CS456: Machine Learning

Logistic Regression

Jakramate Bootkrajang

Department of Computer Science
Chiang Mai University

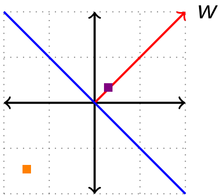
Objectives

- To understand how Logistic Regression works
- To understand what likelihood and log-likelihood is
- To understand basic concept of gradient descent optimisation method

Outlines

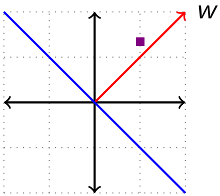
- LR's philosophy
- Sigmoid function
- Likelihood, log-likelihood and negative log-likelihood
- Gradient descent
- LR in sklearn

Decision boundary and prediction



- Recall that we are more certain about the prediction of point that is far away from the boundary (orange point).
- while we are less sure about points lie closer to the decision boundary (violet point)

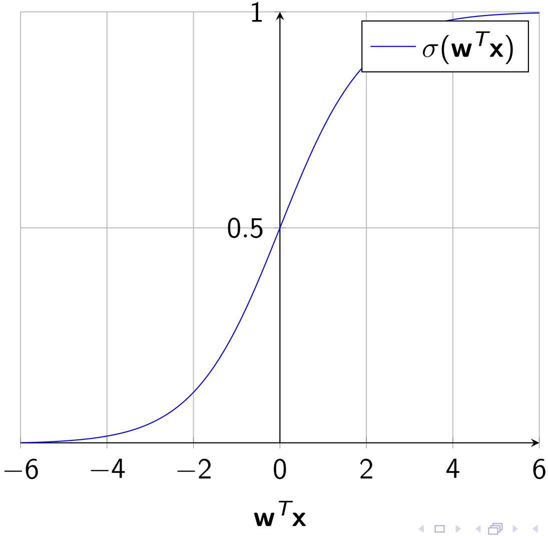
Probability supporting prediction



- Unlike SVM, Logistic Regression converts the distance into probability which increases as the distance from decision boundary increases
- This is done by using the sigmoid function $\frac{1}{1+e^{-t}}$ where $e \approx 2.71828\ldots$

Sigmoid function

The function, $\frac{1}{1+e^{-t}}$, maps real-valued input t to $[0, 1]$ range



Probability for the positive class

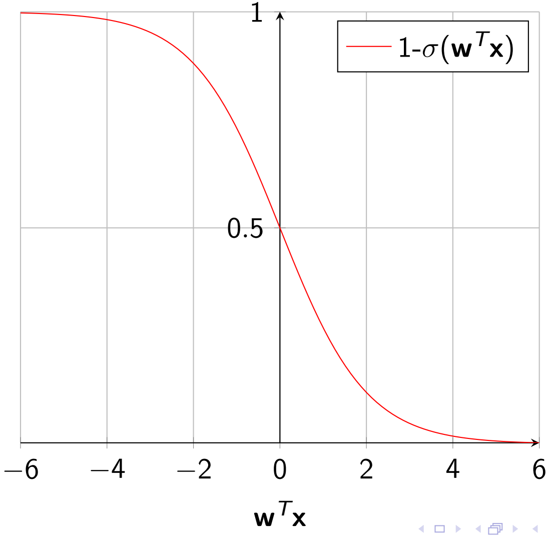
- LR uses $\sigma(\mathbf{w}^T\mathbf{x})$ to represent the probability that $\mathbf{x}$ comes from positive class $(y = 1)$
- The probability is denoted as $p(y = 1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$
- The probability is 0.5 when $\mathbf{x}$ is on the decision boundary

What about the negative class ?

- Negative examples are the ones with $\mathbf{w}^T\mathbf{x} < 0$
- We want a function which is the opposite of the sigmoid i.e., its value increases as $\mathbf{w}^T\mathbf{x}$ decreases

Function for negative class

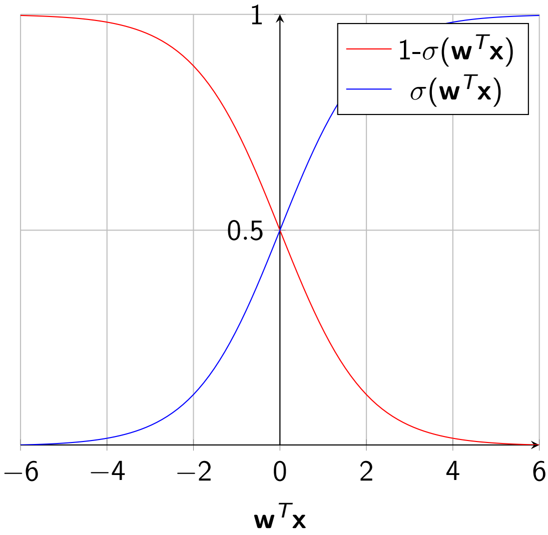
The function, $\frac{e^{-t}}{1+e^{-t}}$, maps real-valued input t to $[0, 1]$



Probability for the negative class

- LR uses $1 - \sigma(\mathbf{w}^T\mathbf{x})$ to represent the probability that $\mathbf{x}$ comes from negative class
- The probability is denoted as $p(y = -1|\mathbf{x}) = \frac{e^{-\mathbf{w}^T\mathbf{x}}}{1+e^{-\mathbf{w}^T\mathbf{x}}} = \frac{1}{1+e^{\mathbf{w}^T\mathbf{x}}}$
- The probability is 0.5 when $\mathbf{x}$ is on the decision boundary

Putting them together



LR's philosophy

- Find a parameter **w** which
  - maximise $\sigma(\mathbf{w}^T\mathbf{x})$ for inputs from positive class
  - maximising $1 - \sigma(\mathbf{w}^T\mathbf{x})$ for inputs from negative class
- Suppose there are N_- data points from negative class and N_+ data points from positive class, the probability for all data points is

$$\prod_{i=1}^{N_-}(1 - \sigma(\mathbf{w}^T\mathbf{x}_i))\prod_{j=1}^{N_+}\sigma(\mathbf{w}^T\mathbf{x}_j) \tag{1}$$

  - This is valid under assumption that data points are independent from each other

Likelihood function

$$\prod_{i=1}^{N_-}(1-\sigma(\mathbf{w}^T\mathbf{x}_i))\prod_{j=1}^{N_+}\sigma(\mathbf{w}^T\mathbf{x}_j)$$

- Since $y_i \in \{-1, 1\}$, the above expression can be written compactly as

$$\prod_{i=1}^{N}\sigma(y_i\mathbf{w}^T\mathbf{x}_j) := \mathcal{L}(\mathbf{w}) \tag{2}$$

- We call the above equation: the likelihood function

Log-likelihood function

- If we take logarithm of the likelihood function we end up with the log-likelihood function

$$llh(\mathbf{w}) = \sum_{i=1}^{N} \log \sigma(y_i \mathbf{w}^T \mathbf{x}_j) \tag{3}$$

- The $llh()$ is a function of parameter $\mathbf{w}$

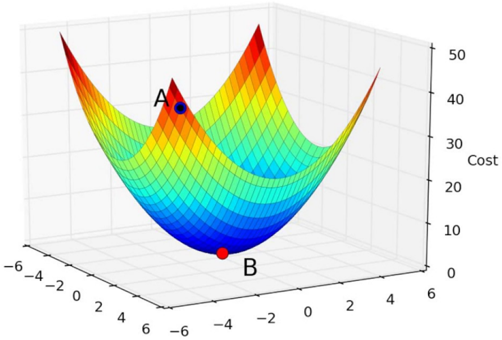
- There's also its negative version, negative log-likelihood

$$nllh(\mathbf{w}) = -\sum_{i=1}^{N} \log \sigma(y_i \mathbf{w}^T \mathbf{x}_j) \tag{4}$$

Model learning

- Learning of the model is to find the best $\mathbf{w}$ which maximises llh
- Or equivalently, minimising the negative log-likelihood function
- Does minimising some objective function sound familiar ?

The solution landscape



Where the best $\mathbf{w}$ is?

- It must be at the stationary point (gradient $= 0$)
- Let's find the gradient of the negative log-likelihood with respect to $\mathbf{w}$ (on the board)

Gradient of nllh

$$\nabla_{\mathbf{w}} nllh = -\sum_{i=1}^{N} (1 - \sigma(y_i \mathbf{w}^T \mathbf{x}_i)) y_i \mathbf{x}_i$$

Equating the above to zero yields...

$$-\sum_{i=1}^{N} (1 - \sigma(y_i \mathbf{w}^T \mathbf{x}_i)) y_i \mathbf{x}_i = 0 \tag{5}$$

Unfortunately, we cannot isolate **w** to get a closed-form solution

Gradient descent

Need to use iterative method e.g., gradient descent of the form

$$\mathbf{w}^{new} = \mathbf{w}^{old} - \eta \nabla_{\mathbf{w}} nllh \tag{6}$$

Gradient of a function always points in direction which increases function value, so we update **w** to the opposite direction

$$\nabla_{\mathbf{w}} nllh = -\sum_{i=1}^{N} (1 - \sigma(y_i \mathbf{w}^T \mathbf{x}_i)) y_i \mathbf{x}_i \tag{7}$$

Here η is the learning rate

Examples of Gradient

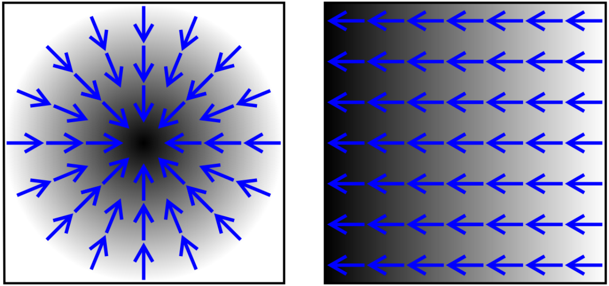


Figure: The direction of gradients are always towards higher function values (darker regions)

LR learning algorithm

1. Input data in the form $\{\mathbf{x}_i, y_i\}, i = 1 : N$ and $y_i \in \{-1, 1\}$
2. Initialise random $1 \times m$ weight vector $\mathbf{w}$, (m is data dimension)
3. Repeat until convergence (no change in $\mathbf{w}$)

$$\mathbf{w}^{new} = \mathbf{w}^{old} - \eta \nabla_{\mathbf{w}} nllh$$

4. To classify new data $\mathbf{x}_q$, decide $y_q = 1$

   if $p(y_q = 1 | \mathbf{x}_q) = \frac{1}{1+e^{-\mathbf{w}^T \mathbf{x}_q}} > 0.5$

LR in sklearn

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
>>> clf.score(X, y)
0.97...
```

Sklearn uses $y = \{0, 1\}$ notation.

Objectives: revisited

- To understand how Logistic Regression works
- To understand what likelihood and log-likelihood is
- To understand basic concept of gradient descent optimisation method

Lastly

Questions please ..