

# 개인 프로젝트

스팸 메일 분류

발표자 : 김성윤

# 목차

1. 프로젝트 목적
2. 모듈·데이터 불러오기 & 데이터 확인
3. 데이터 전처리
4. 인코딩 & 임베딩
5. 모델링
6. 결과 확인
7. 활용
8. 개선점


# 1. 프로젝트 목적

- 배운 내용을 활용하여 실습 코드 확인
- 모델링 과정을 이해

## 2. 모듈·데이터 불러오기 & 데이터 확인

✓ 6초 [2] 

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from keras_preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, BatchNormalization, Embedding, LSTM
from keras import callbacks
import matplotlib.pyplot as plt
```

✓ 0초 

```
dt = pd.read_csv('./SPAM_text_message.csv')
```

✓ 0초 [4] 

```
dt.sample()
```

	Category	Message
770	ham	Lol I know! They're so dramatic. Schools alrea...



### 3. 데이터 전처리

```
✓ [5] # 결측치 확인  
0초 print('결측값 여부 :', dt.isnull().values.any())  
  
# 데이터 중복 확인  
print('Message 중 중복이 아닌 값 :', dt['Message'].nunique())  
  
# 중복 데이터 제거  
dt.drop_duplicates(subset=['Message'], inplace=True)  
  
# 제거되었는지 점검  
print("중복 제거 후 데이터 크기 :", len(dt))
```

```
결측값 여부 : False  
Message 중 중복이 아닌 값 : 5157  
중복 제거 후 데이터 크기 : 5157
```

### 3. 데이터 전처리

✓  
0초

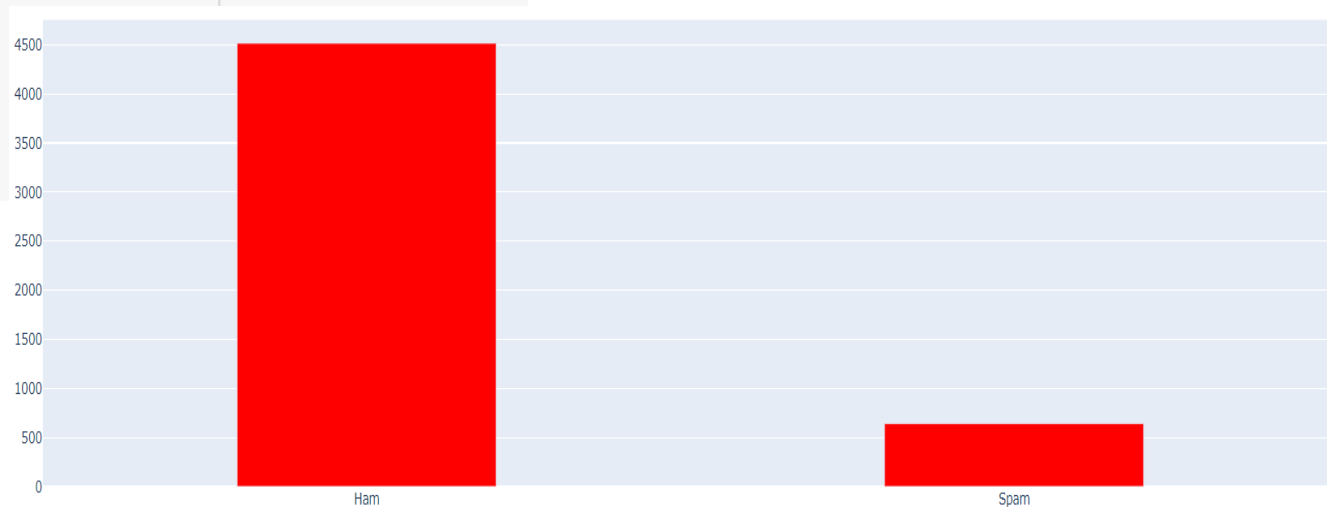
[6] #레이블 비율 확인

```
dt.replace('ham', 0, inplace=True)  
dt.replace('spam', 1, inplace=True)
```

```
print(f'전체 데이터 중 정상 메일의 비율 = {round(dt["Category"].value_counts()[0]/len(dt) * 100,2)}%')  
print(f'전체 데이터 중 스팸 메일의 비율 = {round(dt["Category"].value_counts()[1]/len(dt) * 100,2)}%')
```

```
groups = dt.groupby(by='Category').count().Message  
CNT_HAM = groups[0]  
CNT_SPAM = groups[1]  
fig = go.Figure()  
fig.add_trace(go.Bar(  
    x=['Ham', 'Spam'],  
    y=[CNT_HAM, CNT_SPAM],  
    marker_color='red',  
    width=[0.4,0.4]))
```

전체 데이터 중 정상 메일의 비율 = 87.57%  
전체 데이터 중 스팸 메일의 비율 = 12.43%



## 4. 인코딩 & 임베딩

✓  
0초

```
[7] X_data = dt['Message']  
     y_data = dt['Category']
```

```
# 정수 인코딩
```

```
NUM_WORDS = 10000
```

```
tokenizer = Tokenizer(num_words=NUM_WORDS)
```


```
tokenizer.fit_on_texts(X_data)
```

```
word_to_index = tokenizer.word_index
```

```
print(word_to_index)
```

```
{ 'i': 1, 'to': 2, 'you': 3, 'a': 4, 'the': 5, 'u': 6, 'and': 7, 'in': 8, 'is': 9, 'me': 10, 'my': 11, 'for': 12,
```

## 4. 인코딩 & 임베딩

✓  # 정제 및 정규화

```
threshold = 2
total_cnt = len(word_to_index) # 단어의 수
rare_cnt = 0 # 등장 빈도수가 threshold보다 작은 단어의 개수를 카운트
total_freq = 0 # 훈련 데이터의 전체 단어 빈도수 총 합
rare_freq = 0 # 등장 빈도수가 threshold보다 작은 단어의 등장 빈도수의 총 합

# 단어와 빈도수의 쌍을 key와 value로 받는다.
for key, value in tokenizer.word_counts.items():
    total_freq = total_freq + value

    if(value < threshold):
        rare_cnt = rare_cnt + 1
        rare_freq = rare_freq + value

print('등장 빈도가 %s번 이하인 희귀 단어의 수: %s' %(threshold - 1, rare_cnt))
print("단어 집합(vocabulary)에서 희귀 단어의 비율:", (rare_cnt / total_cnt)*100)
print("전체 등장 빈도에서 희귀 단어 등장 빈도 비율:", (rare_freq / total_freq)*100)
```

📄 등장 빈도가 1번 이하인 희귀 단어의 수: 5001  
 단어 집합(vocabulary)에서 희귀 단어의 비율: 55.54198134162595  
 전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 6.18438137636802



## 4. 인코딩 & 임베딩

```
[9] #워드 임베딩
    sequences = tokenizer.texts_to_sequences(X_data)

    #패딩
    MAX_TEXT_LEN = 100
    X = pad_sequences(sequences, maxlen=MAX_TEXT_LEN)
    y = y_data.copy()

    # 데이터 분리
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True, random_state=42, stratify = y)
```

## 5. 모델링

[10] # 모델링

```
model = Sequential()
model.add(Embedding(NUM_WORDS, 64, input_length=MAX_TEXT_LEN))
model.add(LSTM(3, return_sequences=True))
model.add(LSTM(5, return_sequences=True))
model.add(BatchNormalization())
model.add(LSTM(12))
model.add(Dense(1, activation='sigmoid'))

callbacks_list = [
    callbacks.EarlyStopping(monitor='loss', min_delta=0.01, patience=2, verbose=1),
    callbacks.ReduceLROnPlateau(monitor='loss', factor=0.1, min_delta=0.01, min_lr=1e-10, patience=4, verbose=1, mode='auto')
]
```

[11] model.compile(metrics=['Accuracy'], loss='binary\_crossentropy', optimizer='Adam')

## 5. 모델링

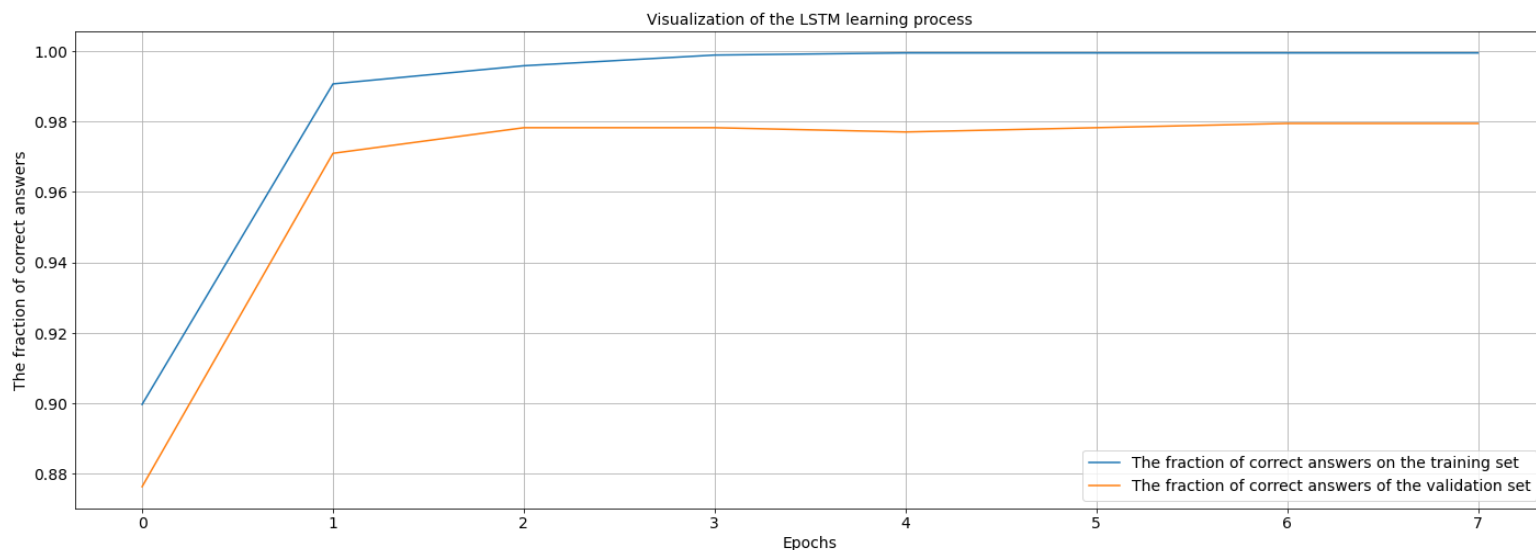
```
[12] history = model.fit(X_train, y_train, batch_size=50, epochs=10, validation_split=0.2, callbacks=callbacks_list)
```

```
Epoch 1/10
66/66 [=====] - 25s 202ms/step - loss: 0.3565 - Accuracy: 0.8997 - val_loss: 0.3959 - val_Accuracy: 0.8764 - lr: 0.0010
Epoch 2/10
66/66 [=====] - 6s 93ms/step - loss: 0.1035 - Accuracy: 0.9906 - val_loss: 0.2739 - val_Accuracy: 0.9709 - lr: 0.0010
Epoch 3/10
66/66 [=====] - 7s 110ms/step - loss: 0.0511 - Accuracy: 0.9958 - val_loss: 0.1947 - val_Accuracy: 0.9782 - lr: 0.0010
Epoch 4/10
66/66 [=====] - 6s 84ms/step - loss: 0.0271 - Accuracy: 0.9988 - val_loss: 0.1086 - val_Accuracy: 0.9782 - lr: 0.0010
Epoch 5/10
66/66 [=====] - 7s 106ms/step - loss: 0.0178 - Accuracy: 0.9994 - val_loss: 0.0984 - val_Accuracy: 0.9770 - lr: 0.0010
Epoch 6/10
66/66 [=====] - 6s 85ms/step - loss: 0.0120 - Accuracy: 0.9994 - val_loss: 0.0890 - val_Accuracy: 0.9782 - lr: 0.0010
Epoch 7/10
66/66 [=====] - 7s 108ms/step - loss: 0.0088 - Accuracy: 0.9994 - val_loss: 0.0880 - val_Accuracy: 0.9794 - lr: 0.0010
Epoch 8/10
66/66 [=====] - 6s 84ms/step - loss: 0.0068 - Accuracy: 0.9994 - val_loss: 0.0880 - val_Accuracy: 0.9794 - lr: 0.0010
Epoch 8: early stopping
```

## 5. 모델링

# 학습과정 시각화

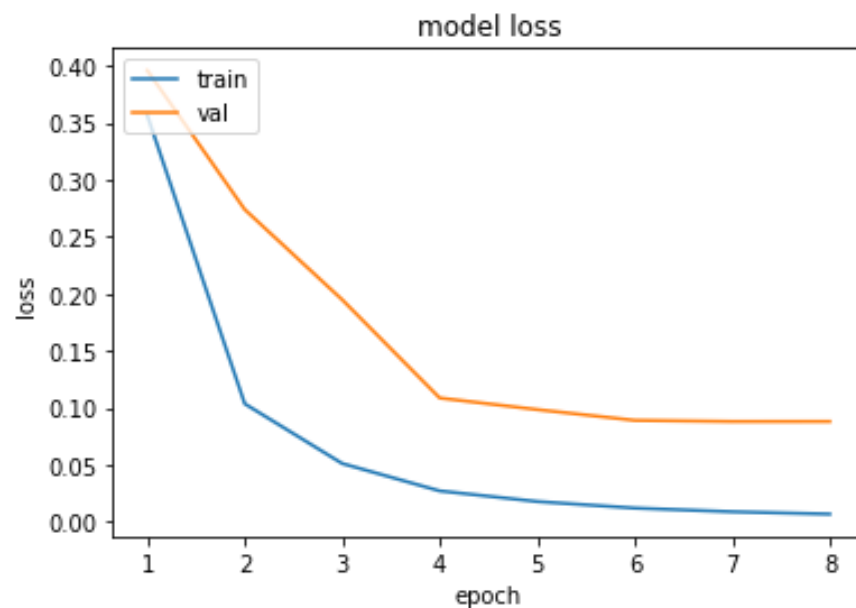
```
plt.figure(figsize=(24, 8))
plt.title('Visualization of the LSTM learning process', fontsize=14)
plt.plot(history.history['Accuracy'], label='The fraction of correct answers on the training set')
plt.plot(history.history['val_Accuracy'], label='The fraction of correct answers of the validation set')
plt.xlabel('Epochs', fontsize=14)
plt.ylabel('The fraction of correct answers', fontsize=14)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid()
plt.legend(fontsize=14)
plt.show()
```



## 5. 모델링

# 오차함수 시각화

```
epochs = range(1, len(history.history['Accuracy']) + 1)
plt.plot(epochs, history.history['loss'])
plt.plot(epochs, history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



## 6. 결과 확인

```
[15] # 결과 확인
      print(f'테스트 데이터 정확도 : {model.evaluate(X_test, y_test)[1]}')
```

```
33/33 [=====] - 1s 18ms/step - loss: 0.1183 - Accuracy: 0.9719
테스트 데이터 정확도 : 0.9718992114067078
```

```
[16] model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 64)	640000
lstm (LSTM)	(None, 100, 3)	816
lstm_1 (LSTM)	(None, 100, 5)	180
batch_normalization (Batch Normalization)	(None, 100, 5)	20
lstm_2 (LSTM)	(None, 12)	864
dense (Dense)	(None, 1)	13

```
=====
Total params: 641,893
Trainable params: 641,883
Non-trainable params: 10
=====
```

## 7. 활용

```
[17] # 스팸 메일 판별 함수
def ml_pipeline(text: str) -> str:
    """LSTM model prediction function for this sample"""
    try:
        sequence = tokenizer.texts_to_sequences([text])
        sequence = pad_sequences(sequence, maxlen=MAX_TEXT_LEN)
        if sequence.max() == 0:
            return 'Enter the words in English'
        else:
            predict = model.predict(sequence, verbose=0)
            if predict > 0.5:
                return 'The text is spam'
            else:
                return 'The text is not spam'
    except AttributeError:
        return 'Enter the text'
```

```
[18] ml_pipeline('if you subscribe, you can use freely this service. only today you should buy this model 40$')

'The text is not spam'
```

## 7. 개선점

- 다른 데이터셋으로 점검하지 못함
- 전형적인 스팸 메일의 형태를 벗어날 경우 잘 구분하지 못함
- 정확도가 높은 이유를 파악하지 못함
- 새롭고 유의미한 결과물로 엮어내지 못함