Last discussion! (except for final review discussion)

CS 188 Discussion 12:

Logistic Regression, Neural Networks

Kenny Wang (kwkw@berkeley.edu)
Wed Nov 15, 2023

Slides based on Sashrika + Joy

Administrivia

- Project 4 deadline extended Mon, Nov 6 => Fri, Nov 10
- Homework is due on Tuesdays
- We have office hours pretty much all day every weekday (12-7), come to Soda 341B! (my hours are 1-3 PM on Mondays)
- Discussion slides are on Ed

Today’s Topics

- Sigmoid Function
- Logistic Regression
- Gradient Descent
- Neural Networks!!

Sigmoid Function

mildly important in CS 189!

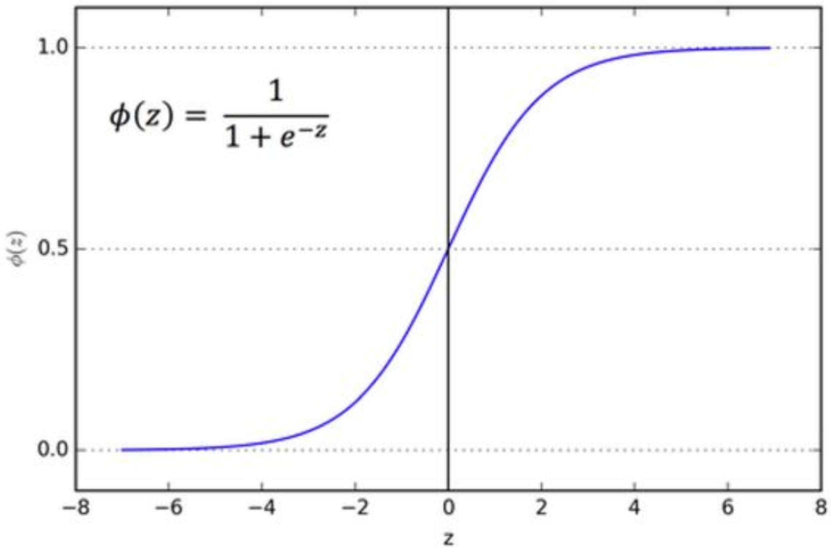
- **Sigmoid (aka Logistic) function:**

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

- All outputs bounded by 0 and 1, is 0.5 when z is 0

- Can convert linear outputs to probabilities (logistic regression)

- Used to introduce nonlinearity in neural networks



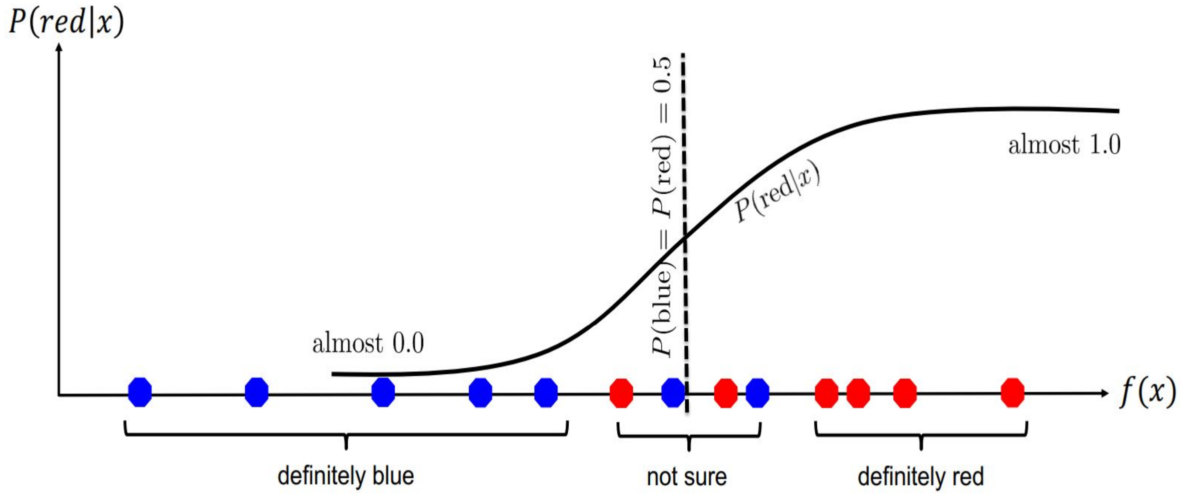
Logistic Regression

- **Idea:** Instead of simply using w^Tx, apply sigmoid function on w^Tx

- **Logistic Regression**

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

- Can predict probabilities, unlike binary perceptron



- How to compute **w**? Use gradient descent!
- See also: multiclass logistic regression, using the **softmax function**

Optimization: Gradient Ascent/Descent

- **Goal:** Want to find the parameters that maximize **objective function** or minimize **loss function**
- If closed-form formula for global optimum does not exist, can use **gradient ascent/descent**
- Observation: gradient is direction of steepest increase … by repeatedly following the gradient, we can chase maxima/minima

- **Gradient Ascent**

Randomly initialize $\mathbf{w}$
while $\mathbf{w}$ *not converged* **do**
| $\mathbf{w} \leftarrow \mathbf{w} + \alpha \nabla_{\mathbf{w}} f(\mathbf{w})$
end

- **Gradient Descent**

Randomly initialize $\mathbf{w}$
while $\mathbf{w}$ *not converged* **do**
| $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} f(\mathbf{w})$
end

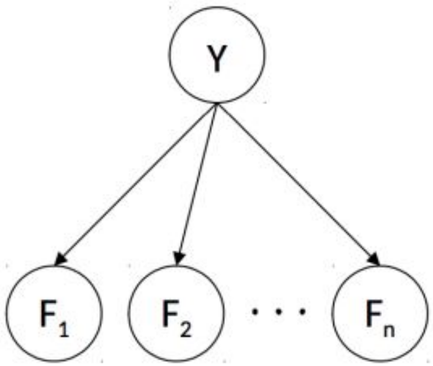
Learning rate α is a hyperparameter

Musheen Lurning!!1

- **Core Idea:** We give machines access to data and they learn for themselves!
- Data is often split into training, validation, and test sets
  - **Training set:** Used to fit the model
  - **Validation set:** Used to tune **hyperparameters** (learning rate, model structure, etc)
  - **Test set:** Used to test the entire model
- Some types of machine learning problems
  - **Classification problems:** try to classify data into discrete classes
  - **Regression problems:** try to estimate some numerical value from data
  - **Clustering problems:** try to group similar data into clusters
- Types of learning
  - **Supervised learning:** training data has labels, e.g. classification
  - **Unsupervised learning:** training data has no labels, e.g. clustering

Naive Bayes

- **Goal:** create a model that can predict a label Y given features, where we assume all features are independently affected by the label
- Ex: Spam filter
  - Y is in {Spam, Ham}
  - F_i in {0, 1} is whether word i appears in the email.
- Label email based on the higher of these two probabilities:



$$P(Y = ham | F_1 = f_1, \ldots, F_n = f_n) \qquad P(Y = spam | F_1 = f_1, \ldots, F_n = f_n)$$

- Generalized:

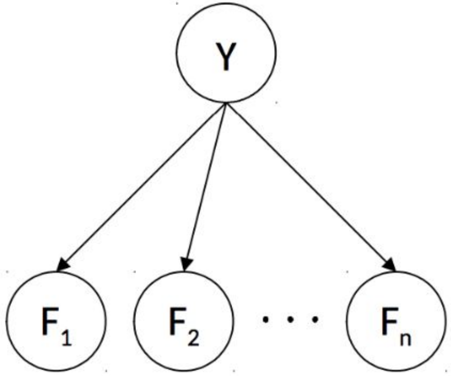
$$\begin{aligned} prediction(f_1, \cdots f_n) &= \underset{y}{\operatorname{argmax}}\, P(Y = y | F_1 = f_1, \ldots F_N = f_n) \\ &= \underset{y}{\operatorname{argmax}}\, P(Y = y, F_1 = f_1, \ldots F_N = f_n) \\ &= \underset{y}{\operatorname{argmax}}\, P(Y = y) \prod_{i=1}^{n} P(F_i = f_i | Y = y) \end{aligned}$$

Maximum Likelihood Estimation

- How to estimate CPTs, since we don’t actually know them?
  - Parameter estimation with MLE
- Find the probabilities (CPT values) $\boldsymbol{\theta}$ **= P(·)** such that we maximize the likelihood of observing our observations, P(observations | θ)
- Answer is actually fairly intuitive. f/e, given data (F, Y),

P(Y=y) = MLE(θ | (F,Y))
= (# examples with Y=y)
(total # examples)

P(F_i=f | Y=y) = MLE(θ | (F,Y))
= (# examples with (F_i=f, Y=y))
(# examples with Y=y)



Laplace Smoothing **Worksheet Q1**

- Chance of **overfitting** (model doesn't generalize well post-training) with our parameter estimation
- **Laplace Smoothing:** pretend you saw each of the |X| possible outcomes **k** extra times

$$P_{LAP,k}(x) = \frac{count(x) + k}{N + k|X|} \qquad P_{LAP,k}(x|y) = \frac{count(x,y) + k}{count(y) + k|X|}$$

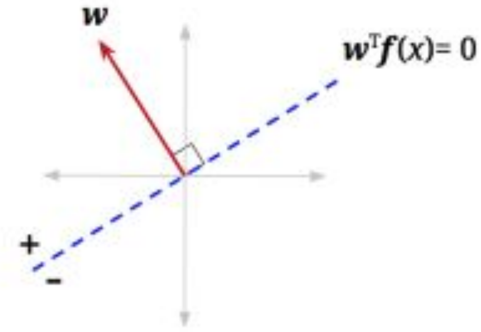
- **k** is a **hyperparameter**, meaning you can choose what to set it to
  - Smaller k means your probability estimates follow the training data more closely
  - Larger k means your probability estimates are more uniform

Perceptrons

- Naive Bayes
- **Perceptrons**

Binary Perceptrons

- Idea: Linearly separate data into two classes using a decision boundary (defined by a set of weights)
- If the data is linearly separable, the algorithm will perfectly classify the data!
- To find boundaries that don't have to cross the origin, incorporate a "bias" feature that always has value 1



Decision Boundary

Perceptron Algorithm

Worksheet Q2

1. Initialize all weights to 0: $\mathbf{w} = \mathbf{0}$

2. For each training sample, with features $\mathbf{f}(x)$ and true class label $y^* \in \{-1, +1\}$, do:

   (a) Classify the sample using the current weights, let y be the class predicted by your current $\mathbf{w}$:

   $$y = \text{classify}(x) = \begin{cases} +1 & \text{if activation}_w(x) = \mathbf{w}^T\mathbf{f}(x) > 0 \\ -1 & \text{if activation}_w(x) = \mathbf{w}^T\mathbf{f}(x) < 0 \end{cases}$$

   (b) Compare the predicted label y to the true label y^*:

   - If $y = y^*$, do nothing
   - Otherwise, if $y \neq y^*$, then update your weights: $\mathbf{w} \leftarrow \mathbf{w} + y^*\mathbf{f}(x)$

3. If you went through **every** training sample without having to update your weights (all samples predicted correctly), then terminate. Else, repeat step 2.

Rest of the Worksheet

Thank you for attending!

Attendance link:

- https://tinyurl.com/cs188fa23

Discussion No: 11

Remember my name is Kenny

My email: kwkw@berkeley.edu