パターン認識
Pattern Recognition

佐藤真一
Shin'ichi Satoh

国立情報学研究所
National Institute of Informatics

May 23, 2023

Course Information

- Course web page: https://research.nii.ac.jp/~satoh/utpr/
- Course materials can be found in the above page
- Course videos can be found in ITC-LMS
- Credits will be given based on final report (mandatory)
- Attendance record will not be taken
- Assignments may be imposed (3 out of 7 are mandatory: subject to change)
- If you fail to submit minimum 3 assignments and final report, you will not obtain credits.

Schedule (subject to change)

4/11		Orientation, Bayes decision theory, probability distribution
4/18		Random variable, random vector, normal distributions
4/25		Parametric density estimation, discriminant function
5/2		Nonparametric density estimation, Parzen windows, k-nearest neighbor estimate
5/9		k-nearest neighbor classification, classification error estimation
5/16		Bayes error estimation, classification error estimation, cross-validation, bootstrap
5/23	Hybrid	Linear classifier, perceptron, MSE classifier, Widrow-Hoff rule
5/30	Online, zoom only	neural network, deep learning
6/6	Hybrid	all about SVM
6/13	Online, zoom only	Orthogonal expansions, Eigenvalue decomposition
6/20		no class
6/27	Hybrid	Clustering, dendrogram, aggromerative clustering, k-means
7/4	Hybrid	Graphs, normalized cut, spectral clustering, Laplacian

Today's agenda

- Linear classifier / linear discriminant function
- Perceptron

Discriminant Functions

Let's consider the problem to classify given observation into one of c classes.
We can formalize this problem using discriminant functions $g_i(x)$, $i = 1, \ldots, c$.
The classifier is then assign a feature vector x to class ω_i if

$$g_i(x) > g_j(x) \text{ for all } j \neq i.$$

Linear Discriminant Functions

Assume that the input is d-dimensional vector:

$$x = [x_1 \, x_2 \, \cdots \, x_d]^T$$

and weight vector:

$$w = [w_1 \, w_2 \, \cdots \, w_d]^T.$$

The linear discriminant function is

$$g(x) = w_0 + \sum_{j=1}^{d} w_j x_j.$$

Linear Discriminant Functions

We consider an augmented feature vector and an augmented weight vector:

$$\hat{x} = [\textcolor{red}{1}\; x_1\; x_2\; \cdots\; x_d]^T$$
$$\hat{w} = [\textcolor{red}{w_0}\; w_1\; w_2\; \cdots\; w_d]^T.$$

The linear discriminant function is then:

$$\begin{aligned} g(x) &= w_0 + \sum_{j=1}^{d} w_j x_j \\ &= w_0 + w^T x \\ &= \hat{w}^T \hat{x}. \end{aligned}$$

(We will use w and $\hat{w}$, x and $\hat{x}$ interchangeably if not ambiguous.)

Linear Discriminant Functions

When a linear discriminant function is used?
Parametric density estimation case when $\Sigma_i = \sigma^2 I$.
Then the discriminant functions are:

$$g_i(x) = -\frac{||x - \mu_i||^2}{2\sigma^2} + \log P(\omega_i)$$
$$= -\frac{1}{2\sigma^2}[x^T x - 2\mu_i^T x + \mu_i^T \mu_i] + \log P(\omega_i).$$

By ommitting common terms we obtain linear discriminant functions:

$$g_i(x) = w_i^T x + w_{i0}$$

where

$$w_i = \frac{1}{\sigma^2}\mu_i \text{ and } w_{i0} = -\frac{1}{2\sigma^2}\mu_i^T \mu_i + \log P(\omega_i).$$

The nearest neighbor classifier is another example.
Assume that there are n prototypes $p_1, p_2, \cdots, p_n$.
The nearest neighbor rule chooses the prototype to whom distance from input vector x is smallest:

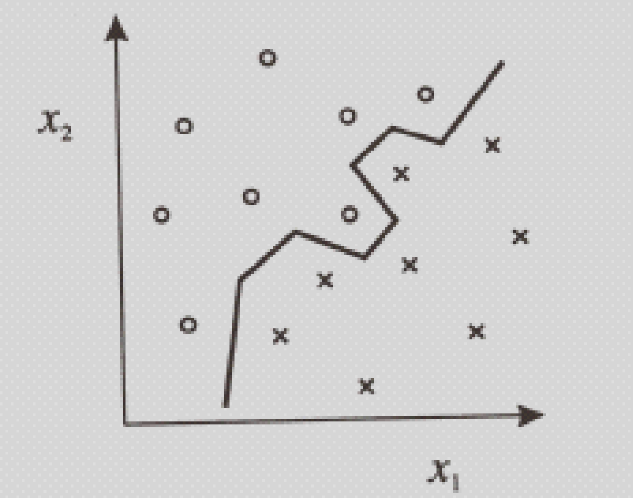
$$\|x - p_i\|^2 = \|x\|^2 - 2p_i^T x + \|p_i\|^2.$$

Thus the discriminant function:

$$g_i(x) \stackrel{\mathrm{def}}{=} p_i^T x - \frac{1}{2}\|p_i\|^2.$$

The decision boundaries are piece-wise linear.

Linear Discriminant Functions



Linear Discriminant Functions

How do we obtain a linear discriminant function from a given set of training data?
Consider the set of training data $\mathcal{X}$ where the set of the training data of each class ω_i is $\mathcal{X}_i$ $(i = 1, 2, \cdots, c)$.
The training of linear discriminant functions is to determine $\hat{w}_i$ so that for all samples in $\mathcal{X}_i$,

$$g_i(x) > g_j(x) \text{ for all } j \neq i.$$

holds.
If there are such $\hat{w}_i$s, $\mathcal{X}$ is said to be linearly separable.

Linear Discriminant Functions

Let's consider two classes case (ω_1 and ω_2).
We then can simplify the problem by

$$
\begin{aligned}
g(x) &= g_1(x) - g_2(x) = (w_1 - w_2)^T x \\
&= w^T x \\
w &\stackrel{\text{def}}{=} w_1 - w_2
\end{aligned}
$$

and

decide ω_1 if $g(x) = w^T x > 0$

decide ω_2 if $g(x) = w^T x < 0$.

We can then normalize data by replacing all samples labeled ω_2 by their negatives.
Then $g(x) = w^T x > 0$ for all training samples.

The Perceptron

Rosenblatt, 1957[1]
We want to determine w so that $w^T x > 0$ for all training samples.
The Perceptron criterion function:

$$J(w) = \sum_{x \in \tilde{\mathcal{X}}} (-w^T x)$$

where $\tilde{\mathcal{X}}$ is the set of samples misclassified.
J is never negative, and we want it to be zero ($\tilde{\mathcal{X}}$ to be empty).

[1] F. Rosenblatt, The perceptron - A perceiving and recognizing automaton, Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January, 1957.

The Perceptron

Since the gradient of J is

$$\nabla J = \sum_{x \in \tilde{\mathcal{X}}} (-x)$$

we can update the weight vector based on Gradient Descent (aka batch Gradient Descent) as

$$w(k+1) = w(k) - \rho \nabla J = w(k) + \rho \sum_{x \in \tilde{\mathcal{X}}} x.$$

ρ is called learning rate.

The Perceptron

Algorithm 1 Batch Perceptron

Initialization: w,ρ, criterion θ, $k = 0$
repeat
    $k \leftarrow k + 1$
    $w \leftarrow w + \rho \sum_{\tilde{x} \in \tilde{\mathcal{X}}} \tilde{x}$
    Recalculate $\tilde{\mathcal{X}}$
until $|\rho \sum_{\tilde{x} \in \tilde{\mathcal{X}}} \tilde{x}| < \theta$
Return w

The Perceptron

We can instead use Stocastic (or on-line) Gradient Descent:

Algorithm 2 Fixed-Increment Single-Sample Perceptron

```
Initialization: $w$, $\tilde{\mathcal{X}} = \{\tilde{x}_1, \tilde{x}_2, \cdots\}$, $k = 0$
repeat
    $k \leftarrow k + 1$
    $i \leftarrow \mathrm{mod}(k, |\tilde{\mathcal{X}}|) + 1$
    $w \leftarrow w + \rho \tilde{x}_i$
    Recalculate $\tilde{\mathcal{X}}$
until all patterns properly classified
Return $w$
```

The Perceptron

The Perceptron is known to converge if given training data is linearly separable.

Minimum Squared-Error and the Pseudoinverse

Let's assume that we have n training data

$$\mathcal{X} = \{x_1, x_2, \cdots, x_n\}.$$

Given p-th data x_p, we observe outputs of c discriminant functions as a vector

$$[g_1(x_p)\, g_2(x_p)\, \cdots\, g_c(x_p)]^T.$$

We further assume that we have a vector as the target signal

$$b_p = [b_{1p}\, b_{2p}\, \cdots\, b_{cp}]^T.$$

Note that $b_{ip} > b_{jp}$ $(j \neq i)$ if $x_p \in \mathcal{X}_i$.
For example,

$$b_p = [0\, \cdots\, 0\, \underset{i}{1}\, 0\, \cdots\, 0]^T$$

for $x_p \in \mathcal{X}_i$.
Then we want to determine w_p such that $g_i(x_p) \approx b_{ip}$.

Minimum Squared-Error and the Pseudoinverse

The error for a pattern x_p is $\varepsilon_{ip} = g_i(x_p) - b_{ip}$.
The criterion function:

$$\begin{aligned} J_p(\hat{w}_1, \hat{w}_2, \cdots, \hat{w}_c) &= \frac{1}{2}\sum_{i=1}^{c} \varepsilon_{ip}^2 \\ &= \frac{1}{2}\sum_{i=1}^{c} (g_i(x_p) - b_{ip})^2 \\ &= \frac{1}{2}\sum_{i=1}^{c} (\hat{w}_i{}^T \hat{x}_p - b_{ip})^2 \end{aligned}$$

Minimum Squared-Error and the Pseudoinverse

The batch criterion function:

$$
\begin{aligned}
J(w_1, w_2, \cdots, w_c) &= \sum_{p=1}^{n} J_p(w_1, w_2, \cdots, w_c) \\
&= \frac{1}{2} \sum_{p=1}^{n} \sum_{i=1}^{c} (g_i(x_p) - b_{ip})^2 \\
&= \frac{1}{2} \sum_{p=1}^{n} \sum_{i=1}^{c} (\hat{w}_i{}^T \hat{x}_p - b_{ip})^2
\end{aligned}
$$

We want w_is which minimize the function.
In other words,

$$
\begin{bmatrix} \hat{w}_1 \ \hat{w}_2 \ \cdots \ \hat{w}_c \end{bmatrix}^T \begin{bmatrix} \hat{x}_1 \ \hat{x}_2 \ \cdots \ \hat{x}_n \end{bmatrix} \approx \begin{bmatrix} & \vdots & \\ \cdots & b_{ip} & \cdots \\ & \vdots & \end{bmatrix}
$$

Minimum Squared-Error and the Pseudoinverse

Two class case

$$\begin{aligned} J_p(\hat{w}) &= \frac{1}{2}(g(x_p) - b_p)^2 \\ &= \frac{1}{2}(\hat{w}^T \hat{x}_p - b_p)^2 \end{aligned}$$

where b_p can be

$$b_p = \begin{cases} 1 & (x_p \in \mathcal{X}_1) \\ -1 & (x_p \in \mathcal{X}_2) \end{cases}$$

Minimum Squared-Error and the Pseudoinverse

Now we minimize J in closed form using the pseudoinverse.

$$\nabla J = \frac{\partial J}{\partial \hat{w}} = [\frac{\partial J}{\partial w_0} \; \frac{\partial J}{\partial w_1} \; \cdots \; \frac{\partial J}{\partial w_d}]$$

We can minimize J by

$$\frac{\partial J}{\partial \hat{w}_i} = \nabla_i J = 0 \; (i = 1, \cdots, c)$$

namely,

$$\begin{aligned} \frac{\partial J}{\partial \hat{w}_i} &= \sum_{p=1}^{n} \frac{\partial J_p}{\partial \hat{w}_i} \\ &= \sum_{p=1}^{n} (\hat{w}_i{}^T \hat{x_p} - b_{ip}) \hat{x_p} = 0 \end{aligned}$$

Minimum Squared-Error and the Pseudoinverse

We assume

$$X = [\hat{x}_1 \, \hat{x}_2 \, \cdots \, \hat{x}_n]^T$$
$$b_i = [b_{i1} \, b_{i2} \, \cdots \, b_{in}]^T \ (i = 1, \cdots, c)$$

then the criterion function will be

$$J(\hat{w}_1, \hat{w}_2, \cdots, \hat{w}_c) = \frac{1}{2}\sum_{i=1}^{c} \|X\hat{w}_i - b_i\|^2$$
$$\frac{\partial J}{\partial \hat{w}_i} = X^T(X\hat{w}_i - b_i) = 0$$
$$X^T X\hat{w}_i = X^T b_i$$
$$\hat{w}_i = (X^T X)^{-1} X^T b_i$$

This gives MSE solution of $\|Xw_i - b_i\|^2$.

Minimum Squared-Error and the Pseudoinverse

$X^{+} = (X^T X)^{-1} X^T$ is called the pseudoinverse of X.

The Widrow-Hoff or LMS Procedure

We now consider gradient descent:

$$\hat{w}_i(k+1) = \hat{w}_i(k) - \rho\frac{\partial J}{\partial \hat{w}_i} = \hat{w}_i(k) - \rho\nabla_i J$$

$$\Delta\hat{w}_i = -\rho\nabla_i J$$

We can also consider stochastic gradient descent:

$$\Delta\hat{w}_i = -\rho\frac{\partial J_p}{\partial \hat{w}_i}$$

The Widrow-Hoff or LMS Procedure

Here we denote $g_i(x_p)$ as g_{ip}.

$$\frac{\partial J_p}{\partial \hat{w}_i} = \frac{\partial J_p}{\partial g_{ip}} \frac{\partial g_{ip}}{\partial \hat{w}_i}$$

where

$$\frac{\partial J_p}{\partial g_{ip}} = g_{ip} - b_{ip} = \varepsilon_{ip}$$
$$\frac{\partial g_{ip}}{\partial \hat{w}_i} = \hat{x}_p$$

therefore

$$\frac{\partial J_p}{\partial \hat{w}_i} = (g_{ip} - b_{ip})\hat{x}_p = \varepsilon_{ip}\hat{x}_p$$

The Widrow-Hoff or LMS Procedure

The update rule becomes

$$\begin{aligned}\Delta \hat{w}_i &= -\rho \varepsilon_{ip} \hat{x}_p \\ &= -\rho (g_{ip} - b_{ip}) \hat{x}_p \\ &= -\rho (\hat{w}_i{}^T \hat{x}_p - b_{ip}) \hat{x}_p .\end{aligned}$$

This is the Widrow-Hoff or LMS rule (least-mean-squared).

The Widrow-Hoff or LMS Procedure

Algorithm 3 LMS

Initialization: $\hat{w}_i$, $k = 0$
repeat
    $k \leftarrow \mathrm{mod}(k, |\mathcal{X}|) + 1$
    $\hat{w}_i \leftarrow \hat{w}_i - \rho(\hat{w}_i{}^T \hat{x}_k - b_{ip})\hat{x}_k$
until all patterns properly classified
Return $\hat{w}_i$

Assignment

- Programming project and non-programming project are imposed.
- You are expected to solve either programming project OR non-programming project.
- Programming project is recommended.
- Of course you are most welcomed to solve both.
- Due on June 6.

Programming Project

Generate three sets of training data using the following matlab programs:

Linearly separable linear.m

Linearly non-separable nonlinear.m

Skewed linearly separable slinear.m

In Python case, put one of the following lines in your program:

Linearly separable from linear import *

Linearly non-separable from nonlinear import *

Skewed linearly separable from slinear import *

(1) Implement augmented feature/weight vectors and normalization, and the Perceptron (batch and/or on-line). Train the Perceptron with the three data sets. Discuss on its behavior. Hint: augumented feature vectors can be obtained by:

```
ax=np.concatenate((np.ones((1,n)),x))
```

(2) Then implement MSE classifier and train it with the three data sets. Discuss on its behavior.

Programming Project (1)

```
import numpy as np
import matplotlib.pyplot as plt
from linear import *

rho = 0.1
ax = np.concatenate((np.ones((1, n)), x))
aw = (2 * np.random.rand(d + 1) - np.array([1, 1, 1]))[:, np.newaxis]
ax[:, np.where(l == -1)] = -ax[:, np.where(l == -1)]
plt.figure()
k = 0
neg = ((ax.T.dot(aw)).T < 0)[-1]
```

Programming Project (1)

```
while len(np.where(neg)[-1]) > 0:
    k += 1
    aw += rho*<<< some code to update aw >>>
    neg = <<< some code to update neg >>>
    plt.clf()
    plt.xlim([-1, 1])
    plt.ylim([-1, 1])
    plt.plot(x[0, np.where((l == 1) & ~neg)],
             x[1, np.where((l == 1) & ~neg)], 'bo')
    plt.plot(x[0, np.where((l == -1) & ~neg)],
             x[1, np.where((l == -1) & ~neg)], 'bx')
    plt.plot(x[0, np.where((l == 1) & neg)],
             x[1, np.where((l == 1) & neg)], 'ro')
    plt.plot(x[0, np.where((l == -1) & neg)],
             x[1, np.where((l == -1) & neg)], 'rx')
```

Programming Project (1)

```
    if abs(aw[1]) > abs(aw[2]):
        plt.plot([-1, 1], [-(aw[0] - aw[1]) / aw[2], -(aw[0] + aw[1]) / aw[2]])
    else:
        plt.plot([-(aw[0] - aw[2]) / aw[1], -(aw[0] + aw[2]) / aw[1]], [-1, 1])
    print(aw)
    plt.pause(0.2)
plt.show()
```

Programming Project (2)

```
import numpy as np
import matplotlib.pyplot as plt
from linear import *

ax = np.concatenate((np.ones((1, n)), x))
aw = <<< some code to compute aw >>>
neg = (ax.T.dot(aw)).T < 0
# Similar code to perceptron follows...
```

Similar code to perceptron follows...

Non-Programming Project

(1) Show the proof of the Perceptron convergence theorem (batch and/or on-line).
(2) Show that MSE solution is obtained by pseudo inverse. Namely, assuming that

$$J_p(\hat{w}_1, \hat{w}_2, \cdots, \hat{w}_c) = \frac{1}{2}\sum_{i=1}^{c}(\hat{w}_i{}^T \hat{x}_p - b_{ip})^2$$

derive that the solution of $\dfrac{\partial J}{\partial \hat{w}_i} = 0$ is $\hat{w}_i = X^+ b_i$.