Unit 8 Support Vector Machines

EL-GY 6143: 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





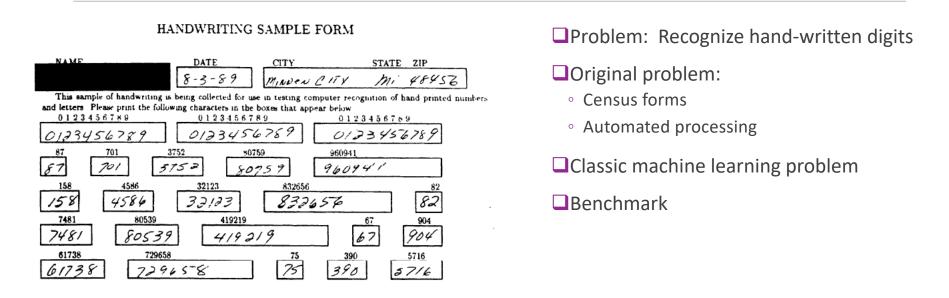
Outline

- 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



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]

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 $\in \{0,1,...,255\}$
 - ∘ 0 = Black and
 - 255 = White

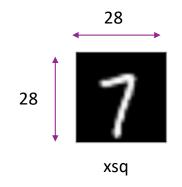


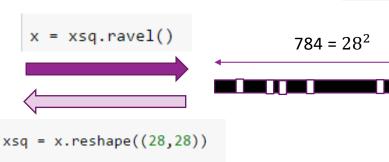


Matrix and Vector Representation

- \square 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







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

$$x = \text{vec}(S) = \begin{bmatrix} x_1 & \cdots & x_{784} \end{bmatrix}$$

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))
                                                 Key command
    plt.imshow(xsq, cmap='Greys_r') ←
    plt.xticks([])
    plt.yticks([])
# Convert data to a matrix
X = mnist.data
v = mnist.target
# Select random digits
                                                 Sample
nplt = 4
nsamp = X.shape[0]
                                                 permutation is
Iperm = np.random.permutation(nsamp)
                                                 necessary for this
# Plot the images using the subplot command
                                                 dataset, as the
for i in range(nplt):
                                                 original data is
    ind = Iperm[i]
    plt.subplot(1,nplt,i+1)
                                                 ordered by digits
    plt_digit(X[ind,:])
```



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

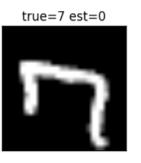


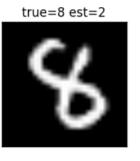
Performance

- □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

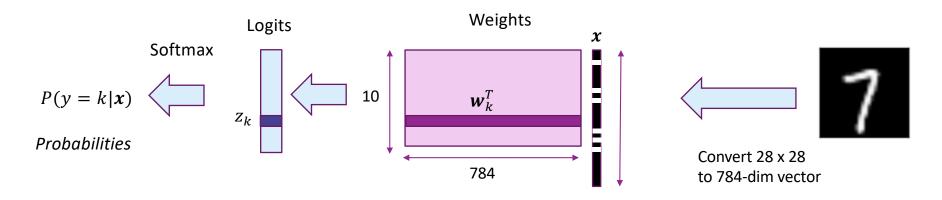








Recap: Logistic Classifier

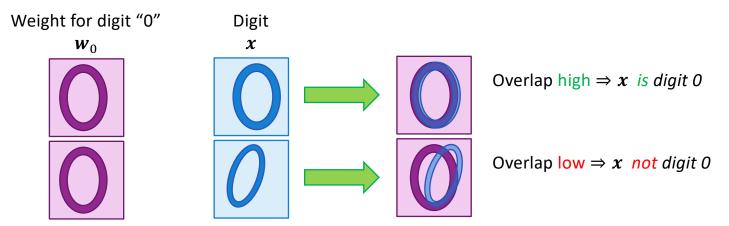


- □ Each logit $z_k = \boldsymbol{w}_k^T \boldsymbol{x}$ = inner product with weight \boldsymbol{w}_k with digit \boldsymbol{x} , k = 0, ..., 9
- $\square \text{Will select } \hat{y} = \arg \max_{k} P(y = k | x) = \arg \max_{k} z_{k}$
 - \circ Output z_k which is largest
- \square When is z_k large?



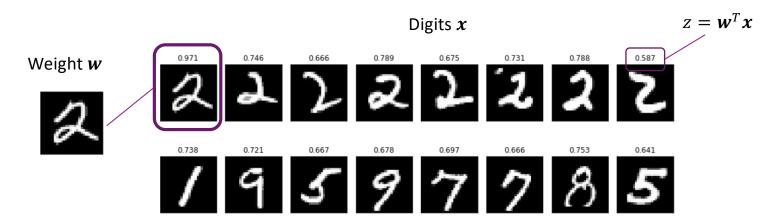
Interpreting the Logistic Classifier Weights

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



Example with Actual Digits

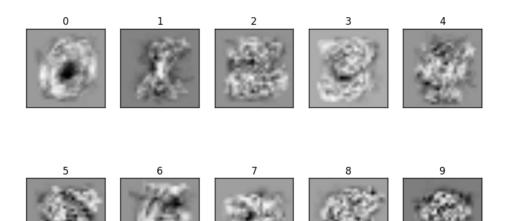
- \square Take weight w from a random digit "2"
- □Inner products $z = w^T x$ are only slightly higher for other digits "2"
- \Box Cannot tell which digit is correct from the inner product $z = w^T x$





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?





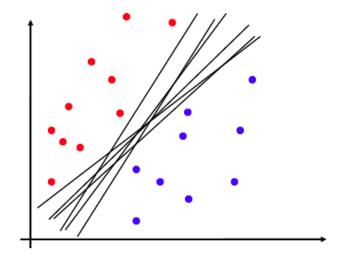
Outline

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



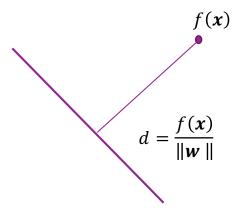
Non-Uniqueness of Separating Plane

- ☐ 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: $f(x) = w^T x + b, x \in \mathbb{R}^d$
- \square Hyperplane in d-dimensional: f(x) = 0
- ☐Parameters:
 - \circ Weight w and bias b
 - Unique up to scaling:
 - \circ (b, w) and $(\alpha b, \alpha w)$ define the same plane.
 - For unique definition, we can require ||w||=1.
- □ Distance of any point **x** to the hyperplane:
 - d = f(x)/||w||, where $f(x) = b + w^T x$.
 - ∘ See ESL Sec. 4.5.
 - ESL: Hastie, Tibshirani, Friedman, "The Elements of Statistical Learning". 2nd Ed. Springer.



Hyperplane

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

Linear Separability and Margin

- \square Given training data (x_i, y_i) , i = 1, ..., N
 - Binary class label: $y_i = \pm 1$
- \square Suppose it is separable with parameters (w, b)
- □ There must exist a $\gamma > 0$ s.t.:

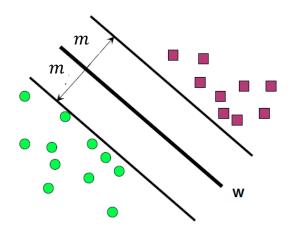
•
$$b + w_1 x_{i1} + \cdots w_d x_{id} > \gamma$$
 when $y_i = 1$

•
$$b + w_1 x_{i1} + \cdots w_d x_{id} < -\gamma$$
 when $y_i = -1$

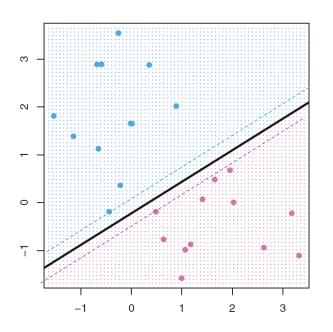
■Single equation form:

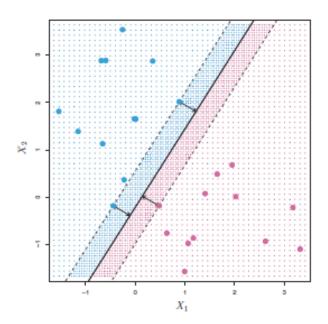
$$y_i(b + w_1x_{i1} + \cdots w_dx_{id}) > \gamma \text{ for all } i = 1, ..., N$$

- \square Margin: $m = \frac{\gamma}{\|w\|}$: minimal distance of a sample to the plane
 - \circ 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

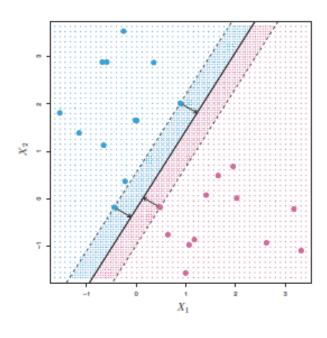
- ☐ For the classifier to be more robust to noise, we want to maximize the margin!
- □ Define maximum margin classifier

- □ 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 midline 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

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

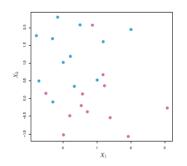
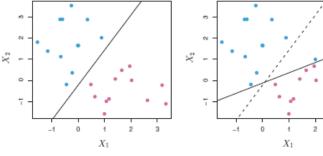


Fig. 9.4





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

```
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]])
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

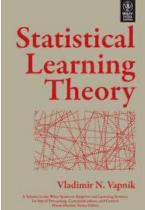




Support Vector Machine

- ■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

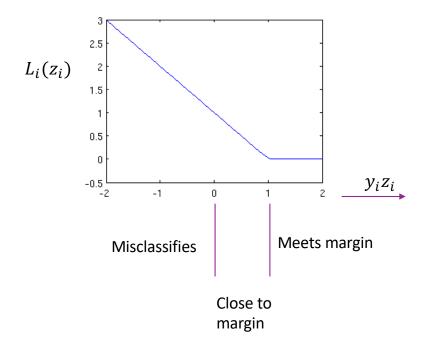






Hinge Loss

- \Box Fix $\gamma = 1$
- □Want ideally: $y_i(\mathbf{w}^T\mathbf{x} + b) \ge 1$ for all samples i
 - \circ Equivalently, $y_i z_i \ge 1$, $z_i = b + \mathbf{w}^T \mathbf{x}$
- ☐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:
 - $y_i z_i \ge 1 \implies$ Sample meets margin target, $L_i(w) = 0$
 - $y_i z_i \in [0,1) \Rightarrow \text{Sample margin too small, small loss}$
 - $\circ~y_iz_i \leq 0~\Rightarrow$ Sample misclassified, large loss



SVM Optimization

- \square Given data (x_i, y_i)

$$J(w,b) = C \sum_{i=1}^{N} \max(0,1 - y_i(w^T x_i + b)) + \frac{1}{2} ||w||^2$$
Fold final margin. Hinge loss term. margin=1/||w||

C controls final margin

Hinge loss term Attempts to reduce Misclassifications

- \square 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:

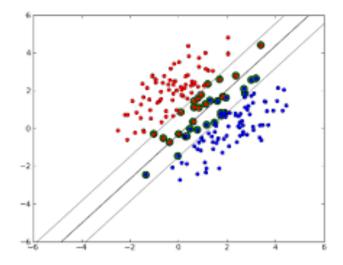
$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \ge 1 - \epsilon_i$$
 for all $i = 1, ..., N$

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

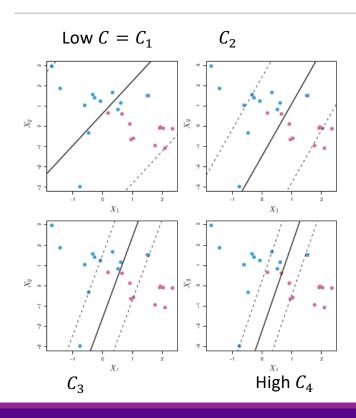


Support Vectors

- □Support vectors: Samples that either:
 - Are exactly on margin: $y_i(\mathbf{w}^T \mathbf{x}_i + b) = 1$
 - \circ Or, on wrong side of margin: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 1$
- ☐ 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

\square Low C:

- Leads to large margin
- But allow many violations of margin.
- Many more SVs
- Reduces variance by using more samples

☐ Large C:

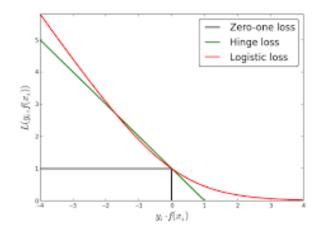
- Leads to small margin
- Reduce number of violations, and fewer SVs.
- Highly fit to data. Low bias, higher variance
- More chance to overfit



Relation to Logistic Regression

□ Logistic regression also minimizes a loss function:

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



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.

```
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]])
y = np.array([1,1,1,1,0,0,0,0,0])

w = np.array([1.5, 1])
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



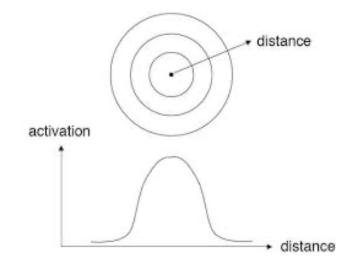
The Kernel Function

■Kernel function:

- Function $K(x_i, x)$
- Key function for SVMs and kernel classifiers
- $^{\circ}$ Measures "similarity" between new sample $oldsymbol{x}$ and training sample $oldsymbol{x}_i$

☐ Typical property

- $\cdot x_i, x \text{ close} \Rightarrow K(x_i, x) \text{ maximum value}$
- $x_i, x \text{ far} \Rightarrow K(x_i, x) \approx 0$



Common Kernels

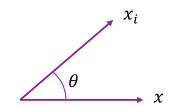
☐Linear SVM:

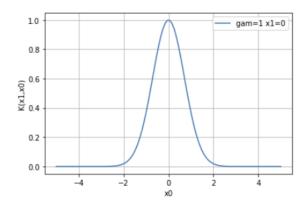
- $\circ K(x_i, x) = x_i^T x = ||x_i|| ||x|| \cos \theta$
- Maximum when angle between vectors is small
- □ Radial basis function:

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

 $\circ~1/\gamma$ indicates width of kernel

• Typically d=2

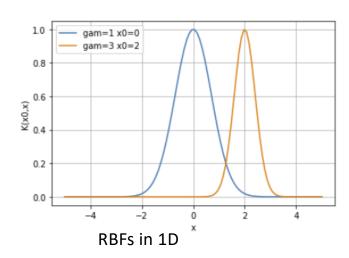


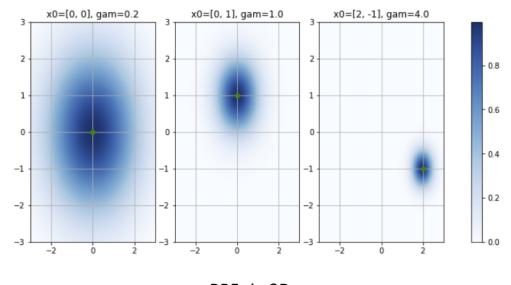


RBF Kernel Examples

□RBF kernel: $K(x_0, x) = \exp[-\gamma ||x - x_0||^2]$

- \circ Peak value of 1 at $x=x_0$
- Width $\propto \frac{1}{\gamma}$









Kernel Classifier

□Given:

- Training data (x_i, y_i) with binary labels $y_i = \pm 1$
- Kernel $K(x_i, x)$

\square 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:

- $\circ z$ 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

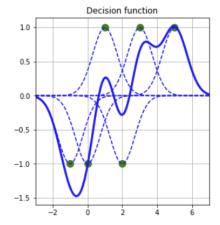
- \blacksquare Example data with 6 points (x_i, y_i)
 - RBF kernel: $K(x_i, x) = e^{-\gamma(x_i x)^2}$, $\gamma = 1$
- □ Decision function:

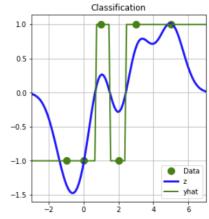
$$\circ z = \sum_{i=1}^n y_i K(x_i, x)$$

- Sum of bell curves
- Positive when near positive samples
- Negative when near negative samples
- **Classification:**

$$\circ \hat{y} = sign(z)$$

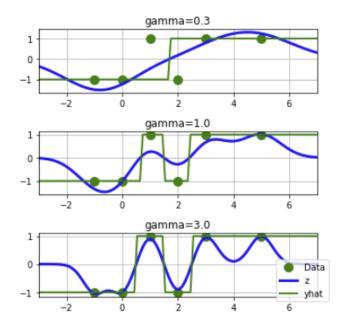
i	1	2	3	4	5	6
x_i	-1	0	1	2	3	5
y_i	-1	-1	1	-1	1	1





Effect of Gamma

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



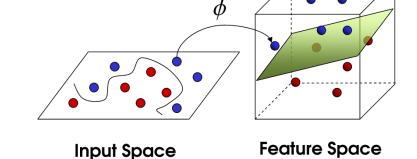
SVMs with Non-Linear Transformations

■ Non-linear transformation:

- Replace x with $\phi(x)$
- 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

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

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

- $\alpha_i \geq 0$ 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
- \square Consequence: The linear discriminant on any other sample x is:

$$z = b + \mathbf{w}^{T} \phi(\mathbf{x}) = b + \sum_{i=1}^{N} \alpha_{i} y_{i} \boxed{\phi(\mathbf{x}_{i})^{T} \phi(\mathbf{x})} - K(\mathbf{x}_{i}, \mathbf{x}) = \text{"kernel"}$$

Kernel Form of the SVM Classifier

□SVM classifier can be written with the kernel $K(x_i, x)$ and values $\alpha_i \ge 0$:

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

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

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

$$\hat{y} = \operatorname{sign}(z)$$

- □ Dual parameters: $\alpha_i \ge 0, i = 1, ..., N$
 - Called the dual parameters due to constrained optimization see next section
- ☐Kernel trick:
 - $^{\circ}$ Directly solve the parameters lpha instead of the weights w
 - \circ Can show that the optimization only needs the kernel $K(x_i, x)$
 - Does not need to explicitly use $\phi(x)$



SVM Example in 1D

- ☐ Same data as in the Kernel classifier example
- \square Fit SVM with RBF with different γ
- \square 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

gamma = 0.5	gamma = 2.0	gamma = 5.0		
10	1.0	1.0		
0.5	0.5	0.5		
-0.5	0.0	0.0		
-1.0	-0.5 -1.0	-0.5 Data z vhat		

2

0

-1

-1

-1

 x_i

 y_i

3

1

1





5

1

6

5

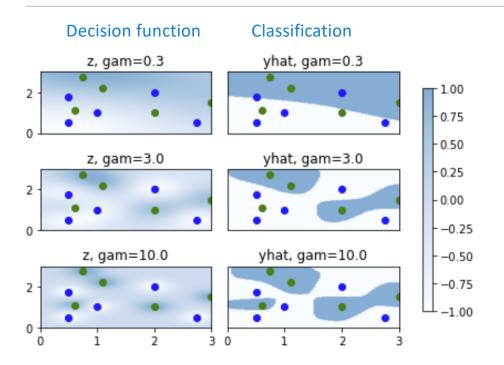
1

4

2

-1

Example in 2D



■Example:

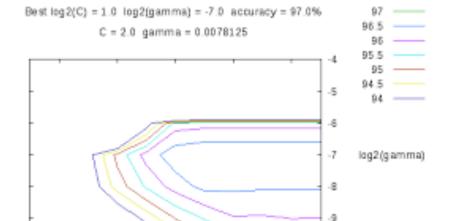
- 10 data points with binary labels
- \circ Fit SVM with C=1 and RBF
- $\gamma = 0.3, 3 \text{ and } 10$

□Plot:

- z= linear discriminant
- $\hat{y} = sign(z) = classification decision$
- \square 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
 - \circ Kernel width $\gamma > 0$
- \square 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



mnist_train_10000_-1_1.svm

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

log2(C)

-2





-10

Multi-Class SVMs

- \square 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:
 - \circ Train K SVMs: train each class k against all other classes
 - \circ 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
- \square Accuracy ≈ 0.984
 - Much better than logistic regression
- □Can get better with:
 - pre-processing
 - More training data
 - Optimal parameter selection

```
# 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)
```

```
Accuaracy = 0.984000
```

from sklearn import svm

yhat1 = svc.predict(Xts)
acc = np.mean(yhat1 == yts)

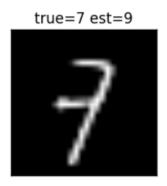
print('Accuaracy = {0:f}'.format(acc))

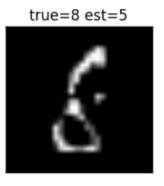


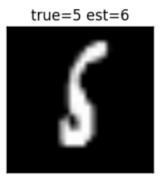


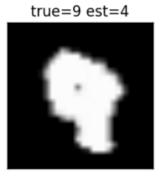
MNIST Errors

■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(w)
 - Constraints: $g_1(\mathbf{w}) \leq 0, ..., g_M(\mathbf{w}) \leq 0$
- ■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
- \square Often write constraints in vector form: Write $g(w) \leq 0$

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



Lagrangian

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

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

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



KKT Conditions

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



General Procedure for Single Constraint

■Suppose:

- $\mathbf{w} = (w_1, ..., w_d)^T$: d unknown primal variables
- ∘ $g(\mathbf{w}) \le 0$: scalar constraint

□ Case 1: Assume constraint is active:

- Solve \mathbf{w} and λ : $\partial L(\mathbf{w}, \lambda)/\partial w_i = 0$ and $g(\mathbf{w}) = 0$ (resulting from setting $\partial L(\mathbf{w}, \lambda)/\partial \lambda = 0$)
- $\circ d + 1$ unknowns and d + 1 equations
- \circ Verify that $\lambda \geq 0$

□ Case 2: Assume constraint is inactive

- Solve primal objective $\partial f(\mathbf{w})/\partial w_i = 0$ ignoring constraint
- $\circ \ d$ unknowns and d equations
- Verify that constraint is satisfied: $g(\mathbf{w}) \leq 0$

KKT Conditions Illustrated

■ Example 1: Constraint is "active"

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

■ Example 2: Constraint is "inactive"

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

- ☐ Examples worked on board with illustration
- ☐ For more information on KKT conditions, check the following lecture on youtube
 - <u>UAMathCamp</u> Lecture 40(A): Kuhn-Tucker Conditions: Conceptual and geometric insight
 - https://www.youtube.com/watch?v=HIm3Z0L90Co

Multiple Constraints

- □Now consider constraint: $g(\mathbf{w}) = [g_1(\mathbf{w}), ..., g_M(\mathbf{w})]^T \le 0$.
- ☐ Lagrangian is:

$$L(\mathbf{w}, \lambda) = f(\mathbf{w}) + \lambda^{T} g(\mathbf{w}) = f(\mathbf{w}) + \sum_{m=1}^{M} \lambda_{m} g_{m}(\mathbf{w})$$

- \circ Weighted sum of all M constraints
- \circ λ is called the dual vector
- □KKT conditions extend to:
 - $\widehat{\boldsymbol{w}}$ minimizes the Lagrangian: $\widehat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}, \widehat{\lambda})$
 - \circ For each $m=1,\ldots,M$
 - $g_m(\widehat{\pmb{w}}) = 0$ and $\hat{\lambda}_m \geq 0$ [active constraint]
 - $g_m(\widehat{\pmb{w}}) < 0$ and $\hat{\lambda}_m = 0$ [inactive constraint]

SVM Constrained Optimization

□ Recall: SVM constrained optimization

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

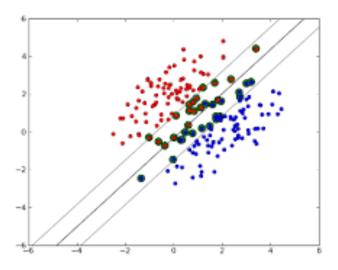
- Constraints: $y_i(\mathbf{w}^T \mathbf{x}_i + b) \ge 1 \epsilon_i$ and $\epsilon_i \ge 0$ for all i = 1, ..., N
- □ After applying KKT conditions and some algebra [beyond this class], solution is
 - \circ Optimal weight vector: $m{w} = \sum_{i=1}^N lpha_i y_i x_i$ linear combination of instances
 - \circ Dual parameters α_i minimize

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



Support Vectors

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



