Recap: Summary of Last Lecture

- **Decision Tree classifier**
  - Each non-leaf node makes a decision about a feature and the classification decision is made at the leaf nodes
  - We prefer shorter trees (Occam's Razor)
  - Greedy tree construction algorithm via information gain principle
  - Flexibility comes at the cost of over-fitting
  - Pruning the tree to avoid over-fitting

Lecture #6: Logistic Regression

Probabilistic Classifier

- Given an instance $\mathbf{x}$, what does a probabilistic classifier do differently compared to, say, perceptron?
- It does not directly predict y
- Instead, it first computes the probability that the instance belongs to different classes, i.e., $P(y|\mathbf{x})$ – the posterior probability of y given $\mathbf{x}$
- Given $p(y|\mathbf{x})$, it then makes a prediction using decision theory

Decision Theory

- **Goal 1:** Minimizing the probability of mistake
  - $y^* = \arg\max_y P(y|\mathbf{x})$
  - i.e., predict the class that has the maximum posterior probability
- **Goal 2:** minimizing the expected loss
  - Given a cost matrix specifying the cost of different types of mistakes

True label→ Predicted ↓	Spam	Non-spam
Spam	0	10
Non-spam	1	0

$$y^* = \arg\min_y \sum_{y'} \underline{L(y, y')P(y'|\mathbf{x})}$$

Expected cost if we predict y

A Simple Example

- Suppose our probabilistic spam-filter gives the following posterior for an incoming email $\mathbf{x}$:
  - $P(y = spam|\mathbf{x}) = 0.6$

True label→ Predicted ↓	Spam	Non-spam
Spam	0	10
Non-spam	1	0

- The expected cost if predict spam?
  - If it is a spam: no cost (0.6 prob)
  - If it is not – cost of 10 (0.4 prob)
  - $0.6 \times 0 + 0.4 \times 10 = 4$
- What if we predict non-spam?
  - $0.6 \times 1 + 0.4 \times 0 = 0.6$

$$y^* = \arg\min_y \sum_{y' \in \{s,ns\}} L(y, y') P(y'|\mathbf{x}) = \text{non-spam}$$

Two Main Approaches

- To learn a probabilistic classifier, there are two types of approaches
- **Generative:**
  - Learn $P(y)$ and $P(\mathbf{x}|y)$
  - Compute $P(y|\mathbf{x})$ using Bayes rule

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} = \frac{P(\mathbf{x}|y)P(y)}{\sum_y P(\mathbf{x}, y)}$$

- **Discriminative:**
  - Learn $P(y|\mathbf{x})$ directly
  - Logistic regression is one of such techniques

Logistic Regression

- Given training set D, logistic regression directly learns the conditional distribution $P(y \mid \boldsymbol{x})$
- We will assume only two classes $y \in \{0,1\}$ and a parametric form for $P(y = 1 \mid \mathbf{x}; \mathbf{w})$, where $\mathbf{w}$ is the weight (parameter) vector

$$P(y=1 \mid \mathbf{x};\mathbf{w}) = p_1(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

$$P(y=0 \mid \mathbf{x};\mathbf{w}) = 1 - p_1(\mathbf{x})$$

Logistic Regression

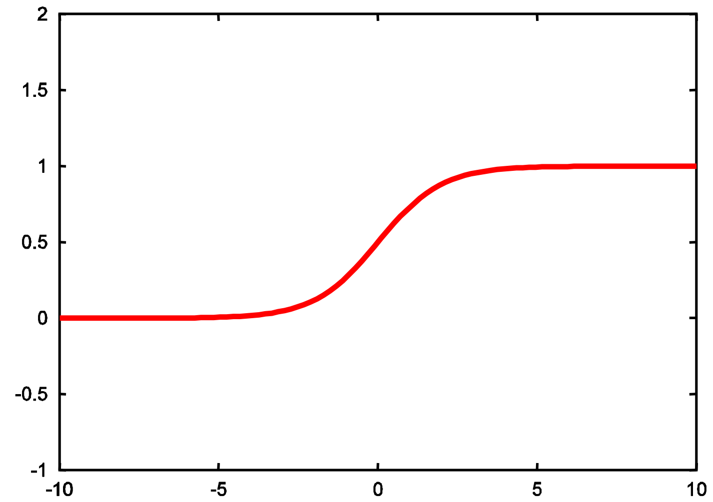
- It is easy to show that this is equivalent to

$$\log \frac{P(y=1\,|\,\mathbf{x};\mathbf{w})}{P(y=0\,|\,\mathbf{x};\mathbf{w})} = \mathbf{w}^T\mathbf{x}$$

- i.e. the ***log odds*** of class 1 is a linear function of $\mathbf{x}$

The Logistic (Sigmoid) Function

$$g(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$



- The output of a linear function $\mathbf{w}^T\mathbf{x}$ has range $(-\infty, \infty)$. A logistic sigmoid function transforms the value of $\mathbf{w}^T\mathbf{x}$ into a range between 0 and 1

Logistic Regression Yields a Linear Classifier

- Given $P(y \mid \mathbf{x})$, suppose we use the decision rule for minimizing classification error: i.e., predict $y^* = 1$ if $P(y = 1 \mid \mathbf{x}) > P(y = 0 \mid \mathbf{x})$
  - More generally, $P(y = 1 \mid \mathbf{x}) > \theta$, where θ is a threshold
  - Depending on the loss function, θ can be different values

- This yields a linear classifier

$$P(y=1|\mathbf{x}) > P(y=0|\mathbf{x}) \Rightarrow \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} > 1$$

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

For more general decision rule, this will be replaced with a different threshold w_0

Learning Setup

- Given a set of training examples: $(\mathbf{x}^1, y^1), \cdots, (\mathbf{x}^N, y^N)$
- We assume that $\mathbf{x}$ and y are probabilistically related (parameterized by $\mathbf{w}$)

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

- **Goal:** learn $\mathbf{w}$ from the training data using Maximum Likelihood Estimation (MLE)

Likelihood Function

- We assume each training example $(\mathbf{x}^i, y^i)$ is drawn **IID** from the same (but unknown) distribution $P(\mathbf{x}, y)$:

$$\log P(D;\mathbf{w}) = \log \prod_i P(\mathbf{x}^i, y^i;\mathbf{w}) = \sum_i \log P(\mathbf{x}^i, y^i;\mathbf{w})$$

- Joint distribution $P(a,b)$ can be factored as $P(a \mid b)P(b)$

$$\arg\max_{\mathbf{w}} \log P(D;\mathbf{w}) = \arg\max_{\mathbf{w}} \sum_i \log P(\mathbf{x}^i, y^i;\mathbf{w})$$
$$= \arg\max_{\mathbf{w}} \sum_i \log P(y^i \mid \mathbf{x}^i;\mathbf{w})P(\mathbf{x}^i;\mathbf{w})$$

- Further, $P(\mathbf{x};\ \mathbf{w})$ can be dropped because it does not depend on **w**:

$$\arg\max_{\mathbf{w}} \log P(D;\mathbf{w}) = \arg\max_{\mathbf{w}} \sum_i \log P(y^i \mid \mathbf{x}^i;\mathbf{w})$$

Computing the Likelihood

$$\arg\max_{\mathbf{w}} \log P(D \mid \mathbf{w}) = \arg\max_{\mathbf{w}} \sum_i \log P(y^i \mid \mathbf{x}^i, \mathbf{w})$$

Let

$$\hat{y}^i = p(y^i = 1 \mid \mathbf{x}^i; \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}^i}}$$

$$p(y^i = 0 \mid \mathbf{x}^i; \mathbf{w}) = 1 - \hat{y}^i$$

This can be compactly written as

$$p(y^i \mid \mathbf{x}^i; \mathbf{w}) = (\hat{y}^i)^{y^i} (1 - \hat{y}^i)^{(1-y^i)}$$

We will take our learning objective function to be:

$$l(\mathbf{w}) = \sum_i \log P(y^i \mid \mathbf{x}^i, \mathbf{w})$$

$$= \sum_i [y^i \log \hat{y}^i + (1 - y^i) \log(1 - \hat{y}^i)]$$

Gradient Ascent

$$l(\mathbf{w}) = \sum_i \log P(y^i \mid \mathbf{x}^i; \mathbf{w}) = \sum_i [y^i \log \hat{y}^i + (1-y^i)\log(1-\hat{y}^i)]$$

$$\frac{\partial l_i(\mathbf{w})}{\partial w_j} = \frac{\partial}{\partial w_j}[y^i \log \hat{y}^i + (1-y^i)\log(1-\hat{y}^i)]$$

$$= \frac{y^i}{\hat{y}^i}\left(\frac{\partial \hat{y}^i}{\partial w_j}\right) + \frac{1-y^i}{1-\hat{y}^i}\left(-\frac{\partial \hat{y}^i}{\partial w_j}\right) = \left[\frac{y^i}{\hat{y}^i} - \frac{1-y^i}{1-\hat{y}^i}\right]\frac{\partial \hat{y}^i}{\partial w_j} = \left[\frac{y^i - \hat{y}^i}{\hat{y}^i(1-\hat{y}^i)}\right]\frac{\partial \hat{y}^i}{\partial w_j}$$

Recall that $\hat{y}^i = \dfrac{1}{1+\exp(-\mathbf{w}^T\mathbf{x}^i)}$

Small tip

for $g(t) = \dfrac{1}{1+\exp(-t)}$ we have

$$\mathrm{g}'(\mathrm{t}) = \frac{\exp(-t)}{(1+\exp(-t))^2} = g(t)(1-g(t))$$

$$\frac{\partial \hat{y}^i}{\partial w_j} = \hat{y}^i(1-\hat{y}^i)\frac{\partial(\mathbf{w}^T\mathbf{x}^i)}{\partial w_j} = \hat{y}^i(1-\hat{y}^i)x_j^i$$

$$\boxed{\frac{\partial l_i(\mathbf{w})}{\partial w_j} = (y^i - \hat{y}^i)x_j^i} \quad \boxed{\frac{\partial l(\mathbf{w})}{\partial w_j} = \sum_{i=1}^N (y^i - \hat{y}^i)x_j^i} \quad \boxed{\nabla l(\mathbf{w}) = \sum_{i=1}^N (y^i - \hat{y}^i)\mathbf{x}^i}$$

Batch Gradient Ascent for LR

$$\text{Given: training examples } (\mathbf{x}^i, y^i),\ i = 1,\ldots,N$$

$$\text{Let } \mathbf{w} \leftarrow (0,0,0,\ldots,0)$$

$$\text{Repeat until convergence}$$

$$\mathbf{d} \leftarrow (0,0,0,\ldots,0)$$

$$\text{For } i = 1 \text{ to } N \text{ do}$$

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

$$error = y^i - \hat{y}^i$$

$$\mathbf{d} = \mathbf{d} + error \cdot \mathbf{x}^i$$

$$\mathbf{w} \leftarrow \mathbf{w} + \eta\,\mathbf{d}$$

- **Online** gradient ascent algorithm can be easily constructed

Optional Notes

- No close-form solution for this optimization problem because of the logistic function
- However, other optimization techniques can also be used, for example Newton's method

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

where H is the Hessian matrix such that $H(i,j) = \frac{\partial^2 l}{\partial w_i \partial w_j}$

- For logistic regression, we have:

$$H = \boldsymbol{X}^T R \boldsymbol{X}$$

where $\boldsymbol{X}$ is our data matrix, each row corresponding to an instance, R is a $N \times N$ diagonal matrix with elements:

$$R_{ii} = \hat{y}^i(\hat{y}^i - 1)$$

- Newton's method enjoys faster convergence, but each step involves computing the Hessian and taking its inverse – expensive computation
- Also called *iterative reweighted least squares (IRLS)* method

Instability of MLE

- For linearly separable data, the maximum likelihood is achieved by
  - finding a linear decision boundary $\mathbf{w}^T\mathbf{x} = 0$ that separates the two classes perfectly
  - Make the magnitude of $\boldsymbol{w}$ go to infinity
- This instability can be avoided by adding a regularization term to the likelihood objective

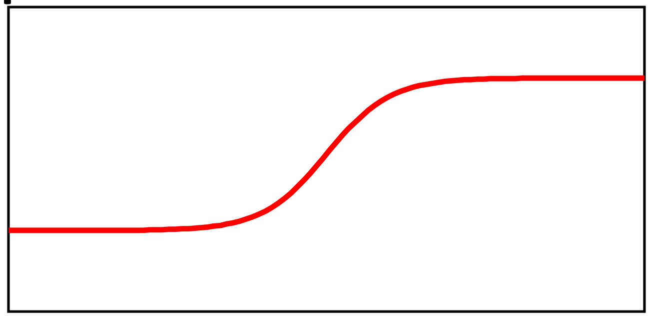
$$\arg\max_{\mathbf{w}} \left\{ \sum_i \log P(y^i \mid \mathbf{x}^i, \mathbf{w}) - \lambda \|\mathbf{w}\|^2 \right\}$$

Gradient ascent Update rule: $$\mathbf{w} \leftarrow \mathbf{w} + \eta\left(\sum_i (y^i - \hat{y}^i)\mathbf{x}^i - \lambda \mathbf{w}\right)$$

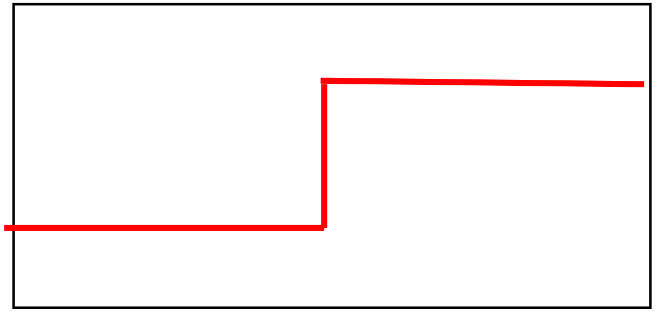
Connection between Logistic Regression and Perceptron Algorithm

- Both methods learn a linear function of the input features

- LR uses the logistic function, Perceptron uses a step function



$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$$



$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}\cdot\mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Both algorithms take a similar update rule:

$$\mathbf{w} = \mathbf{w} + \eta(y^i - h_{\mathbf{w}}(\mathbf{x}^i))\mathbf{x}^i$$

Multi-Class Logistic Regression

- For multiclass classification, we define the posterior probability using the so-called soft-max function

$$p(y = k|\mathrm{x}) = \hat{y}_k = \frac{\exp(\alpha_k)}{\sum_{j=1}^{K} \exp \alpha_j}$$

where α_k is given by

$$\alpha_k = \mathbf{w}_k^T \mathbf{x}$$

- Going through the same MLE derivations, we arrive at the following gradient:

$$\nabla_{\mathbf{w}_k} L = \sum_{i=1}^{N} (y_k^i - \hat{y}_k^i)\mathbf{x}^i$$

where $y_k^i = 1$ if $y^i = k$, and 0 otherwise

Summary of Logistic Regression

- Discriminative classifier
- Learns conditional probability distribution
  - $P(y \mid \mathbf{x})$ defined by a logistic function
  - Produces a linear decision boundary
- Maximum likelihood estimation
  - Gradient ascent bears strong similarity with perceptron
  - Unstable for linearly separable case, should use with regularization term to avoid this issue
  - Easily extended to multi-class problem using the soft-max function