외형 특징에 따른

성별 분류 머신러닝 Project.

3조 Machine learning

순서 a table of contents

주제 선정 배경

- 얼굴 특징에 따른 성별 예측은 보안, 인공지능 카메라 시스템 등 다양한 산업에서 중요한 역할을 한다

- 간단한 머신러닝 모델을 사용해 봄으로써 생체 데이터를 분석하는 방법을 배울 수 있다

결정 트리

Decision Tree

안효준

Part 1

데이터 불러오기

```
gender = pd.read_csv('gender_classification_v7.csv')
gender
```

✓ 0.1s Python

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long	gender
0	1	11.8	6.1	1	0	1	1	Male
1	0	14.0	5.4	0	0	1	0	Female
2	0	11.8	6.3	1	1	1	1	Male
3	0	14.4	6.1	0	1	1	1	Male
4	1	13.5	5.9	0	0	0	0	Female
...	...	...	...	...	...	...	...	...
4996	1	13.6	5.1	0	0	0	0	Female
4997	1	11.9	5.4	0	0	0	0	Female
4998	1	12.9	5.7	0	0	0	0	Female
4999	1	13.2	6.2	0	0	0	0	Female
5000	1	15.4	5.4	1	1	1	1	Male

5001 rows × 8 columns

5001 x 8 데이터

출처: 캐글(Kaggle)

데이터 정보 확인

```
gender.info()
```

✓ 0.0s Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001 entries, 0 to 5000
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   long_hair                 5001 non-null   int64
 1   forehead_width_cm         5001 non-null   float64
 2   forehead_height_cm        5001 non-null   float64
 3   nose_wide                 5001 non-null   int64
 4   nose_long                 5001 non-null   int64
 5   lips_thin                 5001 non-null   int64
 6   distance_nose_to_lip_long 5001 non-null   int64
 7   gender                    5001 non-null   object
dtypes: float64(2), int64(5), object(1)
memory usage: 312.7+ KB
```

> 컬럼에 대한 설명

- long hair : 머리가 긴지 여부 (0: 짧음, 1: 길)
- forehead_width_cm : 이마 너비 (단위: cm)
- forehead_height_cm : 이마 높이 (단위: cm)
- nose_wide : 코가 넓은지 여부 (0: 좁음, 1: 넓음)
- nose_long : 코가 긴지 여부 (0: 짧음, 1: 길)
- lips_thin : 입술이 얇은지 여부 (0: 얇음, 1: 두꺼움)
- distance_nose_to_lip_long : 코와 입술 사이의 거리가 긴지 여부 (0: 짧음, 1: 길)
- gender : 성별 (Male: 남자, Female: 여자)

타겟인 gender를 제외한 모든 피쳐가 숫자로 입력된 데이터

결측치, 이상치, 데이터 불균형 확인

```
gender.isna().sum().sum() # 결측치 없음
```

✓ 0.0s Python

```
0
```

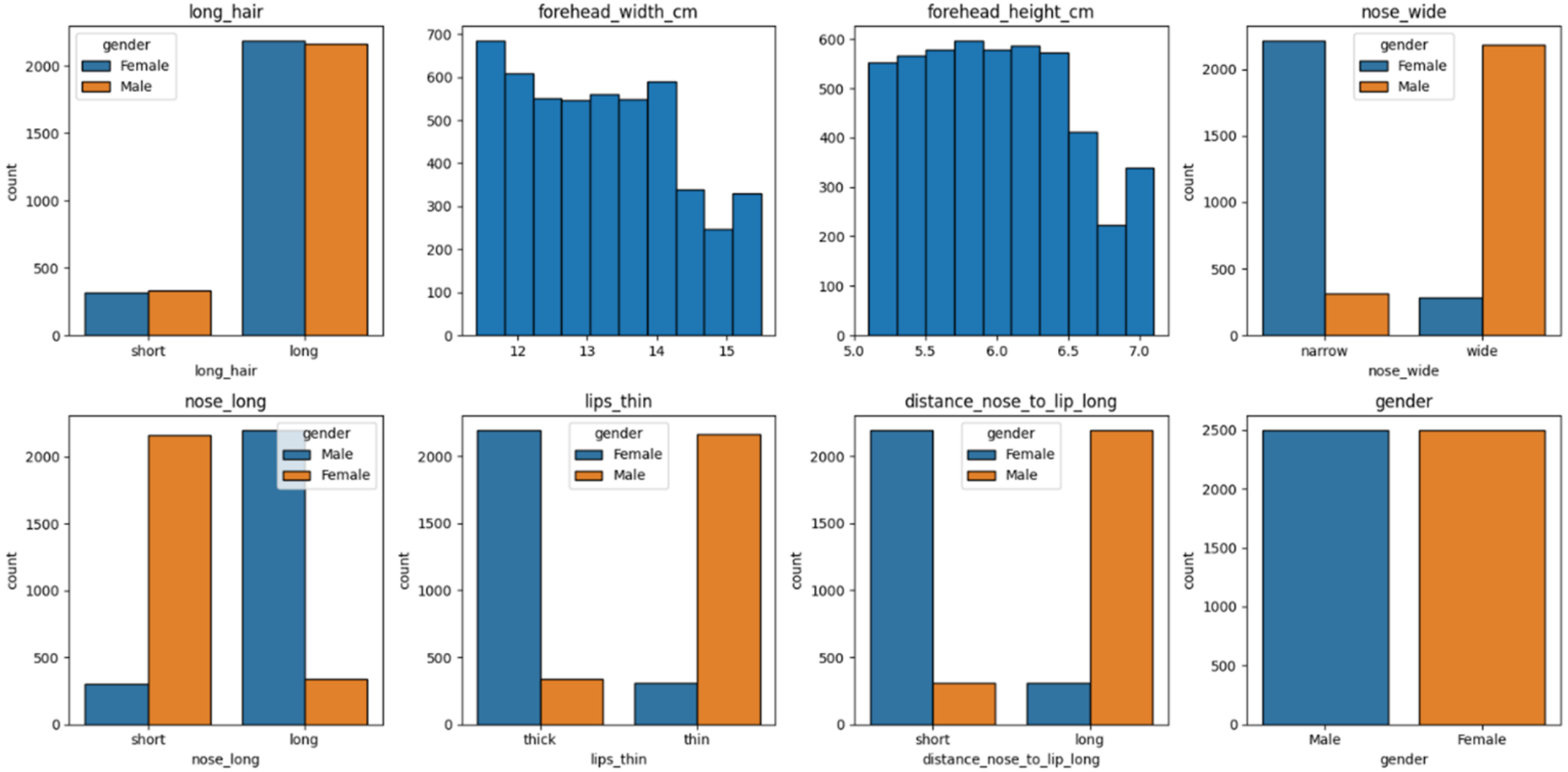

```
gender['gender'].value_counts() # 균형 데이터
```

✓ 0.0s Python

```
gender
Female    2501
Male      2500
Name: count, dtype: int64
```

> 결측치와 이상치가 없으며 균형 잡힌 데이터

컬럼들의 분포 시각화



성별에 따라 **분포 차이**를 보이는 피쳐가 존재

부트스트랩 샘플링 (오버 샘플링)

- 샘플 수가 부족해서 1000개 추가 샘플링

+ Code + Markdown

```
bootstrap_samples = gender.sample(n=1000, replace=True, random_state=10)
boot_gender = pd.concat([gender, bootstrap_samples]).reset_index(drop=True)
boot_gender
```

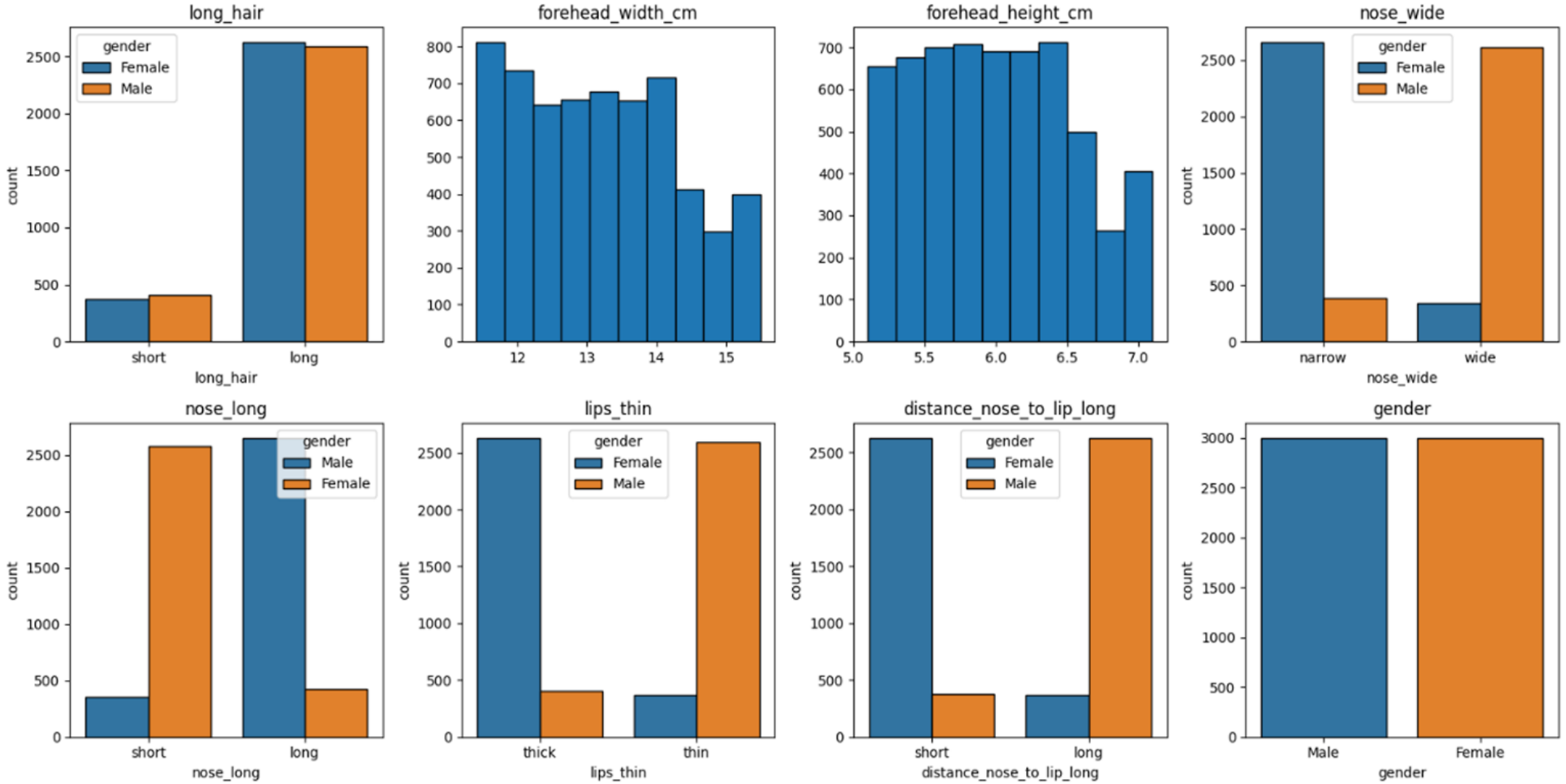
✓ 0.0s Python

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long	gender
0	1	11.8	6.1	1	0	1	1	Male
1	0	14.0	5.4	0	0	1	0	Female
2	0	11.8	6.3	1	1	1	1	Male
3	0	14.4	6.1	0	1	1	1	Male
4	1	13.5	5.9	0	0	0	0	Female
...	...	...	...	...	...	...	...	...
5996	1	13.1	5.6	1	1	0	0	Male
5997	1	13.5	5.9	0	1	0	0	Female
5998	1	13.1	5.6	0	0	1	0	Female
5999	1	12.3	5.8	1	0	0	1	Male
6000	1	13.1	6.6	1	1	1	1	Male

6001 rows × 8 columns

1000개 복원 추출하여 6001개의 행으로 변경

샘플링 후 컬럼들의 분포 시각



이전의 데이터 분포를 유지하면서 잘 샘플링 됨

훈련/테스트 세트 분리

```
featureDF = boot_gender.iloc[:, :-1]
targetSR = boot_gender['gender']
```

✓ 0.0s Python

```
X_train, X_test, y_train, y_test = train_test_split(featureDF, targetSR, stratify = targetSR,
                                                    train_size = 0.8, random_state = 10)
```

✓ 0.0s Python

```
print(f"X_train : {X_train.shape}, y_train : {y_train.shape}")
print(f"X_test : {X_test.shape}, y_test : {y_test.shape}")
```

✓ 0.0s Python

```
X_train : (4800, 7), y_train : (4800,)
X_test : (1201, 7), y_test : (1201,)
```

훈련 세트: 4800개, 테스트 세트: 1201개

Part 1

수치형 피쳐 스케일링

MinMax Scaler

```
X_train_mm
```

✓ 0.0s

	long_hair	forehead_width_cm	forehead_height_cm
2265	1	0.487805	0.20
1721	1	0.707317	0.40
3159	1	0.243902	0.75
3642	1	0.951220	0.70
3420	1	0.512195	0.65
...	...	...	...
3594	1	0.121951	0.40
3189	1	0.121951	0.70
4734	1	1.000000	0.95
4726	1	0.487805	0.30
1370	1	0.682927	0.25

4800 rows × 7 columns

Standard Scaler

```
X_train_ss
```

✓ 0.0s

	long_hair	forehead_width_cm	forehead_height_cm
2265	1	0.187940	-0.816103
1721	1	1.001060	-0.076712
3159	1	-0.715527	1.217223
3642	1	1.904526	1.032375
3420	1	0.278286	0.847527
...	...	...	...
3594	1	-1.167260	-0.076712
3189	1	-1.167260	1.032375
4734	1	2.085219	1.956615
4726	1	0.187940	-0.446408
1370	1	0.910713	-0.631255

4800 rows × 7 columns

GridSearchCV로 하이퍼 파라미터 튜닝

max_depth: 트리의 최대 깊이 지정
min_samples_split: 노드를 분할하기 위한 최소 샘플 수 지정
min_samples_leaf: 리프 노드가 가져야 하는 최소 샘플 수 지정
max_features: 각 노드에서 분할에 사용할 특성의 최대 수 지정
criterion: 노드에서의 분할 기준 선택 (기본값은 'gini'이며, 'entropy'도 사용 가능)

```python
# 탐색할 하이퍼 파라미터 그리드 설정
param_dt = {
    'max_depth': [1, 3, 5, 7],
    'min_samples_split': [2, 4, 6, 8, 10],
    'min_samples_leaf': [1, 2, 4, 6, 8],
    'max_features': ['sqrt', 'log2'],
    'criterion': ['gini', 'entropy']
}
```

✓ 0.0s Python

탐색을 진행할 하이퍼 파라미터 그리드 설정

최적의 모델 평가 지표 확인

MinMax 스케일링

----------훈련 세트 평가 지표----------

	precision	recall	f1-score	support
0	0.98	0.95	0.97	2401
1	0.96	0.98	0.97	2399
accuracy			0.97	4800
macro avg	0.97	0.97	0.97	4800
weighted avg	0.97	0.97	0.97	4800

----------테스트 세트 평가 지표----------

	precision	recall	f1-score	support
0	0.97	0.96	0.96	601
1	0.96	0.97	0.96	600
accuracy			0.96	1201
macro avg	0.96	0.96	0.96	1201
weighted avg	0.96	0.96	0.96	1201

Standard 스케일링

----------훈련 세트 평가 지표----------

	precision	recall	f1-score	support
0	0.98	0.95	0.97	2401
1	0.96	0.98	0.97	2399
accuracy			0.97	4800
macro avg	0.97	0.97	0.97	4800
weighted avg	0.97	0.97	0.97	4800

----------테스트 세트 평가 지표----------

	precision	recall	f1-score	support
0	0.97	0.96	0.96	601
1	0.96	0.97	0.96	600
accuracy			0.96	1201
macro avg	0.96	0.96	0.96	1201
weighted avg	0.96	0.96	0.96	1201

기본 데이터

----------훈련 세트 평가 지표----------

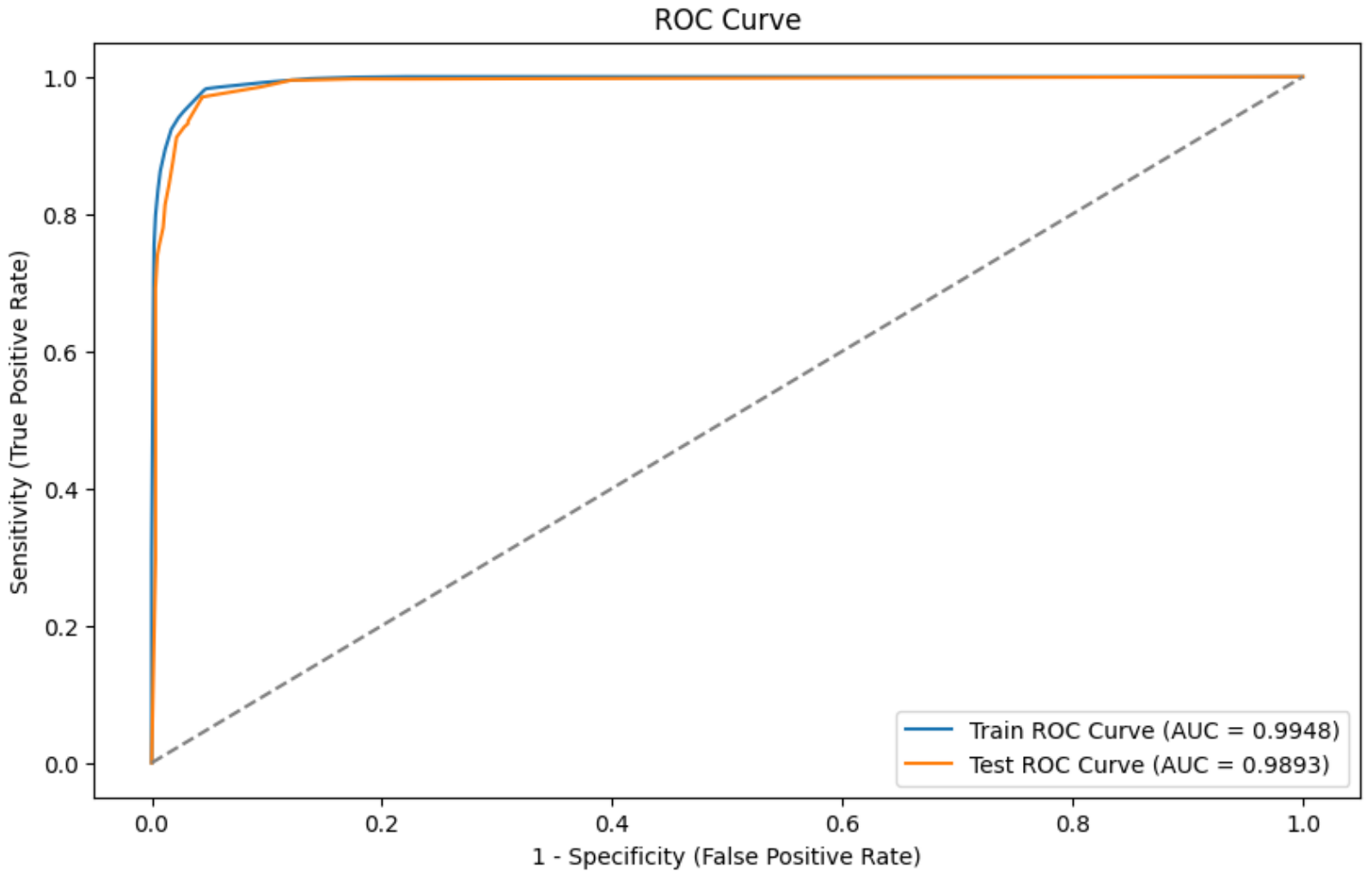
	precision	recall	f1-score	support
0	0.98	0.95	0.97	2401
1	0.96	0.98	0.97	2399
accuracy			0.97	4800
macro avg	0.97	0.97	0.97	4800
weighted avg	0.97	0.97	0.97	4800

----------테스트 세트 평가 지표----------

	precision	recall	f1-score	support
0	0.97	0.96	0.96	601
1	0.96	0.97	0.96	600
accuracy			0.96	1201
macro avg	0.96	0.96	0.96	1201
weighted avg	0.96	0.96	0.96	1201

Decision Tree는 스케일링에 따른 성능 차이가 없었다 (MinMax로 진행)

ROC Curve, AUC 확인



훈련 세트와 테스트 세트 **모두 1에 가깝게** 그려진다

피쳐 최적화

```python
feature_importances_dict = {key : value for key,
                            value in zip(dt_grid_search.best_estimator_.feature_names_in_,
                                         dt_grid_search.best_estimator_.feature_importances_.round(3))}

sorted(feature_importances_dict.items(), key = lambda x : x[1], reverse = True)
```

Python

```
[('distance_nose_to_lip_long', 0.443),
 ('forehead_width_cm', 0.208),
 ('nose_wide', 0.139),
 ('lips_thin', 0.108),
 ('forehead_height_cm', 0.058),
 ('nose_long', 0.043),
 ('long_hair', 0.001)]
```

변수 중요도가 높은 순서로 **4개**, **3개** 넣어보기

최적화된 피쳐로 평가 지표 확인

피쳐 4개

----------훈련 세트 평가 지표----------

	precision	recall	f1-score	support
0	0.96	0.97	0.97	2401
1	0.97	0.96	0.97	2399
accuracy			0.97	4800
macro avg	0.97	0.97	0.97	4800
weighted avg	0.97	0.97	0.97	4800

----------테스트 세트 평가 지표----------

	precision	recall	f1-score	support
0	0.96	0.97	0.96	601
1	0.97	0.96	0.96	600
accuracy			0.96	1201
macro avg	0.96	0.96	0.96	1201
weighted avg	0.96	0.96	0.96	1201

피쳐 3개

----------훈련 세트 평가 지표----------

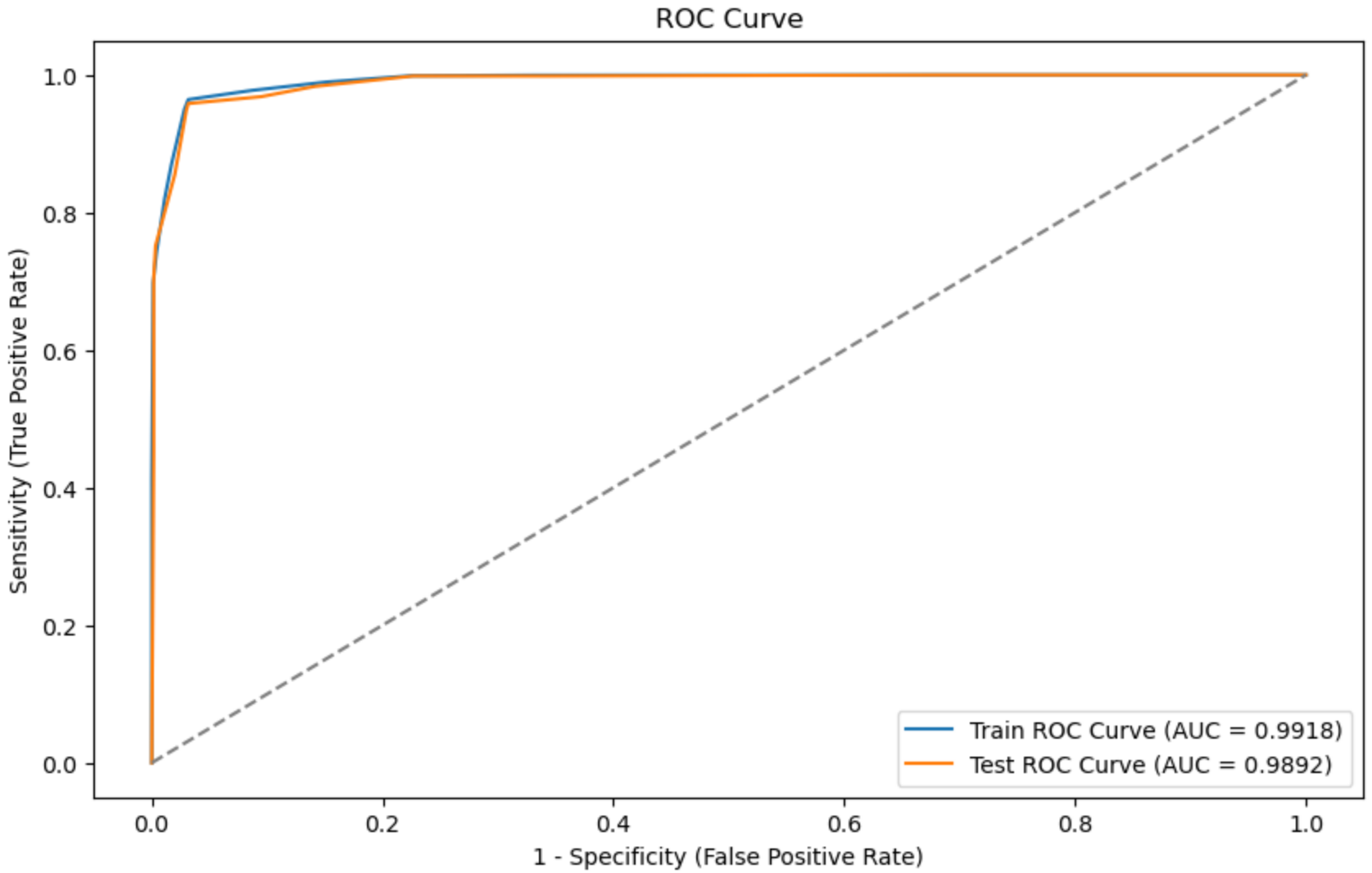
	precision	recall	f1-score	support
0	0.95	0.89	0.92	2401
1	0.90	0.95	0.92	2399
accuracy			0.92	4800
macro avg	0.92	0.92	0.92	4800
weighted avg	0.92	0.92	0.92	4800

----------테스트 세트 평가 지표----------

	precision	recall	f1-score	support
0	0.93	0.88	0.91	601
1	0.89	0.94	0.91	600
accuracy			0.91	1201
macro avg	0.91	0.91	0.91	1201
weighted avg	0.91	0.91	0.91	1201

피쳐가 3개로 줄어든 순간 성능이 많이 줄어든다 (피쳐 4개 선택)

최종 모델의 ROC Curve, AUC



모든 피쳐를 넣은 모델과 성능 차이가 거의 없다

분석 결과 (Decision Tree)

- 피쳐 : **‘distance_nose_to_lip_long’, ‘forehead_width_cm’, ‘nose_wide’, ‘lips_thin’**

- 스케일링 : **MinMaxScaler()**

- 하이퍼 파라미터 : **criterion = ‘entropy’, max_depth = 5, max_features = ‘sqrt’, min_samples_leaf = 1, min_samples_split = 2**

훈련 세트 정확도 : **0.97**
테스트 세트 정확도 : **0.96**

임시 데이터로 모델 시연해보기

```python
# 남자의 특징이 있는 데이터
man_feature_data = pd.DataFrame([[1, X_train_mm['forehead_width_cm'].mean(), 1, 1]],
                                columns=['distance_nose_to_lip_long', 'forehead_width_cm',
                                         'nose_wide', 'lips_thin'])

woman_feature_data = pd.DataFrame([[0, X_train_mm['forehead_width_cm'].mean(), 0, 0]],
                                  columns=['distance_nose_to_lip_long', 'forehead_width_cm',
                                           'nose_wide', 'lips_thin'])
```

✓ 0.0s Python

```python
dt_grid_search.best_estimator_.predict(man_feature_data)
```

✓ 0.0s Python

```
array([0])
```

<- **0(남성)**으로 예측

```python
dt_grid_search.best_estimator_.predict(woman_feature_data)
```

✓ 0.0s Python

```
array([1])
```

<- **1(여성)**으로 예측

성별을 잘 예측하는 것으로 보인다

최종 결론

종합적으로 좋은 성능을 보인 모델은
KNN과 **RandomForest**였다

지금은 비록 간단한 성별 예측을 하는 이진 분류 모델을 만들어보았지만 이러한 생체 인식 기술을 발전시킨다면 **cctv**를 이용한 범인 특정과 같은 보안 기술에 활용할 수 있을 것으로 기대된다.

THANK YOU