https://tinyurl.com/cs441chat

Linear Classifiers: Logistic Regression and SVM

Applied Machine Learning
Derek Hoiem

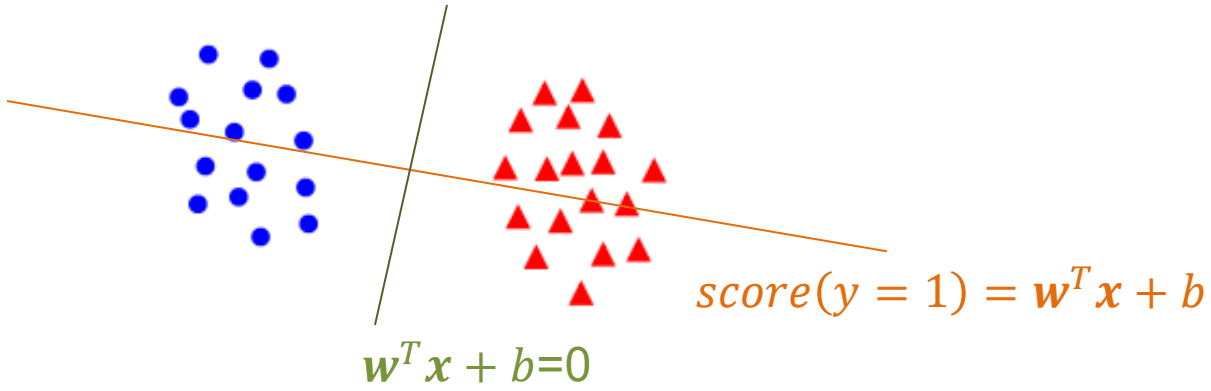
Dall-E

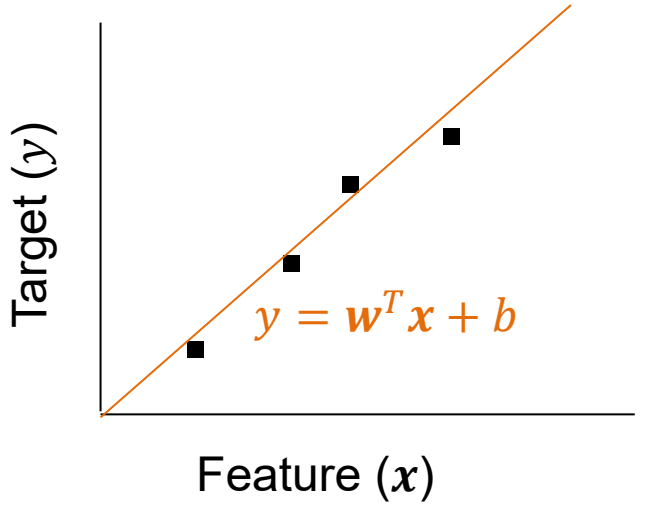
Linear Models

- A model is **linear** in x if it is based on a weighted sum of the values of x (optionally, plus a constant)

$$\boldsymbol{w}^T\boldsymbol{x} + b = \left[\sum_i w_i x_i\right] + b$$

- A **linear classifier** projects the features onto a score that indicates whether the label is positive or negative (i.e., one class or the other). We often show the boundary where that score is equal to zero.
- A **linear regressor** finds a linear model that approximates the prediction value for each set of features.





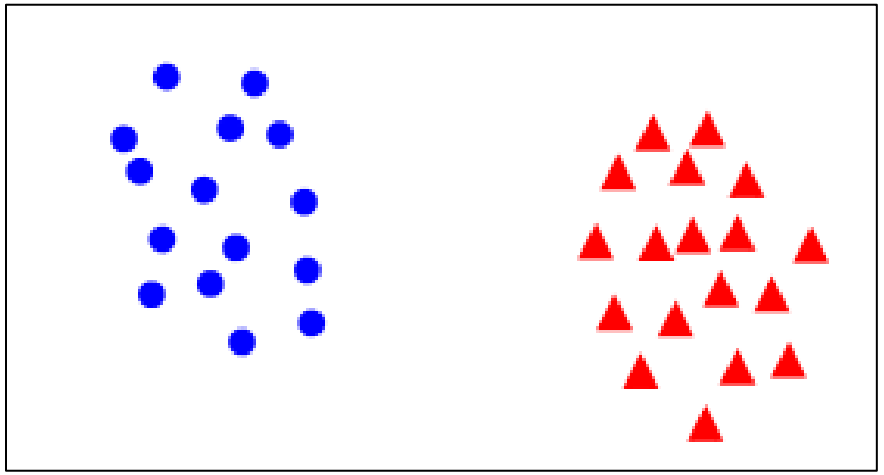
Today’s Lecture

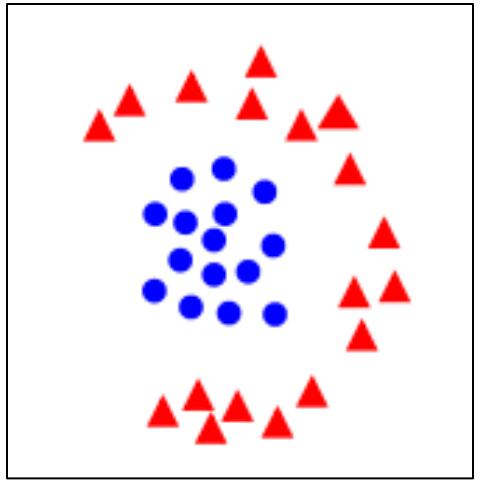
- Linear logistic regression: maximize likelihood of target labels given the features

- SVM: maximize the number of data points with confidently correct predictions

Linear Classifiers and Linear Separability

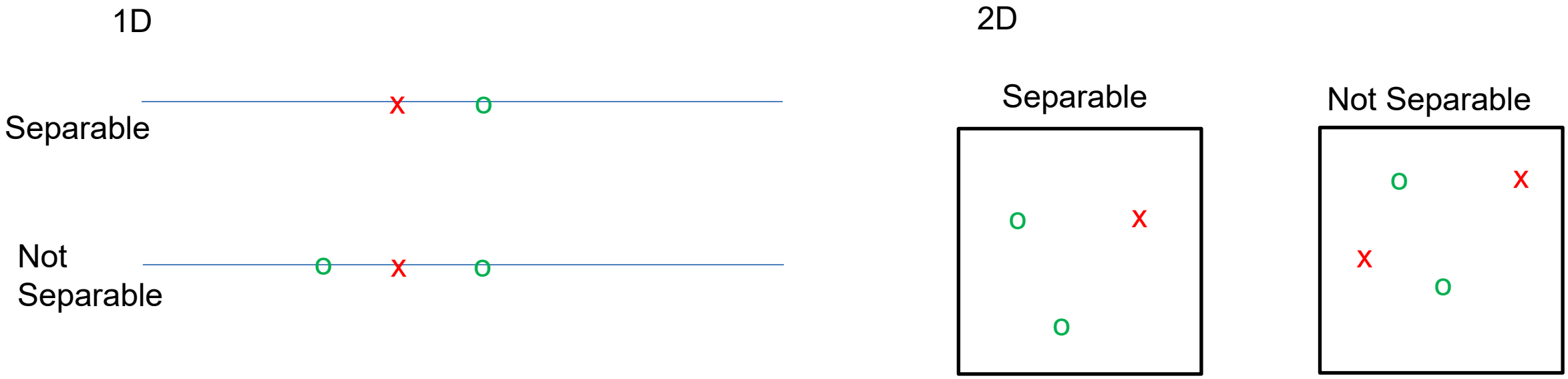
- **Linear classifier**: $\mathrm{y} = 1$ if $\boldsymbol{w}^T \boldsymbol{x} + b > 0$
- **Linearly separable**: a line (or hyperplane) in feature space can split the two labels
- Which of these are linearly separable?





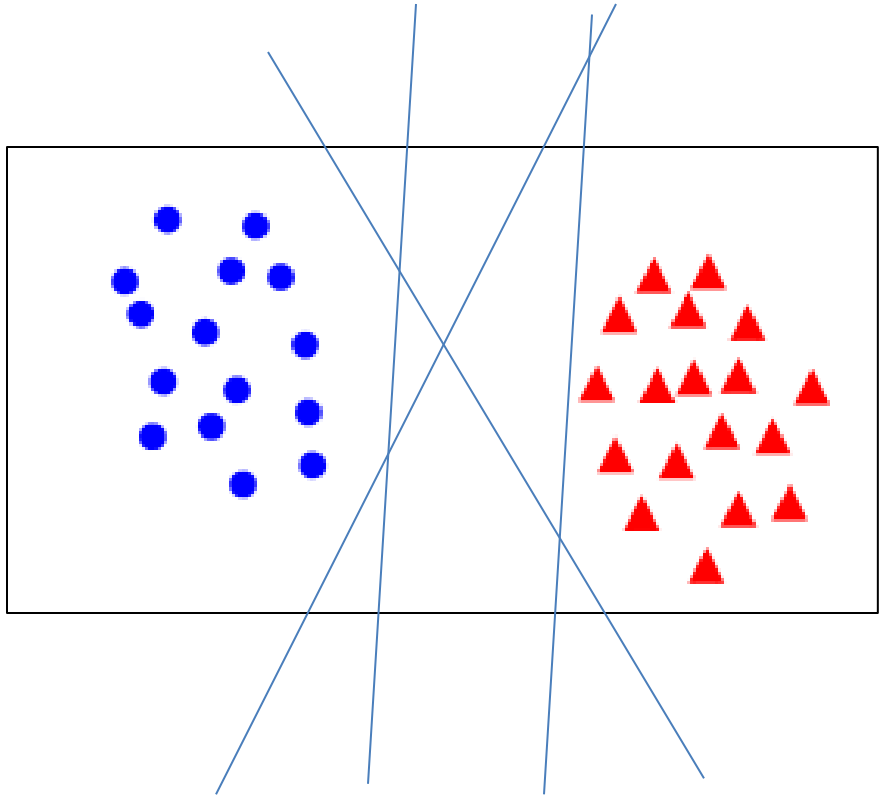
Linear Classifiers and Linear Separability

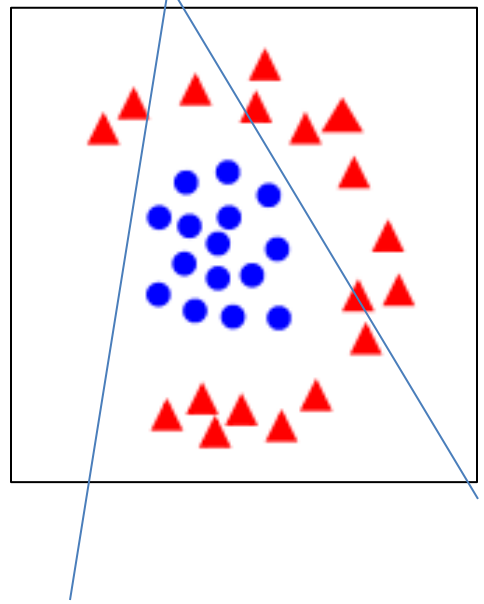
- In high dimensions, a lot more things are linearly separable
- If you have D dimensions, you can separate D+1 points with any arbitrary labeling



Linear Classifiers and Linear Separability

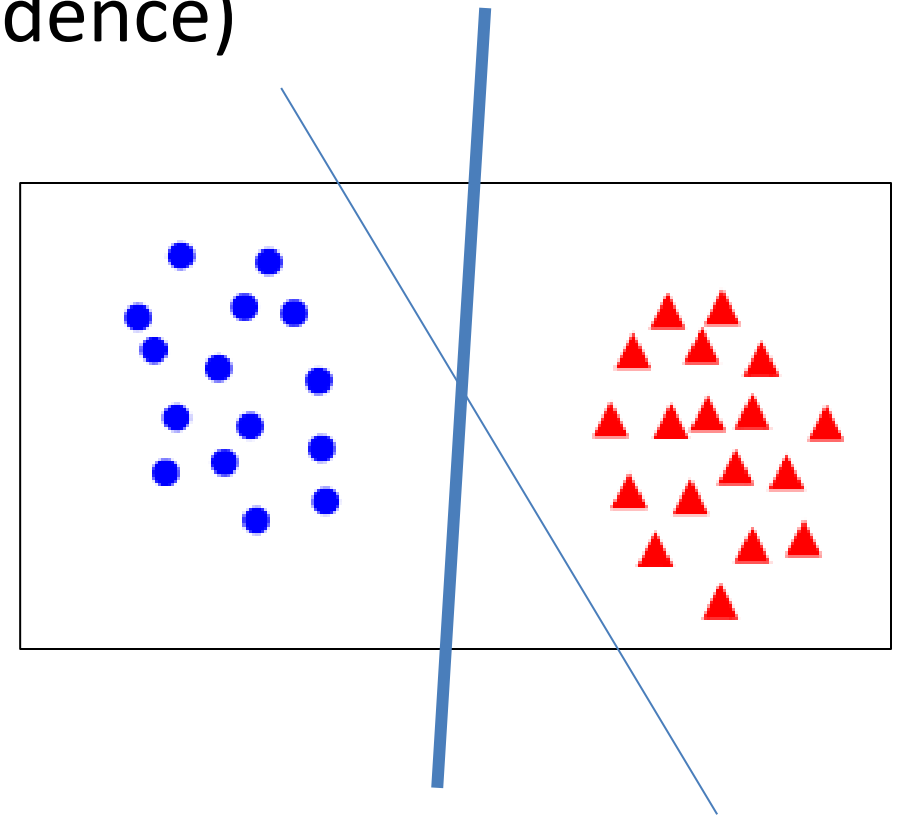
- But how do you choose which line is best?
- Different classifiers use different objectives to choose the line





Linear Classifiers and Linear Separability

- Different classifiers use different objectives to choose the line
- Common principles are that you want training samples on the correct side of the line (low classification error) by some margin (high confidence)



Thick line is better classification function than thin line because all the examples have a good margin

(Linear) Logistic Regression Model

$$\text{maximize } P(\boldsymbol{y}|\boldsymbol{x}), \quad y \in \{-1, 1\}$$

$$\text{where } P(y = 1|\boldsymbol{x}) = \frac{1}{1 + \exp(-[\boldsymbol{w}^T\boldsymbol{x} + b])}$$

← "Logistic function"

$$\boldsymbol{w}^T\boldsymbol{x} + b \approx \log\frac{P(y = 1|x)}{P(y = -1|x)}$$

← "Logit"

* To simplify notation, I may omit the "b", which can be avoided by adding a "1" to each feature vector

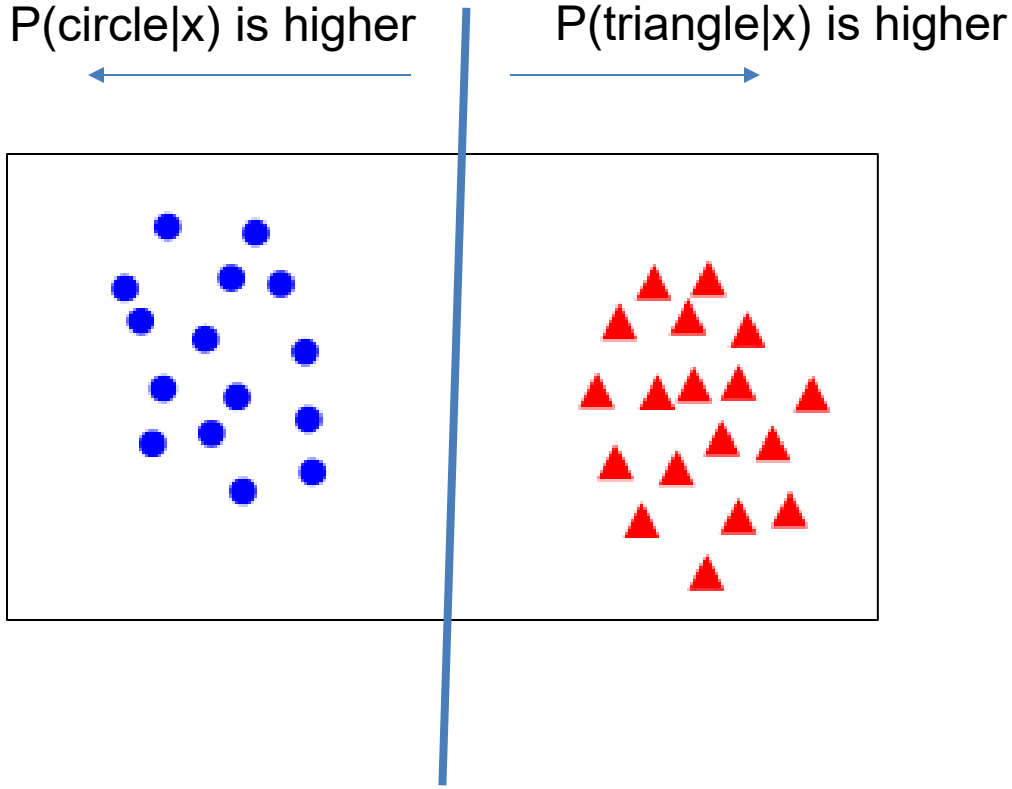
Note:

$$P(y = -1|x) = P(y = 1|x) = \frac{\exp(-[w^T x + b])}{1 + \exp(-[w^T x + b])} = \frac{1}{1 + \exp([w^T x + b])}$$

$$P(y = \hat{y}\,|x) = \frac{1}{1 + \exp(-\hat{y}[w^T x + b])}$$

Linear Logistic Regression

- The further you are from the line, the more confident in a label



Deriving the loss of logistic regression

Maximize probability of correct label given features of each data point, assuming the data points are conditionally independent

$$w^* = \underset{w}{\operatorname{argmax}} \prod_n P(y = y_n | x_n; w)$$

Maximizing $\log\big(f(x)\big)$ is the same as maximizing $f(x)$ because $\log(x)$ monotonically increases with x

Log of product is the sum of logs.

$$w^* = \underset{w}{\operatorname{argmax}} \sum_n \log P(y = y_n | x_n; w)$$

Turn it into a minimization problem (as a convention)

$$w^* = \underset{w}{\operatorname{argmin}} - \sum_n \log P(y = y_n | x_n; w)$$

Linear Logistic Regression algorithm

- Training

$$w^* = \underset{w}{\operatorname{argmin}} - \sum_n \log P(y = y_n | x_n; w) + r(w)$$

regularization

log probability of all labels given features, assuming that each example is i.i.d.

- Prediction

$$y = 1 \text{ if } w^T x > 0$$

$$P(y = 1|x) = \frac{1}{1+\exp(-w^T x)} = \frac{\exp(w^T x)}{\exp(w^T x)+1}$$

← Binary (two-class) case

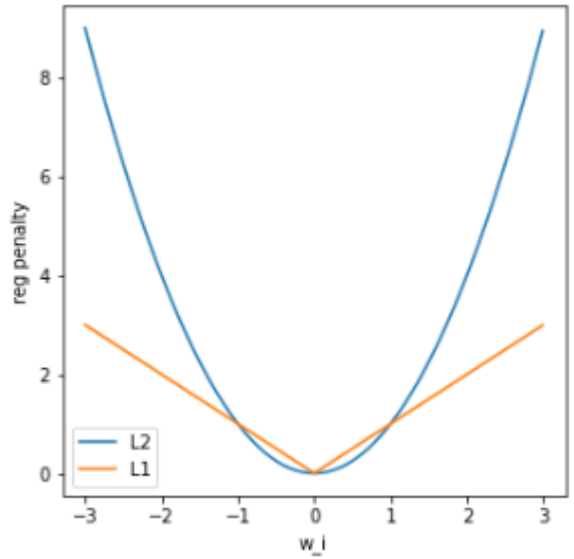
$$P(y = k|x) = \frac{\exp(w_k^T x)}{\sum_j \exp(w_j^T x)}$$

← Multiclass case (one w per class)

Training Logistic Regression

$$w^* = \underset{w}{\text{argmin}} - \sum_n \log P_w(y = y_n | x_n) + r(w)$$

- L2 regularization: $r(w) = \lambda \|w\|_2^2 = \lambda \sum_i w_i^2$
- L1 regularization: $r(w) = \lambda \|w\|_1 = \lambda \sum_i |w_i|$



L2 strongly penalizes really big weights
L1 penalizes increasing the magnitude of big and small weights the same
L1 leads to a sparse weight vector (many zeros) – why?

L1 regularization can be used to select features!

When is regularization absolutely essential?

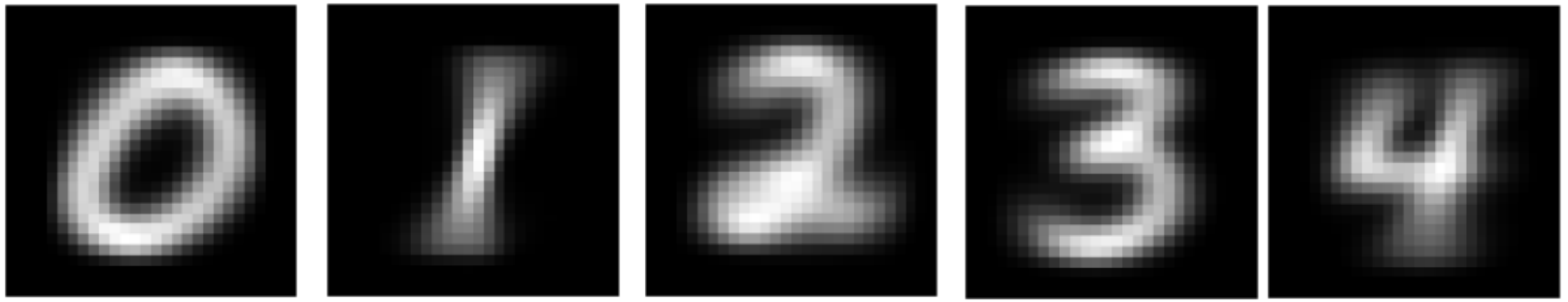
There are many optimizers for L2 and L1 logistic regression. You will want to use a library.

Inspecting weights for digits

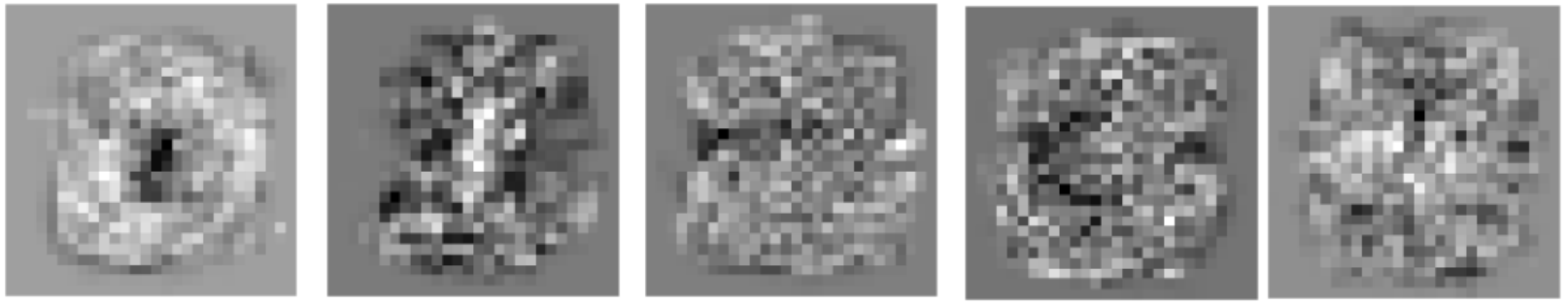
```
model_lr2 = LogisticRegression(max_iter=500, penalty='l2',C=1,verbose=2).fit(x_train[train_indices['m']],y_train[train_indices['m']])
model_lr1 = LogisticRegression(max_iter=500, penalty='l1',C=1,verbose=2,solver='saga').fit(x_train[train_indices['m']],y_train[train_indices['m']])
```

```
for k in np.arange(10):
  print(model_lr1.classes_[k])
  display_mnist(x_train[y_train==k].mean(axis=0))
  display_mnist(model_lr2.coef_[k])
  display_mnist(model_lr1.coef_[k])
```

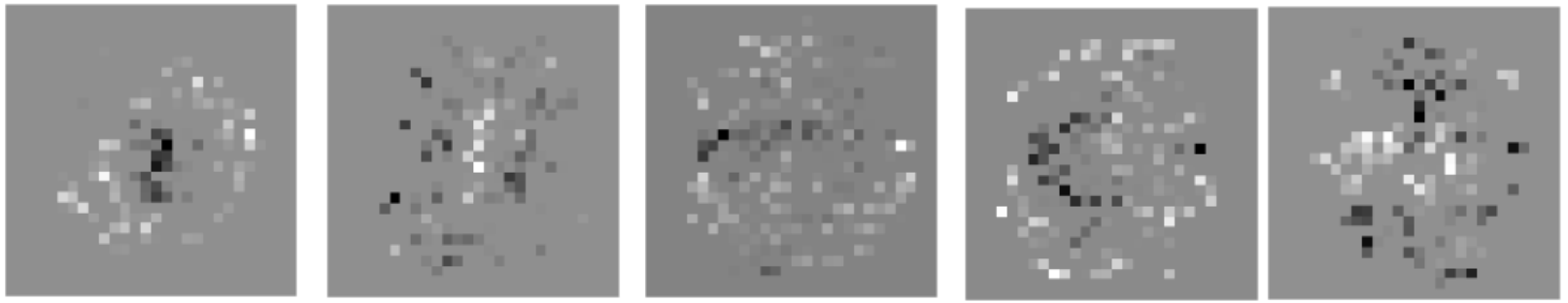
Average Pixels



L2 weights



L1 weights



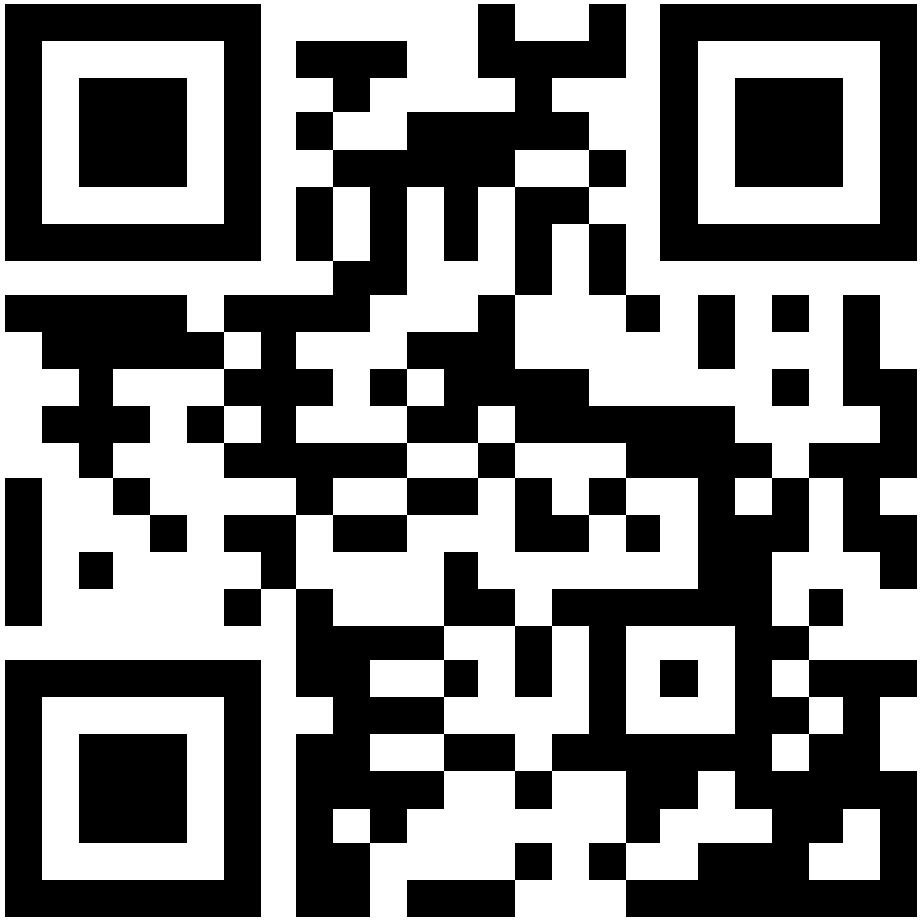
Logistic Regression Summary

- Key Assumptions
  - The log odds ratio $\log \frac{P(y = k|x)}{P(y \neq k|x)}$ can be expressed as a linear combination of features
- Model Parameters
  - One coefficient per feature per class (plus bias term)
- Designs
  - L1 or L2 or elastic (both L1 and L2) regularization weight
- When to Use / Strengths
  - Many features, some of which could be irrelevant or redundant
  - Provides a good estimate of label likelihood
- When Not to Use / Weaknesses
  - Features are low-dimensional (linear function not likely to be expressive enough)

Linear logistic regression is typically the last layer of a classification neural network

Q1

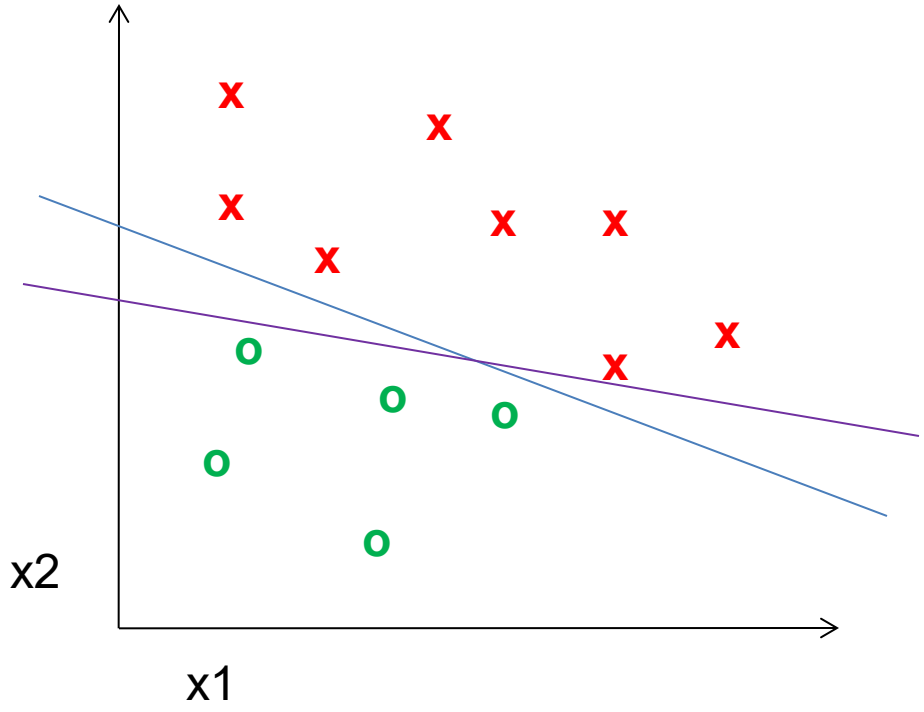
https://tinyurl.com/441-fa24-L7



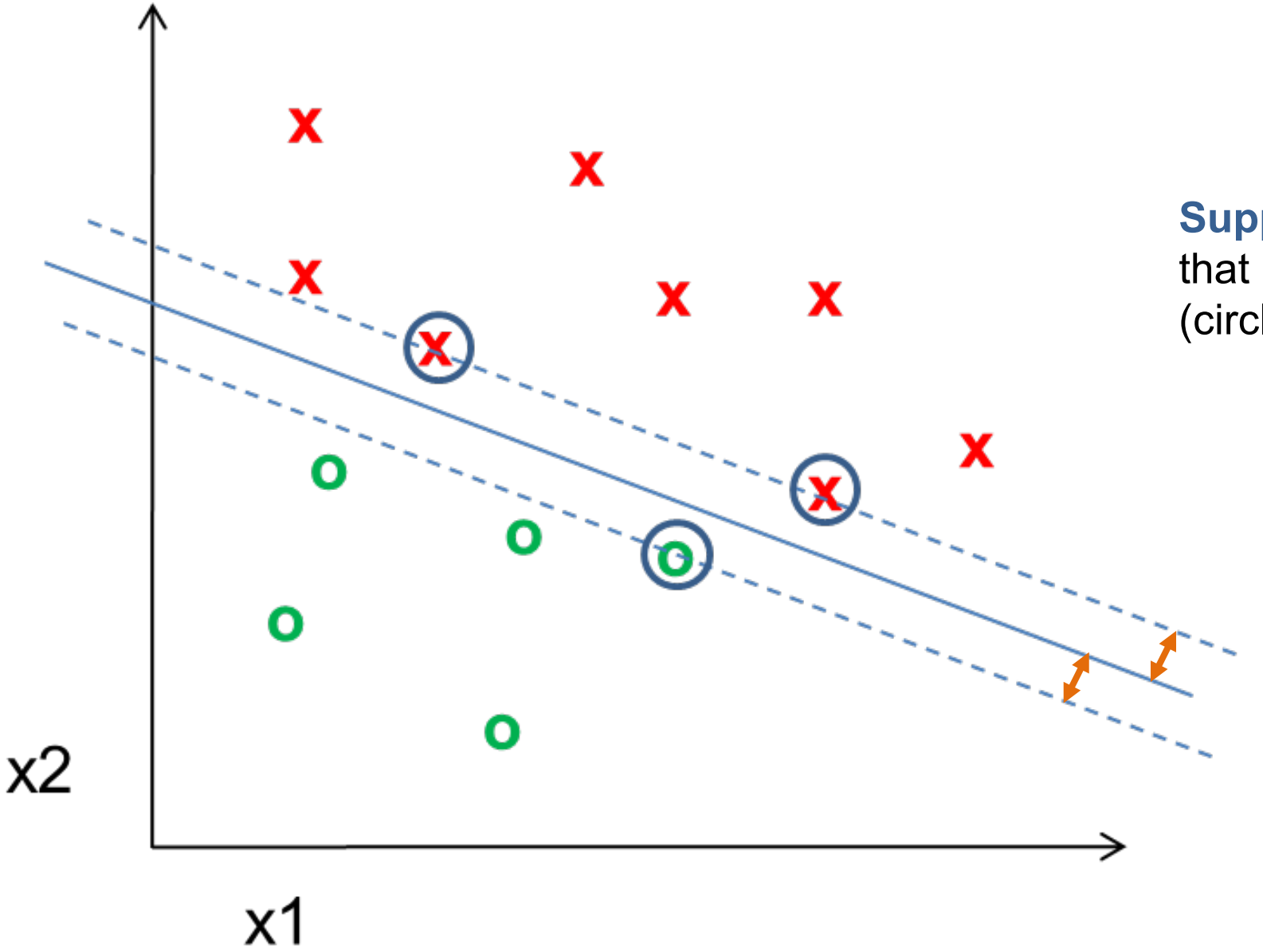
What is the best linear classifier?

- Logistic regression
  - Maximize expected likelihood of true 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



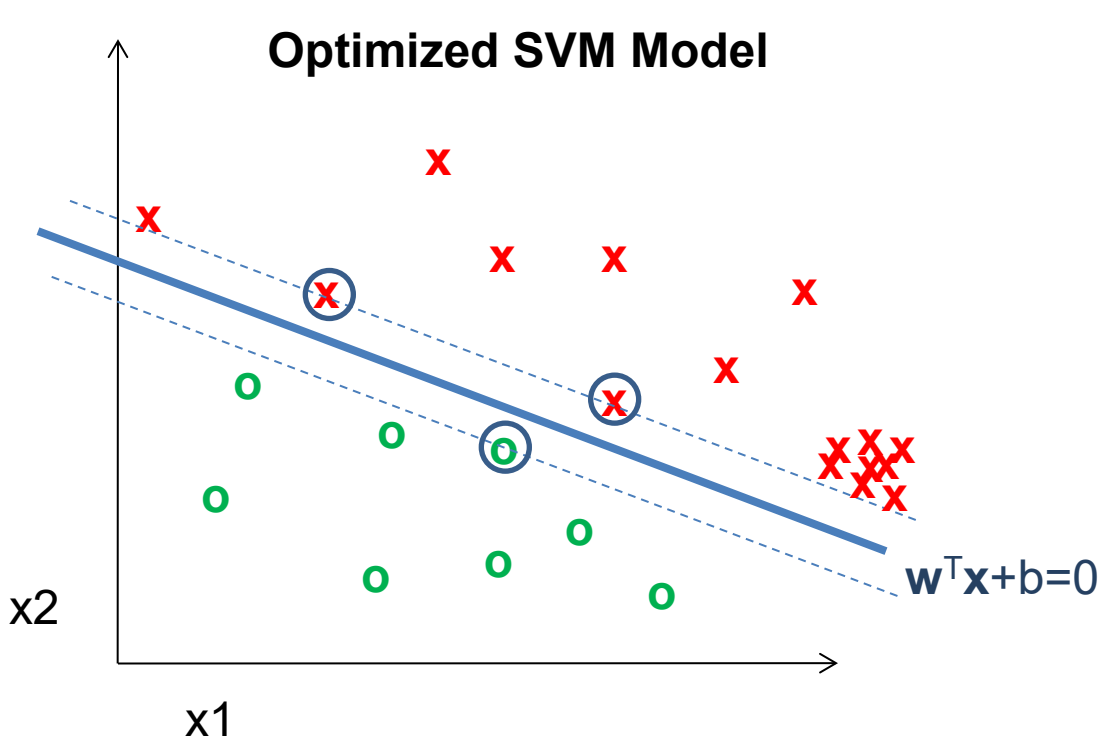
Support Vector: an example that lies on the margin (circled points)

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

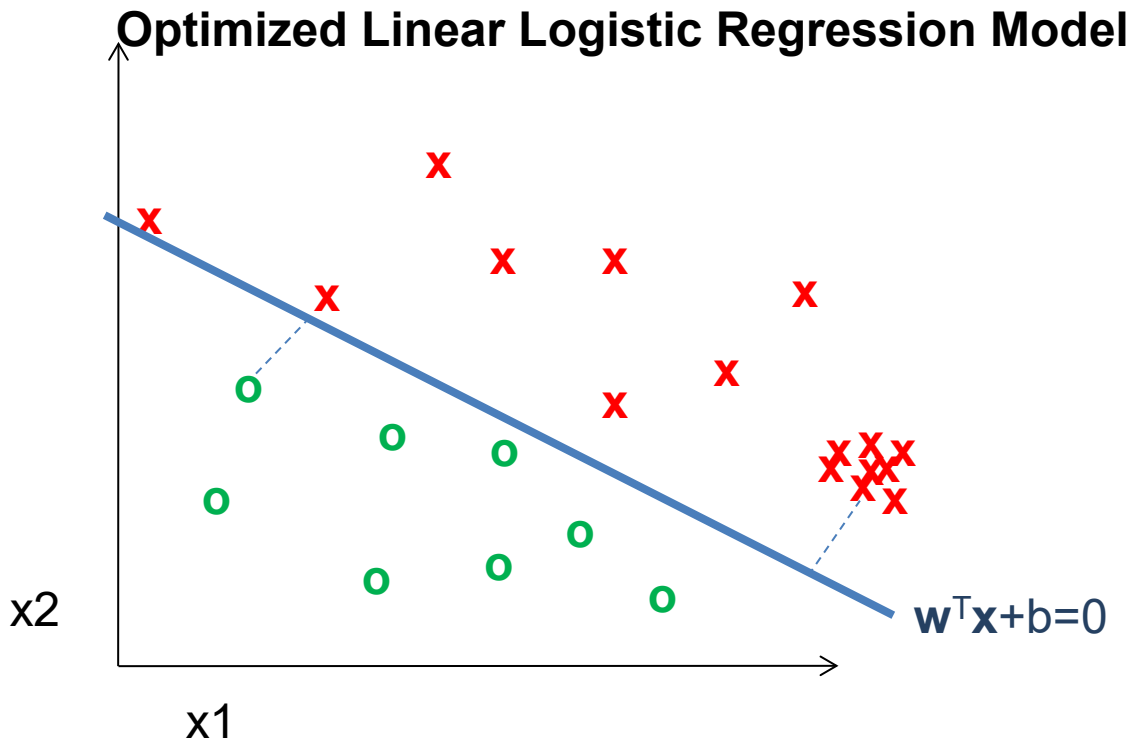
$$m(\boldsymbol{x}) = \frac{y(\boldsymbol{w}^T\boldsymbol{x}+b)}{\|w\|}$$

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

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



Decision boundary depends only on "support vectors" (circled)



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

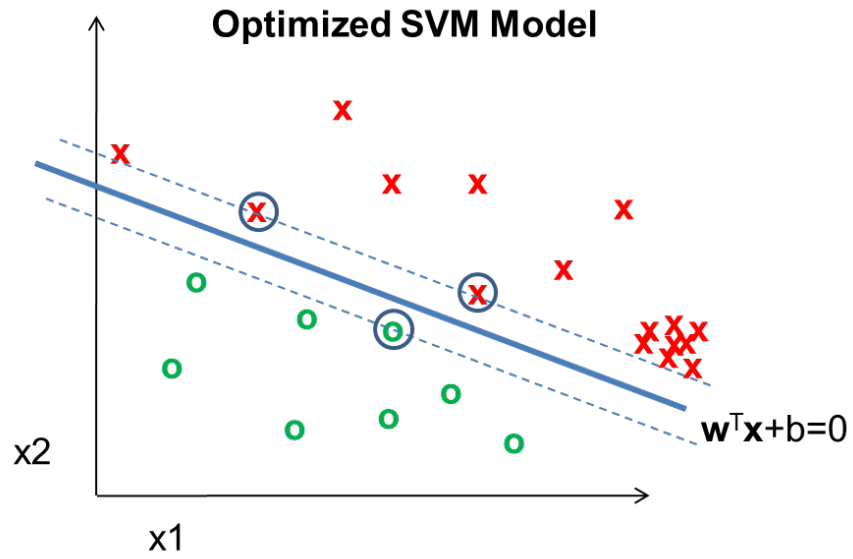
Why SVMs achieve good generalization

- 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

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

Optimization

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

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

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

Optimization

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

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

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

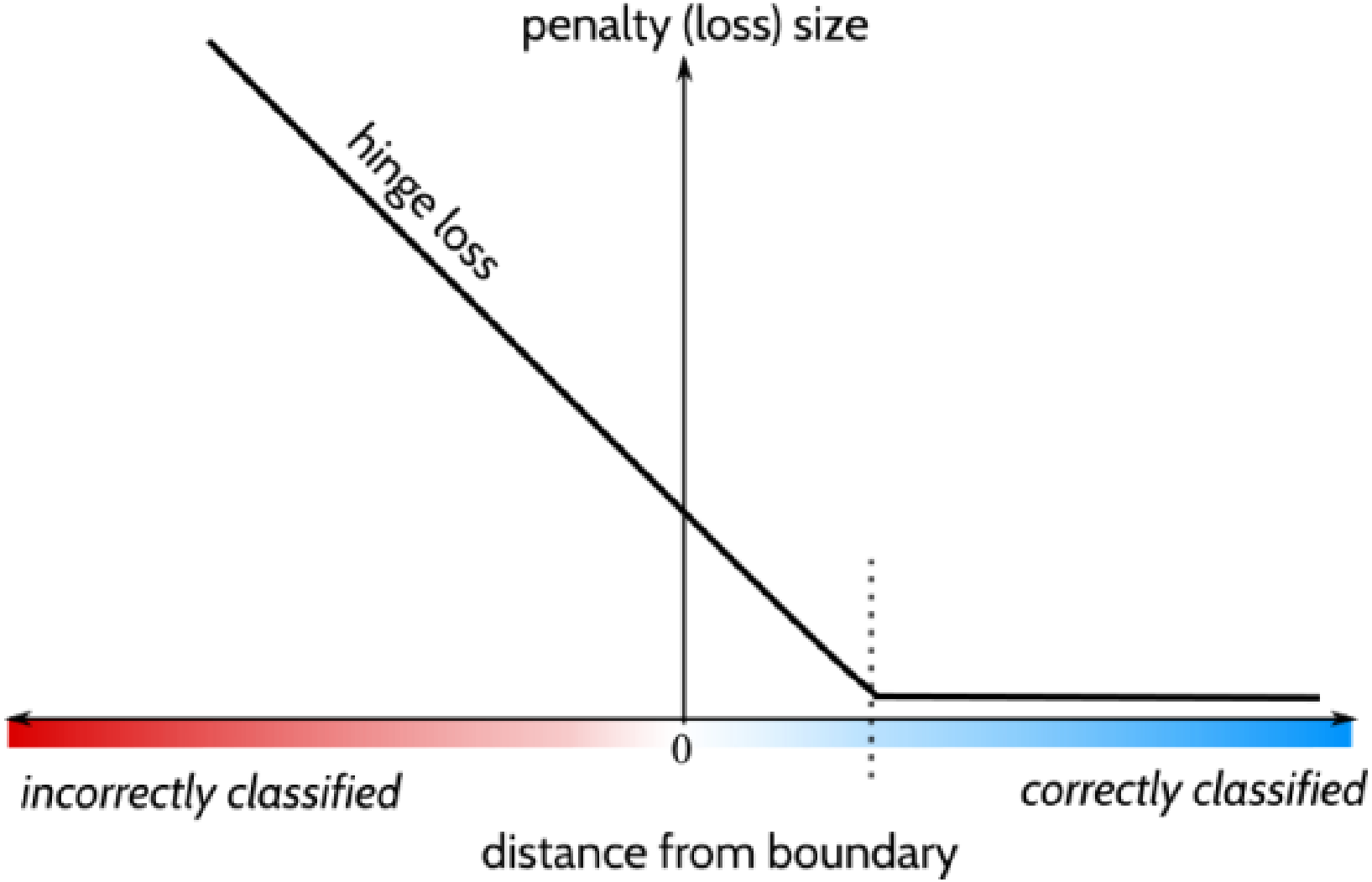
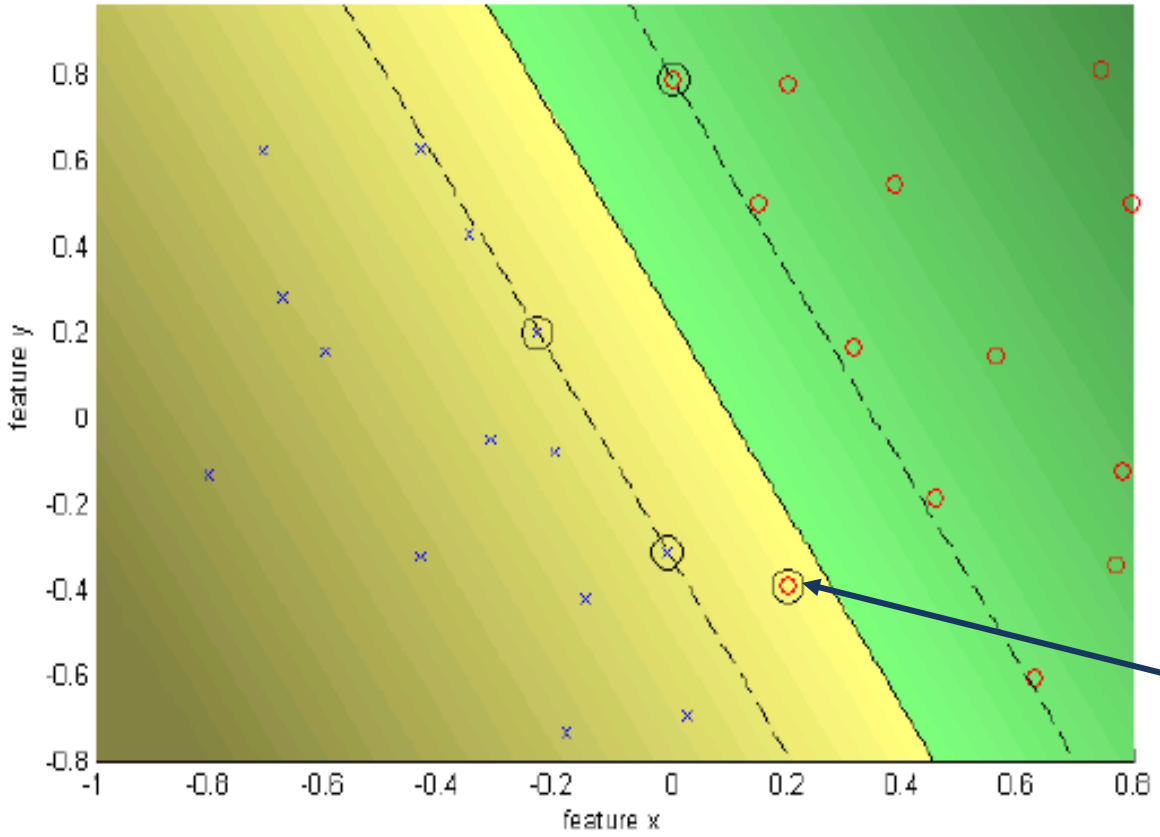

penalty (loss) size
hinge loss
0
incorrectly classified
correctly classified
distance from boundary

Fig source

C = 10 soft margin



Pays a “slack” penalty for violating the margin

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

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

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

So linear SVM is a kind of weighted nearest neighbor with dot product similarity

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

Primal

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

Dual

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

Training Objective

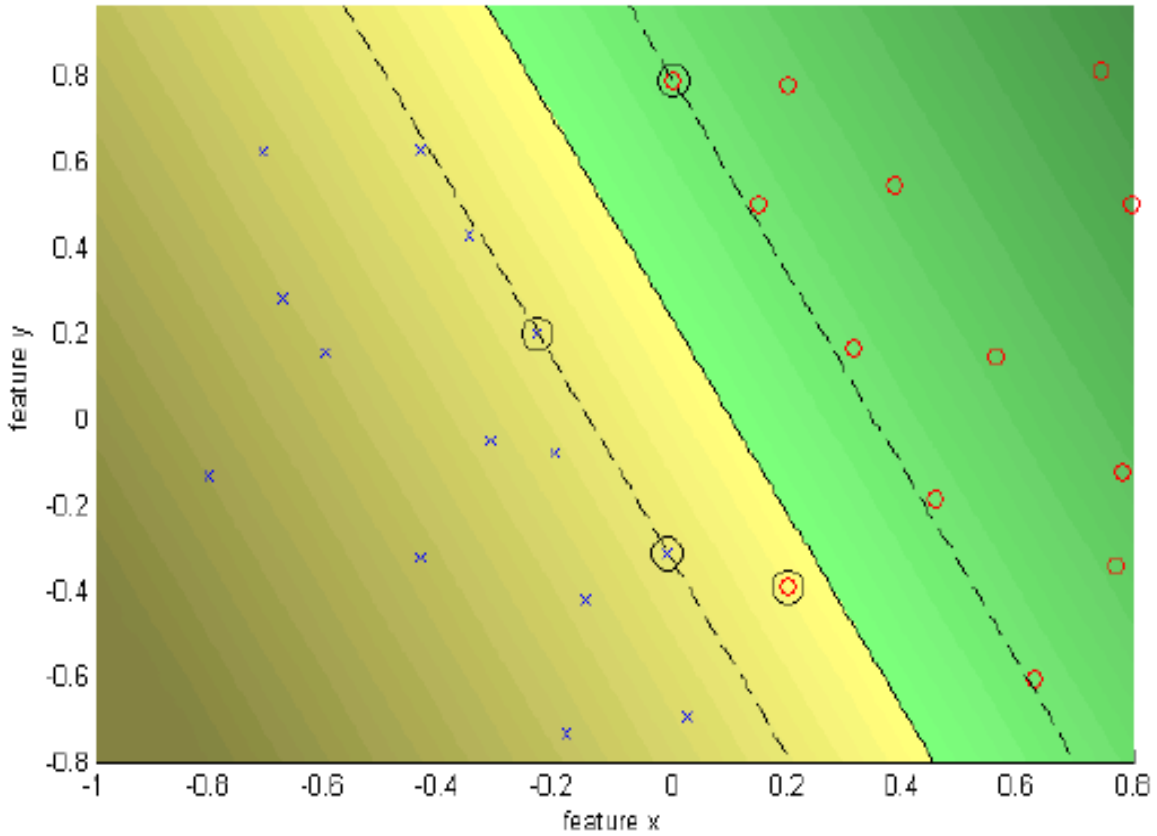
$$w^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left[\|w\|^2 + C \sum_n^N \max(0, 1 - y_n(\boldsymbol{w}^T x_n + b)) \right]$$

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

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

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]

Non-linear SVMs use a different similarity measure, called a “kernel function”, than dot product

- Linear: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j$
- Polynomial: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(1 + \boldsymbol{x}_i^T \boldsymbol{x}_j\right)^d$
- Gaussian: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2})$

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

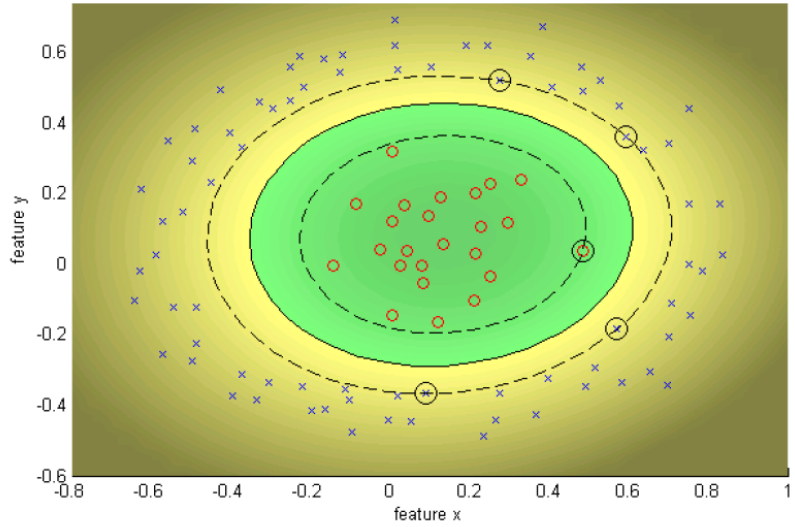
Radial Basis Function (RBF) SVM

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

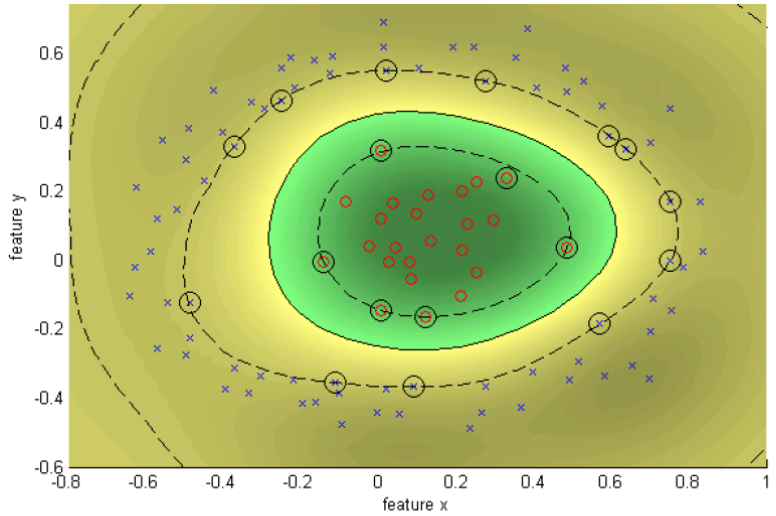
Slide credit: Zisserman [link]

Decreasing sigma makes it more like nearest neighbor

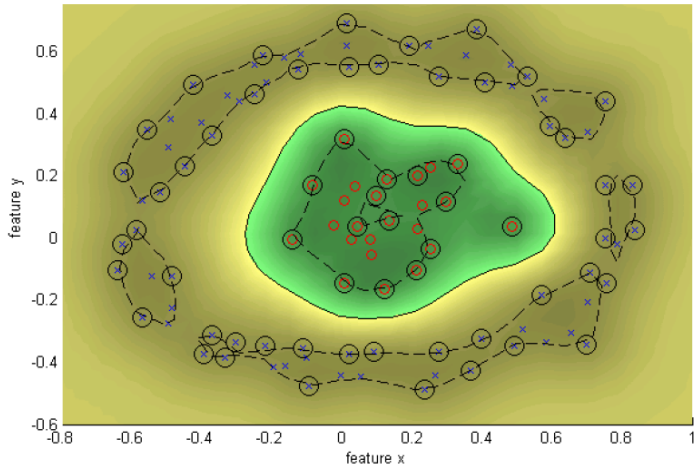
$\sigma = 1.0 \quad C = \infty$



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



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

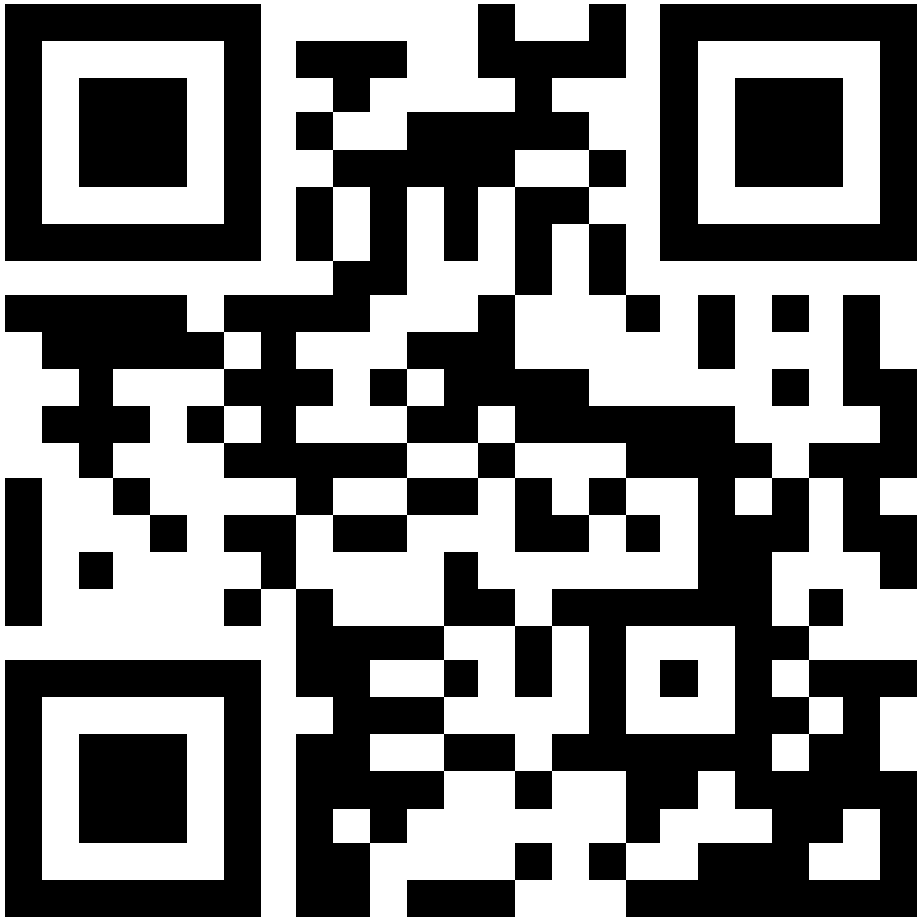


SVM with RBF Kernel Shown

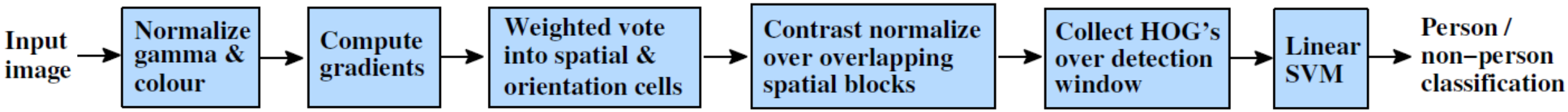
Fig credit: Zisserman [link]

Q2-3

https://tinyurl.com/441-fa24-L7



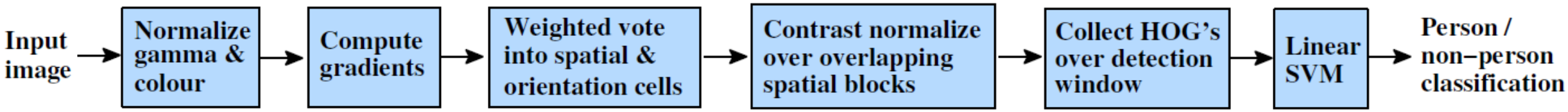
Example application of SVM: Dalal-Triggs 2005



- 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



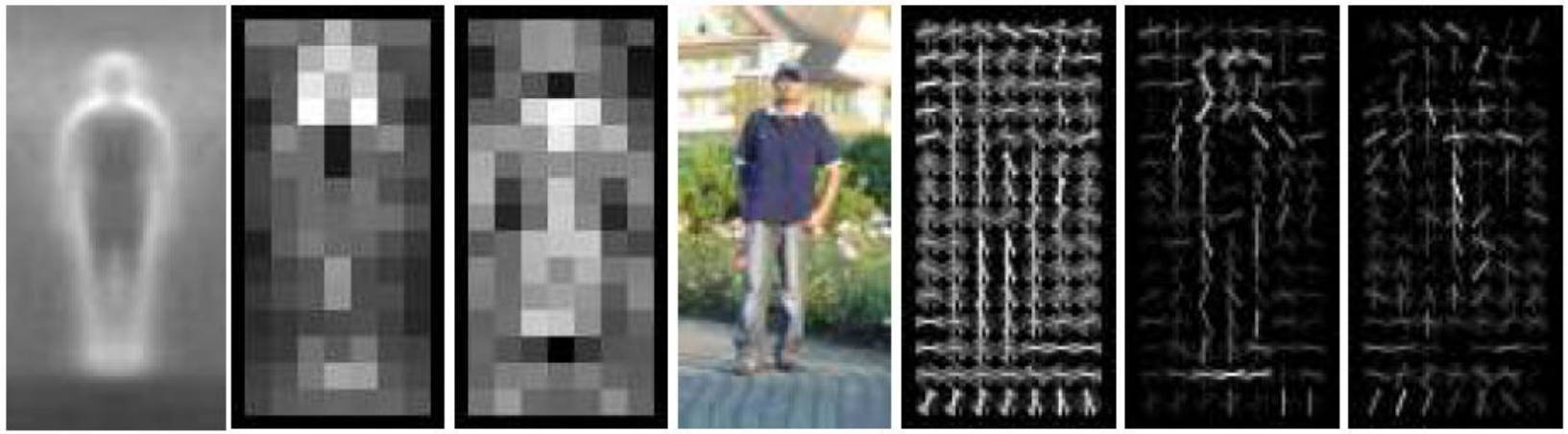
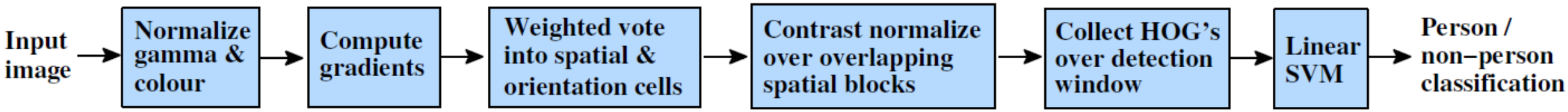
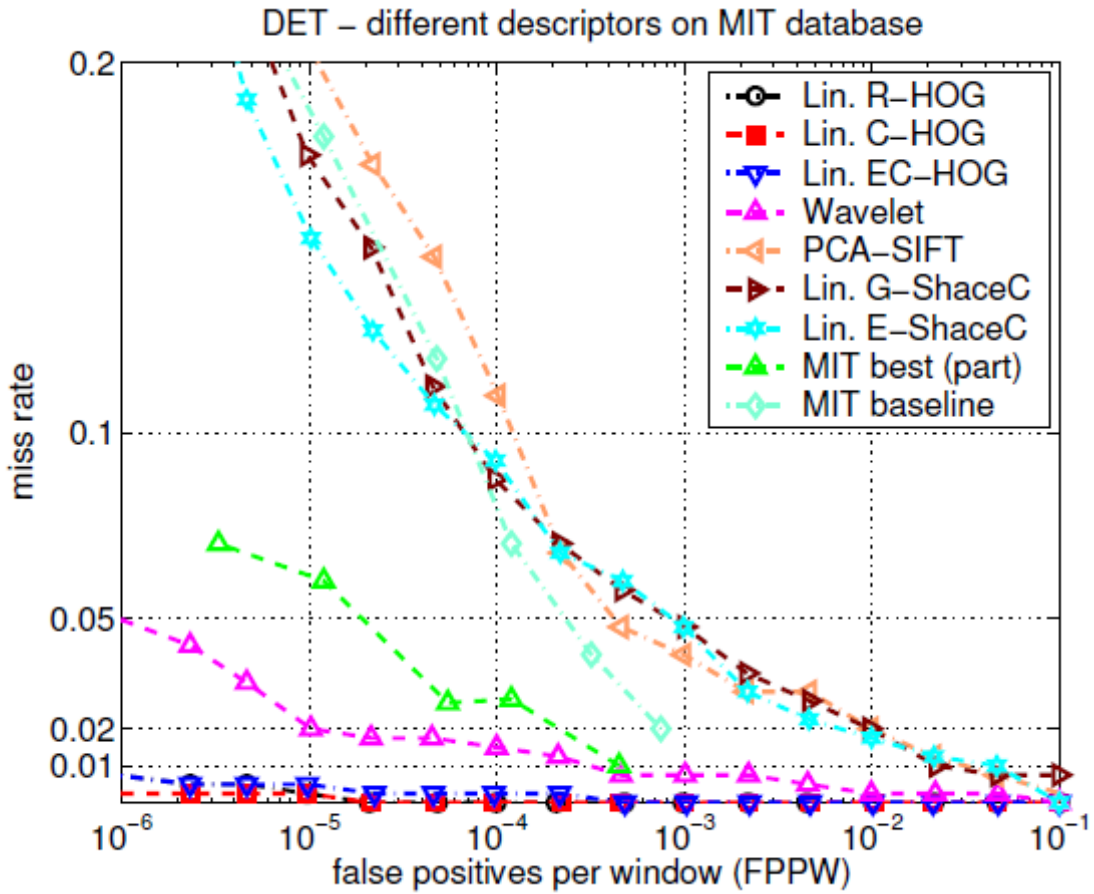


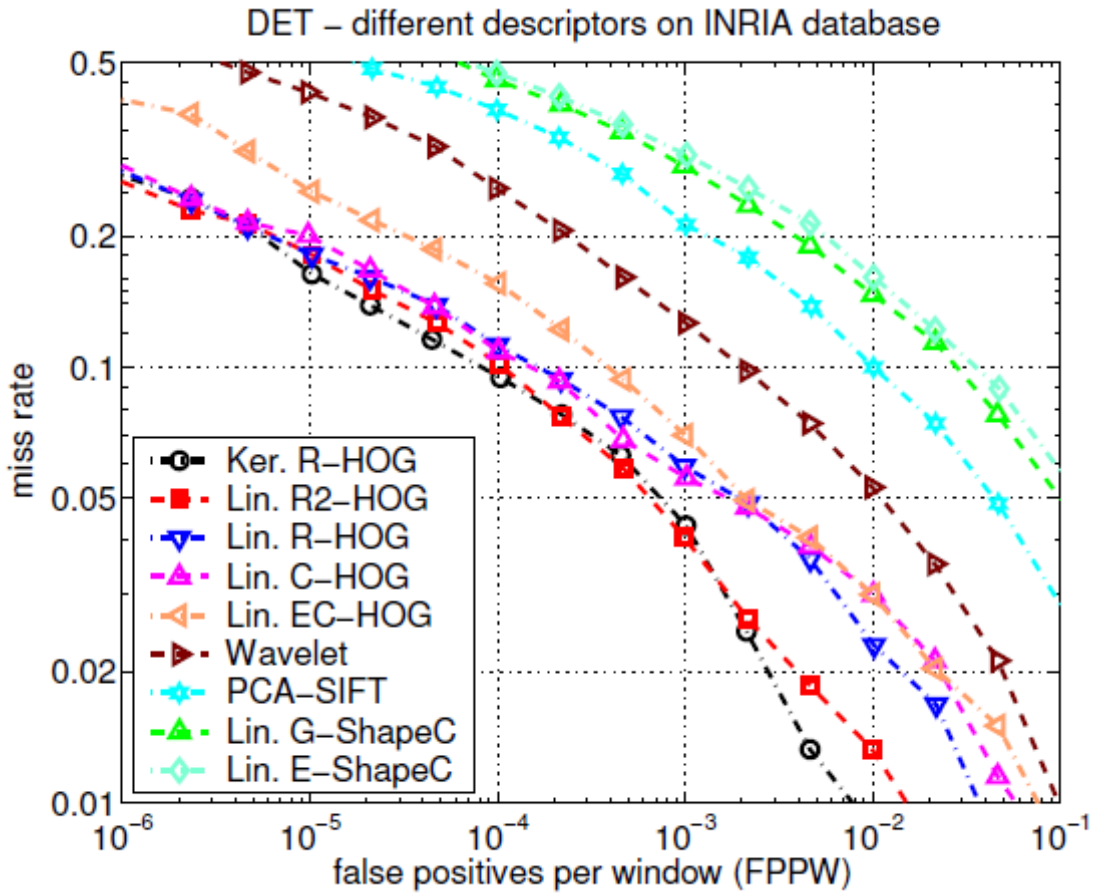
Figure 6. Our HOG detectors cue mainly on silhouette contours (especially the head, shoulders and feet). The most active blocks are centred on the image background just *outside* the contour. (a) The average gradient image over the training examples. (b) Each "pixel" shows the maximum positive SVM weight 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







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!

Recap

- Nearest neighbor is widely used
  - Super-powers: can instantly learn new classes and predict from one or many examples
- Logistic Regression is widely used
  - Super-powers: Effective prediction from high-dimensional features
- Linear Regression is widely used
  - Super-powers: Can extrapolate, explain relationships, and predict continuous values from many variables
- Almost all algorithms involve nearest neighbor, logistic regression, or linear regression
  - The main learning challenge is typically **feature learning**

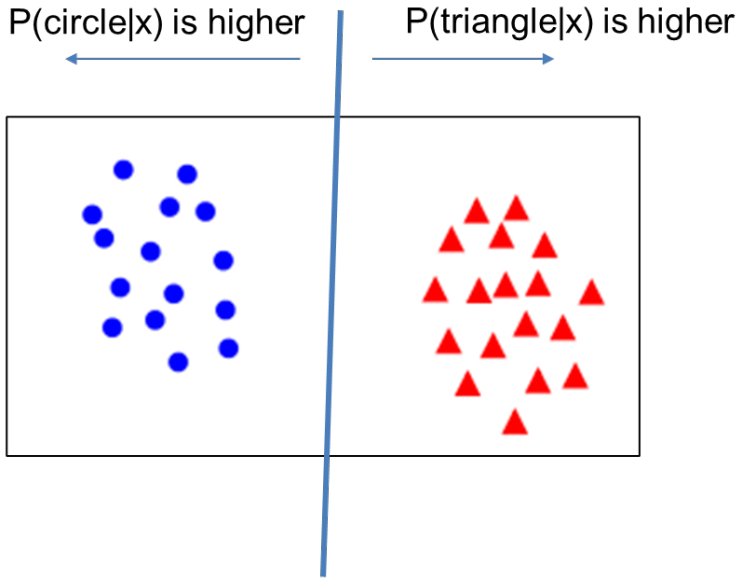
Things to remember

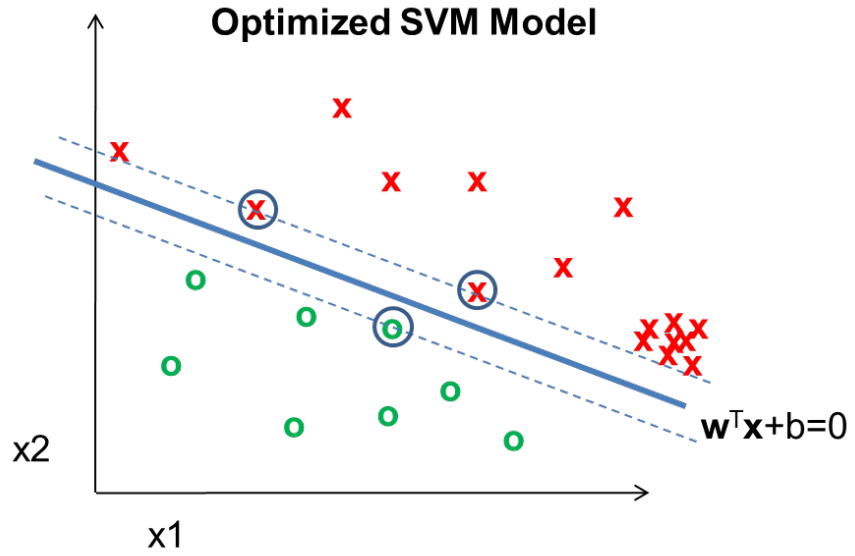
- Linear logistic regression and linear SVM are classification techniques that aims to split features between two classes with a linear model
  - Predict categorical values with confidence

- Logistic regression maximizes confidence in the correct label, while SVM just tries to be confident enough

- Non-linear versions of SVMs can also work well and were once popular (but almost entirely replaced by deep networks)

- Nearest neighbor and linear models are the final predictors of most ML algorithms – the complexity lies in finding features that work well with NN or linear models





Next class

- Naïve Bayes Classifier