Support Vector Machines

Week 4

February 12, 2024

Spring 24 | CUSP-GX 7033 – Machine Learning for Cities | Dr. Anton Rozhkov

Today's Outline

- From Linear to Non-Linear Classifiers with Support Vector Machines
- Linear decision boundaries
- Support vector machines (SVMs) for classification
- Moving from linear to non-linear decision boundaries with kernel SVM
- Lab: SVM in Python

Support Vector Machines

From Linear to Non-Linear Classifiers

Pre 1980: Almost all learning methods learned linear decision surfaces.

Linear learning methods have nice theoretical properties

1980's: Decision trees and NNs allowed efficient learning of non-linear decision surfaces.

Little theoretical basis, and all suffer from local minima

Starting from 1990's: Efficient learning algorithms for non-linear functions based on computational learning theory developed.

Nice theoretical properties.

1980

1990

NYU

Source: Berwick, An Idiot's guide to Support vector machines (SVMs)

Support Vector Machines (SVMs)

Support vector machines are an optimization-based prediction approach used primarily for binary classification and are able to achieve state-of-the-art prediction accuracy on many real-world tasks.

Key idea 1: Learn a **decision boundary** that optimally separates positive and negative training examples. (But what does it mean to be optimal?)

Key idea 2: Learn a **linear** decision boundary in high dimensional space corresponding to a **non-linear** decision boundary for the original problem.

SVM assumes **real-valued** attributes on the **same scale**. Thus, it is very important to pre-process your data before training the model:

- Normalize real-valued attributes (scale either to [0,1] or to mean = 0 and variance = 1). Make sure to use the same scaling for training and test data.
- Replace discrete-valued attributes with **dummy variables**.

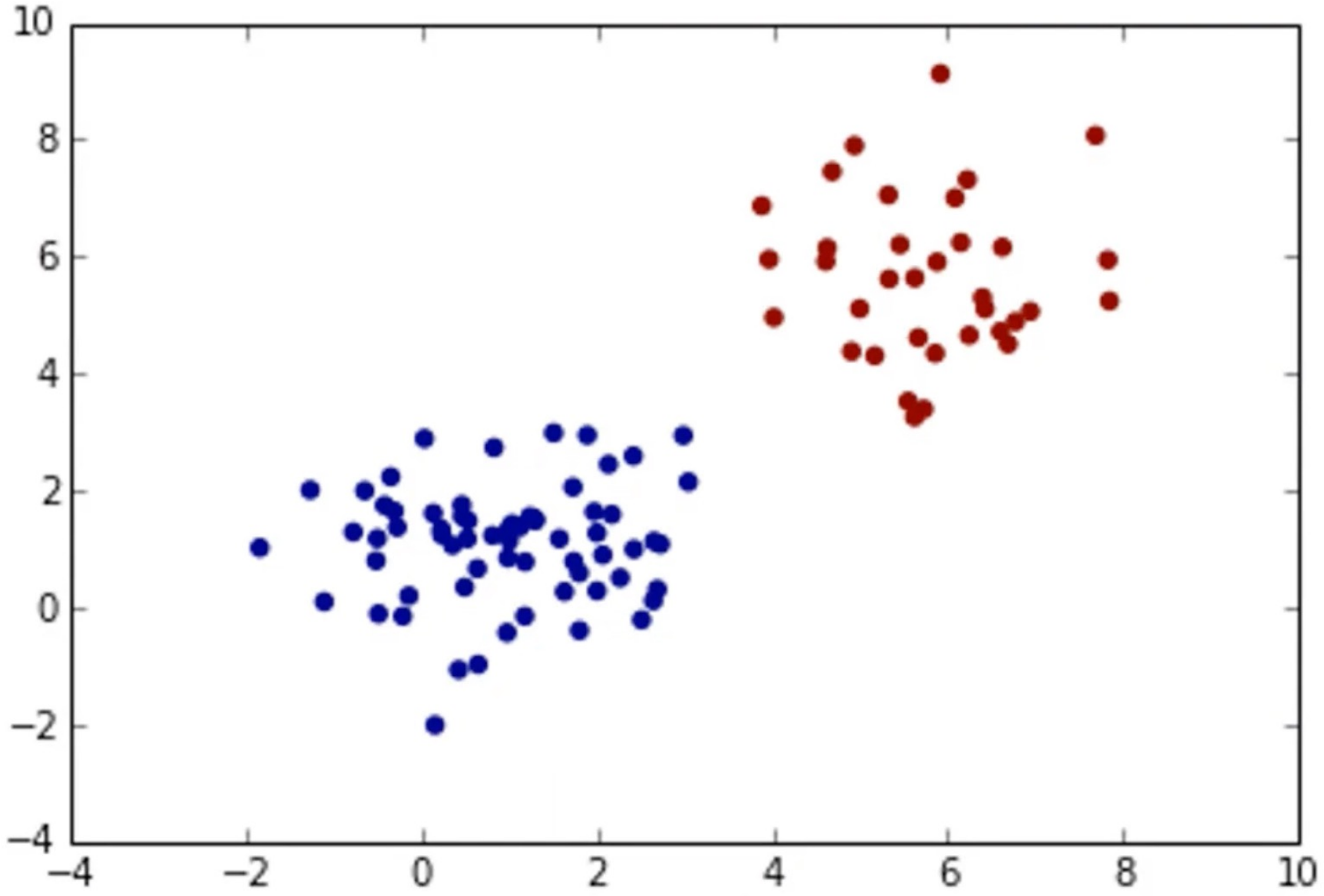
Car	Weight
1	Low
2	Medium
3	Heavy

➡

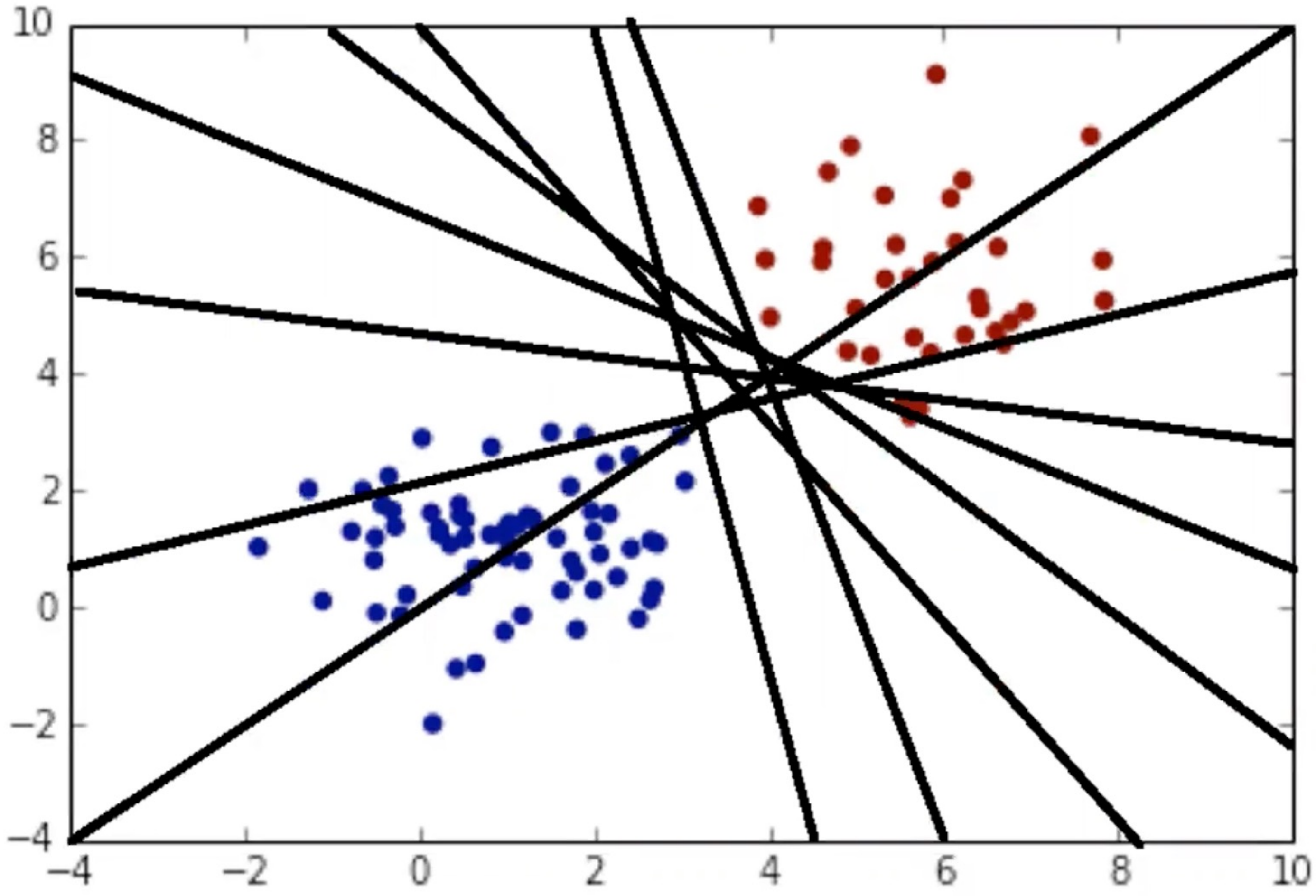
Car	Weight=Medium	Weight=Heavy
1	0	0
2	1	0
3	0	1

NYU

SVMs: Basic Idea (Linearly Separable Case)



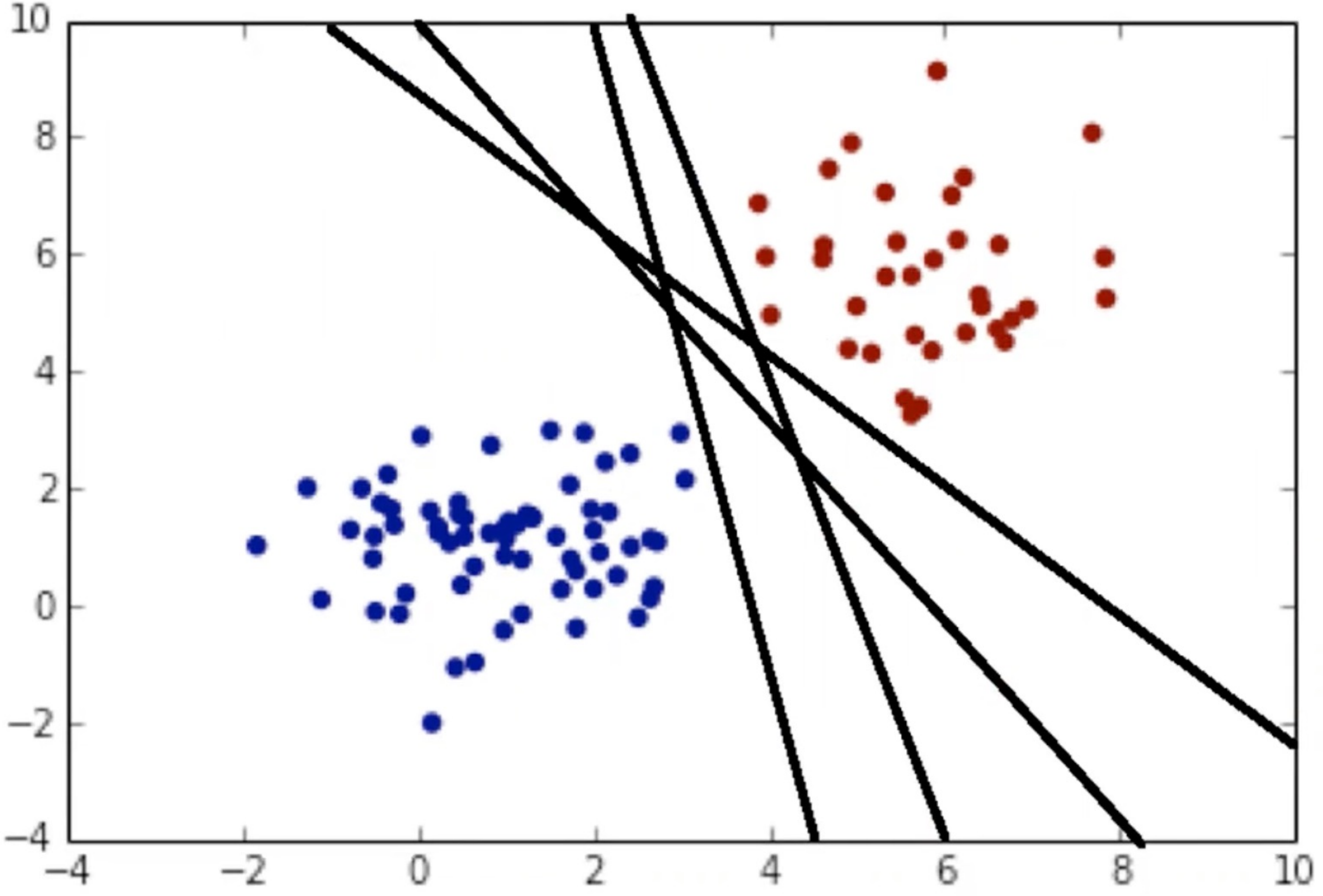
SVMs: Basic Idea (Linearly Separable Case)



In general, there are many possible solutions (an infinite number!)

SVM finds an optimal solution

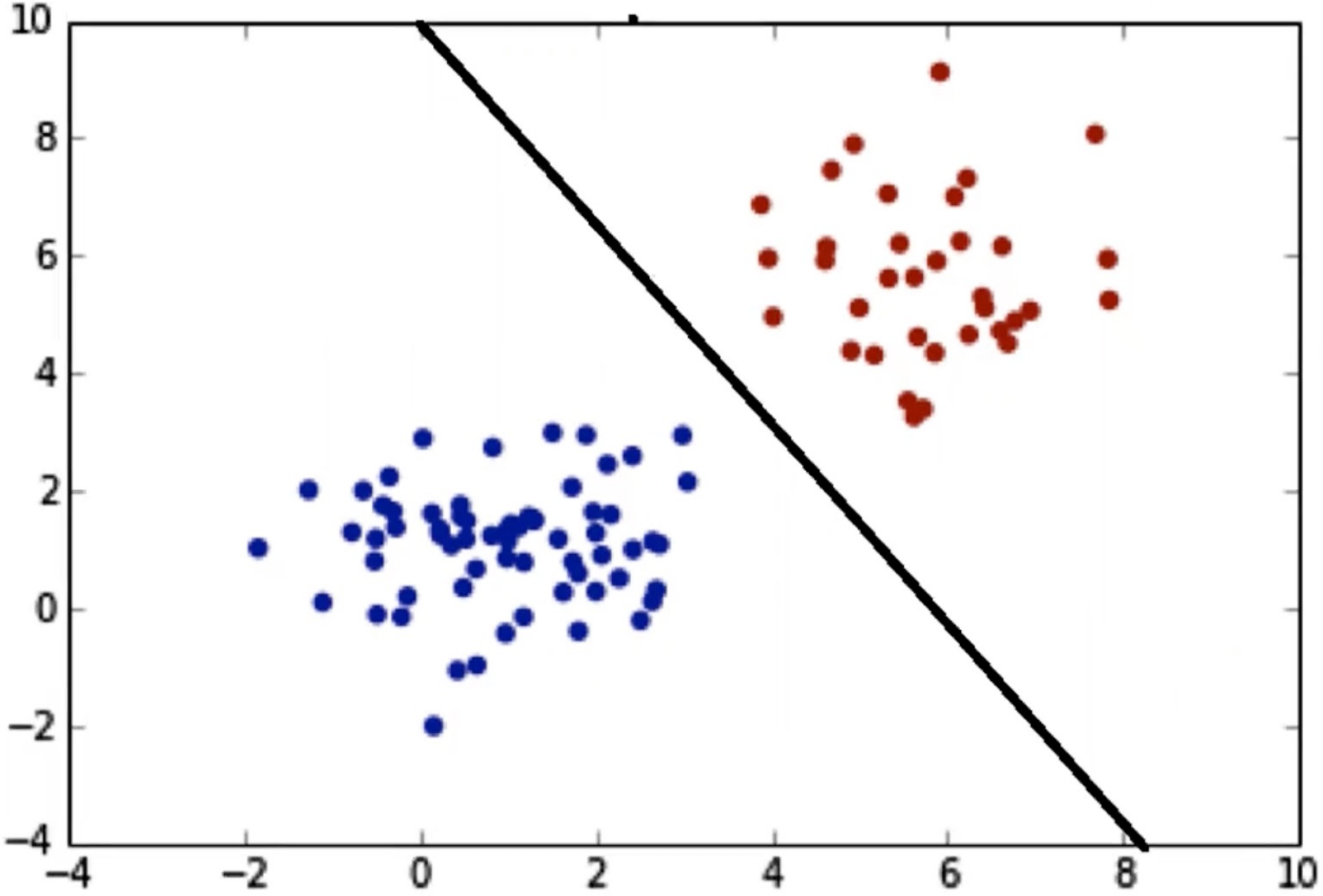
SVMs: Basic Idea (Linearly Separable Case)



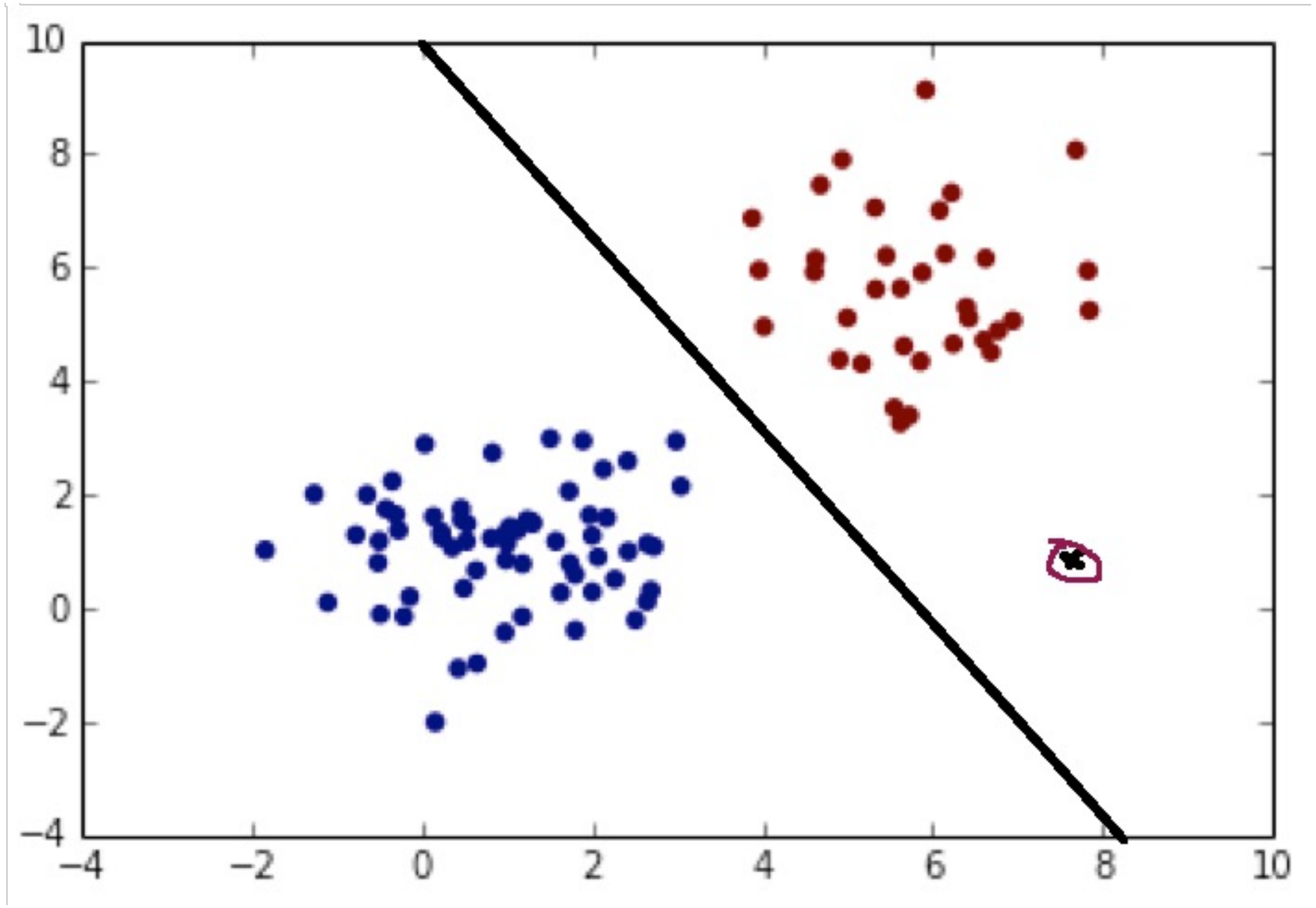
In general, there are many possible solutions (an infinite number!)

SVM finds an optimal solution

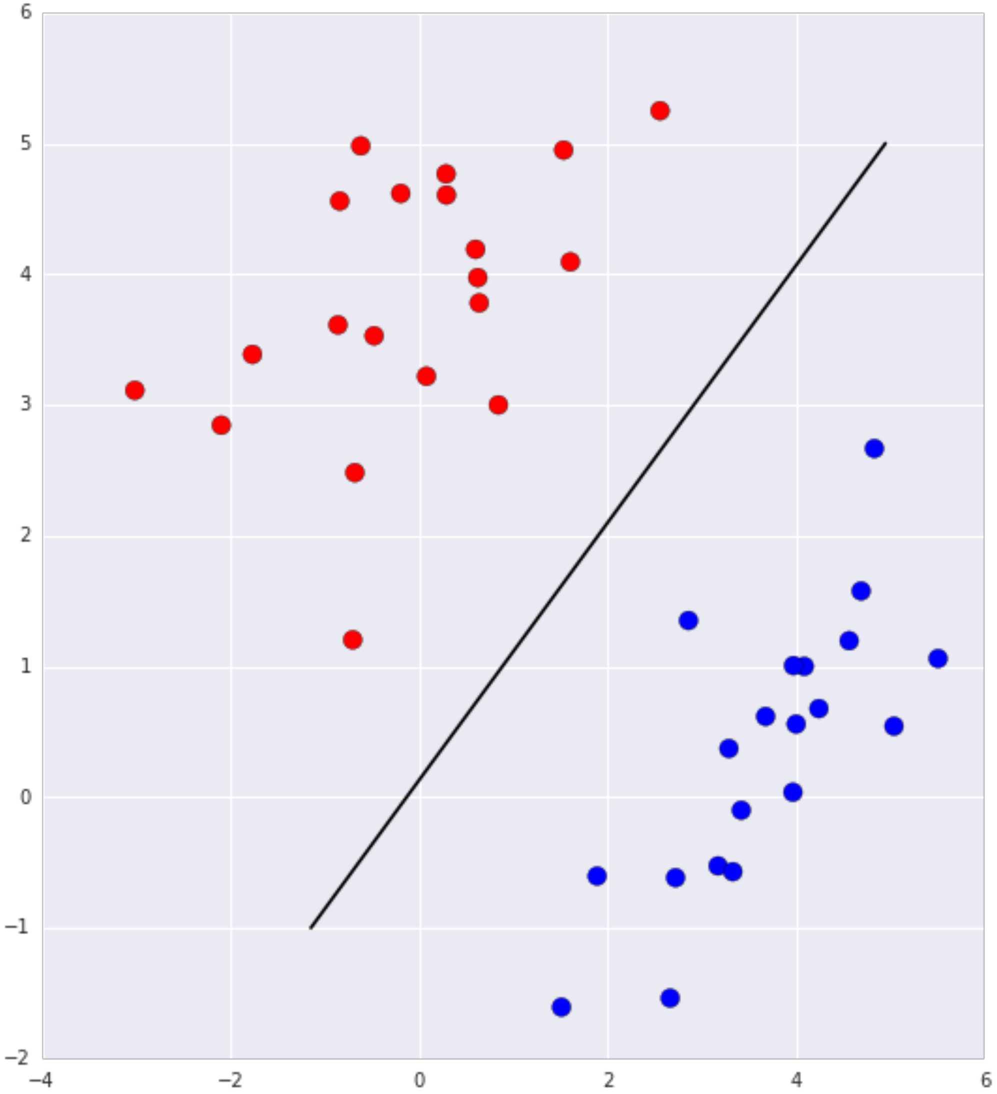
SVMs: Basic Idea (Linearly Separable Case)



SVMs: Basic Idea (Linearly Separable Case)



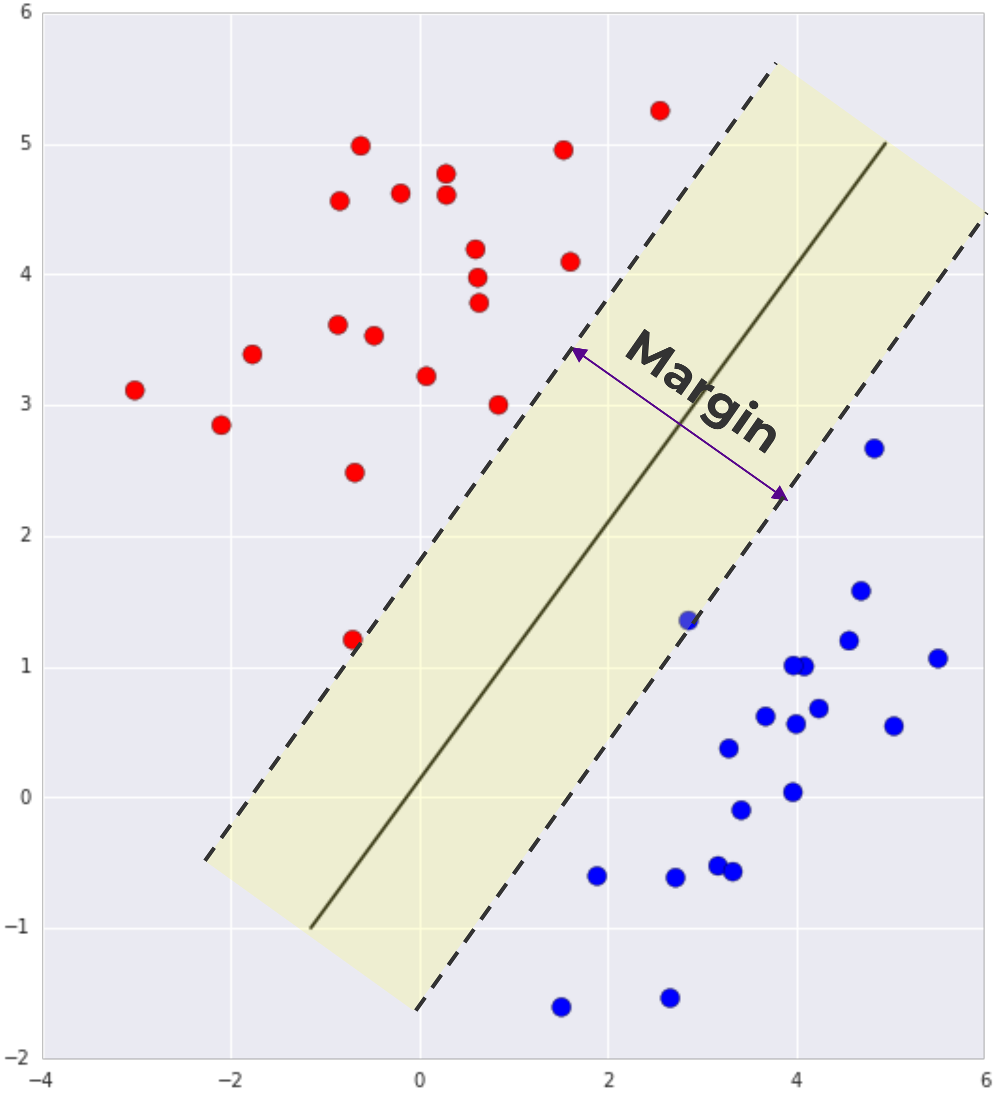
Which Line to Choose?



This is an optimization problem

NYU

Which Line to Choose?

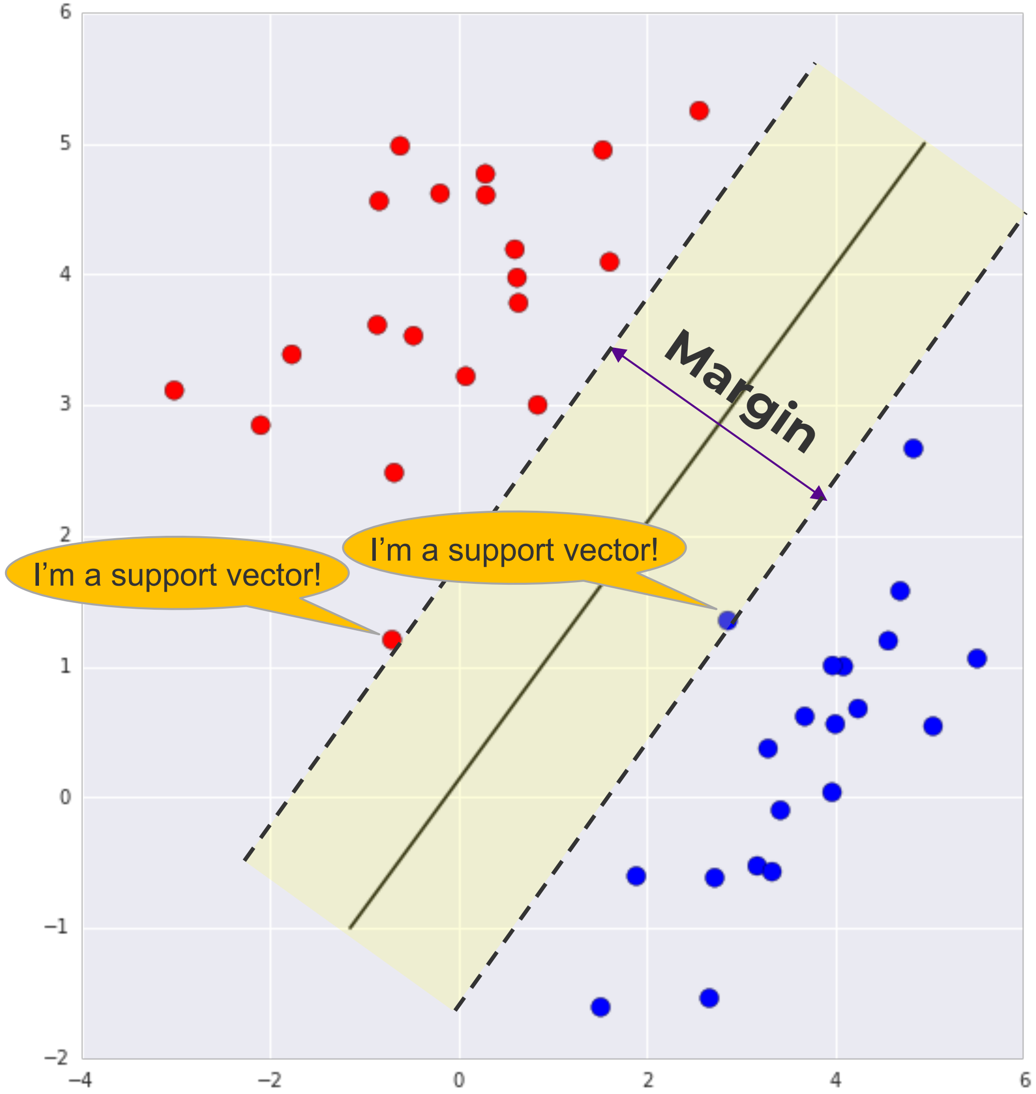


Choose the line that **maximizes** the margin between classes.

Margin = how wide we could make the linear decision boundary before it contacts points from either class.

NYU

Which Line to Choose?



Points on the margin are called **support vectors**.

The classifier can be defined entirely by the set of support vectors.

This fact has lots of useful implications:

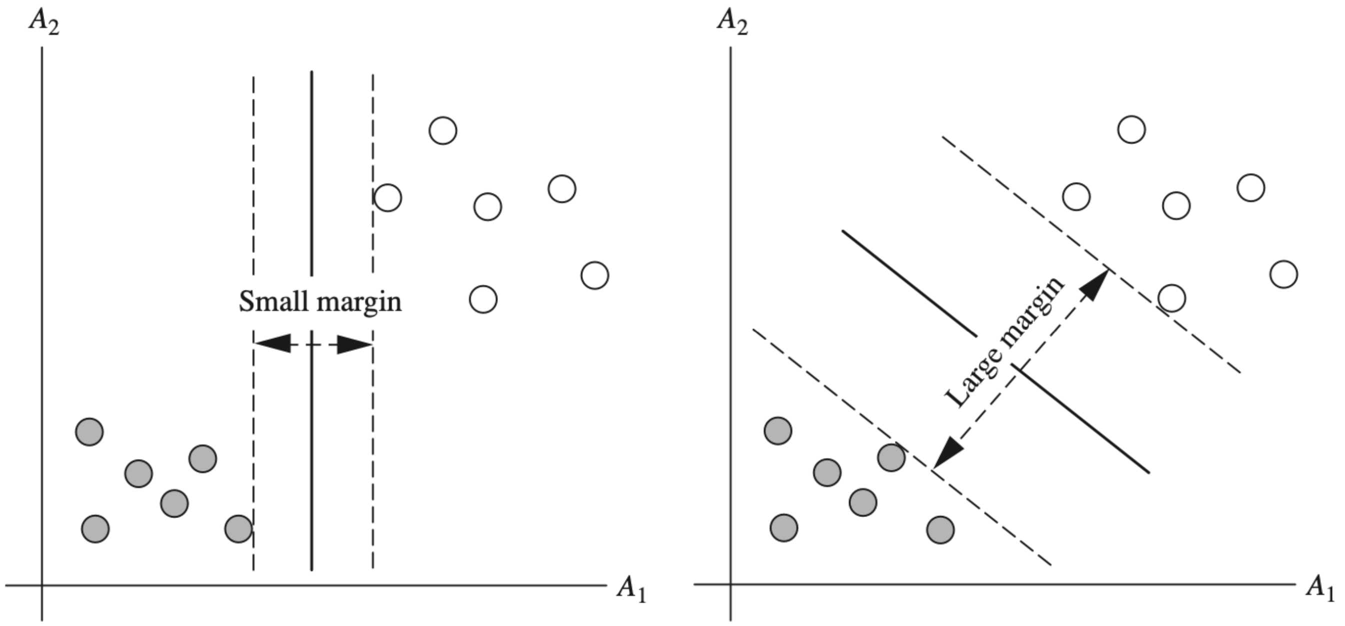
- Fast classification of test points.
- Fast leave-one-out cross-validation.
- Faster, but still expensive, training.

NYU

Which Line to Choose?

Why is maximizing the margin
a good idea?

1. Intuitively, this feels safest.
2. If we've made a small error in the location of the boundary, this gives us the least chance of causing a misclassification.
3. Backed up by statistical learning theory → there are provable bounds on generalization error.
4. Empirically, it works very well.



NYU

How to Maximize the Margin?

To separate, for all points j, we must have:

$$y_j(x_j^T w + b) > 0$$

For the margin, define:

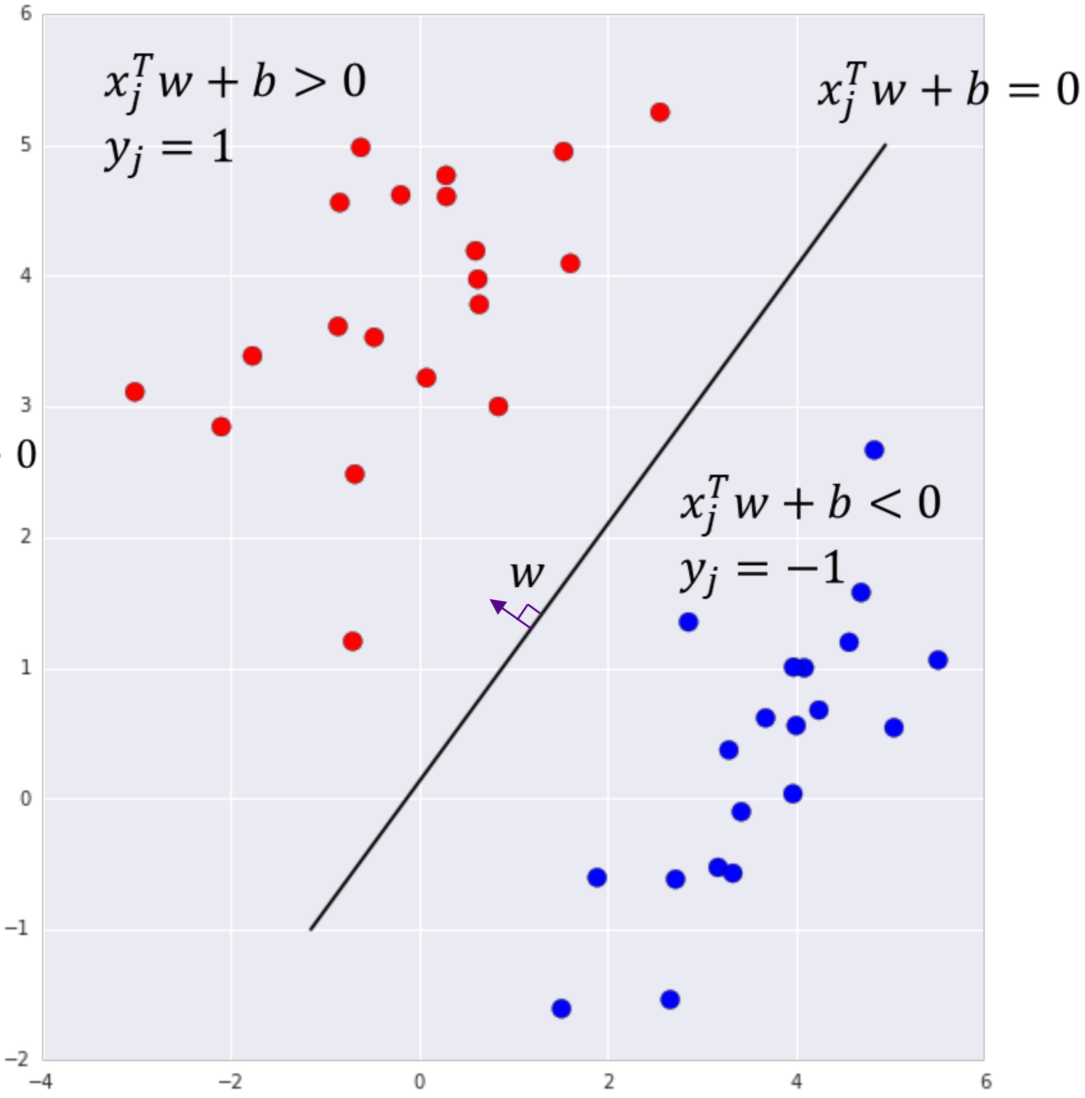
$$M = \min_j y_j\left(x_j^T w + b\right) > 0$$

Then for y_j = 1, we have:

$$x_j^T w + b \geq M$$

For y_j = -1, we have:

$$x_j^T w + b \leq -M$$



Decision boundary (separating line or hyperplane)

Represent each point as (xj, yj), where yj, the class value we are trying to predict, is +1 or -1. Note that xj is a vector of length 2 in this example.

NYU

How to Maximize the Margin?

To separate, for all points j, we must have:

$$y_j(x_j^T w + b) > 0$$

For the margin, define:

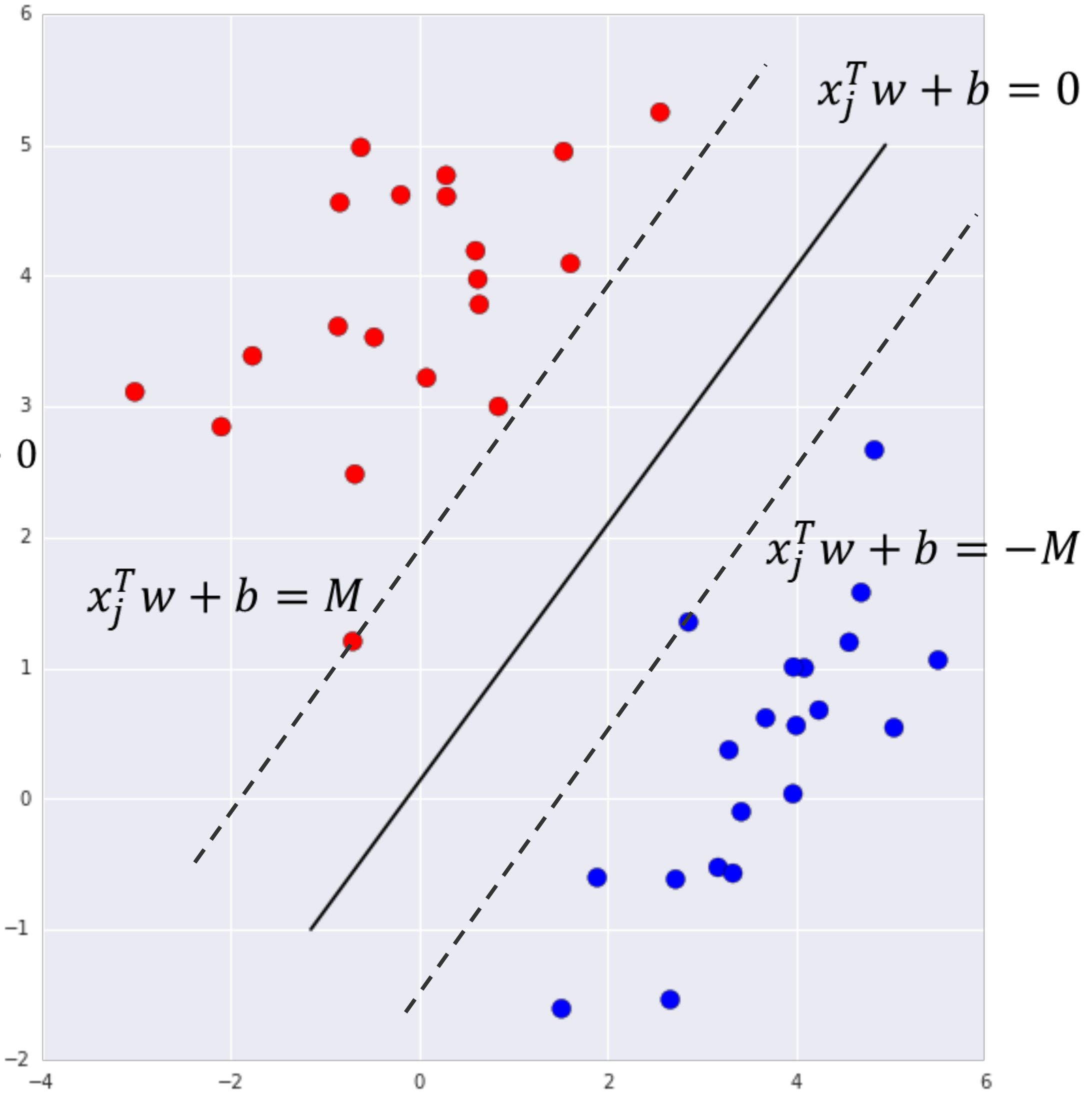
$$M = \min_j y_j\left(x_j^T w + b\right) > 0$$

Then for y_j = 1, we have:

$$x_j^T w + b \geq M$$

For y_j = -1, we have:

$$x_j^T w + b \leq -M$$



Decision boundary (separating line or hyperplane)

Represent each point as (xj, yj), where yj, the class value we are trying to predict, is +1 or -1. Note that xj is a vector of length 2 in this example.

How to Maximize the Margin?

Margin = 2M / ||w||.

This follows from computing the distance between parallel lines.

Goal: maximize 2M / ||w|| subject to constraints, for all j:

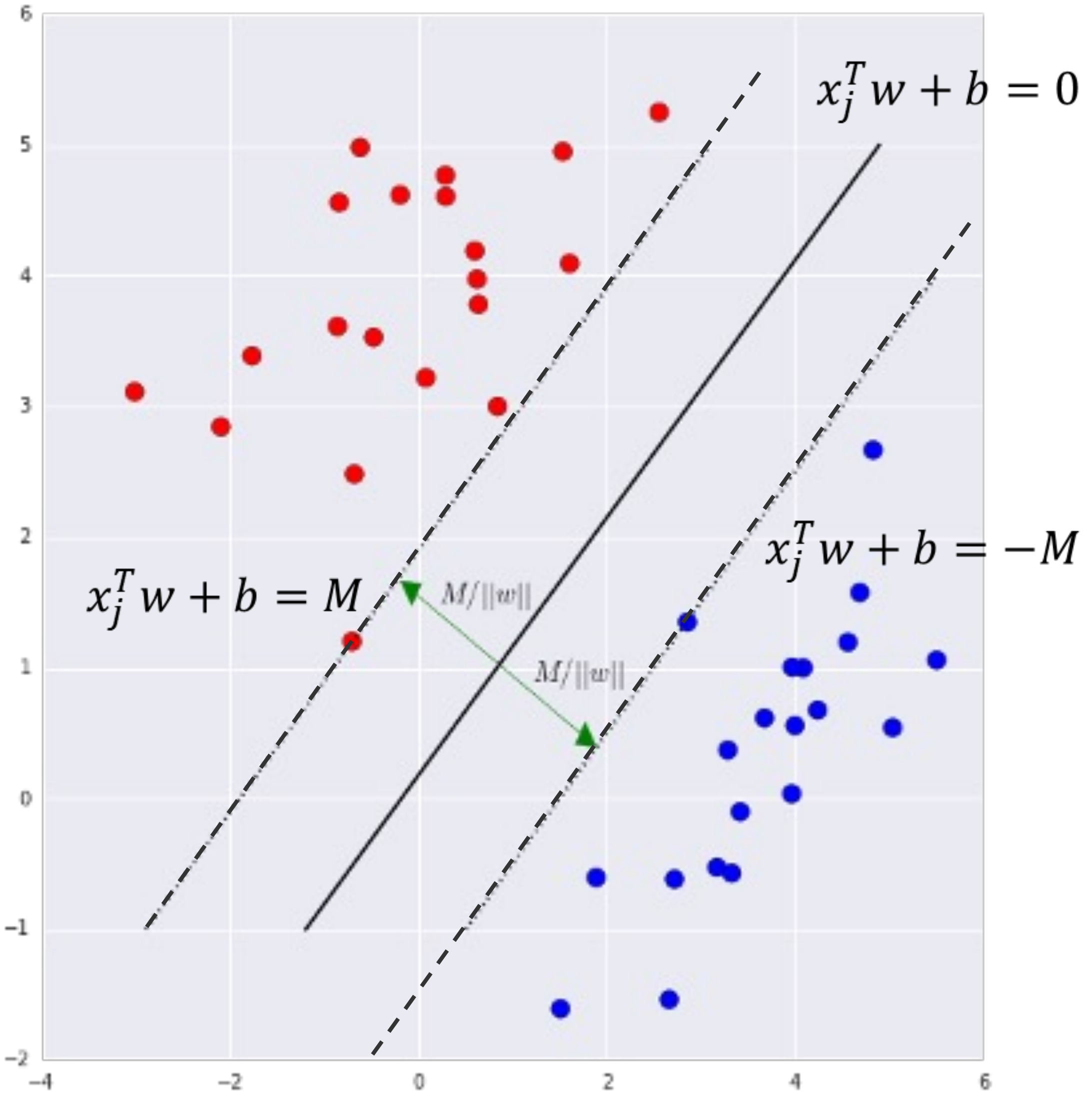
$$y_j(x_j^T w + b) \geq M$$

Simplify by change of variables, dividing w and b through by M.

New goal: minimize ||w|| subject to constraints, for all j:

$$y_j(x_j^T w + b) \geq 1$$

New margin: 2 / ||w||



Decision boundary (separating line or hyperplane)

NYU

How to Maximize the Margin?

Margin = 2M / ||w||.

This follows from computing the distance between parallel lines.

Goal: maximize 2M / ||w|| subject to constraints, for all j:

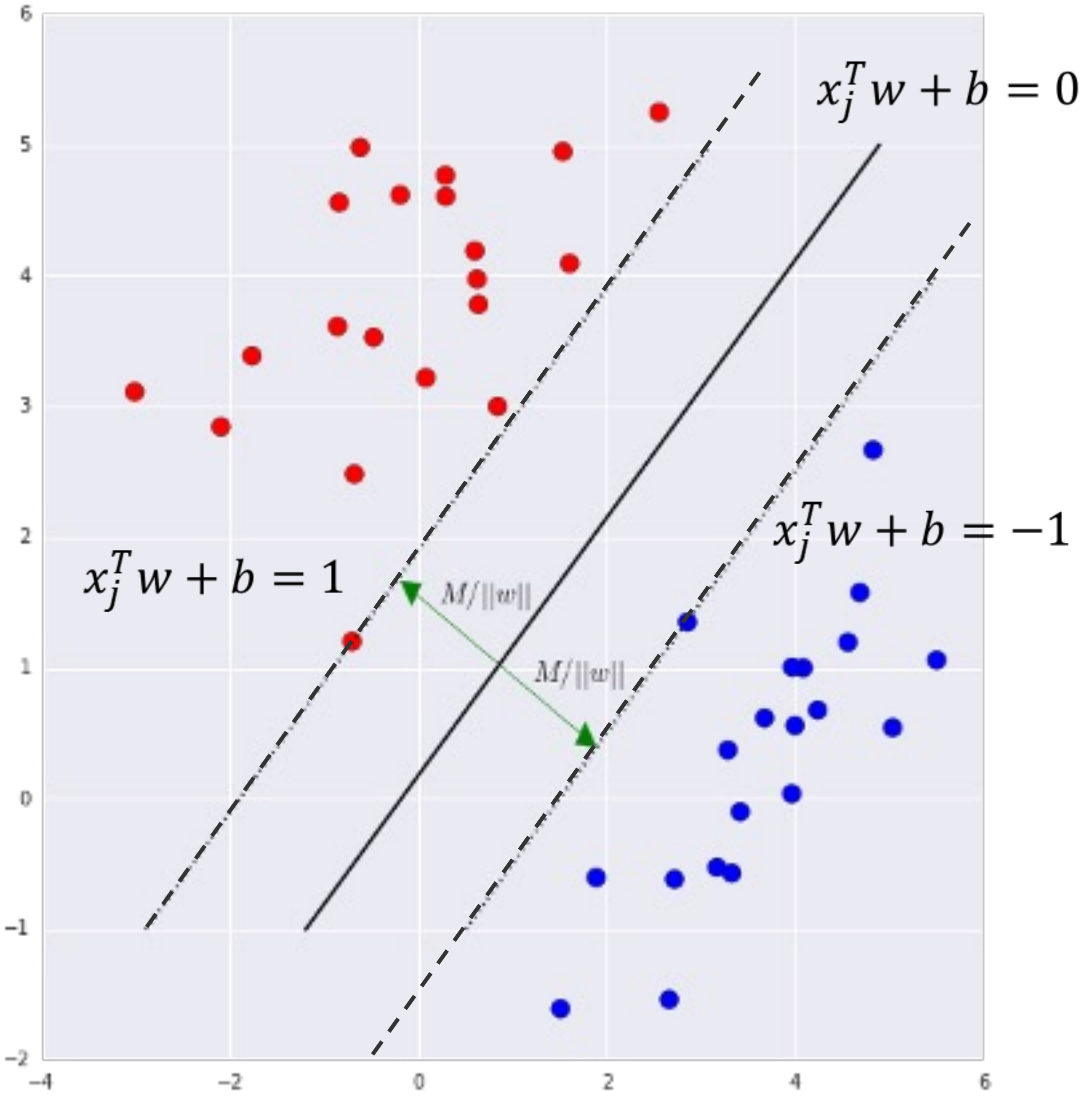
$$y_j(x_j^T w + b) \geq M$$

Simplify by change of variables, dividing w and b through by M.

New goal: minimize ||w|| subject to constraints, for all j:

$$y_j(x_j^T w + b) \geq 1$$

New margin: 2 / ||w||



Decision boundary (separating line or hyperplane)

How to Maximize the Margin?

Margin = 2M / ||w||.

This follows from computing the distance between parallel lines.

Goal: maximize 2M / ||w|| subject to constraints, for all j:

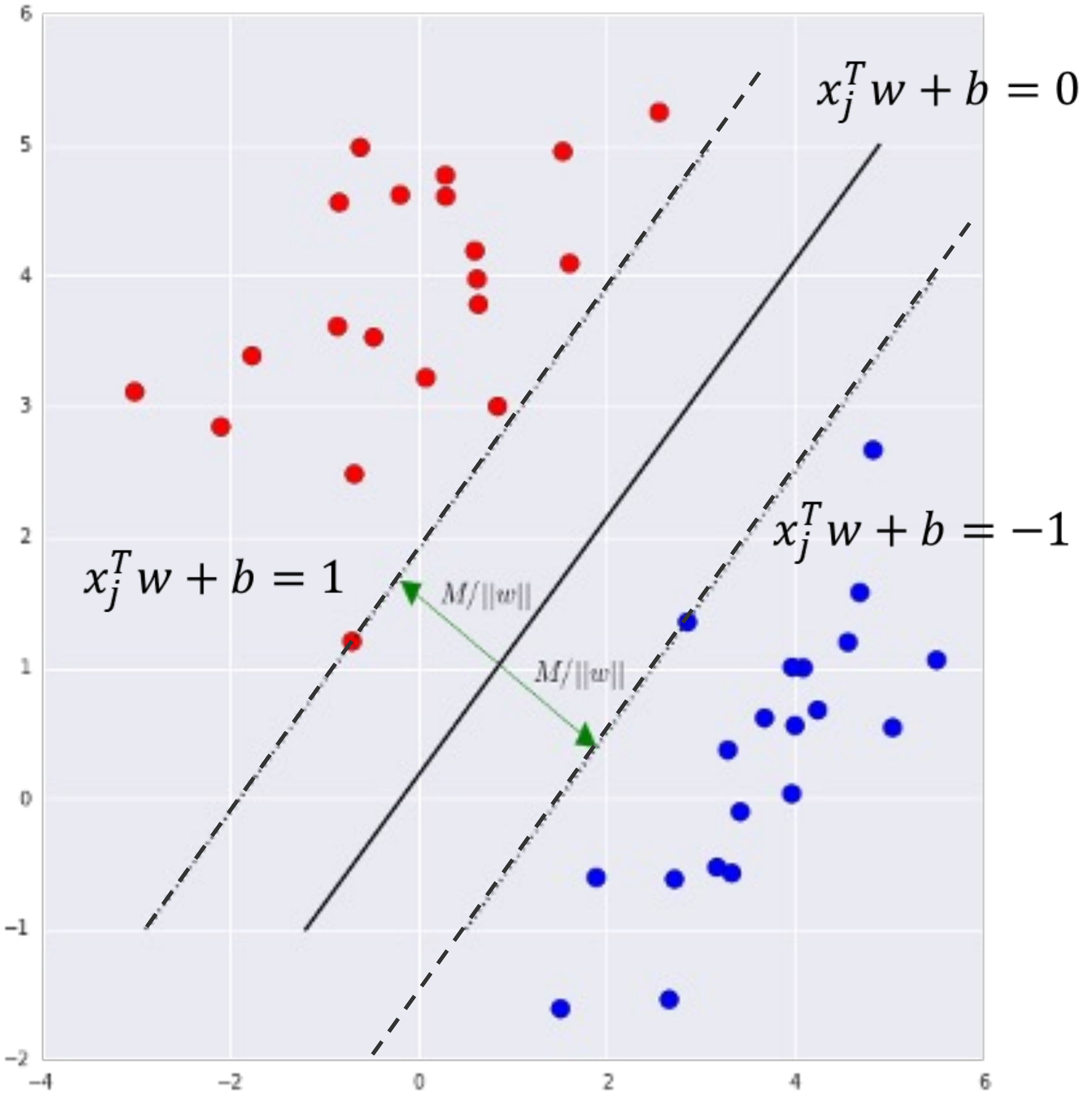
$$y_j(x_j^T w + b) \geq M$$

Simplify by change of variables, dividing w and b through by M.

New goal: minimize ||w|| subject to constraints, for all j:

$$y_j(x_j^T w + b) \geq 1$$

New margin: 2 / ||w||



Decision boundary (separating line or hyperplane)

Question:
What if the points are not linearly separable? Then there is no solution satisfying the constraints!

NYU

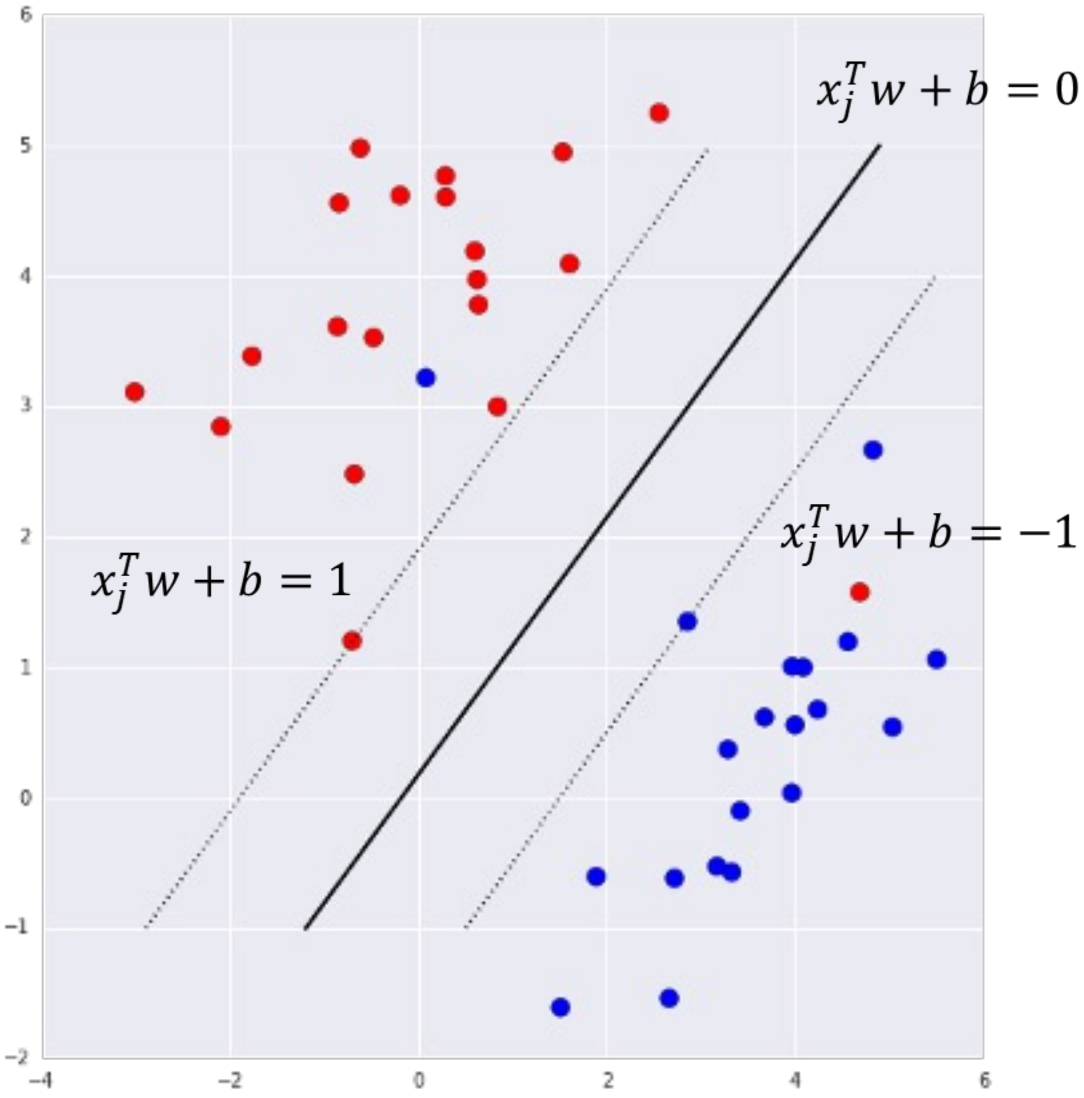
Non-separable Case: Soft Margins

Goal (hard margin): minimize ||w|| subject to constraints, for all j:

$$y_j\left(x_j^T w + b\right) \geq 1$$

Goal (soft margin): minimize subject to constraints, for all j:

$$y_j\left(x_j^T w + b\right) \geq 1 - \xi_j$$
$$\xi_j \geq 0$$



Decision boundary (separating line or hyperplane)

NYU

Non-separable Case: Soft Margins

Goal (hard margin): minimize ||w|| subject to constraints, for all j:

$$y_j\left(x_j^T w + b\right) \geq 1$$

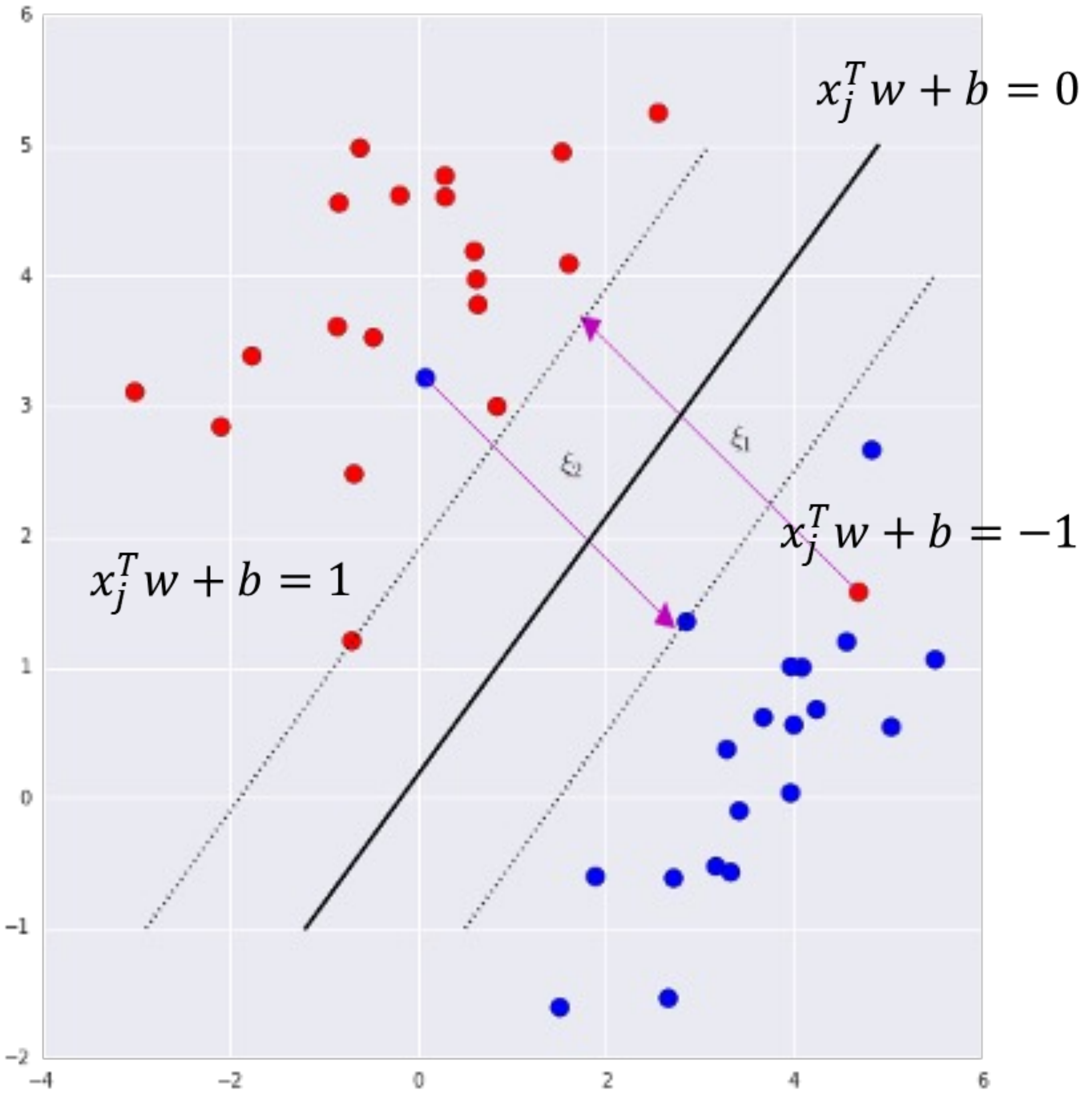
Goal (soft margin): minimize subject to constraints, for all j:

$$y_j\left(x_j^T w + b\right) \geq 1 - \xi_j$$

$$\xi_j \geq 0$$

But what should we minimize? ||w||?

<u>Answer</u>: minimize $\frac{1}{2}\|w\|^2 + C\sum_j \xi_j$



Decision boundary (separating line or hyperplane)

This is a quadratic programming (QP) problem, optimizing a quadratic function with linear constraints.

We can use off-the-shelf QP solvers or faster methods for large problems to find the optimal w, b, and ξ.

Classify test points by $sign(x_j^T w + b)$.

Soft Margins in Practice

In practice, there may be many training points with ξ_j > 0 (all of these are support vectors).

Training points with ξ_j > 1 are misclassifications.

Goal (hard margin): minimize ||w|| subject to constraints, for all j:

$$y_j\left(x_j^T w + b\right) \geq 1$$

Goal (soft margin): minimize subject to constraints, for all j:

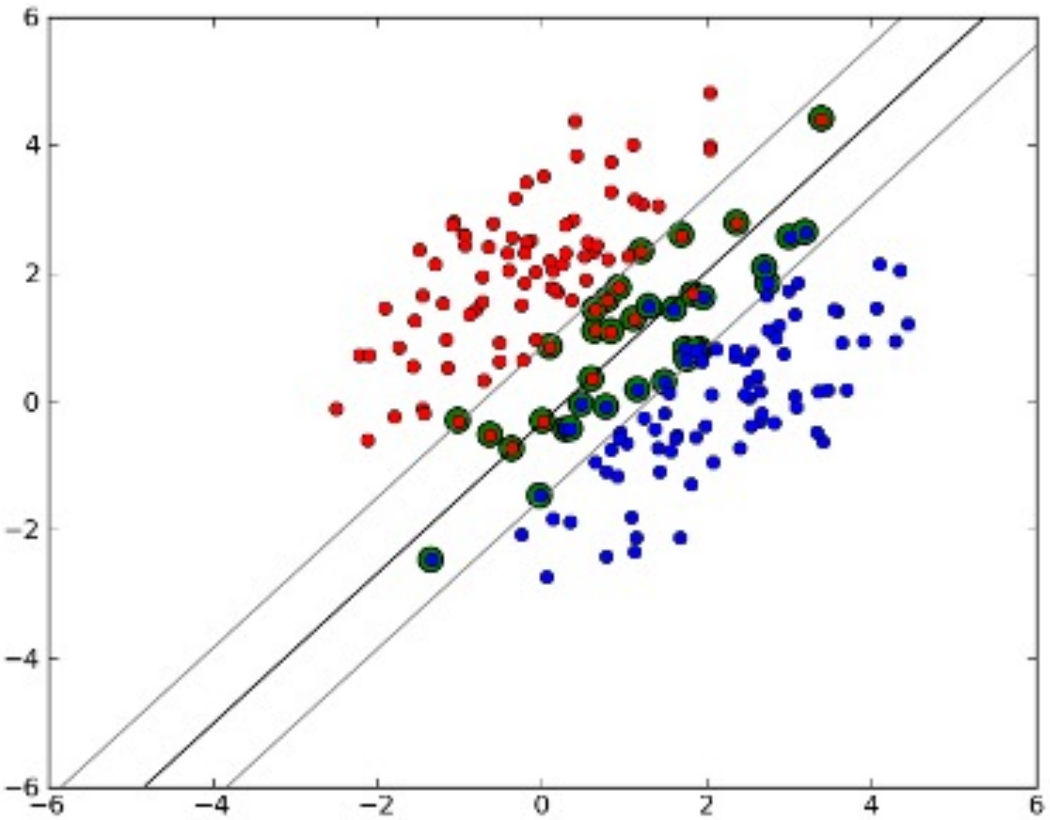
$$y_j\left(x_j^T w + b\right) \geq 1 - \xi_j$$

$$\xi_j \geq 0$$

But what should we minimize? ||w||?

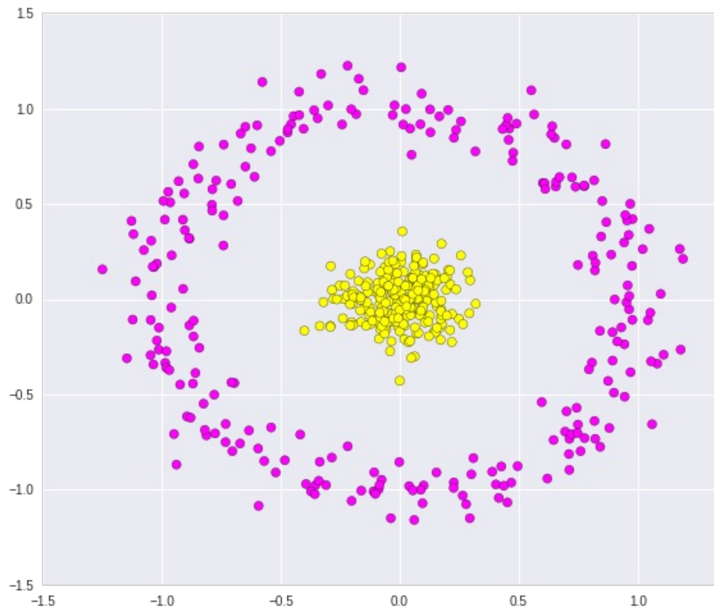
Answer: minimize

$$\frac{1}{2}\|w\|^2 + C\sum_j \xi_j$$



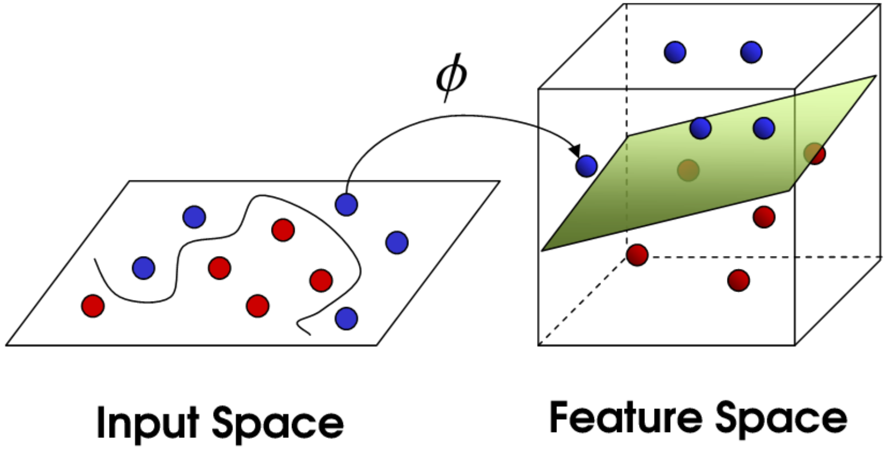
Source: http://www.mblondel.org/journal/2010/09/19/support-vector-machines-in-python/

Non-linear Decision Boundaries



Question:
What do we do in cases like this one?

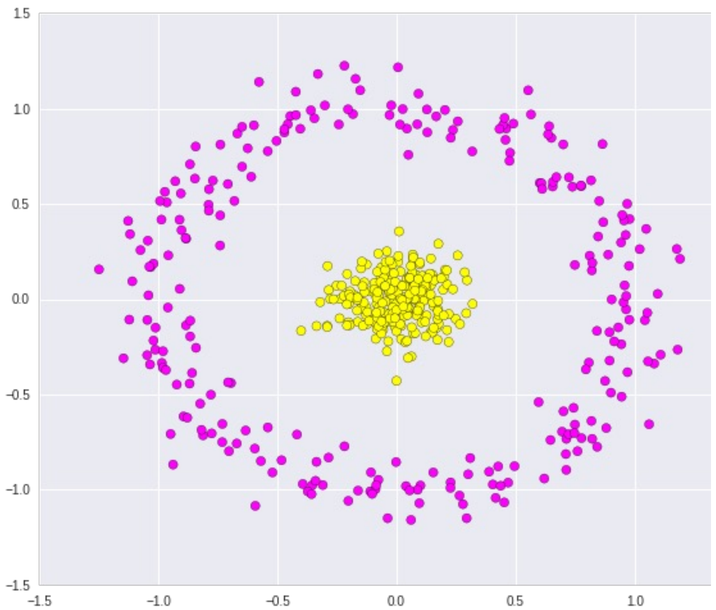
Any linear separator will perform terribly!



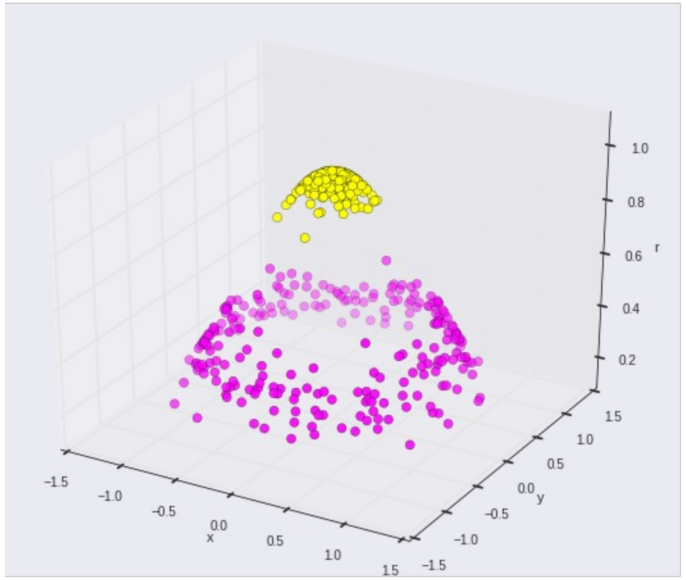
Solution:

1) Map input space to a high-dimensional feature space.
2) Learn a linear decision boundary (hyperplane) in the high-dimensional space.
3) Map back to lower-dimensional space, giving a non-linear boundary.

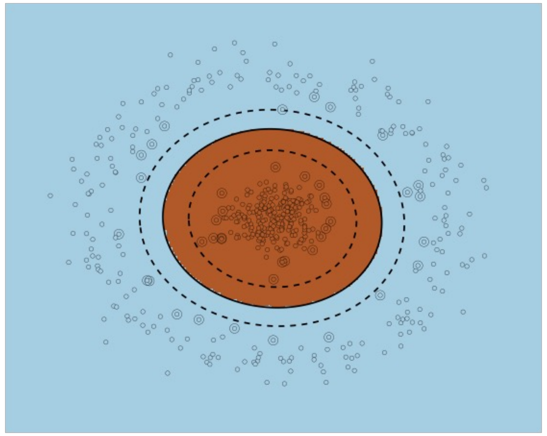
Non-linear Decision Boundaries



$$\phi : (x, y) \to (x, y, r)$$



The resulting classifier perfectly separates the training data.



NYU

Non-linear Decision Boundaries

Non-linear QP problem: $\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_j \xi_j$ subject to: $y_j\left(w^T\boldsymbol{\Phi}(x_j) + b\right) \geq 1 - \xi_j$

$\xi_j \geq 0$

Problem: not efficiently computable, since $\boldsymbol{\Phi}(x_j)$ may be high- or infinite-dimensional!

Solution: transform to equivalent ("dual") QP problem:

$\min_{\alpha} \frac{1}{2}\alpha^T Q\alpha - \sum_j \alpha_j$ subject to: $0 \leq \alpha_j \leq C$ where: $Q_{ij} = y_i y_j(\boldsymbol{\Phi}(x_i) \cdot \boldsymbol{\Phi}(x_j))$

$\sum_j \alpha_j y_j = 0$

$= y_i y_j K(x_i, x_j)$

Very cool trick (the "kernel trick"): instead of mapping both x_i and x_j into a high-dimensional space and computing the dot product in that space, we can just compute a function $K(x_i, x_j)$ of the original data points.

This makes the QP efficiently solvable. To classify a test point x, we just need to compute $sign(\sum_j \alpha_j y_j K(x_j, x) + \rho)$.

Sum is just over the support vectors; other points have $\alpha_j = 0$.

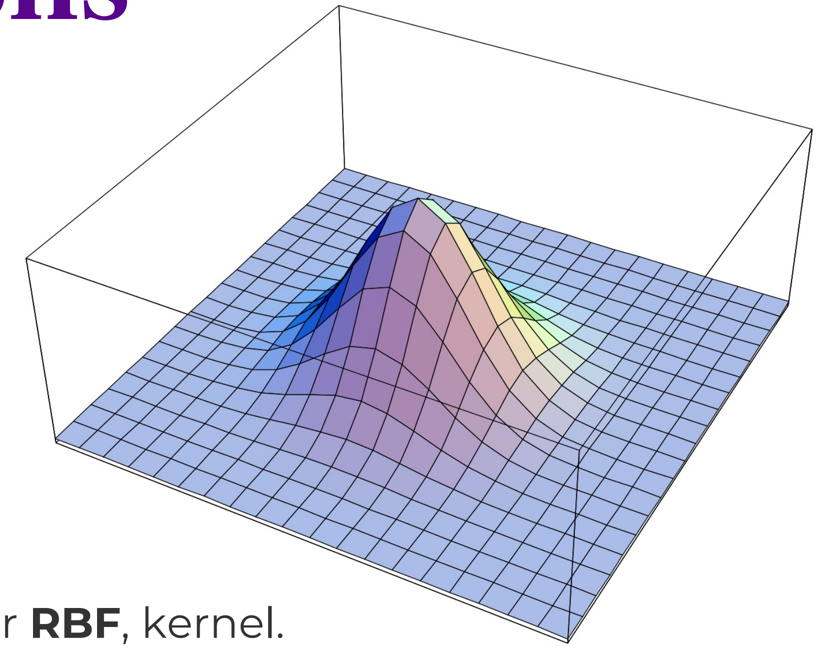
NYU

Some Common Kernel Functions

Linear kernel: $\phi : x \to x \quad K(x_i, x_j) = x_i \cdot x_j$

Non-linear kernels:

- Polynomial kernel: $K(x_i, x_j) = (\gamma(x_i \cdot x_j) + r)^d$
- Sigmoid kernel: $K(x_i, x_j) = \tanh\left(\gamma(x_i \cdot x_j) + r\right)$
- Gaussian kernel: $K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$



The Gaussian kernel is usually called the “radial basis function”, or **RBF**, kernel.
It is one of the most widely used kernel choices and a good default option.

<u>Very cool trick (the “kernel trick”):</u> instead of mapping both x_i and x_j into a high-dimensional space and computing the dot product in that space, we can just compute a function $K(x_i, x_j)$ of the original data points.

This makes the QP efficiently solvable. To classify a test point x, we just need to compute $sign(\sum_j \alpha_j y_j K(x_j, x) + \rho)$.

Sum is just over the support vectors; other points have $\alpha_j = 0$.

Variants and Extensions of SVMs

SVMs are mainly used for **non-probabilistic, binary classification**.

To do multi-class classification:

For each class k, learn a binary classifier (class k vs. rest).

To predict the output for a new test example x, predict with each SVM.

Choose whichever one puts the prediction the furthest into the positive region.

To estimate class probabilities:

SVMs are not really the best for this, but they can do logistic regression using outputs of k(k-1) pairwise SVMs.

Lots of models + additional cross-validation are needed → this approach is very computationally expensive (Wu et al., 2004).

Support vector machines can also be used for **regression** (Smola and Schölkopf, 2003) and for **anomaly detection** (the “one-class SVM,” Schölkopf et al., 2001).

Both are implemented in scikit-learn but are beyond the scope of this class.

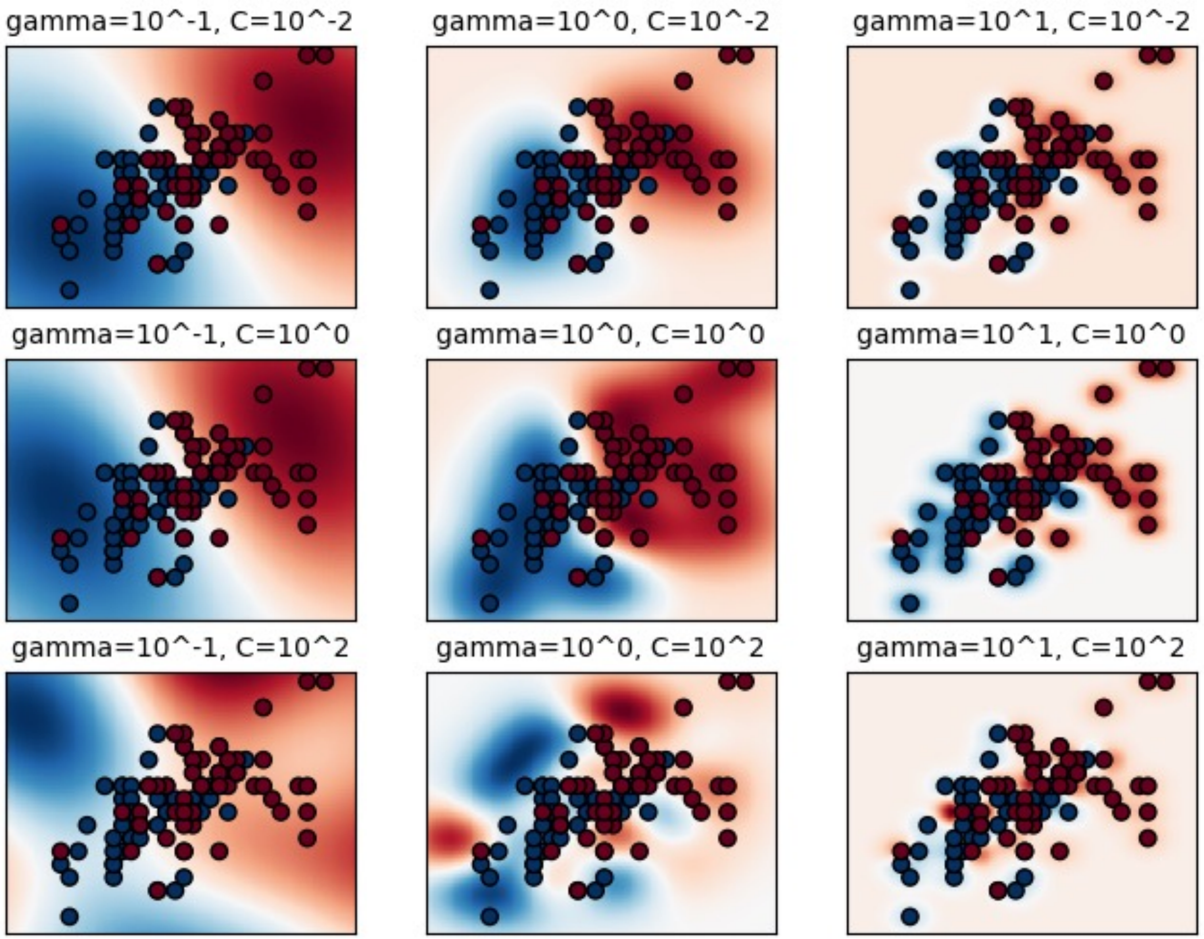
Advantages of SVMs

- Very good performance: though lately outshined by convolutional neural networks on some benchmarks (e.g., the MNIST digit recognition dataset), they often beat basically everything else.
- Theoretical guarantees about their generalization performance (accuracy for labeling test data) based on statistical learning theory.
- SVMs rely on convex optimization and do not get stuck in suboptimal local minima (neural networks have a big problem with these; similarly, decision trees rely on greedy search).
- Fairly robust to the curse of dimensionality → can effectively solve prediction problems with a large number of features.
- Flexible: can choose kernel to fit very complex decision boundaries.
- Will generally avoid overfitting with well-chosen parameters (but can certainly be overfitted for poorly chosen values, e.g., if C is too large).
- Classification of test points relies only on the support vectors → fast and memory efficient, especially when # of support vectors is small.

Disadvantages of SVMs

- Training the model is computationally expensive – dependent on # of support vectors, but typically quadratic to cubic in the number of data points.
- Sensitive to the choice of parameters, particularly the constant C and kernel bandwidth (γ for RBF kernel in sklearn).
  - C trades off the misclassification rate against the simplicity of the decision surface. Low C → smooth decision surface; High C → more training examples classified correctly.
  - Larger γ = lower bandwidth (increased weight on nearest training examples).
  - Proper choice of C and γ is critical to the SVM's performance.
  - For sklearn, use GridSearchCV with C and γ spaced exponentially far apart.
- Not much interpretability for non-linear SVM: it can enumerate the support vectors or (in low dimensions) visualize the decision boundary, but actually obtaining these involves calling a black-box optimization routine.

Non-Linear SVM Parameters



https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

Lab Time

NYU

For the Next Week (Week 5)

No class on February 19, 2024 (Presidents Day).

1. Check readings (optional) and review them
2. Assignment 1

    Due: February 16, 2024 (11:59pm)

References

1. Scikit-learn documentation: http://scikit-learn.org/stable/modules/svm.html
2. C.J.C. Burges. A tutorial on support vector machines for pattern recognition. Data Mining & Knowledge Discovery, 2: 955-974, 1998.
3. A.W. Moore. Support Vector Machines (tutorial slides). https://www.cs.cmu.edu/~./awm/tutorials/svm.html
4. V. Vapnik. Statistical Learning Theory. Wiley: 1998.
5. T.-F. Wu, C.-J. Lin, and R.C. Weng. Probability estimates for multi-class classification by pairwise coupling. Journal of Machine Learning Research 5: 975-1005, 2004.
6. A.J. Smola and B. Schölkopf. A tutorial on support vector regression, Statistics and Computing, 2003. http://alex.smola.org/papers/2003/SmoSch03b.pdf
7. B. Schölkopf et al. Estimating the support of a high-dimensional distribution. Neural Computation 13: 1443-1471, 2001.
8. Berwick. 2009. An Idiot's guide to Support vector machines (SVMs)