University at Buffalo
The State University of New York

CSE574 Introduction to Machine Learning

Project 3

Amir Baghdadi
50135018

Introduction

Logistic regression (LR) is a model for classification of categorical dependent variables. In multiclass LR (MLR), in spite of regular LR in which the output can take only two values (“0” and “1”), the dependent variable has more than two outcome categories. LR has various applications including trauma and injury score or prediction of a disease suffering in a person in medical field or the prediction of whether an American voter will vote Democratic or Republican based on age gender, state of residence, race, etc.

Artificial Neural Network (ANN), is inspired by biological neural networks in brain that learns and progressively improves performance in performing tasks by training through non task-specific programming. An ANN is a collection of connected units called artificial neurons. Each connection transmits a signal to another neuron and the receiving neuron can process all the signals from downstream neurons connected to it. The signal in output layer will determine the desired state of the variable to be classified.

The purpose of this project is to implement a multiclass LR and a single hidden layer ANN classification algorithms on MNIST dataset and evaluate the performance on both MNIST and USPS handwritten digit data to identify them among 0, 1, 2, … , 9. The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems. The database contains 60,000 training images and 10,000 testing images. The images are centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field. Each digit in the USPS dataset has 150 samples of “.png” file available for testing that need to be imported and converted into MNIST data format. The evaluation will be performed by calculating classification error rate in under the one-hot coding scheme.

Tasks

Extracting feature values and labels from the data:

The features for MNIST were imported in Python using tensorflow import command.

```
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

np.shape(mnist.train.images) = (55000, 784)

np.shape(mnist.test.images) = (10000, 784)
```

The USPS data were also imported as image and were resized to match the 28x28 (=784) format of data in MNIST. In the next step, each pixel value were smoothed by taking the average of all adjacent pixel to have intensity value for the pixels similar to MNIST dataset rather than only binary value of 0 or 1 (0 or 255 for the original USPS data).

The MNIST data consisted of 55000 data samples for training and 10000 for testing with 784 features and the USPS testing data consisted of 1500 samples with the same number of features after the aforementioned preprocessing.

Data partition:

The MNIST dataset is originally partitioned into a training set and a testing set and the only the testing portion of USPS data was imported with the detailed number of samples explained above.

Train model parameter using the selected hyper parameters and predict:

Multiclass LR:

After tuning the hyper parameters, i.e., learning rate, the value of 0.5 was selected to train the model in multiclass LR with softmax activation function:

$$p(C_k|\mathbf{x}) = y_k(\mathbf{x}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}\,,$$

and cross-entropy error function:

$$E(\mathbf{x}) = -\sum_{k=1}^{K} t_k \ln y_k.$$

Gradient descent algorithm was applied to modify the variables and reduce the loss function. The number of iterations was considered 1000.

Single hidden layer ANN:

For the given values of hyper parameters, i.e., single hidden layer and 784 nodes, the model was trained in ANN with softmax activation function and stochastic gradient descent to train the neural network. The number of iterations was considered 2000 for testing on MNIST dataset and 10000 for USPS dataset. The model was trained to reduce the cross-entropy error function with a learning rate of 0.0001.

- Backpropagation:

  Backpropagation was also performed to adjust the weights of neurons by calculating the gradient of the loss function. This improves the computational efficiency. The number of iterations were 3000 in this case with the learning rate of 0.25 in gradient descent for optimizing the weights.

- Convolutional NN:

  Convolutional neural network (CNN) was also performed on the data using the Tensorflow CNN package with the predefined hype parameters. The iteration number was considered 2000 for testing on MNIST dataset and 10000 for USPS dataset.

Bayesian LR:

Bayesian LR was applied on the data, hyper parameters were initialized randomly and updated by minimizing the error function by Newton-Raphson method using the Hessian matrix:

$$\mathrm{H} = \sum_{n=1}^{N} y_n(1-y_n)\phi_n\phi_n{}^{\mathrm{T}} = \Phi^{\mathrm{T}}\mathrm{R}\Phi$$

$$\omega^{(\mathrm{new})} = \omega^{(\mathrm{old})} - H^{-1}\nabla E(\omega)$$

Prediction were performed by evaluating the mean and covariance of $a = \omega^{\mathrm{T}}\phi$:

$$\mu_a = \omega_{map}^{\mathrm{T}}\phi$$

$$\sigma_a^2 = \phi^T S_N \phi \qquad\qquad \kappa(\sigma^2) = (1+\pi\sigma^2/8)^{-1/2}$$

Logistic sigmoid function were applied on $\kappa(\sigma^2)\mu_a$ and the class with highest probability will be the predicted class.

Results:

MLR and CNN models were trained on MNIST data with both training and testing accuracy of ~92% in MLR and ~98% in CNN, however, the results of testing on USPS data ends up in weak performances of ~31% in MLR and 68% in CNN. This supports the **"No Free Lunch" theorem** stating that there is no one model that works best for every problem. In other words, the assumptions of a great model for one problem (in this case for MNIST data) may not hold for another problem (in this case USPS data), so we should try multiple models and find one that works best for USPS data, or train the current models on this dataset.

In general ANN methods as semi-parametric methods out performs LR methods due to having advantages that includes allowing a large number of variables in the model, no need to assumptions such as normality, finding the models despite missing data, and detection of complex and nonlinear relationship between independent and dependent variables. Logistic regression model is mainly influenced by the sample size, number of independent variables, potential multicollinearity and missing [1]. In USPS dataset, the number of features was large in compare to the number of sample size which is another reason for the poor performance of multiclass LR in addition to the above mentioned rationales.

[1] Teshnizi SH, Ayatollahi SMT. A Comparison of Logistic Regression Model and Artificial Neural Networks in Predicting of Student's Academic Failure. *Acta Informatica Medica*. 2015; 23(5):296-300. doi:10.5455/aim.2015.23.296-300.

Multiclass LR and CNN results:

Results for MNIST data:

```
UBitName = amirbagh
personNumber = 50135018
```

Multiclass LR testing accuracy:

```
0.9184
```

Single hidden layer ANN training and testing accuracy on MNIST:

```
step 0, training accuracy 0.08

test accuracy 0.1023

test accuracy 0.1222

.

.

test accuracy 0.8258

test accuracy 0.8248

step 100, training accuracy 0.76

test accuracy 0.8255

test accuracy 0.832

.

.

test accuracy 0.9018

test accuracy 0.9036

step 200, training accuracy 0.9

test accuracy 0.9023

test accuracy 0.9043

.

.

test accuracy 0.9744

test accuracy 0.9746

step 1900, training accuracy 0.98

test accuracy 0.975
test accuracy 0.9756

test accuracy 0.9752

.

.

test accuracy 0.9751

test accuracy 0.9752

test accuracy 0.976

test accuracy 0.9762

test accuracy 0.9767
```

##

<u>Results for USPS data:</u>

```
UBitName = amirbagh
personNumber = 50135018
```

Multiclass LR testing accuracy:

```
0.314667
```

Single hidden layer ANN training accuracy on MNIST and testing accuracy on USPS:

```
step 0, training accuracy 0.08
test accuracy 0.115333
test accuracy 0.119333
.
.
test accuracy 0.357333
test accuracy 0.358667
step 100, training accuracy 0.78
test accuracy 0.354667
test accuracy 0.350667
.
.
test accuracy 0.396667
test accuracy 0.4
test accuracy 0.406667
step 200, training accuracy 0.86
test accuracy 0.409333
test accuracy 0.412
test accuracy 0.416667
.
.
step 9100, training accuracy 1
test accuracy 0.670667
test accuracy 0.671333
```

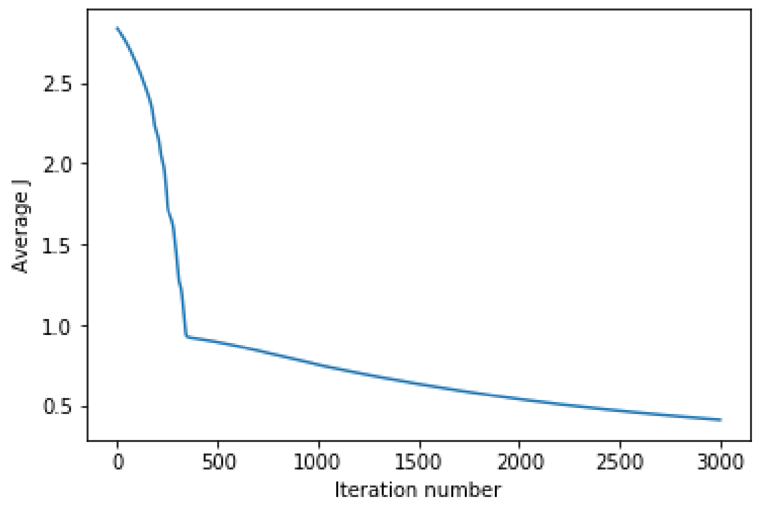
```
.
.
test accuracy 0.679333
test accuracy 0.677333
step 9200, training accuracy 0.98
test accuracy 0.677333
test accuracy 0.678
.
.
test accuracy 0.676667
test accuracy 0.675333
step 9300, training accuracy 0.98
test accuracy 0.676667
test accuracy 0.677333
.
.
test accuracy 0.682
test accuracy 0.685333
step 9400, training accuracy 0.98
test accuracy 0.688667
test accuracy 0.690667
.
.
test accuracy 0.679333
test accuracy 0.68
step 9500, training accuracy 1
test accuracy 0.680667
test accuracy 0.681333
.
.
test accuracy 0.666667
test accuracy 0.666
step 9600, training accuracy 1
test accuracy 0.664
test accuracy 0.662
.
.
test accuracy 0.657333
test accuracy 0.653333
step 9700, training accuracy 1
test accuracy 0.650667
test accuracy 0.648667
.
.
test accuracy 0.68
test accuracy 0.68
```

```
step 9800, training accuracy 0.98
test accuracy 0.680667
test accuracy 0.680667
.
.
step 9900, training accuracy 0.98
test accuracy 0.667333
test accuracy 0.665333
.
.
test accuracy 0.684
test accuracy 0.684667
test accuracy 0.684667
test accuracy 0.688
```

Single hidden layer NN with Back Propagation:

Results for training and testing on MNIST data:

```
Starting gradient descent for 3000 iterations
Iteration 0 of 3000
Iteration 1000 of 3000
Iteration 2000 of 3000
```



Plot of losses for training

```
test accuracy: 0.88734335326
```

##

Results for training on MNIST and testing on USPS data:

```
UBitName = amirbagh
personNumber = 50135018

Starting gradient descent for 3000 iterations
Iteration 0 of 3000
Iteration 1000 of 3000
Iteration 2000 of 3000

test accuracy: 0.156666666667
```

Bayesian LR results:

Results for MNIST data:

```
test accuracy: 0.11
```

##

Results for USPS data:

```
test accuracy: 0.12
```

```
# coding: utf-8

# In[2]:


# Project 3

print('UBitName = amirbagh')
print('personNumber = 50135018')

import numpy as np
from PIL import Image
import glob
from scipy import ndimage

# download and read the MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# read the USPS data
uspsTest_images_1d = []
new_width  = 28
new_height = 28
for filename in glob.glob('USPSTest/*.png'):
    im=Image.open(filename)
    im_resize = im.resize((new_width, new_height), Image.ANTIALIAS)
    im_resize_smooth = ndimage.generic_filter(im_resize, np.nanmean, size=3, 
mode='constant', cval=np.NaN)
    im_1d = np.array(im_resize_smooth, dtype='f').ravel()/-255+1
    uspsTest_images_1d.append(im_1d)

uspsTest_images = np.reshape(uspsTest_images_1d, 
    (int(np.size(uspsTest_images_1d)/(new_width*new_height)), 
(new_width*new_height)))

uspsDigits = list(range(10))
uspsDigits.reverse()
uspsLabels = np.repeat(uspsDigits, 150)
```

```
uspsLabels_oneHot = np.zeros((1500, 10))
uspsLabels_oneHot[np.arange(1500), uspsLabels] = 1

# comment/uncomment for MNIST or USPS test data
trainData = mnist.train.images
#testData = mnist.test.images
testData = uspsTest_images
trainLabel = mnist.train.labels
#testLabel = mnist.test.labels
testLabel = uspsLabels_oneHot


import tensorflow as tf

# Single Hidden Layer Neral Network

x = tf.placeholder(tf.float32, [None, 784])  # input
W = tf.Variable(tf.zeros([784, 10]))         # Parameter Variable
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)       # Softmax regression model

y_ = tf.placeholder(tf.float32, [None, 10])  # add a new placeholder

cross_entropy = tf.reduce_mean(              # cross-entropy function
                -tf.reduce_sum(y_ * tf.log(y),
                reduction_indices=[1]))

# optimize variables and reduce the loss
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

# launch the model in an InteractiveSession
sess = tf.InteractiveSession()

# create an operation to initialize the variables
tf.global_variables_initializer().run()
# we'll run the training step 1000 times
for _ in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

```
# check if our prediction matches the truth
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
# to determine what fraction are correct, we
# cast to floating point numbers and then
# take the mean
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
# finally, we ask for our accuracy on our
# test data
print(sess.run(accuracy, feed_dict={x: testData, y_: testLabel}))



# Convolutional Neural Network

# Weight Initialization
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

# Convolution and Pooling
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
    strides=[1, 2, 2, 1], padding='SAME')

# First Convolutional Layer
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
x_image = tf.reshape(x, [-1, 28, 28, 1])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

# Second Convolutional Layer
W_conv2 = weight_variable([5, 5, 32, 64])
```

```
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) +
b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

# Densely Connected Layer
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
# Dropout
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
# Readout Layer
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

# Train and Evaluate the Model
cross_entropy = tf.reduce_mean(
tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(10000):
        batch = mnist.train.next_batch(50)
        if i % 100 == 0:
            train_accuracy = accuracy.eval(feed_dict={
                x: batch[0], y_: batch[1], keep_prob: 1.0})
            print('step %d, training accuracy %g' % (i, train_accuracy))
        train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
        print('test accuracy %g' % accuracy.eval(feed_dict={
            x: testData, y_: testLabel, keep_prob: 1.0}))
```

```
# coding: utf-8

# In[1]:


# Project 3 Back Propagation

print('UBitName = amirbagh')
print('personNumber = 50135018')

# test size
test_size=0.4

from sklearn.datasets import load_digits
digits = load_digits()
import matplotlib.pyplot as plt
from PIL import Image
import glob
from scipy import ndimage
#print(digits.data.shape)
#plt.gray()
#plt.matshow(digits.images[1])
#plt.show()


# Normalize Data
from sklearn.preprocessing import StandardScaler
X_scale = StandardScaler()
X = X_scale.fit_transform(digits.data)

# Split Data
from sklearn.model_selection import train_test_split
y = digits.target
X_train, X_test_mnist, y_train, y_test_mnist = train_test_split(X, y,
test_size=test_size)

# Convert to one-hot representation
import numpy as np
```

```
def convert_y_to_vect(y):
    y_vect = np.zeros((len(y), 10))
    for i in range(len(y)):
        y_vect[i, y[i]] = 1
    return y_vect
y_v_train = convert_y_to_vect(y_train)
y_v_test_mnist = convert_y_to_vect(y_test_mnist)


# read the USPS data
uspsTest_images_1d = []
new_width  = 8
new_height = 8
for filename in glob.glob('USPSTest/*.png'):
    im=Image.open(filename)
    im_resize = im.resize((new_width, new_height), Image.ANTIALIAS)
    im_resize_smooth = ndimage.generic_filter(im_resize, np.nanmean, size=3,
mode='constant', cval=np.NaN)
    im_1d = np.array(im_resize_smooth, dtype='f').ravel()/-255+1
    uspsTest_images_1d.append(im_1d)

X_test_usps = StandardScaler().fit_transform(np.reshape(uspsTest_images_1d,
    (int(np.size(uspsTest_images_1d)/(new_width*new_height)),
(new_width*new_height))))


uspsDigits = list(range(10))
uspsDigits.reverse()
y_test_usps = np.repeat(uspsDigits, 150)
y_v_test_usps = np.zeros((1500, 10))
y_v_test_usps[np.arange(1500), y_test_usps] = 1

# comment/uncomment for using USPS or MNIST data
#X_test = X_test_usps
#y_v_test =y_v_test_usps
#y_test = y_test_usps

X_test = X_test_mnist
y_v_test =y_v_test_mnist
```

```
y_test = y_test_mnist


# Creating the Neural Network (one hidden layer)
nn_structure = [64, 30, 10]

# Sigmoid activation function
def f(x):
    return 1 / (1 + np.exp(-x))
def f_deriv(x):
    return f(x) * (1 - f(x))

# Randomly initialise the weights
import numpy.random as r
def setup_and_init_weights(nn_structure):
    W = {}
    b = {}
    for l in range(1, len(nn_structure)):
        W[l] = r.random_sample((nn_structure[l], nn_structure[l-1]))
        b[l] = r.random_sample((nn_structure[l],))
    return W, b

# Initialize the weight and bias values
def init_tri_values(nn_structure):
    tri_W = {}
    tri_b = {}
    for l in range(1, len(nn_structure)):
        tri_W[l] = np.zeros((nn_structure[l], nn_structure[l-1]))
        tri_b[l] = np.zeros((nn_structure[l],))
    return tri_W, tri_b

# Feed forward pass
def feed_forward(x, W, b):
    h = {1: x}
    z = {}
    for l in range(1, len(W) + 1):
        # if it is the first layer, then the input into the weights is x,
otherwise,
        # it is the output from the last layer
        if l == 1:
            node_in = x
        else:
            node_in = h[l]
        z[l+1] = W[l].dot(node_in) + b[l] # z^(l+1) = W^(l)*h^(l) + b^(l)
        h[l+1] = f(z[l+1]) # h^(l) = f(z^(l))
    return h, z

# calculate the output layer delta δ(nl) and hidden layer delta δ(l)
def calculate_out_layer_delta(y, h_out, z_out):
    # delta^(nl) = -(y_i - h_i^(nl)) * f'(z_i^(nl))
    return -(y-h_out) * f_deriv(z_out)

def calculate_hidden_delta(delta_plus_1, w_l, z_l):
    # delta^(l) = (transpose(W^(l)) * delta^(l+1)) * f'(z^(l))
    return np.dot(np.transpose(w_l), delta_plus_1) * f_deriv(z_l)

# Train NN with backpropagation of errors
def train_nn(nn_structure, X, y, iter_num=3000, alpha=0.25):
    W, b = setup_and_init_weights(nn_structure)
    cnt = 0
    m = len(y)
    avg_cost_func = []
    print('Starting gradient descent for {} iterations'.format(iter_num))
    while cnt < iter_num:
        if cnt%1000 == 0:
            print('Iteration {} of {}'.format(cnt, iter_num))
        tri_W, tri_b = init_tri_values(nn_structure)
        avg_cost = 0
        for i in range(len(y)):
            delta = {}
            # perform the feed forward pass and return the stored h and z
values, to be used in the
            # gradient descent step
            h, z = feed_forward(X[i, :], W, b)
            # loop from nl-1 to 1 backpropagating the errors
            for l in range(len(nn_structure), 0, -1):
                if l == len(nn_structure):
                    delta[l] = calculate_out_layer_delta(y[i,:], h[l], z[l])
                    avg_cost += np.linalg.norm((y[i,:]-h[l]))
                else:
                    if l > 1:
                        delta[l] = calculate_hidden_delta(delta[l+1], W[l],
z[l])
                    # triW^(l) = triW^(l) + delta^(l+1) * transpose(h^(l))
                    tri_W[l] += np.dot(delta[l+1][:,np.newaxis],
np.transpose(h[l][:,np.newaxis]))
                    # trib^(l) = trib^(l) + delta^(l+1)
                    tri_b[l] += delta[l+1]
        # perform the gradient descent step for the weights in each layer
        for l in range(len(nn_structure) - 1, 0, -1):
            W[l] += -alpha * (1.0/m * tri_W[l])
            b[l] += -alpha * (1.0/m * tri_b[l])
        # complete the average cost calculation
        avg_cost = 1.0/m * avg_cost
        avg_cost_func.append(avg_cost)
        cnt += 1
    return W, b, avg_cost_func

# Run training and plot
W, b, avg_cost_func = train_nn(nn_structure, X_train, y_v_train)

plt.plot(avg_cost_func)
plt.ylabel('Average J')
plt.xlabel('Iteration number')
plt.show()

# Testing the NN
def predict_y(W, b, X, n_layers):
    m = X.shape[0]
    y = np.zeros((m,))
    for i in range(m):
        h, z = feed_forward(X[i, :], W, b)
        y[i] = np.argmax(h[n_layers])
    return y

from sklearn.metrics import accuracy_score
y_pred = predict_y(W, b, X_test, 3)
```

```
print('test accuracy:', accuracy_score(y_test, y_pred))
```

proj3code_bayesian.py

```
# coding: utf-8

# In[5]:


# Project 3 Bayesian LR

print('UBitName = amirbagh')
print('personNumber = 50135018')

import numpy as np
import math
from sklearn.cluster import KMeans
from random import randint
import random
from PIL import Image
import glob
from scipy import ndimage

# Read MNIST Data
from tensorflow.examples.tutorials.mnist import input_data
mnist_onehot = input_data.read_data_sets("/tmp/data/", one_hot=True)
mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)


# read the USPS data
uspsTest_images_1d = []
new_width  = 28
new_height = 28
for filename in glob.glob('USPSTest/*.png'):
    im=Image.open(filename)
    im_resize = im.resize((new_width, new_height), Image.ANTIALIAS)
    im_resize_smooth = ndimage.generic_filter(im_resize, np.nanmean, size=3,
mode='constant', cval=np.NaN)
    im_1d = np.array(im_resize_smooth, dtype='f').ravel()/-255+1
    uspsTest_images_1d.append(im_1d)

uspsTest_images = np.reshape(uspsTest_images_1d,
    (int(np.size(uspsTest_images_1d)/(new_width*new_height)),
(new_width*new_height)))

uspsDigits = list(range(10))
uspsDigits.reverse()
uspsLabels = np.repeat(uspsDigits, 150)
uspsLabels_oneHot = np.zeros((1500, 10))
uspsLabels_oneHot[np.arange(1500), uspsLabels] = 1

mnist_train_size = 5000
mnist_test_size = 1000

mnist_train_rand_idx = random.sample(range(1,
mnist_onehot.train.images.shape[0]), mnist_train_size)
mnist_test_rand_idx = random.sample(range(1,
mnist_onehot.test.images.shape[0]), mnist_test_size)

# comment/uncomment for MNIST or USPS test data
X_train = mnist_onehot.train.images[mnist_train_rand_idx]
#X_test = mnist_onehot.test.images[mnist_test_rand_idx]
X_test = uspsTest_images

y_v_train = mnist_onehot.train.labels[mnist_train_rand_idx]
#y_v_test = mnist_onehot.test.labels[mnist_test_rand_idx]
y_v_test =uspsLabels_oneHot

y_train = mnist.train.labels[mnist_train_rand_idx].reshape(mnist_train_size,1)
#y_test = mnist.test.labels[mnist_test_rand_idx].reshape(mnist_test_size,1)
y_test = uspsLabels

input_data_train = X_train
output_data_train = y_v_train

input_data_test = X_test
output_data_test = y_test

# Hyper parameters acquisition
def hyper_para_bayes_logistic(m_0, S_0, Theta, y):
    w_map = m_0
    S_N = np.linalg.inv(S_0)
    Theta = Theta.T
    for i in range(Theta.shape[0]):
        S_N = S_N + y[i]*(1-y[i])*np.matmul(Theta[i].T, Theta[i])
    return w_map, S_N

def pred_bayes_logistic(w_map, S_N, theta):
    mu_a = np.dot(w_map.T, theta)
    var_a = np.dot(np.dot(theta.T, S_N), theta)
    kappa_var = (1 + math.pi*var_a/8)**(-0.5)
    x = kappa_var*mu_a
    return 1/(1 + np.exp(-x))

def training(Theta, y):
    w0 = np.random.normal(0, 1, Theta.shape[1])
    S0 = np.diag(np.random.normal(0, 1, Theta.shape[1]))
    # Theta n*m (n samples, m features), y n*1
    w_map, S_N = hyper_para_bayes_logistic(w0, S0, Theta, y)
    return w_map, S_N

def predict(w_map, S_N, theta):
    return pred_bayes_logistic(w_map, S_N, theta)

# Multi-class classification
def multiclass_sigmoid_logistic(Theta, Y):
    n_class = Y.shape[1]
    n_sample = Theta.shape[0]
    n_feature = Theta.shape[1]
    models_w = np.zeros((n_class,n_feature))
    models_s = np.zeros((n_class,n_feature,n_feature))
    for i in range(n_class):
        w_m , s_n = training(Theta, Y[:, i])
        models_w[i] = w_m
        models_s[i] = s_n
    return models_w, models_s

# Prediction
def pred_multiclass_sigmoid_logistic(theta, Theta, Y):
    models_w, models_s = multiclass_sigmoid_logistic(Theta, Y)
    props = []
    for i in range(len(models_w)):
        props.append(predict(models_w[i], models_s[i], theta))
    return np.nanmax(props), np.nanargmax(props), props

idx_v = []
max_props_v = []
max_props_idx_v = []
props_v = []

for t in range(0, 100):
    idx = randint(1, np.shape(input_data_test)[0]-1)
    max_props, max_props_idx, props =
pred_multiclass_sigmoid_logistic(theta=input_data_test[idx],

Theta=input_data_train,

Y=output_data_train)
    idx_v.append(idx)
    max_props_v.append(max_props)
    max_props_idx_v.append(max_props_idx)
    props_v.append(props)

true_testlabels = output_data_test[idx_v]
pred_testlabels = max_props_idx_v

from sklearn.metrics import accuracy_score
print('test accuracy:', accuracy_score(true_testlabels, pred_testlabels))
```