호흡기 증상 및 치료

1조 우상민

CONTENTS

- 주제 선정 및 설명
- 데이터 분석
- 분석 과정
- 결과 및 소감

주제 선정

주제 내용

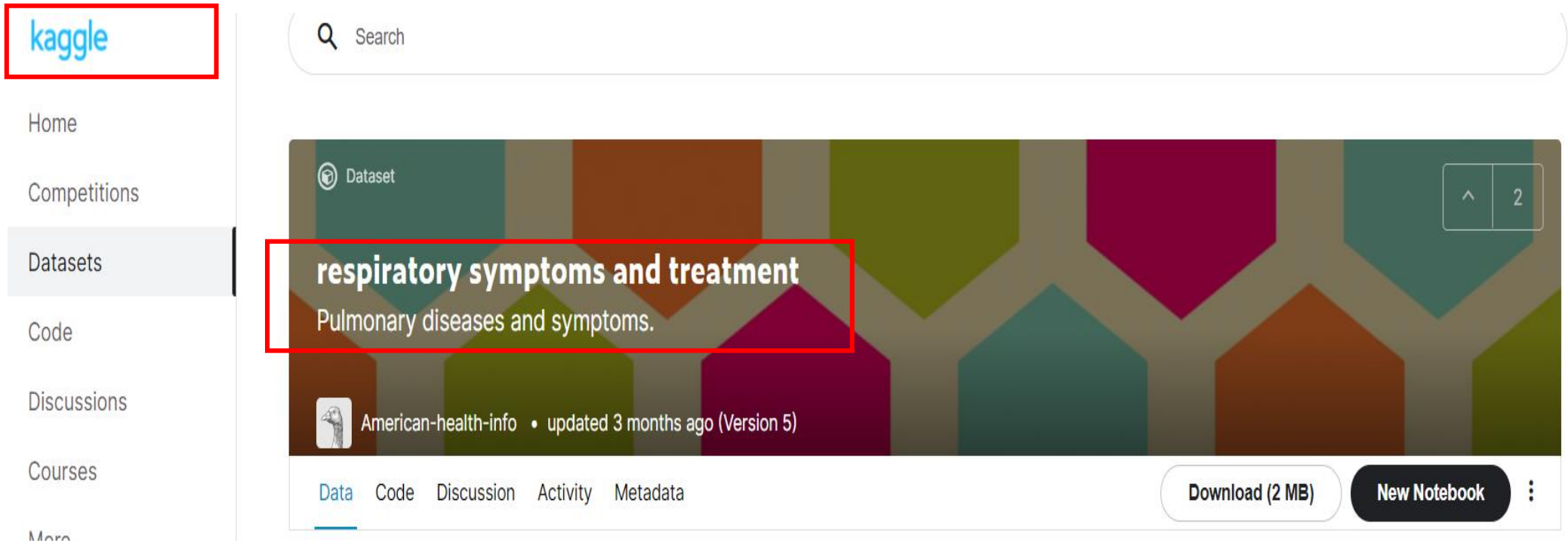
- 호흡기 질환은 엄청난 건강 부담을 유발
- 전 세계적으로 천식 환자는 2억 3천명, 폐질환(copd) 환자는 2억명 이상,

  수면 장애 환자는 1억 명 이상, 만성 호흡기 환자 수는 10억 명
- 미세먼지 때문에 한국에서도 많은 사람들이 호흡기 질환을 겪음

문제 정의

증상, 나이, 성별, 질병을 토대로 올바른 처방이 가능할까?

데이터 수집



※뉴 멕시코주의 공중 보건국 데이터

데이터 분석

데이터 형식

- 6개 컬럼
- 38,538개 데이터로 구성
- Symptoms , Age ,Sex, Disease, Treatment , Nature 총 6개 컬럼으로 구성

Symptoms	Age	Sex	Disease	Treatment	Nature
coughing	5	female	Asthma	Omalizuma	high
tight feelir	4	female	Asthma	Mepolizun	high
wheezing	6	male	Asthma	Mepolizun	high
shortness	7	male	Asthma	Mepolizun	high
shortness	9	male	Asthma	Mepolizun	high
tight feeling in the ch		male	Asthma	Mepolizun	high
shortness of breath		male	Asthma	Mepolizun	high
tight feelir	8	female	Asthma	Mepolizun	high
shortness	36	female	Asthma	Mepolizun	medium
wheezing	40	female	Asthma	Omalizuma	medium
coughing	46	female	Asthma	Omalizuma	medium
shortness	56	female	Asthma	Omalizuma	medium
shortness	58	female	Asthma	Omalizuma	medium
shortness	67	female	Asthma	Omalizuma	high
coughing	87	male	Asthma	Omalizuma	high
shortness	65	male	Asthma	Mepolizun	high
tight feelir	54	female	Asthma	Mepolizun	medium

```
1 df = pd.read_csv('/content/호흡기완성본.csv') # 파일을 불러오고
2 df.head()
```

	Symptoms	Age	Sex	Disease	Treatment	Nature
0	coughing	5.0	female	Asthma	Omalizumab	high
1	tight feeling in the chest	4.0	female	Asthma	Mepolizumab	high
2	wheezing	6.0	male	Asthma	Mepolizumab	high
3	shortness of breath	7.0	male	Asthma	Mepolizumab	high
4	shortness of breath	9.0	male	Asthma	Mepolizumab	high

데이터 전처리

- 각 컬럼에 있는 데이터 null을 제거 -> index 34409 일치 확인
- 필요 없는 컬럼 Nature 삭제
- 성별 코드 안에 not to say '말하지 않겠다' 라는 불필요한 값 삭제

```
df.dropna(subset=['Symptoms','Age','Sex','Disease','Treatment'], axis =0, inplace=True)
```

```
df.info() # 정보 확인
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 34409 entries, 0 to 38536
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Symptoms   34409 non-null  object
 1   Age        34409 non-null  float64
 2   Sex        34409 non-null  object
 3   Disease    34409 non-null  object
 4   Treatment  34409 non-null  object
 5   Nature     32724 non-null  object
dtypes: float64(1), object(5)
memory usage: 1.8+ MB
```

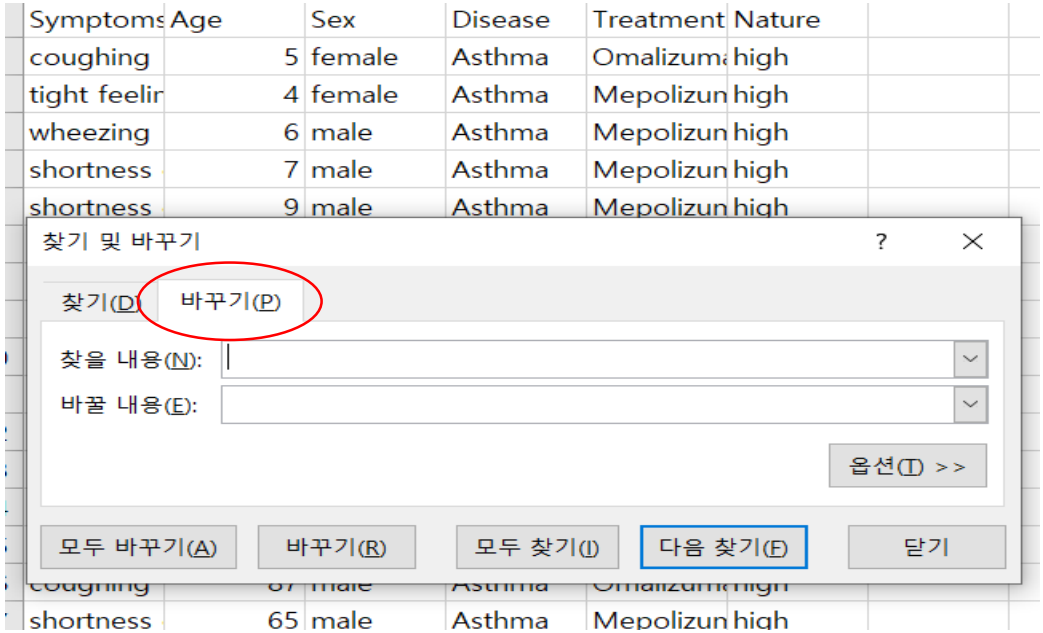
```
df.drop(['Nature'], axis = 1,inplace= True)
df.info()
```

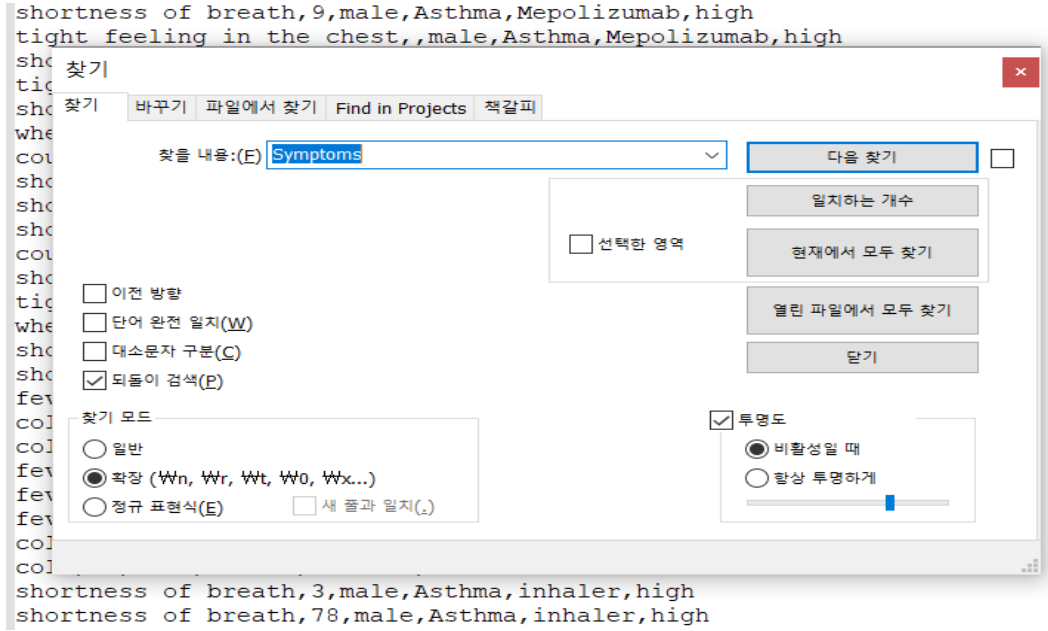
```
df['Sex'].replace('not to say',np.nan, inplace =True)
df.head()
```

```
df.dropna(subset=['Sex'], axis =0, inplace=True)
df.info()
```

데이터 전처리

-대문자 및 중복된 값은 엑셀에서 전처리
-전처리 작업 중 \n 줄 바꿈 형식이 수정 되지 않아서 notepad에서 수정





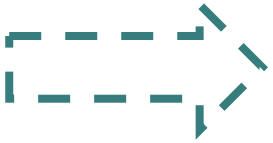
데이터 분석

One-Hot Encoding을 위해 Dummies 함수 사용

```
1 onehot_sex = pd.get_dummies(df[['Sex']])
2 onehot_sex
```

	Sex_female	Sex_male
0	1	0
1	1	0
2	0	1

```
1 onehot_symptoms = pd.get_dummies(df[['Symptoms']])
2 onehot_symptoms.info()
3
```



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33749 entries, 0 to 38536
Data columns (total 82 columns):
```

```
1 onehot_dummies = pd.get_dummies(df[['Disease']])
2 onehot_dummies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33749 entries, 0 to 38536
Data columns (total 17 columns):
```

데이터 분석

- **One-Hot Encoding 으로 변환 한 컬럼들을 데이터프레임에 추가**

```
df = pd.concat([df, onehot_sex], axis = 1)
df.head()
```

```
1 df = pd.concat([df, onehot_symptoms], axis = 1)
2
```

```
1 df = pd.concat([df, onehot_dummies], axis = 1)
2 df.head()
```

	Symptoms	Age	Sex	Disease	Treatment	Sex_female	Sex_male	Symptoms_A cough that lasts more than three weeks	Symptoms_A dry, crackling sound in the lungs while breathing in	Symptoms_Bluish skin	Symptoms_Chest congestion	Symptoms_Chest pain	Symptoms_Chills
0	coughing	5.0	female	Asthma	Omalizumab	1	0	0	0	0	0	0	0
1	tight feeling in the chest	4.0	female	Asthma	Mepolizumab	1	0	0	0	0	0	0	0
2	wheezing	6.0	male	Asthma	Mepolizumab	0	1	0	0	0	0	0	0
3	shortness of breath	7.0	male	Asthma	Mepolizumab	0	1	0	0	0	0	0	0
4	shortness of breath	9.0	male	Asthma	Mepolizumab	0	1	0	0	0	0	0	0

5 rows × 106 columns

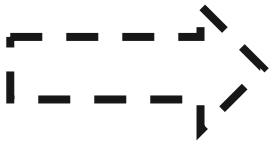
데이터 분석

- label encoder 사용하여 문자를 숫자 형식으로 변환
- Categorical 사용하여 숫자 값을 범주형 형태로 변환

```
from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder()

onehot_location = label_encoder.fit_transform(df['Treatment'])

```

```
df['y_label'] = onehot_location
```

```
from tensorflow.keras.utils import  to_categorical

#각 데이터의 레이블 0~32 숫자 값을 범주형 형태로 변경

y = to_categorical(y)
y
```



```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

데이터 분석 결과

```
1 log_reg =LogisticRegression().fit(X_train, y_train)
```

```
1 #정확도 확인
2 print('Train data score:', log_reg.score(X_train,y_train))
3 print('test data score:', log_reg.score(X_test,y_test))
```

```
Train data score: 0.77167287504232 98
test data score: 0.7746172839506172
```

```
tree = DecisionTreeClassifier(max_depth = 5,
                              min_samples_leaf = 20,
                              min_samples_split = 40).fit(X_train, y_train)
```

```
3 print('Train data score:', tree.score(X_train,y_train))
4 print('test data score:', tree.score(X_test,y_test))
```

```
Train data score: 0.5165932949542837
test data score: 0.5217777777777778
```

데이터 분석 결과

```
boost = GradientBoostingClassifier(max_depth=3, #트리의 갯수
                                   learning_rate =0.05).fit(X_train, y_train)
```

```
#정확도 확인
print('Train data score:', boost.score(X_train,y_train))
print('test data score:', boost.score(X_test,y_test))
```

```
Train data score: 0.9824331188621741
test data score: 0.9816296296296296
```

```
random = RandomForestClassifier(n_estimators=300,
                                random_state=0).fit(X_train,y_train)
```

```
#정확도 확인
print('Train accuracy score is:', random.score(X_train,y_train))
print('test accuracy score is:', random.score(X_test,y_test))
```

```
Train accuracy score is: 0.9910684050118523
test accuracy score is: 0.9899259259259259
```

데이터 분석

```
X= df2[['Age','Sex_female','Sex_male','Symptoms_wheezing','Symptoms_fatigue','Symptoms_shortness of breath','Symptoms_coughing','Sympto
y = df2['y_label']
```

```
from sklearn import preprocessing

X = preprocessing.StandardScaler().fit(X).transform(X)
```

```
from sklearn.model_selection import  train_test_split

X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.33, random_state=777)
```

```
#훈련 데이터
print(X_train.shape,y_train.shape)
print(y_train)

#테스트 데이터
print(X_test.shape, y_test.shape)
print(y_test)
```

```
(23624, 112) (23624, 33)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]]
(10125, 112) (10125, 33)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

데이터 분석

```
from sklearn.model_selection import  train_test_split

X_train, X_val, y_train, y_val = train_test_split(X_train,          # 독립 변수
                                                  y_train,          # 종속변수
                                                  test_size=0.33,   # 검증 33%
                                                  random_state=777)  # 랜덤 추출 값
```

```
print(f'훈련 데이터 {X_train.shape} 레이블 {y_train.shape}')
print(f'검증 데이터 {X_val.shape} 레이블 {y_val.shape}')
```

```
훈련 데이터 (15828, 112) 레이블 (15828, 33)
검증 데이터 (7796, 112) 레이블 (7796, 33)
```

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print(X_val.shape, y_val.shape)
print(X_train.shape, X_val.shape)

```

```
(15149, 112) (15149, 33)
(11138, 112) (11138, 33)
(7462, 112) (7462, 33)
(15149, 112) (7462, 112)
```

데이터 분석

- 다중 분류에서 가장 확률이 높은 하나 값만 가져오기 위해 sortmax 사용
- Metrics = acc (정확도)로 설정
- Epochs = 학습을 100번 반복

```
1 from keras.models import Sequential
2 from keras.layers import Dense
3
4 model = Sequential()
5 model.add(Dense(100, activation = 'relu', input_shape = (112, )))
6 model.add(Dense(50, activation = 'relu'))
7 model.add(Dense(33,activation = 'softmax'))
8
```

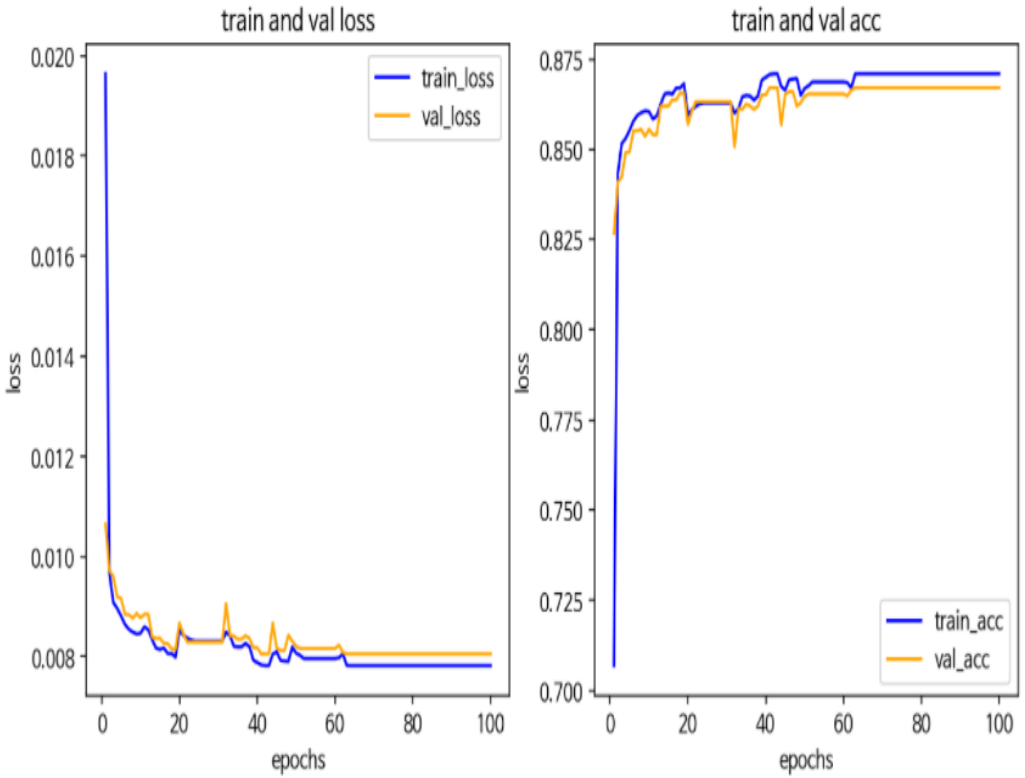
```
1 model.compile(loss='mae',
2               optimizer = 'adam',
3               metrics =['acc'] ) # 정확도
```

```
1 history = model.fit(X_train,y_train,
2                     epochs = 100,
3                     validation_data =(X_val,y_val))
```

데이터 분석 결과 확인

```
Epoch 1/100
474/474 [==============================] - 15s 4ms/step - loss: 0.0319 - acc: 0.5387 - val_loss: 0.0107 - val_acc: 0.8267
Epoch 2/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0100 - acc: 0.8386 - val_loss: 0.0097 - val_acc: 0.8409
Epoch 3/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0092 - acc: 0.8491 - val_loss: 0.0096 - val_acc: 0.8423
Epoch 4/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0089 - acc: 0.8544 - val_loss: 0.0092 - val_acc: 0.8491
Epoch 5/100
474/474 [==============================] - 1s 3ms/step - loss: 0.0088 - acc: 0.8562 - val_loss: 0.0092 - val_acc: 0.8492
Epoch 6/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0089 - acc: 0.8541 - val_loss: 0.0088 - val_acc: 0.8551
Epoch 7/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0086 - acc: 0.8584 - val_loss: 0.0088 - val_acc: 0.8551
Epoch 8/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0085 - acc: 0.8604 - val_loss: 0.0088 - val_acc: 0.8555
Epoch 9/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0089 - acc: 0.8541 - val_loss: 0.0089 - val_acc: 0.8535
Epoch 10/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0082 - acc: 0.8652 - val_loss: 0.0088 - val_acc: 0.8555
Epoch 11/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0083 - acc: 0.8639 - val_loss: 0.0088 - val_acc: 0.8541
Epoch 12/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0084 - acc: 0.8614 - val_loss: 0.0088 - val_acc: 0.8541
Epoch 13/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0081 - acc: 0.8658 - val_loss: 0.0084 - val_acc: 0.8620
Epoch 14/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0086 - acc: 0.8581 - val_loss: 0.0084 - val_acc: 0.8620
```

```
Epoch 86/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0078 - acc: 0.8717 - val_loss: 0.0081 - val_acc: 0.8671
Epoch 87/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0078 - acc: 0.8721 - val_loss: 0.0081 - val_acc: 0.8671
Epoch 88/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0076 - acc: 0.8747 - val_loss: 0.0081 - val_acc: 0.8671
Epoch 89/100
474/474 [==============================] - 1s 3ms/step - loss: 0.0083 - acc: 0.8632 - val_loss: 0.0081 - val_acc: 0.8671
Epoch 90/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0077 - acc: 0.8736 - val_loss: 0.0081 - val_acc: 0.8671
Epoch 91/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0078 - acc: 0.8715 - val_loss: 0.0081 - val_acc: 0.8671
Epoch 92/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0077 - acc: 0.8722 - val_loss: 0.0081 - val_acc: 0.8671
Epoch 93/100
474/474 [==============================] - 1s 3ms/step - loss: 0.0081 - acc: 0.8660 - val_loss: 0.0081 - val_acc: 0.8671
Epoch 94/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0075 - acc: 0.8769 - val_loss: 0.0081 - val_acc: 0.8671
Epoch 95/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0078 - acc: 0.8706 - val_loss: 0.0081 - val_acc: 0.8671
Epoch 96/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0078 - acc: 0.8715 - val_loss: 0.0081 - val_acc: 0.8671
Epoch 97/100
474/474 [==============================] - 1s 2ms/step - loss: 0.0080 - acc: 0.8679 - val_loss: 0.0081 - val_acc: 0.8671
Epoch 98/100
```



혼동 행렬

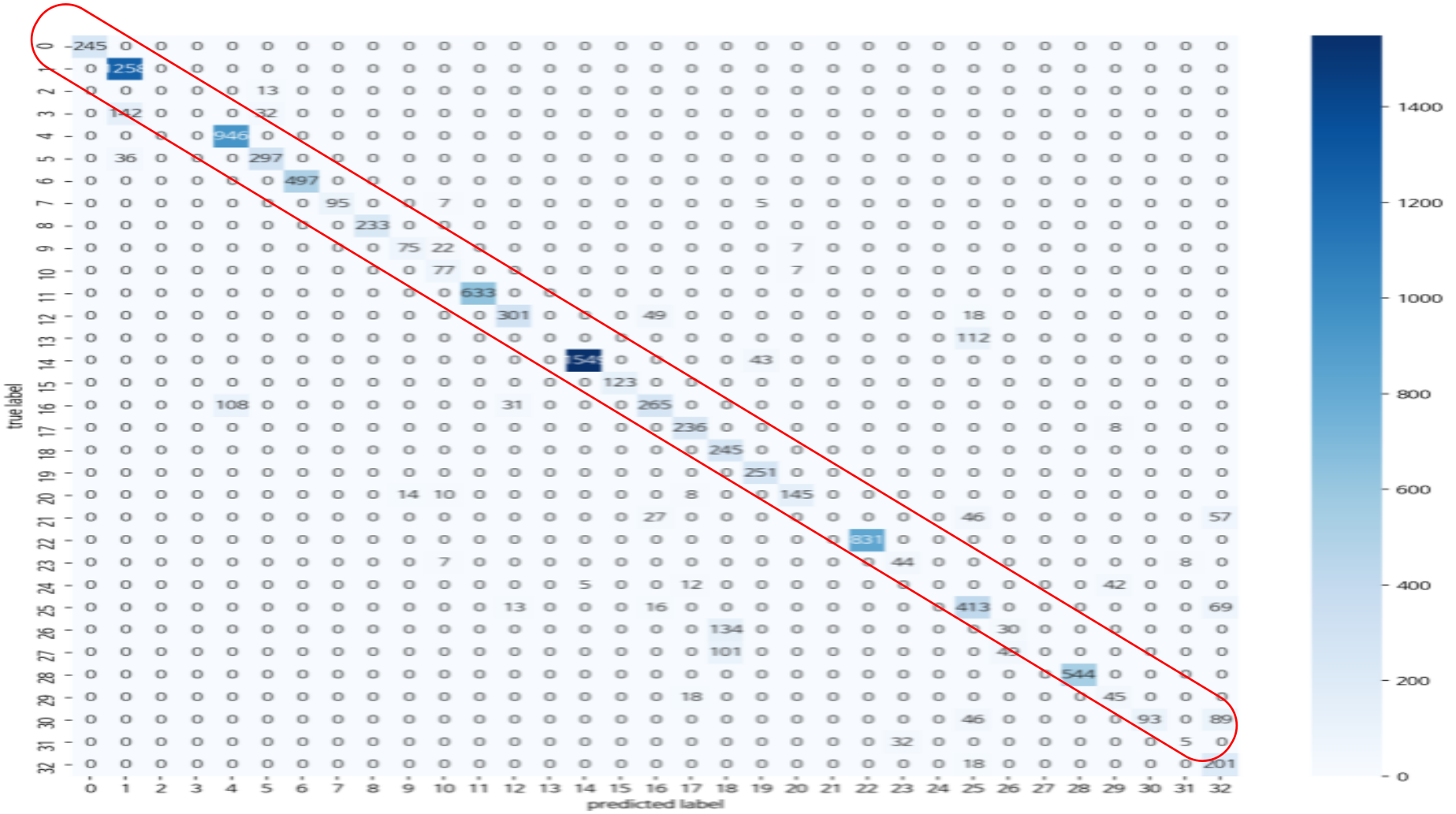
-33개 레이블의 예측 확인이 어려움
-혼동 행렬을 그려서 각 레이블을 어떻게 예측하는지 확인

```python
import numpy as np
np.set_printoptions(suppress=True)

results = model.predict(X_test)
print(results.shape)
np.set_printoptions(precision=7)
print(f'각 클래스에 속할 확률 : \n{results[0]}')
```

```
(11138, 33)
각 클래스에 속할 확률 :
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```python
# sklearn.metrics 모듈은 여러가지 평가 지표에 관한 기능을 제공합니다.
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# 혼동 행렬을 만듭니다.
plt.figure(figsize = (15, 15))
cm = confusion_matrix(np.argmax(y_test, axis = -1), np.argmax(results, axis = -1))
sns.heatmap(cm, annot = True, fmt = 'd',cmap = 'Blues')
plt.xlabel('predicted label')
plt.ylabel('true label')
plt.show()

```

혼동 행렬 그래프



혼동 행렬

```
1 print('\n', classification_report(np.argmax(y_test, axis = -1), np.argmax(results, axis = -1)))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	245
1	0.88	1.00	0.93	1258
2	0.00	0.00	0.00	13
3	0.00	0.00	0.00	174
4	0.90	1.00	0.95	946
5	0.87	0.89	0.88	333
6	1.00	1.00	1.00	497
7	1.00	0.89	0.94	107
8	1.00	1.00	1.00	233
9	0.84	0.72	0.78	104
10	0.63	0.92	0.74	84
11	1.00	1.00	1.00	633
12	0.87	0.82	0.84	368
13	0.00	0.00	0.00	112
14	1.00	0.97	0.98	1592
15	1.00	1.00	1.00	123
16	0.74	0.66	0.70	404
17	0.86	0.97	0.91	244
18	0.51	1.00	0.68	245
19	0.84	1.00	0.91	251
20	0.91	0.82	0.86	177
21	0.00	0.00	0.00	130
22	1.00	1.00	1.00	831
23	0.58	0.75	0.65	59
24	0.00	0.00	0.00	59
25	0.63	0.81	0.71	511
26	0.38	0.18	0.25	164
27	0.00	0.00	0.00	150
28	1.00	1.00	1.00	544
29	0.47	0.71	0.57	63
30	1.00	0.41	0.58	228
31	0.38	0.14	0.20	37
32	0.48	0.92	0.63	219
accuracy			0.87	11138
macro avg	0.66	0.68	0.66	11138
weighted avg	0.84	0.87	0.84	11138

Antibiotic	1	항생제
Antibiotics	2	항생제
Antibiotics.	3	항생제.

consult a doctor	16	의사와 상담하다
consult doctor	17	의사의 진찰을 받다

inhaler	20	흡입기
inhealer	21	치료 중인

oxygen	24	산소
oxyzen	25	산소의

시사점

“군 병원 오진 다리 못 쓸 뻔했다”

구본호 | 입력 2021.03.09 | 5면 | 댓글 0

병원 오진으로 10년간 누워지낸 환자..약 바꾸고 일주일만에 일어나

- 오진으로 인해 환자들이나 일반 시민들이 피해를 입는 상황이 발생
- 환자의 새로운 데이터들이 계속 수집하고 분석 할 수 있다면 오진 및 약 처방 오진을 줄일 수 있음

환자들의 피해 감소

느낀점 및 향후 발전 방향

1. 머신러닝 모델에 직접 수집한 데이터를 적용하여 학습하고 분석하는 과정이 흥미로웠음

2. 데이터를 수집의 한계로 아이디어 구현의 어려움

3. 오류 발생 극복의 어려움

더욱 내 자신을 발전시키자!

감사합니다