In [1]:

```
import math
import random

import numpy as np
import pandas as pd
from sklearn.metrics import log_loss, mean_squared_error, zero_one_loss
```

In [2]:

```
trainDf = pd.read_table("https://www.csie.ntu.edu.tw/~htlin/course/ml20fall/hw3/hw3_train.dat", h
eader=None, names=["x1", "x2", "x3", "x4", "x5", "x6", "x7", "x8", "x9", "x10", "y"])
trainDf.insert(loc=0, column='x0', value=1)
trainDf.head()
```

Out[2]:

	x0	x1	x2	x3	x4	x5	x6	x7	x8
0	1	2.965153	2.447427	1.958754	-0.073541	-2.088459	-1.156375	-0.361324	1.621895
1	1	-4.303194	-0.032933	2.568076	3.488505	1.726636	0.230978	1.483300	-4.560341
2	1	-0.261568	0.974854	-1.132005	-0.495228	-1.851622	-4.663177	-0.368071	-4.031803
3	1	-1.968794	2.500225	-0.230466	-0.063074	1.461600	-0.245143	0.248745	-3.694138
4	1	-1.383537	-3.535660	-2.023477	-0.592387	-0.790183	3.946650	1.643966	2.944593

In [3]:

```
testDf = pd.read_table("https://www.csie.ntu.edu.tw/~htlin/course/ml20fall/hw3/hw3_test.dat", he
ader=None, names=["x1", "x2", "x3", "x4", "x5", "x6", "x7", "x8", "x9", "x10", "y"])
testDf.insert(loc=0, column='x0', value=1)
testDf.head()
```

Out[3]:

	x0	x1	x2	x3	x4	x5	x6	x7	x8
0	1	1.406809	1.629765	0.137603	0.096895	-0.671920	0.086663	0.502709	4.436260
1	1	-4.507169	-0.944514	-1.634057	-1.075809	-0.117050	1.328398	-0.349176	-4.273740
2	1	-1.559574	-4.985006	0.946881	-0.567474	1.249903	1.728784	-1.341989	2.405128
3	1	0.657474	-0.884554	0.274763	0.561744	0.624655	1.556778	0.802202	-0.575164
4	1	-0.533280	-1.945261	1.507659	1.944149	0.000440	1.010056	1.876235	1.531695

In [4]:

```
def problem14(trainD: pd.DataFrame) -> float:
    X, y = trainDf.drop(["y"], axis=1).to_numpy(), trainDf["y"].to_numpy()
    psuedoInverseX = np.linalg.pinv(X)
    weightLIN = np.dot(psuedoInverseX, y)
    y_pred = np.dot(X, weightLIN)

    return mean_squared_error(y, y_pred)

MSE = problem14(trainDf)
print(f"Average squared error of Ein(Wlin) is {MSE:.2f}")
```

Average squared error of Ein(Wlin) is 0.61

In [5]:

```
def gradient_direction(xn: np.array, yn: float, wt: np.array) -> np.array:
    tmp = yn - np.dot(wt.T, xn)
    return 2 * np.dot(tmp, xn)


def problem15(trainDf: pd.DataFrame, eta: float, seed: int) -> int:
    run = 0
    random.seed(seed)
    rowNumber = trainDf.shape[0]
    X, y = trainDf.drop(["y"], axis=1).to_numpy(), trainDf["y"].to_numpy()

    ein = float("inf")
    weight = np.zeros(11)
    MSE = problem14(trainDf) * 1.01
    while ein > MSE:
        number = random.randint(0, rowNumber - 1)
        xn, yn = X[number], y[number]
        weight += eta * gradient_direction(xn, yn, weight)
        y_pred = np.dot(X, weight)
        ein = mean_squared_error(y, y_pred)
        run += 1

    return run

result = [problem15(trainDf, 0.001, i) for i in range(1, 1001)]
print(f"Aaverage number of iterations is {sum(result) // len(result)}")
```

Aaverage number of iterations is 1877

In [6]:

```
def sigmoid(x):
    return 1 / (1 + math.exp(-x))


def gradient_direction(xn: np.array, yn: float, wt: np.array) -> np.array:
    return sigmoid(-yn * np.dot(wt.T, xn)) * yn * xn


def problem16(trainDf: pd.DataFrame, eta: float, seed: int) -> float:
    random.seed(seed)
    rowNumber = trainDf.shape[0]
    X, y = trainDf.drop(["y"], axis=1).to_numpy(), trainDf["y"].map({1: 1, -1: 0}).to_numpy()

    weight = np.zeros(11)
    MSE = problem14(trainDf) * 1.01
    for _ in range(500):
        number = random.randint(0, rowNumber - 1)
        xn, yn = X[number], y[number]
        weight += eta * gradient_direction(xn, yn, weight)

    y_pred = list(map(sigmoid, list(np.dot(X, weight))))
    # print(y_pred, y)
    ein = log_loss(y, y_pred)
    return ein

result = [problem16(trainDf, 0.001, i) for i in range(1, 1001)]
print(f"Aaverage cross-entropy error is {sum(result) / len(result):.2f}")
```

Aaverage cross-entropy error is 0.62

In [7]:

```
def sigmoid(x):
    return 1 / (1 + math.exp(-x))


def gradient_direction(xn: np.array, yn: float, wt: np.array) -> np.array:
    return sigmoid(-yn * np.dot(wt.T, xn)) * yn * xn


def problem17(trainDf: pd.DataFrame, eta: float, seed: int) -> float:
    random.seed(seed)
    rowNumber = trainDf.shape[0]
    X, y = trainDf.drop(["y"], axis=1).to_numpy(), trainDf["y"].map({1: 1, -1: 0}).to_numpy()

    psuedoInverseX = np.linalg.pinv(X)
    weightLIN = np.dot(psuedoInverseX, y)
    weight = weightLIN
    MSE = problem14(trainDf) * 1.01
    for _ in range(500):
        number = random.randint(0, rowNumber - 1)
        xn, yn = X[number], y[number]
        weight += eta * gradient_direction(xn, yn, weight)

    y_pred = list(map(sigmoid, list(np.dot(X, weight))))
    ein = log_loss(y, y_pred)
    return ein

result = [problem17(trainDf, 0.001, i) for i in range(1, 1001)]
print(f"Aaverage cross-entropy error is {sum(result) / len(result):.2f}")
```

Aaverage cross-entropy error is 0.61

In [8]:

```
def problem18(trainDf: pd.DataFrame, testDf: pd.DataFrame) -> float:
    train_X, train_y = trainDf.drop(["y"], axis=1).to_numpy(), trainDf["y"].map({1: 1, -1: 0}).to_numpy()
    psuedoInverseX = np.linalg.pinv(train_X)
    weightLIN = np.dot(psuedoInverseX, train_y)

    y_pred = np.dot(train_X, weightLIN)
    y_pred = (y_pred >= 0.5).astype(int)
    ein = zero_one_loss(train_y, y_pred)

    test_X, test_y = testDf.drop(["y"], axis=1).to_numpy(), testDf["y"].map({1: 1, -1: 0}).to_numpy()
    y_pred = np.dot(test_X, weightLIN)
    y_pred = (y_pred >= 0.5).astype(int)
    eout = zero_one_loss(test_y, y_pred)

    return abs(ein - eout)

print(f"|Ein - Eout| is {problem18(trainDf, testDf):.2f}")
```

|Ein - Eout| is 0.32

In [9]:

```
def problem19(trainDf: pd.DataFrame, testDf: pd.DataFrame, order: int) -> float:
    trainDf, testDf = trainDf.copy(deep=True), testDf.copy(deep=True)
    for i in range(1, 11):
        for j in range(1, order + 1):
            trainDf[f"x{i}^{j}"] = trainDf[f"x{i}"].pow(j)
    for i in range(1, 11):
        for j in range(1, order + 1):
            testDf[f"x{i}^{j}"] = testDf[f"x{i}"].pow(j)

    train_X, train_y = trainDf.drop(["y"], axis=1).to_numpy(), trainDf["y"].map({1: 1, -1: 0}).to_numpy()
    psuedoInverseX = np.linalg.pinv(train_X)
    weightLIN = np.dot(psuedoInverseX, train_y)

    y_pred = np.dot(train_X, weightLIN)
    y_pred = (y_pred >= 0.5).astype(int)
    ein = zero_one_loss(train_y, y_pred)

    test_X, test_y = testDf.drop(["y"], axis=1).to_numpy(), testDf["y"].map({1: 1, -1: 0}).to_numpy()
    y_pred = np.dot(test_X, weightLIN)
    y_pred = (y_pred >= 0.5).astype(int)
    eout = zero_one_loss(test_y, y_pred)

    return abs(ein - eout)

print(f"|Ein - Eout| is {problem19(trainDf, testDf, 3):.2f}")
```

|Ein - Eout| is 0.37

In [10]:

```
def problem20(trainDf: pd.DataFrame, testDf: pd.DataFrame, order: int) -> float:
    trainDf, testDf = trainDf.copy(deep=True), testDf.copy(deep=True)
    for i in range(1, 11):
        for j in range(1, order + 1):
            trainDf[f"x{i}^{j}"] = trainDf[f"x{i}"].pow(j)
    for i in range(1, 11):
        for j in range(1, order + 1):
            testDf[f"x{i}^{j}"] = testDf[f"x{i}"].pow(j)

    train_X, train_y = trainDf.drop(["y"], axis=1).to_numpy(), trainDf["y"].map({1: 1, -1: 0}).to_numpy()
    psuedoInverseX = np.linalg.pinv(train_X)
    weightLIN = np.dot(psuedoInverseX, train_y)

    y_pred = np.dot(train_X, weightLIN)
    y_pred = (y_pred >= 0.5).astype(int)
    ein = zero_one_loss(train_y, y_pred)

    test_X, test_y = testDf.drop(["y"], axis=1).to_numpy(), testDf["y"].map({1: 1, -1: 0}).to_numpy()
    y_pred = np.dot(test_X, weightLIN)
    y_pred = (y_pred >= 0.5).astype(int)
    eout = zero_one_loss(test_y, y_pred)

    return abs(ein - eout)

print(f"|Ein - Eout| is {problem20(trainDf, testDf, 10):.2f}")
```

|Ein - Eout| is 0.45