Logistic Regression

CS114 Lab 4

Kenneth Lai

February 7, 2020

Generative and Discriminative Models

- Generative models

Generative and Discriminative Models

- Generative models
  - Model joint distribution $P(c, d)$

Generative and Discriminative Models

- Generative models
  - Model joint distribution $P(c, d)$
    - $$\begin{aligned} P(c|d) &\propto P(d|c)P(c) \\ &\propto P(c, d) \end{aligned}$$

Generative and Discriminative Models

- Generative models
  - Model joint distribution $P(c, d)$
    - $$\begin{aligned} P(c|d) &\propto P(d|c)P(c) \\ &\propto P(c, d) \end{aligned}$$
  - Naïve Bayes, Hidden Markov Models, Bayesian Networks, etc.

Generative and Discriminative Models

- Generative models
  - Model joint distribution $P(c, d)$
    - $$\begin{aligned} P(c|d) &\propto P(d|c)P(c) \\ &\propto P(c, d) \end{aligned}$$
  - Naïve Bayes, Hidden Markov Models, Bayesian Networks, etc.
- Discriminative models

Generative and Discriminative Models

- Generative models
  - Model joint distribution $P(c, d)$
    - $$\begin{aligned} P(c|d) &\propto P(d|c)P(c) \\ &\propto P(c, d) \end{aligned}$$
  - Naïve Bayes, Hidden Markov Models, Bayesian Networks, etc.
- Discriminative models
  - Model conditional distribution $P(c|d)$ directly

Generative and Discriminative Models

- Generative models
  - Model joint distribution $P(c, d)$
    - $$\begin{aligned} P(c|d) &\propto P(d|c)P(c) \\ &\propto P(c, d) \end{aligned}$$
  - Naïve Bayes, Hidden Markov Models, Bayesian Networks, etc.
- Discriminative models
  - Model conditional distribution $P(c|d)$ directly
  - Logistic Regression, Conditional Random Fields, Neural Networks, etc.

Logistic Regression

- Suppose we observe a document $d =$ "Chinese Chinese Chinese Tokyo Japan". Is the document Chinese or Japanese?

Logistic Regression

- Training data:

document	class
Chinese Beijing Chinese	Chinese
Chinese Chinese Shanghai	Chinese
Chinese Macao	Chinese
Tokyo Japan Chinese	Japanese

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 i:
  - Value x_i
  - Weight w_i

Logistic Regression

- Documents are characterized by features
  - No independence assumptions
- For each feature i:
  - Value x_i
  - Weight w_i
- Bias term b

Logistic Regression

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

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:
  $L(\hat{y}, y) = -\log P(y|\mathbf{x}) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$

Cross-entropy Loss

- Minimize average loss for each example j:

$$Cost(\mathbf{w}, b) = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}^{(j)}, y^{(j)})$$

Cross-entropy Loss

- Minimize average loss for each example j:

$$Cost(\mathbf{w}, b) = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}^{(j)}, y^{(j)})$$

- How?

Gradient Descent

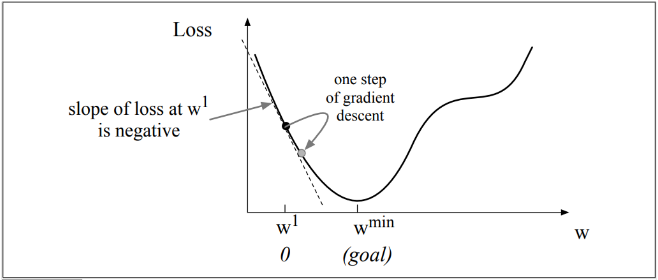

Figure 5.3 The first step in iteratively finding the minimum of this loss function, by moving w in the reverse direction from the slope of the function. Since the slope is negative, we need to move w in a positive direction, to the right. Here superscripts are used for learning steps, so w^1 means the 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
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ randomly
- 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
- 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
- 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
- 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
- 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
- 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

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

Gradient Descent

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

Gradient Descent

- $L(\mathbf{w}, b) = -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1 - y) \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b))]$
- ...
- (calculus)
- ...
- $\dfrac{\partial L}{\partial w_i} = [\sigma(\mathbf{w} \cdot \mathbf{x} + b) - y]x_i$
- $\dfrac{\partial L}{\partial b} = \sigma(\mathbf{w} \cdot \mathbf{x} + b) - y$

Example

- Python time!

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 = sum (Eisenstein) or average (Jurafsky and Martin) of individual gradients—use average for HW

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 $=$ sum (Eisenstein) or average (Jurafsky and Martin) of individual gradients—use average for HW

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

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

$$Cost(\mathbf{w}, b) = \left(\frac{1}{m} \sum_{j=1}^{m} L(\hat{y}^{(j)}, y^{(j)}) \right) + \alpha R(\mathbf{w})$$

Regularization

$$Cost(\mathbf{w}, b) = \left(\frac{1}{m} \sum_{j=1}^{m} L(\hat{y}^{(j)}, y^{(j)}) \right) + \alpha R(\mathbf{w})$$

- $R(\mathbf{w})$ = regularization term
- α = amount of regularization

Regularization

$$Cost(\mathbf{w}, b) = \left(\frac{1}{m}\sum_{j=1}^{m} L(\hat{y}^{(j)}, y^{(j)})\right) + \alpha R(\mathbf{w})$$

- $R(\mathbf{w})$ = regularization term
- α = amount of regularization
  - Another hyperparameter

L1 Regularization

$$R(\mathbf{w}) = ||\mathbf{w}||_1 = \sum_{i=1}^{n} |w_i|$$

L1 Regularization

$$R(\mathbf{w}) = ||\mathbf{w}||_1 = \sum_{i=1}^{n} |w_i|$$

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

L2 Regularization

$$R(\mathbf{w}) = ||\mathbf{w}||_2^2 = \sum_{i=1}^{n} w_i^2$$

L2 Regularization

$$R(\mathbf{w}) = ||\mathbf{w}||_2^2 = \sum_{i=1}^{n} w_i^2$$

- $=$ 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

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{i=1}^{n} w_{i,c} x_i \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{i=1}^{n} w_{i,c} x_i \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

- Softmax function $\sigma(z_c) = \frac{e^{z_c}}{\sum_{k \in C} e^{z_k}} = P(y = c|\mathbf{x})$