```python
from sklearn.datasets import load_iris
import numpy as np
import stat_helper as sh
import datetime as dt


iris = load_iris()

iris_classes =np.array(iris['target_names'])
iris_data = np.array(iris['data'])
iris_feature = np.array(iris['feature_names'])
iris_target = np.array(iris['target'])

def makePredictionMatrix(W, x):
    predictionMatrix = np.array(sh.sigmoid(np.matmul(W,x))).T
    return predictionMatrix

def trainWMatrix(W, TTD, alpha):
    n_classes = len(TTD.trainingData)
    gradW_MSE = 0
    for c in range(n_classes):
        x = np.c_[TTD.trainingData[c],np.ones(len(TTD.trainingData[c])).T].T
        t = TTD.trainingTarget[c]
        g = makePredictionMatrix(W, x)

        gradW_MSE += sh.calculate_grad_W_MSE(g,t,x)

    W-=alpha*gradW_MSE

    return W

def trainUntilSatisfactory(W, TTD, alpha, itt):
    for i in range(itt):
        W = trainWMatrix(W, TTD, alpha)
    return W

def makeConfusionMatricies(W, TTD):
    n_classes = len(TTD.testData)
    confusionMatrixTestSet = np.zeros((n_classes, n_classes))
    confusionMatrixTrainingSet = np.zeros((n_classes, n_classes))
    for c in range(n_classes):
        x_test = np.c_[TTD.testData[c],np.ones(len(TTD.testData[c])).T].T
        x_train =
np.c_[TTD.trainingData[c],np.ones(len(TTD.trainingData[c])).T].T
        predictionTestSet = makePredictionMatrix(W, x_test)
        predictionTrainingSet = makePredictionMatrix(W, x_train)
        answearTestSet = TTD.testTarget[c]
        answearTrainingSet = TTD.trainingTarget[c]
        for i in range(len(TTD.testTarget[0])):
            confusionMatrixTestSet[np.argmax(answearTestSet[i])]
[np.argmax(predictionTestSet[i])] += 1
        for i in range(len(TTD.trainingTarget[0])):
            confusionMatrixTrainingSet[np.argmax(answearTrainingSet[i])]
[np.argmax(predictionTrainingSet[i])] += 1
    return confusionMatrixTestSet, confusionMatrixTrainingSet

def makePercentErrorRate(confusionMatrix):
```

```
55      errorPercent = 0
56      n_classes = len(confusionMatrix[0])
57      n_pred = np.sum(confusionMatrix)
58      for i in range(n_classes):
59          for j in range(n_classes):
60              if i != j:
61                  errorPercent += confusionMatrix[i][j]/n_pred
62      return np.around(errorPercent*100,2)
63
64  class TrainingAndTestDataClass(object):
65      def __init__(self):
66          self.trainingData = []
67          self.trainingTarget = []
68          self.testData = []
69          self.testTarget = []
70      def AddToTTS(self, type, obj):
71          match type:
72              case "trainingData":
73                  self.trainingData.append(obj)
74              case "trainingTarget":
75                  self.trainingTarget.append(obj)
76              case "testData":
77                  self.testData.append(obj)
78              case "testTarget":
79                  self.testTarget.append(obj)
80
81  def makeTrainingAndTestDataClass(data, target, trainingStart, trainingStop,
    classStart, classLength, n_classes):
82      n_trainingData = trainingStop - trainingStart
83      n_testData = classLength - n_trainingData
84      n_trainingOffset = trainingStart-classStart
85      TTD = TrainingAndTestDataClass()
86      for c in range(n_classes):
87          c_off = c*classLength
88          trainingTarget = [target[c_off+trainingStart + i] for i in
    range(n_trainingData)]
89          testTarget = [target[c_off+classStart + i] if i < n_trainingOffset else
    target[c_off+trainingStop-n_trainingOffset +i] for i in range(n_testData)]
90
91          TTD.AddToTTS("trainingData",np.array([data[c_off+trainingStart + i] for i
    in range(n_trainingData)]))
92
    TTD.AddToTTS("trainingTarget",np.array(sh.fix_target(trainingTarget,n_classes)))
93          TTD.AddToTTS("testData",np.array([data[c_off+classStart + i] if i <
    n_trainingOffset else data[c_off+trainingStop-n_trainingOffset + i] for i in
    range(n_testData)]))
94          TTD.AddToTTS("testTarget",np.array(sh.fix_target(testTarget,n_classes)))
95
96      return TTD
97
98  def runIrisTask(alphaStart, n_alphas, itt, TTD, file=0):
99      alpErrList = np.array([[0.]*3 for i in range(n_alphas)])
100     startTime = dt.datetime.now().replace(microsecond=0)
101     n_classes = len(TTD.trainingData)
102     n_features = len(TTD.trainingData[0][0])
103     for i in range(n_alphas):
104         if n_alphas > 1:
105             alphaStart+=1*10**(-4)
```

```
106         W = np.zeros((n_classes, n_features))
107         bias = np.zeros((n_classes,))
108         W = np.c_[W,bias]
109 
110         W = trainUntilSatisfactory(W, TTD, alphaStart, itt)
111 
112         confusionMatrixTestSet, confusionMatrixTrainingSet =
    makeConfusionMatricies(W, TTD)
113         errorPercentTestSet = makePercentErrorRate(confusionMatrixTestSet)
114         errorPercentTrainingSet =
    makePercentErrorRate(confusionMatrixTrainingSet)
115 
116         alpErrList[i][0] = alphaStart
117         alpErrList[i][1] = errorPercentTestSet
118         alpErrList[i][2] = errorPercentTrainingSet
119 
120         if i%100==0:
121             print("Itterations ", i)
122             print("Time taken: ", dt.datetime.now().replace(microsecond=0)-
    startTime)
123     print("ConfusionMatrixTestSet: \n", confusionMatrixTestSet)
124     print("ConfusionMatrixTrainingSet: \n", confusionMatrixTrainingSet)
125     min = 100
126     minitt= 0
127     for i in range (n_alphas):
128         if alpErrList[i][1] < min:
129             min = alpErrList[i][1]
130             minitt = i
131     print("Best Alpha and ErrorMargin, with ErrorRates was: ",
    alpErrList[minitt])
132     stopTime = dt.datetime.now().replace(microsecond=0)
133     print("Time taken: ", stopTime-startTime)
134     if file != 0:
135         np.savetxt(file, alpErrList, delimiter=",")
136 
137 
138 
139 
140 
141 
142 alpha = 0.00370
143 n_a = 1
144 itt = 1000
145 
146 newData, newFeatures = sh.removeListOfFeatures(iris_data, iris_feature, [0])
    #leave list empty to include all
147 sh.hist(newData,newFeatures,iris_classes,"RemovedWorsedOneHist.png") #add a file
    as last input if you want to save
148 TTD = makeTrainingAndTestDataClass(newData, iris_target, 0, 30, 0, 50, 3)
149 runIrisTask(alpha,n_a, itt, TTD)
```