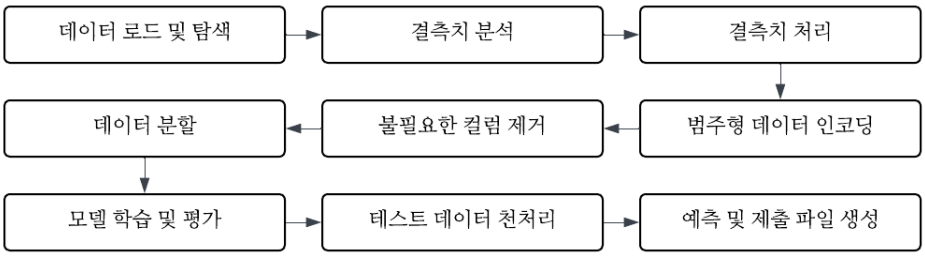
# 3주차 결과보고서

전공 : 경영학과 학년 : 4학년 학번 : 20190808 이름 : 방지혁

**1. 실험시간에 작성한 코드에 전체적인 진행 흐름을 diagram으로 그리고 각 단계별로 어떤 기능을 구현했는지 서술하세요.**




1단계는 데이터 로드 및 탐색 단계입니다.

```
train = pd.read_csv("/content/train.csv")
test = pd.read_csv("/content/test.csv")
print(train.shape, test.shape)
```

Titanic의 train과 test dataset을 모두 불러옵니다. 데이터 기본 정보를 불러오면 891, 12) (418, 11) 이라는 것을 확인할 수 있습니다.

2단계는 결측치 분석 단계입니다.

```
print("column 별 결측치 비율: ")
missing_ratio = train.isnull().sum() / len(train) * 100
for col, ratio in missing_ratio.items():
    print(f"{col:<12}   :    {ratio:.2f}%")

print("결측치를 포함하고 있는 row 수 : ", train.isnull().any(axis=1).sum())
```

컬럼별 결측치 비율을 계산했더니 Age: 19.87%, Cabin: 77.10%, Embarked: 0.22%라는 것을 확인할 수 있었습니다. 해당 row 개수는 708개입니다.

3단계는 결측치 처리 단계입니다.

```
train.Age = train['Age'].fillna(train.Age.median())
train = train.drop(columns='Cabin')
train.Embarked = train.Embarked.fillna('C')
```

해당 코드와 같이 Age 컬럼은 중앙값으로 대체, Cabin 컬럼은 결측치 비율이 너무 높이게 전체 컬럼 자체를 삭제하고, Embarked 컬럼은 'C'로 해당 결측치를 대체합니다.

4단계는 범주형 데이터 인코딩 단계입니다.

```
train = pd.get_dummies(train, columns=['Sex', 'Embarked'])
train
```

해당 one-hot encoding은 sex(성별), embarked(승선 위치)에 대해서 이뤄졌습니다. Sex에 대해 예를 들어 설명하자면 머신러닝은 숫자만 이해합니다. Male과 female은 1과 2로 되어 있는데 이 상태에서는 남성 < 여성이라는 잘못된 관계를 만들기 때문입니다.

5단계는 불필요한 컬럼 제거입니다.

```
drop_cols = ['PassengerId', 'Name', 'Ticket']
train = train.drop(columns = drop_cols)
train
```

이는 passengerid, name, ticket으로 생존 예측에 직접적 영향을 주지 않기 때문에 제거해줍니다.

6단계는 데이터 분할입니다.

```
from sklearn.model_selection import train_test_split
# feature vector
X = train.drop('Survived', axis = 1)
# target value
y = train.Survived
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.25,
random_state=42)
print(X_train.shape, X_val.shape, y_train.shape, y_val.shape)
```

Sklearn의 train_test_split 함수를 가져옵니다. feature vector인 X를 정의하는데 입력 변수만 남겨야 하기 때문에 train 데이터프레임에서 survived 칼럼을 제거합니다. 그리고 target value인 y를 survived 컬럼만 따로 추출해서 정의합니다. 이후 이것들을 3:1 비율로 분할하여 훈련용, 검증용 데이터를 추출합니다.

7단계는 모델 학습 및 평가 단계입니다.

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
# 학습한 모델을 평가
from sklearn.metrics import accuracy_score
pred_train = clf.predict(X_train)
pred_val = clf.predict(X_val)
print("Train ACC : %.4f" % accuracy_score(y_train, pred_train))
print("Validation ACC : %.4f" % accuracy_score(y_val, pred_val))
```

RandomForestClassifier를 사용하여 Train Accuracy: 98.05%, Validation Accuracy: 78.48%라는 결과를 도출할 수 있었습니다.

이후 8단계는 테스트 데이터 전처리 단계로 fare 컬럼의 결측치는 train 데이터의 평균값으로 대체합니다. 이전의 단계와 동일하기에 코드는 생략하겠습니다.

9단계는 예측 및 submission.csv라는 제출 파일 생성입니다.

**2. Random Forest Classifier의 hyper-parameter를 바꿔가면서 실험을 해보세요. 그리고 다음 3가지의 실험을 수행했을 때 결과를 설명해보세요.**

**2-1) max_depth를 3에서 15까지 바꿔가면서 실험**

```
# 예측에 사용할 모델을 가져와서 학습
from sklearn.ensemble import RandomForestClassifier
# 학습한 모델을 평가
from sklearn.metrics import accuracy_score

max_depths = range(3, 16)

for depth in max_depths:
  clf = RandomForestClassifier(max_depth=depth, random_state=42)
  clf.fit(X_train, y_train)

  pred_train = clf.predict(X_train)
  pred_val = clf.predict(X_val)

  print("Train ACC : %.4f" % accuracy_score(y_train, pred_train))
  print("Validation ACC : %.4f" % accuracy_score(y_val, pred_val))
```

```
Train ACC : 0.8308
Validation ACC : 0.7982
Train ACC : 0.8548
Validation ACC : 0.8117
Train ACC : 0.8653
Validation ACC : 0.7937
Train ACC : 0.8817
Validation ACC : 0.8117
Train ACC : 0.8997
Validation ACC : 0.8117
Train ACC : 0.9162
Validation ACC : 0.8341
Train ACC : 0.9401
Validation ACC : 0.7982
Train ACC : 0.9446
Validation ACC : 0.8072
Train ACC : 0.9491
Validation ACC : 0.8117
Train ACC : 0.9611
Validation ACC : 0.7848
Train ACC : 0.9656
Validation ACC : 0.7937
Train ACC : 0.9731
Validation ACC : 0.7937
Train ACC : 0.9760
Validation ACC : 0.7758
```

Max_depth가 중가할 수록 train data에 대해서는 정확도가 상승합니다. 그러나 test data에 대해서는 8일때 가장 좋은 양상이 보여지지만 그 이상일 때는 오히려 오버 피팅으로 성능이 하락하는 양상이 보여집니다. Validation 성능이 제일 높은 8인 경우가 좋아 보입니다.

다음 장에 이어서

**2-2) max_features를 0.3에서 1.0까지 바꿔가면서 실험**

```python
# 예측에 사용할 모델을 가져와서 학습
from sklearn.ensemble import RandomForestClassifier
# 학습한 모델을 평가
from sklearn.metrics import accuracy_score

max_features_range = [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

for max_feature in max_features_range:
  clf = RandomForestClassifier(max_features=max_feature, random_state=42)
  clf.fit(X_train, y_train)

  pred_train = clf.predict(X_train)
  pred_val = clf.predict(X_val)

  print("max_features: %.1f" %max_feature)
  print("Train ACC : %.4f" % accuracy_score(y_train, pred_train))
  print("Validation ACC : %.4f" % accuracy_score(y_val, pred_val))
```

```
max_features: 0.3
Train ACC : 0.9805
Validation ACC : 0.7803
max_features: 0.4
Train ACC : 0.9805
Validation ACC : 0.7848
max_features: 0.5
Train ACC : 0.9805
Validation ACC : 0.7848
max_features: 0.6
Train ACC : 0.9805
Validation ACC : 0.7758
max_features: 0.7
Train ACC : 0.9805
Validation ACC : 0.7668
max_features: 0.8
Train ACC : 0.9805
Validation ACC : 0.7713
max_features: 0.9
Train ACC : 0.9805
Validation ACC : 0.7758
max_features: 1.0
Train ACC : 0.9805
Validation ACC : 0.7803
```

max_features가 0.4, 0.5일 때 validation accurracy에서 최고의 성능을 보였습니다. 모든 경우에서 train data에 대해 98.05%가 나타났는데 이는 max_depth를 설정하지 않아 트리가 끝까지 깊어져서 개별 훈련 샘플까지 완벽 학습한 결과라고 볼 수 있습니다.

**2-3) n_estimators를 50에서 1000까지 바꿔가면서 실험**

```python
# 예측에 사용할 모델을 가져와서 학습
from sklearn.ensemble import RandomForestClassifier
# 학습한 모델을 평가
from sklearn.metrics import accuracy_score

n_estimators_range = [50, 100, 200, 300, 500, 700, 1000]

for n_est in n_estimators_range:
  clf = RandomForestClassifier(n_estimators=n_est, random_state=42)
  clf.fit(X_train, y_train)

  pred_train = clf.predict(X_train)
  pred_val = clf.predict(X_val)

  print("n_est: %d" %n_est)
  print("Train ACC : %.4f" % accuracy_score(y_train, pred_train))
  print("Validation ACC : %.4f" % accuracy_score(y_val, pred_val))
```

```
n_est: 50
Train ACC : 0.9760
Validation ACC : 0.7937
n_est: 100
Train ACC : 0.9805
Validation ACC : 0.7803
n_est: 200
Train ACC : 0.9805
Validation ACC : 0.7892
n_est: 300
Train ACC : 0.9805
Validation ACC : 0.7848
n_est: 500
Train ACC : 0.9805
Validation ACC : 0.7758
n_est: 700
Train ACC : 0.9805
Validation ACC : 0.7758
n_est: 1000
Train ACC : 0.9805
Validation ACC : 0.7758
```

N_estimators가 50일 때 validation이 최고의 성능을 보인 것을 확인할 수 있었습니다. 그 이후로는 오히려 오버피팅이 증가합니다. 트리가 증가하면서 필요 없는 세부 노이즈(특이 케이스)까지 학습하는 것으로 보여집니다.

**3. 제출했을 때의 test accuracy가 train, validation accuracy보다 낮다면 왜 차이가 발생했는지에 대한 생각을 서술해보세요.**

과적합 때문이라고 볼 수 있습니다. 모델이 너무 훈련 데이터에 맞춰져서, 해당 데이터의 노이즈, 특이 패턴까지 학습한 것입니다. 그렇기에 테스트 데이터에 대한 일반화 능력이 떨어진 것입니다. 이를 해결하기 위해 앞서 우리가 한 파라미터 조정을 통해 적당한 균형을 찾는 것이 중요해 보입니다. 또한, 훈련 데이터 양 자체가 애초에 적어 전체 데이터 시나리오를 대표하지 못하는 것일수도 있습니다.