McMaster University, Physics 3G03 February 11 - March 4 2022
`http://avenue.mcmaster.ca/`

Homework 3 : Two-Class and Multi-Class Classification

You should submit source a python notebook along with a pdf of the notebook for all exercises.

Due: Friday March 4 2022, 11:30pm on avenue, with a grace period till the following Monday at 11:30pm

☛**Reading:** Parts of the first lectures are based on *The Hundred Page Machine Learning Book* by A. Burkov. Found at `http://themlbook.com`. Since then we have gone through Chapters 2,3,4,5,6,7 and parts of App A,B in *Machine Learning Refined*. An early version of *Machine Learning Refined* is available at `https://github.com/jermwatt/machine_learning_refined`

Exercise 1 *Classifying Breast Cancer*

In this exercise you will compare the efficacy of the Softmax and the Perceptron cost functions in terms of the minimal number of misclassifications each can achieve by proper minimization via gradient descent on a breast cancer dataset, a description of which you can find here: `https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)` . This dataset consists of $P = 699$ data points, each point consisting of $N = 9$ input of attributes of a single individual and output label indicating whether or not the individual does or does not have breast cancer. According to the website the rows correspond to:

1. Sample code number: id number
2. Clump Thickness: 1 - 10
3. Uniformity of Cell Size: 1 - 10
4. Uniformity of Cell Shape: 1 - 10
5. Marginal Adhesion: 1 - 10
6. Single Epithelial Cell Size: 1 - 10
7. Bare Nuclei: 1 - 10
8. Bland Chromatin: 1 - 10
9. Normal Nucleoli: 1 - 10
10. Mitoses: 1 - 10
11. Class: (2 for benign, 4 for malignant)

In the attached data file `breast_cancer_data.csv` the `Sample code number: id number` has been removed, the `Class` has been changed to 1 for benign, -1 for malignant. Finally, the data has been arranged *column-wise* so that each individual correspond to a single column, with the final row being the label of each point. (In the original data set the data is arranged *row-wise* with each row corresponding to an individual). Also, the data for the Single Epithelial Cell Size has been removed so that the data in the end has $N = 8$ attributes. We can then import the feature vectors and the labels with the following commands:

isn't this the way it should be??

```
# data input
csvname = datapath + 'breast_cancer_data.csv'
data1 = np.loadtxt(csvname,delimiter = ',')

# get input and output of dataset
x = data1[:-1,:]
y = data1[-1:,:]
```

1.1 We now need to implement the cost functions for the softmax and perceptron. Finish the implementation of the perceptron below:

```
# compute linear combination of input points
def model(x,w):
    a = w[0] + np.dot(x.T,w[1:])
    return a.T

# an implementation of the softmax cost
def softmax(w):
    # compute the least squares cost
    cost = np.sum(np.log(1 + np.exp(-y*model(x,w))))
    return cost/float(np.size(y))

# an implementation of the perceptron cost
def perceptron(w):
```

1.2 Use gradient descent (detailed on the previous assignments) with `max_its=1000`, and a random starting vector `w = 0.1*np.random.randn(9,1)` for both cost functions. For the perceptron cost function use `alpha = 0.1` to determine the weight *and* cost history. For the softmax use `alpha = 1.0` to determine the weight *and* cost history.

1.3 Use the weight history to track the number of misclassifications as a function of the iterations by implementing a function `miscount(w,x,y)` that counts the number of misclassifications for each `w` such that we can easily construct a misclassfication history using the command `miscount_history = [miscount(v,x,y) for v in weight_history]`

1.4 Plot the cost function history and the misclassifcation history versus the number of iterations for the 2 cost functions together. Determine the minimum number of misclassifications for the 2 cost functions. For each of the cost functions modify `miscount(w,x,y)` to calculate only the misclassified `malignant` cases.

1.5 We now want to check that we can get very similar results by directly doing a logistic regression on the data with a cross entropy cost. In order to do that we create a new *integer* vector `yc` which is equal to `y` except all the values of -1 in `y` have been replaced with a 0. Furthermore, note that `yc` has only a single index. We can do that (clumsily) using:

```
a=np.argwhere(y>0.9)
b=np.argwhere(y<-0.9)
yc=np.arange(699)
yc[a]=1
yc[b]=0
```

We need `yc` to be integer because the cross entropy function (p. 134 in the book) has statements like `np.argwhere(yc==0)`. We can the use this slightly modified version of the cross entropy function where we have added a L2 regularization:

```
# define sigmoid function
def sigmoid(t):
    return 1/(1 + np.exp(-t))

# the convex cross-entropy cost function
lam = 2*10**(-3)
def cross_entropy(w):
    # compute sigmoid of model
    a = sigmoid(model(x,w))

    # compute cost of label 0 points
    ind = np.argwhere(yc == 0)
    cost = -np.sum(np.log(1 - a[:,ind]))

    # add cost on label 1 points
    ind = np.argwhere(yc==1)
    cost -= np.sum(np.log(a[:,ind]))

    # add regularizer
    cost += lam*np.sum(w[1:]**2)

    # compute cross-entropy
    return cost/float(np.size(yc))
```

Using $\alpha = 0.6$ and again `max_its=1000` along with a similar random starting vector calculate the weight *and* cost history using logistic regression with gradient descent. Use the weight history to calculate the number of misclassifications. In this case you will need a modified `miscount` function. How will you define a misclassification in this case ?

Exercise 2 *Spam Detection*

In this study spam detection using the data set available at `https://archive.ics.uci.edu/ml/datasets/Spambase`. Again the input datapoints have been stacked *column-wise* in the dataset `spambase_data.csv` provided on avenue, with the final row being the label of each point which has been set to -1 (not spam) and 1 (spam). There are 57 attributes (features) for each of the 4601 samples corresponding to word frequencies and character frequencies among others.

There are two main problems with a data set like this that is very common for real life data sets. First it is quite possible that some of the entries are 'not a number' or `NaN`, secondly, how should we compare the frequency of *different* words or characters. A simple solution to this is

for each feature n calculate the mean and the standard deviation:

$$\mu_n = \frac{1}{P}\sum_{p=1}^{P} x_{p,n} \quad \sigma_n = \sqrt{\frac{1}{P}\sum_{p=1}^{P}(x_{p,n} - \mu_n)^2} \tag{1}$$

This calculation has to be done *skipping* occurrences of `NaN`. We can then replace each occurence of `NaN` with the relevant μ_n. That solves the first problem. The second problem can be handled by standard normalizing each feature, shifting by the mean and dividing with the standard deviation. To obtain:

$$x_{p,n} \to \frac{x_{p,n} - \mu_n}{\sigma_n}. \tag{2}$$

After this is done each row (feature) will have a mean of 0 and a standard deviation of 1 making it much more sensible to compare different features. We can use the following code to do that:

```
def standard_normalizer(x):
    # compute the mean and standard deviation of the input
    x_means = np.nanmean(x,axis = 1)[:,np.newaxis]
    x_stds = np.nanstd(x,axis = 1)[:,np.newaxis]

    # check to make sure that x_stds > small threshold, for those not
    # divide by 1 instead of original standard deviation
    ind = np.argwhere(x_stds < 10**(-2))
    if len(ind) > 0:
        ind = [v[0] for v in ind] # Just keep the row index
        adjust = np.zeros((x_stds.shape))
        adjust[ind] = 1.0
        x_stds += adjust
    # fill in any nan values with means
    ind = np.argwhere(np.isnan(x) == True)
    for i in ind:
        x[i[0],i[1]] = x_means[i[0]]

    # create standard normalizer function
    normalizer = lambda data: (data - x_means)/x_stds
    # create inverse standard normalizer
    inverse_normalizer = lambda data: data*x_stds + x_means
    # return normalizer
    return normalizer,inverse_normalizer
```

The standard normalization can then be done in a single command `x = normalizer(x)` as shown in the jupyter notebook for the homework.

2.1 Use gradient descent (detailed on the previous assignments) with `max_its=1000`, and a random starting vector `w = 0.1*np.random.randn(N+1,1)` to classify the e-mails using the softmax and perceptron cost functions. For the perceptron cost function use `alpha = 0.1` to determine the weight *and* cost history. For the softmax use `alpha = 1.` to determine the weight *and* cost history. Use the weight histories to construct the miscount histories for the two cost functions and make plots of the cost function histories and miscount histories versus the iterations. Comment on your observations.

2.2 For the two cost functions determine the smallest number of misclassifications achieved and convert it to an accuracy. Comment on your observations.

2.3 For the softmax results determine the best `w` and construct the confusion matrix.

Exercise 3 *Credit Check*

In this exercise we use a data sheet with a credit rating in `credit_dataset.csv`. This data set is described in Example 6.11 in the book by Watt et al. As usual the $P = 1000$ samples are arranged column wise with the last row being 1 (good rating) -1 (bad rating). The $N = 20$ input features describe things like current account balance, the duration in months of previous credit with the bank etc.

3.1 As in the previous exercise it is difficult to compare the importance of the different features and it therefore makes sense to *standard normalize*. Perform a standard normalization of the input data.

3.2 Use the perceptron cost function to fit a model using gradient descent with `max_its=1000`, and a random starting vector `w = 0.1*np.random.randn(N+1,1)` and `alpha = 0.1`. Plot the cost function history along with the misclassification history versus the number of iterations. Can you achieve the 75% accuracy mentioned in the book ?

3.3 Determine the confusion matrix for the optimal weights determined with the perceptron cost.

Exercise 4 *3-Class, classification*

In this exercise we will use the 3-class toy data set `3class_data.csv` shown in Fig. 7.9. We shall try to reproduce the data in that figure using the multi-class Perceptron. The implementation of the L2 regularized multiclass perceptron is discussed in the book:

```
lam = 10**-5  # our regularization paramter
def multiclass_perceptron(w):
    # pre-compute predictions on all points
    all_evals = model(x,w)

    # compute maximum across data points
    a = np.max(all_evals,axis = 0)

    # compute cost in compact form using numpy broadcasting
    b = all_evals[y.astype(int).flatten(),np.arange(np.size(y))]
    cost = np.sum(a - b)

    # add regularizer
    cost = cost + lam*np.linalg.norm(w[1:,:],'fro')**2
    # return average
    return cost/float(np.size(y))
```

Note that in this case we do not strictly enforce $||w||^2 = 1$. Instead, we have relaxed this constraint by instead including a regularizer.

4.1 Use the multi-class perceptron cost function to fit a model using gradient descent with `max_its=1000`, and a random starting vector `w = 0.1*np.random.randn(3,3)` and `alpha = 0.1`. Plot the cost function history versus the number of iterations. Can you achieve as good a classification as shown in Fig. 7.9 in the book ?

4.2 Plot the data in the plane along with the decision boundaries. You should get something similar but not idetical to what is in the book byt Watt et al.