

In [55]:

```
import pandas as pd
from sklearn.metrics import zero_one_loss
from liblinearutil import *
```

In [8]:

```
trainUrl = "https://www.csie.ntu.edu.tw/~htlin/course/ml20fall/hw4/hw4_train.dat"
testUrl = "https://www.csie.ntu.edu.tw/~htlin/course/ml20fall/hw4/hw4_test.dat"
```

In [32]:

```
colName = ["x1", "x2", "x3", "x4", "x5", "x6", "y"]
trainDf = pd.read_csv(trainUrl, sep=" ", names=colName)
trainDf.insert(loc=0, column="x0", value=1)
trainDf.head()
```

Out[32]:

	x0	x1	x2	x3	x4	x5	x6	y
0	1	0.991	0.8	1.0	0.9	0.941	1	1
1	1	0.950	1.0	0.8	1.0	0.953	1	1
2	1	0.866	0.6	0.8	0.5	0.816	0	-1
3	1	0.916	0.8	0.8	0.9	0.843	0	1
4	1	0.916	0.8	0.9	0.8	0.903	1	1

In [33]:

```
testDf = pd.read_csv(testUrl, sep=" ", names=colName)
testDf.insert(loc=0, column="x0", value=1)
testDf.head()
```

Out[33]:

	x0	x1	x2	x3	x4	x5	x6	y
0	1	0.825	0.4	0.8	0.4	0.766	0	-1
1	1	0.766	0.2	0.4	0.4	0.794	0	-1
2	1	0.941	1.0	0.8	0.8	0.935	1	1
3	1	0.891	0.6	0.7	0.6	0.875	1	1
4	1	0.850	0.2	0.4	0.3	0.806	0	-1

In [30]:

```
def polynomial_transformation(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy(deep=True)
    for i in range(1, 7):
        df[f"x{i}^2"] = df[f"x{i}"].pow(2)

    for i in range(1, 7):
        for j in range(1, 7):
            if i <= j:
                continue
            df[f"x{i}*x{j}"] = df[f"x{i}"] * df[f"x{j}"]

    return df
```

In [36]:

```
trainPoly = polynomial_transformation(trainDf)
testPoly = polynomial_transformation(testDf)
```

In [42]:

```
xTrain, yTrain = trainPoly.drop(["y"], axis=1).to_numpy(), trainPoly["y"].to_numpy()
xTest, yTest = testPoly.drop(["y"], axis=1).to_numpy(), testPoly["y"].to_numpy()
```

In [75]:

```
lambdaList = [-4, -2, 0, 2, 4]
prob = problem(yTrain, xTrain)

result = []
for i in lambdaList:
    lam = pow(10, i)
    param = parameter(f"-s 0 -c {1/lam} -e 0.000001")
    model = train(prob, param)
    p_labels, p_acc, p_vals = predict(yTest, xTest, model)
    result.append([p_acc[0], i])

answer = sorted(result, key=lambda x: (x[0], x[1]), reverse=True)[0]
print(f"The optimal lambda to minimize Eout is {answer[1]})
```

Accuracy = 86% (258/300) (classification)
 Accuracy = 87.6667% (263/300) (classification)
 Accuracy = 82% (246/300) (classification)
 Accuracy = 74.3333% (223/300) (classification)
 Accuracy = 51.6667% (155/300) (classification)
 The optimal lambda to minimize Eout is -2

In [78]:

```

lambdaList = [-4, -2, 0, 2, 4]
prob = problem(yTrain, xTrain)

result = []
for i in lambdaList:
    lam = pow(10, i)
    param = parameter(f"-s 0 -c {1/lam} -e 0.000001")
    model = train(prob, param)
    p_labels, p_acc, p_vals = predict(yTrain, xTrain, model)
    result.append([p_acc[0], i])

answer = sorted(result, key=lambda x: (x[0], x[1]), reverse=True)[0]
print(f"The optimal lambda to minimize Eout is {answer[1]}")

```

Accuracy = 91% (182/200) (classification)
 Accuracy = 91% (182/200) (classification)
 Accuracy = 87.5% (175/200) (classification)
 Accuracy = 80.5% (161/200) (classification)
 Accuracy = 46.5% (93/200) (classification)
 The optimal lambda to minimize Eout is -2

In [89]:

```

xTrain, yTrain = trainPoly.drop(["y"], axis=1).to_numpy()[:120], trainPoly["y"].to_numpy()[:120]
xVal, yVal = trainPoly.drop(["y"], axis=1).to_numpy()[120:], trainPoly["y"].to_numpy()[120:]

lambdaList = [-4, -2, 0, 2, 4]
prob = problem(yTrain, xTrain)

result = []
for i in lambdaList:
    lam = pow(10, i)
    param = parameter(f"-s 0 -c {1/pow(10, bestLambda)} -e 0.000001")
    model = train(prob, param)
    p_labels, p_acc, p_vals = predict(yVal, xVal, model)
    result.append([p_acc[0], i])

bestLambda = sorted(result, key=lambda x: (x[0], x[1]), reverse=True)[0][1]
bestParam = parameter(f"-s 0 -c {1/pow(10, bestLambda)} -e 0.000001")
bestModel = train(prob, bestParam)
p_labels, p_acc, p_vals = predict(yTest, xTest, bestModel)
print(f"The optimal Eout is {1 - p_acc[0] / 100}")

```

Accuracy = 81.25% (65/80) (classification)
 Accuracy = 86.25% (69/80) (classification)
 Accuracy = 80% (64/80) (classification)
 Accuracy = 75% (60/80) (classification)
 Accuracy = 42.5% (34/80) (classification)
 Accuracy = 86% (258/300) (classification)
 The optimal Eout is 0.14

In [90]:

```
xTrain, yTrain = trainPoly.drop(["y"], axis=1).to_numpy(), trainPoly["y"].to_numpy()

prob = problem(yTrain, xTrain)
bestLambda = sorted(result, key=lambda x: (x[0], x[1]), reverse=True)[0][1]
bestParam = parameter(f"-s 0 -c {1/pow(10, bestLambda)} -e 0.000001")
bestModel = train(prob, bestParam)
p_labels, p_acc, p_vals = predict(yTest, xTest, bestModel)
print(f"The optimal Eout is {1 - p_acc[0] / 100}")
```

Accuracy = 87.6667% (263/300) (classification)

The optimal Eout is 0.1233333333333333

In [97]:

```
xTrain, yTrain = trainPoly.drop(["y"], axis=1).to_numpy(), trainPoly["y"].to_numpy()

result = []
lambdaList = [-4, -2, 0, 2, 4]
for i in lambdaList:
    tempt = []
    lam = pow(10, i)
    param = parameter(f"-s 0 -c {1/lam} -e 0.000001")
    for fold in range(1, 6):
        xFold, yFold = trainPoly.drop(["y"], axis=1).to_numpy()[: fold * 40], trainPoly["y"].to_numpy()[:fold * 40]
        xVal, yVal = trainPoly.drop(["y"], axis=1).to_numpy()[(fold - 1) * 40: fold * 40], trainPoly["y"].to_numpy()[(fold - 1) * 40: fold * 40]
        prob = problem(yFold, xFold)
        model = train(prob, param)
        p_labels, p_acc, p_vals = predict(yVal, xVal, model)
        tempt.append(p_acc[0])

    result.append([sum(tempt) / len(tempt), i])

bestLambda = sorted(result, key=lambda x: (x[0], x[1]), reverse=True)[0][1]
bestParam = parameter(f"-s 0 -c {1/pow(10, bestLambda)} -e 0.000001")
bestModel = train(prob, bestParam)
p_labels, p_acc, p_vals = predict(yTest, xTest, bestModel)
print(f"The optimal Eout is {1 - p_acc[0] / 100}")
```

Accuracy = 97.5% (39/40) (classification)
 Accuracy = 90% (36/40) (classification)
 Accuracy = 97.5% (39/40) (classification)
 Accuracy = 87.5% (35/40) (classification)
 Accuracy = 95% (38/40) (classification)
 Accuracy = 92.5% (37/40) (classification)
 Accuracy = 87.5% (35/40) (classification)
 Accuracy = 92.5% (37/40) (classification)
 Accuracy = 87.5% (35/40) (classification)
 Accuracy = 95% (38/40) (classification)
 Accuracy = 85% (34/40) (classification)
 Accuracy = 92.5% (37/40) (classification)
 Accuracy = 85% (34/40) (classification)
 Accuracy = 82.5% (33/40) (classification)
 Accuracy = 90% (36/40) (classification)
 Accuracy = 77.5% (31/40) (classification)
 Accuracy = 87.5% (35/40) (classification)
 Accuracy = 85% (34/40) (classification)
 Accuracy = 72.5% (29/40) (classification)
 Accuracy = 77.5% (31/40) (classification)
 Accuracy = 65% (26/40) (classification)
 Accuracy = 57.5% (23/40) (classification)
 Accuracy = 47.5% (19/40) (classification)
 Accuracy = 40% (16/40) (classification)
 Accuracy = 45% (18/40) (classification)
 Accuracy = 86% (258/300) (classification)
 The optimal Eout is 0.14