

질병예측 머신

팀번호 : 12

과목명 : 오픈소스AI응용 1분반

교수명 : 김동재 교수님

과제명 : 기말대체텀프 발표

문제 제시

현대 사회에서 건강에 대한 염려는 지속적으로 증가하고 있는 상황인데, 정보들이 지나치게 흩어져 있으며 정제되어 있지 않은 환경에서 필요한 정보를 얻기보다 혼란스러움만 얻는 경험을 할 수 있었습니다.



예상되는 문제

데이터 수집

충분한 양의 질병과 증상들을 확보할 수 있는 데이터를 찾는 것이 쉽지 않을 것 같습니다.

데이터의 신뢰도

신뢰할 수 있는 의료용 데이터인지 판단하는 과정이 필요합니다.

질병과의 관련성

질병을 예측하는데 영향을 줄 수 있는 변수들인지 파악해야 합니다.

진행 순서



데이터 수집

증상과 나이, 성별과 같은 변수들로 질병을 예측하는 머신을 학습시키기에 적절한 데이터를 찾아야 합니다.



데이터 전처리

확보한 데이터를 효과적으로 활용하기 위해서 전처리를 해야 합니다.



모델, 신경망구조 작성

분류 문제에 맞는 모델과 활성화 함수, 신경망 구조 등을 작성해 줘야 합니다.



분석, 문제점 발견

과제를 통해서 유의미한 결과를 얻었는지 분석해보고 진행 중에 어떤 문제점들이 있었는지, 개선해야 할 부분은 있는지에 대해 분석해야 합니다.

데이터 소개

target

feature

증상

증상의 강도

나이

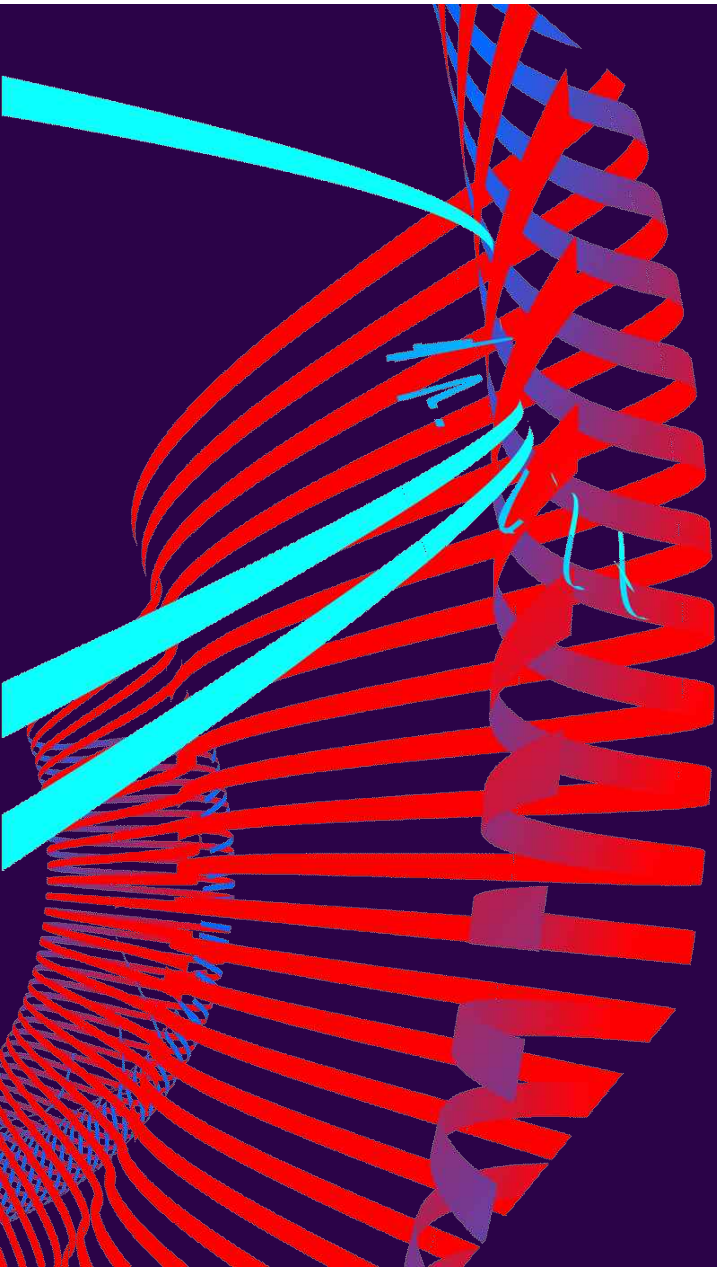
성별

BMI

계절

	A	B	C	D	E	F	G	H				K	L	M
1	Disease	Disease_C	Symptom	Symptom_	Weight	Height	Intensity	Severity	Age	Gender	BMI_Leve	Region	Season	
2	influenza	C0162565	uncoordin	C0039239	68	180	high	medium	24	female	27.9	southwest	Summer	
3	influenza	C0162565	fever	C0000737	68	170	low	medium	23	male	33.77	southeast	Summer	
4	influenza	C0162565	pleuritic p	C0235704	68	162	low	low	24	male	33	southeast	Summer	
5	influenza	C0162565	snuffle	C0030554	68	162	high	medium	34	male	22.705	northwest	Summer	
6	influenza	C0162565	throat sore	C0030552	68	185	low	high	21	male	28.88	northwest	Winter	
7	influenza	C0162565	malaise	C0020538	68	185	medium	medium	21	female	25.74	southeast	Winter	
8	influenza	C0162565	debilitation	C0020555	68	185	medium	medium	25	female	33.44	southeast	Winter	

Datasetfinal.csv



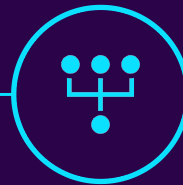
코드 설명

핵심 코드



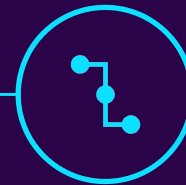
전처리

데이터가 스트링이기 때문에
Numerical Data로 변환해주는
전처리를 합니다.



MLP

CNN이나 RNN을 사용하기에는
부적합해 보여서 MLP를 활용하
려고 합니다.



신경망 구조

레이어를 세 개 넣으려고 합니
다.



전처리

LabelEncoder

Numerical Data로 변환해서 학습하려고 하기 때문에 disease, symptom, intensity, age, gender, bmi, season에 각각 정수값 라벨링을 해줍니다.

```
▶ from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

encoder.fit(df_die)
train_die = pd.Series(encoder.transform(df_die))

encoder.fit(df_sym)
train_sym = pd.Series(encoder.transform(df_sym))

encoder.fit(df_int)
train_int = pd.Series(encoder.transform(df_int))

train_age = df_age

encoder.fit(df_gen)
train_gen = pd.Series(encoder.transform(df_gen))

train_bmi = df_bmi

encoder.fit(df_sea)
train_sea = pd.Series(encoder.transform(df_sea))
```


NORMALIZATION

```
▶ from sklearn.preprocessing import MinMaxScaler  
minmax_scaler = MinMaxScaler()  
  
x_train = minmax_scaler.fit_transform(x_train)  
x_test = minmax_scaler.fit_transform(x_test)
```

데이터 값을 0에서 1사이의 값
으로 fitting 해주는 노말라이제
이션도 중요합니다.

MLP

세 개의 레이어

세 개의 레이어로 구성하려고 합니다.

첫 번째 레이어는 input을 6개의 디멘션으로 합니다.

출력 노드 수는 32개이고 따라서 32채널로 출력이 됩니다.

두 번째 레이어는 input을 32개로 받고 16개를 출력하게 됩니다.

세 번째 레이어는 16개를 입력 받고 질병의 개수인 148개를 출력하게 됩니다.

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, output_size):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, 32)
        self.fc2 = nn.Linear(32, 16)
        self.fc3 = nn.Linear(16, output_size)

    def forward(self, x):
        out_fc1 = F.relu(self.fc1(x))
        # out_ac1 = F.relu(self.activation(out_fc1))
        out_fc2 = F.relu(self.fc2(out_fc1))
        # out_ac2 = self.activation(out_fc2)
        out_fc3 = F.softmax(self.fc3(out_fc2))
        return out_fc3
```

렐루와 소프트맥스를 사용했는데 마지막 레이어에 소프트맥스를 사용한 이유는 각 클래스별 확률을 출력해주기 때문입니다.

```
# Define Optimizer: Stochastic Gradient Descent method  
loss_fn = nn.CrossEntropyLoss()  
# optimizer = torch.optim.SGD(model.parameters(), lr=0.05)  
optimizer = torch.optim.SGD(model.parameters(), lr=0.05, momentum=0.9)  
# optimizer = torch.optim.Adam(model.parameters(), lr=0.05)
```

Loss function은 CrossEntropyLoss를
optimizer는 SGD를 사용했습니다.

학습

```
# Define loss function (Cross Entropy Loss here)
# Train the model
total_size = len(train_data_loader)
epochs = 100
for epoch in range(epochs):
    for i, (images, labels) in enumerate(train_data_loader): # min batch for 100

        # Upload to gpu
        images = images.to(device)
        labels = labels.to(device)

        # print : images shape

        # Forward pass
        outputs = model(images) # Forward: Prediction
        # labels = labels.squeeze(dim=1)
        loss = loss_fn(outputs, labels) # Calculate the loss w/ th the pred class labels (outputs) & ground-truth labels (labels)

        # Backward pass & Optimize
        optimizer.zero_grad()
        loss.backward() # Calculate a gradient calculation (auto grad)
        optimizer.step() # Update model's parameter with requires_grad=True

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

결과

만족스러운 결과가 나오지는
않았습니다...

아무래도 데이터의 숫자가 적은
편이기 때문에 로스가 떨어
지지 않는 것과 제대로 학습되
지 않고 있다는 것을 확인할 수
있었습니다.

그리고 비정형 데이터가 많은
것도 요인중에 있는 것 같습니
다.

```

▶ Epoch [82/100], Step [600/1596], Loss: 5.5773
▶ Epoch [82/100], Step [700/1596], Loss: 5.4021
▶ Epoch [82/100], Step [800/1596], Loss: 5.2790
▶ Epoch [82/100], Step [900/1596], Loss: 4.4092
▶ Epoch [82/100], Step [1000/1596], Loss: 4.3961
▶ Epoch [82/100], Step [1100/1596], Loss: 4.6453
▶ Epoch [82/100], Step [1200/1596], Loss: 4.6633
▶ Epoch [82/100], Step [1300/1596], Loss: 5.5394
▶ Epoch [82/100], Step [1400/1596], Loss: 4.5442
▶ Epoch [82/100], Step [1500/1596], Loss: 4.4099
▶ Epoch [83/100], Step [100/1596], Loss: 4.5644
▶ Epoch [83/100], Step [200/1596], Loss: 4.4099
▶ Epoch [83/100], Step [300/1596], Loss: 4.6089
▶ Epoch [83/100], Step [400/1596], Loss: 4.5824
▶ Epoch [83/100], Step [500/1596], Loss: 4.5917
▶ Epoch [83/100], Step [600/1596], Loss: 4.5582
▶ Epoch [83/100], Step [700/1596], Loss: 5.5781
▶ Epoch [83/100], Step [800/1596], Loss: 4.5388
▶ Epoch [83/100], Step [900/1596], Loss: 5.6447
▶ Epoch [83/100], Step [1000/1596], Loss: 4.4022
▶ Epoch [83/100], Step [1100/1596], Loss: 4.5851
▶ Epoch [83/100], Step [1200/1596], Loss: 5.5299
▶ Epoch [83/100], Step [1300/1596], Loss: 4.6109
▶ Epoch [83/100], Step [1400/1596], Loss: 4.6128
▶ Epoch [83/100], Step [1500/1596], Loss: 4.5310
▶ Epoch [84/100], Step [100/1596], Loss: 4.6494
▶ Epoch [84/100], Step [200/1596], Loss: 4.6498
▶ Epoch [84/100], Step [300/1596], Loss: 5.5397
▶ Epoch [84/100], Step [400/1596], Loss: 4.6103

```


개선해야 할 문제점

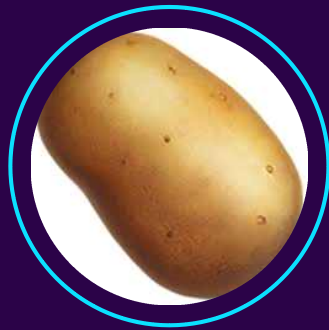
데이터 양 부족

데이터의 절대적인 양이 적어서 제대로 학습이 되지 않았다는 문제가 있습니다.

지역에 의한 차이

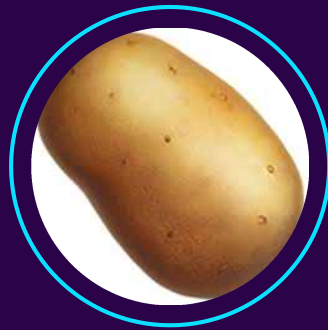
지역에 따라서 걸리기 쉬운 질병과 어려운 질병이 상이한 차이를 보일 수 있으나 데이터의 부족으로 지역마다의 증상과 질병의 관련성을 입증하지 못했습니다.

팀 소개



32207825 정재호

데이터 수집
데이터 전처리
코딩



32207265 김형주

데이터 수집



???????? 교수님

마법



감사합니다.

부족한 내용이지만 들어주셔서 감사합니다.