Project 1: Forest & Mice Discrimination

Sami Rodrigue

1 Introduction

For this project, I will be going over the Forest Type Mapping and the Mice Protein Expression datasets. The Forest Type Mapping dataset contains information on visible-to-near infrared wavelengths for different forest types and the goal is to classify the forests using this spectral characteristics. The Mice Protein Expression dataset contains expression levels of 77 proteins/protein modifications and the goal is to use these proteins to discriminate between mice with Down Syndrome and the ones without it.

The project has two main components: Exploratory Analysis and Prediction. The first section identifies the variables that will be used for the next section.The second section uses logistic regression and multilayer perceptron to classify the datasets based on their target features.

2 Exploratory Analysis

2.1 Forest Type Mapping Dataset

The target variable for the Forest Type Mapping dataset has 4 types:

- s: Sugi Forest
- h: Hinoki Forest
- d: Mixed deciduous Forest
- o: Others Forest

The frequency of each class can be found in Figure 1. In the dataset, Sugi forests occurs most frequently and other forests have the fewest number of occurrences. Five features have been chosen by visually inspecting their ability to discriminate between classes (Figure 2). Following features will be used for the classification of tree types:

1. ASTER image band 1 (b1)
2. ASTER image band 4 (b4)
3. ASTER image band 6 (b5)
4. Predicted spectral values minus actual spectral values for class 's' for band 1 (pred_minus_obs_S_b1)
5. Predicted spectral values minus actual spectral values for class 's' for band 9 (pred_minus_obs_S_b9)

The summary statistics of these features can be found in Table 1. The class variable's summary values are omitted from the table since it is a categorical variable.

	b1	b4	b6	pred_minus_obs_H_b1	pred_minus_obs_H_b9
mean	62.95	101.41	100.65	50.82	-5.59
std	12.78	14.80	11.19	12.84	9.77
25%	54.00	92.25	92.00	40.67	-6.63
50%	60.00	99.50	98.00	53.03	-2.26
75%	70.75	111.75	107.00	59.92	0.25

Table 1: Summary statistics of the chosen features.

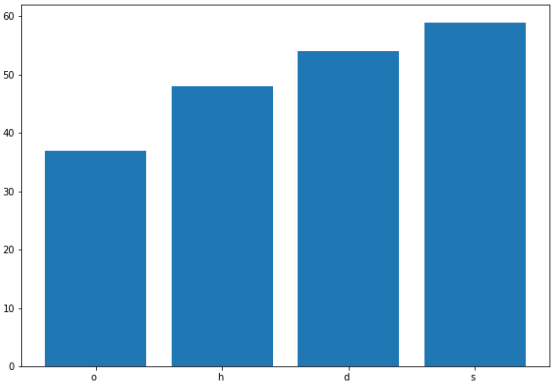

Figure 1: Frequency of each class. s corresponds to Sugi forest, h corresponds to Hinoki forest, d corresponds to Mixed deciduous forest and o is others.

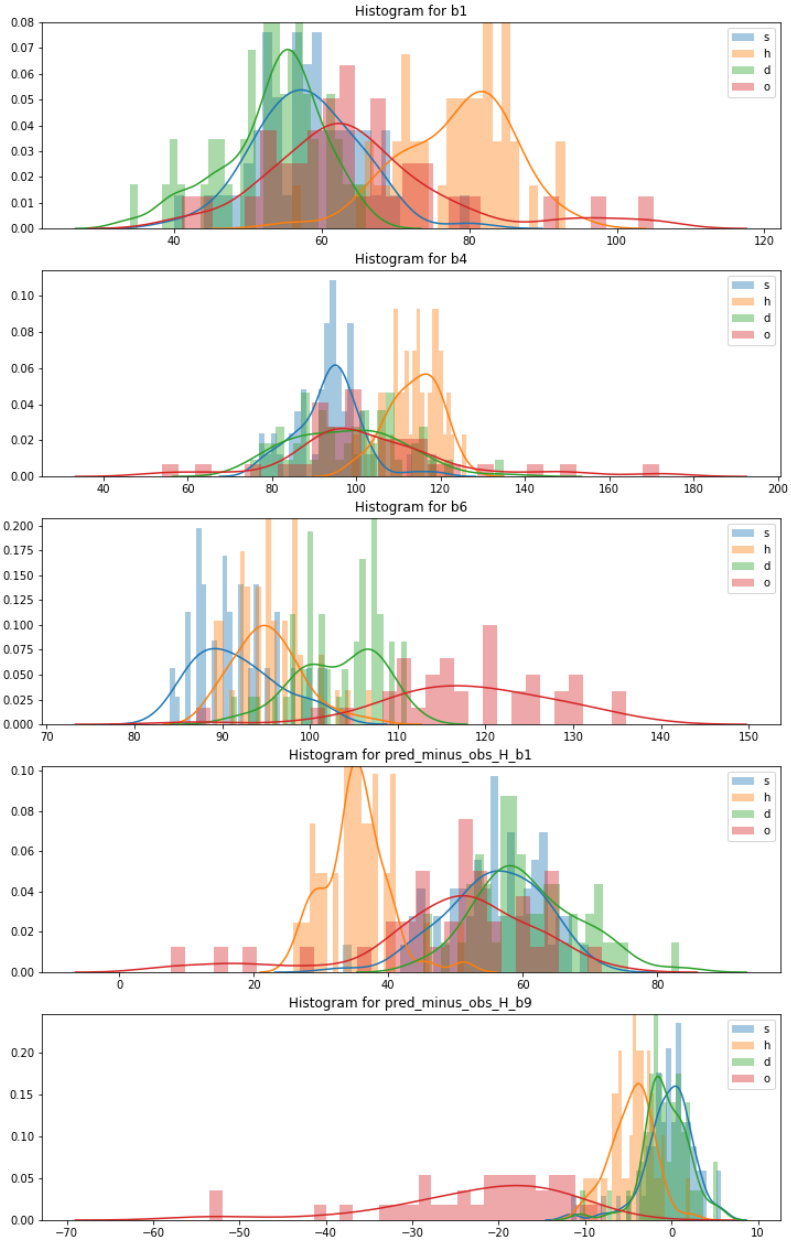

Figure 2: Observed probability distribution of each feature based on the respective class

Figure 3 contains the histograms and the scatter matrix of the features. b1, b4 and pred_minus_obs_H_b1 follow a normal distribution and the rest are either left or right skewed. b1 is highly correlated with pred_minus_obs_H_b1 because the later feature is derived using b1. This can indicate that these features are explaining the same aspects of the target variable. Figure 2 provides further evidence for this claim since these variables appear to be the mirror image of one another. Pairs b1-b4 and b6-pred_minus_obs_H_b9 also have a strong correlation but this can be spurious correlation.

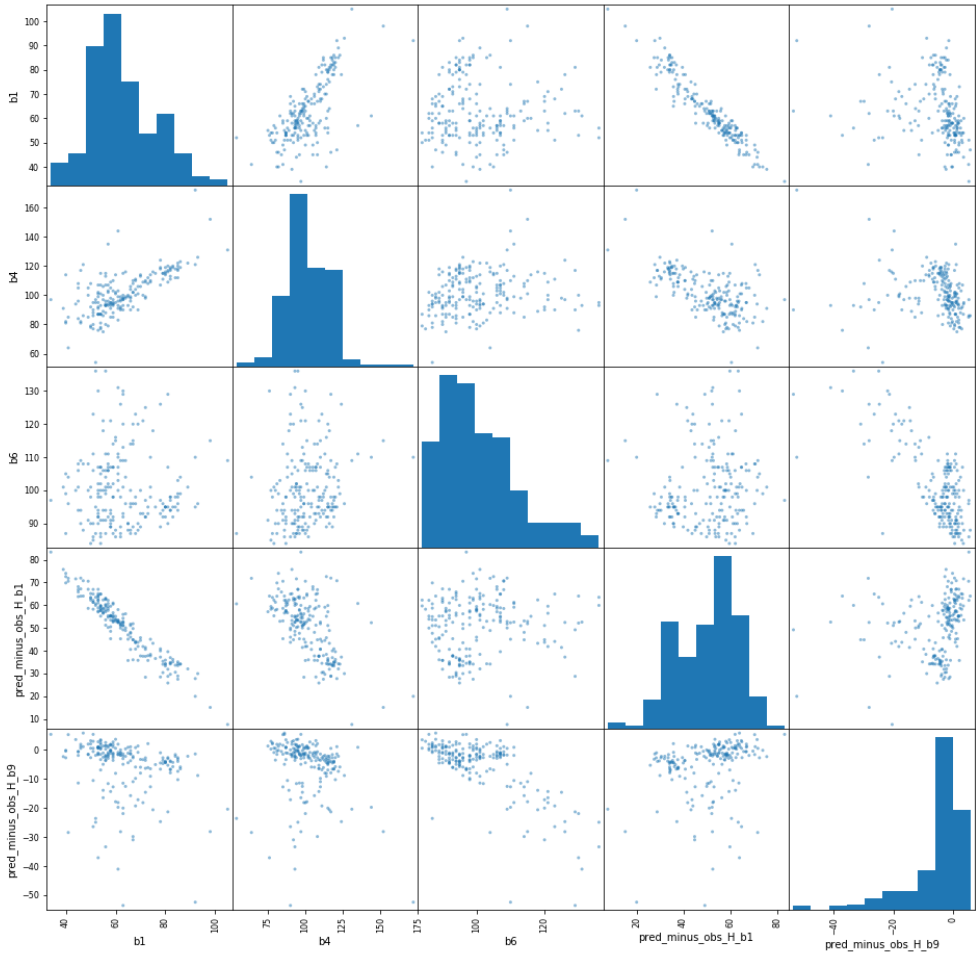

Figure 3: Scatter matrix of the features

	b1	b4	b6	pred_minus_obs_H_b1	pred_minus_obs_H_b9
b1	1.00	0.68	0.01	-0.96	-0.25
b4	0.68	1.00	0.17	-0.62	-0.21
b6	0.01	0.17	1.00	0.06	-0.72
pred_minus_obs_H_b1	-0.96	-0.62	0.06	1.00	0.24
pred_minus_obs_H_b9	-0.25	-0.21	-0.72	0.24	1.00

Table 2: Correlation of the features.

2.2 Mice Protein Expression Dataset

The target variable for the Mice Protein Expression dataset has 8 classes and they are:

- c-CS-s: control mice, stimulated to learn, injected with saline
- c-CS-m: control mice, stimulated to learn, injected with memantine
- c-SC-s: control mice, not stimulated to learn, injected with saline

- c-SC-m: control mice, not stimulated to learn, injected with memantine
- t-CS-s: trisomy mice, stimulated to learn, injected with saline
- t-CS-m: trisomy mice, stimulated to learn, injected with memantine
- t-SC-s: trisomy mice, not stimulated to learn, injected with saline
- t-SC-m: trisomy mice, not stimulated to learn, injected with memantine

In the dataset, there are approximately 15 different measurements for each mice. Hence, even though there are more than 100 observations for each class, there are only 72 different mice once we control for multiple measurements (Figure 4).

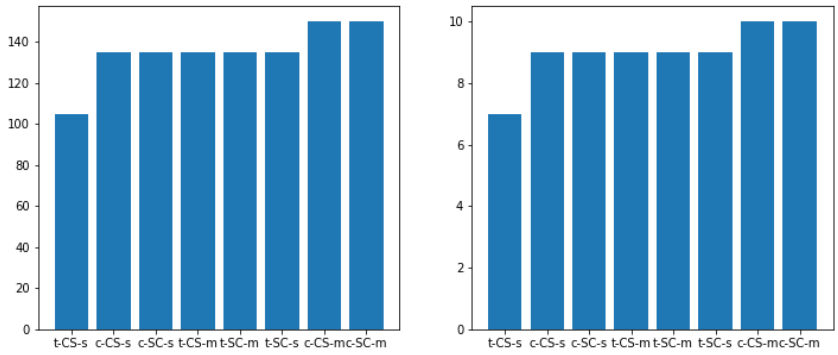

Figure 4: Frequency of observations.

Unlike the previous dataset, the features are chosen randomly due to the sheer volume of variables and lack of domain knowledge to supplement the decision making process. The chosen proteins are:

1. ADARB1_N
2. EGR1_N
3. ELK_N
4. MEK_N
5. CAMKII_N

The summary statistics of these features can be found in Table 3. Figure 5 contains the histograms and the correlation matrix of the chosen features and their correlations can be found at Table 4. ADARB1, EGR1 and ELK exhibit a right skewed distribution which is confirmed by Table 3 since their means lie on the right of the median. All the chosen features exhibit a normal distribution. MEK has relatively strongly correlation with ELK and CAMKII but they simply can be correlated randomly since 70+ features will have features to be correlated by pure chance.

	ADARB1_N	EGR1_N	ELK_N	MEK_N	CAMKII_N
mean	1.20	0.18	1.17	0.27	0.36
std	0.36	0.04	0.34	0.04	0.05
25%	0.93	0.16	0.94	0.25	0.33
50%	1.13	0.17	1.10	0.27	0.36
75%	1.38	0.20	1.32	0.30	0.39

Table 3: Summary Statistics.

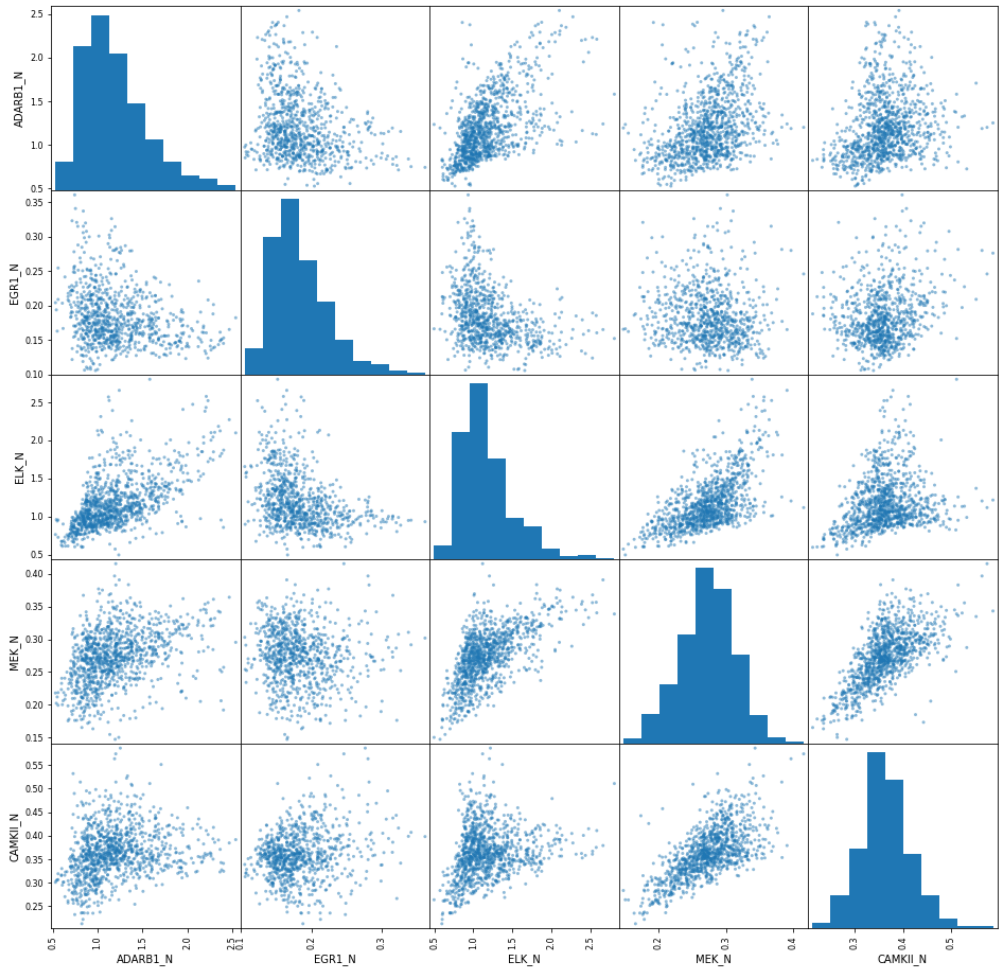

Figure 5: Scatter matrix of the features

	ADARB1_N	EGR1_N	ELK_N	MEK_N	CAMKII_N
ADARB1_N	1.00	-0.26	0.57	0.38	0.22
EGR1_N	-0.26	1.00	-0.35	-0.07	0.21
ELK_N	0.57	-0.35	1.00	0.62	0.23
MEK_N	0.38	-0.07	0.62	1.00	0.67
CAMKII_N	0.22	0.21	0.23	0.67	1.00

Table 4: Correlation Matrix.

3 Solution Method

For the project, I have implemented logistic regression and Multilayer Perceptron (MLP) with a single hidden layer to classify the target variable. The algorithms are written using the pseudocode presented in Ethem Alpaydin's Introduction to Machine Learning book.

The logistic regression is a linear discriminant which requires a gradient descent solution due to its nonlinear output space. Since it is a linear discriminant, it requires the classes to be linearly separable. MLP also implements linear discriminants in its hidden layer, however since it combines these discriminants to reach its output layer it can be used with classes that are not linearly separable.

When implementing the models, I discovered steps that needs to be implemented to improve the results. For both logistic regression and MLP, I normalized the input values, otherwise the models did not converge to any meaningful results. I have also added a decay factor to the learning rate to both algorithms. However, only the logistic regression used for the Mice dataset benefited

from this change. Finally, to prevent overflow of the softmax functions, I have incorporated the exp-normalization trick.

4 Results

The results for the Forest dataset can be found in Tables 5 and 7. Overall, MLP performs better than logistic regression for this dataset. MLP that performs the best is with 30 hidden states and 600 epochs. It seems having more than 30 states causes the model to overfit the training set. Generally, longer a MLP is trained, better the performance will be. However, this adage does not always hold. For example, when it has 50 hidden states, the model trained for 200 epochs performs better than the model trained for 400 epochs. Initializations for MLP's weights play a key role on its convergence. Hence, even though it was trained longer, it failed to converge to a better result. For logistic regression, the iteration time is not as important as MLP and it tends to return the same results after training for 10 epochs.

Epoch	Accuracy
10	0.713846
25	0.710769
50	0.713846
100	0.710769
200	0.713846

Table 5: Logistic Regression results for the forest dataset.

Epoch	Accuracy
10	0.272031
25	0.272031
50	0.272031
100	0.272031
200	0.272031

Table 6: Logistic Regression results for the Mice dataset (with a decay factor = 0.99).

Hidden State	Epoch	Accuracy
30	200	0.753846
30	300	0.756923
30	400	0.747692
30	500	0.772308
30	600	0.781538
40	200	0.720000
40	300	0.753846
40	400	0.772308
40	500	0.710769
40	600	0.741538
50	200	0.741538
50	300	0.756923
50	400	0.701538
50	500	0.778462
50	600	0.772308
60	200	0.747692
60	300	0.763077
60	400	0.738462
60	500	0.747692
60	600	0.775385

Table 7: MLP results for the forest dataset.

Hidden State	Epoch	Accuracy
30	200	0.440613
30	300	0.455939
30	400	0.498084
30	500	0.432950
30	600	0.463602
40	200	0.421456
40	300	0.421456
40	400	0.494253
40	500	0.463602
40	600	0.501916
50	200	0.429119
50	300	0.417625
50	400	0.528736
50	500	0.490421
50	600	0.471264
60	200	0.448276
60	300	0.436782
60	400	0.551724
60	500	0.482759
60	600	0.505747

Table 8: MLP results for the Mice dataset.

The results for the Mice dataset can be found in Tables 6 and 8. Again, MLP does a better job in classifying the test set. The model with the highest accuracy has 60 hidden states and has an epoch of 400. As in the forest dataset, initialization plays a key role in the result of the MLP. The poor performance of logistic regression and the high number of hidden states for MLP hints that the dataset is not linearly separable using the chosen features. Other variables can produce better results.

5 Appendix

This section includes code for the visualizations and the analysis

5.1 Code for Visualization

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.gridspec as gridspec
from pandas.plotting import scatter_matrix
import random

#### FOREST Dataset
path = 'D:/OneDrive/Online/SelfLearn/Ryerson/MSc/2018W/DS8004/Projects/P1'
train = pd.read_csv(path+'/training.csv')

summary = train.groupby('class').count().b1.sort_values()

plt.figure(figsize=(10,7))
plt.bar(x = range(4), height= summary, label=summary.index )
plt.xticks(range(4), summary.index)
plt.show();

binCount = 30
features = train.loc[:,['b1', 'b4', 'b6', 'pred_minus_obs_H_b1', '
    pred_minus_obs_H_b9']].columns

plt.figure(figsize=(12,5*4))
gs = gridspec.GridSpec(5, 1)
for i, cn in enumerate(train[features]):
    ax = plt.subplot(gs[i])
    sns.distplot(train[cn][train['class'] == 's '], bins=binCount, label='s')
    sns.distplot(train[cn][train['class'] == 'h '], bins=binCount, label='h')
    sns.distplot(train[cn][train['class'] == 'd '], bins=binCount, label='d')
    sns.distplot(train[cn][train['class'] == 'o '], bins=binCount, label='o')
    ax.set_xlabel('')
    ax.legend()
    ax.set_title('Histogram for ' + str(cn))
plt.show()

scatter_matrix( train.loc[:,['b1', 'b4', 'b6', 'pred_minus_obs_H_b1', '
    pred_minus_obs_H_b9']], figsize=(16, 16))
plt.show()

#### MICE Dataset
path = 'D:/OneDrive/Online/SelfLearn/Ryerson/MSc/2018W/DS8004/Projects/P1'
train = pd.read_csv(path+'/Data_Cortex_Nuclear.csv')
train['MouseID2'] = [x.partition("_")[0] for x in train.MouseID]

random.seed(24)
features = random.sample(range(1,78),5)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,5))

summary = train.groupby('class').count().ADARB1_N.sort_values()
ax1.bar(x = range(8), height= summary)
ax1.set_xticks(range(8))
ax1.set_xticklabels(labels=summary.index)

summary = train.loc[:,['class','MouseID2']].drop_duplicates().groupby('class').
    MouseID2.count().sort_values()
ax2.bar(x = range(8), height= summary)
ax2.set_xticks(range(8))
ax2.set_xticklabels(labels=summary.index)
plt.show();

scatter_matrix( train.iloc[:,features], figsize=(16, 16))
plt.show()
```

5.2 Code for Prediction

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import random
from sklearn.metrics import accuracy_score, confusion_matrix

def sigmoid(a):
    size = a.shape[0]
    output = np.zeros(size)
    for i in range(size):
        if a[i] > -100:
            output[i] = 1. / (1 + np.exp(-a[i]))
    return(output)

def softmax(x):
    # exp-normalize trick: https://timvieira.github.io/blog/post/2014/02/11/exp-
        normalize-trick/
    e = np.exp(x - np.max(x))
    if e.ndim == 1:
        return e / np.sum(e, axis=0)
    else:
        return e / np.array([np.sum(e, axis=1)]).T

def normalize(X):
    max_ = np.max(X,axis=0)
    min_ = np.min(X,axis=0)
    return((X - min_)/(max_ - min_))

def logDiscrimination(X, Y, step = 0.1, totIter= 100, decay = 0.95):
    X = normalize(X)
    X = np.c_[np.ones(X.shape[0]).reshape(-1,1), X]

    totElem = X.shape[0]

    # one hot coding
    Y = Y - np.min(Y)
    n_values = np.max(Y) + 1
    Y = np.eye(n_values)[Y]

    k = Y.shape[1]
    d = X.shape[1]
    w = np.random.uniform(low = -0.01, high = 0.01, size = (k,d))

    for i in range(totIter):
        idx = random.sample(range(totElem), totElem)
        X = X[idx,:]
        Y = Y[idx,:]

        entropy = 0
        deltaW = np.zeros((k,d))

        for t in range(totElem):
            x = X[t,:]
            r = Y[t,:]

            o = np.dot(x, w.transpose())
            y = softmax(o)
            w += step * (r - y).reshape(k,1) * x

        step *= decay

    return(w)

def prediction(w, x):
    x_ = np.append(1, x).astype(float)
    o_=np.dot(x_, w.transpose())
    return(np.argmax(softmax(o_)))

def backprop(X, Y, step=0.1, H=3, epoch=100, decay = 0.99):
    J = X.shape[1]
```

```
    totElem = X.shape[0]

    X = normalize(X)
    X = np.c_[np.ones(X.shape[0]).reshape(-1,1), X]

    # one hot coding
    Y = Y - np.min(Y)
    n_values = np.max(Y) + 1
    Y = np.eye(n_values)[Y]

    I = Y.shape[1]

    #xavier = (6/(J+I))**0.5
    xavier = 0.01
    v = np.random.uniform(low = -xavier, high = xavier, size = (I, H+1))
    w = np.random.uniform(low = -xavier, high = xavier, size = (H, J+1))

    for e in range(epoch):
        idx = random.sample(range(totElem), totElem)
        X = X[idx,:]
        Y = Y[idx,:]

        for t in range(totElem):
            x = X[t,:]
            r = Y[t,:]

            z = np.ones(H+1)
            z[1:] = sigmoid(np.dot(x, w.transpose()))

            y = softmax(np.dot(z, v.transpose()))

            error = r - y

            deltaV = np.zeros((I, H+1))
            for h in range(H):
                deltaV[:,h] = error * z[h]

            deltaW = np.zeros((H, J+1))
            for h in range(H):
                deltaW[h,:] = np.sum(error * v[:,h+1]) * z[h+1] * (1 - z[h+1]) * x

            v += step * deltaV
            w += step * deltaW

        step *= decay
    return(v, w)

def feedforward(v,w, x, H):
    x_ = np.append(1, x).astype(float)
    z = np.ones(H+1)
    z[1:] = sigmoid(np.dot(x_, w.transpose()))
    y = softmax(np.dot(z, v.transpose()))
    return(y)

#### FOREST Dataset
path = 'D:/OneDrive/Online/SelfLearn/Ryerson/MSc/2018W/DS8004/Projects/P1'
train = pd.read_csv(path+'/training.csv')
test = pd.read_csv(path+'/testing.csv')
train = train.loc[:,['class','b1', 'b4', 'b6', 'pred_minus_obs_H_b1', '
    pred_minus_obs_H_b9']]
test = test.loc[:,['class','b1', 'b4', 'b6', 'pred_minus_obs_H_b1', '
    pred_minus_obs_H_b9']]
train['class'] = train['class'].astype('category').cat.codes
test['class'] = test['class'].astype('category').cat.codes

iterCount = [10, 25, 50, 100, 200]

table = pd.DataFrame(columns=['Epoch', 'Accuracy'])

for j in iterCount:
    weights = logDiscrimination(step = 0.01, X=train.iloc[:,range(1,6)].as_matrix()
        , \
                                Y=train['class'].astype(int), totIter=i)
```

```
    results = test.copy()
    results['pred'] = np.repeat(-1, test.shape[0])

    max_ = np.max(train.iloc[:,range(1,6)].as_matrix(),axis=0)
    min_ = np.min(train.iloc[:,range(1,6)].as_matrix(),axis=0)
    testNormalized = (test.iloc[:,range(1,6)].as_matrix() - min_)/(max_ - min_)

    for i in range(results.shape[0]):
        results.iloc[i,6] = prediction(w=weights, x=testNormalized[i,:])

    table = table.append({'Epoch': int(j),
                          'Accuracy':accuracy_score(y_true=results['class'].values,
                                                    y_pred=results['pred'].values)},
                                                        ignore_index=True)

table['Epoch'] = table['Epoch'].astype('int64')
print(table.to_latex())

hiddenCount = [30, 40, 50, 60, 70]
iterCount = [200, 300, 400, 500, 600]
table = pd.DataFrame(columns=['Hidden_State', 'Epoch', 'Accuracy'])

for s in hiddenCount:
    for j in iterCount:
        v,w = backprop(X=train.iloc[:,range(1,6)].as_matrix(),
               Y= train['class'].astype(int), step=0.1, H=s, epoch=j, decay=1)
        results = test.copy()
        results['pred'] = np.repeat(-1, test.shape[0])

        max_ = np.max(train.iloc[:,range(1,6)].as_matrix(),axis=0)
        min_ = np.min(train.iloc[:,range(1,6)].as_matrix(),axis=0)
        testNormalized = (test.iloc[:,range(1,6)].as_matrix() - min_)/(max_ - min_)

        for i in range(results.shape[0]):
            results.iloc[i,6] = np.argmax(feedforward(v=v,w=w,x=testNormalized[i
                ,:], H = s))

        table = table.append({'Hidden_State': int(s),
                              'Epoch': int(j),
                               'Accuracy':accuracy_score(y_true=results['class'].
                                   values,
                                                         y_pred=results['pred'].values
                                                             )}, ignore_index=True)

table['Hidden_State'] = table['Hidden_State'].astype('int64')
table['Epoch'] = table['Epoch'].astype('int64')
print(table.to_latex())

#### MICE Dataset
path = 'D:/OneDrive/Online/SelfLearn/Ryerson/MSc/2018W/DS8004/Projects/P1'
data = pd.read_csv(path+'/Data_Cortex_Nuclear.csv')
data['MouseID2'] = [x.partition("_")[0] for x in data.MouseID]
data = data.loc[:,['class', 'MouseID', 'MouseID2', 'ADARB1_N', 'EGR1_N', 'ELK_N', '
    MEK_N', 'CAMKII_N']]
data['class2'] = data['class'].astype('category').cat.codes
data = data.dropna()

random.seed(42)
idx = random.sample(range(data.shape[0]), data.shape[0])
idxTrain = idx[0:int(data.shape[0]*0.7)]
idxTest = idx[int(data.shape[0]*0.7):data.shape[0]]

train = data.iloc[idxTrain].reset_index(drop=True)
test = data.iloc[idxTest].reset_index(drop=True)

iterCount = [10, 25, 50, 100, 200]
table = pd.DataFrame(columns=['Epoch', 'Accuracy'])

for j in iterCount:
    weights = logDiscrimination(step = 0.01, X=train.iloc[:,range(3,8)].as_matrix()
        , Y=train['class2'].astype(int), decay = 0.99, totIter=i)
    results = test.copy()
    results['pred'] = np.repeat(-1, test.shape[0])
```

```
    max_ = np.max(train.iloc[:,range(3,8)].as_matrix(),axis=0)
    min_ = np.min(train.iloc[:,range(3,8)].as_matrix(),axis=0)
    testNormalized = (test.iloc[:,range(3,8)].as_matrix() - min_)/(max_ - min_)

    for i in range(results.shape[0]):
        results.iloc[i,9] = prediction(w=weights, x=testNormalized[i,:])

    table = table.append({'Epoch': int(j),
                          'Accuracy':accuracy_score(y_true=results['class2'].values
                              ,
                                                    y_pred=results['pred'].values)},
                              ignore_index=True)

table['Epoch'] = table['Epoch'].astype('int64')
print(table.to_latex())

hiddenCount = [20, 30, 40, 50, 60]
iterCount = [200, 300, 400, 500, 600]
table = pd.DataFrame(columns=['Hidden_State', 'Epoch', 'Accuracy'])

for s in hiddenCount:
    for j in iterCount:
        v,w = backprop(X=train.iloc[:,range(3,8)].as_matrix(),
               Y= train['class2'].astype(int), step=0.1, H=s, epoch=j, decay=1)
        results = test.copy()
        results['pred'] = np.repeat(-1, test.shape[0])

        max_ = np.max(train.iloc[:,range(3,8)].as_matrix(),axis=0)
        min_ = np.min(train.iloc[:,range(3,8)].as_matrix(),axis=0)
        testNormalized = (test.iloc[:,range(3,8)].as_matrix() - min_)/(max_ - min_)

        for i in range(results.shape[0]):
            results.iloc[i,9] = np.argmax(feedforward(v=v,w=w,x=testNormalized[i
                ,:], H = s))

        table = table.append({'Hidden_State': int(s),
                              'Epoch': int(j),
                               'Accuracy':accuracy_score(y_true=results['class2'].
                                   values,
                                                         y_pred=results['pred'].values
                                                             )}, ignore_index=True)

table['Hidden_State'] = table['Hidden_State'].astype('int64')
table['Epoch'] = table['Epoch'].astype('int64')
print(table.to_latex())
```