

Preclass 03: Linear Classification

[SCS4049] Machine Learning and Data Science

Seongsik Park (s.park@dgu.edu)

School of AI Convergence & Department of Artificial Intelligence, Dongguk University

Linear classification

MNIST dataset

- 70,000 images of 28×28 handwritten digits from 0 to 9
- “Hello, World!” of machine learning



Figure 3-1. A few digits from the MNIST dataset

Classification problem

The goal in classification

- Take an input vector $\mathbf{x}$
- Assign it to one-of- K discrete class $\mathcal{C}_k$ where $k = 1, 2, \dots, K$
- Assign each input to one and only one class (disjoint)

The input space divided into decision region, whose boundaries are called **decision boundaries** or **decision surfaces**.

- $(D-1)$ -dimensional hyperplanes within the D -dimensional input space
- Linearly separable dataset that can be separated exactly by linear decision surface

Target output $\mathbf{t}$

- Two-class problem, $t \in \{0, 1\}$, e.g., $t = 1$ represents $\mathcal{C}_1$
- $K > 2$ classes, $\mathbf{t} \in \{0, 1\}^K$ and $\sum_k t_k = 1$
- t_k is probability of $\mathcal{C}_k$ and $\sum_k t_k = 1$

Confusion matrix

Confusion matrix (contingency matrix)

		Predicted	
		Negative	Positive
Ground truth	Negative	True Negative	False Positive (Type I Error)
	Positive	False Negative (Type II Error)	True Positive

True positive rate (TPR): $TPR = \frac{TP}{TP + FN}$ (recall, sensitivity, hit rate)

Positive predictive value (PPV): $PPV = \frac{TP}{TP + FP}$ (precision)

Accuracy (ACC): $ACC = \frac{TP + TN}{TP + FN + TN + FP}$

True negative rate (TNR): $TNR = \frac{TN}{TN + FP}$ (specificity)

False negative rate (FNR): $FNR = 1 - TPR$

False positive rate (FPR): $FPR = 1 - TNR$

Linear discriminant function

Two classes

$$y(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} + \theta_0$$

$\mathbf{x}$ is assigned to class $\mathcal{C}_1$ if $y(\mathbf{x}) \geq 0$
to class $\mathcal{C}_2$ otherwise

Decision surface

- $\boldsymbol{\theta}$ determines the orientation
- θ_0 determines the location
- $-\theta_0/\|\boldsymbol{\theta}\|$ is the normal distance from the origin

Linear discriminant function

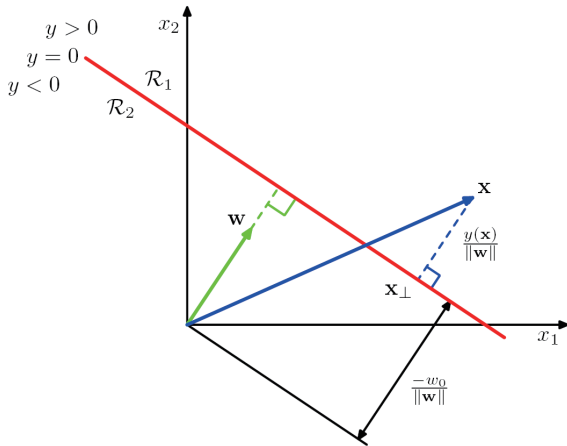


Figure 1: Decision surface of linear discriminant function

Normal equation for linear classification

Two classes

$$y(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} + \theta_0$$

$\mathbf{x}$ is assigned to class $\mathcal{C}_1$ if $y(\mathbf{x}) \geq 0$
to class $\mathcal{C}_2$ otherwise

Minimizing sum of squared errors

→ closed form solution for given a training dataset

$\{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_m, t_m)\}$

$$J(\boldsymbol{\theta}) = \|\mathbf{X}\boldsymbol{\theta} - \mathbf{t}\|_2^2$$

where $\mathbf{t} \in \{0, 1\}^m$ $\mathbf{X} \in \mathbb{R}^{m \times (n+1)}$

$$\text{then } \hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$$

Is it working?

Least squares for classification

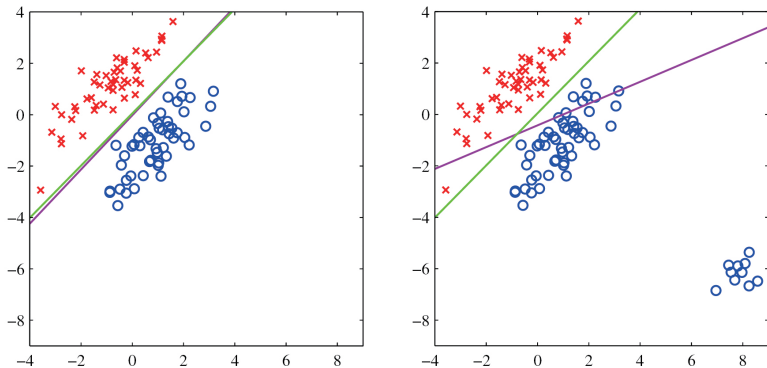


Figure 2: Two classes: least square and logistic regression

Generalized linear model

- $y(\mathbf{x}) = f(\boldsymbol{\theta}^T \mathbf{x} + \theta_0)$
- Activation function $f: \Re \rightarrow (0, 1)$
- Even if the function $f(\cdot)$ is nonlinear, the decision surfaces $\boldsymbol{\theta}^T \mathbf{x} + \theta_0 = \text{const}$ are linear functions of $\mathbf{x}$

Perceptron

The perceptron algorithm

Generalized linear model

$$y(\mathbf{x}) = f(\boldsymbol{\theta}^T \mathbf{x})$$

Nonlinear activation function given by a step function

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

Target values $t = +1$ for class $\mathcal{C}_1$ and $t = -1$ for class $\mathcal{C}_2$.

The perceptron algorithm

Perceptron criterion that we want to minimize

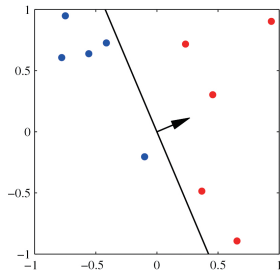
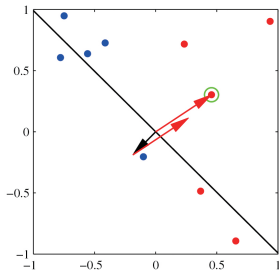
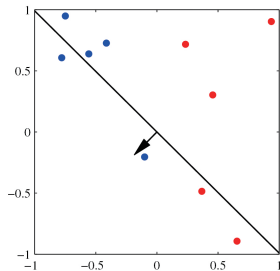
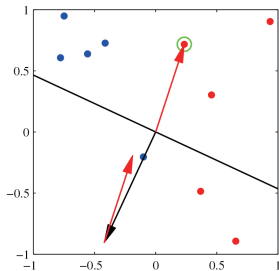
$$J(\boldsymbol{\theta}) = - \sum_{n \in \mathcal{M}} \boldsymbol{\theta}^T \mathbf{x}_n t_n$$

Misclassified pattern $\mathcal{M} = \{\mathbf{x}_n : \boldsymbol{\theta}^T \mathbf{x}_n t_n < 0\}$

The stochastic gradient descent algorithm

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \eta \nabla J(\boldsymbol{\theta}) = \boldsymbol{\theta}^{(\tau)} + \eta \mathbf{x}_n t_n$$

The perceptron algorithm



Logistic regression

We begin our treatment of generalized linear models by considering the problem of two-class classification. Under rather general assumptions, the posterior probability of class $\mathcal{C}_1$ can be written as a logistic sigmoid acting on a linear function of the feature vector $\mathbf{x}$ so that

$$p(\mathcal{C}_1 \mid \mathbf{x}) = y(\mathbf{x}) = \sigma(\boldsymbol{\theta}^T \mathbf{x}) \quad (1)$$

with $p(\mathcal{C}_2 \mid \mathbf{x}) = 1 - p(\mathcal{C}_1 \mid \mathbf{x})$. Here, $\sigma(\cdot)$ is the *logistic sigmoid* function defined by

$$\sigma(a) = \frac{1}{1 + \exp(-a)}. \quad (2)$$

In the terminology of statistics, this model is known as *logistic regression*, although it should be emphasized that this is a model for classification rather than regression.

Likelihood probability

For a data set $\{\mathbf{x}_n, t_n\}$, where $t_n \in \{0, 1\}$ with $n = 1, \dots, N$, the likelihood function can be written

$$p(\mathbf{t} \mid \boldsymbol{\theta}) = \prod_{n=1}^N y_n^{t_n} \{1 - y_n\}^{1-t_n} \quad (3)$$

where $\mathbf{t} = (t_1, \dots, t_N)^T$ and $y_n = p(\mathcal{C}_1 \mid \phi_n)$.

Cross entropy and gradient

As usual, we can define an error function by taking the negative logarithm of the likelihood, which gives the *cross-entropy* error function in the form

$$J(\boldsymbol{\theta}) = -\log p(\mathbf{t} \mid \boldsymbol{\theta}) = -\sum_{n=1}^N \{t_n \log y_n + (1 - t_n) \log(1 - y_n)\} \quad (4)$$

where $y_n = \sigma(a_n)$ and $a_n = \boldsymbol{\theta}^T \boldsymbol{\phi}_n$. Taking the gradient of the error function with respect to $\boldsymbol{\theta}$, we obtain

$$\nabla J(\boldsymbol{\theta}) = \sum_{n=1}^N (y_n - t_n) \mathbf{x}_n. \quad (5)$$

The contribution to the gradient from data point n is given by the error $y_n - t_n$ between the target value and the prediction of the model, times the basis function vector $\mathbf{x}_n$. This takes precisely the same form as the gradient of the sum-of-squares error function for the linear regression model.

Reference and further reading

- “Chap 4” of C. Bishop, Pattern Recognition and Machine Learning
- “Chap 3” of A. Geron, Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow