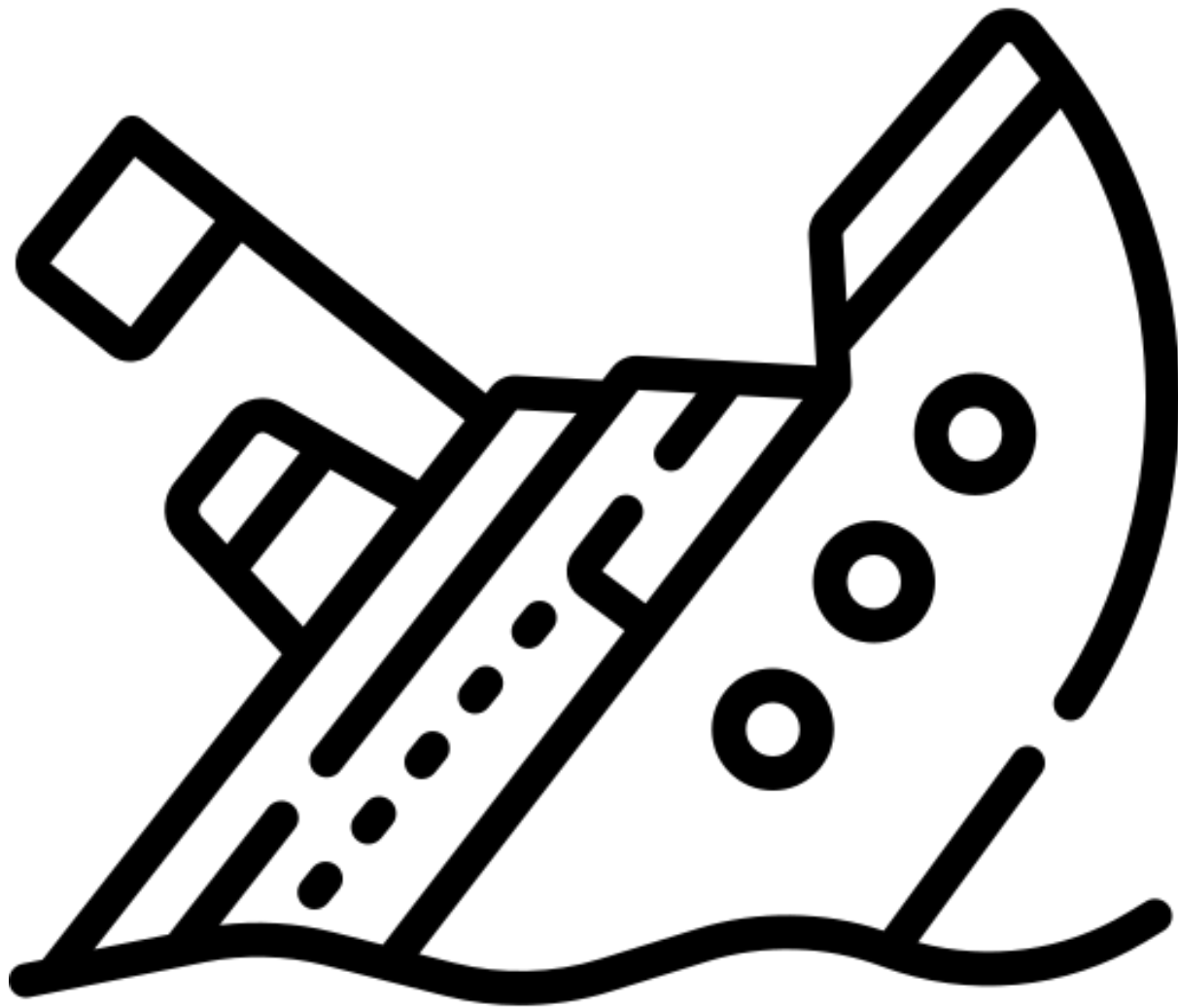


타이타닉 생존율 예측

• 팀 원 : 박윤수 김경태 김혜민 방이현



CONTENTS

01 개요

- 출처 및 목적
- 분석 배경

02 데이터 수집

- 조회
- 데이터 전처리

03 데이터 분석

- 시각화

04 생존자 예측

- SVM
- Random Forest
- Decision Tree

05 결론

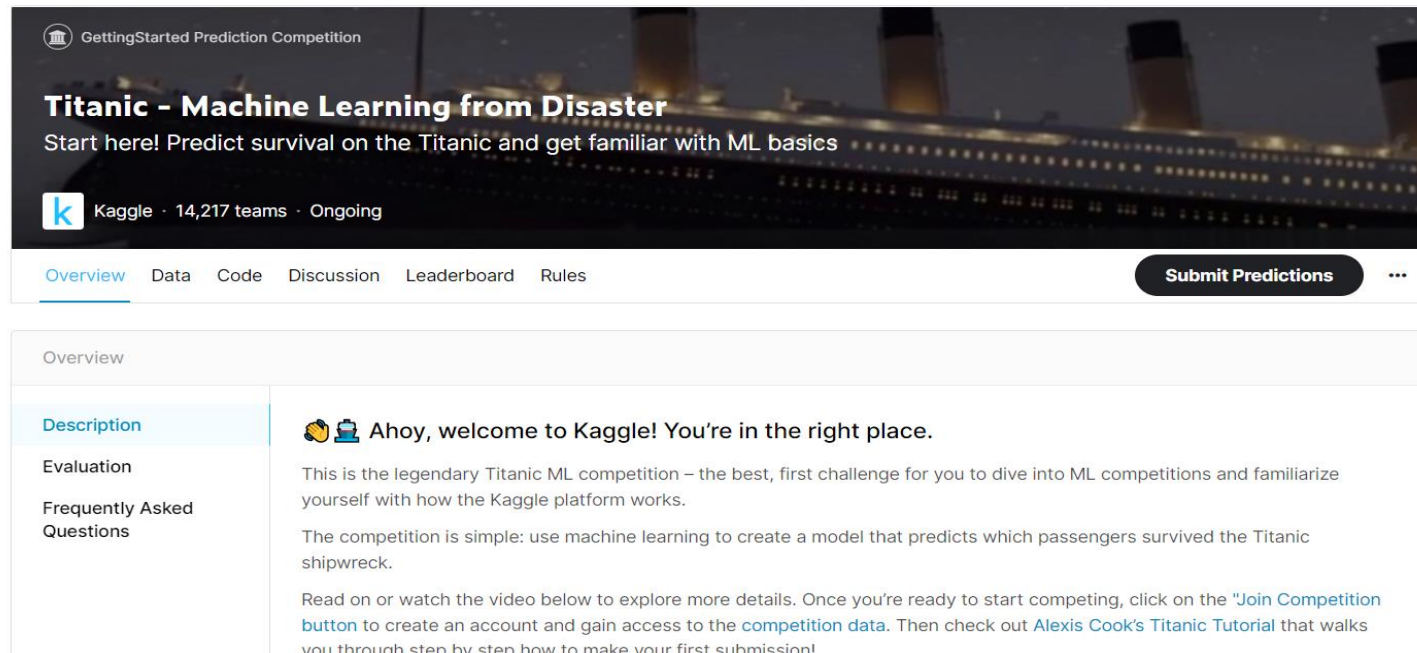
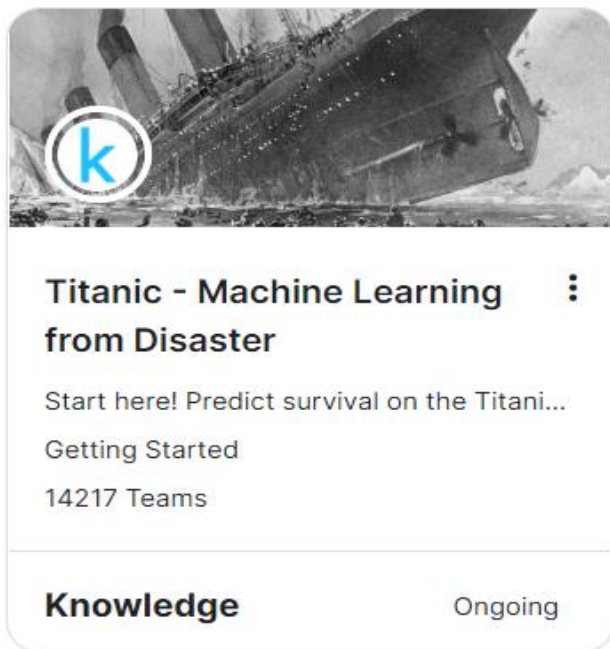


01

출처 및 목적

kaggle

Kaggle은 2010년 설립된 예측모델 및 분석 대회 플랫폼이다. 기업 및 단체에서 데이터와 해결과제를 등록하면, 데이터 과학자들이 이를 해결하는 모델을 개발하고 경쟁한다. Kaggle에서 주최하는 경진 대회 중 대표적인 것은 **Titanic: Machine Learning from Disaster**이다.



대회의 목표는 “어떤 사람들이 생존 할 가능성이 더 높은가”라는 질문에 답하는 예측 모델을 구축하는 것이다.



제공되는 Data Set으로는 학습 데이터(train.csv), 테스트 데이터(test.csv), 제출 양식(gender_submission.csv)가 있다.

01

분석 배경



1912년 시대적 배경에는
“Lady First”라는 개념이 존재 했다.

어린이와 여자부터
우선적으로 구조 했다고 한다.

“Lady First”

라는 개념이 없었다면

급박한 상황에서 구명정으로
달려들 때 **힘센 남성**이
구명정을 차지할 가능성이 높아
생존에 유리했을 것이다.



02

조회

1. 필요한 라이브러리 불러오기

```
# 필요한 라이브러리 import
import pandas as pd
import numpy as np
```

2. 정보 보기

```
train = pd.read_csv('./train.csv')
test = pd.read_csv('./test.csv')
train.head()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

타이타닉 dataset은 test데이터와 train데이터로 나뉘어 있습니다.

train데이터는 모델 훈련에 쓰이고, test데이터는 모델 검증에 쓰입니다.

test데이터에는 train데이터와 다르게 Survived 행이 존재하지 않습니다.
test데이터 셋의 Survived행을 예측하기 위해서 입니다.



#정보보기

train.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 891 entries, 0 to 890

Data columns (total 12 columns):

#	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

PassengerId: 각 승객의 고유 번호

Survived : 0 = 사망, 1 = 생존

Pclass : 1 = 1등석, 2 = 2등석, 3 = 3등석

Sex : male = 남성, female = 여성

Age : 나이

SibSp : 타이타닉 호에 동승한 자매/ 배우자의 수

Parch : 타이타닉 호에 동승한 부모/ 자식의 수

Ticket : 티켓 번호

Fare : 승객 요금

Cabin : 방 호수

Embarked : 탑승지 , C= 세르부르, Q = 퀸즈타운, S = 사우샘프턴



02

전처리 할 부분

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 891 entries, 0 to 890
```

```
Data columns (total 12 columns):
```

#	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
```

보기와 같이 생긴 dataset은 어떤 모델에 적용시켜도 수많은 에러와 낮은 정확도를 나타낼 것 이므로 앞으로 전처리를 하겠습니다.

1. Name, Sex, Embarked 자료형은 컴퓨터가 학습하기에 나쁘기 때문에 숫자로 바꾸도록 하겠습니다.

2. Age행에 Nan, 즉 결측 치 값을 채워야 합니다.

Nan값을 훈련할 때 사용한다면 잘못된 가중치로 받아 들일 수 있기 때문입니다.

3. Cabin과 같이 Nan값이 너무 많은 컬럼은 삭제합니다.



02

데이터 전처리

```
dic = {"male":0,"female":1}  
for dataset in whole:  
    dataset["Sex"] = dataset["Sex"].map(dic)  
train.head
```

	PassengerId	Survived	Pclass	Name	Sex	Age
0	1	0	3	Braund, Mr. Owen Harris	0	22.0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0
2	3	1	3	Heikkinen, Miss. Laina	1	26.0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0
4	5	0	3	Allen, Mr. William Henry	0	35.0

Dictionary를 이용해 남성(male)은 0으로 여성(female)은 1로 mapping합니다.



02

데이터 전처리

1. Null 개수 확인

```
#컬럼별 null 개수
train.isnull().sum()
```

```
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
```

2. Name에 호칭 추출

```
# 정규표현식을 사용해서 이름앞에 있는 호칭을 추출합니다.
for dataset in whole:
    dataset['Title']=dataset['Name'].str.extract('([A-Za-z]+)\.', expand=False)
train['Title'].value_counts().sort_index()
```

4. 호칭 Mapping

```
dic = {"Mr": 0, "Miss": 1, "Mrs": 2,
       "Master": 3, "Dr": 4, "Rev": 4, "Col": 4, "Major": 4, "Mlle": 4, "Countess": 4,
       "Ms": 4, "Lady": 4, "Jonkheer": 4, "Don": 4, "Dona": 4, "Mme": 4, "Capt": 4, "Sir": 4 }

for dataset in whole:
    dataset.drop("Name", axis=1, inplace=True)
    dataset["Title"] = dataset["Title"].map(dic)
    dataset.set_index("PassengerId", inplace=True)
```

3. 타이틀 종류와 개수 확인

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

	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Title
PassengerId											
1	0	3	0	22.0	1	0	A/5 21171	7.2500	NaN	S	0
2	1	1	1	38.0	1	0	PC 17599	71.2833	C85	C	2
3	1	3	1	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	1
4	1	1	1	35.0	1	0	113803	53.1000	C123	S	2

기존에 Name 컬럼을 삭제하고 호칭을 추출한 Title 컬럼을 호칭에 따라 (0,1,2,3,4)로 매핑합니다



02

데이터 전처리

```
for dataset in whole:
    dataset["Age"].fillna(unl.groupby("Title")["Age"].transform("mean"), inplace=True)
```

Age의 결측치 값 호칭의 평균나으로 처리 했습니다.

```
for dataset in whole:
    dataset["Agecut"] = pd.qcut(dataset["Age"], 4, labels=[0, 1, 2, 3])
    dataset.drop("Age", axis=1, inplace=True)
train.head()
```

Pandas에서 제공하는 qcut 함수를 통해

1. 0~22 세
2. 23~29 세
3. 30~35 세
4. 35세 초과

정해진 비율대로 Age값을 카테고리화 합니다.

Agecut이 생기고 모델 훈련에 사용하지 않을 Age를 삭제합니다.

	Survived	Pclass	Sex	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Title	Agecut
PassengerId											
1	0	3	0	1	0	A/5 21171	7.2500	NaN	S	0	1
2	1	1	1	1	0	PC 17599	71.2833	C85	C	2	3
3	1	3	1	0	0	STON/O2. 3101282	7.9250	NaN	S	1	1
4	1	1	1	1	0	113803	53.1000	C123	S	2	2
5	0	3	0	0	0	373450	8.0500	NaN	S	0	2



02

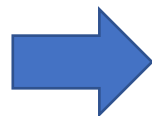
데이터 전처리

```
# Fare에 결측치 값을 가장 관련있는 Pclass와 관련지어 평균을 대입합니다.
for dataset in whole:
    dataset["Fare"].fillna(uni.groupby("Pclass")["Fare"].transform("mean"), inplace=True)
uni.info()
```

Test데이터의 Fare컬럼에 존재하는 Nan값을 처리하기 위해 Age에서 Nan값 처리하는 방법과 동일하게 평균을 대입합니다.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 1 to 1309
Data columns (total 10 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Pclass      1309 non-null    int64
1   Sex         1309 non-null    int64
2   SibSp       1309 non-null    int64
3   Parch       1309 non-null    int64
4   Ticket      1309 non-null    object
5   Fare        1309 non-null    float64
6   Cabin       295 non-null     object
7   Embarked    1307 non-null    object
8   Title       1309 non-null    int64
9   Agecut      1309 non-null    category
dtypes: category(1), float64(1), int64(5), object(3)
memory usage: 136.0+ KB
```

```
# Age를 카테고리화 할때와 같은 방식으로 Fare도 카테고리화를 진행합니다.
for dataset in whole:
    dataset["Farecut"] = pd.cut(dataset["Fare"], 4, labels=[0, 1, 2, 3])
    dataset.drop("Fare", axis=1, inplace=True)
train.head()
```



	Survived	Pclass	Sex	SibSp	Parch	Ticket	Cabin	Embarked	Title	Agecut	Farecut
PassengerId											
1	0	3	0	1	0	A/5 21171	NaN	S	0	1	0
2	1	1	1	1	0	PC 17599	C85	C	2	3	0
3	1	3	1	0	0	STON/O2. 3101282	NaN	S	1	1	0
4	1	1	1	1	0	113803	C123	S	2	2	0
5	0	3	0	0	0	373450	NaN	S	0	2	0

Age를 카테고리화 할 때 와 동일하게 Fare컬럼도 카테고리화 진행합니다.



02

데이터 전처리

```
# Embarked 행의 Nan drop  
train.dropna(axis=0, inplace=True)
```

Embarked의 존재하는 Nan 값을 가진 데이터를 제거합니다.

```
# S는 0, C는 1, Q는 2로 매핑시켜 자료형 변환해줍니다.  
dic2 = {"S": 0, "C": 1, "Q": 2}  
train.loc[:, "Embarked"] = train.loc[:, "Embarked"].map(dic2)  
test.loc[:, "Embarked"] = test.loc[:, "Embarked"].map(dic2)
```

S선승지에서 탄 경우 : 0
C선승지에서 탄 경우 : 1
Q선승지에서 탄 경우 : 2

	Pclass	Sex	Embarked
PassengerId			
1	3	0	0
2	1	1	1
3	3	1	0
4	1	1	0
5	3	0	0

02

데이터 전처리

```
# 동반자가 있는지 없는지 판단하기 위해  
# SibSp, Parch를 합해줍니다.
```

```
for dataset in whole:  
    dataset["FamilySize"] = dataset["SibSp"] + dataset["Parch"] + 1
```

동반자가 있는지 없는지 판단하기 위해

SibSp : 타이타닉 호에 동승한 자매/ 배우자의 수

Parch : 타이타닉 호에 동승한 부모/ 자식의 수

각 컬럼을 더해 FamilySize라는 컬럼을 생성합니다.

또

```
# isAlone이 0 이면 동반자가 있고, 1이면 동반자가 없습니다.
```

```
for dataset in whole:  
    dataset["IsAlone"] = 0 # 동반자가 있을  
    dataset.loc[dataset["FamilySize"] == 1, "IsAlone"] = 1 # 동반자가 없을
```

FamilySize를 이용해 isAlone = 0 이면 동반자가 있고, isAlone = 1 이면 동반자가 없다고 처리합니다.

```
label = train["Survived"]  
train.drop("Survived", axis=1, inplace=True)  
train.head()
```

지도학습이기때문에 라벨을 분리시킬 필요가 있습니다.

생존 유무를 파악하기 위함이므로 label에 Survived행을 저장합니다.

```
# 테이블 drop  
for dataset in whole:  
    dataset.drop(labels="SibSp", axis=1, inplace=True)  
    dataset.drop(labels="Parch", axis=1, inplace=True)  
    dataset.drop(labels="FamilySize", axis=1, inplace=True)  
    dataset.drop(labels="Title", axis=1, inplace=True)
```

```
# 테이블 drop  
for dataset in whole:  
    dataset.drop(labels="Cabin", axis=1, inplace=True)  
    dataset.drop(labels="Ticket", axis=1, inplace=True)
```

```
train.info()
```

필요 없는 테이블 삭제



02

데이터 전처리

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S



	Pclass	Sex	Embarked	Agecut	Farecut	IsAlone
PassengerId						
1	3	0	0	1	0	0
2	1	1	1	3	0	0
3	3	1	0	1	0	1
4	1	1	0	2	0	0
5	3	0	0	2	0	1

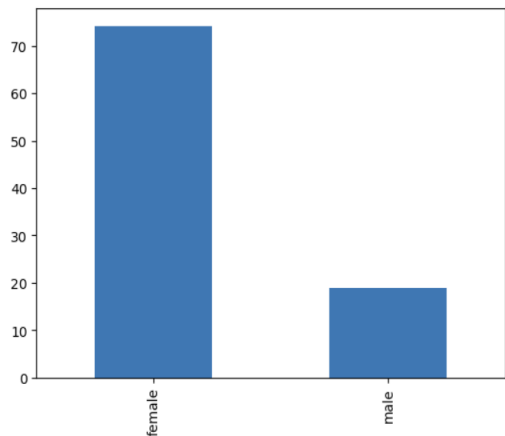


03

시각화

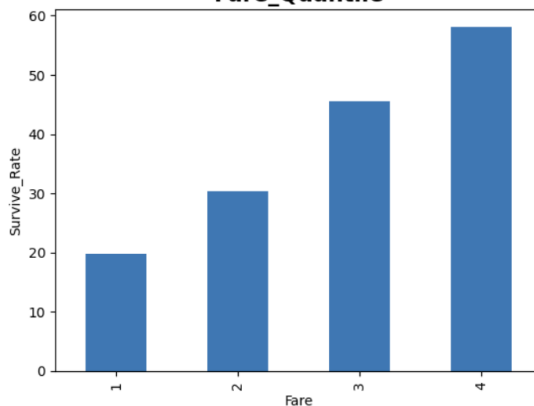
해당 feature들은 생존과 사망에 영향을 미치는 승객 정보들이다.

성별



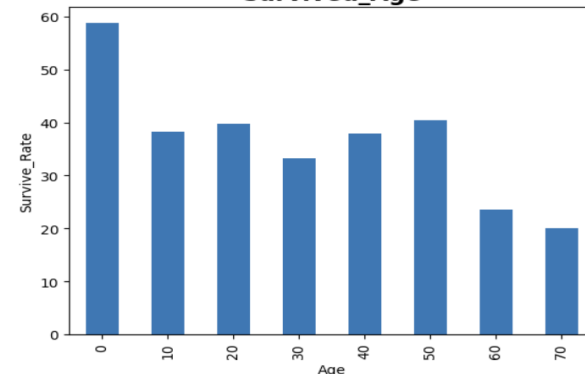
요금

Fare_Quantile



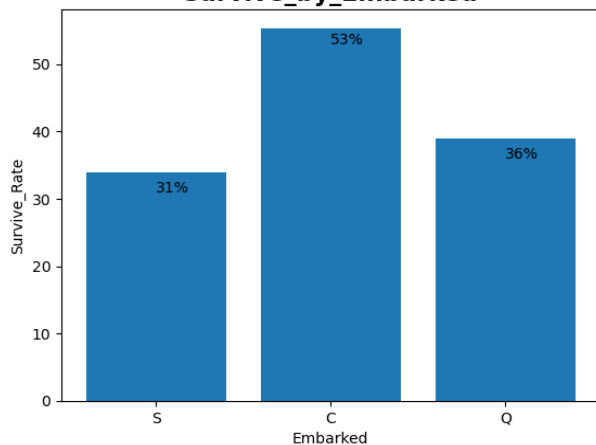
나이

Survived_Age



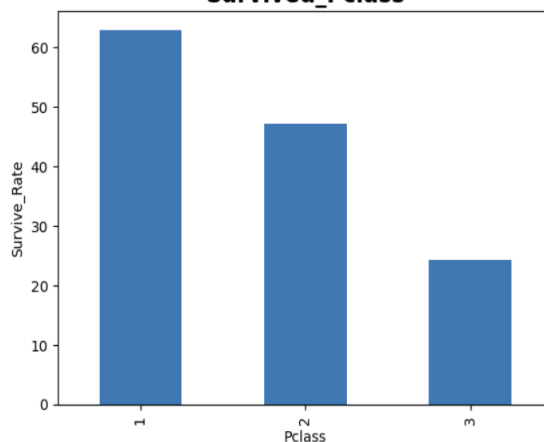
선승지

Survive_by_Embarked

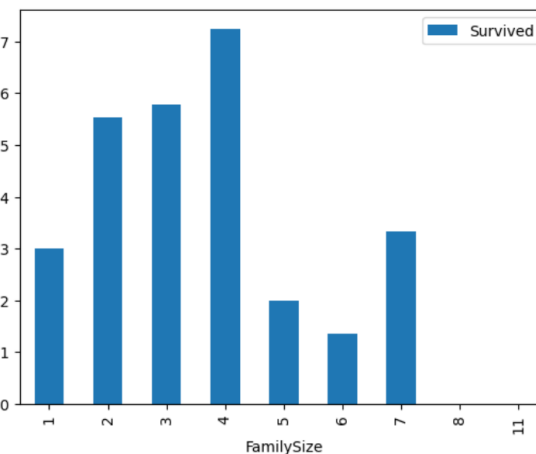


객실등급

Survived_Pclass



동승자



생존율이 높은 승객

여자

비싼 티켓

0~10세

c항

1등급 객실

동승자와 탑승

04 모델링

```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score
# k_fold = KFold(n_splits=20, shuffle=True, random_state=1)
```

SVM, Random Forest, Decision Tree
세가지 모델로 선정하여 학습

04 SVM

생존자 예측

```
param1={"C": [0.001, 0.01, 0.1, 1, 10, 100],  
        "gamma": [0.001, 0.01, 0.1, 1, 10, 100]}
```

```
clf = SVC()  
grid_search = GridSearchCV(clf, param_grid=param1, cv=20,  
                           refit=True,  
                           return_train_score=True)
```

```
grid_search.fit(train, label)
```

```
GridSearchCV(cv=20, estimator=SVC(),  
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100],  
                        'gamma': [0.001, 0.01, 0.1, 1, 10, 100]},  
             return_train_score=True)
```

파라미터들을 dictionary 형태로 설정한 후

Grid search를 이용하여 모델에 하이퍼 파라미터에 넣을 수 있는 값들을 순차적으로 입력한 뒤

가장 높은 성능을 보이는 파라미터를 찾아준다.

04 SVM

생존자 예측

```
# 최고 정확도를 나타냅니다  
grid_search.best_score_
```

```
0.8098484848484848
```

```
# 최고 파라미터를 나타냅니다.  
grid_search.best_params_
```

```
{'C': 1, 'gamma': 1}
```

```
# refit=True로 지정했기 때문에 최고의 파라미터로 미리 훈련했고  
# 그 최적의 모델을 반환합니다.  
best_model = grid_search.best_estimator_
```

```
# 최적의 모델의 score  
best_model.score(train, label)
```

```
0.8368953880764904
```

최고 정확도와 최고 정확도의 파라미터를 구한 후

Refit=True로 지정하여 최고의 파라미터로 훈련.

반환된 최적의 모델을 이용해 train 데이터의 score값을 구함

04 SVM

생존자 예측

```
clf = SVC(C=1, gamma=0.1, probability=True)
clf.fit(train, label)
prediction = clf.predict(test)
pred = pd.DataFrame({"PassengerId" : test.index,
                     "Survived" : prediction})
pred.head(10)
```

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1
5	897	0
6	898	1
7	899	0
8	900	1
9	901	0

최적의 C와 gamma를 설정하고 훈련데이터를 사용해
SVM모델 생성 후
test데이터로 예측 값을 뽑습니다.

04 Random forest

```
param2 = {'max_depth': range(5,20,1), # 15  
          'min_samples_split': range(2,100,10), # 10  
          'min_impurity_decrease': np.arange(0.0001, 0.001, 0.0001) # 9  
          }
```

```
rf=RandomForestClassifier()  
rs = RandomizedSearchCV(rf,  
                        param_distributions=param2,  
                        scoring="accuracy",  
                        n_jobs=-1,  
                        refit=True,  
                        cv=5,  
                        verbose=1,  
                        n_iter=500)
```

```
rs.fit(train, label)
```

파라미터들을 dictionary 형태로 설정 후

RandomizedSearchCV 이용하여
랜덤하게 추출된 샘플링을 사용해서
최고의 파라미터를 찾아준다.

Fitting 5 folds for each of 500 candidates, totalling 2500 fits

```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=500,  
                  n_jobs=-1,  
                  param_distributions={'max_depth': range(5, 20),  
                                      'min_impurity_decrease': array([0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008,  
0.0009]),  
                                      'min_samples_split': range(2, 100, 10)},  
                  scoring='accuracy', verbose=1)
```

04 Random forest

```
# 최고 정확도를 나타냅니다  
rs.best_score_ 0.8087856281343235
```

```
# 최고 파라미터를 나타냅니다  
rs.best_params_
```

```
{'min_samples_split': 12,  
 'min_impurity_decrease': 0.00030000000000000003,  
 'max_depth': 13}
```

최고 정확도와 최고 정확도의 파라미터를 구한 후

Refit=True로 지정하여 최고의 파라미터로 훈련.

반환된 최적의 모델을 이용해 train 데이터의 score값을 구함

```
best_model = rs.best_estimator_  
best_model
```

```
RandomForestClassifier(max_depth=13,  
                        min_impurity_decrease=0.00030000000000000003,  
                        min_samples_split=12)
```

```
# 최적의 모델의 score  
best_model.score(train, label) 0.8335208098987626
```

04 Random forest

생존자 예측

```
rf = RandomForestClassifier(max_depth=5,  
                           min_impurity_decrease=0.00030000000000000003,  
                           random_state=1)  
rf.fit(train, label)  
prediction = rf.predict(test)  
pred = pd.DataFrame({"PassengerId" : test.index,  
                    "Survived" : prediction})  
pred.head(10)
```

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0
5	897	0
6	898	1
7	899	0
8	900	1
9	901	0

최적의 파라미터를 설정하고 랜덤포레스트 모델 생성 후
test 데이터로 예측값을 뽑아냅니다

04

결정 트리(Decision Tree)

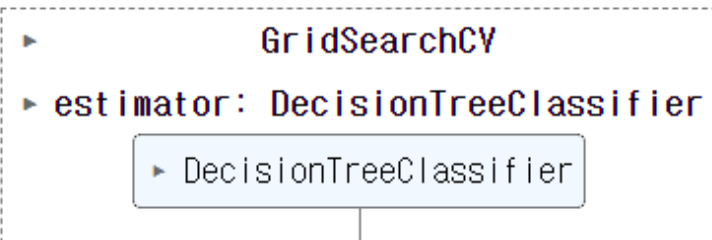
생존자 예측

```
param3 = {'max_depth': range(3,20,1),  
          'min_samples_split': range(2,50,1),  
          'min_impurity_decrease': np.arange(0.0001, 0.001, 0.0001)  
          }
```

```
dt = DecisionTreeClassifier()  
gs = GridSearchCV(dt, param_grid=param3, scoring="accuracy",  
                  n_jobs=-1, refit=True, cv=5, verbose=1)
```

```
gs.fit(train, label)
```

Fitting 5 folds for each of 7344 candidates, totalling 36720 fits



파라미터들을 dictionary 형태로 설정

Grid search를 이용하여 가장 높은 성능을 보이는 파라미터를 찾아줍니다.

Refit = True로 지정하여 최고의 파라미터로 다시 훈련하도록 해줍니다.

04

생존자 예측

결정 트리(Decision Tree)

```
gs.best_score_
```

```
0.8065574811147082
```

```
gs.best_params_
```

```
{'max_depth': 7,  
'min_impurity_decrease': 0.00030000000000000003,  
'min_samples_split': 12}
```

```
best_model = gs.best_estimator_  
best_model
```

```
DecisionTreeClassifier  
DecisionTreeClassifier(max_depth=7,  
                        min_impurity_decrease=0.00030000000000000003,  
                        min_samples_split=12)
```

```
best_model.score(train, label)
```

```
0.8323959505061868
```

Best_score_와 best_params_를 이용하여
최고 정확도와 최고 정확도의 파라미터를
구합니다

best_estimator_를 이용하여 최적의 모델을
반환합니다

반환된 최적의 모델을 이용해 train 데이터
의 score값을 구해줍니다.

04

결정 트리(Decision Tree)

생존자 예측

```
dt = DecisionTreeClassifier(max_depth=7,  
                             min_impurity_decrease=0.00030000000000000003,  
                             min_samples_split=12, random_state=1)  
  
dt.fit(train, label)  
prediction = clf.predict(test)  
pred = pd.DataFrame({"PassengerId" : test.index,  
                     "Survived" : prediction})  
pred.head(10)
```

```
pred.to_csv("./gender_submission", index=False)
```

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1
5	897	0
6	898	1
7	899	0
8	900	1
9	901	0

최적의 파라미터로 설정한 모델을 생성

Train 데이터로 훈련후, Test 데이터로 예측값을 얻습니다

Kaggle에 제출하기 위해 'gender_submission'파일에 예측값을 적용시켜줍니다

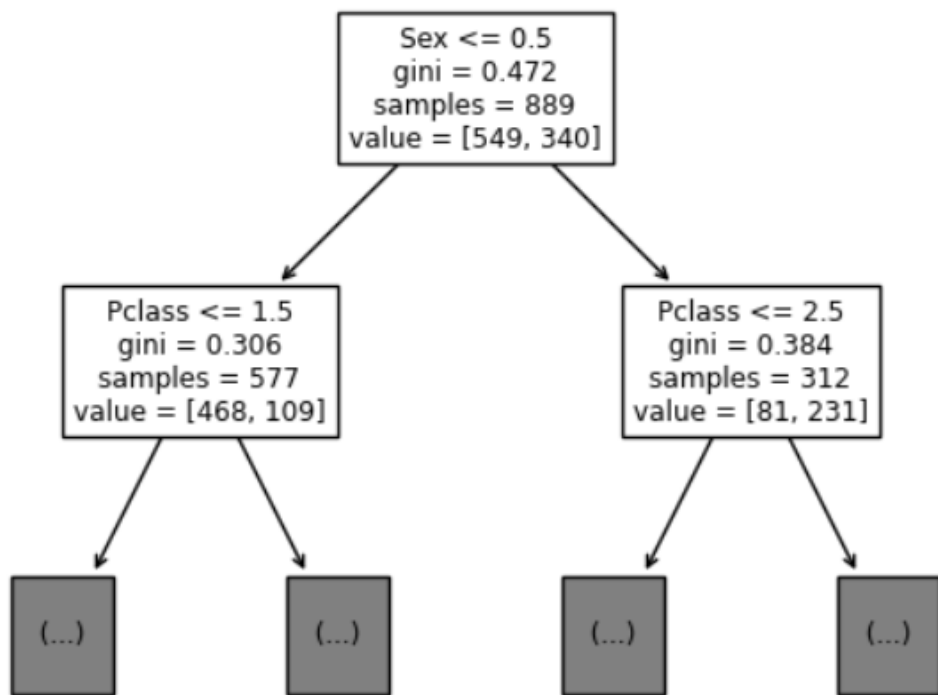
04

결정 트리(Decision Tree)

생존자 예측

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
%matplotlib inline
```

```
plot_tree(dt, feature_names=['Pclass', 'Sex', 'Embarked', 'Agecut', 'Farecut', 'IsAlone'], max_depth=1)
plt.show()
```



%matplotlib inline을 이용하여 브라우저에서 바로 그림을 볼 수 있게 해줍니다


plot_tree를 사용해 결정트리를 시각화 하였습니다

05

결론

Kaggle 제출 결과 SVM

YOUR RECENT SUBMISSION

 **gender_submission**
Submitted by ParkYoonSoo · Submitted just now

Score: 0.76555

↓ Jump to your leaderboard position

Kaggle 제출 결과 결정 트리

dt.csv

14 minutes ago by KIM KYUNGTAE

[add submission details](#)

0.76555

Kaggle 제출 결과 Random forest

YOUR RECENT SUBMISSION



RF.csv

Submitted by KIM KYUNGTAE · Submitted just now

Score: 0.77990

↓ Jump to your leaderboard position



06

