Logistic regression

cs434

Logistic regression

- Recall the problem of regression
  - Learns a linear mapping from input vector $\mathbf{x}$ to a continuous output $y \in (-\infty, \infty)$
- Logistic regression can be viewed as extending linear regression to handle binary output y by warping the output of a linear function to the range between 0 and 1
- For convenience, we will assume $y \in \{0,1\}$ for logistic regression

Logistic regression (cont.)

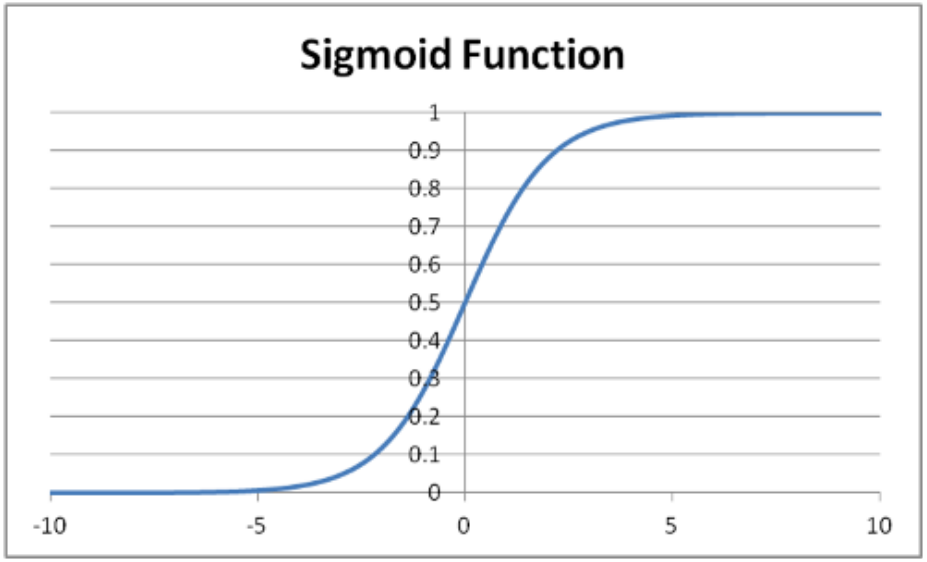
- Consider the linear regression function

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} = w_0 + w_1 x_1 + \cdots + w_m x_m$$

- We introduce a function σ:

$$\sigma(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T\boldsymbol{x})}$$

Referred to as the sigmoid function or logistic function



Logistic Regression Makes Probabilistic Prediction

- We interpret the output of logistic regression as a probability:

$$P(y = 1|\boldsymbol{x}; \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})}$$

i.e., probability of $y = 1$ given the input $\boldsymbol{x}$ and the model's parameter is $\boldsymbol{w}$

$$P(y = 0|\boldsymbol{x}; \boldsymbol{w}) = 1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}) = \frac{\exp(-\boldsymbol{w}^T \boldsymbol{x})}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})}$$

Logistic Regression learns a **linear** decision boundary

- We predict $y = 1$ if

$$P(y = 1|\boldsymbol{x}; \boldsymbol{w}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})} > P(y = 0|\boldsymbol{x}; \boldsymbol{w}) = \frac{\exp(-\boldsymbol{w}^T \boldsymbol{x})}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})}$$

$$\Rightarrow \frac{P(y = 1|\boldsymbol{x})}{P(y = 0|\boldsymbol{x})} = \exp(\boldsymbol{w}^T \boldsymbol{x}) > 1$$

$$\Rightarrow \log \frac{P(y = 1|\boldsymbol{x})}{P(y = 0|\boldsymbol{x})} = \boldsymbol{w}^T \boldsymbol{x} > 0$$

- Decision boundary:

$$\boldsymbol{w}^T \boldsymbol{x} = 0$$

Learning **w** for logistic regression

- Given a set of training data points $(\boldsymbol{x_i}, y_i)$, $i = 1, \ldots, n$, where $y_i \in \{0,1\}$
- The goal is to find a weight vector $\mathbf{w}$ s.t. the probability of the correct y_i for $\boldsymbol{x}_i$ is high for $i = 1, .. n$

$$\text{maximize } P(y = y_i | \boldsymbol{x}_i; \boldsymbol{w})$$

- Or equivalently

$$\text{minimize } -\sum_{i=1}^{n} \log P(y = y_i | \boldsymbol{x}_i; \boldsymbol{w})$$

Negative log likelihood, or cross-entropy loss

Learning **w** for logistic regression

$$\text{minimize } -\sum_{i=1}^{n} \log P(y = y_i | \boldsymbol{x}_i; \boldsymbol{w})$$

$$P(y = y_i | \boldsymbol{x}_i; \boldsymbol{w}) = \begin{cases} \sigma(\boldsymbol{w}^T \boldsymbol{x}_i) & \text{if } y_i = 1 \\ 1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i) & \text{if } y_i = 0 \end{cases}$$

$$= \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)^{y_i} (1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i))^{(1-y_i)}$$

Our final objective:

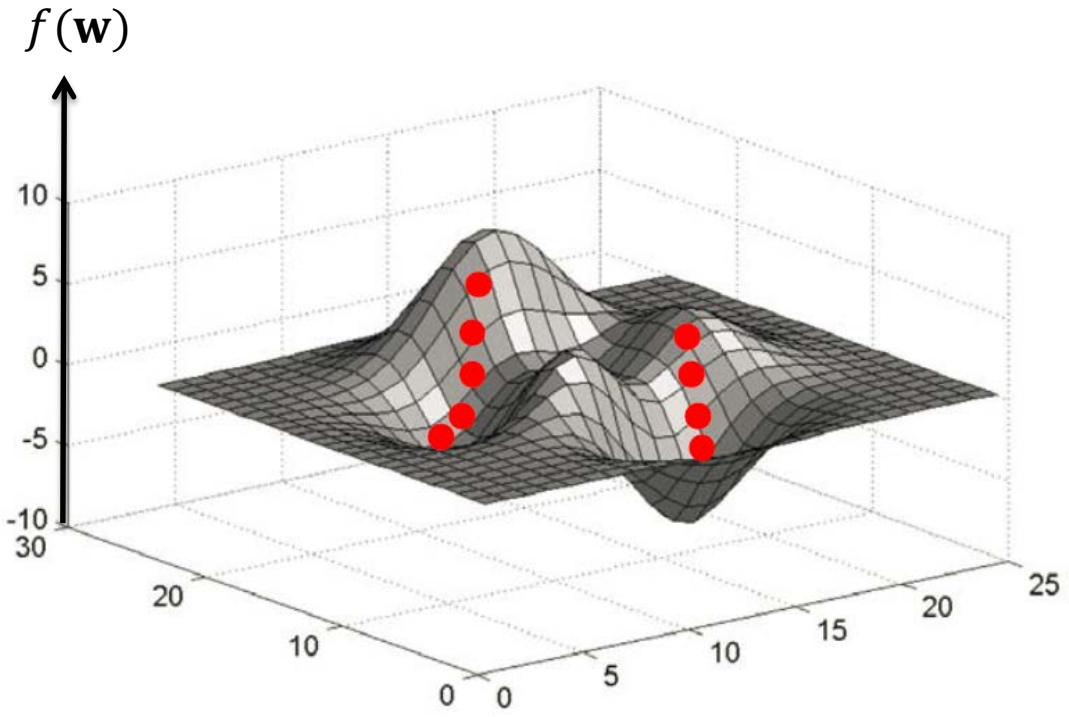
$$\text{minimize}$$

$$L(\boldsymbol{w}) = -\sum_{i=1}^{n} \left(y_i \log \sigma(\boldsymbol{w}^T \boldsymbol{x}_i) + (1 - y_i) \log \left(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i) \right) \right)$$

Minimizing $L(\mathbf{w})$

- Unfortunately this does not have a close form solution
  - You take the derivative, set it to zero, but no closed form solution
- Instead, we iteratively search for the optimal **w**
- Start with a random **w**, iteratively improve **w** (similar to Perceptron) by taking the negative gradient
- This is referred to as gradient descent

Gradient descent to minimize $L(\mathbf{w})$



1. Start from some initial guess $\mathbf{w}^0$

2. Find the direction of steepest descent – opposite of the gradient direction $\nabla f(\mathbf{w})$

3. Take a step toward that direction

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \nabla f(\mathbf{w}^t)$$

4. Repeat until no local improvement is possible

($|\nabla f(\mathbf{w}^t)| \leq \epsilon$)

Different starting points may lead to different local minimum if objective is not convex (like what's shown in the above figure)

Gradient of $L(\mathbf{w})$

Focus on the loss of a particular example $(\boldsymbol{x_i}, y_i)$:

$$l_i = -y_i \log \sigma\left(\boldsymbol{w^T x_i}\right) - (1 - y_i) \log\left(1 - \sigma\left(\boldsymbol{w^T x_i}\right)\right)$$

Useful fact: $\sigma'(t) = \sigma(1 - \sigma)$

$$\nabla l_i = \frac{-y_i}{\sigma(\boldsymbol{w^T x_i})} \nabla\, \sigma(\boldsymbol{w^T x_i}) + \frac{1 - y_i}{1 - \sigma(\boldsymbol{w^T x_i})} \nabla \sigma(\boldsymbol{w^T x_i})$$

$$\nabla\, \sigma\left(\boldsymbol{w^T x_i}\right) = \sigma\left(\boldsymbol{w^T x_i}\right)\left(1 - \sigma\left(\boldsymbol{w^T x_i}\right)\right)\boldsymbol{x_i}$$

$$\nabla l_i = (\sigma(\boldsymbol{w}^T \boldsymbol{x_i}) - y_i)\boldsymbol{x_i}$$

Online gradient descent for Logistic Regression

Recall: y takes 0/1 here, not 1/-1

Given: training examples $(\boldsymbol{x}_i, y_i), i = 1, \ldots n$

Let $\boldsymbol{w} \leftarrow (0, \ldots, 0)$ //initialization

Repeat until convergence //

for $i = 1, \ldots, n$

$$\hat{y}_i \leftarrow \frac{1}{1+e^{-\boldsymbol{w}^T \boldsymbol{x_i}}}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta(\hat{y}_i - y_i)\boldsymbol{x_i}$$

Note the striking similarity between LR and perceptron

Both learn a linear decision boundary

The iterative algorithm takes very similar form

Batch Learning for Logistic Regression

Note: y takes 0/1 here, not 1/-1

Given: training examples $(\boldsymbol{x}_i, y_i), i = 1, \ldots n$

Let $\boldsymbol{w} \leftarrow (0, \ldots, 0)$ //initialization

Repeat

$\nabla = (0, \ldots, 0)$

for $i = 1, \ldots, n$

$\hat{y}_i \leftarrow \frac{1}{1+e^{-\boldsymbol{w}^T \boldsymbol{x}_i}}$

$\nabla \leftarrow \nabla + (\hat{y}_i - y_i)\boldsymbol{x}_i$

$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\nabla$

Until $|\nabla| \leq \epsilon$

Logistic regression: Summary

- LR uses σ to warp the output of a linear function to $(0,1)$, which is interpreted as $P(y = 1|\boldsymbol{x}; \boldsymbol{w})$
- Learn a vector $\boldsymbol{w}$ s.t. for input $\boldsymbol{x}_i$ the desired correct output y_i has high probability, and other incorrect outputs have low probabilities
  - Maximize the log likelihood or
  - Minimize the negative log likelihood
- Learn $\boldsymbol{w}$ iteratively using gradient descent
- Strong similarity with Perceptron
- Logistic regression learns a linear decision boundaries
  - By introducing nonlinear features (i.e., $x_1^2, x_2^2, x_1 x_2, \ldots$), can be extended to nonlinear boundary.