model

September 10, 2023

En este documento estamos tomando un dataset que se nos dio por medio de canvas con la informacion de diferentes cliente de una aseguradora, en la que se tiene informacion personal de los clientes asi como el costo del seguro, con esto el objetivo sera crear un modelo que nos permita predecir el costo del seguro de un cliente nuevo.

En base a lo que hemos aprendido decidi que para esta entrega realizare una red neuronal, simplemente por el hecho de que es un modelo que no hemos tenido la oportunidad de practir como otros modelos de machine learning, por lo que me parecio una buena oportunidad para aprender a usarlo.

[68]:
```
import pandas as pd

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')
```

Importamos el dataset

Para este proyecto utilizaremos una base de datos de personas aseguradas, en la que se tiene informacion personal de los clientes asi como el costo del seguro, con esto el objetivo sera crear un modelo que nos permita predecir el costo del seguro de un cliente nuevo. este es el dataset Insurance que se puede encontrar en: https://experiencia21.tec.mx/courses/406127/files/149437937?module_item_id=24944904

[69]:
```
df = pd.read_csv(r'E:
↪\Github\Portafolio_Implementacion\Final\ConFrameworks\insurance.csv',␣
↪index_col=0)
```

checamos las columnas del dataset, podemos ver que no hay valores nulos y que hay 3 varaibles categoricas que necesitaremos cambiar para poder crear nuestros modelos.

[70]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1338 entries, 0 to 1337
```

```
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 83.6+ KB
```

Cambiamos las variables categoricas a numericas

[71]:
```
df['sex'] = df['sex'].map({'female': 0, 'male': 1})
df['smoker'] = df['smoker'].map({'no': 0, 'yes': 1})

df = pd.get_dummies(df, columns=['region'], drop_first=False)
```

creamos los conjuntos de input y output

[72]:
```
X = df.drop(columns=['charges'])
y = df['charges']
```

creamos los conjuntos de entrenamiento y prueba

[73]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

escalamos los datos, esto es necesario para que la red neuronal pueda trabajar con ellos

[74]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

creamos los tensores de pytorch, estos son los datos que usaremos para entrenar y probar el modelo

[75]:
```
X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.float32).view(-1, 1)
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32).view(-1, 1)
```

creamos la red neuronal, en este caso es una red neuronal simple con 3 capas, la primera capa tiene 32 neuronas, la segunda 16 y la ultima 1, la funcion de activacion es relu para las primeras 2 capas y no hay funcion de activacion para la ultima capa ya que es una regresion.

Nuestro problema a resolver en esta etapa es una regresion, que es predecir un valor numerico no discreto en base a ciertos datos de entrada.

[76]:
```
class SimpleInsuranceNN(nn.Module):
    def __init__(self, input_dim):
        super(SimpleInsuranceNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, 32)
        self.fc2 = nn.Linear(32, 16)
        self.fc3 = nn.Linear(16, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x)

model = SimpleInsuranceNN(X_train_tensor.shape[1])

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

entrenamos el modelo, en este caso lo entrenamos por 1000 epocas, cada 10 epocas imprimimos el loss para ver como va mejorando el modelo

[77]:
```
epochs = 100
for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    loss.backward()
    optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.1f}')
```

```
Epoch [10/100], Loss: 322370816.0
Epoch [20/100], Loss: 322090688.0
Epoch [30/100], Loss: 321328544.0
Epoch [40/100], Loss: 319632960.0
Epoch [50/100], Loss: 316373088.0
Epoch [60/100], Loss: 310778816.0
Epoch [70/100], Loss: 302017696.0
Epoch [80/100], Loss: 289305248.0
Epoch [90/100], Loss: 272045472.0
Epoch [100/100], Loss: 250019872.0
```

evaluamos el modelo con los datos de prueba, podemos ver que la perdida es muy alta por lo que el modelo no es muy bueno, hay que recordar que las redes neuronales no son muy buenas en datos tabulares, ademas de esto el dataset es muy pequeño por lo que no hay muchos datos para entrenar el modelo.

[78]:
```
model.eval()
with torch.no_grad():
    test_outputs = model(X_test_tensor)
    test_loss = criterion(test_outputs, y_test_tensor)
    print(f'Test Loss: {test_loss.item():.1f}')
```

```
Test Loss: 245675632.0
```

El test loss representa el error cuadrado promedio entre las predicciones y los valores reales podemos ver que este es muy alto, por lo que el modelo no es muy bueno.

Aqui tenemos predicciones para 3 elementos del conjunto de prueba, podemos ver que las predicciones son muy diferentes a los valores reales, y que el modelo no es bueno para predecir el costo del seguro.

[79]:
```
model(torch.tensor(X_test_tensor[:3], dtype=torch.float32))
```

[79]:
```
tensor([[1442.8323],
        [1333.4028],
        [4724.2036]], grad_fn=<AddmmBackward0>)
```

[80]:
```
y_test_tensor[:3]
```

[80]:
```
tensor([[ 9095.0684],
        [ 5272.1758],
        [29330.9824]])
```

para hacer un intento mas con redes neuronales modificaremos los datos para que la variable de salida sea un problema de clasificacion.

para hacer esto lo que haremos es dividir el costo del seguro en 3 categorias, bajo, medio y alto, para esto usaremos los cuartiles de los datos, los datos que esten en el primer cuartil seran bajo (0), los que esten en el segundo cuartil seran medio (1) y los que esten en el tercer cuartil seran alto (2).

[81]:
```
Q1 = df['charges'].quantile(0.25)
Q3 = df['charges'].quantile(0.75)

df['category'] = 1
df.loc[df['charges'] < Q1, 'category'] = 0
df.loc[df['charges'] > Q3, 'category'] = 2
```

hacemos los mismos procesos que antes, solo que ahora la variable de salida es la categoria.

[82]:
```
X = df.drop(columns=['charges', 'category'])
y = df['category']
```

[83]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[84]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[85]:
```
X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.long).view(-1)
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.long).view(-1)
```

Este nuevo modelo es de clasificacion que a diferencia del pasado que era de regresion su proposito es categorizar los datos de entrada en una de las 3 categorias que definimos anteriormente.

[86]:
```
class ClassificationNN(nn.Module):
    def __init__(self, input_dim):
        super(ClassificationNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, 32)
        self.fc2 = nn.Linear(32, 16)
        self.fc3 = nn.Linear(16, 3)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return F.softmax(self.fc3(x), dim=1)

model = ClassificationNN(X_train_tensor.shape[1])

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

[87]:
```
epochs = 100
for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    loss.backward()
    optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')
```

```
Epoch [10/100], Loss: 0.9585
Epoch [20/100], Loss: 0.7667
Epoch [30/100], Loss: 0.6707
Epoch [40/100], Loss: 0.6552
Epoch [50/100], Loss: 0.6492
Epoch [60/100], Loss: 0.6466
Epoch [70/100], Loss: 0.6451
```

```
Epoch [80/100], Loss: 0.6441
Epoch [90/100], Loss: 0.6434
Epoch [100/100], Loss: 0.6429
```

[88]:
```
model.eval()
with torch.no_grad():
    test_outputs = model(X_test_tensor)
    _, predicted = torch.max(test_outputs, 1)
    accuracy = (predicted == y_test_tensor).sum().item() / y_test_tensor.size(0)
    print(f'Accuracy: {accuracy*100:.2f}%')
```

```
Accuracy: 90.30%
```

Aqui podemos ver los resultados de la clasificacion de los seguros, en la que en un 90.3% de los casos el modelo clasifico correctamente el seguro.

En esta parte podemos ver las probabilidades que que las primeras 3 pruebas esten en cada una de las categorias.

[89]:
```
model(torch.tensor(X_test_tensor[:3], dtype=torch.float32))
```

[89]:
```
tensor([[2.1184e-12, 1.0000e+00, 1.8245e-06],
        [1.0079e-01, 8.9821e-01, 1.0006e-03],
        [6.4255e-17, 4.3202e-03, 9.9568e-01]], grad_fn=<SoftmaxBackward0>)
```

Aqui podemos ver para las primeras 3 pruebas a que categoria se estan clasificando y cual es la probabilidad de que esten en ella.

[90]:
```
torch.max(model(torch.tensor(X_test_tensor[:3], dtype=torch.float32)), 1)
```

[90]:
```
torch.return_types.max(
values=tensor([1.0000, 0.8982, 0.9957], grad_fn=<MaxBackward0>),
indices=tensor([1, 1, 2]))
```

Aqui podemos ver las categorias reales de las primeras 3 pruebas.

[91]:
```
y_test_tensor[:3]
```

[91]:
```
tensor([1, 1, 2])
```

En esta ultima prueba podemos cer que el modelo tiene una precision buena ahora que solo estamos clasificando si el seguro estara en una de 3 categorias.