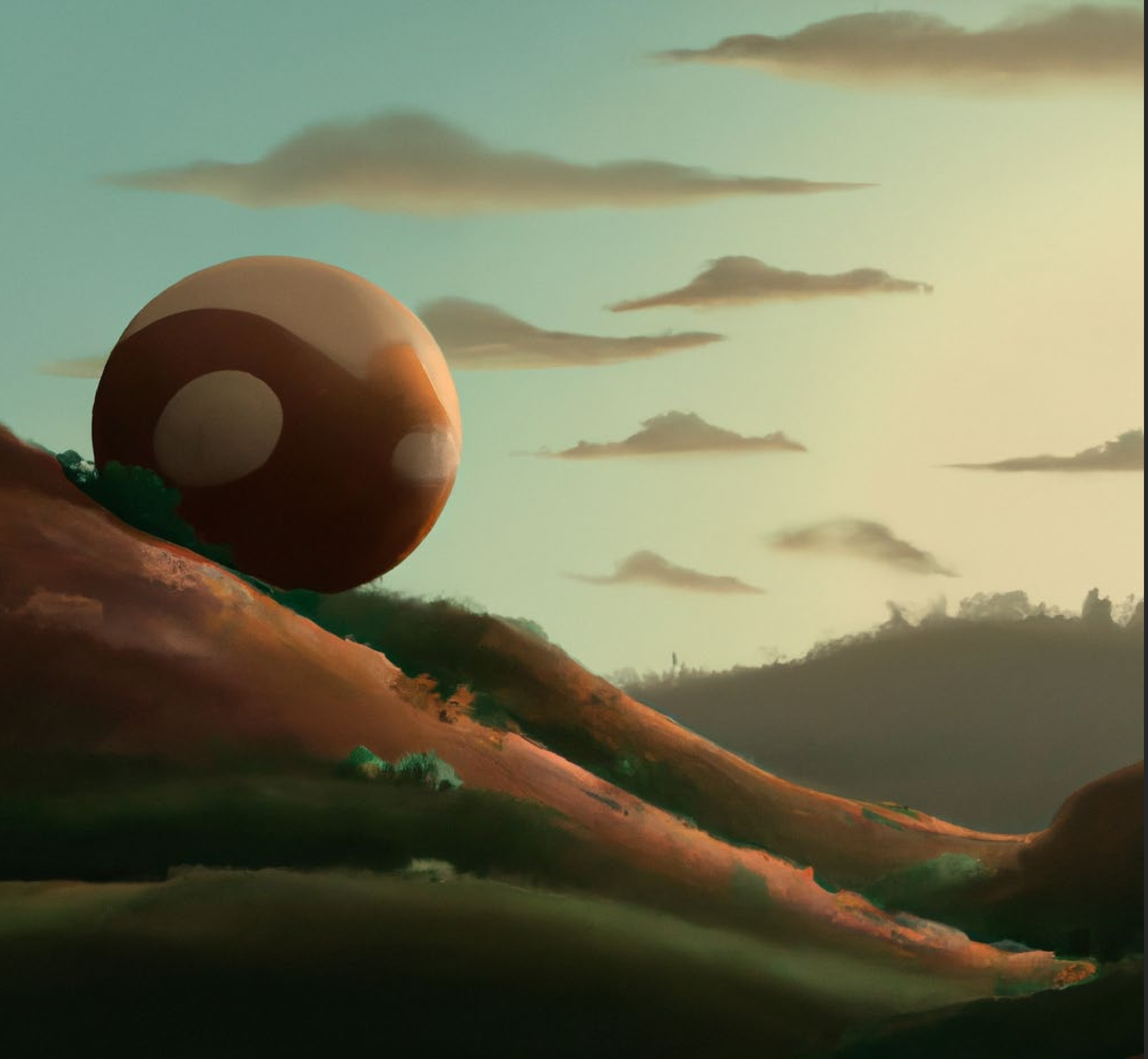


Dall-E

SVMs and SGD

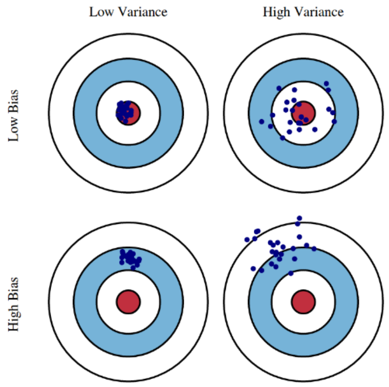
Applied Machine Learning
Derek Hoiem

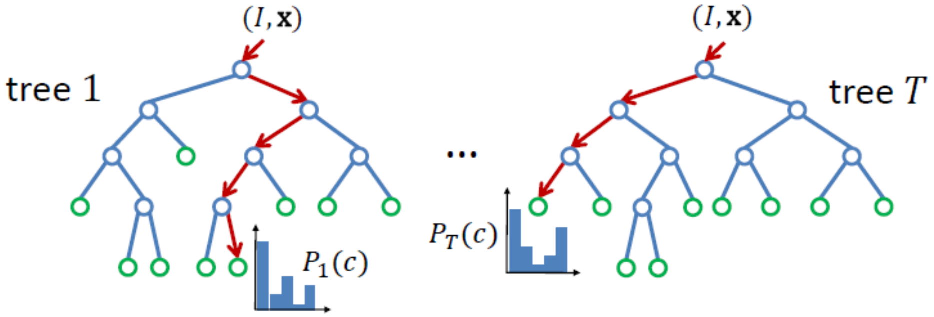
Previously, we learned…

- Ensembles improve accuracy by reducing bias and/or variance

- Boosted trees and random forests are powerful and widely applicable ensemble methods

$$\underbrace{E_{\mathbf{x},y,D}\left[\left(h_D(\mathbf{x}) - y\right)^2\right]}_{\text{Expected Test Error}} = \underbrace{E_{\mathbf{x},D}\left[\left(h_D(\mathbf{x}) - \bar{h}(\mathbf{x})\right)^2\right]}_{\text{Variance}} + \underbrace{E_{\mathbf{x},y}\left[\left(\bar{y}(\mathbf{x}) - y\right)^2\right]}_{\text{Noise}} + \underbrace{E_{\mathbf{x}}\left[\left(\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x})\right)^2\right]}_{\text{Bias}^2}$$





Support Vector Machines (SVMs)

- Developed in the 1990’s by Vapnik and colleagues at Bell Labs based on statistical learning theory
- One of the most popular learning techniques until deep learning resurgence
- What is interesting about SVMs
  - **Generalization properties**, including achieving a margin and structural risk minimization
  - Extension to non-linear classifier via **kernels**
  - Dual form that shows how **linear classifiers can be seen as a weighted average of training examples**
  - Optimization via **stochastic gradient descent**, also used for neural networks

This lecture

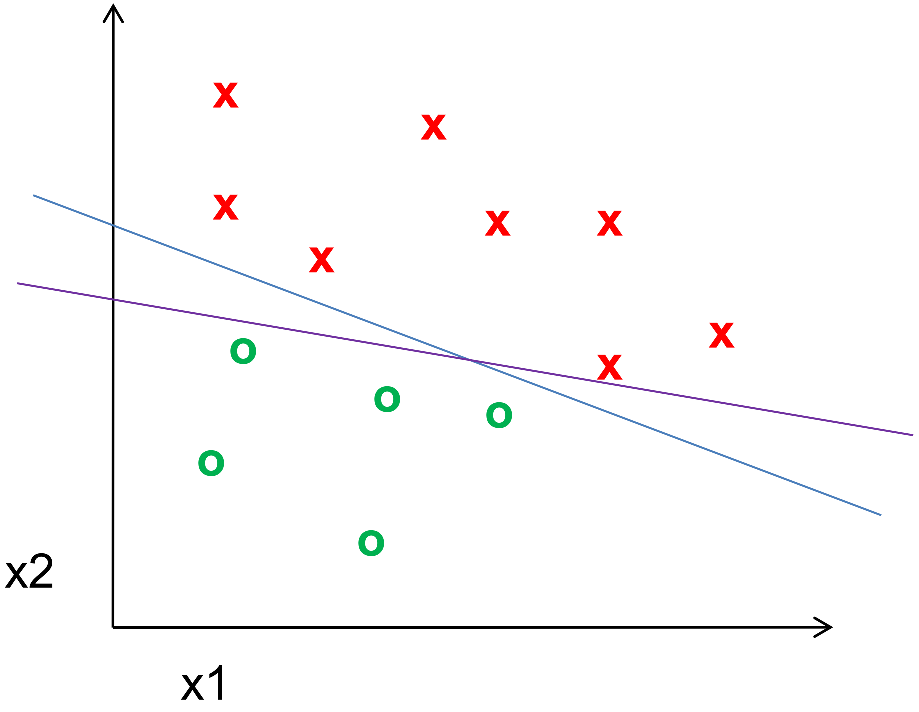
1. Linear SVM

2. Kernels and Non-Linear SVM

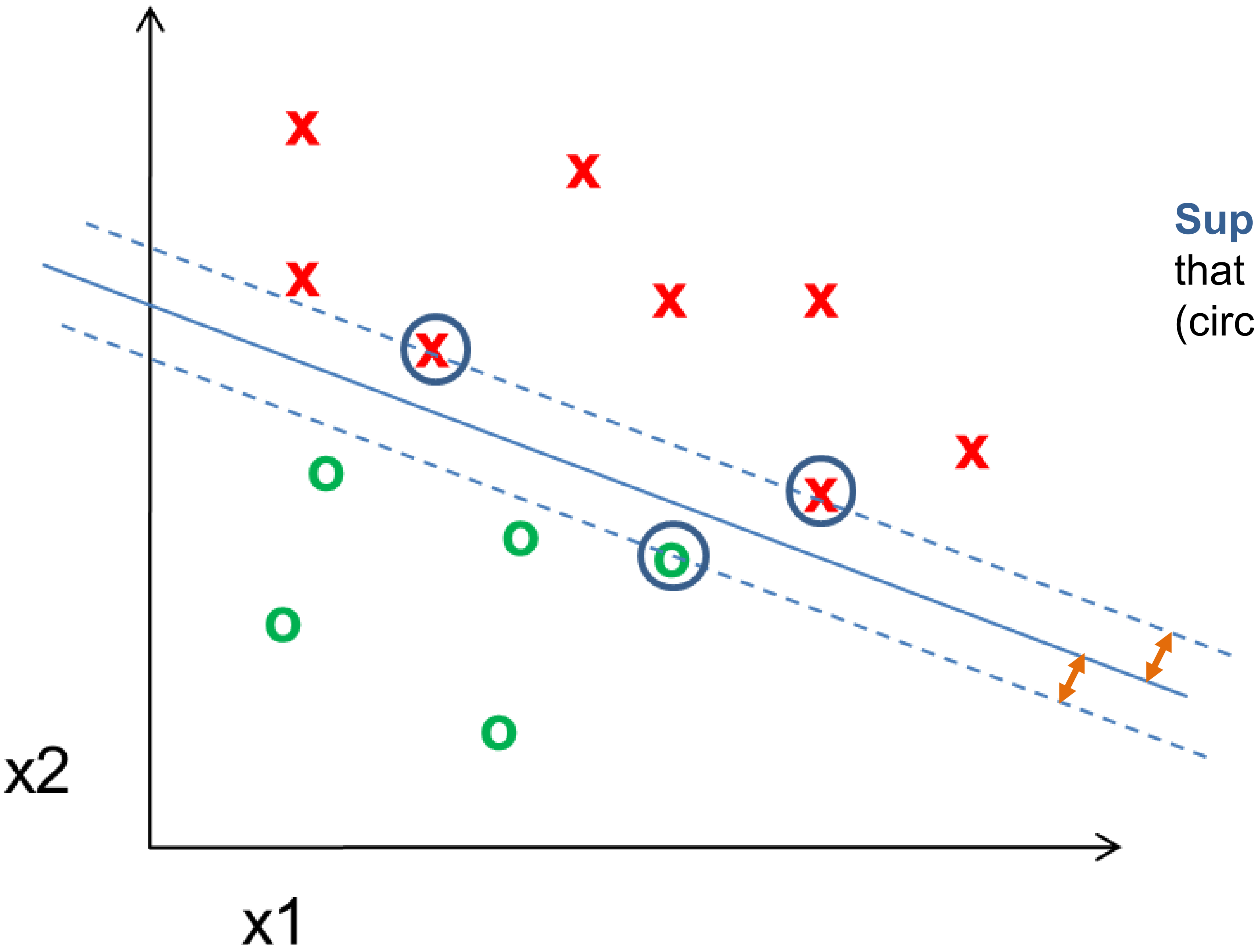
3. SVM Optimization with Stochastic Gradient Descent

What is the best linear classifier?

- Logistic regression
  - Maximize expected likelihood of label given data
  - Every example contributes to loss
- SVM
  - Make all examples at least minimally confident
  - Base decision on a minimal set of examples



SVM Terminology



Support Vector: an example that lies on the margin (circled points)

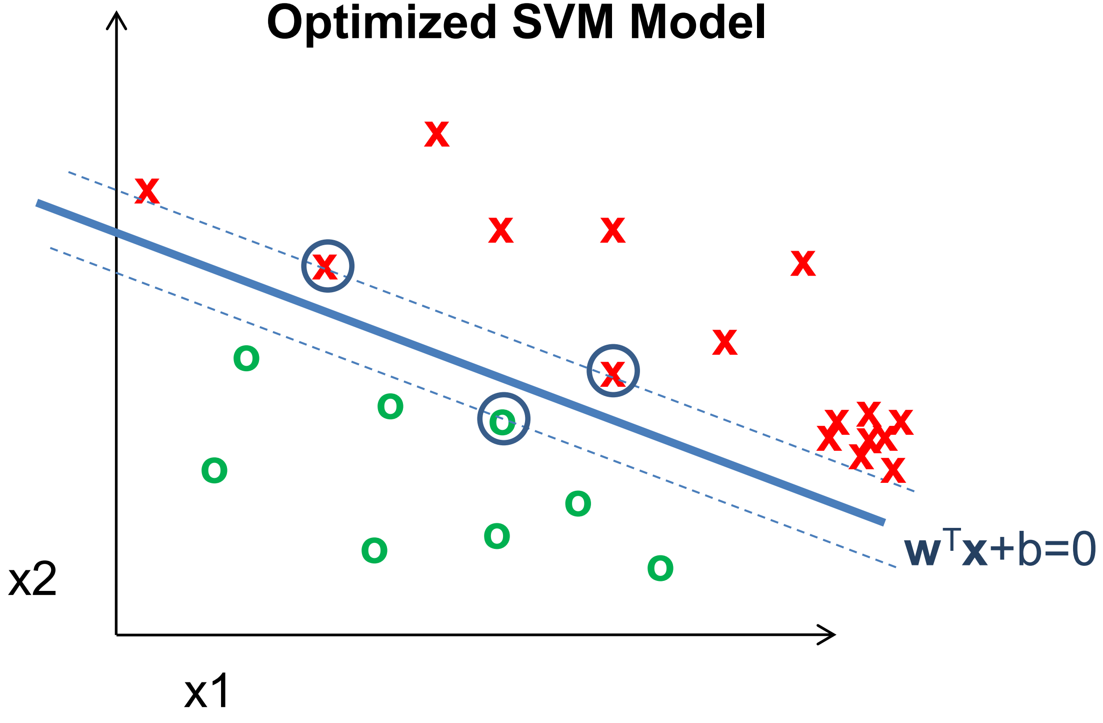
Margin: the distance of examples (in feature space) from the decision boundary

$$m(\boldsymbol{x}) = \frac{y(\boldsymbol{w}^T\boldsymbol{x}+b)}{\|w\|}$$

$y \in \{-1,1\}$

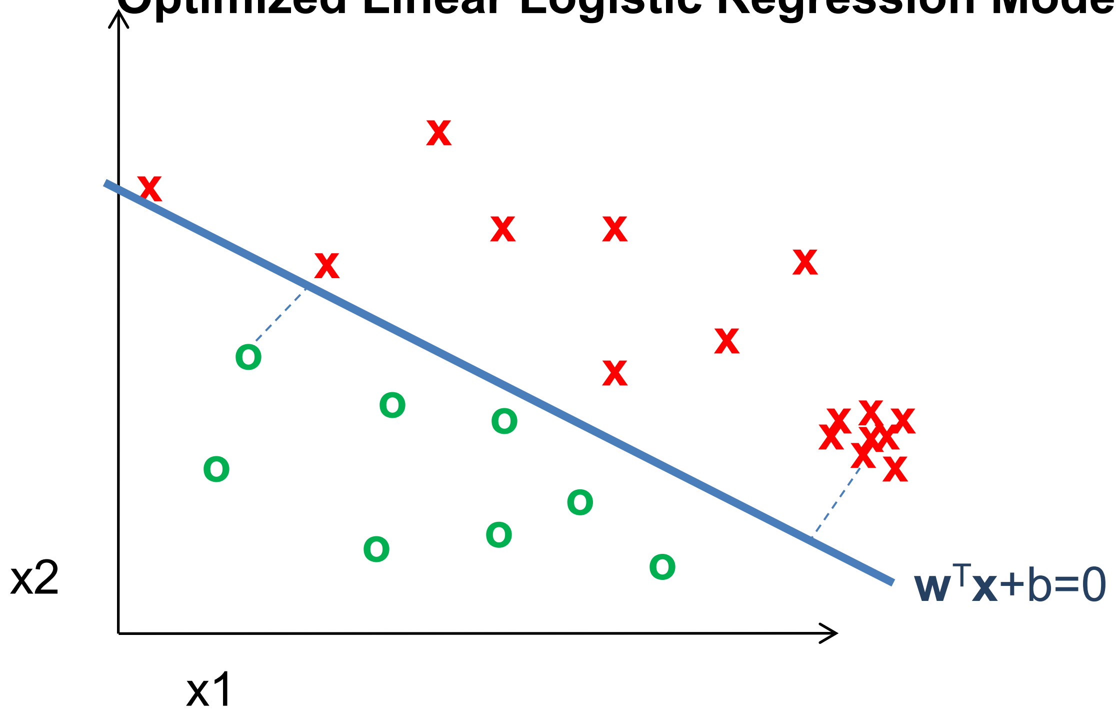
SVMs minimize $\boldsymbol{w}^T\boldsymbol{w}$ while preserving a margin of 1

Optimized SVM Model



Decision boundary depends only on “support vectors” (circled)

Optimized Linear Logistic Regression Model

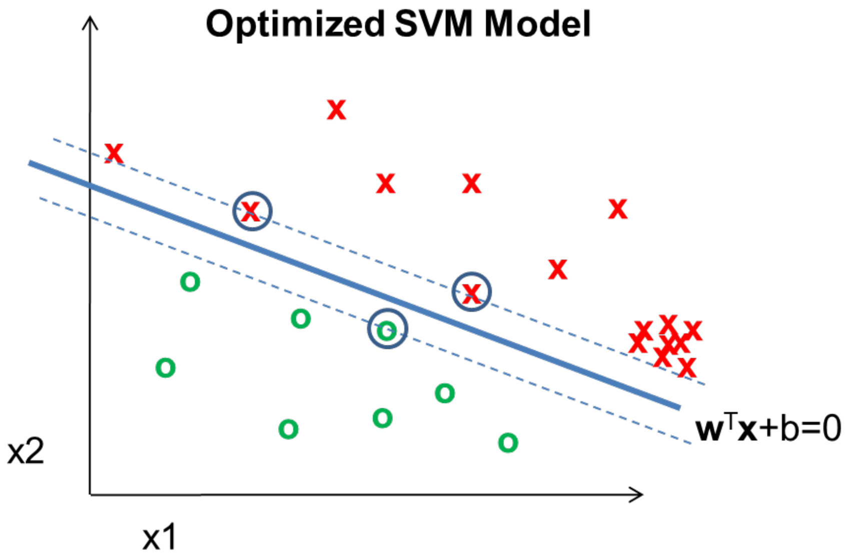


Minimizes the sum of logistic error on all samples, so boundary should be further from dense regions

Why SVMs achieve good generalization

- Maximizing the margin – if all examples are far from the boundary, it is less likely that some test sample will end up on the wrong side of the boundary
  - If classes are linearly separable, the scores can be arbitrarily increased by scaling **w**, so optimization is expressed as minimize $\boldsymbol{w}^T\boldsymbol{w}$ while preserving a margin of 1
- Dependence on few training samples – most training data points could be removed without affecting the decision boundary, which gives an upper bound on the generalization error
- E.g., expected test error is <= than the smaller of:
  a. % of training samples that are support vectors
  b. $D^2/m^2/N$, the diameter of the data compared to the margin divided by the number of examples

(see proof)



SVM in Linearly Separable Case

Prediction

$$y_n = \operatorname{sign}(\boldsymbol{w}^T x_n + b)$$

Optimization

$$w^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} ||w||^2$$

subject to

$$y_n(\boldsymbol{w}^T x_n + b) \geq 1 \text{ for all } n$$

Here, $y \in \{-1,1\}$ which is a common convention that simplifies notation for binary classifiers

SVM in **Non**-Linearly Separable Case

Prediction

$$y_n = \operatorname{sign}(\boldsymbol{w}^T x_n + b)$$

Optimization

Known as “hinge loss”
Penalty is paid if margin is less than 1

$$w^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left[\|w\|^2 + C \sum_{n}^{N} \max(0, 1 - y_n(\boldsymbol{w}^T x_n + b)) \right]$$

Here, $y \in \{-1,1\}$ which is a common convention that simplifies notation for binary classifiers

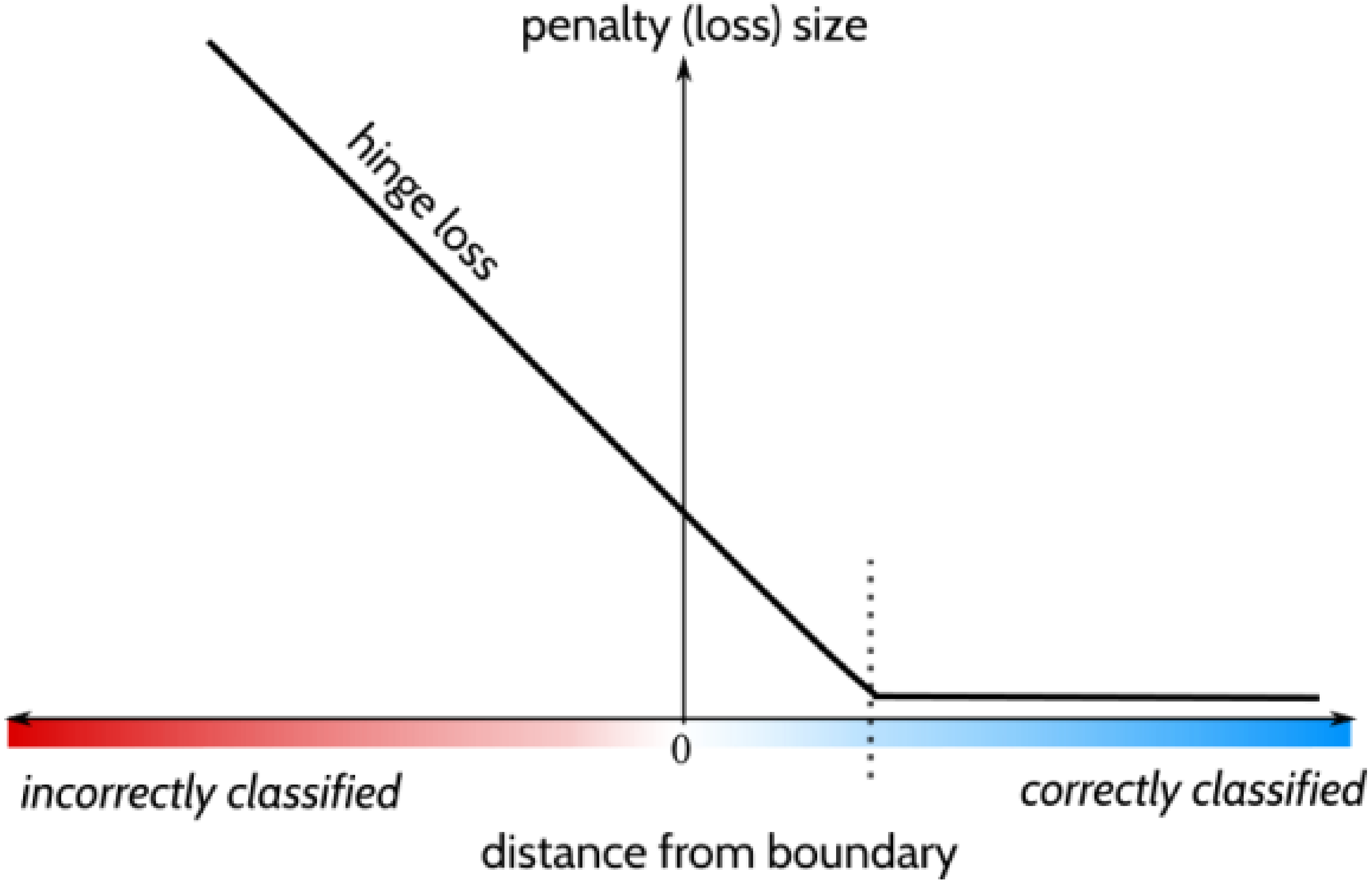
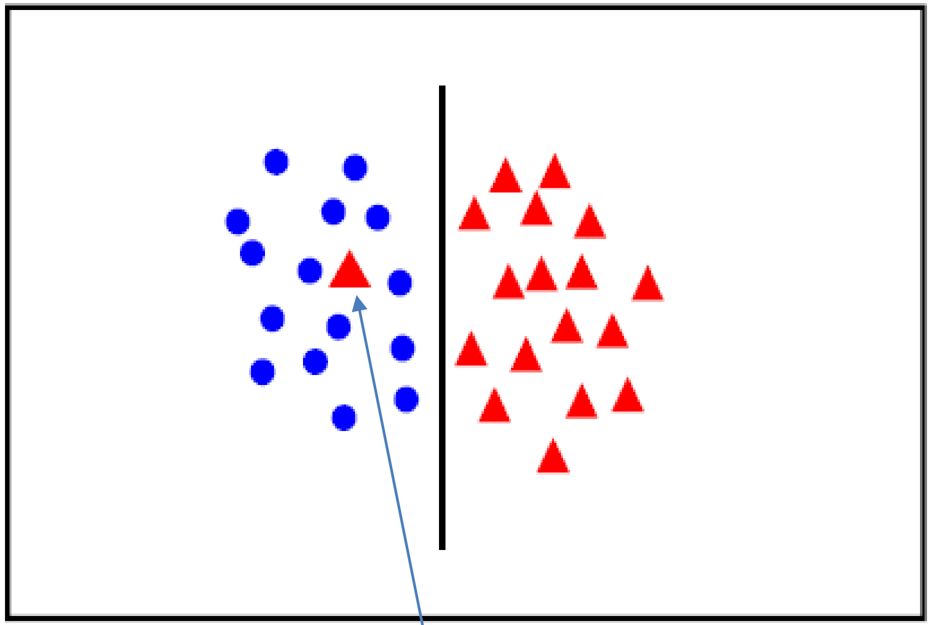

penalty (loss) size
hinge loss
0
incorrectly classified
correctly classified
distance from boundary

Fig source

Sometimes non-linear optimization is written in terms of “slack variables”

$$w^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left[\|w\|^2 + C \sum_{n}^{N} \max(0, 1 - y_n(\boldsymbol{w}^T x_n + b)) \right]$$



Pay slack penalty

is equivalent to

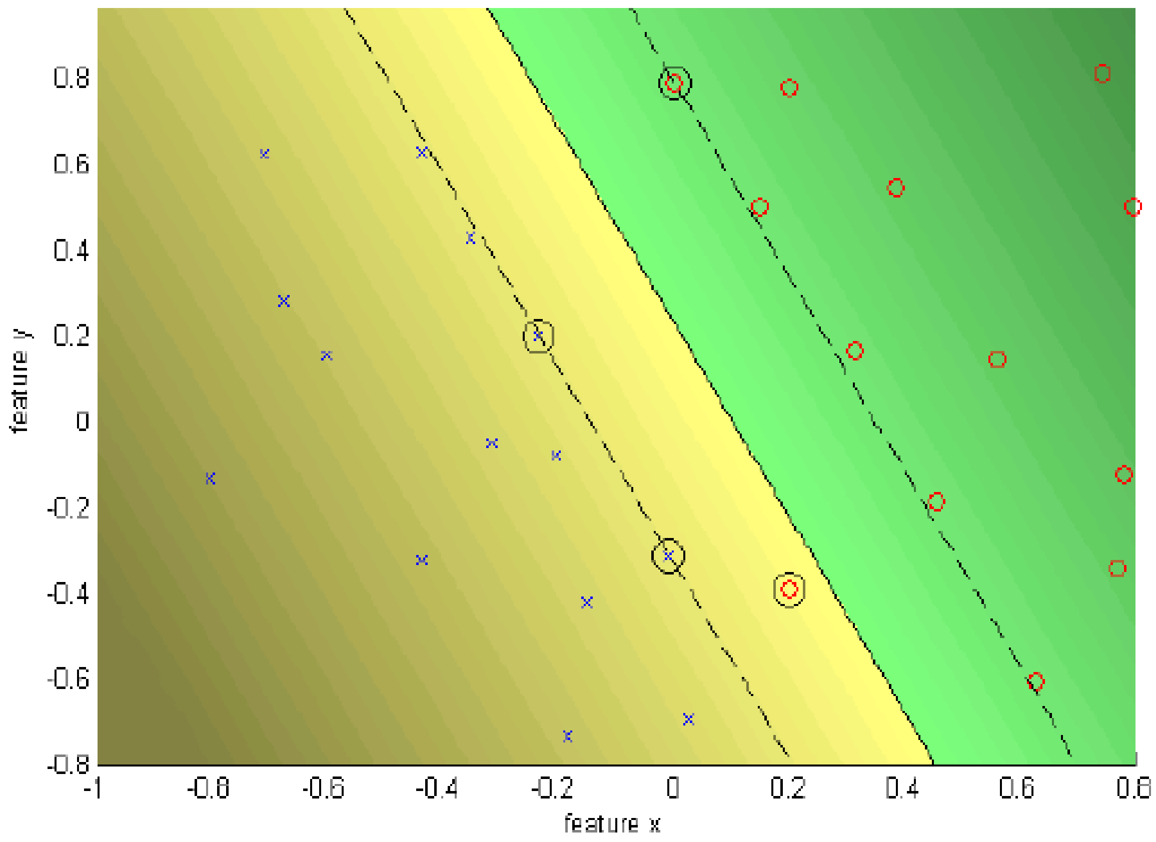
$$\min_{\mathbf{w} \in \mathbb{R}^d, \xi_i \in \mathbb{R}^+} \|\mathbf{w}\|^2 + C \sum_{i}^{N} \xi_i$$

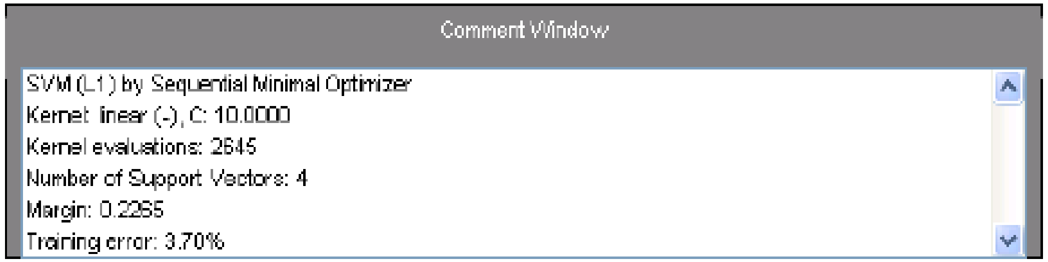
slack variables

subject to

$$y_i \left(\mathbf{w}^\top \mathbf{x}_i + b \right) \geq 1 - \xi_i \text{ for } i = 1 \ldots N$$

C = 10   soft margin





```
SVM (L1) by Sequential Minimal Optimizer
Kernel: linear (-), C: 10.0000
Kernel evaluations: 2645
Number of Support Vectors: 4
Margin: 0.2265
Training error: 3.70%
```

Slide credit: Zisserman [link]

Representer theorem

Optimal weights for many L2-regularized classification and regression functions can be expressed as a weighted combination of training examples

$$\boldsymbol{w}^* = \sum_n \alpha_n y_n \boldsymbol{x}_n$$

$$\alpha_n \geq 0, y_n \in \{-1,1\}$$

Conditions apply, e.g. function must be regularized in a Reproducing Kernel Hilbert Space (details)

Does *not* apply to L1 weight regularization because that can’t be expressed as a dot product of weights

Primal vs. Dual Formulations of SVM

Prediction **Training Objective**

Primal

$f(x) = \boldsymbol{w}^T \boldsymbol{x} + b$

$$w^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left[\|w\|^2 + C \sum_{n}^{N} \max(0, 1 - y_n(\boldsymbol{w}^T x_n + b)) \right]$$

Dual

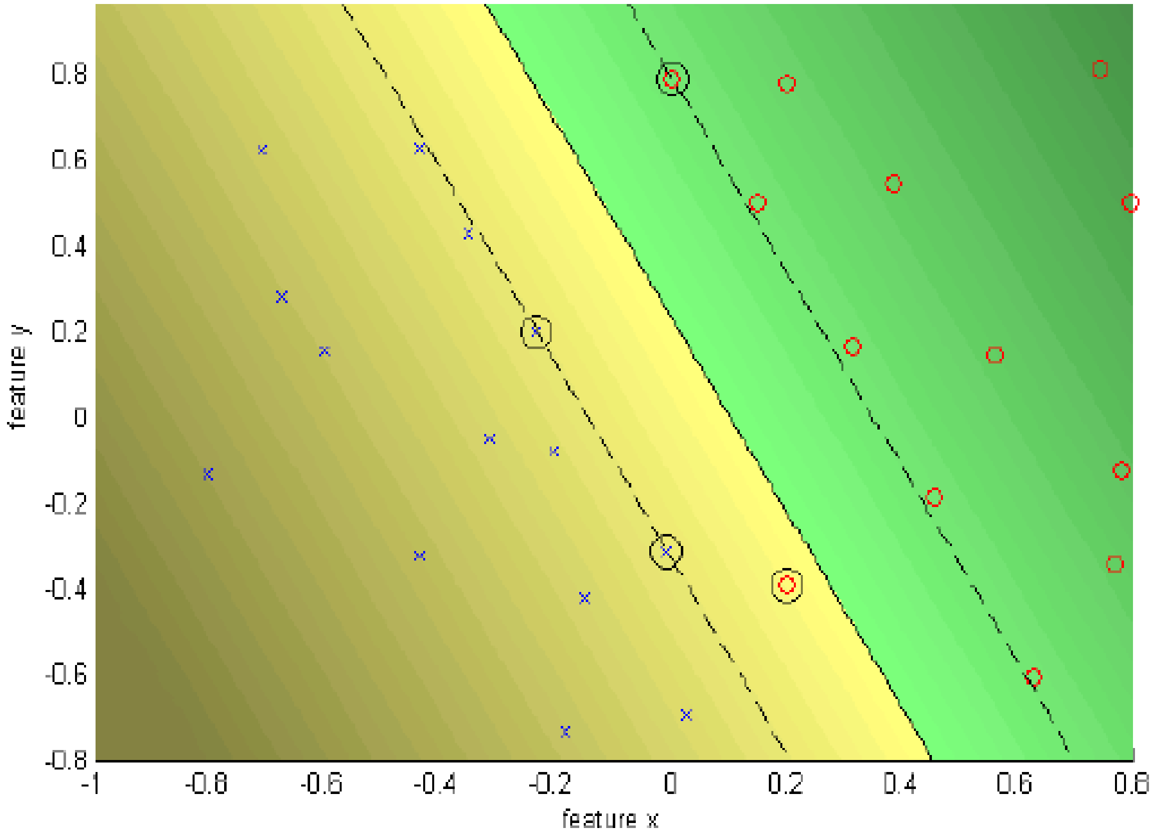
$f(\boldsymbol{x}) = \sum_n \alpha_n y_n (\boldsymbol{x}_n^T \boldsymbol{x}) + b$

$$\boldsymbol{\alpha}^* = \underset{\boldsymbol{\alpha}}{\operatorname{argmax}} \left[\sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k (\boldsymbol{x}_j^T \boldsymbol{x}_k) \right]$$

$$\text{s.t.} \;\; 0 \le \alpha_i \le C \; \forall i \quad \text{and} \quad \sum_i \alpha_i \, y_i = 0$$

Primal: parameter for each feature
Dual: parameter for each training example

For SVM, α is sparse (most values are zero)

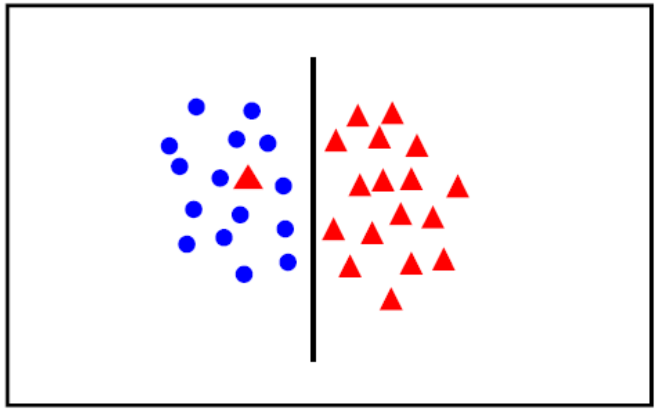


In dual, $\alpha_i > 0$ only for support vectors

$\alpha_i = 0$ for all others

Slide credit: Zisserman [link]

But what if the decision boundary is not even close to linear?

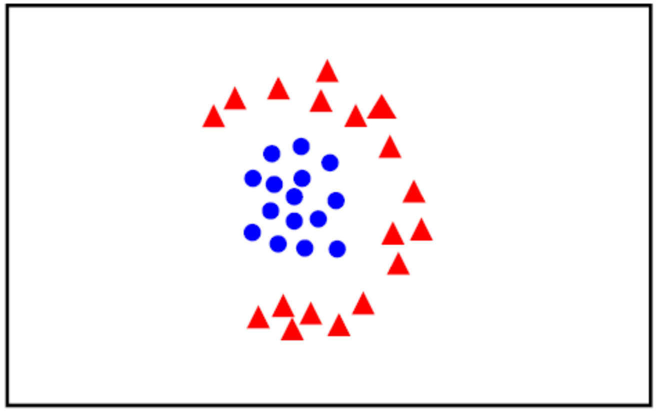


- introduce slack variables

$$\min_{\mathbf{w}\in\mathbb{R}^d,\xi_i\in\mathbb{R}^+} ||\mathbf{w}||^2 + C\sum_i^N \xi_i$$

subject to

$$y_i\left(\mathbf{w}^\top\mathbf{x}_i + b\right) \geq 1 - \xi_i \text{ for } i = 1\ldots N$$

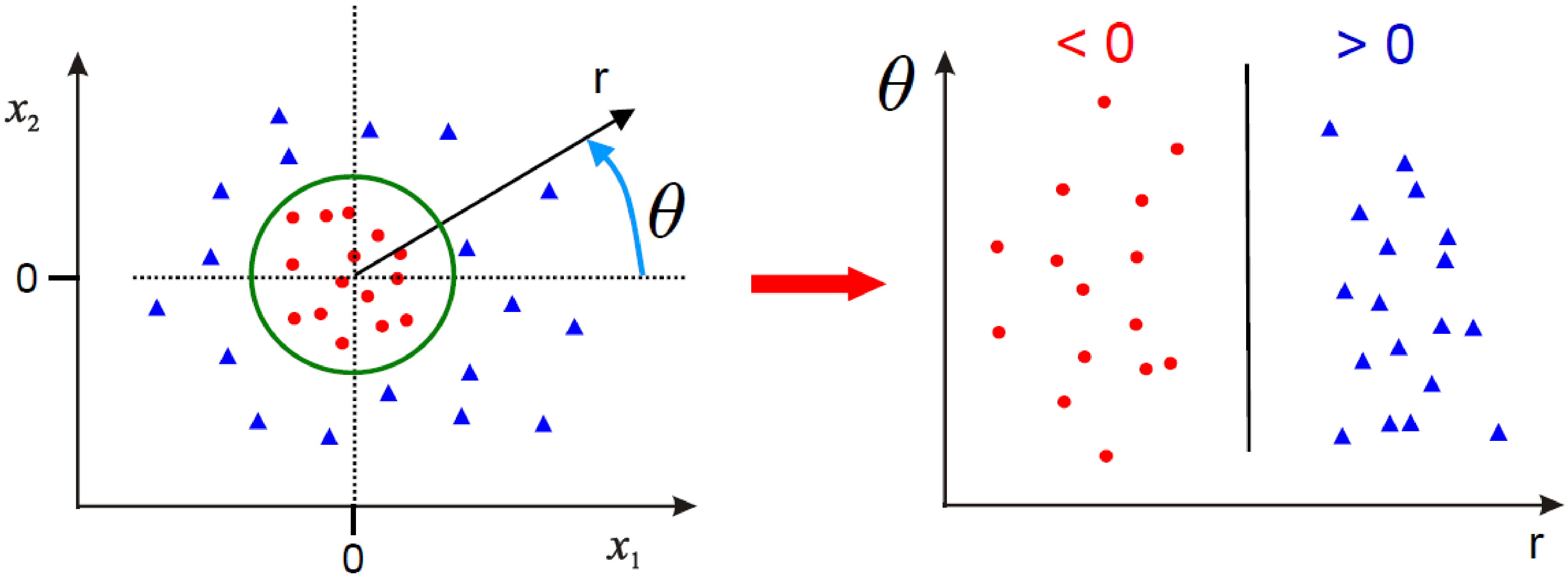


- linear classifier not appropriate

??

Slide credit: Zisserman [link]

Solution 1: use polar coordinates



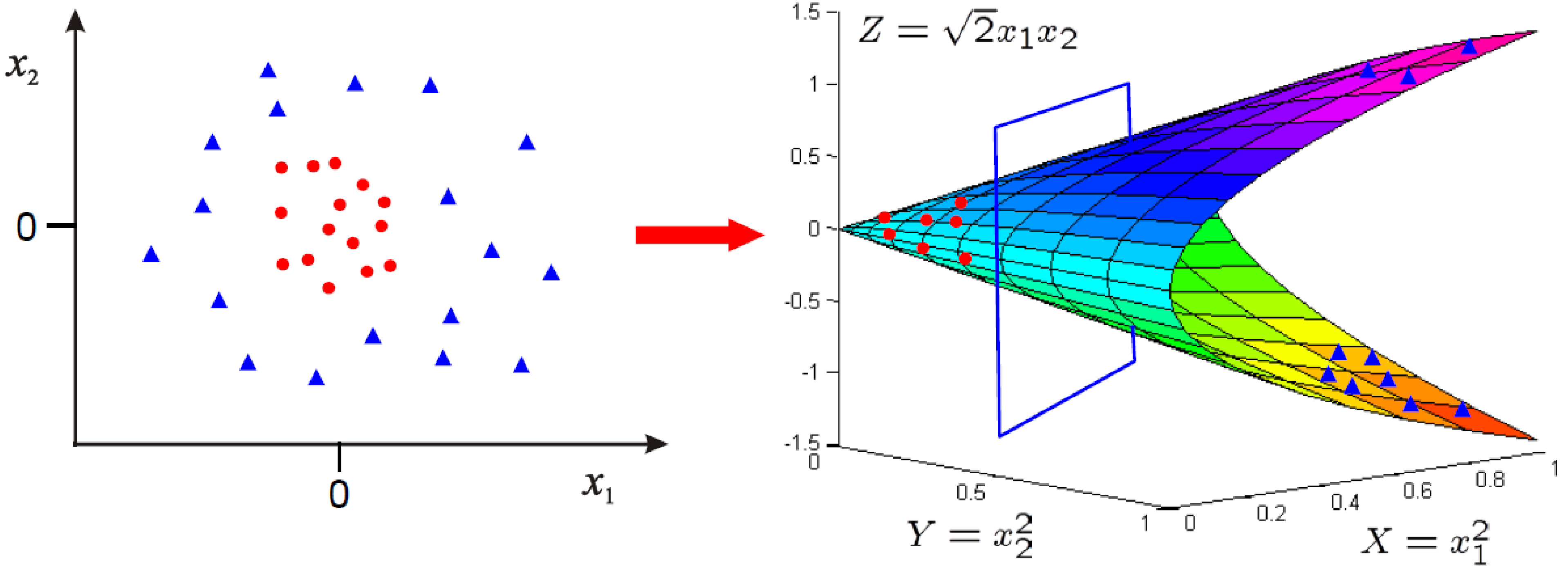
- Data is linearly separable in polar coordinates
- Acts non-linearly in original space

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} r \\ \theta \end{pmatrix} \qquad \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

Slide credit: Zisserman [link]

Solution 2: map data to higher dimension

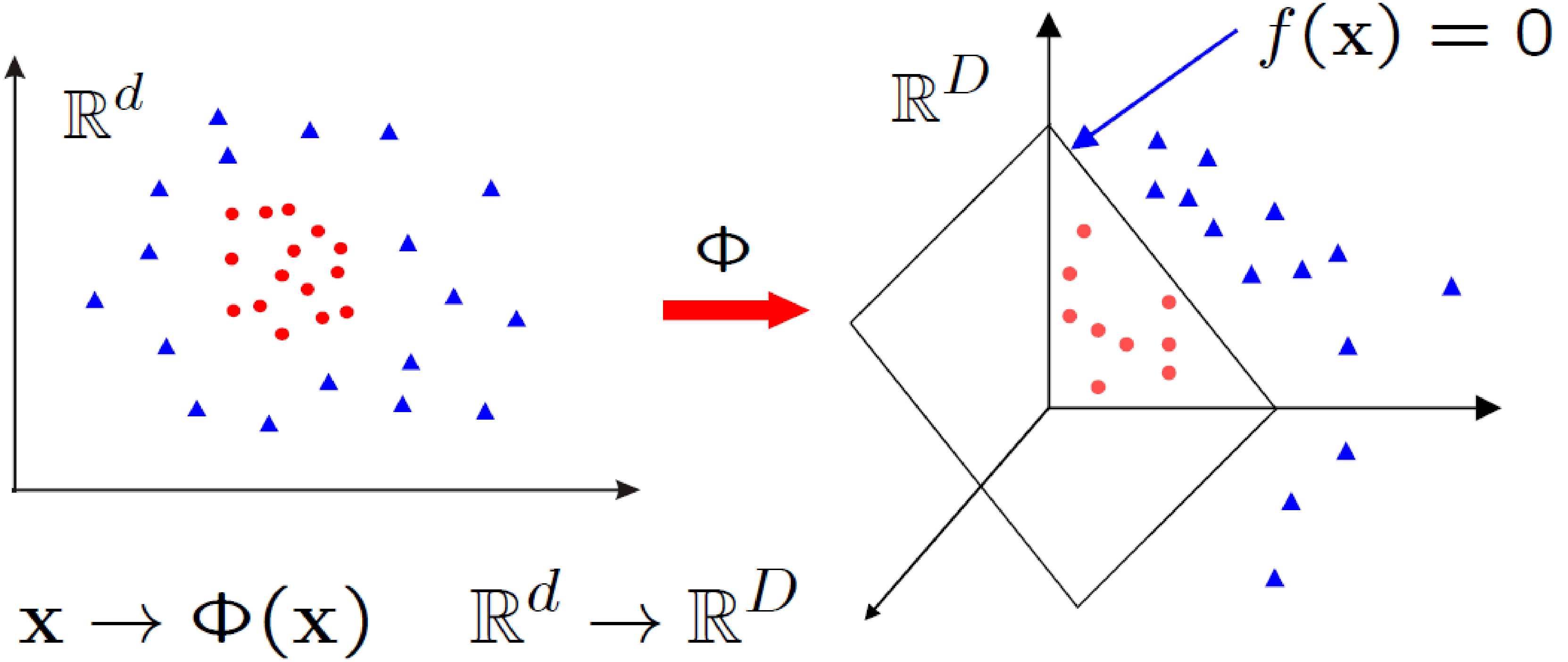
$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \qquad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$



- Data is linearly separable in 3D
- This means that the problem can still be solved by a linear classifier

Slide credit: Zisserman [link]

SVM classifiers in a transformed feature space



$$\Phi : \mathbf{x} \rightarrow \Phi(\mathbf{x}) \qquad \mathbb{R}^d \rightarrow \mathbb{R}^D$$

Learn classifier linear in $\mathbf{w}$ for $\mathbb{R}^D$:

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

$\Phi(\mathbf{x})$ is a feature map

Slide credit: Zisserman [link]

Primal Classifier in transformed feature space

Classifier, with $\mathbf{w} \in \mathbb{R}^D$:

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

Learning, for $\mathbf{w} \in \mathbb{R}^D$

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

- Simply map $\mathbf{x}$ to $\Phi(\mathbf{x})$ where data is separable
- Solve for $\mathbf{w}$ in high dimensional space $\mathbb{R}^D$
- If $D >> d$ then there are many more parameters to learn for $\mathbf{w}$. Can this be avoided?

Slide credit: Zisserman [link]

Dual Classifier in transformed feature space

Classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \, \mathbf{x}_i^\top \mathbf{x} + b$$

$$\rightarrow f(\mathbf{x}) = \sum_i^N \alpha_i y_i \, \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}) + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \mathbf{x}_j^\top \mathbf{x}_k$$

$$\rightarrow \max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_k)$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and} \sum_i \alpha_i y_i = 0$$

Slide credit: Zisserman [link]

Dual Classifier in transformed feature space

- Note, that $\Phi(\mathbf{x})$ only occurs in pairs $\Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$
- Once the scalar products are computed, only the N dimensional vector $\boldsymbol{\alpha}$ needs to be learnt; it is not necessary to learn in the D dimensional space, as it is for the primal
- Write $k(\mathbf{x}_j, \mathbf{x}_i) = \Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$. This is known as a Kernel

Classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \, k(\mathbf{x}_j, \mathbf{x}_k)$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

Slide credit: Zisserman [link]

Special transformations

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \qquad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$\begin{aligned} \Phi(\mathbf{x})^\top \Phi(\mathbf{z}) &= \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right) \begin{pmatrix} z_1^2 \\ z_2^2 \\ \sqrt{2}z_1z_2 \end{pmatrix} \\ &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\ &= (x_1z_1 + x_2z_2)^2 \\ &= (\mathbf{x}^\top \mathbf{z})^2 \end{aligned}$$

Kernel Trick

- Classifier can be learnt and applied without explicitly computing $\Phi(\mathbf{x})$
- All that is required is the kernel $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^2$
- Complexity of learning depends on N (typically it is $O(N^3)$) not on D

Slide credit: Zisserman [link]

Example kernels

- Linear kernels $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$
- Polynomial kernels $k(\mathbf{x}, \mathbf{x}') = \left(1 + \mathbf{x}^\top \mathbf{x}'\right)^d$ for any $d > 0$
  - Contains all polynomials terms up to degree d
- Gaussian kernels $k(\mathbf{x}, \mathbf{x}') = \exp\left(-||\mathbf{x} - \mathbf{x}'||^2 / 2\sigma^2\right)$ for $\sigma > 0$
  - Infinite dimensional feature space

Slide credit: Zisserman [link]

SVM classifier with Gaussian kernel

N = size of training data

$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b$$

weight (may be zero)

support vector

Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp\left(-||\mathbf{x} - \mathbf{x}'||^2 / 2\sigma^2\right)$

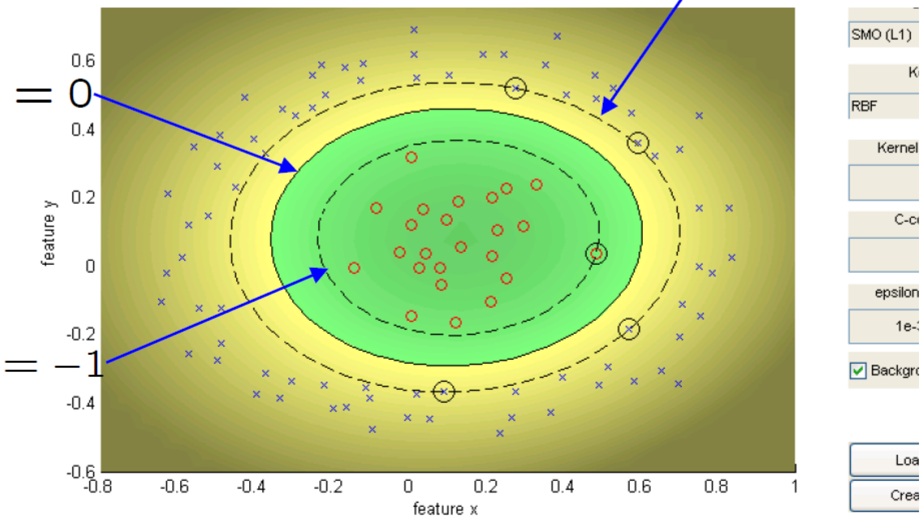
Radial Basis Function (RBF) SVM

$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i \exp\left(-||\mathbf{x} - \mathbf{x}_i||^2 / 2\sigma^2\right) + b$$

Slide credit: Zisserman [link]

Decreasing C gives a wider soft margin

$\sigma = 1.0 \quad C = \infty$

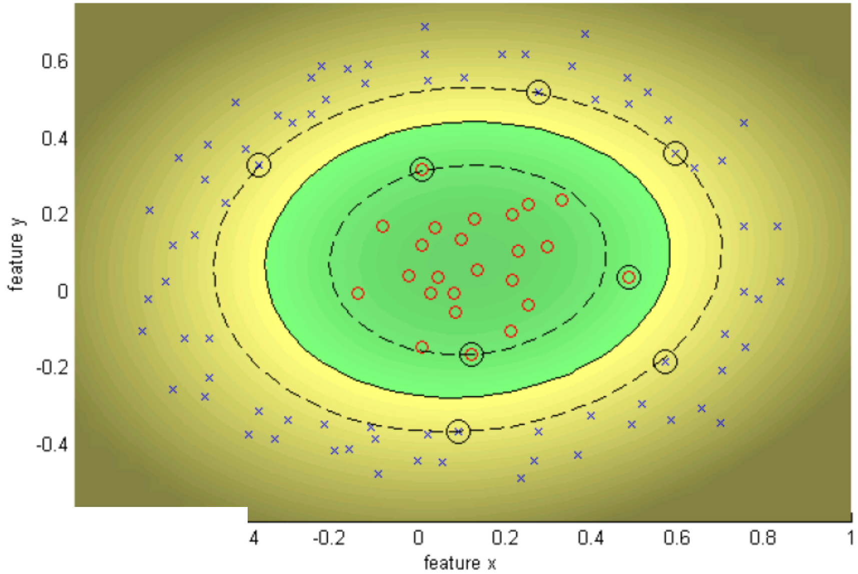


$f(\mathbf{x}) = 1$

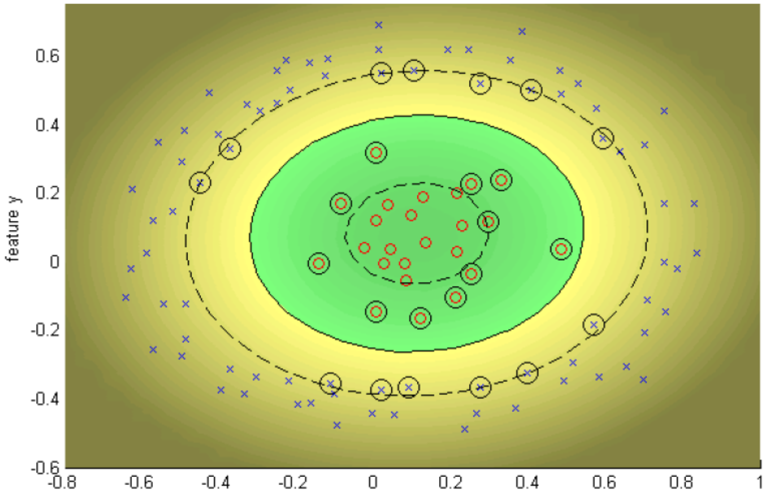
$f(\mathbf{x}) = 0$

$f(\mathbf{x}) = -1$

$\sigma = 1.0 \quad C = 100$



$\sigma = 1.0 \quad C = 10$

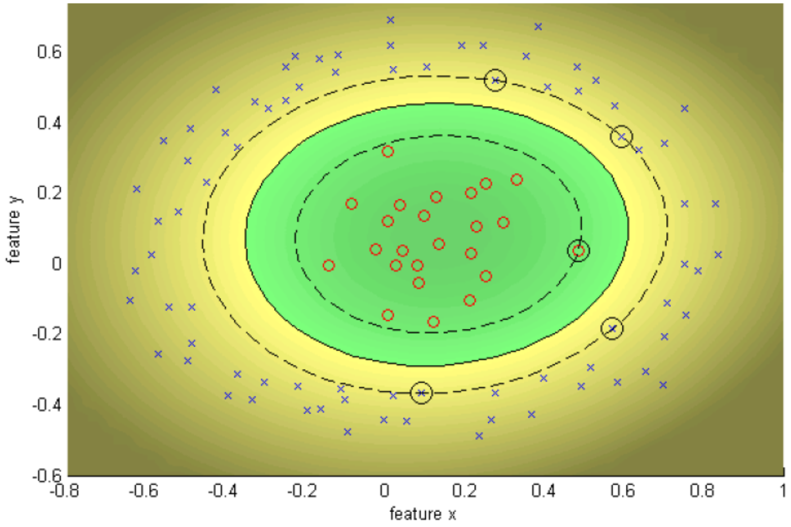


SVM with RBF Kernel Shown

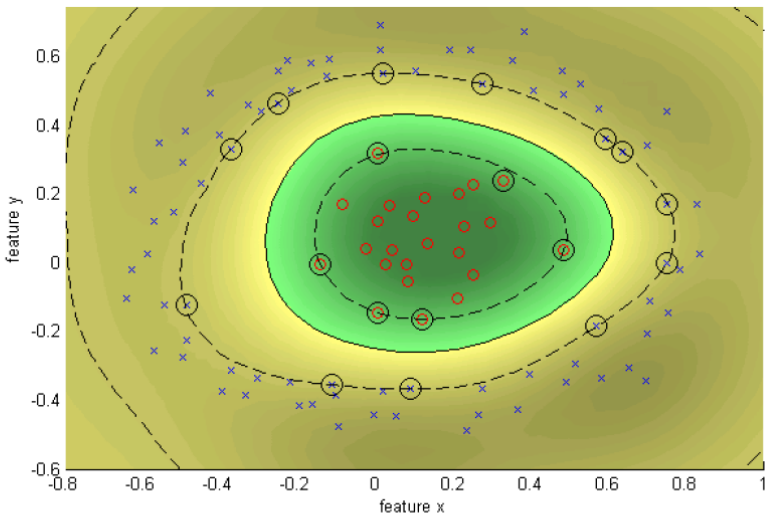
Fig credit: Zisserman [link]

Decreasing sigma makes it more like nearest neighbor

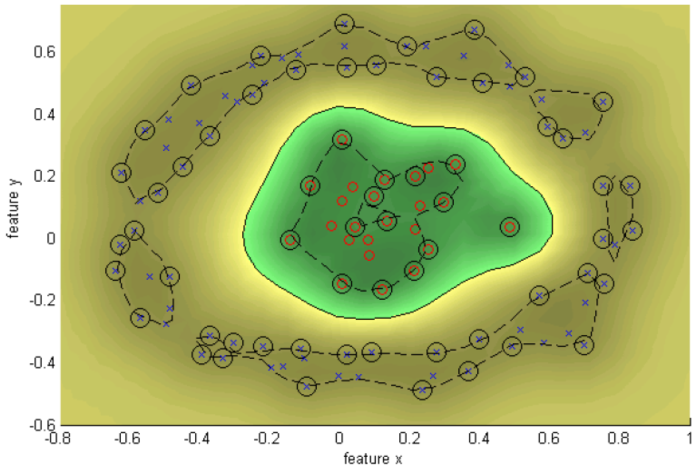
$\sigma = 1.0 \quad C = \infty$



$\sigma = 0.25 \quad C = \infty$



$\sigma = 0.1 \quad C = \infty$



SVM with RBF Kernel Shown

Fig credit: Zisserman [link]

Kernel Trick - Summary

- Classifiers can be learnt for high dimensional features spaces, without actually having to map the points into the high dimensional space

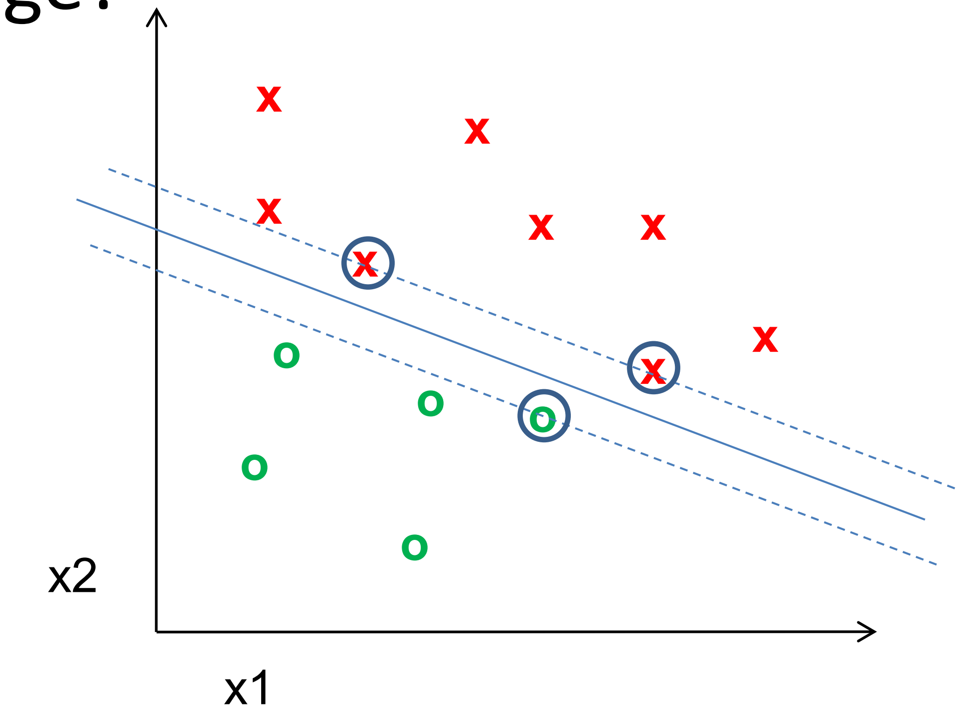
- Data may be linearly separable in the high dimensional space, but not linearly separable in the original feature space

- Kernels can be used for an SVM because of the scalar product in the dual form, but can also be used elsewhere – they are not tied to the SVM formalism

Slide credit: Zisserman [link]

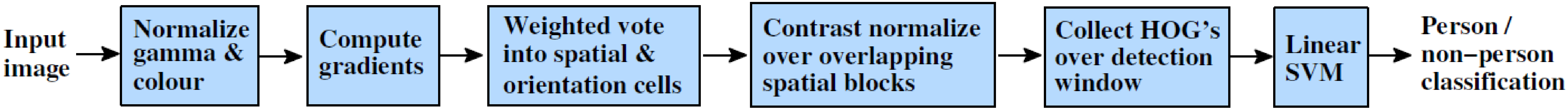
Stretch break

- If you were to remove a support vector from the training set, would the decision boundary change?



- After break
  - Application example
  - Pegasos – SGD optimization

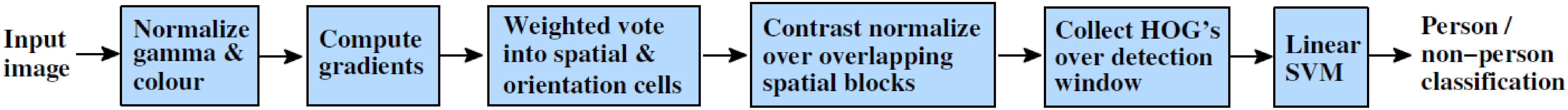
Example application of SVM: Dalal-Triggs 2005


Input image
Normalize gamma & colour
Compute gradients
Weighted vote into spatial & orientation cells
Contrast normalize over overlapping spatial blocks
Collect HOG's over detection window
Linear SVM
Person / non−person classification

- Detection by scanning window
  - Resize image to multiple scales and extract overlapping windows
  - Classify each window as positive or negative
- Very highly cited (40,000+) paper, mainly for HOG
- One of the best pedestrian detectors for several years

https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

Example application of SVM: Dalal-Triggs 2005



Input image → Normalize gamma & colour → Compute gradients → Weighted vote into spatial & orientation cells → Contrast normalize over overlapping spatial blocks → Collect HOG's over detection window → Linear SVM → Person / non–person classification

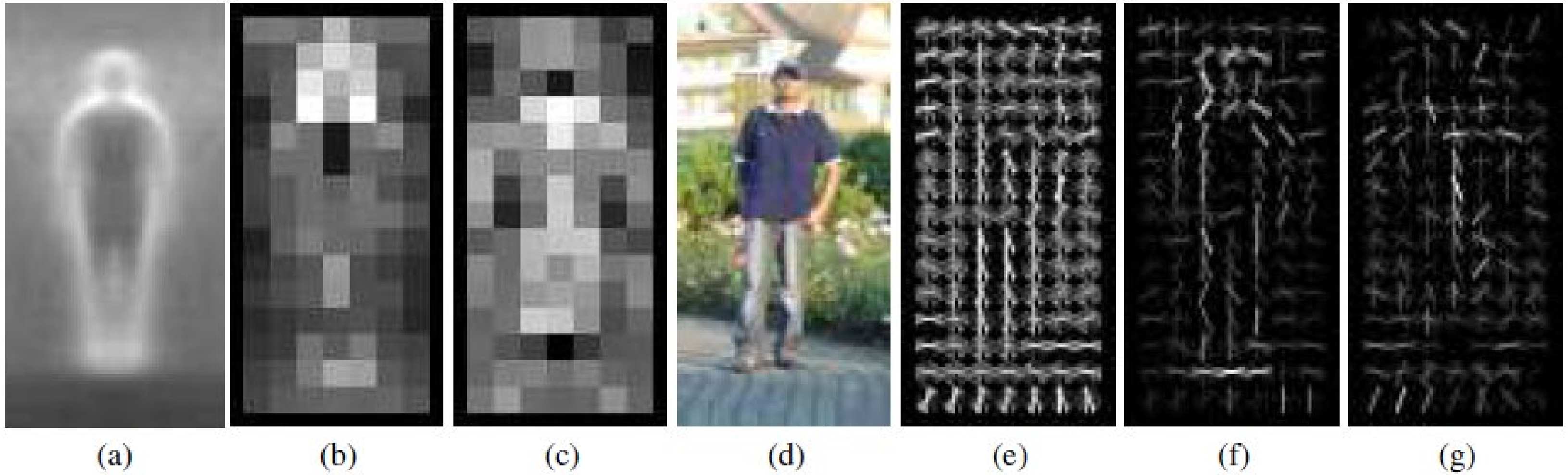
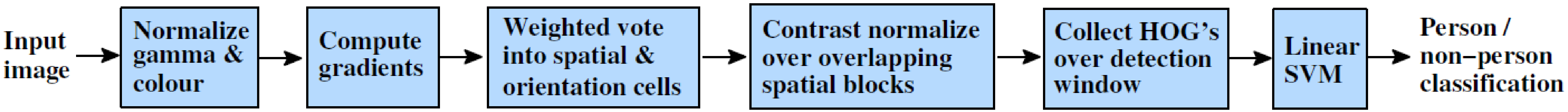


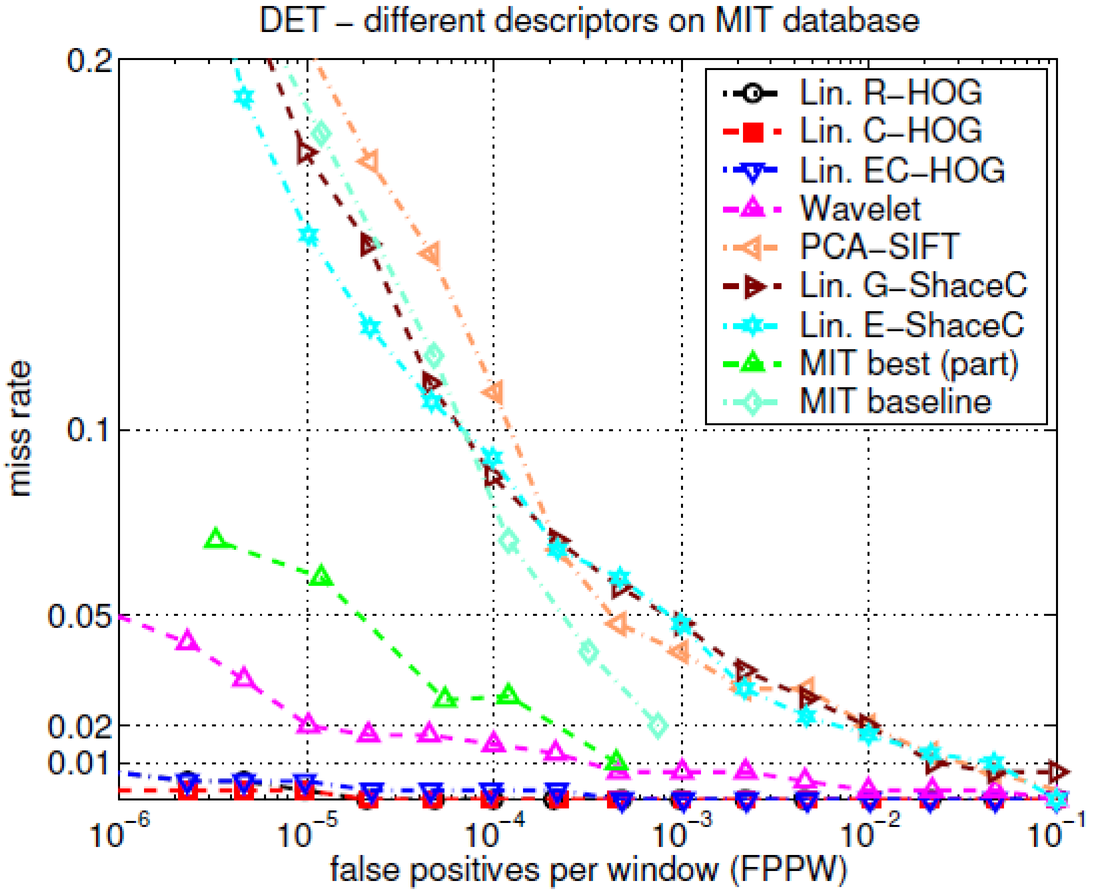
Figure 6. Our HOG detectors cue mainly on silhouette contours (especially the head, shoulders and feet). The most active blocks are centred on the image background just *outside* the contour. (a) The average gradient image over the training examples. (b) Each "pixel" shows the maximum positive SVM weight in the block centred on the pixel. (c) Likewise for the negative SVM weights. (d) A test image. (e) It's computed R-HOG descriptor. (f,g) The R-HOG descriptor weighted by respectively the positive and the negative SVM weights.

- Very highly cited (40,000+) paper, mainly for HOG
- One of the best pedestrian detectors for several years

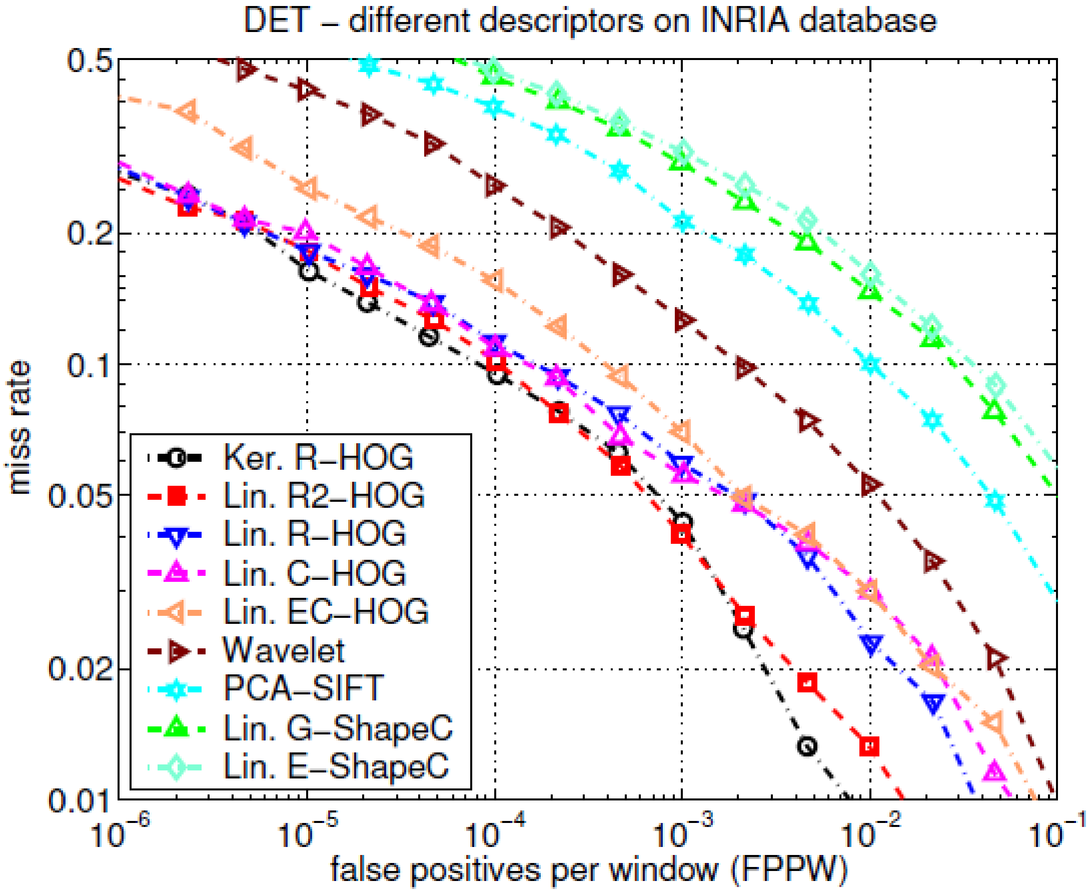
Example application of SVM: Dalal-Triggs 2005



Input image → Normalize gamma & colour → Compute gradients → Weighted vote into spatial & orientation cells → Contrast normalize over overlapping spatial blocks → Collect HOG's over detection window → Linear SVM → Person / non−person classification



DET – different descriptors on MIT database



DET – different descriptors on INRIA database

Using SVMs

- Good broadly applicable classifier
  - Strong foundation in statistical learning theory
  - Works well with many weak features
  - Requires parameter tuning for C
  - Non-linear SVM requires defining a kernel, and slower optimization/prediction
    - RBF: related to neural networks, nearest neighbor (requires additional tuning)
    - Chi-squared, histogram intersection: good for histograms (but slower, esp. chi-squared)
    - Can learn a kernel function

- Negatives
  - Feature learning is not part of the framework (vs trees and neural nets)
  - Slow training (especially for kernels) – until Pegasos!

Pegasos: Primal Estimated sub-GrAdient SOlver for SVM (2011)

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{(\mathbf{x},y)\in S} \ell(\mathbf{w};(\mathbf{x},y))$$

$$\ell(\mathbf{w};(\mathbf{x},y)) \;=\; \max\{0, 1 - y\,\langle \mathbf{w},\mathbf{x}\rangle\}$$

SVM problem that we want to solve (Minimize weights square + sum of hinge losses on all samples)

$$f(\mathbf{w};i_t) \;=\; \frac{\lambda}{2}\|\mathbf{w}\|^2 + \ell(\mathbf{w};(\mathbf{x}_{i_t},y_{i_t}))$$

Problem in terms of **one** sample

$$\nabla_t = \lambda\,\mathbf{w}_t - \mathbb{1}[y_{i_t}\,\langle \mathbf{w}_t,\mathbf{x}_{i_t}\rangle < 1]\,y_{i_t}\mathbf{x}_{i_t}$$

Gradient in terms of one sample

- Direction to move to improve solution

https://home.ttic.edu/~nati/Publications/PegasosMPB.pdf

Gradient Descent Visualization

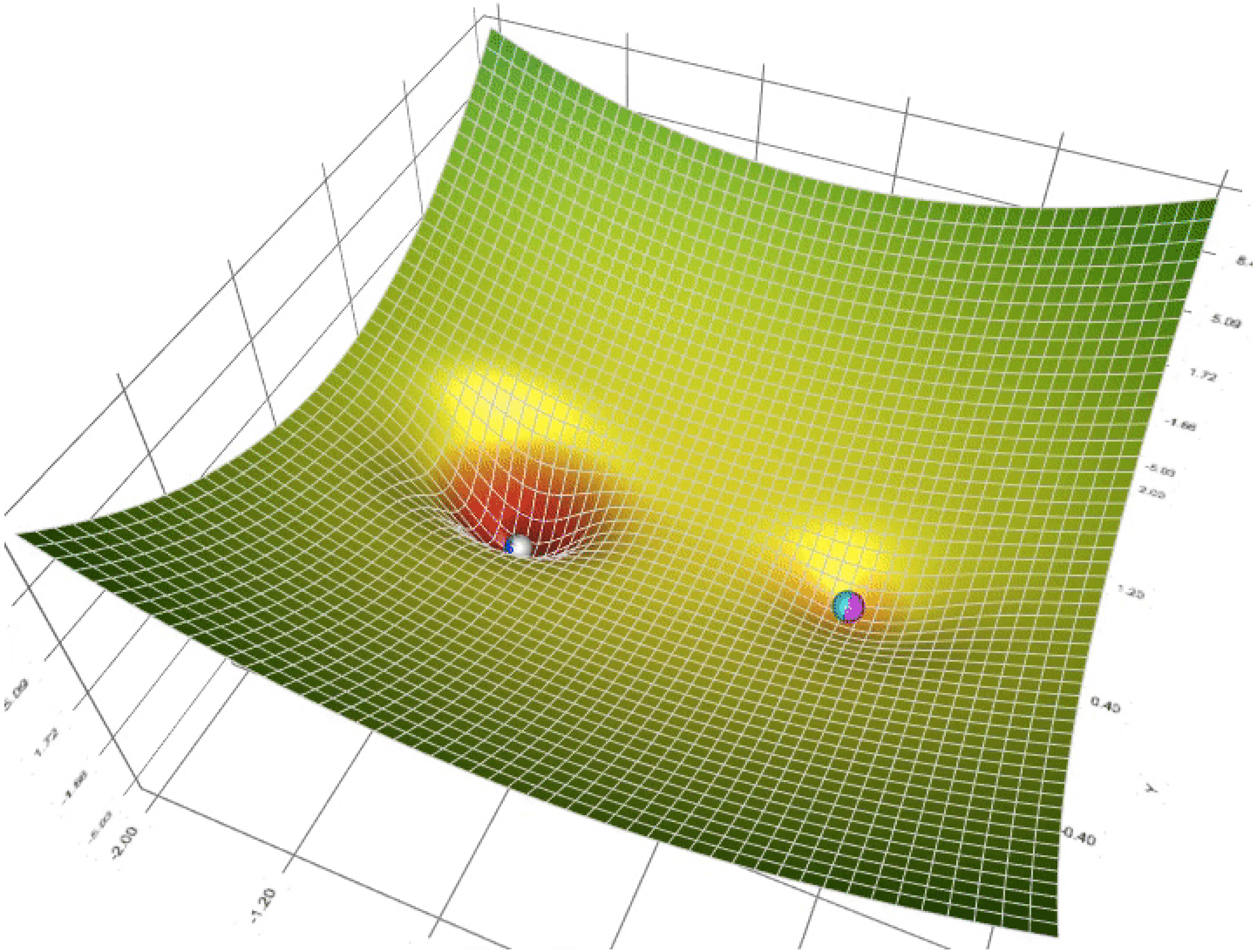
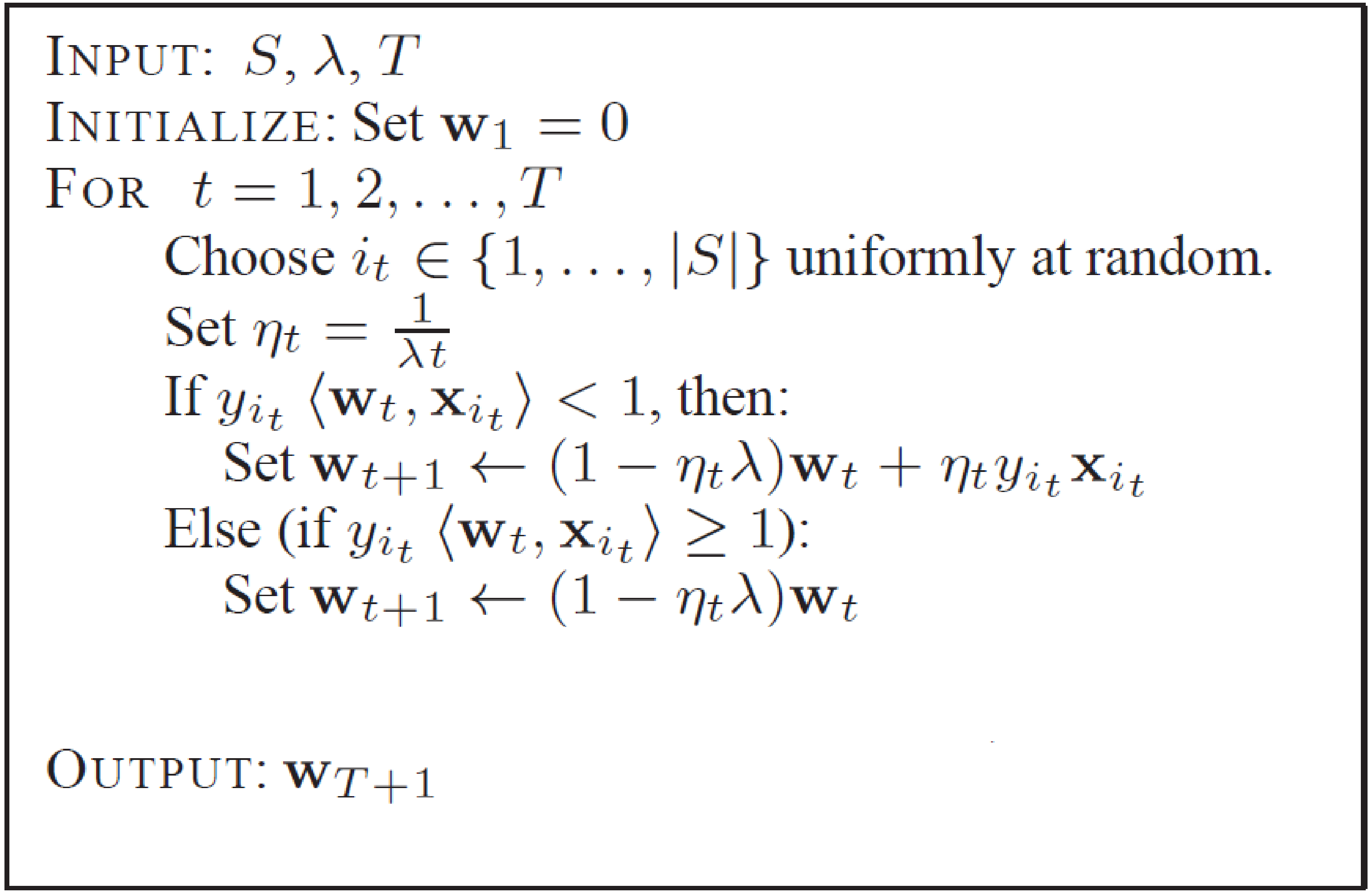


Figure source

Pegasos algorithm: Stochastic Gradient Descent (SGD)



```
INPUT: $S, \lambda, T$
INITIALIZE: Set $\mathbf{w}_1 = 0$
FOR $t = 1, 2, \ldots, T$
        Choose $i_t \in \{1, \ldots, |S|\}$ uniformly at random.
        Set $\eta_t = \frac{1}{\lambda t}$
        If $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1$, then:
            Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \eta_t y_{i_t} \mathbf{x}_{i_t}$
        Else (if $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle \geq 1$):
            Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t$

OUTPUT: $\mathbf{w}_{T+1}$
```

Notation

S: training set

λ: regularization weight

T: number iterations

$\boldsymbol{w}_t$: model weights

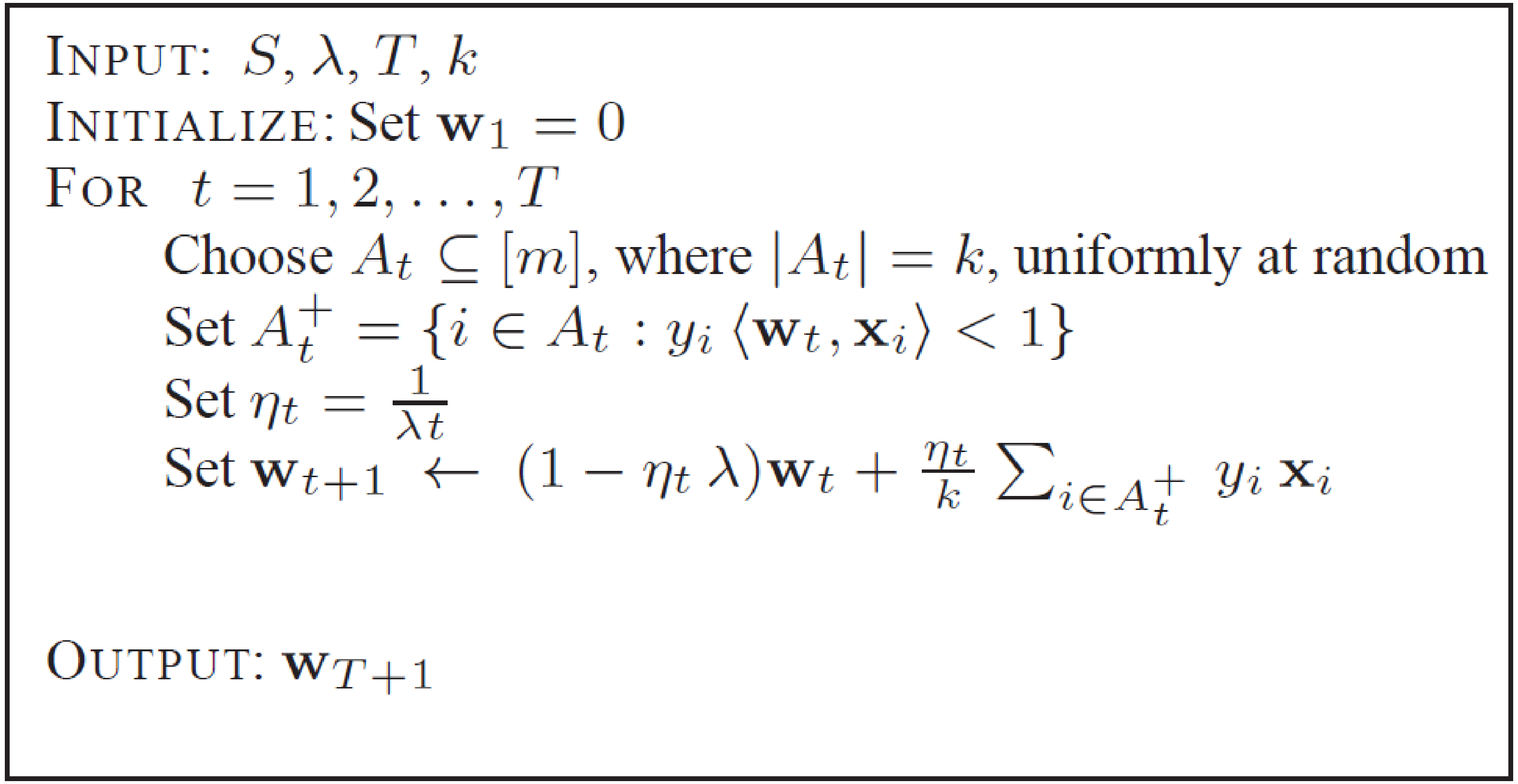
$\boldsymbol{x}_{i_t}$: features for example i_t

y_{i_t}: label for example i_t

η_t: step size (“learning rate”)

Pegasos with **mini-batch**

- Calculating gradient based on multiple examples reduces variance of gradient estimate



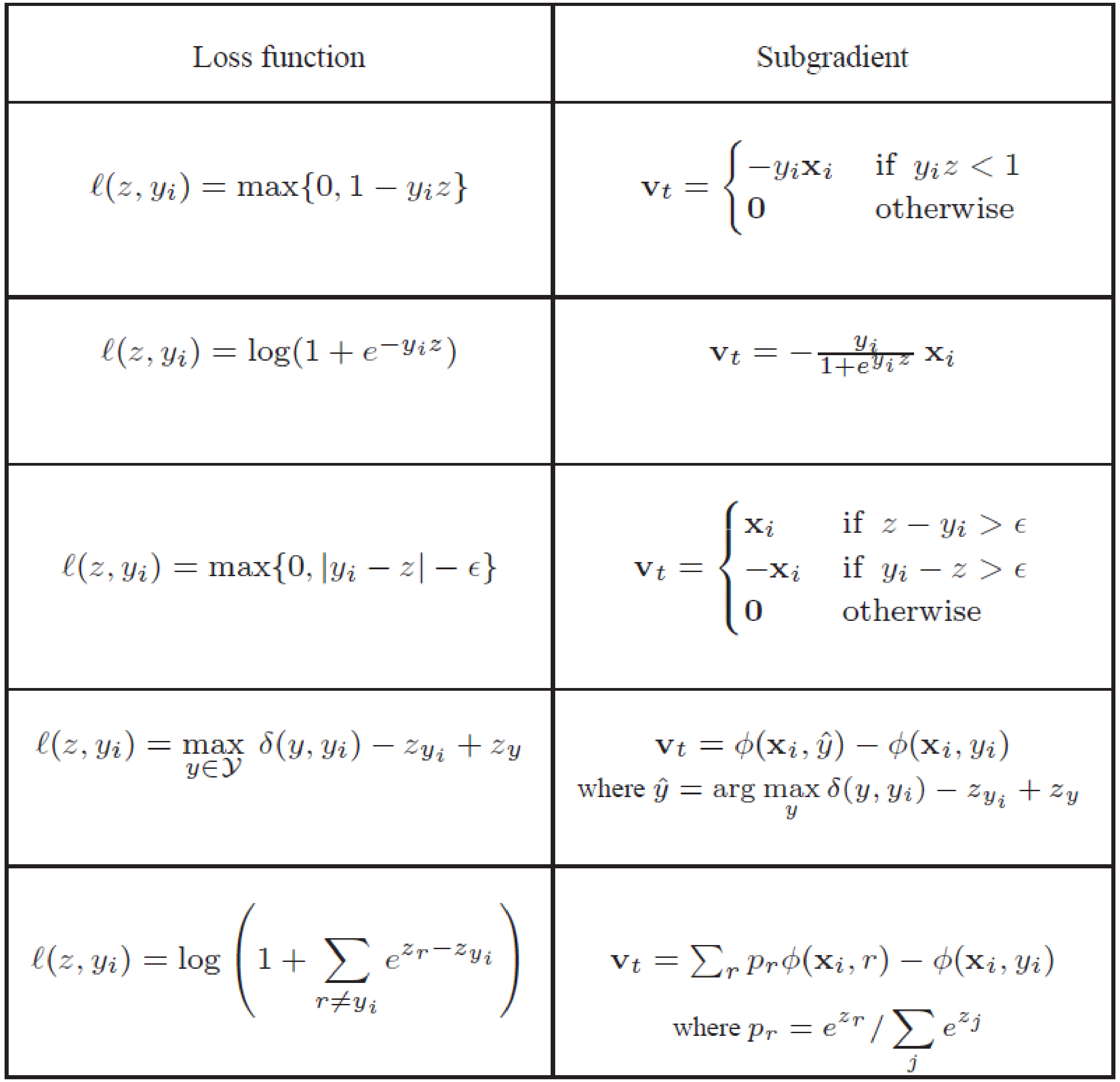
```
INPUT: $S, \lambda, T, k$
INITIALIZE: Set $\mathbf{w}_1 = 0$
FOR $t = 1, 2, \ldots, T$
    Choose $A_t \subseteq [m]$, where $|A_t| = k$, uniformly at random
    Set $A_t^+ = \{i \in A_t : y_i \langle \mathbf{w}_t, \mathbf{x}_i \rangle < 1\}$
    Set $\eta_t = \frac{1}{\lambda t}$
    Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \frac{\eta_t}{k} \sum_{i \in A_t^+} y_i \mathbf{x}_i$


OUTPUT: $\mathbf{w}_{T+1}$
```

k: batch size
m: number of training samples
A_t: batch of examples
A_t^+: examples within margin

S: training set
λ: regularization weight
T: number iterations
$\boldsymbol{w}_t$: model weights
$\boldsymbol{x}_i$: features for example i
y_i: label for example i
η_t: step size (“learning rate”)

SGD applies to many losses

z is the score
for y=1



	Loss function	Subgradient
SVM (hinge loss)	$\ell(z, y_i) = \max\{0, 1 - y_i z\}$	$\mathbf{v}_t = \begin{cases} -y_i\mathbf{x}_i & \text{if } y_i z < 1 \\ 0 & \text{otherwise} \end{cases}$
Logistic regression / sigmoid loss	$\ell(z, y_i) = \log(1 + e^{-y_i z})$	$\mathbf{v}_t = -\frac{y_i}{1+e^{y_i z}}\, \mathbf{x}_i$
Hinge L1 regression	$\ell(z, y_i) = \max\{0, \lvert y_i - z\rvert - \epsilon\}$	$\mathbf{v}_t = \begin{cases} \mathbf{x}_i & \text{if } z - y_i > \epsilon \\ -\mathbf{x}_i & \text{if } y_i - z > \epsilon \\ 0 & \text{otherwise} \end{cases}$
Margin loss between scores of most likely and correct label	$\ell(z, y_i) = \max_{y \in \mathcal{Y}} \delta(y, y_i) - z_{y_i} + z_y$	$\mathbf{v}_t = \phi(\mathbf{x}_i, \hat{y}) - \phi(\mathbf{x}_i, y_i)$ where $\hat{y} = \arg\max_y \delta(y, y_i) - z_{y_i} + z_y$
Variant of a logistic loss	$\ell(z, y_i) = \log\left(1 + \sum_{r \neq y_i} e^{z_r - z_{y_i}}\right)$	$\mathbf{v}_t = \sum_r p_r \phi(\mathbf{x}_i, r) - \phi(\mathbf{x}_i, y_i)$ where $p_r = e^{z_r} / \sum_j e^{z_j}$

SGD is fast compared to other optimization approaches

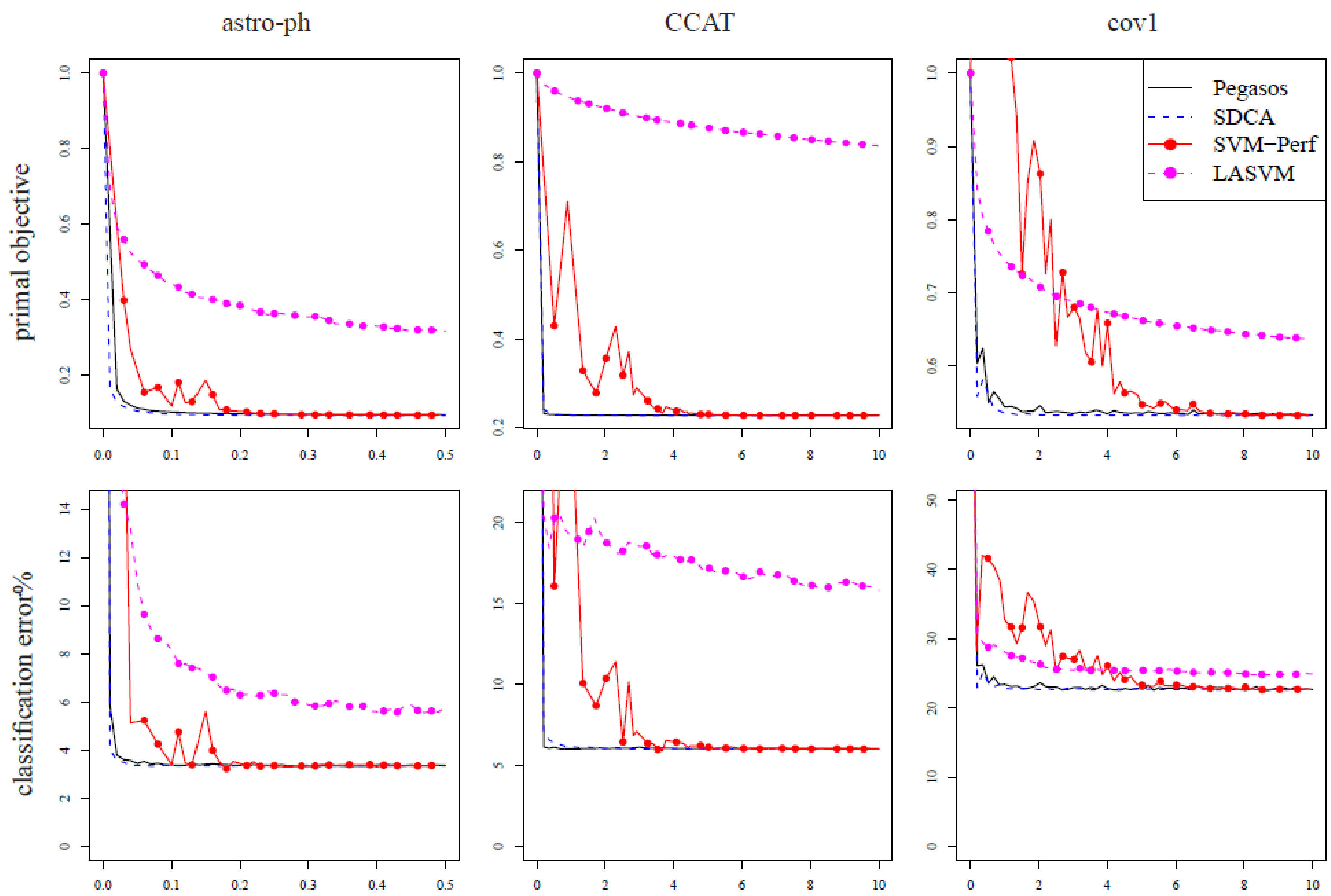
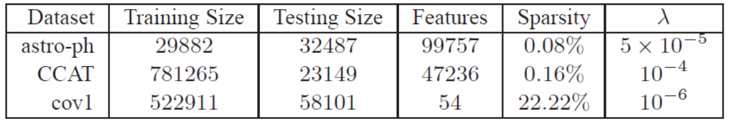


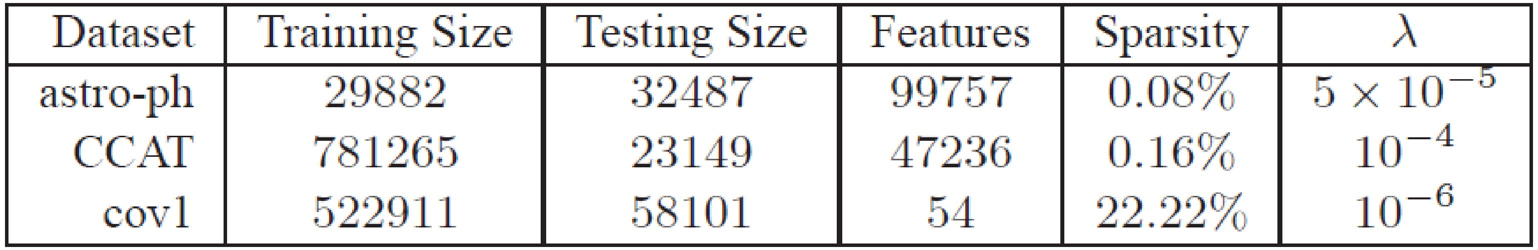
Fig. 4 Comparison of linear SVM optimizers. Primal suboptimality (top row) and testing classification error (bottom row), for one run each of Pegasos, stochastic DCA, SVM-Perf, and LASVM, on the astro-ph (left), CCAT (center) and cov1 (right) datasets. In all plots the horizontal axis measures runtime in seconds.



Dataset	Training Size	Testing Size	Features	Sparsity	λ
astro-ph	29882	32487	99757	0.08%	5×10^{-5}
CCAT	781265	23149	47236	0.16%	10^{-4}
cov1	522911	58101	54	22.22%	10^{-6}

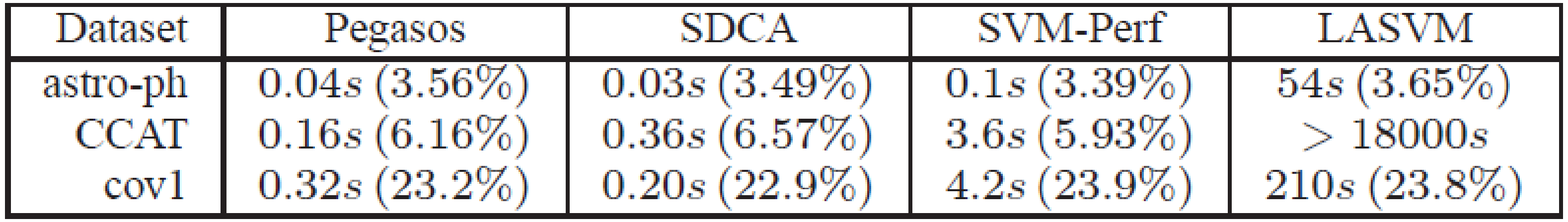
SDCA = stochastic dual coordinate descent, another form of sub-gradient optimization that chooses learning rate dynamically

Experiments with Linear SVM



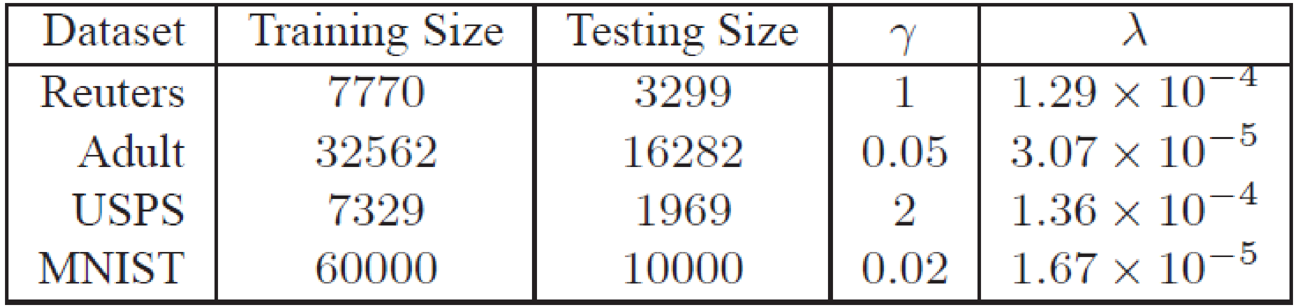
Dataset	Training Size	Testing Size	Features	Sparsity	λ
astro-ph	29882	32487	99757	0.08%	5×10^{-5}
CCAT	781265	23149	47236	0.16%	10^{-4}
cov1	522911	58101	54	22.22%	10^{-6}

Training time and test error

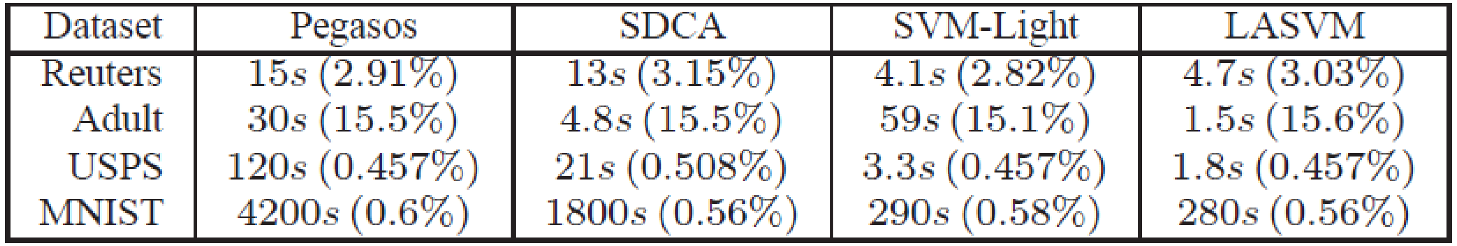


Dataset	Pegasos	SDCA	SVM-Perf	LASVM
astro-ph	$0.04s$ (3.56%)	$0.03s$ (3.49%)	$0.1s$ (3.39%)	$54s$ (3.65%)
CCAT	$0.16s$ (6.16%)	$0.36s$ (6.57%)	$3.6s$ (5.93%)	$> 18000s$
cov1	$0.32s$ (23.2%)	$0.20s$ (22.9%)	$4.2s$ (23.9%)	$210s$ (23.8%)

Experiments using Gaussian kernel SVM (see paper for kernelized Pegasos algorithm)



Dataset	Training Size	Testing Size	γ	λ
Reuters	7770	3299	1	1.29×10^{-4}
Adult	32562	16282	0.05	3.07×10^{-5}
USPS	7329	1969	2	1.36×10^{-4}
MNIST	60000	10000	0.02	1.67×10^{-5}



Dataset	Pegasos	SDCA	SVM-Light	LASVM
Reuters	$15s$ (2.91%)	$13s$ (3.15%)	$4.1s$ (2.82%)	$4.7s$ (3.03%)
Adult	$30s$ (15.5%)	$4.8s$ (15.5%)	$59s$ (15.1%)	$1.5s$ (15.6%)
USPS	$120s$ (0.457%)	$21s$ (0.508%)	$3.3s$ (0.457%)	$1.8s$ (0.457%)
MNIST	$4200s$ (0.6%)	$1800s$ (0.56%)	$290s$ (0.58%)	$280s$ (0.56%)

Effect of mini-batch size

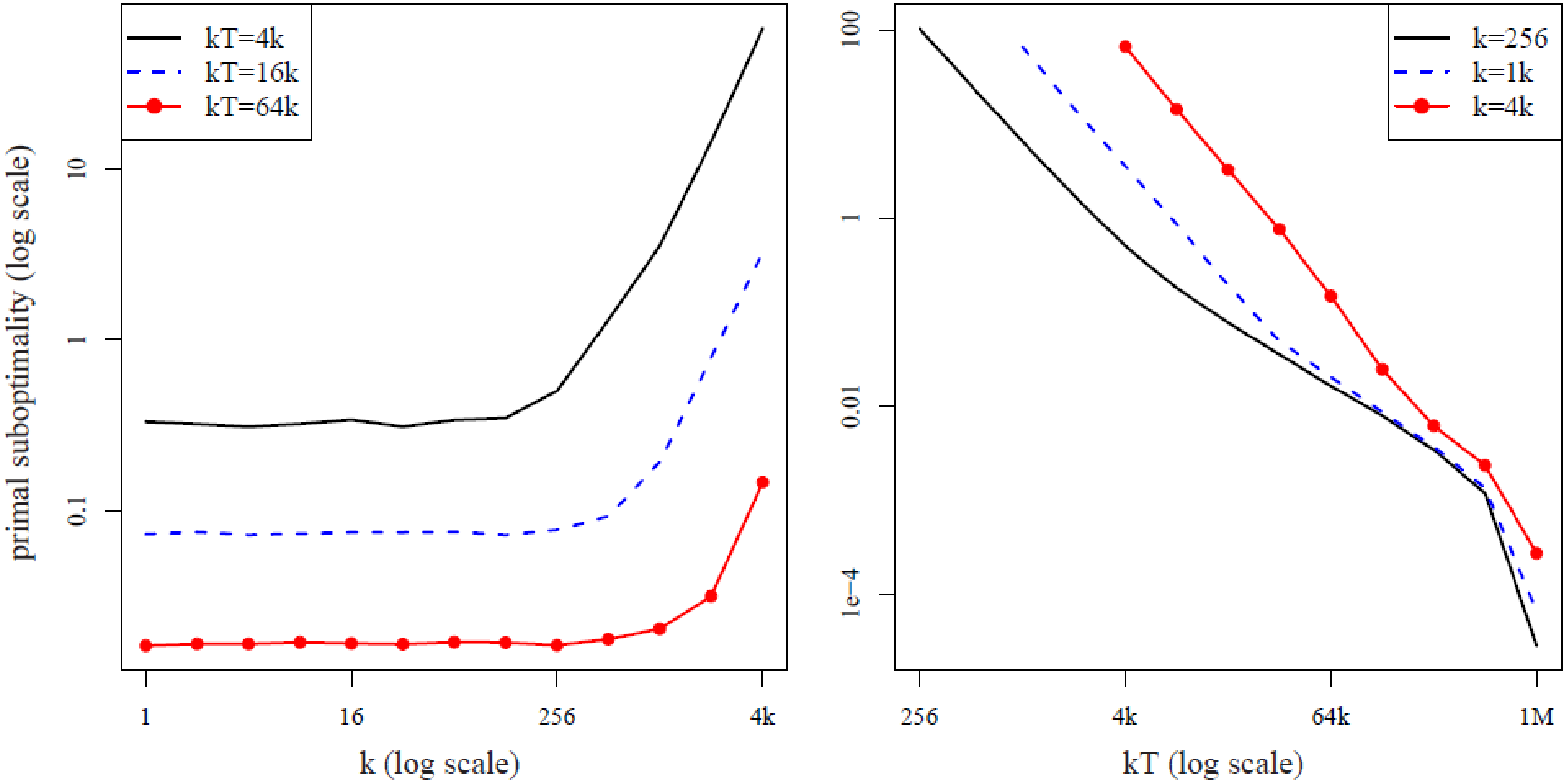
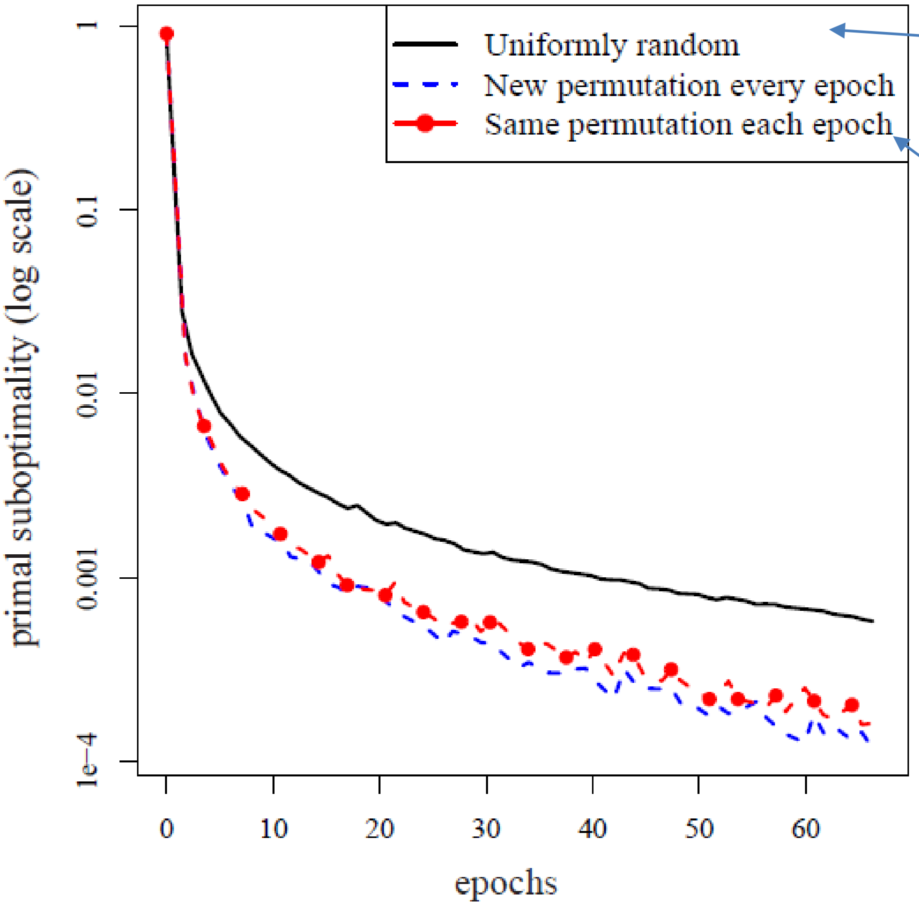


Fig. 7 The effect of the mini-batch size on the runtime of Pegasos for the astro-ph dataset. The first plot shows the primal suboptimality achieved for certain fixed values of overall runtime kT, for various values of the mini-batch size k. The second plot shows the primal suboptimality achieved for certain fixed values of k, for various values of kT. Very similar results were achieved for the CCAT dataset.

Effect of sampling procedure: randomly ordered epochs is best



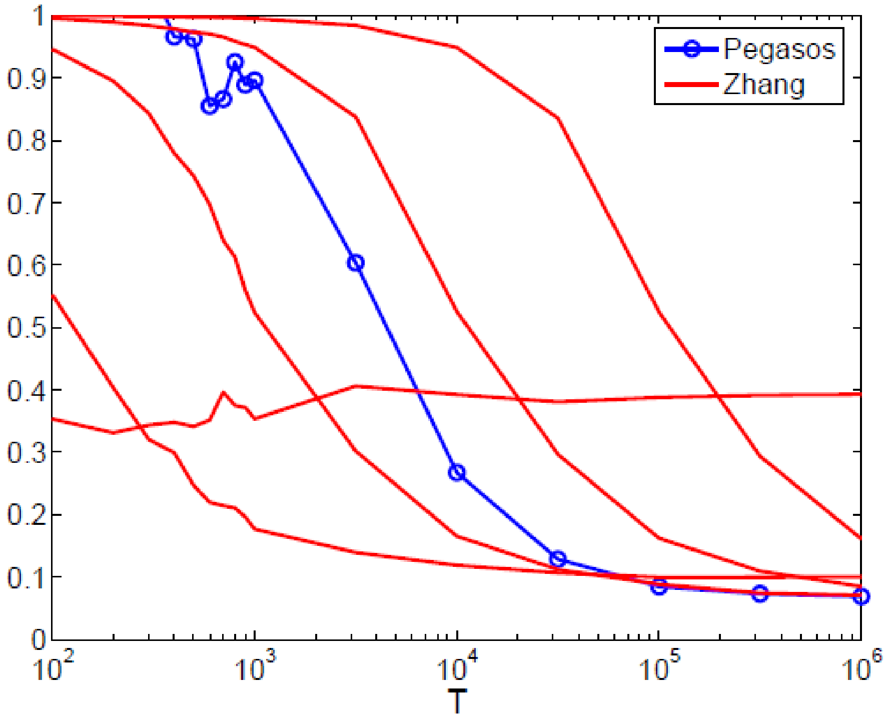
Sampling with replacement

Use different random order for each “epoch”

Use same order for each epoch

Epoch: one run through the training set

Learning rate comparison



Zhang uses fixed learning rate

Plots show error over iterations for several rates

Pegasos: take-ways and surprising facts

- SGD is very simple and effective optimization algorithm – step toward better solution based on a small sample of training data

- Not very sensitive to mini-batch size (but larger batches can be much faster with parallel processing)

- The same learning schedule is effective across several problems

- A *larger training set* makes it *faster* to obtain the same test performance

Next week

- Neural networks
  - Multi-layer perceptrons (MLP)
  - Deep networks