Diego Fernando Montaño Pérez A01282875

Aprendizaje AutomáticoTC3020

Reporte Práctica 4

“Limpieza de datos y su efecto”

Dr. Jesús Guillermo Falcón Cardona

Introducción

La practica fue programada en Python y se usó Jupyter para estar probando todos los pasos que se describirán en el reporte, me apoye de librarías de sklearn y pandas para lograr resultados satisfactorios.

Reporte

1. Lo primero que se hizo fue con ayuda de la librería Pandas leer el archivo de Default.txt el cual se encuentra en mi GitHub de forma pública, después se tuvieron que cambiar los valores en las columnas “default” y “student” pues eran categóricos, se usaron los valores de ‘0’ para la palabra YES y ‘1’ para NO. Por último, se dividió la tabla con las variables X y Y para separar las features.

```
def readFile():
  url = "https://raw.githubusercontent.com/DiegoMontano2705/MachineLearning/main/Parcial1/RegresionLogistica/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
  return x,y
```

Ya teniendo los datos en tablas de dividió el training set usando un 80% de la tabla de forma aleatoria. El resto se usó para el training set.

```
def main():
  x,y = readFile()
  # Creando vector de entrenamiento
  x_train,x_test,y_train,y_test = train_test_split(x, y, test_size=0.2, random_state=42)
  x_train = x_train.values  #dataset datos aletorios
  y_train = y_train.values  #dataset datos aletorios
  y_test = y_test.values
```

2. Después se generó una función la cual recibía el training set y lo modificaba, se hizo el uso de un loop que iteraba cada registro del set, se generaba un número aleatorio usando la función Random.random() que regresa un valor entre 0 y 1, luego se checaba si era menor a 0.2 y con esto se simulaba la probabilidad que se tenía para que se eliminar los valores de “balance” e “income”. Lo mismo se hizo de nuevo pero esta ves con una probabilidad mas baja para solo eliminar el valor de balance.

```
#Funcion para modificar dataset original e introducirle valores nulos
def modifySet(x):

  x2 = x.copy()

  # Recorrer todos los registros del training set y con una probabilidad de 0.2
  # Elimina los valores delos features balancee income para una instancia en especifico
  i = 0
  for row in x2:
    if (random.random()) < 0.2:
      np.put(x2[i],[1,2],[np.nan,np.nan])
    i = i + 1

  #Para la columna de features 'balance', recorrer todos los valores y con una proabilidad de 0.1,eliminar el valor correspondiente
  z = 0
  for row in x2:
    if (random.random()) < 0.1:
      np.put(x2[z],1,np.nan)
    z = z + 1

  return x2
```

3. X2 es lo equivalente a T’ modificado

4. Lo siguiente ya teniendo el traning set modificado, fue rellenar todos los espacios vacíos usando diferentes métodos.

```
# Llama funcion donde se relleneran datos faltantes de diferente manera
x2Mod = supplyValuesMean(x2_train)
x3Mod = supplyValuesLinearReg(x2_train)
x4Mod = supplyValuesKkNeigh(x2_train)
```

El primero fue el de rellenar estos datos usando los valores promedios de cada feature. Se uso la función de numpy.nanmean() la cual regresa una lista con las medias de cada columna ignorando los valores nulos(Nan). Después dentro de un loop se revisaba cada renglón y si tenía un valor nulo se le asignaba con el ya antes calculado.

```
#Rellena los datos faltantes a traves del uso de los valores promedio de cada feature.
def supplyValuesMean(x):
  x2 = x.copy()
  mean = np.nanmean(x2,axis=0)
  i = 0
  for row in x2:
    if np.isnan(row[1]):
      np.put(x2[i],1,mean[1])
    if np.isnan(row[2]):
      np.put(x2[i],2,mean[2])
    i = i + 1
  return x2
```

El segundo fue el de rellenar los valores usando regresión lineal, este fue muy sencillo pues la librería de Pandas tiene incluida una función que hace esto mismo, solo se debe poner que método es el que se quiere usar y automáticamente se rellenan los datos (Como estaba usando el dataset en un arreglo de numpy se tuvo que volver a hacer una conversión a Series para usar la función de Pandas)

```
#Rellena los datos faltantes a traves del uso de valores generados a traves de regresion lineal.
def supplyValuesLinearReg(x):
  x3 = x.copy()
  df = pd.DataFrame(data=x3, columns=["student","balance", "income"])
  df['balance'].interpolate(method='linear', inplace=True)
  df['income'].interpolate(method='linear', inplace=True)
  return df
```

5. El último método fue a través de la búsqueda de los 20 vecinos más cercanos de cada instancia faltante. Para esto se usó la librería KNNImputer() la hace esta función, solo se le debe especificar la cantidad de vecinos que se desean buscar y luego juntarlo junto con el dataset a trabajar.

```
#Rellena los datos faltantes a traves de la busqueda de los 20 vecinos mas cercanos de cada instancia faltante.
def supplyValuesKkNeigh(x):
  x4 = x.copy()
  imputer = KNNImputer(n_neighbors=20)
  x4 = imputer.fit_transform(x4)
  return x4
```

6. La última parte de la practica era de usar la librería de Scikit Learn para usar diferentes técnicas de clasificación con los distintos datasets que se generaron. Estos los saque de mis practicas pasadas y fue lo más sencillo del reporte.
   En las siguientes imágenes dejo la función que obtenía todos los valores y una impresión de pantalla de los resultados finales:

```
# Aplicar las tecnicas de clasificacion de regresion lineal,
# arboles dedecision y 50-Nearest Neighbors,
# empleando las funciones proporcionadas por SciKit-Learn.
print("T'1:")
applySciKitTools(x2Mod,y_train,x_test,y_test)
print("T'2:")
applySciKitTools(x3Mod,y_train,x_test,y_test)
print("T'3:")
applySciKitTools(x4Mod,y_train,x_test,y_test)
#Original
print("T Original:")
applySciKitTools(x_train,y_train,x_test,y_test)
```

Diego Fernando Montaño Pérez A01282875

```python
def applySciKitTools(x,y,x_test,y_test):
  #Usando Regresion lineal
  model = LogisticRegression()
  model.fit(x,y)
  accuracy = model.score(x_test,y_test)
  print("Precision Regresion Lineal Scikit-Learn:",accuracy)

  #Usando arboles de decision
  model = tree.DecisionTreeClassifier(max_depth=2, random_state=30)
  model.fit(x,y)
  accuracy= model.score(x_test, y_test)
  print("Precision Arboles de Decision Scikit-Learn",accuracy)

  #Usando 50-NearestNeighbors
  model = KNeighborsClassifier(n_neighbors=50, algorithm='brute')
  model.fit(x, y)
  accuracy = model.score(x_test, y_test)
  print("Precision 50-NearestNeighbors Scikit-Learn:",accuracy)
```

Resultados

```
T'1:
Precision Regresion Lineal Scikit-Learn: 0.9675
Precision Arboles de Decision Scikit-Learn 0.968
Precision 50-NearestNeighbors Scikit-Learn: 0.9655
T'2:
Precision Regresion Lineal Scikit-Learn: 0.9665
Precision Arboles de Decision Scikit-Learn 0.968
Precision 50-NearestNeighbors Scikit-Learn: 0.9655
T'3:
Precision Regresion Lineal Scikit-Learn: 0.9655
Precision Arboles de Decision Scikit-Learn 0.968
Precision 50-NearestNeighbors Scikit-Learn: 0.9655
T Original:
Precision Regresion Lineal Scikit-Learn: 0.9655
Precision Arboles de Decision Scikit-Learn 0.968
Precision 50-NearestNeighbors Scikit-Learn: 0.9655
```

Diego Fernando Montaño Pérez A01282875

Reflexión

De las practicas que hemos tenido esta ha sido en la que menos complicaciones he tenido, esto se debió por que se usaron muchas librerías que ahorraban el trabajo que mientras en otras prácticas nosotros desarrollábamos por completo el algoritmo.
En cuanto a los resultados obtenidos dentro de este programa se pudo observar que la precisión aplicada en cada método usando los datasets que fueron completados de diferentes maneras, tuvieron resultados muy parecidos y en algunos fueron iguales, esto se debió a que las maneras en que se llenaron los datos incompletos dan valores muy parecidos o que no afectan tanto al momento de clasificar. La base de datos usada también tuvo mucha influencia en estos resultados pues puede que otra con datos diferentes de otra precisión ya se mejor o peor.

```
T'1:
Precision Regresion Lineal Scikit-Learn: 0.9675
Precision Arboles de Decision Scikit-Learn 0.968
Precision 50-NearestNeighbors Scikit-Learn: 0.9655
T'2:
Precision Regresion Lineal Scikit-Learn: 0.9665
Precision Arboles de Decision Scikit-Learn 0.968
Precision 50-NearestNeighbors Scikit-Learn: 0.9655
T'3:
Precision Regresion Lineal Scikit-Learn: 0.9655
Precision Arboles de Decision Scikit-Learn 0.968
Precision 50-NearestNeighbors Scikit-Learn: 0.9655
T Original:
Precision Regresion Lineal Scikit-Learn: 0.9655
Precision Arboles de Decision Scikit-Learn 0.968
Precision 50-NearestNeighbors Scikit-Learn: 0.9655
```