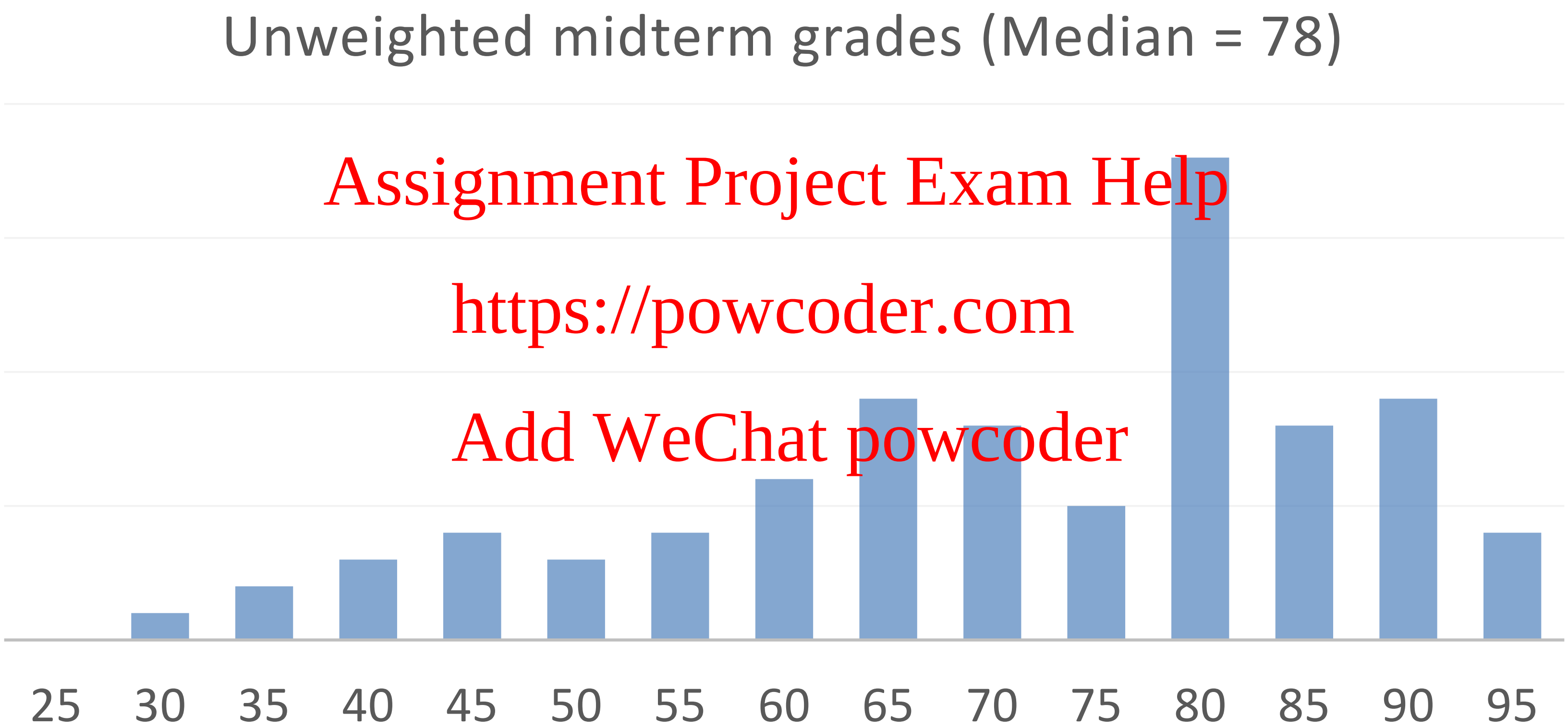
Announcements

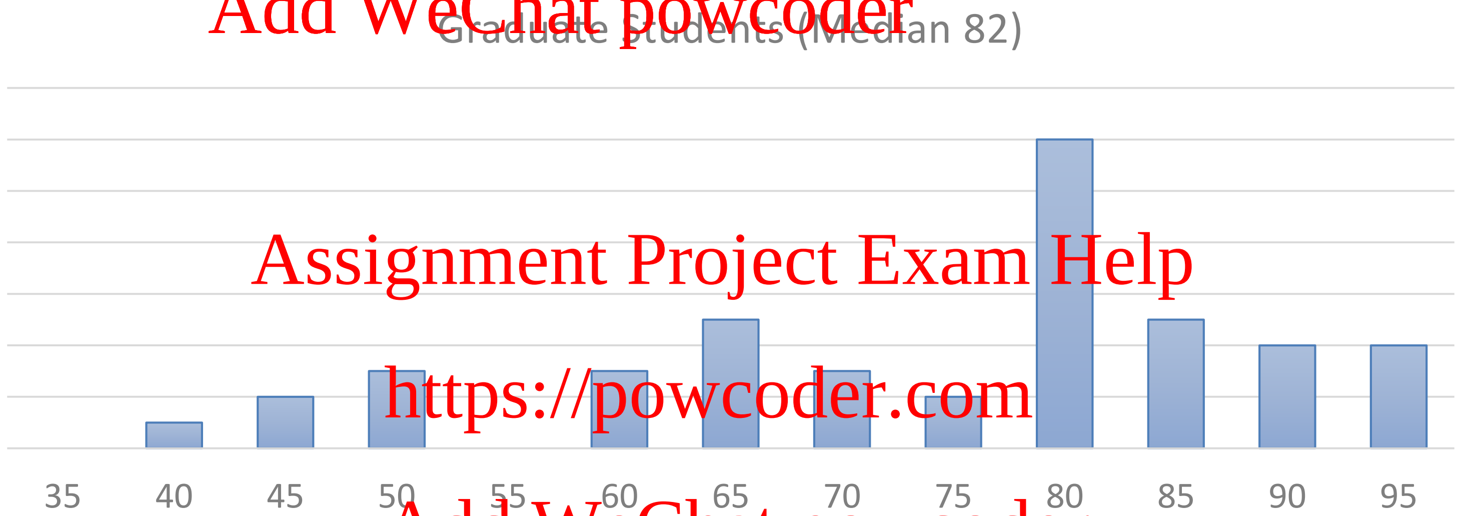
Reminder: ps5 out, due Thursday 11/5

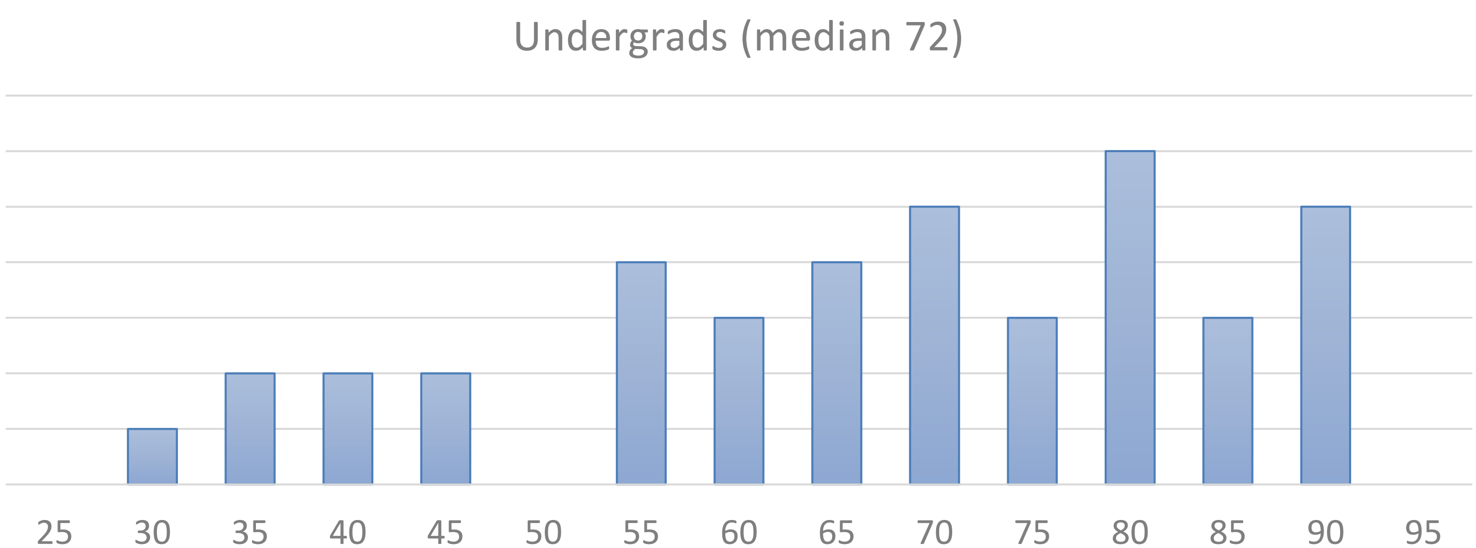
- pset 4 grades up on blackboard by Monday 11/9

Midterm grades out!



Graduate students did better overall





Class grading

- 20% midterm
- 20% final
- 15% class challenge
- 45% homeworks

Assignment Project Exam Help

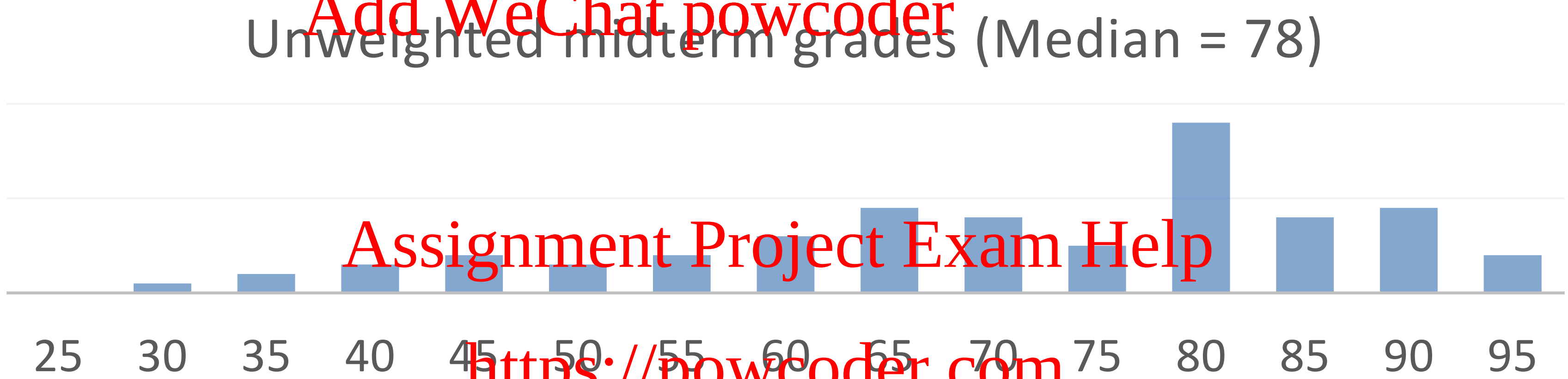
https://powcoder.com

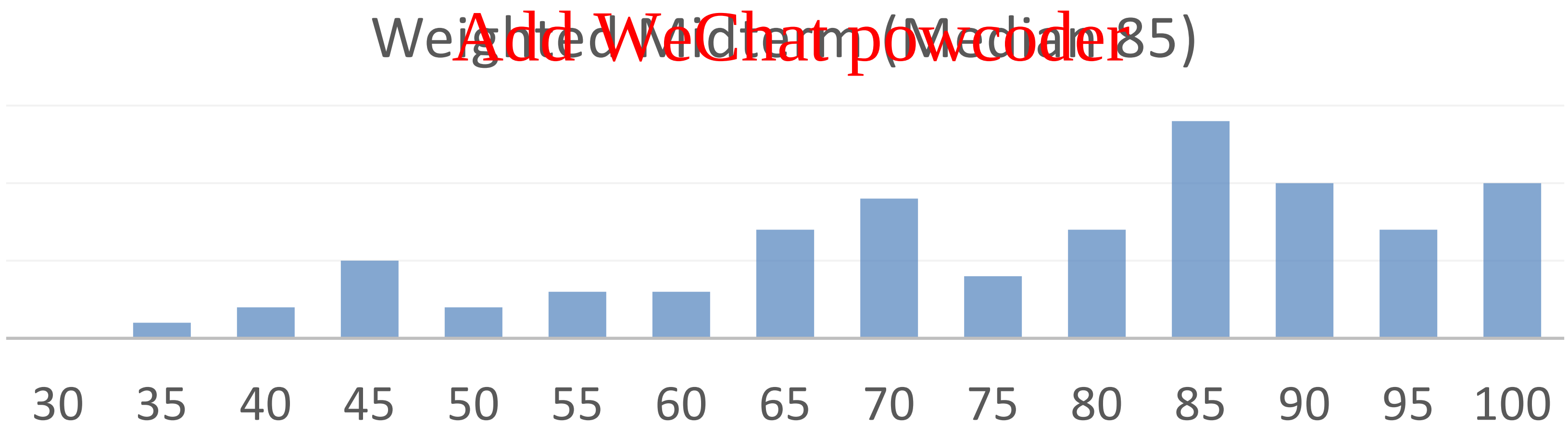
Add WeChat powcoder

Assume student gets 72% on midterm and final, 85% on homework/challenge= ~80% (B-)

72% on midterm, 85% final, 95% homework/challenge=~88% (B+)

Two questions < 50% points awarded, retroactively make them bonus points





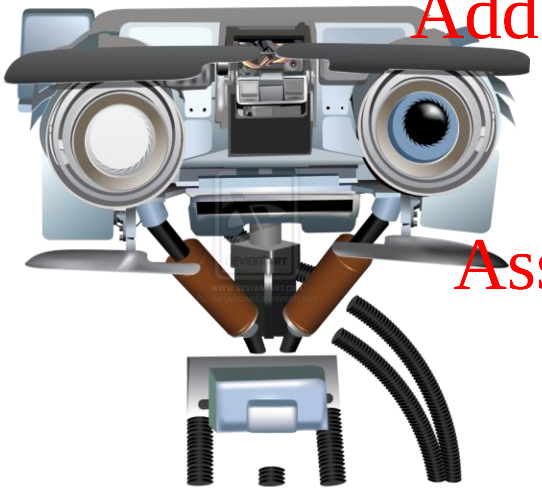
New median for graduates: 89%
New median for undergraduates: 78%

Class grading

- 20% midterm
- 20% final
- 15% class challenge
- 45% homeworks

Assume student gets 78% on midterm and final, 85% on homework/challenge= ~82% (B-/B)

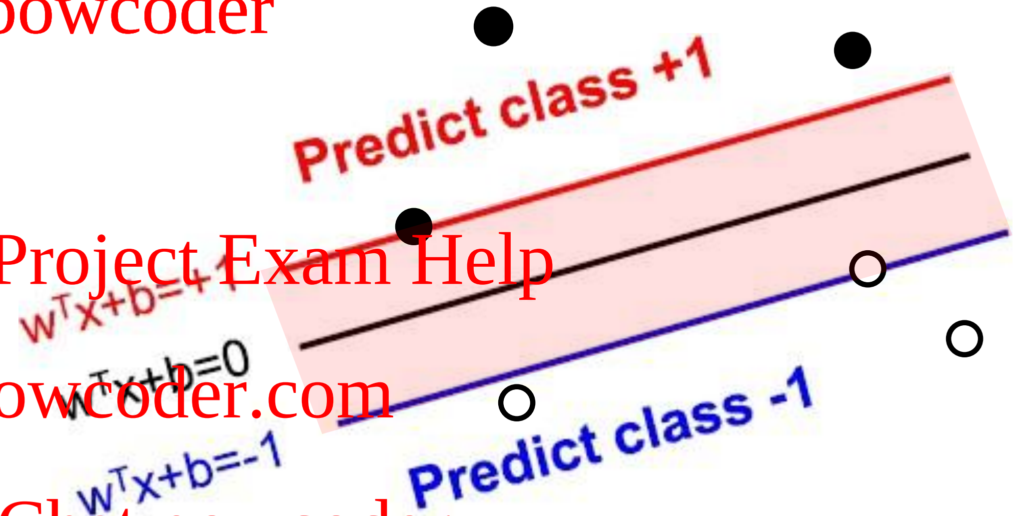
78% on midterm, 88% final, 95% homework/challenge=~90% (A-)



Soft-margin SVM

Max Margin Classifier

“Expand” the decision boundary to include a margin (until we hit first point on either side)



Use margin of 1

Inputs in the margins are of unknown class

Classify as +1	if	$w^Tx+b \geq 1$
Classify as -1	if	$w^Tx+b \leq -1$
Undefined	if	$-1 < w^Tx+b < 1$

Dual vs Primal SVM

n is the number of training points, d is dimension of $\mathbf{x}$, $\mathbf{w}$

Primal problem: for $\mathbf{w} \in \mathbb{R}^{\boldsymbol{d}}$, hyperparameter C, the unconstrained

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|^2 + C \sum_{i}^{n} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)$$

Dual problem: for $\boldsymbol{\alpha} \in \mathbb{R}^{\boldsymbol{n}}$

$$L = \max_{\alpha_i \geq 0} \{ \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) \} \quad \text{s.t } \alpha_i \geq 0; \ \sum_{i=1}^{n} \alpha_i y_i = 0$$

- Efficiency: need to learn d parameters for primal, n for dual

What if data is not linearly separable?

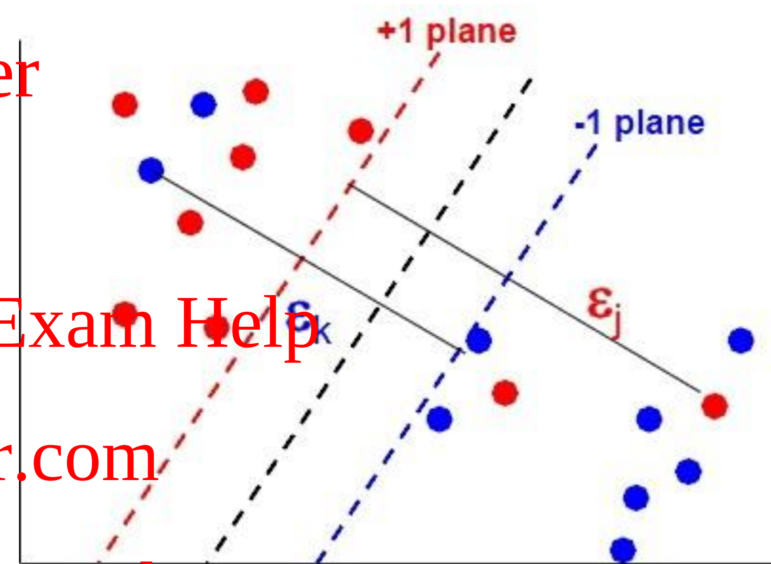
- Introduce slack variables ξ_i

$$\min\left[\frac{1}{2}\|\mathbf{w}\|^2+\lambda\sum_{i=1}^{n}\xi_i\right]$$

subject to constraints (for all *i*):

$$y_i\,(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1-\xi_i$$

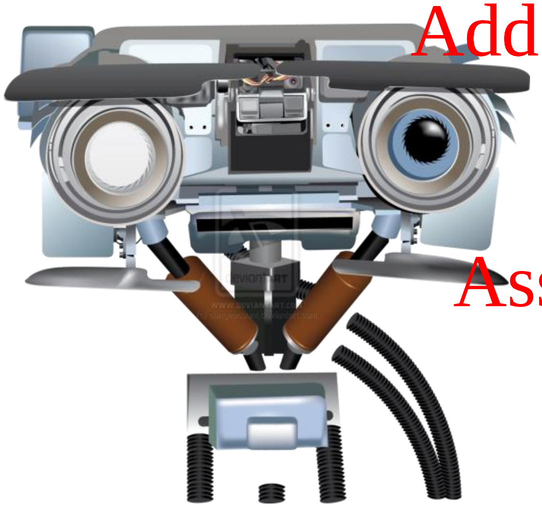
$$\xi_i\geq 0$$



- Example lies on wrong side of hyperplane: $\xi_i>1\Rightarrow\sum_i\xi_i$ is upper bound on number of training errors

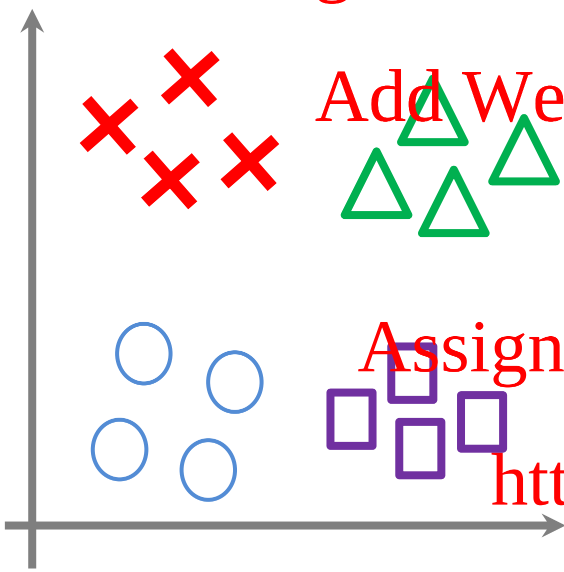
- λ trades off training error versus model complexity

- This is known as the soft-margin extension



Multi-class SVMs

Multi-class classification

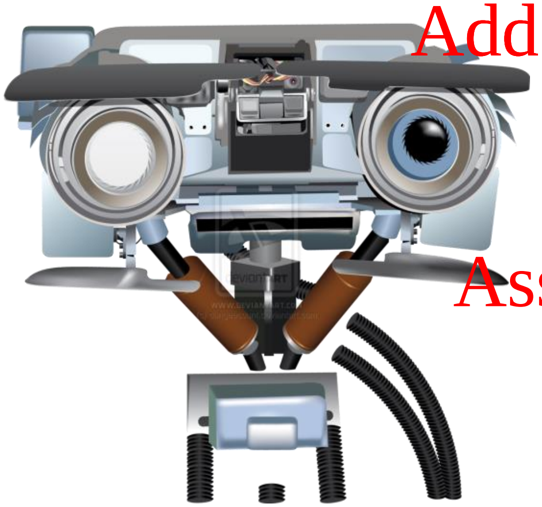


$$y \in \{1, 2, 3, \ldots, K\}$$

Many SVM packages already have built-in multi-class classification functionality.

Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish class i from the rest), for $i = 1, \ldots, K$, get $\mathbf{w}^{(1)}, b^{(1)}, \ldots, \mathbf{w}^{(K)}, b^{(K)}$

Pick class $y = i$ with largest score $\mathbf{w}^{(\mathrm{i})^T} \boldsymbol{x} + b^{(i)}$



Kernel SVM

Non-linear decision boundaries

- Note that both the learning objective and the decision function depend only on dot products between patterns

$$L = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) \qquad y = \mathrm{sign}[b + \mathbf{x} \cdot (\sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i)]$$

- How to form non-linear decision boundaries in input space?

- Basic idea:

1. Map data into feature space $\mathbf{x} \rightarrow \phi(\mathbf{x})$
2. Replace dot products between inputs with feature points
   $$\mathbf{x}_i \cdot \mathbf{x}_j \rightarrow \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$$
3. Find linear decision boundary in feature space

- Problem: what is a good feature function φ(**x**)?

Kernel Trick

- Kernel trick: dot-products in feature space can be computed as a kernel function

$$\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j)$$

- Idea: work directly on **x**, avoid having to compute ϕ(**x**)

- Example:

$$K(\mathbf{a},\mathbf{b}) = (\mathbf{a}\cdot\mathbf{b})^3 = ((a_1, a_2)\cdot(b_1, b_2))^3$$

$$= (a_1 b_1 + a_2 b_2)^3$$

$$= a_1^3 b_1^3 + 3a_1^2 b_1^2 a_2 b_2 + 3a_1 b_1 a_2^2 b_2^2 + a_2^3 b_2^3$$

$$= (a_1^3, \sqrt{3}a_1^2 a_2, \sqrt{3}a_1 a_2^2, a_2^3)\cdot(b_1^3, \sqrt{3}b_1^2 b_2, \sqrt{3}b_1 b_2^2, b_2^3)$$

$$= \phi(\mathbf{a})\cdot\phi(\mathbf{b})$$

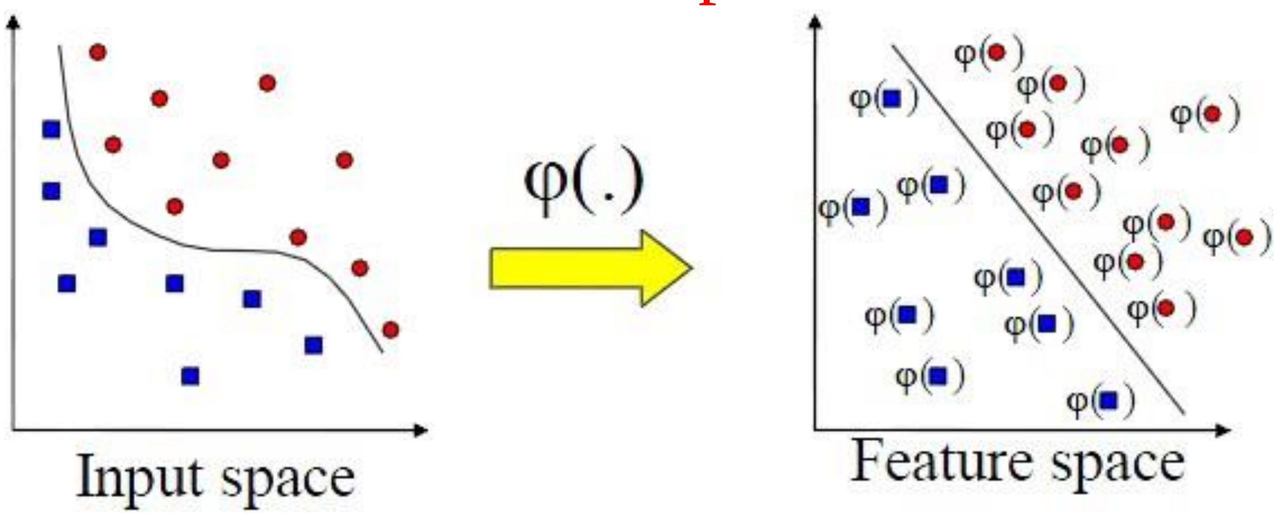
Input transformation

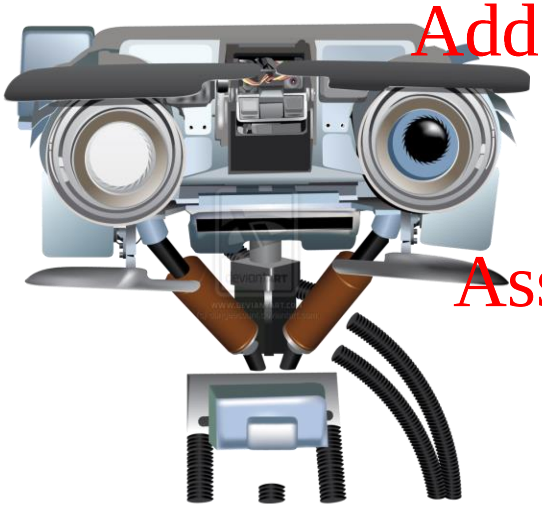
Mapping to a feature space can produce problems:

- High computational burden due to high dimensionality
- Many more parameters

SVM solves these two issues simultaneously

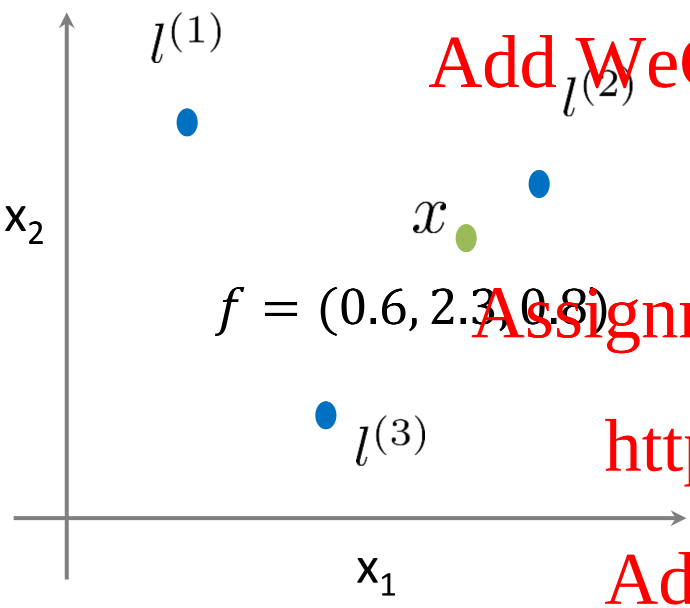
- Kernel trick produces efficient classification
- Dual formulation only assigns parameters to samples, not features





Kernels

Kernels as Similarity Functions



Given x, compute new feature depending on proximity to "landmarks" $l^{(1)}, l^{(2)}, l^{(3)}$

Example: Gaussian (RBF) kernel

$$f_1 = similarity(x, l^{(1)}) = \exp\left(-\frac{\left\| x - l^{(1)} \right\|^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

similarity is high

If x if far from $l^{(1)}$:

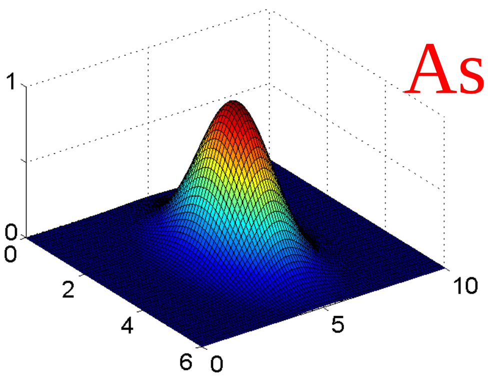
similarity is low

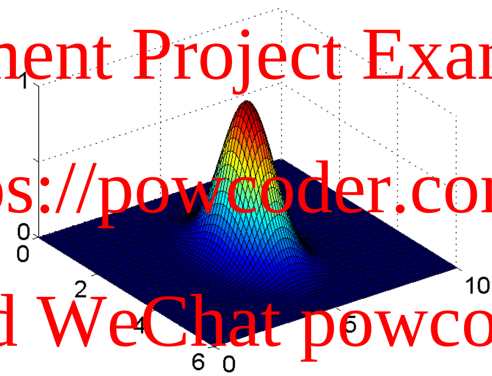
Predict label "1" when $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

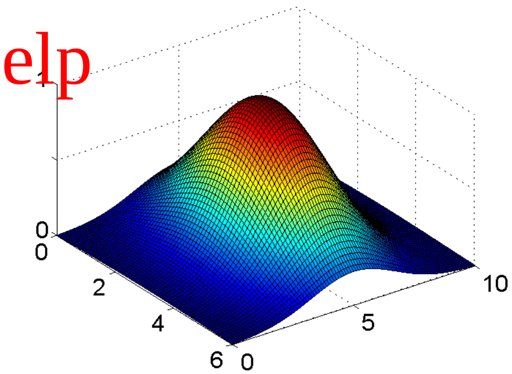
Example:

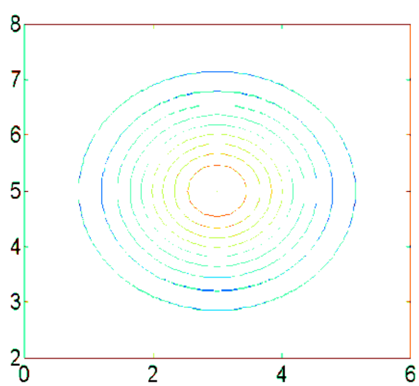
$$l^{(1)} = \begin{bmatrix}3\\5\end{bmatrix}, \quad f_1 = \exp\left(-\frac{\|x-l^{(1)}\|^2}{2\sigma^2}\right)$$

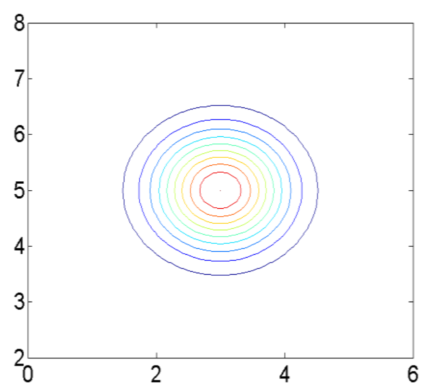
$\sigma^2 = 1$ $\sigma^2 = 0.5$ $\sigma^2 = 3$

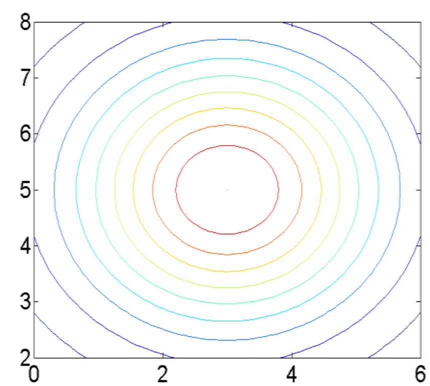




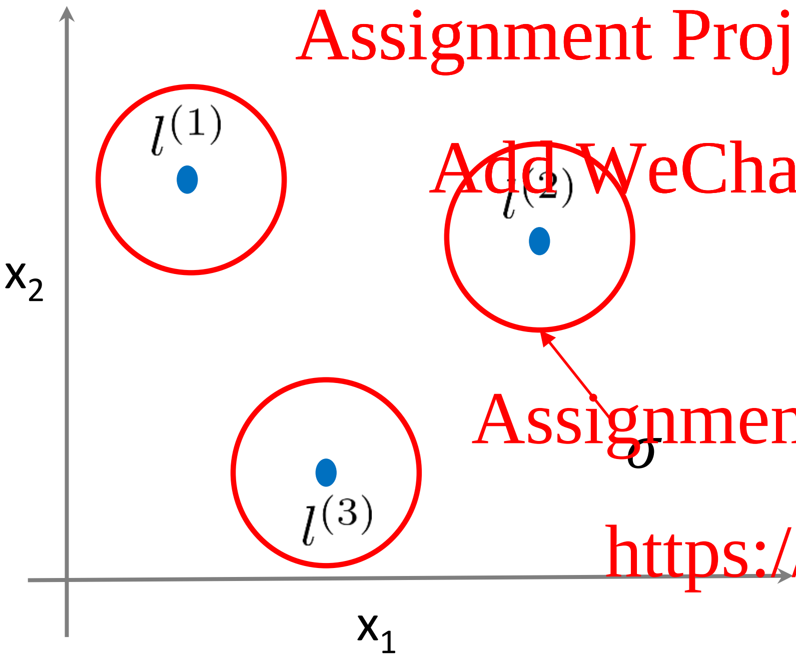








Landmarks for Gaussian kernel



Given x:

$$f_i = \text{similarity}(x, l^{(i)})$$

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

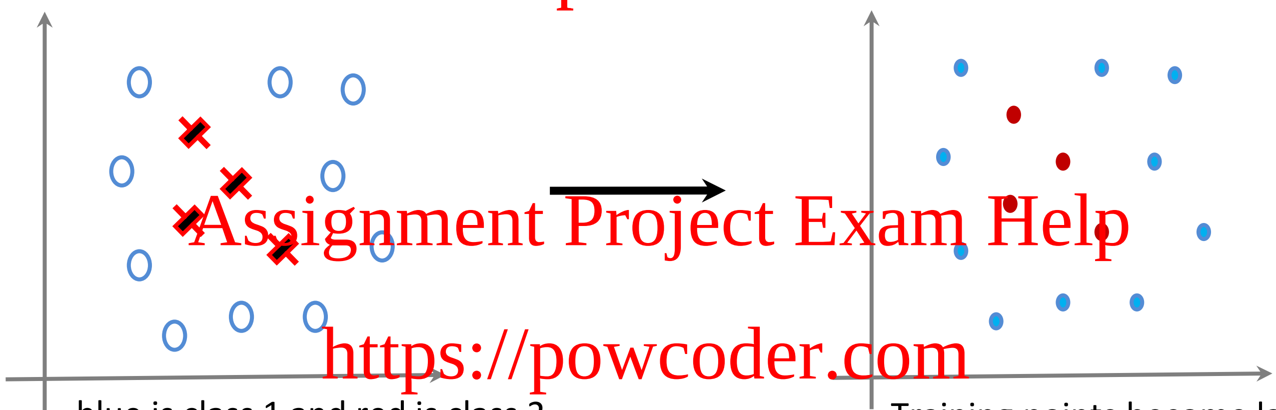
Predict $y = 1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

So the new features f measure how close the example is to each "landmark" point

Where do the landmarks come from?

Landmarks for Gaussian kernel

Where do $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$ come from? They are the training points!



blue is class 1 and red is class 2

Training points become landmarks

So the "landmarks" are points we can use to compute a new feature representation for a point x, by representing it as the similarity to each landmark point (measured using a Gaussian centered at the landmark)

In SVMs with RBF (Gaussian) kernels, we place a Gaussian centered at **each** training point to compute the nonlinear features. This is equivalent to using all of the training data as landmarks.

SVM with Kernels

Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$,

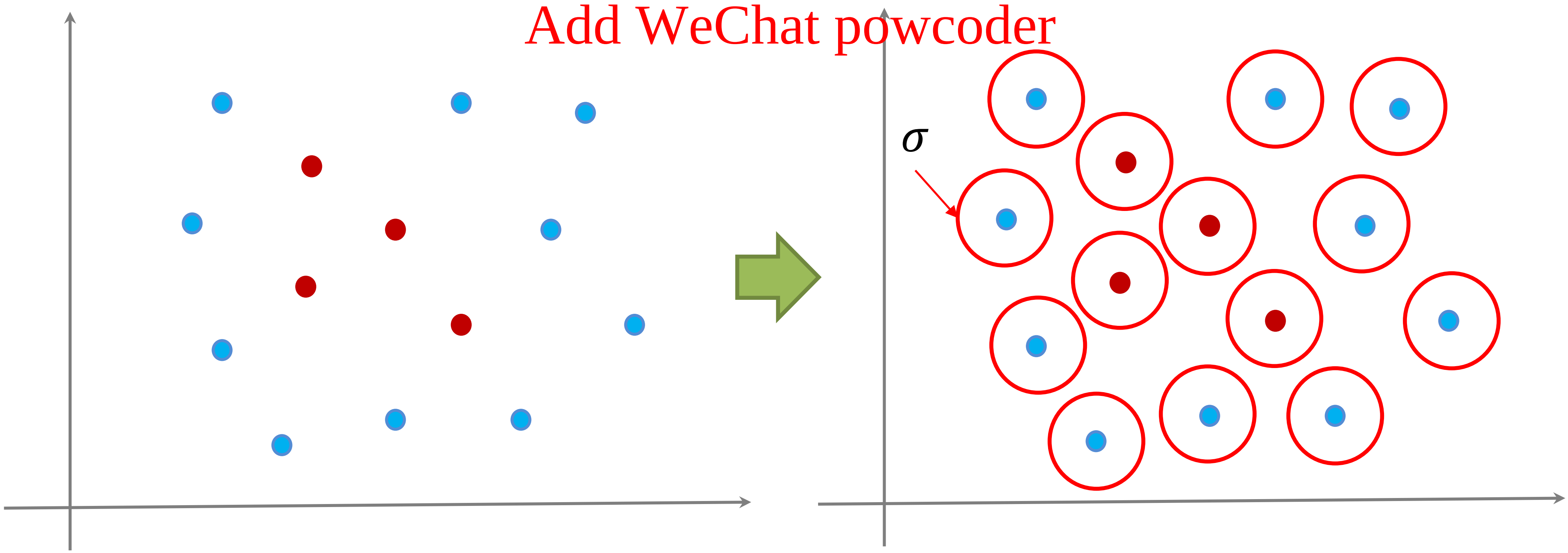
choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

Given example x:

$$f_1 = \text{similarity}(x, l^{(1)})$$

$$f_2 = \text{similarity}(x, l^{(2)})$$

$$\ldots$$



Kernels

Examples of kernels (kernels measure similarity):

1. Polynomial $K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2 + 1)^2$
2. Gaussian $K(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\left\|\mathbf{x}_1 - \mathbf{x}_2\right\|^2 / 2\sigma^2)$
3. Sigmoid $K(\mathbf{x}_1, \mathbf{x}_2) = \tanh(\kappa(\mathbf{x}_1 \cdot \mathbf{x}_2) + a)$

Each kernel computation corresponds to dot product calculation for particular mapping φ(x): implicitly maps to high-dimensional space

Why is this useful?

1. Rewrite training examples using more complex features
2. Dataset not linearly separable in original space may be linearly separable in higher dimensional space

Classification with non-linear SVMs

Non-linear SVM using kernel function *K()*:

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

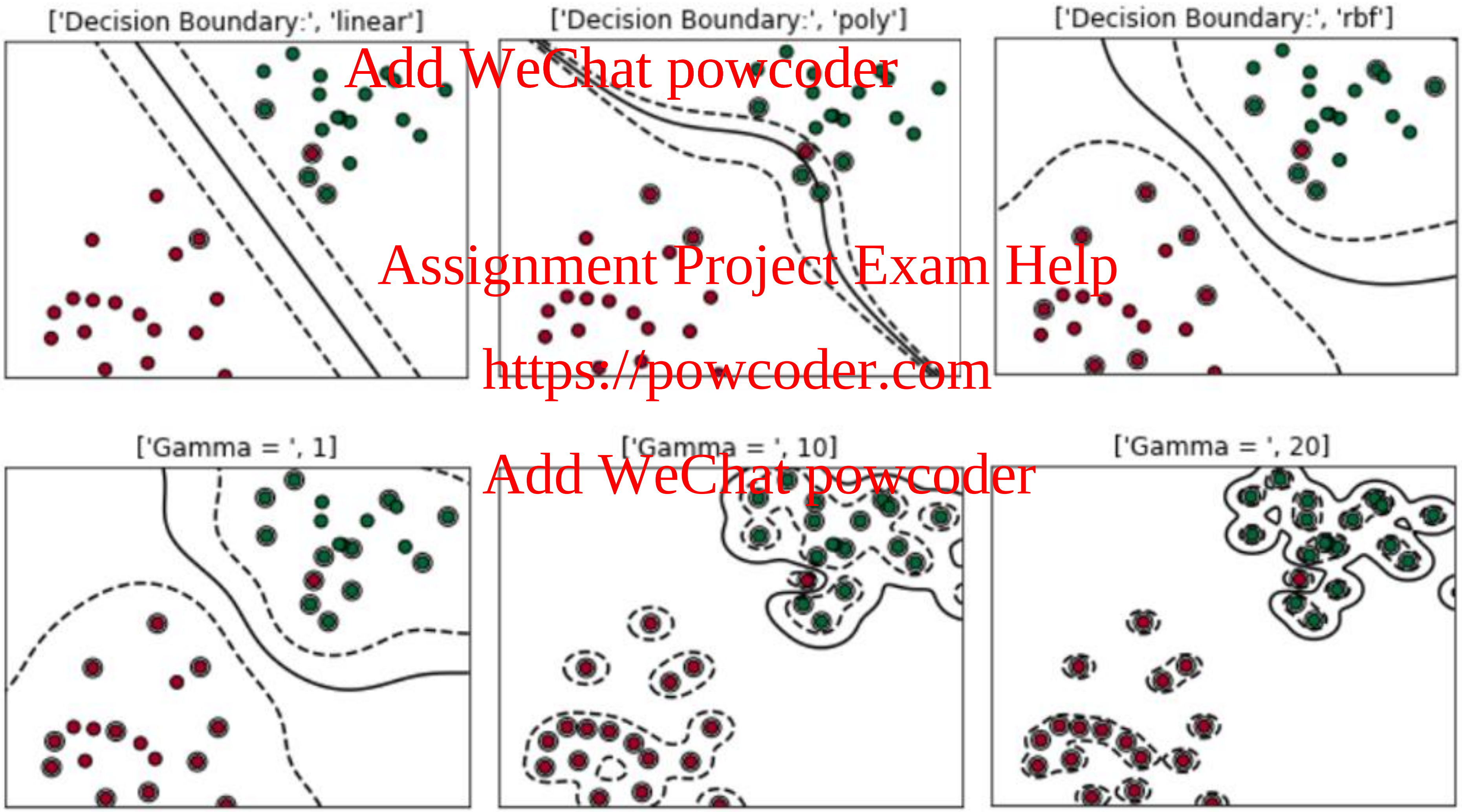
Maximize L_K w.r.t. {α}, under constraints α≥0

Unlike linear SVM, cannot express w as linear combination of support vectors – now must retain the support vectors to classify new examples

Final decision function:

$$y = \mathrm{sign}[b + \sum_{i=1}^{n} y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i)]$$

Decision boundary in Gaussian kernel SVM



Gammma = $1/\sigma^2$

towardsdatascience.com

Kernel SVM Summary

Advantages:

- Kernels allow very flexible hypotheses
- Poly-time exact optimization methods rather than approximate methods
- Soft-margin extension permits mis-classified examples
- Variable-sized hypothesis space
- Excellent results (1.1% error rate on handwritten digits vs. LeNet's 0.9%)

Disadvantages:

- Must choose kernel hyperparameters
- Very large problems computationally intractable
- Batch algorithm

Kernel Functions

Mercer's Theorem (1909): any reasonable kernel corresponds to some feature space

Reasonable means that the Gram matrix is positive definite

$$K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$$

Feature space can be very large, e.g., polynomial kernel $(1+ \mathbf{x}_i + \mathbf{x}_j)^d$ corresponds to feature space exponential in d

Linear separators in these super high-dim spaces correspond to highly nonlinear decision boundaries in input space

Kernelizing

A popular way to make an algorithm more powerful is to develop a kernelized version of it

- We can rewrite a lot of algorithms to be defined only in terms of inner product

- For example: k-nearest neighbors

$$\mathbf{z} = \varphi(\mathbf{x})$$

$$(\mathbf{z}_i - \mathbf{z}_j)^2 = K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_j, \mathbf{x}_j) - 2K(\mathbf{x}_i, \mathbf{x}_j)$$

Techniques for constructing valid kernels

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}') \tag{6.13}$$

$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \tag{6.14}$$

$$k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}')) \tag{6.15}$$

$$k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}')) \tag{6.16}$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \tag{6.17}$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \tag{6.18}$$

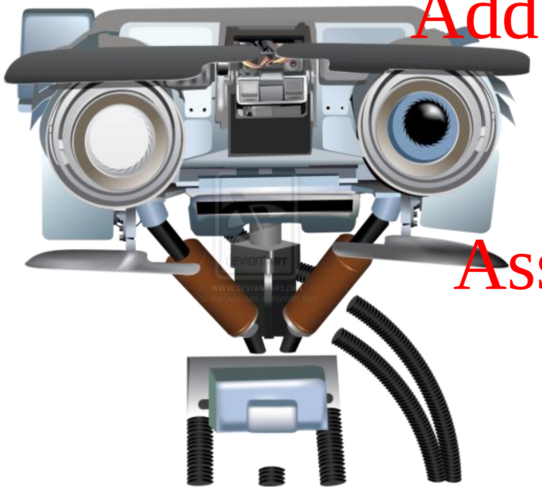
$$k(\mathbf{x}, \mathbf{x}') = k_3(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')) \tag{6.19}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' \tag{6.20}$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \tag{6.21}$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a)k_b(\mathbf{x}_b, \mathbf{x}'_b) \tag{6.22}$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.



Summary of SVMs

Summary

Software:

- A list of SVM implementations can be found at http://www.kernel-machines.org/software.html
- Some implementations (such as LIBSVM) can handle multi-class classification
- SVMLight is among the earliest implementations
- Several Matlab toolboxes for SVM are also available

Key points:

- Difference between logistic regression and SVMs
- Maximum margin principle
- Target function for SVMs
- Slack variables for mis-classified points
- Kernel trick allows non-linear generalizations

History of SVMs

- The original SVM algorithm was invented by Vladimir Vapnik and Alexey Chervonenkis in 1963.

- In 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes.[13]

- The soft margin was proposed by Corinna Cortes and Vapnik in 1993 and published in 1995.[1]

- SVMs were very popular in the 90's-00's until neural networks took over circa 2012

https://en.wikipedia.org/wiki/Support-vector_machine

Next Class

Reinforcement Learning I

reinforcement learning; Markov Decision Process (MDP); policies, value functions, Q-learning