Logistic Regression

Classification vs. Regression

- In *classification* problems, we use ML algorithms (e.g., *k*NN, decision trees, perceptrons) to predict *discrete*-valued (categorical with no numerical relationship) outputs

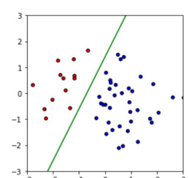
    - Given email, predict ham or spam
    - Given medical info, predict diabetes or not
    - Given tweets, predict positive or negative sentiment
    - Given Titanic passenger info, predict survival or not
    - Given images of handwritten numbers, predict intended digit

- In *regression* problems, we use ML algorithms (e.g., linear regression) to predict *real*-valued outputs

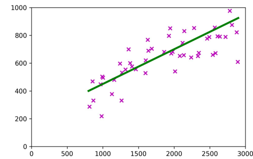
    - Given student info, predict exam scores
    - Given physical attributes, predict age
    - Given medical info, predict blood pressure
    - Given real estate ad, predict housing price
    - Given review text, predict numerical rating

Classification vs. Regression

- In *classification* problems, we use ML algorithms (e.g., *k*NN, decision trees, perceptrons) to predict *discrete*-valued (categorical with no numerical relationship) outputs



- In *regression* problems, we use ML algorithms (e.g., linear regression) to predict *real*-valued outputs

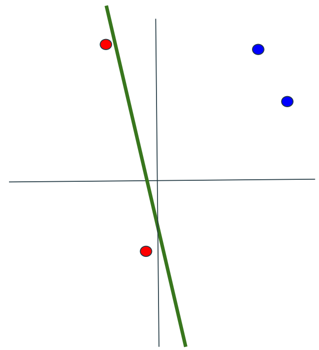


Logistic Regression

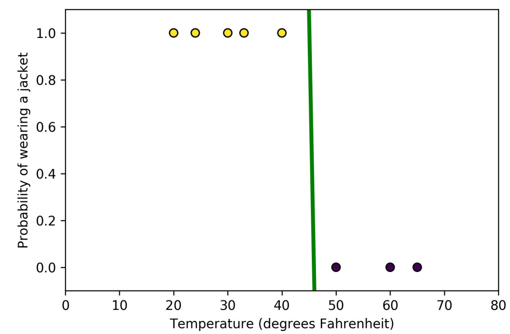
- Logistic regression is used for *classification*, not regression!
- Logistic regression has some commonalities with linear regression, but you should think of it as classification, not regression!
- In many ways, logistic regression is a more advanced version of the perceptron classifier.

Perceptron Limitations

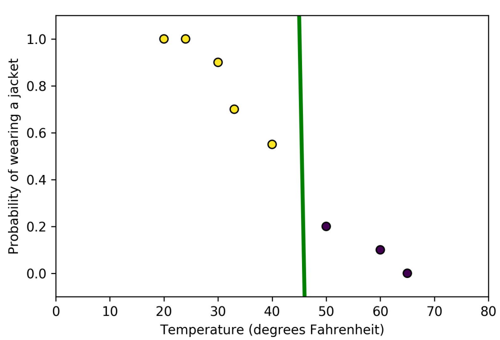
- Perceptron training algorithm finds an arbitrary hyperplane that separates the two classes, not an "optimal" one
- Perceptron predictions have no probabilistic interpretation or confidence estimates
- Perceptron learning algorithm has no principled way of preventing overfitting. Workarounds (e.g., averaged perceptron) are heuristics



Should I wear a jacket?

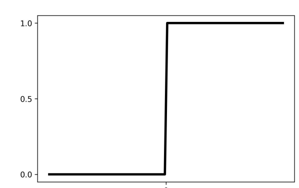


Should I wear a jacket? (softer)



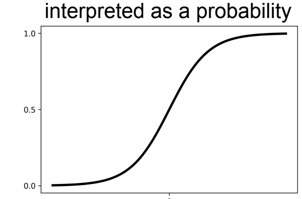
Hard Threshold vs. Sigmoid (Logistic) Function

Returns either 0 or 1



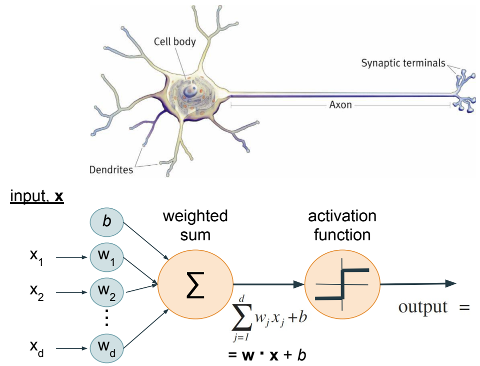
$$g(z) = \begin{cases} 0 \text{ if } & z<0 \\ 1 \text{ if } & z\geq 0 \end{cases}$$

Returns a number between 0.0 and 1.0 that can be interpreted as a probability



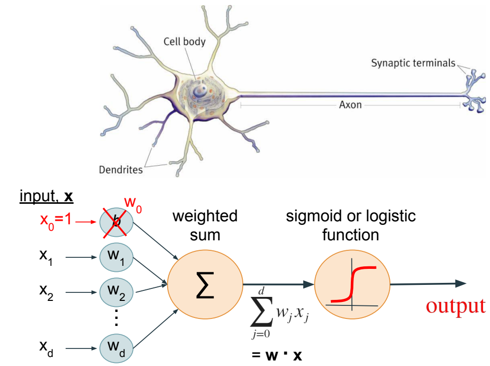
$$g(z) = \frac{1}{1+e^{-z}}$$

Perceptron Motivation



$$\text{output} = \begin{cases} 1 & \text{if } \sum_{j=1}^{d} w_j x_j + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

Logistic Regression Motivation



$$\text{output} = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$$

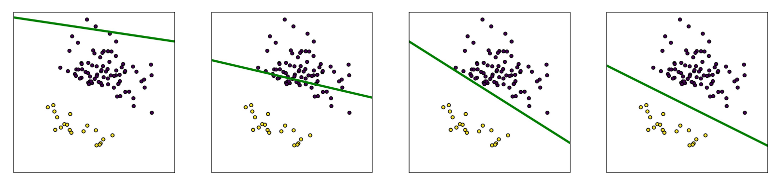
Hypothesis

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

- $h(x)$ is interpreted as the probability that $y = 1$ for input x
- For example, what is the probability that some email message ***x*** is spam (1) as opposed to ham (0)?
  - For a particular set of parameters ***w***, if $h(\boldsymbol{x})$ is 0.25 we would estimate the probability that the message is spam as 25% and classify the message as ham (0)
  - For a particular set of parameters ***w***, if $h(\boldsymbol{x})$ is 0.75 we would estimate the probability that the message is spam as 75% and classify the message as spam (1)

Parameters *w*

Different values for the parameters ***w*** lead to different decision boundaries



We want to quantify the *cost* associated with a given boundary (value settings for ***w***) for our data

Then we can find the values of ***w*** that have the lowest cost

Cost

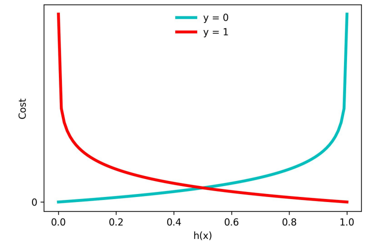
$$J(w) = -\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}\log(h(x^{(i)})) + (1-y^{(i)})\log(1-h(x^{(i)}))\right)$$

Suppose for a given setting of parameters ***w***, we have 4 training data points that:

result in the following hypotheses	have the following classifications
$h(x^{(1)}) = 0.001$	$y^{(1)} = 0$
$h(x^{(2)}) = 0.999$	$y^{(2)} = 0$
$h(x^{(3)}) = 0.001$	$y^{(3)} = 1$
$h(x^{(4)}) = 0.999$	$y^{(4)} = 1$

Cost

$$J(w) = -\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}\log(h(x^{(i)})) + (1-y^{(i)})\log(1-h(x^{(i)}))\right)$$



Gradient Descent

$$J(w) = -\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}\log(h(x^{(i)})) + (1-y^{(i)})\log(1-h(x^{(i)}))\right)$$

We want to find ***w*** that minimizes the cost *J*(***w***).

Repeat (in parallel for each component of ***w***):

$$w_j = w_j - \alpha\frac{\partial}{\partial w_j}J(\mathbf{w})$$

$$= w_j - \alpha\frac{1}{n}\sum_{i=1}^{n}(h(x^{(i)})-y^{(i)})x_j^{(i)}$$

Batch gradient descent

Gradient Descent

$$J(w) = -\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}\log(h(x^{(i)})) + (1-y^{(i)})\log(1-h(x^{(i)}))\right)$$

We want to find ***w*** that minimizes the cost *J*(***w***).

Repeat (in parallel for each component of ***w***), iterating over each data point (*x*, *y*):

$$w_j = w_j - \alpha\frac{\partial}{\partial w_j}J(\mathbf{w})$$

$$= w_j - \alpha(h(x)-y)x$$

Stochastic gradient descent

New Prediction

To make a new prediction, e.g., on a test data point $\boldsymbol{x}$, use the learned model parameters $\boldsymbol{w}$ to output:

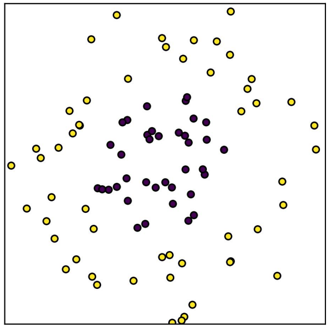
$$h(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{w}\cdot\boldsymbol{x}}}$$

Non-Linear Logistic Regression

Suppose we have data with two features and we don't think the data are linearly separable.

x_1	x_2	y
2	4	1
5	1	0
...	...	...
3	-2	1

$$h(\boldsymbol{w}) = g(w_0+w_1x_1+w_2x_2)$$



Non-Linear Logistic Regression

Suppose we have data with two features and we don't think the data are linearly separable.

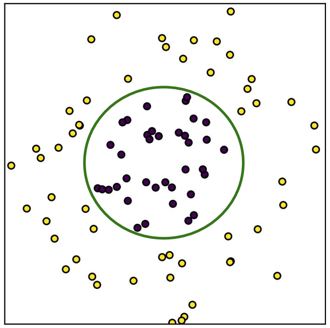
We could add higher order features

x_1	x_2	x_1^2	x_2^2	y
2	4	4	16	1
5	1	25	1	0
...	...	...	...	...
3	-2	9	4	1

$$h(\boldsymbol{w}) = g(w_0+w_1x_1+w_2x_2+w_3x_1^2+w_4x_2^2)$$

Our classifier might learn some $\boldsymbol{w}$=(-1,0,0,1,1), with corresponding decision boundary:

$$-1+0x_1+0x_2+1x_1^2+1x_2^2 = 0 \qquad x_1^2+x_2^2 = 1$$



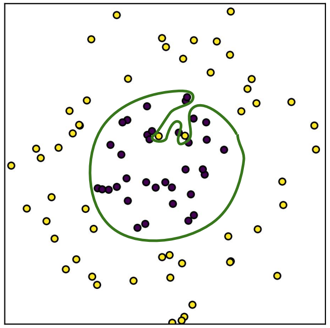
Overfitting

Suppose we have data with two features and we don't think the data are linearly separable.

We could add higher order features

x_1	x_2	y
2	4	1
5	1	0
...	...	...
3	-2	1

$$h(w) = g(w_0+w_1x_1+w_2x_2+w_3x_1^3x_2^2 +w_4x_1^5+w_5x_1^2x_2^4+w_6x_2^9+\dots)$$



Regularized Logistic Regression

- Smaller values for the parameters w_1, w_2, w_3, …, w_d lead to simpler hypotheses that are less prone to overfitting.
- We modify our cost function so that it not only
  - (1) finds a good fitting hypothesis (penalizes error of hypothesis on training data)

  but also
  - (2) considers the complexity of the hypothesis (penalizing more complex hypotheses and favoring simpler hypotheses)

$$J(w) = -\left[\frac{1}{n}\sum_{i=1}^{n}(y^{(i)}\log(h(x^{(i)})) + (1-y^{(i)})\log(1-h(x^{(i)})))\right] + \frac{\lambda}{2n}\sum_{j=1}^{d}w_j^2$$

λ is regularization parameter

Regularized Gradient Descent

$$w_j = w_j - \alpha\frac{\partial}{\partial w_j}J(\mathbf{w})$$

Logistic Regression

$$w_j = w_j - \alpha\frac{1}{n}\sum_{i=1}^{n}(h(x^{(i)})-y^{(i)})x_j^{(i)}$$

Regularized Logistic Regression

$$w_j = w_j - \alpha\left[\frac{1}{n}\sum_{i=1}^{n}(h(x^{(i)})-y^{(i)})x_j^{(i)} + \frac{\lambda}{n}w_j\right]$$

Putting It All Together

- If the data are assumed to be non-linear, add higher order features
- Randomly shuffle the data and split into training, validation, and testing
- Perform feature scaling (in the case where there are multiple features and their range of values is quite different in magnitude) on the training data
- Add a new feature x_0 whose value is always 1, i.e., add a column of ones to the beginning of the data matrix
- Using different hyperparameter settings (e.g., for α and λ):
  - ➢ Train the model, e.g., using *regularized* gradient descent to find the model parameters ***w*** that minimize the cost of the model on the training data while favoring simpler models
  - ➢ Evaluate the model's performance on the (feature scaled) validation data
- Choose the best hyperparameters and gauge the model's performance on new data based on its performance on the (feature scaled) testing data

Multiclass Classification

Song genres:

Blues, Country, Hip Hop, Jazz, Pop, Rock

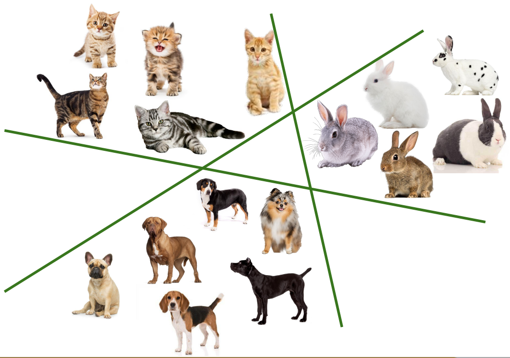
Handwritten digits:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Email labeling:

Family, School, Summer, Friends, CS305

One vs. Rest (One vs. All)



Overview