8. Logistic Regression

STA3142 Statistical Machine Learning

Kibok Lee

Assistant Professor of

Applied Statistics / Statistics and Data Science

Mar 26, 2024

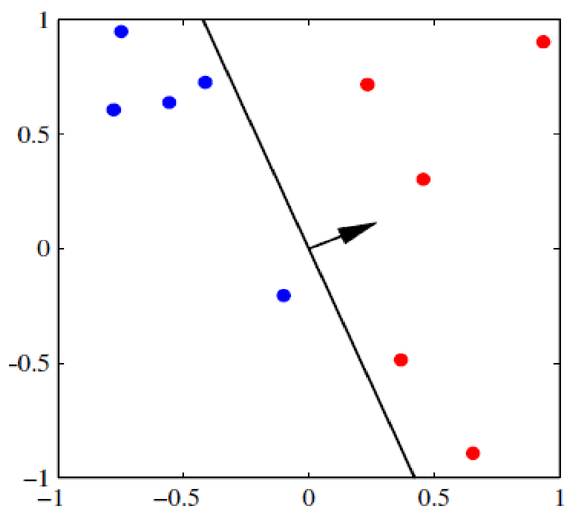
Assignment 1

- Due **Friday 3/29, 11:59pm**
- Topics
  - (Programming) NumPy basics
  - (Programming) Linear regression on a polynomial
  - (Math) Derivation and proof for linear regression

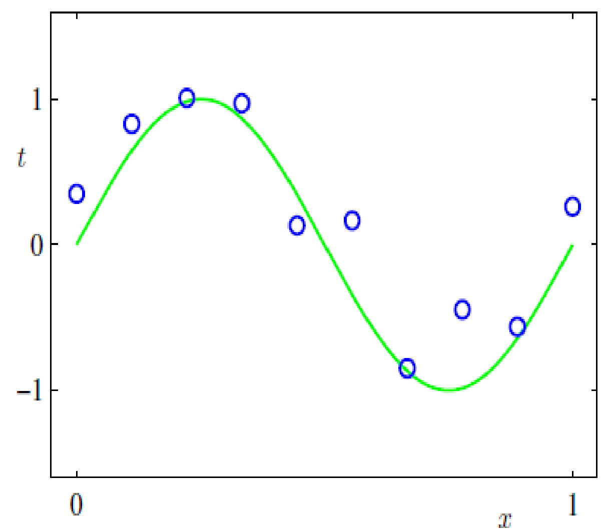
- Please read the instruction carefully!
  - Submit one pdf and one zip file separately
  - Write your code only in the designated spaces
  - Do not import additional libraries
  - …
- If you feel difficult, consider to take **option 2**.

Recap: Supervised Learning

- Learning a function $h: \mathcal{X} \to \mathcal{Y}$
- Labels could be discrete or continuous
  - Discrete labels: **classification** (today's topic)
  - Continuous labels: **regression**



classification



regression

Classification Problem

- Task: Given an input x, assign it to one of K distinct classes C_k where $k \in \{1, \dots, K\}$.
- Two-class classification ($K = 2$)
  - $y = 1$ means that x is in C_1 (or positive class).
  - $y = 0$ means that x is in C_2 (or negative class).
    - Or, $y = -1$ can be used depending on algorithms.
    - Cf. for one-class classification ($K = 1$), we still have a negative class, but we don't have training data for it.
- Multi-class classification ($K \geq 2$),
  - $y \in \{1, \dots, K\}$
  - Or, 1-of-K coding (or one-hot encoding) as target vector
    - e.g., $y = [0, 1, 0, 0, 0]^T$ means that x is in C_2.

Classification Problem

- Training: train a classifier $h(x)$ with training data
  - Training data: $\left\{\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(N)}, y^{(N)}\right)\right\}$

- Test (evaluation):
  - Test data: $\left\{\left(x_{\text{test}}^{(1)}, y_{\text{test}}^{(1)}\right), \left(x_{\text{test}}^{(2)}, y_{\text{test}}^{(2)}\right), \ldots, \left(x_{\text{test}}^{(N_{\text{test}})}, y_{\text{test}}^{(N_{\text{test}})}\right)\right\}$
  - The trained classifier produces predictions

$$\left\{h\left(x_{\text{test}}^{(1)}\right), h\left(x_{\text{test}}^{(2)}\right), \ldots, h\left(x_{\text{test}}^{(N_{\text{test}})}\right)\right\}$$

- Evaluation metric: 0-1 loss

Indicator function: $I(a) = \begin{cases} 1 & \text{if } a \text{ is true} \\ 0 & \text{if } a \text{ is false} \end{cases}$

$$\text{classification error} = \frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} I\left[h\left(x_{\text{test}}^{(j)}\right) \neq y_{\text{test}}^{(j)}\right]$$

Classification Strategies

- Learning the distributions $p(C_k|x)$
  - Discriminative models: Directly model $p(C_k|x)$ and learn parameters from the training set.
  - Generative models: Learn class densities $p(x|C_k)$ and priors $p(C_k)$ to obtain $p(x, C_k) = p(x|C_k)p(C_k)$
- Nearest neighbor classification
  - Given query data x, find the closest training points and do majority vote.
- Discriminant functions
  - Learn a function $h(x)$ that maps x onto some C_k.

Classification Strategies

- Learning the distributions $p(C_k|x)$
  - Discriminative models: Directly model $p(C_k|x)$ and learn parameters from the training set.
  - Generative models: Learn class densities $p(x|C_k)$ and priors $p(C_k)$ to obtain $p(x, C_k) = p(x|C_k)p(C_k)$
- Nearest neighbor classification
  - Given query data x, find the closest training points and do majority vote.
- Discriminant functions
  - Learn a function $h(x)$ that maps x onto some C_k.

Outline

- Learning $p(C_k|x)$ with discriminative models
  - Logistic Regression
  - Softmax Regression
    - Cross-Entropy Loss

- Convex Functions

Logistic Regression

Probabilistic Discriminative Models

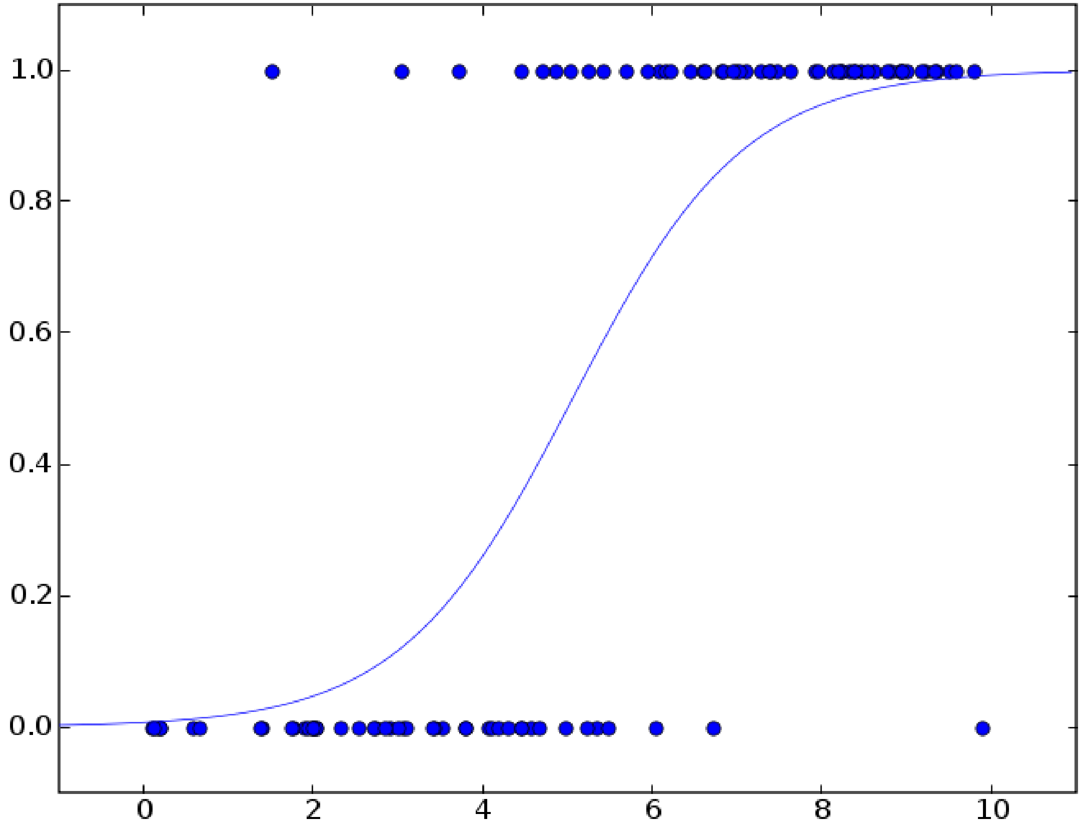
- Modeling decision boundary as a function of x
  - Learn $p(C_k|x)$ over data
    - Classification is done by taking argmax: $y = \mathrm{argmax}_k\, p(C_k|x)$
  - Directly predict class labels from inputs

- Cf. Probabilistic Generative Models
  - Learn $p(x|C_k)$ and $p(C_k)$ over data
    (maximum likelihood and prior)
    Then use Bayes' rule to predict $p(C_k|x)$

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)}$$
$$\propto p(x|C_k)p(C_k)$$

Example: Classification

- $x \in \mathbb{R}$



$p(y = 1|x)$

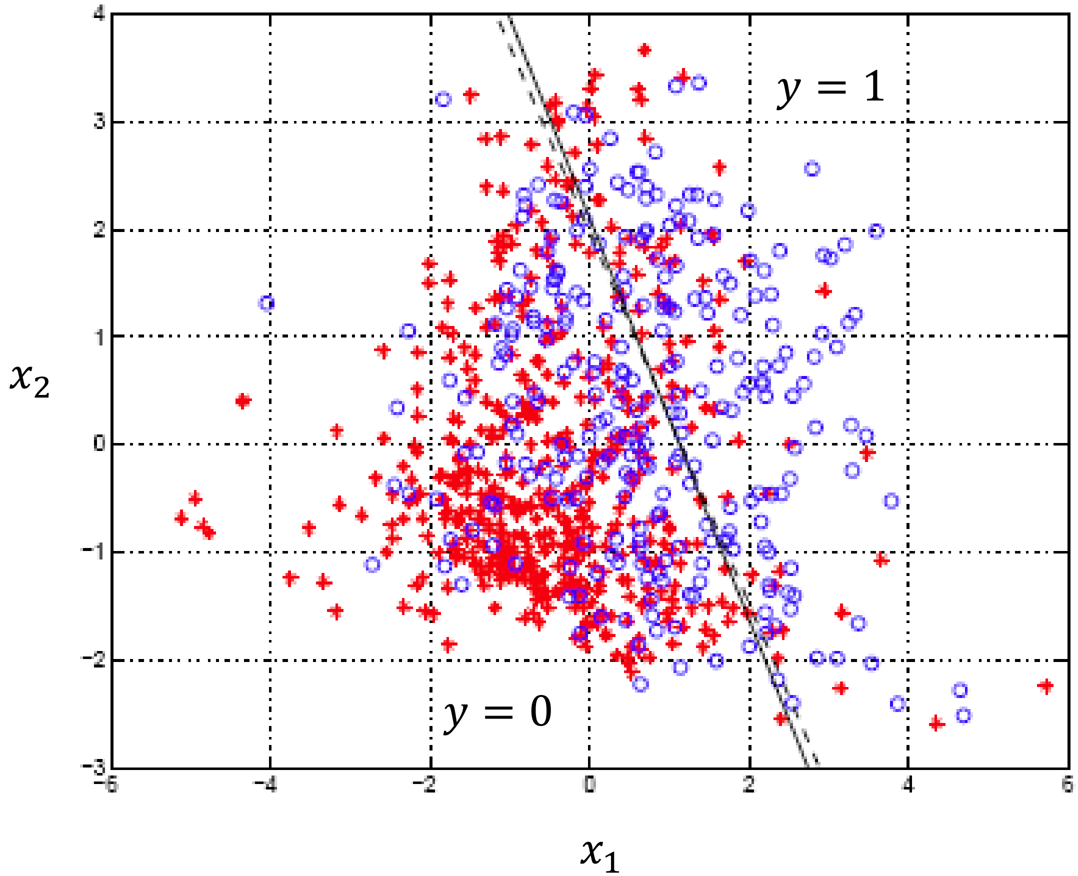
x

$y = 1$

$y = 0$

Example: Classification

- $x = [x_1, x_2]^T \in \mathbb{R}^2$



Logistic Regression

- Models the class posterior using a sigmoid applied to a linear function of the feature vector:

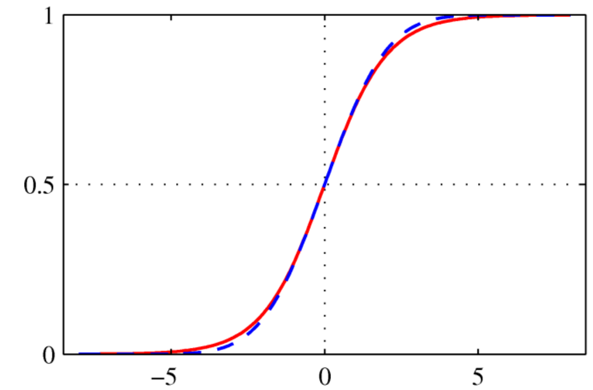
$$p(C_1|\phi) = h(\phi) = \sigma(\mathbf{w}^T \phi(\mathbf{x}))$$

- We can solve the parameter $\mathbf{w}$ by maximizing the likelihood of the training data.

Sigmoid and Logit Functions

- The **logistic sigmoid** function is:

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$



- Its inverse is the **logit** function (a.k.a. the log of the odds ratio):

$$a = \ln\left(\frac{\sigma}{1-\sigma}\right)$$

- It generalizes to **normalized exponential**, or **softmax**.

$$p_i = \frac{\exp(q_i)}{\sum_j \exp(q_j)}$$

Likelihood Function

- Depending on the value of the label y, the likelihood is defined as:

$$P(y = 1|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T \phi(\mathbf{x}))$$
$$P(y = 0|\mathbf{x}, \mathbf{w}) = 1 - \sigma(\mathbf{w}^T \phi(\mathbf{x}))$$

- Integrating all cases:

$$P(y|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T \phi(\mathbf{x}))^y (1 - \sigma(\mathbf{w}^T \phi(\mathbf{x})))^{(1-y)}$$

Learning Objective: Log-likelihood

- For a dataset $\{(\phi(\mathbf{x}^{(n)}), y^{(n)})\},\ \text{where } y^{(n)} \in \{0,1\}$ the likelihood function is

$$p(\mathbf{y}|\mathbf{w}) = \prod_{n=1}^{N} (h^{(n)})^{y^{(n)}} (1 - h^{(n)})^{1-y^{(n)}}$$

  - where

$$h^{(n)} = p(C_1|\phi(\mathbf{x}^{(n)})) = \sigma(\mathbf{w}^T \phi(\mathbf{x}^{(n)}))$$

- Define a loss function
  - Minimizing $E(\mathbf{w})$ maximizes likelihood.

$$E(\mathbf{w}) = -\log p(\mathbf{y}|\mathbf{w})$$

Derivation

- $\log P(\mathbf{y}|\mathbf{w}) = \sum_{n=1}^{N} y^{(n)} \log h^{(n)} + (1 - y^{(n)}) \log(1 - h^{(n)})$
- Gradient (matrix calculus)

$$\nabla_{\mathbf{w}} \log P(\mathbf{y}|\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(N)}, \mathbf{w})$$

$$= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \left(y^{(n)} \log h(\mathbf{x}^{(n)}, \mathbf{w}) + (1 - y^{(n)}) \log(1 - h(\mathbf{x}^{(n)}, \mathbf{w})) \right)$$

Derivation

- $\log P(\mathbf{y}|\mathbf{w}) = \sum_{n=1}^{N} y^{(n)} \log h^{(n)} + (1 - y^{(n)}) \log(1 - h^{(n)})$
- Gradient (matrix calculus)

$$\sigma^{(n)} \triangleq \sigma\left(\mathbf{w}^T \phi(\mathbf{x}^{(n)})\right) = h(\mathbf{x}^{(n)}, \mathbf{w}) = h^{(n)}$$

$$\nabla_{\mathbf{w}} \log P(\mathbf{y}|\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(N)}, \mathbf{w})$$

$$= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \left(y^{(n)} \log h(\mathbf{x}^{(n)}, \mathbf{w}) + (1 - y^{(n)}) \log(1 - h(\mathbf{x}^{(n)}, \mathbf{w})) \right)$$

Derivation

- $\log P(\mathbf{y}|\mathbf{w}) = \sum_{n=1}^{N} y^{(n)} \log h^{(n)} + (1 - y^{(n)}) \log(1 - h^{(n)})$
- Gradient (matrix calculus)

$$\sigma^{(n)} \triangleq \sigma\left(\mathbf{w}^T \phi(\mathbf{x}^{(n)})\right) = h(\mathbf{x}^{(n)}, \mathbf{w}) = h^{(n)}$$

$$\nabla_{\mathbf{w}} \log P(\mathbf{y}|\mathbf{x}^{(1)}, \dots \mathbf{x}^{(N)}, \mathbf{w})$$

$$= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \left(y^{(n)} \log h(\mathbf{x}^{(n)}, \mathbf{w}) + (1 - y^{(n)}) \log(1 - h(\mathbf{x}^{(n)}, \mathbf{w})) \right)$$

$$= \sum_{n=1}^{N} \left(y^{(n)} \frac{\sigma^{(n)}(1 - \sigma^{(n)})}{\sigma^{(n)}} - (1 - y^{(n)}) \frac{\sigma^{(n)}(1 - \sigma^{(n)})}{1 - \sigma^{(n)}} \right) \nabla_{\mathbf{w}}(\mathbf{w}^T \phi(\mathbf{x}^{(n)}))$$

$$\frac{\partial}{\partial s}\sigma(s) = \frac{\partial}{\partial s}\left(\frac{1}{1 + \exp(-s)}\right) = \sigma(s)(1 - \sigma(s))$$

Derivation

- $\log P(\mathbf{y}|\mathbf{w}) = \sum_{n=1}^{N} y^{(n)} \log h^{(n)} + (1 - y^{(n)}) \log(1 - h^{(n)})$
- Gradient (matrix calculus)

$$\sigma^{(n)} \triangleq \sigma\left(\mathbf{w}^T \phi(\mathbf{x}^{(n)})\right) = h(\mathbf{x}^{(n)}, \mathbf{w}) = h^{(n)}$$

$$\nabla_{\mathbf{w}} \log P(\mathbf{y}|\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(N)}, \mathbf{w})$$

$$= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \left(y^{(n)} \log h(\mathbf{x}^{(n)}, \mathbf{w}) + (1 - y^{(n)}) \log(1 - h(\mathbf{x}^{(n)}, \mathbf{w})) \right)$$

$$= \sum_{n=1}^{N} \left(y^{(n)} \frac{\sigma^{(n)}(1 - \sigma^{(n)})}{\sigma^{(n)}} - (1 - y^{(n)}) \frac{\sigma^{(n)}(1 - \sigma^{(n)})}{1 - \sigma^{(n)}} \right) \nabla_{\mathbf{w}}(\mathbf{w}^T \phi(\mathbf{x}^{(n)}))$$

$$= \sum_{n=1}^{N} \left(y^{(n)}(1 - \sigma^{(n)}) - (1 - y^{(n)})\sigma^{(n)} \right) \nabla_{\mathbf{w}}(\mathbf{w}^T \phi(\mathbf{x}^{(n)}))$$

Derivation

- $\log P(\mathbf{y}|\mathbf{w}) = \sum_{n=1}^{N} y^{(n)} \log h^{(n)} + (1 - y^{(n)}) \log(1 - h^{(n)})$
- Gradient (matrix calculus)

$$\sigma^{(n)} \triangleq \sigma\left(\mathbf{w}^T \phi(\mathbf{x}^{(n)})\right) = h(\mathbf{x}^{(n)}, \mathbf{w}) = h^{(n)}$$

$$\nabla_{\mathbf{w}} \log P(\mathbf{y}|\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(N)}, \mathbf{w})$$

$$= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \left(y^{(n)} \log h(\mathbf{x}^{(n)}, \mathbf{w}) + (1 - y^{(n)}) \log(1 - h(\mathbf{x}^{(n)}, \mathbf{w})) \right)$$

$$= \sum_{n=1}^{N} \left(y^{(n)} \frac{\sigma^{(n)}(1 - \sigma^{(n)})}{\sigma^{(n)}} - (1 - y^{(n)}) \frac{\sigma^{(n)}(1 - \sigma^{(n)})}{1 - \sigma^{(n)}} \right) \nabla_{\mathbf{w}}(\mathbf{w}^T \phi(\mathbf{x}^{(n)}))$$

$$= \sum_{n=1}^{N} \left(y^{(n)}(1 - \sigma^{(n)}) - (1 - y^{(n)})\sigma^{(n)} \right) \nabla_{\mathbf{w}}(\mathbf{w}^T \phi(\mathbf{x}^{(n)}))$$

$$= \sum_{n=1}^{N} \left(y^{(n)} - \sigma^{(n)} \right) \phi(\mathbf{x}^{(n)})$$

Logistic Regression: Gradient Descent

- Taking the gradient of $E(\mathbf{w})$ gives us

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \sum_{n=1}^{N} \left(h^{(n)} - y^{(n)} \right) \phi(\mathbf{x}^{(n)})$$

  - where $h^{(n)} = p(C_1 | \phi(\mathbf{x}^{(n)})) = \sigma(\mathbf{w}^T \phi(\mathbf{x}^{(n)}))$

  - This is essentially the same gradient expression that appeared in linear regression with least-squares.
  - Note the error term between model prediction and target value:
    - Logistic regression: $h^{(n)} - y^{(n)} = \sigma(\mathbf{w}^T \phi(x^{(n)})) - y^{(n)}$
    - Cf. Linear regression: $h^{(n)} - y^{(n)} = \mathbf{w}^T \phi(x^{(n)}) - y^{(n)}$

Recall: Newton's Method

- We want to minimize $E(\mathbf{w})$
  - Convert $E'(\mathbf{w}) = f(\mathbf{w})$
  - Repeat until convergence

$$\mathbf{w} := \mathbf{w} - \frac{E'(\mathbf{w})}{E''(\mathbf{w})}$$

Newton update when $\mathbf{w}$ is a scalar

- This method can be extended for multivariate case:

$$\mathbf{w} := \mathbf{w} - H^{-1}\nabla_{\mathbf{w}}E$$

Newton update when $\mathbf{w}$ is a vector

where H is a hessian matrix (evaluated at $\mathbf{w}$)

$$H_{ij}(\mathbf{w}) = \frac{\partial^2 E(\mathbf{w})}{\partial \mathbf{w}_i \partial \mathbf{w}_j}$$

Logistic Regression Has No Closed-form Solution

- Recall: For linear regression, least squares has a closed-form solution:

$$\mathbf{w}_{ML} = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{y}$$

- Hessian of linear regression is $\mathbf{\Phi}^T\mathbf{\Phi}$.

- Hessian of logistic regression is $\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi}$, where $\mathbf{R}$ is a diagonal matrix satisfying

$$R_{nn} = h^{(n)}(1 - h^{(n)})$$

Logistic Regression Has No Closed-form Solution

- Recall: For linear regression, least squares has a closed-form solution:

$$\mathbf{w}_{ML} = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{y}$$

- It generalizes to weighted least squares with an $N \times N$ diagonal weight matrix $\mathbf{R}$.

$$\mathbf{w}_{WLS} = \left(\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{R}\mathbf{y}$$

- For logistic regression, however, there is no closed-form solution ($\mathbf{R}$ depends on $\mathbf{w}$).
  - Hence, we need to apply an iterative method, e.g., gradient descent or Newton's method.
  - It is convex, so we will arrive at the optimal solution.

Iterative Solution

- Apply Newton-Raphson method to iterate to a solution $\mathbf{w}$ for $\nabla E(\mathbf{w}) = 0$
- This involves least squares with weights $\mathbf{R}$:

$$R_{nn} = h^{(n)}(1 - h^{(n)})$$

- Since $\mathbf{R}$ depends on $\mathbf{w}$ (and vice versa), we get iterative reweighted least squares (IRLS)

  - where $$\mathbf{w}^{(new)} = (\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{R}\mathbf{z}$$
$$\mathbf{z} = \mathbf{\Phi}\mathbf{w}^{(old)} - \mathbf{R}^{-1}(\mathbf{h} - \mathbf{y})$$

Softmax Regression

Softmax Regression

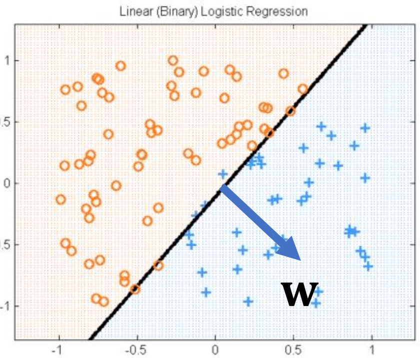
- Softmax regression is a multiclass generalization of logistic regression.
  - Also known as multinomial logistic regression, softmax classifier, maximum entropy classifier, …

- When $K = 2$, logistic regression models class conditional probabilities as:

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

$$\sigma(s) = \frac{1}{1+\exp(-s)} = \frac{\exp(s)}{1+\exp(s)}$$

$$p(y=0|\mathbf{x};\mathbf{w}) = \frac{1}{1+\exp(\mathbf{w}^T\phi(\mathbf{x}))}$$



  - Geometrical intuition: $\mathbf{w}$ is direction to increase the classification score (orthogonal to decision boundary)

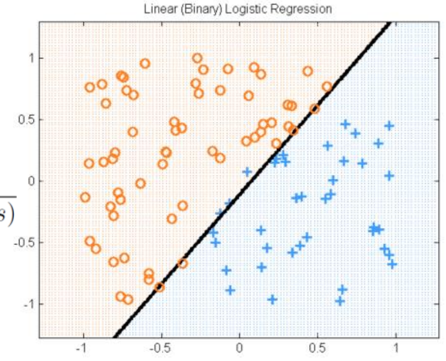
Softmax Regression

- When $K = 2$, logistic regression models class conditional probabilities as:

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

$$p(y=0|\mathbf{x};\mathbf{w}) = \frac{1}{1+\exp(\mathbf{w}^T\phi(\mathbf{x}))}$$

$$\sigma(s) = \frac{1}{1+\exp(-s)} = \frac{\exp(s)}{1+\exp(s)}$$

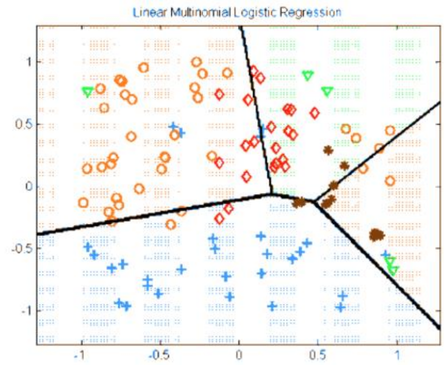


- When $K \geq 2$, we extend the formulation as:

$$p(y=k|\mathbf{x};\mathbf{w}) = \frac{\exp(\mathbf{w}_k^T\phi(\mathbf{x}))}{1+\sum_{j=1}^{K-1}\exp(\mathbf{w}_j^T\phi(\mathbf{x}))} \quad \text{(for } k = 1, \dots, K-1\text{)}$$

$$p(y=K|\mathbf{x};\mathbf{w}) = \frac{1}{1+\sum_{j=1}^{K-1}\exp(\mathbf{w}_j^T\phi(\mathbf{x}))}$$

  - Where is $\mathbf{w}_K$?



Softmax Regression

- Softmax regression with $\{\mathbf{w}_1, \dots, \mathbf{w}_{K-1}\}$

$$p(y=k|\mathbf{x};\mathbf{w}) = \frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}))}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^T \phi(\mathbf{x}))} \quad (\text{for } k = 1, \dots, K-1)$$

$$p(y=K|\mathbf{x};\mathbf{w}) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^T \phi(\mathbf{x}))}$$

- By shifting all weight vectors by an arbitrary $\mathbf{w}_K$ $(\mathbf{w}_j \leftarrow \mathbf{w}_j - \mathbf{w}_K)$, we have the K-th weight vector.

$$p(y=k|\mathbf{x};\mathbf{w}) = \frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}))}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^T \phi(\mathbf{x}))} \quad (\text{for } k = 1, \dots, K)$$

  - Derivation?

Softmax Regression

- Softmax regression with $\{\mathbf{w}_1, \dots, \mathbf{w}_K\}$

$$
\begin{aligned}
p(y=k|\mathbf{x};\mathbf{w}) &= \frac{\exp((\mathbf{w}_k - \mathbf{w}_K)^T \phi(\mathbf{x}))}{1 + \sum_{j=1}^{K-1} \exp((\mathbf{w}_j - \mathbf{w}_K)^T \phi(\mathbf{x}))} \\
&= \frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}) - \mathbf{w}_K^T \phi(\mathbf{x}))}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^T \phi(\mathbf{x}) - \mathbf{w}_K^T \phi(\mathbf{x}))} \\
&= \frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x})) / \exp(\mathbf{w}_K^T \phi(\mathbf{x}))}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^T \phi(\mathbf{x})) / \exp(\mathbf{w}_K^T \phi(\mathbf{x}))} \\
&= \frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}))}{\exp(\mathbf{w}_K^T \phi(\mathbf{x})) + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^T \phi(\mathbf{x}))} \\
&= \frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}))}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^T \phi(\mathbf{x}))}
\end{aligned}
$$

Learning Objective: Log-likelihood

- Softmax regression:

Indicator function: $I(a) = \begin{cases} 1 & \text{if } a \text{ is true} \\ 0 & \text{if } a \text{ is false} \end{cases}$

$$p(y = k|\mathbf{x}; \mathbf{w}) = \frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}))}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^T \phi(\mathbf{x}))}$$

- Or,

$$p(y|\mathbf{x}; \mathbf{w}) = \prod_{k=1}^{K} \left[\frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}))}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^T \phi(\mathbf{x}))} \right]^{I(y=k)}$$

- Log-likelihood: $\log p(D|\mathbf{w}) = \sum_i \log p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w})$

$$= \sum_i \log \prod_{k=1}^{M} \left[\frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}^{(i)}))}{\sum_{j=1}^{M} \exp(\mathbf{w}_j^T \phi(\mathbf{x}^{(i)}))} \right]^{I(y^{(i)}=k)}$$

- We can learn parameters $\mathbf{w}$ by gradient ascent or Newton's method.

Cross-Entropy Loss

- Softmax regression is a multiclass generalization of logistic regression.
  - Also known as multinomial logistic regression, softmax classifier, maximum entropy classifier, …
- If we take $\mathbf{s} \in \mathbb{R}^K$ as a general input, we call the negative log-likelihood (NLL) as **cross-entropy loss**.
  - Softmax regression = linear regression + softmax

    $\phi(\mathbf{x}) \to \mathbf{p}$ $\quad\quad$ $\phi(\mathbf{x}) \to \mathbf{s}$ $\quad$ $\mathbf{s} \to \mathbf{p}$
    - Classification is done by taking argmax:
      $y = \text{argmax}_k \mathbf{p} = \text{argmax}_k \mathbf{s}$, where $\mathbf{p} = [p_1, \dots, p_K] \in [0,1]^K$
    - At test time, you can skip softmax.
  - Cross-entropy loss = softmax + NLL loss

    $\mathbf{s} \to L$ $\quad\quad$ $\mathbf{s} \to \mathbf{p}$ $\quad$ $\mathbf{p} \to L$

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores

$$\mathbf{s} = f(\mathbf{x}, \mathbf{w})$$

Softmax function

$$p_k = p(y = k | \mathbf{x}; \mathbf{w}) = \frac{\exp(s_k)}{\sum_j \exp(s_j)}$$

cat	**3.2**
car	5.1
frog	-1.7

Slide credit: Justin Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Softmax function

Classifier scores

$$\mathbf{s} = f(\mathbf{x}, \mathbf{w})$$

$$p_k = p(y = k|\mathbf{x}; \mathbf{w}) = \frac{\exp(s_k)}{\sum_j \exp(s_j)}$$

Probabilities must be >= 0

Probabilities must sum to 1

			exp		normalize	
cat		**3.2**		**24.5**		**0.13**
car		5.1	$\rightarrow$	164	$\rightarrow$	0.87
frog		-1.7		0.18		0.00

Unnormalized log-probabilities / logits

unnormalized probabilities

probabilities

Slide credit: Justin Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Softmax function

Classifier scores

$\mathbf{s} = f(\mathbf{x}, \mathbf{w})$

$$p_k = p(y = k | \mathbf{x}; \mathbf{w}) = \frac{\exp(s_k)}{\sum_j \exp(s_j)}$$

Loss

$$L_i = -\log(p_{y^{(i)}})$$

Probabilities must be >= 0

Probabilities must sum to 1

	Unnormalized log-probabilities / logits	$\xrightarrow{\text{exp}}$	unnormalized probabilities	$\xrightarrow{\text{normalize}}$	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164		0.87
frog	-1.7		0.18		0.00

L_i = -log(0.13)

= **2.04**

Maximum Likelihood Estimation
Choose weights to maximize the likelihood of the observed data

Slide credit: Justin Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Softmax function

$$p_k = p(y = k|\mathbf{x};\mathbf{w}) = \frac{\exp(s_k)}{\sum_j \exp(s_j)}$$

Classifier scores

$$\mathbf{s} = f(\mathbf{x}, \mathbf{w})$$

Probabilities must be >= 0

Probabilities must sum to 1

Loss

$$L_i = -\log(p_{y^{(i)}})$$

	Unnormalized log-probabilities / logits		unnormalized probabilities		probabilities		Correct probs
cat	**3.2**	exp →	**24.5**	normalize →	**0.13**	→ Compare ←	**1.00**
car	5.1		164		0.87		0.00
frog	-1.7		0.18		0.00		0.00

Kullback-Leibler Divergence

$$D_{KL}(\boldsymbol{Q} \,||\, \boldsymbol{P}) = \sum_y \boldsymbol{Q}(y) \log \frac{\boldsymbol{Q}(y)}{\boldsymbol{P}(y)}$$

Slide credit: Justin Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Softmax function

Classifier scores

$$\mathbf{s} = f(\mathbf{x}, \mathbf{w})$$

$$p_k = p(y = k|\mathbf{x}; \mathbf{w}) = \frac{\exp(s_k)}{\sum_j \exp(s_j)}$$

Probabilities must be >= 0

Probabilities must sum to 1

Loss

$$L_i = -\log(p_{y^{(i)}})$$

cat	**3.2**	exp →	**24.5**	normalize →	**0.13**	→ Compare ←	**1.00**
car	5.1		164		0.87		0.00
frog	-1.7		0.18		0.00		0.00
	Unnormalized log-probabilities / logits		unnormalized probabilities		probabilities		Correct probs

Cross-Entropy:

$$H(Q, P) = H(Q) + D_{KL}(Q \,||\, P)$$

Slide credit: Justin Johnson

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

cat	**3.2**
car	5.1
frog	-1.7

Classifier scores

$$\mathbf{s} = f(\mathbf{x}, \mathbf{w})$$

Softmax function

$$p_k = p(y = k|\mathbf{x}; \mathbf{w}) = \frac{\exp(s_k)}{\sum_j \exp(s_j)}$$

Putting it all together:

$$L_i = -\log p\left(y = y^{(i)} | \mathbf{x} = \mathbf{x}^{(i)}; \mathbf{w}\right)$$

$$= -\log \frac{\exp\left(s_{y^{(i)}}\right)}{\sum_j \exp(s_j)}$$

$$= -s_{y^{(i)}} - \log \sum_j \exp(s_j)$$

Slide credit: Justin Johnson

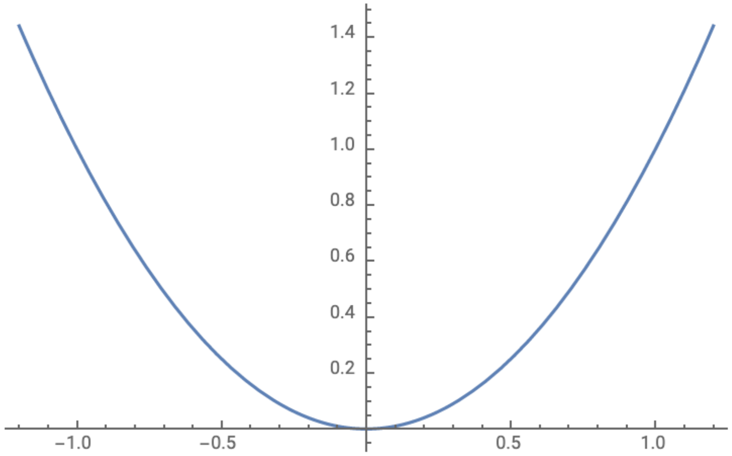
Convex Functions

Convex Functions

A function $f : X \subseteq \mathbb{R}^N \to \mathbb{R}$ is **convex** if for all $x_1, x_2 \in X, t \in [0,1]$,

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

Example: $f(x) = x^2$ is convex:

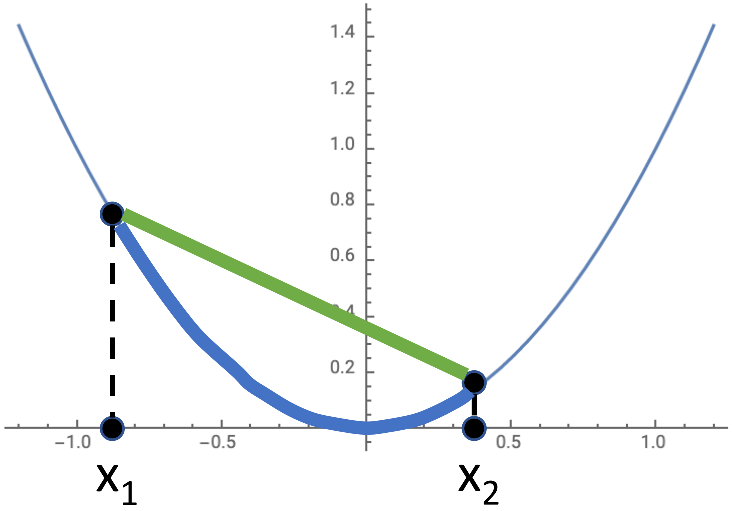


Convex Functions

A function $f : X \subseteq \mathbb{R}^N \to \mathbb{R}$ is **convex** if for all $x_1, x_2 \in X, t \in [0, 1]$,

$$\boxed{f(tx_1 + (1-t)x_2)} \leq \boxed{tf(x_1) + (1-t)f(x_2)}$$

Example: $f(x) = x^2$ is convex:

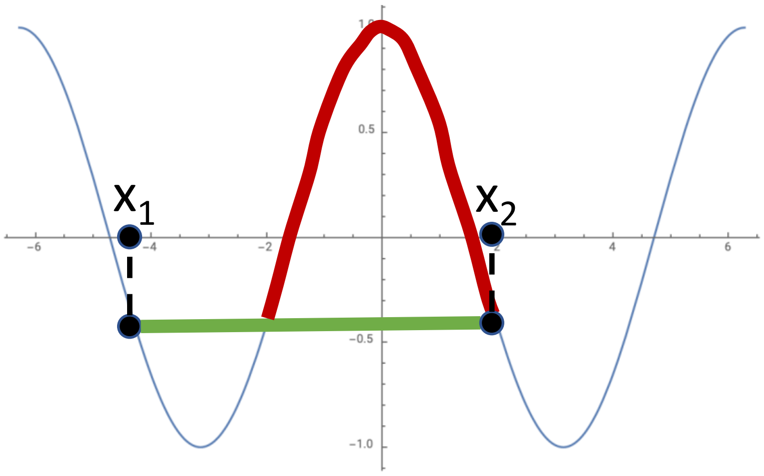


Convex Functions

A function $f : X \subseteq \mathbb{R}^N \to \mathbb{R}$ is **convex** if for all $x_1, x_2 \in X, t \in [0, 1]$,

$$\boxed{f(tx_1 + (1-t)x_2)} \leq \boxed{tf(x_1) + (1-t)f(x_2)}$$

Example: $f(x) = \cos(x)$
is <u>not</u> convex:

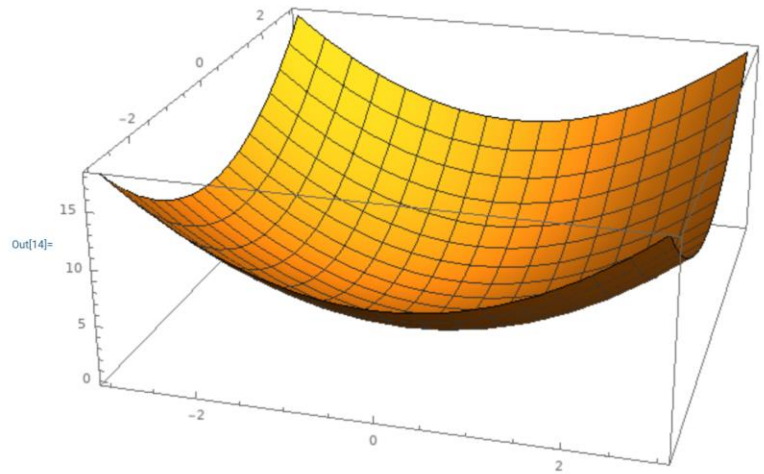


Convex Functions

A function $f : X \subseteq \mathbb{R}^N \to \mathbb{R}$ is **convex** if for all $x_1, x_2 \in X, t \in [0, 1]$,

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

Intuition: A convex function is a (multidimensional) bowl



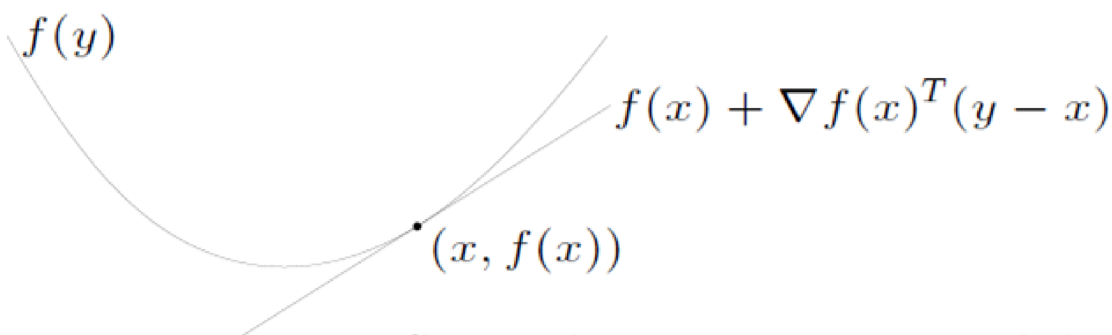
Generally speaking, convex functions are **easy to optimize**: can derive theoretical guarantees about **converging to global minimum***

*Many technical details inside!

Convex Functions

- A differentiable f is convex if and only if

$$f(y) \geq f(x) + \nabla f(x)^T (y - x)$$



first-order approximation of f is global underestimator

- A twice-differentiable f is convex if and only if its Hessian is positive semi-definite ($H = \nabla^2 f(x) \succcurlyeq 0$).
  - This is handy when proving convexity of a function.

Image credit: Stephen Boyd

Next:
Generative Classifiers