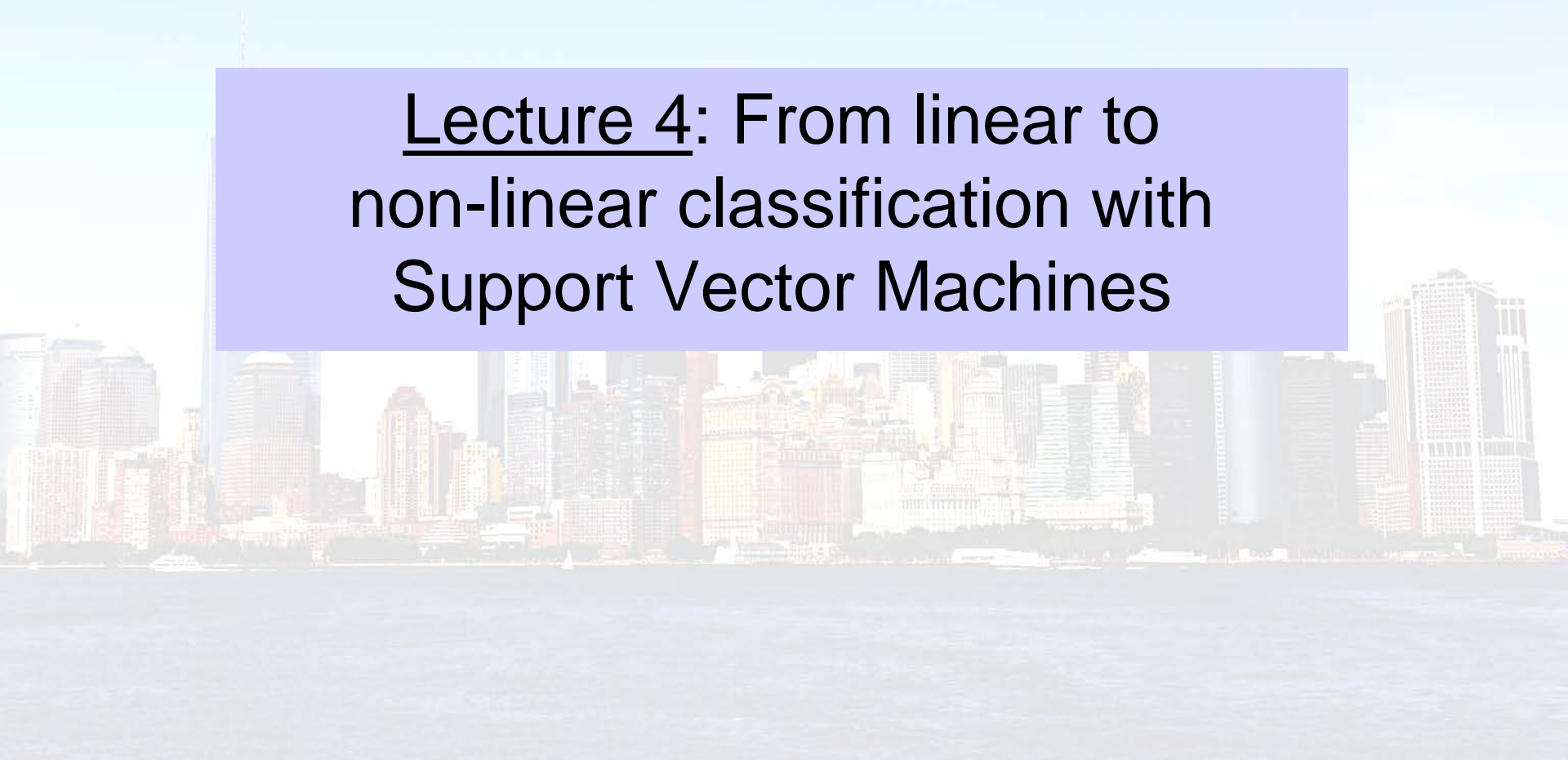


Acknowledgement: these slides are partially based on Machine Learning for Cities course material from Dr. Stanislav Sobolevsky, used with permission.

Machine Learning for Cities

CUSP-GX 5006.001, Spring 2019

Lecture 4: From linear to non-linear classification with Support Vector Machines



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?)

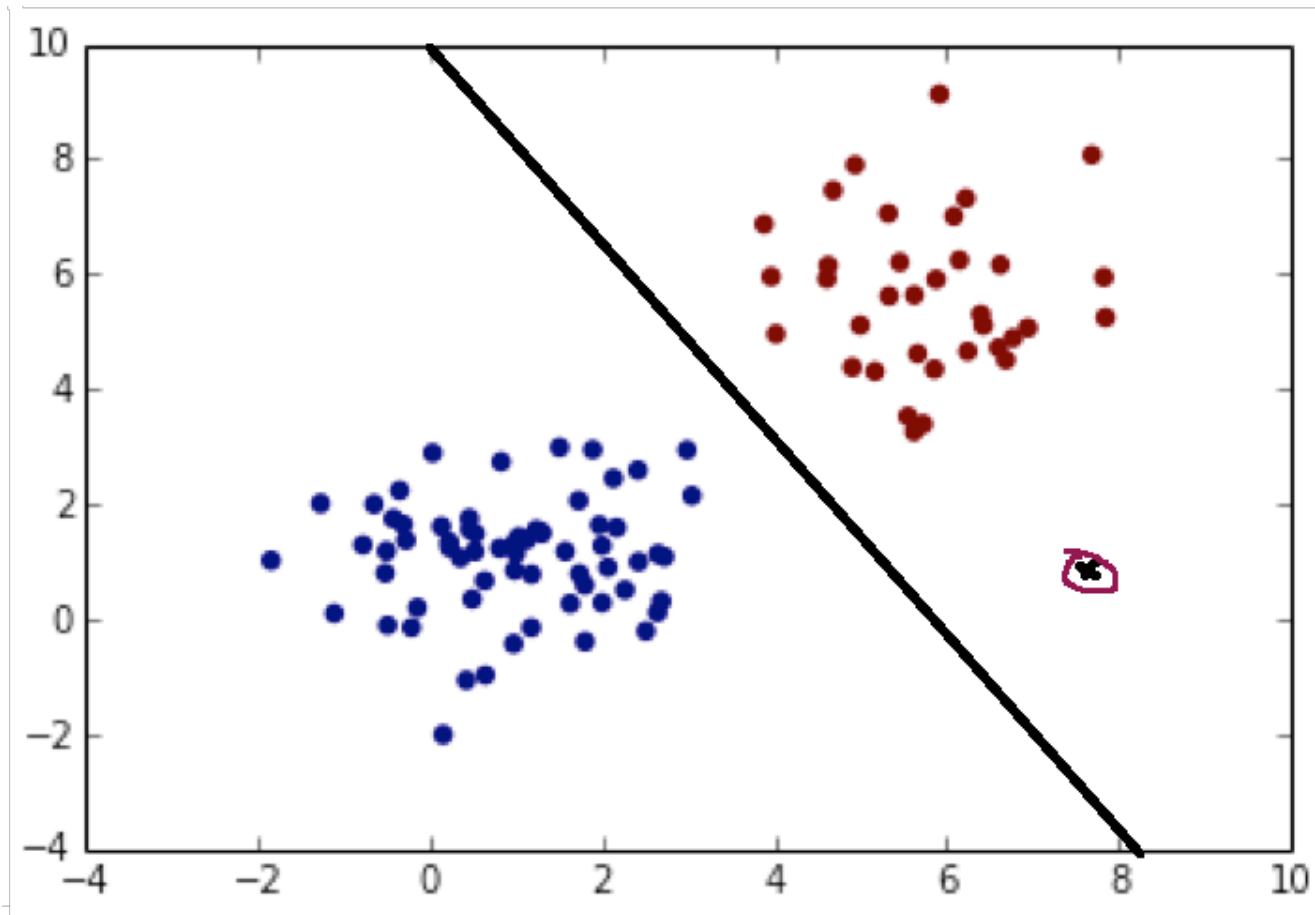
Key idea 2: Learn a **linear** decision boundary in high dimensional space corresponding to a **non-linear** decision boundary for the original problem.

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 same scaling for training and test data.
- Replace discrete-valued attributes with **dummy variables**.

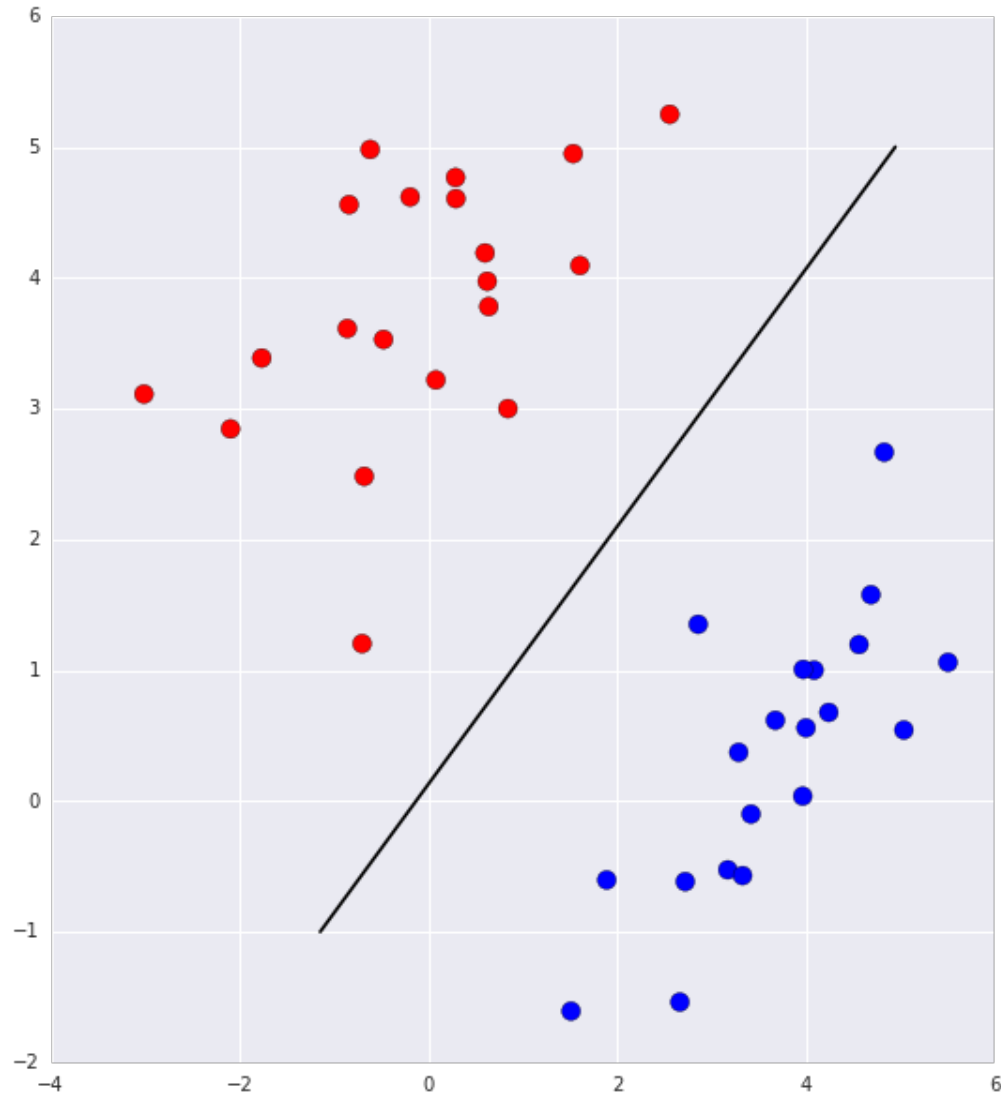
<u>Car</u>	<u>Weight</u>		<u>Car</u>	<u>Weight=Medium</u>	<u>Weight=Heavy</u>
1	Low		1	0	0
2	Medium	⇒	2	1	0
3	Heavy		3	0	1

SVMs: the basic idea (linearly separable case)



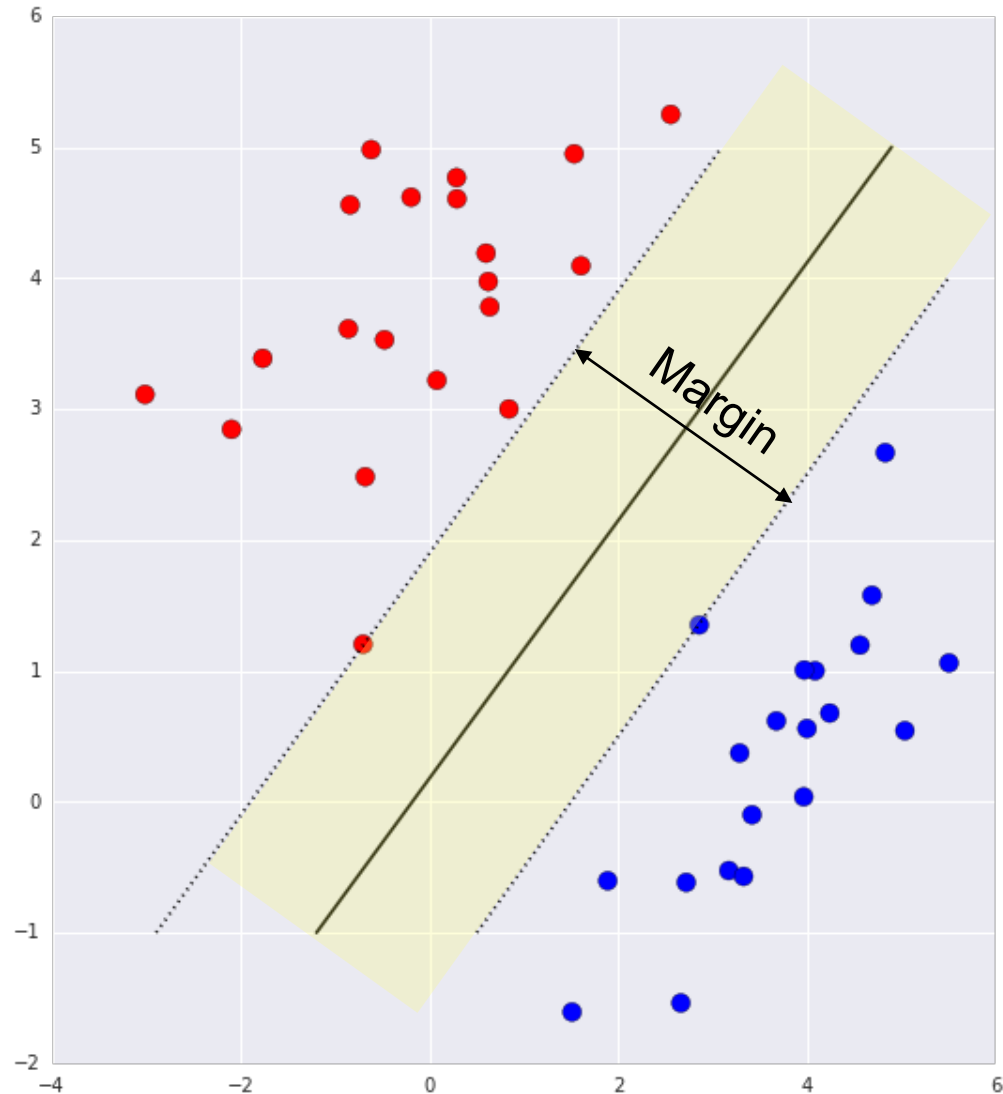
Which line to choose?

This is an optimization problem.



Which line to choose?

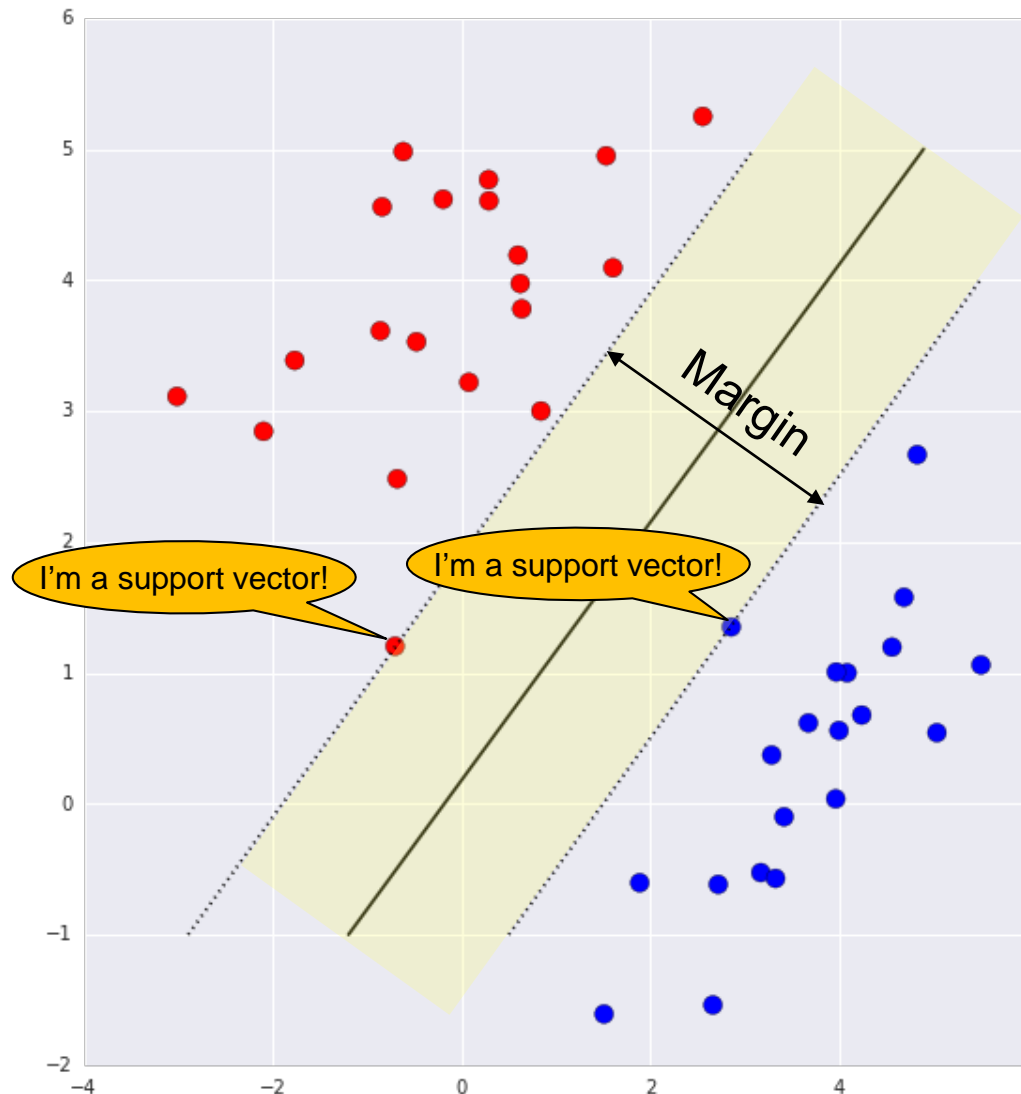
Choose the line that maximizes the **margin** between classes.



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

Which line to choose?

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



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

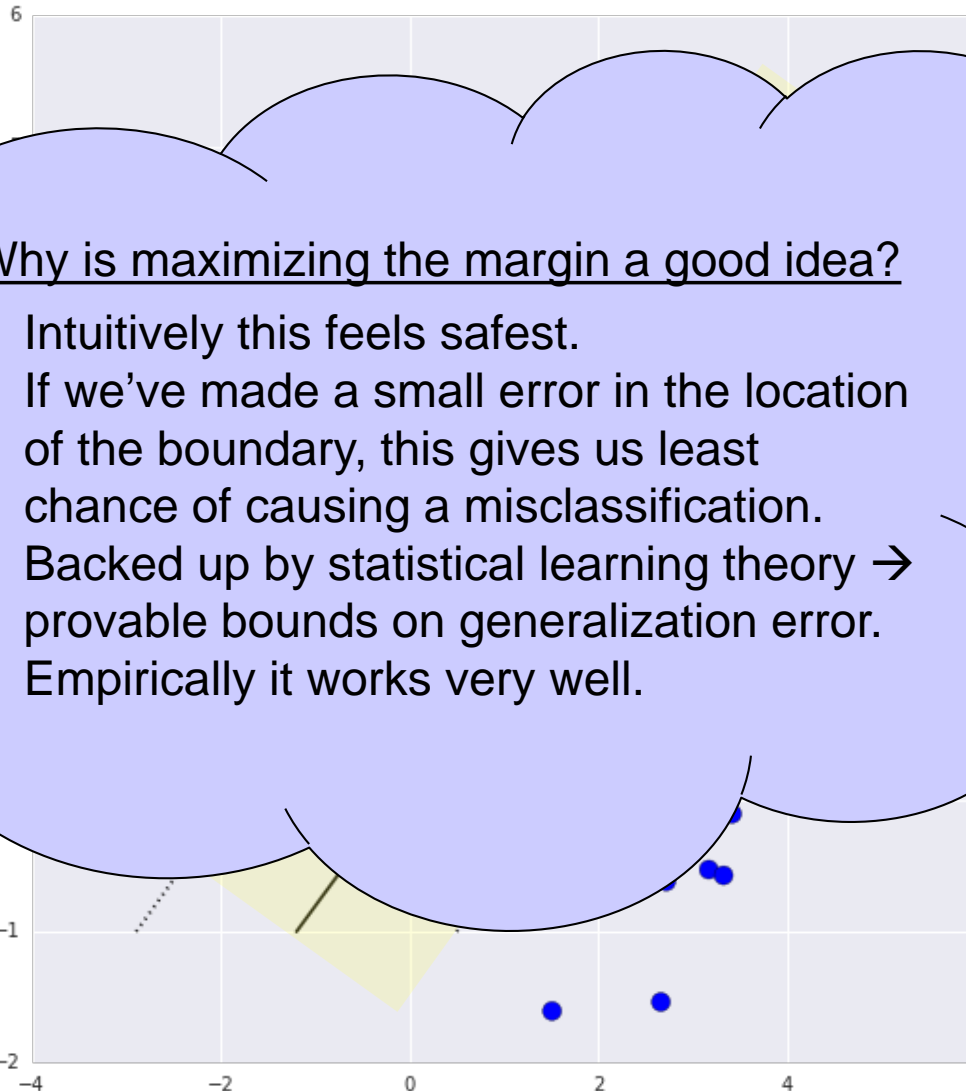
The classifier can be defined entirely by the set of support vectors.

This fact has lots of useful implications:

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

Which line to choose?

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



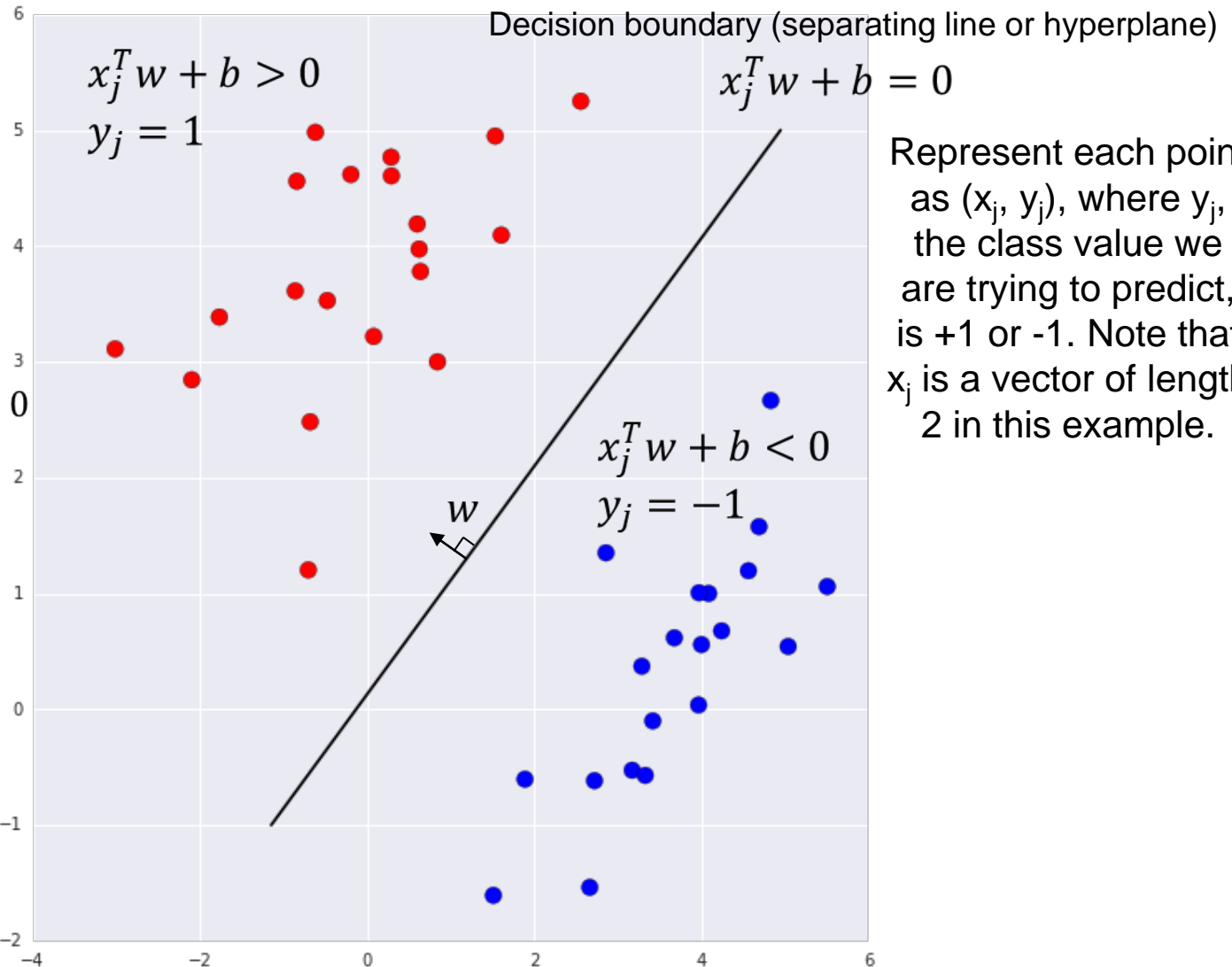
Points on the margin are called **support vectors**.

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 least chance of causing a misclassification.
3. Backed up by statistical learning theory → provable bounds on generalization error.
4. Empirically it works very well.

...er can be
...ely by the
...rt vectors.
...s lots of
...ations:
...ification
...points.
...ast leave-one-out
cross-validation.
Faster, but still
expensive, training.

How to maximize the margin?



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

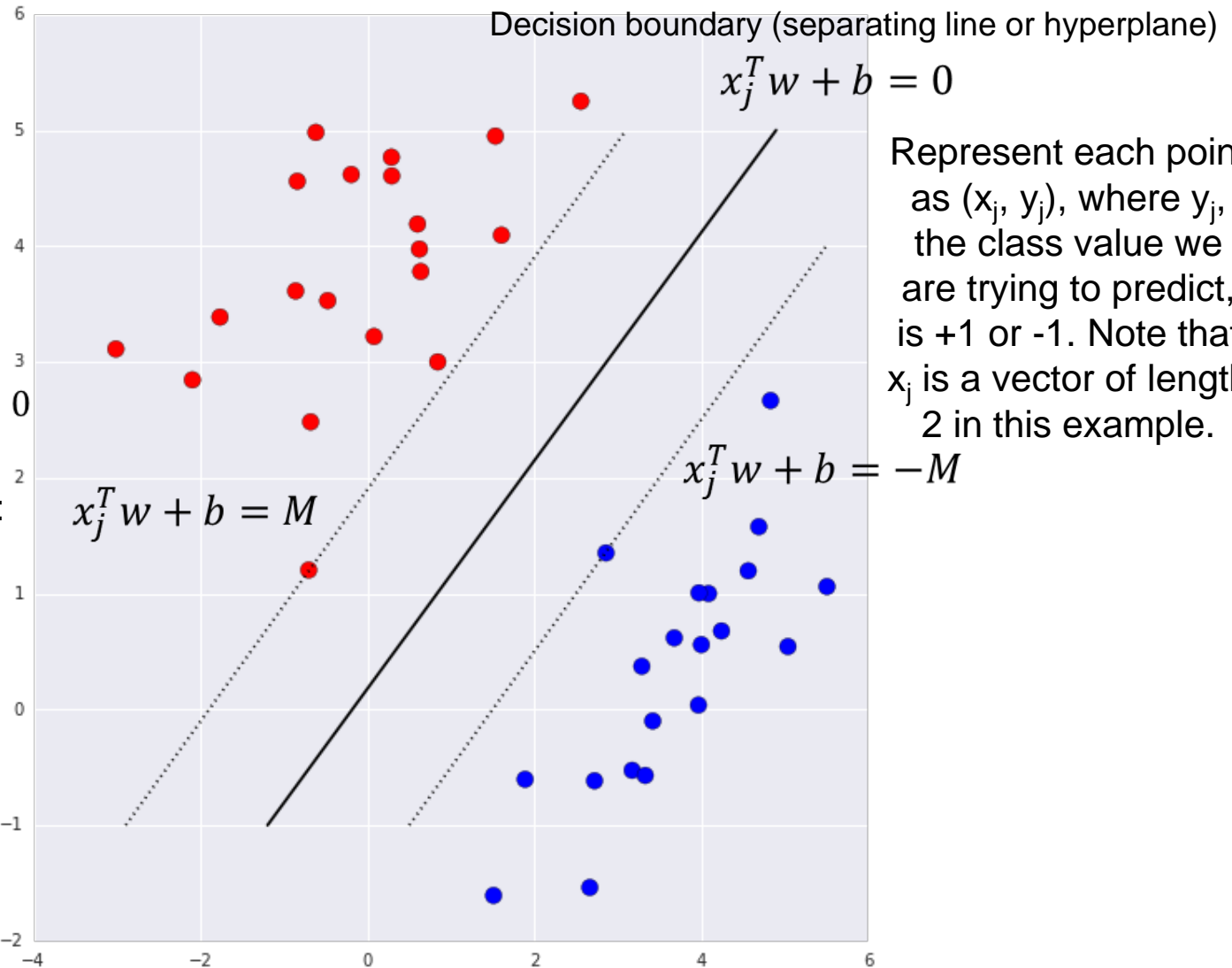
To separate, for all points j , we must have:
 $y_j(x_j^T w + b) > 0$

For the margin, define:
 $M = \min_j y_j(x_j^T w + b) > 0$

Then for $y_j = 1$, we have:
 $x_j^T w + b \geq M$

For $y_j = -1$, we have:
 $x_j^T w + b \leq -M$

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:

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

Then for $y_j = 1$, we have:

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

For $y_j = -1$, we have:

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

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

How to maximize the margin?

$$\text{Margin} = 2M / \|w\|.$$

This follows from computing distance between parallel lines.

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

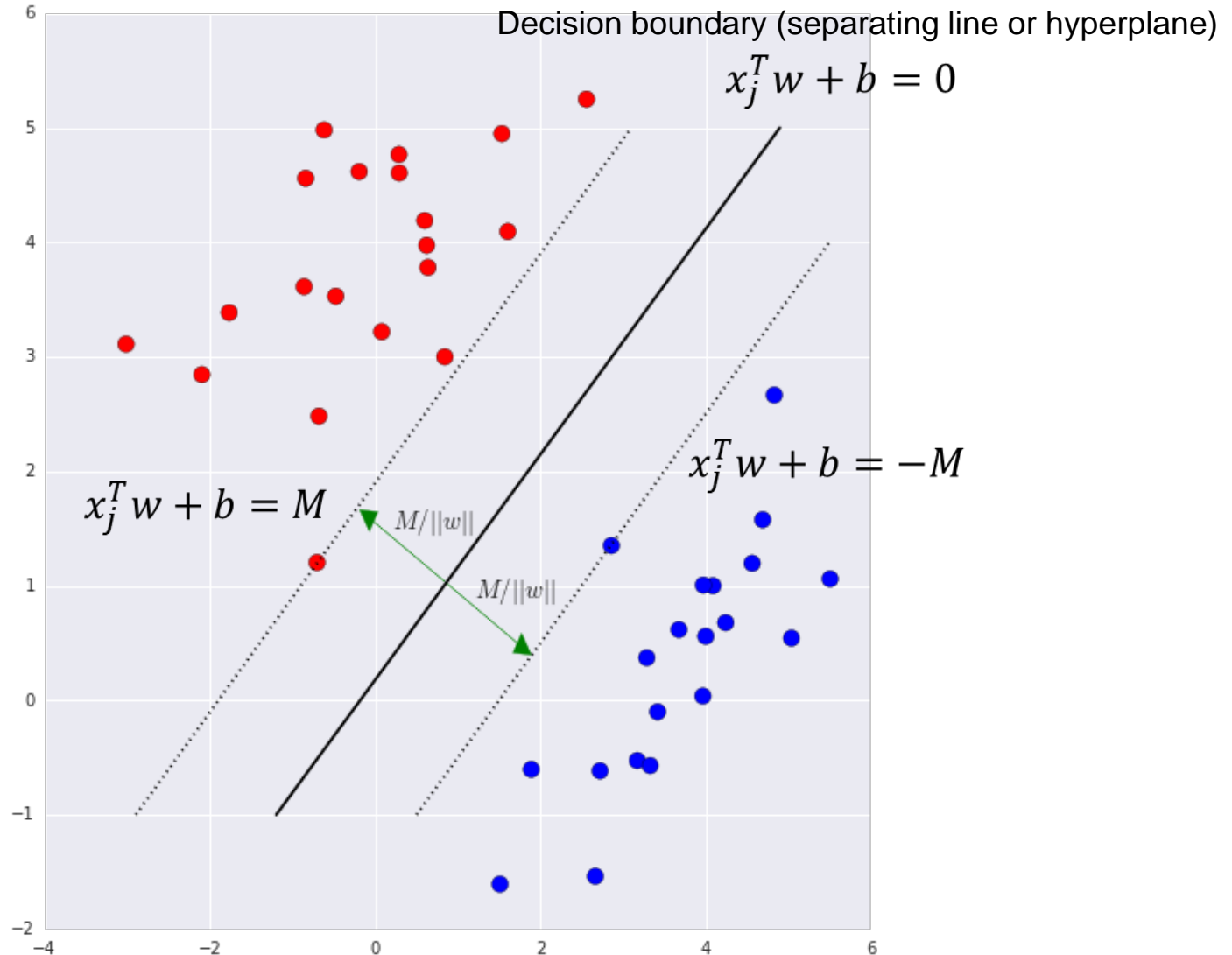
$$y_j(x_j^T w + b) \geq M$$

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: $2 / \|w\|$



How to maximize the margin?

$$\text{Margin} = 2M / \|w\|.$$

This follows from computing distance between parallel lines.

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

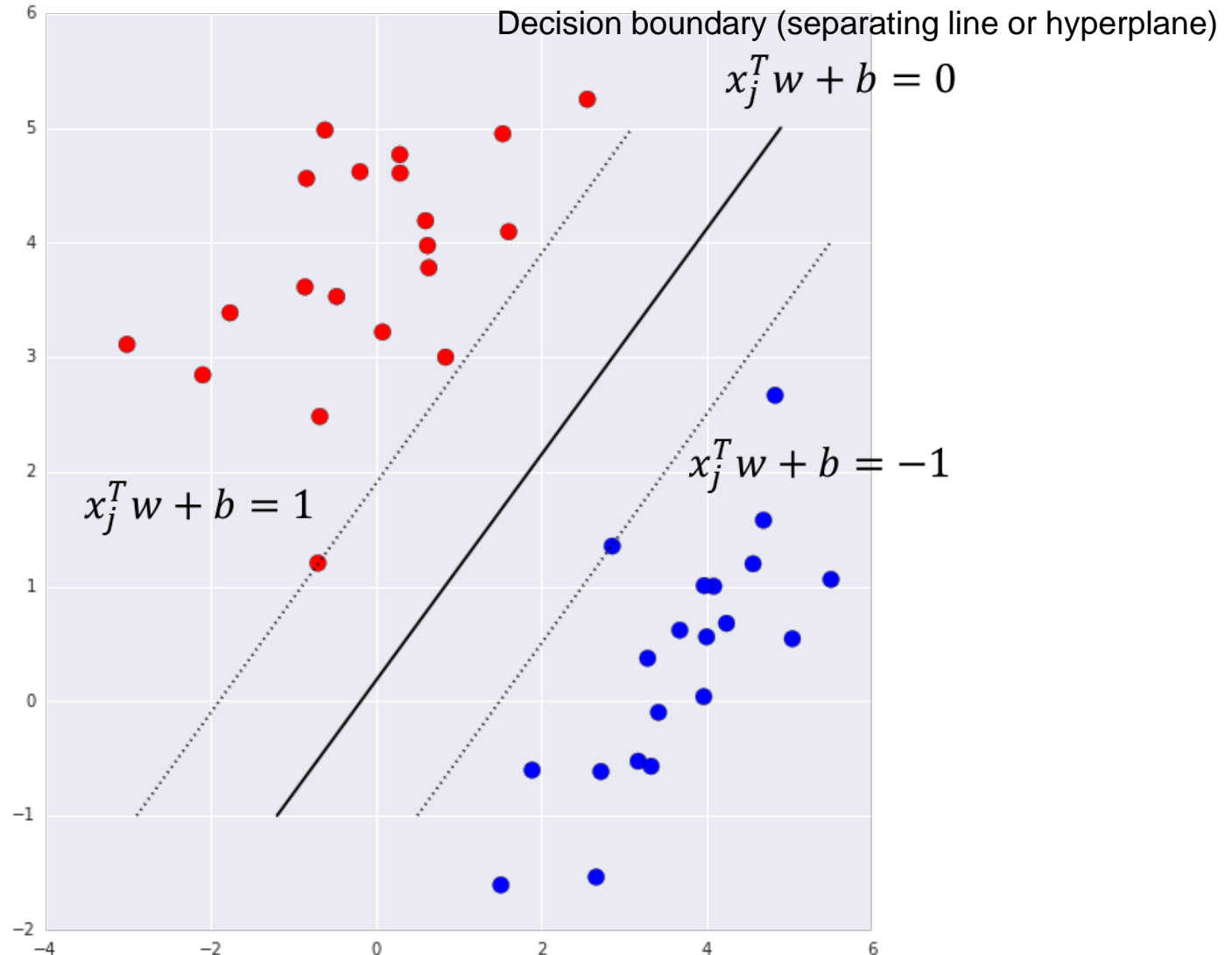
$$y_j(x_j^T w + b) \geq M$$

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: $2 / \|w\|$



How to maximize the margin?

$$\text{Margin} = 2M / \|w\|.$$

This follows from computing distance between parallel lines.

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

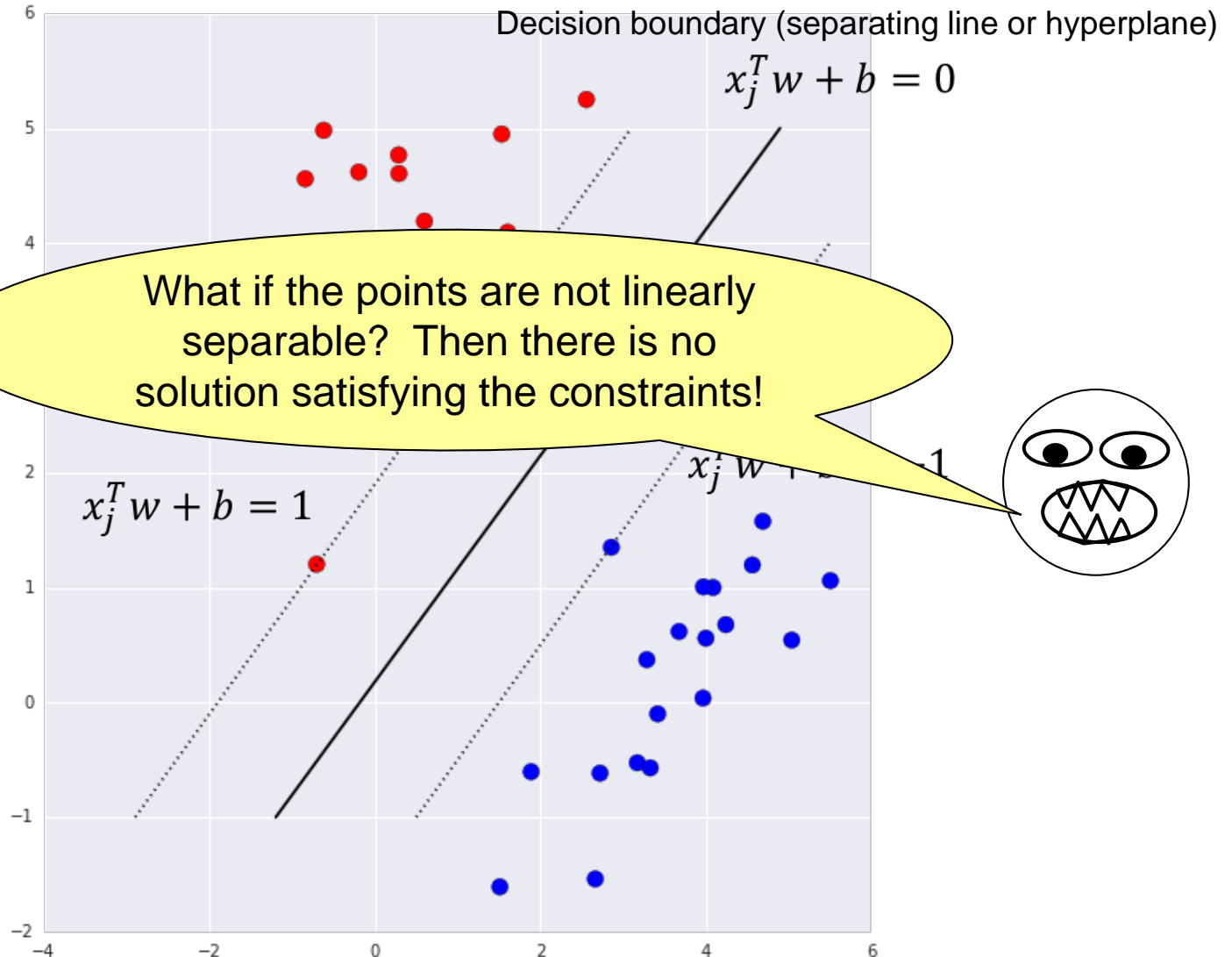
$$y_j(x_j^T w + b) \geq M$$

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

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

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

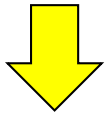
New margin: $2 / \|w\|$



Non-separable case: soft margins

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

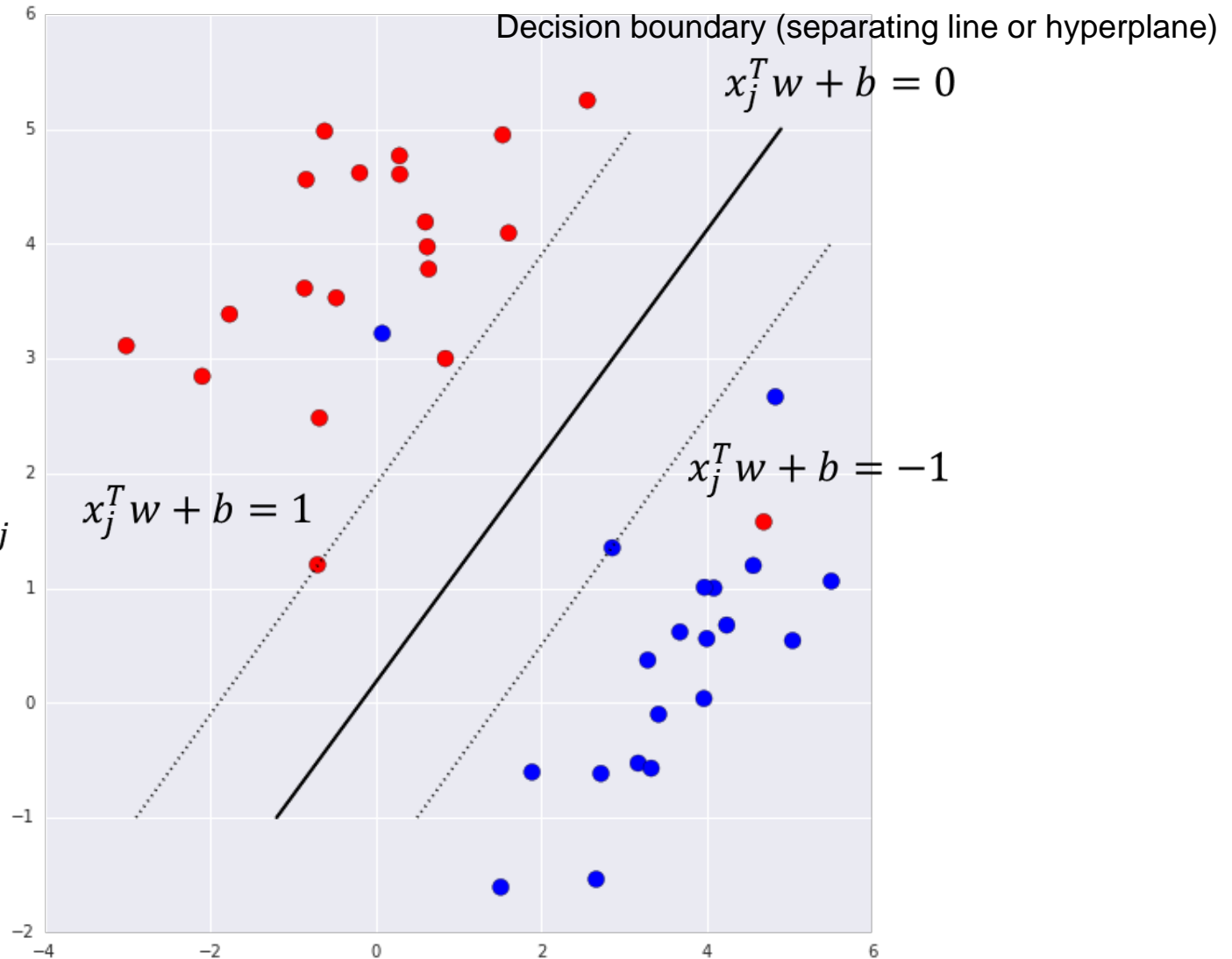
$$y_j(x_j^T w + b) \geq 1$$



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

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

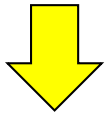
$$\xi_j \geq 0$$



Non-separable case: soft margins

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

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



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

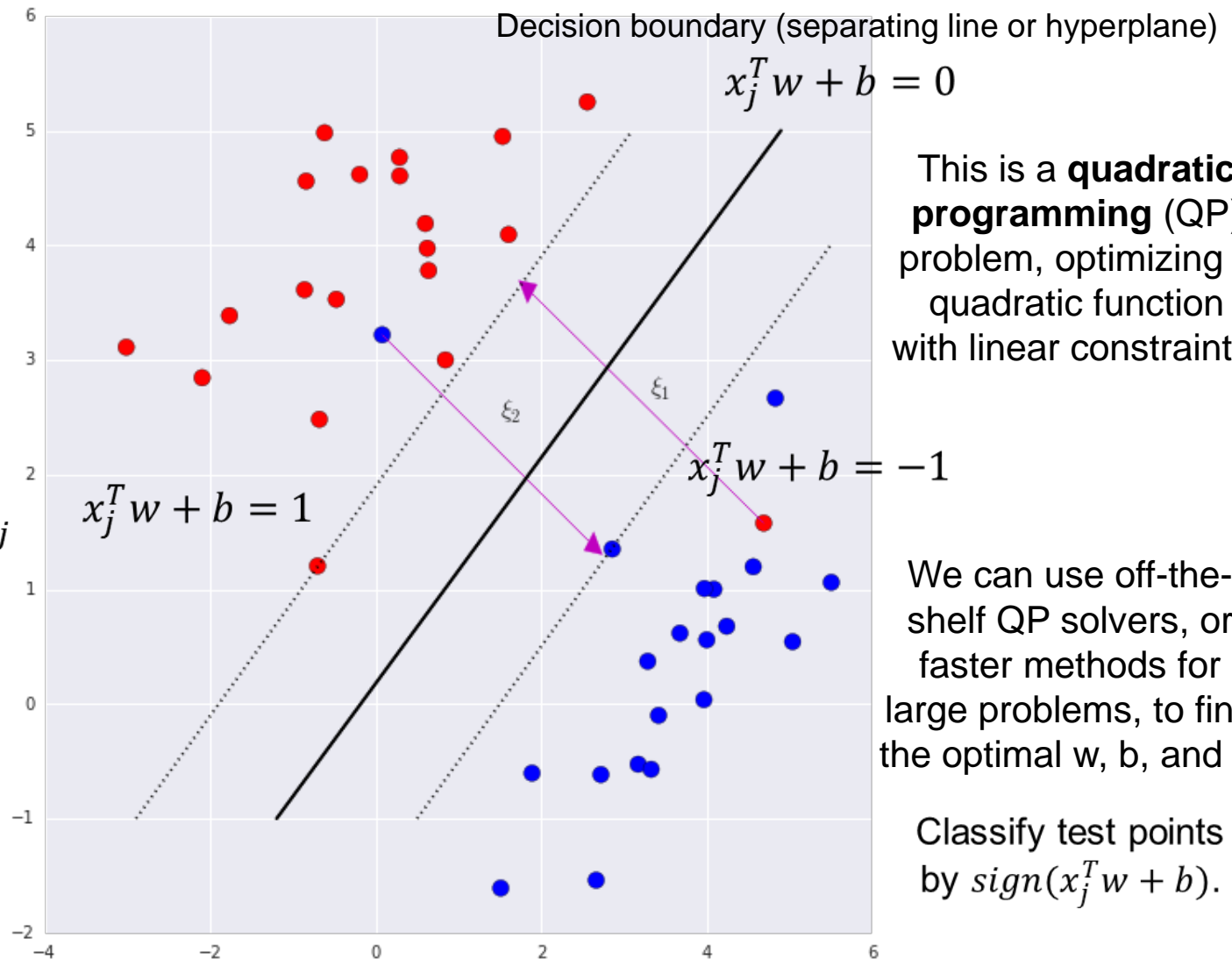
$$y_j(x_j^T w + b) \geq 1 - \xi_j$$

$$\xi_j \geq 0$$

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

Answer: minimize

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



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

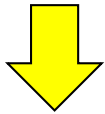
We can use off-the-shelf QP solvers, or faster methods for large problems, to find the optimal w , b , and ξ .

Classify test points by $\text{sign}(x_j^T w + b)$.

Soft margins in practice

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

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



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

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

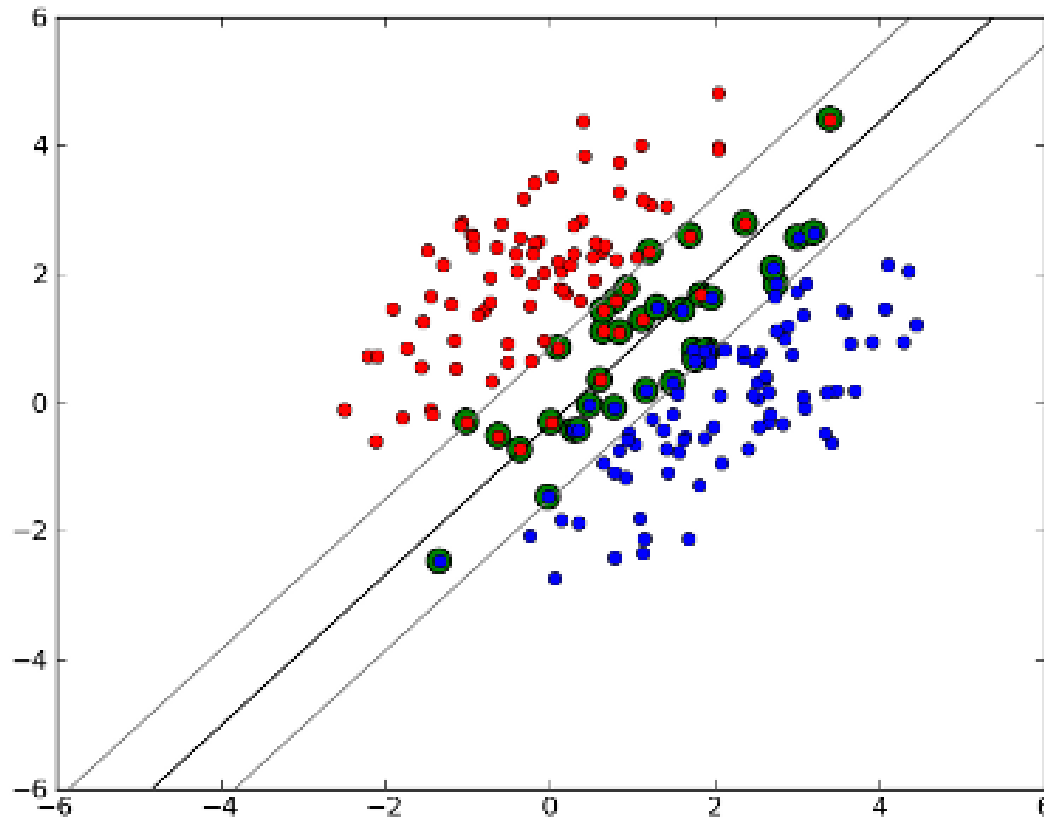
But what should we minimize? $\|w\|$?

Answer: minimize

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

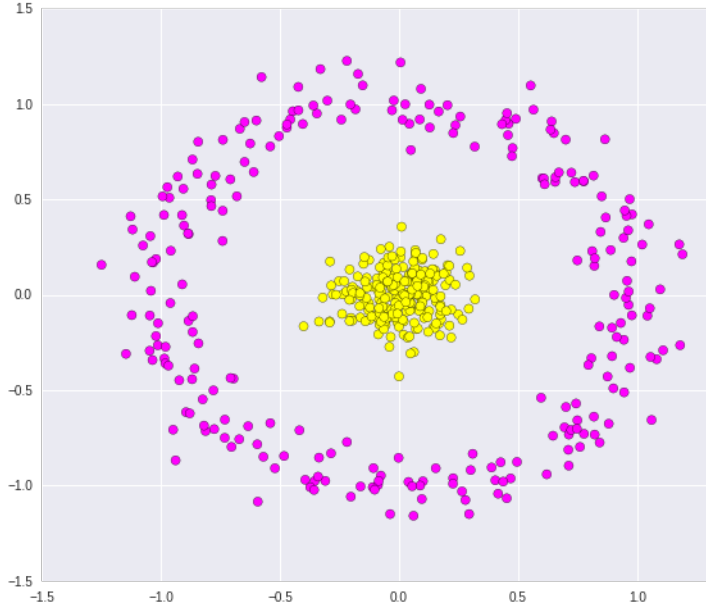
In practice, there may be many training points with $\xi_j > 0$ (all of these are support vectors).

Training points with $\xi_j > 1$ are misclassifications.

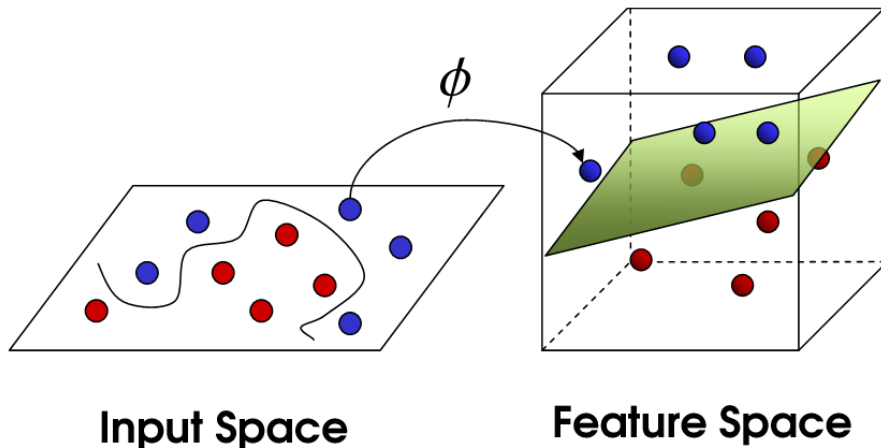
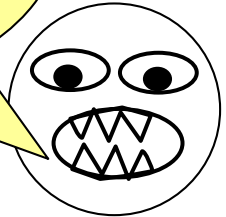


<http://www.mblondel.org/journal/2010/09/19/support-vector-machines-in-python/>

Non-linear decision boundaries



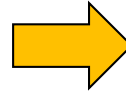
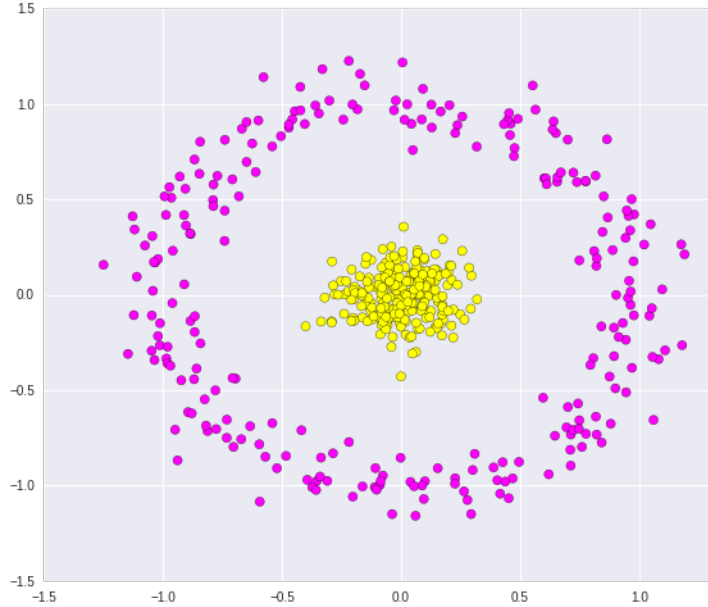
What do we do in cases like this one?
Any linear separator will perform terribly!



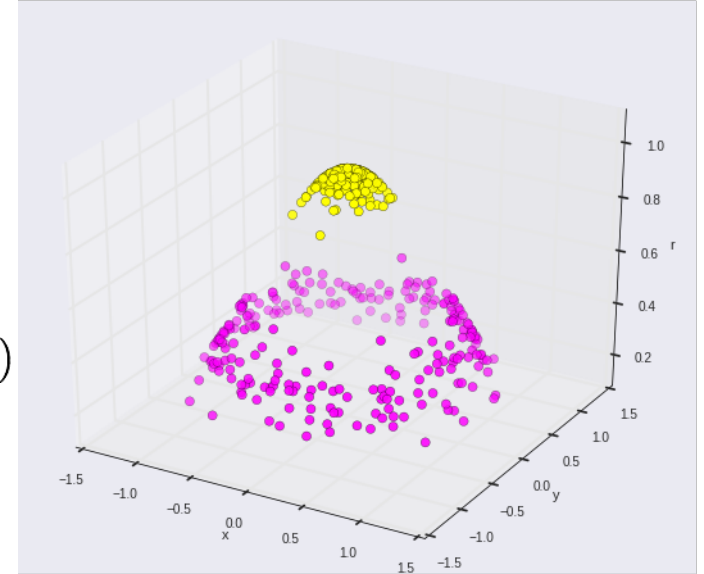
Solution:

- 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.

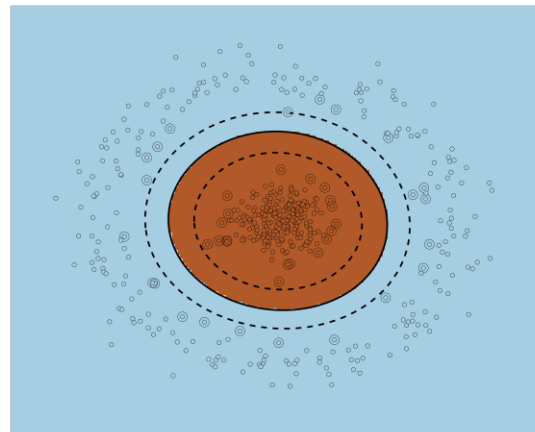
Non-linear decision boundaries



$$\phi : (x, y) \rightarrow (x, y, r)$$
$$r = e^{-(x^2+y^2)}$$



The resulting classifier perfectly separates the training data.



Non-linear decision boundaries

Non-linear QP problem: $\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C \sum_j \xi_j$ subject to: $y_j(w^T \Phi(x_j) + b) \geq 1 - \xi_j$
 $\xi_j \geq 0$

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


Solution: transform to equivalent (“dual”) QP problem:

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

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 $\text{sign}(\sum_j \alpha_j y_j K(x_j, x) + \rho)$.

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



Some common kernel functions

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

Non-linear kernels {

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

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

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 $\text{sign}(\sum_j \alpha_j y_j K(x_j, x) + \rho)$.

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 can do logistic regression using outputs of $k(k-1)$ pairwise SVMs.

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

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.

Some 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 overfit 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.

Some 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 choice of **parameters**, particularly the constant C and kernel bandwidth (γ for RBF kernel in sklearn).
 - C trades off misclassification rate against simplicity of the decision surface. Low $C \rightarrow$ smooth decision surface; High $C \rightarrow$ 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: can enumerate the support vectors or (in low dimensions) visualize the decision boundary, but actually obtaining these involves calling a black-box optimization routine.

References

- Scikit-learn documentation: <http://scikit-learn.org/stable/modules/svm.html>
- C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining & Knowledge Discovery*, 2: 955-974, 1998.
<http://research.microsoft.com/en-us/um/people/cburges/papers/svmtutorial.pdf>
- A.W. Moore. Support Vector Machines (tutorial slides).
<https://www.autonlab.org/tutorials/svm.html>
- V. Vapnik. *Statistical Learning Theory*. Wiley: 1998.
- 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.
- A.J. Smola and B. Schölkopf. A tutorial on support vector regression, *Statistics and Computing*, 2003.
<http://alex.smola.org/papers/2003/SmoSch03b.pdf>
- B. Schölkopf et al. Estimating the support of a high-dimensional distribution. *Neural Computation* 13: 1443-1471, 2001.

Up next: a short break, and then Python examples for support vector machines.