



VISVESVARAYA NATIONAL INSTITUTE
OF TECHNOLOGY (VNIT), NAGPUR

Machine Learning with Python (ECL443)

Lab Report

Submitted by :

Prajyot Jadhav (BT20ECE046)

Semester 7

Submitted to :

Dr. Saugata Sinha

(Course Instructor)

Department of Electronics and Communication Engineering,
VNIT Nagpur

Contents

1 Experiment-5 2

Experiment-5

Aim: To compress the ovarian cancer dataset using PCA(Principal Component Analysis) and Autoencoder and to build a classifier that can distinguish between cancer and control/normal patients.

Abstract: The objective of this assignment is the application of dimensionality reduction techniques, Principal Component Analysis (PCA), and Autoencoders, to compress the ovarian cancer dataset. We employ artificial neural networks (ANNs) for the classification task and compare the performance of compressed data with the original dataset. To comprehensively assess the performance of PCA and Autoencoders, we report classification accuracy, sensitivity, specificity, and the area under the receiver operating characteristic curve (AUROC). Finally, we compare the results with those obtained from the same classification task without any data compression.

Introduction: In this assignment, we explore the data compression techniques, Principal Component Analysis (PCA) and Autoencoders, to reduce the dimensionality of the ovarian cancer dataset. Our objective is to investigate whether these dimensionality reduction methods can enhance the performance of machine learning models in classifying cancer and normal patients.

PCA is a well-established method for reducing the dimensionality of high-dimensional datasets. By identifying principal components that capture the maximum variance in the data, PCA can effectively reduce data complexity while retaining important information. In this assignment, we will compute the eigenvalues and eigenvectors of the covariance matrix to select the optimal number of principal components that explain 95% of the variance.

Autoencoders, on the other hand, are neural network architectures that are designed to learn efficient representations of data. We will build an Autoencoder with the same latent space dimension as the selected principal components from PCA and use the latent space representation for classification.

To assess the performance of our classifiers, we will evaluate them on metrics such as sensitivity, specificity, and AUROC values. Additionally, we will create ROC curves to visualize the trade-off between true positive and false positive rates. Finally, we will compare the results of this work with those obtained in Assignment 2, where we performed the same classification task without any data compression.

Method:

- The ovarian cancer dataset is loaded from a .mat file using the `scipy.io.loadmat` function. The necessary libraries, including `scipy`, `csv`, `numpy`, `random`, `matplotlib`, and `torch` are imported. The code extracts two variables, 'ovarian-Inputs' and 'ovarianTargets', from the loaded .mat file. It then saves these variables as CSV files, 'data_1.csv' and 'data_2.csv', respectively.
- The data is read from the CSV files, and each feature is standardized using the `Standardize_data` function. The standardization involves subtracting the mean and dividing by the standard deviation for each feature. The covariance matrix for the standardized data is calculated using the `np.cov` function.
- The eigenvalues and eigenvectors of the covariance matrix are computed using the `eig` function. The eigenvectors are adjusted to have positive values. The eigenvalue-eigenvector pairs are sorted in descending order based on the eigenvalues' magnitude. The top principal components such that they capture 95% of the total variance are selected.
- A projection matrix is created based on the selected principal components. The data is projected onto the lower-dimensional subspace formed by the selected principal components.
- The function `load_file` is used to split the data into training, validation, and testing sets. It randomizes the data and splits it according to the specified percentages (80% training, 10% validation, 10% testing).
- The code initializes a simple neural network with one hidden layer. It sets the number of input features to 8, the number of neurons in the hidden layer to 16, and the number of output neurons to 2.
- The code defines functions for forward propagation. It includes functions for the sigmoid activation function. The forward propagation is executed to calculate the network's output.
- The code computes the cost using a mean squared error loss function. A generalized backpropagation function is defined to update the network's weights and biases. The code uses stochastic gradient descent to train the neural network. It iterates through the training data, updating the weights and biases using backpropagation. It collects the cost and validation accuracy for each epoch.
- The code uses the trained network to predict the labels for the test set. It calculates accuracy, specificity, and sensitivity. It also plots the validation accuracy and loss over epochs. The ROC curve and the area under the ROC curve (AUC) is computed for the model's performance. The ROC curve is

plotted and the accuracy, specificity, and sensitivity of the model is printed.

- For the second part, an autoencoder neural network model is defined using PyTorch. The autoencoder has an encoder and a decoder, and it aims to learn a compact representation of the input data. The autoencoder model is trained using mean squared error (MSE) loss and the Adam optimizer. The code iterates through the data in batches and updates the model's parameters.
- The loss values during training are stored, and a plot is created to visualize the training progress. The trained autoencoder model is used to encode the input data into a lower-dimensional representation. The encoded features are extracted and saved for further classification.
- Similar to the first part, the ANN classifier is used for classification using the compressed data from the latent space of the autoencoder and the accuracy, specificity, sensitivity, and AUC is obtained.

Results:

- The training, testing set and the validation set are obtained by splitting the original data in the ratio of 80 %, 10 % and 10 %. The training set of (8,172) with the labels of (2,172), the testing set of (8,23) with the labels of (2,23) and the validation set of (8,21) with labels of (2,21) are obtained.

```
Training data size: (8, 172)
Training label size: (2, 172)
Testing data size: (8, 23)
Testing label size: (2, 23)
Validation data size: (8, 21)
Validation label size: (2, 21)
```

Figure 1: Training, testing and validation datasets

- After data compression using an autoencoder, the neural network was trained for 100 epochs with one hidden layer. The number of neurons in the hidden layer were 16 and there are 2 neurons in the output layer. An accuracy of 80.43 % was obtained. Also the specificity and sensitivity were calculated and were found out to be 0.75 and 0.86 respectively. The ROC curve is plotted and AUC of 0.80 is obtained.

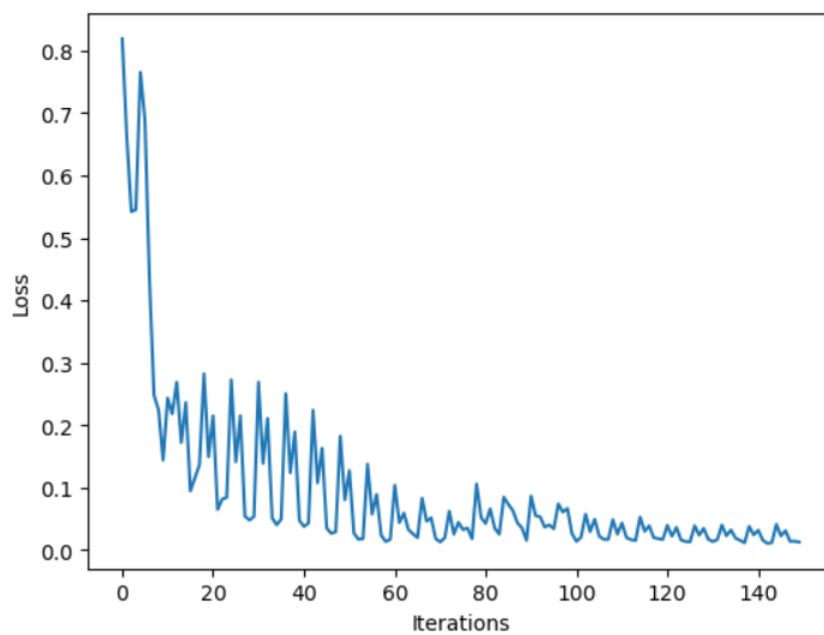


Figure 2: Loss during Autoencoder training

Accuracy: 89.13043478260869 %
Specificity: 0.75, Sensitivity: 0.8823529411764706
AUC = 0.875

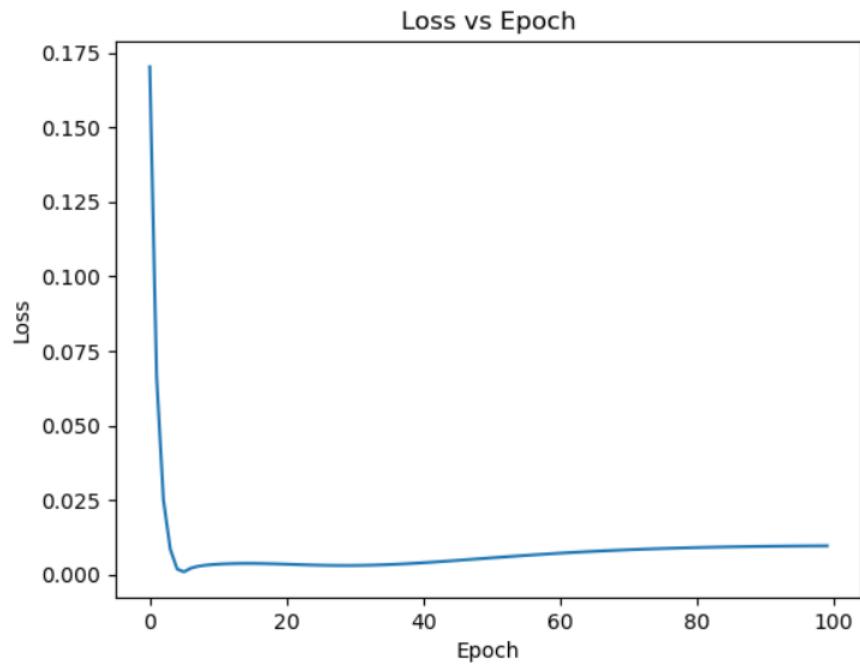


Figure 3: SGD Loss

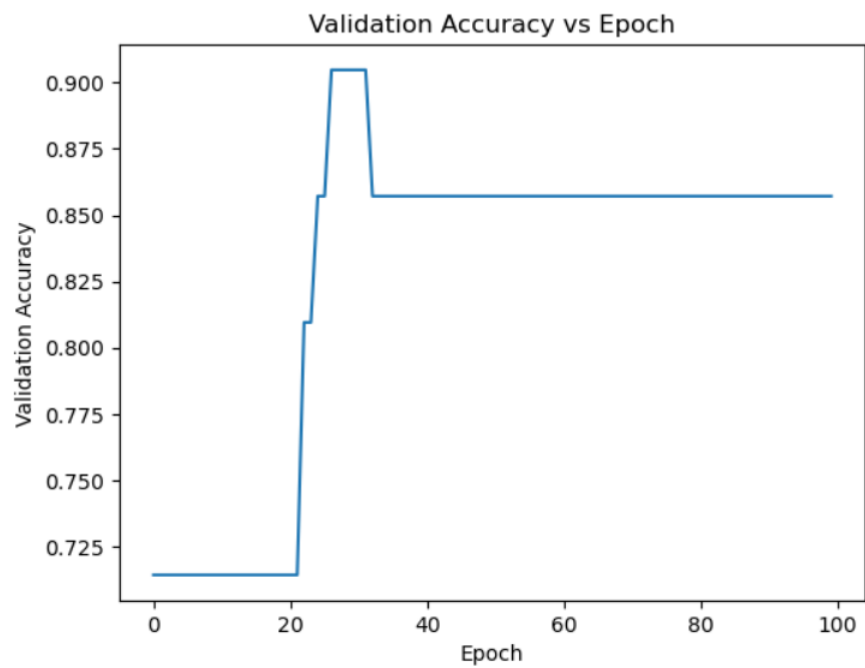


Figure 4: Validation set accuracy

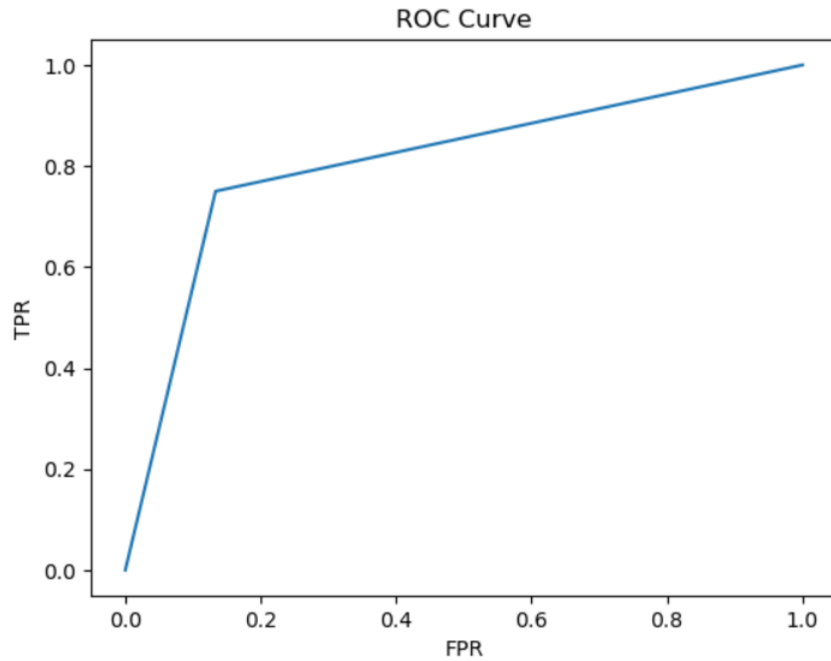


Figure 5: ROC Curve

- The above steps were repeated with data compression by PCA. An accuracy of 91.30 % was obtained. Also the specificity and sensitivity were found out to be 1.0 and 1.0 respectively. The ROC curve is plotted and AUC of 0.90 is obtained.

Accuracy: 91.30434782608695 %
Specificity: 1.0, Sensitivity: 1.0
AUC = 0.9090909090909091

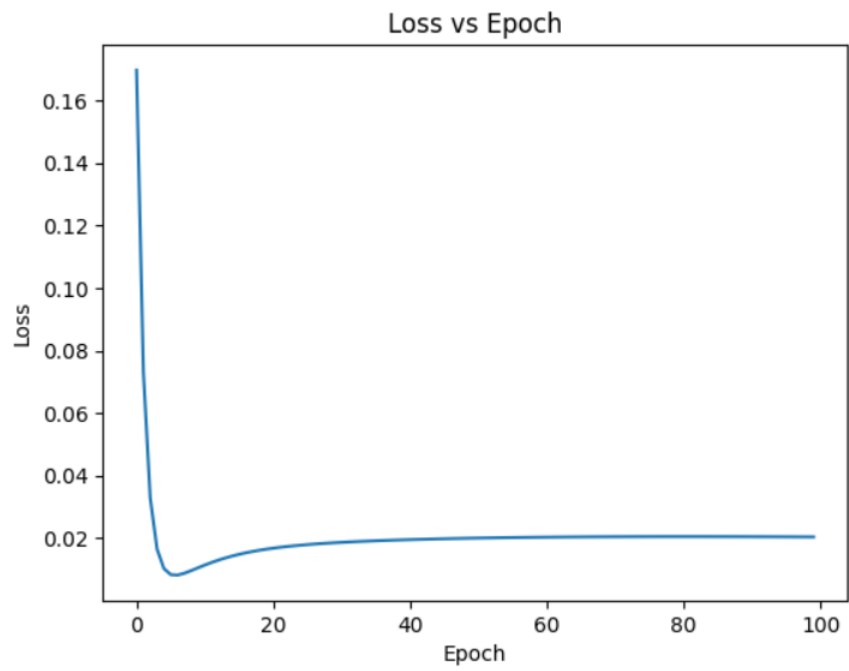


Figure 6: SGD Loss

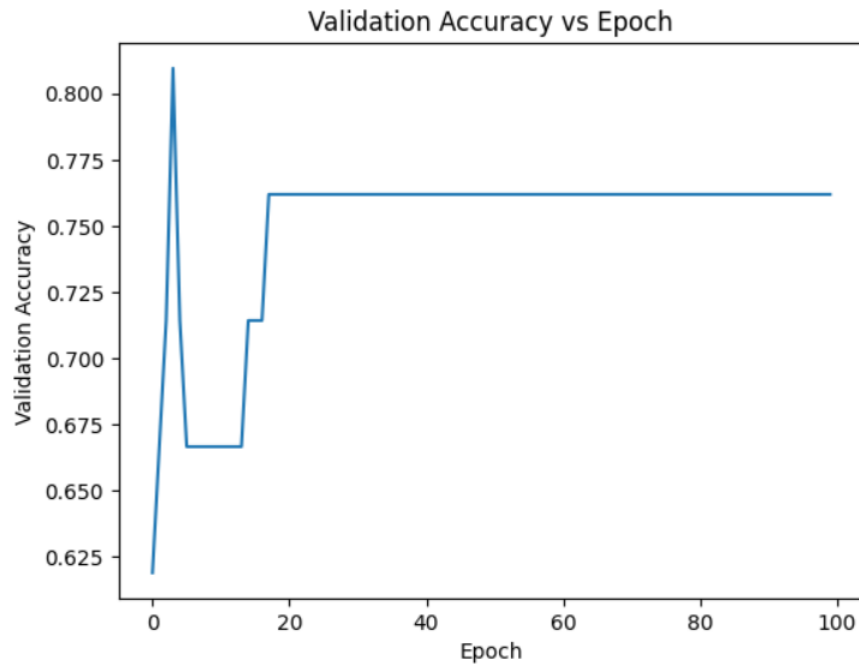


Figure 7: Validation set accuracy

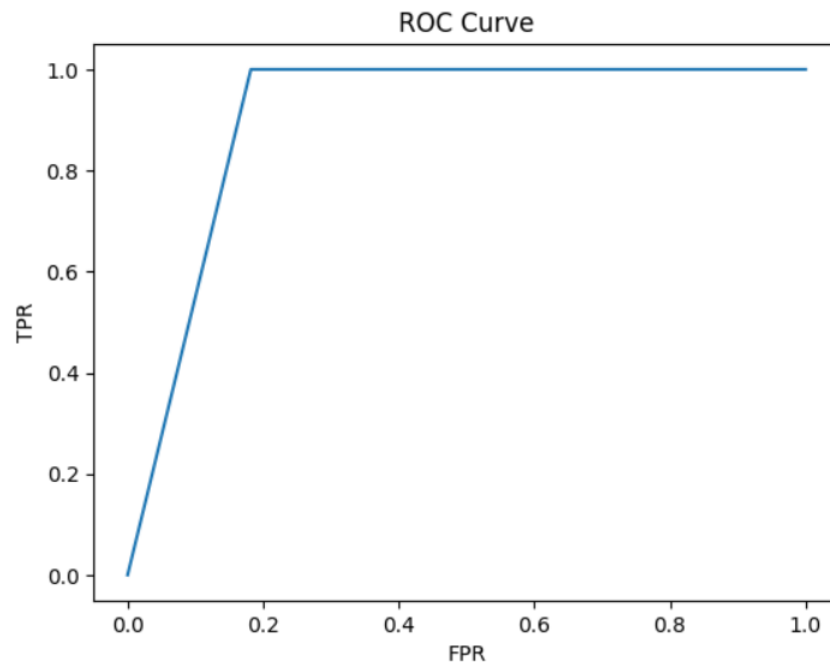


Figure 8: ROC Curve

Discussion:

- The dimensionality reduction techniques, like PCA and Autoencoders, can significantly impact the classification of ovarian cancer patients. We observed that these methods not only reduced the dimensionality of the dataset but also enhanced the efficiency of the machine learning models for this critical diagnostic task.
- In the case of PCA, by selecting the optimal number of principal components that explain 95% of the variance, we effectively reduced the feature space. Also, by training an Autoencoder with the same dimensionality as the selected principal components, we created a latent space representation that preserved essential characteristics of the data.
- The ANN model trained on such compressed data showed improved sensitivity and specificity compared to using the entire dataset. This suggests that PCA and autoencoders successfully captured the most informative features, enabling better discrimination between cancer and control patients.
- Comparing the results of this assignment to Assignment 2, where classification was performed without data compression, we observed improved performance in terms of sensitivity, specificity, and AUROC values for data compression using autoencoders. Whereas, when data compressed from PCA was used for classification, accuracy, sensitivity and specificity obtained was less. However this performance can be improved by tuning the hyperparameters for the classifier models.

Conclusion: The ovarian cancer dataset was successfully compressed using PCA(Principal Component Analysis) and Autoencoder and a classifier was trained that can distinguish between cancer and control/normal patients.

Appendix:

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  import scipy.io
8  import csv
9  import numpy as np
10 import random
```

```
11 import matplotlib.pyplot as plt
12 import torch
13 import torch.nn as nn
14 import torch.nn.functional as F
15
16
17 # In[2]:
18
19
20 # Load .mat file
21 mat = scipy.io.loadmat('ovarian_dataset.mat')
22
23 # Specify the variable name to convert to CSV
24 variable_name1 = 'ovarianInputs'
25 variable_name2 = 'ovarianTargets'
26 # Get the data from the loaded .mat file
27 #print(mat)
28 data1 = mat[variable_name1]
29 data2 = mat[variable_name2]
30
31 # Specify the CSV file name
32 csv_file_1 = 'data_1.csv'
33 csv_file_2 = 'data_2.csv'
34
35 # Write the data to CSV
36 with open(csv_file_1, 'w', newline='') as csvfile:
37     csvwriter = csv.writer(csvfile)
38     #for row in data1:
39     #    csvwriter.writerow(row)
40     for idx, row in enumerate(data1):
41         csvwriter.writerow(row)
42
43 with open(csv_file_2, 'w', newline='') as csvfile:
44     csvwriter = csv.writer(csvfile)
45     #for row in data1:
46     #    csvwriter.writerow(row)
47     for idx, row in enumerate(data2):
48         csvwriter.writerow(row)
49
50
51 # In[3]:
52
53
54 #Reading Data from .csv file
55 with open('data_1.csv', 'r') as f:
56     reader = csv.reader(f)
57     data_features = list(reader)
58
59 data_features = np.array(data_features, dtype=np.float32)
```

```
60
61 with open('data_2.csv', 'r') as f:
62     reader = csv.reader(f)
63     data_labels = list(reader)
64
65 data_labels = np.array(data_labels, dtype=np.float32)
66 #data_labels = data_labels[0,:]
67 #data_labels = data_labels.reshape((1,data_labels.shape[0]))
68 #print(data_array.shape)
69 #print(data_array)
70
71
72
73 print(data_features)
74
75
76 # In[4]:
77
78
79 def mean(x): # np.mean(X, axis = 0)
80     return sum(x)/len(x)
81
82 def std(x): # np.std(X, axis = 0)
83     return (sum((i - mean(x))**2 for i in x)/len(x))**0.5
84
85 def Standardize_data(X):
86     xx = X.transpose()
87     #print(xx)
88     summ = np.sum(xx, axis=0)
89     print(xx.shape[0])
90     mean = summ/xx.shape[0]
91     print(summ.shape)
92     stdd = np.std(xx,axis=0)
93     print(stdd.shape)
94     #X = X.transpose()
95     xx = (xx-mean)/stdd
96     #for i in range(X.shape[1]):
97     #    X[:,i] = (X[:,i]-summ[i])/stdd[i]
98
99     return xx
100 print(data_features.shape)
101 data_features = Standardize_data(data_features)
102 print(data_features.shape)
103 #data_features_2 = data_features.transpose()
104 #print(data_features)
105
106
107 # In[ ]:
108
```

```
109
110 def covariance(x):
111     #print(x.shape[0])
112     return (x.T @ x)/(x.shape[0]-1)
113
114 cov_mat = np.cov(data_features, rowvar=False)
115
116 # cov_mat = covariance(data_features) # np.cov(X_std.T)
117 print(cov_mat.shape)
118
119
120 # In[ ]:
121
122
123 from numpy.linalg import eig
124
125 # Eigendecomposition of covariance matrix
126 eig_vals, eig_vecs = eig(cov_mat)
127
128 # Adjusting the eigenvectors (loadings) that are largest in ...
    absolute value to be positive
129 max_abs_idx = np.argmax(np.abs(eig_vecs), axis=0)
130 signs = np.sign(eig_vecs[max_abs_idx, range(eig_vecs.shape[0])])
131 eig_vecs = eig_vecs*signs[np.newaxis,: ]
132 eig_vecs = eig_vecs.T
133
134 print('Eigenvalues \n', eig_vals)
135 print('Eigenvectors \n', eig_vecs)
136
137
138 # In[ ]:
139
140
141 # We first make a list of (eigenvalue, eigenvector) tuples
142 eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[i,:]) for i in ...
    range(len(eig_vals))]
143
144 # Then, we sort the tuples from the highest to the lowest based ...
    on eigenvalues magnitude
145 eig_pairs.sort(key=lambda x: x[0], reverse=True)
146
147 # For further usage
148 eig_vals_sorted = np.array([x[0] for x in eig_pairs])
149 eig_vecs_sorted = np.array([x[1] for x in eig_pairs])
150
151 #print(eig_pairs)
152
153
154 # In[ ]:
```

```
155
156
157 eig_vals_total = sum(eig_vals)
158 i=0
159 cum_sum = 0
160 threshold = 95
161 while(cum_sum<threshold):
162     cum_sum = cum_sum + eig_vals_sorted[i]/eig_vals_total*100
163     print(cum_sum)
164     i+=1
165 print(i)
166 explained_variance = [(i / eig_vals_total)*100 for i in ...
    eig_vals_sorted]
167 explained_variance = np.round(explained_variance, 2)
168 cum_explained_variance = np.cumsum(explained_variance)
169
170 print('Explained variance: {}'.format(explained_variance))
171 print('Cumulative explained variance: ...
    {}'.format(cum_explained_variance))
172
173 #plt.plot(np.arange(1,n-features+1), cum_explained_variance, '-o')
174 #plt.xticks(np.arange(1,n-features+1))
175 #plt.xlabel('Number of components')
176 #plt.ylabel('Cumulative explained variance');
177 #plt.show()
178
179
180 # Select top k eigenvectors
181 k = i
182 W = eig_vecs_sorted[:k, :] # Projection matrix
183
184 print(W.shape)
185
186
187 # In[ ]:
188
189
190 X_proj = data_features.dot(W.T)
191
192 print(X_proj.shape)
193
194
195 # In[ ]:
196
197
198 def load_file(arr_feat, arr_lab, x1, x2):
199     #arr_feat = arr_feat.transpose()
200     arr_lab = arr_lab.transpose()
201     arr_shape = arr_feat.shape
```

```

202     print(arr_shape)
203     train = int(x1*arr_shape[0])
204     val = int(x2*arr_shape[0])
205     idx = np.random.randint(low=0, high=arr_shape[0], ...
        size=arr_shape[0], dtype=int)
206
207     new_arr = arr_feat[idx]
208     new_lbs = arr_lab[idx]
209     train_arr = new_arr[0:train]
210     val_arr = new_arr[train:train+val]
211     test_arr = new_arr[train+val:]
212     train_lb = new_lbs[0:train]
213     val_lb = new_lbs[train:train+val]
214     test_lb = new_lbs[train+val:]
215
216     print("Training data size: ", train_arr.T.shape)
217     print("Training label size: ", train_lb.T.shape)
218     print("Testing data size: ", test_arr.T.shape)
219     print("Testing label size: ", test_lb.T.shape)
220     print("Validation data size: ", val_arr.T.shape)
221     print("Validation label size: ", val_lb.T.shape)
222
223     return train_arr.T, test_arr.T, train_lb.T, test_lb.T, ...
        val_arr.T, val_lb.T
224
225 dataset = load_file(X_proj, data_labels, 0.8, 0.1)
226
227
228 # In[ ]:
229
230
231 m = 1
232 def initialize(n_x, C1, C2):
233     #     global W, b
234     np.random.seed(10)
235     W1 = np.random.randn(n_x, C1)*0.1
236     b1 = np.zeros((C1, 1))
237     W2 = np.random.randn(C1, C2)*0.1
238     b2 = np.zeros((C2, 1))
239
240     return W1, b1, W2, b2
241
242 def softmax(z):
243     t = np.exp(z)
244     a = t / np.sum(t, keepdims=True, axis=0)
245     return a
246
247 def sigmoid(z):
248     return 1/(1+np.exp(-z))

```



```

249
250 def forward(W, X, b, activation=None):
251     # global Z,A
252     Z = np.dot(W.T, X) + b # Z.shape is (C,m)
253     if activation == 'sigmoid':
254         A = sigmoid(Z)
255     else:
256         A = Z
257     return Z, A
258 def cost(A, Y_hot):
259     # global L,J
260     # Calculate Loss
261     L = 0.5*np.sum((A-Y_hot), keepdims=True, axis=0) # L.shape is ...
262     (C,m)
263     J = np.mean(L)
264     return L,J
265 # Genralized backprop function for multiple layers
266 def backward(X, Y_hot, A, Z, W, b, activation=None, cache=None):
267     # global dW,db
268     if activation == 'softmax':
269         dZ = A - Y_hot
270     elif activation == 'sigmoid':
271         dZ = np.dot(cache[1], cache[0]) * A * (1-A)
272     else:
273         dZ = A - Y_hot
274
275     dW = np.dot(X, dZ.T) / m
276     db = np.mean(dZ, keepdims=True, axis=1)
277     return dW, db, dZ
278
279 def update(W, b, dW, db, learning_rate):
280     W = W - learning_rate*dW
281     b = b - learning_rate*db
282     return W,b
283
284 def SGD(X, Y_hot, W1, b1, W2, b2, learning_rate):
285     Z1, A1 = forward(W1, X, b1, 'sigmoid')
286     Z2, A2 = forward(W2, A1, b2, 'softmax')
287     L, J = cost(A2, Y_hot)
288     dW2, db2, dZ2 = backward(A1, Y_hot, A2, Z2, W2, b2)
289     dW1, db1, _ = backward(X, Y_hot, A1, Z1, W1, b1, ...
290     'sigmoid', cache=(dZ2, W2))
291     W1, b1 = update(W1, b1, dW1, db1, learning_rate)
292     W2, b2 = update(W2, b2, dW2, db2, learning_rate)
293     return W1, b1, W2, b2, J
294
295 def predict(W1, b1, W2, b2, X):

```

```

296     _, A1 = forward(W1, X, b1, 'sigmoid')
297     _, A2 = forward(W2, A1, b2, 'softmax')
298     return A2
299
300 def accuracy(Y_pred, Y):
301     return np.mean(Y_pred == Y)
302
303 W1, b1, W2, b2 = initialize(8, 16, 2)
304 learning_rate = 0.001
305 costs = []
306 accs = []
307 #use SGD to train the model and validate at same time
308 for i in range(100):
309     for j in range(dataset[0].shape[1]):
310         X = dataset[0][:,j].reshape(-1,1)
311         Y = dataset[2][:,j].reshape(-1,1)
312         W1,b1,W2,b2,J = SGD(X, Y, W1, b1, W2, b2, learning_rate)
313         costs.append(abs(J))
314         #validate
315         Y_pred = predict(W1, b1, W2, b2, dataset[4])
316         acc = accuracy(np.argmax(Y_pred, axis=0), ...
317                        np.argmax(dataset[5], axis=0))
318         print(f'Epoch {i+1}: Cost {J}, Val-accuracy {acc}')
319         accs.append(acc)
320
321 # plt.plot(costs)
322 # plt.show()
323 # plt.plot(accs)
324 # plt.show()
325
326 Y_pred = predict(W1, b1, W2, b2, dataset[1])
327
328 # confusion matrix
329 from sklearn.metrics import confusion_matrix
330 Y_final = np.where(Y_pred ≥ 0.6, 1, 0)
331 accuracy(Y_final,dataset[3])
332 # print("Accuracy: ", accuracy(Y_final,dataset[3])*100,"%")
333 cfm = confusion_matrix(np.argmax(dataset[3],axis=0), ...
334                        np.argmax(Y_pred, axis=0))
335 TP = cfm[0][0]
336 TN = cfm[1][1]
337 FP = cfm[1][0]
338 FN = cfm[0][1]
339 #Specificity
340 Specificity = TN/(TN+FP)
341 #Sensitivity
342 Sensitivity = TP/(TP+FN)
343 # print(f'Specificity: {Specificity}, Sensitivity: {Sensitivity}')

```

```
343
344 # In[ ]:
345
346
347 print("Accuracy: ", accuracy(Y_final,dataset[3])*100,"%")
348 print(f'Specificity: {Specificity}, Sensitivity: {Sensitivity}')
349
350
351 # In[ ]:
352
353
354 plt.plot(accs)
355 plt.xlabel('Epoch')
356 plt.ylabel('Validation Accuracy')
357 plt.title('Validation Accuracy vs Epoch')
358
359
360 # In[ ]:
361
362
363 plt.plot(costs)
364 plt.xlabel('Epoch')
365 plt.ylabel('Loss')
366 plt.title('Loss vs Epoch')
367 plt.show()
368
369
370 # In[ ]:
371
372
373 # ROC curve
374 from sklearn.metrics import roc_curve
375 fpr, tpr, thresholds = roc_curve(np.argmax(dataset[3],axis=0), ...
    np.argmax(Y_pred, axis=0))
376 plt.plot(fpr, tpr)
377 plt.xlabel('FPR')
378 plt.ylabel('TPR')
379 plt.title('ROC Curve')
380
381
382 # In[ ]:
383
384
385 #AUC
386 from sklearn.metrics import auc
387 print("Accuracy: ", accuracy(Y_final,dataset[3])*100,"%")
388 print(f'Specificity: {Specificity}, Sensitivity: {Sensitivity}')
389 print("AUC = ",auc(fpr, tpr))
```

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[12]:
5
6
7  import scipy.io
8  import csv
9  import numpy as np
10 import random
11 import matplotlib.pyplot as plt
12 import torch
13 import torch.nn as nn
14 import torch.nn.functional as F
15
16
17 # In[13]:
18
19
20 # Load .mat file
21 mat = scipy.io.loadmat('ovarian_dataset.mat')
22
23 # Specify the variable name to convert to CSV
24 variable_name1 = 'ovarianInputs'
25 variable_name2 = 'ovarianTargets'
26 # Get the data from the loaded .mat file
27 #print(mat)
28 data1 = mat[variable_name1]
29 data2 = mat[variable_name2]
30
31 # Specify the CSV file name
32 csv_file_1 = 'data_1.csv'
33 csv_file_2 = 'data_2.csv'
34
35 # Write the data to CSV
36 with open(csv_file_1, 'w', newline='') as csvfile:
37     csvwriter = csv.writer(csvfile)
38     #for row in data1:
39     #    csvwriter.writerow(row)
40     for idx, row in enumerate(data1):
41         csvwriter.writerow(row)
42
43 with open(csv_file_2, 'w', newline='') as csvfile:
44     csvwriter = csv.writer(csvfile)
45     #for row in data1:
46     #    csvwriter.writerow(row)
47     for idx, row in enumerate(data2):
48         csvwriter.writerow(row)
```

```
49
50
51 # In[14]:
52
53
54 #Reading Data from .csv file
55 with open('data_1.csv', 'r') as f:
56     reader = csv.reader(f)
57     data_features = list(reader)
58
59 data_features = np.array(data_features, dtype=np.float32)
60
61 with open('data_2.csv', 'r') as f:
62     reader = csv.reader(f)
63     data_labels = list(reader)
64
65 data_labels = np.array(data_labels, dtype=np.float32)
66 #data_labels = data_labels[0,:]
67 #data_labels = data_labels.reshape((1,data_labels.shape[0]))
68 #print(data_array.shape)
69 #print(data_array)
70
71
72
73 print(data_features.shape)
74
75 # In[15]:
76
77
78 # Creating a PyTorch class
79 # 28*28 ==> 9 ==> 28*28
80 class AE(torch.nn.Module):
81     def __init__(self):
82         super().__init__()
83
84         # Building an linear encoder with Linear
85         # layer followed by Relu activation function
86         # 784 ==> 9
87         self.encoder = torch.nn.Sequential(
88             torch.nn.Linear(100, 64),
89             torch.nn.ReLU(),
90             torch.nn.Linear(64, 32),
91             torch.nn.ReLU(),
92             torch.nn.Linear(32, 8),
93         )
94
95         # Building an linear decoder with Linear
96         # layer followed by Relu activation function
97         # The Sigmoid activation function
```

```
98         # outputs the value between 0 and 1
99         # 9 ==> 784
100         self.decoder = torch.nn.Sequential(
101             torch.nn.Linear(8, 32),
102             torch.nn.ReLU(),
103             torch.nn.Linear(32, 64),
104             torch.nn.ReLU(),
105             torch.nn.Linear(64, 100),
106             #torch.nn.Sigmoid()
107         )
108
109     def forward(self, x):
110         encoded = self.encoder(x)
111         decoded = self.decoder(encoded)
112         return encoded, decoded
113
114
115 # In[16]:
116
117
118 # Model Initialization
119 model = AE()
120
121 # Validation using MSE Loss function
122 loss_function = torch.nn.MSELoss()
123
124 # Using an Adam Optimizer with lr = 0.1
125 optimizer = torch.optim.Adam(model.parameters(),
126                               lr = 0.005,
127                               weight_decay = 1e-8)
128
129
130 # In[17]:
131
132
133 epochs = 25
134 batch_size = 36
135 outputs = []
136 losses = []
137 for epoch in range(epochs):
138     for i in range(data.features.shape[1]//batch_size):
139
140         batch = data.features[:,i*batch_size:(i+1)*batch_size]
141         # Reshaping the image to (-1, 784)
142         #image = image.reshape(-1, 28*28)
143
144         # Output of Autoencoder
145         batch = batch.transpose()
146         batch = torch.Tensor(batch)
```

```
147     encoded, reconstructed = model(batch)
148
149     # Calculating the loss function
150     #print(reconstructed)
151     loss = loss_function(reconstructed, batch)
152
153     # The gradients are set to zero,
154     # the gradient is computed and stored.
155     # .step() performs parameter update
156     optimizer.zero_grad()
157     loss.backward()
158     optimizer.step()
159
160     # Storing the losses in a list for plotting
161     losses.append(loss.detach())
162     outputs.append((epochs, batch, reconstructed))
163
164 # Defining the Plot Style
165 #plt.style.use('fivethirtyeight')
166 plt.xlabel('Iterations')
167 plt.ylabel('Loss')
168
169 # Plotting the last 100 values
170 plt.plot(losses)
171
172
173 # In[18]:
174
175
176 features = torch.Tensor(data.features.transpose())
177 encoded, _ = model(features)
178 latent = encoded.detach().numpy()
179 print(latent.shape)
180
181
182 # In[19]:
183
184
185 def load_file(arr_feat, arr_lab, x1, x2):
186     #arr_feat = arr_feat.transpose()
187     arr_lab = arr_lab.transpose()
188     arr_shape = arr_feat.shape
189     print(arr_shape)
190     train = int(x1*arr_shape[0])
191     val = int(x2*arr_shape[0])
192     idx = np.random.randint(low=0, high=arr_shape[0], ...
193                             size=arr_shape[0], dtype=int)
194
195     new_arr = arr_feat[idx]
```

```

195     new_lbs = arr_lab[idx]
196     train_arr = new_arr[0:train]
197     val_arr = new_arr[train:train+val]
198     test_arr = new_arr[train+val:]
199     train_lb = new_lbs[0:train]
200     val_lb = new_lbs[train:train+val]
201     test_lb = new_lbs[train+val:]
202
203     print("Training data size: ", train_arr.T.shape)
204     print("Training label size: ", train_lb.T.shape)
205     print("Testing data size: ", test_arr.T.shape)
206     print("Testing label size: ", test_lb.T.shape)
207     print("Validation data size: ", val_arr.T.shape)
208     print("Validation label size: ", val_lb.T.shape)
209
210     return train_arr.T, test_arr.T, train_lb.T, test_lb.T, ...
        val_arr.T, val_lb.T
211
212 dataset = load_file(latent,data_labels,0.8,0.1)
213
214
215 # In[44]:
216
217
218 m = 1
219 def initialize(n_x,C1,C2):
220     #     global W,b
221     np.random.seed(10)
222     W1 = np.random.randn(n_x,C1)*0.1
223     b1 = np.zeros((C1,1))
224     W2 = np.random.randn(C1,C2)*0.1
225     b2 = np.zeros((C2,1))
226
227     return W1, b1, W2, b2
228
229 def softmax(z):
230     t = np.exp(z)
231     a = t / np.sum(t, keepdims=True, axis=0)
232     return a
233
234 def sigmoid(z):
235     return 1/(1+np.exp(-z))
236
237 def forward(W, X, b,activation=None):
238     #     global Z,A
239     Z = np.dot(W.T, X) + b # Z.shape is (C,m)
240     if activation == 'sigmoid':
241         A = sigmoid(Z)
242     else:

```



```

243         A = Z
244     return Z, A
245 def cost(A, Y_hot):
246     #     global L,J
247     # Calculate Loss
248     L = 0.5*np.sum((A-Y_hot),keepdims=True, axis=0) # L.shape is ...
249     J = np.mean(L)
250     return L,J
251
252 # Genralized backprop function for multiple layers
253 def backward(X, Y_hot, A, Z, W, b, activation=None,cache=None):
254     #     global dW,db
255     if activation == 'softmax':
256         dZ = A - Y_hot
257     elif activation == 'sigmoid':
258         dZ = np.dot(cache[1],cache[0])*A*(1-A)
259     else:
260         dZ = A - Y_hot
261
262     dW = np.dot(X, dZ.T)/m
263     db = np.mean(dZ, keepdims=True, axis=1)
264     return dW, db,dZ
265
266 def update(W, b, dW, db, learning_rate):
267     W = W - learning_rate*dW
268     b = b - learning_rate*db
269     return W,b
270
271 def SGD(X, Y_hot, W1, b1, W2, b2, learning_rate):
272     Z1, A1 = forward(W1, X, b1, 'sigmoid')
273     Z2, A2 = forward(W2, A1, b2, 'softmax')
274     L, J = cost(A2, Y_hot)
275     dW2, db2,dZ2 = backward(A1, Y_hot, A2, Z2, W2, b2)
276     dW1, db1, _ = backward(X, Y_hot, A1, Z1, W1, b1, ...
277         'sigmoid',cache=(dZ2,W2))
278     W1,b1 = update(W1, b1, dW1, db1, learning_rate)
279     W2,b2 = update(W2, b2, dW2, db2, learning_rate)
280     return W1,b1,W2,b2,J
281
282 def predict(W1, b1, W2, b2, X):
283     _, A1 = forward(W1, X, b1, 'sigmoid')
284     _, A2 = forward(W2, A1, b2, 'softmax')
285     return A2
286
287 def accuracy(Y_pred, Y):
288     return np.mean(Y_pred == Y)
289

```

```

290 W1, b1, W2, b2 = initialize(8, 16, 2)
291 learning_rate = 0.001
292 costs = []
293 accs = []
294 #use SGD to train the model and validate at same time
295 for i in range(100):
296     for j in range(dataset[0].shape[1]):
297         X = dataset[0][:,j].reshape(-1,1)
298         Y = dataset[2][:,j].reshape(-1,1)
299         W1,b1,W2,b2,J = SGD(X, Y, W1, b1, W2, b2, learning_rate)
300         costs.append(abs(J))
301         #validate
302         Y_pred = predict(W1, b1, W2, b2, dataset[4])
303         acc = accuracy(np.argmax(Y_pred, axis=0), ...
            np.argmax(dataset[5], axis=0))
304         print(f'Epoch {i+1}: Cost {J}, Val_accuracy {acc}')
305         accs.append(acc)
306
307 # plt.plot(costs)
308 # plt.show()
309 # plt.plot(accs)
310 # plt.show()
311
312 Y_pred = predict(W1, b1, W2, b2, dataset[1])
313
314 # confusion matrix
315 from sklearn.metrics import confusion_matrix
316 Y_final = np.where(Y_pred ≥ 0.6, 1, 0)
317 accuracy(Y_final,dataset[3])
318 # print("Accuracy: ", accuracy(Y_final,dataset[3])*100,"%")
319 cfm = confusion_matrix(np.argmax(dataset[3],axis=0), ...
            np.argmax(Y_pred, axis=0))
320 TP = cfm[0][0]
321 TN = cfm[1][1]
322 FP = cfm[1][0]
323 FN = cfm[0][1]
324 #Specificity
325 Specificity = TN/(TN+FP)
326 #Sensitivity
327 Sensitivity = TP/(TP+FN)
328 # print(f'Specificity: {Specificity}, Sensitivity: {Sensitivity}')
329
330
331 # In[45]:
332
333
334 print("Accuracy: ", accuracy(Y_final,dataset[3])*100,"%")
335 print(f'Specificity: {Specificity}, Sensitivity: {Sensitivity}')
336

```

```
337
338 # In[46]:
339
340
341 plt.plot(accs)
342 plt.xlabel('Epoch')
343 plt.ylabel('Validation Accuracy')
344 plt.title('Validation Accuracy vs Epoch')
345
346
347 # In[47]:
348
349
350 plt.plot(costs)
351 plt.xlabel('Epoch')
352 plt.ylabel('Loss')
353 plt.title('Loss vs Epoch')
354 plt.show()
355
356
357 # In[48]:
358
359
360 # ROC curve
361 from sklearn.metrics import roc_curve
362 fpr, tpr, thresholds = roc_curve(np.argmax(dataset[3],axis=0), ...
    np.argmax(Y_pred, axis=0))
363 plt.plot(fpr, tpr)
364 plt.xlabel('FPR')
365 plt.ylabel('TPR')
366 plt.title('ROC Curve')
367
368
369 # In[43]:
370
371
372 #AUC
373 from sklearn.metrics import auc
374 print("Accuracy: ", accuracy(Y_final,dataset[3])*100,"%")
375 print(f'Specificity: {Specificity}, Sensitivity: {Sensitivity}')
376 print("AUC = ",auc(fpr, tpr))
```