Review: Logistic Regression

CS114B Lab 4

Kenneth Lai

February 26, 2021

Logistic Regression

- Suppose we observe a movie review $d =$ "predictable with no fun". Is the review positive or negative?

Logistic Regression

- Documents are characterized by features

Logistic Regression

- Documents are characterized by features
  - No independence assumptions

Logistic Regression

- Documents are characterized by features
  - No independence assumptions
- For each feature j:
  - Value x_j
  - Weight w_j

Logistic Regression

- Documents are characterized by features
  - No independence assumptions
- For each feature j:
  - Value x_j
  - Weight w_j
- Bias term b

Logistic Regression

- Documents are characterized by features
  - No independence assumptions
- For each feature j:
  - Value x_j
  - Weight w_j
- Bias term b
- "Score" (log-odds) $z = \left(\sum_{j=1}^{n} w_j x_j \right) + b = \mathbf{w} \cdot \mathbf{x} + b$

Logistic Regression

- Logistic function $\sigma(z) = \dfrac{1}{1 + e^{-z}}$

Logistic Regression

- Logistic function $\sigma(z) = \dfrac{1}{1 + e^{-z}}$
- $p(y = 1|\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \dfrac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$
- $p(y = 0|\mathbf{x}) = 1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \dfrac{e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$

Cross-entropy Loss

- Let $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$ be the classifier output for some $\mathbf{x}$

Cross-entropy Loss

- Let $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$ be the classifier output for some $\mathbf{x}$
- What is the probability that the classifier is correct?

Cross-entropy Loss

- Let $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$ be the classifier output for some $\mathbf{x}$
- What is the probability that the classifier is correct?
  - If $y = 1$, then $P(y = 1|\mathbf{x}) = \hat{y}$
  - If $y = 0$, then $P(y = 0|\mathbf{x}) = 1 - \hat{y}$

Cross-entropy Loss

- In general, $P(y|\mathbf{x}) = \hat{y}^y (1 - \hat{y})^{1-y}$

Cross-entropy Loss

- In general, $P(y|\mathbf{x}) = \hat{y}^y(1-\hat{y})^{1-y}$
- Take the log of both sides:
  $\log P(y|\mathbf{x}) = y\log\hat{y} + (1-y)\log(1-\hat{y})$

Cross-entropy Loss

- In general, $P(y|\mathbf{x}) = \hat{y}^y (1 - \hat{y})^{1-y}$
- Take the log of both sides:
  $\log P(y|\mathbf{x}) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$
- Turn this into a loss function:
  $L_{CE}(\hat{y}, y) = -\log P(y|\mathbf{x}) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$

Cross-entropy Loss

- Minimize average loss for each example i:

$$Cost(\hat{y}, y) = \frac{1}{m} \sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})$$

Cross-entropy Loss

- Minimize average loss for each example i:

$$Cost(\hat{y}, y) = \frac{1}{m} \sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})$$

- How?

Gradient Descent

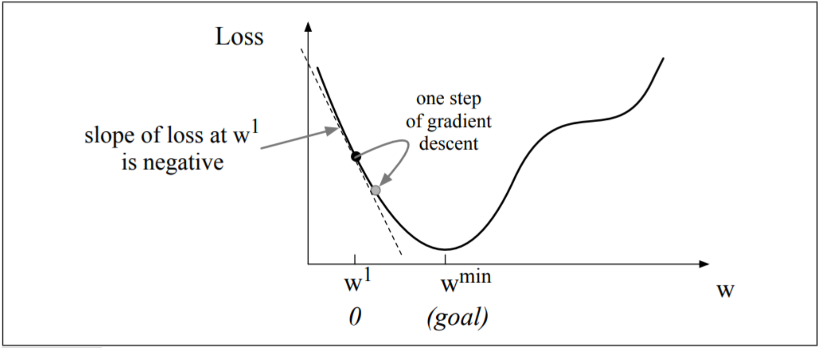

Figure 5.3 The first step in iteratively finding the minimum of this loss function, by moving w in the reverse direction from the slope of the function. Since the slope is negative, we need to move w in a positive direction, to the right. Here superscripts are used for learning steps, so w^1 means the initial value of w (which is 0), w^2 at the second step, and so on.

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
    - $\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_n} \\ \frac{\partial L}{\partial b} \end{bmatrix}$

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
    - $\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_n} \\ \frac{\partial L}{\partial b} \end{bmatrix}$
    - $\approx$ slope of loss function L

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient
- $\theta_{t+1} = \theta_t - \eta \nabla L$

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient
- $\theta_{t+1} = \theta_t - \eta \nabla L$
  - $\eta =$ learning rate

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient
- $\theta_{t+1} = \theta_t - \eta \nabla L$
  - $\eta =$ learning rate
    - "Hyperparameter": parameter set before training

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient

- $\theta_{t+1} = \theta_t - \eta \nabla L$
  - $\eta =$ learning rate
    - "Hyperparameter": parameter set before training
    - Trade-off between speed of convergence and "zig-zag" behavior

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient

- $\theta_{t+1} = \theta_t - \eta \nabla L$
  - $\eta =$ learning rate
    - "Hyperparameter": parameter set before training
    - Trade-off between speed of convergence and "zig-zag" behavior
    - Often a function of t

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient
- $\theta_{t+1} = \theta_t - \eta \nabla L$
  - $\eta =$ learning rate
    - "Hyperparameter": parameter set before training
    - Trade-off between speed of convergence and "zig-zag" behavior
    - Often a function of t
- Because L is convex, we eventually reach a global minimum

Gradient Descent

- $L_{CE}(\hat{y}, y) = -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1 - y) \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b))]$

Gradient Descent

- $L_{CE}(\hat{y}, y) = -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1 - y) \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b))]$
- ...
- (calculus)
- ...

Gradient Descent

- $L_{CE}(\hat{y}, y) = -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1 - y) \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b))]$
- ...
- (calculus)
- ...
- $\frac{\partial L}{\partial w_j} = [\sigma(\mathbf{w} \cdot \mathbf{x} + b) - y]x_j$
- $\frac{\partial L}{\partial b} = \sigma(\mathbf{w} \cdot \mathbf{x} + b) - y$

Gradient Descent

- Stochastic gradient descent: update θ after every training example

Gradient Descent

- Stochastic gradient descent: update θ after every training example
  - Can result in very choppy movements

Gradient Descent

- Stochastic gradient descent: update θ after every training example
  - Can result in very choppy movements
- Batch gradient descent: update θ after processing the entire training set

Gradient Descent

- Stochastic gradient descent: update θ after every training example
  - Can result in very choppy movements
- Batch gradient descent: update θ after processing the entire training set
- Minibatch gradient descent: update θ after m training examples

Gradient Descent

- Stochastic gradient descent: update θ after every training example
  - Can result in very choppy movements
- Batch gradient descent: update θ after processing the entire training set
- Minibatch gradient descent: update θ after m training examples
  - Gradient = average of individual gradients

Gradient Descent

- Stochastic gradient descent: update θ after every training example
  - Can result in very choppy movements
- Batch gradient descent: update θ after processing the entire training set
- Minibatch gradient descent: update θ after m training examples
  - Gradient = average of individual gradients

$$\frac{\partial Cost}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}[\sigma(\mathbf{w}\cdot\mathbf{x}^{(i)} + b) - y^{(i)}]x_j^{(i)}$$

Regularization

- Problem of overfitting

Regularization

- Problem of overfitting
  - Models can fit the training data too well

Regularization

- Problem of overfitting
  - Models can fit the training data too well
    - Accidental correlations get high weights

Regularization

- Problem of overfitting
  - Models can fit the training data too well
    - Accidental correlations get high weights
    - Poor generalization performance

Regularization

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \left(\sum_{i=1}^{m} \log P(y^{(i)} | \mathbf{x}^{(i)}) \right) - \alpha R(\theta)$$

Regularization

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \left(\sum_{i=1}^{m} \log P(y^{(i)} | \mathbf{x}^{(i)}) \right) - \alpha R(\theta)$$

- $R(\theta) =$ regularization term
- $\alpha =$ amount of regularization

Regularization

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \left(\sum_{i=1}^{m} \log P(y^{(i)} | \mathbf{x}^{(i)}) \right) - \alpha R(\theta)$$

- $R(\theta)$ = regularization term
- α = amount of regularization
    - Another hyperparameter

L1 Regularization

$$R(\theta) = ||\theta||_1 = \sum_{j=1}^{n} |\theta_j|$$

L1 Regularization

$$R(\theta) = ||\theta||_1 = \sum_{j=1}^{n} |\theta_j|$$

- $=$ sum of absolute values of weights
- Manhattan distance
- Lasso regression
- Some large weights, many zero weights

L2 Regularization

$$R(\theta) = ||\theta||_2^2 = \sum_{j=1}^{n} \theta_j^2$$

L2 Regularization

$$R(\theta) = ||\theta||_2^2 = \sum_{j=1}^{n} \theta_j^2$$

- $=$ sum of squares of weights
- Euclidean distance
- Ridge regression
- Many small weights

Multinomial Logistic Regression

- (aka maximum entropy classifier)

Multinomial Logistic Regression

- (aka maximum entropy classifier)
- Logistic regression with more than two classes

Logistic Regression

- Suppose we observe a movie review $d =$ "predictable with no fun". Is the review positive, negative, or neutral?

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{j=1}^{n} w_{jc} x_j\right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{j=1}^{n} w_{jc} x_j \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

- $\mathrm{softmax}(z_c) = \frac{e^{z_c}}{\sum_{k=1}^{K} e^{z_k}} = P(y = c|\mathbf{x}) = \hat{y}_c$

Multinomial Logistic Regression

- Cross-entropy loss $L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$

Multinomial Logistic Regression

- Cross-entropy loss $L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$
- Gradient ∇L becomes a matrix, where

Multinomial Logistic Regression

- Cross-entropy loss $L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$
- Gradient ∇L becomes a matrix, where
  - $\frac{\partial L}{\partial w_{jk}} = (\hat{y}_k - y_k) x_j$
  - $\frac{\partial L}{\partial b_k} = \hat{y}_k - y_k$