REPORT

COMPUTATIONAL INTELLIGENCE

ASHITA DIWAN (6)
SURBHI MITTAL (43)
VINEETA GOSWAMI (48)

PROBLEM

Given a dataset ("train_cancer.h5") containing:

- a training set of 380 records containing diagnostic cancer information labeled as benign (y=1) or malignant(y=0)
- a test set of 189 records labeled as benign or malignant

The problem is to build a simple text recognition algorithm that can correctly classify records as benign or malignant.

DATASET

Breast Cancer Dataset

```
Files used          : cancer_train.h5, cancer_test.h5
Total instances     : 569
Training instances: 380
Testing instances : 189
```

About train_set_x and test_set_x:

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)

The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 1 is Mean Radius, field 11 is Radius SE, field 21 is Worst Radius.

All feature values are recoded with four significant digits.

About train_set_y and test_set_y:
Classes used:

1. Malignant: 0
2. Benign   : 1

ALGORITHM

Build a Logistic Regression, using a Neural Network mindset.
Steps for building a Neural Network are:

1. Define the model structure (such as number of input features)
2. Initialize the model's parameters
3. Loop:
    - Calculate current loss (forward propagation)
    - Calculate current gradient (backward propagation)
    - Update parameters (gradient descent)

LOGISTIC REGRESSION

To train the parameters w and b, we need to define a cost function.
$y\hat{}(i) = \sigma(wTx(i) + b)$, where $\sigma(z(i)) = 1\ 1 + e{-}z(i)$

$Given$ $\{(x(1), y(1)), \cdots, (x(m), y(m))\}$, $we\ want$ $y\hat{}(i) \approx y(i)$

Loss (error) function:
The loss function measures the discrepancy between the prediction ($y\hat{}(i)$) and the desired output ($y(i)$). In other words, the loss function computes the error for a single training example.

$L(y\hat{}(i), y(i)) = 12\ (y\hat{}(i) - y(i))2$

$L(y\hat{}(i), y(i)) = -(\ y(i)\ \log(y\hat{}(i)) + (1 - y(i))\log\ (1 - y\hat{}(i))$

- If $y(i) = 1$: $L(y\hat{}(i), y(i)) = -\log(y\hat{}(i))$ where $\log(y\hat{}(i))$ and $y\hat{}(i)$ should be close to 1
- If $y(i) = 0$: $L(y\hat{}(i), y(i)) = -\log(1 - y\hat{}(i))$ where $\log(1 - y\hat{}(i))$ and $y\hat{}(i)$ should be close to 0

Cost function:
The cost function is the average of the loss function of the entire training set. We are going to find the parameters w and b that minimize the overall cost function. $J(w, b) = 1\ m\ \Sigma\ L(y\hat{}(i), y(i))\ m\ i{=}1 = -\ 1\ m\ \Sigma[(\ y(i)\ \log(y\hat{}(i)) + (1 - y(i))\log\ (1 - y\hat{}(i))]$

OPERATING ENVIRONMENT

Operating System : Windows 10
Machine Architecture: 64-bit
Language : Python 2.7
Packages : numpy, scipy, matplotlib, h5py, PIL

CODE

```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy
from PIL import Image
from scipy import ndimage
from lr_utils import load_dataset

train_set_x_orig, train_set_y, test_set_x_orig, test_set_y, classes =
load_dataset()

m_train = train_set_x_orig.shape[0]
m_test = test_set_x_orig.shape[0]
num_px = train_set_x_orig.shape[1]

print ("Number of training examples: m_train = " + str(m_train))
print ("Number of testing examples: m_test = " + str(m_test))
print ("train_set_x shape: " + str(train_set_x_orig.shape))
print ("train_set_y shape: " + str(train_set_y.shape))
print ("test_set_x shape: " + str(test_set_x_orig.shape))
print ("test_set_y shape: " + str(test_set_y.shape))
print ("classes: " + str(classes))
print ('\n' + "-------------------------------------------------------" +
'\n')


max_val_train = np.max(train_set_x_orig,axis =0) # (1,30)
max_val_test = np.max(test_set_x_orig,axis =0) # (1,30)

train_set_x_norm = train_set_x_orig/max_val_train
test_set_x_norm = test_set_x_orig/max_val_test

train_set_x = train_set_x_norm.T
test_set_x = test_set_x_norm.T

print(train_set_x)

def sigmoid(z):

      s = 1.0/(1.0+np.exp(-z))

      return s


def propagate(w, b, X, Y):

      m = X.shape[1]
```

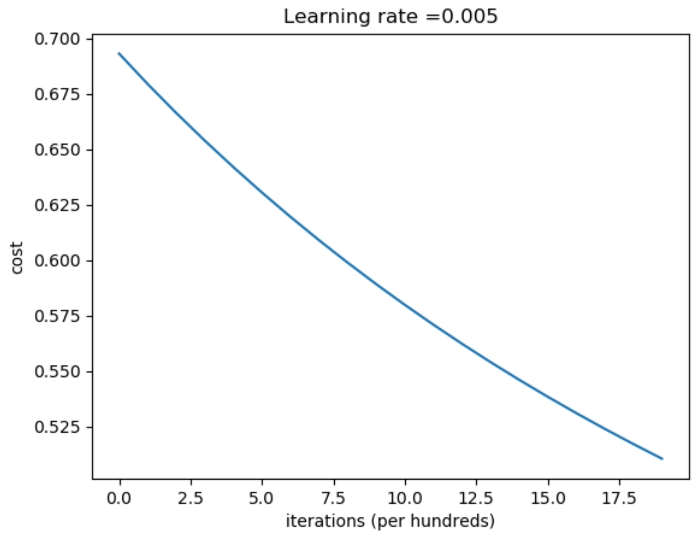
```
    A = sigmoid(np.dot(w.T, X) + b)
    cost = (-1.0/m)*np.sum((Y*np.log(A)+ (1-Y)*np.log(1-A)), axis = 1)
    dw = (1.0/m)*np.dot(X, (A-Y).T)
    db = (1.0/m)*np.sum(A-Y, axis = 1)

    assert(dw.shape == w.shape)
    assert(db.dtype == float)
    cost = np.squeeze(cost)
    assert(cost.shape == ())

    grads = {"dw": dw, "db": db}

    return grads, cost


def optimize(w, b, X, Y, num_iterations, learning_rate, print_cost = False):

    costs = []

    for i in range(num_iterations):

        grads, cost = propagate(w, b, X, Y)
        dw = grads["dw"]
        db = grads["db"]

        w = w - learning_rate*dw
        b = b - learning_rate*db

        if i % 100 == 0:
            costs.append(cost)

        if print_cost and i % 100 == 0:
            print ("Cost after iteration %i: %f" %(i, cost))

    params = {"w": w,
              "b": b}

    grads = {"dw": dw,
             "db": db}

    return params, grads, costs

def predict(w, b, X):

    m = X.shape[1]
    Y_prediction = np.zeros((1,m))
    w = w.reshape(X.shape[0], 1)
```

```
    A = sigmoid(np.dot(w.T, X) + b)

    p = np.zeros(m).reshape(1,m)
    for i in range(A.shape[1]):

        if A[0,i]>0.5:
            Y_prediction[0,i] = 1


    assert(Y_prediction.shape == (1, m))

    return Y_prediction


def model(X_train, Y_train, X_test, Y_test, num_iterations = 2000,
learning_rate = 0.5, print_cost = False):

    w, b = np.zeros(X_train.shape[0]).reshape(X_train.shape[0],1), 0.0

    parameters, grads, costs = optimize(w, b, X_train, Y_train,
num_iterations, learning_rate, print_cost)

    w = parameters["w"]
    b = parameters["b"]

    Y_prediction_test = predict(w, b, X_test)
    Y_prediction_train = predict(w, b, X_train)

    print("train accuracy: {} %".format(100 -
np.mean(np.abs(Y_prediction_train - Y_train)) * 100))
    print("test accuracy: {} %".format(100 -
np.mean(np.abs(Y_prediction_test - Y_test)) * 100))


    d = {"costs": costs,
         "Y_prediction_test": Y_prediction_test,
         "Y_prediction_train" : Y_prediction_train,
         "w" : w,
         "b" : b,
         "learning_rate" : learning_rate,
         "num_iterations": num_iterations}

    return d

#end of functions

d = model(train_set_x, train_set_y, test_set_x, test_set_y, num_iterations =
2000, learning_rate = 0.005, print_cost = True)
```

```
print('\n' + "-------------------------------------------------------" +
'\n')

costs = np.squeeze(d['costs'])
plt.plot(costs)
plt.ylabel('cost')
plt.xlabel('iterations (per hundreds)')
plt.title("Learning rate =" + str(d["learning_rate"]))
plt.show()


learning_rates = [0.01, 0.001, 0.0001]
models = {}
for i in learning_rates:
      print ("learning rate is: " + str(i))
      models[str(i)] = model(train_set_x, train_set_y, test_set_x,
test_set_y, num_iterations = 1500, learning_rate = i, print_cost = True)
      print ('\n' + "-------------------------------------------------------"
+ '\n')

for i in learning_rates:
      plt.plot(np.squeeze(models[str(i)]["costs"]), label=
str(models[str(i)]["learning_rate"]))

plt.ylabel('cost')
plt.xlabel('iterations')

legend = plt.legend(loc='upper center', shadow=True)
frame = legend.get_frame()
frame.set_facecolor('0.90')
plt.show()
```

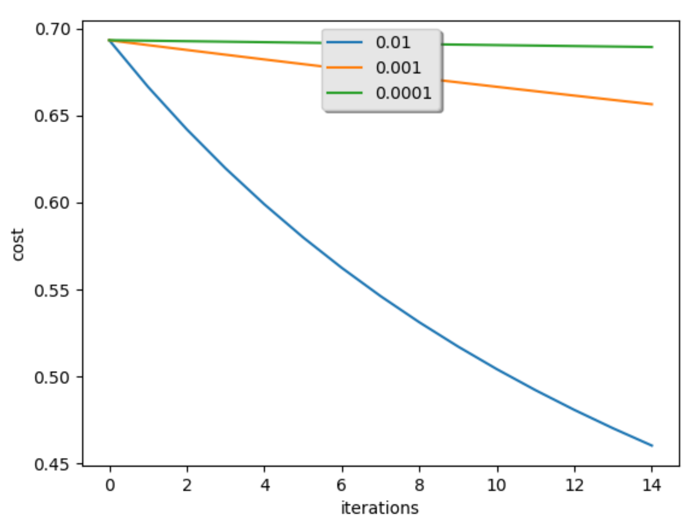
OUTPUT

```
Number of training examples: m_train = 380
Number of testing examples: m_test = 189
train_set_x shape: (380, 30)
train_set_y shape: (1, 380)
test_set_x shape: (189, 30)
test_set_y shape: (1, 189)
classes: [b'maligna' b'benign']

------------------------------------------------------

[[ 0.63998574  0.73176801  0.70046246 ...,  0.47883314  0.48594806
   0.39416575]
 [ 0.26425663  0.45239311  0.54098779 ...,  0.7181772   0.38569248
   0.47937882]
 [ 0.65145892  0.70503974  0.68965518 ...,  0.45564985  0.46827585
   0.38885942]
 ...,
 [ 0.91202742  0.63917524  0.83505154 ...,  0.19865979  0.36219931
   0.86735398]
 [ 0.69313043  0.41428143  0.54429042 ...,  0.40584514  0.51024407
   0.62579089]
 [ 0.57301205  0.42901206  0.42207232 ...,  0.34028918  0.46448195
   0.6761446 ]]
Cost after iteration 0: 0.693147
Cost after iteration 100: 0.679428
Cost after iteration 200: 0.666393
Cost after iteration 300: 0.653930
Cost after iteration 400: 0.641993
Cost after iteration 500: 0.630552
Cost after iteration 600: 0.619583
Cost after iteration 700: 0.609061
Cost after iteration 800: 0.598965
Cost after iteration 900: 0.589274
Cost after iteration 1000: 0.579968
Cost after iteration 1100: 0.571027
Cost after iteration 1200: 0.562433
Cost after iteration 1300: 0.554169
Cost after iteration 1400: 0.546218
Cost after iteration 1500: 0.538564
Cost after iteration 1600: 0.531193
Cost after iteration 1700: 0.524091
Cost after iteration 1800: 0.517244
Cost after iteration 1900: 0.510640
train accuracy: 92.10526315789474 %
test accuracy: 94.17989417989418 %
```




```
-------------------------------------------------------

learning rate is: 0.01
Cost after iteration 0: 0.693147
Cost after iteration 100: 0.666390
Cost after iteration 200: 0.641988
Cost after iteration 300: 0.619576
Cost after iteration 400: 0.598957
Cost after iteration 500: 0.579958
Cost after iteration 600: 0.562423
Cost after iteration 700: 0.546207
Cost after iteration 800: 0.531182
Cost after iteration 900: 0.517232
Cost after iteration 1000: 0.504255
Cost after iteration 1100: 0.492158
Cost after iteration 1200: 0.480859
Cost after iteration 1300: 0.470285
Cost after iteration 1400: 0.460370
train accuracy: 92.10526315789474 %
test accuracy: 95.76719576719577 %

-------------------------------------------------------

learning rate is: 0.001
Cost after iteration 0: 0.693147
Cost after iteration 100: 0.690338
Cost after iteration 200: 0.687564
Cost after iteration 300: 0.684823
Cost after iteration 400: 0.682112
Cost after iteration 500: 0.679429
Cost after iteration 600: 0.676773
```

```
Cost after iteration 700: 0.674142
Cost after iteration 800: 0.671536
Cost after iteration 900: 0.668954
Cost after iteration 1000: 0.666395
Cost after iteration 1100: 0.663859
Cost after iteration 1200: 0.661345
Cost after iteration 1300: 0.658853
Cost after iteration 1400: 0.656383
train accuracy: 92.10526315789474 %
test accuracy: 88.35978835978835 %

-------------------------------------------------------

learning rate is: 0.0001
Cost after iteration 0: 0.693147
Cost after iteration 100: 0.692865
Cost after iteration 200: 0.692582
Cost after iteration 300: 0.692300
Cost after iteration 400: 0.692019
Cost after iteration 500: 0.691738
Cost after iteration 600: 0.691457
Cost after iteration 700: 0.691177
Cost after iteration 800: 0.690897
Cost after iteration 900: 0.690617
Cost after iteration 1000: 0.690338
Cost after iteration 1100: 0.690059
Cost after iteration 1200: 0.689781
Cost after iteration 1300: 0.689502
Cost after iteration 1400: 0.689225
train accuracy: 84.21052631578948 %
test accuracy: 73.01587301587301 %

-------------------------------------------------------
```



BIBLIOGRAPHY

1. Coursera - Deep Leaning AI by Andrew NG
2. UCI - Machine Learning Repository
3. www.google.com