Practica 4: Multi-class Classification and Neural Networks

Alberto Muñoz Fernández

Óscar García Castro

```python
# one-vs-all
#
def oneVsAll(X, y, n_labels, lambda_):
    """
     Trains n_labels logistic regression classifiers and returns
     each of these classifiers in a matrix all_theta, where the i-th
     row of all_theta corresponds to the classifier for label i.

     Parameters
     ----------
     X : array_like
         The input dataset of shape (m x n). m is the number of
         data points, and n is the number of features.

     y : array_like
         The data labels. A vector of shape (m, ).

     n_labels : int
         Number of possible labels.

     lambda_ : float
         The logistic regularization parameter.

     Returns
     -------
     all_theta : array_like
         The trained parameters for logistic regression for each class.
         This is a matrix of shape (K x n+1) where K is number of classes
         (ie. `n_labels`) and n is number of features without the bias.
     """
    all_theta = np.zeros((n_labels, X.shape[1] + 1))

    for i in range(n_labels):
        w = np.zeros(len(X[0]))
        b = 0
        y_aux = np.where(y == i, 1, 0)
        wf, bf, jHis = lgr.gradient_descent(X, y_aux, w, b,
        lgr.compute_cost_reg, lgr.compute_gradient_reg, 1, 1500, lambda_)
        all_theta[i, 0] = bf
        all_theta[i, 1:] = wf
```

```
    return all_theta


def predictOneVsAll(all_theta, X):
    """
    Return a vector of predictions for each example in the matrix X. 
    Note that X contains the examples in rows. all_theta is a matrix 
where
    the i-th row is a trained logistic regression theta vector for the 
    i-th class. You should set p to a vector of values from 0..K-1 
    (e.g., p = [0, 2, 0, 1] predicts classes 0, 2, 0, 1 for 4 examples) .

    Parameters
    ----------
    all_theta : array_like
        The trained parameters for logistic regression for each class.
        This is a matrix of shape (K x n+1) where K is number of classes
        and n is number of features without the bias.

    X : array_like
        Data points to predict their labels. This is a matrix of shape 
        (m x n) where m is number of data points to predict, and n is 
number 
        of features without the bias term. Note we add the bias term for 
X in 
        this function. 

    Returns
    -------
    p : array_like
        The predictions for each data point in X. This is a vector of 
shape (m, ).
    """
    p = np.zeros(len(X))

    p = np.argmax(lgr.sigmoid(X @ all_theta[:, 1:].T + all_theta[:, 0]), 
1)

    return p


##########################################################################
# NN
#
def predict(theta1, theta2, X):
    """
    Predict the label of an input given a trained neural network.
```

```
    Parameters
    ----------
    theta1 : array_like
        Weights for the first layer in the neural network.
        It has shape (2nd hidden layer size x input size)

    theta2: array_like
        Weights for the second layer in the neural network.
        It has shape (output layer size x 2nd hidden layer size)

    X : array_like
        The image inputs having shape (number of examples x image
dimensions).

    Return
    ------
    p : array_like
        Predictions vector containing the predicted label for each
example.
        It has a length equal to the number of examples.
    """
    #X_b = deepcopy(X)                        Size para este caso:
    a1 = np.c_[np.ones(len(X)), X]           #Size(5000 * 401)
    z2 = np.dot(theta1, a1.T)
    a2 = lgr.sigmoid(z2)
    a2 = np.c_[np.ones(len(a2[0])), a2.T]    #Size(5000 * 26)
    z3 = np.dot(theta2, a2.T)
    a3 = lgr.sigmoid(z3)
    a3 = np.argmax(a3.T, 1)                  #Size(5000 * 1)
    return a3

def main():
    data = sc.loadmat('data/ex3data1.mat', squeeze_me=True)
    X = data['X']
    y = data['y']

    #PARTE 1
    Theta = oneVsAll(X, y, 10, 0.20)
    yP = predictOneVsAll(Theta, X)

    #PARTE 2
    weights = sc.loadmat('data/ex3weights.mat')
    theta1, theta2 = weights['Theta1'], weights['Theta2']
    yP2 = predict(theta1, theta2, X)

    count = 0
    for i in range(len(y)):
        if(y[i] == yP[i]) :
            count+=1
```

```
    print("Part A accuracy: ", count/len(y)*100, "%")

    count = 0
    for i in range(len(y)):
        if(y[i] == yP2[i]) :
            count+=1

    print("Part B accuracy: ", count/len(y)*100, "%")

    rand_indices = np.random.choice(X.shape[0], 100, replace=False)
    utils.displayData(X[rand_indices, :])

main()
```