기계학습기초 팀 프로젝트 보고서

Titanic – Machine Learning from Disaster

7 조

201804235 이현중

202101297 이소정

201803225 한만형

목차

1. 목표

팀 별 프로젝트를 통해, 수업에서 학습한 분류 이론 및 실습 관련 실전 데이터 경험과 협업 경험을 쌓는다. 가장 높은 정확도를 목표로 할 것

2. 데이터 전처리 과정

주어진 데이터

- Passengerid, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked
- 각 데이터를 가공하고 유효한지 비교하여 선정하는 것이 주요
- 우선 Passengerid 는 각 승객의 고유번호이고, Ticket 의 경우 영향이 없다고 판단하여 사용하지 않음

Train

```
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Test

```
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

- 결측값

Train

PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

dtype: int64

Test

PassengerId	0
Pclass	0
Name	0
Sex	0
Age	86
SibSp	0
Parch	0
Ticket	0
Fare	1
Cabin	327
Embarked	0

dtype: int64

데이터 결측값 채우기 및 가공 과정

Age

- Age 의 경우 단순 평균으로 채워넣기에는 승객의 연령 분포도가 드러난 것만 해도 다양함, 연령대에 맞는 평균을 넣어줄 필요가 있음
- 우선 연령대를 맞추기 위해, Name 의 앞 부분을 추출해 Title 이라는 새로운 항목으로 만들어 냄, 앞의 칭호에 따라 연령대가 어느 정도 비슷할 것으로 추측

```
train['Title'] = train['Name'].str.extract(' ([A-Za-z]+)₩.', expand=False)
test['Title'] = test['Name'].str.extract(' ([A-Za-z]+)₩.', expand=False)

train['Title'].value_counts()
```

```
Mr          517
Miss        182
Mrs         125
Master       40
Dr            7
Rev           6
Mlle          2
Major         2
Col           2
Countess      1
Capt          1
Ms            1
Sir           1
Lady          1
Mme           1
Don           1
Jonkheer      1
Name: Title, dtype: int64
```

- 수가 적은 Title 항목은 단일화하고, Test 데이터도 똑같이 진행

```
Mr        517
Miss      185
Mrs       126
Master     40
Other      23
Name: Title, dtype: int64
```

```
Mr        240
Miss       79
Mrs        72
Master     21
Other       6
Name: Title, dtype: int64
```

- Title 별 평균 연령에 따라 Age 의 결측값을 채움

```
train["Age"].fillna(train.groupby("Title")["Age"].transform("median"), inplace=True)
test["Age"].fillna(test.groupby("Title")["Age"].transform("median"), inplace=True)
```

- 그 후, Age 는 구간화

```
# Train
train.loc[ train['Age'] <= 10, 'Age_clean'] = 0
train.loc[(train['Age'] > 10) & (train['Age'] <= 16), 'Age_clean'] =
train.loc[(train['Age'] > 16) & (train['Age'] <= 20), 'Age_clean'] =
train.loc[(train['Age'] > 20) & (train['Age'] <= 26), 'Age_clean'] =
train.loc[(train['Age'] > 26) & (train['Age'] <= 30), 'Age_clean'] =
train.loc[(train['Age'] > 30) & (train['Age'] <= 36), 'Age_clean'] =
train.loc[(train['Age'] > 36) & (train['Age'] <= 40), 'Age_clean'] =
train.loc[(train['Age'] > 40) & (train['Age'] <= 46), 'Age_clean'] =
train.loc[(train['Age'] > 46) & (train['Age'] <= 50), 'Age_clean'] =
train.loc[(train['Age'] > 50) & (train['Age'] <= 60), 'Age_clean'] =
train.loc[ train['Age'] > 60, 'Age_clean'] = 10

# Test
test.loc[ test['Age'] <= 10, 'Age_clean'] = 0
test.loc[(test['Age'] > 10) & (test['Age'] <= 16), 'Age_clean'] = 1
test.loc[(test['Age'] > 16) & (test['Age'] <= 20), 'Age_clean'] = 2
test.loc[(test['Age'] > 20) & (test['Age'] <= 26), 'Age_clean'] = 3
test.loc[(test['Age'] > 26) & (test['Age'] <= 30), 'Age_clean'] = 4
test.loc[(test['Age'] > 30) & (test['Age'] <= 36), 'Age_clean'] = 5
test.loc[(test['Age'] > 36) & (test['Age'] <= 40), 'Age_clean'] = 6
test.loc[(test['Age'] > 40) & (test['Age'] <= 46), 'Age_clean'] = 7
test.loc[(test['Age'] > 46) & (test['Age'] <= 50), 'Age_clean'] = 8
test.loc[(test['Age'] > 50) & (test['Age'] <= 60), 'Age_clean'] = 9
test.loc[ test['Age'] > 60, 'Age_clean'] = 10
```

Cabin

- 결측값의 비중이 훨씬 높지만, 사용하기로 결정하고, Cabin 의 앞 문자만 활용

```
train['Cabin'].str[:1].value_counts()
```

```
C    59
B    47
D    33
E    32
A    15
F    13
G     4
T     1
Name: Cabin, dtype: int64
```

- 앞 글자를 숫자로 변환

```
mapping = {
    'A': 0,
    'B': 1,
    'C': 2,
    'D': 3,
    'E': 4,
    'F': 5,
    'G': 6,
    'T': 7
}
```

- Map 함수 사용해 여러 데이터의 형태를 한 번에 변경

```
train['Cabin_clean'] = train['Cabin'].str[:1]
train['Cabin_clean'] = train['Cabin_clean'].map(mapping)
```

- Age 와 같이, 연관되어 있는 항목인 Pclass 로 그룹화한 중앙값을 넣어줌

```
train['Cabin_clean'] = train.groupby('Pclass')['Cabin_clean'].transform('median')
```

- Test 데이터도 변환

```
test['Cabin_clean'] = test['Cabin'].str[:1]
test['Cabin_clean'] = test['Cabin_clean'].map(mapping)
test['Cabin_clean'] = test.groupby('Pclass')['Cabin_clean'].transform('median')
```

Emabrked

- Embarked 값 확인

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

- S 의 비중이 압도적으로 높다는 점에 기인해 결측값 2 개는 S 로 대체

```
train['Embarked'].fillna('S', inplace=True)
```

- Embarked 항목 또한 숫자로 변경, 카테고리 형식으로 변경

```
train['Embarked_clean'] = train['Embarked'].astype('category').cat.codes
test['Embarked_clean'] = test['Embarked'].astype('category').cat.codes
```

Fare

- Fare 값도 5 구간화, pd.qcut 활용

```
train['FareBin'] = pd.qcut(train['Fare'], 5)
test['FareBin'] = pd.qcut(test['Fare'], 5)

train['FareBin'].value_counts()
```

```
(7.854, 10.5]      184
(21.679, 39.688]   180
(-0.001, 7.854]    179
(39.688, 512.329]  176
(10.5, 21.679]     172
Name: FareBin, dtype: int64
```

- 단순한 값으로 변경

```
train['Fare_clean'] = train['FareBin'].astype('category').cat.codes
test['Fare_clean'] = test['FareBin'].astype('category').cat.codes

train['Fare_clean'].value_counts()
```

```
1    184
3    180
0    179
4    176
2    172
Name: Fare_clean, dtype: int64
```

Sex

- 결측값이 없으므로, 카테고리화로 숫자로 변경

```
train['Sex_clean'] = train['Sex'].astype('category').cat.codes
test['Sex_clean'] = test['Sex'].astype('category').cat.codes
```

SibSp, Parch

- 두 특성을 활용해 혼자 탑승한 사람, 아닌 사람을 구분하기로 결정
- Sibsp(함께 탑승한 형제 혹은 배우자 수), Parch(함께 탑승한 부모 또는 자녀 수)
- 두 항목을 합친 것에 1 을 더하면 함께 탄 가족의 총 숫자가 됨

```
train['Family'] = train['SibSp'] + train['Parch'] + 1
test['Family'] = test['SibSp'] + test['Parch'] + 1
```

```
0      2
1      2
2      1
3      2
4      1
      ..
886    1
887    1
888    4
889    1
890    1
```

- 그렇게 만들어진 Faimly 의 값이 1 이면 혼자 탄 것

```
train['Solo'] = (train['Family'] == 1)
test['Solo'] = (test['Family'] == 1)
```

- 가공하고, 사용하기로 한 특성

  Pclass, Sibsp, Parch, Sex, Embarked, Family, Solo, Title, Age, Cabin, Fare

- 사용 이유

  가공한 특성만 사용해보기, 결측값 있는 항목 아예 삭제하기 등 여러 방법으로 시도했으나 기존에 비해 높은 정확도가 나타나지 않음

3. 분류기 선택(로지스틱 회귀, 랜덤포레스트)

- 출력 변수와 연관이 있는 속성을 잘 제어한다는 가정 하에 좋은 결과를 낼 **로지스틱 회귀**를 선택
- 과대적합을 하이퍼파라미터를 통해 잘 해결할 수 있다 생각하였고, 당시 배운 훈련 방법 중 가장 좋은 결과를 낼 것으로 기대되었던 **랜덤포레스트**를 선정했음
- 결정트리, KNN 등 다양한 분류기를 사용하여 시도해 봤지만 앞서 설명한 2 개의 분류기의 정확도가 가장 높았기 때문.
-

4. 하이퍼파라미터 선정 및 수행 결과

- 사용할 특성 및 타겟 설정
- feature2 = ['Pclass', 'SibSp', 'Parch', 'Sex_clean', 'Embarked_clean', 'Family', 'Solo', 'Title_clean', 'Age_clean', 'Cabin_clean', 'Fare_clean']
- target2 = ['Survived']

4-1. 로지스틱회귀

- 표준화 작업 : 정확한 독립 변수간의 영향력을 확인하고, 표본의 크기에 의해서 독립 변수의 영향력이 민감하게 변화하지 않도록 함

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(train2)
train_scaled = ss.transform(train2)
test_scaled = ss.transform(test2)
```

- 이진 분류

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(train2, target2)
```

- 교차 검증

```
import numpy as np
from sklearn.model_selection import cross_validate
from sklearn.model_selection import StratifiedKFold

splitter = StratifiedKFold(n_splits=5, shuffle=True)
scores = cross_validate(lr, train_scaled, target2, return_train_score=True, cv=splitter)
```

트레인 세트: 0.8044360748223205

검증세트: 0.7878789780930261

- 변수별 중요도 확인

```
print(lr.coef_, lr.intercept_)
```

```
[[-0.91006458 -0.31283129  0.01994204 -2.53915212 -0.17588928 -0.29298642
  -0.68951609 -0.25537667 -0.16362942  0.06509205  0.21750585]] [4.94893991]
```

- 예측 데이터 submission 에 삽입

```
pred = lr.predict(test_scaled)
pred = pd.DataFrame(pred)
submission['Survived'] = pred
submission
```

- 최종 제출 결과

4-2 랜덤포레스트

- 랜덤포레스트 수행, Cross_Validate 를 이용해 교차검증

```
from sklearn.model_selection import cross_validate
from sklearn.ensemble import RandomForestClassifier
```

- 데이터 학습

```
rf = RandomForestClassifier(n_jobs=-1)
rf.fit(train3, target3)
rf.score(train3, target3)
```

0.9102132435465768

- Train 스코어가 너무 높음

- 그리드 서치를 이용해 하이퍼파라미터 구하기

```
from sklearn.model_selection import GridSearchCV
params = {
          'max_depth': range(1, 10, 1),  #깊이
          'n_estimators' : range(30, 100, 10)  #트리의 개수
          }
```

```
gs = GridSearchCV(RandomForestClassifier(), params, n_jobs=-1)
gs.fit(train3, target3)
print(gs.best_params_)
```

```
{'max_depth': 5, 'n_estimators': 60}
```

- 구한 하이퍼파라미터를 이용해 다시 학습

```
rf = RandomForestClassifier(n_estimators = 30, max_depth = 4, n_jobs=-1)
rf.fit(train3, target3)
rf.score(train3, target3)
```

- 5-Fold 교차 검증

```
import numpy as np
from sklearn.model_selection import cross_validate
from sklearn.model_selection import StratifiedKFold

splitter = StratifiedKFold(n_splits=5, shuffle=True)
scores = cross_validate(rf, train3, target3, return_train_score=True, cv=splitter)

print('트레인 세트: ',np.mean(scores['train_score']))
print('검증세트 : ',np.mean(scores['test_score']))
```

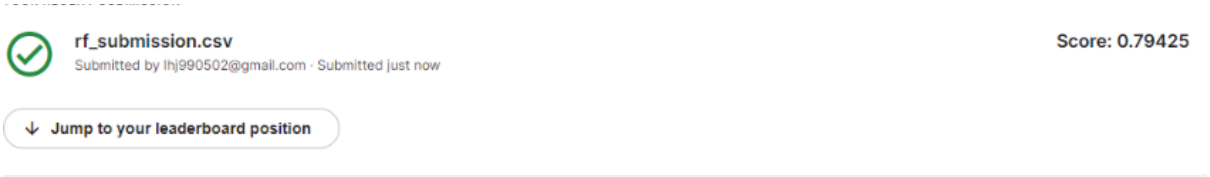
트레인 세트: 0.8423136927368139

검증세트: 0.8327976900382901

- 예측 데이터 submission 에 삽입

```
pred2 = rf.predict(test3)
pred2 = pd.DataFrame(pred2)
submission['Survived'] = pred2
submission
```

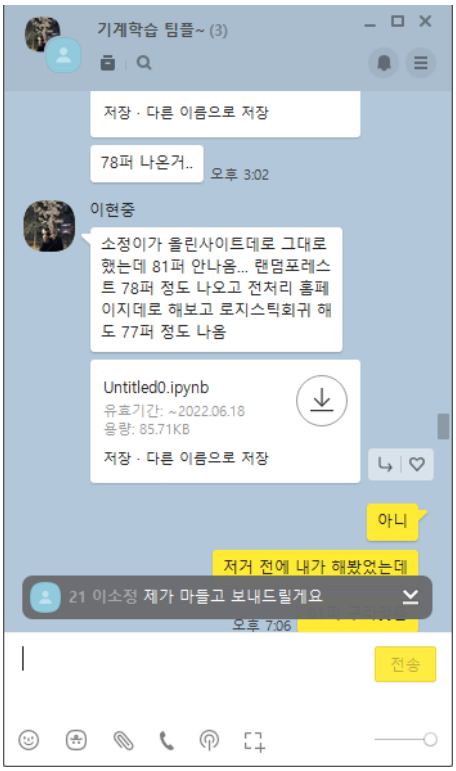
- 최종 제출 결과

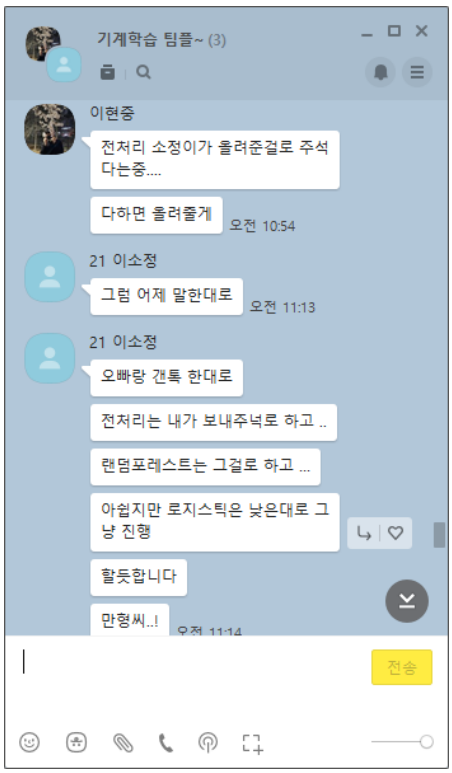


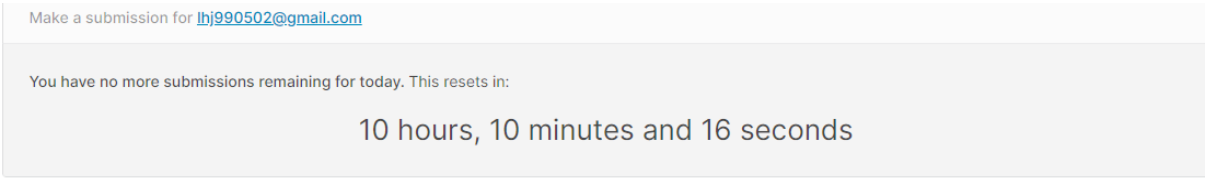
5. 아쉬운 점

- 속성을 잘 제어한다는 전제 하에 좋은 결과를 낼 것으로 기대하였던 로지스틱 회귀의 경우, 여러 속성을 가공하고 선정하는 시행착오를 겪어도 큰 지장이 없는 기간이었음에도 선형 회귀 분석이라는 근본적인 한계점이 있어 정확도를 더 높이기 어려웠음
- 랜덤포레스트의 경우 성능이 굉장히 우수한 방식임을 알 수 있었으나, 트레인 데이터에 큰 변화를 줘도 결과에는 영향력이 적다는 점에서 가공되지 않은 데이터를 삽입했을 때에 비해 큰 변화를 얻기는 어려웠음

6. 마치며







- 한 학기동안 배운 내용을 팀 프로젝트를 통해 다시 한번 공부할 수 있는 계기가 되어 주었고, 결과물과 별개로 협업과 많은 시도를 경험할 수 있었다는 점에서 앞으로도 겪게 될 팀 프로젝트에서의 태도 및 역할에 대해서도 다시 한번 고민해 볼 수 있는 계기가 되었음

- Kaggle 에서 가장 기초적인 타이타닉 데이터에 대해서 제출이 불가할 때까지 여러 번 바꿔 제출해 보고 조금이나마 더 나은 결과를 얻기 위한 노력도 즐거운 경험이었고, 학습 목표에 걸맞게 기계학습의 기초를 닦고 더 관심을 갖게 된 계기가 되어줄 것