Machine Learning

Daniel Mao

Contents

1

Perceptron

1.1 The Perceptron Algorithm

Algorithm 1: The Perceptron Algorithm (Rosenblatt 1958)

Input: Dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1..n\}$, initialization $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, threshold $\delta \geq 0$.

Output: Approximate solutions $\boldsymbol{x}$ and b.

```
while true do
    receive training example index i ∈ {1..n};
    if y_i(w · x_i + b) ≤ δ then
        w ← w + y_i x_i;
        b ← b + y_i;
```

Theorem 1. *Assume there exists some $\boldsymbol{z}$ such that $A^\top \boldsymbol{z} > 0$, then the perceptron algorithm converges to some $\boldsymbol{z}^*$. If each column of A is selected indefinitely often, then $A^\top \boldsymbol{z}^* > \delta$.*

Corollary. *Let $\delta = 0$ and $\boldsymbol{z}_0 = \boldsymbol{0}$. Then the perceptron algorithm converges after at most $(R/\gamma)^2$ steps, where R and γ are dataset properties given by*

$$R := \|A\|_{2,\infty} = \max_i \|\boldsymbol{a}_i\|_2 \quad \text{and} \quad \gamma := \max_{\|z\|_2 \leq 1} \min_i \langle \boldsymbol{z}, \boldsymbol{a}_i \rangle.$$

Note that the parameters R and γ are independent of the dataset size n.

Theorem 2. *The iterate $\boldsymbol{z} = (\boldsymbol{w}; b)$ of the perceptron algorithm is always bounded. In particular, if there is no separating hyperplane, then perceptron cycles.*

2

Logistic Regression

2.1 Hypothesis Representation

Logistic Regression Model
Want $0 \leq h_\theta(x) \leq 1$.

$$h_\theta(x) = g(\theta^\top x)$$

where

$$g(z) = \frac{1}{1+e^{-z}}.$$

So

$$h_\theta(x) = \frac{1}{1+e^{-\theta^\top x}}.$$

Sigmoid function (logistic function)

$$g(z) = \frac{1}{1+e^{-z}}.$$

Properties:

- $g(0) = 0.5$.
- $\lim_{z\to+\infty} g(z) = 1$.
- $\lim_{z\to-\infty} g(z) = 0$.

Interpretation of the Hypothesis Output
$h_\theta(x)$ is the probability that $y = 1$ on input x. i.e. $h_\theta(x) = P(y = 1 \mid x; \theta)$, the probability that $y = 1$, given x, parameterizad by θ.

Since y is either 0 or 1,

$$P(y = 0 \mid x; \theta) + P(y = 1 \mid x; \theta) = 1, \text{ or}$$
$$P(y = 0 \mid x; \theta) = 1 - P(y = 1 \mid x; \theta).$$

2.2 Decision Boundary

Predict $y = 1$ if $h_\theta(x) \geq 0.5$ and predict $y = 0$ if $h_\theta(x) < 0.5$.

$$h_\theta(x) \geq 0.5 \iff \theta^\top x \geq 0.$$

Consider $\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$. Predict "$y = 1$" if $\theta^\top x = -3 + x_1 + x_2 \geq 0$. That is, $x_2 \geq 3 - x_1$. This gives a "half space" in $\mathbb{R}^2$. The line $x_1 + x_2 = 3$ separates $\mathbb{R}^2$ into a region where the model predicts "$y = 0$" and a region where the model predicts "$y = 1$".

Non-linear Decision Boundary

Consider

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

and $\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$. Then the model predicts "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$. That is, if $x_1^2 + x_2^2 \geq 1$.

2.3 Cost Function

Training set: $\{(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})\}$. $x_0 = 1$. $x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$. $y \in \{0, 1\}$.

If

$$cost(h_\theta(x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

as in linear regression, then the cost function is non-convex.

Logistic Regression Cost Function

$$cost(h_\theta(x), y) := \begin{cases} -\log(h_\theta(x)), & \text{if } y = 1 \\ -\log(1 - h_\theta(x)), & \text{if } y = 0. \end{cases}$$

Property:

- If $y = 1$ and $h_\theta(x) \uparrow 1$, then $cost(h_\theta(x), y) \approx 0$.
- If $y = 1$ and $h_\theta(x) \downarrow 0$, then $cost(h_\theta(x), y) \to +\infty$.
- Similar for $y = 0$.

Captures intuition that if $h_\theta(x) = 0$ (the model predicts $P(y = 1 \mid x; \theta) = 0$), but $y = 1$, we'll penalize the learning algorithm by a very large cost.

Cost Function for the Training Set

$$J(\theta) := \frac{1}{m} \sum_{i=1}^{m} cost(h_\theta(x^{(i)}, y^{(i)}))$$

where *cost* is the function given above.
Rewrite the cost function:

$$cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x)).$$

So

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \left(h_\theta(x^{(i)}) \right) + (1 - y^{(i)}) \log \left(1 - h_\theta(x^{(i)}) \right) \right].$$

To fit parameter θ: $\min_\theta J(\theta)$.

2.4 Optimization

Gradient Descent

$$\frac{\partial}{\partial \theta_j} J(\theta) = \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}.$$

This looks identical to linear regression.

Other Optimization Algorithms

- Conjugate Gradient
- BFGS
- L-BFGS

Advantages:

- No need to pick learning rate α.
- Often faster than gradient descent.

2.5 Multiclass Classification

One-vs-rest

$$h_\theta^{(i)}(x) = P(y = i \mid x; \theta) \quad i = 1, 2, 3.$$

Train a logistic regression classifier for each i to predict the probability that $y = i$. On a new input x, pick class $i = \underset{i}{\text{argmax}} h_\theta^{(i)}(x)$.

3

Dimensionality Reduction

3.1 Motivations

- Data compression.
- Data visualization.

3.2 Principal Component Analysis

PCA is trying to find a lower dimensional surface onto which to project the data so as to minimize squared projection error.
Principal component analysis vs. linear regression:

- Linear regression tries to minimize residuals. PCA tries to minimize projection error.
- In linear regression, we try to predict one feature with other features. In PCA, all features are equivalent.

Algorithm: (from n dimensions to k dimensions)

- Preprocessing: mean normalization and feature scaling (standardization).
- Compute covariance matrix

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T.$$

- Compute the eigenvalues of Σ.

$$\Sigma = USV.$$

  Let U_{reduce} denote the first k columns of U. Then $z = U_{reduce}^T x$.

Choosing k:

- Average squared projection error:

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2.$$

- Total variation in the data:

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2.$$

- Typically, we choose k to be the smallest value so that

$$\frac{\text{average squared projection error}}{\text{total variation}} < 0.01,$$

  i.e. 99% of variance is retained.

- The above ratio can be computed in an easier way. Say

$$S = \begin{bmatrix} s_{11} & & O \\ & \ddots & \\ O & & s_{nn} \end{bmatrix}.$$

  Then

$$\frac{\text{average squared projection error}}{\text{total variation}} = 1 - \frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}}.$$

  So we can start with $k = 1$ and increase k and check whether

$$\frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}} > 0.99.$$