Tufts

Class #08:
Logistic Regression

Machine Learning (COMP 135)

1

Reminder: Threshold Functions

1. We have data-points with n features:
$$\mathbf{x} = (x_1, x_2, \ldots, x_n)$$
2. We have a linear function defined by $n+1$ weights:
$$\mathbf{w} = (w_0, w_1, w_2, \ldots, w_n)$$
3. We can write this linear function as:
$$\mathbf{w} \cdot \mathbf{x}$$
4. We can then find the linear boundary, where:
$$\mathbf{w} \cdot \mathbf{x} = 0$$
5. And use it to define our threshold between classes:
$$h_{\mathbf{w}} = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \mathbf{w} \cdot \mathbf{x} < 0 \end{cases}$$

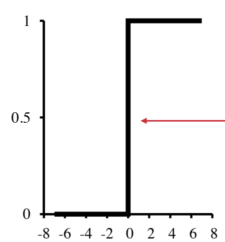
Outputs 1 and 0 here are ***arbitrary labels*** for one of two possible classes

2

Hard Thresholds are Hard!

$$h_{\mathbf{w}} = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \mathbf{w} \cdot \mathbf{x} < 0 \end{cases}$$

- The hard threshold function used by the perceptron algorithm (among others) produces some conceptual and mathematical challenges
- Gives a yes/no answer everywhere, which can be tricky when our data isn't linearly separable



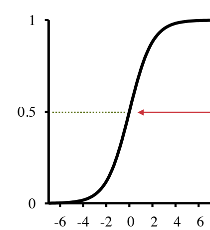
Function is discontinuous (non-differentiable) at $x = 0$

3

The Logistic Function

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

- We can generate a smooth curve by instead using the logistic function as a threshold
- We can treat this value as a ***probability*** of belonging to one class or another



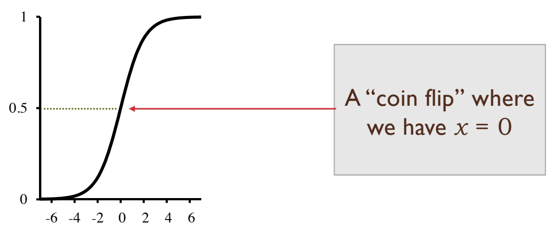
Probability function is 0.5 at $x = 0$

4

Using the Logistic for Classification

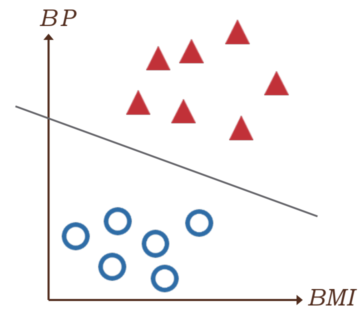
$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}}}$$

- Treated as a probability, the logistic can still be used to *classify* data, where the class is the one that has highest probability overall, while also supplying a probability for that outcome



5

Issues with Linear Classification



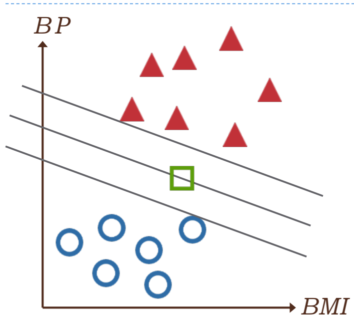
- Consider data about heart-attack risk, based upon body mass index (BMI) and blood pressure (BP)
- Even assuming linearly separable training data, linear classification gives a hard cut-off that may not be appropriate

heart attack

no heart attack

6

Issues with Linear Classification



- Given that ***multiple*** possible lines can separate this data, how do we classify a new instance when it lies in the region between the training instances?

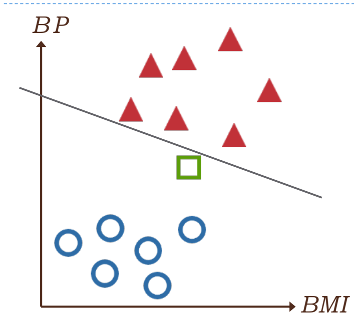
heart attack

don't know?

no heart attack

7

Issues with Linear Classification



- Even if we did settle on some fixed line, what do we do with something that is *very close* to the separator?

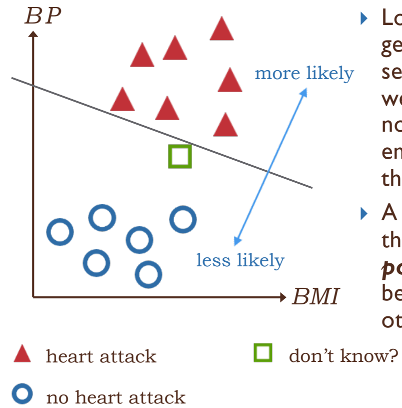
heart attack

don't know?

no heart attack

8

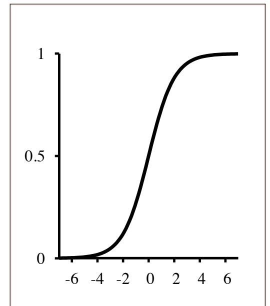
Using Probabilistic Classification



- Logistic regression also generates a linear separator (where the weight-function = 0), but now it is giving us an empirical distribution over the data-set
- A new data point close to the line still has ***some positive probability*** of being in the class on the other side of it

9

Properties of the Logistic Function



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

- Also known as the Sigmoid, from the shape of its plot
- It always has a value in range:
  $0 \leq x \leq 1$
- The function is everywhere differentiable, and has a *derivative* that is easy to calculate, which turns out to be useful for learning:

$$h'_{\mathbf{w}}(\mathbf{x}) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))$$

10

Logistic Regression

- In perceptron learning we update the weight vector in each case based upon a mis-classified instance, using the equation:

$$w_j \leftarrow w_j + \alpha(y_i - h_{\mathbf{w}}(\mathbf{x}_i)) \times x_{i,j}$$

- For the logistic, using the same loss function (squared error), we would do the same, but add an extra term:

$$w_j \leftarrow w_j + \alpha\underbrace{(y_i - h_{\mathbf{w}}(\mathbf{x}_i))}_{\text{The difference between what output } \textit{should} \text{ be, and what our weights make it}} \times \underbrace{h_{\mathbf{w}}(\mathbf{x}_i)(1 - h_{\mathbf{w}}(\mathbf{x}_i))}_{\text{The derivative of the logistic}} \times \underbrace{x_{i,j}}_{\text{The } j\text{th feature-value}}$$

11

Gradient Descent for Logistic Regression, I

- We could then use the same approach as for linear classification, starting with some random (or uniform) weights and then:

1. Choose an input $\mathbf{x}_i$ from our data set that is wrongly classified.
2. Update vector of weights, $\mathbf{w} = (w_0, w_1, w_2, \ldots, w_n)$:

$$w_j \leftarrow w_j + \alpha(y_i - h_{\mathbf{w}}(\mathbf{x}_i)) \times h_{\mathbf{w}}(\mathbf{x}_i)(1 - h_{\mathbf{w}}(\mathbf{x}_i)) \times x_{i,j}$$

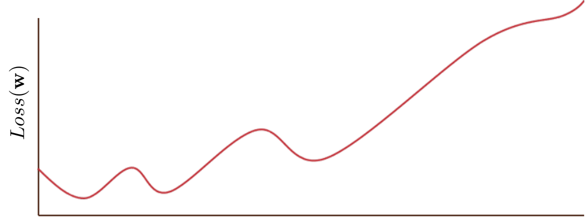
3. Repeat until weights no longer change; modify learning parameter α over time to guarantee this.

- Again, we make α smaller and smaller over time, and the algorithm converges as $\alpha \to 0$

12

A Problem: Local Minima

$$w_j \leftarrow w_j + \alpha(y_i - h_{\mathbf{w}}(\mathbf{x}_i)) \times h_{\mathbf{w}}(\mathbf{x}_i)(1 - h_{\mathbf{w}}(\mathbf{x}_i)) \times x_{i,j}$$



- While this sort of weight update **does** drive the error down, it has a flaw: squared-error loss for logistic regression is ***not*** convex
  - Gradient descent can get stuck in locally optimal solutions that aren't ideal
  - Solution: change the error function!

Gradient Descent for Logistic Regression, II

- We change the weight-update in our gradient descent approach:

1. Choose an input $\mathbf{x}_i$ from our data set that is wrongly classified.
2. Update vector of weights, $\mathbf{w} = (w_0, w_1, w_2, \ldots, w_n)$:

$$\boxed{w_j \leftarrow w_j + \alpha\, x_{i,j}\,(y_i - h_{\mathbf{w}}(\mathbf{x}_i))}$$

3. Repeat until weights no longer change; modify learning parameter α over time to guarantee this.

- Note: the update ***looks*** like the update for linear regression, but:
  1. It only uses a ***single*** incorrect data-point, not the sum of ***all*** errors
  2. The hypothesis h is an application of the logistic function
- The reason we can do this update is that we ***don't*** use the squared-error loss, but use a ***different*** loss function: logistic loss

Gradient Descent for Logistic Regression

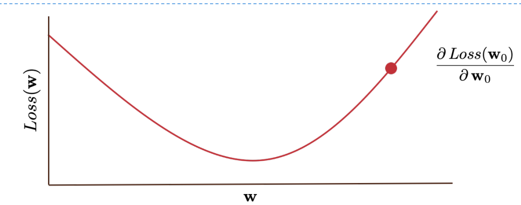
$$w_j \leftarrow w_j + \alpha\, x_{i,j}\,(y_i - h_{\mathbf{w}}(\mathbf{x}_i))$$

- The logistic update equation, via gradient descent, minimizes the log-loss (also known as logistic loss or binary cross entropy):

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log p_i + (1 - y_i)\log(1 - p_i)$$

  - For these purposes, we treat the output of the logistic as the probability we are interested in: $p_i \triangleq h_{\mathbf{w}}(\mathbf{x}_i)$
  - Over time, we drive the loss towards 0

Gradient Descent for Logistic Regression



$$\boxed{\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log p_i + (1 - y_i)\log(1 - p_i)}$$

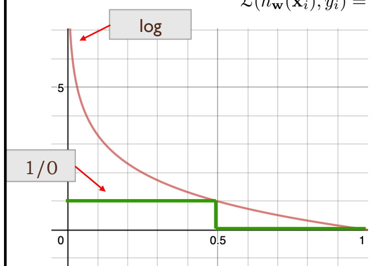
- The log-loss is a convex function
  - Like the losses used in linear regression and the perceptron
- This means that the gradient descent process (with suitable values of α) will converge upon a near-optimal solution

Logarithmic Loss vs. Thresholded Error

- For an individual data element, the log loss is an upper bound on a threshold-based (1/0) loss:

$$\mathcal{E}(h_{\mathbf{w}}(\mathbf{x}_i), y_i) = \begin{cases} 0 & \text{if } h_{\mathbf{w}}(\mathbf{x}_i) = y_i \\ 1 & \text{if } h_{\mathbf{w}}(\mathbf{x}_i) \neq y_i \end{cases}$$

$$\mathcal{L}(h_{\mathbf{w}}(\mathbf{x}_i), y_i) = -[y_i \log h_{\mathbf{w}}(\mathbf{x}_i) + (1 - y_i) \log(1 - h_{\mathbf{w}}(\mathbf{x}_i))]$$



- This graph assumes:
  1. True label is 1
  2. Threshold used is 0.5 (i.e., $h_{\mathbf{w}} = 1$ if probability assigned is $p \geq 0.5$)
  3. Log base 2 is used

17

Linear vs. Logistic Regression

Linear Regression	Logistic Regression
A value $x \in \mathbb{R}$	A value $0 \leq x \leq 1$
Output value of an arbitrary function	Probability of belonging to a certain class
Tries to find line that best ***fits*** to the data	Tries to find separator that best ***divides*** the classes

18

Linear vs. Logistic Regression in Mathematical Terms

Linear	
Loss function	$Loss(\mathbf{w}) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(\mathbf{x}_j))^2$
Weight-update equation	$w_i \leftarrow w_i + \alpha \sum_j x_{j,i} (y_j - h_{\mathbf{w}}(\mathbf{x}_j))$

Logistic	
Loss function	$-\frac{1}{N} \sum_{j=1}^{N} [y_j \log h_{\mathbf{w}}(\mathbf{x}_j) + (1 - y_j) \log(1 - h_{\mathbf{w}}(\mathbf{x}_j))]$
Weight-update equation	$w_i \leftarrow w_i + \alpha\, x_{j,i} (y_j - h_{\mathbf{w}}(\mathbf{x}_j))$

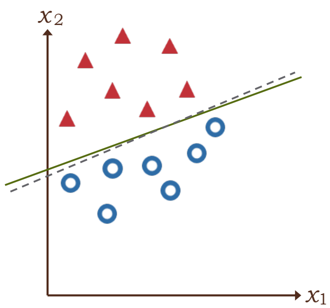
19

Another Approach: ADALINE classifiers

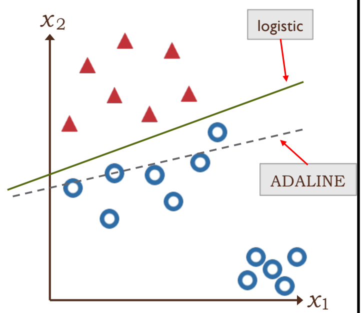
- Rather than a perceptron or logistic approach, what if we tried to use linear regression itself to build a classifier?
- For two classes, we could:
  1. Label data using two class-labels, $y \in \{+1, -1\}$
  2. Fit a linear regression to this data using squared loss (now measured as the difference between the linear value and the class-label, not some other real number) and the same weight-updates as before
  3. Classify data based upon whether the resulting linear function is ≥ 0 (in which case it is assigned $+1$) or not (-1)
- This is known as a least-squares or ADALINE (Adaptive Linear Neuron) classifier

20

Treatment of Outliers in Data



- Logistic regression (solid line) and ADALINE (dashed) give similar results on ***some*** data



- The ADALINE is skewed by outliers, however, as loss function sees them as “too correct”