Unit 8
Support Vector Machines

EL-GY 6143/CS-GY 6923: INTRODUCTION TO MACHINE LEARNING

PROF. PEI LIU

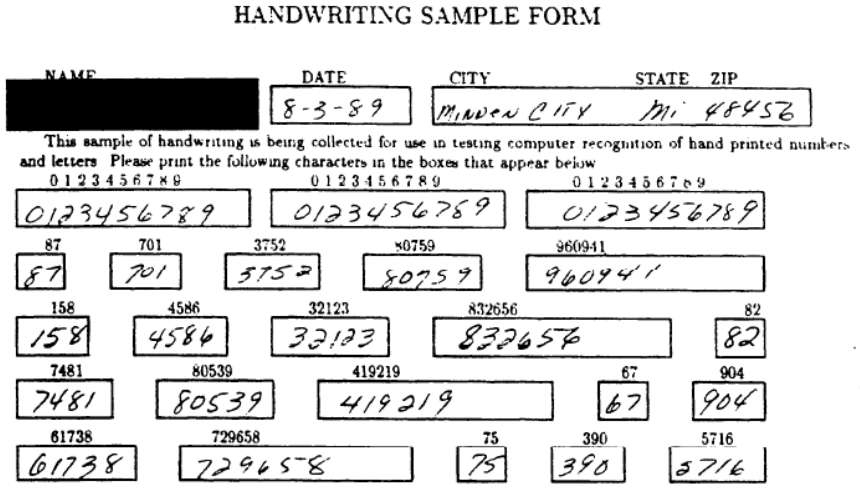
Learning Objectives

- ❑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.
- ❑Write equations for solutions of constrained optimization using the Lagrangian.
- ❑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
- Constrained optimization

MNIST Digit Classification



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

- 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]

Tensorflow and GPU Acceleration

- Tensorflow, which is developed by Google, can be used to training and inference of deep neural networks.
- If you are running the demo on your own computer, you need to install Tensorflow, which can be done in the terminal using command “conda create -n tf tensorflow”
  - It creates a new environment in Anaconda called “tf”
  - Many standard packages are not in, you need to install scikit-learning, matplotlib etc yourself
- If you want to use GPU to accelerate Tensorflow on your computer
  - Windows computer with Nvidia graphics card: It should be straightforward
  - MAC with Apple M1/M1 Pro/M1 Max Chip: You can use the build-in Apple GPU
    - Currently, Anaconda does not support Apple ARM based M1 processor
    - Use Miniforge to install: https://developer.apple.com/metal/tensorflow-plugin/ and https://makeoptim.com/en/deep-learning/tensorflow-metal
- Or, just use Google Colab
  - You might want to consider to pay for Google Colab Pro and pay a monthly fee (about $10)
    - After open your .ipynb, choose “Runtime”->”Change runtime type” and choose “Hardware accelerator=GPU/TPU” and “Runtime shape=High-RAM”

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)
```

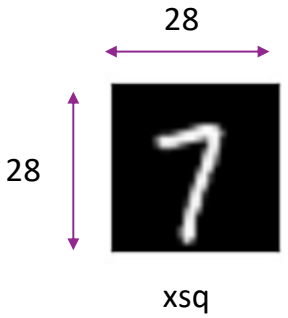
- MNIST data is available in many sources
  - Note: It has been removed from sklearn
- Tensorflow version:
  - 60000 training samples
  - 10000 test samples
- Each sample is a 28 x 28 images
- Grayscale: Pixel values $\in \{0,1,\ldots,255\}$
  - 0 = Black and
  - 255 = White

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))
```




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

$784 = 28^2$

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

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

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

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

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 (lbfgs)
  - Others can be very slow. Even this will take minutes

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

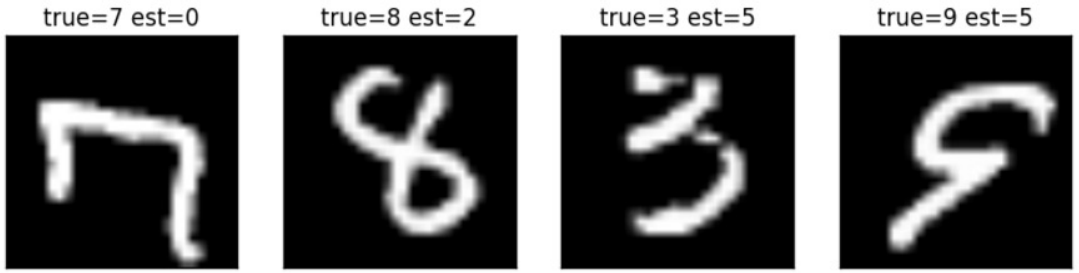
```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758:
e. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.3min remaining:    0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.3min finished
```

Performance

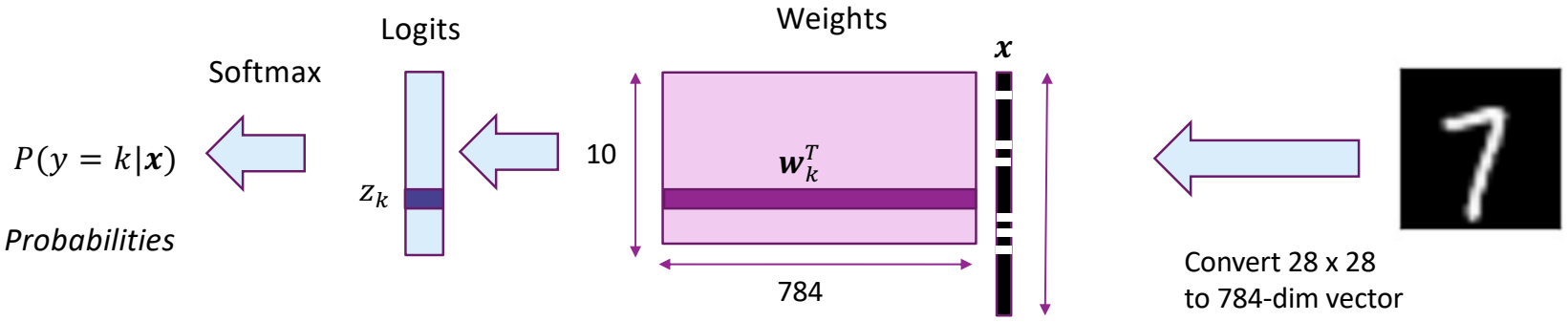
- Accuracy = 89%. Very bad
- Some of the errors seem like they should have been easy to spot
- What went wrong?

```
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
```



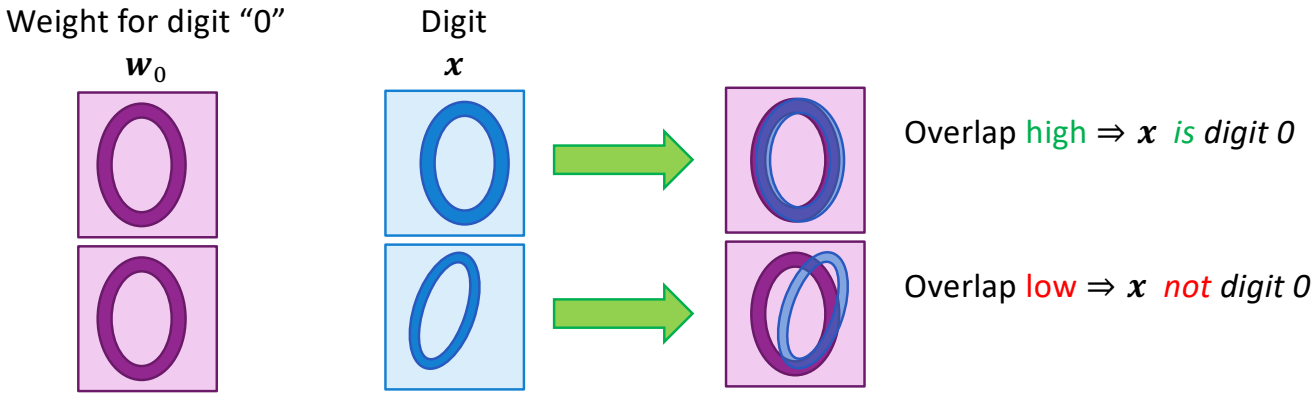
Recap: Logistic Classifier



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

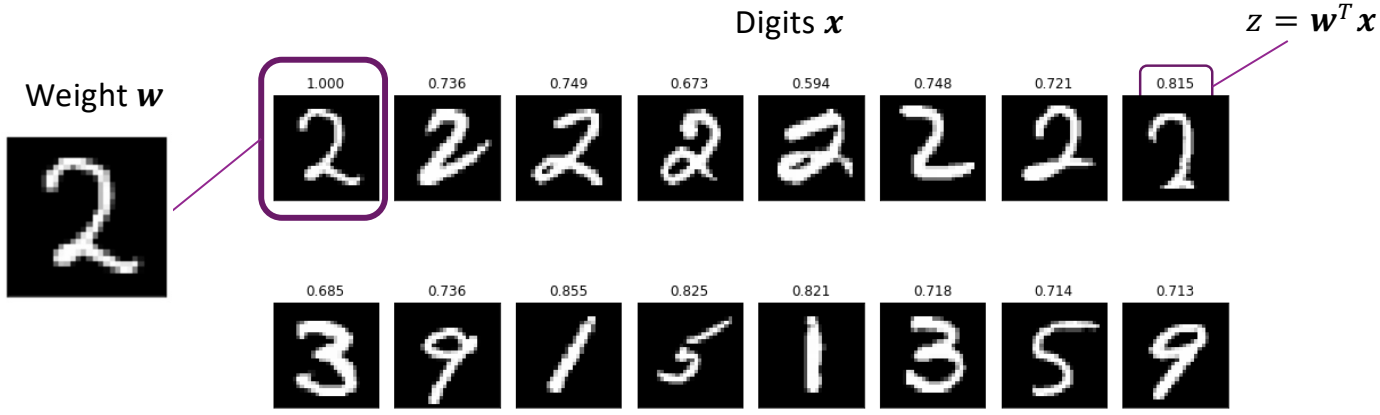
Interpreting the Logistic Classifier Weights

- 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



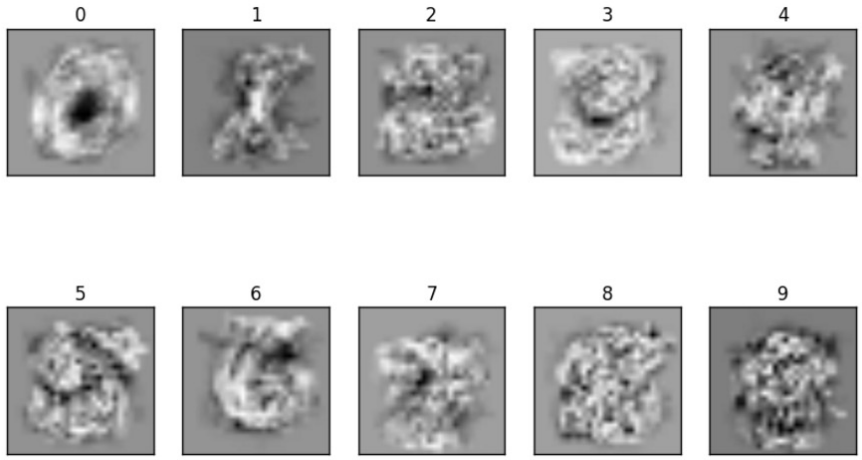
Example with Actual Digits

- 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

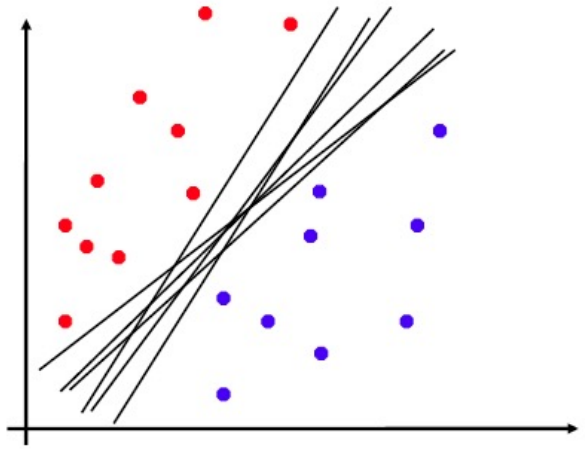
- Linear weighting cannot capture many deformities in image
  - Rotations
  - Translations
  - Variations in relative size of digit components
- Can be improved with preprocessing
  - E.g. deskewing, contrast normalization, many methods
- Is there a better classifier?

Outline

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

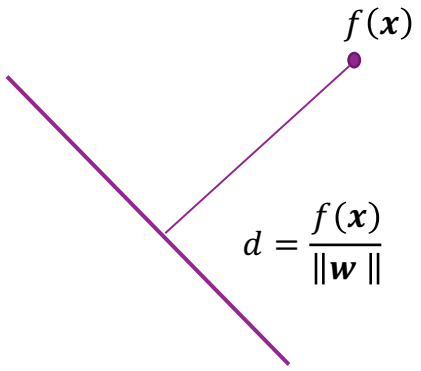
Non-Uniqueness of Separating Plane

- 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.



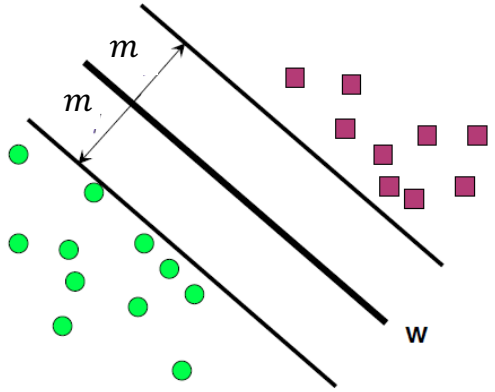
Hyperplane
$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = 0$

Linear Separability and Margin

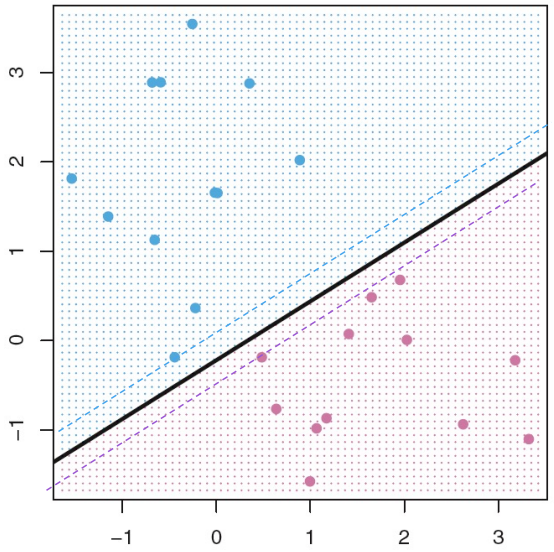
- Given training data $(\boldsymbol{x}_i, y_i), i = 1, \ldots, N$
  - Binary class label: $y_i = \pm 1$
- Suppose it is separable with parameters $(\boldsymbol{w}, b)$
- There must exist a $\gamma > 0$ s.t.:
  - $b + w_1 x_{i1} + \cdots w_d x_{id} > \gamma$ when $y_i = 1$
  - $b + w_1 x_{i1} + \cdots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

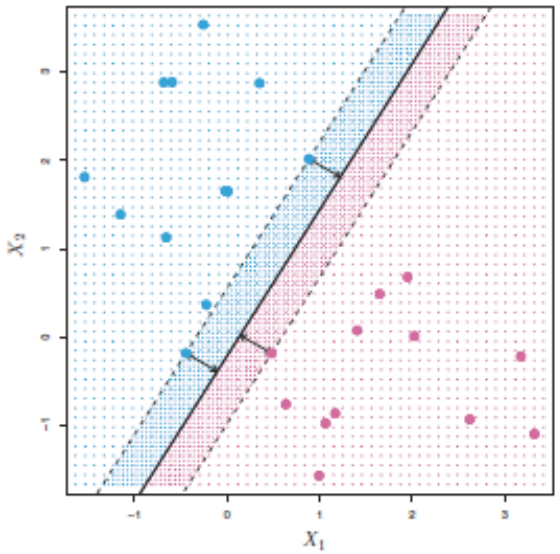
$$y_i(b + w_1 x_{i1} + \cdots w_d x_{id}) > \gamma \text{ for all } i = 1, \ldots, N$$

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

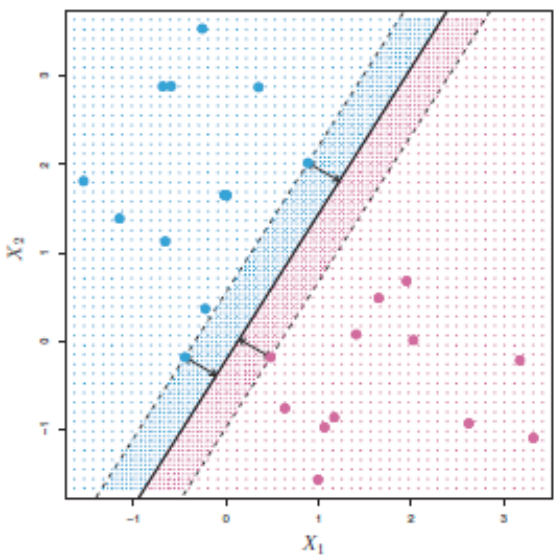
$$\max_{w,b} \gamma$$ ← Maximizes the margin

  - 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.

- 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
  - Only want to correctly separate most points

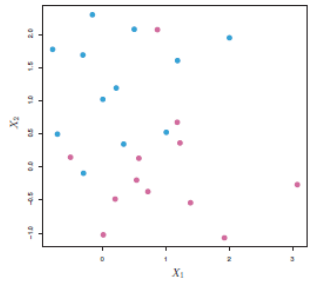


Fig. 9.4

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

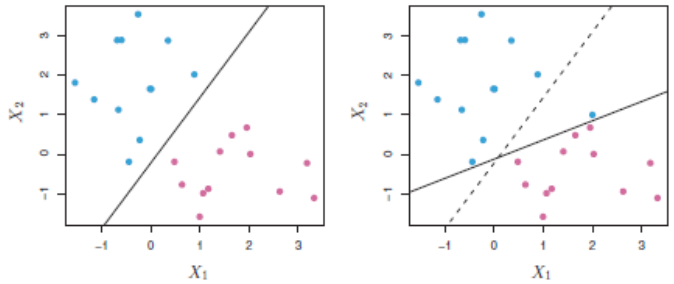


Fig. 9.5

In-Class Exercise

- Found in github site: svm_inclass.ipynb

Problem 1. Margin

For the points below with binary labels:

- Create a scatter plot of the points with different markers for the two classes
- Find the weight and bias of the classifier that separates the two classes
- Compute the distance to the classifier boundary for the points
- Find the margin

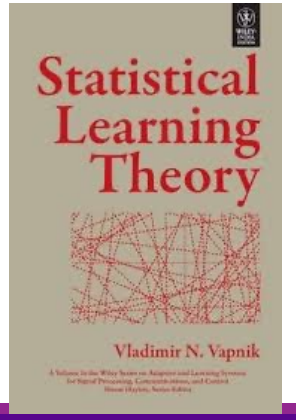
```
1 X = np.array([[0.5,0.5], [1,0.5],[0.5,1.75], [0.75,2.75], [1.1,2.2], [2,1], [3,1.5]])
2 y = np.array([1,1,1,0,0,0,0])
```

Outline

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

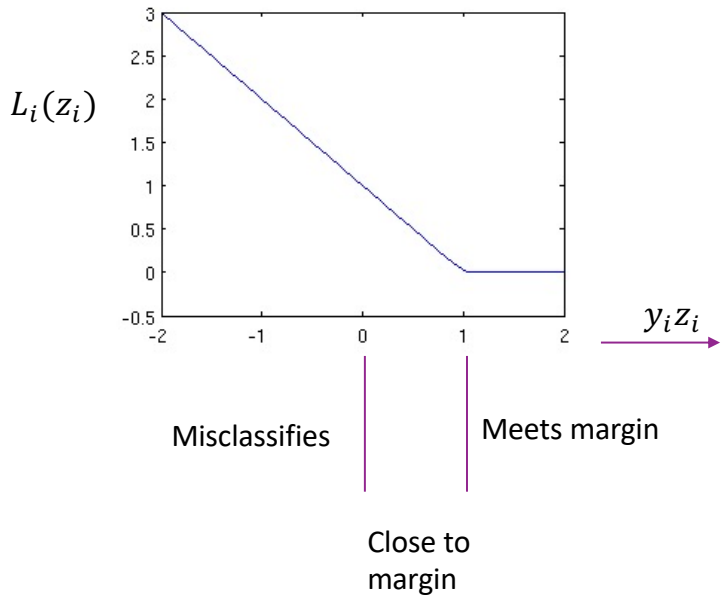
Support Vector Machine

- 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, $y_i z_i \geq 1, \quad z_i = b + \boldsymbol{w}^T\boldsymbol{x}$
- 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

- 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:

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

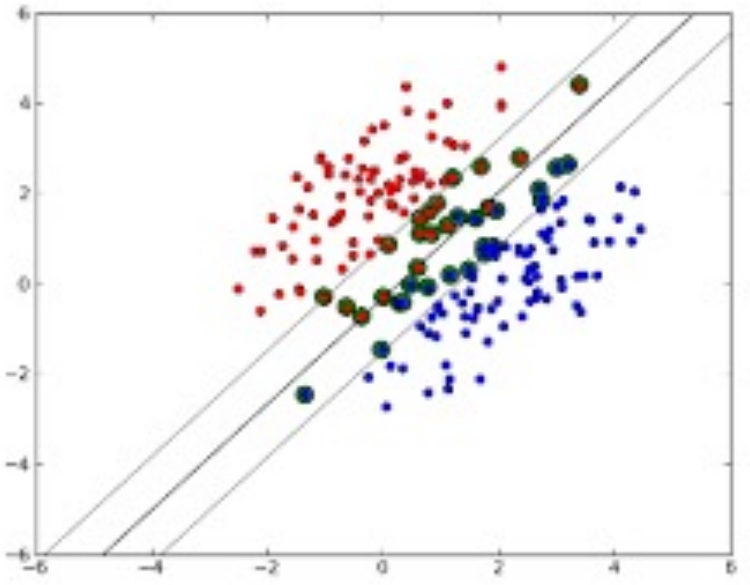
  - ϵ_i = amount sample i misses margin target

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

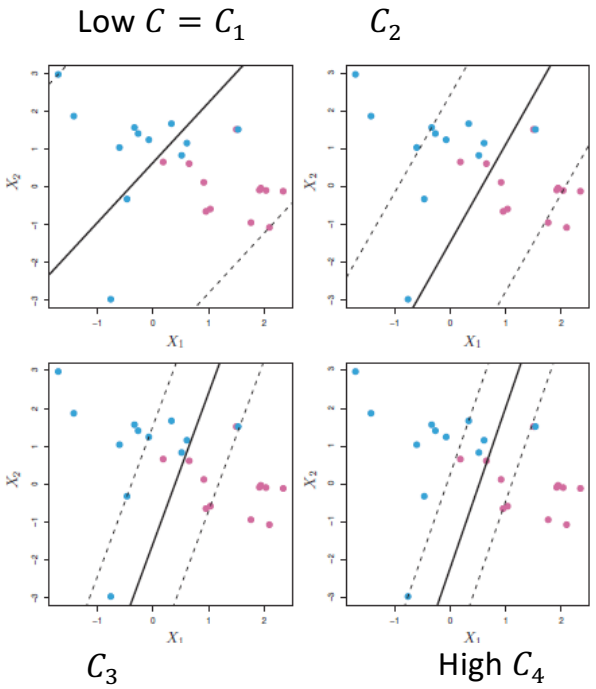
  - $\|\boldsymbol{\epsilon}\|_1 = \sum_{i=1}^{N} \epsilon_i$ called the "one-norm"
  - Generally one-norm would have absolute sign over ϵ_i.
  - But in this case, when the constraint is met, ϵ_i>=0.

Support Vectors

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



Illustrating Effect of C

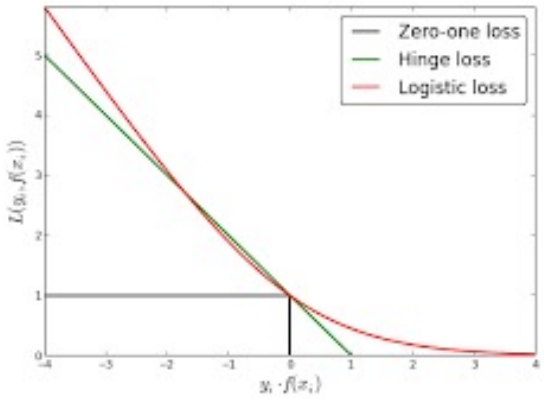


- Fig. 9.7 of ISL
  - Note: C has opposite meaning in ISL than python
  - Here, we use python meaning
- Low C:
  - Leads to large margin
  - But allow many violations of margin.
  - Many more SVs
  - Reduces variance by using more samples
- Large C:
  - Leads to small margin
  - Reduce number of violations, 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:

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



In-Class Exercise

Problem 2. Minimizing the Hinge Loss

For the data below, first create a scatter plot of the points with different markers for the two classes. You should see that the data is not linearly separable.

Then, consider a set of classifiers:

```
yhat = sign(z),    z = w.dot(x)+b
```

Use the the `w` below, plot the hinge loss as a function of the bias `b` where the hinge loss is:

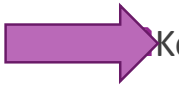
```
J  = sum( maximum(0, 1-ypm*z) )
```

Here `ypm=2*y-1` so that it is a value +1 or -1. Find the `b` that minimizes the hinge loss and plot the boundary of the classifier.

```
1 X = np.array([[0.5,0.5], [1,0.5],[0.5,1.75], [2,2], [0.75,0.75], [0.75,2.75], [1.1,2.2], [2,1], [3,1.5]])
2 y = np.array([1,1,1,1,0,0,0,0,0])
3
4 w = np.array([1.5, 1])
5 w = w / np.linalg.norm(w)
```

Outline

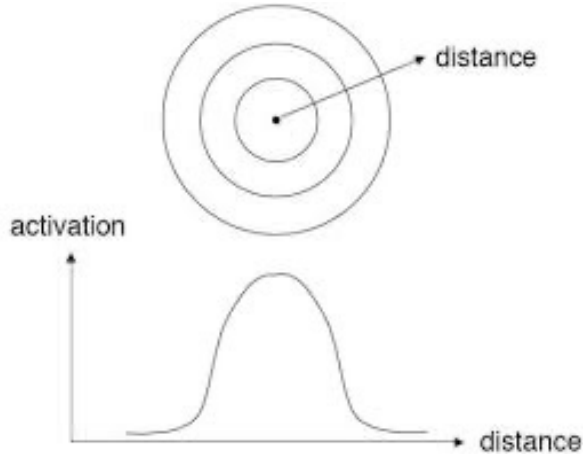
- Motivating example: Recognizing handwritten digits
  - Why logistic regression doesn't work well.
- Maximum margin classifiers
- Support vector machines
- Kernel trick
- Constrained optimization



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 $\boldsymbol{x}_i$

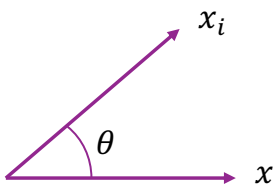
- Typical property
  - $\boldsymbol{x}_i, \boldsymbol{x}$ close $\Rightarrow K(\boldsymbol{x}_i, \boldsymbol{x})$ maximum value
  - $\boldsymbol{x}_i, \boldsymbol{x}$ far $\Rightarrow K(\boldsymbol{x}_i, \boldsymbol{x}) \approx 0$



Common Kernels

❑Linear SVM:

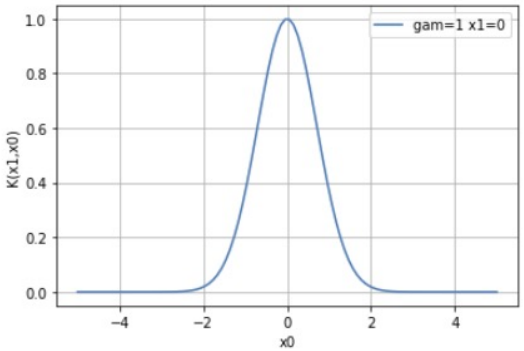
- $K(x_i, x) = x_i^T x = \|x_i\|\|x\| \cos\theta$
- Maximum when angle between vectors is small



❑Radial basis function:

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

- $1/\gamma$ indicates width of kernel

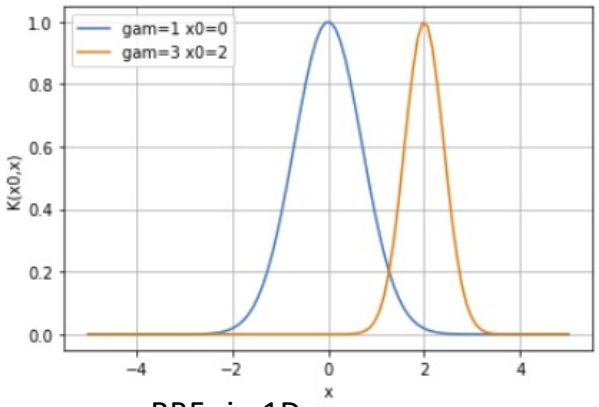


❑Polynomial kernel: $K(x_i, x) = \left|x_i^T x\right|^d$

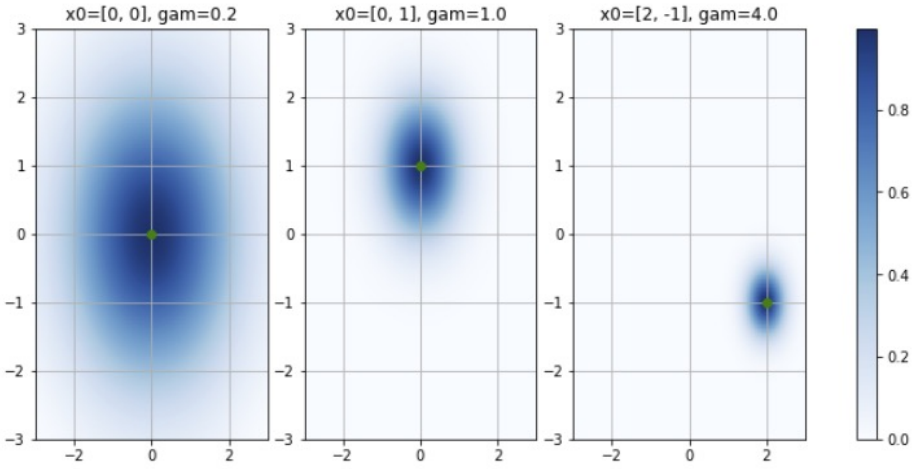
- Typically d=2

RBF Kernel Examples

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



RBFs in 1D



RBFs in 2D

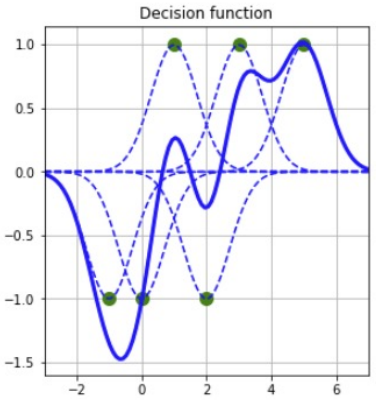
Kernel Classifier

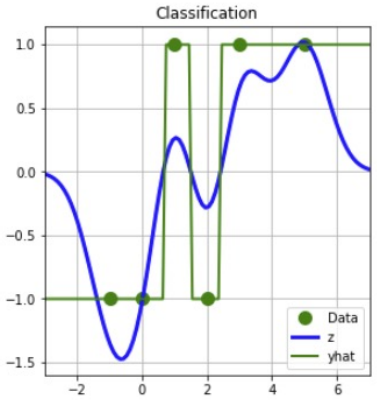
- Given:
  - Training data (x_i, y_i) with binary labels $y_i = \pm 1$
  - Kernel $K(x_i, x)$
- To classify a new point x:
  - Decision function: $z = \sum_{i=1}^{n} y_i K(x_i, x)$
  - Classify: $\hat{y} = sign(z)$
- Idea:
  - z is large positive when x is close to samples x_i with $y_i = 1$
  - z is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

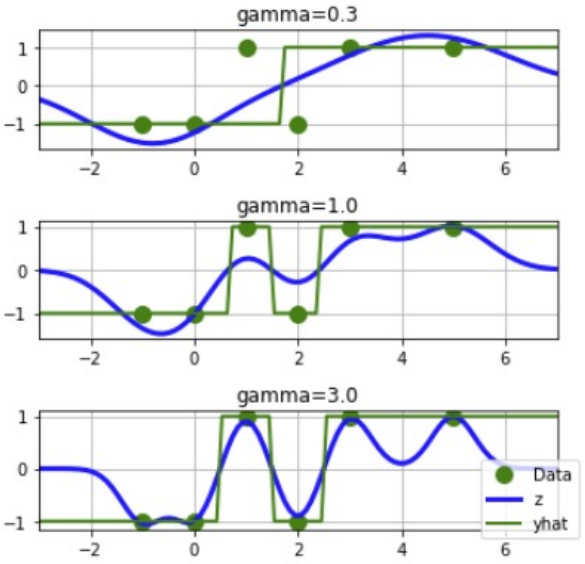
i	1	2	3	4	5	6
x_i	-1	0	1	2	3	5
y_i	-1	-1	1	-1	1	1





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
  - But, higher variance error

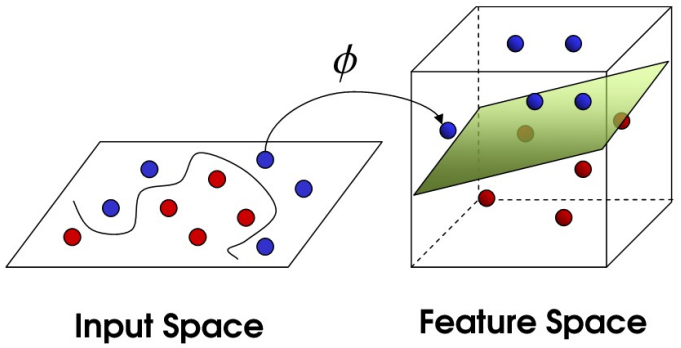


SVMs with Non-Linear Transformations

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

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

- Tries to find separation in a feature space



- 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:

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

  - $\alpha_i \geq 0$ 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:

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

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

Kernel Form of the SVM Classifier

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

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

Decision function

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

Classification decision

- Key point: SVM classifier is approximately Kernel classifier
- But there are two differences:
  - 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):

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

❑Dual parameters: $\alpha_i \geq 0, i = 1, \ldots, N$

- Called the dual parameters due to constrained optimization – see next section

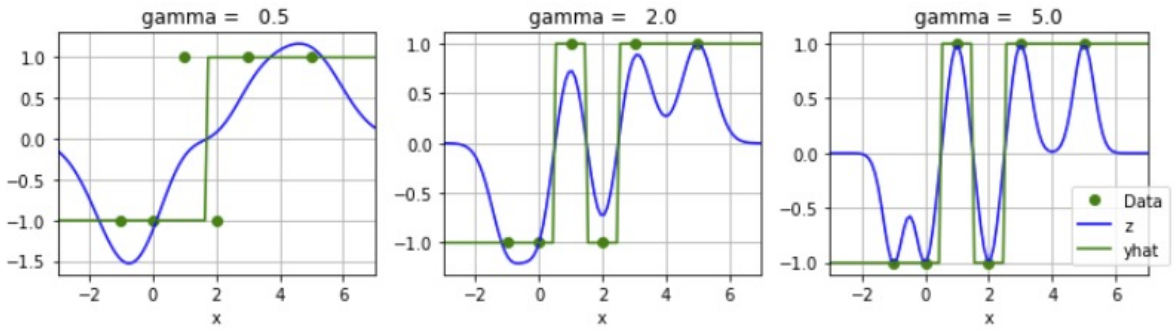
❑Kernel trick:

- 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 $\phi(\boldsymbol{x})$

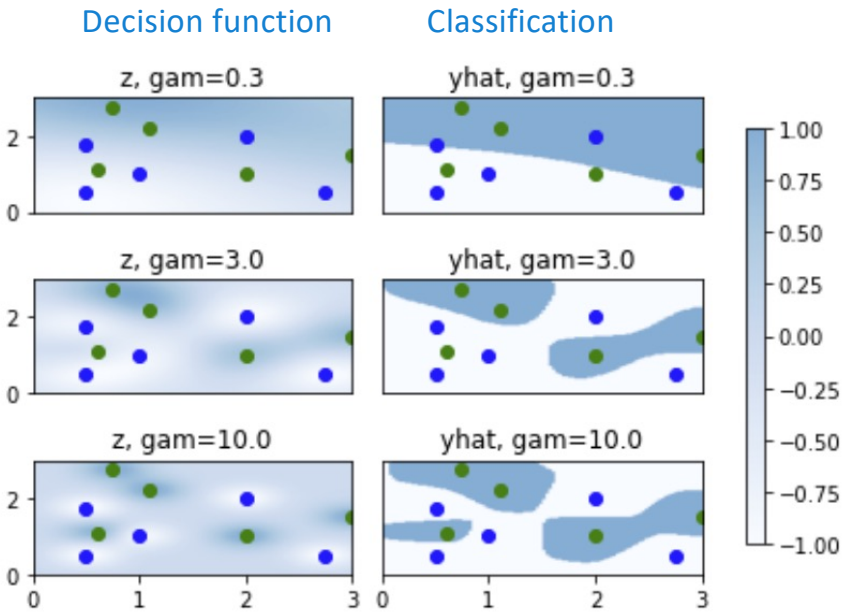
SVM Example in 1D

- 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

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



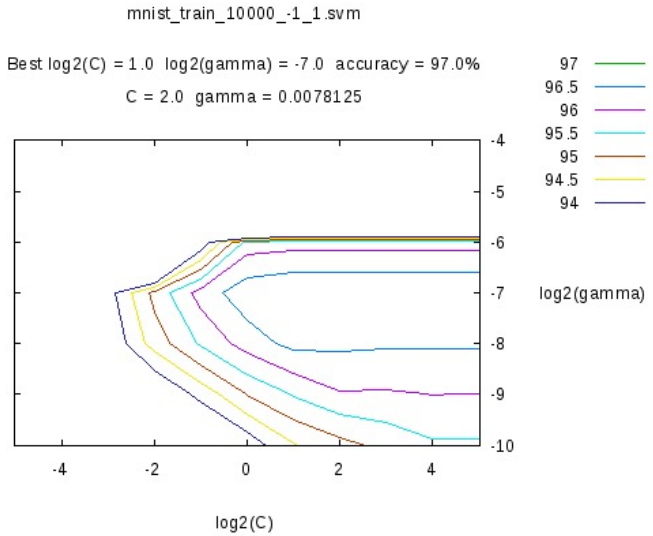
Example in 2D



- 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)
```

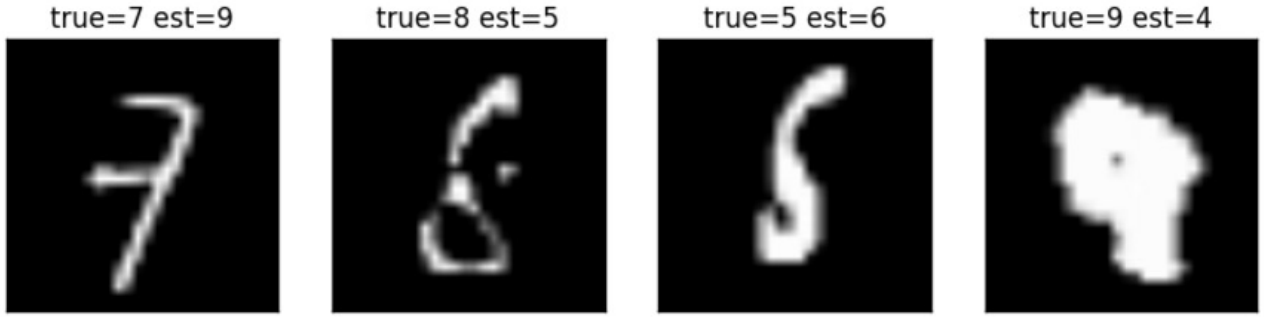
```
[LibSVM]
SVC(C=2.8, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma=0.0073, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=10)
```

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

```
Accuaracy = 0.984000
```

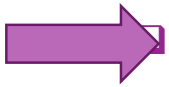
MNIST Errors

- Some of the error are hard even for a human



Outline

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



Constrained Optimization

- In many problems, variables are constrained
- Constrained optimization formulation:
  - Objective: Minimize $f(\boldsymbol{w})$
  - Constraints: $g_1(\boldsymbol{w}) \leq 0, \ldots, g_M(\boldsymbol{w}) \leq 0$
- 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 $g(\boldsymbol{w}) \leq 0$

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

Lagrangian

- Constrained optimization: Min $f(\boldsymbol{w})$ 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 objective 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 satisfy KKT conditions

General Procedure for Single Constraint

- Suppose:
  - $\boldsymbol{w} = (w_1, \ldots, w_d)^T$: 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 $g(\boldsymbol{w}) = 0$ (resulting from setting $\partial L(\boldsymbol{w}, \lambda)/\partial \lambda = 0$)
  - $d + 1$ 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 with illustration

Multiple Constraints

- Now consider constraint: $g(\boldsymbol{w}) = [g_1(\boldsymbol{w}), \ldots, g_M(\boldsymbol{w})]^T \leq 0$.
- 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:
  - $\hat{\boldsymbol{w}}$ minimizes the Lagrangian: $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}, \hat{\lambda})$
  - For each $m = 1, \ldots, M$
    - $g_m(\hat{\boldsymbol{w}}) = 0$ and $\hat{\lambda}_m \geq 0$ [active constraint]
    - $g_m(\hat{\boldsymbol{w}}) < 0$ and $\hat{\lambda}_m = 0$ [inactive constraint]

Multiple Constraints

- If there are M constraints, there could be 2^M cases to discuss
- In practice, the number of cases are much smaller
- For more information on KKT conditions, check the following lecture on youtube
  - UAMathCamp Lecture 40(A): Kuhn-Tucker Conditions: Conceptual and geometric insight
  - https://www.youtube.com/watch?v=HIm3Z0L90Co

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 $i = 1, \ldots, N$

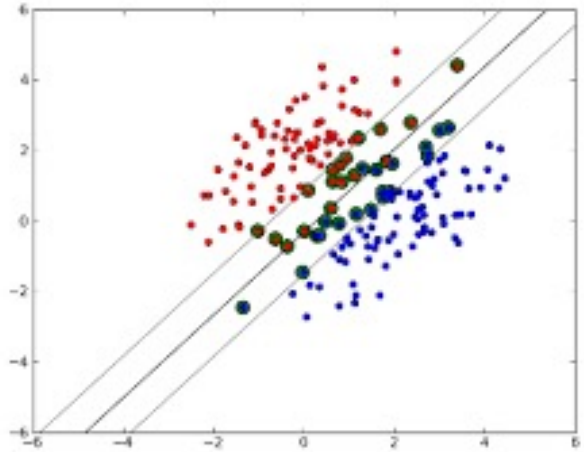
- After applying KKT conditions and some algebra [beyond this class], solution is
  - Optimal weight vector: $\boldsymbol{w} = \sum_{i=1}^{N}\alpha_i y_i \boldsymbol{x}_i$ linear combination of instances
  - Dual parameters α_i minimize

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

  - Details can be found on textbook ESLII, section 12.2.1 Computing the Support Vector Classifier

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 $\alpha_i = 0$



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)
- ❑Solve constrained optimization using the Lagrangian.
  - Understand KKT conditions for a single constraint
- ❑Extend to nonlinear classifier by feature transformation: SVM with nonlinear kernels
- ❑Select SVM parameters from cross-validation