Q2,3 - Zhangsheng Lai (1002554)

October 7, 2018

Q2. Logistic regression algorithm using stochastic gradient descent to perform binary classification

In [1]:
```
import cv2
import os
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
sns.set()

from utils import *
%matplotlib inline
```

In [2]:
```
def load_data(path, feature = 'raw'):
    """
    Loads data into pixel values from the list of path given. Returns
    Input:
    - path (list): list of path to load the data from.
    - feature: either 'raw' or 'hist' for raw pixel values and 3D histogram respective
    """
    x_train=[]
    y_train=[]
    for c,i in enumerate(path):
        os.chdir(i)
        l = os.listdir()
        for i in l:
            if feature == 'raw':
                vf = convert2pixel_value(i)
            else:
                vf = convert2color_3Dhist(i)
            x_train.append(vf)
            y_train.append(c)

    x_train = np.concatenate([i[np.newaxis] for i in x_train])
    y_train = np.array(y_train)

    # comment below to remove the shuffling of the data
    arr = np.arange(x_train.shape[0])
```

```
        np.random.shuffle(arr)
        x_train = x_train[arr]
        y_train = y_train[arr]

        return x_train, y_train
```

Define the path to the directories containing the images then load the data set using `load_data`.

```
In [3]: train_bird = "C:/Users/zlai/Documents/repo/HomeworkTex/ML/hw/homework 1/data/train/bird
        train_cat = "C:/Users/zlai/Documents/repo/HomeworkTex/ML/hw/homework 1/data/train/cat"
        test_bird = "C:/Users/zlai/Documents/repo/HomeworkTex/ML/hw/homework 1/data/test/bird"
        test_cat = "C:/Users/zlai/Documents/repo/HomeworkTex/ML/hw/homework 1/data/test/cat"
```

Load the data from `bird` and `cat` folders, and we shall let `x1`, `y1` refer to the raw pixel value feature and x2, y2 refer to the feature obtained by using the 3D histogram in this jupyter notebook.

```
In [4]: x1_train, y1_train = load_data([train_cat, train_bird])
        x1_test, y1_test = load_data([test_cat, test_bird])

In [5]: x1_train.shape

Out[5]: (40, 3072)
```

We do some preprocessing of the training data by normalizing the pixel values to between $[0, 1]$ and the labels to be $\{-1, +1\}$.

```
In [6]: def add_bias(dataset):
            """
            Add a one to each sample for bias. Dataset must be of the
            form rows: samples, columns: features
            """
            n, m = dataset.shape
            out = np.ones((n, m+1))
            out[:,:-1] = dataset
            return out

In [7]: x1_train = x1_train/255
        x1_test = x1_test/255
        y1_train = y1_train*2 - 1
        y1_test = y1_test*2 - 1

In [8]: x1_train = add_bias(x1_train)
        x1_test = add_bias(x1_test)

In [9]: print (x1_train.shape[0], 'training samples')
        print (x1_test.shape[0], 'test samples')

40 training samples
40 test samples
```

Using features obtained from the raw pixels to do the logistic loss

In [10]:
```
def sigmoid(x):
    """
    Applies the sigmoid function on the given vector.
    Input(s):
    - x : numpy vector of values
    """
    return 1/(1+np.exp(-x))
```

In [11]:
```
def initialize_params(size=3073, seed=123):
    """
    Initialize parameters W weights and b biases.
    Input(s):
    - size (int): size of the parameters
    - seed (int): seed for the random number generator
    """
    rng = np.random.RandomState(seed)

    return rng.normal(size=(size,))
```

In [12]:
```
def log_loss(x_train, y_train, W):
    """
    Computes the loss value of the logistic loss.
    Input(s):
    - x_train, y_train: training data and labels. x_train takes
    different forms depending on the features used and y_train
    is {-1,+1}.
    - W: value of the parameters.
    """
    z = y_train * np.dot(x_train, W)
    h = sigmoid(z)

    return -np.mean(np.log(h))
```

In [13]:
```
def log_grad(x_train, y_train, W):
    """
    Computes the gradient of the logistic loss function.
    Input(s):
    - x_train, y_train: training data and labels. x_train takes
    different forms depending on the features used and y_train
    is {-1,+1}.
    - W: value of the parameters.
    """
    z = y_train * np.dot(x_train, W)
    h = sigmoid(z)
    n = x_train.shape[0]

    return 1/n * np.dot(x_train.T,(y_train * (h-1)))
```

```
In [14]: def next_batch(x_train, y_train, batch_size=2):
             """
             Returns a batch of size batch_size for stochastic gradient descent.
             - x_train, y_train: training data and labels. x_train takes different
             forms depending on the features used and y_train is {-1,+1}.
             - batch_size (int): size of each batch.
             """
             for i in np.arange(0, x_train.shape[0], batch_size):
                 yield (x_train[i:i+batch_size],y_train[i:i+batch_size])

In [15]: def log_classifier(x, learnt_W):
             """
             Takes in the test set and learnt parameters and returns the
             accuracy of the classifier on the test set.
             Inputs:
             - x: data for classification
             - learnt_W: learnt parameters
             """
             return (sigmoid(np.dot(x, learnt_W)) >= .5) * 2 - 1

In [16]: def log_accuracy(x, y, learnt_W):
             """
             Returns the accuracy of the model with parameters learnt_W.
             Input(s):
             - x: data for classification
             - y: corresponding labels to the data x
             - learnt_W: learnt parameters
             """
             output = log_classifier(x, learnt_W)
             return np.sum(np.absolute(y - output) == 0)/y.shape[0]

In [17]: def log_train(x_train, y_train, x_test, y_test, W, alpha=0.01, batch_size = 4, epoch =
             """
             Trains the log loss model with given learning rate alpha, batch_size, epoch and i
             Returns the history of the loss, train accuracy, test accuracy and the value of t
             at different epochs.
             Input(s):
             - x_train, y_train: train data and labels
             - x_test, y_test: test data and labels
             - W: parameters of the model
             - alpha: learning rate
             - batch_size: batch size for stochastic gradient descent
             - epoch: number of times the dataset is passed through the model.
             """
             loss_history = []
             train_acc_history = []
             test_acc_history = []
             W_history = []
```