Logistic Regression

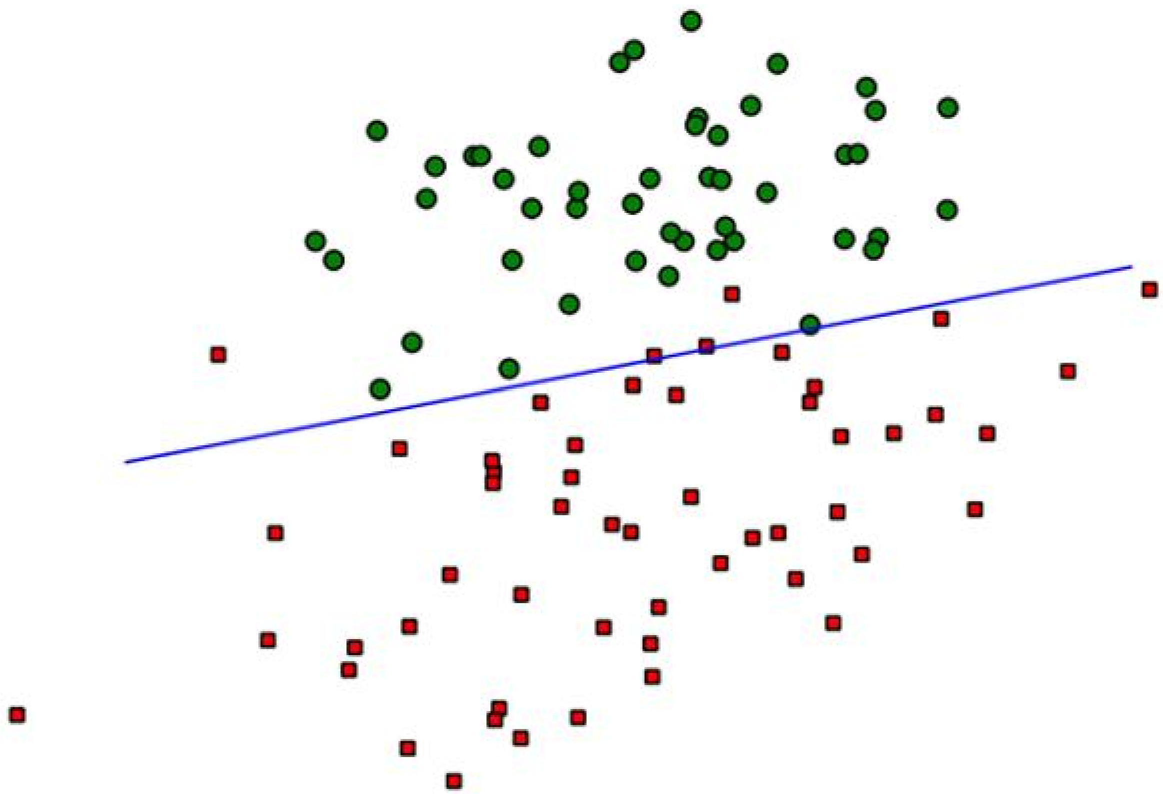
葉建華

jhyeh@mail.au.edu.tw

http://jhyeh.csie.au.edu.tw/

What is Regression?

- Some data points and then someone fit a line called the *best-fit* line to these points



Logistic Regression

- We have a bunch of data, and with the data we try to build an equation to do classification for us
  - The regression aspects means that we try to find a best-fit set of parameters
- Finding the best fit is similar to regression
  - That's how we train the classifier
- Use optimization algorithms to find the best-fit parameters

Logistic Regression

Logistic regression

Pros: Computationally inexpensive, easy to implement, knowledge representation easy to interpret

Cons: Prone to underfitting, may have low accuracy

Works with: Numeric values, nominal values

Sigmoid Function

- In two-class prediction, we want to have a function to spit out a 0 or 1
  - (Heaviside) step function
- Problem of step function
  - Instantly from 0 to 1
  - Sometimes difficult to deal with
- Sigmoid: a not-so-drastic curve function

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

Sigmoid Function

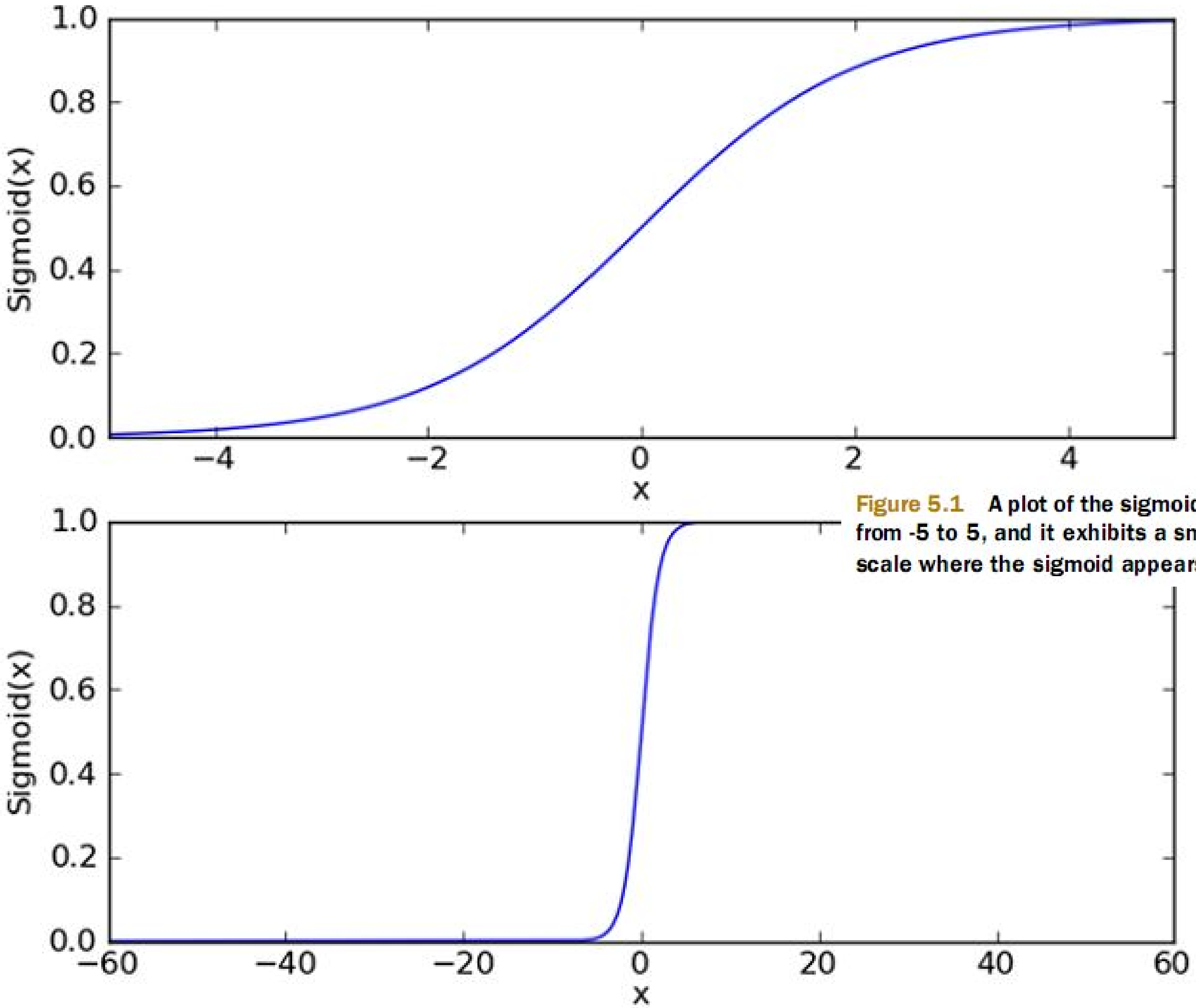


Figure 5.1 A plot of the sigmoid function on two scales; the top plot shows the sigmoid from -5 to 5, and it exhibits a smooth transition. The bottom plot shows a much larger scale where the sigmoid appears similar to a step function at x=0.

Regression Coefficients

- z is given by the following

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

$$z = w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

- Or in vector notation

$$z = w^T x$$

- x: input data, w: coefficients to find

真理大學 Aletheia University

Gradient Ascent

- Based on the idea that if we want to find the maximum point on a function
  - Best way to move: in the direction of the gradient

$$\nabla f(x,y) = \begin{pmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{pmatrix}$$

  - Moving amount for x direction: $\dfrac{\partial f(x,y)}{\partial x}$
  - Moving amount for y direction: $\dfrac{\partial f(x,y)}{\partial y}$

Gradient Ascent Algorithm

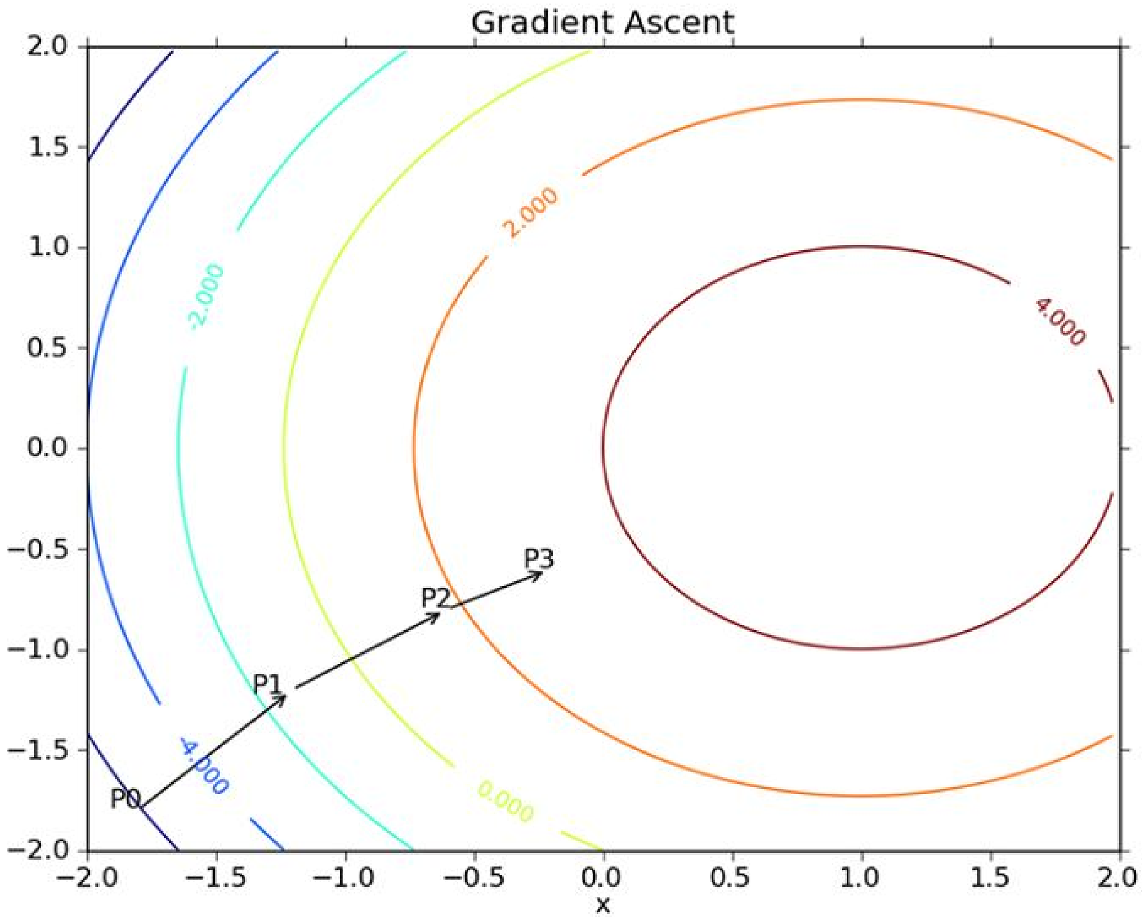


Figure 5.2 The gradient ascent algorithm moves in the direction of the gradient evaluated at each point. Starting with point P0, the gradient is evaluated and the function moves to the next point, P1. The gradient is then reevaluated at P1, and the function moves to P2. This cycle repeats until a stopping condition is met. The gradient operator always ensures that we're moving in the best possible direction.

Magnitude of Movement

- By defining parameter α

$$\boldsymbol{w} := \boldsymbol{w} + \alpha \nabla_{\mathbf{W}} f(w)$$

- Recall:

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

$$\nabla f(x,y) = \begin{pmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{pmatrix}$$

$$z = w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

- Stopping condition
  - A specified number of steps, or
  - within a certain tolerance margin

Gradient Descent

- Same thing except finding the minimum point

Gradient descent

Perhaps you've also heard of gradient descent. It's the same thing as gradient ascent, except the plus sign is changed to a minus sign. We can write this as

$$w := w - \alpha \nabla_{\mathbf{w}} f(w)$$

With gradient descent we're trying to minimize some function rather than maximize it.

A Simple Dataset

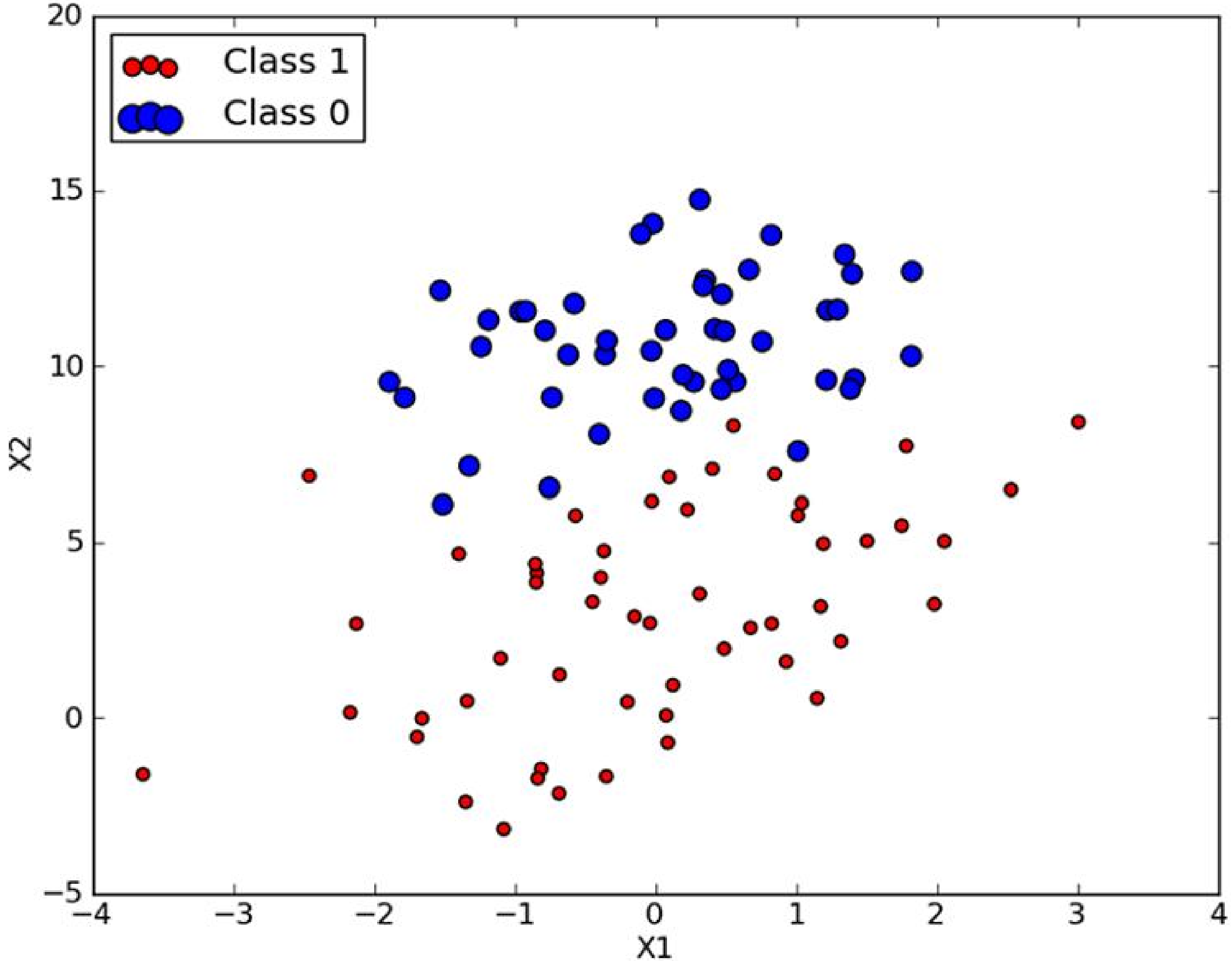


Figure 5.3 Our simple dataset. We're going to attempt to use gradient descent to find the best weights for a logistic regression classifier on this dataset.

Gradient Ascent Pseudocode

Start with the weights all set to 1
Repeat R number of times:
    Calculate the gradient of the entire dataset
    *Update the weights vector by alpha*gradient*
    Return the weights vector

Gradient Ascent Optimization Function

Listing 5.1 Logistic regression gradient ascent optimization functions

```
def loadDataSet():
    dataMat = []; labelMat = []
    fr = open('testSet.txt')
    for line in fr.readlines():
        lineArr = line.strip().split()
        dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])])
        labelMat.append(int(lineArr[2]))
    return dataMat,labelMat

def sigmoid(inX):
    return 1.0/(1+exp(-inX))

def gradAscent(dataMatIn, classLabels):
    dataMatrix = mat(dataMatIn)
    labelMat = mat(classLabels).transpose()
    m,n = shape(dataMatrix)
    alpha = 0.001
    maxCycles = 500
    weights = ones((n,1))
    for k in range(maxCycles):
        h = sigmoid(dataMatrix*weights)
        error = (labelMat - h)
        weights = weights + alpha * dataMatrix.transpose()* error
    return weights
```

1 Convert to NumPy matrix data type

2 Matrix multiplication

Decision Boundary Plotting

Listing 5.2 Plotting the logistic regression best-fit line and dataset

```
def plotBestFit(wei):
    import matplotlib.pyplot as plt
    weights = wei.getA()
    dataMat,labelMat=loadDataSet()
    dataArr = array(dataMat)
    n = shape(dataArr)[0]
    xcord1 = []; ycord1 = []
    xcord2 = []; ycord2 = []
    for i in range(n):
        if int(labelMat[i])== 1:
            xcord1.append(dataArr[i,1]); ycord1.append(dataArr[i,2])
        else:
            xcord2.append(dataArr[i,1]); ycord2.append(dataArr[i,2])
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(xcord1, ycord1, s=30, c='red', marker='s')
    ax.scatter(xcord2, ycord2, s=30, c='green')
    x = arange(-3.0, 3.0, 0.1)
    y = (-weights[0]-weights[1]*x)/weights[2]
    ax.plot(x, y)
    plt.xlabel('X1'); plt.ylabel('X2');
    plt.show()
```

1 Best-fit line

Decision Boundary Plotting

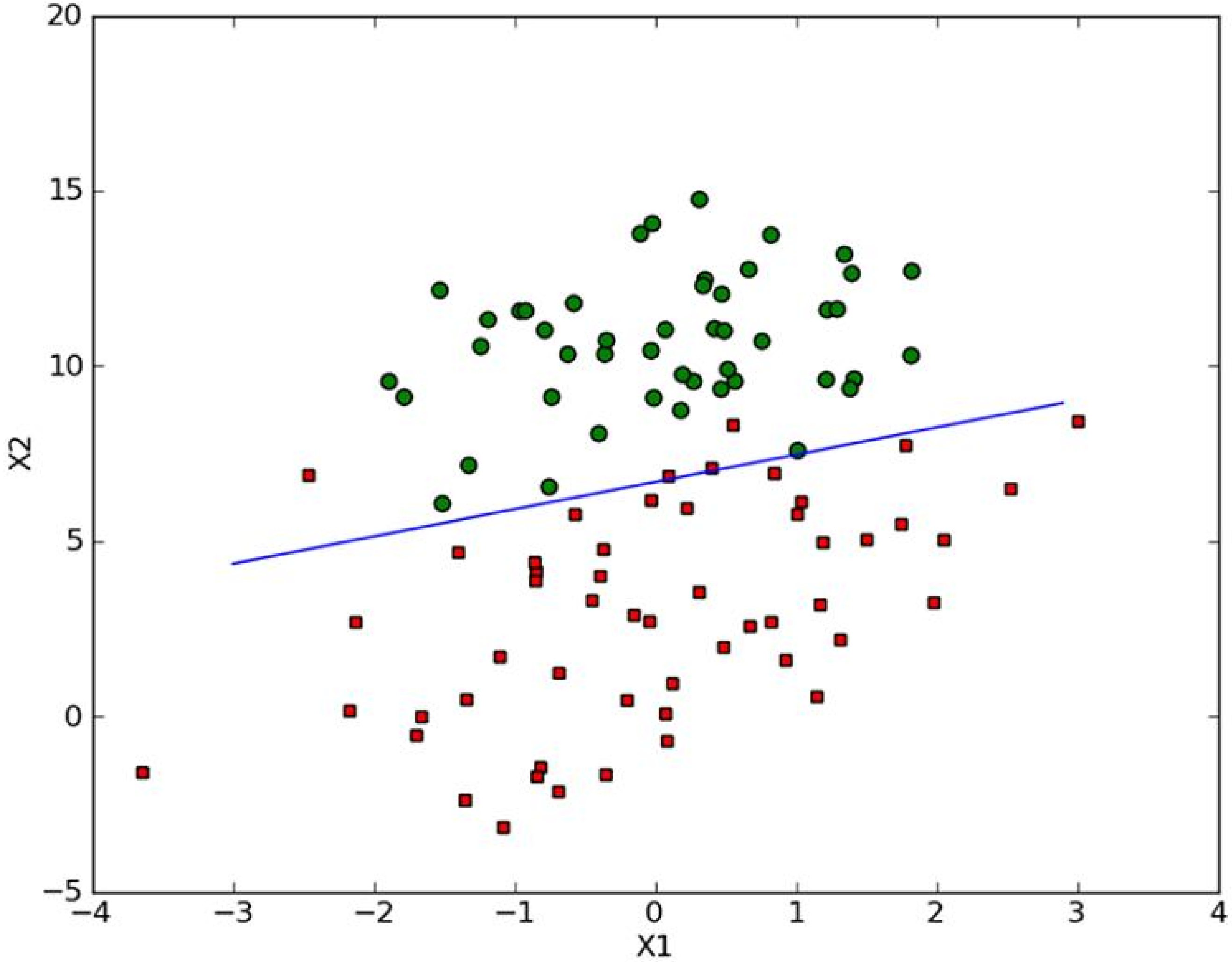


Figure 5.4 The logistic regression best-fit line after 500 cycles of gradient ascent

Train: Stochastic Gradient Ascent

- Use the whole dataset to calculate update?

$$z = w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

- Stochastic gradient ascent
  - Update the weights using only one instance at a time
  - An example of an online learning algorithm, incrementally update!

Stochastic Gradient Ascent Pseudocode

Start with the weights all set to 1
For each piece of data in the dataset:
    Calculate the gradient of one piece of data
    *Update the weights vector by alpha*gradient*
    Return the weights vector

Stochastic Gradient Ascent Optimization

Listing 5.3 Stochastic gradient ascent

```
def stocGradAscent0(dataMatrix, classLabels):
    m,n = shape(dataMatrix)
    alpha = 0.01
    weights = ones(n)
    for i in range(m):
        h = sigmoid(sum(dataMatrix[i]*weights))
        error = classLabels[i] - h
        weights = weights + alpha * error * dataMatrix[i]
    return weights
```

Decision Boundary Plotting

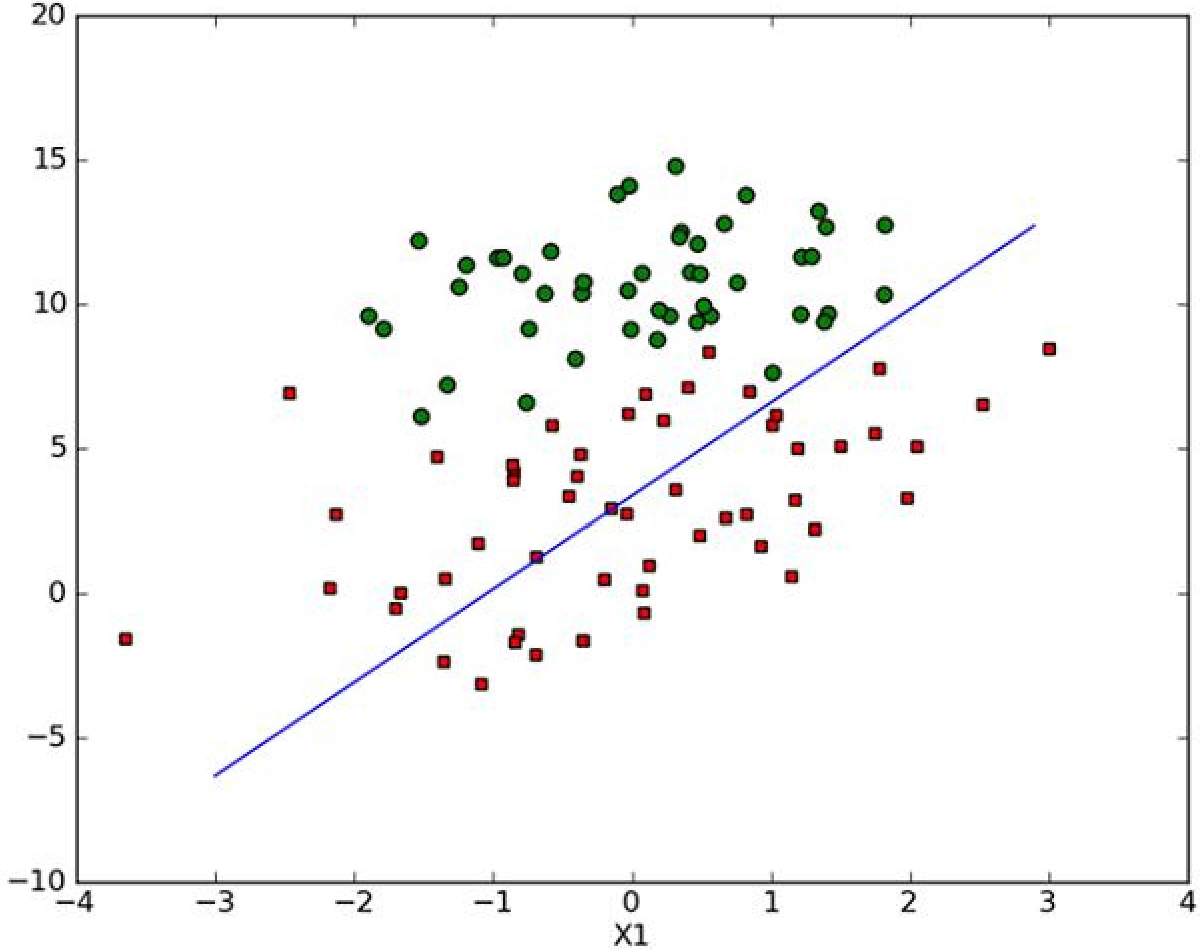


Figure 5.5 Our simple dataset with solution from stochastic gradient ascent after one pass through the dataset. The best-fit line isn't a good separator of the data.

Weight vs. Iteration

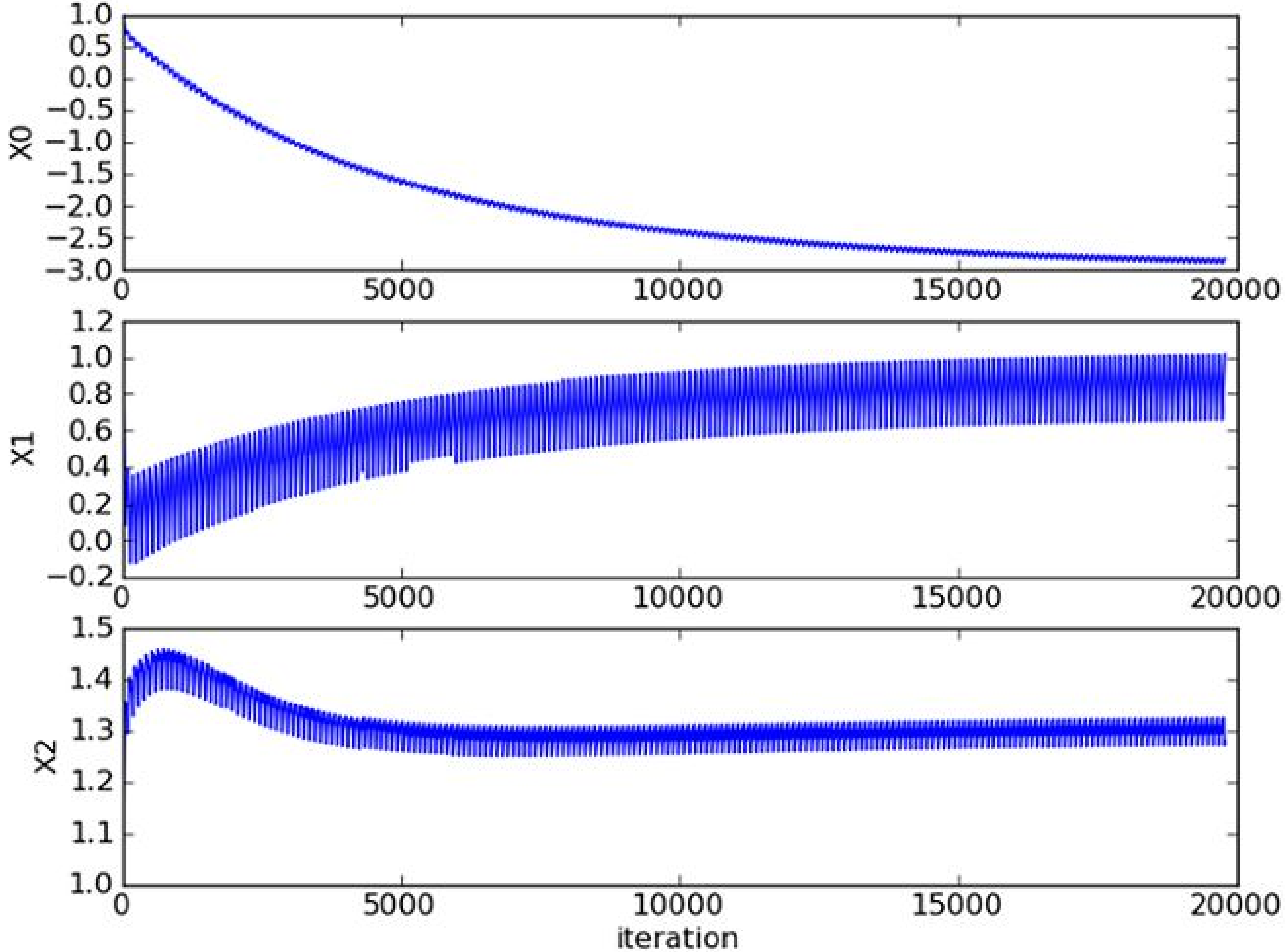


Figure 5.6 Weights versus iteration number for one pass through the dataset, with this method. It takes a large number of cycles for the weights to reach a steady-state value, and there are still local fluctuations.

The Problems?

- Weight oscillation

Modified Algorithm

Listing 5.4 Modified stochastic gradient ascent

```
def stocGradAscent1(dataMatrix, classLabels, numIter=150):
    m,n = shape(dataMatrix)
    weights = ones(n)
    for j in range(numIter):          dataIndex = range(m)
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.01
            randIndex = int(random.uniform(0,len(dataIndex)))
            h = sigmoid(sum(dataMatrix[randIndex]*weights))
            error = classLabels[randIndex] - h
            weights = weights + alpha * error * dataMatrix[randIndex]
            del(dataIndex[randIndex])
    return weights
```

1 Alpha changes with each iteration

2 Update vectors are randomly selected

Modifications

- Alpha decreases as the number of iterations increases
  – Will reach a minimum constant
- Randomly selecting each instance to update
  – But this will not applicable for online training

Weight vs. Iteration

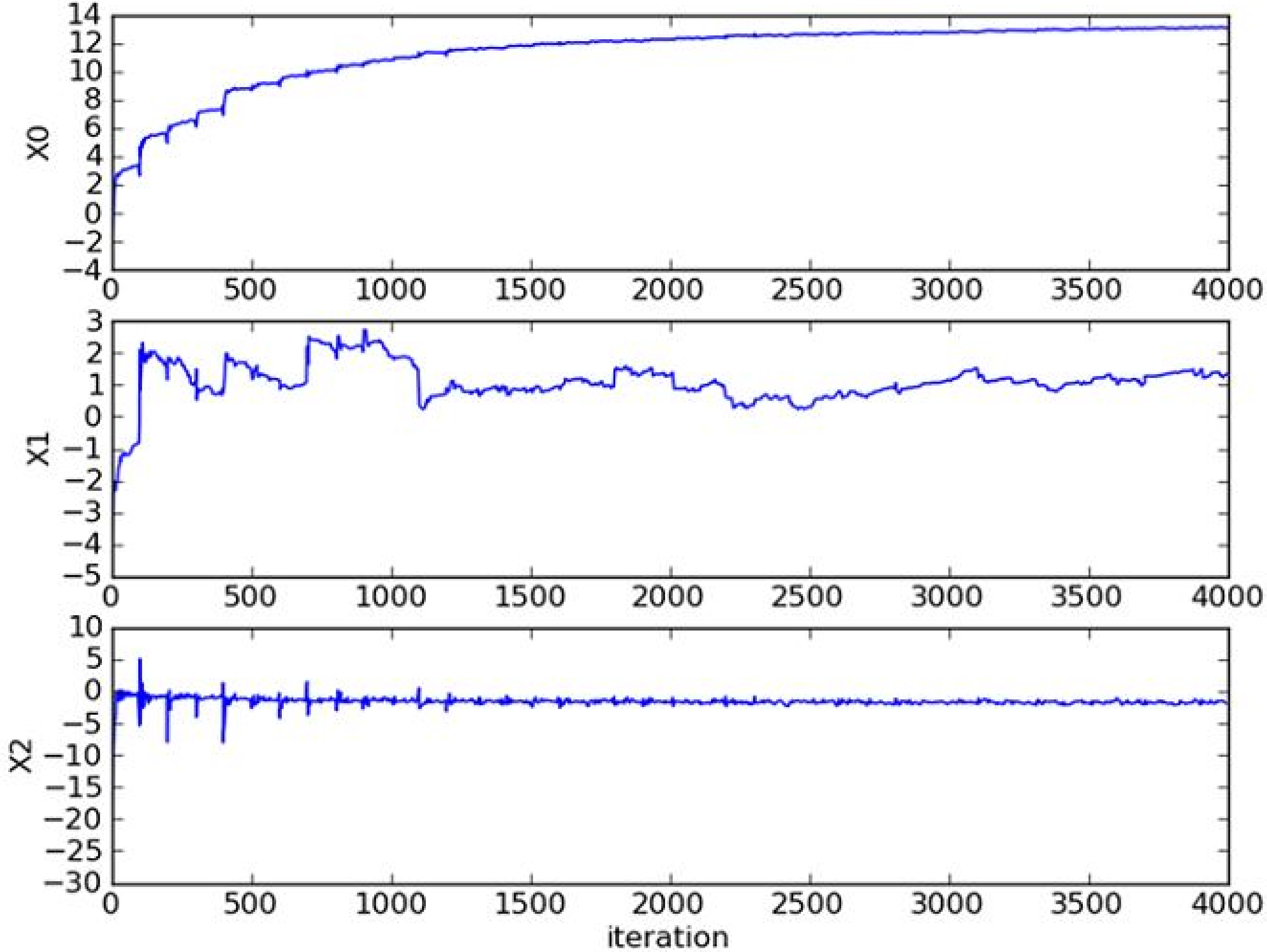


Figure 5.7 **Coefficient convergence in `stocGradAscent1()` with random vector selection and decreasing alpha. This method is much faster to converge than using a fixed alpha.**

Decision Boundary Plotting

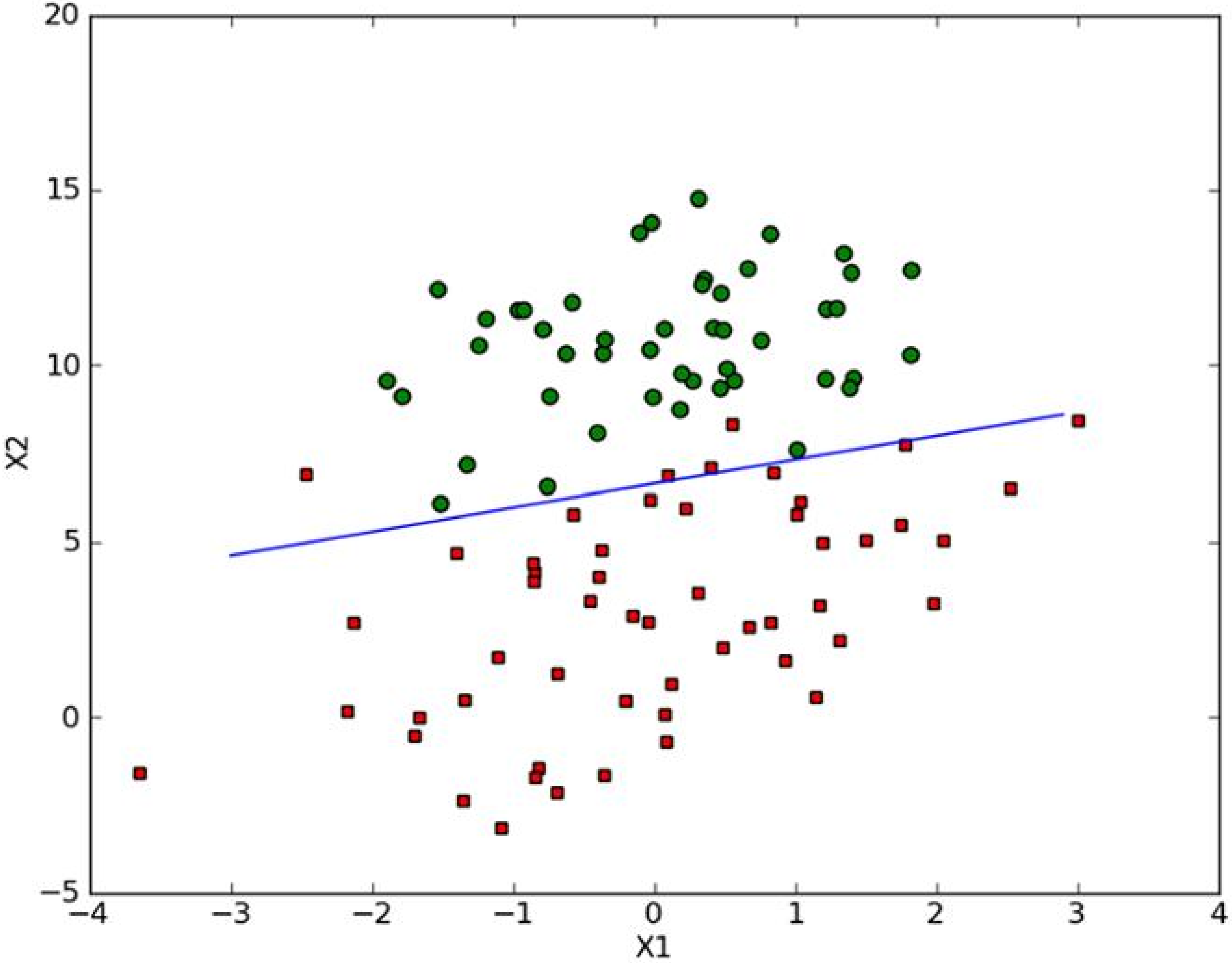


Figure 5.8 The coefficients with the improved stochastic gradient descent algorithm

Example: Estimating Horse Fatalities from Colic

- From UCI Machine Learning Repository
- 368 instances with 28 features

Example: using logistic regression to estimate horse fatalities from colic

1. Collect: Data file provided.
2. Prepare: Parse a text file in Python, and fill in missing values.
3. Analyze: Visually inspect the data.
4. Train: Use an optimization algorithm to find the best coefficients.
5. Test: To measure the success, we'll look at error rate. Depending on the error rate, we may decide to go back to the training step to try to find better values for the regression coefficients by adjusting the number of iterations and step size.
6. Use: Building a simple command-line program to collect horse symptoms and output live/die diagnosis won't be difficult. I'll leave that up to you as an exercise.

Missing Data Problem

- Handling options
  - Use the feature's mean value from all the available data
  - Fill in the unknown with a special value like -1
  - Ignore the instance
  - Use a mean value from similar items
  - Use another machine learning algorithm to predict the value

Testing Code

Listing 5.5 Logistic regression classification function

```
def classifyVector(inX, weights):
    prob = sigmoid(sum(inX*weights))
    if prob > 0.5: return 1.0
    else: return 0.0

def colicTest():
    frTrain = open('horseColicTraining.txt')
    frTest = open('horseColicTest.txt')
    trainingSet = []; trainingLabels = []
    for line in frTrain.readlines():
        currLine = line.strip().split('\t')
        lineArr =[]
        for i in range(21):
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        trainingLabels.append(float(currLine[21]))
    trainWeights = stocGradAscent1(array(trainingSet), trainingLabels, 500)
    errorCount = 0; numTestVec = 0.0
    for line in frTest.readlines():
        numTestVec += 1.0
        currLine = line.strip().split('\t')
        lineArr =[]
        for i in range(21):
            lineArr.append(float(currLine[i]))
        if int(classifyVector(array(lineArr), trainWeights))!=
            int(currLine[21]):
            errorCount += 1
    errorRate = (float(errorCount)/numTestVec)
    print "the error rate of this test is: %f" % errorRate
    return errorRate

def multiTest():
    numTests = 10; errorSum=0.0
    for k in range(numTests):
        errorSum += colicTest()
    print "after %d iterations the average error rate is:
        %f" % (numTests, errorSum/float(numTests))
```

Summary

- Logistic regression is finding best-fit parameters to a nonlinear function called the sigmoid
- Gradient ascent can be simplified with stochastic version
  - Do as well as gradient ascent using far fewer computing resources
- Deal with missing values