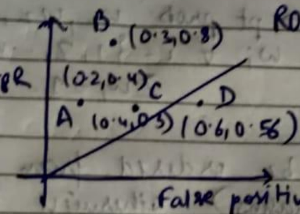
Advanced Statistical Algorithms MA691 Midsem

Name: **Naman Goyal**
Roll No: **180123029**

Ques.**1**

Sol^n (1) :-



ROC → Receiver Operating characteristic
[ROC Curve]

This plot shows :-

TPR (True positive rate) $= \frac{TP}{TP+FN}$

FPR (False positive rate) $= \frac{FP}{(FP+TN)}$

(i) Comparison b/w A & B can't be done because as we move from A → B both TPR and FPR are increasing. Hence, no comparison could be done.

(ii) classifier A is better than C because it has higher TPR than C and lower FPR than C. Similarly, comparison b/w B and D can be done we can say that classifier B is better than D.

(iii) Similarly to (i) C, D can't compare ~~btw~~ as moving from C → D both TPR and FPR are increasing.

(iv) B is better than C as TPR is high and FPR is low. (A, D) ~~can~~ can't be compared as both TPR and FPR increases.

Ques.**2**

given

Sol^n (2): $x_1, x_2 \ldots x_n \rightarrow$ pt prob. $w_1, w_2, \ldots, w_n$.

$f(x = x_i) = w_i \;\forall\; i = \{1, 2, \ldots n\}$

$\sum w_i = 1$.

Let $x_1 x_2 \ldots x_n$ be ordered from smallest to largest (increasing) otherwise that increasing sequence could be taken.

Let $F_n(x) = \sum_{i=1}^{n} f(x_i)\, I(x_i \le x)$

$\Rightarrow F_n(x) = \sum_{i=1}^{n} w_i\, I(x_i \le x)$

↳ α^{th} quantile point $\Rightarrow$

$F^{-1}(\alpha) = \{ x_i : \sum_{j=0}^{i-1} w_j \le \alpha \le \sum_{j=0}^{i} w_j$

Sol^n (3): Let $\theta = \sigma^2$

$p(\theta | y_2 y_2 \ldots y_n) \propto p(y_1 y_2 \ldots y_n | \theta)\, p(\theta)$

$\left(\frac{1}{\sigma}\right)^n e^{-\sum y_i^2/2\sigma^2} \cdot (\sigma^2)^{-7/2} e^{-1/2\sigma^2}$

Ques.**3**

to largest (increasing) otherwise that increasing sequence could be taken.

Let $F_n(x) = \sum_{i=1}^{n} f(x_i)\, I(x_i \le x)$

$\Rightarrow F_n(x) = \sum_{i=1}^{n} w_i\, I(x_i \le x)$

↳ α^{th} quantile point ~~could be~~ ⇒

$$F^{-1}(\alpha) = \left\{ x_i : \sum_{j=0}^{i-1} w_j \le \alpha \le \sum_{j=0}^{i} w_j \right\}$$

Q(3) Let $\theta = \sigma^2$

$$p(\theta \mid y_1\, y_2 \ldots y_n) \propto p(y_1, y_2 \ldots y_n \mid \theta)\, p(\theta)$$

$$= \left(\frac{1}{\sqrt{2\pi}}\right)^n e^{-\sum y_i^2/2\sigma^2} \cdot (\sigma^2)^{-7/2} e^{-1/2\sigma^2}$$

$$= e^{-1/2\sigma^2 (1+\sum y_i^2)} (\sigma^2)^{-7/2-n/2} \left(\frac{1}{\sqrt{2\pi}}\right)^n$$

Hence, we can see that σ^2 is sampled from inverse gamma distributⁿ with params

$$\boxed{\beta = \frac{(1+\sum y_i^2)}{2}} \qquad \alpha + 1 = -\left(-7/2 - n/2\right) \qquad \boxed{\alpha = 5/2 + n/2}$$

Code:

```
import numpy as np
import scipy.stats

mu, sigma = 0, 5 # mean and standard deviation
s = np.random.normal(mu, sigma, 100)

param_alpha = (5+100)/2
param_beta = (1+sum(i*i for i in s))/2;
```

```
print('Alpha = ',param_alpha)
print('Beta = ',param_beta)
print("--------------------------------------------")

i = 0
generated_random = []
while(i<30):
   val = scipy.stats.invgamma.rvs(param_alpha, loc=0, scale=param_beta)
   i=i+1
   generated_random.append(val)

print("Random Values from Inv Gamma distribution")
print(generated_random)

# Bayes Estimate
bayes_estimate = np.mean(generated_random)
std = np.std(generated_random)
print('Bayes_estimate = ',bayes_estimate)
```

Output:
Alpha = 52.5
Beta = 1136.372296241155
--
Random Values from Inv Gamma distribution
[22.797837731420774, 23.187597932973755, 27.245572105094723, 24.810422006808906, 21.26600704906269, 21.31013588446922, 24.026473496915976, 24.930910962542526, 21.979419152804986, 21.930959918563467, 20.983002020287564, 22.029057488293002, 15.410251681804173, 17.427716078975685, 22.78137895504229, 20.22157044513737, 18.134987317591737, 19.748526784908307, 16.541718749700152, 23.43779597691197, 21.03585409565021, 22.501141743918975, 24.830374839492276, 16.37346435406361, 20.710265285619702, 15.847295435070711, 22.0800058152737, 21.25749507589313, 18.54254544859192, 20.96076616931359]
Bayes_estimate = 21.144685000073235

Ques.**4**
Code:

```python
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt
y = []
with open("FRWRD.txt", 'r') as f:
   for line in f.readlines():
       try:
           y.append(float(line))
       except:
           pass
y = np.asarray(y)
x = np.asarray([i for i in range(1,37)]).reshape(-1,1)
m = 8

print(f"Polynomial degree : {m}")
poly = PolynomialFeatures(degree = m)
X_poly = poly.fit_transform(x)

kf = KFold(n_splits=6, shuffle=True, random_state=0)
i = 0
print("Metric used: Mean squared error")
for train, test in kf.split(X_poly):
   i = i+1
   X_train, y_train, X_validation, y_validation = X_poly[train], y[train],
X_poly[test], y[test]
   model = LinearRegression()
   model.fit(X_train, y_train)
   y_hat_train = model.predict(X_train)
   y_hat_val = model.predict(X_validation)
   print(f"Fold {i}:\n train set:
{mean_squared_error(y_hat_train,y_train)} \n val set:
{mean_squared_error(y_hat_val,y_validation)}")
```

```
not_best = model

def score_model(model,X_train,y_train,X_test,y_test):
   y_hat_train = model.predict(X_train)
   y_hat_val = model.predict(X_validation)
   return
mean_squared_error(y_hat_train,y_train),mean_squared_error(y_hat_val,y_val
idation)

X_train, X_test, y_train, y_test=train_test_split(X_poly,y,test_size=0.3)

for alp in [0.01, 0.1,1, 10,100]:
   rr = Ridge(alpha=alp)
   rr.fit(X_train, y_train)
   Ridge_score = score_model(rr, X_train,y_train, X_test, y_test)
   print(f"for alpha = {alp}, score fpr train and test set is
{Ridge_score}")

for alp in [0.01, 0.1,1, 10,100]:
   rr = Lasso(alpha=alp)
   rr.fit(X_train, y_train)
   Ridge_score = score_model(rr, X_train,y_train, X_test, y_test)
   print(f"for alpha = {alp}, score fpr train and test set is
{Ridge_score}")


plt.plot(x, model.predict(X_poly))
plt.plot(x, rr.predict(X_poly))
print("---------------------------------------------")

plt.legend(["Not best","Regularised", "True label"],loc='upper right')
plt.scatter(x,y)
plt.show()
```

Output:
Polynomial degree : 8
Metric used: Mean squared error
Fold 1:
train set: 0.005387561645726775

val set: 0.0064165682137984565
Fold 2:
 train set: 0.004526272826527307
 val set: 0.0794103262436119
Fold 3:
 train set: 0.005272414647275464
 val set: 0.006237627821274647
Fold 4:
 train set: 0.0057277187284529634
 val set: 0.004654875670639628
Fold 5:
 train set: 0.003919724661867668
 val set: 0.020955286543234017
Fold 6:
 train set: 0.003369667646965825
 val set: 0.04554862266601486
for alpha = 0.01, score fpr train and test set is (0.002457304953997490, 0.013602013291232477)
for alpha = 0.1, score fpr train and test set is (0.0024672212937505377, 0.013536114036611986)
for alpha = 1, score fpr train and test set is (0.0024846442219727503, 0.013518291292908746)
for alpha = 10, score fpr train and test set is (0.002547627563608066, 0.013746356601089237)
for alpha = 100, score fpr train and test set is (0.0027793108993350857, 0.014519094067409777)
Duality gap: 0.09969034708805465, tolerance: 3.393329600000001e-05
  model = cd_fast.enet_coordinate_descent(
for alpha = 0.01, score fpr train and test set is (0.007713126588888095, 0.030599464828291567)
You might want to increase the number of iterations. Duality gap: 0.10065931966022318, tolerance: 3.393329600000001e-05
  model = cd_fast.enet_coordinate_descent(
for alpha = 0.1, score fpr train and test set is (0.008040788802023524, 0.031649699972755116)
Duality gap: 0.10102968244035052, tolerance: 3.393329600000001e-05
  model = cd_fast.enet_coordinate_descent(

for alpha = 1, score fpr train and test set is (0.008026724674587035, 0.031773922075666976)
Duality gap: 0.10429472209436998, tolerance: 3.393329600000001e-05
model = cd_fast.enet_coordinate_descent(
for alpha = 10, score fpr train and test set is (0.008072799291230599, 0.030779860028718187)
Duality gap: 0.10499973539172826, tolerance: 3.393329600000001e-05
model = cd_fast.enet_coordinate_descent(
for alpha = 100, score fpr train and test set is (0.008177991488490554, 0.029939663595976464)

Ques.5
Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

#Logistic Regression

file = pd.read_csv("ecoli.csv")
df = file.values
X = df[:,0:7]
y = df[:,7]

clf = LogisticRegression()
clf = clf.fit(X,y)

X_train, X_test, y_train, y_test = train_test_split(X, y)

clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy Score',accuracy*100)
precision = precision_score(y_test, y_pred)
print('Precision Score',precision*100)
recall = recall_score(y_test, y_pred, average = 'macro')
print('Recall Score',recall*100)
conf_mat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print('Confusion matrix:\n', conf_mat)
```

Output:
Accuracy Score 79.76190476190477
Precision Score 51.96825396825396
Recall Score 45.65870558749816
Confusion matrix:

```
[[29  0  0  0  0  0  1]
 [ 1 17  0  0  0  0  1]
 [ 0  1  0  0  0  0  0]
 [ 1  6  0  3  0  0  0]
 [ 1  0  0  0  1  0  3]
 [ 1  0  0  0  1  0  0]
 [ 3  1  0  0  0  0 13]]
```

Ques.6
Code:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Class for 2-class logistic regression
class Model(nn.Module):
   def _init_(self, n_input_features):
       super(Model, self)._init_()
```

```python
        self.linear = nn.Linear(n_input_features, 1)

    def forward(self, x):
        y_pred = torch.sigmoid(self.linear(x))
        return y_pred

def train_loop(X_train, y_train, model, criterion, optimizer, num_epochs):
    for epoch in range(num_epochs):
        # Forward pass and loss
        y_pred = model(X_train)
        loss = criterion(y_pred, y_train)
        # Backward pass and update
        loss.backward()
        optimizer.step()
        # zero grad before new step
        optimizer.zero_grad()
        if (epoch+1) % 100 == 0:
            print(f'epoch: {epoch+1}, loss = {loss.item():.4f}'.format())

def get_accuracy(X_test, y_test, model):
    with torch.no_grad():
        y_predicted = model(X_test)
        _,y_predicted_cls = torch.max(y_predicted.data, 1)
        acc = y_predicted_cls.eq(y_test).sum() / float(y_test.shape[0])
        print(" Accuracy")
        print(f'accuracy: {acc.item():.4f}')

def normalize(X,y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 
0.2, random_state = 3)
    # Scale the input features
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
    # Get the torch tensors
    X_train = torch.from_numpy(X_train.astype(np.float32))
    X_test = torch.from_numpy(X_test.astype(np.float32))
    y_train = torch.from_numpy(y_train.astype(np.float32)).long()
    y_test = torch.from_numpy(y_test.astype(np.float32)).long()
    return X_train, X_test, y_train, y_test
```

```python
def logistic_regression(X, y, num_epochs, learning_rate):
   n_samples, n_features = X.shape
   X_train, X_test, y_train, y_test = normalize(X,y)
   y_train = y_train.view(y_train.shape[0], 1).type(torch.FloatTensor)
   y_test = y_test.view(y_test.shape[0], 1).type(torch.FloatTensor)
   # 1) Create the model
   model = Model(n_features)
   # 2) Loss and optimizer
   criterion = nn.BCELoss()
   optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
   # 3) Training loop
   train_loop(X_train, y_train, model, criterion, optimizer, num_epochs)
   # 4) Evaluation
   correct,total = 0,0
   confusion_matrix = np.zeros((2,2))

   with torch.no_grad():
       y_predicted = model(X_test)
       y_predicted_cls = y_predicted.round()
       acc = y_predicted_cls.eq(y_test).sum() / float(y_test.shape[0])
       for i in range(y_predicted_cls.shape[0]):
           confusion_matrix[1 -
int(y_test[i].item())][1-int(y_predicted_cls[i].item())]+=1
       print("Accuracy")
       print(f'accuracy: {acc.item():.4f}')

   # Cost Matrix for confusion matrix
   cost_matrix = np.array([[0,5],[1,0]])
   print("Confusion Matrix")
   print(confusion_matrix)
   print("Cost Matrix")
   print(cost_matrix)
   print("Cost")
   # Get the cost form the confusion matrix
   cost = np.sum(confusion_matrix*cost_matrix)
   print(f'cost: {cost.item():.4f}')

# Load breast cancer dataset
bc = datasets.load_breast_cancer()
```

```
print("Breast cancer dataset")
logistic_regression(bc.data, bc.target, num_epochs = 1000, learning_rate =
0.01)
```

Output:
Breast cancer dataset
epoch: 100, loss = 0.2597
epoch: 200, loss = 0.1935
epoch: 300, loss = 0.1634
epoch: 500, loss = 0.1333
epoch: 600, loss = 0.1245
epoch: 700, loss = 0.1177
epoch: 800, loss = 0.1123
epoch: 900, loss = 0.1078
epoch: 1000, loss = 0.1041

Accuracy
accuracy: 0.9649

Confusion Matrix
[[73. 1.]
[3. 37.]]

Cost Matrix
[[0 5]
[1 0]]

Cost
cost: 8.0000
