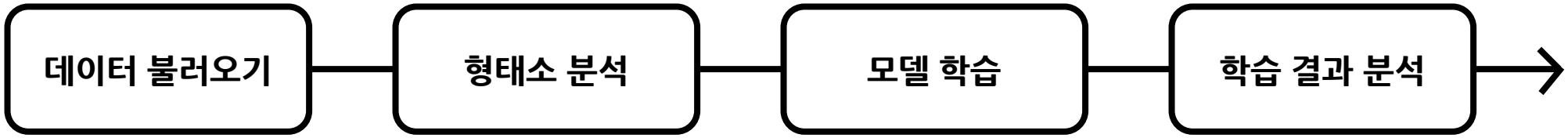
TTA 인공지능 모델 분석

2022. 06. 14.

INDEX



학습에 사용할 데이터 불러오기

471_train_data.txt

id la content

00071 0 요거에요! 톤28 세정젤이네용

00073 1 진정팩인데 수분감 좋아요! 거즈마스크인 것두 좋아요!

00074 1 롬앤 더스티로즈 강추여! 처음엔 붉고 웜하게 올라오는데 갈수록 살짝 안개낀 핑크느낌 들면서 붉은데 너무 웜하지 않은 느낌! 쿨톤인데 완전 잘쓰고 있어요!

00079 0 사실 저 키보드 삼,, 키보드 소리 넘 귀여워 소듕해 다들 내 키보드 소리 들어줘쓰면 조케써,,☆

00112 1 문샷틴트핏블러 발림성 완전 버터 같아요! 지속력이 재금 아쉽지만 겉보속촉 넘 부드럽게 발려서 계속 바르고 싶은 고론느낌 #문샷 #틴트핏블러 #틴트추천

00115 1 #클리오#킬커버워터프루프마스카라이거 진짜 찐탱이에요..벌써 다써감.. 올영 세일할 때 1+1으로 꼭 겟하세요!!!!

00119 0 화알못이에요 저

00122 1 요즘 잘 쓰고있는 웜톤한테 진짜 형광등 킨 것 같은에스쁘아 문릿입니다진짜 추천해요

00123 1 #이즈앤트리 #스팟 세이버 쑥 파우더 워시엄청 세정력이 강한 건 아닌데휴대가 찐 간편하고알갱이? 같은 게 있어서 각질제거도 되공징짜 갱춘 ~!

00125 1 민감 + 지성인 노답피부는 닥치고 써야댐집에선 큰사이즈쓰고 헬스장용으로 200 씀사용감에 한번 익숙해지면 빠져나올 수 없다#바이오더마 #이름어려운 #클렌저

00127 0 #이지아#톤업크림 진짜 잘 썼어요 ㅋㅋㅋ진짜 sns 안믿는데 이거 진짜 성공했씀당진짜 기초 크림 바를 때만 너무 유뷴기 많은 것만 안쓰면하나도 안뜨고 진짜 자연스럽게 하얘져서요즘 맨날 써요 자연스럽게 하얘진…

00129 0 #노베브#언더아이마스터지속력 진짜 미쳤어용!!!!듀얼이라 간편하고, 솔직히영상보면 알 수 있듯이 색이 엄청 막 예쁜 건 아닌데색 별로 상관없고 자연스러운+ 지속력 좋은 애교살 메이커 찾는다고 하면 무적권 …

00131 1 #이글립스 #블러파우더틴트 진짜 저 겨울만 되면 각질 장난 아닌데솔직히 이거 제형이 각질 잘 부각되는 제형이긴 해요근데 저는 막상 바르니까 부드럽게 잘 발리긴했고각질 부각은 별루.. 안됐음!!오히려 그래서

학습에 사용할 데이터 불러오기

```
train_data = read_data('471_train_data.txt')
test_data = read_data('471_test_data.txt')


print(len(train_data))
print(len(train_data[0]))
print(len(test_data))
print(len(test_data[0]))
```

```
6094 3 504 3
```

형태소 분석

KoNLPy - Okt Class

오픈 소스 한국어 분석기

- Json 라이브러리
- os 라이브러리

설치 방법

```
from konlpy.tag import Okt
import json
import os
from pprint import pprint

# Okt 객체 생성
okt = Okt()
```

형태소 분석 과정

“진정팩인데 수분감 좋아요!”

JSON 데이터

단어	형태
진정	Noun
팩	Noun
인데	Josa
수분	Noun
감	Noun
좋다	Adjective
!	Punctuation

형태소 분석

```
def tokenize(doc):
    # norm은 정규화, stem은 근어로 표시하기를 나타냄
    return ['/'.join(t) for t in okt.pos(doc, norm=True, stem=True)]
```

[파라미터 종류]

① 형태소 분석을 하고 싶은 문장

② norm: 그래욬ㅋㅋ -> 그래요로 변환하는 옵션

③ stem: 그래요 -> 그렇다로 원형을 찾는 옵션

형태소 분석

```
# 형태소 분석 json 데이터가 이미 존재한다면 데이터를 불러옴
If os.path.isfile('471_rating_data.json'):
    with open('471_rating_data.json', encoding='UTF8') as f:
        train_docs = json.load(f)
    with open('471_test_data.json', encoding='UTF8') as f:
        test_docs = json.load(f)

# 그렇지 않다면 tokenize()로 형태소 분석을 진행
else:
    train_docs = [(tokenize(row[2]), row[1]) for row in train_data]
    test_docs = [(tokenize(row[2]), row[1]) for row in test_data]

    # JSON 파일로 저장
    with open('471_rating_data.json', 'w', encoding='UTF8') as make_file:
        json.dump(train_docs, make_file, ensure_ascii=False, indent="\t")
    with open('471_test_data.json', 'w', encoding='UTF8') as make_file:
        json.dump(test_docs, make_file, ensure_ascii=False, indent="\t")
```

형태소 분석 결과

```
tokens = [t for d in train_docs for t in d[0]]

println(tokens)
```

```
[
    [
        [
            "요/Modifier",
            "거/Noun",
            "에요/Josa",
            "!/Punctuation",
            "톤/Noun",
            "28/Number",
            "세/Modifier",
            "정젤/Noun",
            "이네/Josa",
            "용/Noun"
        ],
        "0"
    ],
    [
        [
            "진정/Noun",
            "팩/Noun",
            "인데/Josa",
            "수분/Noun",
            "감/Noun",
            "좋다/Adjective",
            "!/Punctuation",
            "거즈/Noun",
            "마스크/Noun",
            "인/Josa",
            "것/Noun",
            "두/Josa",
            "좋다/Adjective",
            "!/Punctuation"
        ],
        "1“
    ],
    …
]
```

NLTK - Text 클래스

문서 분석에 유용한 여러가지 메서드 제공 → 토큰열을 입력해 생성

```
import nltk
text = nltk.Text(tokens, name='NMSC’)

print('# 전체 토큰의 개수')
print(len(text.tokens))

print('# 중복을 제외한 토큰의 개수’)
print(len(set(text.tokens)))
```

```
# 전체 토큰의 개수
131096
# 중복을 제외한 토큰의 개수
11117
```

데이터셋 준비

```
# 가장 빈번하게 나타난 단어 10000개만 사용
selected_words = [f[0] for f in text.vocab().most_common(10000)]

# 각 단어 별 등장 횟수를 세어 리턴하는 함수
def term_frequency(doc):
    return [doc.count(word) for word in selected_words]

# x에는 등장 횟수를, y에는 단어를 담아 훈련용 데이터셋과 테스트용 데이터셋 생성
train_x = [term_frequency(d) for d, _ in train_docs]
test_x = [term_frequency(d) for d, _ in test_docs]
train_y = [c for _, c in train_docs]
test_y = [c for _, c in test_docs]
```

numpy

- 다차원 배열을 처리하는데 필요한 여러 기능을 제공
- 리스트보다 속도가 빠르고, 메모리 사용이 효율적

```
import numpy as np

x_train = np.asarray(train_x).astype('float32')
x_test = np.asarray(test_x).astype('float32')

y_train = np.asarray(train_y).astype('float32')
y_test = np.asarray(test_y).astype('float32')
```

Tensorflow.keras

딥러닝 모델을 빌드하고 학습시키기 위한 API

```
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras import losses
```

Scikit_learn(사이킷런)

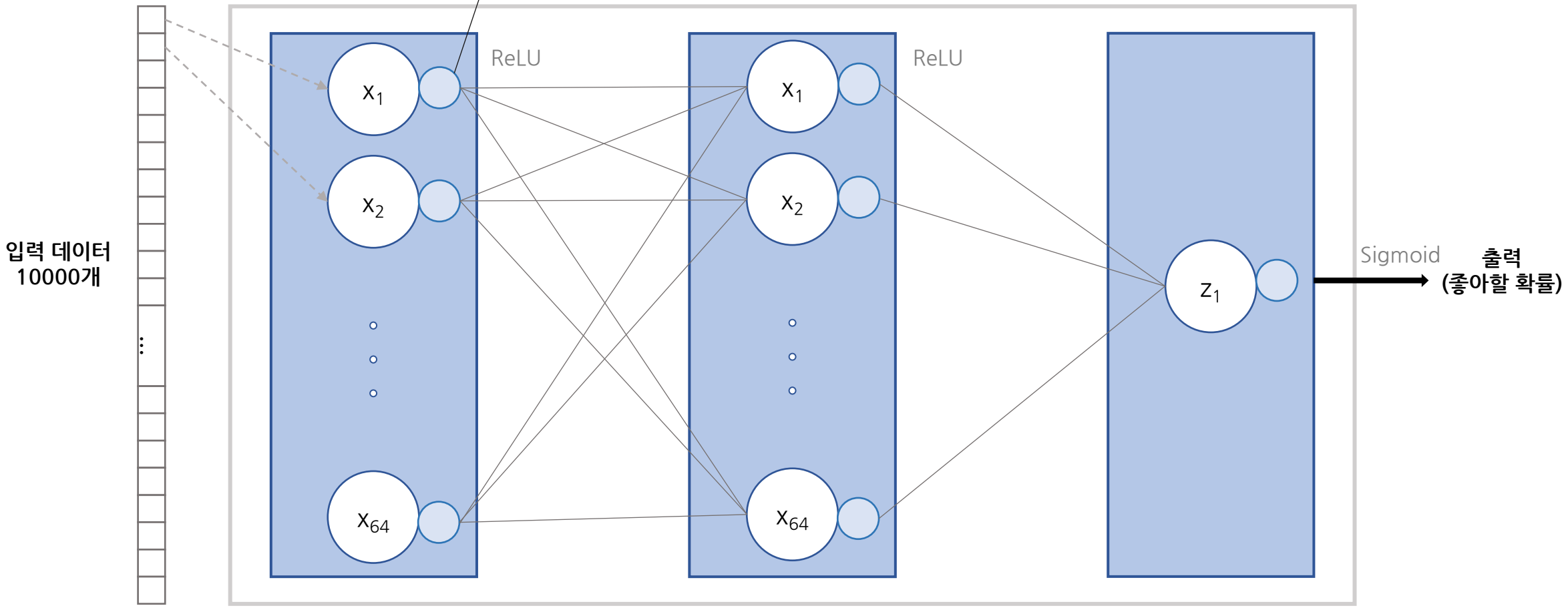
데이터 분석 및 머신러닝 적용을 위한 파이썬 기반 라이브러리

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
```

모델 학습

활성화 함수

이전 층의 결과값을 변환해 다른 층의 뉴런으로 신호를 전달하는 역할



입력층

64개의 뉴런

은닉층

64개의 뉴런

출력층

1개의 뉴런

모델 학습

- **Sequantial()**: 모델 객체 생성
- **Layers.Dense(***`뉴런 개수`, `활성화 함수`, `입력 데이터 개수`***)**: 모델의 층 생성

```
# 모델 객체 생성
Model = models.Sequential()

# 64개의 뉴런을 가진 입력층 추가
Model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
# 64개의 뉴런을 가진 은닉층 추가
model.add(layers.Dense(64, activation='relu’))
# 1개의 뉴런을 가진 출력층 추가
model.add(layers.Dense(1, activation='sigmoid'))
```

모델 학습

- **compile()** : 모델에 추가적인 지표 설정
    - Optimizer: 값 보정 함수
    - Loss: 손실 함수
    - Metrics: 평가 지표 설정

```
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=["acc"])
```

모델 학습

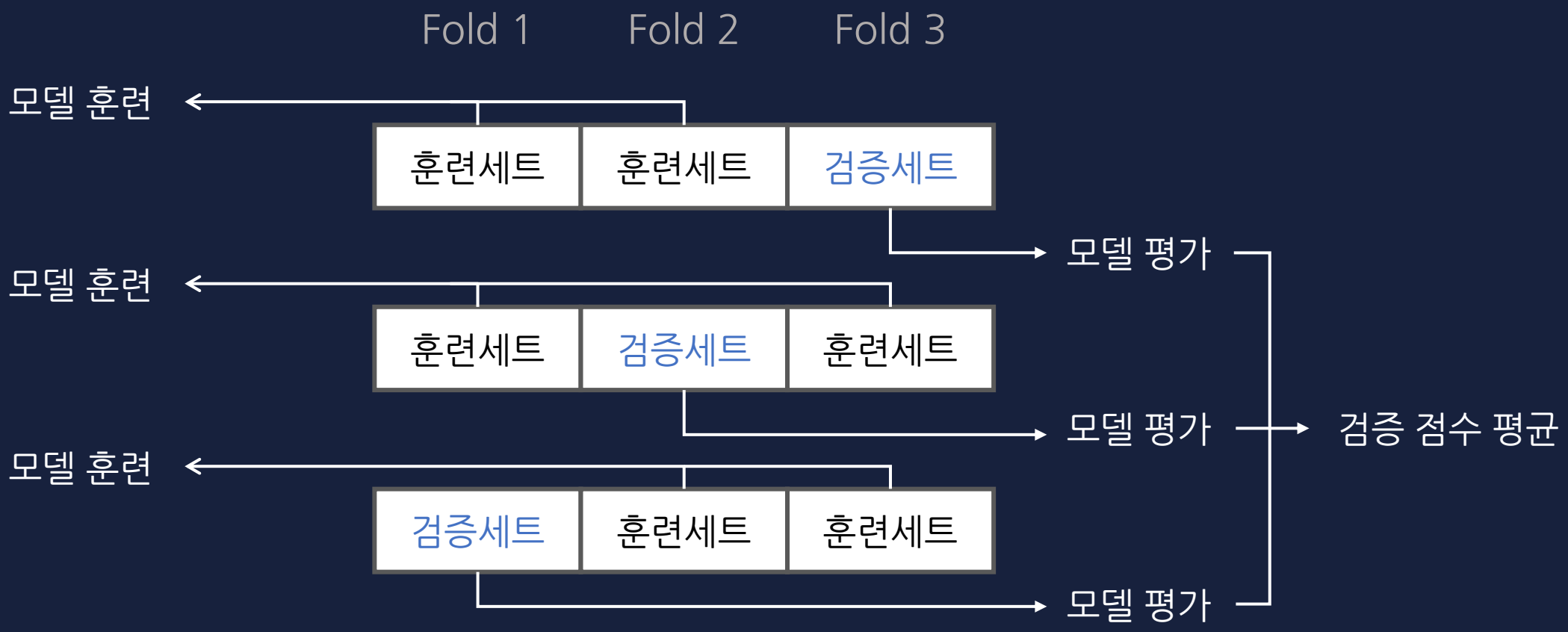
```
def create_model():

 # create model
 model = models.Sequential()
 model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
 model.add(layers.Dense(64, activation='relu'))
 model.add(layers.Dense(1, activation='sigmoid'))

 # Compile model
 model.compile(optimizer=optimizers.RMSprop(lr=0.001),
               loss=losses.binary_crossentropy,
               metrics=["acc"])
 return model
```

교차 검증

검증 데이터셋으로 모델의 매개변수를 튜닝하는 과정



Kfold (k = 3)

교차 검증

```
seed = 7
np.random.seed(seed)

model = KerasClassifier(build_fn=create_model, epochs=10, batch_size=512,
                        validation_data=(x_test, y_test) ,verbose = 0)

kfold = KFold(n_splits=5, shuffle=True, random_state=seed)        # k = 5
results = cross_val_score(model, x_train, y_train, cv=kfold)
print("교차 검증 점수: ", results)
print("교차 검증 평균: {:.2f}".format(results.mean()))
```

```
교차 검증 점수: [0.67104185 0.62428218 0.67022151 0.63412631 0.66584563]
교차 검증 평균: 0.65
```

모델 학습

모델 학습

```
# 모델 학습
hist = model.fit(x_train, y_train,
                 epochs=10, batch_size=512,
                 validation_data=(x_test, y_test))

# 모델 평가
results = model.evaluate(x_test, y_test)
print(results)
```

```
[1.667404055595398, 0.6507936716079712]
```

fit 메소드의 파라미터

- **x_train** 학습시킬 값
- **y_train** x_train을 학습시켰을 때 나와야 하는 결과 값
- **epochs** 반복 테스트 횟수
- **batch_size** 한번의 테스트에 주어지는 데이터 샘플 수
- **validation_data** 학습 모델을 평가할 때 사용할 데이터셋

모델 학습

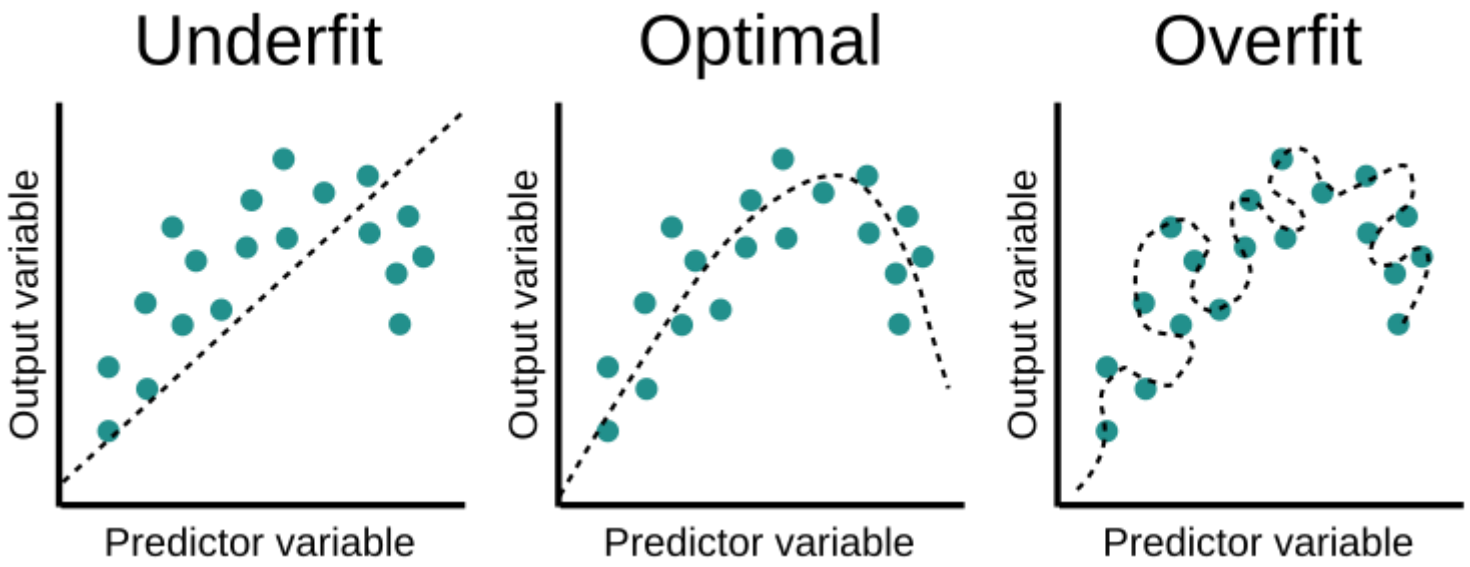
Epoch란?

- 여러 번 학습을 반복해 최적의 값을 구하는 것
- 적절한 epoch 값을 설정해 underfitting과 overfitting을 방지

```
Epoch 1/10 12/12 [==============================] - 3s 129ms/step - loss: 0.6739 - acc: 0.6114 - val_loss: 0.6960 - val_acc: 0.5972
Epoch 2/10 12/12 [==============================] - 1s 56ms/step - loss: 0.5797 - acc: 0.7443 - val_loss: 0.8203 - val_acc: 0.5933
Epoch 3/10 12/12 [==============================] - 1s 69ms/step - loss: 0.4897 - acc: 0.7923 - val_loss: 0.8996 - val_acc: 0.6012
Epoch 4/10 12/12 [==============================] - 1s 61ms/step - loss: 0.4162 - acc: 0.8269 - val_loss: 0.8360 - val_acc: 0.6548
Epoch 5/10 12/12 [==============================] - 1s 50ms/step - loss: 0.3606 - acc: 0.8515 - val_loss: 0.9618 - val_acc: 0.6409
Epoch 6/10 12/12 [==============================] - 1s 44ms/step - loss: 0.3164 - acc: 0.8730 - val_loss: 1.1590 - val_acc: 0.6052
Epoch 7/10 12/12 [==============================] - 1s 64ms/step - loss: 0.2827 - acc: 0.8825 - val_loss: 1.1199 - val_acc: 0.6548
Epoch 8/10 12/12 [==============================] - 1s 44ms/step - loss: 0.2556 - acc: 0.8902 - val_loss: 1.3534 - val_acc: 0.6349
Epoch 9/10 12/12 [==============================] - 0s 41ms/step - loss: 0.2360 - acc: 0.8945 - val_loss: 1.6144 - val_acc: 0.6190
Epoch 10/10 12/12 [==============================] - 1s 76ms/step - loss: 0.2233 - acc: 0.8981 - val_loss: 1.6674 - val_acc: 0.6508
16/16 [==============================] - 0s 3ms/step - loss: 1.6674 - acc: 0.6508
```

모델 학습

Undefitting과 Overfitting



matplotlib 라이브러리

데이터 시각화 라이브러리로 그래프를 그릴 때 유용

```
import matplotlib.pyplot as plt
from matplotlib import font_manager, rc
```

학습 결과 분석

그래프로 나타내고 싶은 데이터들을 history 객체로부터 불러온다.

- **acc, loss** : 훈련 데이터셋에서의 정확도와 loss 값
- **val_acc, val_loss**: 테스트 데이터셋에서의 정확도와 loss 값
- **epochs** : 그래프의 x축으로 보여줄 epoch의 범위

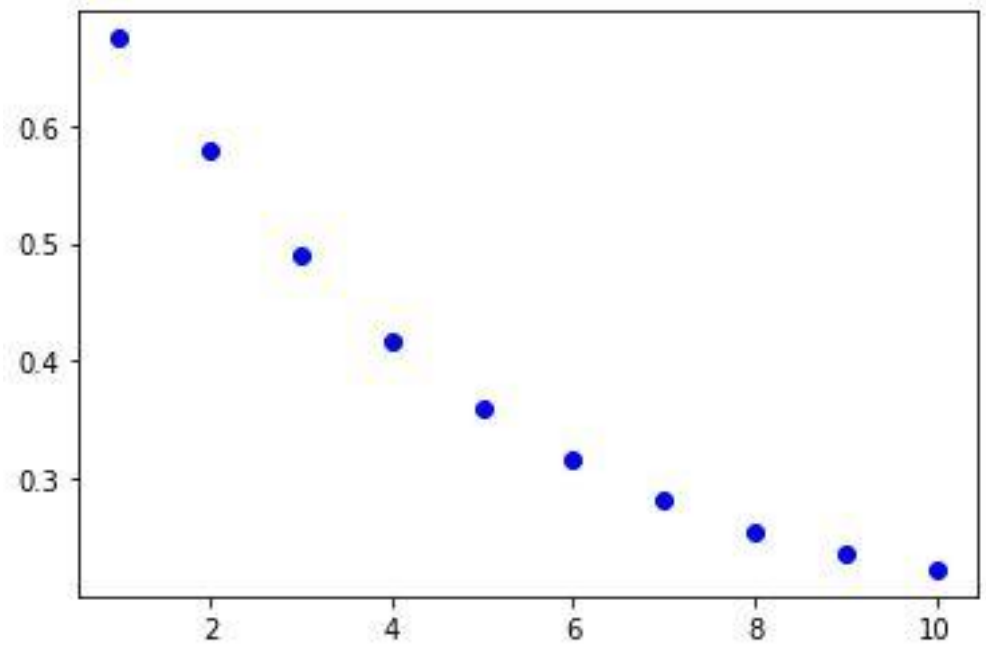
```
acc = hist.history['acc']
val_acc = hist.history['val_acc']
loss = hist.history['loss']
val_loss = hist.history['val_loss']

epochs = range(1, len(acc) + 1)
```

학습 결과 분석

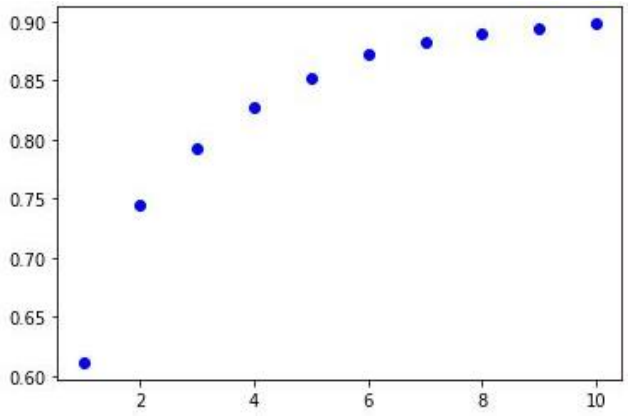
plot: x축, y축과 보여줄 데이터를 설정하면 그래프 출력

```
plt.plot(epochs, loss, 'bo', label='Training loss')
```



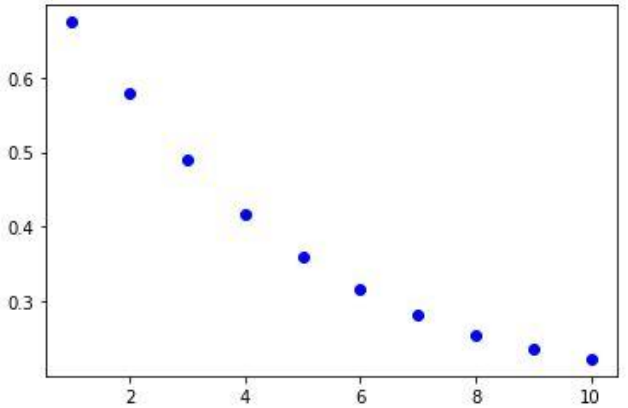
TTA 딥러닝 모델 학습 결과

훈련 데이터셋에서의 정확도



훈련한 epoch 수에 비례하여 정확도가 증가해 **99%의 정확도를 달성**

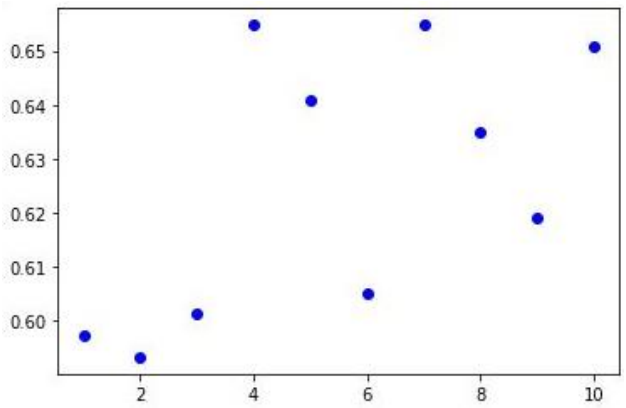
훈련 데이터셋에서의 손실값



훈련한 epoch 수에 비례하여 손실값이 우하향해 **손실값이 0.1에 수렴**

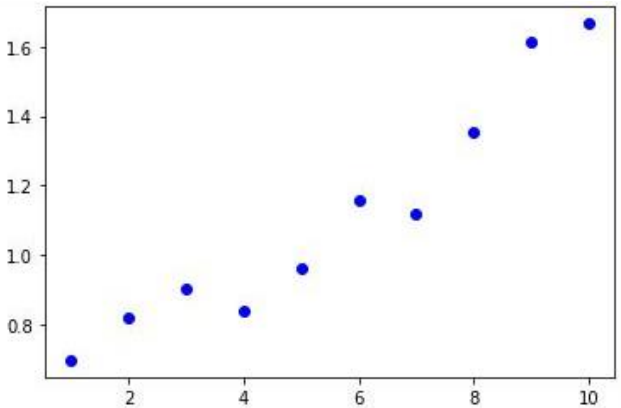
TTA 딥러닝 모델 학습 결과

테스트 데이터셋에서의 정확도



Epoch=4일 때, 가장 높은 정확도를 보이고 있으며 이 때 **정확도는 65%**

테스트 데이터셋에서의 손실값



훈련한 epoch 수에 비례하여 손실값이 우상향해 **1.66을 달성**