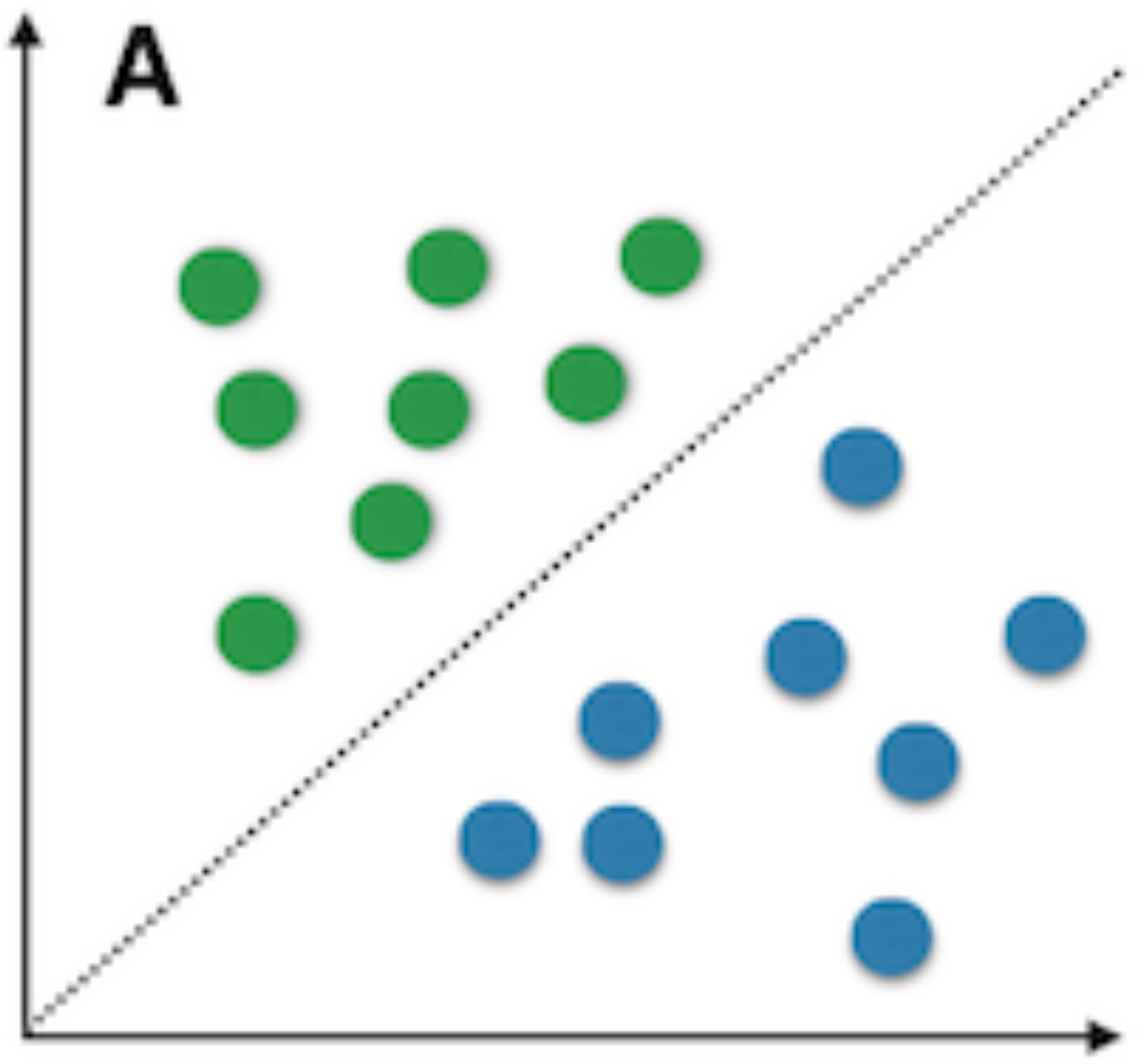
Soft & Kernel SVMs

Recap

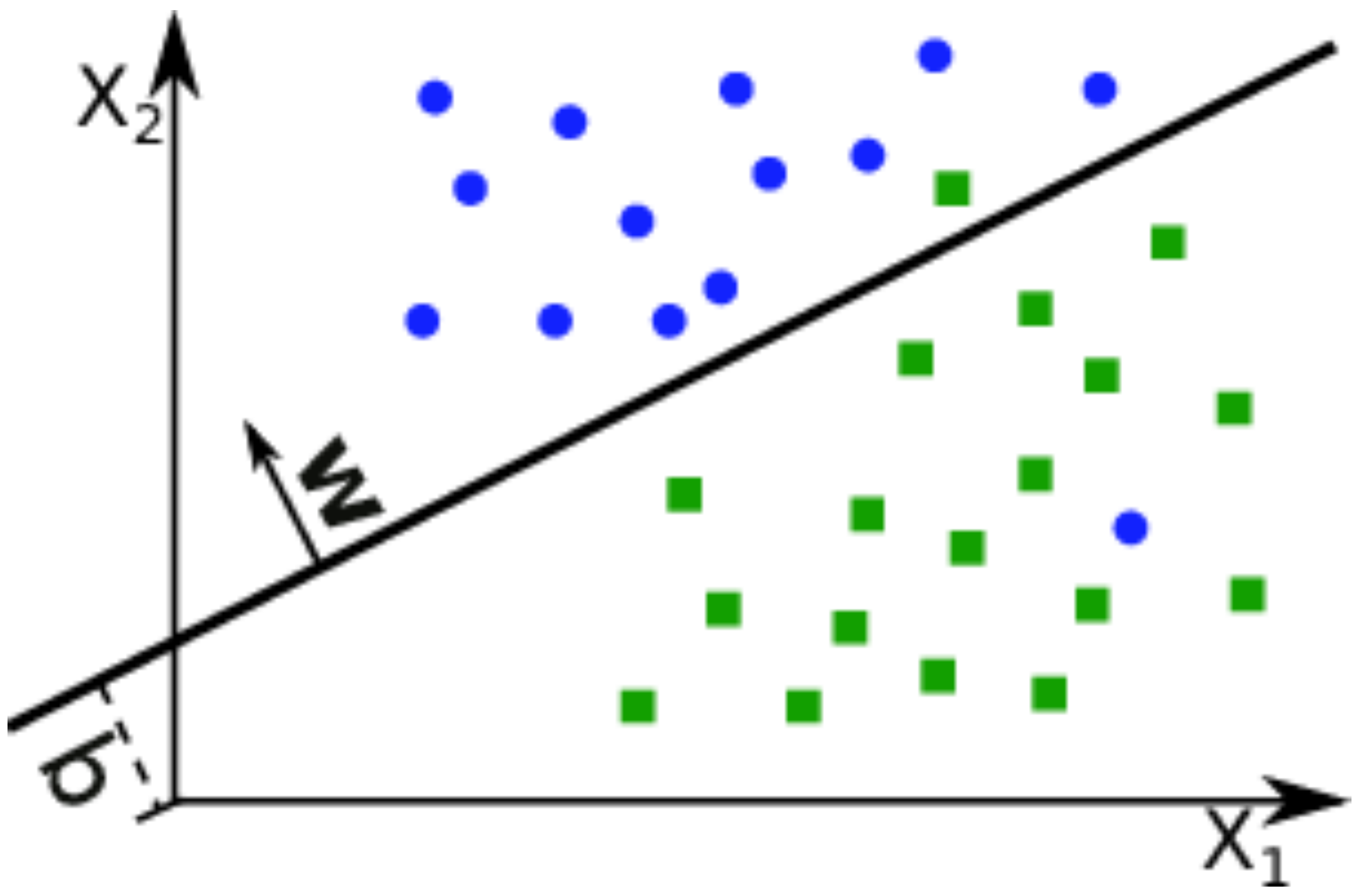
- Logistic Regression
  - Bayesian interpretation
  - Gradient descent on convex function

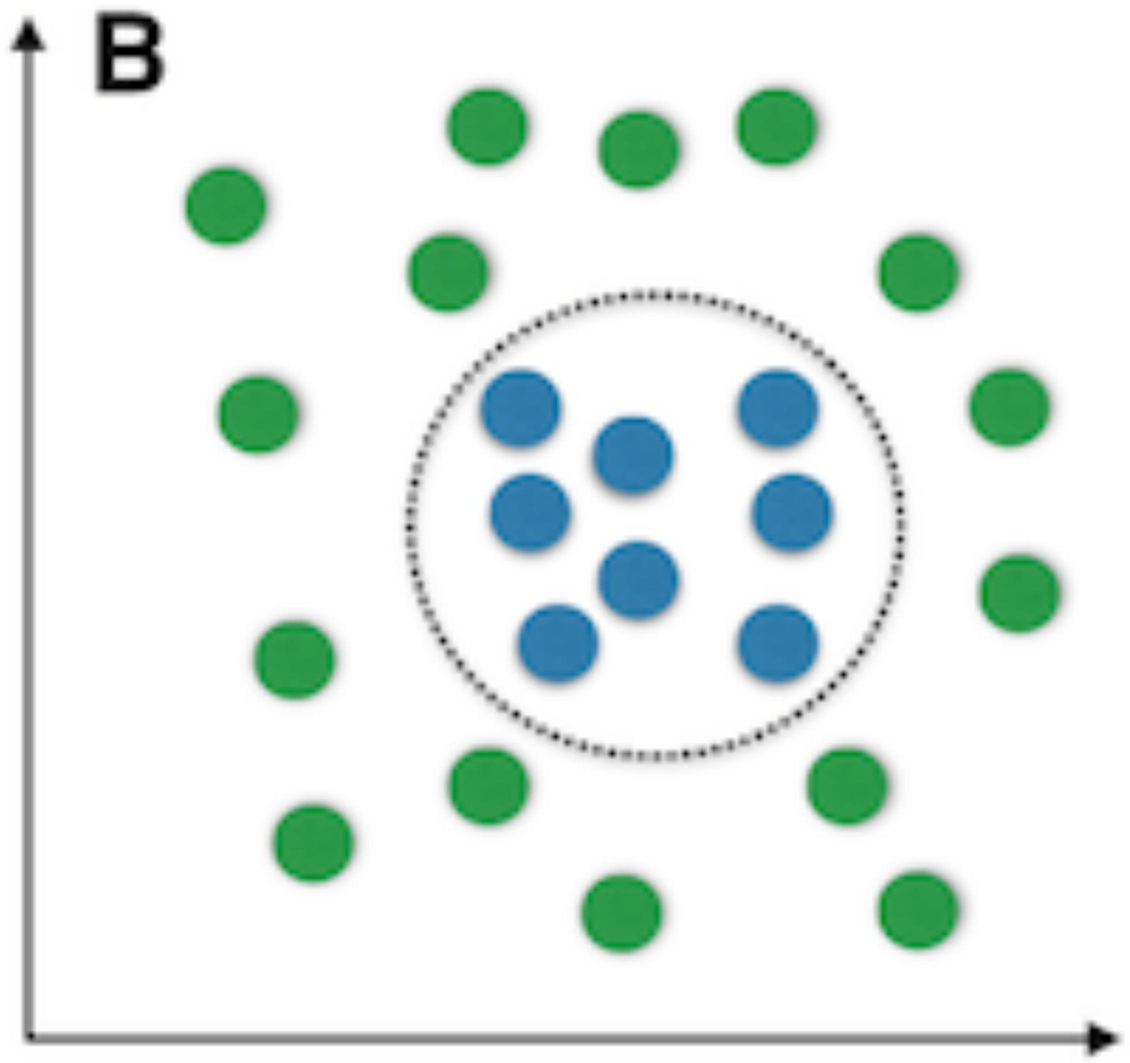
- Support Vector Machine
  - Margin maximization
  - Analytic solution via Lagrangian dual
  - Required. Linearly separable data



Today

- SVMs for handling non-separable data
  - Soft-margin SVM
  - Kernel SVM

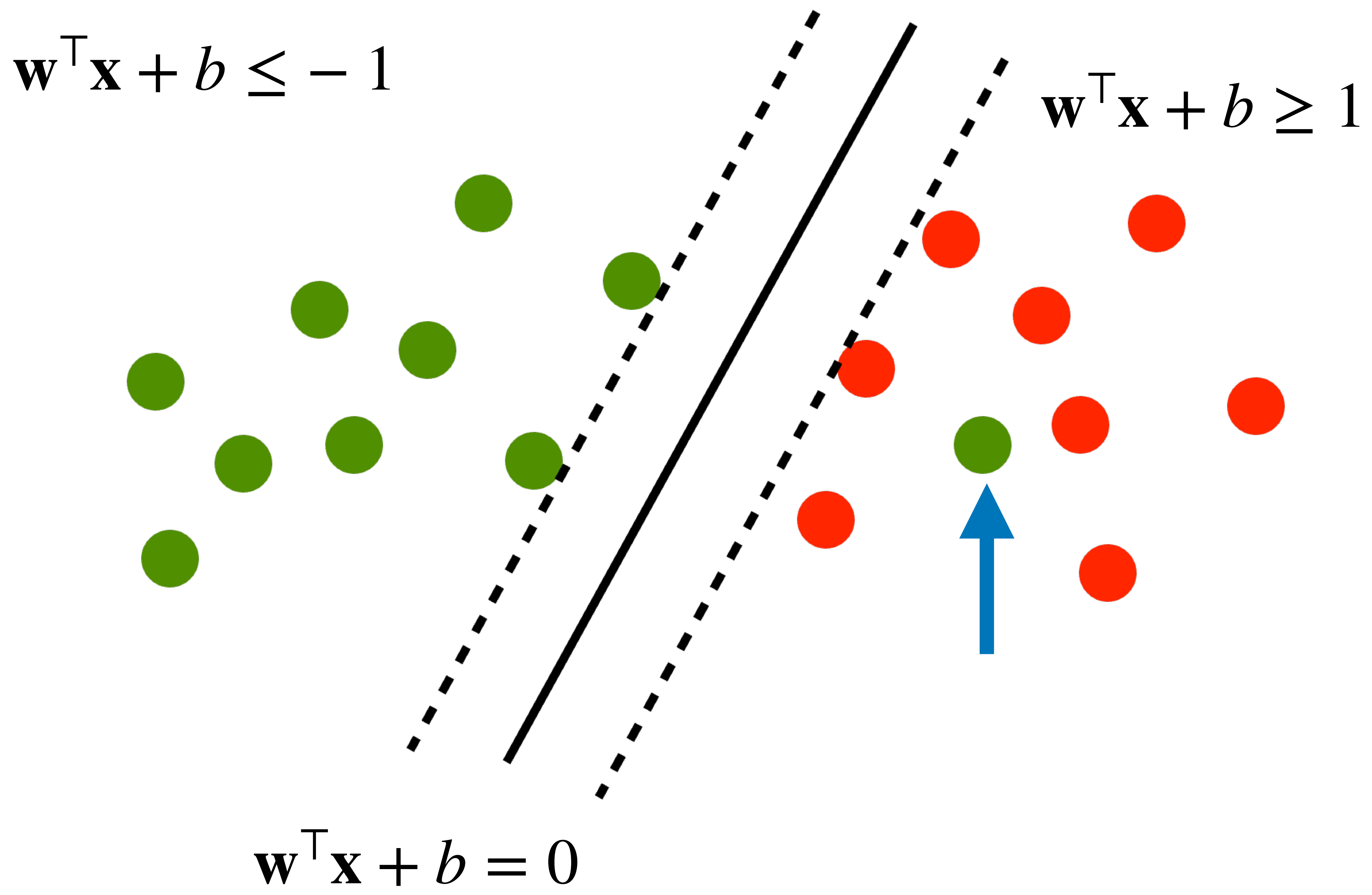




Soft(-Margin) SVM

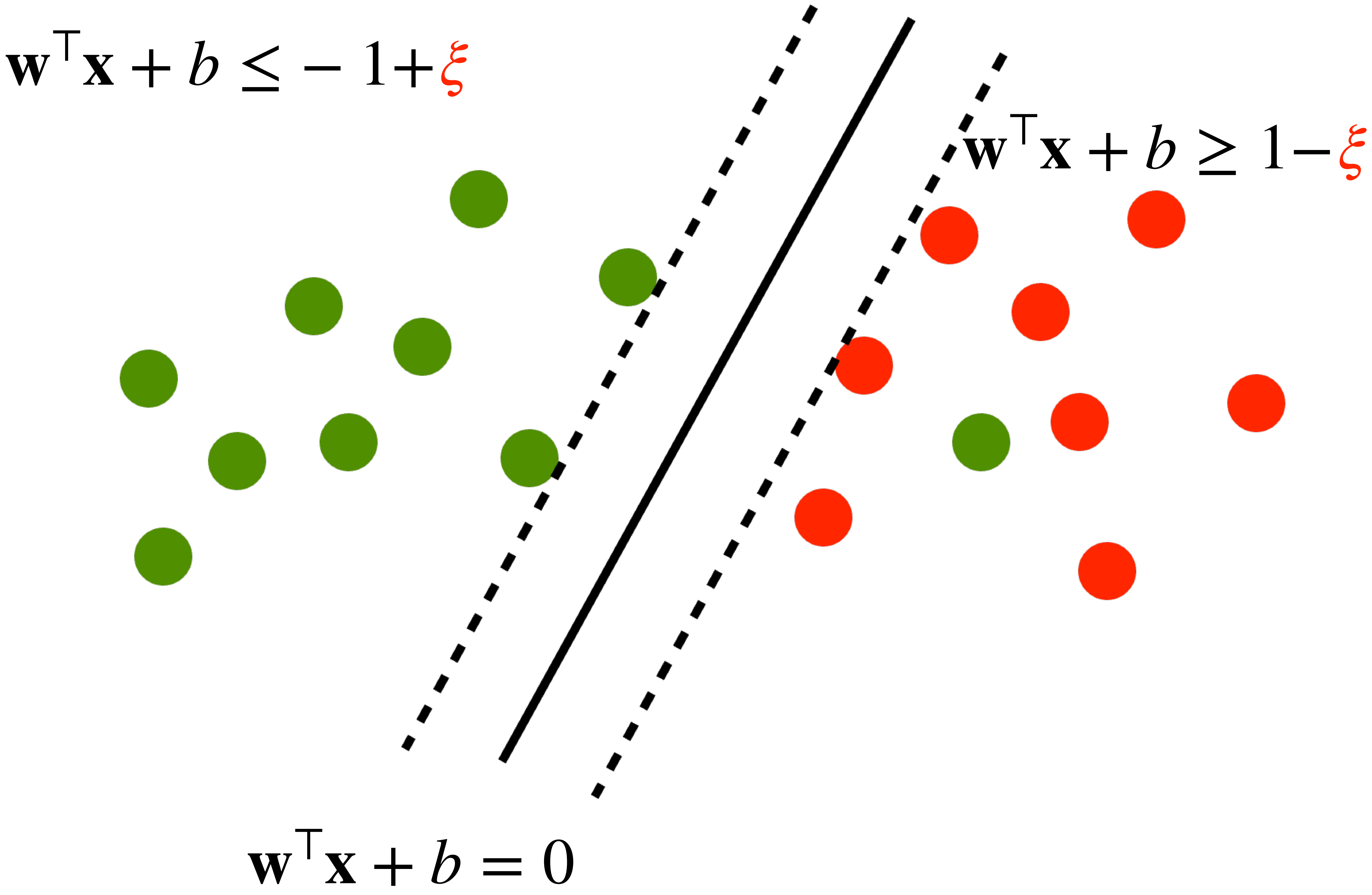
Data with outliers

- Suppose that there exists some **outliers** in data
  - Then, there exists no linear separator
  - Finding a minimum-error separating hyperplane becomes NP-hard (Minsky & Papert, 1969)



Data with outliers

- **Idea.** Handle this by introducing a "slack" ξ
  - i.e., error allowed for each sample
  - We want to (1) maximize the margin, while (2) minimizing the slack



Formulation

- More formally, we solve the optimization problem

$$\ell^* = \min_{\mathbf{w}, b, \xi} \frac{\|\mathbf{w}\|^2}{2} + C \cdot \sum_i \xi_i$$

$$\text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \qquad \xi_i \geq 0$$

- ξ_i: Slack we allow for sample i
- C: Hyperparameter
  - like k in nearest neighbor, or η in gradient descent
  - sending $C \to \infty$ recovers the vanilla SVM

Formulation

$$\ell^* = \min_{\mathbf{w},b,\xi} \frac{\|\mathbf{w}\|^2}{2} + C \cdot \sum_i \xi_i$$

$$\text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \qquad \xi_i \geq 0$$

- We know that this problem is always feasible
  - i.e., the search space is nonempty, regardless of the data drawn
  - Any idea?

Formulation

$$\ell^* = \min_{\mathbf{w},b,\xi} \frac{\|\mathbf{w}\|^2}{2} + C \cdot \sum_i \xi_i$$

$$\text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \qquad \xi_i \geq 0$$

- We know that this problem is always feasible
    - i.e., the search space is nonempty, regardless of the data drawn
    - Any idea?
        - Let $(\mathbf{w}, b) = (\mathbf{0}, 0)$ and $\xi_i = 1$

Optimization

- Again, to solve this constrained optimization, we invoke its dual form

$$\min_{\mathbf{w},b,\xi} \max_{\alpha,\eta\geq 0} \left(\frac{\|\mathbf{w}\|^2}{2} + C\sum_i \xi_i - \sum_i \alpha_i \Big(y_i(\mathbf{x}_i^\top \mathbf{w} + b) + \xi_i - 1 \Big) - \sum_i \eta_i \xi_i \right)$$

  - This time, we have additional dual variables η to handle the nonnegativity constraints on the slack

  - The optimal $(\mathbf{w}, b, \xi)$ is at the saddle point with (α, η)

Optimization

$$\min_{\mathbf{w},b,\xi} \max_{\alpha,\eta} \left(\frac{\|\mathbf{w}\|^2}{2} + C\sum_i \xi_i - \sum_i \alpha_i \big(y_i(\mathbf{x}_i^\top \mathbf{w} + b) + \xi_i - 1 \big) - \sum_i \eta_i \xi_i \right)$$

- Analyzing the derivatives with respect to $(\mathbf{w}, b, \xi)$:

$$\nabla_{\mathbf{w}} \mathcal{L} = \mathbf{w} - \sum \alpha_i y_i \mathbf{x}_i \quad = 0$$

$$\nabla_b \mathcal{L} = \sum \alpha_i y_i \quad = 0$$

$$\nabla_{\xi_i} \mathcal{L} = C - \alpha_i - \eta_i \quad = 0$$

Optimization

$$\min_{\mathbf{w},b,\xi} \max_{\alpha,\eta} \left(\frac{\|\mathbf{w}\|^2}{2} + C \sum_i \xi_i - \sum_i \alpha_i \left(y_i(\mathbf{x}_i^\top \mathbf{w} + b) + \xi_i - 1 \right) - \sum_i \eta_i \xi_i \right)$$

- Analyzing the derivatives with respect to $(\mathbf{w}, b, \xi)$:

$$\nabla_{\mathbf{w}} \mathscr{L} = \mathbf{w} - \sum \alpha_i y_i \mathbf{x}_i \quad = 0$$

(same as in SVM)

$$\nabla_b \mathscr{L} = \sum \alpha_i y_i \quad = 0$$

$$\nabla_{\xi_i} \mathscr{L} = C - \alpha_i - \eta_i \quad = 0$$

$$C = \alpha_i + \eta_i$$

i.e., given α_i, η_i is unique

Optimization

- Via similar steps as in vanilla SVM, we get the Lagrangian

$$-\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + \sum_i \alpha_i - \sum_i \alpha_i \xi_i + C\sum_i \xi_i - \sum_i \eta_i \xi_i$$

- Plugging in the condition $C = \alpha_i + \eta_i$, we get

$$-\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + \sum_i \alpha_i$$

  - Surprisingly, the optimand did not change at all!

Optimization

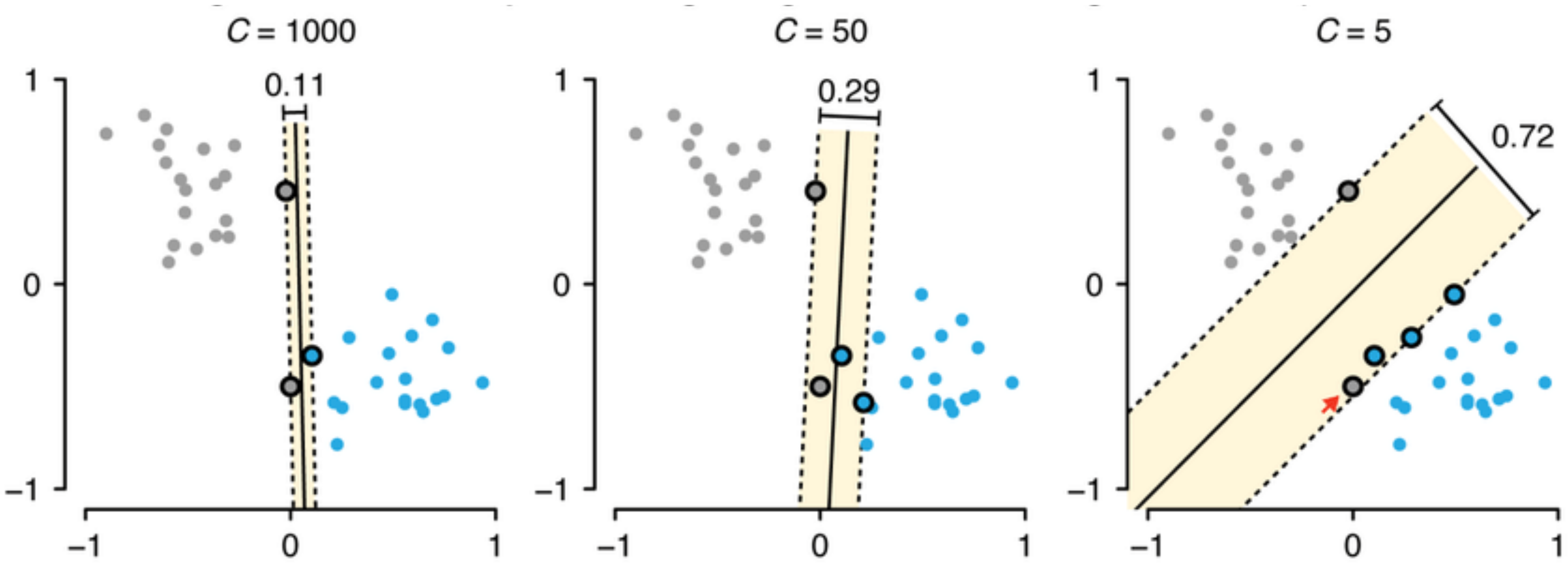
- Softness only matters in terms of the search space:

$$\max_{\alpha} \left(-\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i \right) \quad \text{subject to} \quad \sum_i \alpha_i y_i = 0 \qquad 0 \leq \alpha_i \leq C$$

  - In vanilla SVM, we could have very large α
  - In soft SVM, the maximum size of α is constrained by C
    - Recalling that $\mathbf{w}^* = \sum \alpha_i y_i \mathbf{x}_i$, this means that each datapoint has limited impact on $\mathbf{w}^*$

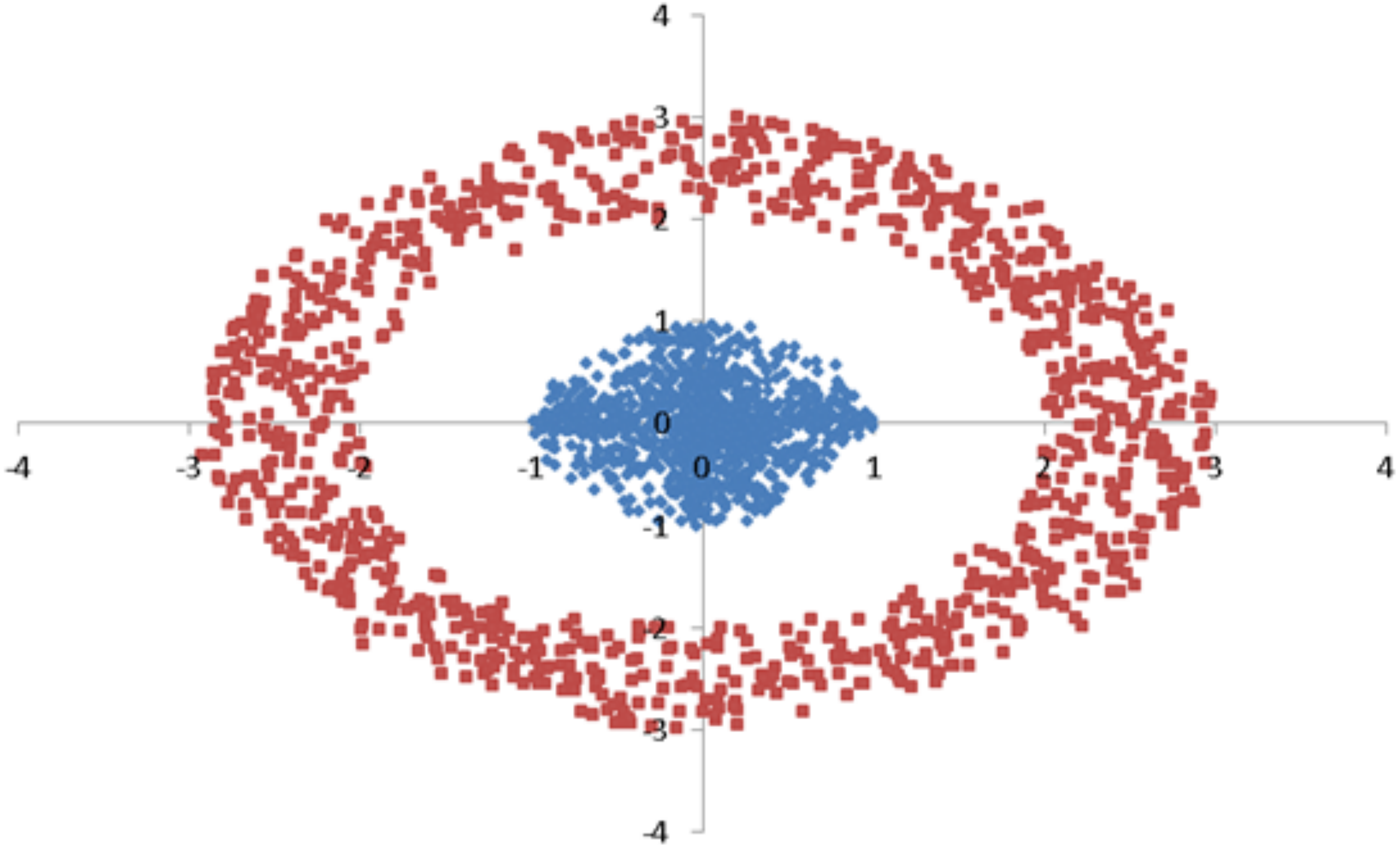
Impact of C

- With larger C, the soft-SVM looks for a smaller slack solution



Limitations

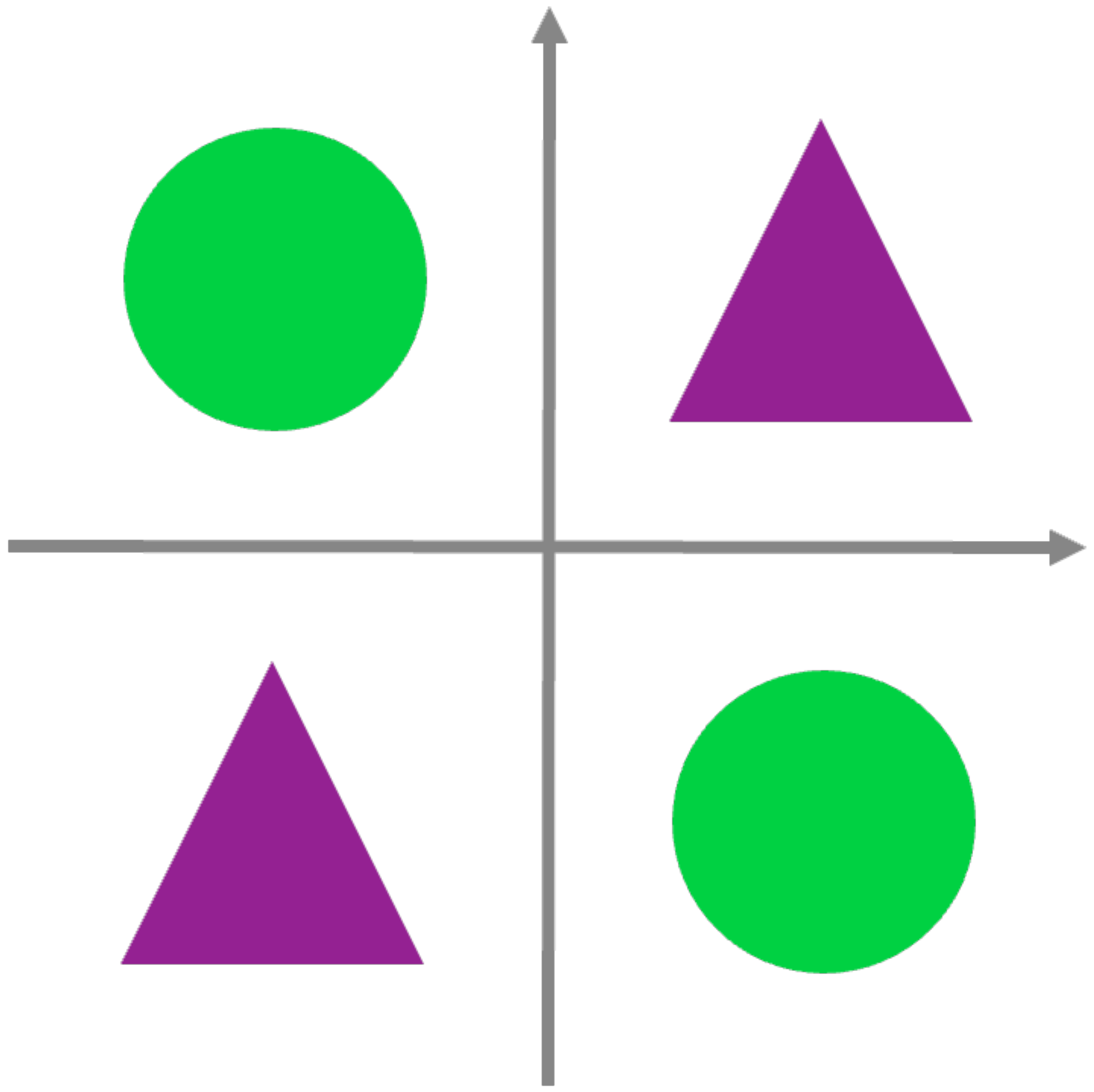
- Still, "allowing some errors" cannot be a fundamental solution for nonlinear data



Kernel SVM

Nonlinear data

- Suppose that we have an **XOR**-like data
  - Not linearly separable
  - Yet highly structured — we can think of nice predictors
- How do we handle this data?

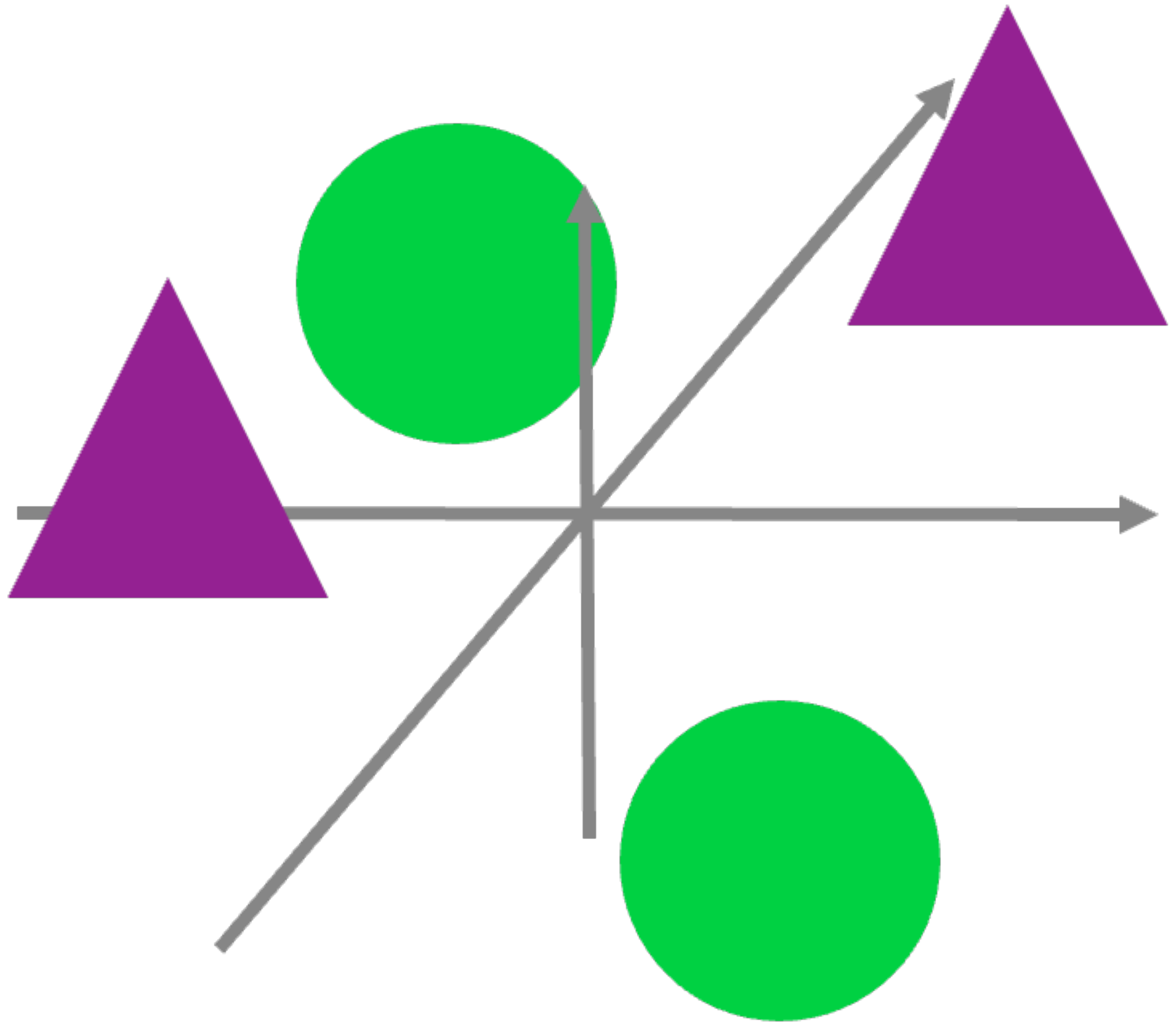


(x_1, x_2)

Nonlinear data

- **Idea.** Map it to a high-dimensional space
  - In the lifted space, there exists a clean linear classifier

$$f(\mathbf{x}) = \text{sign}\left(\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right)$$



$(x_1, x_2, x_1 x_2)$

Formalization

- Formally, we consider mapping data to nonlinear **feature** using

$$\Phi(\,\cdot\,) : \mathbb{R}^d \rightarrow \mathbb{R}^k$$

where, typically $d < k$ (but not necessarily)

- Then, we can consider predictors of the form

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} a_i \cdot \boxed{\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) \rangle} + b \right)$$

  - This form comes from the SVMs, where

$$f(\mathbf{x}) = \text{sign}\left(\sum a_i \cdot \langle \mathbf{x}_i, \mathbf{x} \rangle + b \right)$$

Selecting the feature

- **Question.** How should we choose $\Phi(\cdot)$?
  - Handcrafting (classical & compute-light)
    - Design “good” kernels
    - Test them on data
    - Select the one that works best
  - Data-driven (modern & compute-heavy)
    - Build a parameterized set of kernels
    - Optimize the kernel parameters, jointly with SVM params

Handcrafting the feature

- **Question.** How do we handcraft the feature?

Handcrafting the feature

- **Answer (naïve).** Simply throw in many features,
  and let SVM choose the useful dimension

$$\Phi(\mathbf{x}) = (x_1, \cdots, x_d, x_1x_2, \cdots, x_{d-1}x_d, \cdots, x_k^{100})$$

  - Overfitting to a weird feature
  - Computational cost
    - Both computing $\Phi(\,\cdot\,)$ and computing $\langle\,\cdot\,,\,\cdot\,\rangle$ is expensive

$$f(\mathbf{x}) = \mathrm{sign}\Big(\sum a_i \cdot \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x})\rangle + b\Big)$$

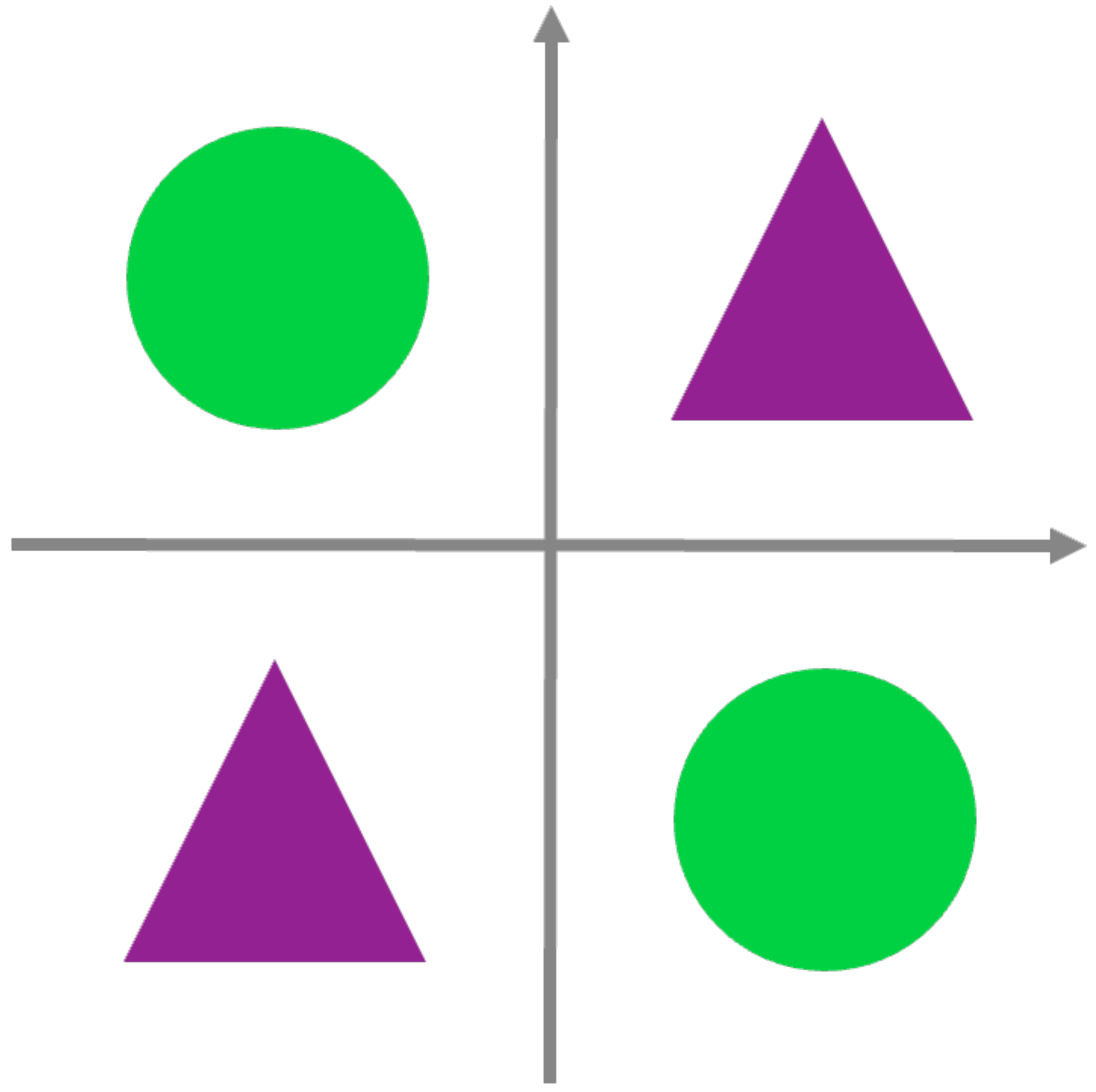
Handcrafting the feature

- Interestingly, some features admit **computational shortcut**

- **Example.** Recall the XOR, and consider two features

$$\Phi_a((x_1, x_2)) = (x_1, x_2, x_1x_2)$$

$$\Phi_b((x_1, x_2)) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$



  - Looks similar…
    - However, one is computationally much better than the other
    - Which one is better?

Handcrafting the feature

$$\Phi_a((x_1, x_2)) = (x_1, x_2, x_1 x_2)$$

$$\Phi_b((x_1, x_2)) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2)$$

- **Answer.** Surprisingly, Φ_b is better!
  - Feature Φ_a

$$\langle \Phi_a(\mathbf{x}), \Phi_a(\mathbf{y}) \rangle = x_1 y_1 + x_2 y_2 + x_1 x_2 y_1 y_2$$

    - Compute 3D features $\phi_{\mathbf{x}} = \Phi_a(\mathbf{x}), \phi_{\mathbf{y}} = \Phi_a(\mathbf{y})$
    - Compute 3D inner prod $\langle \phi_{\mathbf{x}}, \phi_{\mathbf{y}} \rangle$

Handcrafting the feature

$$\Phi_a((x_1, x_2)) = (x_1, x_2, x_1 x_2)$$

$$\Phi_b((x_1, x_2)) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2)$$

- **Answer.** Surprisingly, Φ_b is better!
  - Feature Φ_b

$$\langle \Phi_b(\mathbf{x}), \Phi_b(\mathbf{y}) \rangle = x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 x_2 y_1 y_2 = (\langle \mathbf{x}, \mathbf{y} \rangle)^2$$

    - Compute 2D inner prod $\langle \mathbf{x}, \mathbf{y} \rangle$
    - Take a square $(\cdot)^2$
  - **Reason.** Can compute dot products of features, without actually computing features <- called "kernels"

Kernel SVM

- Inspired by this, the **Kernel SVM** does the following:
  - Choose some similarity metric $K(\,\cdot\,,\cdot\,)$
  - Build predictors of form

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum a_i \cdot K(\mathbf{x}_i, \mathbf{x}) + b\right)$$

  - Optimize a_i, b to fit the training data
    - As in vanilla SVM, will resort to solving

$$\max_{\alpha}\left(-\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{n}\alpha_i\right)$$

    - Simply a quadratic program — use solvers

Kernel SVM

- But can we use any $K(\cdot, \cdot)$?
  - What if there is no corresponding $\Phi(\cdot)$?

Mercer's Theorem

If $K(\cdot, \cdot)$ is a Mercer kernel, then there always exists $\Phi(\cdot)$ such that

$$K(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$$

- Thus, as long as K is well-behaved, it is a valid SVM

Kernel SVM

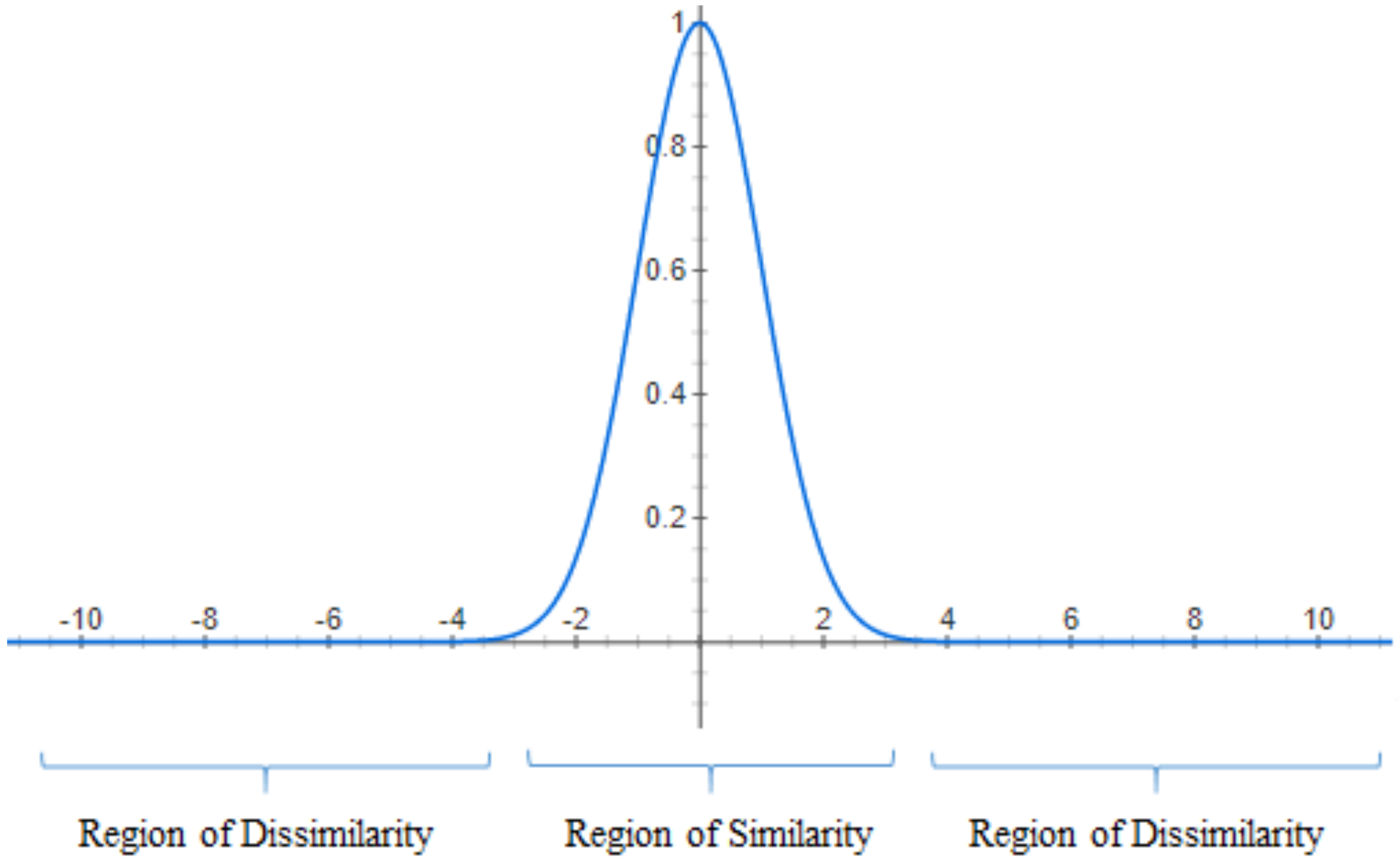
Definition (Mercer Kernel)

A real-valued function $K(\cdot\,,\cdot)$ is a Mercer kernel, if

- $K(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}', \mathbf{x})$ i.e., symmetric
- $\lim_{n\to\infty} K(\mathbf{x}^{(n)}, \mathbf{x}) \to K\left(\lim_{n\to\infty} \mathbf{x}^{(n)}, \mathbf{x}\right)$ i.e., continuous
- $\sum_{i,j} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0, \quad \forall \alpha_i, \alpha_j, \mathbf{x}_i, \mathbf{x}_j$ i.e., positive-semidefinite

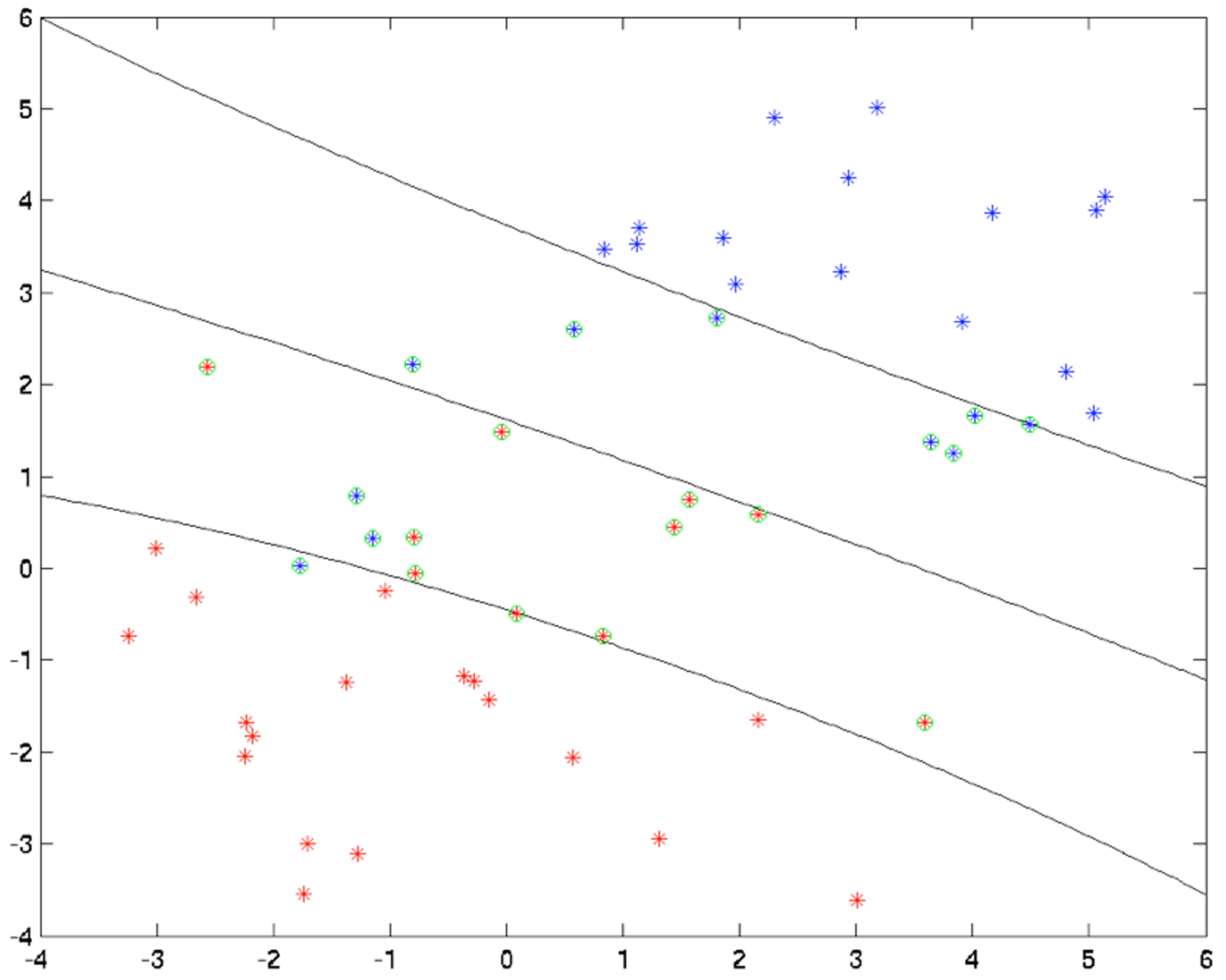
Kernels for kernel SVM

- Here are some popular kernels:
  - Gaussian RBF $K(\mathbf{x}, \mathbf{x}') = \exp(-\lambda \|\mathbf{x} - \mathbf{x}'\|_2^2)$
  - Laplacian RBF $K(\mathbf{x}, \mathbf{x}') = \exp(-\lambda \|\mathbf{x} - \mathbf{x}'\|_2)$
  - Polynomial $K(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + c)^d$
  - B-Spline (…)

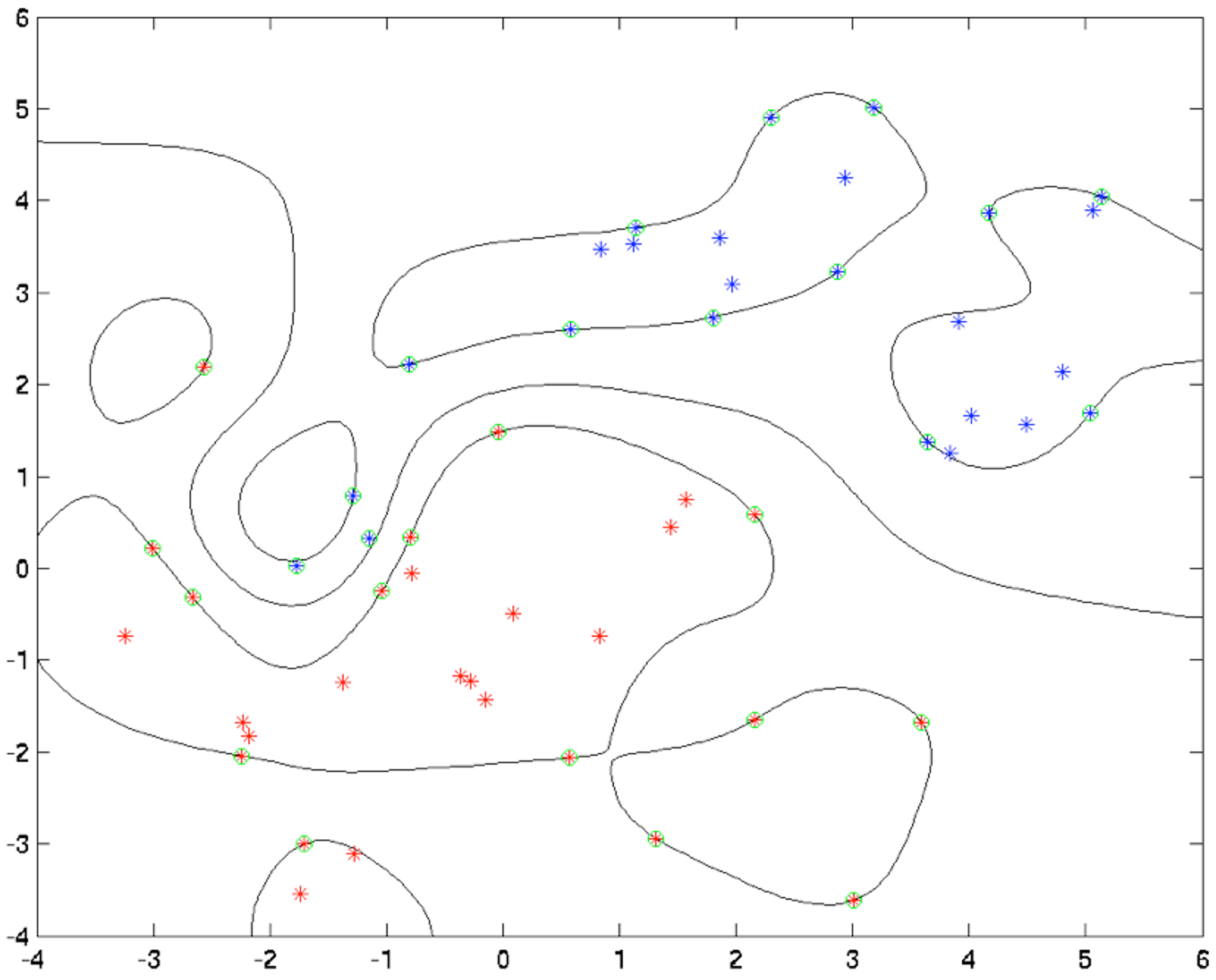


Kernels for kernel SVM

- For Gaussian kernel ($\exp(-\lambda\|\mathbf{x}-\mathbf{x}'\|_2^2)$), large λ means narrower region of similarity
  - Thus more fine-grained decision boundary



Small λ



Large λ

Wrapping up

- In large-scale ML, we usually model $\Phi(\,\cdot\,)$ using neural nets, and tune its parameters with data
  - Expensive, but we now have GPUs for compute
  - Conduct logistic regression, instead of SVD
    - Ease of joint training
    - When train long enough, tend to maximize margin
  - Use nice augmentations to find good similarity metrics such that

$$\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}_{\text{aug}}) \rangle \gg \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$$

Next up

- Unsupervised learning — K-means!

</lecture 6>