Logistic Regression
Q:
In recitation, we only covered the Perceptron mistake bound for **linearly separable data**. Isn’t that an unrealistic setting?

A:
Not at all! Even if your data isn’t linearly separable to begin with, we can often add features to make it so.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>+1</td>
<td>+</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>-</td>
</tr>
<tr>
<td>-1</td>
<td>+1</td>
<td>-</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>+</td>
</tr>
</tbody>
</table>

**Exercise:** Add another feature to transform this nonlinearly separable data into linearly separable data.
Reminders

• Homework 3: KNN, Perceptron, Lin.Reg.
  – Out: Wed, Feb 6
  – Due: Fri, Feb 15 at 11:59pm

• Homework 4: Logistic Regression
  – Out: Fri, Feb 15
  – Due: Fri, Mar 1 at 11:59pm

• Midterm Exam 1
  – Thu, Feb 21, 6:30pm – 8:00pm

• Today’s In-Class Poll
PROBABILISTIC LEARNING
Probabilistic Learning

**Function Approximation**

Previously, we assumed that our output was generated using a deterministic target function:

\[ x^{(i)} \sim p^* (\cdot) \]
\[ y^{(i)} = c^* (x^{(i)}) \]

Our goal was to learn a hypothesis \( h(x) \) that best approximates \( c^*(x) \)

**Probabilistic Learning**

Today, we assume that our output is sampled from a conditional probability distribution:

\[ x^{(i)} \sim p^* (\cdot) \]
\[ y^{(i)} \sim p^* (\cdot | x^{(i)}) \]

Our goal is to learn a probability distribution \( p(y|x) \) that best approximates \( p^*(y|x) \)
## Robotic Farming

<table>
<thead>
<tr>
<th></th>
<th>Deterministic</th>
<th>Probabilistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification (binary output)</td>
<td>Is this a picture of a wheat kernel?</td>
<td>Is this plant drought resistant?</td>
</tr>
<tr>
<td>Regression (continuous output)</td>
<td>How many wheat kernels are in this picture?</td>
<td>What will the yield of this plant be?</td>
</tr>
</tbody>
</table>
Bayes Optimal Classifier

Whiteboard

– Bayes Optimal Classifier
– Reducible / irreducible error
– Ex: Bayes Optimal Classifier for 0/1 Loss
Learning from Data (Frequentist)

Whiteboard

– Principle of Maximum Likelihood Estimation (MLE)

– Strawmen:
  • Example: Bernoulli
  • Example: Gaussian
  • Example: Conditional #1 (Bernoulli conditioned on Gaussian)
  • Example: Conditional #2 (Gaussians conditioned on Bernoulli)
MOTIVATION: LOGISTIC REGRESSION
Example: Image Classification

- **ImageNet LSVRC-2010 contest:**
  - **Dataset:** 1.2 million labeled images, 1000 classes
  - **Task:** Given a new image, label it with the correct class
  - **Multiclass** classification problem
- **Examples from** [http://image-net.org/](http://image-net.org/)
Bird

Warm-blooded egg-laying vertebrates characterized by feathers and forelimbs modified as wings

- marine animal, marine creature, sea animal, sea creature (1)
- scavenger (1)
- biped (0)
- predator, predatory animal (1)
- larva (49)
- acrodont (0)
- feeder (0)
- stunt (0)
- chordate (3087)
  - tunicate, urochordate, urochord (6)
  - cephalochordate (1)
  - vertebrate, craniate (3077)
    - mammal, mammalian (1169)
    - bird (871)
      - dickeybird, dickey-bird, dickybird, dicky-bird (0)
      - cock (1)
      - hen (0)
      - nester (0)
      - night bird (1)
      - bird of passage (0)
      - protoavis (0)
      - archaeopteryx, archeopteryx, Archaeopteryx lithographi
      - Sinornis (0)
      - ibero-mesornis (0)
      - archaeornis (0)
      - ratite, ratite bird, flightless bird (10)
      - carinate, carinate bird, flying bird (0)
      - passerine, passeriform bird (279)
      - nonpasserine bird (0)
      - bird of prey, raptor, raptorial bird (80)
      - gallinaceous bird, gallinaceous (114)
German iris, Iris kochii

Iris of northern Italy having deep blue-purple flowers; similar to but smaller than Iris germanica

- halophyte (0)
- succulent (39)
- cultivar (0)
- cultivated plant (0)
- weed (54)
- evergreen, evergreen plant (0)
- deciduous plant (0)
- vine (272)
- creeper (0)
- woody plant, ligneous plant (1868)
- geophyte (0)
- desert plant, xerophyte, xerophytic plant, xerophile, xerophilic
- mesophyte, mesophytic plant (0)
- aquatic plant, water plant, hydrophyte, hydrophytic plant (11)
- tuberous plant (0)
- bulbous plant (179)
- iridaceous plant (27)
- iris, flag, fleur-de-lis, sword lily (19)
  - bearded iris (4)
    - Florentine iris, orris, Iris germanica florentina, Iris germanica (0)
    - German iris, Iris germanica (0)
    - German iris, Iris kochii (0)
    - Dalmatian iris, Iris pallida (0)
  - beardless iris (4)
  - bulbous iris (0)
  - dwarf iris, iris cristata (0)
  - stinking iris, gladdon, gladdon iris, stinking gladwyn, Persian iris, Iris persica (0)
  - yellow iris, yellow flag, yellow water flag, Iris pseudacorus (0)
  - dwarf iris, vernal iris, Iris verna (0)
  - blue flag, Iris versicolor (0)
Court, courtyard

An area wholly or partly surrounded by walls or buildings; "the house was built around an inner court."

- ImageNet 2011 Fall Release (32326)
  - plant, flora, plant life (4486)
  - geological formation, formation (175)
  - natural object (1112)
  - sport, athletics (176)
  - artifact, artefact (10504)
    - instrumentality, instrumentation (5494)
  - structure, construction (1405)
    - airdock, hangar, repair shed (0)
    - altar (1)
    - arcade, colonnade (1)
    - arch (31)
    - area (344)
      - aisle (0)
      - auditorium (1)
      - baggage claim (0)
      - box (1)
      - breakfast area, breakfast nook (0)
      - bullpen (0)
      - chancel, sanctuary, bema (0)
      - choir (0)
      - corner, nook (2)
      - court, courtyard (6)
        - atrium (0)
        - bailey (0)
        - cloister (0)
        - food court (0)
        - forecourt (0)
        - narthex (0)
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each.

4 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random $224 \times 224$ patches (and their horizontal reflections) from the $256 \times 256$ images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five $224 \times 224$ patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components.

This is the reason why the input images in Figure 2 are $224 \times 224 \times 3$-dimensional.
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

**4 Reducing Overfitting**

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

**4.1 Data Augmentation**

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random $224 \times 224$ patches (and their horizontal reflections) from the $256 \times 256$ images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five $224 \times 224$ patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network’s softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components.

---

**Example: Image Classification**

**CNN for Image Classification**
(Krizhevsky, Sutskever & Hinton, 2011)
17.5% error on ImageNet LSVRC-2010 contest

- Input image (pixels)
- 5 convolutional layers (w/max-pooling)
- Three fully connected layers
- 1000-way softmax

The rest is *just* some fancy feature extraction (discussed later in the course)

This “softmax” layer is Logistic Regression!
LOGISTIC REGRESSION
Logistic Regression

**Data:** Inputs are continuous vectors of length M. Outputs are discrete.

\[ D = \{ x^{(i)}, y^{(i)} \}_{i=1}^{N} \] where \( x \in \mathbb{R}^{M} \) and \( y \in \{0, 1\} \)

We are back to classification.

Despite the name logistic **regression**.
Linear Models for Classification

Key idea: Try to learn this hyperplane directly

Directly modeling the hyperplane would use a decision function:

\[ h(x) = \text{sign}(\theta^T x) \]

for:

\[ y \in \{-1, +1\} \]

Looking ahead:
- We’ll see a number of commonly used Linear Classifiers
- These include:
  - Perceptron
  - Logistic Regression
  - Naïve Bayes (under certain conditions)
  - Support Vector Machines
Background: Hyperplanes

**Notation Trick**: fold the bias $b$ and the weights $\mathbf{w}$ into a single vector $\mathbf{\theta}$ by prepending a constant to $\mathbf{x}$ and increasing dimensionality by one!

Hyperplane (Definition 1):
$$\mathcal{H} = \{ \mathbf{x} : \mathbf{w}^T \mathbf{x} = b \}$$

Hyperplane (Definition 2):
$$\mathcal{H} = \{ \mathbf{x} : \mathbf{\theta}^T \mathbf{x} = 0 \text{ and } x_0 = 1 \}$$
$$\mathbf{\theta} = [b, w_1, \ldots, w_M]^T$$

Half-spaces:
$$\mathcal{H}^+ = \{ \mathbf{x} : \mathbf{\theta}^T \mathbf{x} > 0 \text{ and } x_0 = 1 \}$$
$$\mathcal{H}^- = \{ \mathbf{x} : \mathbf{\theta}^T \mathbf{x} < 0 \text{ and } x_0 = 1 \}$$

Recall...
Using gradient ascent for linear classifiers

Key idea behind today’s lecture:

1. Define a linear classifier (logistic regression)
2. Define an objective function (likelihood)
3. Optimize it with gradient descent to learn parameters
4. Predict the class with highest probability under the model
Using gradient ascent for linear classifiers

This decision function isn’t differentiable:

\[ h(x) = \text{sign}(\theta^T x) \]

Use a differentiable function instead:

\[ p_\theta(y = 1|x) = \frac{1}{1 + \exp(-\theta^T x)} \]

\[ \text{logistic}(u) \equiv \frac{1}{1 + e^{-u}} \]
Using gradient ascent for linear classifiers

This decision function isn’t differentiable:

\[ h(x) = \text{sign}(\theta^T x) \]

Use a differentiable function instead:

\[ p_\theta(y = 1|x) = \frac{1}{1 + e^{-\theta^T x}} \]

logistic\((u) \equiv \frac{1}{1 + e^{-u}}\)
Logistic Regression

Data: Inputs are continuous vectors of length M. Outputs are discrete.
\[ \mathcal{D} = \{ \mathbf{x}^{(i)}, y^{(i)} \}_{i=1}^{N} \text{ where } \mathbf{x} \in \mathbb{R}^{M} \text{ and } y \in \{0, 1\} \]

Model: Logistic function applied to dot product of parameters with input vector.
\[ p_{\theta}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\theta^{T}\mathbf{x})} \]

Learning: finds the parameters that minimize some objective function.
\[ \theta^* = \arg\min_{\theta} J(\theta) \]

Prediction: Output is the most probable class.
\[ \hat{y} = \arg\max_{y \in \{0,1\}} p_{\theta}(y|\mathbf{x}) \]
Logistic Regression

Whiteboard

- Bernoulli interpretation
- Logistic Regression Model
- Decision boundary
Learning for Logistic Regression

Whiteboard

– Partial derivative for Logistic Regression
– Gradient for Logistic Regression
Logistic Regression
Logistic Regression

Logistic Regression Distribution
LEARNING LOGISTIC REGRESSION
**Maximum Conditional Likelihood Estimation**

**Learning:** finds the parameters that minimize some objective function.

\[ \theta^* = \arg\min_{\theta} J(\theta) \]

We minimize the **negative** log conditional likelihood:

\[ J(\theta) = -\log \prod_{i=1}^{N} p_{\theta}(y^{(i)}|x^{(i)}) \]

Why?

1. We can’t maximize likelihood (as in Naïve Bayes) because we don’t have a joint model \( p(x,y) \)
2. It worked well for Linear Regression (least squares is MCLE)
Maximum Conditional Likelihood Estimation

**Learning:** Four approaches to solving $\theta^* = \arg\min_{\theta} J(\theta)$

**Approach 1:** Gradient Descent
(take larger – more certain – steps opposite the gradient)

**Approach 2:** Stochastic Gradient Descent (SGD)
(take many small steps opposite the gradient)

**Approach 3:** Newton’s Method
(use second derivatives to better follow curvature)

**Approach 4:** Closed Form???
(set derivatives equal to zero and solve for parameters)
Maximum Conditional Likelihood Estimation

Learning: Four approaches to solving $\theta^* = \arg\min_{\theta} J(\theta)$

**Approach 1:** Gradient Descent
(take larger – more certain – steps opposite the gradient)

**Approach 2:** Stochastic Gradient Descent (SGD)
(take many small steps opposite the gradient)

**Approach 3:** Newton’s Method
(use second derivatives to better follow curvature)

**Approach 4:** Closed Form???
(set derivatives equal to zero and solve for parameters)

Logistic Regression does not have a closed form solution for MLE parameters.
SGD for Logistic Regression

Question:
Which of the following is a correct description of SGD for Logistic Regression?

Answer:
At each step (i.e. iteration) of SGD for Logistic Regression we...

A. (1) compute the gradient of the log-likelihood for all examples (2) update all the parameters using the gradient

B. (1) ask Matt for a description of SGD for Logistic Regression, (2) write it down, (3) report that answer

C. (1) compute the gradient of the log-likelihood for all examples (2) randomly pick an example (3) update only the parameters for that example

D. (1) randomly pick a parameter, (2) compute the partial derivative of the log-likelihood with respect to that parameter, (3) update that parameter for all examples

E. (1) randomly pick an example, (2) compute the gradient of the log-likelihood for that example, (3) update all the parameters using that gradient

F. (1) randomly pick a parameter and an example, (2) compute the gradient of the log-likelihood for that example with respect to that parameter, (3) update that parameter using that gradient
In order to apply GD to Logistic Regression all we need is the gradient of the objective function (i.e. vector of partial derivatives).
Stochastic Gradient Descent (SGD)

We can also apply SGD to solve the MCLE problem for Logistic Regression.

We need a per-example objective:

Let \( J(\theta) = \sum_{i=1}^{N} J^{(i)}(\theta) \)

where \( J^{(i)}(\theta) = -\log p_\theta(y^i|x^i) \).
Mini-Batch SGD

• Gradient Descent:
  Compute true gradient exactly from all N examples

• Mini-Batch SGD:
  Approximate true gradient by the average gradient of K randomly chosen examples

• Stochastic Gradient Descent (SGD):
  Approximate true gradient by the gradient of one randomly chosen example
Mini-Batch SGD

while not converged: \( \theta \leftarrow \theta - \lambda g \)

Three variants of first-order optimization:

Gradient Descent: \( g = \nabla J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla J^{(i)}(\theta) \)

SGD: \( g = \nabla J^{(i)}(\theta) \) where \( i \) sampled uniformly

Mini-batch SGD: \( g = \frac{1}{S} \sum_{s=1}^{S} \nabla J^{(i_s)}(\theta) \) where \( i_s \) sampled uniformly \( \forall s \)
Question:
True or False: Just like Perceptron, one step (i.e. iteration) of SGD for Logistic Regression will result in a change to the parameters only if the current example is incorrectly classified.

Answer:
Summary

1. Discriminative classifiers directly model the **conditional**, $p(y|x)$

2. Logistic regression is a **simple linear classifier**, that retains a **probabilistic semantics**

3. Parameters in LR are learned by **iterative optimization** (e.g. SGD)
Logistic Regression Objectives

You should be able to...

• Apply the principle of maximum likelihood estimation (MLE) to learn the parameters of a probabilistic model
• Given a discriminative probabilistic model, derive the conditional log-likelihood, its gradient, and the corresponding Bayes Classifier
• Explain the practical reasons why we work with the log of the likelihood
• Implement logistic regression for binary or multiclass classification
• Prove that the decision boundary of binary logistic regression is linear
• For linear regression, show that the parameters which minimize squared error are equivalent to those that maximize conditional likelihood