ML_sin_frameworks

September 10, 2023

1 Implementacion de regresion logistica para clasificacion multiclase sin librerias de machine learning

El objetivo de este proyecto es hacer una implementacion del modelo de regresion logistica para clasificacion multiclase. y con el modelo desarrollado poder hacer uso sobre los datos de iris. Todo esto sin hacer uso de librerias de machine learning.

El problema que estamos intentando resolver es de clasificacion esto consiste en dado un conjunto de datos con caracteristicas y etiquetas, encontrar una funcion que pueda predecir la etiqueta de una nueva muestra. En este caso el problema es de clasificacion multiclase, esto quiere decir que las etiquetas pueden tomar mas de dos valores.

[26]:
```
# Librerias de manejo de datos
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Funciones propias con la implementacion del modelo
from funciones import predecir_ovr, entrenar_ovr, evaluar
```

Los datos que estamos utilizando son los datos de iris, estos datos son muy conocidos en el area de machine learning y son utilizados para hacer pruebas de modelos de clasificacion. Los datos consisten en 150 muestras de flores de iris, cada muestra tiene 4 caracteristicas y una etiqueta que indica a que clase pertenece la muestra. Las clases son 3: Iris-setosa, Iris-versicolor, Iris-virginica.

Fuente: https://www.kaggle.com/datasets/uciml/iris

[27]:
```
# Se carga el dataset original
df = pd.read_csv('iris.csv')

df.head()
```

[27]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

[28]:
```
# Se elimina el id ya que no se va a utilizar
df = df.drop(['Id'], axis=1)

# Se cambian los nombres de las clases para convertir variable categoricas a␣
↪numericas
df['Species'] = df['Species'].map({'Iris-setosa': 0, 'Iris-versicolor': 1,␣
↪'Iris-virginica': 2})
```

[29]:
```
# Se separan los datos de las etiquetas
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Se convierten los datos a numpy arrays
X = np.array(X)
y = np.array(y)

# Se agrega una columna de unos para el termino independiente, esto forma parte␣
↪de la implementacion del modelo
X = np.hstack([np.ones((X.shape[0], 1)), X])
```

[30]:
```
# Se separan los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42,␣
↪test_size=0.2)
```

[31]:
```
# Se entrenar un clasificador para cada clase
thetas = []
clases = np.unique(y)
for clase in clases:
    theta = entrenar_ovr(X_train, y_train, clase, 0.01, 1000)
    thetas.append(theta)
```

[32]:
```
# Se hacen las predicciones
predicciones = predecir_ovr(X_test, thetas)

predicciones = pd.DataFrame(predicciones, columns=['prediccion'])

pred_test = predicciones.merge(pd.DataFrame(y_test, columns=['test']),␣
↪left_index=True, right_index=True)
```

En la siguiente tabla se muestran las predicciones del modelo y las etiquetas reales de las muestras.

Podemos observar que en estos 5 ejemplos el modelo predijo correctamente la etiqueta de 4 muestras y fallo en una.

[33]:
```
pred_test.head()
```

[33]:

	prediccion	test
0	1	1

```
1           0     0
2           2     2
3           2     1
4           1     1
```

Para nuestra implementacion solo desarrollamos la metrica de exactitud, esta metrica consiste en el porcentaje de muestras que fueron clasificadas correctamente. Para calcular la exactitud se compara la prediccion con la etiqueta real y se calcula el porcentaje de muestras que coinciden.

[34]:
```
# exactitud sobre datos de prueba
exactitud = evaluar(X_test, y_test, thetas)

print('Exactitud: ', exactitud)
```

```
Exactitud:  0.8666666666666667
```

En nuestro conjunto de datos de prueba el modelo tiene una exactitud de 86.6%, esto quiere decir que el modelo clasifico correctamente el 86.6% de las muestras.

[35]:
```
# exactitud sobre datos de entrenamiento
exactitud = evaluar(X_train, y_train, thetas)

print('Exactitud: ', exactitud)
```

```
Exactitud:  0.9083333333333333
```

En nuestro conjunto de datos de entrenamiento el modelo tiene una exactitud de 90.8%, esto quiere decir que el modelo clasifico correctamente el 90.8% de las muestras.

Como podemos ver el modelo tiene una exactitud ligeramente mejor sobre los datos de entrenamiento que sobre los datos de prueba, esto es un indicador de que el modelo no esta sobreajustando los datos de entrenamiento.