Diego Fernando Montaño Pérez A01282875

Aprendizaje automático

Práctica 01

Nota para profesor: Al hacer mi predicción y taza de error me estaba dando valores muy pequeños y por mas que buscaba en que me fallaba no encontré el error. También me falto agregar graficas para el archivo Default.txt. A pesar de todo esto entiende bien como funciona el modelo y no me quede con dudas, fueron más problemas de programación que tuve pues no había usado mucho Python y me costo algo de trabajo esa parte.

Como correr programa desde consola:

python practica1 genero.txt 0.8 0.2 0.005 0.0001 37

Resultados en consola que muestra el programa con archivo “genero.txt”:

```
Vector beta = [[ 0.00225448]
 [ 0.10712826]
 [-0.04497361]]
Interacciones necesarias para entrenar modelo = 786
Valor de alpha empleado = 0.5
Valor de threshold empleado = 0.0001
Taza de error que tiene el modelo = 0.085

Regresion Logistica con Scikit-Learn:
Vector beta usando Scikit-Learn: Coeficiente = [[ 0.4714644  -0.19397827]] Intercepcion = [0.00990834]
Taza de error usando Scikit-Learn: 0.91825
```

Programa

Parte 1:

```
def main():
  # genero.txt  Default.txt
  dataSet =  sys.argv[1] # param1: nombre del archivo del dataset.
  trainSetProc = sys.argv[2]#  param2: porcentaje de elementos del training set (ocupar valores enteros).
  testSetProc =  sys.argv[3]#  param3: porcentaje de elementos del test set (ocupar valores enteros).
  alpha = sys.argv[4] # param4: valor de α.  0.5 o 0.0005
  threshold =  sys.argv[5] # param5: valor del threshold (1x10^-4).0.0001
  semillaGem = sys.argv[6] # param6: semilla del generador de números pseudoaleatorios. 37

  #Links a los archivos que se encuentran en Github y se guarda la información en una matriz
  if dataSet == "Default.txt":
    url = "https://raw.githubusercontent.com/DiegoMontano2705/MachineLearning/main/Parcial1/Default.txt"
    data = pd.read_csv(url, sep='\t')
    default = {'Yes': 0, 'No': 1}
    student = {'Yes': 0, 'No': 1}
    data.default = [default[item] for item in data.default]
    data.student = [student[item] for item in data.student]
    x = data[['student','balance','income']] #features del vector de entrada
    y = data['default'] #Categoria a ser predecida
  elif dataSet == "genero.txt":
    url = "https://raw.githubusercontent.com/DiegoMontano2705/MachineLearning/main/Parcial1/genero.txt"
    data = pd.read_csv(url)
    gender = {'Male': 0, 'Female': 1}
    data.Gender = [gender[item] for item in data.Gender]
    x = data[['Height','Weight']] #features del vector de entrada
    y = data['Gender'] #Categoria a ser predecida
  else:
    breastCancer = datasets.load_breast_cancer()
    #Modelando arbol de decision
    x = breastCancer.data
    y = breastCancer.target
    print("Cant process file")

  #Creando vector de entrenamiento
  x_train,x_test,y_train,y_test = train_test_split(x, y, test_size=testSetProc, random_state=semillaGem)
  x_train = x_train.values  #dataset datos aletorios
  y_train = y_train.values  #dataset datos aletorios
  y_test = y_test.values
  betaFinal,n = batchGradientDescent(x_train,y_train,alpha,threshold)
  #Error cuadratico medio
  m1,n1 = x_test.shape
  aux = np.ones(m1)
  aux = aux.reshape(m1,1)
  x_test = np.append(aux,x_test,axis=1)
  y_test = y_test.reshape(-1,1)
```

Diego Fernando Montaño Pérez A01282875

En la primera parte del código es donde se leen los datos que manda el usuario al correr el programa y dependiendo del nombre del archivo guarda las tablas de la base de datos de diferente manera pues están organizados de diferente manera, también se modifican valores dentros de las tablas para maneras todo con números, como en el caso de genero.txt donde en la parte de genero de la tabla, se sustituye la palabra 'Male' a 0 y Female a '1', lo mismo pasa en el caso de Default.txt donde la palabra Yes = '0' y No = '1'.

Una ves que se tienen los datos y se separa en las variables X y Y los datos y el objetivo de predicción, se crean los vectores de entrenamiento y pruebas en donde de manera aleatorio se escogen quedando un 80% en entrenamiento y un 20% en pruebas. También se modifican las tablas agregando una columna de '0' para poder hacer las operaciones más adelante.

Parte 2:

```
#Funcion logistica
def logit(t):
  return 1.0 / (1 + np.exp(-t))

def batchGradientDescent(x,y,alpha,threshold):
  #Añadiendo una columna con 1's a x
  m1,n1 = x.shape
  aux = np.ones(m1)
  aux = aux.reshape(m1,1)
  x = np.append(aux,x,axis=1)
  #Creando beta
  m2,n2 = x.shape
  aux = np.ones(n2)
  beta = np.zeros(n2)
  beta = beta.reshape(n2,1)          #beta = np.ones((np.shape(x)[1],1))
  #beta[0] = 0.5
  #Cambiando forma de y sin perder sus datos
  y = y.reshape(-1,1)
  x_transpose = x.transpose()
  j = 0
  betasResta = 1
  #Algoritmo
  while betasResta > threshold:
    B0 = beta
    t = np.dot(x,beta)
    functLogistica = logit(t) - y
    beta = beta - (alpha/m2)*((np.dot(x_transpose,functLogistica))/m2)
    betasResta = np.linalg.norm(B0-beta)
    j = j + 1
  return beta,j
```

Este es el algoritmo de Batch Gradient Descent que programe para obtener el valor de theta, que en este caso lo llame beta. Al inicio les doy forma a las matrices para poder hacer las operaciones correspondientes apoyándome de la librería numpy para el manejo de matrices. Al final del algoritmo se regresan el vector theta y el numero de iterraciónes que se requirieron para obtener el un resultado menos al threshold.

Diego Fernando Montaño Pérez A01282875

Parte 3:

```
def tasaDeError(x,y,beta):
  mse = 0
  j = 0
  b = beta.transpose()
  probTotal = []
  vectorProb = []
  for i in x:
    prob = logit(np.dot(b,i))
    vectorProb.append(prob)
    if prob >= 0.5:
      probTotal.append(1.0)
    else:
      probTotal.append(0)
  for i in y:
    #print(probTotal[j]," ", i)
    if probTotal[j] != i :
      mse = mse + 1
    j = j+1
  mse = (1/len(x))* mse
  return mse,probTotal,vectorProb
```

Esta función de tasaDeError me ayuda a encontrar todas las predicciones posibles para después apoyarme con estas mismas para saber la tasa de error del modelo para saber la precisión de nuestro modelo. Al final se regresa E, Y^, y p^.

Parte 4:

```
def sciKitLearnMethod(x,y,xTest,yTest,alpha,threshold):
  model = LogisticRegression()
  #Entrenando modelo
  model.fit(x,y)
  y_pred = model.predict(x)
  #Encontrando valores de theta
  beta1 = model.coef_
  beta0 = model.intercept_
  print("Vector beta usando Scikit-Learn: Coeficiente =",beta1,"Intercepcion =",beta0)
  #Encontrando precison del modelo
  accuracy = accuracy_score(y,y_pred)
  print("Taza de error usando Scikit-Learn:",accuracy)
```

La penúltima parte del código se hace de nuevo la regresión logística, pero esta apoyándonos de la librería Scikit-learn, donde a través de métodos se crea un modelo y entrena un modelo de manera más sencilla, después encontramos los valores de theta que en este caso son el coeficiente e intercepción. Por último, se regresa la precisión del modelo

Parte 5:

```
pred,y_gorrito,vectorProb = tasaDeError(x_test,y_test,betaFinal)

#Valores finales
print("Vector beta =",betaFinal)
print("Interacciones necesarias para entrenar modelo =",n)
print("Valor de alpha empleado =",alpha)
print("Valor de threshold empleado =",threshold)
print("Taza de error que tiene el modelo =",pred)

#Vectores finales guardados en archivo
file = open('vectores.txt', 'w')
file.write("Vector X\n")
np.savetxt(file,x_test,fmt='%1.3f')
file.write("Vector Y_^\n")
file.write(str(y_gorrito))
file.write('\n')
file.write("Vector Y\n")
np.savetxt(file,y_test, fmt='%1.3f')
file.write("Vector Probabilidades(P^) Y\n")
file.write(str(vectorProb))
file.close()

#Grafica de la clasificacion para archivo Default.txt
#if dataSet == "Default.txt":
  #Y^ vs X

print("\nRegresion Logistica con Scikit-Learn:")
sciKitLearnMethod(x_train,y_train,x_test,y_test,alpha,threshold)
```

Por último, se imprimen los resultados como se encuentra en la parte del inicio del documento y se guardan los vectores correspondientes en un archivo de texto llamado “vectores.txt”