LECTURE 8: SUPPORT VECTOR MACHINES

Ehsan Aryafar
earyafar@pdx.edu
http://web.cecs.pdx.edu/~aryafare/ML.html

Recall: Gradient Defined

- Consider scalar-valued function $f(\boldsymbol{w})$
- Vector input $\boldsymbol{w}$. Then gradient is:

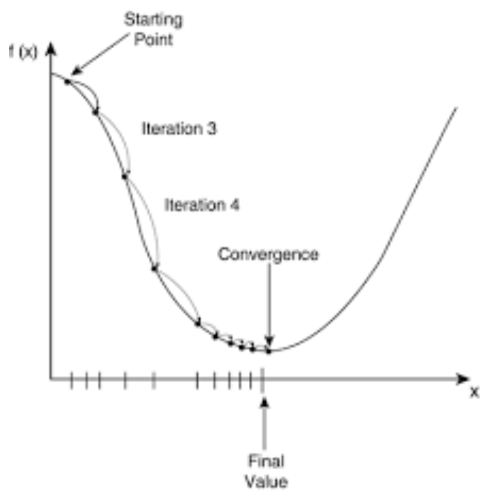
$$\nabla_w f(\boldsymbol{w}) = \begin{bmatrix} \partial f(\boldsymbol{w})/\partial w_1 \\ \vdots \\ \partial f(\boldsymbol{w})/\partial w_N \end{bmatrix}$$

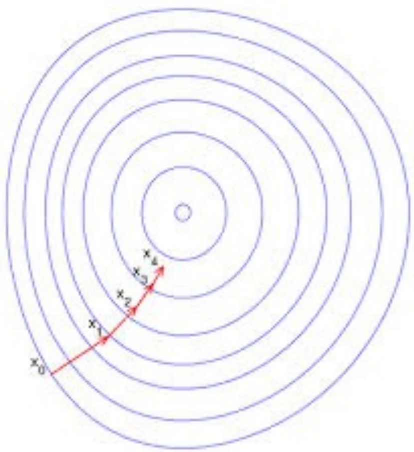
- Matrix input $\boldsymbol{W}$, size $M \times N$. Then gradient is:

$$\nabla_w f(\boldsymbol{W}) = \begin{bmatrix} \partial f(\boldsymbol{W})/\partial W_{11} & \cdots & \partial f(\boldsymbol{W})/\partial W_{1N} \\ \vdots & \vdots & \vdots \\ \partial f(\boldsymbol{W})/\partial W_{M1} & \cdots & \partial f(\boldsymbol{W})/\partial W_{MN} \end{bmatrix}$$

- Gradient is same size as the argument!

Recall: Gradient Descent Illustrated





- $M = 1$
- $M = 2$

Recall: Convex Sets

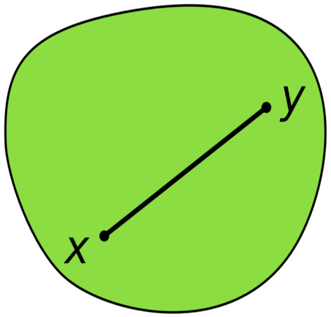
- Definition: A set X is convex if for any $x, y \in X$,

$$tx + (1-t)y \in X \text{ for all } t \in [0,1]$$

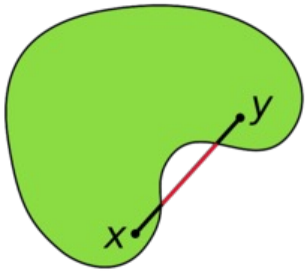
- Any line between two points remains in the set.
- Examples:
  - Square, circle, ellipse
  - $\{x \mid Ax \leq b\}$ for any matrix A and vector b

Recall: Convex Set Visualized

- Convex



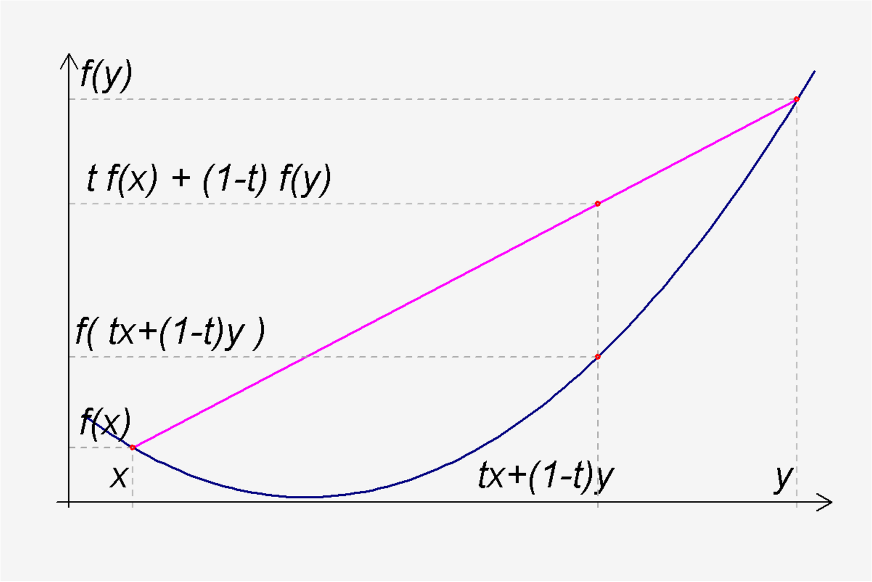
- Not convex



Recall: Convex Functions

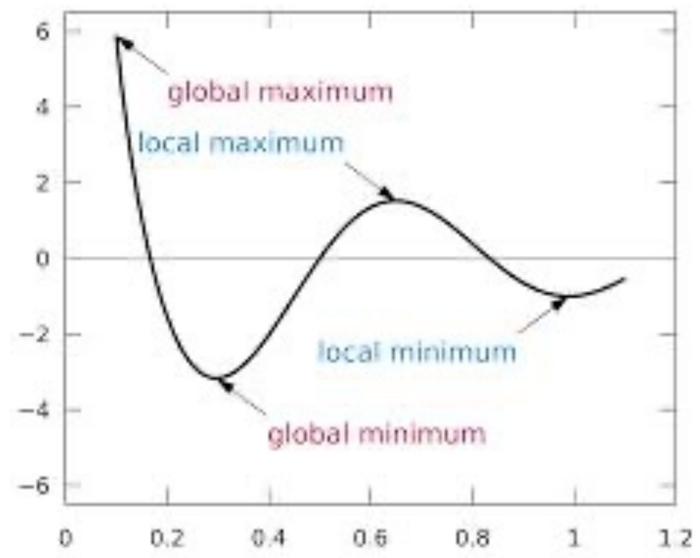
- A real-valued function $f(x)$ is convex if:
  - Its domain is a convex set, and
  - For all x, y and $t \in [0,1]$:

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$$



Recall: Global Minima and Convex Function

- Theorem: If $f(w)$ is convex and w is a local minima, then w is a global minima
- Implication for optimization:
  - Gradient descent only converges to local minima
  - In general, cannot guarantee optimality
  - Depends on initial condition
  - But, for convex functions can always obtain optimal



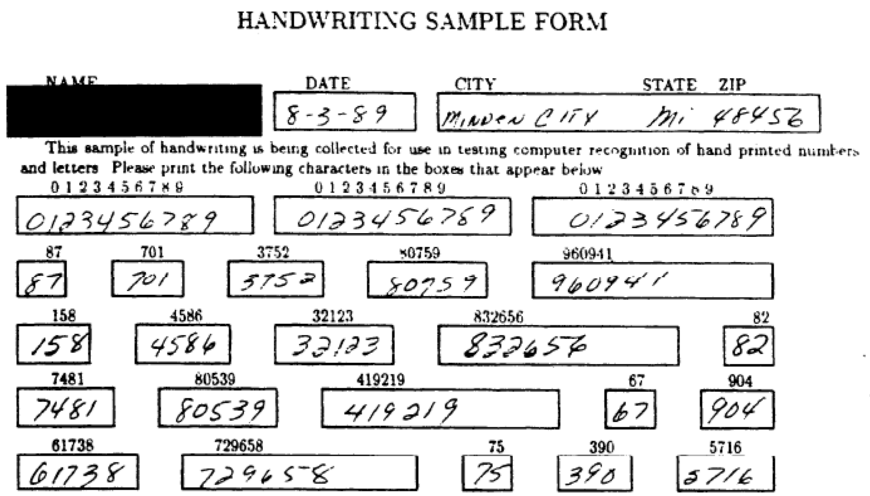
Learning Objectives

- Interpret weights in linear classification of images
- Describe why linear classification for images does not work
- Define the margin in linear classification
- Describe the SVM classification problem.
- Describe a kernel SVM problem for non-linear classification
- Implement SVM classifiers in python
- 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
- 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 | 0 1 2 3 4 5 6 7 8 9 | 0 1 2 3 4 5 6 7 8 9

87 | 701 | 3752 | 80759 | 960941

158 | 4586 | 32123 | 832656 | 82

7481 | 80539 | 419219 | 67 | 904

61738 | 729658 | 75 | 390 | 5716

From Patrick J. Grother, NIST Special Database, 1995

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

A Widely-Used Benchmark

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]

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

Downloading MNIST

```
import tensorflow as tf

(Xtr,ytr),(Xts,yts) = tf.keras.datasets.mnist.load_data()

print('Xtr shape: %s' % str(Xtr.shape))
print('Xts shape: %s' % str(Xts.shape))

ntr = Xtr.shape[0]
nts = Xts.shape[0]
nrow = Xtr.shape[1]
ncol = Xtr.shape[2]
```

```
Xtr shape: (60000, 28, 28)
Xts shape: (10000, 28, 28)
```

- MNIST data is available in many sources
  - Note: It has been removed from sklearn
- Tensorflow version:
  - 60000 training samples
  - 10000 test samples
- Each sample is a 28 x 28 image
- Grayscale: Pixel values $\in \{0,1,\ldots,255\}$
  - 0 = Black and
  - 255 = White

Matrix and Vector Representation

- For this demo, we reshape data from $N \times 28 \times 28$ to $N \times 784$
- But, you can easily go back and forth
- Also, scale the pixel values from -1 to 1

```
npix = nrow*ncol
Xtr = 2*(Xtr/255 - 0.5)
Xtr = Xtr.reshape((ntr,npix))

Xts = 2*(Xts/255 - 0.5)
Xts = Xts.reshape((nts,npix))
```

28

28

xsq

```
x = xsq.ravel()
```

```
xsq = x.reshape((28,28))
```

$784 = 28^2$

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

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

Displaying Images in Python

4 random images in the dataset

We want to classify each digit

A human can classify these easily

Getting a computer to do is harder

```
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

Sample permutation is necessary for this dataset, as the original data is ordered by digits

Try a Logistic Classifier

```
ntr1 = 5000
Xtr1 = Xtr[Iperm[:ntr1],:]
ytr1 = ytr[Iperm[:ntr1]]
```

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

```
from sklearn import linear_model
logreg = linear_model.LogisticRegression(verbose=10, solver='sag',\
                                         max_iter=500)
logreg.fit(Xtr1,ytr1)
```

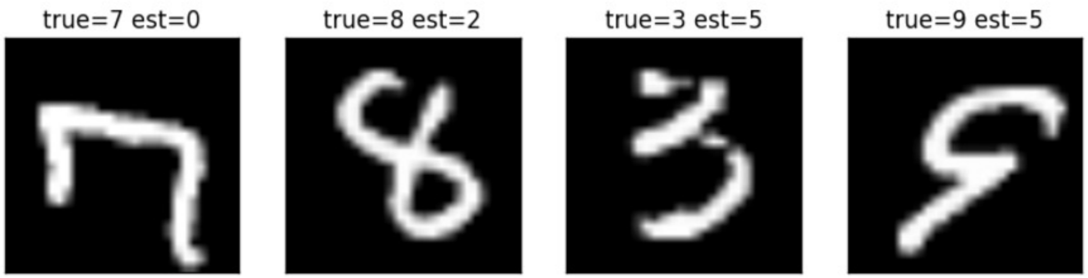
```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
convergence after 408 epochs took 64 seconds
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.1min remaining:    0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.1min finished
```

Performance

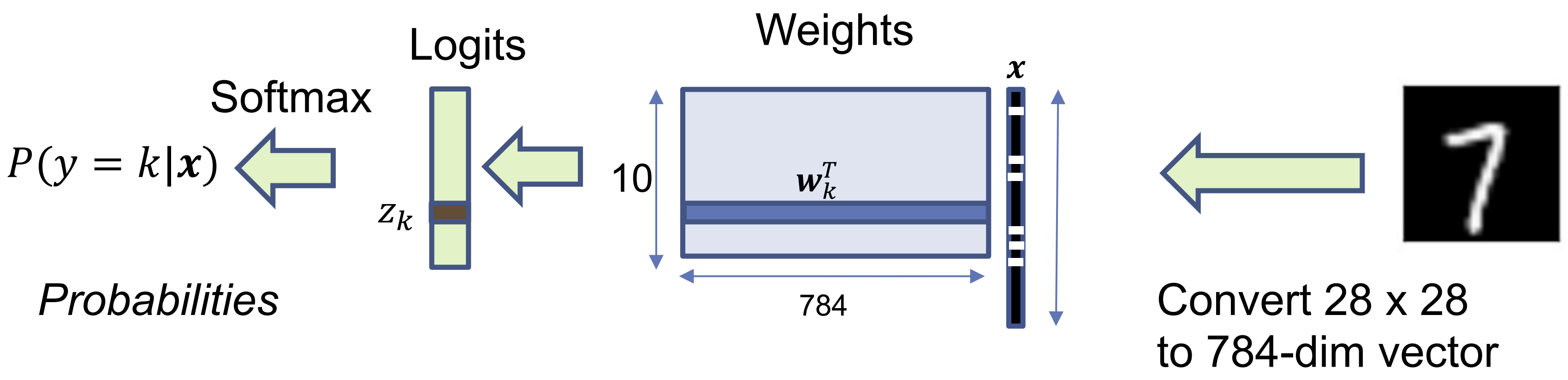
- Accuracy = 89%. Very bad
- Some of the errors seem like they should have been easy to spot
- What went wrong?

```
nts1 = 5000
Iperm_ts = np.random.permutation(nts)
Xts1 = Xts[Iperm_ts[:nts1],:]
yts1 = yts[Iperm_ts[:nts1]]
yhat = logreg.predict(Xts1)
acc = np.mean(yhat == yts1)
print('Accuaracy = {0:f}'.format(acc))
```

```
Accuaracy = 0.891000
```



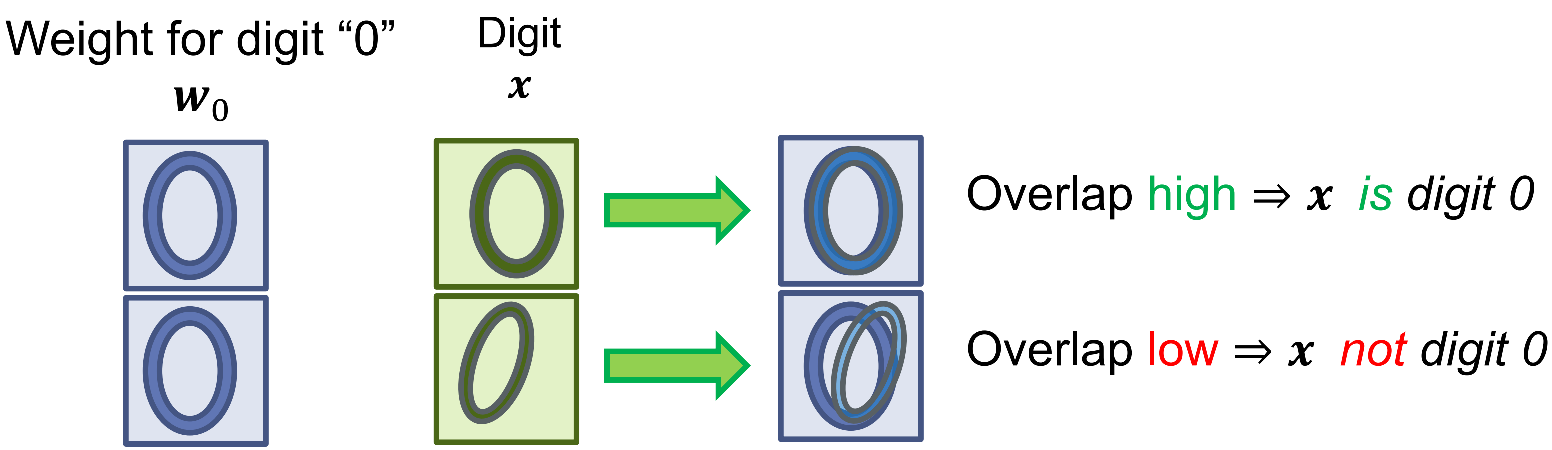
Recap: Logistic Classifier



- Each logit $z_k = \boldsymbol{w}_k^T \boldsymbol{x}$ = inner product with weight $\boldsymbol{w}_k$ with digit $\boldsymbol{x}$, $k = 0, \dots, 9$
- 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

- A logit $z_k = \boldsymbol{w}_k^T \boldsymbol{x}$ is high when there is high overlap between $\boldsymbol{w}_k$ with digit $\boldsymbol{x}$
  - Visualize each weight as an image
  - Suppose pixels are 0 or 1
  - $z_k = \boldsymbol{w}_k^T \boldsymbol{x} = \sum_i w_{ki} x_i$ = number of pixels that overlap with $\boldsymbol{w}_k$ and $\boldsymbol{x}$
- Conclusion: Small variations in digits can cause low overlap

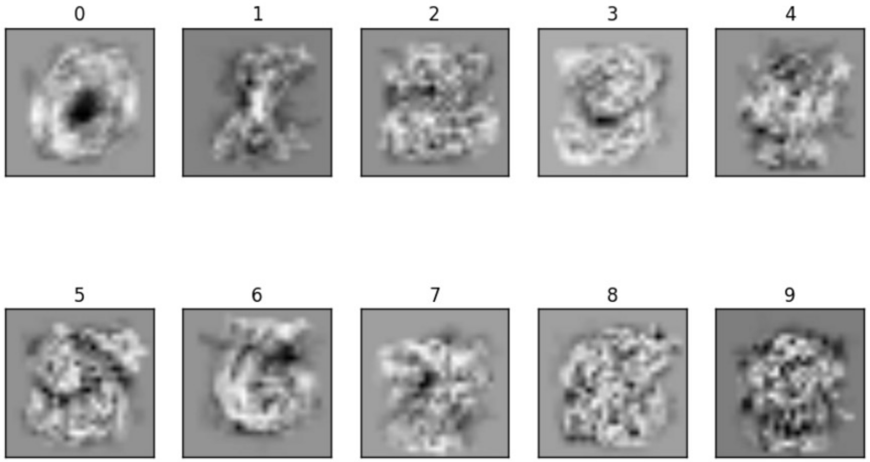


Example with Actual Digits

- Take weight $\boldsymbol{w}$ from a random digit "2"
- Inner products $z = \boldsymbol{w}^T\boldsymbol{x}$ are only slightly higher for other digits "2"
- Cannot tell which digit is correct from the inner product $z = \boldsymbol{w}^T\boldsymbol{x}$

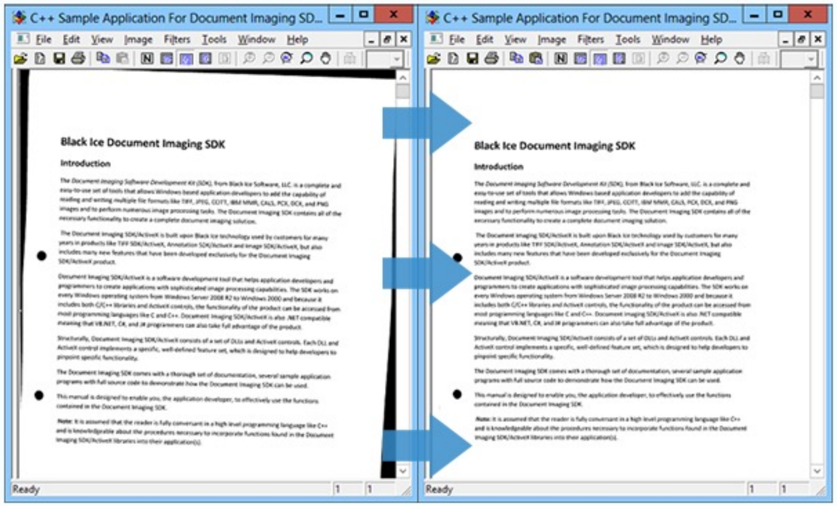
Visualizing the Weights

- Optimized weights of the classifier
- Blurry versions of image to try to capture rotations, translations, …



Problems with Logistic Classifier

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

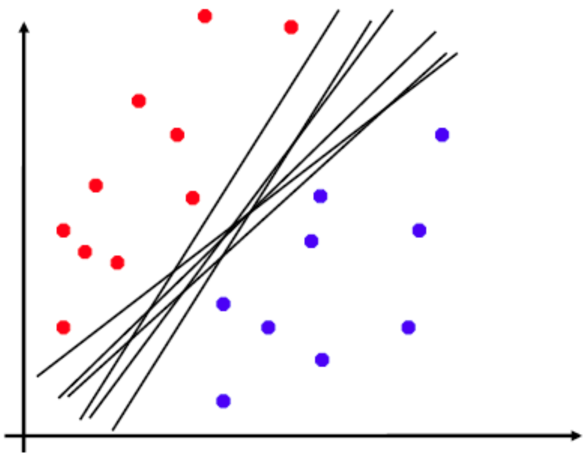


Outline

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

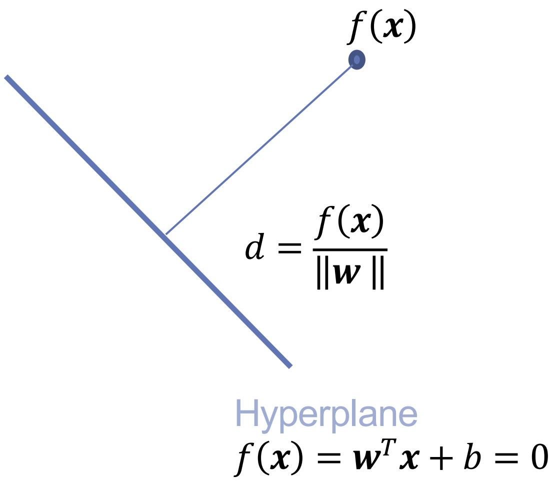
Non-Uniqueness of Separating Plane

- Linearly separable data:
  - Can find a separating hyper-plane as a linear classifier.
- Separating hyper-plane is not unique
  - Fig. on right: Many separating planes
- Which one is optimal?



Hyperplane Basics

- Linear function: $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b, \boldsymbol{x} \in R^d$
- Hyperplane in d-dimensional: $f(\boldsymbol{x}) = 0$
- Parameters:
  - Weight $\boldsymbol{w}$ and bias b
  - Unique up to scaling:
  - $(b, \boldsymbol{w})$ and $(\alpha b, \alpha\boldsymbol{w})$ define the same plane.
  - For unique definition, we can require $\|\boldsymbol{w}\|$=1.



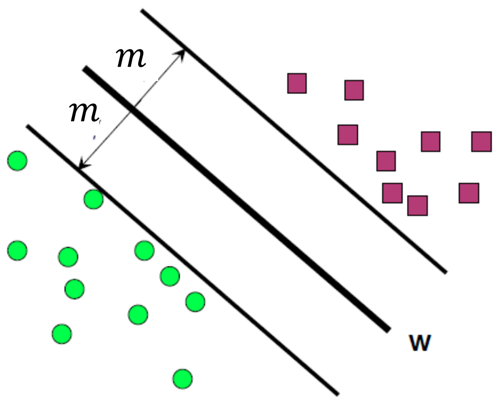
- Distance of any point **x** to the hyperplane:
  - $d = f(\boldsymbol{x})/\|\boldsymbol{w}\|$, where $f(\boldsymbol{x}) = b + \boldsymbol{w}^T\ \boldsymbol{x}$.
  - See ESL Sec. 4.5.
  - ESL: Hastie, Tibshirani, Friedman, “The Elements of Statistical Learning”. 2nd Ed. Springer.

Linear Separability and Margin

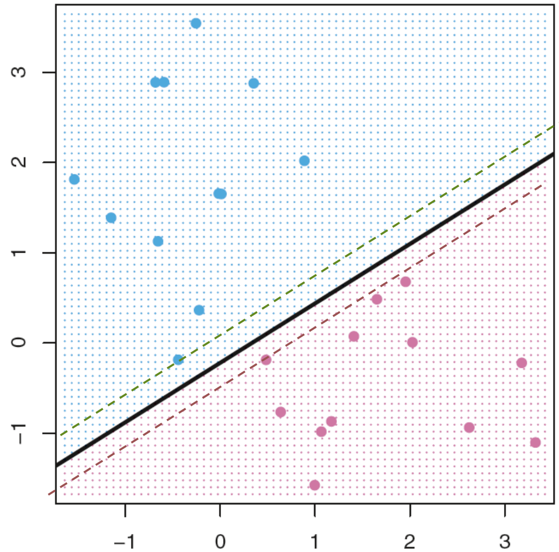
- Given training data $(\boldsymbol{x}_i, y_i), i = 1, \dots, N$
  - Binary class label: $y_i = \pm 1$
- Suppose it is separable with parameters $(\boldsymbol{w}, b)$
- There must exist a $\gamma > 0$ s.t.:
  - $b + w_1 x_{i1} + \cdots w_d x_{id} > \gamma$ when $y_i = 1$
  - $b + w_1 x_{i1} + \cdots w_d x_{id} < -\gamma$ when $y_i = -1$
- Single equation form:

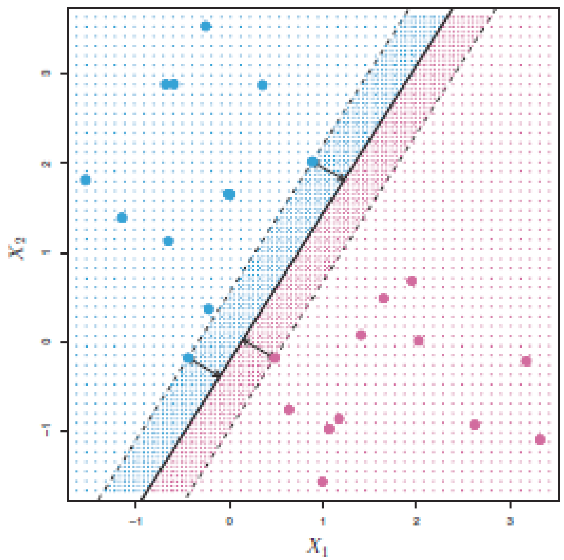
$$y_i(b + w_1 x_{i1} + \cdots w_d x_{id}) > \gamma \text{ for all } i = 1, \dots, N$$

- Margin: $\mathrm{m} = \frac{\gamma}{\|\boldsymbol{w}\|}$: minimal distance of a sample to the plane
  - γ is the minimum value satisfying the above constraints



Which separating plane is better ?





From Fig. 9.2 and Fig. 9.3 in ISL.

Maximum Margin Classifier

- For the classifier to be more robust to noise, we want to maximize the margin!

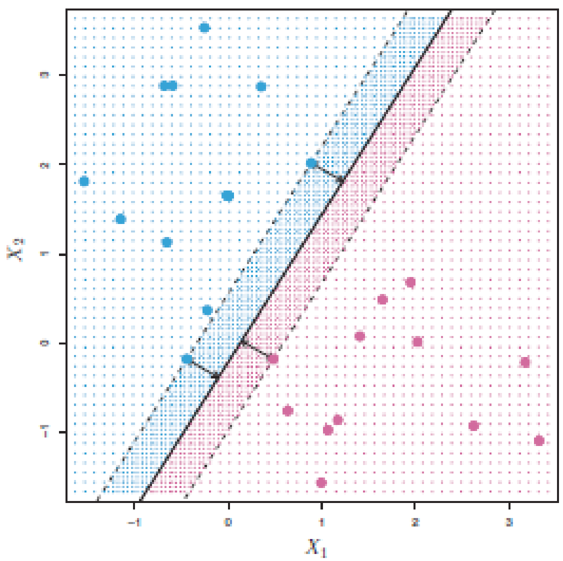
- Define maximum margin classifier optimization problem

$$\max_{w,\gamma} \gamma$$ 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 problem
  - Objective function and constraints
  - More on this later.
- See closed form solution in Sec. 4.5.2 in ESL. Note notation difference.

Visualizing Maximum Margin Classifier



- Fig. 9.3 of ISL
- Margin determined by closest points to the line
  - The maximal margin hyperplane represents the mid-line of the widest “slab” that we can insert between two classes
- In this figure, there are 3 points at the margin

ISL: James, Witten, Hastie, Tibshirani, An Introduction to Statistical Learning, Springer. 2013.

Problems with MM classifier

- Data is often not perfectly separable
  - You cannot talk about margin
  - Only want to correctly separate most points

- MM classifier is not robust
  - A single sample can radically change line
  - Suggests generalization errors may not be good

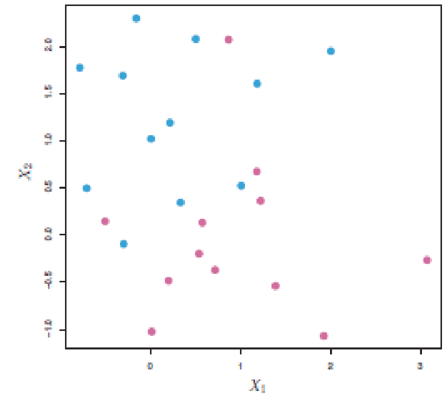


Fig. 9.4

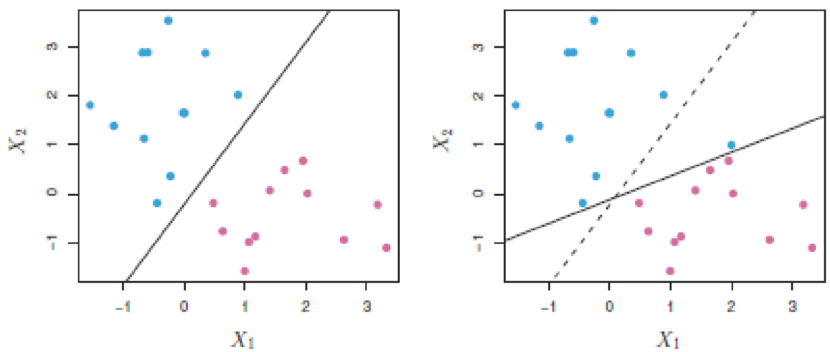


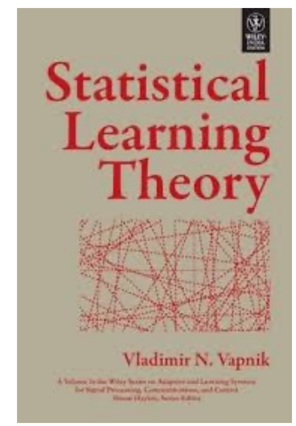
Fig. 9.5

Outline

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

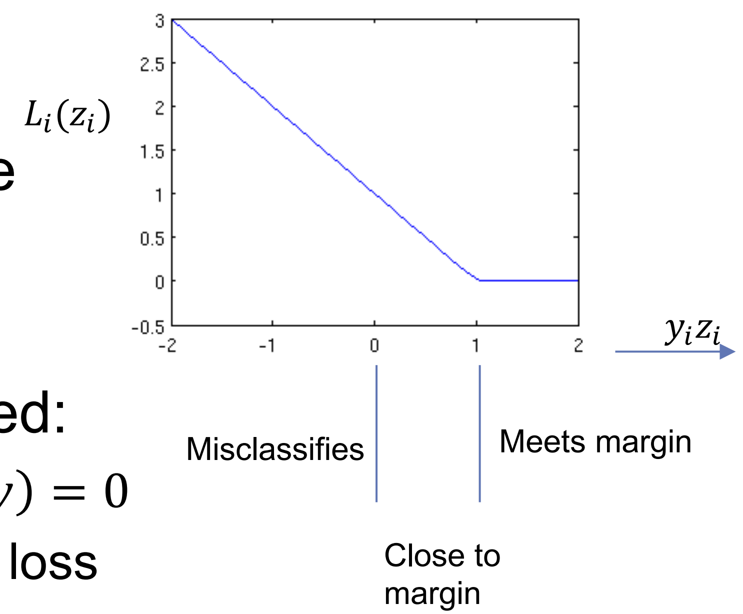
Support Vector Machine

- 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
  - Support vector classifier (SVC): Directly use raw features. Good when the original feature space is roughly linearly separable
  - Support vector machine (SVM): Map the raw features to some other domain through a kernel function



Hinge Loss

- Fix $\gamma = 1$
- Want ideally: $y_i(\boldsymbol{w}^T\boldsymbol{x} + b) \geq 1$ for all samples i
  - Equivalently, $y_i z_i \geq 1, \;\; z_i = b + \boldsymbol{w}^T\boldsymbol{x}$
  - Note that y_i is + or - one
- But perfect separation may not be possible
- Define hinge loss or soft margin:
  - $L_i(\boldsymbol{w}, b) = \max(0, 1 - y_i z_i)$
- 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, small loss
  - $y_i z_i \leq 0 \;\Rightarrow$ Sample misclassified, large loss



SVM Optimization

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

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

C controls final margin

Hinge loss term
Attempts to reduce
Misclassifications

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 (Constrained Optimization Format)

- 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:

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

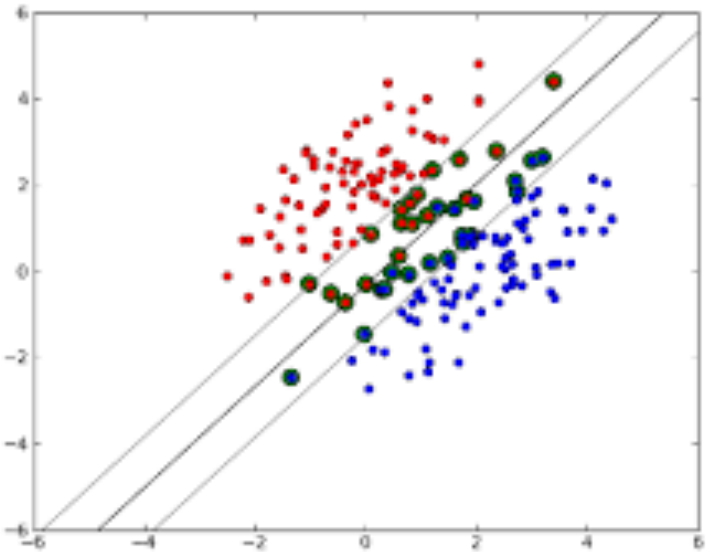
  - ϵ_i = amount sample i misses margin target
- Sometimes written as $J_1(\boldsymbol{w}, b, \boldsymbol{\epsilon}) = C\|\boldsymbol{\epsilon}\|_1 + \frac{1}{2}\|\boldsymbol{w}\|^2$
  - $\|\boldsymbol{\epsilon}\|_1 = \sum_{i=1}^{N} \epsilon_i$ called the "one-norm"
  - Generally one-norm would have absolute sign over ϵ_i.
  - But in this case, when the constraint is met, ϵ_i>=0.

Interpreting Parameters

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

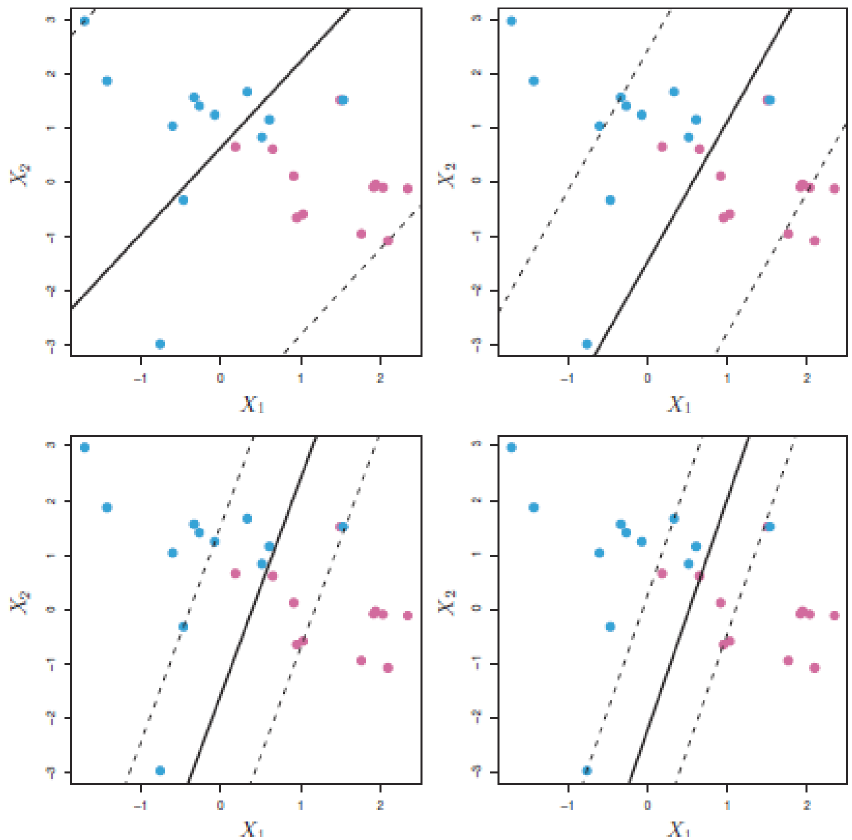
Support Vectors

- 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: $y_{\mathrm{i}}(\boldsymbol{w}^T\boldsymbol{x}_i + b) \le 1$
- Changing samples that are not SVs
  - Does not change solution
  - Provides robustness



Illustrating Effect of C

Low $C = C_1$ C_2



C_3 High C_4

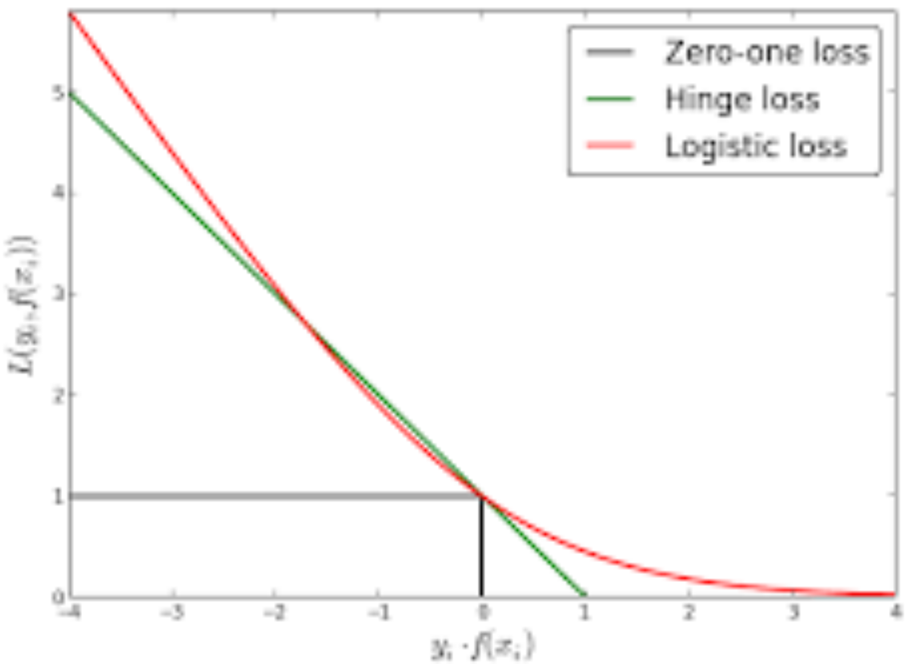
- Fig. 9.7 of ISL
  - Note: C has opposite meaning in ISL than python
  - Here, we use python meaning
- Low C:
  - Leads to large margin
  - But allow many violations of margin.
  - Many more SVs
  - Reduces variance or increases bias by using more samples
- Large C:
  - Leads to small margin
  - Reduce number of violations, and fewer SVs.
  - Highly fit to data. Low bias, higher variance
  - More chance to overfit

Relation to Logistic Regression

- Logistic regression also minimizes a loss function:

$$J(\boldsymbol{w}, b) = \sum_{i=1}^{N} L_i(w, b)\,,$$

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

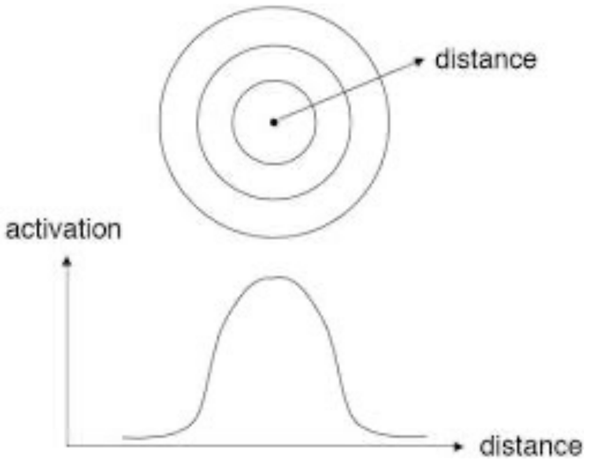


Outline

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

The Kernel Function

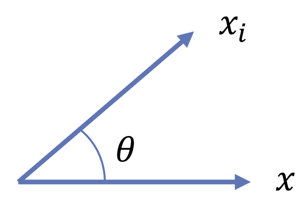
- Kernel function:
  - Function $K(\boldsymbol{x}_i, \boldsymbol{x})$
  - Key function for SVMs and kernel classifiers
  - Measures "similarity" between new sample $\boldsymbol{x}$ and training sample $\boldsymbol{x}_i$



- Typical property
  - $\boldsymbol{x}_i, \boldsymbol{x}$ close $\Rightarrow K(\boldsymbol{x}_i, \boldsymbol{x})$ maximum value
  - $\boldsymbol{x}_i, \boldsymbol{x}$ far $\Rightarrow K(\boldsymbol{x}_i, \boldsymbol{x}) \approx 0$

Common Kernels

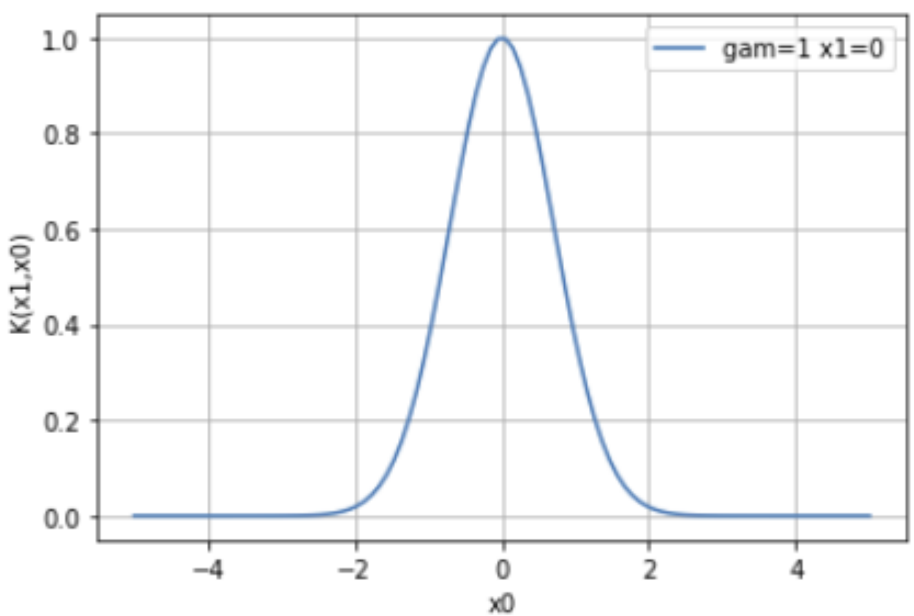
- Linear SVM:
  - $K(x_i, x) = x_i^T x = \|x_i\| \|x\| \cos\theta$
  - Maximum when angle between vectors is small



- Radial basis function:

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

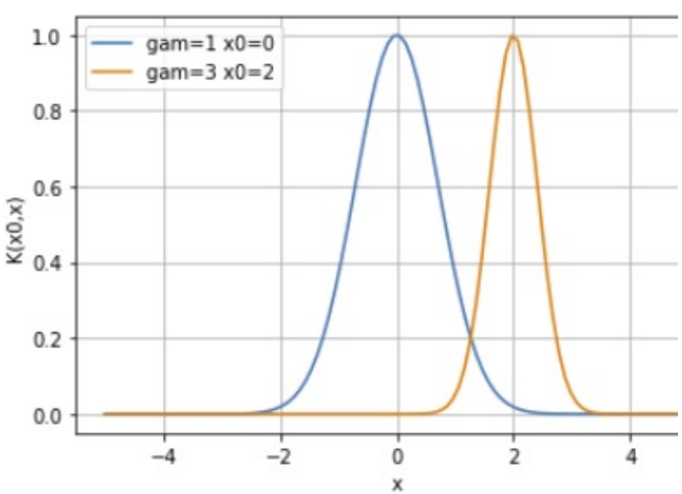
  - $1/\gamma$ indicates width of kernel



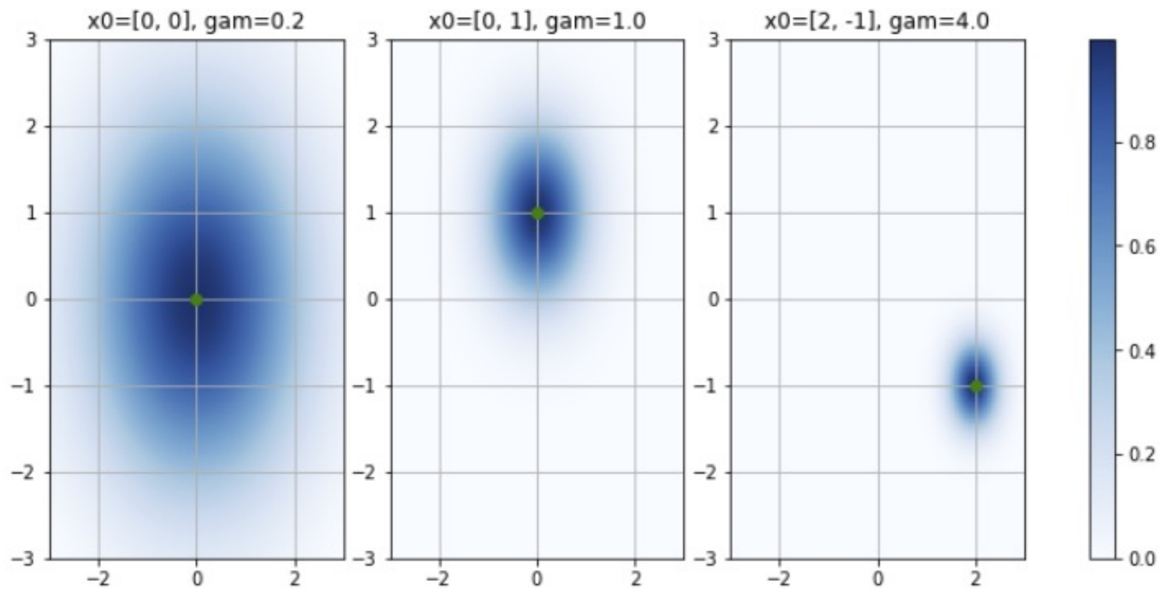
- Polynomial kernel: $K(x_i, x) = |x_i^T x|^d$
  - Inner product to the power of d!
  - Typically d=2

RBF Kernel Examples

- RBF kernel: $K(x_0, x) = \exp[-\gamma \|x - x_0\|^2]$
  - Peak value of 1 at $x = x_0$
  - Decay with a rate of $\frac{1}{\gamma}$
  - Width $\propto \frac{1}{\gamma}$



RBFs in 1D



Kernel Classifier

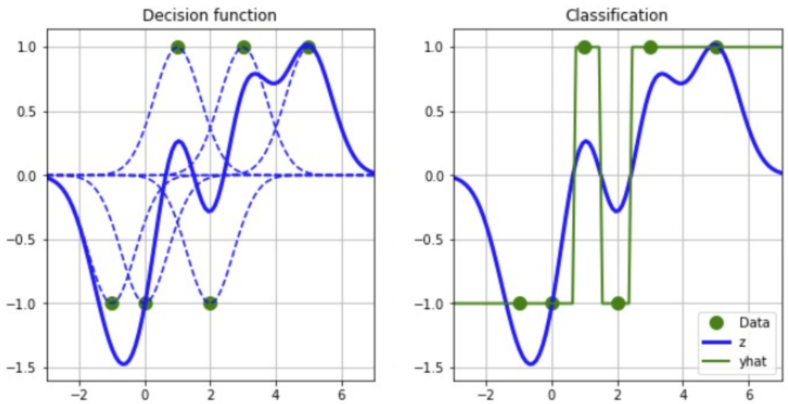
- Given:
  - Training data (x_i, y_i) with binary labels $y_i = \pm 1$
  - Kernel $K(x_i, x)$
- To classify a new point x:
  - Decision function: $z = \sum_{i=1}^{n} y_i K(x_i, x)$
  - Classify: $\hat{y} = sign(z)$
- Idea:
  - z is large positive when x is close to samples x_i with $y_i = 1$
  - z is large negative when x is close to samples x_i with $y_i = -1$

- Kernel classifiers are a subject on their own
  - We just mention them here to explain connection to SVMs

Example in 1D

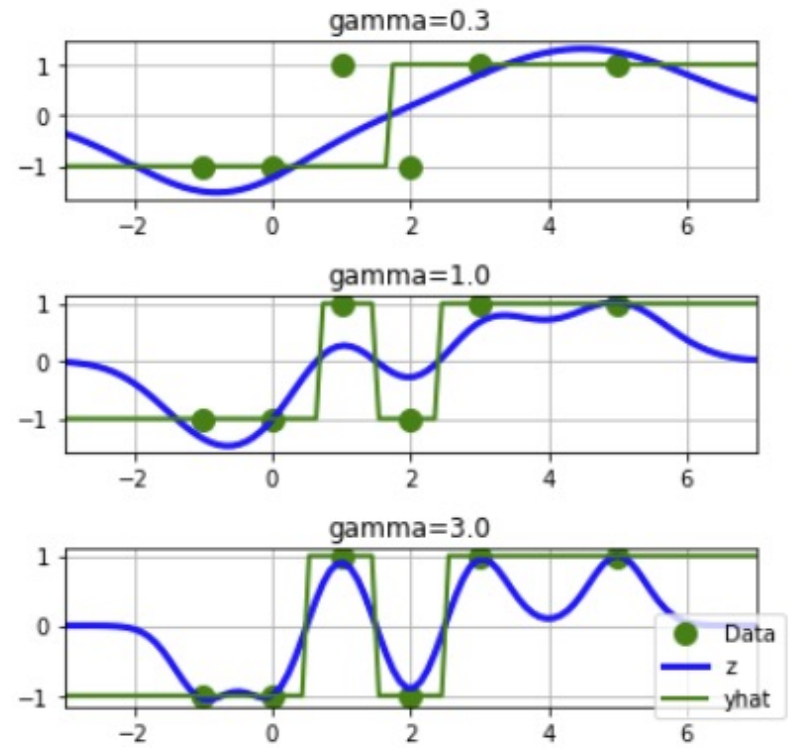
- Example data with 6 points (x_i, y_i)
  - RBF kernel: $K(x_i, x) = e^{-\gamma(x_i - x)^2}$, $\gamma = 1$
- Decision function:
  - $z = \sum_{i=1}^{n} y_i K(x_i, x)$
  - Sum of bell curves
  - Positive when near positive samples
  - Negative when near negative samples
- Classification:
  - $\hat{y} = sign(z)$

i	1	2	3	4	5	6
x_i	-1	0	1	2	3	5
y_i	-1	-1	1	-1	1	1



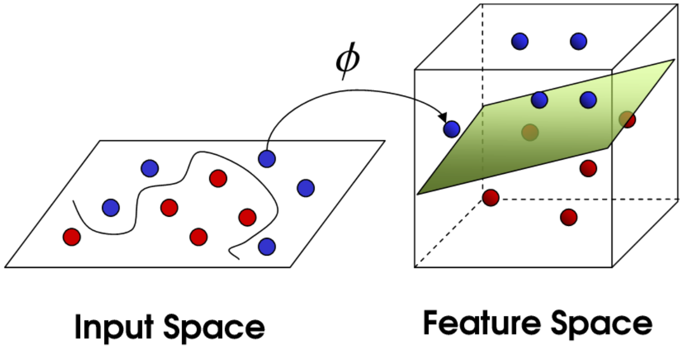
Effect of Gamma

- Same data as before
- RBF kernel: $K(x_i, x) = e^{-\gamma(x_i - x)^2}$
- As γ increases:
  - Decision function $z \approx y_i$ when $x = x_i$
  - Classifier fits training data better
  - Classification region more complex
- As a classifier, higher γ results in:
  - Lower bias error (fits training data)
  - But, higher variance error
  - Overfitting



SVMs with Non-Linear Transformations

- Non-linear transformation:
  - Replace $\boldsymbol{x}$ with $\phi(\boldsymbol{x})$
  - Enables more rich, non-linear classifie
  - Examples: polynomial classification



$$\phi(x) = [1, x, x^2, \ldots, x^{d-1}]$$

- Tries to find separation in a feature space (e.g., classification in the picture)
  - You can do this with any classifier (we have already done this)

- Kernel trick in SVMs:
  - Makes applying non-linear transformations easy

SVM with the Transformation

- Consider SVM model with $\boldsymbol{x}$ replaced by $\phi(\boldsymbol{x})$
- Minimize SVM cost function as before (i.e. Hinge loss + inverse margin)
- Theorem: The optimal weight is of the form (linear):

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

  - $\alpha_i \geq 0$ for all i
  - $\alpha_i > 0$ if and only if sample i is a support vector
  - Will show this fact later using results in constrained optimization
- Consequence: The linear discriminant on any other sample x is:

$$z = b + \boldsymbol{w}^T \phi(\boldsymbol{x}) = b + \sum_{i=1}^{N} \alpha_i y_i \boxed{\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x})}$$

  -

$K(\boldsymbol{x}_i, \boldsymbol{x})$ = "kernel"

Kernel Form of the SVM Classifier

- SVM classifier can be written with the kernel $K(\boldsymbol{x}_i, \boldsymbol{x})$ and values $\alpha_i \geq 0$:

$$z = b + \sum_{i=1}^{N} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}),$$

$$\hat{y} = \operatorname{sign}(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{if } z < 0 \end{cases}$$

Decision function

Classification decision

- Key point: SVM classifier is approximately Kernel classifier
- But there are two differences:
  - introduction of weights $\alpha_i \geq 0$ on the samples (the weights are only non-zero on the SVs)
  - A bias term b (can be positive or negative)

"Kernel Trick" and Dual Parameterization

- Kernel form of SVM classifier (previous slide):

$$z = b + \sum_{i=1}^{N} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}),$$
$$\hat{y} = \mathrm{sign}(z)$$

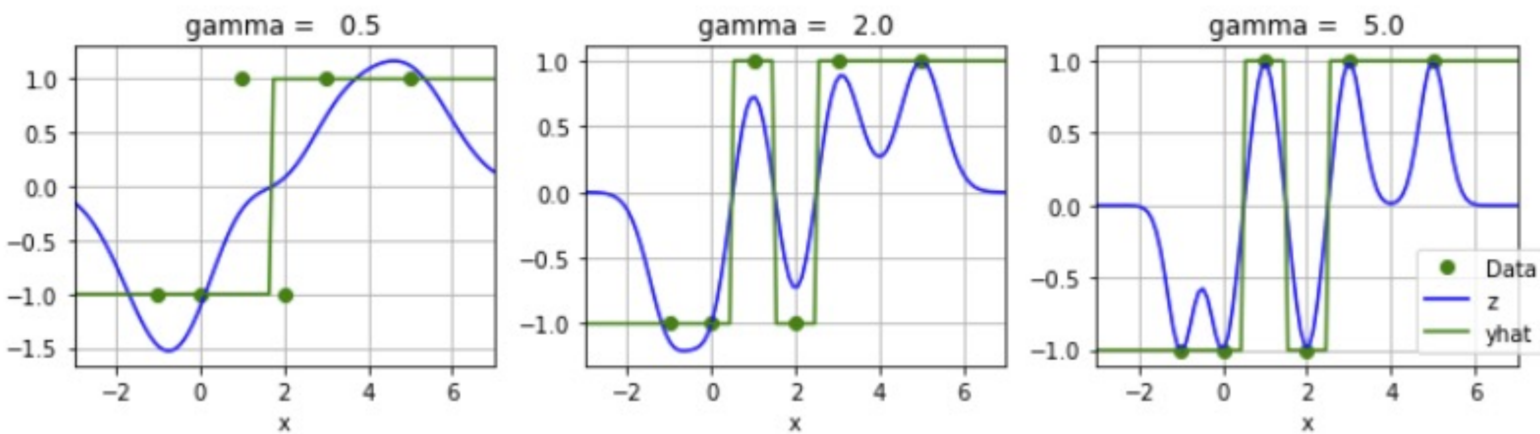
- Dual parameters: $\alpha_i \geq 0, i = 1, \ldots, N$
  - Problem based on α_i parameters
  - Called the dual parameters due to constrained optimization – see next section

- Kernel trick:
  - Directly solve the parameters $\boldsymbol{\alpha}$ instead of the weights $\boldsymbol{w}$
  - Can show that the optimization only needs the kernel $K(\boldsymbol{x}_i, \boldsymbol{x})$
  - Does not need to explicitly use $\phi(\boldsymbol{x})$

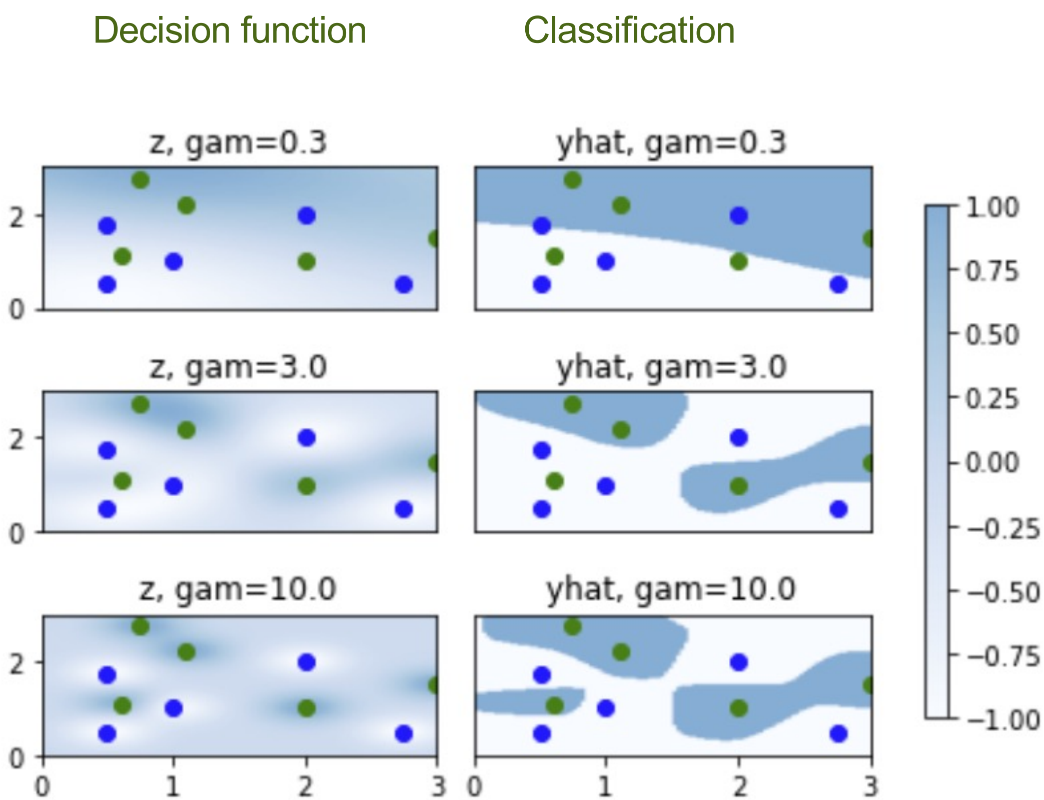
SVM Example in 1D

i	1	2	3	4	5	6
x_i	-1	0	1	2	3	5
y_i	-1	-1	1	-1	1	1

- Same data as in the Kernel classifier example
- Fit SVM with RBF with different γ
- Similar trends as kernel classifier: As γ increases
  - z "fits" data (x_i, y_i) closer
  - Leads to more complex decision regions.
  - Enables nonlinear decision regions



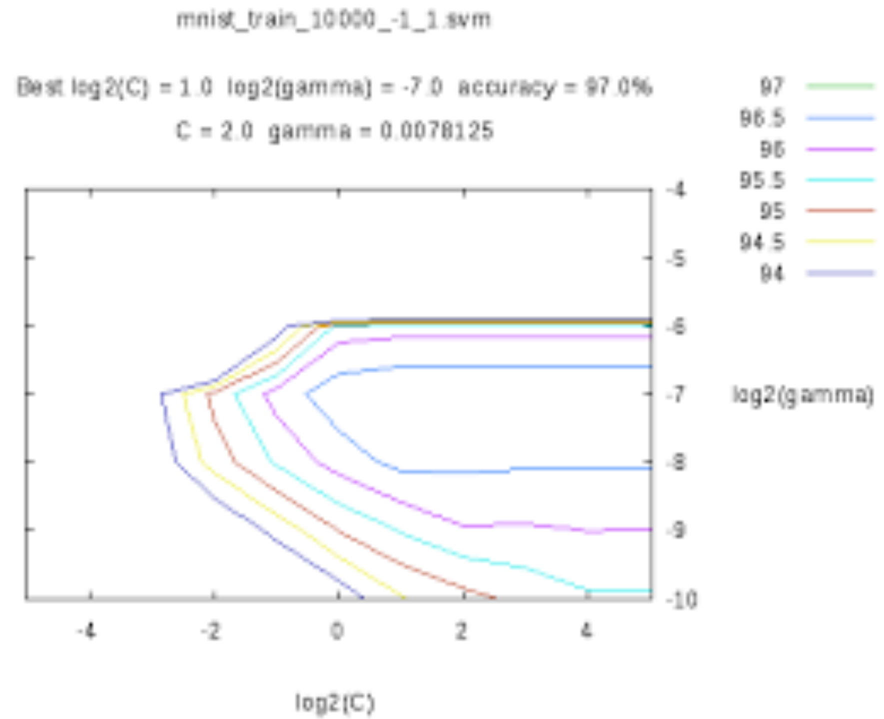
Example in 2D



- Example:
  - 10 data points with binary labels
  - Fit SVM with $C = 1$ and RBF
  - $\gamma = 0.3$, 3 and 10
- Plot:
  - z= linear discriminant
  - $\hat{y} = sign(z) =$ classification decision
- Observe: As γ increases
  - Fits training data better
  - More complex decision region

Parameter Selection

- For SVMs with RBFs we need to select:
  - Parameter $C > 0$ in the loss function
  - Kernel width $\gamma > 0$
- Higher C or γ
  - Fewer SVs
  - Classifiers averages over smaller set
  - Lower bias, but higher variance
- Typically select via cross-validation
  - Try out different (C, γ) pairs
  - 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”
  - Best results but very slow
- One-vs-rest:
  - Train K SVMs: train each class k against all other classes
  - Pick class with highest z_k
- Sklearn has both options

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)
```

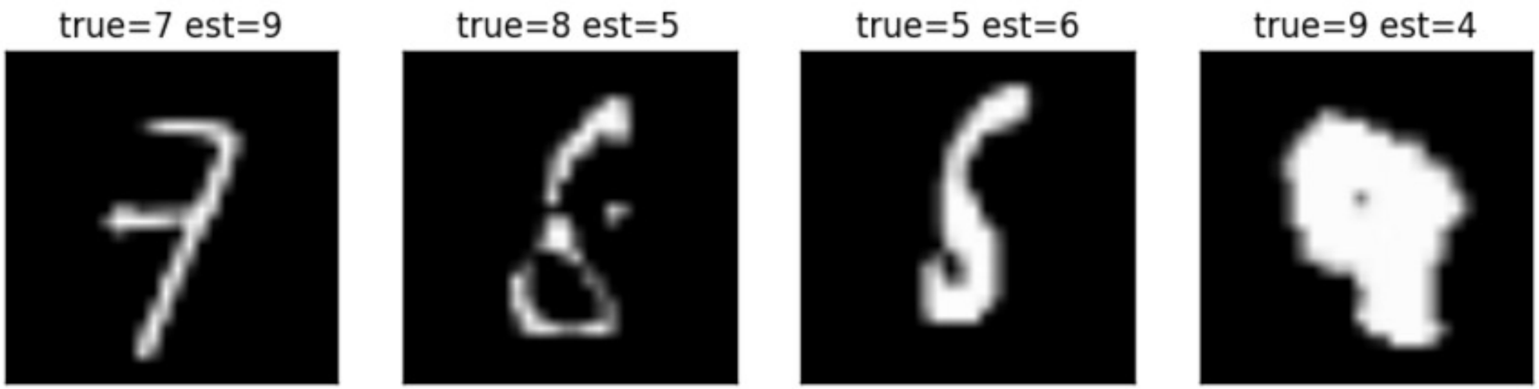
```
[LibSVM]

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



What you should know

- Interpret weights in linear classification of images (logistic regression): Match filters
- Understand the margin in linear classification and maximum margin classifier
- SVM classifier: Allow violation of margin by introducing slack variables (More robust than linear classifier)
- Extend to nonlinear classifier by feature transformation: SVM with nonlinear kernels
- Select SVM parameters from cross-validation