Lecture 8
Support Vector Machines

EE-UY 4563/EL-GY 9123: INTRODUCTION TO MACHINE LEARNING

PROF. SUNDEEP RANGAN

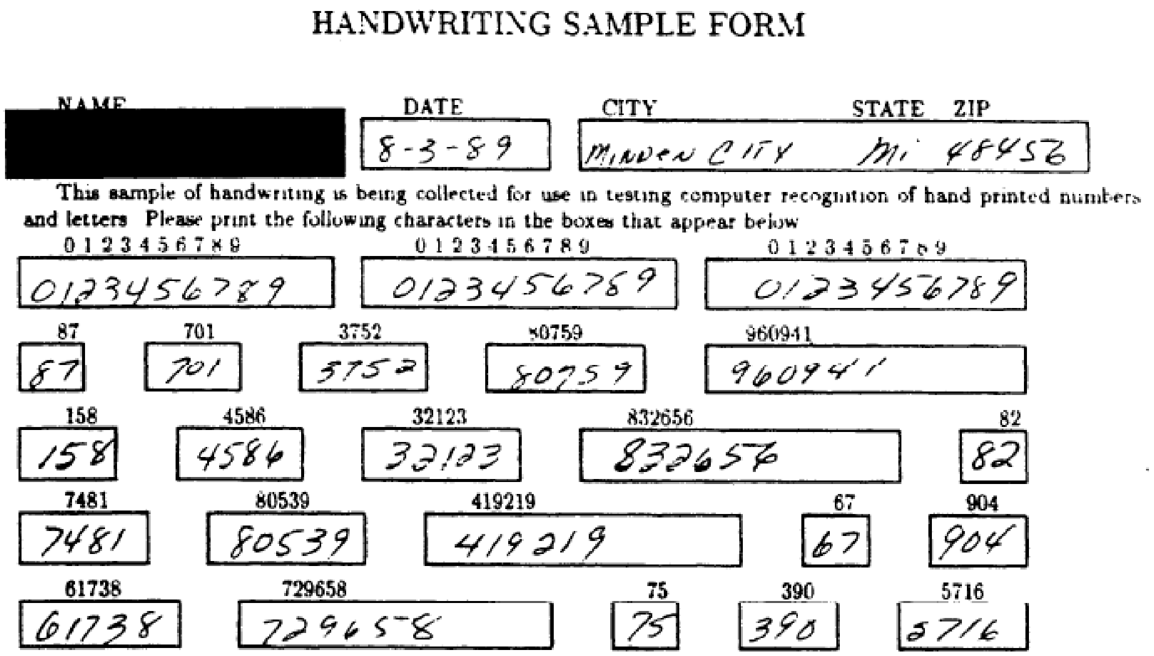
Learning Objectives

- ❑Describe image classification as an classification problem
- ❑Interpret weights in linear classification of images
- ❑Define the margin in linear classification
- ❑Describe the SVM classification problem.
- ❑Write equations for solutions of constrained optimization using the Lagrangian.
- ❑Describe a kernel SVM problem
- ❑Select SVM parameters from cross-validation

Outline

- Motivating example: Recognizing handwritten digits
  - Why logistic regression doesn't work well.
- Maximum margin classifiers
- Support vector machines
- Constrained optimization
- Kernel trick

MNIST Digit Classification


HANDWRITING SAMPLE FORM

NAME [redacted] DATE 8-3-89 CITY MINDEN CITY STATE MI ZIP 48456

This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below.

0 1 2 3 4 5 6 7 8 9 — 0123456789
0 1 2 3 4 5 6 7 8 9 — 0123456789
0 1 2 3 4 5 6 7 8 9 — 0123456789

87 — 87; 701 — 701; 3752 — 3752; 80759 — 80759; 960941 — 960941

158 — 158; 4586 — 4586; 32123 — 32123; 832656 — 832656; 82 — 82

7481 — 7481; 80539 — 80539; 419219 — 419219; 67 — 67; 904 — 904

61738 — 61738; 729658 — 729658; 75 — 75; 390 — 390; 5716 — 5716

- Problem: Recognize hand-written digits
- Originally problem:
  - Census forms
  - Automated processing
- Classic machine learning problem
- Benchmark

From Patrick J. Grother, NIST Special Database, 1995

A Widely-Used Benchmark

- We will look at SVM today
- Not the best algorithm
- But quite good
- ...and illustrates the main points

Classifiers [edit]

This is a table of some of the machine learning methods used on the database and their error rates, by type of classifier:

Type	Classifier	Distortion	Preprocessing	Error rate (%)
Linear classifier	Pairwise linear classifier	None	Deskewing	7.6[9]
K-Nearest Neighbors	K-NN with non-linear deformation (P2DHMDM)	None	Shiftable edges	0.52[14]
Boosted Stumps	Product of stumps on Haar features	None	Haar features	0.87[15]
Non-Linear Classifier	40 PCA + quadratic classifier	None	None	3.3[9]
Support vector machine	Virtual SVM, deg-9 poly, 2-pixel jittered	None	Deskewing	0.56[16]
Neural network	2-layer 784-800-10	None	None	1.6[17]
Neural network	2-layer 784-800-10	elastic distortions	None	0.7[17]
Deep neural network	6-layer 784-2500-2000-1500-1000-500-10	elastic distortions	None	0.35[18]
Convolutional neural network	Committee of 35 conv. net, 1-20-P-40-P-150-10	elastic distortions	Width normalizations	0.23[8]

Loading The Data: 8 x 8 Images

```
from sklearn import datasets, linear_model, preprocessing
digits = datasets.load_digits()
images = digits.images
labels = digits.target
images.shape
```

```
(1797, 8, 8)
```

- Directly in sklearn datasets

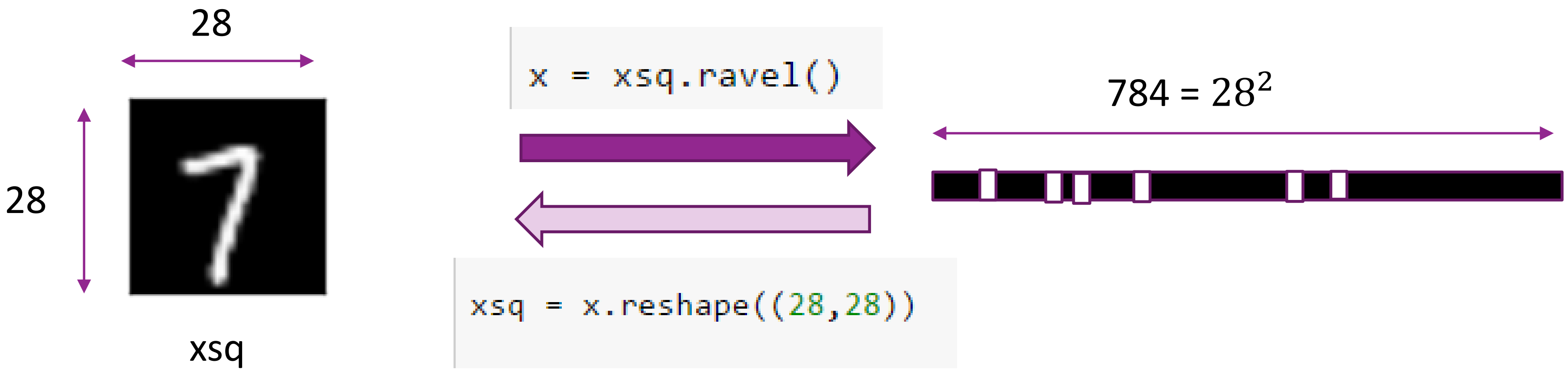
Loading the Data: 28 x 28 Images

```
In [3]: from sklearn.datasets import fetch_mldata
        mnist = fetch_mldata("MNIST original")

In [4]: mnist.data.shape

Out[4]: (70000, 784)
```

- Will look at higher resolution version
- Also, in sklearn
- But needs to download from external site
- 70,000 samples.
- Each image sample is a row

Matrix and Vector Representation

- Images can be represented as 2D matrices or 1D vectors
- Grayscale: Each pixel value is between 0 (black) and 255 (white)



$$S = \mathrm{Mat}(x) = \begin{bmatrix} s_{11} & \cdots & s_{1,28} \\ \vdots & \vdots & \vdots \\ s_{28,1} & \cdots & s_{28,28} \end{bmatrix}$$

$$x = \mathrm{vec}(S) = [x_1 \quad \cdots \quad x_{784}]$$

Displaying Images in Python

4 random images in the dataset

A human can classify these easily

```
def plt_digit(x):
    nrow = 28
    ncol = 28
    xsq = x.reshape((nrow,ncol))
    plt.imshow(xsq,  cmap='Greys_r')
    plt.xticks([])
    plt.yticks([])

# Convert data to a matrix
X = mnist.data
y = mnist.target

# Select random digits
nplt = 4
nsamp = X.shape[0]
Iperm = np.random.permutation(nsamp)

# Plot the images using the subplot command
for i in range(nplt):
    ind = Iperm[i]
    plt.subplot(1,nplt,i+1)
    plt_digit(X[ind,:])
```

Key command

Try a Logistic Classifier

```
y = mnist.target
ntr = 5000
Xtr = X[Iperm[:ntr],:]
ytr = y[Iperm[:ntr]]
```

- Train on 5000 samples
  - To reduce training time.
  - In practice want to train with ~40k
- Select correct solver (lbfgs)
  - Others can be very slow. Even this will take minutes

```
logreg = linear_model.LogisticRegression(verbose=10, multi_class='multinomial', solver='lbfgs', max_iter=500)
logreg.fit(Xtr,ytr)
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.3s finished
```

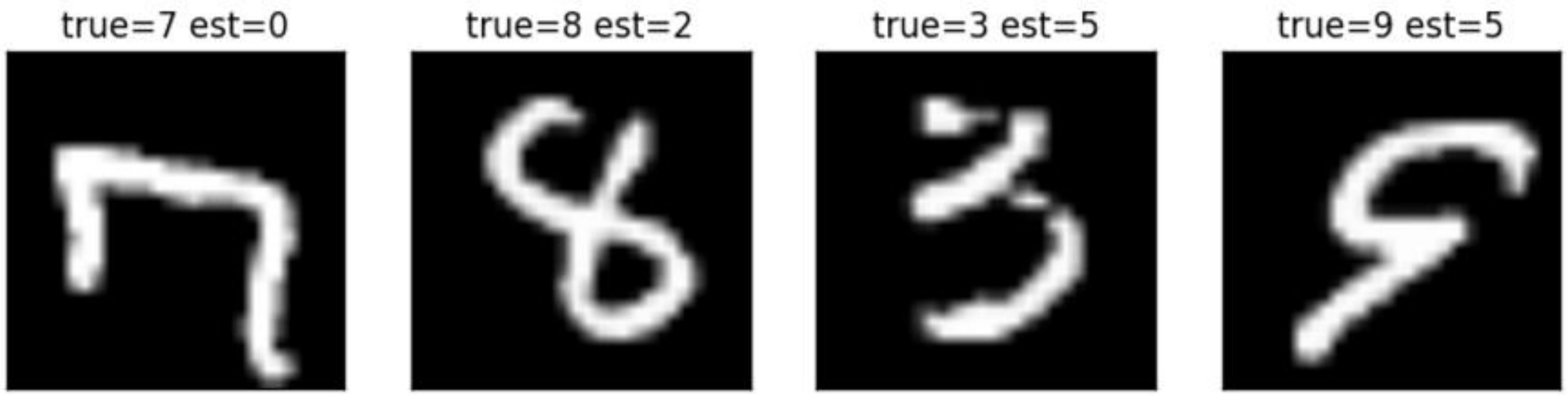
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=500, multi_class='multinomial',
          n_jobs=1, penalty='l2', random_state=None, solver='lbfgs',
          tol=0.0001, verbose=10, warm_start=False)
```

Performance

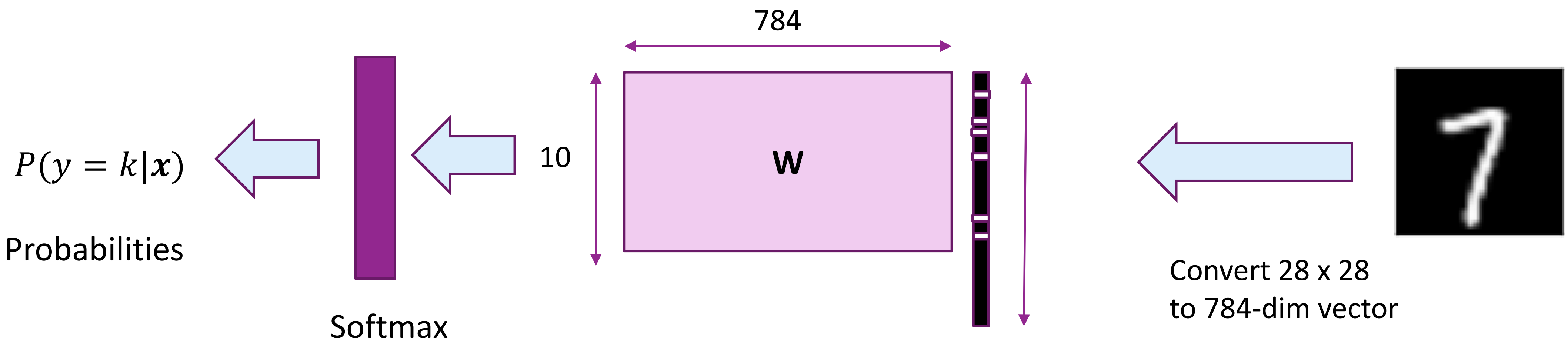
- Accuracy = 85%. Very bad
- Some of the errors seem like they should have been easy to spot
- What went wrong?

```
Xts = X[Iperm[ntr:],:]
yts = y[Iperm[ntr:]]
yhat = logreg.predict(Xts)
acc = np.mean(yhat == yts)
print('Accuaracy = {0:f}'.format(acc))
```

```
Accuaracy = 0.849767
```



Recap: Logistic Classifier



- Will select $\hat{y} = \arg\max_k P(y = k|x) = \arg\max_k z_k$
  - Output z_k which is largest
- When is z_k large?

Interpreting the Logistic Classifier Weights

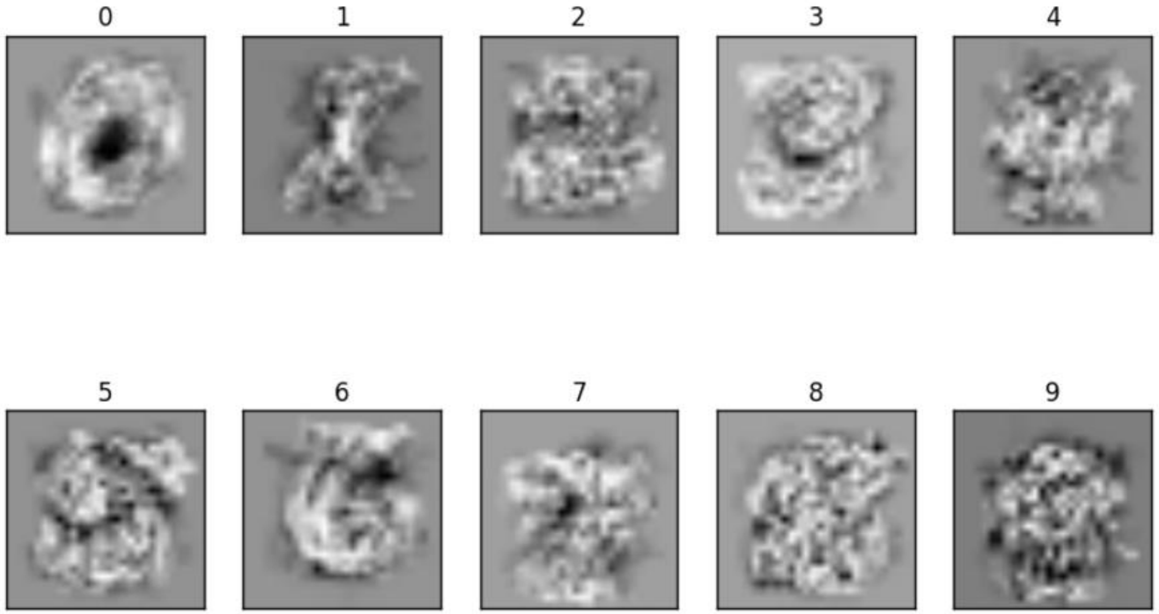
- Suppose $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$. Then: $z_k = \boldsymbol{w}_k^T\boldsymbol{x}$
  - $\boldsymbol{w}_k$ is 784-dim row of W
- When is z_k large?
- Theorem (proof on board): If $\boldsymbol{u}$ is any vector, then

$$\arg\max_{\|\boldsymbol{x}\|=1} \boldsymbol{u}^T\boldsymbol{x} = \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|}$$

- Conclusion: For a given $\|\boldsymbol{x}\|$, $z_k = \boldsymbol{w}_k^T\boldsymbol{x}$ is maximized when $\boldsymbol{x} = \alpha\boldsymbol{w}_k$
  - Output of class k will be large when $\boldsymbol{x}$ is aligned with $\boldsymbol{w}_k$
  - Called the "matched filter" in signal processing

Visualizing the Weights

- ❑ Each class weight can be viewed as an image.
- ❑ Class weight output z_k will be large when it is aligned with $\boldsymbol{w}_k$



Optimized weights for logistic classifier

Why are they blurry?

Problems with Logistic Classifier

- Linear weighting cannot capture many deformities in image
  - Rotations
  - Translations
  - Variations in relative size of digit components
- Can be improved with preprocessing
  - E.g. deskewing, contrast normalization, many methods
- Is there a better classifier?

Outline

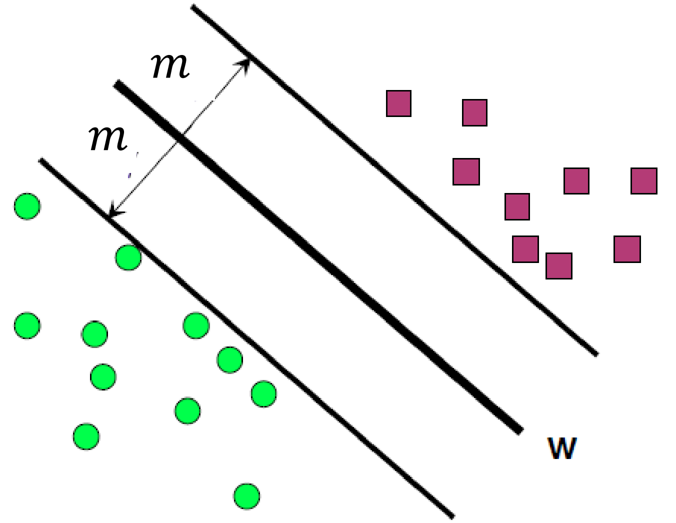
- Motivating example: Recognizing handwritten digits
  - Why logistic regression doesn't work well.
- Maximum margin classifiers
- Support vector machines
- Constrained optimization
- Kernel trick

Recap: Linear Separability and Margin

- Given training data $(\boldsymbol{x}_i, y_i), i = 1, \dots, N$
- Binary class label: $y_i = \pm 1$
- Perfectly linearly separable if there exists a $\boldsymbol{\theta} = (b, w_1, \dots, w_d)$ s.t.:
  - $b + w_1 x_{i1} + \cdots w_d x_{id} > \gamma$ when $y_i = 1$
  - $b + w_1 x_{i1} + \cdots w_d x_{id} < -\gamma$ when $y_i = -1$
- $\boldsymbol{w}$ is the separating hyperplane, m is the margin
- b is the bias
- Single equation form:

$$y_i(b + w_1 x_{i1} + \cdots w_d x_{id}) > \gamma \text{ for all } i = 1, \dots, N$$

$$\mathrm{m} = \frac{\gamma}{\|\boldsymbol{w}\|}$$



Maximum Margin Classifier

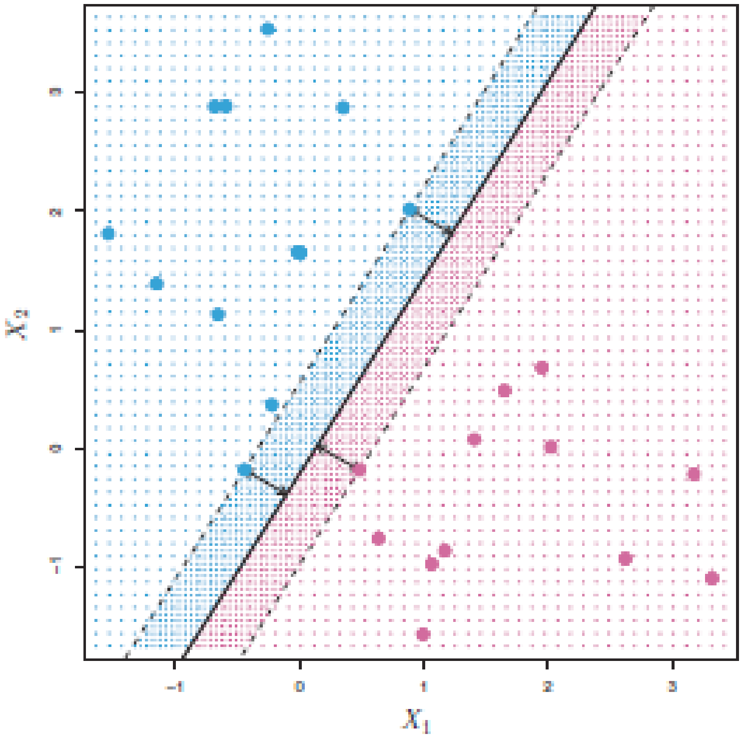
- Suppose data is perfectly linearly separable.
- Define maximum margin classifier

  $\max_{w,\gamma} \gamma$ — Maximizes the margin

  - Such that $y_i(b + \boldsymbol{w}^T\boldsymbol{x}) \geq \gamma \quad \text{for all } i$ — Ensures all points are correctly classified
  - $\sum_{j=1}^{d} w_j^2 \leq 1$ — Scaling on weights

- Called a constrained optimization
  - Objective function and constraints
  - More on this later.

Visualizing Maximum Margin Classifier



- Fig. 9.3 of ISL
- Margin determined by closest points to the line
- In this figure, there are 3 points at the margin

Problems with MM classifier

- Data is often not perfectly separable
  - Only want to correctly separate most points

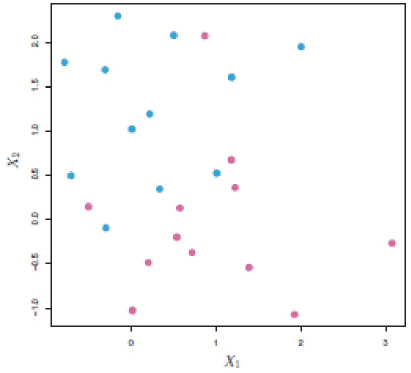


Fig. 9.4

- MM classifier is not robust
  - A single sample can radically change line

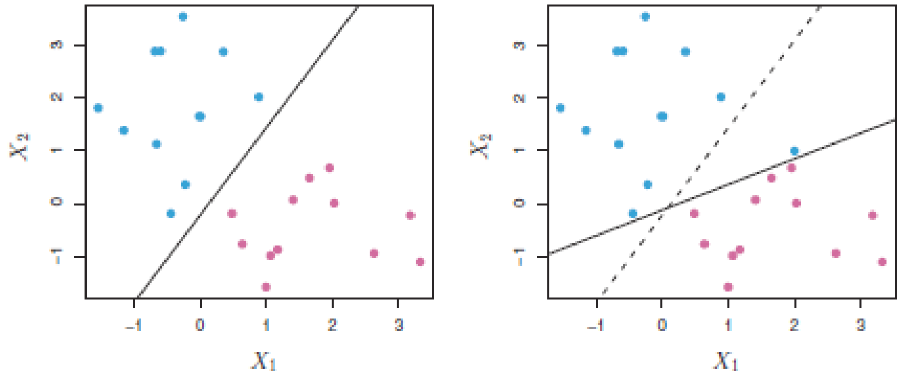


Fig. 9.5

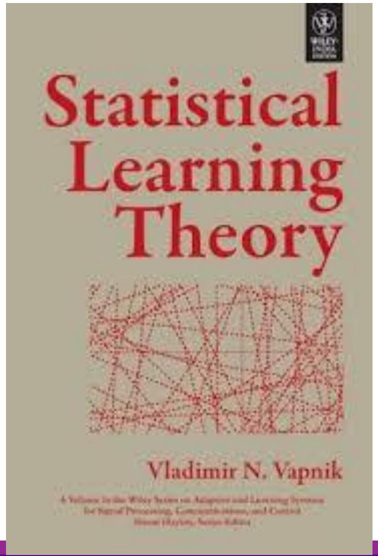
Outline

- Motivating example: Recognizing handwritten digits
  - Why logistic regression doesn’t work well.
- Maximum margin classifiers
- Support vector machines
- Constrained optimization
- Kernel trick

Support Vector Machine

- Support Vector Machine (SVM)
  - Vladimir Vapnik, 1963
  - But became widely-used with kernel trick, 1993
  - More on this later

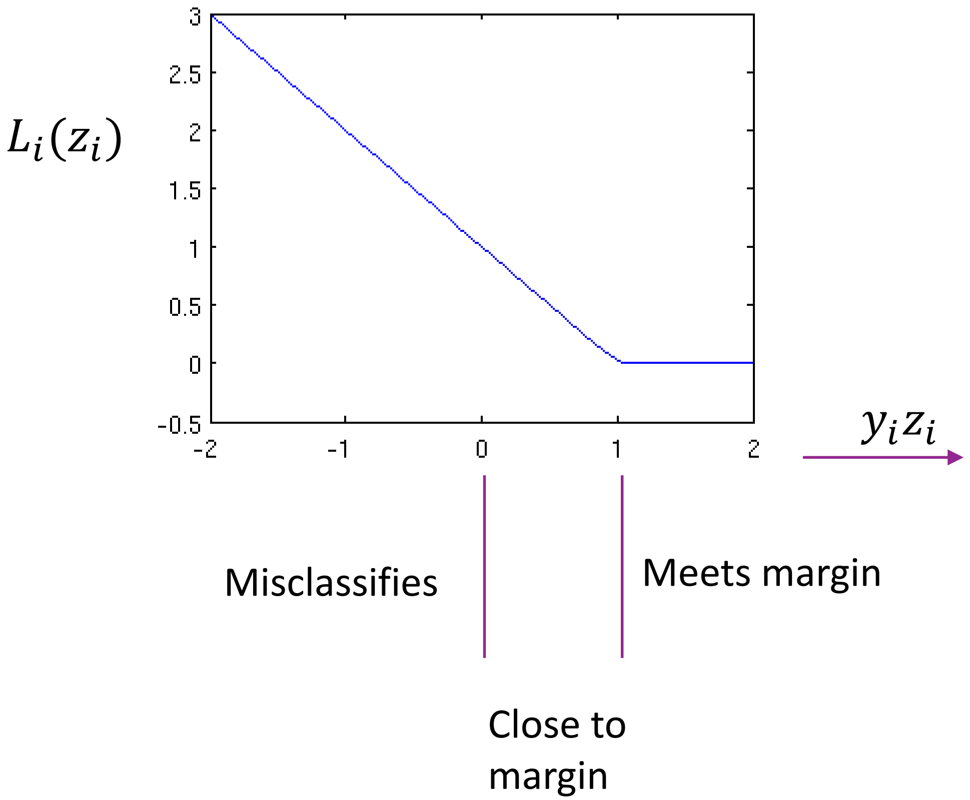
- Got best results on character recognition



- Key idea: Allow “slack” in the classification

Hinge Loss

- Fix $\gamma = 1$
- Want ideally: $y_i(\boldsymbol{w}^T\boldsymbol{x} + b) \geq 1$ for all samples i
  - Equivalently, $y_i z_i \geq 1, \;\; z_i = b + \boldsymbol{w}^T\boldsymbol{x}$
- But, perfect separation may not be possible
- Define hinge loss or soft margin:
  - $L_i(\mathbf{w}, b) = \max(0, 1 - y_i z_i)$
- Starts to increase as sample is misclassified:
  - $y_i z_i \geq 1 \;\Rightarrow$ Sample meets margin target, $L_i(w) = 0$
  - $y_i z_i \in [0,1) \;\Rightarrow$ Sample margin too small
  - $y_i z_i \leq 0 \;\Rightarrow$ Sample misclassified



SVM Optimization

- Given data $(\boldsymbol{x}_i, y_i)$
- Optimization $\min_{w,b} J(\boldsymbol{w}, b)$

$$J(\boldsymbol{w}, b) = C \sum_{i=1}^{N} \max(0, 1 - y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)) + \frac{1}{2} \|\boldsymbol{w}\|^2$$

Hinge loss terms
Attempts to reduce
Misclassifications

Controls margin

$1/\|\boldsymbol{w}\|$

- Constant $C > 0$ will be discussed below
- Note: ISL book uses different naming conventions.
  - We have followed convention in sklearn

Alternate Form of SVM Optimization

- Equivalent optimization:

$$\min J_1(\boldsymbol{w}, b, \boldsymbol{\epsilon}), \qquad J_1(\boldsymbol{w}, b, \boldsymbol{\epsilon}) = C\sum_{i=1}^{N} \epsilon_i + \frac{1}{2}\|\boldsymbol{w}\|^2$$

- Subject to constraints:

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \epsilon_i \ \text{ for all } i = 1, \ldots, N$$

  - ϵ_i = amount sample i misses margin target

- Sometimes write as $J_1(\boldsymbol{w}, b, \boldsymbol{\epsilon}) = C\|\boldsymbol{\epsilon}\|_1 + \frac{1}{2}\|\boldsymbol{w}\|^2$

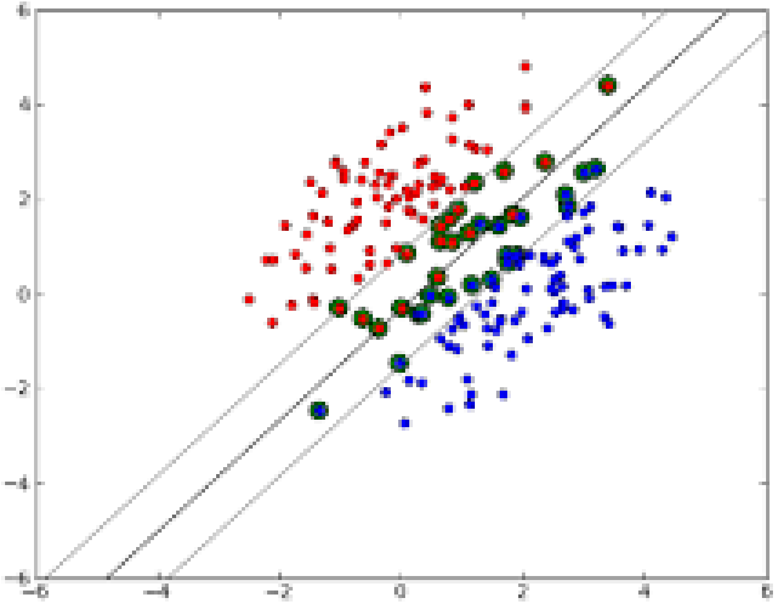
  - $\|\boldsymbol{\epsilon}\|_1 = \sum_{i=1}^{N} \epsilon_i$ called the "one-norm"

Interpreting Parameters

- Margin is $1/\|\boldsymbol{w}\|$
- Parameter ϵ_i called the slack variable
  - $\epsilon_i = 0 \Rightarrow$ Sample on correct side of margin
  - $\epsilon_i \geq 0 \Rightarrow$ Sample violates the margin
  - $\epsilon_i \geq 1 \Rightarrow$ Sample misclassified (wrong side of hyperplane)
- Parameter C:
  - C large: Forces minimum number of violations. Highly fit to data. Low bias, higher variance
  - C small: Enables more samples violations. Higher bias, lower variance
  - Found by cross-validation

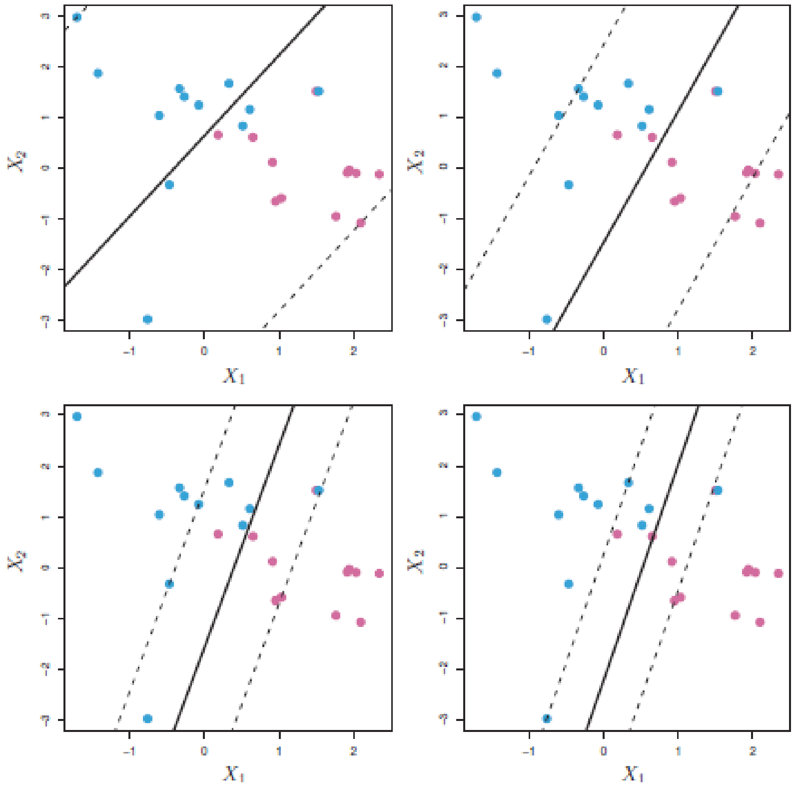
Support Vectors

- Support vectors: Samples that either:
  - Are exactly on margin: $y_{\mathrm{i}}(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1$
  - Or, on wrong side of margin: $y_{\mathrm{i}}(\boldsymbol{w}^T\boldsymbol{x}_i + b) \leq 1$
- Changing samples that are not SVs
  - Does not change solution
  - Provides robustness



Illustrating Effect of C

Low $C = C_1$ $\quad$ C_2



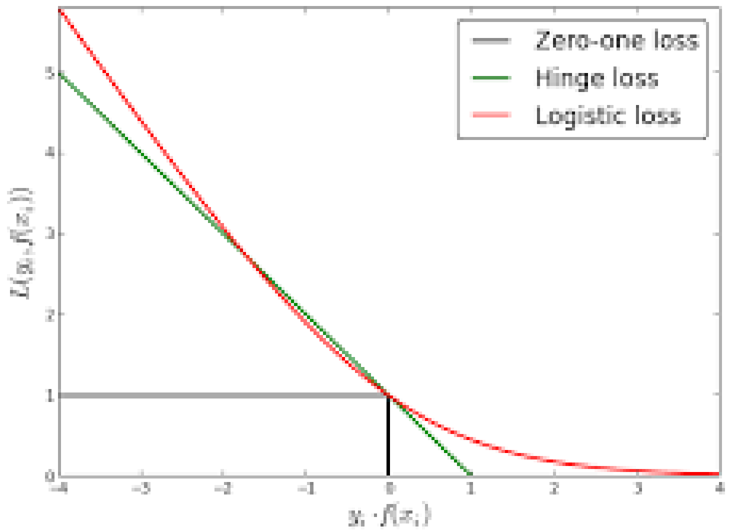
C_3 $\quad$ High C_4

- Fig. 9.7 of ISL
  - Note: C has opposite meaning in ISL than python
  - Here, we use python meaning
- Low C:
  - Many violations of margin.
  - Many more SVs
  - Reduces variance by using more samples

Relation to Logistic Regression

- Logistic regression also minimizes a loss function (proof on board):

$$J(\boldsymbol{w}, b) = \sum_{i=1}^{N} L_i(w, b)\,, \qquad L_i(w, b) = \ln P(y_i|\boldsymbol{x}_i) = -\ln(1 + e^{-y_i z_i})$$



Outline

- Motivating example: Recognizing handwritten digits
  - Why logistic regression doesn’t work well.
- Maximum margin classifiers
- Support vector machines
- Constrained optimization
- Kernel trick

Constrained Optimization

- In many problems, variables are constrained
- Constrained optimization formulation:
  - Objective: Minimize $f(\boldsymbol{w})$
  - Constraints: $g_1(\boldsymbol{w}) \leq 0, \dots, g_M(\boldsymbol{w}) \leq 0$
- Examples:
  - Minimize the mpg of a car subject to a cost or meeting some performance
  - In ML: weight vector may have constraints from physical knowledge
- Often write constraints in vector form: Write $g(\boldsymbol{w}) \leq 0$

$$g(\boldsymbol{w}) = [g_1(\boldsymbol{w}), \dots, g_m(\boldsymbol{w})]^T$$

Lagrangian

- Constrained optimization $f(\boldsymbol{w})$ s.t. $g(\boldsymbol{w}) \leq 0$
- Consider first a single constraint: $g(\boldsymbol{w})$ is a scalar
- Define Lagrangian: $L(\boldsymbol{w}, \lambda) = f(\boldsymbol{w}) + \lambda g(\boldsymbol{w})$
  - $\boldsymbol{w}$ is called the primal variable
  - λ is called the dual variable
- Dual minimization: Given a dual parameter λ, minimize

$$\widehat{\boldsymbol{w}}(\lambda) = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}, \lambda), \qquad L^*(\lambda) = \min_{w} L(\boldsymbol{w}, \lambda)$$

  - Minimizes a weighted combination of objection and constraint.
  - Higher $\lambda \Rightarrow$ Weight constraint more (try to make $g(\boldsymbol{w})$ smaller)
  - Lower $\lambda \Rightarrow$ Weight objective more (try to make $f(\boldsymbol{w})$ smaller)

KKT Conditions

- Given objective $f(\boldsymbol{w})$ and constraint $g(\boldsymbol{w})$
- KKT Conditions: $\hat{\boldsymbol{w}}, \hat{\lambda}$ satisfy:
  - $\hat{\boldsymbol{w}}$ minimizes the Lagrangian: $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}, \hat{\lambda})$
  - Either
    - $g(\hat{\boldsymbol{w}}) = 0$ and $\hat{\lambda} \geq 0$ [active constraint]
    - $g(\hat{\boldsymbol{w}}) < 0$ and $\hat{\lambda} = 0$ [inactive constraint]
- Theorem: Under some technical conditions,
  - if $\hat{\boldsymbol{w}}, \hat{\lambda}$ are local mimima of the constrained optimization, they must satify KKT conditions

General Procedure for Single Constraint

- Suppose:
  - $\boldsymbol{w} = (w_1, \dots, w_d)^T$: d unknown primal variables
  - $g(\boldsymbol{w}) \leq 0$: scalar constraint
- Case 1: Assume constraint is active:
  - Solve $\boldsymbol{w}$ and λ: $\partial L(\boldsymbol{w}, \lambda)/\partial w_j = 0$ and $g(\boldsymbol{w}) = 0$
  - $d + 1$ unknowns and $d + 1$ equations
  - Verify that $\lambda \geq 0$
- Case 2: Assume constraint is inactive
  - Solve primal objective $\partial f(\boldsymbol{w})/\partial w_j = 0$ ignoring constraint
  - d unknowns and d equations
  - Verify that constraint is satisfied: $g(\boldsymbol{w}) \leq 0$

KKT Conditions Illustrated

- Example 1: Constraint is "active"

$$\min_{w} w^2 \quad s.t.\ w+1 \leq 0$$

- Example 2: Constraint is "inactive"

$$\min_{w} w^2 \quad s.t.\ w-1 \leq 0$$

- Examples worked on board

Multiple Constraints

- Now consider constraint: $g(\boldsymbol{w}) = [g_1(\boldsymbol{w}), \dots, g_M(\boldsymbol{w})]^T \leq 0.$
- Lagrangian is:

$$L(\boldsymbol{w}, \boldsymbol{\lambda}) = f(\boldsymbol{w}) + \boldsymbol{\lambda}^T g(\boldsymbol{w}) = f(\boldsymbol{w}) + \sum_{m=1}^{M} \lambda_m \, g_m(\boldsymbol{w})$$

  - Weighted sum of all M constraints
  - $\boldsymbol{\lambda}$ is called the dual vector

- KKT conditions extend to:
  - $\widehat{\boldsymbol{w}}$ minimizes the Lagrangian: $\widehat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}, \hat{\lambda})$
  - For each $m = 1, \dots, M$
    - $g_m(\widehat{\boldsymbol{w}}) = 0$ and $\hat{\lambda}_m \geq 0$ [active constraint]
    - $g_m(\widehat{\boldsymbol{w}}) < 0$ and $\hat{\lambda}_m = 0$ [inactive constraint]

SVM Constrained Optimization

- Recall: SVM constrained optimization

$$\min J_1(\boldsymbol{w}, b, \boldsymbol{\epsilon}), \qquad J_1(\boldsymbol{w}, b, \boldsymbol{\epsilon}) = C \sum_{i=1}^{N} \epsilon_i + \frac{1}{2}\|\boldsymbol{w}\|^2$$

  - Constraints: $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \epsilon_i$ and $\epsilon_\mathrm{i} \geq 0$ for all $i = 1, \ldots, N$

- After applying KKT conditions and some algebra [beyond this class], solution is
  - Optimal weight vector: $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$ linear combination of instances
  - Dual parameters α_i minimize

$$\sum_{i=1}^{N} \alpha_i + \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \quad \mathrm{s.t.}\ \ 0 \leq \alpha_i \leq C$$

Correlation Interpretation of SVM

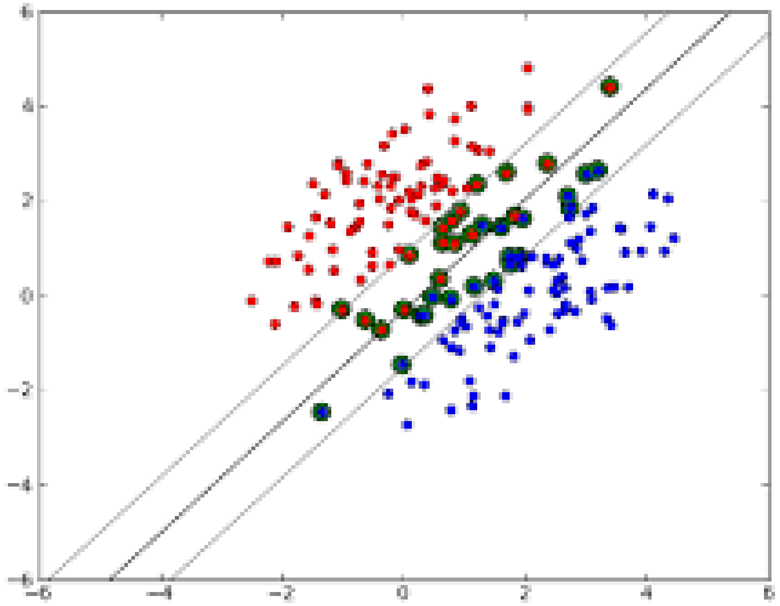
- Classifier weight is:

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$$

- Now suppose we are given a new sample $\boldsymbol{x}$ to classify
- Classifier linear discriminant is:
  - Measure "correlation" $\boldsymbol{x}_i^T \boldsymbol{x}$ of new sample $\boldsymbol{x}$ with each $\boldsymbol{x}_i$ in training data
  - Weight output z by the y_i on the samples where $\boldsymbol{x}_i^T \boldsymbol{x}$ is large

Support Vectors

- Classifier weight is: $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$
- Can show that $\alpha_i > 0$ only when $\boldsymbol{x}_i$ is a support vector
  - On boundary or violating constraint
  - Otherwise $\alpha_i = 0$

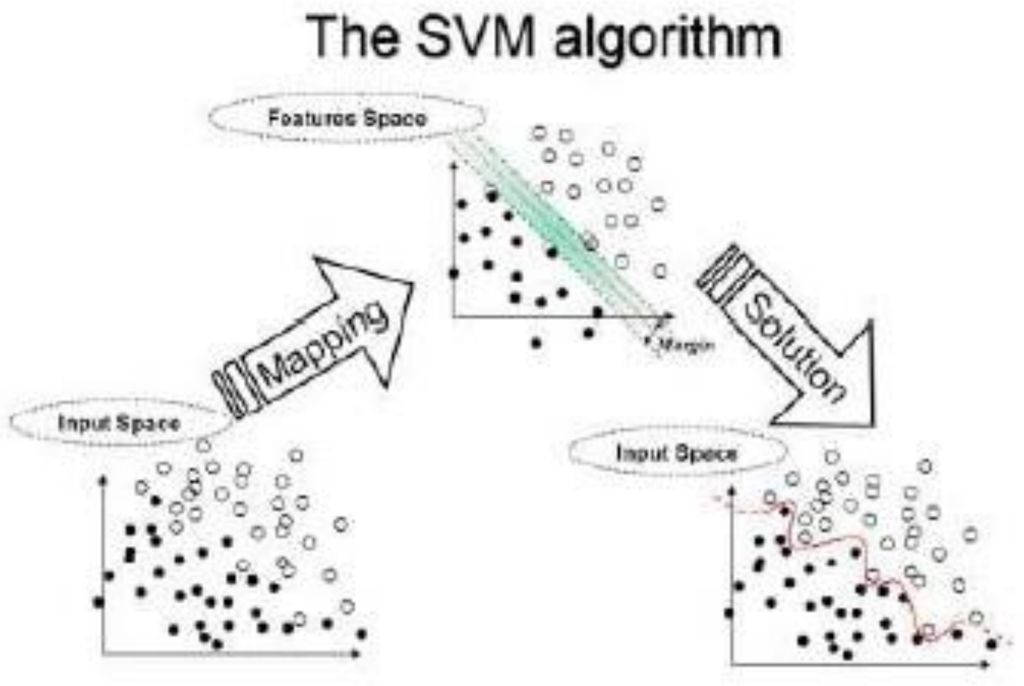


Outline

- Motivating example: Recognizing handwritten digits
  - Why logistic regression doesn’t work well.
- Maximum margin classifiers
- Support vector machines
- Constrained optimization
- Kernel trick

Transform Problem

- Transform problem: replace $\boldsymbol{x}$ with $\phi(\boldsymbol{x})$
  - Enables more rich classifiers
  - Examples: polynomial classification $\phi(x) = [1, x, x^2, \dots, x^{d-1}]$
- Tries to find separation in a feature space



From https://www.dtreg.com/solution/view/20

Transform Problem

- SVM problem in transformed domain:

$$J(\boldsymbol{w}, b) = C\sum_{i=1}^{N} \max(0,1 - y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b)) + \frac{1}{2}\|\boldsymbol{w}\|^2$$

- Solution is of the form:

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \phi(\boldsymbol{x}_i)$$

- Classifier discriminant function:

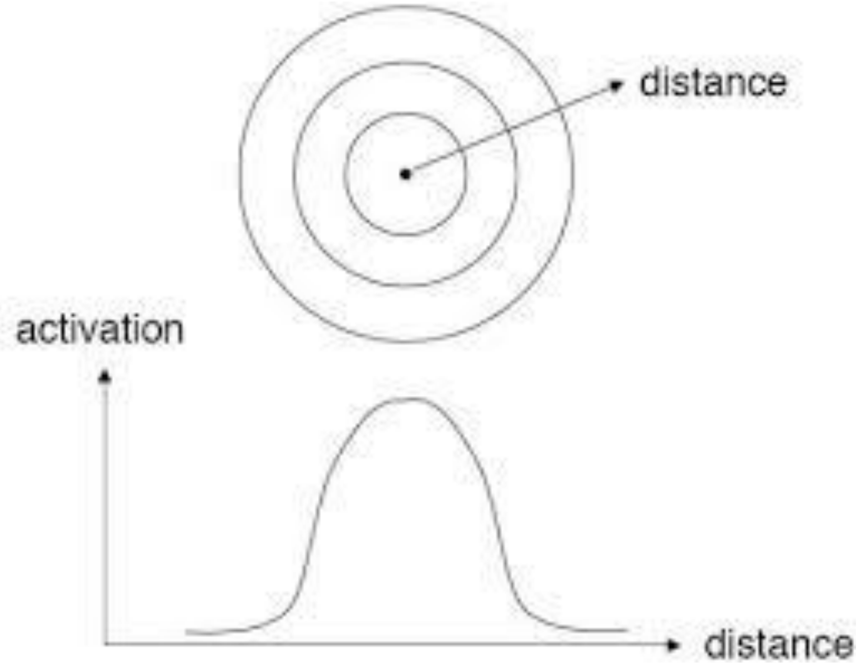
$$z = b + \boldsymbol{w}^T\boldsymbol{x} = b + \sum_{i=1}^{N} \alpha_i y_i \underbrace{\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x})}_{K(\boldsymbol{x}_i,\boldsymbol{x}) = \text{“kernel”}}$$

Kernel Trick

- Classifier is:
  - $z = b + \boldsymbol{w}^T \boldsymbol{x} = b + \sum_{i=1}^{N} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x})$, $\quad K(\boldsymbol{x}_i, \boldsymbol{x}) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x})$
  - $\hat{y} = \text{sign}(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{if } z < 0 \end{cases}$
- Do not need to explicitly compute $\phi(\boldsymbol{x})$
- Can directly compute kernel $K(\boldsymbol{x}_i, \boldsymbol{x})$
  - Provided kernel corresponds to some $\phi(\boldsymbol{x})$

Understanding the Kernel

- Kernel function $K(\boldsymbol{x}_i, \boldsymbol{x})$:
  - measures "distance" between new sample $\boldsymbol{x}$ and training data $\boldsymbol{x}_i$
  - $K(\boldsymbol{x}_i, \boldsymbol{x})$ large $\Rightarrow \boldsymbol{x}_i, \boldsymbol{x}$ close
  - $K(\boldsymbol{x}_i, \boldsymbol{x}) \approx 0 \Rightarrow \boldsymbol{x}_i, \boldsymbol{x}$ far



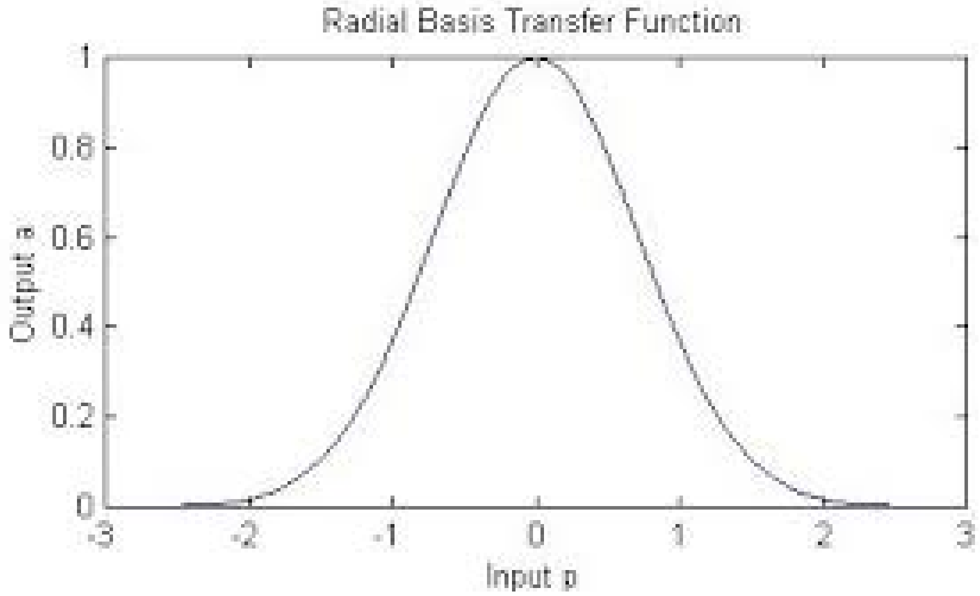
- Linear discriminant $z = b + \sum_{i=1}^{N} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x})$
  - Weighs sample $\boldsymbol{x}_i$ that are close to $\boldsymbol{x}$

Possible Kernels

- Radial basis function:

$$K(x, x') = \exp[-\gamma \|x - x'\|^2]$$

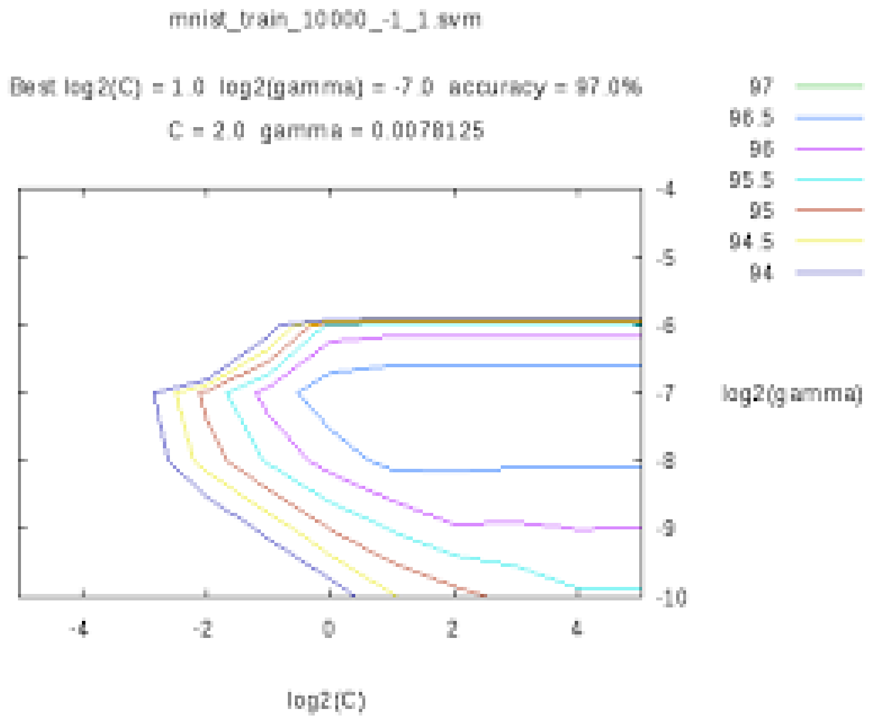
  - $1/\gamma$ indicates width of kernel



- Polynomial kernel: $K(x, x') = |x^T x|^d$
  - Typically d=2

Parameter Selection

- Consider SVM with:
  - Parameter $C > 0$ RBF with $\gamma > 0$
- Higher C or γ
  - Fewer SVs
  - Classifiers averages over smaller set
  - Lower bias, but highe variance
- Typically select via cross-validation
  - Try out different (C, γ)
  - Find which one provides highest accuracy on test set
- Python can automatically do grid search



http://peekaboo-vision.blogspot.com/2010/09/mnist-for-ever.html

Multi-Class SVMs

- Suppose there are K classes
- One-vs-one:
  - Train $\binom{K}{2}$ SVMs for each pair of classes
  - Test sample assigned to class that wins "majority of votes"
  - Used in sklearn. Best results but very slow
- One-vs-all:
  - Train K SVMs: train each class k against all other classes
  - Pick class with highest z_k

MNIST Results

- Run classifier
- Very slow
  - Several minutes for 40,000 samples
  - Slow in training and test
  - Major drawback of SVM
- Accuracy ≈ 0.984
  - Much better than logistic regression
- Can get better with:
  - pre-processing
  - More training data
  - Optimal parameter selection

```
from sklearn import svm

# Create a classifier: a support vector classifier
svc = svm.SVC(probability=False,  kernel="rbf", C=2.8, gamma=.0073,verbose=10)
```

```
svc.fit(Xtr,ytr)
```

```
[LibSVM]

SVC(C=2.8, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma=0.0073, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=10)
```

```
yhat1 = svc.predict(Xts)
acc = np.mean(yhat1 == yts)
print('Accuaracy = {0:f}'.format(acc))
```

```
Accuaracy = 0.984000
```

MNIST Errors

- Some of the error are hard even for a human