CMSC5741 Big Data Tech. & Apps.

Lecture 9: Large Scale Support Vector Machines

Prof. Michael R. Lyu
Computer Science & Engineering Dept.
The Chinese University of Hong Kong

Motivation

- Introduce the widely used classification tool: Support Vector Machine (SVM)

- Understand the model and parameter estimation method in terms of big data

Motivation

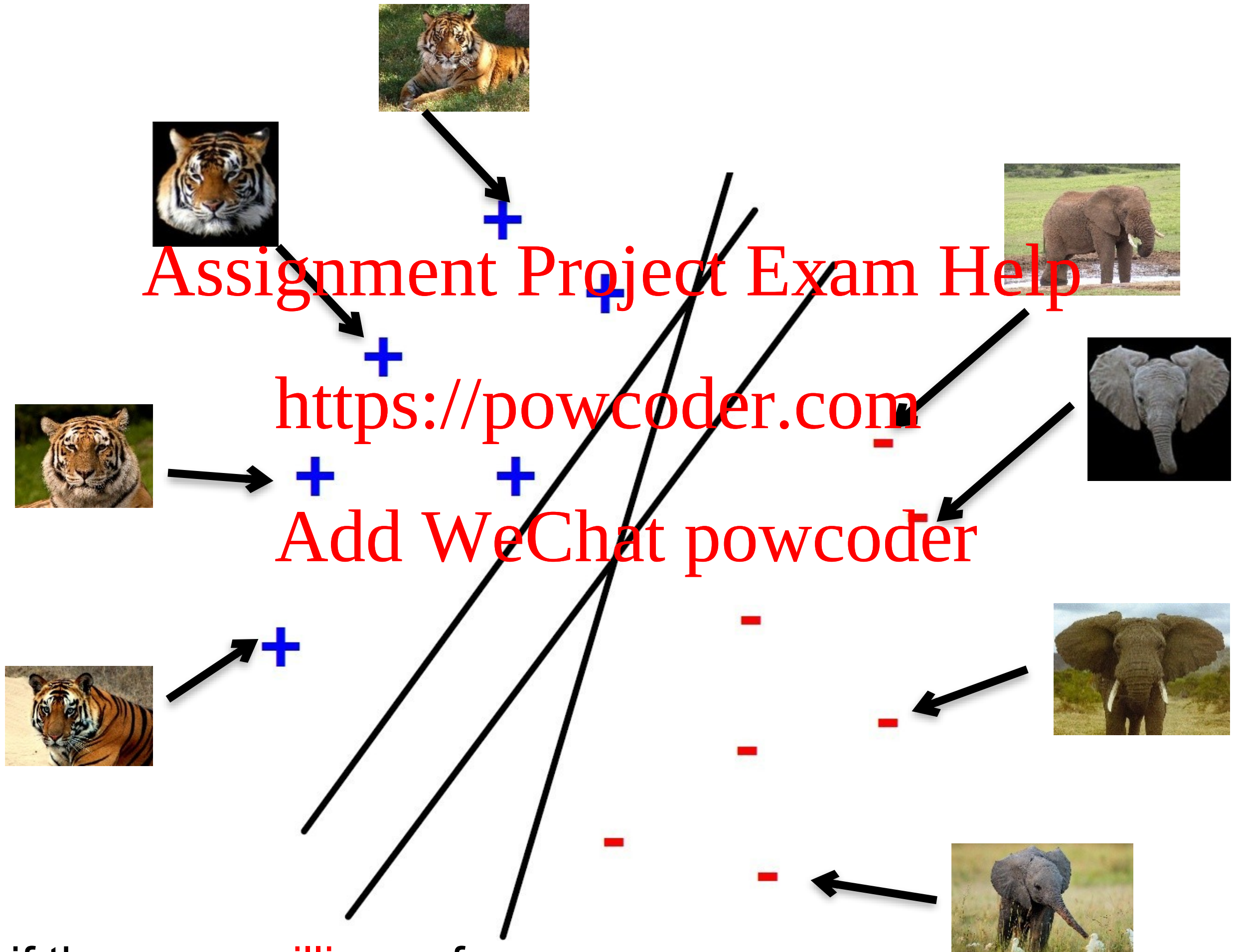
Suppose we have 50 photographs of elephants and 50 photos of tigers.

vs.

We digitize them into 100 x 100 pixel images, so we have $x \in R^n$ where $n = 10,000$.

Now, given a new (different) photograph we want to answer the question: **is it an elephant or a tiger?** [we assume it is one or the other.]

Motivation



What if there are millions of photos, how to make the SVM training scalable?

Outline

- Support Vector Machines
  - History
  - Linear Separable SVMs
  - Non-linear Separable SVMs
    - Soft Margin
    - Kernel Trick
- Parameter Estimation
- Further Reading

Outline

- Support Vector Machines
  - History
  - Linear Separable SVMs
  - Non-linear Separable SVMs
    - Soft Margin
    - Kernel Trick
- Parameter Estimation
- Further Reading

SVMs: History

- SVMs introduced in COLT-92 by Boser, Guyon & Vapnik. Became rather popular since.

- Theoretically well motivated algorithm: developed from Statistical Learning Theory (Vapnik & Chervonenkis) since the 60s.

- Empirically good performance: successful applications in many fields (bioinformatics, text, image recognition, . . .)

SVMs: History

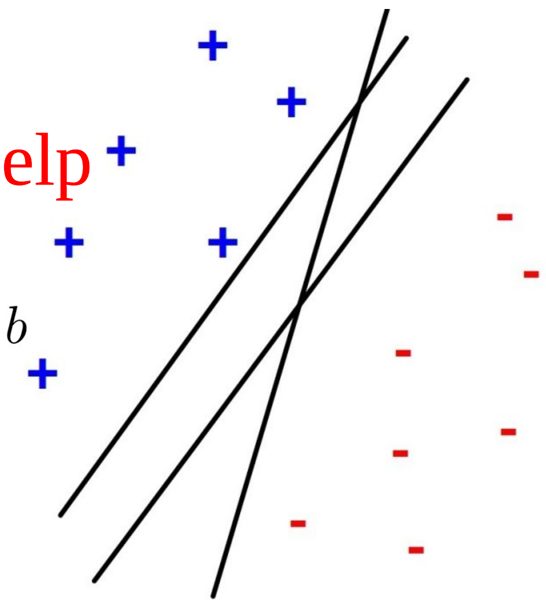
- Centralized website: www.kernel-machines.org.
- Several textbooks, e.g. "An introduction to Support Vector Machines" by Cristianini and Shawe-Taylor is one.
- A large and diverse community work on them: from machine learning, optimization, statistics, neural networks, functional analysis, etc.

Outline

- Support Vector Machines
  - History
  - Linear SVMs
  - Non-linear SVMs
    - Soft Margin
    - Kernel Trick
- Parameter Estimation
- Further Reading

Linear SVMs

- Data
  - Training examples: $(x_1, y_1), \ldots, (x_n, y_n)$
  - Each $x_i \in \mathbb{R}^d, y_i \in \{+1, -1\}$
  - Want to find a hyperplane $y = w^T x + b$ to separate "+" from "-"
- What's the **best** hyperplane defined by w ?

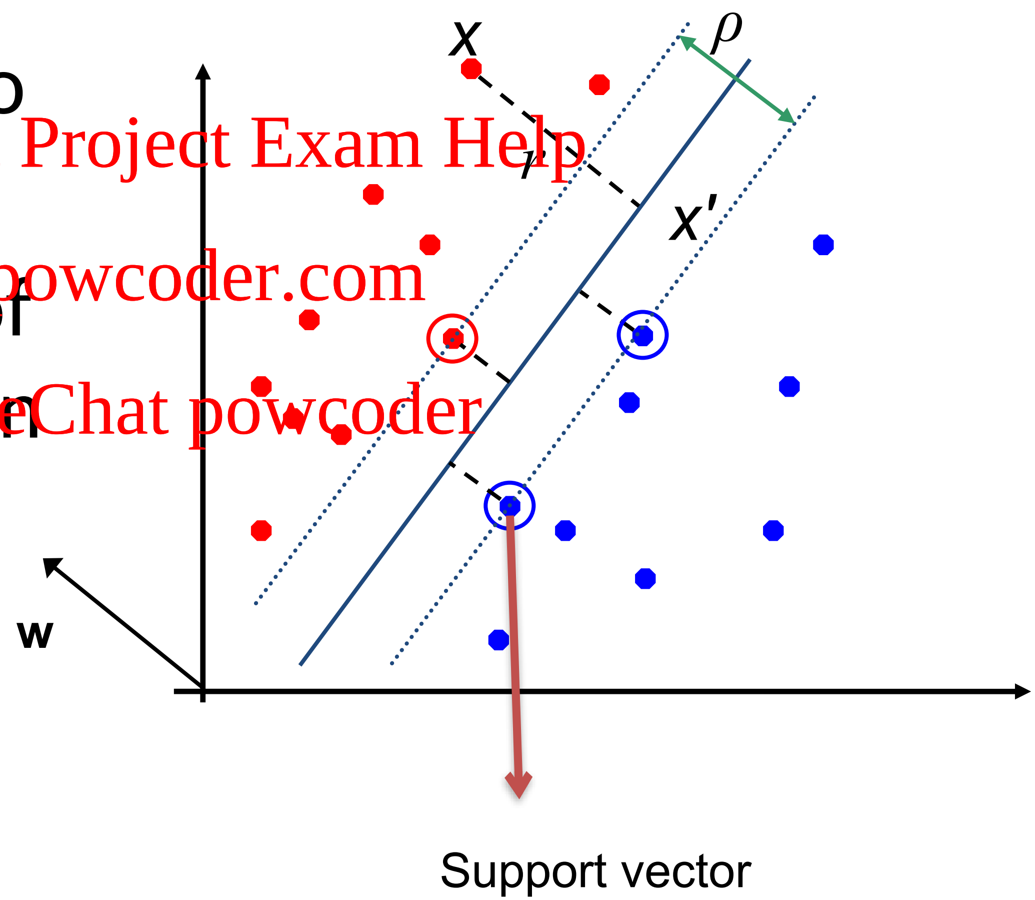


Largest Margin

- Distance from the separating hyperplance corresponds to the "confidence" of prediction
- Example: We have more confidence to say A and B belong to "+" than C

Largest Margin

- **Support Vectors**: Examples closest to the hyperplane
- **Margin** ρ : width of separation between support vectors of classes.



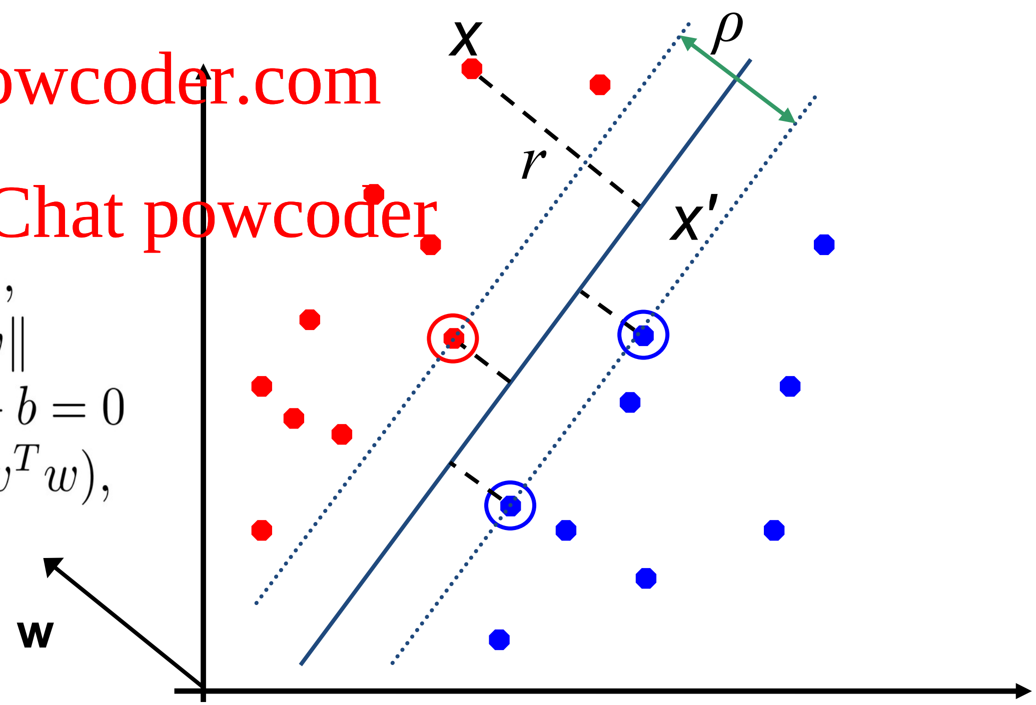
Largest Margin

- Distance from example to the separator is :

$$r = y\frac{w^T x + b}{\|w\|}$$

- Proof:

$x' - x // w$, unit vector is $w/\|w\|$,
so line is $rw/\|w\|$, $x' = x - yrw/\|w\|$
since x' is on the separator, $w^T x' + b = 0$
so $w^T(x - yrw/\|w\|)$+b=0,||w||= $\sqrt{(w^T w)}$,
so $w^T x - yr\|w\| + b = 0$,
then we get $r = y\frac{w^T x+b}{\|w\|}$



Largest Margin

- Assume that all data is at least distance 1 from the hyperplane, then the following constraints follow for a training set $\{(x_i, y_i)\}_{i=1}^{n}$

$$y_i(w^T x_i + b) \geq 1$$

- For support vectors, the inequality becomes an equality
- Recall that $$r = y\frac{w^T x + b}{\|w\|}$$
- Margin is: $$\rho = \frac{2}{\|w\|}$$

Linear SVMs

- Note that we assume that all data points are linearly separated by the hyperplane.
- The margin is invariant to scaling of parameters.
  - i.e. by changing w, b to 5w, 5b, the margin doesn't change

Linear SVMs

- Maximize the margin
  - Good according to intuition, theory (VC dimension) & practice

- The problem of linear SVMs is formulated as:

$$\max_{w} \rho = \frac{2}{\|w\|}$$
$$s.t. \quad y_i(w^T x_i + b) \geq 1 \quad \forall i = 1, \ldots, n$$

- An equivalent form is:

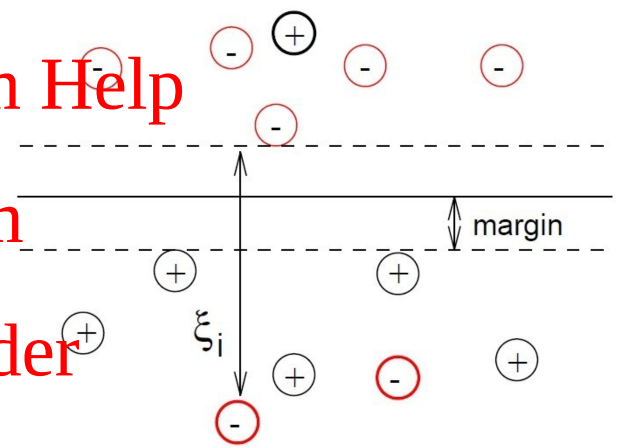
$$\min_{w} \frac{1}{2}\|w\|^2$$
$$s.t. \quad y_i(w^T x_i + b) \geq 1 \quad \forall i = 1, \ldots, n$$

Outline

- Support Vector Machines
  - History
  - Linear Separable SVMs
  - Non-linear Separable SVMs
    - Soft Margin
    - Kernel Trick
- Parameter Estimation
- Further Reading

Non-Linear Separable SVMs

- In reality, training samples are usually not linearly separable.
- Soft Margin Classification
  - Idea: allow errors but introduce slack variable ξ_i to penalize errors
  - Still try to minimize training set errors, and to place hyperplane “far” from each class (large margin)



Soft Margin Classification

- The problem becomes:

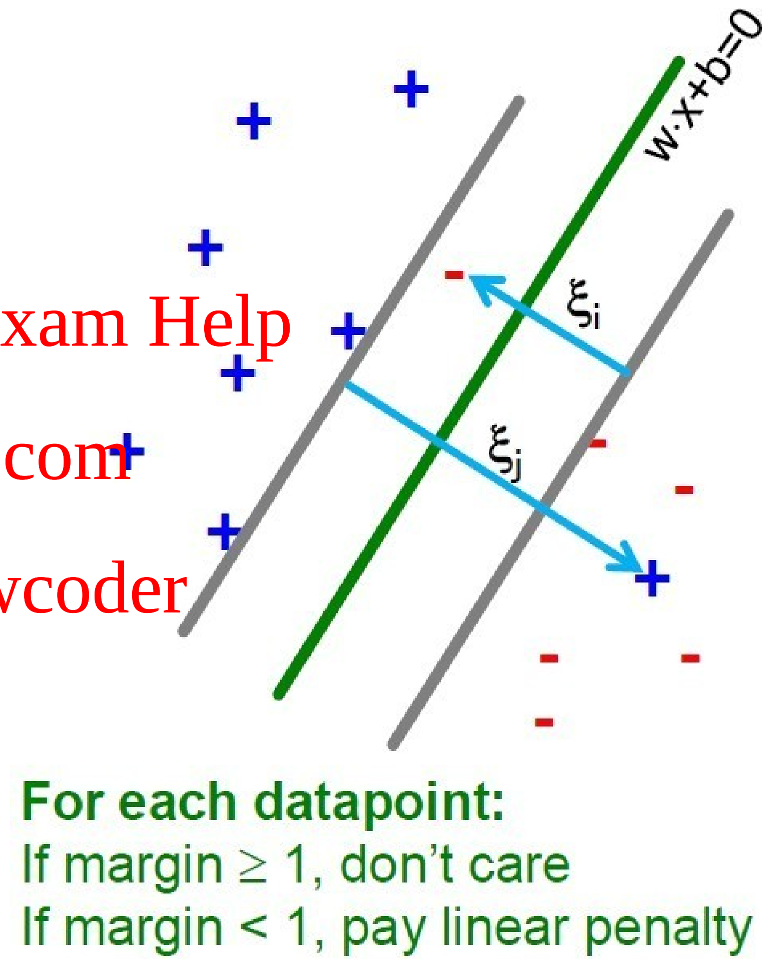
$$\min_w \frac{1}{2}\|w\|^2 + C\sum \xi_i$$

$$s.t. \quad y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i = 1, \ldots, n$$

  - Minimize $\|w\|^2$ plus the number of training mistakes
  - Set C using cross validation

Soft Margin Classification

- If point x_i is on the wrong side of the margin then get penalty ξ_i
- Thus all mistakes are not equally bad!

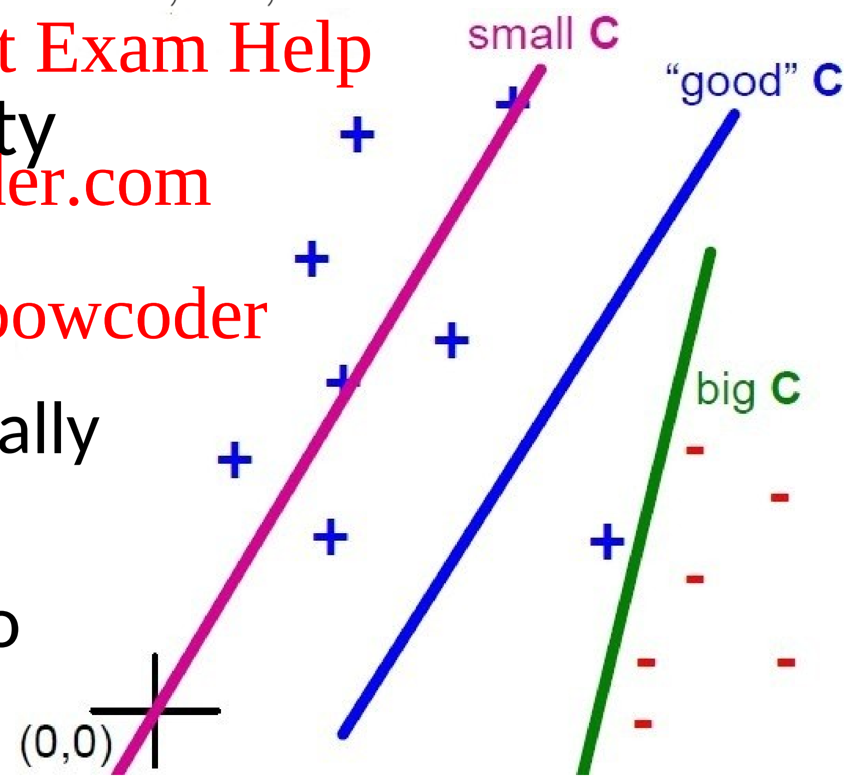


Slack Penalty C

$$\min_{w} \frac{1}{2}\|w\|^2 + C\sum \xi_i$$

$$s.t. \quad y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i = 1, \ldots, n$$

- What is the role of penalty C:
  - $C = 0$: can set ξ_i to anything, then w=0 (basically ignore the data)
  - $C = \infty$: Only want w, b to separate the data



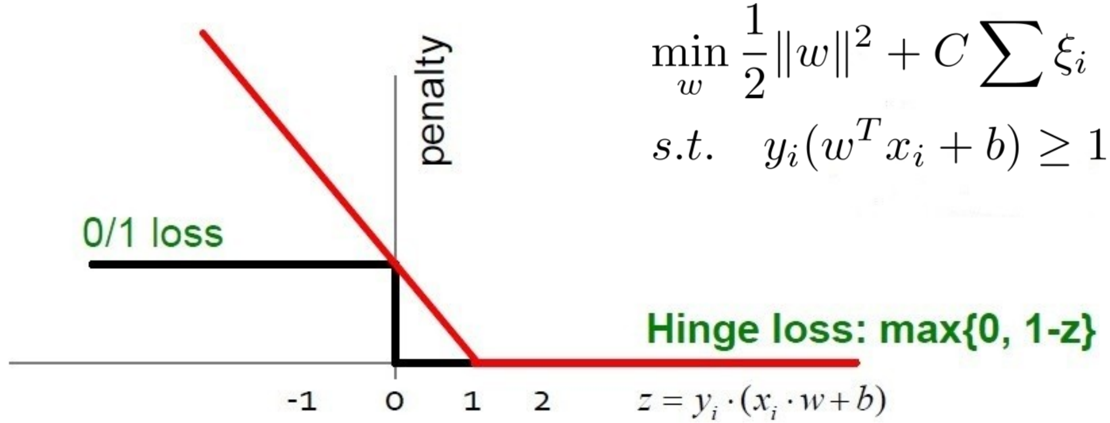
Soft Margin Classification

- SVM in the “natural” form

$$\arg\min_{w,b} \frac{1}{2}\underbrace{\|w\|^2}_{\text{Margin}} + C \underbrace{\sum_{i=1}^{n} \max\{0, 1 - y_i(w^T x_i + b)\}}_{\text{Empirical loss L}}$$

Regularization Parameter

- SVM uses “Hinge Loss”:



$$\min_{w} \frac{1}{2}\|w\|^2 + C \sum \xi_i$$

$$s.t. \quad y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i = 1, \ldots, n$$

In-class Practice

- Go to practice

Outline

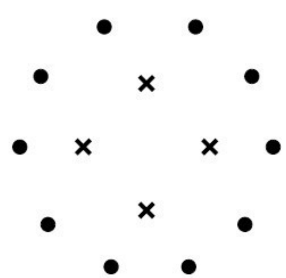
- Support Vector Machines
  - History
  - Linear SVMs
  - Non-linear SVMs
    - Soft Margin
    - Kernel Trick
- Parameter Estimation
- Further Reading

Non-linear Separable SVMs

- Linear classifiers aren't complex enough sometimes.

  - Map data into a richer feature space including non-linear features

  - Then construct a hyperplane in that space so all other equations are the same



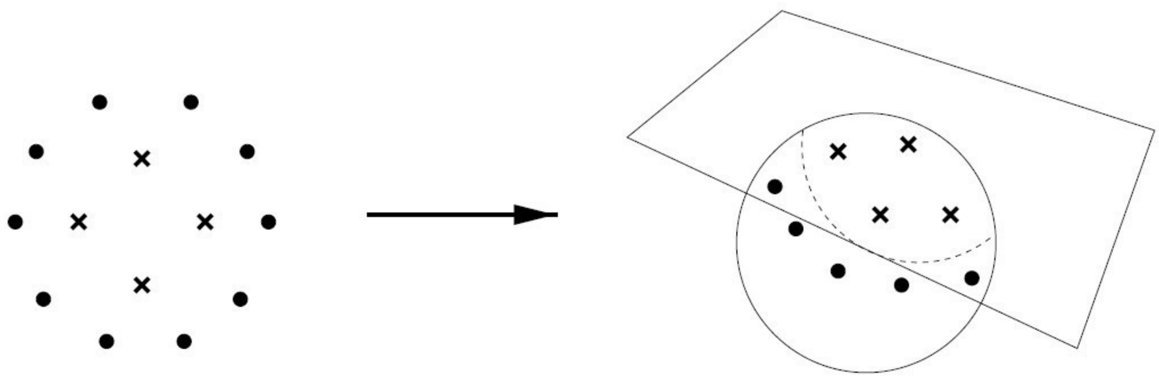
Non-linear Separable SVMs

- Formally, process the data with:

$$x \mapsto \Phi(x)$$

- Then learn the map from $\Phi(x)$ to y

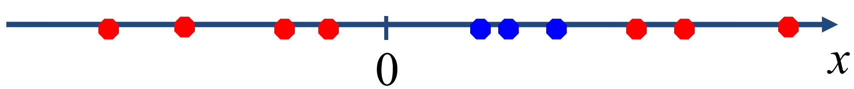
$$f(x) = w \cdot \Phi(x) + b$$

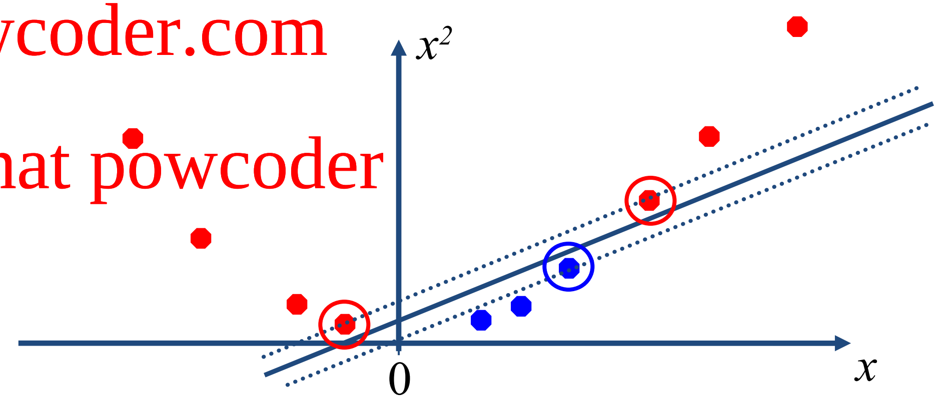


Example: Polynomial Mapping

$$\Phi : R \rightarrow R^2$$

$$(x_1) \mapsto (z_1, z_2) := (x_1, x_1^2)$$

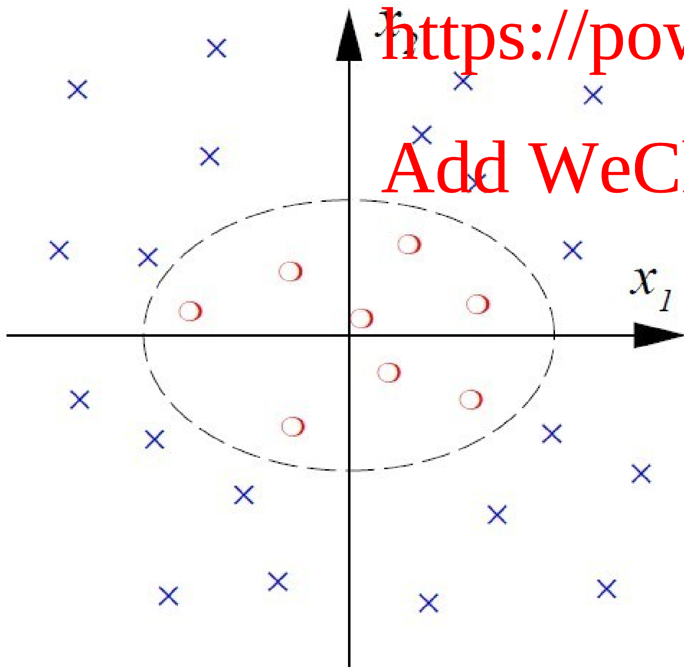


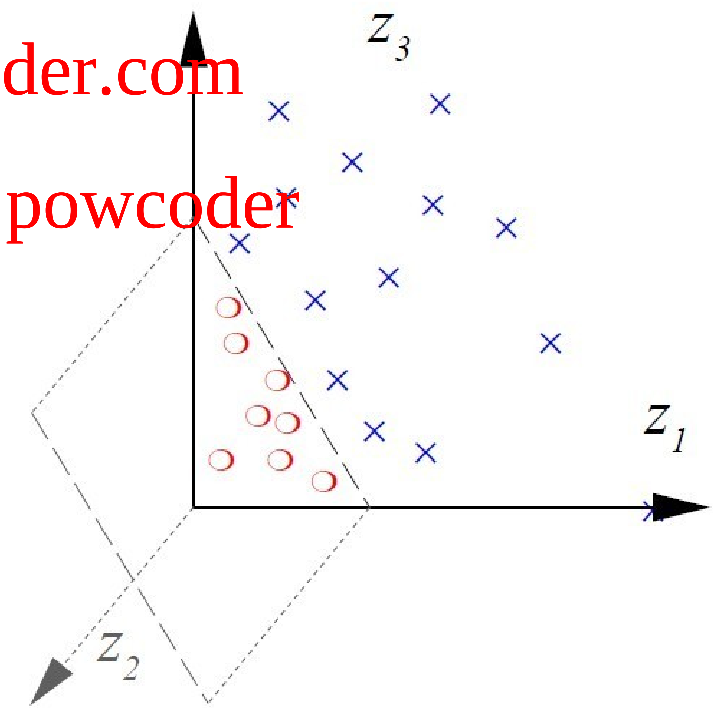


Example: Polynomial Mapping

$$\Phi : R^2 \rightarrow R^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$





Example: MNIST

- Data: 60,000 training examples, 10000 test examples, 28x28
- Linear SVM has around 8.5% test error. Polynomial SVM has around 1% test error.

MINST Results

Classifier	Test Error
linear	8.4%
3-nearest-neighbor	2.4%
RBF-SVM	1.4 %
Tangent distance	1.1 %
LeNet	1.1%
Boosted LeNet	0.7 %
Translation invariant SVM	0.56 %

Choosing a good mapping $\Phi(\cdot)$ (encoding prior knowledge + getting right complexity of function class) for your problem improves results.

SVMs: Kernel Trick

- The Representer theorem (Kimeldorf & Wahba, 1971) shows that (for SVMs as a special case):

$$w = \sum_{i=1}^{m} \alpha_i \Phi(x_i)$$

  for some variables α, instead of optimizing w directly, we can optimize α.
- The decision rule is: $f(x) = \sum_{i=1}^{m} \alpha_i \Phi(x_i) \cdot \Phi(x) + b$
  – We call $K(x_i, x) = \Phi(x_i) \cdot \Phi(x)$ the *kernel function*.

Kernels

- Why kernels?
  - Make non-separable problem separable.
  - Map data into better representational space
- Common used kernels
  - Linear
  - Polynomial $K(x_i, x_j) = (1 + x_i^T \cdot x_j)^d$
    - Gives feature conjunctions
  - Radial basis function

$$K(x_i, x_j) = e^{-\|x_i - x_j\|^2 / 2\sigma^2}$$

Outline

- Support Vector Machines
  - History
  - Linear Separable SVMs
  - Non-linear Separable SVMs
    - Soft Margin
    - Kernel Trick
- Parameter Estimation
- Further Reading

SVM: How to Estimate w, b

- We take the soft margin classification for example:

$$\min_{w} \frac{1}{2}\|w\|^2 + C\sum \xi_i$$

$$s.t. \quad y_i(w^T x_i + b) \ge 1 - \xi_i, \quad \xi_i \ge 0 \quad \forall i = 1, \dots, n$$

- Standard way: Use a solver!
  - Solver: software for finding solutions to “common” optimization problems, e.g. LIBSVM (http://www.csie.ntu.edu.tw/~cjlin/libsvm/)
- Problems: Solvers are inefficient for big data!

SVM: How to Estimate w, b

- Want to estimate w, b !

$$\min_w \frac{1}{2}\|w\|^2 + C\sum \xi_i$$

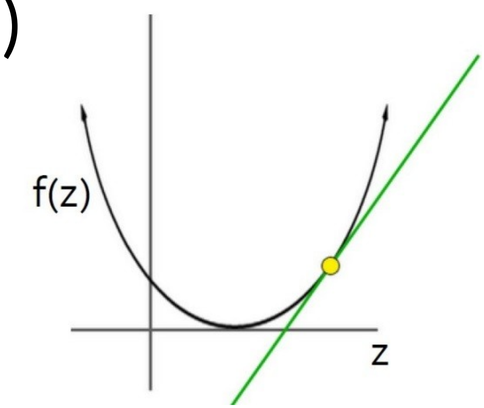
$$s.t. \forall i \; y_i(w^T x_i + b) \geq 1 - \xi_i, \; \xi_i \geq 0$$

- Alternative approach:

  – Want to minimize ***f(w,b)***

$$f(w,b) = \frac{1}{2}\sum_{j=1}^{d}(w^{(j)})^2 + C\sum_{i=1}^{n}\max\{0, 1 - y_i(\sum_{j=1}^{d} w^{(j)} x_i^{(j)} + b)\}$$

  – How to minimize convex functions f(z)
  – Use gradient descent: $\min_z f(z)$
  – Iterate: $z_{t+1} \leftarrow z_t - \eta f'(z_t)$



SVM: How to Estimate w?

- Want to minimize ***f(w,b)***:

$$f(w,b) = \frac{1}{2}\sum_{j=1}^{d}(w^{(j)})^2 + C\sum_{i=1}^{n}\underbrace{\max\{0, 1 - y_i(\sum_{j=1}^{d} w^{(j)} x_i^{(j)} + b)\}}_{\text{Empirical loss } L}$$

- Compute the gradient $\nabla(j)\ w.r.t\ w^{(j)}$

$$\nabla(j) = \frac{\partial f(w,b)}{\partial w^{(j)}} = w^{(j)} + C\sum_{i=1}^{n}\frac{\partial L(x_i, y_j)}{\partial w^{(j)}}$$

$$\frac{\partial L(x_i, y_j)}{\partial w^{(j)}} = \begin{cases} 0 & \text{if } y_i(w \cdot x_i + b) \geq 1 \\ -y_i x_i^{(j)} & \text{otherwise} \end{cases}$$

SVM: How to Estimate w?

- Gradient descent:

  Iterate untial convergence:

  - For $j = 1, \ldots, d$

    – Evaluate: $\nabla(j) = \frac{\partial f(w,b)}{\partial w^{(j)}} = w^{(j)} + C \sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$

    – Update: $w^{(j)} = w^{(j)} - \eta \nabla(j)$

  $\eta \ldots$ learning rate parameter
  $C \ldots$ regularization parameter

- Problem:
  – Computing $\nabla(j)$ takes O(n) time
    - n … size of the training dataset

SVM: How to Estimate w?

We just had:
$\nabla(j) = w^{(j)} + C\sum_{i=1}^{n} \frac{\partial L(x_i,y_i)}{\partial w^{(j)}}$

- Stochastic Gradient Descent
  - Instead of evaluating gradient over all examples, evaluate it for each individual training example

$$\nabla(j,i) = w^{(j)} + C\frac{\partial L(x_i,y_i)}{\partial w^{(j)}}$$

- Stochastic gradient descent (SGD):

Iterate untial convergence:

- For $i = 1, \ldots, n$
  - For $j = 1, \ldots, d$
    - Evaluate: $\nabla(j,i)$
    - Upadate: $w^{(j)} \leftarrow w^{(j)} - \eta\nabla(j,i)$

Example: Text Categorization

- Example by Leon Bottou:
  - **Reuters RCV1** document corpus
    - Predict a category of a document
      - One **vs.** the rest classification
  - ***n = 781,000*** training examples (documents)
  - 23,000 test examples
  - **d = 50, 000** features
    - One feature per word
    - Remove stop-words
    - Remove low frequency words

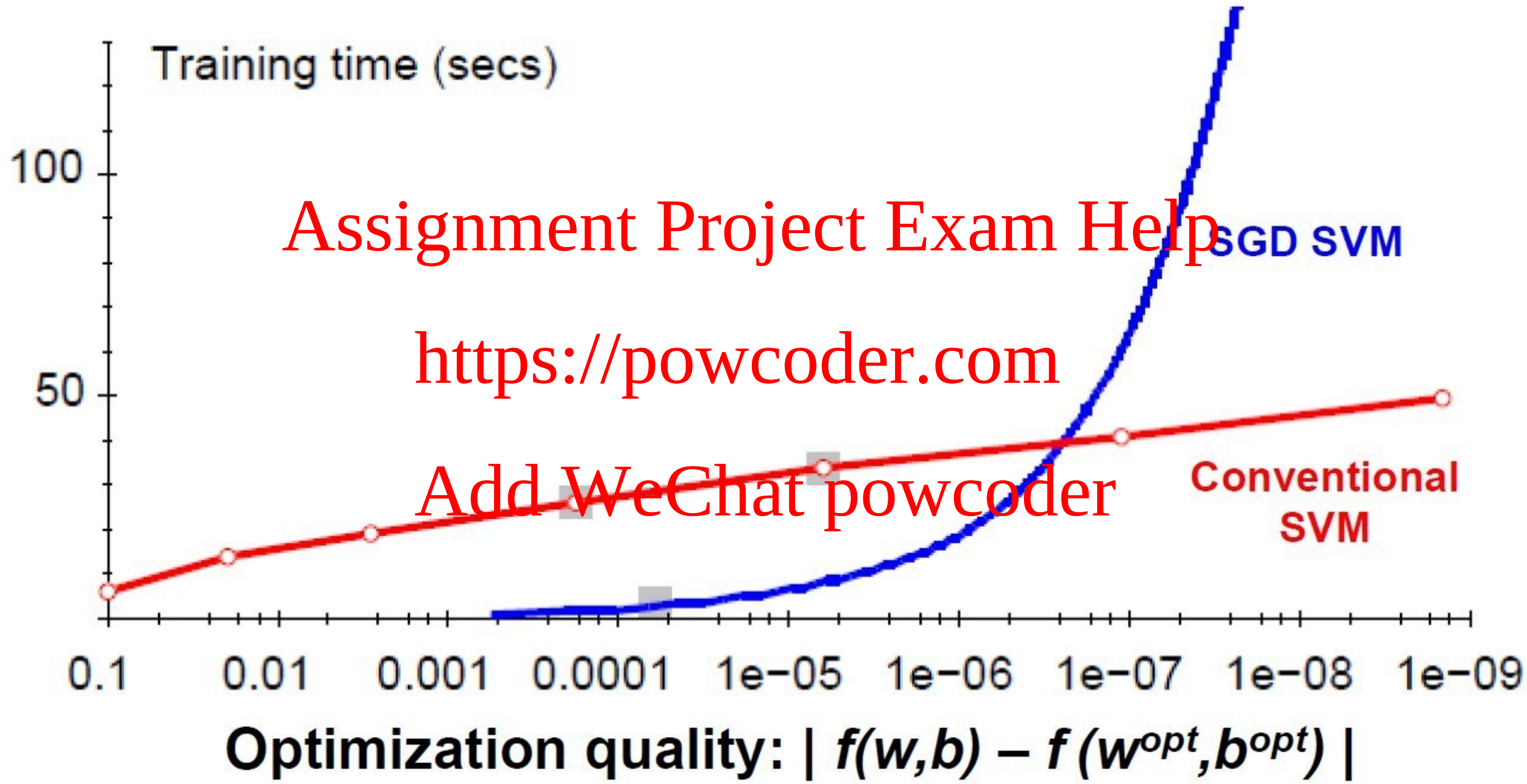
Examples: Text Categorization

- Questions:
  - Is SGD successful at minimizing ***f(w,b)***?
  - How quickly does SGD find the min of ***f(w,b)***?
  - What is the error on a test set?

	Training time	*Value of f(w,b)*	*Test error*
Standard SVM	23,642 secs	0.2275	6.02%
"Fast SVM"	66 secs	0.2278	6.03%
SGD SVM	1.4 secs	0.2275	6.02%

- SGD-SVM is successful at minimizing the value of ***f(w,b)***
- SGD-SVM is super fast
- SGD-SVM test set error is comparable

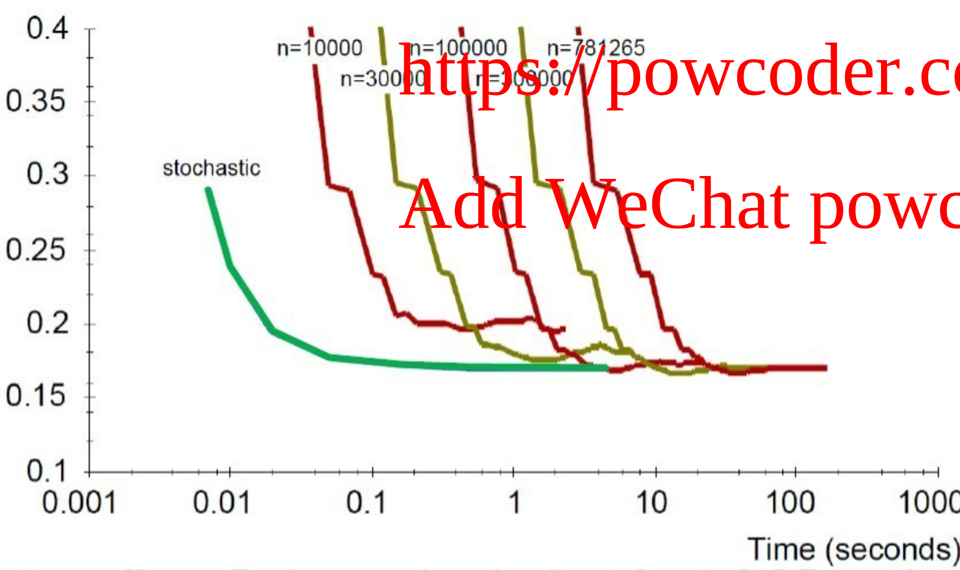
Optimization “Accuracy”



For optimizing *f(w,b) within reasonable* quality
SGD-SVM is super fast

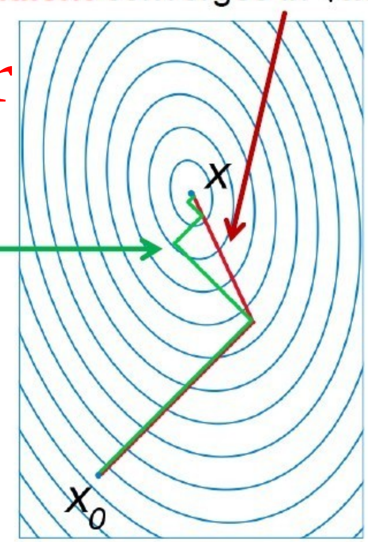
SGD vs. Batch Conjugate Gradient

- SGD on full dataset vs. Batch Conjugate
  - Gradient on a sample of n training examples



Bottom line: Doing a simple (but fast) SGD update many times is better than doing a complicated (but slow) BCG update a few times

Theory says: Gradient descent converges in linear time k. **Conjugate gradient** converges in $\sqrt{k}$.



k… condition number

Practical Considerations

- Need to choose learning rate η and t_0

$$w_{t+1} \leftarrow w_t - \frac{\eta_t}{t + t_0}\left(w_t + C\frac{\partial L(x_i, y_i)}{\partial w}\right)$$

- Leon suggests:

  - Choose t_0 so that the expected initial updates are comparable with the expected size of the weights

  - Choose η:
    - Select a small subsample
    - Try various rates η (e.g., 10,1,0.1,0.01,...)
    - Pick the one that most reduces the cost
    - Use η for next 100k iterations on the full dataset

Practical Considerations

- Sparse Linear SVM:
  - Feature vector x_i is sparse (contains many zeros)
    - Do not do: $x_i = [0, 0, 0, 1, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, \ldots]$
    - But represent x_i as a sparse vector $\mathbf{x_i} = [(\mathbf{4}, \mathbf{1}), (\mathbf{9}, \mathbf{5}), \ldots]$
  - Can we do the SGD update more efficiently?

$$w \leftarrow w - \eta \left(w + C \frac{\partial L(x_i, y_i)}{\partial w} \right)$$

  - Approximated in 2 steps:

$$w \leftarrow w - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$$

$$w \leftarrow w(1 - \eta)$$

Cheap: $\mathbf{x_i}$ is sparse and so few coordinates **j** of **w** will be updates

Expensive: **w** is not sparse, all coordinates need to be updated

Practical Considerations

- Solution 1: $\mathbf{w} = \mathbf{s} \cdot \mathbf{v}$
  - Represent vector **w** as the product of scalar **s** and the vector **v**
  - Then the update procedure is:
    - 1) $v = v - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$
    - 2) $s = s(1 - \eta)$
- Solution 2:
  - Perform only step 1) for each training example
  - Perform step 2) with lower frequency and higher η

Two step update procedure:

1. $w \leftarrow w - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$
2. $w \leftarrow w(1 - \eta)$

Practical Considerations

- Stopping criteria:

  How many iterations of SGD?

  – Early stopping with cross validation

    - Create validation set
    - Monitor cost function on the validation set

    - Stop when loss stops decreasing

Practical Considerations

- Stopping criteria:

  How many iterations of SGD?

  – Early Stopping

    - Extract two disjoint subsamples **A** and **B** of training data
    - Train on **A**, stop by validating on **B**

    - Number of epochs is an estimate of **k**
    - Train for **k** epochs on the full dataset

What about Multiple Classes?

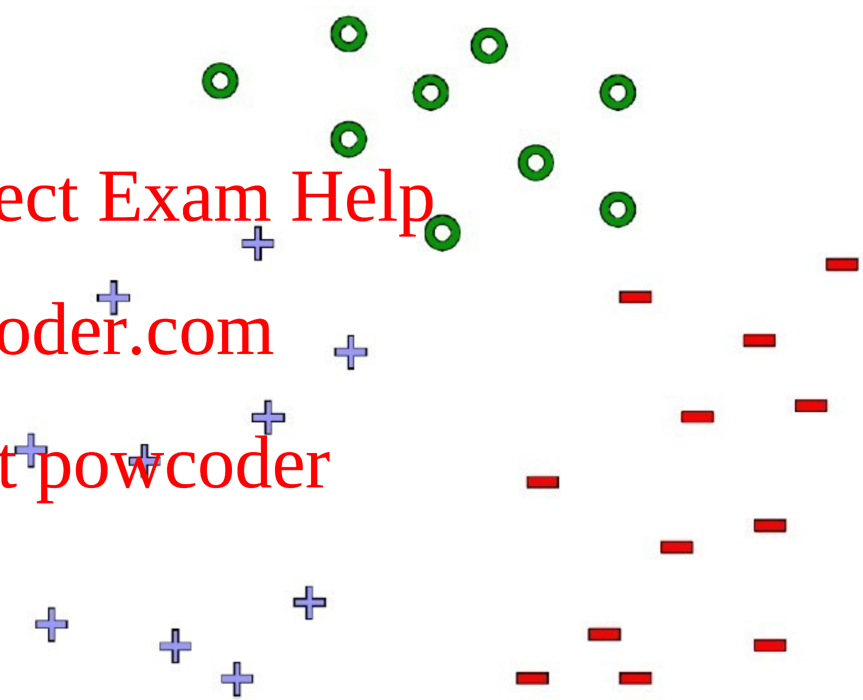
- Idea 1:
  - One against all

  Learn 3 classifiers
    - + vs. {o,-}
    - - vs. {o,+}
    - o vs. {+,-}

  Obtain: $w_+ b_+, w_- b_-, w_o b_o$

  - Return class *c*

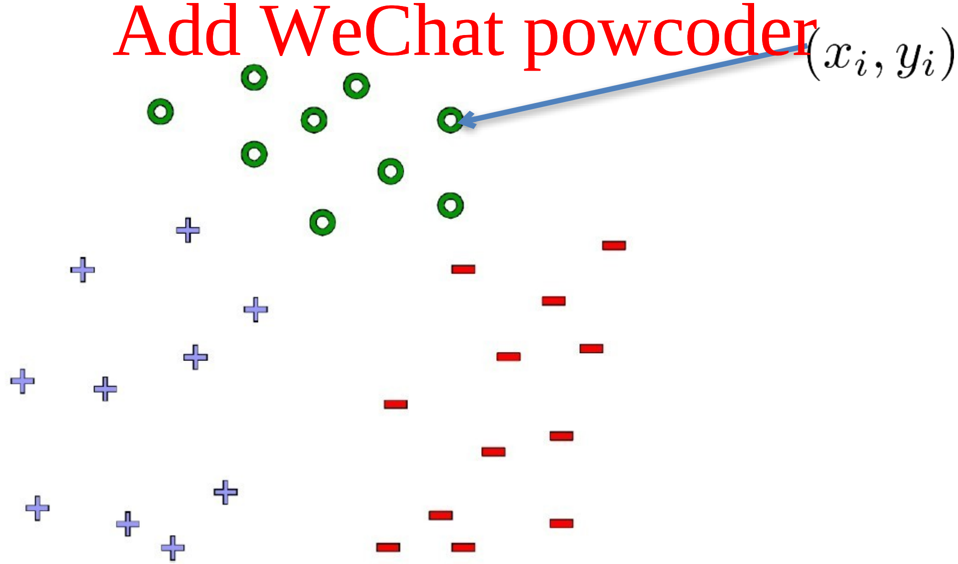
$$\arg\max_c w_c x + b_c$$



What about Multiple Classes?

- Idea 2:
  - Learn 3 sets of weights simultaneously
  - Want the correct class to have highest margin:

$$w_{y_i} x_i + b_{y_i} \geq 1 + w_c x_i + b_c \; \forall c \neq y_i, \forall i$$



Multiclass SVM

- Optimization problem:

$$\min_{w,b} \frac{1}{2}\sum_c \|w_c\|^2 + C\sum_{i=1}^n \xi_i$$

$$w_{y_i} x_i + b_{y_i} \geq w_c x_i + b_c + 1 - \xi_i \; \forall c \neq y_i, \xi_i \geq 0, \forall i$$

  – To obtain parameters w_c, b_c for each class c, we can use similar techniques as for 2 class SVM

- SVM is widely perceived a very powerful learning algorithm

Demo

Libsvm package for R:

http://cran.r-project.org/web/packages/e1071/index.html

Demo

```
> # load library, class, a dependence for the SVM library
> library(class)
> # load library, SVM
> library(e1071)
```



```
> # load library, mlbench, a collection of some datasets from the UCI repository
> library(mlbench)
```



```
> # load data, has 7 classes, details of data:
    http://archive.ics.uci.edu/ml/datasets/Glass+Identification
```



```
> data(Glass, package = "mlbench")
> # get the index of all data
> index <- 1:nrow(Glass)
> # generate test index
> testindex <- sample(index, trunc(length(index)/3))
> # generate test set
> testset <- Glass[testindex, ]
> # generate trainin set
> trainset <- Glass[-testindex, ]
```

Demo

```
> # train svm on the training set
> # cost=100: the penalizing parameter for C-classification
> # gamma=1: the radial basis function-specific kernel parameter
> # Output values include SV, index, coefs, rho, sigma, probA, probB
> svm.model <- svm(Type~ ., data = trainset, cost = 100, gamma = 1)
> # a vector of predicted values,
> # for classification: a vector of labels
> svm.pred <- predict(svm.model, testset[, -10])
> # a cross-tabulation of the true
> # versus the predicted values
> table(pred = svm.pred, true = testset[, 10])
```

One-slide Takeaway

- SVM:
  - Linear Separable SVMs
  - Non-linear Separable SVMs: Soft Margin and Kernel Trick

- Parameter Estimation:
  - Solver: e.g. libsvm, not efficient
  - Stochastic gradient descent

Outline

- Support Vector Machines
  - History
  - Linear Separable SVMs
  - Non-linear Separable SVMs
    - Soft Margin
    - Kernel Trick
- Parameter Estimation
- Further Reading

Further Reading

- Early paper about SVM algorithm: http://link.springer.com/content/pdf/10.1007%2FBF00994018.pdf

- More kernel techniques:
  - Schölkopf, Bernhard; Burges, Christopher J. C.; and Smola, Alexander J. (editors); *Advances in Kernel Methods: Support Vector Learning*, MIT Press, Cambridge, MA, 1999. ISBN 0-262-19416-3.

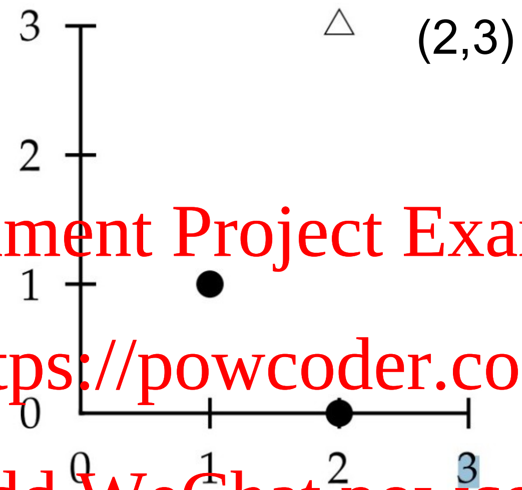
Further Reading

- More efficient learning algorithm for SVM:
  – Parallelizing Support Vector Machines on Distributed Computers: https://code.google.com/p/psvm/

Reference

- http://www.stanford.edu/class/cs246/slides/13-svm.pdf
- http://www.stanford.edu/class/cs276/handouts/lecture14-SVMs.ppt
- http://i.stanford.edu/~ullman/pub/ch12.pdf
- http://www.svms.org/tutorials/
- http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf
- http://www.csie.ntu.edu.tw/~cjlin/libsvm/
- Chang, E, Zhu, K, Wang, H, Bai, H, Li, J, Qiu, Z, and Cui, H. PSVM: Parallelizing support vector machines on distributed computers. NIPS, 20:257-264. 2007.

In-class Practice



- Consider building an SVM over the (very little) data set shown in above figure, compute the each SVM decision boundary.