Homework 1

Andrew Walters, A08443596 10/18/2015

Perceptron

Problem 1

Decision Boundary:

$$w_1x_1 + w_2x_2 - \theta = 0 \rightarrow w^Tx = \theta$$

Distance Proof - The unit vector for weights w shows the perpendicular relationship of the decision boundary to the origin. Therefore for any x the dot product of the unit vector w will result in l:

$$l = \frac{\theta}{||w||}$$

$$\theta = w^Tx$$

$$l = \frac{w^Tx}{||w||}$$

Problem 2

Learning Rule:

$$w_i(t+1) = w_i(t) + \alpha(teacher - output)x_i$$

Learning Pattern:

x_1	x_2	w_1	w_2	Net	Output	Teacher	Threshold (θ)
1	1	0	0	0	1	0	0
0	0	-1	-1	0	0	1	1
0	1	-1	-1	-1	0	1	0
1	1	-1	0	-1	1	0	-1
1	0	-2	-1	-2	0	1	0
		-1	-1				-1

$$Note : \alpha = 1$$

The solution is not unique. There are three possible solutions that result in convergence. The possible solutions respectively are:

$$w_1 \rightarrow -1, -1, -2$$
$$w_2 \rightarrow -1, -2, -1$$
$$\theta \rightarrow -1, -2, -2$$

Problem 3

Z-Score - The z-score is used to normalize the data. By doing so it allows for a more consistent learning rate when training the weights. In cases where the data sets contain a wider range of values this can help a lot. See attached code for z-score calculation (flower$_classifier.py - findZScore()$)

Plots - The data is linearly separable relative to the flower type. This is clear by looking at the graphs due to the two clusters of data points. See attached files for graph images.

Average Error Rate - 0.0

Learning Rate - My learning rate is set to 1. By altering it you can scale the weights differently. In this case altering the learning rate does not have much of an impact on the outcome.

Code - Code for this problem can be found in the appendix and in the attached file flower$_classifier.py$.

Regression

Problem 1

Logistic Regression

$$E(\theta) = -\sum_{i=1}^{N} y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))$$

$$E(\theta) = -\sum_{i=1}^{N} y^{(i)} log(\frac{1}{1 + e^{-\theta^T x^{(i)}}})) + (1 - y^{(i)}) log(1 - \frac{1}{1 + e^{-\theta^T x^{(i)}}})$$

$$E(\theta) = \sum_{i=1}^{N} log(1 + e^{-\theta^T x^{(i)}})) - y^{(i)} \theta x^{(i)}$$

$$\frac{\partial E(\theta)}{\partial \theta} = \sum_{i=1}^{N} \frac{x^{(i)}}{1 + e^{-\theta^T x^{(i)}}} - y^{(i)} x^{(i)}$$

$$\frac{\partial E(\theta)}{\partial \theta} = \sum_{i=1}^{N} x^{(i)} (h_\theta(x^{(i)}) - y^{(i)})$$

Problem 2

Softmax Regression

$$E(\theta) = -\sum_{i=1}^{N} \sum_{j=1}^{K} 1\{y^{(i)} = j\} log \frac{e^{\theta^{(j)T} x^{(i)}}}{\sum_{l=1}^{K} e^{\theta^{(l)T} x^{(i)}}}$$

$$E(\theta) = -\sum_{i=1}^{N} \sum_{j=1}^{K} 1\{y^{(i)} = j\} (log(e^{\theta^{(j)T} x^{(i)}}) - log \sum_{l=1}^{K} e^{\theta^{(l)T} x^{(i)}})$$

$$E(\theta) = -\sum_{i=1}^{N} \sum_{j=1}^{K} 1\{y^{(i)} = j\} (\theta^{(j)T} x^{(i)} - log \sum_{l=1}^{K} e^{\theta^{(l)T} x^{(i)}})$$

$$\nabla_{\theta^{(k)}} E(\theta) = -\sum_{i=1}^{N} 1\{y^{(i)} = k\} (x^{(i)} - \frac{x^{(i)} e^{\theta^{(j)T} x^{(i)}}}{\sum_{j=1}^{K} e^{\theta^{(j)T} x^{(i)}}})$$

$$\nabla_{\theta^{(k)}} E(\theta) = -\sum_{i=1}^{N} x^{(i)} (1\{y^{(i)} = k\} - \frac{e^{\theta^{(j)T} x^{(i)}}}{\sum_{j=1}^{K} e^{\theta^{(j)T} x^{(i)}}})$$

$$\nabla_{\theta^{(k)}} E(\theta) = -\sum_{i=1}^{N} x^{(i)} (1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta))$$

Problem 3

See code in Appendix or attached file regression.py

Problem 4

I implemented Logistic Regression with Stochastic Batch Gradient Descent. The default parameters results in a run with 100 learning iterations on random batch sets of size 1000. The learning rate defaults to 1e-4.

Logistic Regression Overall Accuracy: 0.8725
Label 0 Accuracy: 0.977142857143
Label 1 Accuracy: 0.961538461538
Label 2 Accuracy: 0.835616438356
Label 3 Accuracy: 0.850241545894
Label 4 Accuracy: 0.861751152074
Label 5 Accuracy: 0.821229050279
Label 6 Accuracy: 0.921348314607
Label 7 Accuracy: 0.819512195122
Label 8 Accuracy: 0.822916666667
Label 9 Accuracy: 0.855670103093

Problem 5

See attached file for graph image

I implemented Softmax Regression with Stochastic Batch Gradient Descent. The default parameters results in a run with 100 learning iterations on random batch sets of size 1000. The learning rate defaults to 1e-4.

Softmax Regression Overall Accuracy: 0.881
Label 0 Accuracy: 0.988571428571
Label 1 Accuracy: 0.974358974359
Label 2 Accuracy: 0.826484018265
Label 3 Accuracy: 0.869565217391
Label 4 Accuracy: 0.898617511521
Label 5 Accuracy: 0.832402234637
Label 6 Accuracy: 0.910112359551
Label 7 Accuracy: 0.853658536585
Label 8 Accuracy: 0.817708333333
Label 9 Accuracy: 0.835051546392

The accuracy of softmax does seem to consistently be slightly better. This could be due to it working by training weights for each individual case.

Appendix

Listing 1: Perceptron – Flower Classifier

```
import random
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

def findZScore(data):
    """
    Predicts Z-Scores of data
    :param data: List of x vectors and labels ex: [[array[], label],...]
    :return: data with x vectors normalized to Z-Scores
    """
    for col in range(4):
        temp = []
        for row in range(len(data)):
            temp.append(data[row][0][col])
        zScores = stats.zscore(np.array(temp))
        for row in range(len(data)):
            data[row][0][col] = zScores[row]
    return data

def plotData(data, x, y, name, zScore):
    """
    Scatter plots two columns in data
    :param data: List of x vectors and labels ex: [[array[], label],...]
    :param x: column in data
    :param y: column in data
    :param name: list of attribute name for x, y
    :param zScore: flag if data has been normailized to Z-Scores
    """
    xAxis, yAxis = name
    figure = plt.figure()
    ax = figure.add_subplot(1,1,1)
    xSet, ySet, xVer, yVer = [[],[],[],[]]
    for line in data:
        if line[1] == 0:
            xSet.append(line[0][x])
            ySet.append(line[0][y])
        else:
            xVer.append(line[0][x])
            yVer.append(line[0][y])
    ax.scatter(xSet, ySet, color='red')
    ax.scatter(xVer, yVer, color='blue')
    if zScore:
        ax.set_title('Z Scores: ' + xAxis + ' vs. ' + yAxis +
                     '(Setosa=Red, Versicolor=Blue)')
    else:
        ax.set_title(xAxis + ' vs. ' + yAxis +
                     '(Setosa=Red, Versicolor=Blue)')
    ax.set_xlabel(xAxis)
    ax.set_ylabel(yAxis)
    if zScore:
        figure.savefig('ScatterPlots/ZScore/ZScore_' + xAxis + '_' +
                       yAxis + '.png')
    else:
        figure.savefig('ScatterPlots/' + xAxis + '_' + yAxis + '.png')
    return

```

```python
def generatePlots(inputFile, zScore):
    """
    Builds all possible column vs. column plots
    :param inputFile: filename
    :param zScore: flag if data has been normailized to Z-Scores
    """
    name = {0: 'Sepal Length',
            1: 'Sepal Width',
            2: 'Pedal Length',
            3: 'Pedal Width'}
    data = parseData(inputFile)
    if zScore:
        data = findZScore(data)
    for i in range(4):
        for j in range(i+1, 4):
            if i != j:
                plotData(data, i, j, [name[i], name[j]], zScore)
    return

def classLabel(label):
    """
    Determines label of attributes
    :param label: flower name
    :return: binary label
    """
    if 'setosa' in label:
        return 0
    else:
        return 1

def parseData(trainFile):
    """
    Parses data out of input file
    :param trainFile: filename
    :return: List of x vectors and labels ex: [[array[], label],...]
    """
    file = open(trainFile)
    lines = file.readlines()
    for x in range(len(lines)):
        line = lines[x].strip().split(',')
        xVector = np.array([float(line[x]) for x in range(4)])
        lines[x] = [xVector, classLabel(line[4])]
    file.close()
    return lines

def trainPerceptron(trainFile, zScore):
    """
    Trains w vector and threshold
    :param trainFile: filename
    :param zScore: flag if you want the data to be normailized to Z-Scores
    :return: w and threshold
    """
    a = random.randint(1, 1000)
    train = parseData(trainFile)
    if zScore:
        train = findZScore(train)
    w = np.array([0] * 4)
    threshold = 0
    randRange = len(train) - 1
    limit = 0
    while (limit < 100):
        rand = random.randint(0,randRange)
```

```
        x, teacher = train[rand]
        net = np.dot(x, w)
        if net >= threshold:
            output = 1
        else:
            output = 0
        w = w + a * (teacher - output) * x
        threshold = threshold + (teacher - output)
        limit += 1
    return w, threshold

def predict(w, threshold, testFile, zScore):
    """
    Predicts type of flower and prints error rate
    :param w: vector of weights
    :param threshold: threshold constant
    :param testFile: filename
    :param zScore: flag if w and threshold were generated based on Z-Scores
    """
    test = parseData(testFile)
    if zScore:
        test = findZScore(test)
    error = 0
    for x, label in test:
        net = np.dot(x, w)
        if net >= threshold:
            output = 1
        else:
            output = 0
        if output != label:
            error += 1
    print('Error Rate: ', end='')
    print(error / len(test))
    return

#Builds scatter plots
generatePlots('iris_train.data', False)
#Builds Z-Score scatter plots
generatePlots('iris_train.data', True)
#Trains perceptron
w, threshold = trainPerceptron('iris_train.data', False)
#Predicts results with perceptron
predict(w, threshold, 'iris_test.data', False)
```

Listing 2: Logistic/Softmax Regression – Number Classifier

```
import os, struct
import numpy as np
from array import array as pyarray
from numpy import append, array, int8, uint8, zeros
from scipy import stats
import math
import random
import matplotlib.pyplot as plt


def loadMNIST(dataset="training", digits=np.arange(10), path="."):
    """
    Loads MNIST files into 3D numpy arrays

    Adapted from: http://abel.ee.ucla.edu/cvxopt/_downloads/mnist.py
    """
    if dataset == "training":
        fname_img = os.path.join(path, 'train-images.idx3-ubyte')
        fname_lbl = os.path.join(path, 'train-labels.idx1-ubyte')
    elif dataset == "testing":
        fname_img = os.path.join(path, 't10k-images.idx3-ubyte')
        fname_lbl = os.path.join(path, 't10k-labels.idx1-ubyte')
    else:
        raise ValueError("dataset must be 'testing' or 'training'")

    flbl = open(fname_lbl, 'rb')
    magic_nr, size = struct.unpack(">II", flbl.read(8))
    lbl = pyarray("b", flbl.read())
    flbl.close()

    fimg = open(fname_img, 'rb')
    magic_nr, size, rows, cols = struct.unpack(">IIII", fimg.read(16))
    img = pyarray("B", fimg.read())
    fimg.close()

    ind = [ k for k in range(size) if lbl[k] in digits ]
    N = len(ind)

    images = zeros((N, rows, cols), dtype=uint8)
    labels = zeros((N, 1), dtype=int8)
    for i in range(len(ind)):
        images[i] = array(img[ ind[i]*rows*cols : (ind[i]+1)*rows*cols ]).reshape((rows, cols))
        labels[i] = lbl[ind[i]]

    return images, labels

#Lambda function to only grab first 20000 training set
trainingData = lambda : (x[0:20000] for x in loadMNIST())
#Lambda function to only grab first 2000 testing set
testingData = lambda : (x[0:2000] for x in loadMNIST('testing'))

def setupData(dataType):
    """
    Computes the Z-Scores for the images and adds the intercept term
    :param dataType: either training or testing
    :return: images, labels
    """
    if dataType != 'training' and dataType != 'testing':
        raise ValueError('dataType must be training or testing')
```

```python
    if dataType == 'training':
        images, labels = trainingData()
    else:
        images, labels = testingData()
    images = [np.insert(stats.zscore(np.concatenate(x)), 0, 1.0) for x in images]
    return images, labels

class logisticRegression:

    def __init__(self, learn=1, iterations=100, subset=1000):
        """
        Creates a logistic regression object and performs a gradient descent
        :param self: logisticRegression object
        :param learn: learning rate factor, defaults=1
        :param iterations: number of weight training iterations, default=100
        :param subset: size of random stochastic batch sampling, default=1000
        :attribute trainImages: vectorized zscores of training images
        :attribute trainLabels: vectorized labels of training images
        :attribute testImages: vectorized zscores of testing images
        :attribute testLabels: vectorized labels of testing images
        :attribute weights: vector matrix of pixel weights by classifications 785x10
        :attribute labels: 10x10 identity matrix used for label processing
        :attribute learn: learning rate, learn / (subset * 10)
        :attribute error: vector of testing errors on each label
        :attribute totals: vector of the number of each label in the testing set
        """
        self.trainImages, self.trainLabels = setupData('training')
        self.testImages, self.testLabels = setupData('testing')
        self.weights = np.array([[0.0] * len(self.trainImages[0]) for x in range(10)])
        self.labels = np.array([[1 if x == y else 0 for x in range(10)] for y in range(10)])
        self.learn = learn / (subset * 10)
        self.iterations = iterations
        self.subset = subset
        self.error = np.array([0] * 10)
        self.totals = np.array([0] * 10)
        for k in range(10):
            for _ in range(self.iterations):
                self.gradient(k)
        self.predict()

    def probability(self, x, k):
        """
        Determines probability of a label for an image
        :param x: image vector
        :param k: label
        :return: probability
        """
        dot = np.dot(self.weights[k], x)
        return 1 / (1 + math.exp(-1 * dot))

    def gradient(self, k):
        """
        Gradient descent learning algorithm, updates weights
        :param k: label
        """
        subset = random.sample(range(len(self.trainImages)), self.subset)
        temp = sum((self.probability(self.trainImages[i], [k]) - self.labels[k][self.trainLabels[i]])
                   * self.trainImages[i] for i in subset)
```

```
        self.weights[k] = self.weights[k] - self.learn * temp

    def predict(self):
        """
        Predicts testImages labels
        """
        for x, y in zip(self.testImages, self.testLabels):
            guess = 0
            best = 0
            self.totals[y] += 1
            for k in range(10):
                if best < self.probability(x, k):
                    guess = k
                    best = self.probability(x, k)
            if y != guess:
                self.error[y] += 1

class softmaxRegression:

    def __init__(self, learn=1, iterations=100, subset=1000):
        """
        Creates a softmax regression object and performs a gradient descent
        :param self: softmaxRegression object
        :param learn: learning rate factor, defaults=1
        :param iterations: number of weight training iterations, default=100
        :param subset: size of random stochastic batch sampling, default=1000
        :attribute trainImages: vectorized zscores of training images
        :attribute trainLabels: vectorized labels of training images
        :attribute testImages: vectorized zscores of testing images
        :attribute testLabels: vectorized labels of testing images
        :attribute weights: vector matrix of pixel weights by classifications 785x10
        :attribute labels: 10x10 identity matrix used for label processing
        :attribute learn: learning rate, learn / (subset * 10)
        :attribute error: vector of testing errors on each label
        :attribute totals: vector of the number of each label in the testing set
        """
        self.trainImages, self.trainLabels = setupData('training')
        self.testImages, self.testLabels = setupData('testing')
        self.weights = np.array([[0.0] * len(self.trainImages[0]) for x in range(10)])
        self.labels = np.array([[1 if x == y else 0 for x in range(10)] for y in range(10)])
        self.learn = learn / (subset * 10)
        self.iterations = iterations
        self.subset = subset
        self.error = np.array([0] * 10)
        self.totals = np.array([0] * 10)
        for _ in range(self.iterations):
            self.gradient()
        self.predict()

    def probability(self, x):
        """
        Determines probability of a label for an image
        :param x: image vector
        :return: probability vector 10x1
        """
        numerator = np.array([(math.exp(np.dot(self.weights[i], x))) for i in range(10)])
        denominator = sum(math.exp(np.dot(self.weights[i], x)) for i in range(10))
```

```
        return numerator / denominator

    def gradient(self):
        """
        Gradient descent learning algorithm, updates weights
        """
        subset = random.sample(range(len(self.trainImages)), self.subset)
        for k in range(10):
            temp = sum(self.trainImages[i] * (self.labels[k][self.trainLabels[i]]
                       - self.probability(self.trainImages[i])[k]) for i in subset)
            self.weights[k] = self.weights[k] - self.learn * (-1 * temp)

    def predict(self):
        """
        Predicts testImages labels
        """
        for x, y in zip(self.testImages, self.testLabels):
            self.totals[y] += 1
            if y != self.probability(x).argmax():
                self.error[y] += 1

def plotData():
    """
    Plots softmax accuracy relative to iterations of gradient descent
    """
    x = []
    y = []
    for iterations in range(10, 101, 10):
        sr = softmaxRegression(iterations=iterations)
        x.append(iterations)
        y.append(1 - sum(sr.error) / 2000)
    figure = plt.figure()
    ax = figure.add_subplot(1,1,1)
    ax.scatter(x, y)
    ax.set_title('Iterations vs. Test Accuracy')
    ax.set_xlabel('Iterations')
    ax.set_ylabel('Test Accuracy')
    figure.savefig('SoftmaxPlotAccuracy.png')




#builds logisticRegression object with defualt params
lr = logisticRegression()
#prints number of errors for each label type
print(lr.error)
#prints overall test accuracy
print('Logistic Regression Overall Accuracy: ', end='')
print(1 - sum(lr.error) / 2000)
count = 0
for x, y in zip(lr.error, lr.totals):
    print('Label ' + str(count) + ' Accuracy: ' + str(1 - x / y))
    count += 1
print(' ')

#Plots softmax accuracy data
plotData()

#build softmaxRegression object with default params
sr = softmaxRegression()
#prints number of errors for each label type
```

```python
print(sr.error)
#prints overall test accuracy
print('Softmax Regression Overall Accuracy: ', end='')
print(1 - sum(sr.error) / 2000)
count = 0
for x, y in zip(sr.error, sr.totals):
    print('Label ' + str(count) + ' Accuracy: ' + str(1 - x / y))
    count += 1
```