Home Assignment

Names: Lidor Erez, Dvir Rehavi, Guy Mizrahi

First Task

Show that the maximum likelihood estimates for a univariate Normal distribution with unknown mean and variance are given by:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})(x_i - \hat{\mu})^T$$

Solution

The likelihood function for a univariate normal distribution is given by:

$$L(\mu, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

Simplifying the above expression, we get:

$$L(\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{n/2}} e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2}$$

Taking the log of the likelihood function, we get:

$$\log L(\mu, \sigma^2) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2$$

Taking the derivative of the log likelihood function with respect to μ and σ^2, we get:

$$\frac{\partial \log L(\mu, \sigma^2)}{\partial \mu} = \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu) = 0$$

$$\frac{\partial \log L(\mu, \sigma^2)}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4}\sum_{i=1}^{n}(x_i - \mu)^2 = 0$$

Finding $\hat{\mu}$

$\frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu) = 0$

$\sum_{i=1}^{n}(x_i - \mu) = 0$

$\sum_{i=1}^{n} x_i - n\mu = 0$

$\sum_{i=1}^{n} x_i = n\mu$

$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$

Finding $\hat{\sigma}^2$

$-\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4}\sum_{i=1}^{n}(x_i - \mu)^2 = 0$

$\frac{1}{2\sigma^4}\sum_{i=1}^{n}(x_i - \mu)^2 = \frac{n}{2\sigma^2}$

$\sum_{i=1}^{n}(x_i - \mu)^2 = n\sigma^2$

$\sum_{i=1}^{n}(x_i - \hat{\mu})^2 = n\hat{\sigma}^2$

$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2$

Thus the maximum likelihood estimates for a univariate Normal distribution with unknown mean and variance are given by:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2$$

Second Task

In [30]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from NN import NN, NeuralNetworkCV,trainNN
from RF_TASK import RF, RandomForestCV
import torch
```

In [31]:
```
# Loading Data
M1 = np.loadtxt("./data/M1.csv",delimiter=',')
M2 = np.loadtxt("./data/M2.csv",delimiter=',')
S1 = np.loadtxt('./data/Sigma1.csv',delimiter=',')
S2 = np.loadtxt('./data/Sigma2.csv',delimiter=',')

# A priori probabilities
P1 = 0.35
P2 = 0.65
```

** Task 2.1 **

Generate 10,000 observations from the two distributions, proportionate to the a priori probabilities, which will be the training set.

In [32]:
```
# sample size for training set
n = 10000
sample_size_1 = int(n*P1)
sample_size_2 = int(n*P2)

# Create Distribution
np.random.seed(11)
dist1 = np.random.multivariate_normal(M1,S1, sample_size_1)
dist2 = np.random.multivariate_normal(M2,S2,sample_size_2)
y1 = np.ones(sample_size_1)
y2 = np.zeros(sample_size_2)

# stack distributions:
data = np.vstack([dist1,dist2])
labels = np.hstack([y1,y2])
```

** Task 2.2 **

Compute the MLE estimators for each of the class conditional parameters and compare them to the true values.

In [33]:
```
def gaussian_mle(data,dist):
    """
    Function to estimate the parameters of a Gaussian distribution
    """
    mu = np.mean(data, axis=0)
    sigma = (data - mu).T @ (data - mu) / len(data)
    print(f"Estimated mu for dist{dist}:\n{mu}",end='\n\n')
    print(f"Estimated sigma for dist{dist}:\n {sigma}",end='\n\n')
    return mu, sigma
```

In [34]:
```
mu_hat_dist1, sigma_hat_dist1 = gaussian_mle(dist1,1)
mu_hat_dist2, sigma_hat_dist2 = gaussian_mle(dist2,2)
```

```
Estimated mu for dist1:
[6.50938839 5.8243426  6.75761525 7.33637681 7.7731861  4.38719271]

Estimated sigma for dist1:
 [[10.39686125 -0.34800071 -2.75480451  0.43475269  3.17679352  1.6901788
4]
 [-0.34800071  2.57758081  0.85688225 -0.2666876   1.32473075  0.37673872]
 [-2.75480451  0.85688225  4.29628595 -1.03926033 -1.03506258  1.19061338]
 [ 0.43475269 -0.2666876  -1.03926033  3.49492454 -0.91519742  0.29000415]
 [ 3.17679352  1.32473075 -1.03506258 -0.91519742  2.80010793  0.78827087]
 [ 1.69017884  0.37673872  1.19061338  0.29000415  0.78827087  2.4535043
9]]

Estimated mu for dist2:
[6.19666394 5.84103995 6.64614838 6.11156466 6.97733738 4.3451731 ]

Estimated sigma for dist2:
 [[ 9.82420388 -3.88368511 -3.5253374  -0.13725756  1.49088391 -2.9554119
2]
 [-3.88368511  3.16989061  0.09933475 -0.06451439 -1.10436848  2.24385832]
 [-3.5253374   0.09933475  3.69315294  0.41166515  0.3467576  -0.55595673]
 [-0.13725756 -0.06451439  0.41166515  1.8219158  -0.36514266 -0.50389575]
 [ 1.49088391 -1.10436848  0.3467576  -0.36514266  2.50043393 -0.60491785]
 [-2.95541192  2.24385832 -0.55595673 -0.50389575 -0.60491785  2.6195776
4]]
```

In [35]:
```python
print("Difference between estimated and real mean of Distribution 1:")
mu_hat_dist1 - M1
```

```
Difference between estimated and real mean of Distribution 1:
```

Out[35]:
```
array([ 0.06878839, -0.0142574 ,  0.00821525, -0.02172319,  0.0076861 ,
        0.02009271])
```

In [36]:
```python
print("Difference between estimated and real sigma of Distribution 1:")
sigma_hat_dist1 - S1
```

```
Difference between estimated and real sigma of Distribution 1:
```

Out[36]:
```
array([[ 0.11386125, -0.08231071, -0.08600451, -0.00014731, -0.01970648,
        -0.08792116],
       [-0.08231071,  0.01908081,  0.01618225,  0.0361624 , -0.00326925,
         0.02086872],
       [-0.08600451,  0.01618225,  0.08508595,  0.02843967, -0.05339258,
         0.02461338],
       [-0.00014731,  0.0361624 ,  0.02843967, -0.09997546,  0.01554258,
        -0.00651585],
       [-0.01970648, -0.00326925, -0.05339258,  0.01554258,  0.00800793,
        -0.04400913],
       [-0.08792116,  0.02086872,  0.02461338, -0.00651585, -0.04400913,
        -0.06319561]])
```

In [37]:
```python
print("Difference between estimated and real mean of Distribution 2:")
mu_hat_dist2 - M2
```

```
Difference between estimated and real mean of Distribution 2:
```

Out[37]:
```
array([-0.00213606, -0.00536005,  0.03974838, -0.00373534,  0.04643738,
       -0.0245269 ])
```

In [38]:
```python
print("Difference between estimated and real sigma of Distribution 2:")
sigma_hat_dist2 - S2
```

```
Difference between estimated and real sigma of Distribution 2:
```

Out[38]:
```
array([[ 0.00960388,  0.02281489,  0.0246626 , -0.04548556, -0.06581609,
        -0.06191192],
       [ 0.02281489, -0.02790939, -0.00348525,  0.03714561,  0.05813152,
         0.00075832],
       [ 0.0246626 , -0.00348525, -0.05114706, -0.02816485, -0.0139224 ,
         0.05274327],
       [-0.04548556,  0.03714561, -0.02816485,  0.0027158 , -0.07789266,
         0.05503425],
       [-0.06581609,  0.05813152, -0.0139224 , -0.07789266, -0.01586607,
         0.07618215],
       [-0.06191192,  0.00075832,  0.05274327,  0.05503425,  0.07618215,
        -0.00022236]])
```

** Task 2.3 **

Generate another set, with 2,000 observations (this will serve as validation set).

In [39]:
```python
# Sample Size
n_val = 2000
sample_size1_val = int(n_val*P1)
sample_size2_val = int(n_val*P2)

# Create distributions

np.random.seed(11)
dist1_val = np.random.multivariate_normal(M1,S1, sample_size1_val)
dist2_val = np.random.multivariate_normal(M2,S2,sample_size2_val)

# Create labels
y1_val = np.ones(sample_size1_val)
y2_val = np.zeros(sample_size2_val)

# stack distributions for validation
data_val = np.vstack([dist1_val,dist2_val])

# stack labels for validation
labels_val = np.hstack([y1_val,y2_val])
```

** Task 2.4 **

Fit a random forest to the data. Use the validation set to compare a number of forest configurations and choose the best performing one. Then use CV-10 over the training set to estimate the model accuracy and generalization error. (You may not use existing functions for the cross-validation for optimization and estimation part but write your own).

In [40]:
```python
from itertools import product
from tqdm import tqdm
```

Chossing the best performing configuration:

In [41]:
```python
dct = {
    'n_estimator':[300,500],
    'max_depth':[5,7],
```

```
    'min_samples_leaf':[2,4],
    'min_samples_split':[2,5,10]
}
```

In [42]:

```
# This function creates combinations from the feature space dictionary.
def parameterSub(paramter_space):
    values = paramter_space.values()
    feature_combs = {}
    combinations = list(product(*values))

    for i,comb in enumerate(combinations):
        feature_combs[f'combination_{i}'] = comb

    return feature_combs

param_space = parameterSub(dct)
param_space
```

Out[42]:

```
{'combination_0': (300, 5, 2, 2),
 'combination_1': (300, 5, 2, 5),
 'combination_2': (300, 5, 2, 10),
 'combination_3': (300, 5, 4, 2),
 'combination_4': (300, 5, 4, 5),
 'combination_5': (300, 5, 4, 10),
 'combination_6': (300, 7, 2, 2),
 'combination_7': (300, 7, 2, 5),
 'combination_8': (300, 7, 2, 10),
 'combination_9': (300, 7, 4, 2),
 'combination_10': (300, 7, 4, 5),
 'combination_11': (300, 7, 4, 10),
 'combination_12': (500, 5, 2, 2),
 'combination_13': (500, 5, 2, 5),
 'combination_14': (500, 5, 2, 10),
 'combination_15': (500, 5, 4, 2),
 'combination_16': (500, 5, 4, 5),
 'combination_17': (500, 5, 4, 10),
 'combination_18': (500, 7, 2, 2),
 'combination_19': (500, 7, 2, 5),
 'combination_20': (500, 7, 2, 10),
 'combination_21': (500, 7, 4, 2),
 'combination_22': (500, 7, 4, 5),
 'combination_23': (500, 7, 4, 10)}
```

In [43]:

```
res = {}
for key,val in tqdm(param_space.items()):
    rf = RF(*val)
    rf.train(data,labels)
    rf.predict(data_val)
    f1_score,_ = rf.computeMetrics(labels_val)
    res[key] = (f1_score,rf.params)
```

```
100%|██████████| 24/24 [01:15<00:00,  3.13s/it]
```

In [44]:

```
items = list(res.items())
by_f1 = sorted(items, key=lambda item: item[1][0], reverse=True)[0]
by_f1
```

Out[44]:
```
('combination_8',
 (0.8211189913317573,
  {'n_estimators': 300,
   'max_depth': 7,
   'min_samples_leaf': 2,
   'min_samples_split': 10}))
```

Estimating the model accuracy and generalization error using CV-10:

In [45]:
```
best_params = by_f1[1][1] # get best parameters
best_rf = RF(*best_params.values()) # deploy a new RF with the best param

# Train using CV
rf_cv = RandomForestCV(best_rf,10,data,labels)
rf_cv.runCV()
```

Out[45]:
```
(0.6606637806637806, 0.13414414414414416)
```

In [46]:
```
print(f'Random Forest F1 {np.round(rf_cv.avgRFF1*100,3)}%\nRandom Forest
```

```
Random Forest F1 66.066%
Random Forest Generalization Error 13.414%
```

** Task 2.5 **

Fit a neural network to the data. Use the validation set to determine the number of neurons to use for the network. After choosing the number of neurons, use CV-10 over the training set to estimate the classification accuracy

Choosing the number of neurons:

In [47]:
```
input_size = data.shape[1]
output_size = 1
hidden_sizes = [2**i for i in range(4,10)]
result = {}

data_cp = data.copy()
labels_cp = labels.copy()

val_data_cp = data_val.copy()
val_labels_cp = labels_val.copy()

scaler = StandardScaler()

data_cp = scaler.fit_transform(data_cp)
val_data_cp = scaler.fit_transform(val_data_cp)

data_tens = torch.from_numpy(data_cp).to(torch.float32)
labels_tens = torch.from_numpy(labels_cp)
labels_tens = labels_tens.to(torch.float32)
labels_tens = labels_tens.unsqueeze(1)

val_data_tens = torch.from_numpy(val_data_cp).to(torch.float32)
val_labels_tens = torch.from_numpy(val_labels_cp)
val_labels_tens = val_labels_tens.to(torch.float32)
val_labels_tens = val_labels_tens.unsqueeze(1)
labels_size = labels.shape[0]
```

```python
# Handle imbalance of data using weights in loss function
num_zeros = torch.sum(labels_tens == 0).item()
num_ones = torch.sum(labels_tens == 1).item()

pos_weight = num_zeros/num_ones

criterion = torch.nn.BCEWithLogitsLoss(pos_weight=torch.tensor([pos_weigh

for hidden_size in hidden_sizes:
    net = NN(input_size,hidden_size,output_size)
    optimizer = torch.optim.Adam(net.parameters(),lr=1e-1)

    _,val_loss = trainNN(net,
                                  data_tens,
                                  labels_tens,
                                  optimizer,
                                  criterion,
                                  1000,
                                  200,
                                   val_data_tens,
                                   val_labels_tens,
                                   validate=True)

    result[hidden_size] = (val_loss)
```

In [48]:
```python
result_items = list(result.items())

by_loss = sorted(result_items,key=lambda item: item[0])[0]

print(f'{by_loss[0]} neurons in the hidden layer result in {by_loss[1]} l
```

16 neurons in the hidden layer result in 0.34719195067882536 loss value.

Estimating the classification accuracy using CV-10:

In [49]:
```python
best_net  = NN(input_size,by_loss[0],output_size)
nn_cv = NeuralNetworkCV(data_tens,labels_tens,10,best_net)

criterion = torch.nn.BCEWithLogitsLoss(pos_weight=torch.tensor([pos_weigh
optimizer = torch.optim.Adam(best_net.parameters(),lr=1e-1)
nn_cv.runCV(optimizer, criterion)
```

Out[49]: (0.9126984126984128, 0.24489656016230582)

** Task 2.6 **

Choose one of the two models above. We will now consider the overfitting phenomenon as a function of training set size. Fit the model with training sets of size $N = 10, 20, 30, \ldots, 1,000$. Plot the test and training error as a function of N. *For estimating test error, use the validation set.

In [50]:
```python
net_of = NN(input_size, by_loss[0],output_size)
criterion = torch.nn.BCEWithLogitsLoss(pos_weight=torch.tensor([pos_weigh
optimizer = torch.optim.Adam(net_of.parameters(),lr=1e-1)

train_losses = []
```

```
val_losses = []
for sample_size in range(10,1001,10):
    train_loss,val_loss = trainNN(net_of,
            data_tens[:sample_size,:],
            labels_tens[:sample_size,:],
            optimizer,
            criterion,
            sample_size,
            200,
            val_data_tens,
            val_labels_tens,
            validate=True)
    train_losses.append(train_loss/sample_size)
    val_losses.append(val_loss/200)
```

In [51]:

```
idx = list(range(10,1001,10))
plt.figure(figsize=(10,8))
plt.plot(idx,train_losses,label='Train Error')
plt.plot(idx, val_losses,label='Test Error')
plt.title("Train Error VS Test Error as a function of N")
plt.xlabel("$N = 10,20,30,\ldots,1,000$")
plt.ylabel("Error")
plt.legend()
plt.show()
```