Q1: Feed Forward and Backpropagation

1) a

$$f_1(a) = \frac{1}{1+e^{-a}} \qquad f_1'(a) = \frac{e^a}{(e^a+1)^2}$$

$$f_2(a_i) = \frac{e^{a_i}}{\sum_{j=1}^{N} e^{a_j}}$$

$$f_2'(a_i) = \frac{\partial f_2(a_i)}{\partial a_j} = \begin{cases} f_2(a_i) \times (1 - f_2(a_i)) = \frac{e^{a_i}}{\sum_{j=1}^{N} e^{a_j}} \times \left(1 - \frac{e^{a_i}}{\sum_{j=1}^{N} e^{a_j}}\right) & \text{if } i = j \\ -f_2(a_i) \times f_2(a_j) = -\frac{e^{a_i}}{\sum_{j=1}^{N} e^{a_j}} \times \frac{e^{a_j}}{\sum_{j=1}^{N} e^{a_j}} & \text{if } i \neq j \end{cases}$$

1) b

$$L = -t^T \cdot \log(X_2)$$

$$\frac{\partial L}{\partial X_{2i}} = -t_i^T \cdot \frac{1}{X_{2i}}$$

1) c

$$a_1 = W_1 X_0 + b_1 = \begin{bmatrix} 1 & 2 & -3 & 0 & 1 & -3 \\ 3 & 1 & 2 & 1 & 0 & 2 \\ 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 0 & 2 & 1 & -2 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 8 \\ 2 \end{bmatrix}$$

$$X_1 = f_1(a_1) = \begin{bmatrix} \frac{1}{1+e^{-2}} \\ \frac{1}{1+e^{-7}} \\ \frac{1}{1+e^{-8}} \\ \frac{1}{1+8^{-2}} \end{bmatrix} = \begin{bmatrix} 0.8808 \\ 0.9991 \\ 0.9997 \\ 0.8808 \end{bmatrix}$$

$\because \delta_1 = \delta_2 W_2 f_1'(a_1)$

$\therefore$ We need to calculate δ_2 first.

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial W_2} = \frac{\partial L}{\partial X_2} \cdot \frac{\partial X_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial W_2} = \delta_2 \cdot X_1, \text{ where } \delta_2 = \frac{\partial L}{\partial X_2} \cdot \frac{\partial X_2}{\partial a_2}$$

$$a_2 = W_2 \cdot X_1 + b_2 = \begin{bmatrix} 1 & 2 & -2 & 1 \\ 1 & -1 & 1 & 2 \\ 3 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0.8808 \\ 0.9991 \\ 0.9997 \\ 0.8808 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.7604 \\ 3.6430 \\ 4.5226 \end{bmatrix}$$

$$X_2 = f_2(a_2) = \begin{bmatrix} \frac{e^{2.7604}}{e^{2.7604}+e^{3.6430}+e^{4.5256}} \\ \frac{e^{3.6430}}{e^{2.7604}+e^{3.6430}+e^{4.5256}} \\ \frac{e^{4.5256}}{e^{2.7604}+e^{3.6430}+e^{4.5256}} \end{bmatrix} = \begin{bmatrix} 0.1082 \\ 0.2615 \\ 0.6303 \end{bmatrix}$$

$$\frac{\partial X_2}{\partial a_2} = f_2'(a_2) = \begin{bmatrix} f_2(a_2^1)\times(1-f_2(a_2^1)) & -f_2(a_2^1)\times f_2(a_2^2) & -f_2(a_2^1)\times f_2(a_2^3) \\ -f_2(a_2^2)\times f_2(a_2^1) & f_2(a_2^2)\times(1-f_2(a_2^2)) & -f_2(a_2^2)\times f_2(a_2^3) \\ -f_2(a_2^3)\times f_2(a_2^1) & -f_2(a_2^3)\times f_2(a_2^2) & f_2(a_2^3)\times(1-f_2(a_2^3)) \end{bmatrix}$$

$$= \begin{bmatrix} 0.1082\times(1-0.1082) & -0.1082\times 0.2615 & -0.1082\times 0.6303 \\ -0.2615\times 0.1082 & 0.2615\times(1-0.2615) & -0.2615\times 0.6303 \\ -0.6303\times 0.1082 & -0.6303\times 0.2615 & 0.6303\times(1-0.6303) \end{bmatrix}$$

$$= \begin{bmatrix} 0.0965 & -0.0283 & -0.0682 \\ -0.0283 & 0.1931 & -0.1648 \\ -0.0682 & -0.1648 & 0.2330 \end{bmatrix}$$

$$\frac{\partial L}{\partial X_2} = \begin{bmatrix} -0\times\frac{1}{0.1082} \\ -1\times\frac{1}{0.2615} \\ -0\times\frac{1}{0.6303} \end{bmatrix} = \begin{bmatrix} 0 \\ -3.8233 \\ 0 \end{bmatrix}$$

$$\therefore \delta_2 = \begin{bmatrix} 0.0965 & -0.0283 & -0.0682 \\ -0.0283 & 0.1931 & -0.1648 \\ -0.0682 & -0.1648 & 0.2330 \end{bmatrix} \begin{bmatrix} 0 \\ -3.8233 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.1082 \\ -0.7385 \\ 0.6303 \end{bmatrix}$$

$$\delta_1 = \delta_2 W_2 f_1'(a_1) = \underbrace{\begin{bmatrix} 1 & 2 & -2 & 1 \\ 1 & -1 & 1 & 2 \\ 3 & 1 & -1 & 1 \end{bmatrix}^T}_{4\times 3} \underbrace{\begin{bmatrix} 0.1082 \\ -0.7385 \\ 0.6303 \end{bmatrix}}_{3\times 1} \circ \underbrace{\begin{bmatrix} \frac{e^2}{(e^2+1)^2} \\ \frac{e^1}{(e^1+1)^2} \\ \frac{e^8}{(e^8+1)^2} \\ \frac{e^2}{(e^2+1)^2} \end{bmatrix}}_{4\times 1}$$

$$= \begin{bmatrix} 0.4324 \\ 0.0014 \\ -0.0005 \\ -0.0775 \end{bmatrix}$$

1) d. a_2, x_2, δ_2 are calculated above.

1) e. Because $x_2 = \begin{bmatrix} 0.1082 \\ 0.2615 \\ 0.6303 \end{bmatrix}$ ∴ x_0 should be predicted as class 3.

1) f. $L = -t^T \log(x_2) = -[0 \ 1 \ 0] \begin{bmatrix} 0.1082 \\ 0.2615 \\ 0.6303 \end{bmatrix} = 1.3412$

2) a

$$W_2^{new} := W_2 - \eta \cdot \frac{\partial L}{\partial W_2} = W_2 - \eta \cdot \delta_2 \cdot x_1^T$$

$$= \begin{bmatrix} 1 & 2 & -2 & 1 \\ 1 & -1 & 1 & 2 \\ 3 & 1 & -1 & 1 \end{bmatrix} - 0.5 \times \begin{bmatrix} 0.1082 \\ -0.7385 \\ 0.6303 \end{bmatrix} \cdot \begin{bmatrix} 0.8808 & 0.9991 & 0.999 & 0.8808 \end{bmatrix}$$

$$= \begin{bmatrix} 0.9523 & 1.9460 & -2.0541 & 0.9523 \\ 1.3252 & -0.6311 & 1.3691 & 2.3252 \\ 2.7724 & 0.6851 & -1.3150 & 0.7724 \end{bmatrix}$$

2) b

$$b_2^{new} := b_2 - \eta \frac{\partial L}{\partial b_2} = b_2 - \eta \cdot \delta_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - 0.5 \begin{bmatrix} 0.1082 \\ -0.7385 \\ 0.6303 \end{bmatrix} = \begin{bmatrix} 0.9459 \\ 1.3692 \\ 0.6849 \end{bmatrix}$$

2) c

$$W_1^{new} := W_1 - \eta \frac{\delta L}{\delta W_1} = W_1 - \eta \delta_1 \cdot X_0^T$$

$$= \begin{bmatrix} 1 & 2 & -3 & 0 & 1 & -3 \\ 3 & 1 & 2 & 1 & 0 & 2 \\ 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 0 & 2 & 1 & -2 & 2 \end{bmatrix} - 0.5 \begin{bmatrix} 0.1323 \\ 0.00016 \\ -0.00025 \\ -0.00775 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.9338 & 1.9338 & -3.0000 & 0 & 0.9338 & -3.0662 \\ 2.9993 & 0.9993 & 2.0000 & 1.0000 & -0.0007 & 1.9993 \\ 2.0003 & 2.0003 & 2.0000 & 2.0000 & 2.0003 & 1.0003 \\ 1.0388 & 0.0388 & 2.0000 & 1.0000 & -1.9612 & 2.0388 \end{bmatrix}$$

2) d

$$b_1^{new} := b_1 - \eta \frac{\delta L}{\delta b_1} = b_1 - \eta \delta_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - 0.5 \begin{bmatrix} 0.1323 \\ 0.00016 \\ -0.00025 \\ -0.00775 \end{bmatrix} = \begin{bmatrix} 0.9338 \\ 0.9993 \\ 1.0003 \\ 1.0388 \end{bmatrix}$$

2) e

I plug new W_1, b_1, W_2, b_2 into system, and I get $X_2 = [0.0515, 0.8636, 0.0850]^T$, which can be classified as Class 2, same as ground true.

Q2: Classification Toolbox

(a)

In [8]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [22]:
```
# read data
data = pd.read_csv("winequality-red.csv",delimiter = ";")
data = np.array(data)
```

In [29]:
```
# split data
X = data[:,0:11]
Y = data[:,11]
trainX = X[0:1000,:]
testX = X[1000:,:]
trainY = np.zeros((1000))
trainY[Y[0:1000]>=6] = 1
trainY[Y[0:1000]<6] = 0
testY = np.zeros((testX.shape[0]))
testY[Y[1000:]>=6] = 1
testY[Y[1000:]<6] = 0
```

In [42]:
```
# Decision tree
from sklearn.tree import DecisionTreeClassifier
critetrion = ["gini","entropy"]
max_depth = np.arange(1,11,1)
for i in critetrion:
    for j in max_depth:
        clf = DecisionTreeClassifier(criterion=i,max_depth=j)
        clf.fit(trainX,trainY)
        train_acc = clf.score(trainX,trainY)
        test_acc = clf.score(testX,testY)
        print(f"Critetrion:{i}, max_depth:{j},train accuracy:{train_acc},test accuracy:{test_acc}")
```

```
Critetrion:gini, max_depth:1,train accuracy:0.701,test accuracy:0.7011686143572621
Critetrion:gini, max_depth:2,train accuracy:0.701,test accuracy:0.7011686143572621
Critetrion:gini, max_depth:3,train accuracy:0.738,test accuracy:0.7178631051752922
Critetrion:gini, max_depth:4,train accuracy:0.756,test accuracy:0.7278797996661102
Critetrion:gini, max_depth:5,train accuracy:0.794,test accuracy:0.7295492487479132
Critetrion:gini, max_depth:6,train accuracy:0.828,test accuracy:0.7111853088480802
Critetrion:gini, max_depth:7,train accuracy:0.864,test accuracy:0.7161936560934892
Critetrion:gini, max_depth:8,train accuracy:0.891,test accuracy:0.6878130217028381
Critetrion:gini, max_depth:9,train accuracy:0.919,test accuracy:0.669449081803005
Critetrion:gini, max_depth:10,train accuracy:0.933,test accuracy:0.6527545909849749
Critetrion:entropy, max_depth:1,train accuracy:0.701,test accuracy:0.7011686143572621
Critetrion:entropy, max_depth:2,train accuracy:0.701,test accuracy:0.7011686143572621
Critetrion:entropy, max_depth:3,train accuracy:0.734,test accuracy:0.7178631051752922
Critetrion:entropy, max_depth:4,train accuracy:0.757,test accuracy:0.7362270450751253
Critetrion:entropy, max_depth:5,train accuracy:0.777,test accuracy:0.7378964941569283
Critetrion:entropy, max_depth:6,train accuracy:0.811,test accuracy:0.7078464106844741
Critetrion:entropy, max_depth:7,train accuracy:0.845,test accuracy:0.7078464106844741
Critetrion:entropy, max_depth:8,train accuracy:0.869,test accuracy:0.6978297161936561
Critetrion:entropy, max_depth:9,train accuracy:0.895,test accuracy:0.6944908180300501
Critetrion:entropy, max_depth:10,train accuracy:0.905,test accuracy:0.6944908180300501
```

from the numbers above, {"gini", max_depth = 5},{"entropy", max_depth = 4}, {"entropy", max_depth = 5} these three groups of parameters can work best.

In [49]:
```
# K-nearest Neighbor
from sklearn.neighbors import KNeighborsClassifier
n_neighbors = np.arange(1,36,1)
for i in n_neighbors:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(trainX,trainY)
    train_acc = neigh.score(trainX,trainY)
    test_acc = neigh.score(testX,testY)
    print(f"n_neighbors:{i}, train accuracy:{train_acc},test accuracy:{test_acc}")
```

```
n_neighbors:1, train accuracy:1.0,test accuracy:0.5876460767946577
n_neighbors:2, train accuracy:0.866,test accuracy:0.5609348914858097
n_neighbors:3, train accuracy:0.86,test accuracy:0.5876460767946577
n_neighbors:4, train accuracy:0.805,test accuracy:0.5859766277128547
n_neighbors:5, train accuracy:0.789,test accuracy:0.5792988313856428
n_neighbors:6, train accuracy:0.773,test accuracy:0.5792988313856428
n_neighbors:7, train accuracy:0.778,test accuracy:0.5859766277128547
n_neighbors:8, train accuracy:0.755,test accuracy:0.5893155258764607
n_neighbors:9, train accuracy:0.746,test accuracy:0.6093489148580968
n_neighbors:10, train accuracy:0.738,test accuracy:0.5909849749582637
n_neighbors:11, train accuracy:0.735,test accuracy:0.6110183639398998
n_neighbors:12, train accuracy:0.741,test accuracy:0.5959933222036727
n_neighbors:13, train accuracy:0.734,test accuracy:0.6227045075125208
n_neighbors:14, train accuracy:0.732,test accuracy:0.6160267111853088
n_neighbors:15, train accuracy:0.73,test accuracy:0.6193656093489148
n_neighbors:16, train accuracy:0.703,test accuracy:0.6076794657762938
n_neighbors:17, train accuracy:0.714,test accuracy:0.6227045075125208
n_neighbors:18, train accuracy:0.707,test accuracy:0.6143572621035058
n_neighbors:19, train accuracy:0.713,test accuracy:0.6327212020033389
n_neighbors:20, train accuracy:0.705,test accuracy:0.6260434056761269
n_neighbors:21, train accuracy:0.709,test accuracy:0.6327212020033389
n_neighbors:22, train accuracy:0.702,test accuracy:0.6126878130217028
n_neighbors:23, train accuracy:0.708,test accuracy:0.6243739565943238
n_neighbors:24, train accuracy:0.716,test accuracy:0.6126878130217028
n_neighbors:25, train accuracy:0.718,test accuracy:0.6260434056761269
n_neighbors:26, train accuracy:0.711,test accuracy:0.6176961602671118
n_neighbors:27, train accuracy:0.72,test accuracy:0.6193656093489148
n_neighbors:28, train accuracy:0.717,test accuracy:0.6210350584307178
n_neighbors:29, train accuracy:0.719,test accuracy:0.6193656093489148
n_neighbors:30, train accuracy:0.709,test accuracy:0.6210350584307178
n_neighbors:31, train accuracy:0.713,test accuracy:0.6260434056761269
n_neighbors:32, train accuracy:0.705,test accuracy:0.6227045075125208
n_neighbors:33, train accuracy:0.707,test accuracy:0.6293823038397329
n_neighbors:34, train accuracy:0.699,test accuracy:0.6243739565943238
n_neighbors:35, train accuracy:0.706,test accuracy:0.6243739565943238
```

I will choose K = {19,21,33}.

In [79]:
```python
#Logistic Regression
from sklearn.linear_model import LogisticRegression
one_train = np.ones((trainX.shape[0]))
one_test = np.ones((testX.shape[0]))
penalty = ["l1","l2","elasticnet"]
for i in penalty:
    if i == "elasticnet":
        clf = LogisticRegression(penalty=i,solver = 'saga',l1_ratio = 0.5,max_iter = 10000)
    elif i == "l1":
        clf = LogisticRegression(penalty=i,solver = 'liblinear',max_iter = 10000)
    else:
        clf = LogisticRegression(penalty=i,solver = 'liblinear',max_iter = 10000)
    clf.fit(trainX,trainY)
    train_acc = clf.score(trainX,trainY)
    test_acc = clf.score(testX,testY)
    print(f"penalty:{i}, train accuracy:{train_acc},test accuracy:{test_acc}")
```

```
penalty:l1, train accuracy:0.735,test accuracy:0.7629382303839732
penalty:l2, train accuracy:0.728,test accuracy:0.7529215358931552
penalty:elasticnet, train accuracy:0.729,test accuracy:0.7545909849749582
```

In [61]:
```python
# SVM
from sklearn.svm import SVC
kernel = ["linear","rbf"]
C = [0.01,10,1000]
for i in range(2):
    for j in range(3):
        model = SVC(kernel = kernel[i],C = C[j])
        model.fit(trainX,trainY)
        train_acc = model.score(trainX,trainY)
        test_acc = model.score(testX,testY)
        print(f"kernel = {kernel[i]}, C = {C[j]}, train accuracy:{train_acc},test accuracy:{test_acc}")
model = SVC(kernel = "poly")
model.fit(trainX,trainY)
train_acc = model.score(trainX,trainY)
test_acc = model.score(testX,testY)
print(f"kernel = poly,train accuracy:{train_acc},test accuracy:{test_acc}")
```

```
kernel = linear, C = 0.01, train accuracy:0.72,test accuracy:0.7245409015025042
kernel = linear, C = 10, train accuracy:0.733,test accuracy:0.7662771285475793
kernel = linear, C = 1000, train accuracy:0.731,test accuracy:0.7462437395659433
kernel = rbf, C = 0.01, train accuracy:0.623,test accuracy:0.6060100166944908
kernel = rbf, C = 10, train accuracy:0.715,test accuracy:0.7128547579298832
kernel = rbf, C = 1000, train accuracy:0.775,test accuracy:0.7495826377295493
kernel = poly,train accuracy:0.613,test accuracy:0.6193656093489148
```

In [86]:
```
# Neural Networks
from sklearn.neural_network import MLPClassifier
activation = ["logistic","tanh","relu"]
for i in activation:
    clf = MLPClassifier(hidden_layer_sizes=(10,20,10), activation=i,max_iter=5000,random_state = 20)
    clf.fit(trainX,trainY)
    train_acc = clf.score(trainX,trainY)
    test_acc = clf.score(testX,testY)
    print(f"hidden_layer_sizes=(10,20,10),activation = {i}, train accuracy:{train_acc},test accuracy:{test_acc}")

for i in activation:
    clf = MLPClassifier(hidden_layer_sizes=(8,25,8), activation=i,max_iter=5000,random_state = 20)
    clf.fit(trainX,trainY)
    train_acc = clf.score(trainX,trainY)
    test_acc = clf.score(testX,testY)
    print(f"hidden_layer_sizes=(5,20,5),activation = {i}, train accuracy:{train_acc},test accuracy:{test_acc}")

for i in activation:
    clf = MLPClassifier(hidden_layer_sizes=(10,50,10), activation=i,max_iter=5000,random_state = 20)
    clf.fit(trainX,trainY)
    train_acc = clf.score(trainX,trainY)
    test_acc = clf.score(testX,testY)
    print(f"hidden_layer_sizes=(10,50,10),activation = {i}, train accuracy:{train_acc},test accuracy:{test_acc}")
```

```
hidden_layer_sizes=(10,20,10),activation = logistic, train accuracy:0.747,test accuracy:0.7545909849749582
hidden_layer_sizes=(10,20,10),activation = tanh, train accuracy:0.731,test accuracy:0.7479131886477463
hidden_layer_sizes=(10,20,10),activation = relu, train accuracy:0.744,test accuracy:0.7462437395659433
hidden_layer_sizes=(5,20,5),activation = logistic, train accuracy:0.752,test accuracy:0.7595993322203672
hidden_layer_sizes=(5,20,5),activation = tanh, train accuracy:0.741,test accuracy:0.7429048414023373
hidden_layer_sizes=(5,20,5),activation = relu, train accuracy:0.72,test accuracy:0.7228714524207012
hidden_layer_sizes=(10,50,10),activation = logistic, train accuracy:0.755,test accuracy:0.7662771285475793
hidden_layer_sizes=(10,50,10),activation = tanh, train accuracy:0.731,test accuracy:0.7712854757929883
hidden_layer_sizes=(10,50,10),activation = relu, train accuracy:0.505,test accuracy:0.3989983305509182
```

The best parameterss combination for each type of model:

Decision tree: {Critetrion:entropy, max_depth:5,train accuracy:0.777,test accuracy:0.737896494156928 3}

K-nearest Neighbor: {n_neighbors:21, train accuracy:0.709,test accuracy:0.632721202003338 9}

Logistic Regression: {penalty:l1, train accuracy:0.735,test accuracy:0.762938230383973 2}

SVM: {kernel = linear, C = 10, train accuracy:0.733,test accuracy:0.7662771285475793}

Neural Networks: {hidden_layer_sizes=(10,50,10),activation = tanh, train accuracy:0.731,test accuracy:0.771285475792988 3}

(b)

true positive (TP): A test result that correctly indicates the presence of a condition or characteristic.

true negative (TN): A test result that correctly indicates the absence of a condition or characteristic.

false positive (FP): A test result which wrongly indicates that a particular condition or attribute is present.

false negative (FN): A test result which wrongly indicates that a particular condition or attribute is absent.

Confusion matrix is made up of these four terms.

Precision: the proportion of positive identifications was actually correct. (TP/(TP+FP))

Recall: the proportion of actual positives was identified correctly. (TP/(TP+FN))

F1: F1 score is a measure of a model's accuracy on a dataset. F1_score is the harmonic mean of the precision and recall. (2 *precision* recall/precision + recall)

In [88]:
```
# Decision tree
# criterion="entropy",max_depth=5
from sklearn import metrics
clf = DecisionTreeClassifier(criterion="entropy",max_depth=5)
clf.fit(trainX,trainY)
pred = clf.predict(testX)
precision = metrics.precision_score(testY,pred)
recall = metrics.recall_score(testY,pred)
f1_score = metrics.f1_score(testY,pred)
confusion_matrix = metrics.confusion_matrix(testY,pred)
print("precision: ",precision)
print("recall: ",recall)
print("f1_score: ",f1_score)
print("confusion_matrix: ",confusion_matrix)
```

```
precision:  0.7595907928388747
recall:  0.825
f1_score:  0.7909454061251664
confusion_matrix:  [[145  94]
 [ 63 297]]
```

In [89]:
```
# K-nearest Neighbor
# n_neighbors=21
clf = KNeighborsClassifier(n_neighbors=21)
clf.fit(trainX,trainY)
pred = clf.predict(testX)
precision = metrics.precision_score(testY,pred)
recall = metrics.recall_score(testY,pred)
f1_score = metrics.f1_score(testY,pred)
confusion_matrix = metrics.confusion_matrix(testY,pred)
print("precision: ",precision)
print("recall: ",recall)
print("f1_score: ",f1_score)
print("confusion_matrix: ",confusion_matrix)
```

```
precision:  0.7
recall:  0.6805555555555556
f1_score:  0.6901408450704225
confusion_matrix:  [[134 105]
 [115 245]]
```

In [90]:
```
# Logistic regression
# penalty="l1"
clf = LogisticRegression(penalty="l1",solver = 'liblinear',max_iter = 10000)
clf.fit(trainX,trainY)
pred = clf.predict(testX)
precision = metrics.precision_score(testY,pred)
recall = metrics.recall_score(testY,pred)
f1_score = metrics.f1_score(testY,pred)
confusion_matrix = metrics.confusion_matrix(testY,pred)
print("precision: ",precision)
print("recall: ",recall)
print("f1_score: ",f1_score)
print("confusion_matrix: ",confusion_matrix)
```

```
precision:  0.7780612244897959
recall:  0.8472222222222222
f1_score:  0.8111702127659575
confusion_matrix:  [[152  87]
 [ 55 305]]
```

In [91]:
```
# SVM
# kernel = "linear",C = 10
clf = SVC(kernel = "linear",C = 10)
clf.fit(trainX,trainY)
pred = clf.predict(testX)
precision = metrics.precision_score(testY,pred)
recall = metrics.recall_score(testY,pred)
f1_score = metrics.f1_score(testY,pred)
confusion_matrix = metrics.confusion_matrix(testY,pred)
print("precision: ",precision)
print("recall: ",recall)
print("f1_score: ",f1_score)
print("confusion_matrix: ",confusion_matrix)
```

```
precision:  0.8072625698324022
recall:  0.8027777777777778
f1_score:  0.8050139275766017
confusion_matrix:  [[170  69]
 [ 71 289]]
```

In [92]:
```
# Neural Networks
# hidden_layer_sizes=(10,50,10), activation="tanh"
clf = MLPClassifier(hidden_layer_sizes=(10,50,10), activation="tanh",max_iter=5000,random_state = 20)
clf.fit(trainX,trainY)
pred = clf.predict(testX)
precision = metrics.precision_score(testY,pred)
recall = metrics.recall_score(testY,pred)
f1_score = metrics.f1_score(testY,pred)
confusion_matrix = metrics.confusion_matrix(testY,pred)
print("precision: ",precision)
print("recall: ",recall)
print("f1_score: ",f1_score)
print("confusion_matrix: ",confusion_matrix)
```

```
precision:  0.7989276139410187
recall:  0.8277777777777777
f1_score:  0.8130968622100956
confusion_matrix:  [[164  75]
 [ 62 298]]
```

Case 1: Because client doesn't want to miss any good wine, that means we should max the proportion of True Positive in all actual Positive. Therefore, I choose **Logistic regression** which has the highest Recall score.

Case 2: The client doesn't want to lose good wine but also doesn't like wines classified as good wine but actully bad wines(FP). Therefore, I think F1_score can evaluate this case because F1_score combines Recall and Precision. I will choose **Neurual Networks** which has the highest F1_score.