Do not distribute course material

Lecture
Support Vector Machines continued

PROF. LINDA SELLIE

SOME SLIDES FROM PROF. RANGAN

You are not responsible for the material covered in class on March 29th

Outline

- Notation change, intuition, and finding how to compare hyperplanes - mathematically how do compare hyperplanes to find the one with the maximum margin. Can we turn this way of comparing hyperplanes into an objective function
- Support vector machines
  - ★ hard margin - find the objective function when the data is linearly separable
  - ★ Dealing with non-linear data - "Soft" margins for SVM - New objective function for the case where the data is not linearly separable
  - ★ dual formula - a clever trick $g(\mathbf{x}) = \text{sign}\left(w_0 + \sum_{i \in I} \alpha^{(i)} y^{(i)} \underbrace{\phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x})}_{K(\mathbf{x}^{(i)}, \mathbf{x}) = \Phi(\mathbf{x}^{(i)})^T \Phi(\mathbf{x})}\right) = \text{sign}\left(w_0 + \sum_{i \in I} \alpha^{(i)} y^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x})\right)$
  - ★ Dealing with non-linear data - feature transformation with the kernel trick - Show two popular feature maps
- Multiclass

SVMs

- Maximizes distance of training data to boundary (built in regularization)
- Generalizes to nonlinear decision boundaries (I.e. feature transformation)
- Performs well with high dimensional data

Duel formula **motivation**: Polynomial Kernel

$$g(\mathbf{x}) = \text{sign}\left(\sum_{i \in I} \alpha^{(i)} y^{(i)} (\mathbf{x}^{(i)T} \mathbf{x}) + w_0\right)$$

$$\mathbf{x} = [x_1, x_2]^T$$

$$\Phi_2 : R^2 \to R^6$$

$$\Phi_2(\mathbf{x}) = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T$$

$$\Phi_2(\mathbf{x})^T \Phi_2(\mathbf{x}') = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2] \cdot [1, x_1', x_2', x_1' x_2', x_1'^2, x_2'^2]$$

$$\Phi_2(\mathbf{x})^T \Phi_2(\mathbf{x}') = 1 + x_1 x_1' + x_2 x_2' + x_1 x_2 x_1' x_2' + x_1^2 x_1'^2 + x_2^2 x_2'^2$$

$$\Phi_2(-0.75, -0.25) = (1, -0.75, -0.25, (-0.75)\cdot(-0.25), (-0.75)^2, (-0.25)^2)$$

$$\Phi_2(-1,1) = (1, -1, 1, -1\cdot 1, (-1)^2, 1^2)$$

$$\Phi_2(-0.75, -0.25) \cdot \Phi_2(-1,1) = 1 + 0.75 - 0.25 + 0.75 \cdot (-0.25) + (-0.75)^2 + (-0.25)^2$$

But if we use a different (but similar) transformation:

$$\Phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2]^T$$

$$K(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}') = 1 + 2x_1 x_1' + 2x_2 x_2' + 2x_1 x_2 x_1' x_2' + x_1^2 x_1'^2 + x_2^2 x_2'^2$$

$$= (1 + \mathbf{x} \cdot \mathbf{x}')^2$$

$$\Phi(-0.75, -0.25) = \left(1, \sqrt{2}\cdot(-0.75), \sqrt{2}\cdot(-0.25), \sqrt{2}(-0.75)\cdot(-0.25), (-0.75)^2, (-0.25)^2\right)$$

$$\Phi(-1,1) = (1, -\sqrt{2}, \sqrt{2}, -\sqrt{2}, (-1)^2, 1^2)$$

$$K((-1,1), (-0.75, -0.25)) = \underline{2.25}$$

Class exercise

$$= 1 + 2(0.75) + 2(-0.25) - 2(0.75)(0.25) + (-0.75)^2 + (-0.25)^2$$

$$= \left(1 + (0.75 - 0.25)\right)^2$$

So, instead of computing $\Phi(\mathbf{x})^T \Phi(\mathbf{x}')$,
we compute $\mathbf{x} \cdot \mathbf{x}'$ in the original feature space, and then compute $(1 + \mathbf{x} \cdot \mathbf{x}')^2$

The polynomial kernel can be generalized to higher dimensions. A degree k polynomial is $K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x} \cdot \mathbf{x}')^k$

Learn more at: https://en.wikipedia.org/wiki/Polynomial_kernel

Duel formula **motivation**: Polynomial Kernel

$$g(\mathbf{x}) = \text{sign}\left(\sum_{i \in I} \alpha^{(i)} y^{(i)} (\mathbf{x}^{(i)T} \mathbf{x}) + w_0\right)$$

$$\mathbf{x} = [x_1, x_2]^T$$

$$\Phi_2 : R^2 \rightarrow R^6$$

$$\Phi_2(\mathbf{x}) = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T$$

$$\Phi_2(\mathbf{x})^T \Phi_2(\mathbf{x}') = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2] \cdot [1, x_1', x_2', x_1' x_2', x_1'^2, x_2'^2]$$

$$\Phi_2(\mathbf{x})^T \Phi_2(\mathbf{x}') = 1 + x_1 x_1' + x_2 x_2' + x_1 x_2 x_1' x_2' + x_1^2 x_1'^2 + x_2^2 x_2'^2$$

If d=200 then in z-space we have a feature vector of size approx 20,000

$$\Phi_2(-0.75, -0.25) = (1, -0.75, -0.25, (-0.75)\cdot(-0.25), (-0.75)^2, (-0.25)^2)$$

$$\Phi_2(-1,1) = (1, -1, 1, -1\cdot 1, (-1)^2, 1^2)$$

$$\Phi_2(-0.75, -0.25) \cdot \Phi_2(-1,1) = 1 + 0.75 - 0.25 + 0.75 \cdot (-0.25) + (-0.75)^2 + (-0.25)^2$$

But if we use a different (but similar) transformation:

$$\Phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2]^T$$

$$K(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}') = 1 + 2x_1 x_1' + 2x_2 x_2' + 2x_1 x_2 x_1' x_2' + x_1^2 x_1'^2 + x_2^2 x_2'^2$$

$$= (1 + \mathbf{x} \cdot \mathbf{x}')^2$$

If d=200 then in z-space we have a feature vector of size approx 20,000, but we do roughly the same amount of work to compute K(**x**,**x'**) as if we hadn't transformed date features

$$\Phi(-0.75, -0.25) = \left(1, \sqrt{}\right.$$

$$\Phi(-1,1) = (1, -\sqrt{2}, \sqrt{2}, -$$

$$K((-1,1), (-0.75, -0.25)) =$$

2.25

Class exercise

$$= 1 + 2(0.75) + 2(-0.25) - 2(0.75)(0.25) + (-0.75)^2 + (-0.25)^2$$

$$= (1 + (0.75 - 0.25))^2$$

So, instead of computing $\Phi(\mathbf{x})^T \Phi(\mathbf{x}')$,
we compute $\mathbf{x} \cdot \mathbf{x}'$ in the original feature space, and then compute $(1 + \mathbf{x} \cdot \mathbf{x}')^2$

The polynomial kernel can be generalized to higher dimensions. A degree k polynomial is $K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x} \cdot \mathbf{x}')^k$

Learn more at: https://en.wikipedia.org/wiki/Polynomial_kernel

Duality

Transforming our constrained objective function into a new objective function. (Thus we don't solve the original objective function)

- New objective function scales well with high-dimensional data
- Our hypothesis will be a function of the training examples (thus we can use the kernel function)

For SVM's the original constrained objective function (primal problem) and the dual formulation are equivalent problems. (This is not true in general when transforming a problem into its dual.)

Learning in a transformed feature space:

Pair share: if we use $\Phi(\mathbf{x})$, do we need to actually perform the transformation?

Primal

$$\min_{\mathbf{w},w_0} \|\mathbf{w}\|_2^2$$

subject to: $y^{(i)}(\mathbf{w}^T\Phi(\mathbf{x}^{(i)}) + w_0) \geq 1$ for all $i = 1 \ldots N$

Dual

$$\min_{\alpha} \sum_{i,j} \alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)} K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) - \sum_i \alpha^{(i)}$$

$$\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0 \qquad \alpha^{(i)} \geq 0$$

We only need to know the result of $\Phi(\mathbf{x}^{(j)})^T\Phi(\mathbf{x}^{(i)})$

Prediction

$$g(\mathbf{x}) = \text{sign}\left(\mathbf{w}^T\Phi(\mathbf{x}) + w_0\right)$$

$$g(\mathbf{x}) = \text{sign}\left(\sum_{i \in I} \alpha^{(i)}y^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x}) + w_0\right)$$

We only need to know the result of $\Phi(\mathbf{x}^{(i)})^T\Phi(\mathbf{x})$

Support Vector $\mathbf{x}^{(i)}$

By computing the value of K without writing the transformation, we can have a computational advantage

Learning in a transformed feature space:

Pair share: if we use $\Phi(\mathbf{x})$, do we need to actually perform the transformation?

Primal

$$\min_{\mathbf{w},w_0} \|\mathbf{w}\|_2^2$$

subject to: $y^{(i)}(\mathbf{w}^T\Phi(\mathbf{x}^{(i)}) + w_0) \geq 1$ for all $i = 1 \ldots N$

Dual

$$\min_{\alpha} \sum_{i,j} \alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)} K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) - \ldots$$

$$\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0 \qquad \alpha^{(i)} \geq 0$$

Kernel trick: evaluate $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ without competing $\Phi(\mathbf{x}^{(j)})$ or $\Phi(\mathbf{x}^{(i)})$

We only need to know the result of $\Phi(\mathbf{x}^{(j)})^T\Phi(\mathbf{x}^{(i)})$

Prediction

$$g(\mathbf{x}) = \text{sign}\left(\mathbf{w}^T\Phi(\mathbf{x}) + w_0\right)$$

$$g(\mathbf{x}) = \text{sign}\left(\sum_{i \in I} \alpha^{(i)}y^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x}) + w_0\right)$$

We only need to know the result of $\Phi(\mathbf{x}^{(i)})^T\Phi(\mathbf{x})$

Support Vector $\mathbf{x}^{(i)}$

By computing the value of K without writing the transformation, we can have a computational advantage

Lagrangian (hard margin case)

We will transform the hard margin case, the soft margin case is similar. We will assume the data is linearly separable

Primal SVM

- $\min_{\mathbf{w},w_0,\xi} \frac{1}{2}\|\mathbf{w}\|_2^2$

  Subject to $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) \geq 1$ for all $i = 1 \ldots N$

Lagrangian

- $L(\mathbf{w}, w_0, \alpha^{(i)}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \alpha^{(i)} \left(1 - y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0)\right)$

  where $\alpha^{(i)} \geq 0$

Primal variables: $\mathbf{w}, w_0$
Dual variables: $\alpha^{(i)}, i \in \{1, \ldots, N\}$
(Lagrangian multipliers)

Dual formulation of SVM

$$\max_{\alpha^{(i)}, i \in \{1,\ldots,N\}} \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i,j} y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} \mathbf{x}^{(i)T} \mathbf{x}^{(j)}$$

subject to $\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$ and $\alpha^{(i)} \geq 0$

$$\mathbf{w} = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}$$

Find $\alpha^{(j)} \neq 0$ solve

$$w_0 = y^{(j)} - \mathbf{w}^T \mathbf{x}^{(j)}$$

Steps:

* Minimize the Lagrangian function over the primal variables $\mathbf{w}, w_0$ by setting $\nabla_{\mathbf{w}} L = 0, \frac{\partial L}{\partial w_0} = 0.$
* Solve for primal variables wrt dual variables.
* Remove primal variables in the Lagrangian by substituting their equivalent dual variables

Lagrangian (hard margin case)

We will transform the hard margin case, the soft margin case is similar. We will assume the data is linearly separable

Primal SVM

- $\min_{\mathbf{w},w_0,\xi} \frac{1}{2}\|\mathbf{w}\|_2^2$

  Subject to $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) \geq 1$ for all $i = 1 \ldots N$

Lagrangian

- $L(\mathbf{w}, w_0, \alpha^{(i)}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \alpha^{(i)} \left(1 - y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0)\right)$

  where $\alpha^{(i)} \geq 0$

Primal variables: $\mathbf{w}, w_0$
Dual variables: $\alpha^{(i)}, i \in \{1, \ldots, N\}$
(Lagrangian multipliers)

Dual formulation of SVM

$$\max_{\alpha^{(i)}, i \in \{1,\ldots,N\}} \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i,j} y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} \mathbf{x}^{(i)T} \mathbf{x}^{(j)}$$

subject to $\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0$ and $\alpha^{(i)} \geq 0$

Steps:
* Minimize the Lagrangian function
ariables $\mathbf{w}, w_0$ by
), $\frac{\partial L}{\partial w_0} = 0.$
variables wrt dual

riables in the
ostituting their
ariables

Three key ideas from transformation you need to know (you don't need to know how the transformation works):

1. $\alpha^{(i)}(1 - y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + w_0)) = 0$
2. $\mathbf{w} = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}$
3. Find $\alpha^{(j)} \neq 0$ solve $w_0 = y^{(j)} - \mathbf{w}^T\mathbf{x}^{(j)}$

Recovering $\mathbf{w}, w_0$

Dual formulation returns $\alpha = [\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(N)}]^T$

Recovering **w**:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} = \sum_{i \in I} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}$$

Only depends on support vectors! ($\alpha^{(i)} > 0$)

$I = \{i \mid \alpha^{(i)} \neq 0\}$

If $\alpha^{(i)} \neq 0$ then $x^{(i)}$ is a support vector,

The number of nonzero $\alpha^{(i)} \neq 0$ could be less than d

Recovering w_0: There exists some $\alpha^{(j)} \neq 0$

$$\alpha^{(j)} \underbrace{\left(y^{(j)} (w_0 + \mathbf{w}^T \mathbf{x}^{(j)}) - 1 \right)}_{0} = 0$$

Solve for w_0: $y^{(j)}(w_0 + \mathbf{w}^T \mathbf{x}^{(j)}) = 1$

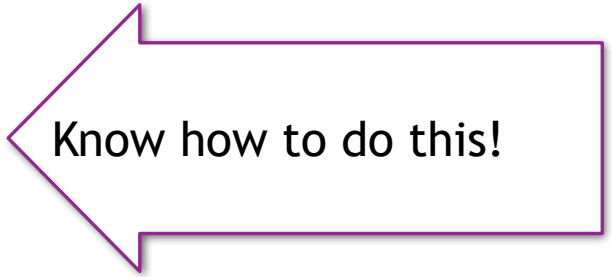
$$w_0 = y^{(j)} - \mathbf{w}^T \mathbf{x}^{(j)} = y^{(j)} - \sum_{i \in I} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \mathbf{x}^{(j)}$$

Approach taken from Learning from Data

Creating the Dual SVM

Another objective function for finding the maximum margin hyperplane.

- Steps to create the dual SVM
  - Create the Lagrangian
  - Minimize the Lagrangian over the primal variables by taking the gradient and setting the gradient to zero (thus we can write the primal variables in terms of the dual variables)
  - Substitute the primal variables with their equivalent dual formulas
  - Maximize the Lagrangian
  - Find $\mathbf{w}, w_0$ from dual variables



Lagrangian

Example: Given training examples (x^T,y):

((1, 2.5),1), ((2, 2),1), ((1,1),-1)

$$\min_{w_0,w_1,w_2} \frac{1}{2}[w_1, w_2]\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

Subject to:

$$y^{(1)}\left(w_0 + w_1 x_1^{(1)} + w_2 x_2^{(1)}\right) \geq 1,$$

$$y^{(2)}\left(w_0 + w_1 x_1^{(2)} + w_2 x_2^{(2)}\right) \geq 1,$$

$$y^{(3)}\left(w_0 + w_1 x_1^{(3)} + w_2 x_2^{(3)}\right) \geq 1$$

$$\min_{w_0,w_1,w_2} \frac{1}{2}[w_1, w_2]\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

Subject to:

$$\left(1-(w_0 + 1w_1 + 2.5w_2)\right) \leq 0$$

$$\left(1 - \left(w_0 + 2w_1 + 2w_2\right)\right) \leq 0$$

$$\left(1 - \left(-w_0 + -1w_1 + -1w_2\right)\right) \leq 0$$

The constrained quadratic optimization function can be rewritten as:

$$\min_{w_0,w_1,w_2} \max_{\alpha^{(1)}\geq 0} \max_{\alpha^{(2)}\geq 0} \max_{\alpha^{(3)}\geq 0} \frac{1}{2}[w_1, w_2]\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \alpha^{(1)}\left(1-(w_0 + 1w_1 + 2.5w_2)\right) + \alpha^{(2)}\left(1 - \left(w_0 + 2w_1 + 2w_2\right)\right) + \alpha^{(3)}\left(1 - \left(-w_0 + -1w_1 + -1w_2\right)\right)$$

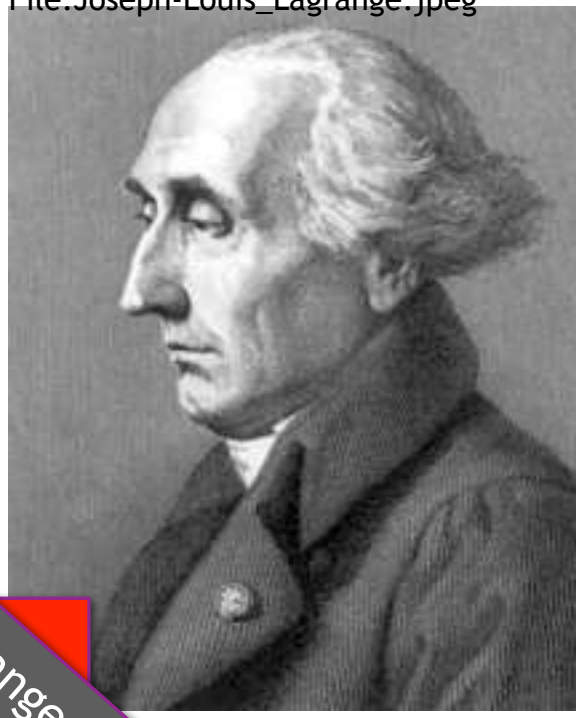
$$L(\mathbf{w}, w_0, \alpha) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \alpha^{(i)}\left(1 - y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)})\right) \qquad L(\mathbf{w}, w_0, \alpha) = \frac{1}{2}\left\|\mathbf{w}\right\|^2 - \sum_{i=1}^{N} \alpha^{(i)}\left(y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) - 1\right)$$

Background on solving constrained optimization

LAGRANGE DUALITY

Background: The Lagrangian

https://commons.wikimedia.org/wiki/File:Joseph-Louis_Lagrange.jpeg



Simplified problem

minimize **u:** $\frac{1}{2}\mathbf{u}^T\mathbf{u}$ — primal problem

subject to: $\mathbf{a}^T\mathbf{u} \geq c$ where $\mathbf{a}^T\mathbf{u} \geq c$ is $a_1u_1 + a_2u_2 + \cdots + a_ru_r > c$

$\mathbf{a}'^T\mathbf{u} \geq c'$

If there is a *valid* solution, this is equal to: (c-aTu) <=0 otherwise

minimize **u:** $\frac{1}{2}\mathbf{u}^T\mathbf{u} + \max_{\alpha \geq 0} \alpha(c - \mathbf{a}^T\mathbf{u}) + \max_{\alpha' \geq 0} \alpha'(c' - \mathbf{a}'^T\mathbf{u})$

No constraint on **u**!

First minimize with unconstrained **u** then maximize wrt α

Lagrange dual problem

The Lagrangian function

$$L(\mathbf{u}, \alpha) = \frac{1}{2}\mathbf{u}^T\mathbf{u} + \alpha(c - \mathbf{a}^T\mathbf{u}) + \alpha'(c' - \mathbf{a}'^T\mathbf{u})$$

The optimization function

$$\min_{\mathbf{u}} \max_{\alpha \geq 0} L(\mathbf{u}, \alpha)$$

If there exists a solution to the primal problem which is a quadratic optimization with linear constraints there is ***strong duality:*** $\min_{\mathbf{u}} \max_{\alpha \geq 0} L(\mathbf{u}, \alpha) = \max_{\alpha \geq 0} \min_{\mathbf{u}} L(\mathbf{u}, \alpha)$

SVM's can appear magical because we can transform the feature space $\Phi(\mathbf{x}^{(i)})$ (e.g. polynomial, RBF) and never have to compute $\Phi(\mathbf{x}^{(i)})$, instead we only need to compute the kernel $K(\mathbf{x}^{(i)},\mathbf{x}^{(j)}) = \Phi(\mathbf{x}^{(i)})^T \Phi(\mathbf{x}^{(j)})$ (In the description below we will not transform the feature space to keep the explanation clearer - but you can replace each $\mathbf{x}^{(i)}$ by $\Phi(\mathbf{x}^{(i)})$ yourself)

To show how this works, we transform our original problem into its duel formalization.

Given our constrained quadratic optimization problem: $\min \frac{1}{2}\|\mathbf{w}\|_2^2$ subject to $y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) \geq 1$ for all i=1,...,N

We create the Lagrangian (each constraint is multiplied by a Lagrangian multiplier $\alpha^{(i)}$, for i = 1,...,N)

$$L(\mathbf{w}, w_0, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N}\alpha^{(i)}\left(y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) - 1\right) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \mathbf{w}^T\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\mathbf{x}^{(i)} - w_0\sum_{i=1}^{N}\alpha^{(i)}y^{(i)} + \sum_{i=1}^{N}\alpha^{(i)} \qquad \alpha^{(i)} \geq 0$$

The original problem is now the same as minimization of L(**w**, w_0, α). We first minimize with respect to **w**, w_0, ($\alpha = [\alpha^{(1)}, \ldots, \alpha^{(N)}]$ is fixed). We take the gradient and find the optimal by setting the gradient to zero

$$\nabla_{\mathbf{w}} L(\mathbf{w}, w_0, \alpha) = \mathbf{w} = \sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\mathbf{x}^{(i)} \qquad \nabla_{w_0} L(\mathbf{w}, w_0, \alpha) = -\sum_{i=1}^{N}\alpha^{(i)}y^{(i)} = 0$$

w is a linear sum of the training examples

Plugging these values back into L(**w**, w_0, α) we get the duel formalization

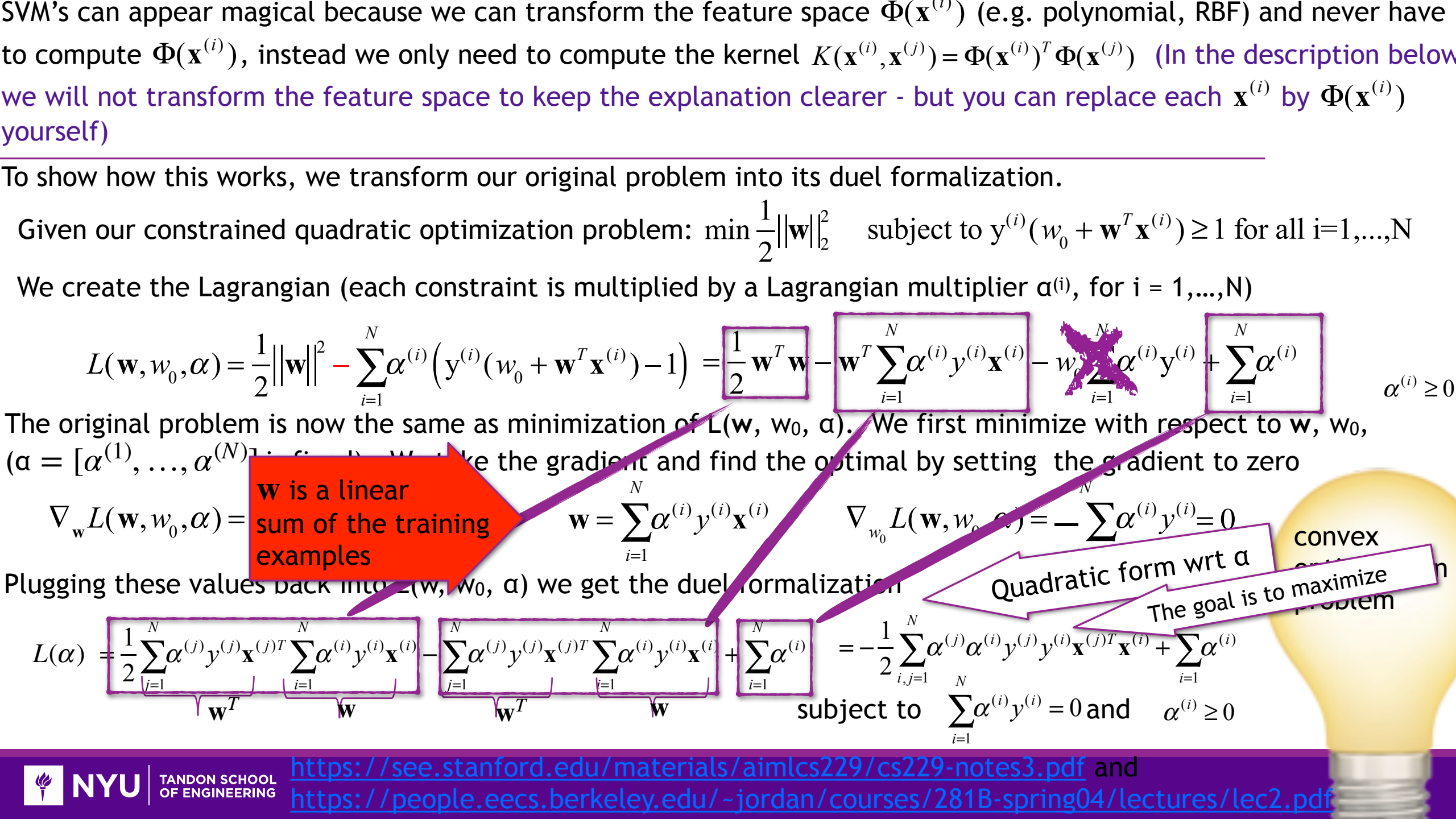
$$L(\alpha) = \frac{1}{2}\underbrace{\sum_{j=1}^{N}\alpha^{(j)}y^{(j)}\mathbf{x}^{(j)T}}_{\mathbf{w}^T}\underbrace{\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\mathbf{x}^{(i)}}_{\mathbf{w}} - \underbrace{\sum_{j=1}^{N}\alpha^{(j)}y^{(j)}\mathbf{x}^{(j)T}}_{\mathbf{w}^T}\underbrace{\sum_{i=1}^{N}\alpha^{(i)}y^{(i)}\mathbf{x}^{(i)}}_{\mathbf{w}} + \sum_{i=1}^{N}\alpha^{(i)} = -\frac{1}{2}\sum_{i,j=1}^{N}\alpha^{(j)}\alpha^{(i)}y^{(j)}y^{(i)}\mathbf{x}^{(j)T}\mathbf{x}^{(i)} + \sum_{i=1}^{N}\alpha^{(i)}$$

Quadratic form wrt α

The goal is to maximize

convex optimization problem

subject to $\sum_{i=1}^{N}\alpha^{(i)}y^{(i)} = 0$ and $\alpha^{(i)} \geq 0$



https://see.stanford.edu/materials/aimlcs229/cs229-notes3.pdf and
https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec2.pdf

Now we can send this to a quadratic programming solver!

There are many specialized QP solvers designed specifically for SVM's. We will not cover them :)

Equality constraints can be made from two inequality constraints

Many commercially available software packages for solving quadratic programming problems

$$\text{minimize}_{u \in R^L} : \frac{1}{2}\mathbf{u}^T Q \mathbf{u} + \mathbf{p}^T \mathbf{u}$$

$$\text{subject to} : \mathbf{a}^{(i)T}\mathbf{u} \geq c^{(i)} \quad (i = 1, \ldots, N)$$

Where the objective function is convex if Q is positive semidefinite: $\mathbf{u}^T Q \mathbf{u} \geq 0$ for all $\mathbf{u}$.

Can we turn our problem into this form?

Maximize:

$$L(\alpha) = -\frac{1}{2}\sum_{i,j=1}^{N}\alpha^{(j)}\alpha^{(i)}y^{(j)}y^{(i)}\mathbf{x}^{(j)T}\mathbf{x}^{(i)} + \sum_{i=1}^{N}\alpha^{(i)} \quad \text{subject to} \sum_{i=1}^{N}\alpha^{(i)}y^{(i)} = 0 \text{ and } \alpha^{(i)} \geq 0$$

Quadratic programming solvers usually are for minimization

$$\min_{\alpha} \frac{1}{2}\sum_{i,j=1}^{N}\alpha^{(j)}\alpha^{(i)}y^{(j)}y^{(i)}\mathbf{x}^{(j)T}\mathbf{x}^{(i)} - \sum_{i=1}^{N}\alpha^{(i)} \quad \text{subject to} \sum_{i=1}^{N}\alpha^{(i)}y^{(i)} = 0 \text{ and } \alpha^{(i)} \geq 0$$

$$\min_{\alpha} \frac{1}{2}\alpha^T \begin{bmatrix} y^{(1)}y^{(1)}\mathbf{x}^{(1)T}\mathbf{x}^{(1)} & y^{(1)}y^{(2)}\mathbf{x}^{(1)T}\mathbf{x}^{(2)} & \cdots & y^{(1)}y^{(N)}\mathbf{x}^{(1)T}\mathbf{x}^{(N)} \\ y^{(2)}y^{(1)}\mathbf{x}^{(2)T}\mathbf{x}^{(1)} & y^{(2)}y^{(2)}\mathbf{x}^{(2)T}\mathbf{x}^{(2)} & \cdots & y^{(2)}y^{(N)}\mathbf{x}^{(2)T}\mathbf{x}^{(N)} \\ \vdots & \vdots & & \vdots \\ y^{(N)}y^{(1)}\mathbf{x}^{(N)T}\mathbf{x}^{(1)} & y^{(N)}y^{(2)}\mathbf{x}^{(N)T}\mathbf{x}^{(2)} & \cdots & y^{(N)}y^{(N)}\mathbf{x}^{(N)T}\mathbf{x}^{(N)} \end{bmatrix} \alpha + \underbrace{(-1^T)\alpha}_{\text{linear term}}$$

quadratic term

subject to :

$$\underbrace{\mathbf{y}^T\boldsymbol{\alpha} = 0}_{\text{linear constraint}}$$

$$\underbrace{0 \leq \alpha^{(i)} \leq \infty}_{\text{range of values}}$$

$$\min_{\alpha} \frac{1}{2}\alpha^T Q \alpha - 1^T\alpha$$

subject to :

$$\mathbf{y}^T\alpha = 0$$

$$\alpha \geq 0$$

Instead of computing $\mathbf{x}^{(i)T}\mathbf{x}^{(j)}$ we could have computed $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ as long as K is symmetric and Q is pos. Semidefinite

We can perform the same calculations in Z-Space

Maximize:

$$L(\alpha) = -\frac{1}{2}\sum_{i,j=1}^{N}\alpha^{(j)}\alpha^{(i)}y^{(j)}y^{(i)}K(\mathbf{x}^{(j)},\mathbf{x}^{(i)}) + \sum_{i=1}^{N}\alpha^{(i)} \quad \text{subject to} \sum_{i=1}^{N}\alpha^{(i)}y^{(i)} = 0 \text{ and } \alpha^{(i)} \geq 0$$

Quadratic programming solvers usually are for minimization

$$\min_{\alpha} \frac{1}{2}\sum_{i,j=1}^{N}\alpha^{(j)}\alpha^{(i)}y^{(j)}y^{(i)}K(\mathbf{x}^{(j)},\mathbf{x}^{(i)}) - \sum_{i=1}^{N}\alpha^{(i)} \quad \text{subject to} \sum_{i=1}^{N}\alpha^{(i)}y^{(i)} = 0 \text{and } \alpha^{(i)} \geq 0$$

$$\min_{\alpha} \frac{1}{2}\alpha^{T}\begin{bmatrix} y^{(1)}y^{(1)}K(\mathbf{x}^{(1)},\mathbf{x}^{(1)}) & y^{(1)}y^{(2)}K(\mathbf{x}^{(1)},\mathbf{x}^{(2)}) & \cdots & y^{(1)}y^{(N)}K(\mathbf{x}^{(1)},\mathbf{x}^{(N)}) \\ y^{(2)}y^{(1)}K(\mathbf{x}^{(2)},\mathbf{x}^{(1)}) & y^{(2)}y^{(2)}K(\mathbf{x}^{(2)},\mathbf{x}^{(2)}) & \cdots & y^{(2)}y^{(N)}K(\mathbf{x}^{(2)},\mathbf{x}^{(N)}) \\ \vdots & \vdots & & \vdots \\ y^{(N)}y^{(1)}K(\mathbf{x}^{(N)},\mathbf{x}^{(1)}) & y^{(N)}y^{(2)}K(\mathbf{x}^{(N)},\mathbf{x}^{(2)}) & \cdots & y^{(N)}y^{(N)}K(\mathbf{x}^{(N)},\mathbf{x}^{(N)}) \end{bmatrix}\alpha + \underbrace{(-1^{T})\alpha}_{\text{linear term}}$$

quadratic term

subject to :

$\mathbf{y}^{T}\alpha = 0$ — linear constraint

$0 \leq \alpha^{(i)} \leq \infty$ — range of values

$$\min_{\alpha} \frac{1}{2}\alpha^{T}Q\alpha - 1^{T}\alpha$$

subject to :

$\mathbf{y}^{T}\alpha = 0$

$\alpha \geq 0$

For symmetric $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^{T}\phi(\mathbf{x}^{(j)})$ and positive semidefinite Q

"Kernel Trick"

Write the algorithm so the data only appeared when it was part of the dot product $\Phi\left(\mathbf{x}^{(i)}\right)^T \Phi\left(\mathbf{x}^{(j)}\right)$

➡ For some special types of transformations, we can efficiently compute $\Phi\left(\mathbf{x}^{(i)}\right)^T \Phi\left(\mathbf{x}^{(j)}\right)$ without transforming $\Phi\left(\mathbf{x}^{(i)}\right)$ or $\Phi\left(\mathbf{x}^{(j)}\right)$

- Remember the polynomial transformation?
- Define $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \Phi\left(\mathbf{x}^{(i)}\right)^T \Phi\left(\mathbf{x}^{(j)}\right)$
- We never need to create $\mathbf{z}^{(j)} = \Phi\left(\mathbf{x}^{(j)}\right)$

The kernel must satisfy Mercer's conditions (beyond the scope of this course)

This means…if I can compute, $\Phi\left(\mathbf{x}^{(i)}\right)^T \Phi\left(\mathbf{x}^{(j)}\right)$ efficiently I never need to worry about the length of $\Phi\left(\mathbf{x}^{(j)}\right)$

Learning in a transformed feature space:

Pair share: if we use $\Phi(\mathbf{x})$, do we need to actually perform the transformation?

Primal

$$\min_{\mathbf{w},w_0} \|\mathbf{w}\|_2^2$$

subject to: $y^{(i)}(\mathbf{w}^T\Phi(\mathbf{x}^{(i)}) + w_0) \geq 1$ for all $i = 1 \ldots N$

Dual

$$\min_{\alpha} \sum_{i,j} \alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)} K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) - \sum_i \alpha^{(i)}$$

$$\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0 \qquad \alpha^{(i)} \geq 0$$

We only need to know the result of $\Phi(\mathbf{x}^{(j)})^T\Phi(\mathbf{x}^{(i)})$

Prediction

$$g(\mathbf{x}) = \text{sign}\left(\mathbf{w}^T\Phi(\mathbf{x}) + w_0\right)$$

$$g(\mathbf{x}) = \text{sign}\left(\sum_{i \in I} \alpha^{(i)}y^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x}) + w_0\right)$$

We only need to know the result of $\Phi(\mathbf{x}^{(i)})^T\Phi(\mathbf{x})$

Support Vector $\mathbf{x}^{(i)}$

By computing the value of K without writing the transformation, we can have a computational advantage

Learning in a transformed feature space:

Pair share: if we use $\Phi(\mathbf{x})$, do we need to actually perform the transformation?

Primal

$$\min_{\mathbf{w},w_0} \|\mathbf{w}\|_2^2$$

subject to: $y^{(i)}(\mathbf{w}^T\Phi(\mathbf{x}^{(i)}) + w_0) \geq 1$ for all $i = 1 \ldots N$

Dual

$$\min_{\alpha} \sum_{i,j} \alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)} K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) - \ldots$$

$$\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0 \qquad \alpha^{(i)} \geq 0$$

Kernel trick: evaluate $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ without competing $\Phi(\mathbf{x}^{(j)})$ or $\Phi(\mathbf{x}^{(i)})$

We only need to know the result of $\Phi(\mathbf{x}^{(j)})^T\Phi(\mathbf{x}^{(i)})$

Prediction

$$g(\mathbf{x}) = \text{sign}\left(\mathbf{w}^T\Phi(\mathbf{x}) + w_0\right)$$

$$g(\mathbf{x}) = \text{sign}\left(\sum_{i \in I} \alpha^{(i)}y^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x}) + w_0\right)$$

We only need to know the result of $\Phi(\mathbf{x}^{(i)})^T\Phi(\mathbf{x})$

Support Vector $\mathbf{x}^{(i)}$

By computing the value of K without writing the transformation, we can have a computational advantage

Soft margin support vectors

MARGIN SUPPORT VECTORS

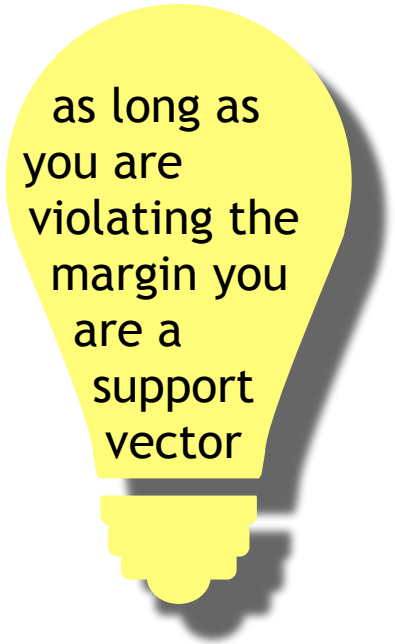
$$y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) = 1$$

$$\xi^{(i)} = 0$$

$$\alpha^{(i)} > 0$$

NON-MARGIN SUPPORT VECTORS

$$y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) < 1 \qquad \xi^{(i)} > 0$$



To decide on the value of C, use cross validation!

A good first guess is: $C = \frac{1}{N}$

Outline

- Notation change, intuition, and finding how to compare hyperplanes - mathematically how do compare hyperplanes to find the one with the maximum margin. Can we turn this way of comparing hyperplanes into an objective function
- Support vector machines
  - ★ hard margin - find the objective function when the data is linearly separable
  - ★ Dealing with non-linear data - "Soft" margins for SVM - New objective function for the case where the data is not linearly separable
  - ★ dual formula - a clever trick
  - ★ Dealing with non-linear data - feature transformation with the kernel trick - Show two popular feature maps

Kernel SVM

What if the data isn't linearly separable (part 2)?

SOME OF THE MATERIAL IN THE NEXT FEW SLIDE ARE FROM SLIDES CREATED BY PROF LISA HELLERSTEIN AND PROF LINDA SELLIE

SOME SLIDES WERE INSPIRED BY SLIDES FROM CMU 18-661

How to deal with non-linearly separable data

- Transform features?
  - By adding enough new features we can make any data linearly separable…

$$\Phi(\mathbf{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

  We can transform $\mathbf{x}$ into $\Phi(\mathbf{x})$ in both the primal and dual SVM formulations
- What could be the problem?
  - high computation costs
  - overfitting
- Dual formulation of support vectors are designed to deal (somewhat) with these issues

Standard Choices of Kernel functions

- Dot product (no transformation):

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \mathbf{x}^{(i)T}\mathbf{x}^{(j)}$$

- Polynomial kernels:

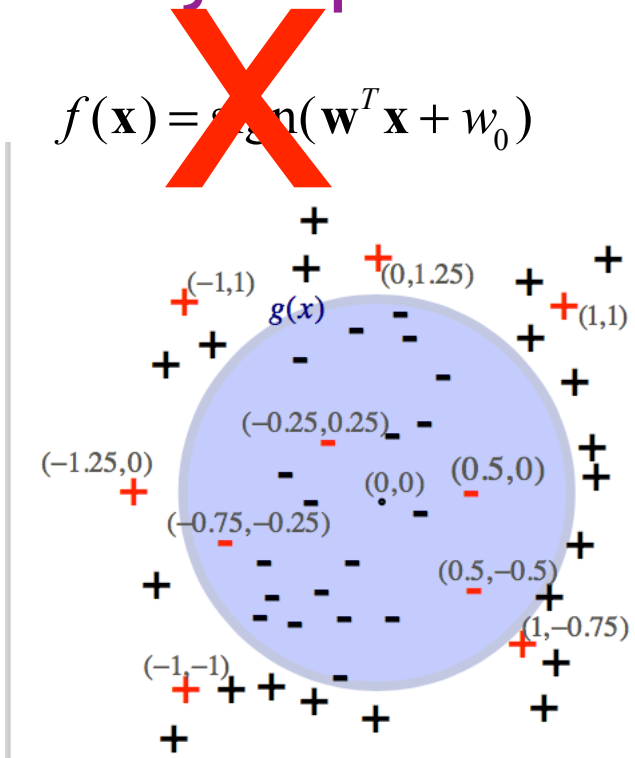
$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (1 + \mathbf{x}^{(i)T}\mathbf{x}^{(j)})^d$$

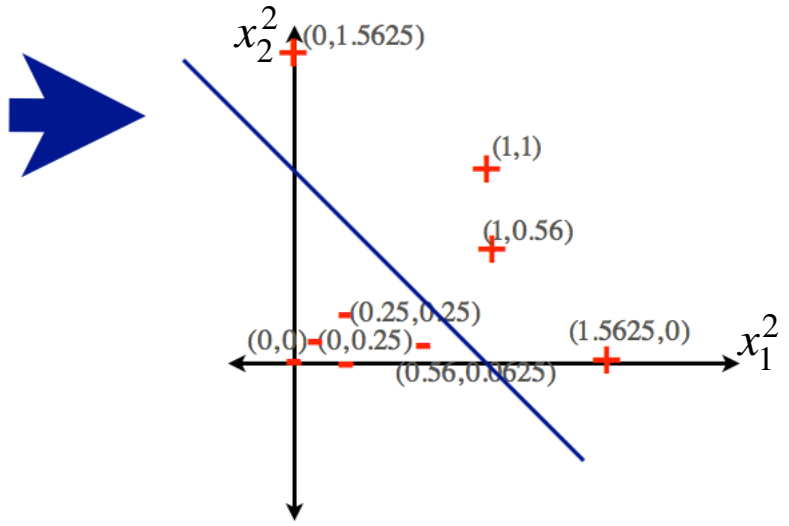
- Radial basis function:

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp\left(-\gamma\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2\right)$$

Map data into a different space

linearly separable?

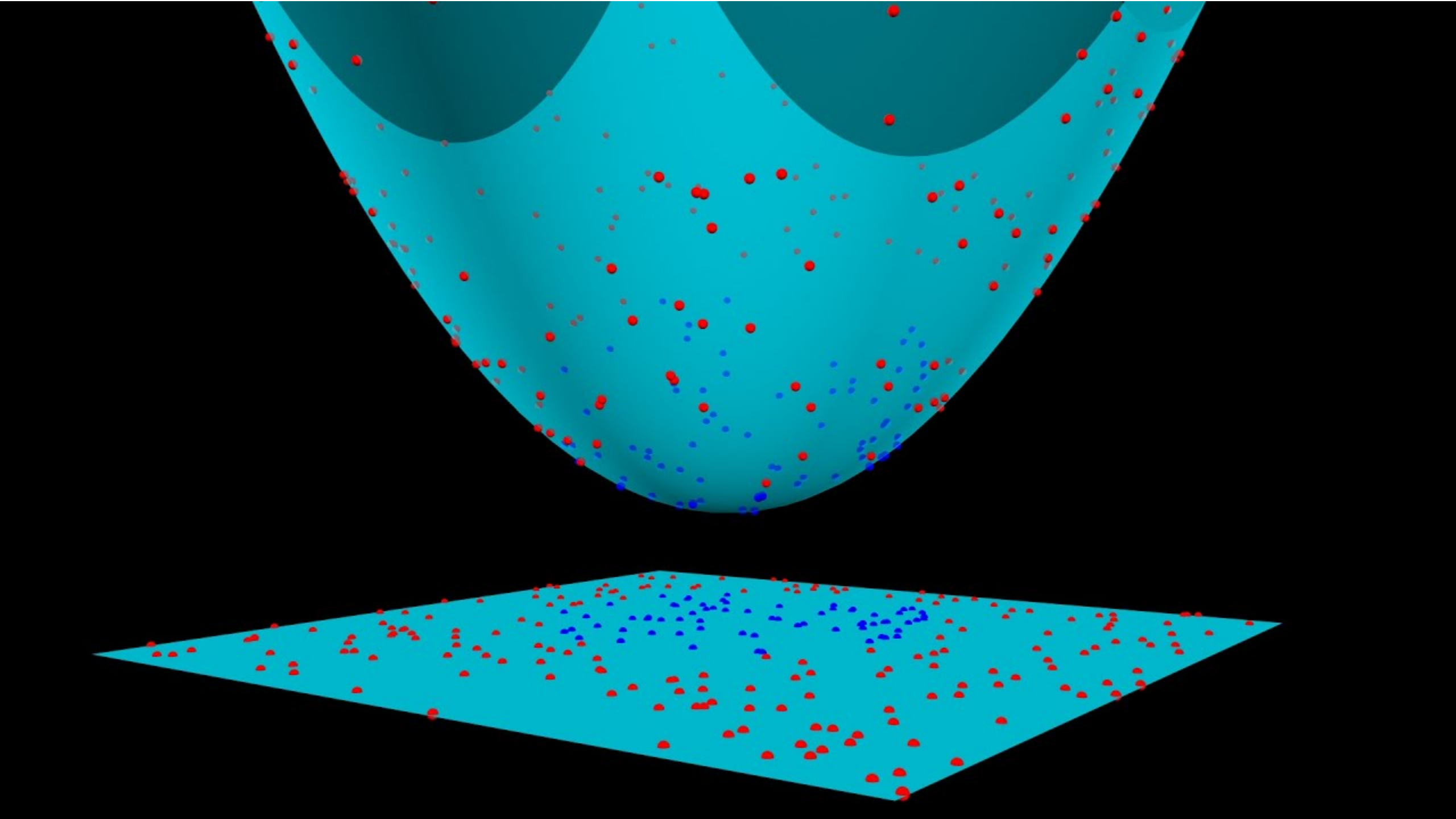




If we transform the features using $\Phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]^T$

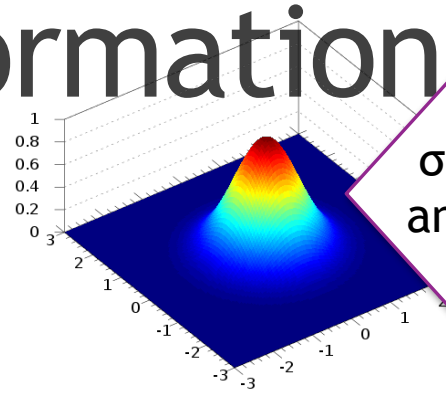
$$\text{sign}\left(\left(\sum_{i\in I}\alpha^{(i)}y^{(i)}\Phi(\mathbf{x}^{(i)})\right)^T\Phi(\mathbf{x}) + w_0\right) \Rightarrow g(\mathbf{x}) = \text{sign}\left(\sum_{i\in I}\alpha^{(i)}y^{(i)}K(\mathbf{x}^{(i)},\mathbf{x}) + w_0\right)$$

$$K(\mathbf{x}^{(i)},\mathbf{x}) = (1 + \mathbf{x}^{(i)T}\mathbf{x})^2$$



picture from https://en.wikipedia.org/wiki/Gaussian_function

Another feature transformation idea



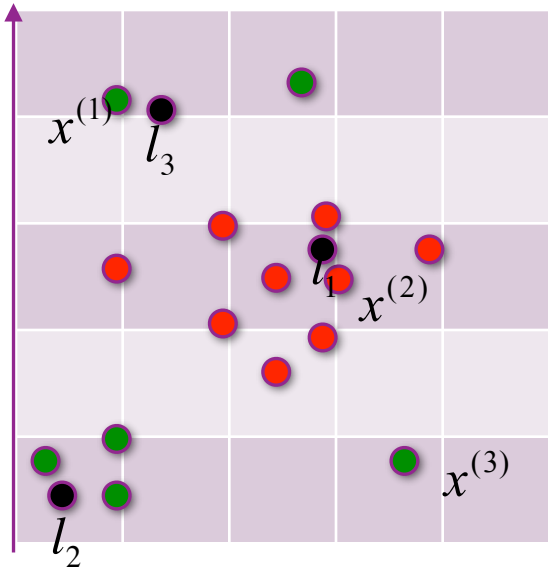
if σ^2 is small, then lower bias and higher variance since the features taper off quicker

- Define new features, f_1, f_2, and f_3 for every example on how close it is to chosen"landmarks" $l^{(1)}$, $l^{(2)}$, and $l^{(3)}$
- For an example **x** we create new features z_1, z_2, z_3:

$$z_1 = exp\left(-\frac{\left\|\mathbf{x} - l^{(1)}\right\|_2^2}{2\sigma^2}\right)$$

$$z_2 = exp\left(-\frac{\left\|\mathbf{x} - l^{(2)}\right\|_2^2}{2\sigma^2}\right)$$

$$z_3 = exp\left(-\frac{\left\|\mathbf{x} - l^{(3)}\right\|_2^2}{2\sigma^2}\right)$$



- $\Phi(x^{(1)})=(z_1, z_2, z_3)$ approx (0, 0, 1)
- $\Phi(x^{(2)})=(z_1, z_2, z_3)$ approx (1, 0, 0)
- The larger the sigma - the more weight is given to features far away from the landmark

This slide is adapted from: https://www.coursera.org/learn/machine-learning/lecture/YOMHn/kernels-i

https://en.wikipedia.org/wiki/Radial_basis_function_kernel

Another feature transfo

- Define new features, f_1, f_2, and f_3 for every examp… how close it is to chosen "landmarks" $l^{(1)}$, $l^{(2)}$, and $l^{(3)}$
- For an example **x** we create new features z_1, z_2, z_3:

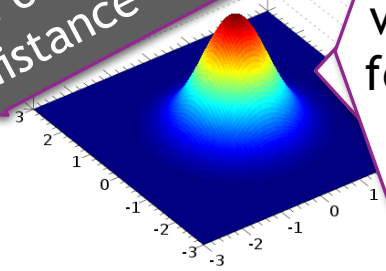
$$z_1 = exp\left(-\frac{\|\mathbf{x} - l^{(1)}\|_2^2}{2\sigma^2}\right)$$

$$z_2 = exp\left(-\frac{\|\mathbf{x} - l^{(2)}\|_2^2}{2\sigma^2}\right)$$

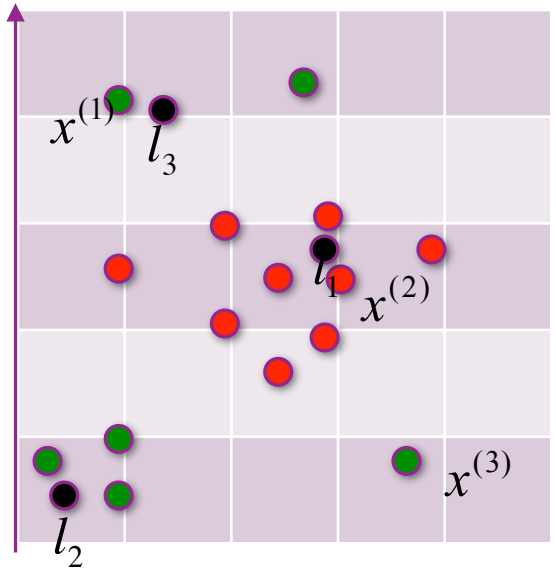
$$z_3 = exp\left(-\frac{\|\mathbf{x} - l^{(3)}\|_2^2}{2\sigma^2}\right)$$

$\|\mathbf{x} - l^{(i)}\|_2^2$ Square of Euclidean distance of x and $l^{(i)}$

picture from https://en.wikipedia.org/wiki/Gaussian_function



if σ^2 is large, then higher bias and lower variance since the features taper off slower. σ^2 is the radius of influence



- $\Phi(x^{(1)}) = (z_1, z_2, z_3)$ approx (0, 0, 1)
- $\Phi(x^{(2)}) = (z_1, z_2, z_3)$ approx (1, 0, 0)
- The larger the sigma - the more weight is given to features far away from the landmark

This slide is adapted from: https://www.coursera.org/learn/machine-learning/lecture/YOMHn/kernels-i

https://en.wikipedia.org/wiki/Radial_basis_function_kernel

Gausian-RBF Kernel

$$\Phi(\mathbf{x}) = exp\left(-\frac{(\mathbf{x})^2}{2\sigma^2}\right)\left(1, \sqrt{\frac{2^1}{1!}}(\mathbf{x})^1, \sqrt{\frac{2^2}{2!}}(\mathbf{x})^2, \sqrt{\frac{2^3}{3!}}(\mathbf{x})^3, \ldots\right)$$

infinite-dimensions transformation

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = exp\left(-\frac{\left\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right\|_2^2}{2\sigma^2}\right) = exp\left(-\frac{(\mathbf{x}^{(i)})^2}{2\sigma^2}\right) exp\left(\frac{2\mathbf{x}^{(i)}\mathbf{x}^{(j)}}{2\sigma^2}\right) exp\left(-\frac{(\mathbf{x}^{(j)})^2}{2\sigma^2}\right)$$

Taylor series

$$= exp\left(-\frac{(\mathbf{x}^{(i)})^2}{2\sigma^2}\right)\left(\sum_{k=0}^{\infty}\frac{2^k(\mathbf{x}^{(i)})^k(\mathbf{x}^{(j)})^k}{k!}\right) exp\left(-\frac{(\mathbf{x}^{(j)})^2}{2\sigma^2}\right)$$

$$= \Phi(\mathbf{x}^{(i)})^T \Phi(\mathbf{x}^{(j)})$$

The Taylor series expansion of e^x

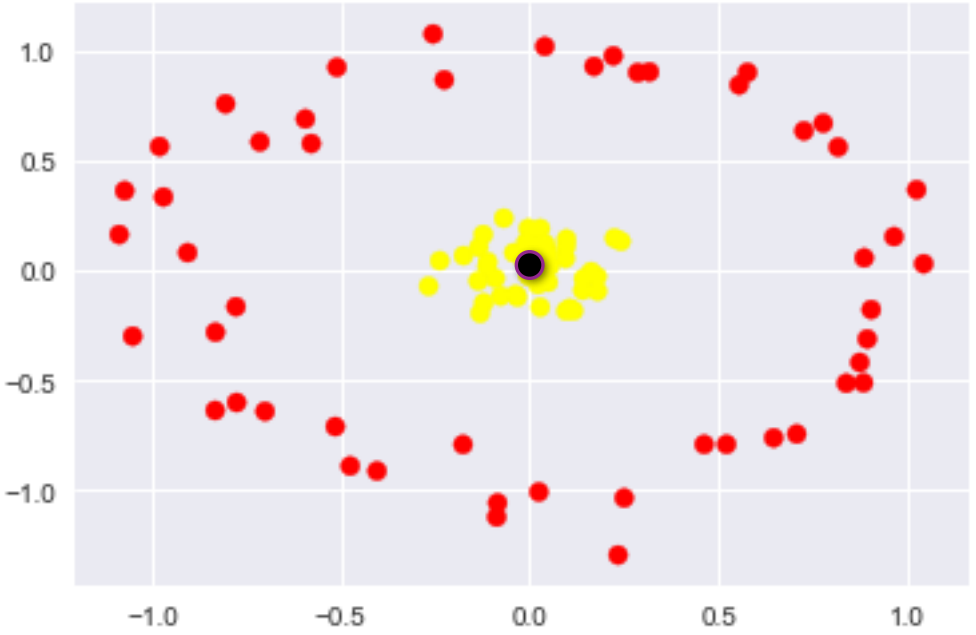
$$e^x = \sum_{k=0}^{\infty} x^k/k!$$

see: http://people.math.sc.edu/girardi/m142/handouts/10sTaylorPolySeries.pdf

To extend this idea see: https://peekaboo-vision.blogspot.com/2012/12/kernel-approximations-for-efficient.html
https://pdfs.semanticscholar.org/b986/6f91d6ce82384e18e8e99f803cf817787f74.pdf

Example

We choose to make a landmark of a center point, l, and compute a new feature for example $x^{(i)}$



https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

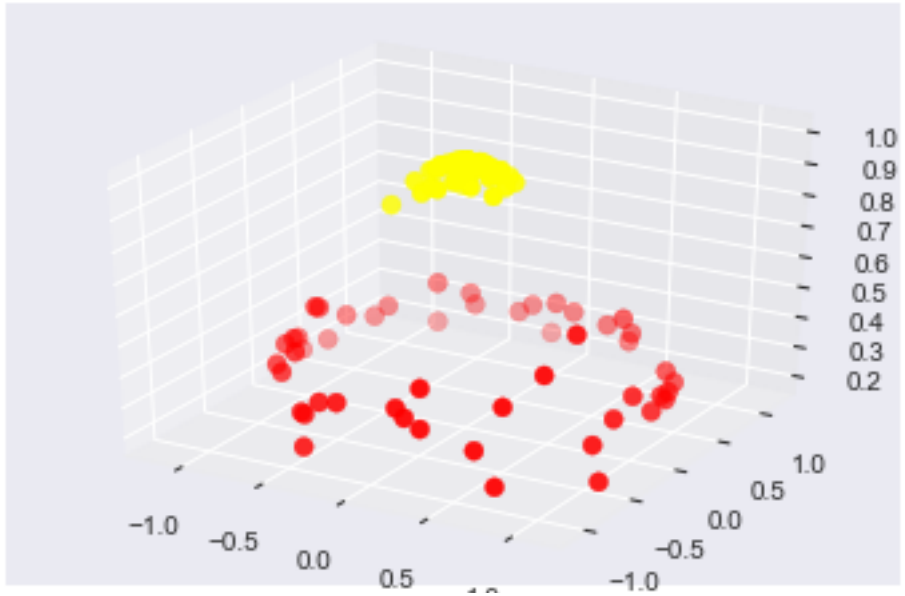
Lets transform the feature space by using a RBF (radial basis function) Kernel SVM

We choose to make a landmark of a center point, l, and compute a new feature for example $x^{(i)}$

$$z_1^{(i)} = \exp\left(-\frac{\|\mathbf{x}^{(i)} - \ell\|_2^2}{(2\sigma^2)}\right)$$

For every example $x^{(i)}$, where $z^{(i)}$ will have only one feature.
Now all the examples have only one feature (instead of 2 features).

We will show the original features and the new feature z



In the new space, it is easy to see we can separate the points

https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

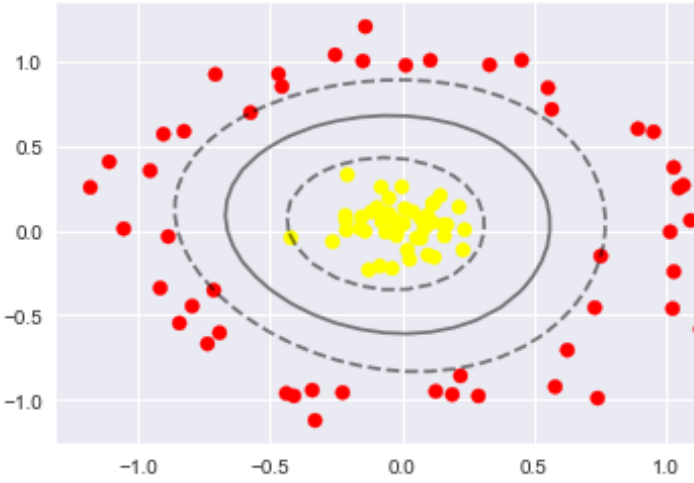
Gaussian Radial Basis Function

- For every example make a landmark: $l^{(1)} = \mathbf{x}^{(1)}$, $l^{(2)} = \mathbf{x}^{(2)}$,, …, $l^{(N)} = \mathbf{x}^{(N)}$
- Then for every x, we have created N new features z_1, z_2,…, z_n:

$$K(\mathbf{x}^{(i)}, l^{(j)}) = z_j^{(i)} = exp\left(-\frac{\left\|\mathbf{x}^{(i)} - l^{(j)}\right\|_2^2}{2\sigma^2}\right)$$

$$z = \begin{bmatrix} \exp(\left\|\mathbf{x} - l^{(1)}\right\|_2^2 / 2\sigma^2) \\ \exp(\left\|\mathbf{x} - l^{(2)}\right\|_2^2 / 2\sigma^2) \\ \vdots \\ \exp(\left\|\mathbf{x} - l^{(N-1)}\right\|_2^2 / 2\sigma^2) \\ \exp(\left\|\mathbf{x} - l^{(N)}\right\|_2^2 / 2\sigma^2) \end{bmatrix}$$

- The number of features is now N




```
support vectors
[-0.7116806 , -0.35318537]
[-0.57297015,  0.69623782]
[ 0.7511019 , -0.1515738 ]
[-0.42044048, -0.04330787]
```

A great description is given at: https://www.coursera.org/learn/machine-learning/lecture/hxdcH/kernels-ii

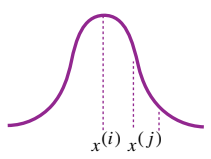
Gaussian Radial Basis Function

- For every example make a landm
  $l^{(1)} = \mathbf{x}^{(1)}$, $l^{(2)} = \mathbf{x}^{(2)}$,, …, $l^{(N)} = \mathbf{x}^{(N)}$
- Then for every x, we have created N new features z_1, z_2,…, z_n:

$$K(\mathbf{x}^{(i)}, l^{(j)}) = z_j^{(i)} = exp\left(-\frac{\left\|\mathbf{x}^{(i)} - l^{(j)}\right\|_2^2}{2\sigma^2}\right)$$

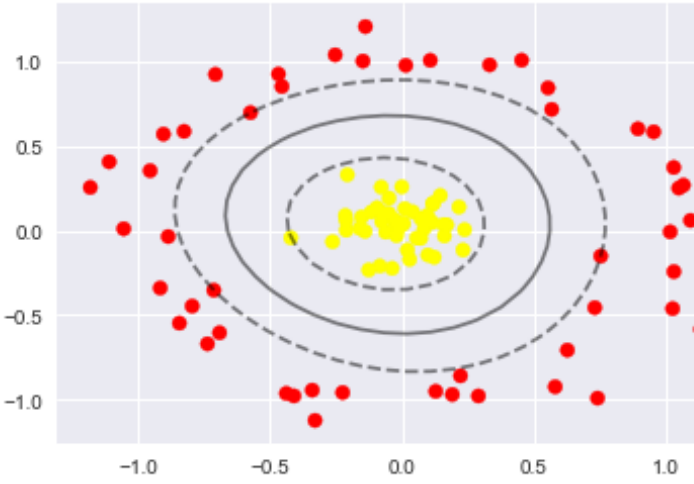
Perform feature scaling!

Similarity measure. $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}$ are "very similar" if they are at most σ distance



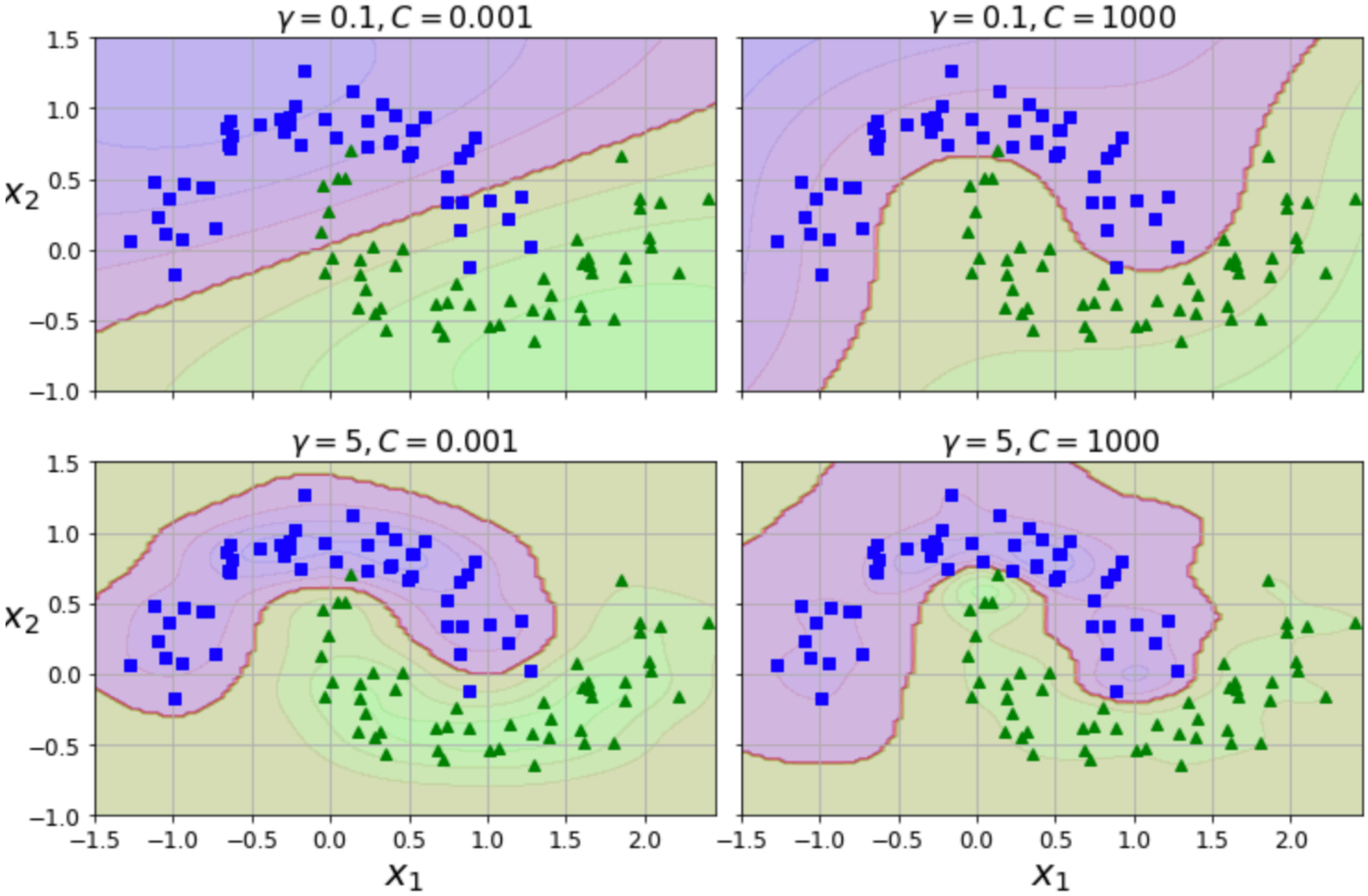
$$\begin{bmatrix} \exp(\left\|\mathbf{x} - l^{(1)}\right\|_2^2 / 2\sigma^2) \\ \exp(\left\|\mathbf{x} - l^{(2)}\right\|_2^2 / 2\sigma^2) \\ \exp(\left\|\mathbf{x} - l^{(N)}\right\|_2^2 / 2\sigma^2) \end{bmatrix}$$

- The number of features is now N




```
support vectors
[-0.7116806 , -0.35318537]
[-0.57297015,  0.69623782]
[ 0.7511019 , -0.1515738 ]
[-0.42044048, -0.04330787]
```

A great description is given at: https://www.coursera.org/learn/machine-learning/lecture/hxdcH/kernels-ii



$$\gamma = \frac{1}{2\sigma^2}$$

Graph from Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow

Read the next slides on how to use the sklearn implementation of SVM on your own

Sklearn

LinearSVC: optimizer for the primal problem

SVC: optimizer for the dual problem

From: https://scikit-learn.org/dev/modules/svm.html#svm-kernels

1.4.6. Kernel functions

The *kernel function* can be any of the following:

- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma \langle x, x' \rangle + r)^d$, where d is specified by parameter `degree`, r by `coef0`.
- rbf: $\exp(-\gamma \|x - x'\|^2)$, where γ is specified by parameter `gamma`, must be greater than 0.
- sigmoid $\tanh(\gamma \langle x, x' \rangle + r)$, where r is specified by `coef0`.

Different kernels are specified by the `kernel` parameter:

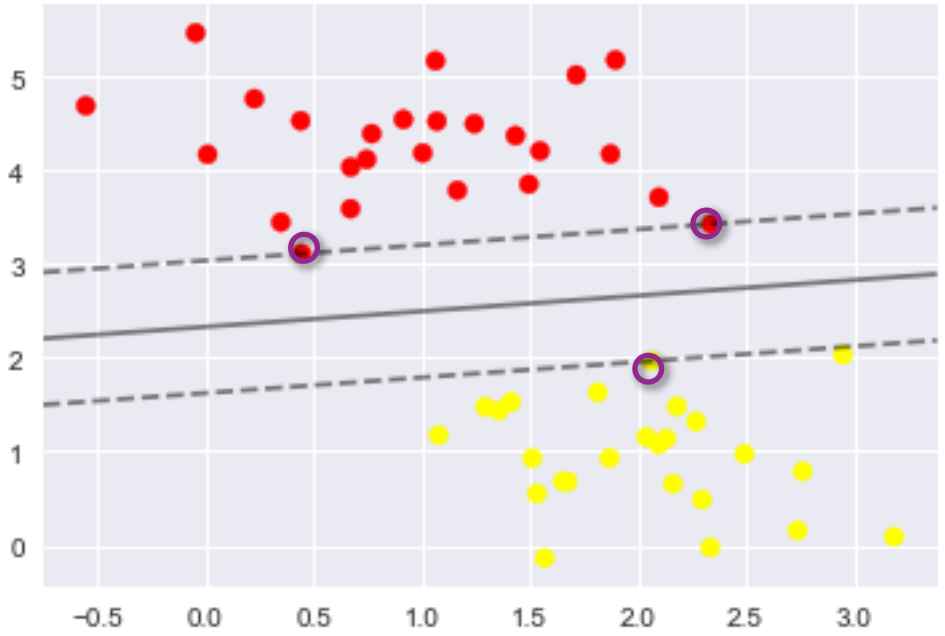
```
>>> linear_svc = svm.SVC(kernel='linear')
>>> linear_svc.kernel
'linear'
>>> rbf_svc = svm.SVC(kernel='rbf')
>>> rbf_svc.kernel
'rbf'
```

Using sklearn's support vector classifier to find the maximum margin on the training examples X

```
from sklearn.svm import SVC # "Support vector
model = SVC(kernel='linear', C=1E10)
model.fit(X, y)
```

By choosing the parameter C to be very large we assure no misclassification points

The classifier finds the optimal model for the training examples



We can observe what the support vectors are:

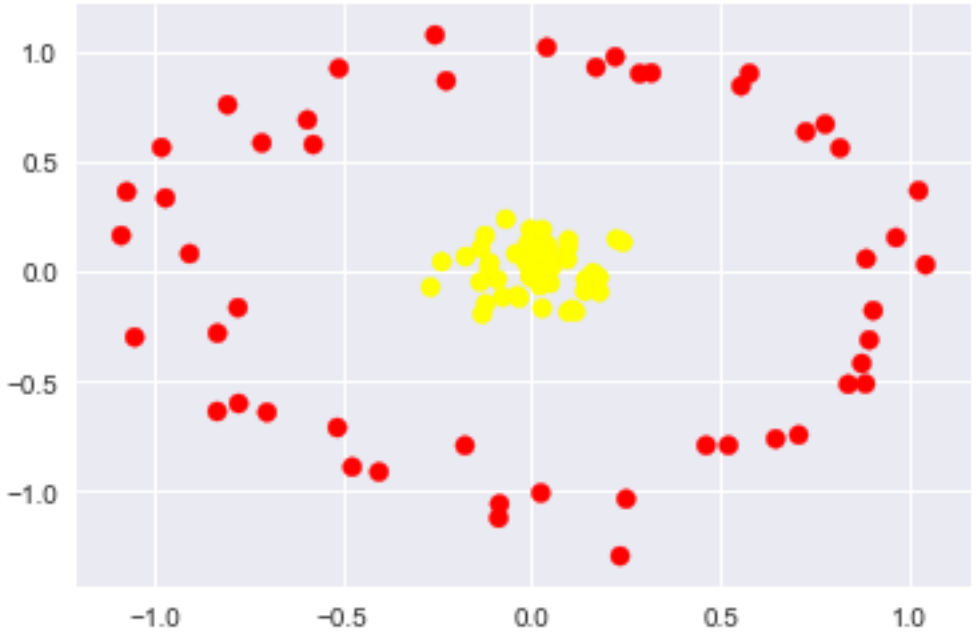
```
model.support_vectors_
```

```
array([[ 0.44359863,  3.11530945],
       [ 2.33812285,  3.43116792],
       [ 2.06156753,  1.96918596]])
```

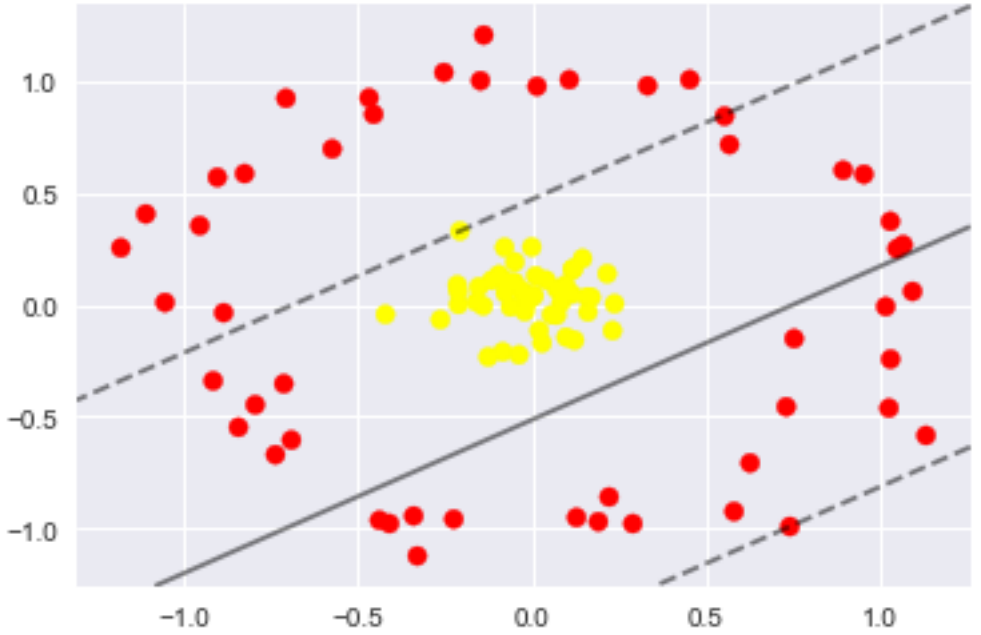
https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

Suppose the data is not linearly separable

Lets try to fit it with a linear support vector machine (we will allow for some misclassification)…




```
clf = SVC(kernel='linear').fit(X, y)
```



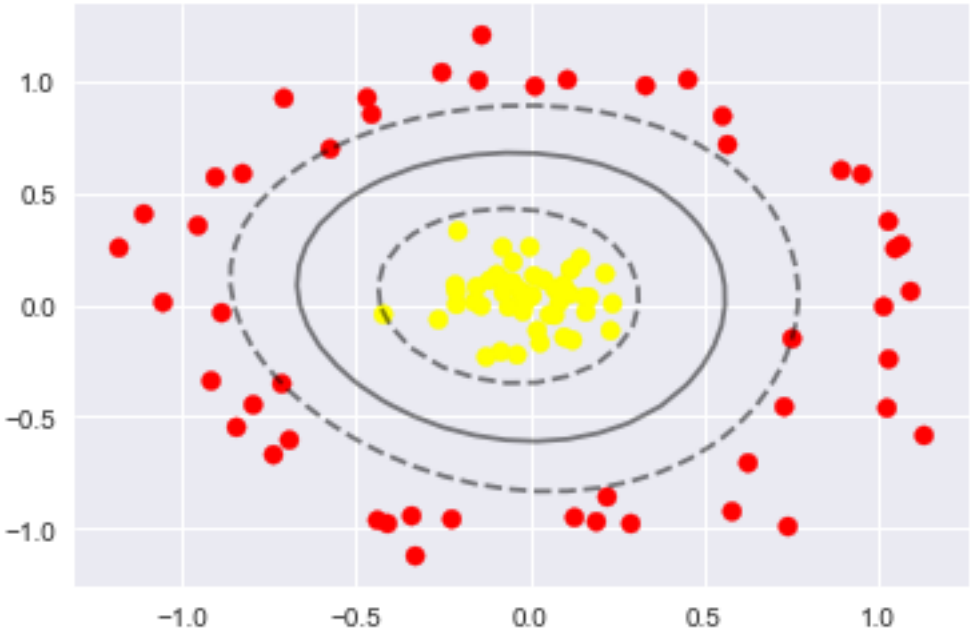
https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

How can we find the right point to transform the points to become linearly separable?

Easiest answer is to compute the distance to every point in the data set!
Problem? The number of features becomes N, the number of training examples

In sklearn, we can apply the radial basis function kernel (RBF kernel)

```
clf = SVC(kernel='rbf', C=1E6)
clf.fit(X, y)
```

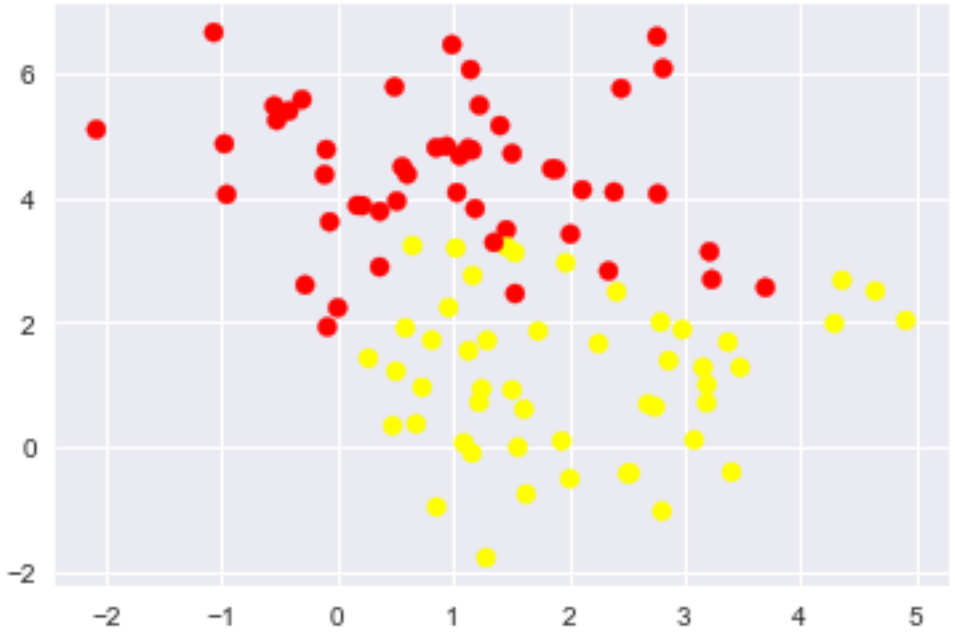



```
clf.support_vectors_
```

```
array([[-0.7116806 , -0.35318537],
       [-0.57297015,  0.69623782],
       [ 0.7511019 , -0.1515738 ],
       [-0.42044048, -0.04330787]])
```

https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

Another way to work with non separable data is to allowing for errors - Soft Margin SVM



https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

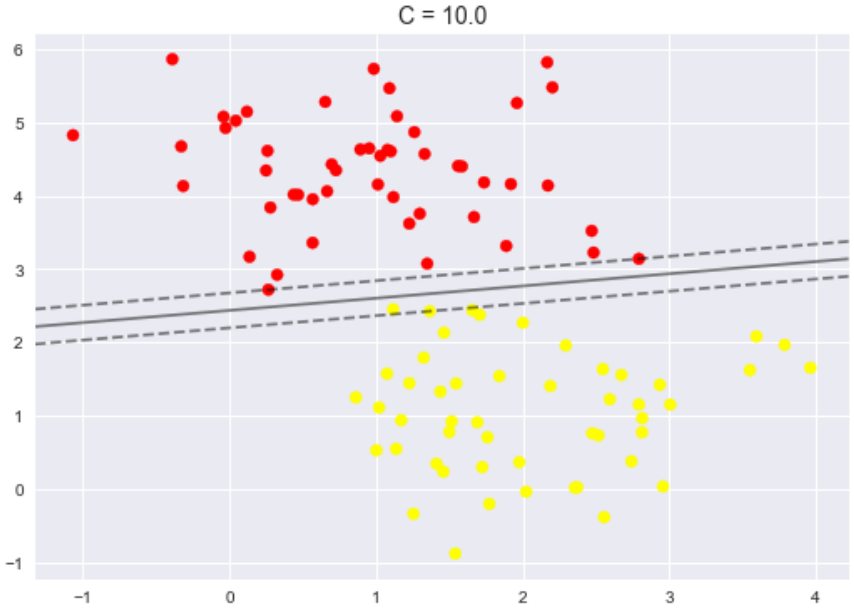
We create a soft- margin classifier by adding a cost to points that are not correctly classified and outside the margin

The primal objective function:

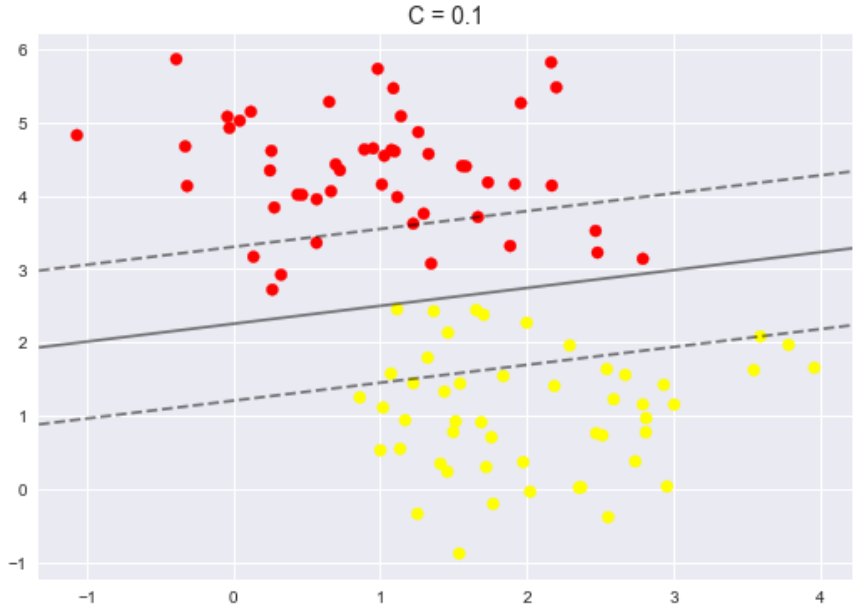
$$\min_{w_0,\mathbf{w},\{\xi^{(i)}\}_{i=1}^{N}} \|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N} \xi^{(i)}$$

$$\text{subject to } y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) \geq 1 - \xi^{(i)} \quad \text{for all } i = 1,\ldots,N \qquad \xi^{(i)} \geq 0$$

The tuning parameter C softens (C smaller) or hardens (C larger)



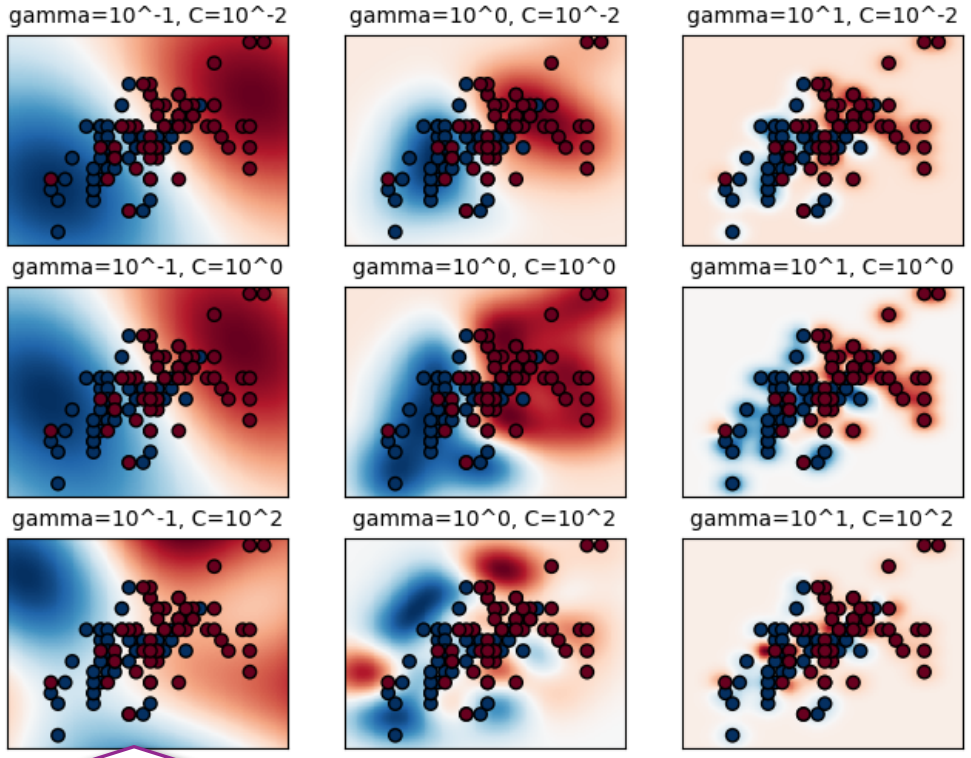
model = SVC(kernel='linear', C=10).fit(X, y)



model = SVC(kernel='linear', C=0.1).fit(X, y)

https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

SVC



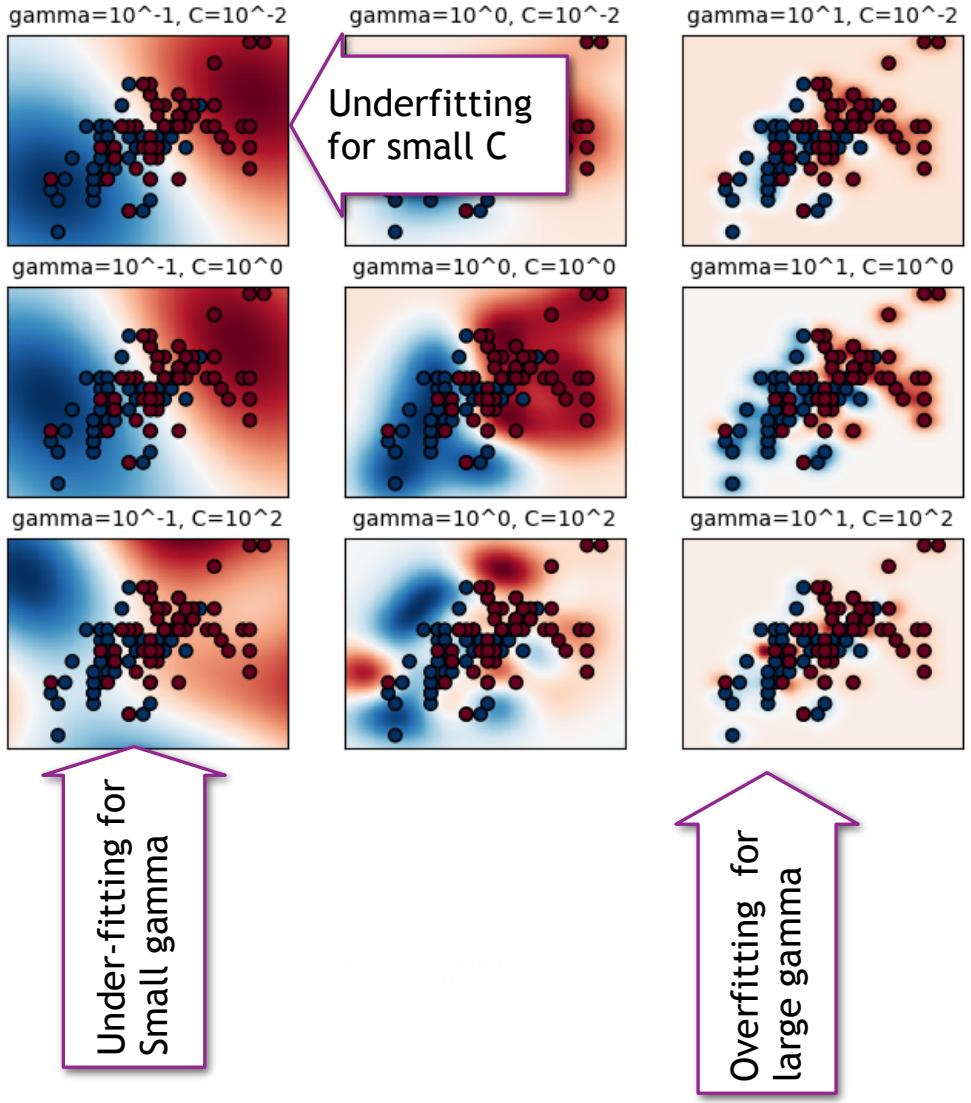

```
from sklearn.svm import SVC


C_2d_range = [1e-2, 1, 1e2]
gamma_2d_range = [1e-1, 1, 1e1]
classifiers = []
for C in C_2d_range:
    for gamma in gamma_2d_range:
        clf = SVC(C=C, gamma=gamma)
        clf.fit(X_2d, y_2d)
        classifiers.append((C, gamma, clf))
```

“Intuitively, the `gamma` parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’.”
“The `C` parameter trades off correct classification of training examples against maximization of the decision function’s margin”

The radius of the RBF kernel alone acts as a good structural regularizer.

Example from http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

SVC




```
from sklearn.svm import SVC


C_2d_range = [1e-2, 1, 1e2]
gamma_2d_range = [1e-1, 1, 1e1]
classifiers = []
for C in C_2d_range:
    for gamma in gamma_2d_range:
        clf = SVC(C=C, gamma=gamma)
        clf.fit(X_2d, y_2d)
        classifiers.append((C, gamma, clf))
```

“Intuitively, the `gamma` parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’.”
“The `C` parameter trades off correct classification of training examples against maximization of the decision function’s margin”

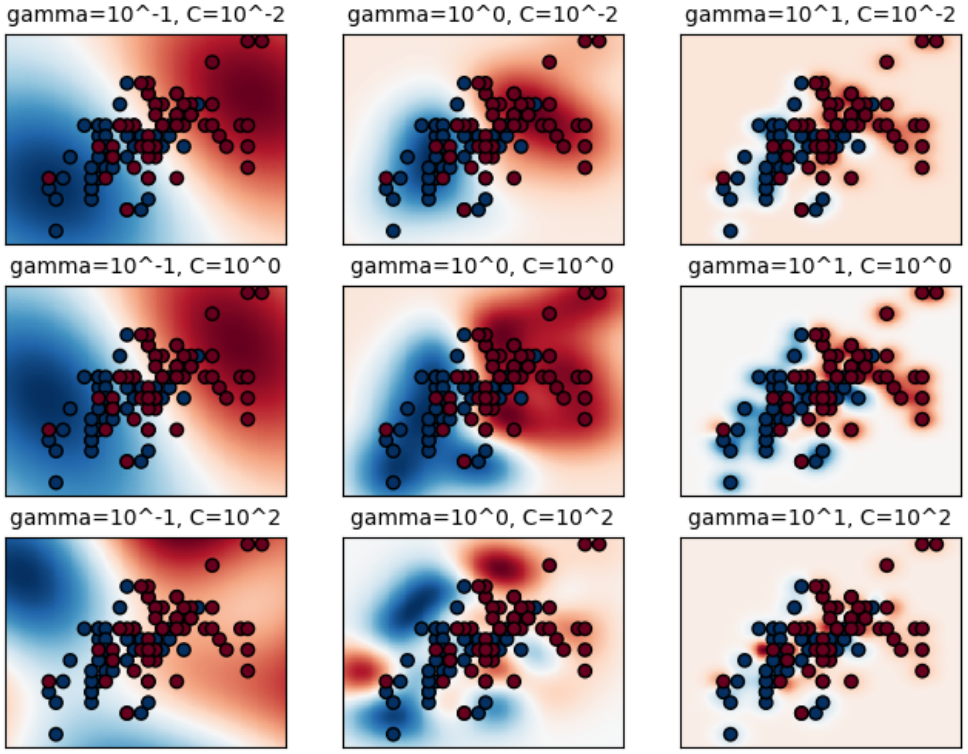
The radius of the RBF kernel alone acts as a good structural regularizer.

Example from http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

Ack!!! There are too many parameters to tune

sklearn's RandomizedSearchCV and GridSearchCV are classes for parameter tuning that methodically builds and evaluates different combinations of parameters as a grid

GridSearchCV



These also need to be imported

```
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import GridSearchCV
# Train classifiers
#
# For an initial search, a logarithmic grid with basis
# 10 is often helpful. Using a basis of 2, a finer
# tuning can be achieved but at a much higher cost.

C_range = np.logspace(-2, 10, 13)
gamma_range = np.logspace(-9, 3, 13)
param_grid = dict(gamma=gamma_range, C=C_range)
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
grid = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
grid.fit(X, y)

print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))
```

Example from http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html