Support vector machines (SVMs)
Lecture 6

David Sontag
New York University

Slides adapted from Luke Zettlemoyer, Vibhav Gogate, and Carlos Guestrin

Pegasos vs. Perceptron

Pegasos Algorithm

Initialize: $w_1 = 0$, $t=0$

For iter = 1,2,…,20

    For j=1,2,…,|data|

        $t = t+1$

        $\eta_t = 1/(t\lambda)$

        If $y_j(w_t\, x_j) < 1$

            $w_{t+1} = (1-\eta_t\lambda)\, w_t + \eta_t\, y_j\, x_j$

        Else

            $w_{t+1} = (1-\eta_t\lambda)\, w_t$

Output: wt+1

Pegasos vs. Perceptron

<u>Perceptron Algorithm</u>

Initialize: $w_1 = 0$, $t=0$

For iter = 1,2,…,20

For j=1,2,…,|data|

$t = t+1$

~~$\eta_t = 1/(t\lambda)$~~

If $y_j(w_t\, x_j) < \cancel{1}\ \mathbf{0}$

$w_{t+1} = \cancel{(1-\eta_t\lambda)}\, w_t + \cancel{\eta_t}\, y_j\, x_j$

~~**Else**~~

~~$w_{t+1} = (1-\eta_t\lambda)\, w_t$~~

Output: wt+1

Much faster than previous methods

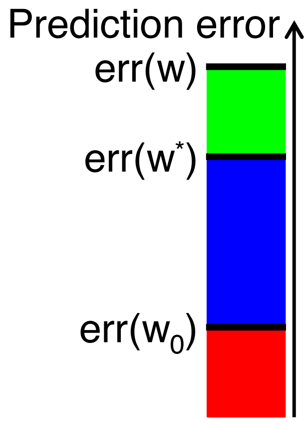
- **3 datasets** (provided by Joachims)
  - Reuters CCAT (800K examples, 47k features)
  - Physics ArXiv (62k examples, 100k features)
  - Covertype (581k examples, 54 features)

Training Time (in seconds):

	Pegasos	SVM-Perf	SVM-Light
Reuters	**2**	77	20,075
Covertype	**6**	85	25,514
Astro-Physics	**2**	5	80

Running time guarantee

[Shalev Schwartz, Srebro '08]

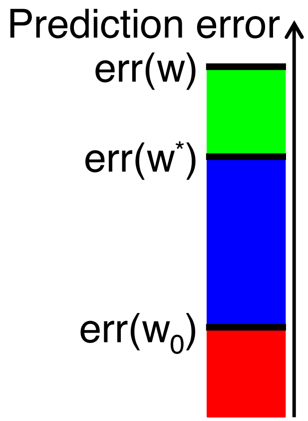


Note: w_0 is redefined in this context (see below) – does *not* refer to initial weight vector

- Approximation error:
  - Best error achievable by large-margin predictor
  - Error of population minimizer
    $w_0 = \text{argmin } E[f(w)] = \text{argmin } \lambda|w|^2 + E_{x,y}[\text{loss}(\langle w,x\rangle;y)]$
- Estimation error:
  - Extra error due to replacing E[loss] with empirical loss
    $w^* = \text{arg min } f_n(w)$
- Optimization error:
  - Extra error due to only optimizing to within finite precision

Running time guarantee

[Shalev Schwartz, Srebro '08]



- Approximation error:
  - Best error achievable by large-marg
  - Error of population minimizer
    w_0 = argmin E[f(w)] = argmin $\lambda|w|^2$
- Estimation error:
  - Extra error due to replacing E[loss]
    w^* = arg min $f_n(w)$
- Optimization error:
  - Extra error due to only optimizing to within finite precision

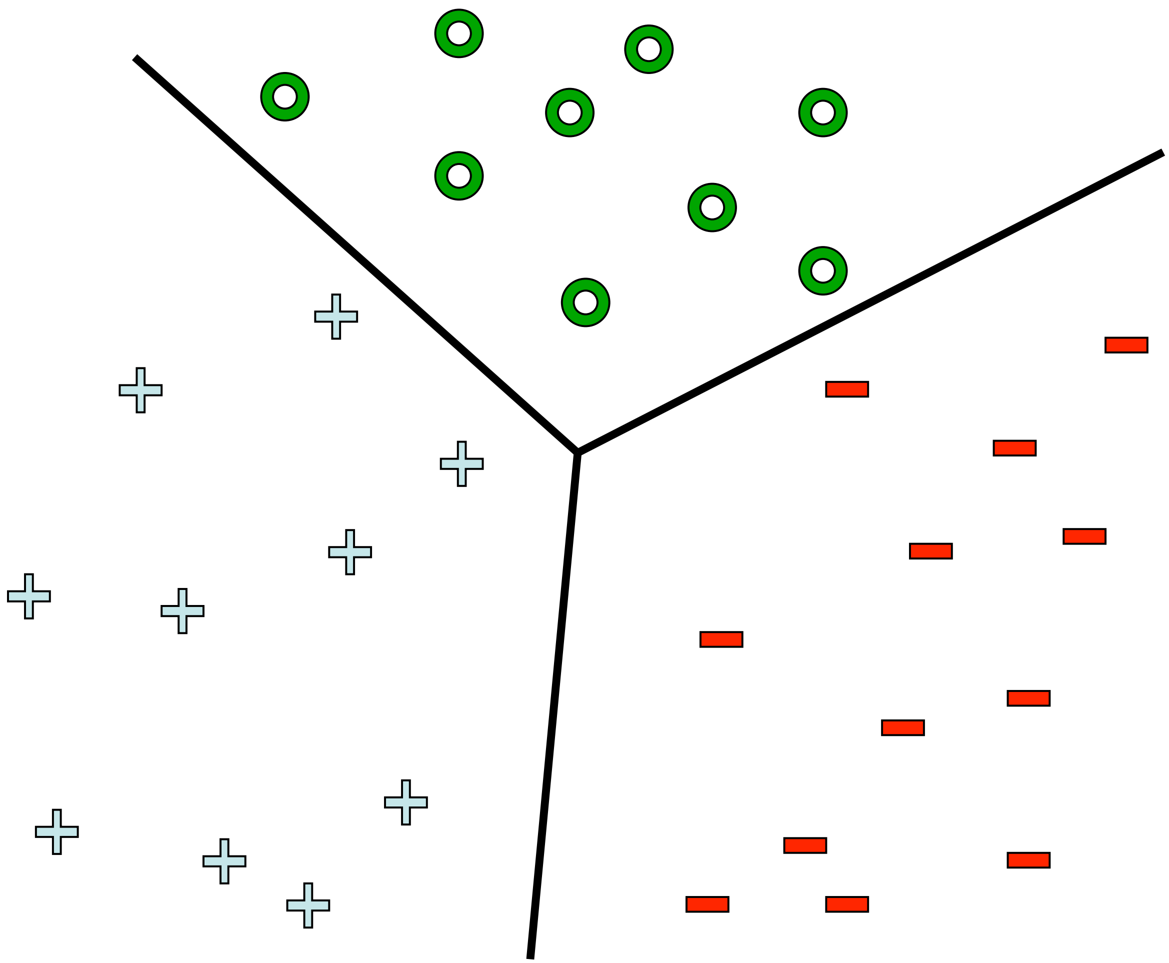
Pegasos

After $T = \tilde{O}\left(\frac{1}{\delta\lambda\epsilon}\right)$ updates:

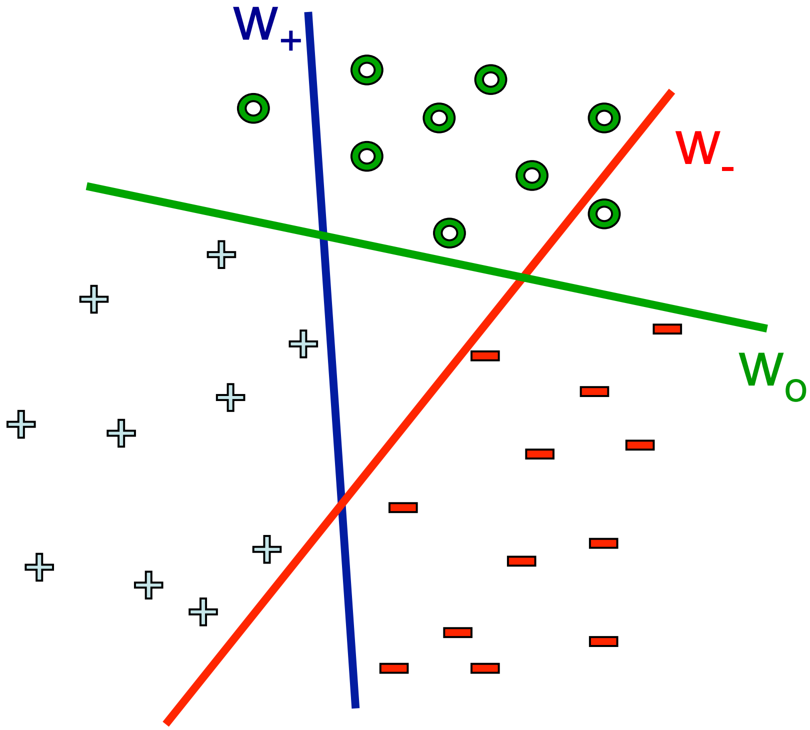
err(w_T) < err(w_0) + ϵ

With probability 1- δ

Extending to multi-class classification



One versus all classification



Learn 3 classifiers:

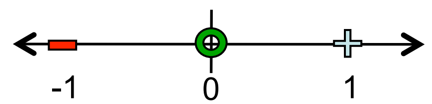
- \- vs {o,+}, weights w_-
- \+ vs {o,-}, weights w_+
- o vs {+,-}, weights w_o

Predict label using:

$$\hat{y} \leftarrow \arg\max_k \; w_k \cdot x + b_k$$

Any problems?

Could we learn this (1-D) dataset? →

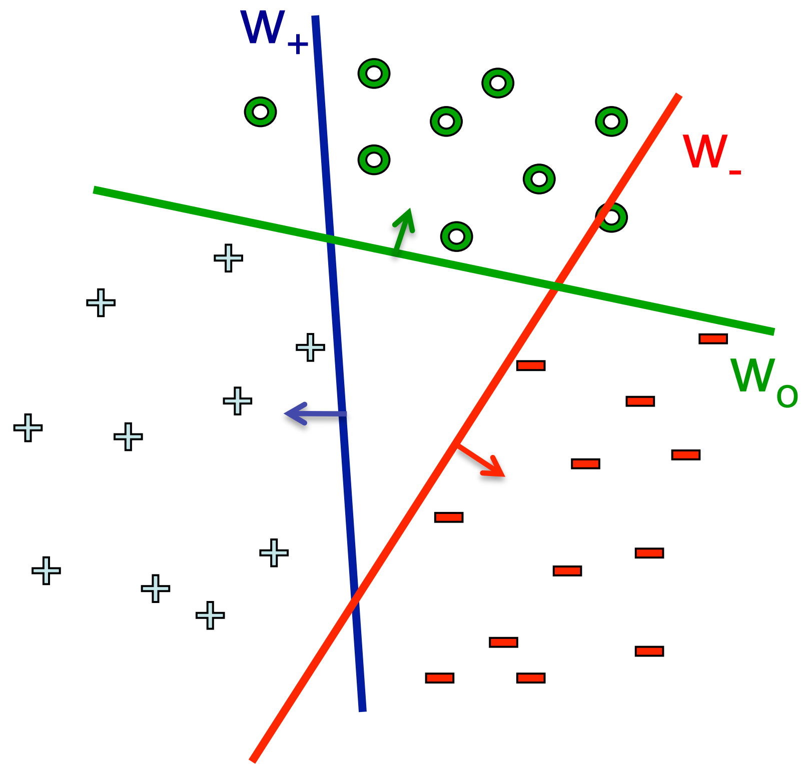


Multi-class SVM

Simultaneously learn 3 sets of weights:

•How do we guarantee the correct labels?

•Need new constraints!



The "score" of the correct class must be better than the "score" of wrong classes:

$$w^{(y_j)} \cdot x_j + b^{(y_j)} > w^{(y)} \cdot x_j + b^{(y)} \qquad \forall j,\ y \neq y_j$$

Multi-class SVM

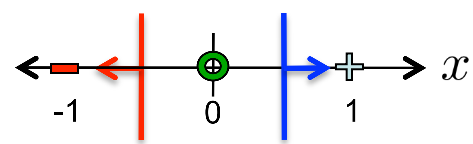
As for the SVM, we introduce slack variables and maximize margin:

$$
\begin{aligned}
&\text{minimize}_{\mathbf{w},b} \quad \textstyle\sum_y \mathbf{w}^{(y)}.\mathbf{w}^{(y)} + C \sum_j \xi_j \\
&\mathbf{w}^{(y_j)}.\mathbf{x}_j + b^{(y_j)} \geq \mathbf{w}^{(y')}.\mathbf{x}_j + b^{(y')} + 1 - \xi_j, \ \ \forall y' \neq y_j, \ \ \forall j \\
&\hfill \xi_j \geq 0, \ \ \forall j
\end{aligned}
$$

To predict, we use:

$$\hat{y} \leftarrow \arg\max_k \ w_k \cdot x + b_k$$

Now can we learn it? →



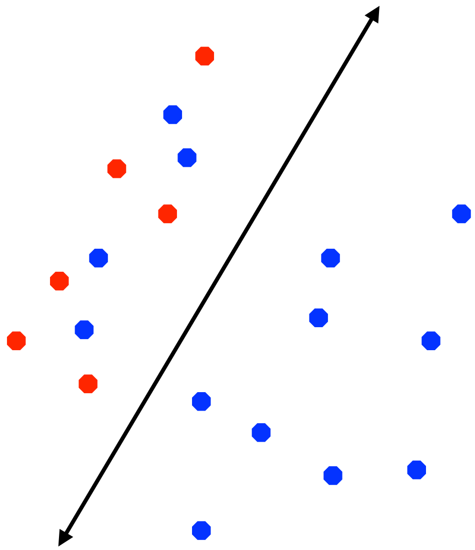
$w_- = -1$ $\quad$ $w_+ = 1$

$b_- = -.5$ $\quad$ $b_+ = -.5$

$w_o = 0$

$b_o = .001$

How to deal with imbalanced data?



- In many practical applications we may have **imbalanced** data sets
- We may want errors to be equally distributed between the positive and negative classes
- A slight modification to the SVM objective does the trick!

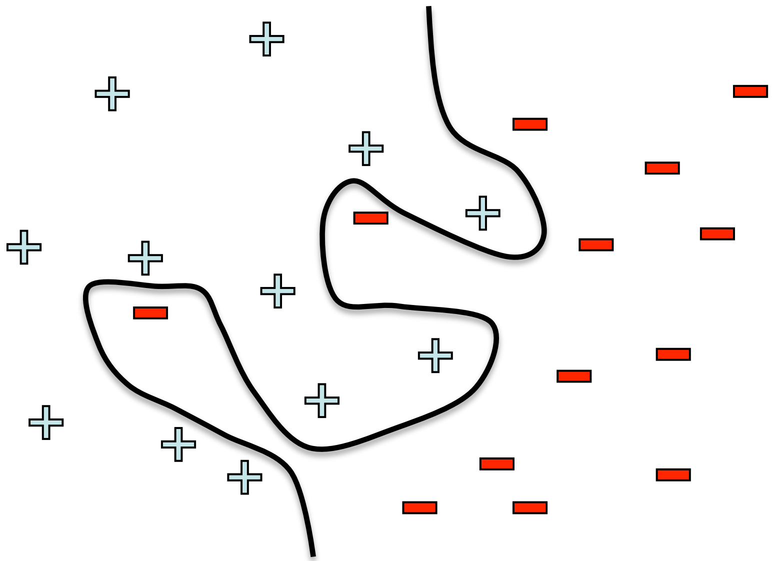
$$N = N_+ + N_-$$

$$\min_{w,b} \; ||w||_2^2 + \frac{CN}{2N_+} \sum_{j:y_j=+1} \xi_j + \frac{CN}{2N_-} \sum_{j:y_j=-1} \xi_j$$

Class-specific weighting of the slack variables

What if the data is not linearly separable?

Use features of features of features of features….



$$\phi(x) = \begin{pmatrix} x^{(1)} \\ \ldots \\ x^{(n)} \\ x^{(1)}x^{(2)} \\ x^{(1)}x^{(3)} \\ \ldots \\ e^{x^{(1)}} \\ \ldots \end{pmatrix}$$

Feature space can get really large really quickly!

Key idea #3: the kernel trick

- High dimensional feature spaces at no extra cost!
- After every update (of Pegasos), the weight vector can be written in the form:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

- As a result, prediction can be performed with:

$$\begin{aligned}
\hat{y} &\leftarrow \mathrm{sign}(\mathbf{w} \cdot \phi(\mathbf{x})) \\
&= \mathrm{sign}\Big((\sum_i \alpha_i y_i \phi(\mathbf{x}_i)) \cdot \phi(\mathbf{x}) \Big) \\
&= \mathrm{sign}\Big(\sum_i \alpha_i y_i (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x})) \Big) \\
&= \mathrm{sign}\Big(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) \Big) \qquad \text{where } K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}').
\end{aligned}$$

Common kernels

- Polynomials of degree exactly *d*

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v})^d$$

- Polynomials of degree up to *d*

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v} + 1)^d$$

- Gaussian kernels

$$K(\vec{u},\vec{v}) = \exp\left(-\frac{||\vec{u}-\vec{v}||_2^2}{2\sigma^2}\right)$$

- Sigmoid

$$K(\mathbf{u},\mathbf{v}) = \tanh(\eta\mathbf{u}\cdot\mathbf{v} + \nu)$$

- And many others: very active area of research!

Polynomial kernel

d=1

$$\phi(u).\phi(v) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} . \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2 = u.v$$

d=2

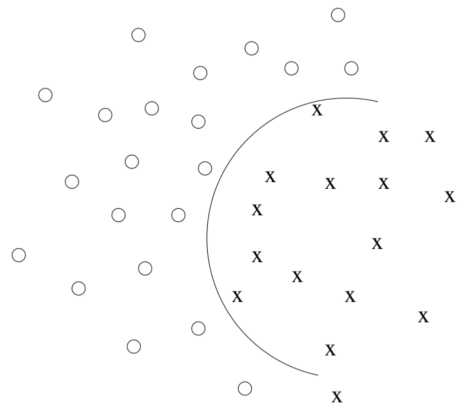
$$\phi(u).\phi(v) = \begin{pmatrix} u_1^2 \\ u_1 u_2 \\ u_2 u_1 \\ u_2^2 \end{pmatrix} . \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2 v_1 \\ v_2^2 \end{pmatrix} \begin{aligned} &= u_1^2 v_1^2 + 2 u_1 v_1 u_2 v_2 + u_2^2 v_2^2 \\ &= (u_1 v_1 + u_2 v_2)^2 \\ &= (u.v)^2 \end{aligned}$$

For any *d* *(we will skip proof):*

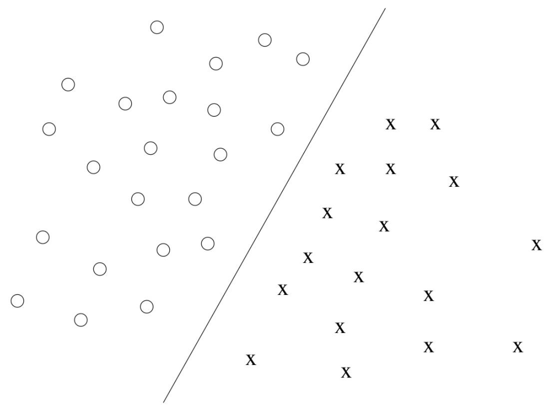
$$\phi(u).\phi(v) = (u.v)^d$$

Polynomials of degree **exactly** *d*

Quadratic kernel



Non-linear separator in the original $\mathbf{x}$-space

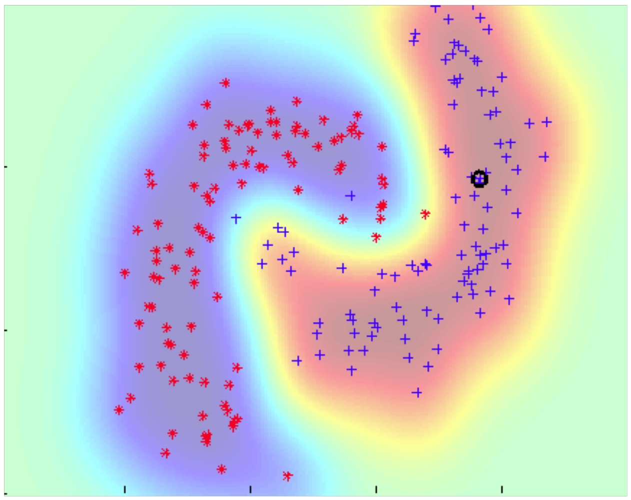


Linear separator in the feature ϕ-space

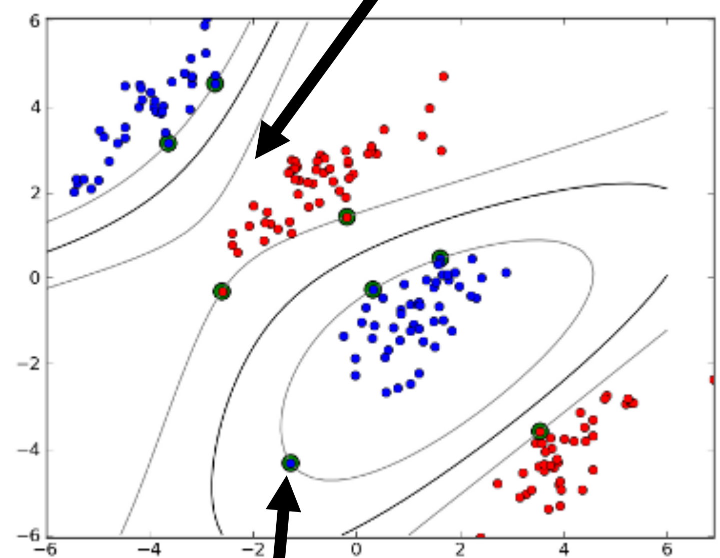
[Tommi Jaakkola]

Gaussian kernel

$$K(\vec{u}, \vec{v}) = \exp\left(-\frac{||\vec{u} - \vec{v}||_2^2}{2\sigma^2}\right)$$



Level sets, i.e. $w \cdot \phi(x) = r$ for some r



Support vectors

$$y \leftarrow \text{sign}\left[\sum_i \alpha_i y_i \exp\left(-\frac{\|\vec{x} - \vec{x}_i\|_2^2}{2\sigma^2}\right) + b\right]$$

[Cynthia Rudin]

[mblondel.org]

Kernel algebra

kernel composition	feature composition
a) $k(\mathbf{x},\mathbf{v}) = k_a(\mathbf{x},\mathbf{v}) + k_b(\mathbf{x},\mathbf{v})$	$\boldsymbol{\phi}(\mathbf{x}) = (\boldsymbol{\phi}_a(\mathbf{x}), \boldsymbol{\phi}_b(\mathbf{x})),$
b) $k(\mathbf{x},\mathbf{v}) = f k_a(\mathbf{x},\mathbf{v}),\ f > 0$	$\boldsymbol{\phi}(\mathbf{x}) = \sqrt{f}\boldsymbol{\phi}_a(\mathbf{x})$
c) $k(\mathbf{x},\mathbf{v}) = k_a(\mathbf{x},\mathbf{v}) k_b(\mathbf{x},\mathbf{v})$	$\phi_m(\mathbf{x}) = \phi_{ai}(\mathbf{x})\phi_{bj}(\mathbf{x})$
d) $k(\mathbf{x},\mathbf{v}) = \mathbf{x}^T A \mathbf{v}$, A positive semi-definite	$\boldsymbol{\phi}(\mathbf{x}) = L^T\mathbf{x}$, where $A = LL^T$.
e) $k(\mathbf{x},\mathbf{v}) = f(\mathbf{x}) f(\mathbf{v}) k_a(\mathbf{x},\mathbf{v})$	$\phi(\mathbf{x}) = f(\mathbf{x})\phi_a(\mathbf{x})$

Q: How would you prove that the "Gaussian kernel" is a valid kernel?
A: Expand the Euclidean norm as follows:

$$\exp\left(-\frac{\|\vec{u}-\vec{v}\|_2^2}{2\sigma^2}\right) = \exp\left(-\frac{\|\vec{u}\|_2^2}{2\sigma^2}\right)\exp\left(-\frac{\|\vec{v}\|_2^2}{2\sigma^2}\right)\exp\left(\frac{\vec{u}\cdot\vec{v}}{\sigma^2}\right)$$

Then, apply (e) from above

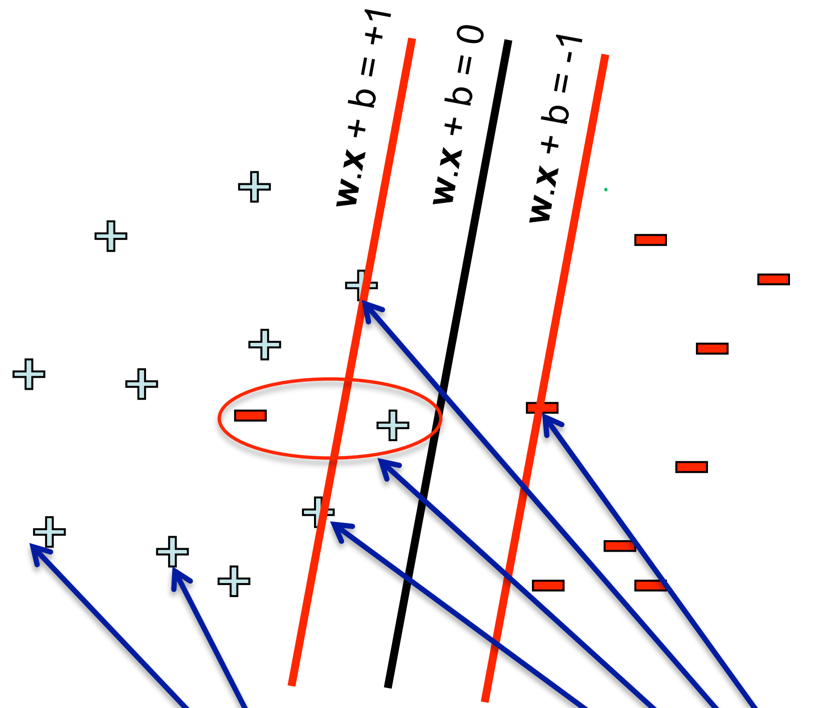
To see that this is a kernel, use the Taylor series expansion of the exponential, together with repeated application of (a), (b), and (c):

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

The feature mapping is infinite dimensional!

[Justin Domke]

Dual SVM interpretation: Sparsity



$$\mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

Final solution tends to be sparse

- α_j=0 for most j
- don’t need to store these points to compute w or make predictions

Non-support Vectors:
- α_j=0
- moving them will not change w

Support Vectors:
- $\alpha_j \geq 0$

Overfitting?

- Huge feature space with kernels: should we worry about overfitting?
  - SVM objective seeks a solution with large **margin**
    - Theory says that large margin leads to good generalization (we will see this in a couple of lectures)
  - But everything overfits sometimes!!!
  - Can control by:
    - Setting C
    - Choosing a better Kernel
    - Varying parameters of the Kernel (width of Gaussian, etc.)