UNIVERSITY OF SAN FRANCISCO

CHANGE THE WORLD FROM HERE

Logistic Regression

Machine Learning

Motivation

- Can you tell when something is political?
    - Consider two examples from Twitter:

      has iced the sugar cookie pumpkins and gangrenous feet, and is handing out candy to adorable wee goblins. So sweet! All of it.

      RT @ConserValidity: Why do ignorant Progressive Liberals believe Obama/Pelosi Care= reform?Do they know it has nothing to do w/ Health Care?

    - How did you make the decision about what is political?
- Let's first figure out how to make a basic "yes/no" classification

Why Logistic Regression?

- Pros
  - Computationally inexpensive*
  - Relatively easy to implement
  - Easy to interpret
- Cons
  - Prone to underfitting (i.e. low performance)
  - Can misbehave with large numbers of features / variables, especially when not linear
- Other stuff
  - Three types of logistic regression: *binomial* (TRUE/FALSE); *multinomial* (many categories)
  - Uses gradient ascent or a version of it — usually stochastic gradient descent — an iterative optimization algorithm

Transposing Matrices

- Example matrix and its transpose

$$\begin{pmatrix} 5 & 4 \\ 4 & 0 \\ 7 & 10 \\ -1 & 8 \end{pmatrix}^T_{4 \times 2} = \begin{pmatrix} 5 & 4 & 7 & -1 \\ 4 & 0 & 10 & 8 \end{pmatrix}_{2 \times 4}$$

- Quiz:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}^T_{2 \times 3} = \ ?$$

Multiplying Matrices

- Example matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

- Scalar multiplication ($y = 3$)
- Vector multiplication

$$y = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$$

- Multiplication with another matrix

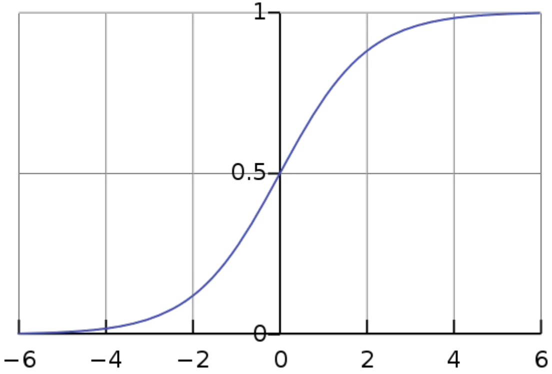
Odds (Ratio)

- Represents the likelihood of some event taking place (vs. not taking place)
  - Sometimes (especially for gambling) expressed as “odds on” and “odds against”
  - NOT the same as probability, but related
- Odds ratio
  - Given two events, A and B:
    - What are the odds that B is seen with A?
    - What are the odds that (not B) is seen with A?
    - What is the ratio of the above?
  - Similar to conditional probability

odds (ratio)	o_f	o_a	p	q
1:1	1	1	50%	50%
0:1	0	∞	0%	100%
1:0	∞	0	100%	0%
2:1	2	0.5	67%	33%
1:2	0.5	2	33%	67%
4:1	4	0.25	80%	20%
1:4	0.25	4	20%	80%
9:1	9	$0.\overline{1}$	90%	10%
10:1	10	0.1	$90.\overline{90}\%$	$9.\overline{09}\%$
99:1	99	$0.\overline{01}$	99%	1%
100:1	100	0.01	$99.\overline{0099}\%$	$0.\overline{90}\%$

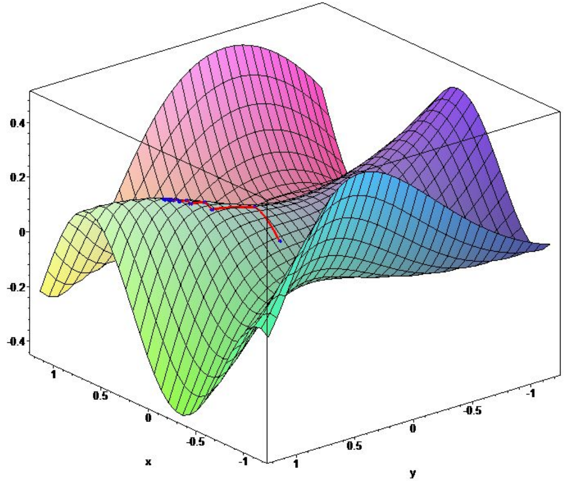
Step Function

- Need a function to decide the class of an instance
  - A step function allows us to do this
  - (There are issues with Heaviside)
  - Of the possibilities, we use the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$
  - Any value above 0.5 = 1; otherwise = 0
- Objective of the Logistic Regression algorithm:
  - Establish values for weight matrix
  - Minimise the error / Maximise the accuracy



Gradient

- Error curves
  - Assume we can quantify the errors of a model as a continuous function of the feature weights
  - Goal: find a point of minimal error by finding the lowest point on our error curve
  - The contrapositive is true for accuracy
- Finding the minimal error point:
  - Quantify error in a consistent manner
  - Move in the direction of maximum error reduction
  - Practically: move slowly to avoid overshooting



From Linear Regression

- For training a linear regression model, we have:
  - Y (target, a vector of real numbers) … and

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}$$

- … and we derive a vector of regression coefficients

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ . \\ . \\ \beta_p \end{bmatrix}$$

  - We sometimes call this the weight matrix (w^T)
- We use B (weight vector) as our odds ratios

Logistic Regression — Algorithm

```
Set all weights to some default value (1?)

REPEAT until convergence:

    Calculate error and gradient

    Update the weight vector by constant (alpha) * gradient

# convergence criteria (either of):

    # a) Number of times through loop

    # b) Error does not improve by greater than threshold
```

Trivial Example

- Assume two points:

  Instance 1 = (1, 1) ⇒ 1
  Instance 2 = (2, 2) ⇒ 0
- Initialise
  - alpha to some scalar (0.1 for this example?)
  - $Y = Y^T$
  - weights to `[ 1 1 ]`^{T}
- Repeat:
  - h = sigmoid(X * weights)
  - error = (Y - h)
  - weights += X^T * error * alpha

Logistic Regression: Implementation

- Classifier (Abstract) Base Class
  - Implemented as classifier.py
  - This will be the base class for all our classifiers
- Needs constructor + two functions:
  - Implemented as logistic_regression.py
  - `fit (self, train_x, train_y)`
    - train_x: a matrix of feature values
    - train_y: a vector of target values — 0 or 1 for binary logistic regression
    - Changes internal state of classifier; does not return any value
  - `predict (self, test_x)`
    - test_x: a matrix of feature values
    - Returns h(test_x): a vector of target hypotheses for test_x

Logistic Regression in Python (1)

- Establish some internal variables

```
def __init__(self):
    self.alpha = 0.001
    self.maxcycles = 500
    self.weights = None  # Placeholder for later
```

- Using numpy, the sigmoid step function is trivial (but maybe this function should be private/protected/"please don't touch"?)

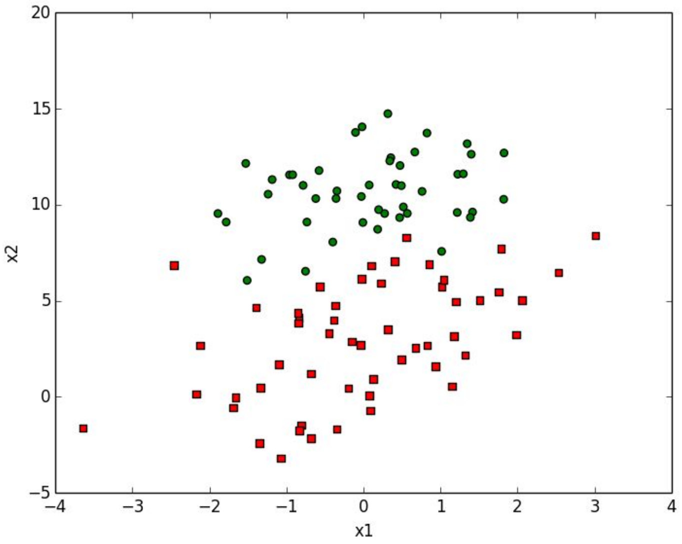
```
def sigmoid(self, x):
    return 1.0 / (1 + np.exp(-x))
```

Logistic Regression in Python (2)

```
def fit(self, Xin, Yin):
    X = np.mat(Xin)
    Y = np.mat(Yin).transpose()
    m, n = X.shape
    self.weights = np.ones((n, 1))
    for k in range(self.maxcycles):
        h = self.sigmoid(X * self.weights) # h = hypotheses
        error = (Y - h)
        self.weights = self.weights + self.alpha * \
            X.transpose() * error
    return self.weights
```

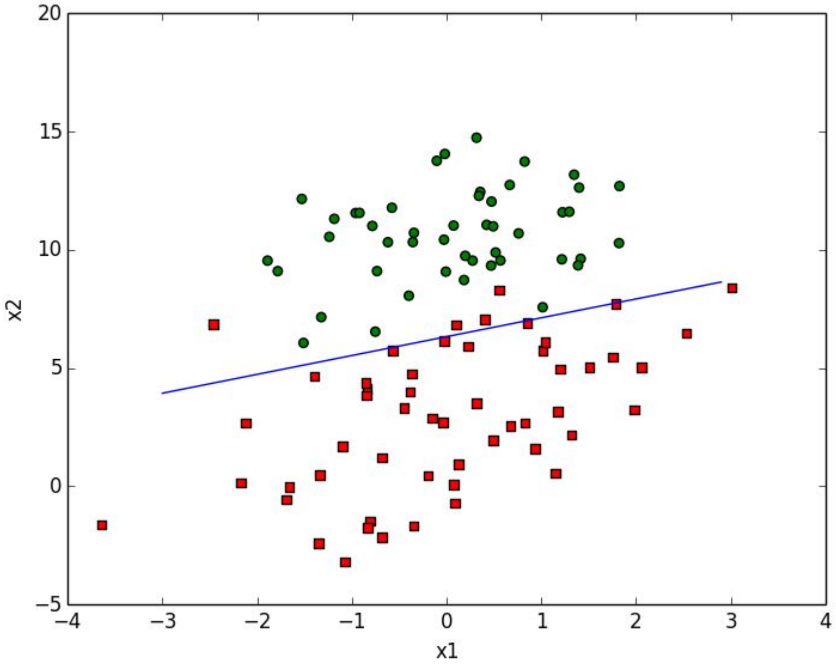
Example (not a lab, but...)

- This is not a lab, but perhaps a good exercise:
  - Import this synthetic dataset
  - Data has two classes (0, 1) and two features
  - Graph the data — should look like this:
- Apply linear regression algorithm
  - The `fit` function as above
  - No need for `predict` function yet
  - Graph the line associated with the weights



Example — Results

- After 1 iteration
- After 10 iterations
- After 50 iterations
- After 100 iterations
- After 250 iterations
- After 500 iterations



Questions about the Example

- What is the effect of increasing the number of iterations?
- What is the effect of changing alpha?
  - Increasing to 1.0?
  - Decreasing to 0.00001?
- What is the effect of changing the initial values of the weight vector?
  - … when the number of iterations is low (1? 10?)
  - … when the number of iterations increases?

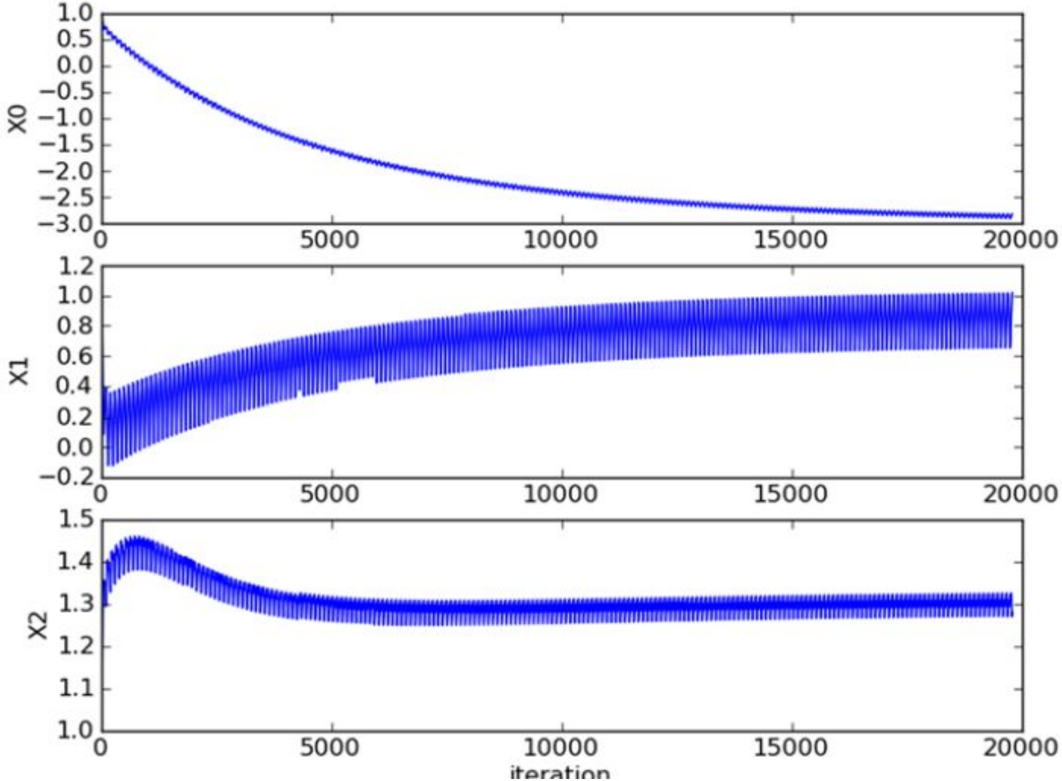
Stochastic Gradient

- Example is a toy dataset
  - N = 100; p = 2
  - 500 iterations to converge
  - What if this were a big data problem?
- A step in the right direction: ***stochastic*** gradient [descent/ascent]
  - Updates weights based on the current instance
  - Algorithm:

```
Set all weights to some default value (1?)
foreach x_i ∈ X:
    Calculate error and gradient for x_i
    Update the weight vector by constant (alpha) * gradient
```

  - Online learning algorithm

Checking Weights



- What is happening?
    - Happens to all weights
    - Especially for x1, x2
- Why is it happening?
- What can be done?

Hypotheses / Predictions

- With Logistic Regression, generating a hypothesis is simple:
  - p(x) = sigmoid(sum(x*self.weights))
  - If p(x) > 0.5, x represents an instance of the class encoded as 1
- Normally, we want to generate a vector of predictions, one for each input:

```
def predict(self, X):
    hypotheses = []
    for x in X:
        prob = self.sigmoid(sum(x*self.weights))
        if prob > 0.5:
            hypotheses.append(1)
        else:
            hypotheses.append(0)
    return hypotheses
```

Model Performance

- How good is our model?
- Accuracy
  - Easy to calculate and understand
  - Does not show where the errors lie — i.e. where the model can be improved
- Confusion Matrix
  - Table used to show performance of a classifier
  - Simple to understand
  - Terminology can be confusing
    - True Positive / True Negative
    - False Positive (Type I error)
    - False Negative (Type II error)

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

From http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/

Testing the Model

- Another good exercise:
  - Predict the classes of this subset of the synthetic dataset
  - Data has two classes (0, 1) and two features
  - Determine the accuracy
  - Show the confusion matrix
- Show solutions

Next Time

- Back to the Tweet Storm
  - Twitter Political Corpus data = https://www.usna.edu/Users/cs/nchamber/data/twitter/
  - L1 & L2 norms
  - Cross Validation
- Multinomial Logistic Regression