Python을 이용한 강수 예측 프로그램

윤유상(대구과학고 1년, fyoon46@gmail.com)
정우용(대구과학고 1년, vrokoli418@gmail.com)
심훈(대구과학고 1년, hun030819@gmail.com)

Ⅰ. 프로그램 개요

1. Import

가. NumPy

NumPy는 'Numerical Python'의 약자로 Python에서 제공하는 Linear Algebra Library 중 하나이다. 대부분의 library가 NumPy에 기반을 둘 정도로 NumPy는 Python에서 중요한 library이다. NumPy의 경우 'import numpy as np'로 작성하여 NumPy의 함수들을 사용하는 것이 가장 일반적이다.

나. Pandas

1) read_csv
CSV 파일을 읽기 위해 불러온다.
2) get_dummies
'Yes' 또는 'No'로 된 데이터를 0과 1로 변환한다.

다. Seaborn

Seaborn은 데이터를 분석하기 위한 함수들을 포함하는 library로, 표와 그래프를 다룰 때 유용하게 쓰이는 기능을 포함한다. 이들은 ( )plot의 형태를 띠는 함수들로 이루어진다. Seaborn을 import 할 때는 'import seaborn as sns'를 입력한다. 이때, '%matplotlib inline'를 함께 실행시키면 그래프에 있어서 더 좋은 성능을 발휘할 수 있다.
아래에서 x = (x축에 대입할 항목), y = (y축에 대입할 항목), data = (데이터베이스), palette = (그래프에 사용할 색)을 대입하면 된다.

1) Pairplot
데이터의 각 항목 간의 관계를 그래프로 나타내 분석할 수 있게 해준다.
2) Heatmap: sns.heatmap(data)
데이터의 선택 항목별 관련성 정도를 표시하는 데에 쓰인다. 때로는 데이터의 비어있는 데이터 값이 얼마나 있는지 알아보기 위해서도 쓰인다.

라. Sklearn

1) train_test_split

데이터를 train set과 test set로 나누어 Logistic Regression을 위한 준비를 한다.

2) LogisticRegression

주어지는 x값을 통해 y값을 예측하는 로지스틱 회귀 모델을 만든다.

3) metrics

모델을 평가하는데 쓰인다.

2. 프로그램 소개

가. 프로그램 주제

부정확한 일기예보를 개선하고자 하는 뜻에서 PIREP 수업 중 배운 내용을 활용해서 직접 정확한 날씨 예측 프로그램을 만들고자 하였다. 우리는 오늘의 날씨 정보를 이용해 다음 날의 강우 여부를 예측하는 모델을 만들고 평가하여 그 정확도를 향상하였다.

나. Logistic Regression Model

강우 여부는 'Yes'와 'No'라는 두 가지 상태만으로 구성되므로 0과1로 결과값이 나오는 Logistic Regression Model이 적합하다고 판단하여 사용하였다.

다. Neural Network Model

Logistic Regression Model에서는 분석하기 어려운 다양한 타입의 데이터를 다룰 수 있게 하고 많은 layer을 이용하여 예측의 정확도를 높인다.

II. Logistic Regression Model

1. Import

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
```

기본적으로 NumPy, Pandas와 그래프를 그리는데 사용되는 Matplotlib, Seaborn을 import 한다. 그리고 로지스틱 회귀 모델을 만들 때 사용되는 함수들을 sklearn으로부터 import하는 것으로 프로그램을 시작한다.

2. 데이터프레임

가. 데이터프레임 만들기

우리는 데이터셋을 CSV 파일로 다운로드를 하였기 때문에 데이터프레임을 만들기 위해서는 Pandas를 사용하면 된다.

```
weathers_df=pd.read_csv("weatherAUS.csv")
```

3. 데이터 정리

가. 특정 열 제거

로지스틱 회귀 모델에서는 이분법으로 나뉘지 않는 문자열을 다루기 어렵기 때문에 이러한 원소들을 포함하고 있는 열은 제거했다. 배열에서 특정 열을 완전하게 제거하기 위해서는 drop 함수를 사용하면 된다.

```
weathers_df.drop(['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm'], axis=1, inplace=True)
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity3pm	Pressure9am	Pressur
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	22.0	1007.7	1
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	25.0	1010.6	1
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	30.0	1007.6	1
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	16.0	1017.6	1
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	33.0	1010.8	1

▲ drop 함수 사용 전

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure
0	13.4	22.9	0.6	NaN	NaN	44.0	20.0	24.0	71.0	22.0	1007.7	10
1	7.4	25.1	0.0	NaN	NaN	44.0	4.0	22.0	44.0	25.0	1010.6	10
2	12.9	25.7	0.0	NaN	NaN	46.0	19.0	26.0	38.0	30.0	1007.6	10
3	9.2	28.0	0.0	NaN	NaN	24.0	11.0	9.0	45.0	16.0	1017.6	10
4	17.5	32.3	1.0	NaN	NaN	41.0	7.0	20.0	82.0	33.0	1010.8	10

▲drop 함수 사용 후

우리는 drop 함수를 사용하여 날짜, 장소, 풍향에 대한 열을 완전히 제거하여 다루기 쉬운 데이터들만 남겼다.

나. 공백이 포함된 행 제거

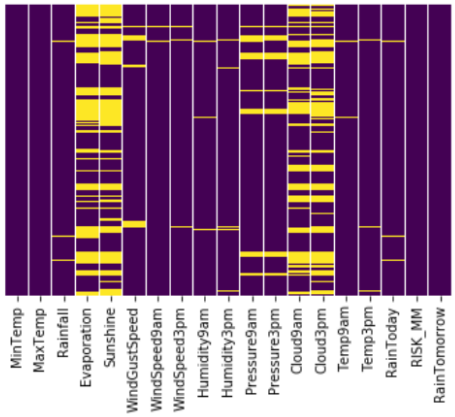
데이터 중 공백이 포함된 것은 Logistic Regression Model의 정확도를 떨어뜨리

므로 이를 제거하기 위해 dropna 함수를 사용하였다.

```
new_weathers_df = weathers_df.dropna(axis=0)
```

```
sns.heatmap(weathers_df.isnull(), yticklabels=False
```

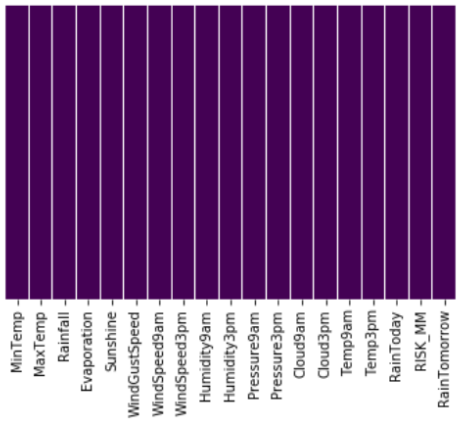
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b047e4b
```



▲dropna 함수 사용 전

```
sns.heatmap(new_weathers_df.isnull(), yticklabels=
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b047e
```



▲dropna 함수 사용 후

데이터들이 정상적으로 제거되어있는지 확인하기 위해서는 heatmap을 사용하면 된다. 위의 그래프에서는 노란색 선이 비어있는 데이터를 표시한다.

다. Binary 데이터로 변환

그다음 Pandas에 포함되어 있는 get_dummies 함수를 이용해 'Yes'와 'No' 데이터로 구성되어있는 'RainToday'와 'RainTomorrow'를 이분법적 데이터로 변환시켰다. 비가 왔으면 0, 안 왔으면 1을 대입하여 데이터베이스를 완성하였다.

```
pd.get_dummies(new_weathers_df['RainToday'])
```

```
new_weathers_df["RainToday"]=pd.get_dummies(new_weathers_df['RainToday'], drop_first=True)
```

이와 같은 작업을 'RainTomorrow'에 똑같이 하면 된다.

```
new_weathers_df.head()
```

ım	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RainToday	RISK_MM	RainTomorrow
5.0	20.0	20.0	13.0	1006.3	1004.4	2.0	5.0	26.6	33.4	0	0.0	0
9.0	19.0	30.0	8.0	1012.9	1012.1	1.0	1.0	20.3	27.0	0	0.0	0
1.0	15.0	42.0	22.0	1012.3	1009.2	1.0	6.0	28.7	34.9	0	0.0	0
5.0	6.0	37.0	22.0	1012.7	1009.1	1.0	5.0	29.1	35.6	0	0.0	0
7.0	13.0	19.0	15.0	1010.7	1007.4	1.0	6.0	33.6	37.6	0	0.0	0

파일을 출력해보면 'RainToday'와 'RainTomorrow'의 데이터 값들이 0과 1로 바뀐 것을 확인할 수 있다.

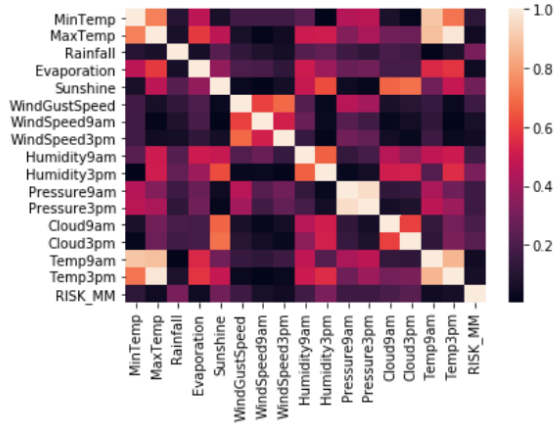
3. 데이터 분석

가. Heatmap

우리의 데이터셋을 heatmap으로 표현하면, 모델을 만들 때 상대적으로 상호연관도가 낮은 데이터 열을 제거하여 만드는 등 여러 가지 경우를 실험해볼 때 도움이 된다.

```
sns.heatmap(abs(weathers_df.corr()))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b047f5e6a0>
```



나. Pairplot

Pairplot은 데이터 열 간의 그래프를 의미한다. Pairplot을 사용하면 heatmap보다 더 정확하게 상호 간의 관계를 알아볼 수 있고, 이 또한 모델을 만들 때 무엇을 제거하고 시도해 볼 것인지 선택하는데 도움이 된다.

4. 모델 생성, 검증

가. 모델 생성

로지스틱 회귀 모델을 생성하기 위해서는 우선 데이터셋을 예측하는데 쓰이는 파일 X와 예측할 값을 포함한 파일 y를 새로 정의해야 한다.

```
new_weathers_df.columns
```

```
Index(['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine',
       'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am',
       'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm',
       'Temp9am', 'Temp3pm', 'RainToday', 'RISK_MM', 'RainTomorrow'],
      dtype='object')
```

```
X=new_weathers_df[['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity
y=new_weathers_df['RainTomorrow']
```

그 후에는 X와 y를 train_test_split 함수를 이용하여 각각 훈련 세트와 테스트 세트로 나뉜다. 이때, 우리는 테스트 세트를 0.2의 비율로 나누었다.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=999)
```

다음으로는 로지스틱 회귀 모델을 생성하는 코드만 작성하면 된다. 이런 함수는 이미 sklearn에 존재하기 때문에 초기에 import한 LogisticRegression 함수를 사용하면 된다. wlog(로지스틱 회귀 모델 이름)을 생성하고나서는 이전에 만든 X_train과 y_train을 이용하여 모델을 훈련시킨다.

```
wlog=LogisticRegression()
```

```
wlog.fit(X_train, y_train)
```

```
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩sklearn₩linear_model₩logistic.py:4
in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

나. 모델 검증

모델을 생성하였다면 테스트 세트로 이 모델이 얼마만큼의 성능을 보이는지 검증해야 한다. 검증하기 위해서는 pred 함수에 X_test를 넣어 y_pred를 정의한 다음 y_pred와 y_test를 비교하면 된다.

```
y_pred=wlog.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.95      0.91      9034
           1       0.74      0.52      0.61      2584

   micro avg       0.85      0.85      0.85     11618
   macro avg       0.81      0.73      0.76     11618
weighted avg       0.84      0.85      0.84     11618
```

위 표에서 확인해야 하는 것은 precision(정밀도)과 recall(재현율) 값이다. 비가 오지 않는 경우를 예측할 때는 높은 수치를 보였지만, 비가 오는 경우에서는 재현율이 낮은 수치를 보였다. 앞에서 데이터 분석을 했을 때, 낮은 관계를 갖는 열을 빼서 해보아도 결과는 비슷했다. 따라서 로지스틱 회귀 모델을 사용했을 때 사용하지 못한 열도 사용함으로써 더 정확한 모델을 만들기 위해 Neural Network Model을 만들기로 하였다.

Ⅲ. Neural Network Model

1. Import and Install

```
!pip install -q sklearn
```

```
from __future__ import absolute_import, division, print_function, unicode_literals

import numpy as np
import pandas as pd

!pip install -q tensorflow==2.0.0-alpha0
import tensorflow as tf

from tensorflow import feature_column
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
```

2. 데이터프레임

가. 데이터프레임 만들기

우리는 데이터셋을 CSV 파일로 다운로드를 하였기 때문에 데이터프레임을 만들기 위해서는 Pandas를 사용하면 된다.

```
weathers_df=pd.read_csv("weatherAUS.csv")
```

데이터프레임을 만든 후, 로지스틱 회귀 모델에서 데이터프레임을 정리한 것과 동일하게 해준다.

나. 훈련, 검증, 테스트 세트로 나누기

모델을 훈련 시키기 위해서는 56420개의 데이터를 훈련 세트, 검증 세트, 테스트 세트로 나누어야 한다. 검증 세트를 훈련 세트로부터 따로 구분한 이유는 모델을 훈련시켜 개발하는 단계에서는 항상 모델을 튜닝시켜야 하기 때문이다. 검증 세트에서 모델의 성능을 평가하여 튜닝을 수행함으로써 모델의 성능을 한 층 더 올릴 수 있게 된다.

```
train, test = train_test_split(new_weathers_df, test_size=0.2)
train, val = train_test_split(train, test_size=0.2)
print(len(train), '훈련 샘플')
print(len(val), '검증 샘플')
print(len(test), '테스트 샘플')
```

```
36108 훈련 샘플
9028 검증 샘플
11284 테스트 샘플
```

훈련 세트와 테스트 세트를 나눌 때는 테스트 세트를 0.2의 비율로, 훈련 세트를 훈련 세트와 검증 세트로 나눌 때는 검증 세트를 0.2의 비율로 나누었다.

3. 입력 파이프라인

가. df.data를 이용하여 입력 파이프라인 생성

데이터프레임을 완성하였다면 df.data를 이용하여 데이터프레임을 Neural Network에서 사용하기 좋도록 데이터셋을 만들어야 한다. 이렇게 하면 데이터프레임의 열(column)을 모델 훈련에 필요한 특성으로 매핑시킬 수 있다.

```
# 판다스 데이터프레임으로부터 tf.data 데이터셋을 만들기 위한 함수
def df_to_dataset(dataframe, shuffle=True, batch_size=32):
  dataframe = dataframe.copy()
  labels = dataframe.pop('RainTomorrow')
  ds = tf.data.Dataset.from_tensor_slices((dict(dataframe), labels))
  if shuffle:
    ds = ds.shuffle(buffer_size=len(dataframe))
  ds = ds.batch(batch_size)
  return ds
```

```
batch_size = 32 # A small batch sized is used for demonstration purposes
train_ds = df_to_dataset(train, batch_size=batch_size)
val_ds = df_to_dataset(val, shuffle=False, batch_size=batch_size)
test_ds = df_to_dataset(test, shuffle=False, batch_size=batch_size)
```

위 함수를 사용하면 데이터프레임으로부터 예측하고 싶은 열을 따로 뽑아낸 뒤, tf.data 데이터셋을 만들고, 데이터들의 순서를 섞을 수 있다.

나. 입력 파이프라인

입력 파이프라인을 형성하게 된다면 데이터셋은 다음과 같이 열의 이름을 key로 하는 dictionary의 형태로 되어있으며, 열의 값들을 매핑하고 있다.

```
for feature_batch, label_batch in train_ds.take(1):
  print('전체 특성:', list(feature_batch.keys()))
  print('MinTemp 특성의 배치:', feature_batch['MinTemp'])
  print('타깃의 배치:', label_batch )
```

```
전체 특성: ['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm', 'RainToday', 'RISK_MM']
MinTemp 특성의 배치: tf.Tensor(
[20.5 23.3 19.5 19.4 15.9  2.3 11.2  9.5  6.8  9.3 12.4  7.9  9.   4.
 20.9 21.   9.8 21.1 11.8 13.4 20.6 22.5 16.3 11.3 12.3 11.6 12.   4.3
 15.8  9.4  8.4 19.9], shape=(32,), dtype=float32)
타깃의 배치: tf.Tensor([1 0 1 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1], shape=(32,), dtype=int32)
```

4. 데이터프레임의 열 변환

데이터프레임의 열을 변환시키는 함수를 정의하여 사용하면 정보가 어떤 식으로 모델에 전달되는지 확인할 수 있다.

```
# 특성 열을 시험해 보기 위해 샘플 배치를 만듭니다.
example_batch = next(iter(train_ds))[0]
```

```
# 특성 열을 만들고 배치 데이터를 변환하는 함수
def demo(feature_column):
  feature_layer = layers.DenseFeatures(feature_column)
  print(feature_layer(example_batch).numpy())
```

가. 수치형 열 변환

수치형 열은 원소가 실수인 것을 말하며, 이 열을 모델로 보낼 때는 데이터프레임의 열의 값을 변형시키지 않고, 그대로 전달한다.

```
Evaporation = feature_column.numeric_column("Evaporation")
demo(Evaporation)
```

```
[[ 2.6]
 [ 2.4]
 [ 0.6]
 [ 2.  ]
```

나. 범주형 열 변환

범주형 열은 쉽게 말하면 문자열 열을 말한다. 로지스틱 모델에서는 문자열 열을 다루기 어려워서 제거하였지만, Neural Network 모델에서는 다음과 같이 one-hot vector 로표현이 가능하다. One-hot vector는 문자를 숫자로 바꾸는 가장 기본적인 기법으로 표현하고 싶은 단어의 인덱스의 위치에는 1을 부여하고, 다른 단어의 인덱스의 위치에는 0을 부여하는 방법이다. 즉, 한 행에서는 하나의 위치에서만 1이 나타난다.

```
WindGustDir = feature_column.categorical_column_with_vocabulary_list(
      'WindGustDir', ['SSW', 'S', 'NNE', 'WNW', 'N', 'SE', 'ENE', 'NE', 'E', 'SW', 'W',
       'WSW', 'NNW', 'ESE', 'SSE', 'NW'])

WindGustDir_one_hot = feature_column.indicator_column(WindGustDir)
demo(WindGustDir_one_hot)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

다. 특성 열 정의

수치형 열과 범주형 열을 사용하는 방법을 이해했으므로, 이제는 이를 모델에 주입하기 위해서 특성 열을 정의해야 한다. 이때, 모델에 사용할 특성 열을 선택할 수 있으므로, 여러 가지 경우로 실험해 볼 때는 이 부분만 바꾸어 주면 된다.

```
feature_columns = []
# numeric cols
for header in ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine',
               'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm',
               'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm']:
  feature_columns.append(feature_column.numeric_column(header))
```

▲ 수치형 열 선언

```
Location = feature_column.categorical_column_with_vocabulary_list(
      'Location', ['Cobar', 'CoffsHarbour', 'Moree', 'NorfolkIsland', 'Sydney',
       'SydneyAirport', 'WaggaWagga', 'Williamtown', 'Canberra', 'Sale',
       'MelbourneAirport', 'Melbourne', 'Mildura', 'Portland', 'Watsonia',
       'Brisbane', 'Cairns', 'Townsville', 'MountGambier', 'Nuriootpa',
       'Woomera', 'PerthAirport', 'Perth', 'Hobart', 'AliceSprings',
       'Darwin'])

Location_one_hot = feature_column.indicator_column(Location)
feature_columns.append(Location_one_hot)
```

▲ 범주형 열 선언

5. 모델 생성, 컴파일, 훈련

가. 특성 층 형성

특성 열을 선언하였으면 이제 keras에 정의되어 있는 DenseFeatures 함수를 이용하여 특성 열들을 keras 모델에 주입할 수 있다.

```
feature_layer = tf.keras.layers.DenseFeatures(feature_columns)
```

나. 모델 생성, 컴파일, 훈련

이제는 위에서 정의한 특성 층을 모델에 사용해야 한다. Keras 모델을 만들어 컴파일 후, 실행하기 위해서는 다음과 같은 코드를 작성하면 된다.

```
model = tf.keras.Sequential([
  feature_layer,
  layers.Dense(128, activation='relu'),
  layers.Dense(128, activation='relu'),
  layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'],
              run_eagerly=True)

model.fit(train_ds,
          validation_data=val_ds,
          epochs=5)
```

다. 결과

Keras 모델을 통해 훈련 시키는 과정에서 정확도는 다음과 같이 나왔다.

```
Epoch 1/5
1129/1129 [==============================] - 27s 24ms/step - loss: 3.3973 - accuracy: 0.7824 - val_loss: 3.4560 - val_accuracy: 0.7760
Epoch 2/5
1129/1129 [==============================] - 27s 24ms/step - loss: 3.3973 - accuracy: 0.7824 - val_loss: 3.4560 - val_accuracy: 0.7760
Epoch 3/5
1129/1129 [==============================] - 28s 25ms/step - loss: 3.3973 - accuracy: 0.7824 - val_loss: 3.4560 - val_accuracy: 0.7760
Epoch 4/5
1129/1129 [==============================] - 29s 26ms/step - loss: 3.3973 - accuracy: 0.7824 - val_loss: 3.4560 - val_accuracy: 0.7760
Epoch 5/5
1129/1129 [==============================] - 27s 24ms/step - loss: 3.3973 - accuracy: 0.7824 - val_loss: 3.4560 - val_accuracy: 0.7760
```

이제는 이 모델을 이용하여 테스트 세트에 대해 예측한 것이 얼마나 정확한지 알아보면 된다. 이를 알아보기 위해서는 evaluate 함수를 사용하여 정확도를 출력하면 된다.

```
loss, accuracy = model.evaluate(test_ds)
print("Accuracy", accuracy)
```

```
353/353 [==============================] - 5s 14ms/step - loss: 3.3521 - accuracy: 0.7826
Accuracy 0.78261256
```

결과는 0.7826로 높은 수준의 정확도를 얻어냈다.

Ⅳ. 결론

로지스틱 회귀 모델은 문자열 열과 같이 사용하기 어려운 데이터 열들이 존재할 때에는 이들을 배제하여 모델을 훈련시켜야 한다는 단점 때문에 결과적으로 비가 오는 경우의 재현율이 낮았다. 하지만, Neural Network는 어떤 데이터 열이든 각각 처리하는 방법이 존재하기 때문에 로지스틱 회귀 모델에서 문제가 되었던 문자열 열을 포함시킬 수 있다는 것이 좋은 장점이었다. 또한, Neural Network는 기본적으로 여러 층들을 생성하여 훈련시키기 때문에 단순하게 상관계수를 구하여 예측하는 로지스틱 회귀 모델보다는 성능이 더 좋을 수 밖에 없었던 것 같다. 앞으로도 CNN을 사용할 정도로 어려운 데이터셋이 아니 경우에 Neural Network를 사용하면 로지스틱 회귀 모델이나 직선형 모델 등보다는 더 좋은 성능을 발휘할 수 있을 것 같다.

Ⅴ. 참고문헌

“Classify structured data”, TensorFlow, https://www.tensorflow.org/beta/tutorials/keras/feature_columns

“원-핫 인코딩(One-hot encoding)”, WikiDocs, https://wikidocs.net/22647

“[Keras Study] 4장. 머신 러닝의 기본 요소”, Hello Subinium!, https://subinium.github.io/Keras-4/