Computing Probabilities from Data

- Various probabilities you will need to compute for Naive Bayesian Classifier (using MLE here):

$$\hat{P}(Y=0) = \frac{\#\text{instances in class} = 0}{\text{total}\,\#\,\text{instances}}$$

$$\hat{P}(X_i = 0, Y = 0) = \frac{\#\text{instances where } X_i = 0 \text{ and class} = 0}{\text{total}\,\#\,\text{instances}}$$

$$\hat{P}(X_i = 0 \mid Y = 0) = \frac{\hat{P}(X_i = 0, Y = 0)}{\hat{P}(Y=0)} \qquad \hat{P}(X_i = 0 \mid Y = 1) = \frac{\hat{P}(X_i = 0, Y = 1)}{\hat{P}(Y=1)}$$

$$\hat{P}(X_i = 1 \mid Y = 0) = 1 - \hat{P}(X_i = 0 \mid Y = 0)$$

$$\hat{y} = \arg\max_y P(\boldsymbol{X} \mid Y)P(Y) = \arg\max_y (\log[P(\boldsymbol{X} \mid Y)P(Y)])$$

$$\log P(\boldsymbol{X} \mid Y) = \log P(X_1, X_2, \ldots X_m \mid Y) = \log \prod_{i=1}^{m} P(X_i \mid Y) = \sum_{i=1}^{m} \log P(X_i \mid Y)$$

From Naive Bayes to Logistic Regression

- Recall the Naive Bayes Classifier
  - Predict $\hat{Y} = \arg\max_y P(\boldsymbol{X}, Y) = \arg\max_y P(\boldsymbol{X} \mid Y)P(Y)$
  - Use assumption that $P(\boldsymbol{X} \mid Y) = P(X_1, X_2, \ldots X_m \mid Y) = \prod_{i=1}^{m} P(X_i \mid Y)$
  - We are really modeling joint probability P(**X**, Y)
- But for classification, really care about P(Y | **X**)
  - Really want to predict $\hat{y} = \arg\max_y P(Y \mid \boldsymbol{X})$
  - Modeling full joint probability P(**X**, Y) is just proxy for this
- So, how do we model P(Y | **X**) directly?
  - Welcome our friend: logistic regression!

Logistic Regression

- Model *conditional* likelihood P(Y | **X**) directly
  - Model this probability with *logistic* function:

$$P(Y=1 \mid \boldsymbol{X}) = \frac{1}{1+e^{-z}} \quad \text{where } z = \alpha + \sum_{j=1}^{m} \beta_j X_j$$

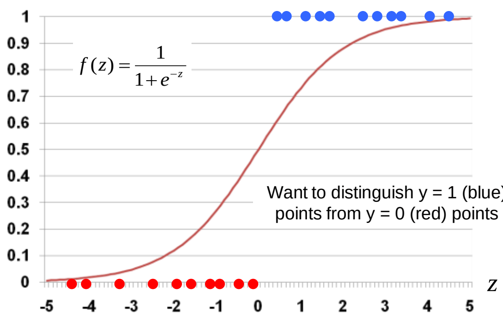
  - For simplicity define $X_0 = 1$ and $\beta_0 = \alpha$, so $z = \sum_{j=0}^{m} \beta_j X_j$
  - Since P(Y = 0 | **X**) + P(Y = 1 | **X**) = 1, we obtain:

$$P(Y=0 \mid \boldsymbol{X}) = \frac{e^{-z}}{1+e^{-z}} \quad \text{where } z = \sum_{j=0}^{m} \beta_j X_j$$

  - Note: log-odds is **linear** function of inputs X_j:

$$\log \frac{P(Y=1 \mid \boldsymbol{X})}{P(Y=0 \mid \boldsymbol{X})} = \log \frac{1}{e^{-z}} = \log e^{z} = z = \sum_{j=0}^{m} \beta_j X_j$$

The Logistic Function



Note: inflection point at z = 0. *f*(0) = 0.5

Learning Logistic Regression Function

- Can write log-conditional likelihood of data as:

$$LL(\theta) = \sum_{i=1}^{n} [y_i \log P(Y=1 \mid \boldsymbol{X}) + (1-y_i)\log P(Y=0 \mid \boldsymbol{X})]$$

where $P(Y=1 \mid \boldsymbol{X}) = \frac{1}{1+e^{-z}}$, $P(Y=0 \mid \boldsymbol{X}) = \frac{e^{-z}}{1+e^{-z}}$ with $z = \sum_{j=0}^{m} \beta_j X_j$

  - No analytic derivation of MLE parameters for β_j
    - But, log-conditional likelihood function is *concave*
    - Has a single global maximum (good times)
  - Compute gradient of $LL(\theta)$ w.r.t. β_j where $0 \le j \le m$:

$$\frac{\partial LL(\theta)}{\partial \beta_j} = x_j\left(y - \frac{1}{1+e^{-z}}\right) = x_j(y - P(Y=1 \mid \boldsymbol{X}))$$

  - Maximize $LL(\theta)$: iteratively update β_j using gradient:

$$\beta_j^{new} = \beta_j^{old} + c\, x_j\left(y - \frac{1}{1+e^{-z}}\right) \quad \text{where } z = \sum_{j=0}^{m} \beta_j^{old} x_j \text{ and } c = \text{constant}$$

Wanna See How We Computed Gradient?

- Log-conditional likelihood of data point *i*, $LL_i(\theta)$:

$$LL_i(\theta) = y_i \log P(Y=1 \mid \boldsymbol{X}) + (1-y_i)\log P(Y=0 \mid \boldsymbol{X})$$

  - Rearrange terms:

$$LL_i(\theta) = y_i \log \frac{P(Y=1 \mid \boldsymbol{X})}{P(Y=0 \mid \boldsymbol{X})} + \log P(Y=0 \mid \boldsymbol{X})$$

  - Substitute values for P(Y | **X**) and simplify:

$$LL_i(\theta) = y_i \sum_{j=0}^{m} \beta_j X_j + \log e^{-z} - \log(1+e^{-z})$$

$$= y_i \sum_{j=0}^{m} \beta_j X_j - \sum_{j=0}^{m} \beta_j X_j - \log(1+e^{-z})$$

  - Compute gradient of $LL(\theta)$ w.r.t. β_j where $0 \le j \le m$:

$$\frac{\partial LL_i(\theta)}{\partial \beta_j} = yx_j - x_j + \frac{x_j e^{-z}}{1+e^{-z}} = x_j\left(y - \frac{1+e^{-z}}{1+e^{-z}} + \frac{e^{-z}}{1+e^{-z}}\right) = x_j\left(y - \frac{1}{1+e^{-z}}\right)$$

“Batch” Logistic Regression Algorithm

```
Initialize: β_j = 0 for all 0 ≤ j ≤ m
// "epochs" = number of passes over data during learning
for (i = 0; i < epochs; i++) {
   Initialize: gradient[j] = 0 for all 0 ≤ j ≤ m
   // Compute "batch" gradient vector
   for each training instance (<x_1, x_2, …, x_m>, y) in data {
      // Add contribution to gradient for each data point
      for (j = 0; j <= m; j++) {
         // Note: x_j below is j-th input variable and x_0 = 1.
         gradient[j] += x_j(y - 1/(1+e^(-z))) where z = Σ_{j=0}^{m} β_j x_j
      }
   }
   // Update all β_j.  Note learning rate η is pre-set constant
   β_j += η * gradient[j] for all 0 ≤ j ≤ m
}
```

Classification with Logistic Regression

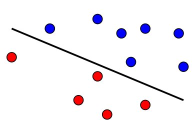
- Training: determine parameters β_j (for all $0 \le j \le m$)
  - After parameters β_j have been learned, test classifier
- To test classifier, for each new (test) instance $\boldsymbol{X}$:
  - Compute: $p = P(Y=1 \mid \boldsymbol{X}) = \dfrac{1}{1+e^{-z}}$, where $z = \sum_{j=0}^{m} \beta_j X_j$
  - Classify instance as: $\hat{y} = \begin{cases} 1 & p > 0.5 \\ 0 & \text{otherwise} \end{cases}$
  - Note about evaluation set-up: parameters β_j are **not** updated during “testing” phase
  - In real systems, model parameters are updated as new data becomes available (on a periodic basis)

Linear Separability

- Recall that log-odds is **<u>linear</u>** function of inputs X_j:

$$\log \frac{P(Y=1 \mid \boldsymbol{X})}{P(Y=0 \mid \boldsymbol{X})} = \sum_{j=0}^{m} \beta_j X_j$$

  - Logistic regression is trying to fit a **<u>line</u>** that separates data instances where $y = 1$ from those where $y = 0$



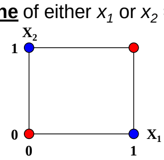
  - We call such data (or the functions generating the data) “**<u>linearly separable</u>**”

Wouldn’t You Like to Be Linear Too?

- Logistic Regression as a linear function
  - As mentioned $\log \frac{P(Y=1 \mid \boldsymbol{X})}{P(Y=0 \mid \boldsymbol{X})} = \sum_{j=0}^{m} \beta_j X_j$ is linear in inputs X_j
  - That is, each X_j is multiplied by separate β_j
  - No direct interaction (multiplication) of multiple X_j
- Such linearity is also true for Naïve Bayes
  - You will show it on the next problem set
    - It’s pretty straight-forward (pay attention to the top of this slide)
  - So, Logistic Regression and Naive Bayes have same functional form!
    - Each is just maximizing a different objective function:
      Naive Bayes: P(**X**, Y)   Logistic Regression: P(Y | **X**)

Data Often Not Linearly Separable

- Many data sets/functions are not linearly separable
  - Consider function: $y = x_1$ XOR x_2
  - Note: y =1 iff **<u>one</u>** of either x_1 or x_2 = 1



  - Not possible to draw a line that successfully separates all the y = 1 points (blue) from the y = 0 points (red)
  - Despite this fact, logistic regression and Naive Bayes still often work well in practice

Logistic Regression vs. Naïve Bayes

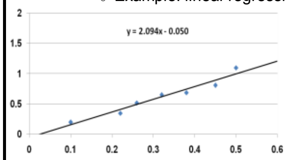
- Compare Naive Bayes and Logistic Regression
  - Recall that Naive Bayes models P(**X**, Y) = P(**X** | Y) P(Y)
  - Logistic Regression directly models P(Y | **X**)
  - We call Naive Bayes a “generative model”
    - Tries to model joint distribution of how data is “generated”
    - I.e., could use P(X, Y) to generate new data points if we wanted
    - But lots of effort to model something that may not be needed
  - We call Logistic Regression a “discriminative model”
    - Just tries to model way to discriminate y = 0 vs. y = 1 cases
    - Cannot use model to generate new data points (no P(X, Y))
    - Note: Logistic Regression can be generalized to more than two output values for y (have multiple sets of parameters β_j)

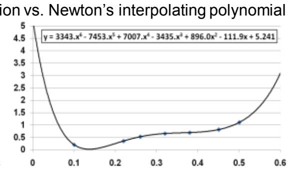
Choosing an Algorithm

- Many trade-offs in choosing learning algorithm
  - Continuous input variables
    - Logistic Regression easily deals with continuous inputs
    - Naive Bayes needs to use some parametric form for continuous inputs (e.g., Gaussian) or "discretize" continuous values into ranges (e.g., temperature in range: <50, 50-60, 60-70, >70)
  - Discrete input variables
    - Naive Bayes naturally handles multi-valued discrete data by using multinomial distribution for $P(X_i \mid Y)$
    - Logistic Regression requires some sort of representation of multi-valued discrete data (e.g., multiple binary features)
    - Say $X_i \in \{A, B, C\}$. Not necessarily a good idea to encode X_i as taking on input values 1, 2, or 3 corresponding to A, B, or C.

Good Machine Learning = Generalization

- Goal of machine learning: build models that ***generalize*** well to predicting new data
  - "Overfitting": fitting the training data too well, so we lose generality of model
    - Example: linear regression vs. Newton's interpolating polynomial





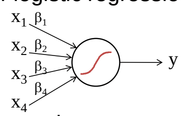
    - Interpolating polynomial fits training data perfectly!
    - Which would you rather use to predict a new data point?

Issues with Logistic Regression

- Logistic Regression can more easily overfit training data than Naive Bayes
  - Logistic Regression is not modeling whole distribution, it is just optimizing prediction of Y
  - Overfitting can be especially problematic if distributions of training data and testing data differ a bit
  - There are methods to mitigate overfitting in Logistic Regression
    - Use Bayesian priors on parameters β_i rather than just maximizing conditional likelihood
    - Intuitively, analogous to using priors in Naive Bayes to get MAP estimates of probabilities
    - But optimization process is more complex in Logistic Regression since conditional likelihood has no analytic solution

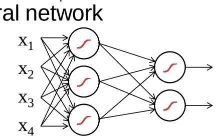
Logistic Regression and Neural Networks

- Consider logistic regression as:



Logistic regression is same as a one node neural network

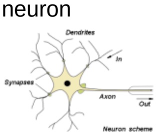
- Neural network

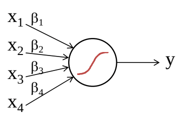


  - Neural network uses "back-propagation" algorithm
    - Like Logistic Regression optimization on "steroids"
    - With enough nodes, can approximate *any* function

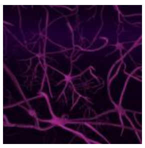
Biological Basis for Neural Networks

- A neuron

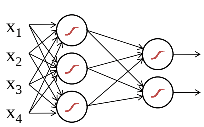




- Your brain



Actually, it's probably someone else's brain



Now You Too Can Build Terminators!

- Be careful! ☺

"My CPU is a neural net processor, a learning computer. The more contact I have with humans, the more I learn."