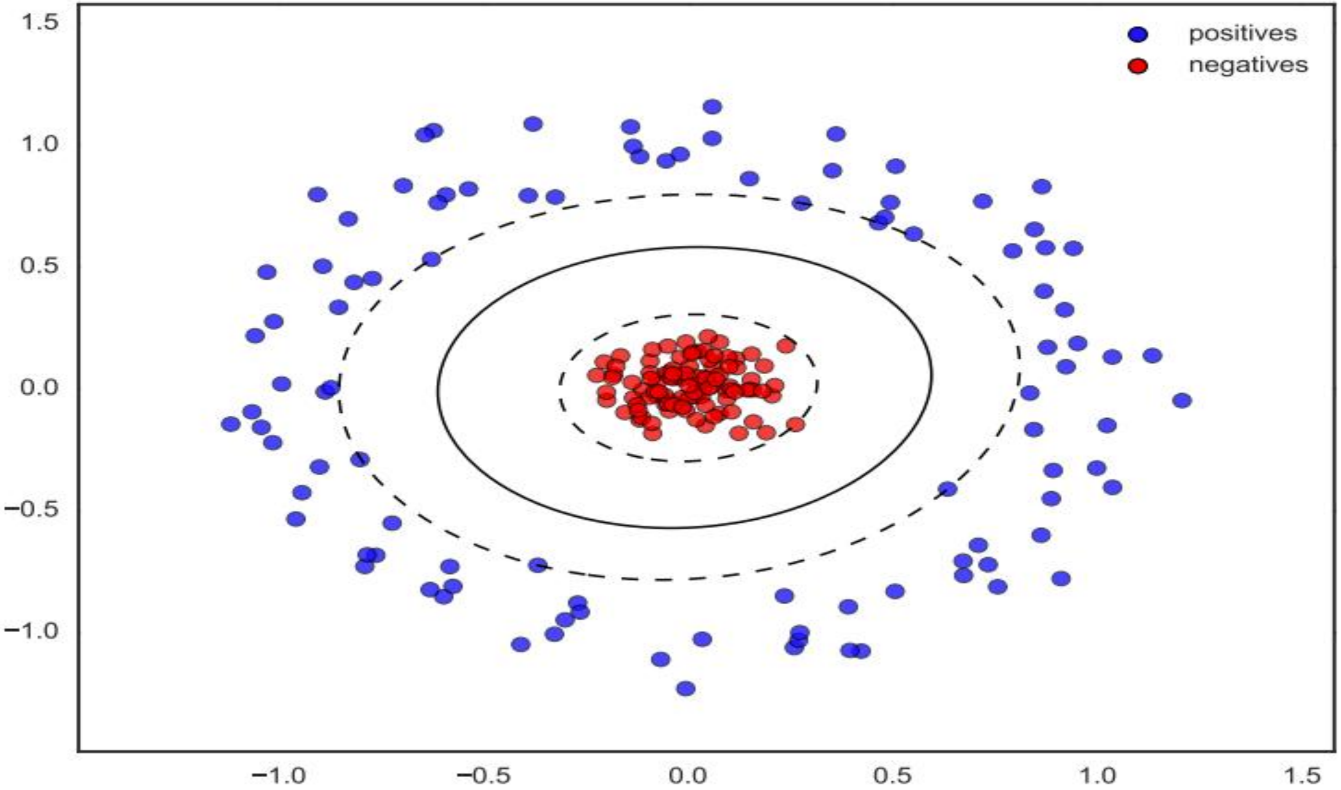
Example: SVM (C=1) with degree-3 polynomial kernel



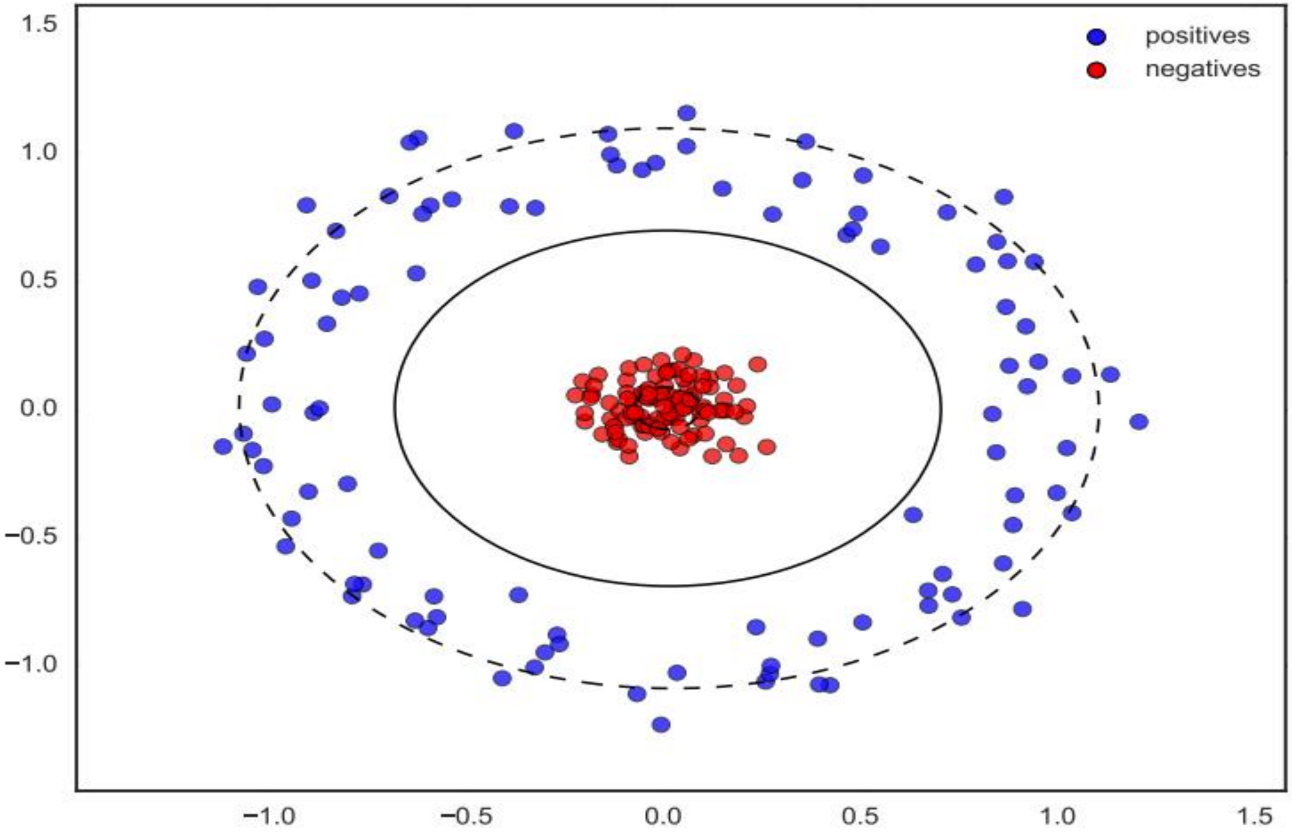
Example: RBF SVM with C = 1



Example: RBF SVM with C = 100



Example: RBF SVM with C = 0.1



Tips for use

Scaling data

The SVM is not scale invariant! Good idea to try **normalizing** your data. Common strategies:

- standardize data to have $\mu = 0$ and $\sigma = 1$.
- min-max scale from to range $[0, 1]$ or $[-1, +1]$

Fit normalization parameters (e.g. min and max or $\mu = 0$ and $\sigma = 1$) **on the training data**. Apply same normalization to validation and test data.

The support vectors

- The support vectors are the training vectors $\mathbf{x}_i$ for which the corresponding α_i is non-zero.
- You do not need to keep the entire training set to predict. Only the support vectors.
- The number of support vectors is typically a small fraction of the training set (only the values that touch/violate the margin).
- This limits the effective number of parameters (α_i), which helps prevent overfitting.

Kernels and hyperparameters

- Kernel SVMs:
  - ➢ typically work well when you have $O(10,000)$ or fewer training examples.
  - ➢ 1M+ training examples makes training very slow.
  - ➢ kernel needs to be evaluated at least once for all pairs of training examples $O(N^2)$.
  - ➢ training a kernel SVM has approx $O(N^2D)$ complexity (SMO algorithm).
  - ➢ the RBF kernel is a good default for many tasks.
- Hyperparameters (C, γ):
  - ➢ try grid search or *random search* on an exponential scale (log space, e.g. $C = \{0.01, 0.1, 1.0, 10, 100\}$).
  - ➢ optimize on a **validation** set, or via **cross-validation**. Never on the test set!

Linear SVMs

- Try first as a baseline!
- Much faster to train, approx $O(ND)$ for LIBLINEAR.
- Can be trained out-of-core using SGD (Pegasos) on very large datasets.
- Very fast at predict time $\mathbf{w}^T\mathbf{x} + b$.
- Often faster to explicitly compute a feature map $\varphi(\mathbf{x})$ and use linear SVM. Also possible to approximate kernels with explicit feature maps.
- Possible to use with deep learning and backprop.

Further reading

- Hastie et al., The Elements of Statistical Learning, Chapter 12 –
  - http://statweb.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf
- Andrew Ng’s Lecture Notes
  - http://cs229.stanford.edu/notes/cs229-notes3.pdf
- Bishop, Pattern Recognition and Machine Learning, Chapter 6, 7