Linear Classification Models

Fabio A. González Ph.D.

Depto. de Ing. de Sistemas e Industrial
Universidad Nacional de Colombia, Bogotá

April 9, 2018

Content

Outline

Classification problems

- $predict(x) = \begin{cases} C_1, & y(x) \geq \text{threshold} \\ C_2, & y(x) < \text{threshold} \end{cases},$
  with $\text{threshold} = 0$ or $\text{threshold} = 0.5$.

Classification problems

- $predict(x) = \begin{cases} C_1, & y(x) \geq \text{threshold} \\ C_2, & y(x) < \text{threshold} \end{cases},$
  with $\text{threshold} = 0$ or $\text{threshold} = 0.5$.
- Three ways to address the classification problem:

Classification problems

- $predict(x) = \begin{cases} C_1, & y(x) \geq \text{threshold} \\ C_2, & y(x) < \text{threshold} \end{cases}$, with $\text{threshold} = 0$ or $\text{threshold} = 0.5$.
- Three ways to address the classification problem:
  1. Directly model the discrimination function: e.g $y(x) = w^T x + w_0$

Classification problems

- $predict(x) = \begin{cases} C_1, & y(x) \geq \text{threshold} \\ C_2, & y(x) < \text{threshold} \end{cases}$,
  with $\text{threshold} = 0$ or $\text{threshold} = 0.5$.
- Three ways to address the classification problem:
  1. Directly model the discrimination function: e.g $y(x) = w^T x + w_0$
  2. Generative model:

$$y(x) = P(C_k|x) = \frac{P(x|C_k)P(C_k)}{P(x)}$$

Classification problems

- $predict(x) = \begin{cases} C_1, & y(x) \geq \text{threshold} \\ C_2, & y(x) < \text{threshold} \end{cases}$,
  with $\text{threshold} = 0$ or $\text{threshold} = 0.5$.
- Three ways to address the classification problem:
  1. Directly model the discrimination function: e.g $y(x) = w^T x + w_0$
  2. Generative model:
  $$y(x) = P(C_k|x) = \frac{P(x|C_k)P(C_k)}{P(x)}$$
  3. Discriminative model:
  $$y(x) = P(C_k|x) = f(x),$$
  with f an arbitrary function.

Linear Models

$$y(x) = f(w^T x + w_0)$$

- $f(\cdot)$: activation function, may be non-linear

Linear Models

$$y(x) = f(w^T x + w_0)$$

- $f(\cdot)$: activation function, may be non-linear
- Even if $f(\cdot)$ is non-linear, the decision boundary is linear

Linear Models

$$y(x) = f(w^T x + w_0)$$

- $f(\cdot)$: activation function, may be non-linear
- Even if $f(\cdot)$ is non-linear, the decision boundary is linear
- Also called *generalized linear models*

Linear Models

$$y(x) = f(w^T x + w_0)$$

- $f(\cdot)$: activation function, may be non-linear
- Even if $f(\cdot)$ is non-linear, the decision boundary is linear
- Also called *generalized linear models*
- Applicable if instead of x we use a vector of basis functions $\phi(x)$, corresponding to features in a feature space

Using regression for classification

- We can use a regression model, such as least squares to fit a linear classification model with a linear activation function

$$\min_{w,w_o} \sum_{i=1}^{\ell} (t_i - w^T x_i + w_0)^2,$$

where $t_i \in \{-1, 1\}$ is the label of the i-th training sample,

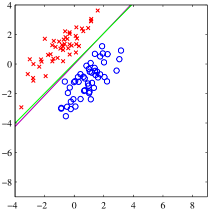
Using regression for classification

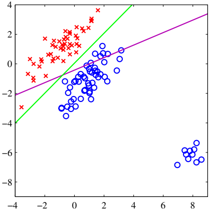
- We can use a regression model, such as least squares to fit a linear classification model with a linear activation function

$$\min_{w,w_o} \sum_{i=1}^{\ell} (t_i - w^T x_i + w_0)^2,$$

where $t_i \in \{-1, 1\}$ is the label of the i-th training sample,

- but this strategy does not work well:





Outline

Rosemblatt's perceptron

- Designed by Frank Rossemblat in 1957

Rosemblatt's perceptron

- Designed by Frank Rossemblat in 1957
- A hardware implementation of the learning algorithm

Rosemblatt's perceptron

- Designed by Frank Rossemblat in 1957
- A hardware implementation of the learning algorithm
- The precursor of neural networks

Rosemblatt's perceptron

- Designed by Frank Rossemblat in 1957
- A hardware implementation of the learning algorithm
- The precursor of neural networks
- Criticized by Marvin Minsky, producing a decline in research funding

Perceptron learning

- Activation function:

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

Perceptron learning

- Activation function:

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

- Loss function:

$$E_p(w) = -\sum_{n=1}^{\ell} w^T \phi_n t_n,$$

where $\phi_n = \phi(x_n)$.

Perceptron learning

- Activation function:

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

- Loss function:

$$E_p(w) = -\sum_{n=1}^{\ell} w^T \phi_n t_n,$$

where $\phi_n = \phi(x_n)$.

- Learning rule:

$$w^{(\tau+1)} = w^{(\tau)} + \eta \phi_n t_n$$

Perceptron convergence

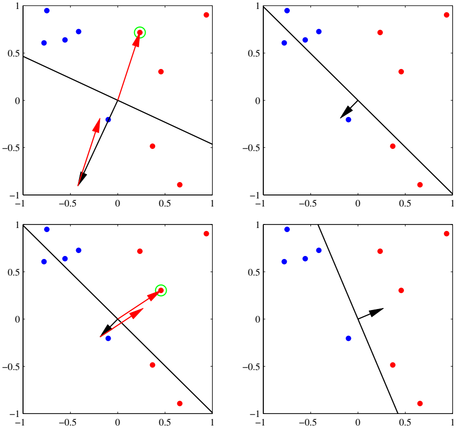
- If the training points are linearly separable, the perceptron algorithms converges (*perceptron convergence theorem*)

Perceptron convergence

- If the training points are linearly separable, the perceptron algorithms converges (*perceptron convergence theorem*)
- It could converge to different solutions depending on the order of presentation of training sample

Perceptron convergence

- If the training points are linearly separable, the perceptron algorithms converges (*perceptron convergence theorem*)
- It could converge to different solutions depending on the order of presentation of training sample



-

Perceptron problems

- Non-probabilistic outputs

Perceptron problems

- Non-probabilistic outputs
- Non-convex optimization problem

Perceptron problems

- Non-probabilistic outputs
- Non-convex optimization problem
- No convergence guarantee if samples are not linearly separable

Perceptron problems

- Non-probabilistic outputs
- Non-convex optimization problem
- No convergence guarantee if samples are not linearly separable
- Its power may be increased by stacking several perceptron layers (multilayer perceptrons) and using smooth activation functions

Outline

Parametric discrimination

- These three conditions are equivalent:
  - $P(C_1|x) \geq 0.5$
  - $\frac{P(C_1|x)}{1-P(C_1|x)} \geq 1$
  - $\text{logit}(P(C_1|x)) = \log \frac{P(C_1|x)}{1-P(C_1|x)} \geq 0$

Parametric discrimination

- These three conditions are equivalent:
  - $P(C_1|x) \geq 0.5$
  - $\frac{P(C_1|x)}{1-P(C_1|x)} \geq 1$
  - $\text{logit}(P(C_1|x)) = \log \frac{P(C_1|x)}{1-P(C_1|x)} \geq 0$
- If we assume that $P(x|C_1)$ and $P(x|C_2)$ are normally distributed sharing the same covariance matrix:

$$\text{logit}(P(C_1|x)) = \log \frac{P(C_1|x)}{P(C_2|x)} = w^T x + w_0,$$

where

$$w = \Sigma^{-1}(\mu_1 + \mu_2)$$
$$w_0 = -\frac{1}{2}(\mu_1 + \mu_2)^T \Sigma^{-1}(\mu_1 + \mu_2) + \log \frac{P(C_1)}{P(C_2)}$$

Logistic function

- The logit function:

$$\text{logit}(P(C_1|x)) = \log \frac{P(C_1|x)}{1 - P(C_1|x)} = w^T x + w_0$$

Logistic function

- The logit function:

$$\mathrm{logit}(P(C_1|x)) = \log \frac{P(C_1|x)}{1 - P(C_1|x)} = w^T x + w_0$$

- The inverse-logit:

$$P(C_1|x) = \sigma(w^T x + w_0) = \frac{1}{1 + e^{-(w^T x + w_0)}}$$

Logistic function

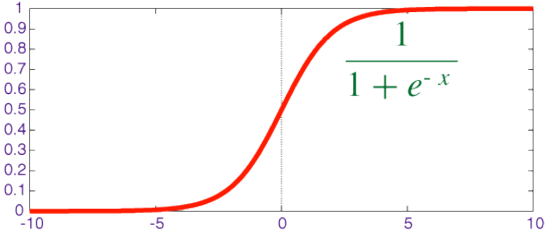
- The logit function:

$$\mathrm{logit}(P(C_1|x)) = \log \frac{P(C_1|x)}{1 - P(C_1|x)} = w^T x + w_0$$

- The inverse-logit:

$$P(C_1|x) = \sigma(w^T x + w_0) = \frac{1}{1 + e^{-(w^T x + w_0)}}$$

- σ is called the logistic or sigmoid function.



Logistic regression

- $$y(x) = P(C_1|x) = \sigma(w^T x)$$

Logistic regression

- $$y(x) = P(C_1|x) = \sigma(w^T x)$$
- Find w using maximum likelihood estimation:

$$p(\mathbf{t}|w) = \prod_{n=1}^{\ell} y_n^{t_n}(1 - y_n)^{1-t_n},$$

where $\mathbf{t} = \{t_1, \ldots, t_\ell\}$ and $y_n = y(x_n)$.

Logistic regression

- $$y(x) = P(C_1|x) = \sigma(w^T x)$$
- Find w using maximum likelihood estimation:

$$p(\mathbf{t}|w) = \prod_{n=1}^{\ell} y_n^{t_n}(1 - y_n)^{1-t_n},$$

where $\mathbf{t} = \{t_1, \ldots, t_\ell\}$ and $y_n = y(x_n)$.

- Cross-entropy error:

$$E(w) = -\ln p(\mathbf{t}|w) = -\sum_{n=1}^{\ell} [t_n \ln y_n + (1 - t_n)\ln(1 - y_n)]$$

Multiclass logistic regression

- $$y_k(x) = P(C_k|x) = \frac{e^{w_k^T x}}{\sum_j e^{w_j^T x}}$$

Multiclass logistic regression

- $$y_k(x) = P(C_k|x) = \frac{e^{w_k^T x}}{\sum_j e^{w_j^T x}}$$

- Likelihood:

$$p(\mathbf{T}|w_1 \dots w_K) = \prod_{n=1}^{\ell} \prod_{k=1}^{K} y_{nk}^{t_{nk}},$$

where $y_{nk} = y_k(x_n)$ and $\mathbf{T} \in \mathbb{R}^{\ell \times K}$ is a matrix of target variables with elements t_{nk}.

Multiclass logistic regression

- $$y_k(x) = P(C_k|x) = \frac{e^{w_k^T x}}{\sum_j e^{w_j^T x}}$$
- Likelihood:

$$p(\mathbf{T}|w_1 \ldots w_K) = \prod_{n=1}^{\ell} \prod_{k=1}^{K} y_{nk}^{t_{nk}},$$

where $y_{nk} = y_k(x_n)$ and $\mathbf{T} \in \mathbb{R}^{\ell \times K}$ is a matrix of target variables with elements t_{nk}.

- Multiclass cross-entropy error:

$$E(w_1, \ldots, w_K) = -\ln p(\mathbf{T}|w_1 \ldots w_K) = -\sum_{n=1}^{\ell} \sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

Outline

Optimization problem

- $$\min_{w} E(w) = \min_{w} - \sum_{n=1}^{\ell} [t_n \ln y_n + (1 - t_n) \ln(1 - y_n)]$$

Optimization problem

- $$\min_{w} E(w) = \min_{w} -\sum_{n=1}^{\ell} [t_n \ln y_n + (1 - t_n) \ln(1 - y_n)]$$

- $$\nabla E(w) = \sum_{n=1}^{\ell} (y_n - t_n) \phi_n$$

Optimization problem

- $$\min_w E(w) = \min_w - \sum_{n=1}^{\ell} [t_n \ln y_n + (1 - t_n) \ln(1 - y_n)]$$
- $$\nabla E(w) = \sum_{n=1}^{\ell} (y_n - t_n)\phi_n$$
- $$w^{(\tau+1)} = w^{(\tau)} - \eta \sum_{n=1}^{\ell} (y_n - t_n)\phi_n$$

Newton-Raphson

- $$w^{(\tau+1)} = w^{(\tau)} - \mathrm{H}^{-1}\nabla E(w)$$

Newton-Raphson

- $$w^{(\tau+1)} = w^{(\tau)} - \mathrm{H}^{-1} \nabla E(w)$$
- $$\nabla E(w) = \Phi^T (\mathbf{y} - \mathbf{t})$$

Newton-Raphson

- $$w^{(\tau+1)} = w^{(\tau)} - \mathrm{H}^{-1}\nabla E(w)$$
- $$\nabla E(w) = \Phi^T(\mathbf{y} - \mathbf{t})$$
- $$\mathbf{H} = \nabla\nabla E(w) = \Phi^T\mathbf{R}\Phi,$$

with $\mathbf{R}$ a diagonal matrix with $R_{nn} = y_n(1 - y_n)$.

Newton-Raphson

- $$w^{(\tau+1)} = w^{(\tau)} - \mathrm{H}^{-1}\nabla E(w)$$
- $$\nabla E(w) = \Phi^T(\mathbf{y} - \mathbf{t})$$
- $$\mathbf{H} = \nabla\nabla E(w) = \Phi^T \mathbf{R} \Phi,$$
  with $\mathbf{R}$ a diagonal matrix with $R_{nn} = y_n(1 - y_n)$.
- The resulting algorithm is called *iterative reweighted least squares.*

Regularization

- $$\min_{w} -C \sum_{n=1}^{\ell} \left[t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right] + \|w\|^2$$

Regularization

- $$\min_{w} -C \sum_{n=1}^{\ell} [t_n \ln y_n + (1 - t_n) \ln(1 - y_n)] + \|w\|^2$$
- Prevents overfitting.

Regularization

- $$\min_{w} -C \sum_{n=1}^{\ell} [t_n \ln y_n + (1 - t_n) \ln(1 - y_n)] + \|w\|^2$$
- Prevents overfitting.
- Equivalent to the inclusion of a prior and finding a MAP solution for W.

Stochastic gradient descent

- $$\min_w Q(w) = \min_w \sum_{i=1}^{n} Q_i(w)$$

Stochastic gradient descent

- $$\min_w Q(w) = \min_w \sum_{i=1}^{n} Q_i(w)$$
- Batch gradient descent:
$$w^{(\tau+1)} = w^{(\tau)} - \alpha \nabla Q(w) = w^{(\tau)} - \alpha \sum_{i=1}^{n} \nabla Q_i(w)$$

Stochastic gradient descent

- $$\min_w Q(w) = \min_w \sum_{i=1}^{n} Q_i(w)$$
- Batch gradient descent:
$$w^{(\tau+1)} = w^{(\tau)} - \alpha \nabla Q(w) = w^{(\tau)} - \alpha \sum_{i=1}^{n} \nabla Q_i(w)$$
- on-line gradient descent:
$$w^{(\tau+1)} = w^{(\tau)} - \alpha \nabla Q_i(w)$$

- Alpaydin, E. 2010 Introduction to Machine Learning, 2Ed. The MIT Press. (Chap 10)
- Russell, S and Norvig, P. 2010 Artificial Intelligence: a Modern Approach, 3rd Ed, Prentice-Hall (Sect 18.6)