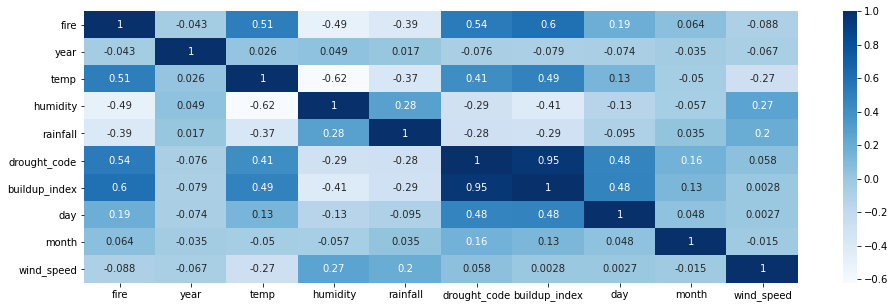
import numpy as np  
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn.metrics import roc\_curve, auc  
from sklearn.preprocessing import StandardScaler

data = pd.read\_table("wildfires.txt", delim\_whitespace=True)  
data['fire'] = data['fire'].map({"yes":1, "no":0})

corr = data.corr()  
plt.figure(figsize = (16,5))  
sns.heatmap(corr, cmap="Blues",annot = True)  
plt.savefig("Heatmap")



# Perceptron From Scratch

class Perceptron:  
   
 #activation function for the perceptron  
 def sigmoid(self, x):  
 return 1/(1+np.exp(-x))  
  
 #constructor call  
 def \_\_init\_\_(self):  
 self.learning\_rate = None  
 self.epochs = None  
 self.weights = None  
 self.bias = None  
  
 #fit function to adjust weights and bias for a certain number of epochs  
 def fit(self, X\_train, Y\_train, epochs = 100, learning\_rate= 0.00001):  
 data\_size, vector\_size = X\_train.shape  
 self.epochs = epochs  
 self.learning\_rate = learning\_rate  
 self.weight = np.random.rand(vector\_size)  
 self.bias = 0  
 for i in range(self.epochs):  
 for x, y in zip(X\_train, Y\_train):  
 Y\_pred = self.sigmoid(np.dot(x, self.weight) + self.bias)  
 self.weight += self.learning\_rate \* (y - Y\_pred) \* x  
 self.bias += self.learning\_rate \* (y - Y\_pred)  
  
 def predict(self, X):  
 return self.sigmoid(np.dot(X, self.weight) + self.bias)

#splitiing datta 1/3 for test 2/3 for training and crossvalidation  
from sklearn.model\_selection import train\_test\_split  
train\_data, test\_data = train\_test\_split(data, test\_size=0.33, random\_state = 100)

#scaling data such that mean is zero and standard deviation as one  
scaler = StandardScaler()  
X\_train = scaler.fit\_transform(train\_data)  
X\_test = scaler.transform(test\_data)

#splitting x and y data  
X\_train = train\_data[["temp","drought\_code","buildup\_index"]]  
Y\_train = train\_data[['fire']]  
X\_test = test\_data[["temp","drought\_code","buildup\_index"]]  
Y\_test = test\_data[['fire']]

#pandas to numpy for mathematical operations  
X\_train = X\_train.to\_numpy().reshape(X\_train.shape[0], 3)  
Y\_train = Y\_train.to\_numpy().reshape(Y\_train.shape[0],)   
X\_test = X\_test.to\_numpy().reshape(X\_test.shape[0], 3)  
Y\_test = Y\_test.to\_numpy().reshape(Y\_test.shape[0],)

from sklearn.model\_selection import KFold  
from sklearn.metrics import accuracy\_score  
  
#function outputs 5 models trained using crossvalidation  
def crossvalidation(X\_train, Y\_train, splits):  
 cv = KFold(n\_splits=splits)  
 models = []  
 for train, test in cv.split(X\_train):  
 roundoff = lambda x: np.round(x)  
 x\_train = X\_train[train]  
 y\_train = Y\_train[train]  
 x\_val = X\_train[test]  
 y\_val = Y\_train[test]  
  
 p = Perceptron()  
 p.fit(x\_train, y\_train, 100, 0.00001)  
 y\_pred = roundoff(p.predict(x\_val))  
 print("model accuracy : ",accuracy\_score(y\_val, y\_pred))  
 models.append(p)  
 return models

models = crossvalidation(X\_train, Y\_train, 5)

model accuracy : 0.8571428571428571  
model accuracy : 0.7407407407407407  
model accuracy : 0.7037037037037037  
model accuracy : 0.8888888888888888  
model accuracy : 0.6666666666666666

#this fucntion outputs probabilites and discrete values of the preditions made on Y\_test as a list  
def predictions\_cross\_val(models, X\_test):  
 probablities = np.empty(0, )  
 discrete\_vals = np.empty(0, )  
 for test in X\_test:  
 roundoff = lambda x: np.round(x)  
 pred = np.average([models[0].predict(test),   
 models[1].predict(test),  
 models[2].predict(test),  
 models[3].predict(test),  
 models[4].predict(test)])  
 probablities = np.append(probablities, pred)  
 discrete\_vals = np.append(discrete\_vals, roundoff(pred))  
   
 return probablities, discrete\_vals

Y\_pred\_probabilities, Y\_pred = predictions\_cross\_val(models, X\_test)

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report  
accuracy\_score(Y\_pred, Y\_test)

0.8676470588235294

confusion\_matrix(Y\_pred, Y\_test)

array([[27, 5],  
 [ 4, 32]], dtype=int64)

print(classification\_report(Y\_test, Y\_pred))

precision recall f1-score support  
  
 0 0.84 0.87 0.86 31  
 1 0.89 0.86 0.88 37  
  
 accuracy 0.87 68  
 macro avg 0.87 0.87 0.87 68  
weighted avg 0.87 0.87 0.87 68

#saving roc data for plotting later  
from sklearn.metrics import roc\_curve, auc  
  
fpr\_perceptron, tpr\_preceptron, threshold = roc\_curve(Y\_test, Y\_pred\_probabilities)  
auc\_perceptron = auc(fpr\_perceptron, tpr\_preceptron)

# MLP Classifier from scratch

class Layer:  
 def \_\_init\_\_(self):  
 self.input = None  
 self.output = None  
  
 # computes the output Y of a layer for a given input X  
 def forward\_propagation(self, input):  
 pass  
  
 # computes partial derivatives for adjusting weights  
 def backward\_propagation(self, output\_error, learning\_rate):  
 pass

class FCLayer(Layer):  
 # input no of layers output no of outputs  
 def \_\_init\_\_(self, input\_size, output\_size):  
 self.weights = np.random.rand(input\_size, output\_size) -0.5  
 self.bias = np.random.rand(1, output\_size) -0.5  
  
 # returns wx+b  
 def forward\_propagation(self, input\_data):  
 self.input = input\_data  
 self.output = np.dot(self.input, self.weights) + self.bias  
 return self.output  
   
 # computes partial derivatives and updates weights {weight = weight - alpha\*partialderivative}  
 def backward\_propagation(self, output\_error, learning\_rate):  
 input\_error = np.dot(output\_error, self.weights.T)  
 weights\_error = np.dot(self.input.T, output\_error)  
   
 self.weights -= learning\_rate \* weights\_error  
 self.bias -= learning\_rate \* output\_error  
 return input\_error

class ActivationLayer(Layer):  
   
 #activation function  
 def sigmoid(self, x):  
 return 1/(1+np.exp(-x))  
   
 #activation derivative  
 def sigmoid\_derivative(self, x):  
 return self.sigmoid(x)\*(1-self.sigmoid(x))  
   
 def \_\_init\_\_(self):  
 pass  
   
 # applying sigmoid to the FClayer output  
 def forward\_propagation(self, input\_data):  
 self.input = input\_data  
 self.output = self.sigmoid(self.input)  
 return self.output  
  
   
 # applying sigmoid derivative to the Activation layer  
 def backward\_propagation(self, output\_error, learning\_rate):  
 return self.sigmoid\_derivative(self.input) \* output\_error

class Network:  
   
 def \_\_init\_\_(self):  
 self.layers = []  
   
 #loss function   
 def mean\_squared\_error\_derivative(self, Y\_true, Y\_pred):  
 return 2\*(Y\_pred - Y\_true)/ Y\_true.size  
  
 # add layers to network  
 def add(self, layer):  
 self.layers.append(layer)  
  
 #predict output for a list of inputdata  
 def predict(self, input\_data):  
 result = np.empty(0, )  
  
 #calculate results for all the data  
 for i in range(len(input\_data)):  
 # forward propagation  
 output = input\_data[i]  
 for layer in self.layers:  
 output = layer.forward\_propagation(output)  
 result = np.append(result, output)  
  
 return result  
  
 # train the network  
 def fit(self, x\_train, y\_train, epochs, learning\_rate):  
  
 #training for a number of epochs  
 for i in range(epochs):  
 for j in range(len(x\_train)):  
 # forward propagation  
 output = x\_train[j]  
 for layer in self.layers:  
 output = layer.forward\_propagation(output)  
  
 # backward propagation  
 error = self.mean\_squared\_error\_derivative(y\_train[j], output)  
 for layer in reversed(self.layers):  
 error = layer.backward\_propagation(error, learning\_rate)

from sklearn.model\_selection import train\_test\_split  
train\_data, test\_data = train\_test\_split(data, test\_size=0.33, random\_state = 100)

X\_train = train\_data[["temp","drought\_code","buildup\_index"]]  
Y\_train = train\_data[['fire']]  
X\_test = test\_data[["temp","drought\_code","buildup\_index"]]  
Y\_test = test\_data[['fire']]

X\_train = X\_train.to\_numpy().reshape(X\_train.shape[0], 1, 3)  
Y\_train = Y\_train.to\_numpy().reshape(Y\_train.shape[0],)  
X\_test = X\_test.to\_numpy().reshape(X\_test.shape[0], 1, 3)  
Y\_test = Y\_test.to\_numpy().reshape(Y\_test.shape[0],)

net = Network()  
net.add(FCLayer(3, 25))   
net.add(ActivationLayer())  
net.add(FCLayer(25, 1))   
net.add(ActivationLayer())  
net.fit(X\_train, Y\_train, epochs = 35, learning\_rate = 0.001)

from sklearn.model\_selection import KFold  
from sklearn.metrics import accuracy\_score  
  
def cross\_validation(X\_train, Y\_train, splits):  
 roundoff = lambda x: np.round(x)  
 cv = KFold(n\_splits=splits)  
 models = []  
 for train, test in cv.split(X\_train):  
 x\_train = X\_train[train]  
 y\_train = Y\_train[train]  
 x\_val = X\_train[test]  
 y\_val = Y\_train[test]  
  
 net = Network()  
 net.add(FCLayer(3, 25))   
 net.add(ActivationLayer())  
 net.add(FCLayer(25, 1))   
 net.add(ActivationLayer())  
 net.fit(x\_train, y\_train, epochs = 35, learning\_rate = 0.001)  
 y\_pred = roundoff(net.predict(x\_val))  
 print(accuracy\_score(y\_val, y\_pred))  
 models.append(net)  
 return models

models = cross\_validation(X\_train, Y\_train, 5)

0.9285714285714286  
0.8148148148148148  
0.8148148148148148  
0.8888888888888888  
0.7037037037037037

Y\_pred\_probabilities, Y\_pred = predictions\_cross\_val(models, X\_test)

from sklearn.metrics import accuracy\_score, confusion\_matrix  
  
accuracy\_score(Y\_test, Y\_pred)

0.8823529411764706

confusion\_matrix(Y\_test, Y\_pred)

array([[23, 8],  
 [ 0, 37]], dtype=int64)

print(classification\_report(Y\_test, Y\_pred))

precision recall f1-score support  
  
 0 1.00 0.74 0.85 31  
 1 0.82 1.00 0.90 37  
  
 accuracy 0.88 68  
 macro avg 0.91 0.87 0.88 68  
weighted avg 0.90 0.88 0.88 68

from sklearn.metrics import roc\_curve, auc  
  
fpr\_mlp, tpr\_mlp, threshold = roc\_curve(Y\_test, Y\_pred\_probabilities)  
auc\_mlp = auc(fpr\_mlp, tpr\_mlp)

# Perceptron Sklearn for Reference

from sklearn.model\_selection import train\_test\_split  
train\_data, test\_data = train\_test\_split(data, test\_size=0.33, random\_state = 100)

#scaling data such that mean is zero and standard deviation as one  
scaler = StandardScaler()  
X\_train = scaler.fit\_transform(train\_data)  
X\_test = scaler.transform(test\_data)

#splitting data in X and y  
X\_train = train\_data[["temp","drought\_code","buildup\_index"]]  
Y\_train = train\_data[['fire']]  
X\_test = test\_data[["temp","drought\_code","buildup\_index"]]  
Y\_test = test\_data[['fire']]

# converting to numpy for mathematical operations  
X\_train = X\_train.to\_numpy().reshape(X\_train.shape[0], 3)  
Y\_train = Y\_train.to\_numpy().reshape(Y\_train.shape[0],)  
X\_test = X\_test.to\_numpy().reshape(X\_test.shape[0], 3)  
Y\_test = Y\_test.to\_numpy().reshape(Y\_test.shape[0],)

#function outputs 5 models trained using crossvalidation  
from sklearn.model\_selection import KFold  
from sklearn.linear\_model import Perceptron  
from sklearn.metrics import accuracy\_score  
  
def cross\_validation(X\_train, Y\_train, splits):  
 roundoff = lambda x: np.round(x)  
 cv = KFold(n\_splits=splits)  
 models = []  
 for train, test in cv.split(X\_train):  
 x\_train = X\_train[train]  
 y\_train = Y\_train[train]  
 x\_val = X\_train[test]  
 y\_val = Y\_train[test]  
  
 p = Perceptron(alpha = 0.0001)  
 p.fit(X\_train, Y\_train)  
 y\_pred = roundoff(p.predict(x\_val))  
 print(accuracy\_score(y\_val, y\_pred))  
 models.append(p)  
 return models

models = cross\_validation(X\_train, Y\_train, 5)

0.9285714285714286  
0.8518518518518519  
0.8148148148148148  
0.8888888888888888  
0.6666666666666666

#this function outputs probabilites and discrete values of the preditions made on Y\_test as a list  
  
def predictions\_cross\_val(models, X\_test):  
 probablities = np.empty(0, )  
 discrete\_vals = np.empty(0, )  
 for test in X\_test:  
 roundoff = lambda x: np.round(x)  
 pred = np.average([models[0].predict(test.reshape(1, -1)),   
 models[1].predict(test.reshape(1, -1)),  
 models[2].predict(test.reshape(1, -1)),  
 models[3].predict(test.reshape(1, -1)),  
 models[4].predict(test.reshape(1, -1))])  
 probablities = np.append(probablities, pred)  
 discrete\_vals = np.append(discrete\_vals, roundoff(pred))  
   
 return probablities, discrete\_vals

Y\_pred\_probabilities, Y\_pred = predictions\_cross\_val(models, X\_test)

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report  
# performance metrics  
accuracy\_score(Y\_test, Y\_pred)

0.8823529411764706

confusion\_matrix(Y\_test, Y\_pred)

array([[24, 7],  
 [ 1, 36]], dtype=int64)

print(classification\_report(Y\_test, Y\_pred))

precision recall f1-score support  
  
 0 0.96 0.77 0.86 31  
 1 0.84 0.97 0.90 37  
  
 accuracy 0.88 68  
 macro avg 0.90 0.87 0.88 68  
weighted avg 0.89 0.88 0.88 68

# MLP Sklearn for reference

from sklearn.model\_selection import train\_test\_split  
train\_data, test\_data = train\_test\_split(data, test\_size=0.33, random\_state = 100)

#scaling data such that mean is zero and standard deviation as one  
scaler = StandardScaler()  
X\_train = scaler.fit\_transform(train\_data)  
X\_test = scaler.transform(test\_data)

#splitting data in X and y  
X\_train = train\_data[["temp","drought\_code","buildup\_index"]]  
Y\_train = train\_data[['fire']]  
X\_test = test\_data[["temp","drought\_code","buildup\_index"]]  
Y\_test = test\_data[['fire']]

# converting to numpy for mathematical operations  
X\_train = X\_train.to\_numpy().reshape(X\_train.shape[0], 3)  
Y\_train = Y\_train.to\_numpy().reshape(Y\_train.shape[0],)  
X\_test = X\_test.to\_numpy().reshape(X\_test.shape[0], 3)  
Y\_test = Y\_test.to\_numpy().reshape(Y\_test.shape[0],)

#function outputs 5 models trained using crossvalidation  
from sklearn.model\_selection import KFold  
from sklearn.neural\_network import MLPClassifier  
from sklearn.metrics import accuracy\_score  
  
def cross\_validation(X\_train, Y\_train, splits):  
 roundoff = lambda x: np.round(x)  
 cv = KFold(n\_splits=splits)  
 models = []  
 for train, test in cv.split(X\_train):  
 x\_train = X\_train[train]  
 y\_train = Y\_train[train]  
 x\_val = X\_train[test]  
 y\_val = Y\_train[test]  
  
 mlp= MLPClassifier(solver = 'sgd', hidden\_layer\_sizes = (25, ) ,alpha= 0.00001)  
 mlp.fit(X\_train, Y\_train)  
 y\_pred = roundoff(mlp.predict(x\_val))  
 print(accuracy\_score(y\_val, y\_pred))  
 models.append(mlp)  
 return models

models = cross\_validation(X\_train, Y\_train, 5)

0.8928571428571429  
0.8888888888888888  
0.7037037037037037  
0.8888888888888888  
0.6666666666666666

Y\_pred\_probabilities, Y\_pred = predictions\_cross\_val(models, X\_test)

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report  
accuracy\_score(Y\_test, Y\_pred)

0.8823529411764706

confusion\_matrix(Y\_test, Y\_pred)

array([[26, 5],  
 [ 3, 34]], dtype=int64)

print(classification\_report(Y\_test, Y\_pred))

precision recall f1-score support  
  
 0 0.90 0.84 0.87 31  
 1 0.87 0.92 0.89 37  
  
 accuracy 0.88 68  
 macro avg 0.88 0.88 0.88 68  
weighted avg 0.88 0.88 0.88 68

plt.plot(fpr\_perceptron, tpr\_preceptron, linestyle='-', label=' scratch percepton auc = %0.3f' % auc\_perceptron)  
plt.plot(fpr\_mlp, tpr\_mlp, linestyle='-', label=' scratch mlp auc = %0.3f)' % auc\_mlp)  
  
  
plt.xlabel('False Positive Rate -->')  
plt.ylabel('True Positive Rate -->')  
   
plt.legend()  
plt.savefig("Roc curve")  
plt.show()

