Machine Intelligence & Deep Learning Workshop

Machine Learning and Pattern Recognition Fundamentals

Majid Rabbani

Visiting Professor

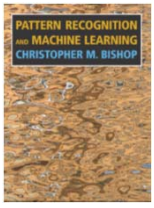
The Kate Gleason COLLEGE OF ENGINEERING

Some Reference Books Used in My Lecture

- S. Theodoridis and K. Koutroumbas, "*Pattern Recognition*", Academic Press, 4th Edition, 2009, ISBN: 9781597492720

- R. Duda, P. Hart, and D. Stork, "*Pattern Classification*", 2nd Edition, John Wiley & Sons, Inc. ISBN 0-471-05669-3

- Christopher Bishop, "*Pattern Recognition and Machine Learning.*" Can be downloaded for free from: http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf



Classification vs Regression

Bayesian Classification

Error Measures

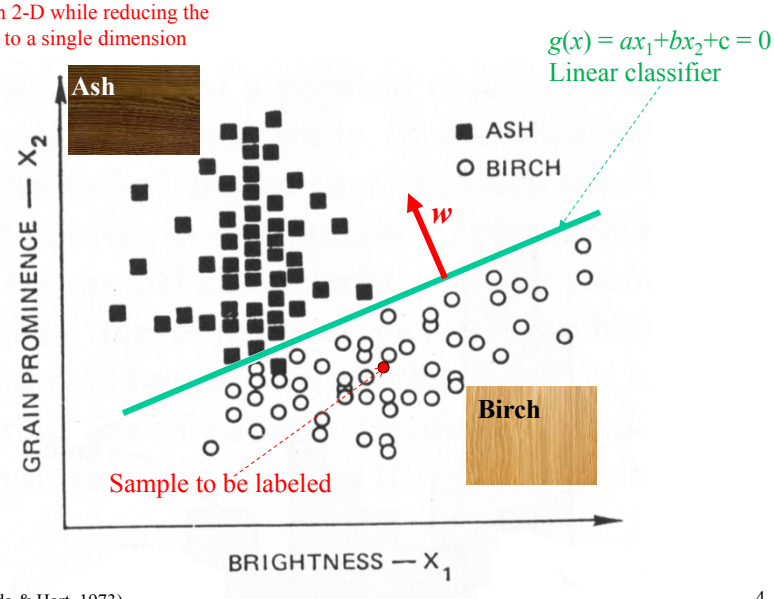
Validation

Nearest Neighbor Classification

Dimensionality Reduction (PCA, FLD)

Note that projecting the data in the direction of *w* appears to provide almost as good of a separation in 1-D as in 2-D while reducing the problem to a single dimension

Classification Task



(Duda & Hart, 1973)

In-Sample vs Out-of-Sample Performance: Generalization

- In most practical situations, we are given a set of training samples that we use to build a classifier to classify unseen data.
- We can build as sophisticated of a classifier as we wish, typically by introducing a large number of model parameters, to the extent that the error on training data can even be reduced to zero. However, is this the right thing to do?
- What we are really interested in is good performance on data that we have not seen yet. This is called **generalization**. Ideally, we'd like to build a classifier that performs reasonably well on the training data but also generalizes well to the data outside the training set.
- It turns out that simpler models generalize better than the more sophisticated models, because the sophisticated models often tend to **overfit** the data.

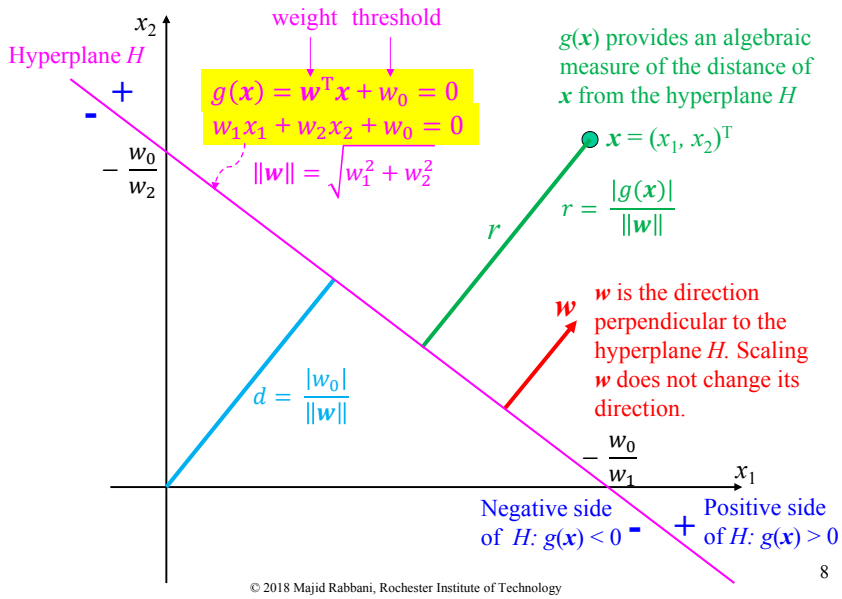
Classification Techniques

- We will briefly review the following paradigms in machine learning:
  - Bayesian Classifier
  - Nearest Neighbor Classifier
  - Linear Classifiers (Linear Discriminant Functions or LDF's)
  - Support Vector Machines (SVM)
  - Adaptive Boosting or AdaBoost
  - Neural Networks

Feature Vectors and Multiple Classes

- In general, our data comes from one of *C* classes or categories. However, in many of our developments, we will consider a two-class problem because it is easier to analyze and/or to visualize.
  - In some cases the solution easily extends to multiple classes in a straightforward manner such as the Bayesian classifier.
  - In some other cases, such as SVM, we extend our solution to multi-class problems by either using "pairwise" (and pick the class with the highest vote) approach, or "one against all" approach.
- In general, our feature space will be *d*-dimensional, where *d* can be as large as several orders of magnitude. However, in most of our illustrations we will use a 2-D feature space because it can be visualized easily (e.g., a linear classifier is a line in 2-D, a plane in 3-D but a hyperplane for $d > 3$). However, most paradigms can be extended to a large *d* in a straightforward manner unless stated otherwise.

Describing a Hyperplane



Classification, Linear Regression, Logistic Regression

- **Classification**: predicts the class of a data sample and has a discrete value. For example, in a 2-class problem, the input sample can be a d-dimensional feature vector $\boldsymbol{x}$, while the output of the classifier is a label $y = +1$ or -1.
  - An example is the task of credit approval. The input features can be factors such as age, gender, annual salary, years in residence, years in job, current debt, etc., while the output would be a binary "yes" or "no."
- **Regression**: is used to predict continuous values instead of discrete.
  - An example is credit line approval. Although the input features could be the same as before, instead of a binary "yes' or "no" answer, the output is a continuous number, i.e., the dollar amount of approved credit.
- **Logistic Regression**: is used when the input is categorical in nature (as in classification) but the output has a limited range of possible values such as the probability that the data point belongs to a given class.
  - An example is using SVM but then taking the output of the discriminant function (distance from the SVM hyperplane) through a sigmoidal function (output range of 0 to 1) to predict the class label probability.

Bayesian Classifier

- In some cases, the probability distribution of the feature vector for each class (i.e., probability of the feature vector conditioned on a given class), $p(\boldsymbol{x}|\omega_i)$, is known or can be approximated well.
- In such cases, if the prior probability of each class, $P(\omega_i)$, is also known (e.g., *priors*), the Bayes formula can be used to find the conditional probability of each class given the feature vector, $P(\omega_i|x)$.
- If the priors are not known, they can be assumed to be equal.
- In the Bayesian case, the classification task becomes very simple: Label the data as that class i that has the highest value of $P(\omega_i|x)$.
- The Bayesian classifier has the **best possible performance** among all classifiers on out-of-sample data and acts as the gold standard.
- A popular approach is: model the training data as samples of a normal distribution, estimate the distribution parameters from the available data, and use the resulting Bayes classifier to classify incoming data.

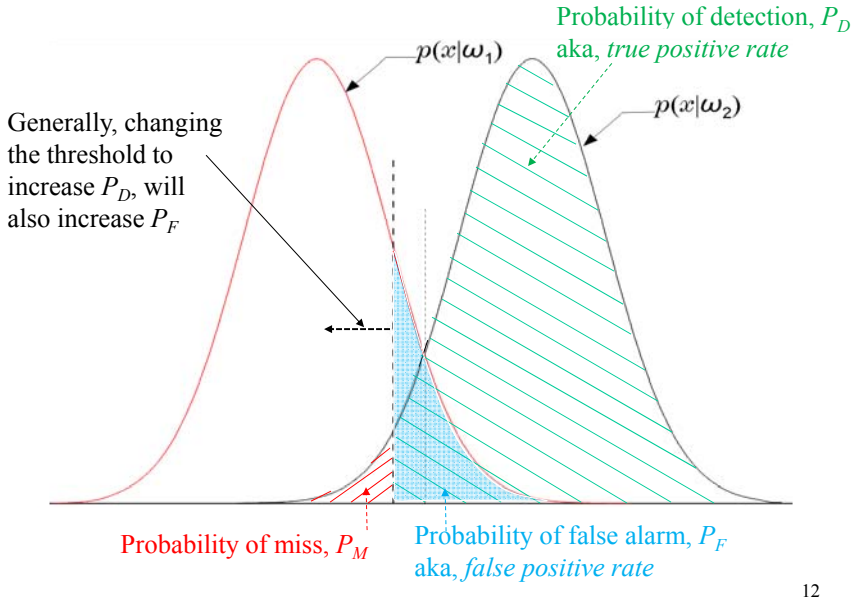
Example of Bayes Statistics

- Suppose that you are worried that you might have a rare disease. You decide to get tested, and suppose that the testing methods for this disease are correct 99% of the time (in other words, if you have the disease, it shows that you do with 99% probability, and if you don't have the disease, it shows that you do not with 99% probability). Suppose this disease is actually quite rare, occurring randomly in the general population in only 1 of every 10,000 people.
- If your test results come back positive, what are the approximate chances that you actually have the disease?

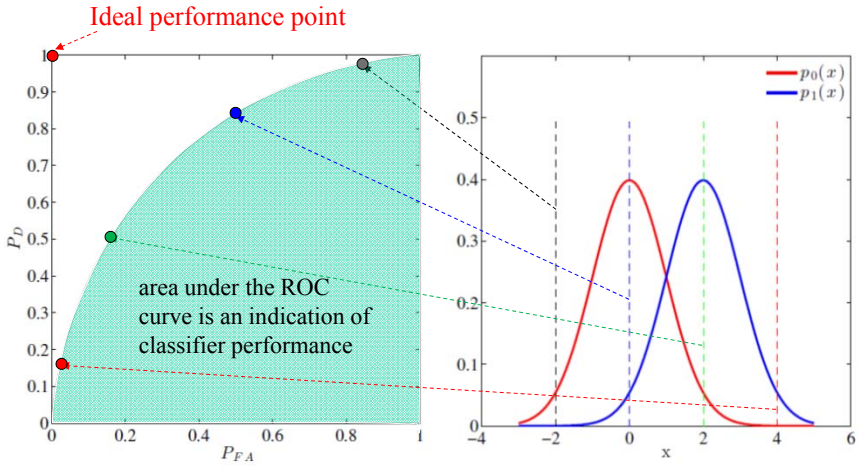
  (a) 0.99, (b) 0.90, (c) 0.10, (d) 0.01?

The answer is (d) or 0.0098!

Error Metrics

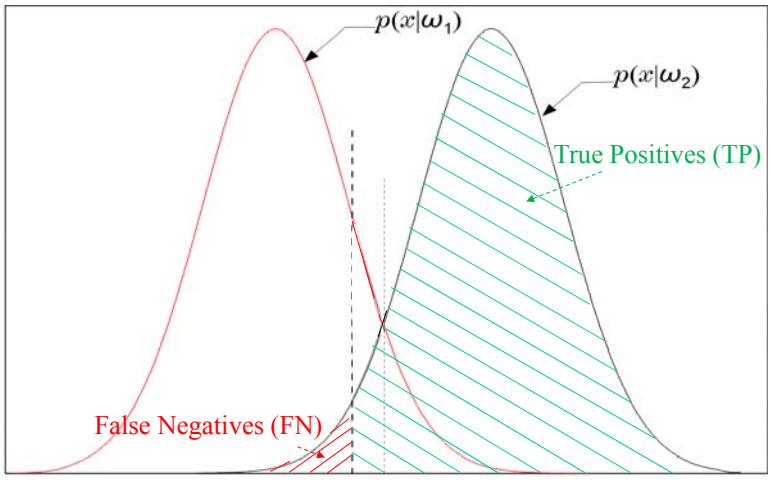


Receiving Operator Characteristics (ROC) Curves



The plot of P_D vs P_F for the various decision strategies is called the **receiver operating characteristics** (ROC) and is used to assess classifier performance

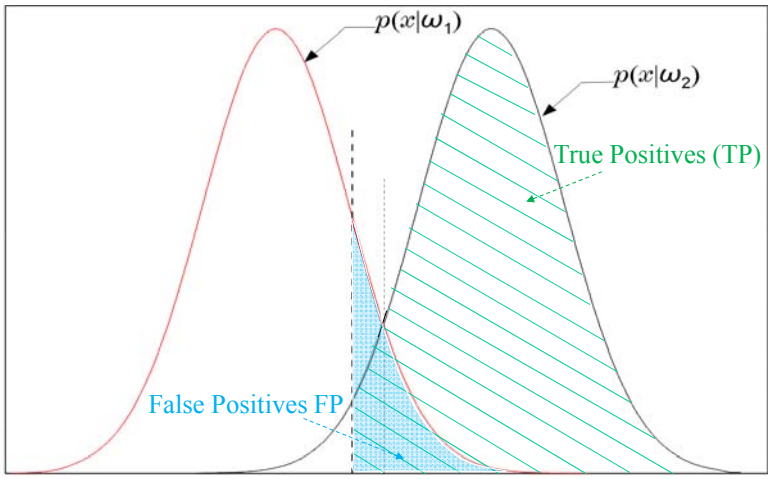
Precision and Recall



$$\textbf{Recall} = \frac{TP}{TP + FN}$$

Recall is the same as **probability of detection**, P_D, or the **true positive rate**

Precision and Recall



$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision has no counterpart in detection theory and is insensitive to the number of TN

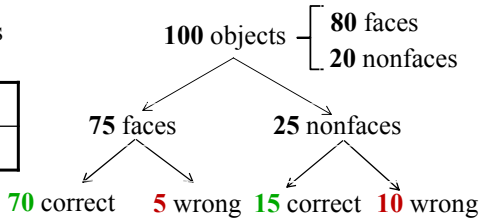
Precision and Recall

- **Precision** (*specificity*) and **recall** (*sensitivity*) come from the information retrieval literature and are often used in computer vision tasks to evaluate classification performance.
- **Recall** is the number of correct relevant items divided by the total number of relevant items in the repository. That is, of all the items that are true, how many did the classifier actually pick up. In the language of detection theory, recall is equivalent to the **probability of detection**, or P_D.
- **Precision** is the number of correct items divided by the number of total items retrieved. That is, of all the items that are marked positive, how many are truly positive. In the language of detection theory, precision has no counterpart.
- A major short coming in using only precision and recall for the evaluation of classification performance is that neither of them include TN, which implies that these measures are insensitive to the data distribution (e.g., one can increase the number of TN as much as one wishes and there will be no change in precision and recall numbers (note that the FP rate is sensitive to the data distribution).

Example

- Consider **100** pictures (**80** w. faces & **20** w. none) inputted to a face detector
- Detector detects **75** pictures w. faces, **70** of which are correct, **5** are wrong
- Detector detects **25** pictures as nonfaces, **15** of which are correct, **10** wrong

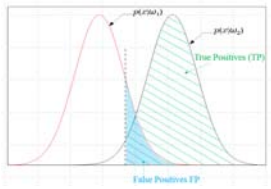
Classification	Actual Class ω_1	Actual Class ω_2
ω_1	70	5
ω_2	10	15



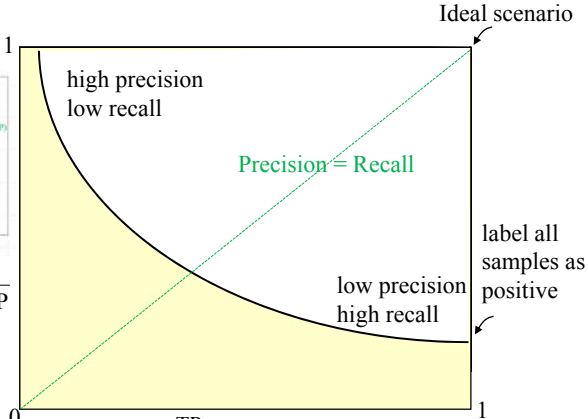
$$\textbf{TP Rate} = \frac{70}{70+10}; \quad \textbf{FP Rate} = \frac{5}{15+5}$$

$$\textbf{Precision} = \frac{70}{70+5}; \quad \textbf{Recall} = \frac{70}{70+10}$$

Trade-Off Between Precision and Recall



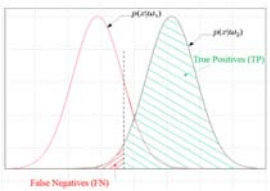
$$\text{Precision} = \frac{TP}{TP+FP}$$



$$\text{Recall} = \frac{TP}{TP+FN}$$

$$F_{score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

F_{score} summarizes the precision and recall measures. Larger values show better performance.



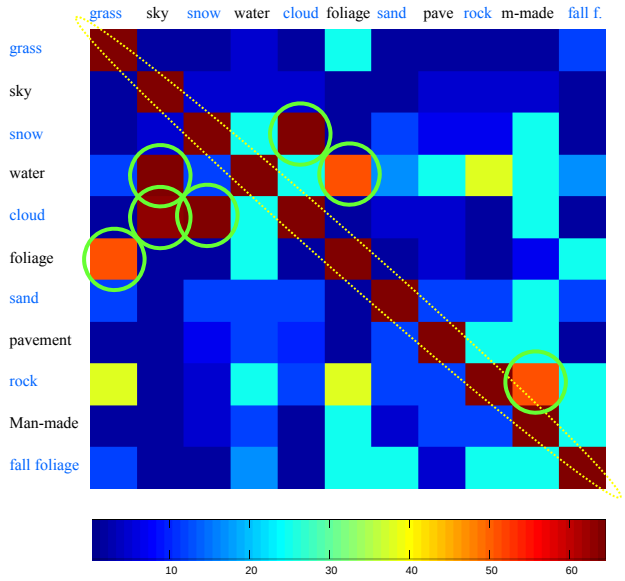
Evaluation of Multi-Class Classification

A multi-class classification performance is evaluated with a **confusion matrix**.

Prediction Outcome \ Actual Class	C_1	C_2	C_3	C_4
C_1	☺	X	X	X
C_2	X	☺	X	X
C_3	X	X	☺	X
C_4	X	X	X	☺

$$\text{Accuracy} = \frac{\text{Number of correctly classified points}}{\text{Total number of points}}$$

Example: Visualization of Confusion Matrix



Examples of confusion

- water and sky
- cloud and snow
- snow and cloud
- cloud and sky
- foliage and grass
- water and foliage
- rock and man-made

Jie Yu, Kodak Research Labs

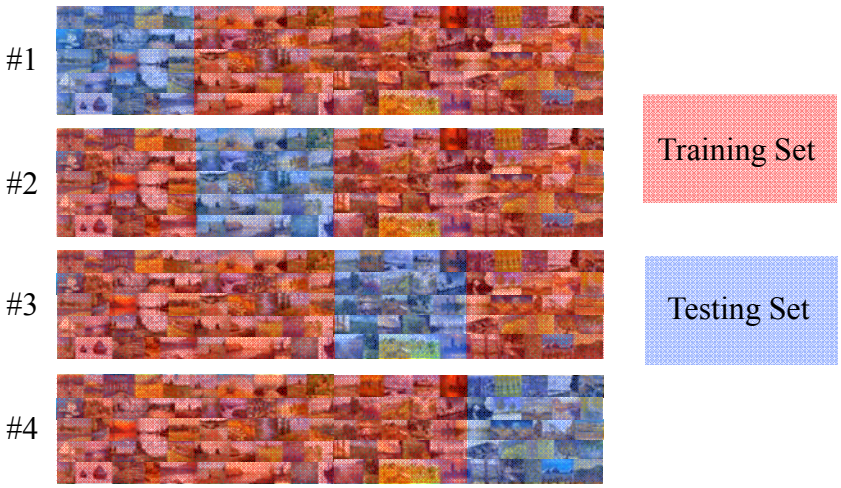
Validation

- **Validation** is the process of determining whether the classifier is performing a good job in modeling the data.
- If the performance of the learner is only measured on the training data, it usually does not provide an accurate indication of how well it will generalize to an unseen data set.
- **Holdout Method:** part of the training data is removed and used to get predictions of the performance of the learner trained on rest of the data. Although superior to traditional validation, this method still suffers from issues such as underfitting and high variance. By reducing the training data, we risk losing important patterns/trends in the data set, which in turn increases the error induced by bias. Furthermore, we don't know which data points will end up in the validation set and the result might be entirely different for different sets.
- A better approach is to use *K*-fold cross validation.

K-Fold Cross-Validation (CV)

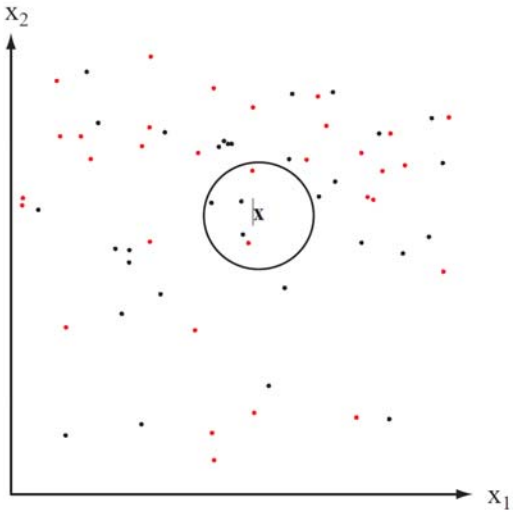
- ***K*-Fold Cross Validation** (CV) is used to test the generalizability (robustness) of a machine learning algorithm.
- The available dataset is randomly divided into K training and testing partitions (folds), e.g., $K=5$.
- The *Holdout* method is repeated K times. That is, one of the K folds is used for testing while the remaining $K-1$ folds are used for training. Every data point gets to be in a validation set exactly once, while it participates in a training set $K-1$ times. This significantly reduces bias as we are using most of the data for fitting, and also significantly reduces variance as most of the data is also being used in validation set.
- The cross validation error is measured as the average error on test samples over all experiments.

Example of 4-Fold Cross-Validation



k-Nearest Neighbor (*k*-NN) Classification

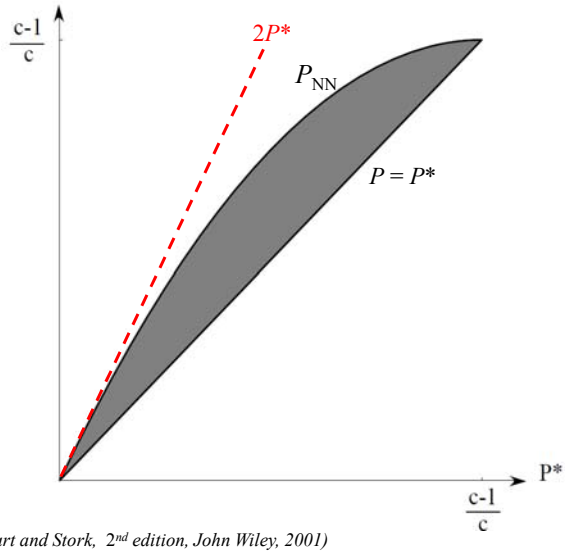
The *k*-nearest neighbor query starts at the test point ***x*** and grows a spherical region until it encloses *k* training samples, and it labels the test point by a majority vote of these samples. In this example, $k = 5$ and the test point would be labeled as the category of the black points.



(Duda, Hart and Stork, 2nd edition, John Wiley, 2001)

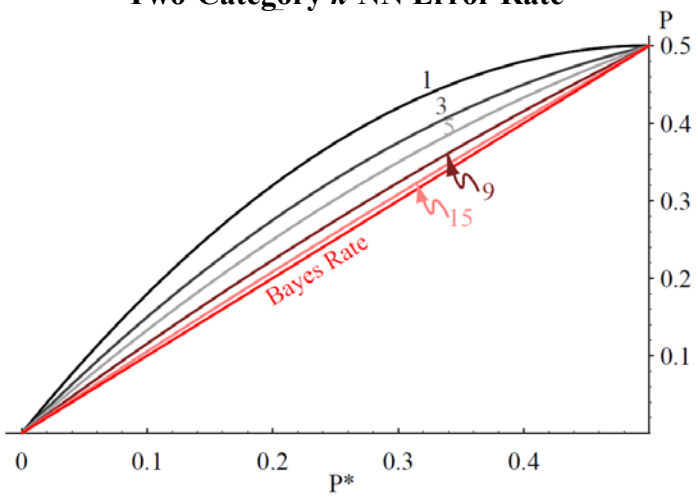
C-Category 1-Nearest Neighbor Classifier Error Rate

Bounds on the nearest neighbor error rate P_{NN} in a c-category problem given infinite training data, where P^* is the Bayes error. At low error rates, the NN error rate is bounded above by twice the Bayes rate.



(Duda, Hart and Stork, 2nd edition, John Wiley, 2001)

Two-Category *k*-NN Error Rate



The error rate for the k-nearest neighbor rule for a 2-category problem. As $k \to \infty$, the estimated probabilities match the true probabilities and the error rate becomes equal to the Bayes rate P^*

(Duda, Hart and Stork, 2nd edition, John Wiley, 2001)

NN-Rule Implementation

- **Editing** algorithms eliminate erroneously labelled prototypes from the original set and 'clean' the overlapping among regions from different classes. These techniques tend to offer improvements in performance. It used primarily to clean erroneously labelled samples from the training set and the main goal is to improve recognition accuracy by producing a sterilised set. The fact that a computational advantage may be gained is a secondary benefit.
- **Reducing** or **condensing** algorithms aim at selecting the minimal subset of prototypes that lead to (approximately) the same performance as the NN rule using the whole training set. They are used primarily for the purpose of reducing the number of samples to gain a computational advantage

Machine Intelligence & Deep Learning Workshop

Dimensionality Reduction Techniques

Majid Rabbani

Visiting Professor

The Kate Gleason COLLEGE OF ENGINEERING

Dimensionality Reduction Techniques

- Aim at reducing the number of dimensions while either preserving the fidelity to the original data or maintaining maximum discrimination.
  - Principal Component Analysis (PCA) – Seeks to reduce the dimension of the data while maintaining fidelity to the original high-dimensional data in the mean-square error sense.
  - Fisher's Linear Discriminant (FLD)
    - For the two-category case, FLD reduces the data to a single dimension while maintaining maximum discrimination between the two classes by maximizing a measure of between-class scatter while minimizing a measure of within-class scatter.
    - For multi-category case (c categories, where $c > 2$), the d-dimensional data is projected onto a $(c-1)$-dimensional subspace in such a way as to create the greatest separation of the projected distributions as defined by the scatter matrices.

Quick Review of Scatter Matrices

Table 10.1: Mean vectors and scatter matrices used in clustering criteria.

	Depend on cluster center?		
	Yes	No	
Mean vector for the ith cluster		×	$\mathbf{m}_i = \frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{D}_i} \mathbf{x}$ (54)
Total mean vector		×	$\mathbf{m} = \frac{1}{n} \sum_{\mathcal{D}} \mathbf{x} = \frac{1}{n} \sum_{i=1}^{c} n_i \mathbf{m}_i$ (55)
Scatter matrix for the ith cluster	×		$\mathbf{S}_i = \sum_{\mathbf{x} \in \mathcal{D}_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^t$ (56)
Within-cluster scatter matrix	×		$\mathbf{S}_W = \sum_{i=1}^{c} \mathbf{S}_i$ (57)
Between-cluster scatter matrix	×		$\mathbf{S}_B = \sum_{i=1}^{c} n_i (\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^t$ (58)
Total scatter matrix		×	$\mathbf{S}_T = \sum_{\mathbf{x} \in \mathcal{D}} (\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^t$ (59)

(Duda, Hart and Stork, 2nd edition, John Wiley, 2001)

Procedure for Calculating PCA

- Consider n, d-dimensional sample points labeled $\boldsymbol{x}_1$ through $\boldsymbol{x}_n$. Class labels do not affect the mathematical derivation of PCA.
- Calculate the mean vector $\mathbf{m}$ and the scatter matrix $\mathbf{S}$ of the entire sample set by summing over all the n samples:

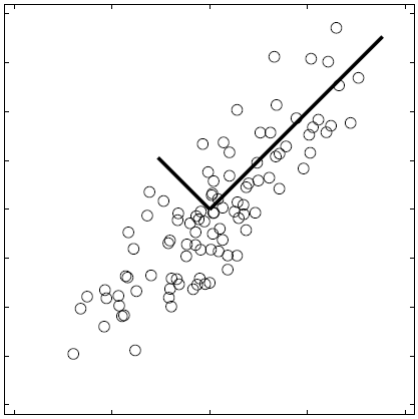
$$\mathbf{m} = 1/n\left[\sum \boldsymbol{x}_k\right]$$

$$\mathbf{S} = \sum (\boldsymbol{x}_k - \mathbf{m})\,(\boldsymbol{x}_k - \mathbf{m})^t$$

- Calculate the eigenvalues λ_i and the corresponding eigenvectors $\mathbf{e}_i$ of the scatter matrix $\mathbf{S}$. Rank the eigenvalues in the order of largest to smallest.
- For a sample $\boldsymbol{x}$, project it on $\mathbf{e}_i$ and call the projection value the scalar a_i.
- The best approximation to a sample vector $\boldsymbol{x}$ (in terms of smallest squared error) using only d' components (where $d' < d$) is found by adding the first d' terms of the expansion:

$$\boldsymbol{x} = \mathbf{m} + \sum a_k \mathbf{e}_k$$

Principal Component Analysis (PCA): Visualization



Data points are represented in a rotated orthogonal coordinate system: the origin is the mean of the data points and the axes are provided by the eigenvectors.

PCA Example – Handwritten Digits

- The MNIST dataset (http://yann.lecun.com/exdb/mnist/) contains over 50,000 handwritten single digit images where each image is of size 28×28 pixels and has a ground truth label.
- Considering each pixel value as a feature (dimension), each image consists of $d = 28 \times 28 = 784$ dimensions. We wish to explore storing these images in less than 784 dimensions by using PCA decomposition.
- 1,000 images were randomly selected as a representative set. Sixteen of these images are shown as an example in the following slides. The mean vector **m** (size 28×28) and the scatter matrix **S** (size 784×784) of the set were calculated.
- The eigenvalues (and corresponding eigenvectors) of the covariance matrix were calculated and ranked from highest to lowest. These eigenvectors constitute the new set of basis functions for representing the images. The mean image and 5 eigenvectors with highest eigenvalues are shown in the next slide.
- The subsequent slides show the example digits reconstructed using 1, 2, 5, 10, 20, 50, 100, and 200 eigenvectors corresponding to the largest eigenvalues.

PCA Example - 1,000 Handwritten Digits (28×28 pixels)

m = mean image (vector) of the 1,000 images 

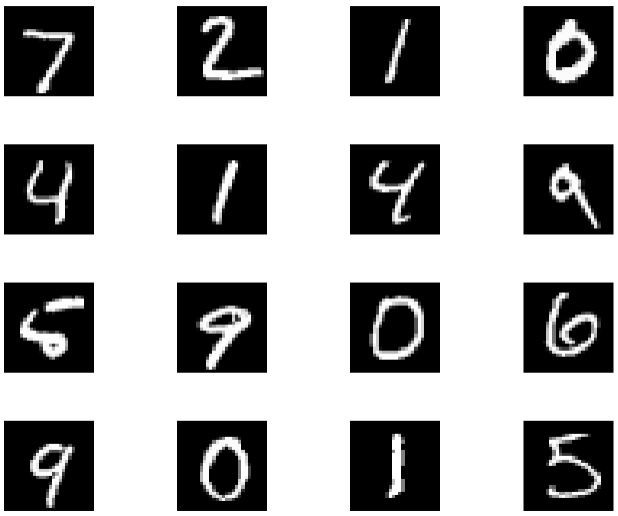
unscaled version of 5 PCA eigenvectors corresponding to 5 largest eigenvalues

λ_1=727.3 λ_2=136.2 λ_3=124.1 λ_4=110.2 λ_5=101.0

scaled version of above eigenvectors to fit the entire displayable range

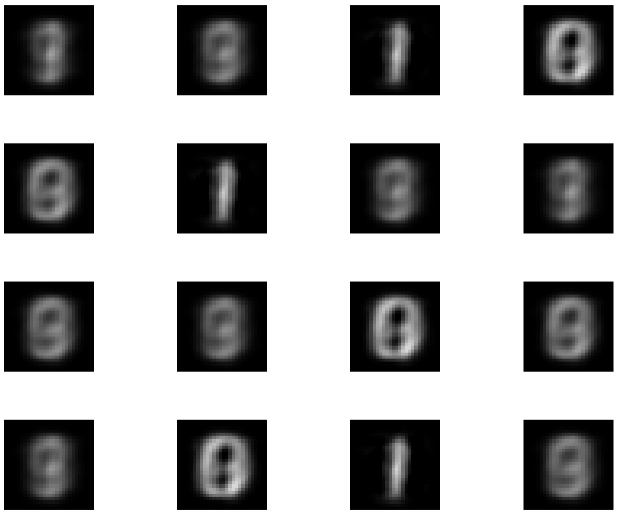
Courtesy of Prasanna Reddy Pulakurthi

PCA Example – Handwritten Digits (Original)



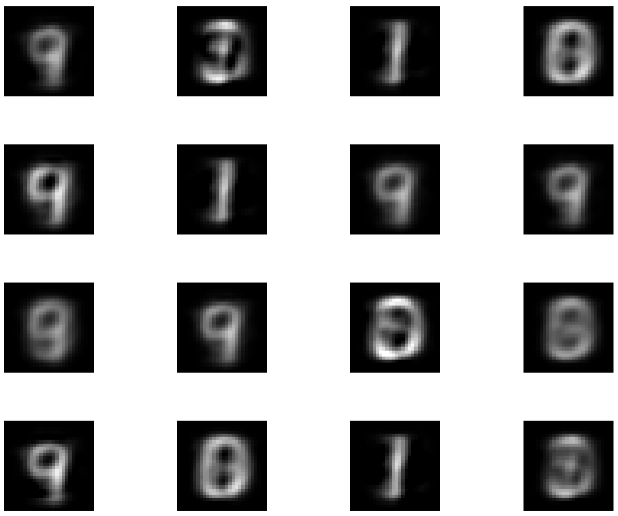
Courtesy of Prasanna Reddy Pulakurthi

Handwritten Digits Reconstructed with 1 PCA Component



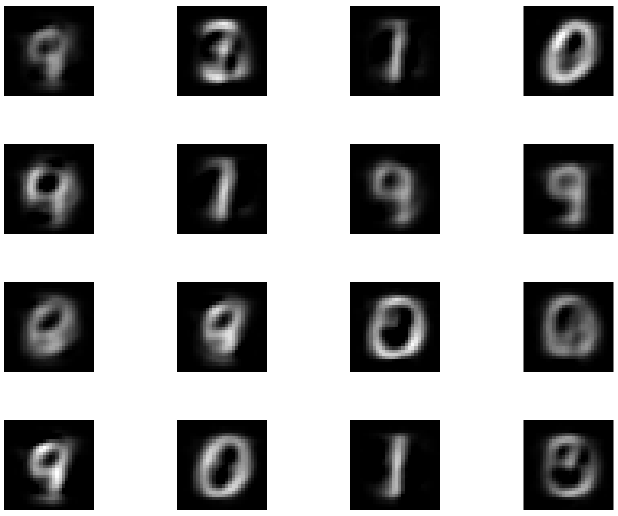
Courtesy of Prasanna Reddy Pulakurthi

Handwritten Digits Reconstructed with 2 PCA Components



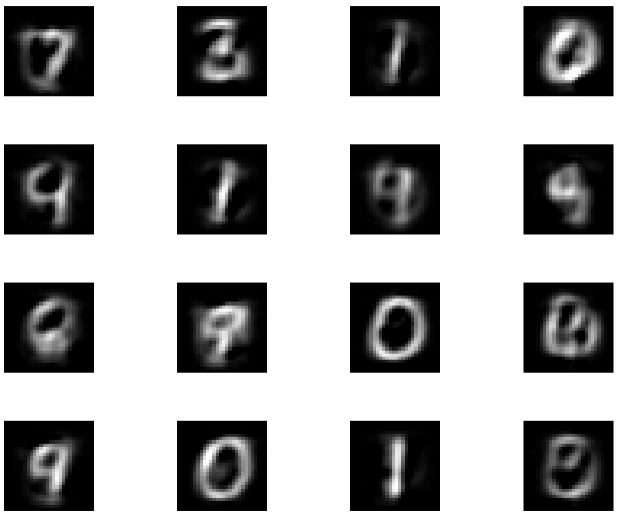
Courtesy of Prasanna Reddy Pulakurthi

Handwritten Digits Reconstructed with 5 PCA Components



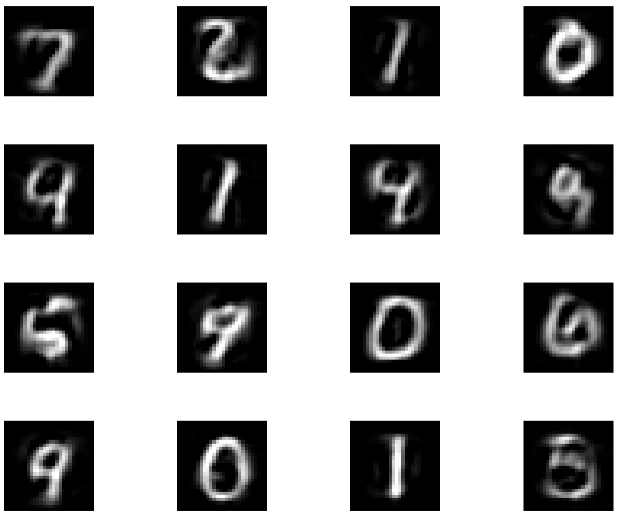
Courtesy of Prasanna Reddy Pulakurthi

Handwritten Digits Reconstructed with 10 PCA Components



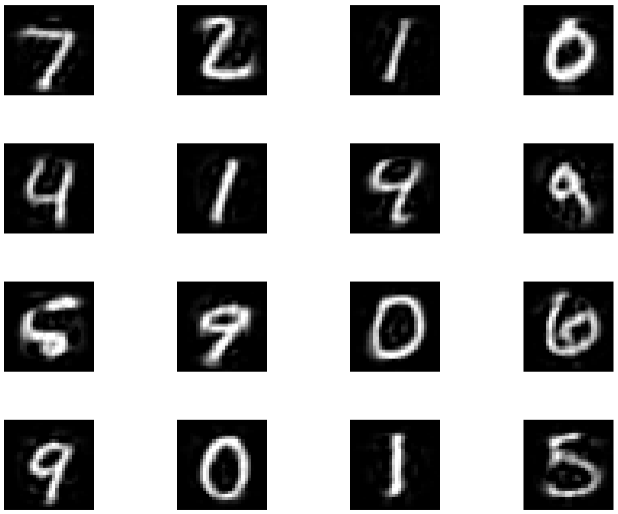
Courtesy of Prasanna Reddy Pulakurthi

Handwritten Digits Reconstructed with 20 PCA Components



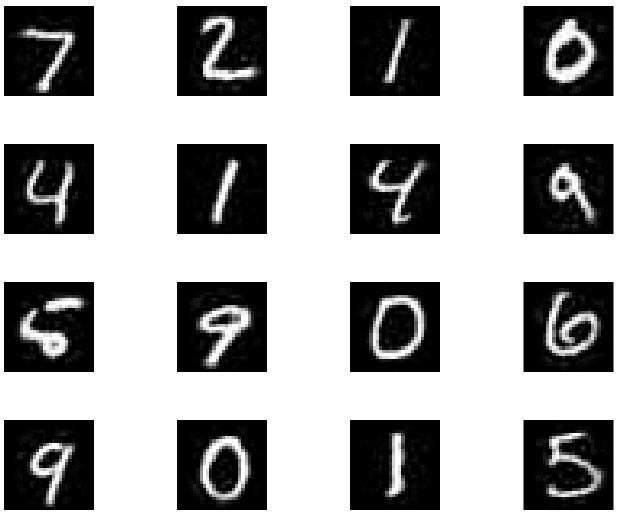
Courtesy of Prasanna Reddy Pulakurthi

Handwritten Digits Reconstructed with 50 PCA Components



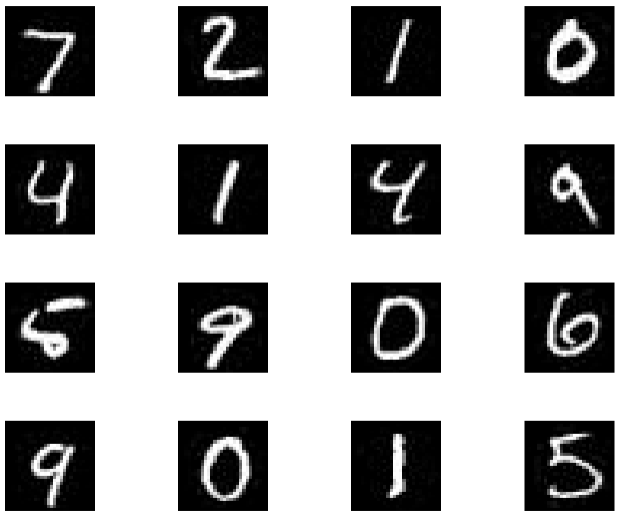
Courtesy of Prasanna Reddy Pulakurthi

Handwritten Digits Reconstructed with 100 PCA Components



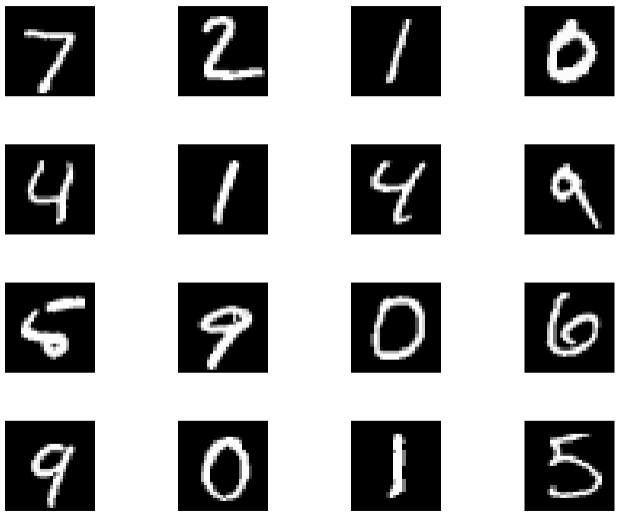
Courtesy of Prasanna Reddy Pulakurthi

Handwritten Digits Reconstructed with 200 PCA Components



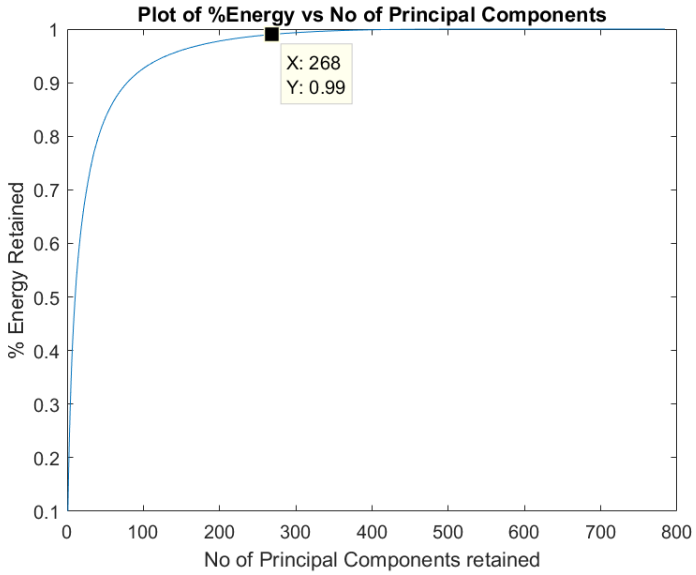
Courtesy of Prasanna Reddy Pulakurthi

Original – 784 Components



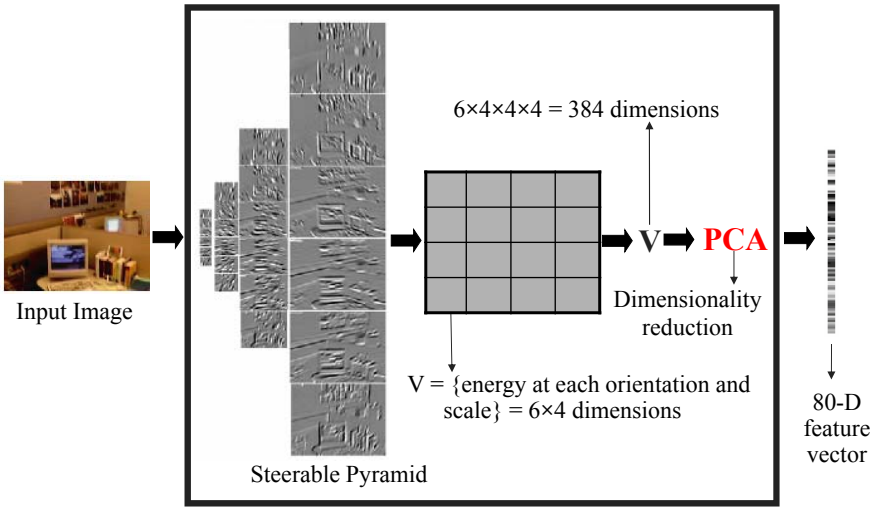
Courtesy of Prasanna Reddy Pulakurthi

PCA Energy Retention



Courtesy of Prasanna Reddy Pulakurthi

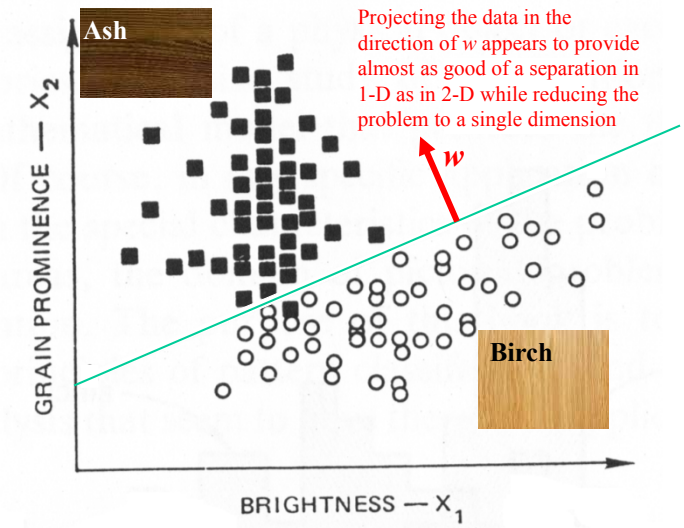
Example of PCA: GIST Feature Descriptor



GIST Features for Scene Classification or Matching

Courtesy of Aude Olivia and Antonio Torralba, MIT

Fisher's Linear Discriminant (FLD)



Portions of this figure are from "Pattern Classification" by Duda and Hart

Fisher's Linear Discriminant

- For the two-category case, FLD aims at finding the direction $\boldsymbol{w}$ (note that $\boldsymbol{w}$ is a vector) so that when the data in projected in that direction, the following objective function, known as the generalized Raleigh Quotient, is maximized:

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T \boldsymbol{S}_{\mathrm{B}} \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{S}_{\mathrm{w}} \boldsymbol{w}}$$

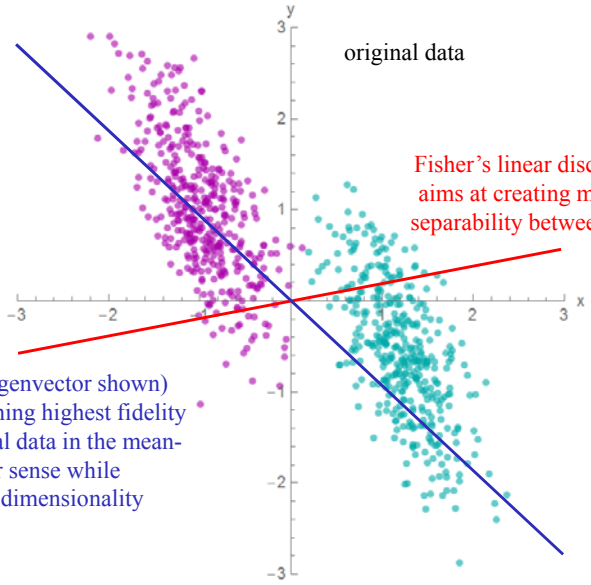
- The optimum direction $\boldsymbol{w}$ is given by:

$$\boldsymbol{w} = S_{\mathrm{W}}^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$$

- For multi-category case, the weight $\boldsymbol{w}$ becomes a matrix $\boldsymbol{W}$ and the objective function to be maximized is (where $|\boldsymbol{A}|$ represents the determinant of the matrix $\boldsymbol{A}$):

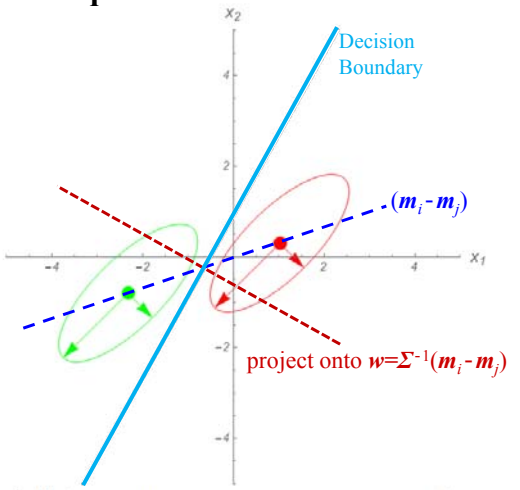
$$J(\boldsymbol{W}) = \frac{|\boldsymbol{W}^T \boldsymbol{S}_{\mathrm{B}} \boldsymbol{W}|}{|\boldsymbol{W}^T \boldsymbol{S}_{\mathrm{w}} \boldsymbol{W}|}$$

PCA vs Fisher's LDA



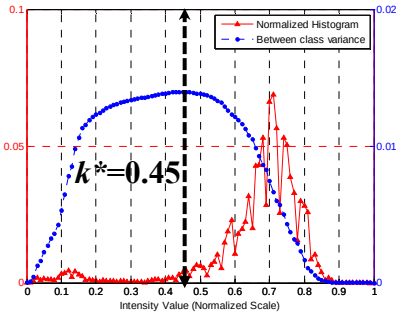
2-Category Gaussian with Equal Covariance Matrices

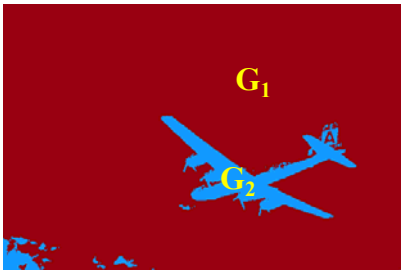
- It is well-known from theory that in the case of two-category Gaussian distributions with equal covariance matrices, $\boldsymbol{\Sigma}$, the decision boundary is a hyperplane perpendicular to the direction of the vector $\boldsymbol{\Sigma}^{-1}(\boldsymbol{m}_i - \boldsymbol{m}_j)$.
- It turns out that this direction is identical to the projection direction of the Fisher's linear discriminant. Note that after the projection, the decision threshold still needs to be determined.



	$P(\omega_i)$	μ_i	σ_i	ρ_i	Σ_i
ω_1	0.5	$\begin{pmatrix}1.02\\0.3\end{pmatrix}$	$\begin{pmatrix}1.53\\1.52\end{pmatrix}$	0.75	$\begin{pmatrix}2.3409 & 1.7442\\1.7442 & 2.3104\end{pmatrix}$
ω_2	0.5	$\begin{pmatrix}-2.32\\-0.8\end{pmatrix}$	$\begin{pmatrix}1.53\\1.52\end{pmatrix}$	0.75	$\begin{pmatrix}2.3409 & 1.7442\\1.7442 & 2.3104\end{pmatrix}$

1-D, 2-Class, K-Means Example - Otsu's Thresholding Segmentation Method





Machine Intelligence & Deep Learning Workshop

Boosting

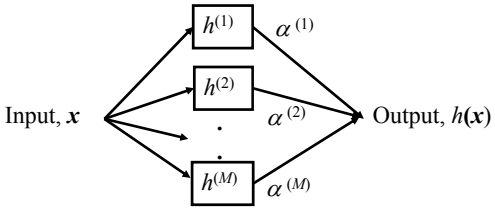
Majid Rabbani

Visiting Professor

The Kate Gleason COLLEGE OF ENGINEERING

Boosting

- A powerful classification method is to construct a team of weak classifiers and weigh their responses based on their confidence, known as **Boosting**. A weak classifier is a simple classifier such as one applying a simple threshold to only one of the dimensions (often chosen randomly) of the feature vector.
- Similar to SVM, Boosting is generally suited to 2-class problems and the extension to multi-class is often handled by either one-against-all or pairwise-comparison strategies. However, a recent publication has addressed a natural extension to the multi-class case†.



Team of M weak classifiers

† *Trevor Hastie, et. al, "Multi-class AdaBoost," Statistics and Its Interface, Vol 2, No 3, Pages: 349 – 360*

Adaptive Boosting (AdaBoost) Summary

$$w_i^1 = \frac{1}{N}, \qquad p_i^1 = \frac{w_i^1}{\sum_{i=1}^{N} w_i^1}, \qquad i = 1, 2, \ldots, N$$

Initialize a weight distribution over data; usually uniform weight

$$h^{*(n)};\ \min\left(\varepsilon^{(n)} = \sum_{i=1}^{N} p_i^{(n)} \left|h^{(n)}(\boldsymbol{x}_i) - c(\boldsymbol{x}_i)\right|\right)$$

Select a weak classifier that minimizes the error with the current distribution

$$w_i^{(n+1)} = \begin{cases} w_i^{(n)}, & \text{if incorrect} \\ w_i^{(n)}\beta^{(n)}, & \text{if correct} \end{cases} \quad \text{where,} \quad \beta^{(n)} = \frac{\varepsilon^{(n)}}{1-\varepsilon^{(n)}}$$

$$p_i^{(n+1)} = \frac{w_i^{(n+1)}}{\sum_{i=1}^{N} w_i^{(n+1)}}, \qquad i = 1, 2, \ldots, N$$

Estimate new data weights by emphasizing misclassified points

Repeat M times

$$\alpha^{(n)} = \ln\left[\frac{1-\varepsilon^{(n)}}{\varepsilon^{(n)}}\right] \qquad h(\boldsymbol{x}) = \begin{cases} 1, & \sum_{k=1}^{M} \alpha^{(k)} h^{(k)}(\boldsymbol{x}) \geq \frac{1}{2}\sum_{k=1}^{M} \alpha^{(k)} \\ 0, & \text{otherwise} \end{cases}$$

Weighted sum of all weak classifier outputs is the final response

Adaptive Boosting – AdaBoost (Step 1)

- In the following, the various steps involved in the application of **AdaBoost** algorithm are outlined.
- STEP 1: Consider N data points $\boldsymbol{x}_i$, each of which has a d-dimensional feature. We start by initializing a weight distribution over all data points, denoted by w_i^1 (where the superscript 1 denotes the first iteration or the design of the first weak classifier), which is usually just assigning equal weight to all data points. We also define a probability distribution, p_i^1, over the data points by normalizing the weights so that they add up to one.

$$w_i^1 = \frac{1}{N}, \qquad p_i^1 = \frac{w_i^1}{\sum_{i=1}^{N} w_i^1}, \qquad i = 1, 2, \ldots, N$$

Adaptive Boosting – AdaBoost (Step 2)

- At each iteration, a new weak classifier is developed. A weak classifier is a simple classifier such as one applying a simple threshold to only one of the dimensions (often chosen randomly) of the feature vector.
- STEP 2: At stage n, we try all the possible weak classifiers, $h^{(n)}$, and choose the one that minimizes the error with the current probability distribution $p_i^{(n)}$, and denote it by $h^{*(n)}$.
- In the following, $h^{(n)}(\boldsymbol{x}_i)$ denotes the classification label (zero or one) of the data point $\boldsymbol{x}_i$ as classified by the classifier $h^{(n)}$ and $c(\boldsymbol{x}_i)$ denotes the true class label of $\boldsymbol{x}_i$ (zero or one).
- The $\varepsilon^{(n)}$ is the total classification error resulting from using the weak classifier $h^{(n)}$. Clearly, for the minimum error classifier, $\varepsilon^{(n)} < 0.5$.

$$h^{*(n)};\ \min\left(\varepsilon^{(n)} = \sum_{i=1}^{N} p_i^{(n)} \left|h^{(n)}(\boldsymbol{x}_i) - c(\boldsymbol{x}_i)\right|\right)$$

Adaptive Boosting – AdaBoost (Step 3)

- STEP 3: The weights $w_i^{(n)}$ are updated to emphasize the misclassified points (essentially, the weights of the misclassified points are kept the same while the weights of the correctly classified points are reduced) and the new probability distribution $p_i^{(n+1)}$ is calculated by normalizing the updated weights.
- Note that in the equation below, $\beta^{(n)} < 1$, because $\varepsilon^{(n)} < 0.5$.

$$w_i^{(n+1)} = \begin{cases} w_i^{(n)}, & \text{if incorrect} \\ w_i^{(n)}\beta^{(n)}, & \text{if correct} \end{cases} \quad \text{where,} \quad \beta^{(n)} = \frac{\varepsilon^{(n)}}{1-\varepsilon^{(n)}}$$

$$p_i^{(n+1)} = \frac{w_i^{(n+1)}}{\sum_{i=1}^{N} w_i^{(n+1)}}, \qquad i = 1, 2, \ldots, N$$

Adaptive Boosting - AdaBoost

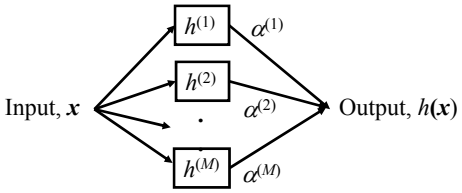
- If the classification error has dropped below a target value or the maximum number of weak classifiers has been reached, the algorithm is terminated, otherwise, go back to STEP 2 and calculate the next optimum weak classifier based on the new distribution $p_i^{(n+1)}$.
- Note that the classification error $\varepsilon^{(n+1)}$ of this classifier would be affected by all the previous classifiers. The new classifier would minimize the weighted classification error at stage $(n + 1)$.
- The classification error $\varepsilon^{(n)}$ at stage n also determines the weight $\alpha^{(n)}$ applied for the final voting to the output of the n^{th} stage classifier, $h^{(n)}$. Once determined, the weight $\alpha^{(n)}$ remains fixed and the output of $h^{(n)}$ will always be weighted with $\alpha^{(n)}$.

$$\alpha^{(n)} = \ln\left[\frac{1-\varepsilon^{(n)}}{\varepsilon^{(n)}}\right]$$

Adaptive Boosting – AdaBoost (Step 4)

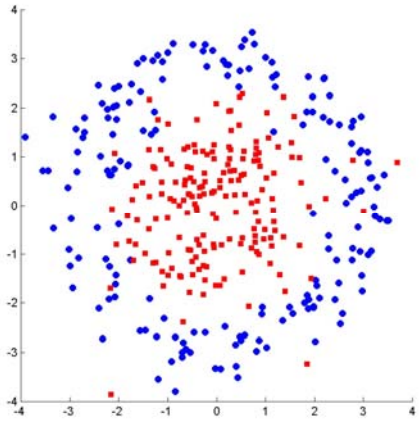
- STEP 4: The iterations are stopped if one is either satisfied with the final classification error or a maximum number of classifiers is reached.
- For a given input sample $\boldsymbol{x}$, its class label $h(\boldsymbol{x})$ (zero or one) is determined by the weighted sum (weighted by $\alpha^{(k)}$) of all the M weak classifier outputs $h^{(k)}(\boldsymbol{x})$.

$$h(\boldsymbol{x}) = \begin{cases} 1, & \sum_{k=1}^{M} \alpha^{(k)} h^{(k)}(\boldsymbol{x}) \geq \frac{1}{2} \sum_{k=1}^{M} \alpha^{(k)} \\ 0, & \text{otherwise} \end{cases}$$



Team of weak classifiers

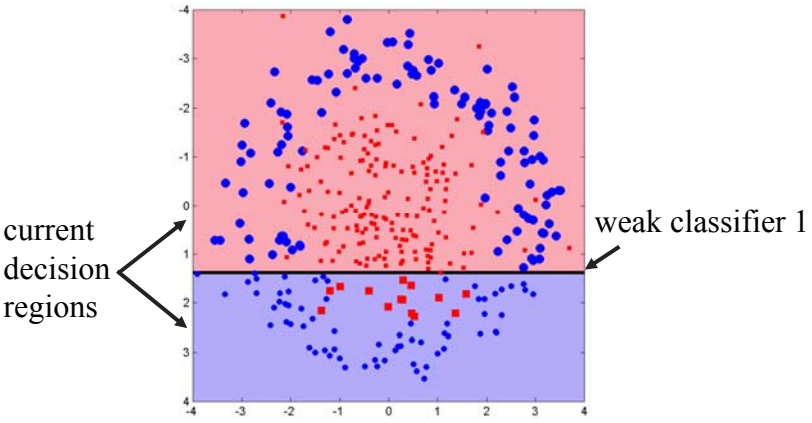
AdaBoost Example



Two classes of data points

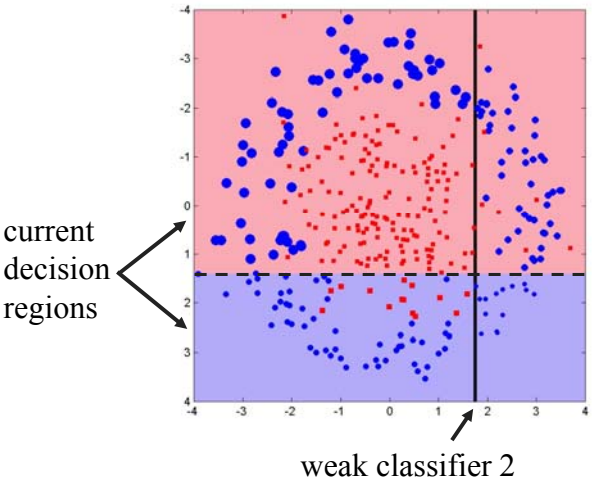
Courtesy of Andrew Gallagher, Eastman Kodak Company

AdaBoost Example

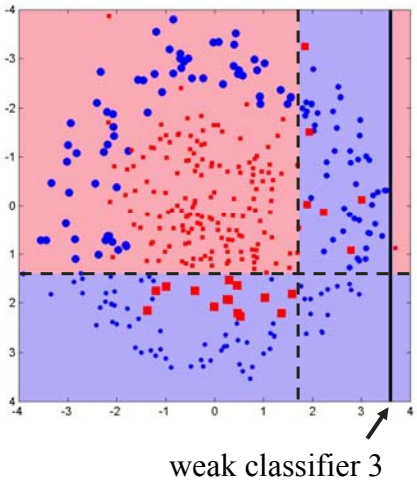


Marker size indicates the weight

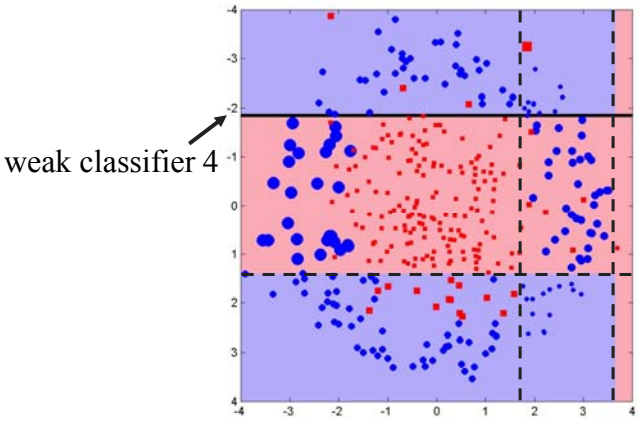
AdaBoost Example



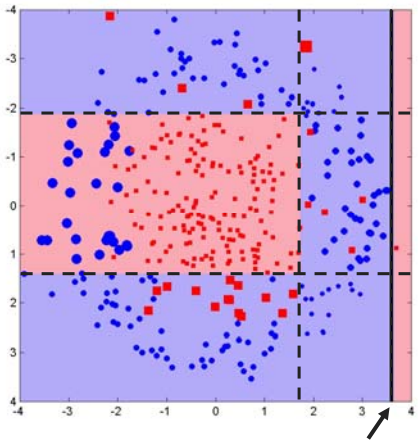
AdaBoost Example



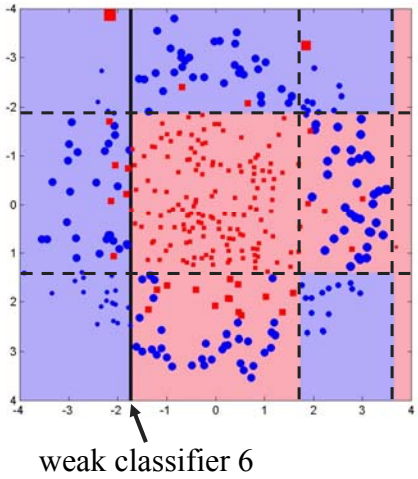
AdaBoost Example



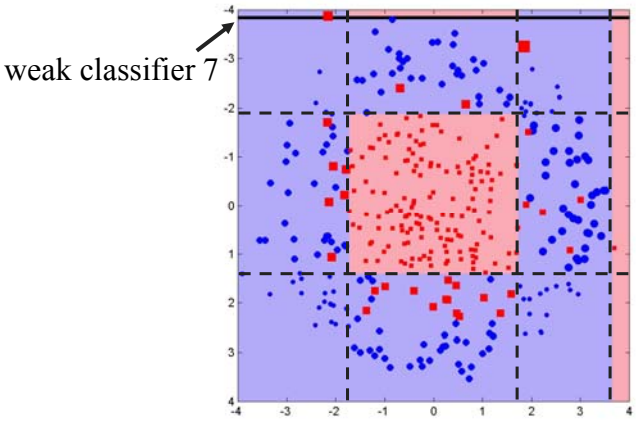
AdaBoost Example



AdaBoost Example



AdaBoost Example



AdaBoost Remarks

- One of the main and desirable properties of boosting is its relative immunity to overfitting. In practice, it has been verified that, although the number of terms M, and consequently the associated number of parameters can be quite high, the error rate on a test set does not increase but keeps decreasing and finally *levels off at a certain value*. It has been observed that the test error continues to decrease long after the error on the training set has become zero.
- Furthermore, boosting is particularly aggressive at improving the margin distribution, since it concentrates on examples with the smallest margins. From this point of view, there is an affinity with the support vectors machines, which also try to maximize the margin of the training samples from the decision surface.
- A comparative study of the performance of boosting and other related algorithms can be found in: *Bauer E., Kohavi R. "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," Machine Learning, Vol. 36, pp. 105–139, 1999.*

Example 4.3 From T&K Text (Page 236-237)

Example 4.3

Let us consider a two-class classification task. The data reside in the 20-dimensional space and obey a Gaussian distribution of unit covariance matrix and mean values $[-a,-a,\ldots,-a]^T$, $[a,a,\ldots,a]^T$, respectively, for each class, where $a = 2/\sqrt{20}$. The training set consists of 200 points (100 from each class) and the test set of 400 points (200 from each class) independently generated from the points of the training set.

To design a classifier using the AdaBoost algorithm, we chose as a seed the weak classifier known as *stump*. This is a very "naive" type of tree, consisting of a single node, and classification of a feature vector $\boldsymbol{x}$ is achieved on the basis of the value of only one of its features, say, x_i. Thus, if $x_i < 0$, $\boldsymbol{x}$ is assigned to class A. If $x_i > 0$, it is assigned to class B. The decision about the choice of the specific feature, x_i, to be used in the classifier was randomly made. Such a classifier results in a training error rate slightly better than 0.5.

The AdaBoost algorithm was run on the training data for 2000 iteration steps. Figure 4.30 verifies the fact that the training error rate converges to zero very fast. The test error rate keeps decreasing even after the training error rate becomes zero and then levels off at around 0.05.

Figure 4.31 shows the margin distributions, over the training data points, for four different training iteration steps. It is readily observed that the algorithm is indeed greedy in increasing the margin. Even when only 40 iteration steps are used for the AdaBoost training, the resulting classifier classifies the majority of the training samples with large margins. Using 200 iteration steps, all points are correctly classified (positive margin values), and the majority of them with large margin values. From then on, more iteration steps further improve the margin distribution by pushing it to higher values.

Example 4.3 From T&K Text (Page 236-237)

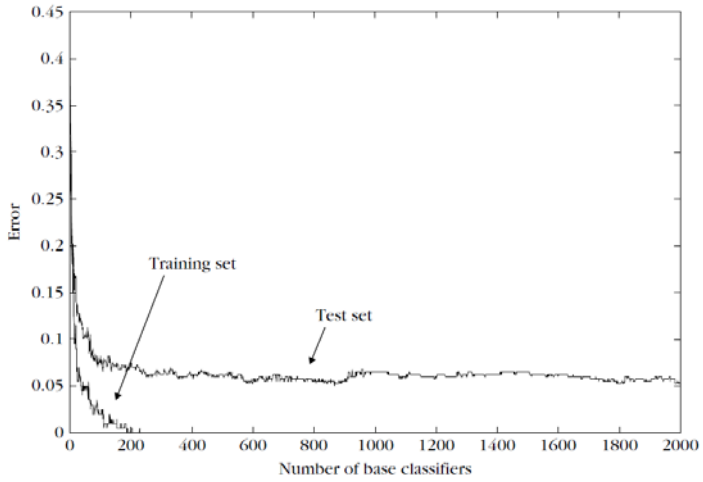


FIGURE 4.30

Training and test error rate curves as functions of the number of iteration steps for the AdaBoost algorithm, using a stump as the weak base classifier. The test error keeps decreasing even after the training error becomes zero.

AdaBoost Remarks

- A main advantage of boosting is its relative immunity to overfitting. In practice, it has been verified that, although the number of terms M, and consequently the associated number of parameters can be quite high, the error rate on a test set does not increase but keeps decreasing and finally *levels off at a certain value*. It has been observed that the test error continues to decrease long after the error on the training set has become zero.
- Furthermore, boosting is particularly aggressive at improving the margin distribution, since it concentrates on examples with the smallest margins. From this point of view, there is an affinity with the support vectors machines, which also try to maximize the margin of the training samples from the decision surface.
- A comparative study of the performance of boosting and other related algorithms can be found in: *Bauer E., Kohavi R. "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," Machine Learning, Vol. 36, pp. 105–139, 1999.*

AdaBoost Example†

Example 4.3

Let us consider a two-class classification task. The data reside in the 20-dimensional space and obey a Gaussian distribution of unit covariance matrix and mean values $[-a,-a,\ldots,-a]^T$, $[a,a,\ldots,a]^T$, respectively, for each class, where $a = 2/\sqrt{20}$. The training set consists of 200 points (100 from each class) and the test set of 400 points (200 from each class) independently generated from the points of the training set.

To design a classifier using the AdaBoost algorithm, we chose as a seed the weak classifier known as *stump*. This is a very "naive" type of tree, consisting of a single node, and classification of a feature vector $\boldsymbol{x}$ is achieved on the basis of the value of only one of its features, say, x_i. Thus, if $x_i < 0$, $\boldsymbol{x}$ is assigned to class A. If $x_i > 0$, it is assigned to class B. The decision about the choice of the specific feature, x_i, to be used in the classifier was randomly made. Such a classifier results in a training error rate slightly better than 0.5.

The AdaBoost algorithm was run on the training data for 2000 iteration steps. Figure 4.30 verifies the fact that the training error rate converges to zero very fast. The test error rate keeps decreasing even after the training error rate becomes zero and then levels off at around 0.05.

Figure 4.31 shows the margin distributions, over the training data points, for four different training iteration steps. It is readily observed that the algorithm is indeed greedy in increasing the margin. Even when only 40 iteration steps are used for the AdaBoost training, the resulting classifier classifies the majority of the training samples with large margins. Using 200 iteration steps, all points are correctly classified (positive margin values), and the majority of them with large margin values. From then on, more iteration steps further improve the margin distribution by pushing it to higher values.

† *Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009, pages 236-237*

AdaBoost Example†

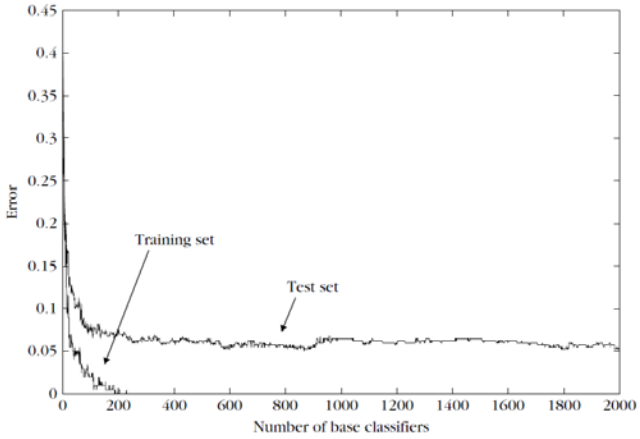


FIGURE 4.30

Training and test error rate curves as functions of the number of iteration steps for the AdaBoost algorithm, using a stump as the weak base classifier. The test error keeps decreasing even after the training error becomes zero.

† *Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009, pages 236-237*

Machine Intelligence & Deep Learning Workshop

Linear Discriminant Functions (LDF)

Majid Rabbani

Visiting Professor

The Kate Gleason COLLEGE OF ENGINEERING

Linear Classification Example

- Consider the problem of approving credit for a bank customer†. This problem can be formulated as a binary classification task where the input is a d-dimensional vector $\boldsymbol{x}$, represented as $(x_1, x_2, \ldots, x_d)^T$, components of which represent a relevant feature, such as age, gender, marital status, years in residence, years employed, annual income, current debt, etc., while the output y is a binary "yes (+)" or "no (-)."
- We will assign weights $\boldsymbol{w} = (w_1, w_2, \ldots, w_d)^T$ to each input feature. Note that some features might be more important than others and hence carry a larger weight (relative magnitudes of w_i's). Also, some features might contribute negatively (sign of w_i's).
- The approval task can be formulated as:

$$\sum_{i=1}^{d} x_i w_i > \text{Threshold} \rightarrow \textbf{Approve} \qquad \sum_{i=1}^{d} x_i w_i < \text{Threshold} \rightarrow \textbf{Reject}$$

†*Caltech lecture series: Yaser S. Abu-Mostafa, Learning from Data - Lecture 1*

Linear Classification Example

- The classification task can be represented by the following single equation, where an output sign of "+" corresponds to approval and "-" corresponds to rejection.

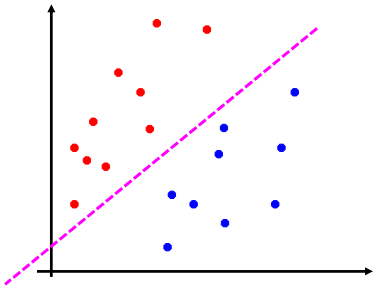
$$\text{Approve}(\boldsymbol{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} x_i w_i\right) - \text{Threshold}\right)$$

- If we introduce an artificial coordinate, $x_0 = 1$, the equation can be simplified by absorbing the threshold into the weight vector as an extra component w_0:

$$\text{Approve}(\boldsymbol{x}) = \text{sign}\left(\sum_{i=0}^{d} x_i w_i\right)$$

Perceptron Learning

- The learning task consists of determining the weight vector $\boldsymbol{w}$ (including the value of the threshold parameter w_0) by training the system with the examples available in the bank's historical database.
- If the training data is linearly separable (which we do not know in advance, and, in real life, almost certainly it will not be), a very simple algorithm called Perceptron can be used to find a linear classifier (although not necessarily the best) that separates the training data.



Perceptron Learning Algorithm

1. Consider a set of n input samples $\boldsymbol{x}_i$ with their corresponding output labels y_i. Start with an arbitrary initial guess $\boldsymbol{w}^{(0)}$.
2. At the k^{th} iteration, select a misclassified point $(\boldsymbol{x}_j, y_j)$ by either cycling through the whole data set, or by random selection, and correct (update) the latest hyperplane $\boldsymbol{w}^{(k-1)}$ according to:

$$\boldsymbol{w}^{(k)} = \boldsymbol{w}^{(k-1)} + \eta\, \boldsymbol{x}_j\, y_j$$

3. Continue STEP 2 until either there are no more misclassified samples (i.e., a separating hyperplane has been found) or a maximum number of iterations has been reached.

- There are many variations to the Perceptron learning algorithm: (i) set the relaxation parameter η to a constant (e.g., 1), (ii) make η a function of the iteration number k, (iii) execute the update in batch mode, (iv) execute the update in single sample stochastic or deterministic mode, ...

Shortcomings of The Perceptron Learning Algorithm

- Although the Perceptron algorithm is desirable due to its simplicity, it has many shortcomings that make it impractical.
  - Even when the data is linearly separable, it can take a very large number of iterations before a separating hyperplane is found. This is particularly true if two or more data points are very close to the separating hyperplane.
  - Even when the data is linearly separable, the solution might not be the best possible and the classifier might generalize poorly.
  - If the data is not linearly separable, the algorithm will not converge. Furthermore, the classification error might increase with the number of iterations. In such a case the classification error is tracked with each iteration and when a maximum number of iteration is reached a past hyperplane that resulted in the smallest error is selected (known as the Pocket algorithm).

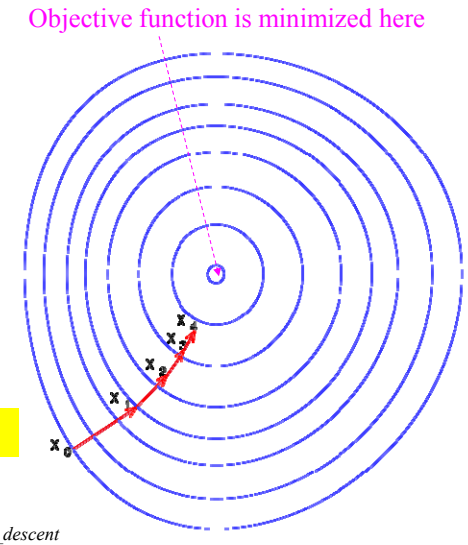
Optimizing Objective Functions by Gradient Descent

- The Perceptron algorithm can be made more robust by defining an objective function related to the misclassified samples and then finding the hyperplane that minimizes (maximizes) that objective function.
- A popular algorithm for optimizing objective functions is gradient descent, where the solution is iteratively updated by moving either in the direction of the gradient (for maximization) or in the opposite direction of the gradient (for minimization).
- For the gradient descent to work, the objective function has to be differentiable. For example, an objective function that is simply the total number of misclassified samples would not work and more sophisticated objective functions must be constructed.
- The same principle applies to the backpropagation algorithm in neural networks. For example, when choosing a logistic function, a step function does not work but sigmoid functions work well.

Gradient Ascent (Descent) Optimization

At each iteration, the value of $\boldsymbol{x}$ is updated by moving in the direction of the maximum variation (gradient) of the objective function. The relaxation parameter η determines the step size and can either be a constant or depend on the iteration number. For minimization:

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} - \eta\,(k)\,\nabla J(\mathbf{x}^{(k)})$$



https://en.wikipedia.org/wiki/Gradient_descent

Machine Intelligence & Deep Learning Workshop

Support Vector Machines (SVM)

Majid Rabbani

Visiting Professor

The Kate Gleason COLLEGE OF ENGINEERING

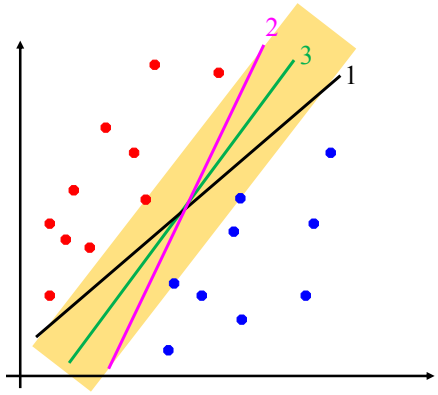
Support Vector Machines (SVM)

- A classifier derived from statistical learning theory by Vapnik, *et al.* in 1992.
- SVM became popular when, using images as input, it gave accuracy comparable to neural-network with hand-designed features in a handwriting and digit recognition tasks.
- SVM's can be formulated for both linearly separable and non-separable data. When combined with nonlinear kernels, SVM's become one of the most effective machine learning tools available today.
- Currently, SVM's are prevalent in all applications of machine learning such as object detection & recognition, content-based image retrieval, text recognition, biometrics, speech recognition, etc.
- SVM's can also be used for regression.

V. Vapnik

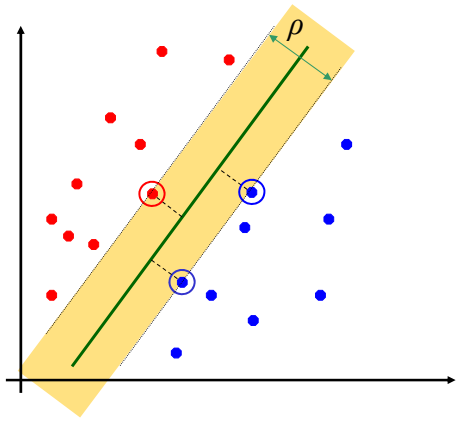
Support Vector Machines (SVM)

- Many linear classifiers exist that can separate the two classes but which one is the best?
- Intuitively, we feel that the third classifier should generalize better than the first two. Can this preference be quantified?

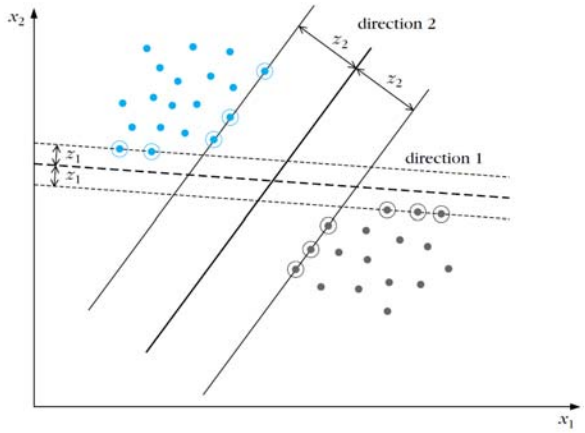


Support Vector Machines (SVM)

- Margin is defined as the width that the boundary could be increased by before hitting a data point.
- It is intuitive to pick the linear discriminant function (linear classifier) with the maximum margin, but why?
- One major reason is that it is robust to outliers and as a result has a stronger generalization ability.



Maximizing The Margin

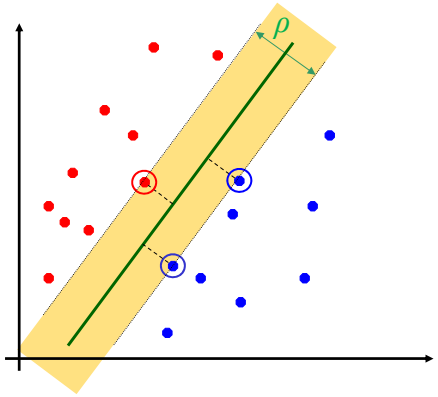


An example of a linearly separable two-class problem with two possible linear classifiers with different margins. From Theodoridis and Koutroumbas, "Pattern Recognition," page 121.

Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009, page 121

Support Vector Machines (SVM)

- Samples closest to the hyperplane are support vectors.
- Margin ρ of the separator is the distance between the support vectors. SVM seeks to find the maximum margin linear classifier.

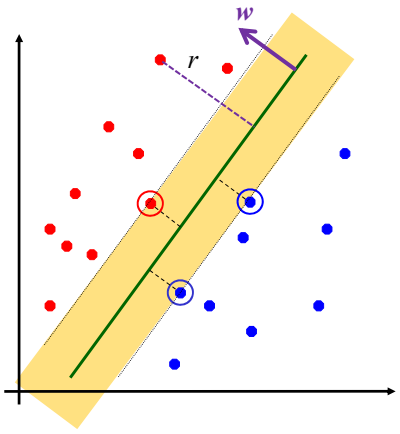


SVM – Linearly Separable Case

- Recall that the distance r of a sample point $\boldsymbol{x}_i$ from the hyperplane $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0$, is given by:

$$r = \frac{|g(\boldsymbol{x}_i)|}{\|\boldsymbol{w}\|} = \frac{|\boldsymbol{w}^T\boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|}$$

- $\boldsymbol{w}$ is the direction of the vector normal to the hyperplane and scaling it does not change its direction, so, in order to perform a meaningful optimization, it has to be normalized in some fashion.



Maximum Margin Linear Classifier Formulation

- Given a set of n data points $\boldsymbol{x}_i$ and their corresponding labels y_i: $i = 1, 2, \ldots n$, where:

  For $y_i = +1$, $\mathbf{w}^T\mathbf{x}_i + b \geq 0$

  For $y_i = -1$, $\mathbf{w}^T\mathbf{x}_i + b < 0$

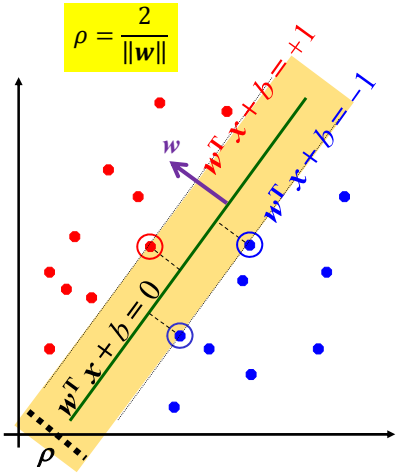
- The vector $\boldsymbol{w}$ is normalized such that the value of $|g(\boldsymbol{x})|$ at support vector samples is 1.

  For $y_i = +1$, $\mathbf{w}^T\mathbf{x}_i + \mathrm{b} \geq 1$

  For $y_i = -1$, $\mathbf{w}^T\mathbf{x}_i + \mathrm{b} \leq -1$

- Combining both equations:

  $y_i(\mathbf{w}^T\mathbf{x}_i + \mathrm{b}) \geq 1$



SVM Optimization Formulation

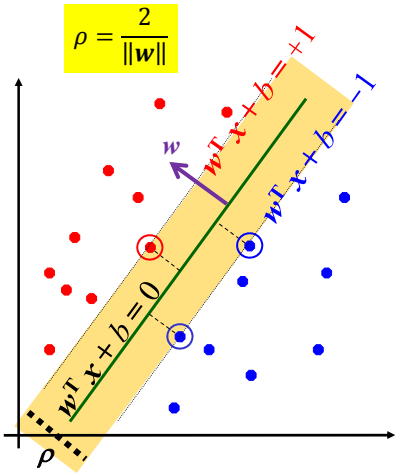
maximize $\dfrac{2}{\|\mathbf{w}\|}$, such that:

For $y_i = +1$, $\mathbf{w}^T\mathbf{x}_i + \mathrm{b} \geq 1$

For $y_i = -1$, $\mathbf{w}^T\mathbf{x}_i + \mathrm{b} \leq -1$

minimize $\dfrac{1}{2}\|\mathbf{w}\|^2$, such that:

$y_i(\mathbf{w}^T\mathbf{x}_i + \mathrm{b}) \geq 1 \quad i = 1, 2, \ldots n$



Solving The Optimization Problem

Quadratic programming with linear constraints

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + \mathrm{b}) \geq 1 \quad \text{for } i = 1, 2, \ldots n$$

Lagrangian Function

$$\text{minimize} \quad L_p(\mathbf{w}, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + \mathrm{b}) - 1 \right)$$

$$\text{s.t.} \quad \alpha_i \geq 0 \quad \text{for } i = 1, 2, \ldots n$$

α_i for i=1, 2, …n are Lagrangian multipliers

Optimizing By Solving The Primal Problem

$$\text{minimize} \quad L_p(\mathbf{w}, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + \mathrm{b}) - 1 \right)$$

$$\text{s.t.} \quad \alpha_i \geq 0 \quad \text{for } i = 1, 2, \ldots n$$

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L_p}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Known as the Karush-Kuhn-Tucker (KKT) conditions, which play a central role in both the theory and practice of constrained optimization.

Optimizing By Solving The Lagrangian Dual Problem

Primal Problem

$$\text{minimize } L_p(\mathbf{w},b,\alpha_i)=\frac{1}{2}\|\mathbf{w}\|^2-\sum_{i=1}^{n}\alpha_i\left(y_i(\mathbf{w}^T\mathbf{x}_i+\text{b})-1\right)$$

$$\text{s.t. } \alpha_i \geq 0 \quad \text{for } i=1,2,\ldots n$$

$$\mathbf{w}=\sum_{\text{j}=1}^{\text{n}}\alpha_{\text{j}}\text{y}_{\text{j}}\mathbf{x}_{\text{j}}$$

Note that we need to maximize L_p(**w**,b, α_i) wrt α_i , and minimize it wrt **w** and b.

Lagrangian Dual Problem

$$\text{maximize } L(\alpha_i)=\sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$

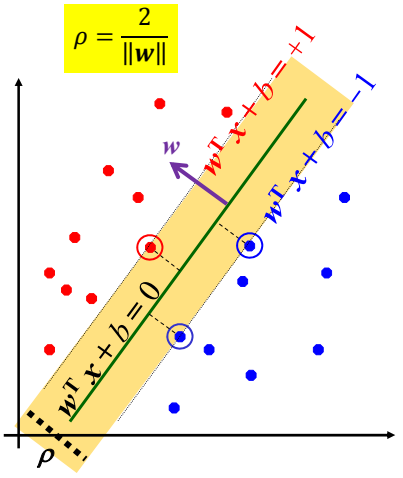
$$k_{ij}=\mathbf{x}_i^T\mathbf{x}_j$$

$$\text{s.t. } \alpha_i \geq 0, \quad \text{and} \quad \sum_{i=1}^{n}\alpha_i y_i=0$$

We got rid of w!!

SVM Optimization Formulation

- For support vectors (SV): $y_i(\mathbf{w}^T\mathbf{x}_i+\text{b})-1=0$
- From KKT condition, we know $\alpha_i\left(y_i(\mathbf{w}^T\mathbf{x}_i+\text{b})-1\right)=0$
- Thus, only SV's have: $\alpha_i \neq 0$
- The solution has the form: $\mathbf{w}=\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i=\sum_{i\in\text{SV}}\alpha_i y_i\mathbf{x}_i$
- get b from $y_i(\mathbf{w}^T\mathbf{x}_i+\text{b})-1=0$, where $\mathbf{x}_i$ is support vector



Linearly Separable Data - The Final Answer

- The maximum-margin linear discriminant function (*i.e.*, the SVM) is:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \mathrm{b} = \sum_{i \in \mathrm{SV}} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + \mathrm{b}$$

- The summation contains only the support vectors. Although there can potentially exist thousands of training samples, the number of SV's are typically significantly less. Furthermore, it relies only on the dot product between the test point and the support vectors.
- SVM's are typically used for 2-class problems, however, they can be extended to multi-class problems by either using "pairwise" (and pick the class with the highest vote), or "one against all" classification.
- The classification step for a sample $\boldsymbol{x}$ involves computing the discriminant function $g(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$, which is proportional to the distance of the sample from the decision hyperplane and can be interpreted as a measure of confidence in the classification.

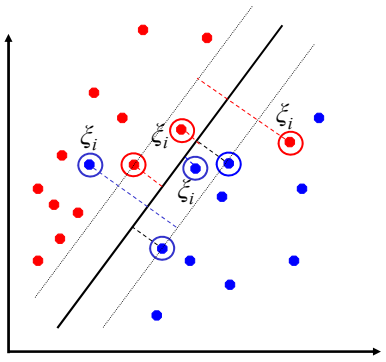
Linearly Non-Separable Case: Soft Margin Classification

- If the training set is not linearly separable, **slack variables** ξ_i can be added to allow misclassification of difficult or noisy examples, and the resulting margin is called **soft**.
- Three different categories of samples are identified:

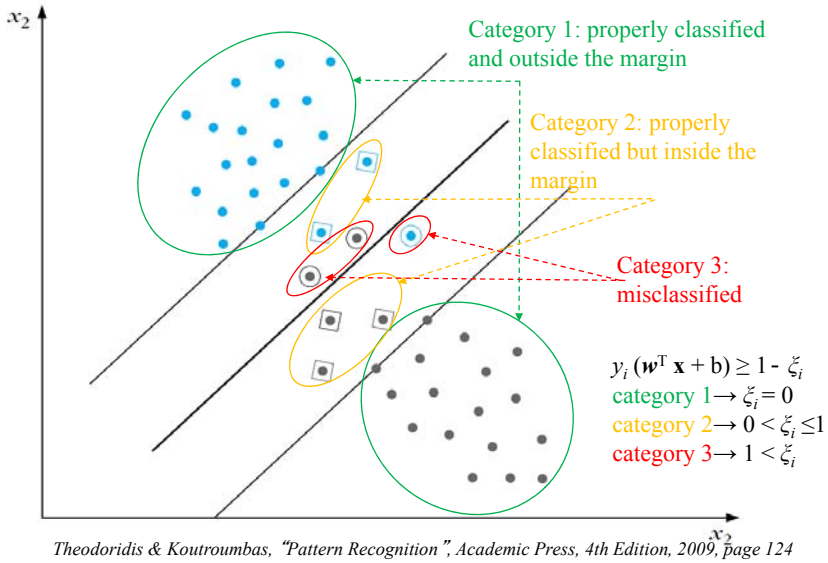
  i. Properly classified and outside the margin, where $\xi_i = 0$

  ii. Properly classified but inside the margin, where $0 < \xi_i \leq 1$,

  iii. Misclassified samples where $1 < \xi_i$.



Linearly Non-Separable Case: Soft Margin Classification



Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009, page 124

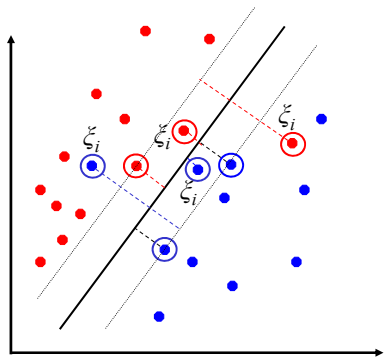
Linearly Non-Separable Case: Soft Margin Classification

Primal Problem Formulation

$$\text{minimize } L(\mathbf{w},\boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{such that } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$



Linearly Non-Separable Case: Soft Margin Classification

Lagrangian Dual Problem Formulation

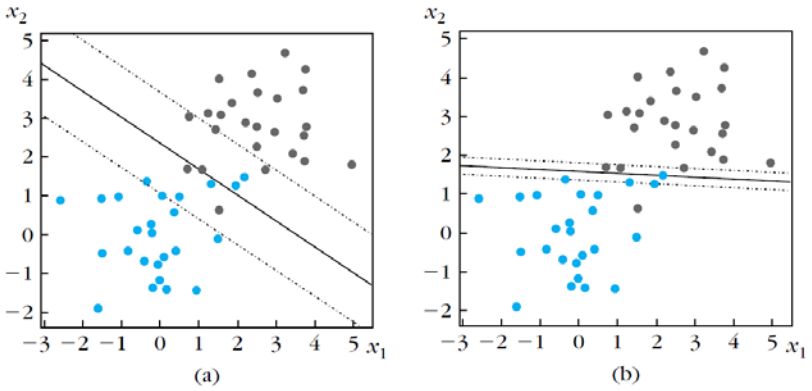
$$\text{maximize } L(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{such that } 0 \le \alpha_i \le C \quad \text{for all } i = 1, \ldots n$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

- Note that slack variables ξ_i do not appear in the dual problem!!
- Parameter C can be viewed as a way to control over-fitting. It controls the tradeoff between complexity of the SVM and the number of nonseparable points (outliers).

Linearly Non-Separable Case: Soft Margin Classification

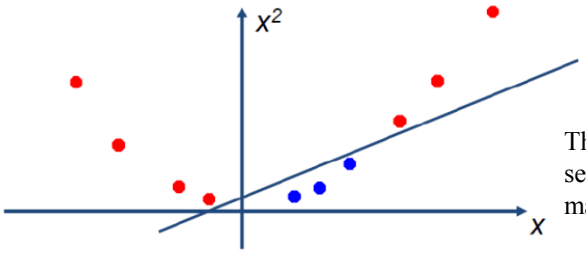


Example of two nonseparable classes and the resulting SVM (full line) with the associated margin (dotted line) for the values of (*a*) C=0.2 and (*b*) C=1000. In case (*b*), the location and direction of the classifier as well as the width of the margin have changed to include a smaller number of points inside the margin.

Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009, page 132

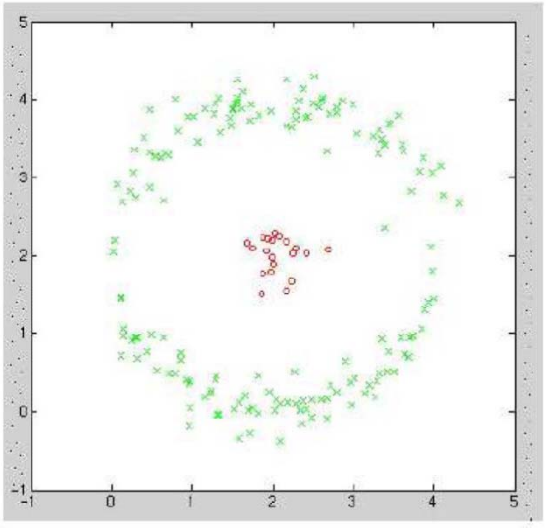
Nonlinear SVM Using Kernels

The two classes are not linearly separable in the input 1-D space



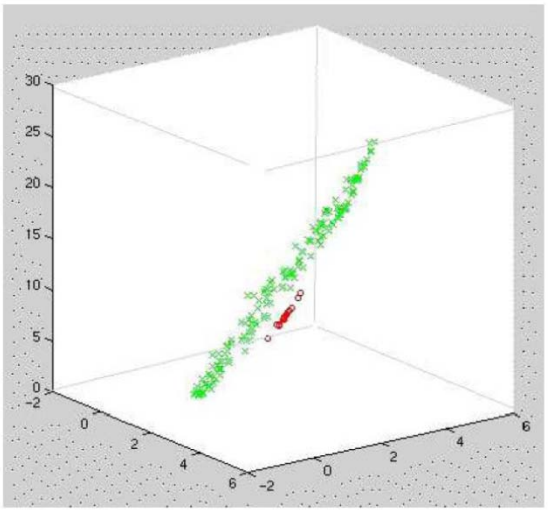
They become linearly separable by a simple mapping to 2-D space

Nonlinear SVM Using Kernels



The two classes are not linearly separable in the original 2-D space

Nonlinear SVM Using Kernels

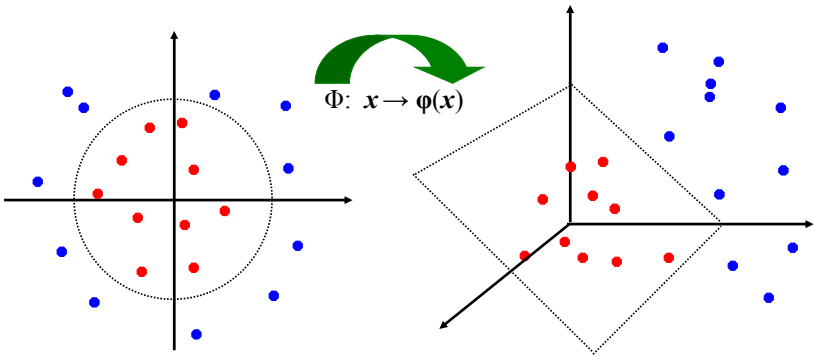


$$\Phi : \mathbf{R}^2 \to \mathbf{R}^3$$
$$(x_1, x_2) \mapsto (x_1, x_2, x_1^2 + x_2^2)$$

The data can be made linearly separable by a simple mapping to the 3-D space

Nonlinear SVM Using Kernels

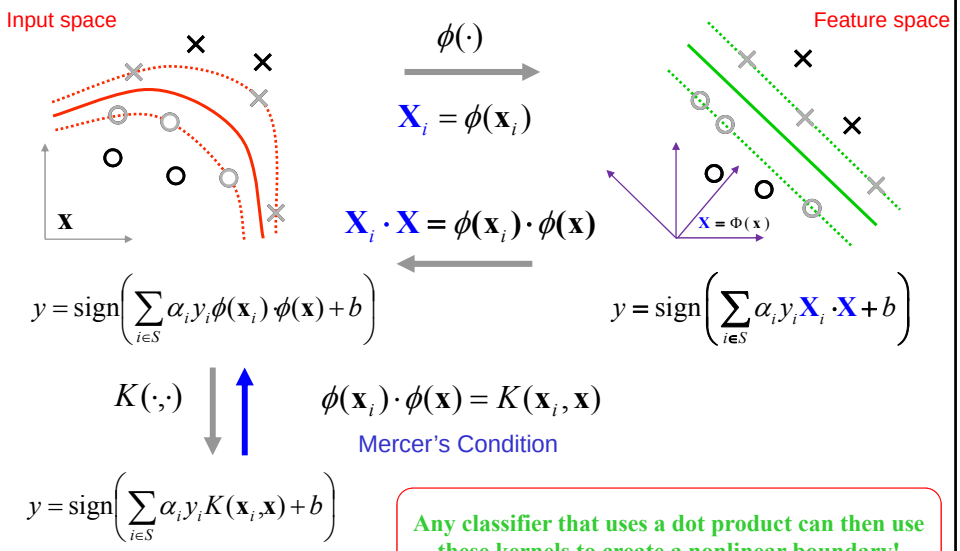
- When the training data is not linearly separable in the original feature space, it can often be mapped to some higher-dimensional space where is becomes separable.
- But, how well does this generalize?



Kernel Learning Methods

- The basic idea behind kernel-based techniques is the following
  - Implicitly map the input data into a high dimensional feature space (possibly infinite).
  - Reformulate the problem in terms of dot products in the feature space.
  - Implement dot products using kernel function in terms of the original data using the so-called **kernel trick**.
- Any classifier that uses a dot product can then use the kernel trick to create a nonlinear boundary!
- Unless the misclassification rate of the linear SVM is extremely low, there is nothing to be lost (computers are very fast these days) by trying a nonlinear kernel!

Using Kernels For Nonlinear Classification



Any classifier that uses a dot product can then use these kernels to create a nonlinear boundary!

Using Kernels For Nonlinear Classification

- With this mapping, our discriminant function is now:

$$g(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{i \in \text{SV}} \text{y}_i \ \alpha_i \ \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b$$

$k(\mathbf{x}_i, \mathbf{x}_j)$

- No need to know this mapping explicitly, because we only use the dot product of feature vectors in both the training and test.

- A *kernel function* is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$k(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Nonlinear SVM: Optimization

- Remember linear SVM formulation (Lagrangian Dual Problem)

$$\text{maximize} \ \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

- Nonlinear SVM Formulation: (Lagrangian Dual Problem)

$$\text{maximize} \ \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{such that} \quad 0 \le \alpha_i \le C$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

- The solution of the discriminant function is

$$g(\mathbf{x}) = \sum_{i \in \text{SV}} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b$$

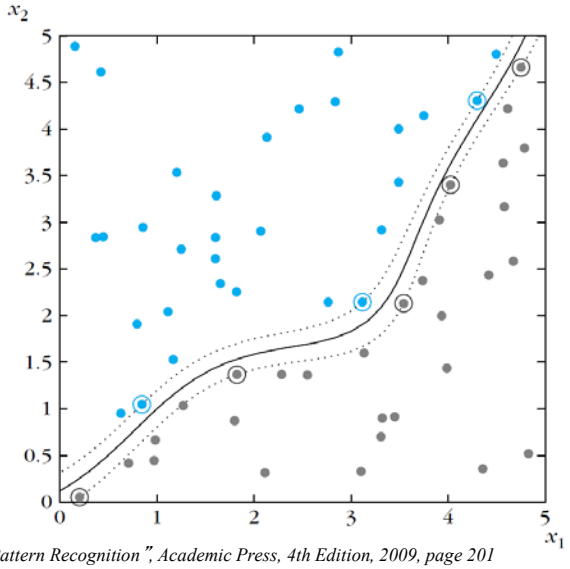
- The optimization technique is the same.

Examples of Common Kernel Functions

- In general, functions that satisfy Mercer's condition satisfy the requirements of being a kernel function. Common examples are
- Linear: $K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y}$
- Polynomial: $K(\boldsymbol{x}, \boldsymbol{y}) = (\theta + \boldsymbol{x}^T \boldsymbol{y})^d$
- Sigmoid: $K(\boldsymbol{x}, \boldsymbol{y}) = tanh(\kappa \boldsymbol{x}^T \boldsymbol{y} - \xi)$
- Radial Basis Function (RBF): $K(\boldsymbol{x}, \boldsymbol{y}) = exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2}\right)$

Nonlinear SVM Using SVM Kernel

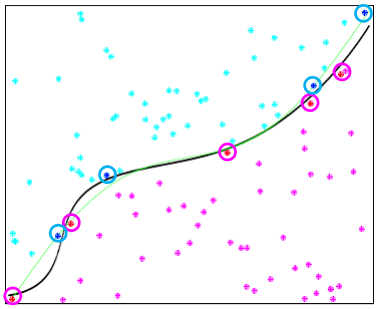
Example of a nonlinear SVM classifier using the Gaussian RBF kernel to separate two nonlinearly separable classes. Dotted lines mark the margin and circled points mark the support vectors. From Theodoridis and Koutroumbas, "Pattern Recognition," page 201.



Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009, page 201

Another Example

- 100, 2-D training points (50 per class) are generated according to a uniform distribution and based on a nonlinear boundary shown as the solid green line labeled GT.
- RBF kernel has been used to find a separating plane in the kernel space, which resulted in 9 SV's.
- The ratio of (9-1)/100 is a good prediction of the expected value of the out of training set error. Thus, the less complicated the model parameters (less SV's), the better it generalizes.



GT

SVM hyperplane in RBF domain mapped back to data domain

From the Caltech lecture series: Yaser S. Abu-Mostafa, Learning from Data- Lecture 15

Machine Intelligence & Deep Learning Workshop

Neural Networks

Majid Rabbani

Visiting Professor

The Kate Gleason COLLEGE OF ENGINEERING

Nonlinear Classifiers: The XOR Problem

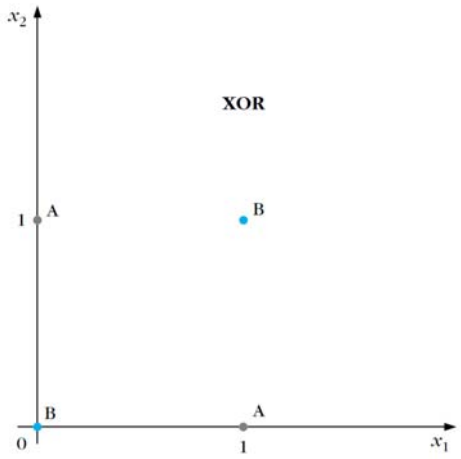


Table 4.1 Truth Table for the XOR Problem

x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B

The XOR Problem is not linearly separable

Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009

Linear Classifier For (a) the AND and (b) OR problems

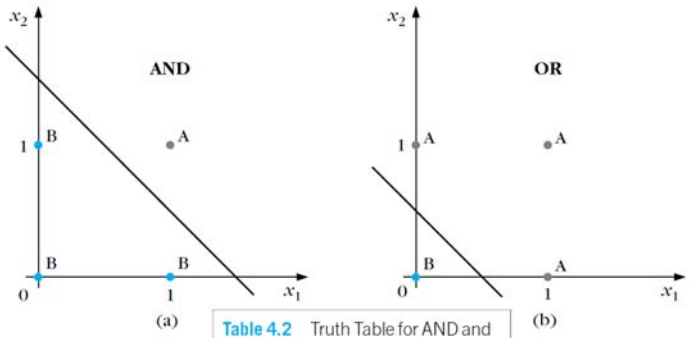
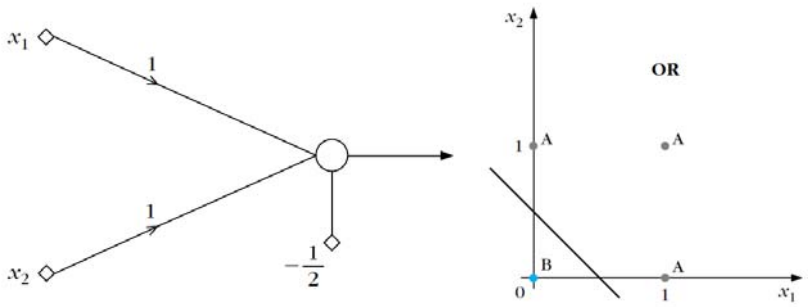


Table 4.2 Truth Table for AND and OR Problems

x_1	x_2	AND	Class	OR	Class
0	0	0	B	0	B
0	1	0	B	1	A
1	0	0	B	1	A
1	1	1	A	1	A

Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009

Linear Classifier (Perceptron) for Realizing The OR Gate



$$g(\boldsymbol{x}) = x_1 + x_2 - \frac{1}{2} = 0$$

Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009

Two-Layer Perceptron For Solving the XOR Problem

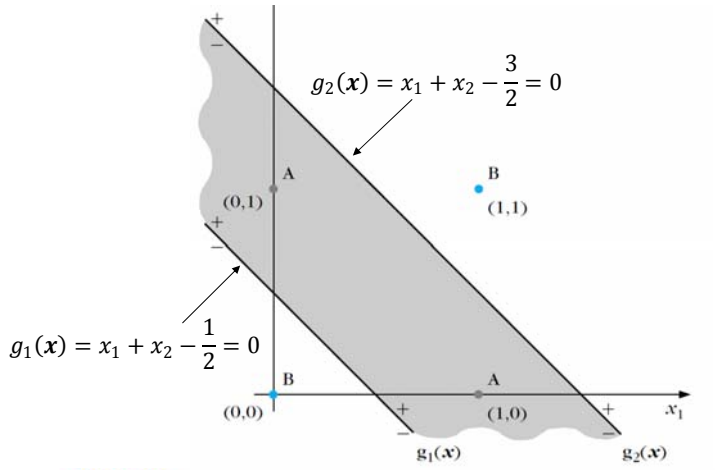


FIGURE 4.4

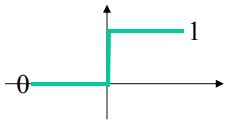
Decision lines realized by a two-layer perceptron for the XOR problem.

Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009

Two-Layer Perceptron For Solving the XOR Problem

$$y_i = f(g_i(\boldsymbol{x})) = \begin{cases} 1, & \text{if } g_i(\boldsymbol{x}) > 0 \\ 0, & \text{if } g_i(\boldsymbol{x}) < 0 \end{cases}$$

"Activation function" or "nonlinearity"



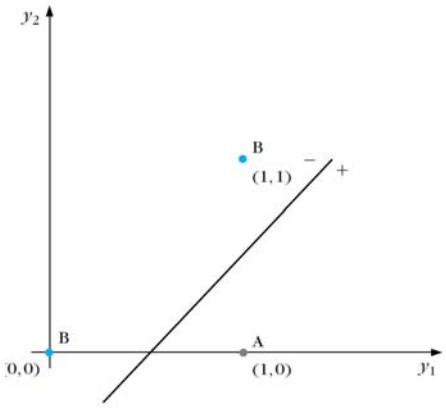
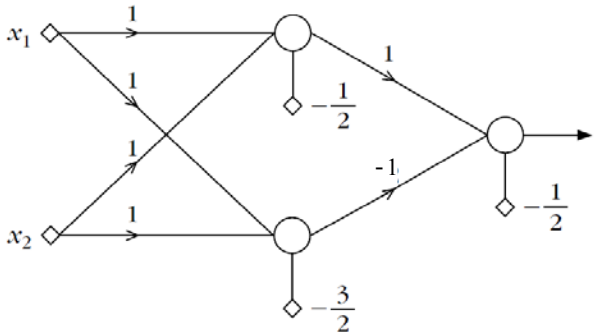


Table 4.3 Truth Table for the Two Computation Phases of the XOR Problem

1st Phase				
x_1	x_2	y_1	y_2	2nd Phase
0	0	0 (−)	0 (−)	B (0)
0	1	1 (+)	0 (−)	A (1)
1	0	1 (+)	0 (−)	A (1)
1	1	1 (+)	1 (+)	B (0)

Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009

Two-Layer Perceptron Solving the XOR Problem



Computations of the first phase perform a mapping that transforms the nonlinearly separable problem to a linearly separable one. This is an example of a 2-layer feed-forward network.

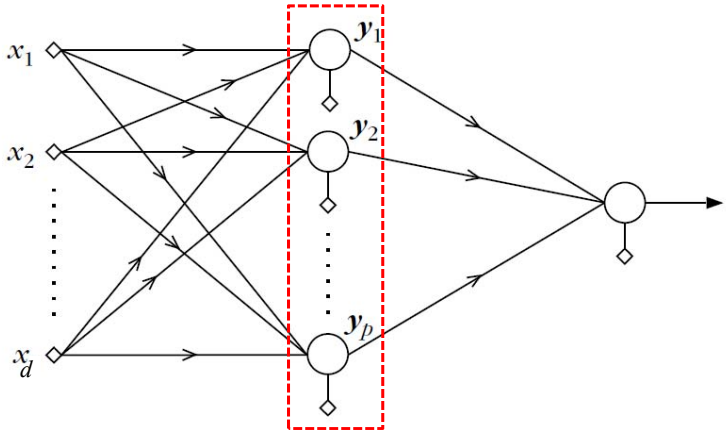
$$g_1(\boldsymbol{x}) = x_1 + x_2 - \frac{1}{2} = 0$$

$$g_2(\boldsymbol{x}) = x_1 + x_2 - \frac{3}{2} = 0$$

$$g(\boldsymbol{y}) = y_1 - y_2 - \frac{1}{2} = 0$$

Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009

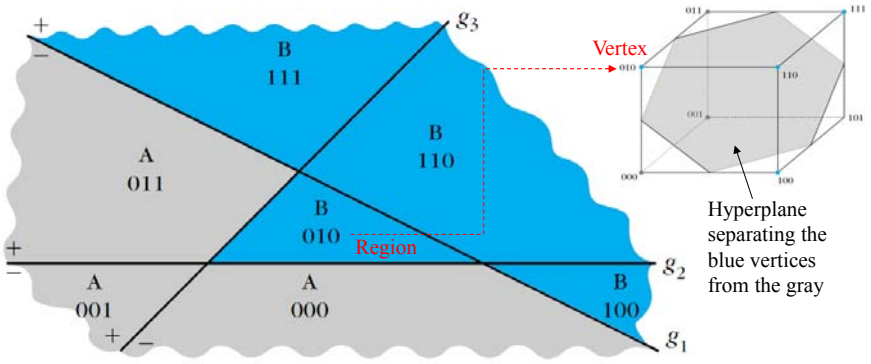
2-Layer Perceptron or 2-Layer Feedforward Neural Net



2-Layer Perceptron or 2-Layer Feedforward Neural Net

- The number of nodes in the input layer equals the dimension of the input space, d. No processing takes place in the input layer.
- The hidden layer applies the mapping with p neurons, each realizing a hyperplane. This layer of neurons divides the d-dimensional input space into polyhydra, which are formed by hyperplane intersections.
- The output of each of these neurons is subjected to an activation function (in this case, a step function, outputting zero or one) whose output depends on the relative position of $\boldsymbol{x}$ with respect to the hyperplane. All vectors located within one of these polyhydra regions are mapped onto a specific vertex of the unit H_p hypercube.
- The output neuron realizes another hyperplane, which separates the hypercube into two parts, having some of its vertices on one and some on the other. It provides the network with the potential to classify the vectors into classes consisting of unions of polyhedral regions.

Example of 2-Layer Neural Net with d=2, p=3

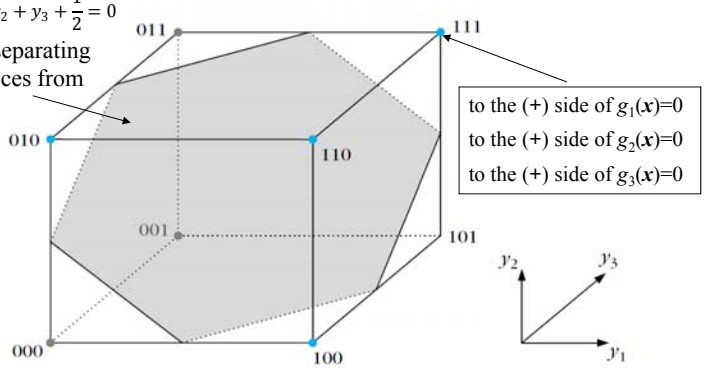


The neurons of the first hidden layer of a multilayer perceptron define hyperplanes in the d-dimensional space, the intersections of which form regions. Each region in the d-dimensional space is mapped into a vertex of the H_p unit hypercube.

2-Layer Perceptron or 2-Layer Feedforward Neural Net

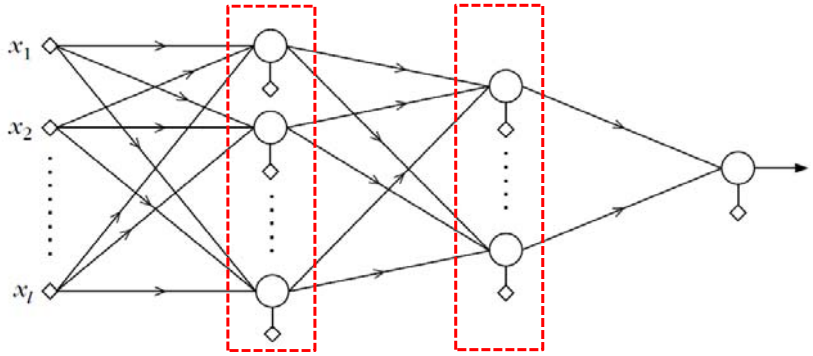
$g(\boldsymbol{y}) = -y_1 - y_2 + y_3 + \frac{1}{2} = 0$

Hyperplane separating the blue vertices from the gray



The two layer perceptron has the capability to classify vectors into **classes that consist of unions of polyhedral regions**. But **NOT ANY** union. For example the union of 000∪111∪110 vs the rest cannot be realized in this configuration.

Multi-layer Perceptron



Input layer Hidden layer 1, p nodes Hidden layer 2, k nodes Output layer

The 3-layer perceptron is capable of classifying vectors into classes consisting of **ANY** union of polyhedral regions. The idea is similar to the XOR problem. It realizes more than one planes in the $\boldsymbol{y} \in R^P$ space.

Three-Layer Perceptron

- Assume that all regions of interest are formed by intersections of p d-dimensional half-spaces defined by the p hyperplanes. These are realized by the p neurons of the first hidden layer, which also perform the mapping of the input space onto the vertices of the Hp hypercube of unit side length.
- Assume that class A consists of the union of k of the resulting polyhedra and class B of the rest. We use k neurons in the second hidden layer. Each of these neurons realizes a hyperplane in the p-dimensional space. The synaptic weights for each of the second-layer neurons are chosen so that the realized hyperplane leaves only one of the Hp vertices on one side and *all* the rest on the other.
- For each neuron a different vertex is isolated, that is, one of the k vertices in class A. That is, each time an input vector from class A enters the network, one of the k neurons of the second layer results in a 1 and the remaining $k-1$ give 0. In contrast, for class B vectors all neurons in the second layer output a 0.
- Classification is now a straightforward task. Choose the output layer neuron to realize an OR gate. Its output will be 1 for class A and 0 for class B vectors.

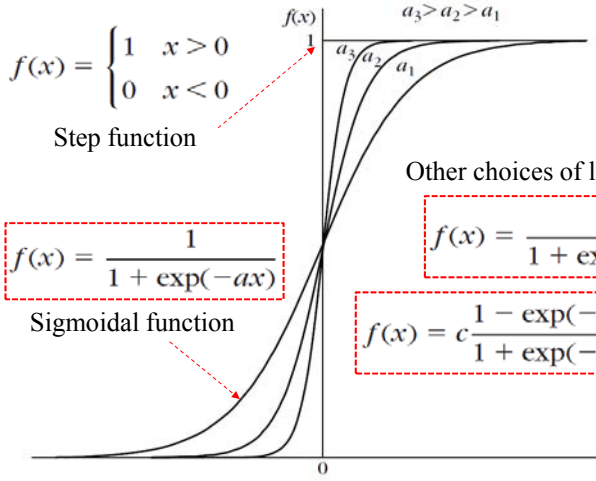
Designing Multi-Layer Neural Networks

- Two major directions exist:
  - The network is constructed in a way that classifies correctly *all* the available training data, by building it as a succession of linear classifiers. This could potentially result in large networks and is not very popular.
  - Construct the network by fixing the architecture and computing its synaptic parameters so as to **minimize an appropriate cost function of its output** by using an iterative procedure. This is most popular, and not only overcomes the drawback of the resulting large networks but also makes these networks powerful tools for a number of other applications, beyond pattern recognition.
  - The main difficulty in the latter approach is the discontinuity of the step (activation) function, prohibiting differentiation with respect to the unknown parameters (synaptic weights) which is a consequence of the cost function minimization procedure.

Logistics Functions

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

Step function



Other choices of logistic functions:

$$f(x) = \frac{2}{1 + \exp(-ax)} - 1$$

$$f(x) = \frac{1}{1 + \exp(-ax)}$$

Sigmoidal function

$$f(x) = c\frac{1 - \exp(-ax)}{1 + \exp(-ax)} = c\tanh\left(\frac{ax}{2}\right)$$

Theodoridis & Koutroumbas, "Pattern Recognition", Academic Press, 4th Edition, 2009

Algorithmic Steps

- Adopt an optimizing cost function between desired responses and actual responses of the network for the available training patterns. Note that we'll have to live with errors. We only try to minimize them, using certain criteria. Examples are:
  - Least Squares Error
  - Relative Entropy
- Adopt an algorithmic procedure for the optimization of the cost function with respect to the synaptic weights, e.g.,
  - Gradient descent
  - Newton's algorithm
  - Conjugate gradient

Backpropagation Algorithm

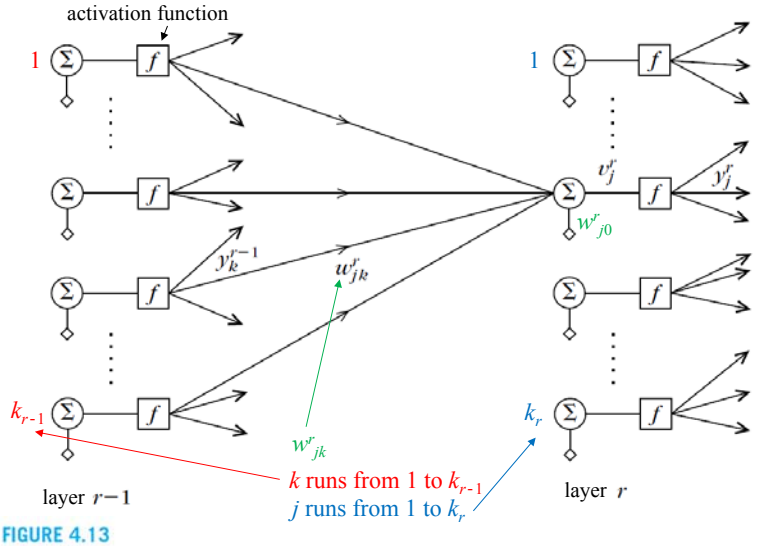


FIGURE 4.13

Definition of variables involved in the backpropagation algorithm.

Backpropagation Algorithmic Steps

1. Initialize unknown weights randomly with small values.
2. Compute the gradient terms backwards, starting with the weights of the last layer and then moving towards the first
3. Update the weights
4. Repeat the procedure until a termination procedure is met

- Two major philosophies exist:
  - Batch mode: The gradients of the last layer are computed once ALL training data have appeared to the algorithm, i.e., by summing up all error terms.
  - Pattern mode: The gradients are computed every time a new training data pair appears. Thus gradients are based on successive individual errors.

Choice of Cost Functions

- A drawback of most cost features is getting trapped in a local minimum. There are a class of cost functions, which guarantee that the gradient descent approach converges to a solution that classifies correctly ALL the training patterns, provided that one exists. They are known as "well formed" cost functions.
  - Least Squares (susceptible to getting trapped in local minima)
  - Cross-entropy (well formed function)
  - Classification error rate (aka as discriminative learning, tries to move the decision surfaces to minimize the classification error. Consequently, it puts more emphasis on the largest of the class *a posteriori* probability estimates. In contrast, the squared error cost function assigns the same importance to all posterior probability estimates. Most of the discriminative learning techniques use a smoothed version of the classification error, so as to be able to apply differentiation in association with gradient descent approaches, which can result in the danger that the minimization procedure will be trapped in a local minimum.

Choice of Cost Functions

Least-Squares cost function: $y_m(i)$ is the desired response of the m^{th} output neuron (1 or 0), while $\hat{y}_m(i)$ is the actual response of the m^{th} output neuron, in the interval [0, 1], for input $\boldsymbol{x}(i)$:

$$E(i) = \sum_{m=1}^{k} e_m^2(i) = \sum_{m=1}^{k} (y_m(i) - \hat{y}_m(i))^2$$

$$i = 1,2,\ldots, N$$

$$J = \sum_{i=1}^{N} E(i)$$

Cross-entropy cost function presupposes an interpretation of y and $\hat{y}$ as probabilities.

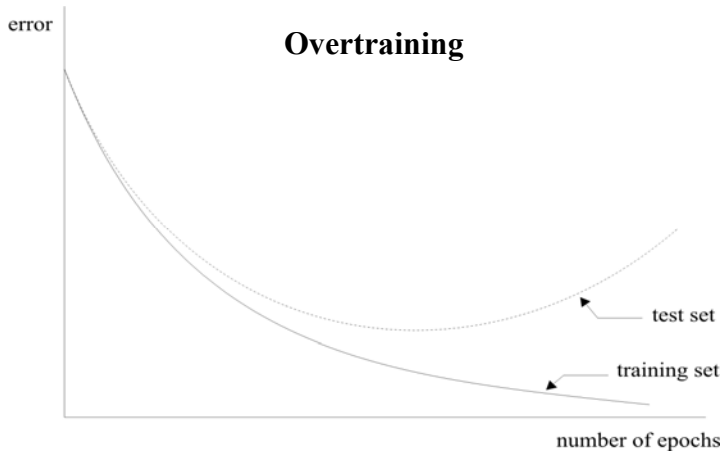
$$E(i) = \sum_{m=1}^{k} \{y_m(i) \ln \hat{y}_m(i) + (1 - y_m(i)) \ln(1 - \hat{y}_m(i))\}$$

$$J = \sum_{i=1}^{N} E(i)$$

Choice of Network Size

- How many layers and neurons per layer is need? Two major directions exist:
  - Pruning Techniques: These techniques start from a large network and then weights and/or neurons are removed iteratively, according to a criterion.
  - Constructive techniques: They start with a small network and keep increasing it, according to a predetermined procedure and criterion.
- Why not start with a large network and leave the algorithm to decide which weights are small? The term generalization refers to the capability of the multilayer neural network (and of any classifier) to classify correctly feature vectors that were not presented to it during the training phase. A large network can result in small errors for the training set, since it can learn the particular details of the training set. However, it will generalize poorly, that is perform poorly when presented with data unknown to it.
- The size of the network should be large enough to learn what makes data of the same class "similar" and data from different classes "dissimilar," and, at the same time, small enough w.r.t. n not to be able to learn underlying differences between data of the same class, which leads to the so called overfitting.

Overtraining



For the curve corresponding to the training set, we observe that the error keeps decreasing as the weights converge. For the curve that corresponds to the error of the validation set, initially the error decreases, but at some later stage it starts increasing. This is because the weights, computed from the training set, adapt to the idiosyncrasies of the specific training set, thus adversely affecting the generalization performance of the network.