중간고사 대체 경진대회 보고서

Machine learning competition 2021

소프트웨어학과 32170578 김산

1. Import Library & Read Data

1) Import Library

In [ ]:
```
!pip install pycaret
!pip install xgboost
!pip install catboost
```

In [2]:
```
%matplotlib inline
from pycaret.classification import *

import time

from sklearn import pipeline
from hyperopt import fmin, tpe, hp, STATUS_OK, Trials

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # data visualization
import seaborn as sns # statistical data visualization

from sklearn import preprocessing

import xgboost as xgb
import lightgbm as lgb
import catboost as ctb

from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

from sklearn.metrics import accuracy_score, roc_curve, auc, confusion_matrix
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import RFE, RFECV
from mlxtend.feature_selection import SequentialFeatureSelector as SFS

import scipy.stats as stats
```

2) Read Data

In [4]:
```
# 인자로 전달된 경로의 csv파일을 읽어옵니다.
def readTrainCSV(path : str) :
    df = pd.read_csv(path, header=None)
    # target인자가 문자열이 아니면 키를 인식하지 못하는 경우가 몇몇 있어서 column을 문자열로 바꾸어 준뒤에 진행하였습니다.
    df.columns = [str(i) for i in range(0, df.shape[1])]
    X = df.drop('22', axis = 1)
    y = df['22'] - 1
    return df, X, y

df, X, y = readTrainCSV("./train_open.csv")
```

2. Model Selection

PyCaret 프레임 워크를 통해 각 모델별로 데이터 셋에 대한 정확도를 비교합니다. PyCaret은 파이썬 오픈소스 라이브러리로 머신러닝 모델들을 한번에 비교하는데 좋은 라이브러리를 제공합니다.

In [ ]:
```
# pycaret setup모듈은 다양한 기계학습 모델을 테스트하기위한 환경을 구성합니다.
# data인자에 훈련데이터를 전달하고 target인자에 class_label을 전달합니다.
clf = setup(data = df, target='22')
```

In [6]:
```
compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lightgbm	Light Gradient Boosting Machine	0.8821	0.9471	0.9268	0.8746	0.8998	0.7569	0.7593	0.148
catboost	CatBoost Classifier	0.8771	0.9465	0.9306	0.8648	0.8964	0.7459	0.7493	2.699
et	Extra Trees Classifier	0.8735	0.9444	0.9424	0.8520	0.8949	0.7372	0.7433	0.632
rf	Random Forest Classifier	0.8700	0.9414	0.9424	0.8474	0.8923	0.7295	0.7364	0.668
gbc	Gradient Boosting Classifier	0.8696	0.9440	0.9205	0.8609	0.8896	0.7308	0.7336	0.586
lr	Logistic Regression	0.8589	0.9326	0.8955	0.8630	0.8788	0.7101	0.7112	0.780
ridge	Ridge Classifier	0.8567	0.0000	0.8949	0.8603	0.8771	0.7056	0.7068	0.019
ada	Ada Boost Classifier	0.8567	0.9298	0.8886	0.8648	0.8763	0.7061	0.7070	0.214
lda	Linear Discriminant Analysis	0.8567	0.9344	0.8949	0.8603	0.8771	0.7056	0.7068	0.058
nb	Naive Bayes	0.8328	0.9118	0.8705	0.8422	0.8560	0.6568	0.6577	0.020
dt	Decision Tree Classifier	0.8196	0.8180	0.8292	0.8512	0.8398	0.6334	0.6340	0.037
knn	K Neighbors Classifier	0.6006	0.6167	0.6627	0.6470	0.6543	0.1813	0.1817	0.139
svm	SVM - Linear Kernel	0.5506	0.0000	0.6600	0.6461	0.5299	0.0639	0.0795	0.061
qda	Quadratic Discriminant Analysis	0.4634	0.5155	0.1484	0.6437	0.2375	0.0274	0.0494	0.038

Out[6]:
```
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
               importance_type='split', learning_rate=0.1, max_depth=-1,
               min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
               n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
               random_state=8334, reg_alpha=0.0, reg_lambda=0.0, silent='warn',
               subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

여기서 lightgbm은 정확도가 높은 모델들과 정확도차이가 크게 나지 않고, 수행시간이 짧아, 이후 작업에서 lightgbm모델을 사용하기로 하였습니다.

LightGBM은 XGBoost와 마찬가지로 결정트리 알고리즘 기반 부스팅 앙상블 기법입니다.

XGBoost는 `level-wise` 알고리즘으로 수직적으로 확장되는 반면, LightGBM은 수평적으로 확장되는 `leaf wise` 알고리즘 입니다. 때문에 다음과 같은 몇몇 장점들을 가집니다.

1. 빠른 훈련 속도, 높은 효율성
2. 낮은 메모리 사용률
3. 다른 부스팅 알고리즘에 비해 좋은 정확도
4. 대용량 데이터셋에 대한 호환성
5. 병렬처리

3. Data Analysis & Preprocessing

1. Data Overview

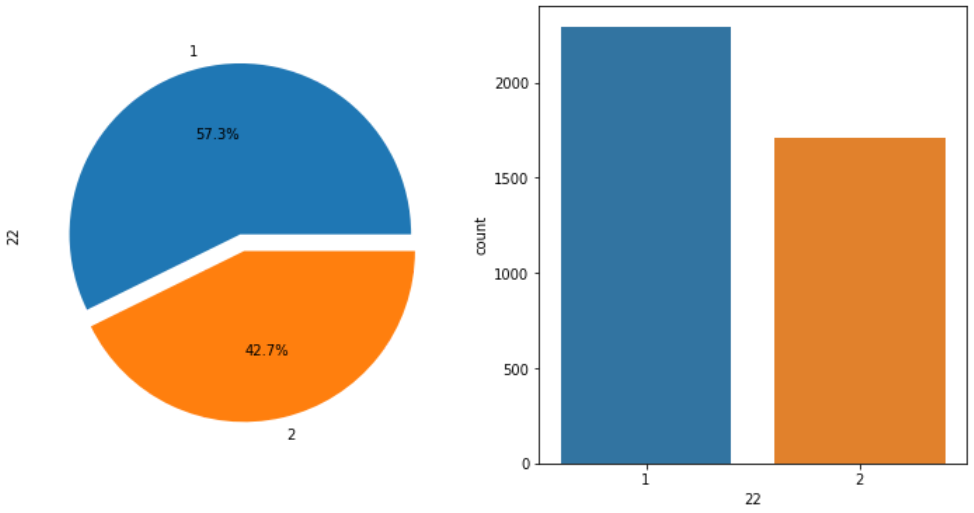
In [7]:
```
df.describe()
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	
count	4000.000000	4000.000000	4000.000000	4000.000000	4000.00000	4000.00000	4000.000000	4000.000000	4000.000000	4000.000000	400
mean	2.974250	1173.674250	3.337000	3.607250	3.33475	3.23925	2.730250	16.700750	3.344500	1.300000	
std	1.282965	992.518991	1.321313	1.188424	1.32538	1.33864	1.331889	41.140228	1.291602	0.458315	
min	1.000000	56.000000	0.000000	1.000000	1.00000	0.00000	0.000000	0.000000	1.000000	1.000000	
25%	2.000000	413.000000	2.000000	3.000000	2.00000	2.00000	2.000000	0.000000	2.000000	1.000000	
50%	3.000000	802.000000	4.000000	4.000000	4.00000	3.00000	3.000000	0.000000	4.000000	1.000000	
75%	4.000000	1721.000000	4.000000	5.000000	4.00000	4.00000	4.000000	15.000000	4.000000	2.000000	
max	5.000000	4983.000000	5.000000	5.000000	5.00000	5.00000	5.000000	794.000000	5.000000	2.000000	

In [8]:
```
f, ax = plt.subplots(1, 2, figsize=(12, 6))

df['22'].value_counts().plot.pie(explode=[0, 0.1], autopct='%1.1f%%', ax=ax[0])
sns.countplot('22', data=df, order=df['22'].value_counts().index, ax=ax[1])
A, B = y.value_counts()
print("Number of 0: ", A)
print("Number of 1: ", B)
```

```
Number of 0:  2291
Number of 1:  1709
```



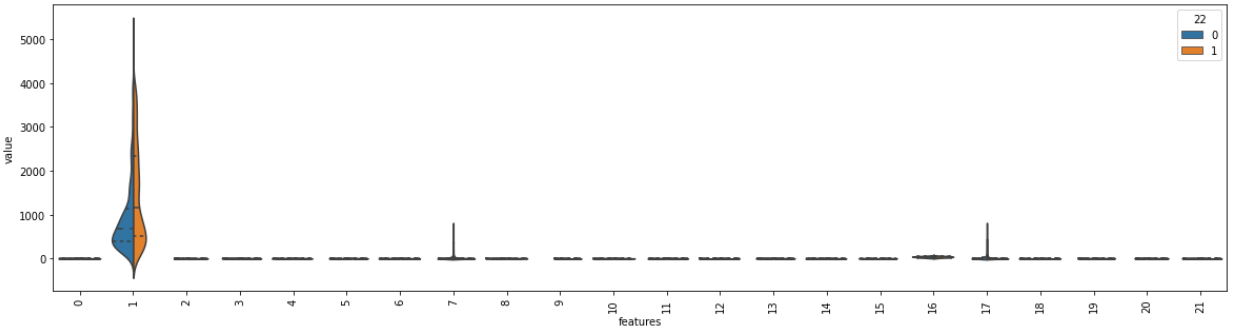
pandas의 read_csv모듈을 통해 csv파일을 pandas Dataframe으로 불러왔습니다. `describe()` 함수를 통해 데이터를 분석을 해보면 몇가지 특징을 확인할 수 있습니다.

1. 먼저 column이름이 따로 정해져 있지 않기 때문에 `read_csv` 함수에서 `header=None` 매개변수를 전달하여 첫번째 행이 column의 이름이 되지 않도록 해야합니다.
2. class_label인 22 컬럼의 경우, 값이 1과 2로 값이 구성이 되어있습니다. Binary Classification을 위한 몇몇 모듈과의 호환을 위해 1을 빼주어 0과 1로 값을 바꾸었습니다.
3. 모든 column이 int형이고, 1, 7, 16, 17 column의 경우 범위가 매우 넓지만, 나머지 column들의 범위는 대부분 1~5로 한정되어있는것을 확인할 수 있습니다. 때문에 normalization을 통해 데이터의 범위를 좁혀줘야 할 필요성이 있다 생각했습니다.

2. Without nomalization

In [9]:
```
data_dia = y
data = X
# don't nomalize features

data = pd.concat([y, data], axis=1)
data = pd.melt(data, id_vars="22",
                    var_name="features",
                    value_name="value")

plt.figure(figsize=(20, 5))
sns.violinplot(x="features", y="value", hue="22", data=data, split=True, inner="quart")
plt.xticks(rotation=90)
```

Out[9]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21]), <a list of 22 Text major ticklabel objects>)
```



1번 feature로 인해 다른 feature를 분석하기 어렵습니다. normalization을 통해 이를 해결합니다.

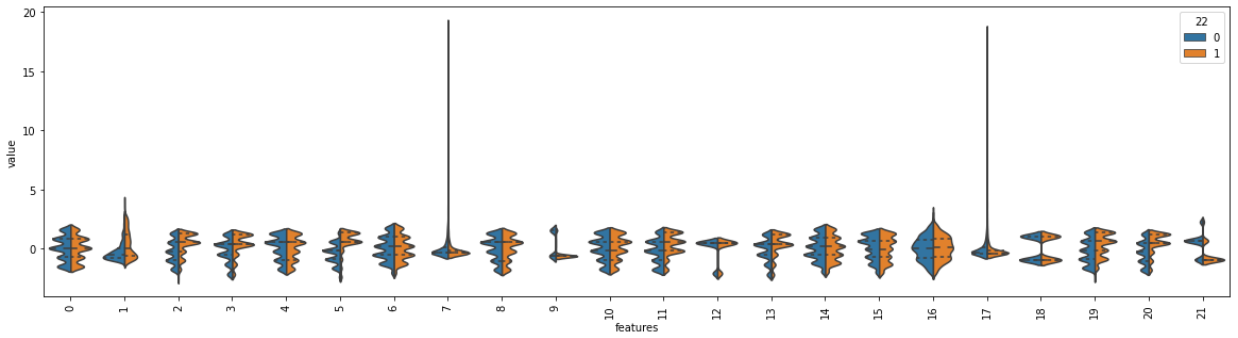
3. Nomalization

In [10]:
```
# first ten features
data_dia = y
data = X
data_n_2 = (data - data.mean()) / data.std()  # pd.std(): return sample standardization over requested axis

data = pd.concat([data_dia, data_n_2], axis=1)
print(data.shape)
# data.head()
data = pd.melt(data, id_vars="22",
                    var_name="features",
                    value_name="value")
print(data.head())
print(data.shape)
```

```
(4000, 23)
   22 features     value
0   0        0  0.020071
1   0        0  0.020071
2   1        0  0.020071
3   0        0  0.799515
4   0        0  0.020071
(88000, 3)
```

In [11]:
```
plt.figure(figsize=(20, 5))
sns.violinplot(x="features", y="value", hue="22", data=data, split=True, inner="quart")
plt.xticks(rotation=90)
```

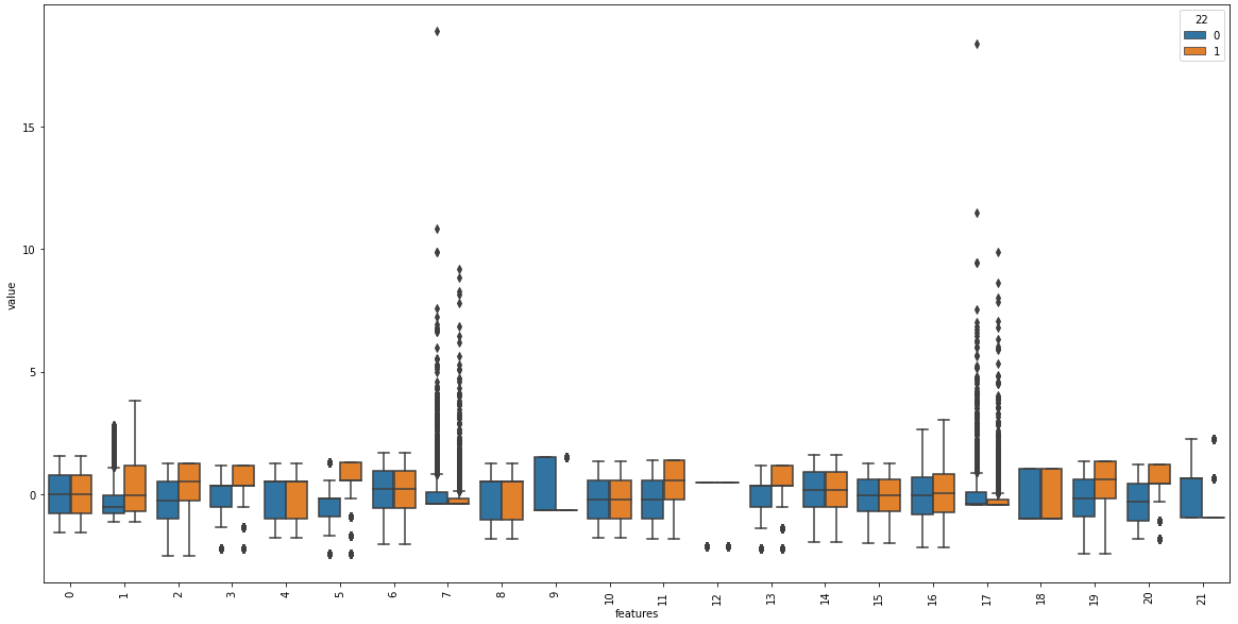
Out[11]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21]), <a list of 22 Text major ticklabel objects>)
```



앞선 그래프를 해석해 보았을때 5,9,21 feature의 0과 1의 중위값이 분류되는 것 처럼 보입니다. 반면 이외 feature들은 0,1을 분류하지 못하는것으로 보입니다

In [12]:
```
plt.figure(figsize=(20, 10))
sns.boxplot(x="features", y="value", hue="22", data=data)
plt.xticks(rotation=90)
```

Out[12]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21]), <a list of 22 Text major ticklabel objects>)
```

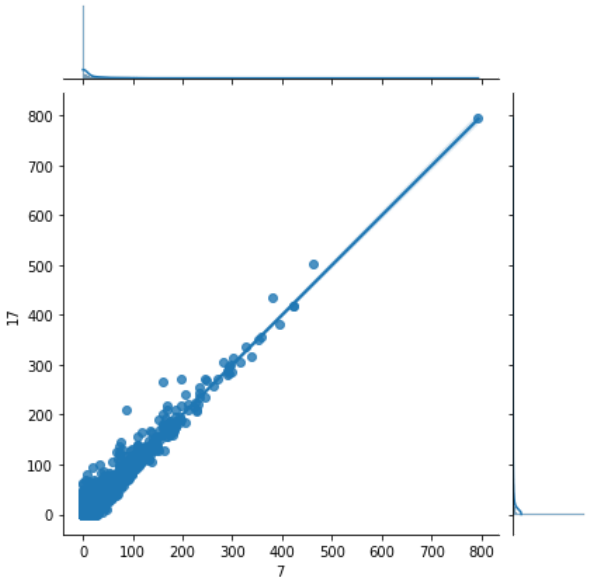


3,5,9,13,20,21 feature의 중위값이 0,1을 분류하는것으로 보이며, 7-17, 9-21 feature의 형태가 유사해 보입니다.

4. Correlation Test

In [13]:
```
sns.jointplot(X.loc[:, '7'], X.loc[:, '17'], kind="reg")
stats.pearsonr(X.loc[:, '7'], X.loc[:, '17'])
```

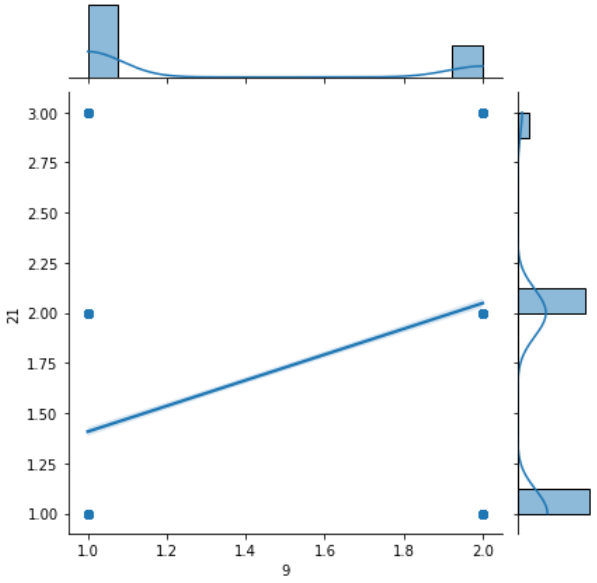
Out[13]: (0.9708695348775973, 0.0)



- jointplot을 통해 7, 17 두 feature의 상관관계를 비교해 보았을때 점들이 선에 가깝게 위치합니다. 이는 변수 사이에 강한 선형 관계가 있다는 것을 나타냅니다. 한 변수가 증가하면 다른 변수도 증가하기 때문에 양의 관계가 있습니다.
- 또한 `stats.pearsonr()` 를 통해 correlation(리턴값중 첫번째)이 1에 가까운것을 확인할 수 있습니다. 이는 높은 양의 상관관계를 의미합니다.
- 따라서 7과 17은 높은 양의 상관관계를 보여주고 있음을 알 수 있습니다.

In [14]:
```
sns.jointplot(X.loc[:, '9'], X.loc[:, '21'], kind="reg")
stats.pearsonr(X.loc[:, '9'], X.loc[:, '21'])
```

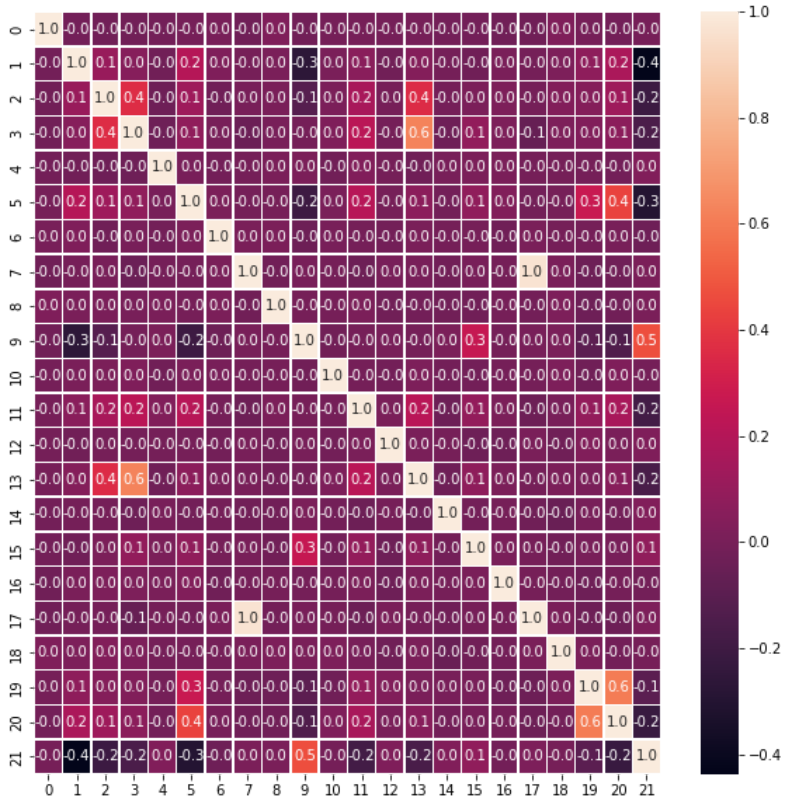
Out[14]: (0.4708169246116344, 7.340464574576601e-220)



반면 9와 21은 큰 상관관계가 없는 것으로 보입니다.

In [15]:
```
# correlation map
f, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(X.corr(), annot=True, linewidth=.5, fmt='.1f', ax=ax)
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7fed887df3d0>



4. Feature Selection

1.Remove correlated feature

앞선 데이터 분석에서 다음과 같은 데이터의 몇가지 특징을 확인할 수 있었습니다.

- 7과 17은 높은 양의 상관 관계를 가진다(p > 0.8)
- 3,5,9,13,20,21 feature의 중위값이 0,1을 분류할 수 있는것으로 보인다

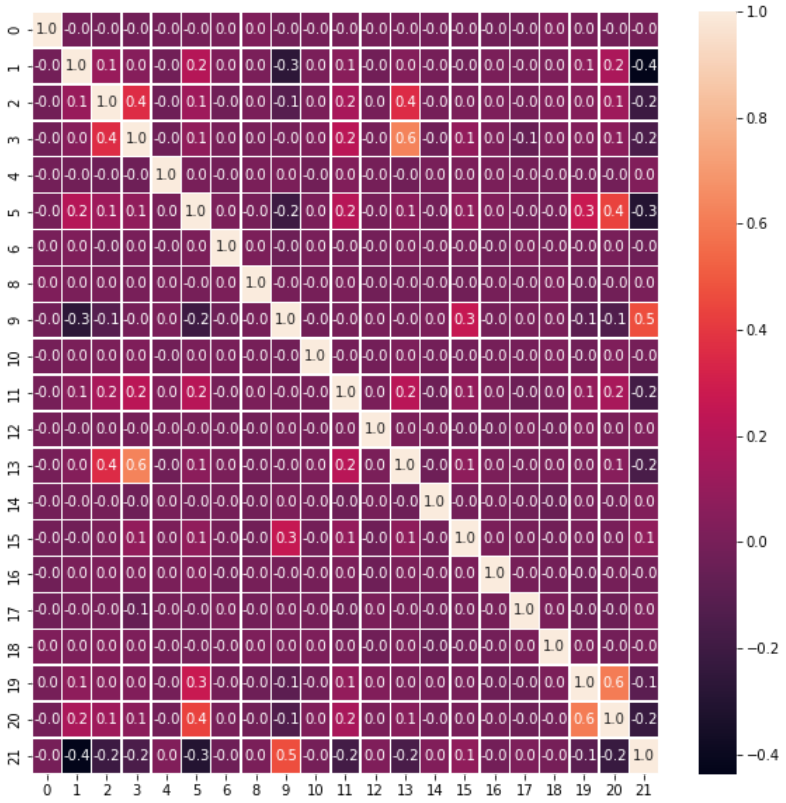
7과 17 feature중 어느 feature가 0, 1을 뚜렷히 구별한다고 보기 어려웠기에 둘 중 어느 feature를 삭제하더라도 상관 없을거라 생각했고, 구글링을 통해 이러한 변수들은 다중공신성 문제를 갖는다라는 사실을 알아냈습니다. 7과 17의 경우 0.8이상의 높은 선형관계가 존재하므로 다중공신성이 있다 볼 수 있습니다. 이는 회귀분석의 전제가정을 위배하므로 변수를 제거하거나 통합하여 관리하는게 맞다고 합니다.

따라서 상관 관계가 높은 feature를 제거해준뒤 heatmap을 그려보았습니다.

In [16]:
```
x_1 = X.drop('7',axis = 1)
```

In [18]:
```
#correlation map
f,ax = plt.subplots(figsize=(10, 10))
sns.heatmap(x_1.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

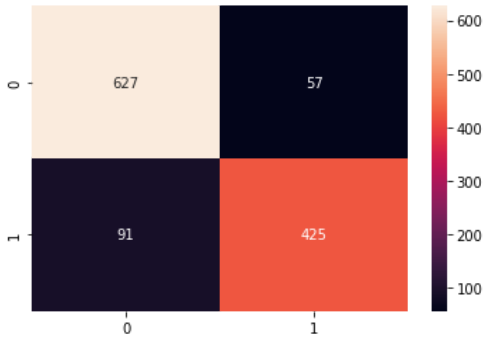
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7fed888c3d50>



In [19]:
```
#Split in 75% train and 25% test set
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.3, random_state= 1234)

lgb_clf = LGBMClassifier(random_state=1234)
lgb_clf = lgb_clf.fit(train_x, train_y)

ac = cross_val_score(lgb_clf, X, y).mean()
# ac = accuracy_score(test_y, lgb_clf.predict(test_x))
print("Accuracy is: ", ac)

cm = confusion_matrix(test_y, lgb_clf.predict(test_x))
sns.heatmap(cm, annot=True, fmt="d")
```

```
Accuracy is:  0.8815000000000002
```

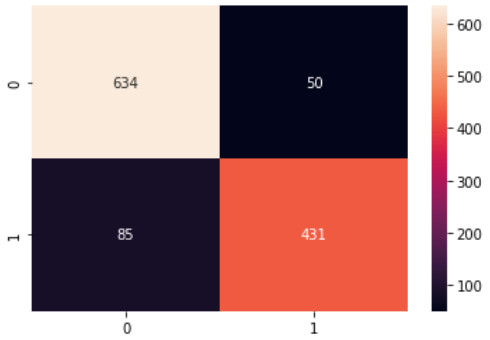
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7fed88ebf3d0>



In [20]:
```
#Split in 75% train and 25% test set
train_x, test_x, train_y, test_y = train_test_split(x_1, y, test_size = 0.3, random_state = 1234)

lgb_clf = LGBMClassifier(random_state=1)
lgb_clf = lgb_clf.fit(train_x, train_y)

ac = cross_val_score(lgb_clf, x_1, y).mean()
# ac = accuracy_score(test_y, lgb_clf.predict(test_x))
print("Accuracy is: ", ac)

cm = confusion_matrix(test_y, lgb_clf.predict(test_x))
sns.heatmap(cm, annot=True, fmt="d")
```

```
Accuracy is:  0.88575
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fed88ddcd50>
```



- 상관관계가 높은 feature를 제거해 줌으로써 정확도가 조금 상승한 것을 확인할 수 있었습니다.
- 하지만 confusion metrix를 보면 여전히 잘못된 예측이 많이 존재하는것을 볼 수 있습니다

2. Feature Selection

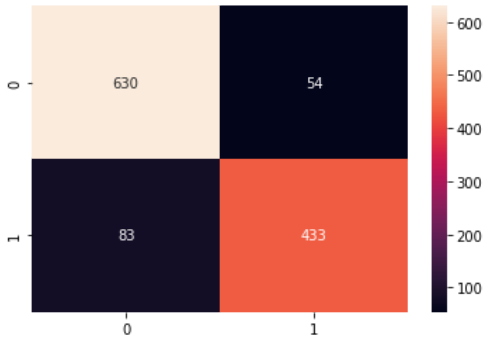
1) SelectKBest

- `SelectKBest` 는 일변량 통계분석모델을 통해 K개의 가장 좋은 feature를 선택합니다.
- `chi2` 또는 `mutual_info_classif` 함수 와 함께 사용되어 class label과 가장 관련성이 높은 상위 k 개의 특징 을 식별하고 선택합니다.

In [41]:
```
# find best scored 13 features
select_feature = SelectKBest(chi2, k=13).fit(train_x, train_y)

all_x_2 = select_feature.transform(x_1)
train_x_2 = select_feature.transform(train_x)
test_x_2 = select_feature.transform(test_x)

# random forest classifier with n_estimators=10 (default)
lgb_clf_2 = LGBMClassifier()
ac_2 = cross_val_score(lgb_clf_2, all_x_2, y).mean()
print("Accuracy is: ", ac_2)

lgb_clf_2 = lgb_clf_2.fit(train_x_2, train_y)
# ac_2 = accuracy_score(test_y, lgb_clf_2.predict(test_x_2))
cm_2 = confusion_matrix(test_y, lgb_clf_2.predict(test_x_2))
sns.heatmap(cm_2, annot=True, fmt="d")
```

```
Accuracy is:  0.8845000000000001
```

Out[41]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fed83fe9490>
```



2) Recursive feature elimination(RFE)

Recursive Feature Elimination(RFE)는 Backward fature selection방식중 하나로 모든 변수를 우선 다 포함 시킨후 반복하여 학습을 진행하면서 중요도가 낮은 변수를 제거하는 방식입니다. 다음의 예에서는 15개의 feature를 선택하여 진행하였습니다.

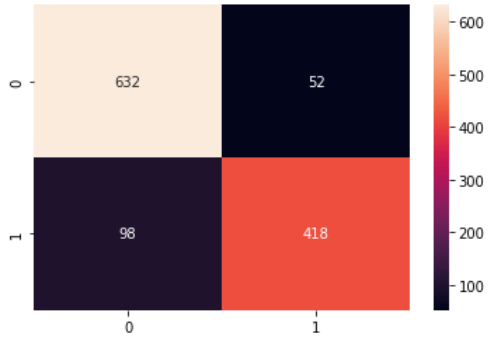
In [22]:
```
# Create the RFE object and rank each pixel
lgb_clf_3 = LGBMClassifier()
rfe = RFE(estimator=lgb_clf_3, n_features_to_select=15)
rfe = rfe.fit(train_x, train_y)

print("Chosen best 15 feature by rfe: ", train_x.columns[rfe.support_])
```

```
Chosen best 15 feature by rfe:  Index(['1', '2', '3', '4', '5', '6', '9', '11', '13', '14', '15', '16', '17',
       '19', '20'],
      dtype='object')
```

In [39]:
```
all_x_3 = x_1.loc[:, x_1.columns[rfe.support_]]
train_x_3 = train_x.loc[:, train_x.columns[rfe.support_]]
test_x_3 = test_x.loc[:, test_x.columns[rfe.support_]]

lgb_clf_3 = LGBMClassifier()
lgb_clf_3.fit(train_x_3, train_y)

ac_3 = cross_val_score(lgb_clf_3, all_x_3, y).mean()
# ac_3 = accuracy_score(test_y, lgb_clf_3.predict(test_x_3))
print("Accuracy: ", ac_3)
# confusion matrics
cm_3 = confusion_matrix(test_y, lgb_clf_3.predict(test_x_3))
sns.heatmap(cm_3, annot=True, fmt="d")
```

```
Accuracy:  0.885
```

Out[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fed83bdc850>
```



In [42]:
```
rfe.support_
```

Out[42]:
```
array([False,  True,  True,  True,  True,  True,  True, False,  True,
       False,  True, False,  True,  True,  True,  True,  True, False,
        True,  True, False])
```

이외에도 feature개수를 바꾸어 가며 다양한 feature개수를 테스트 해보았음에도 불구하고 전반적으로 RFE방식을 통한 feature selection방식은 정확도가 별로 좋지 않았습니다. 21개의 feature를 모두 사용하였을때 가장 정확도가 높게 나왔습니다.

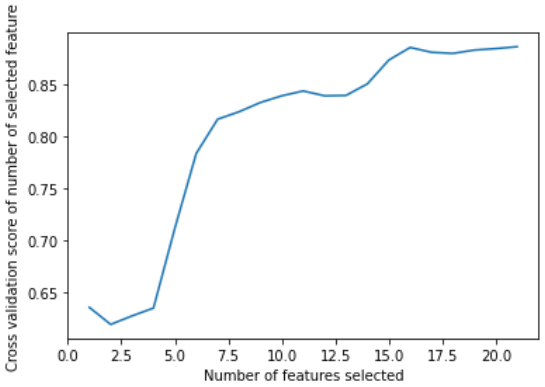
3) RFECV

RFECV를 통해 RFE방식에서의 최적의 feature개수가 몇개인지도 확인해 보았습니다.

In [24]:
```
# The "accuracy" scoring is proportional to the number of correct classifications
lgb_clf_4 = LGBMClassifier()
kfold = StratifiedKFold(n_splits=5)
rfecv = RFECV(estimator=lgb_clf_4, step=1, cv=kfold,scoring='accuracy')   #5-fold cross-validation
rfecv = rfecv.fit(train_x, train_y)

print('Optimal number of features :', rfecv.n_features_)
print('Best features :', train_x.columns[rfecv.support_])
```

```
Optimal number of features : 21
Best features : Index(['0', '1', '2', '3', '4', '5', '6', '8', '9', '10', '11', '12', '13',
       '14', '15', '16', '17', '18', '19', '20', '21'],
      dtype='object')
```

In [25]:
```
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score of number of selected feature")
plt.plot(range(1, len(rfecv.grid_scores_) +1), rfecv.grid_scores_)
plt.show()

print(rfecv.grid_scores_[-1])
```



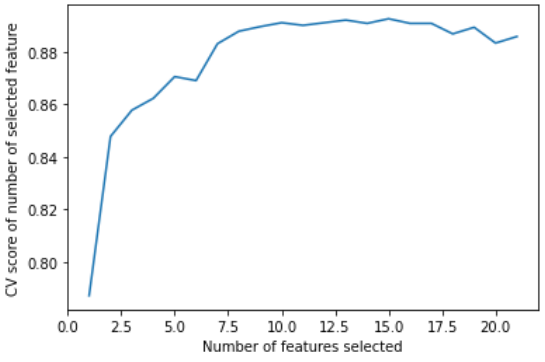

```
0.8857142857142858
```

모든 feature를 사용하였을때 가장 높은 정확도를 보이는것을 확인할수 있었습니다.

4) SequentialFeatureSelector(SFS)

In [26]:
```
lgb_clf_5 = LGBMClassifier()
kfold = StratifiedKFold(n_splits=5)
sfs = SFS(lgb_clf_5,
          k_features=21, # number of features
          verbose=0,
          scoring='accuracy',
          cv=5)

sfs = sfs.fit(x_1, y, custom_feature_names=x_1.columns)
```

In [27]:
```
scores = []

for i in range(1, 22):
    scores.append(sfs.subsets_[i]['cv_scores'].mean())

plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("CV score of number of selected feature")
plt.plot(range(1, 22), scores)
plt.show()

for i in range(11, 21):
    print(str(i) + ':' + str(sfs.subsets_[i]['cv_scores'].mean()))
```




```
11:0.89
12:0.891
13:0.892
14:0.8907499999999999
15:0.8925000000000001
16:0.89075
17:0.8907499999999999
18:0.8867499999999999
19:0.88925
20:0.88325
```

5) result

SFS로 고른 15개의 feature가 가장 정확도가 높았지만, 13개 feature를 골랐을때와 정확도가 큰 차이(0.0005)를 보이지 않았기 때문에 SFS방식으로 고른 13개의 feature로 진행하였고, feature selelction을 진행하지 않았을때(0.88575) 보다 정확도가 조금 향상(0.892)된 것을 확인할 수 있습니다.

SFS로 고른 13개의 feature로 파라미터 튜닝을 진행하였을때 `test.csv` 파일에 대한 예측도가 SelectKBest(0.926)에 비해 낮게(0.920) 나왔습니다.

때문에 SelectKBest방식으로 다시 하이퍼 파라미터 튜닝을 진행하였습니다.

5. Hyperparameter tunning

앞서 고른 feature로 hyperparameter tunning을 진행하였습니다. hyperparameter tunning에는 다음과 같이 몇가지 방법들이 존재합니다.

- GridSearchCV
  - Grid search는 모델의 hyperparameter에 넣을 수 있는 값들을 순차적으로 입력한 뒤에 가장 높은 성능을 보이는 hyperparameter를 찾는 탐색 방법입니다. 일일이 탐색해 보기때문에 너무 오랜시간이 걸려 GridSearchCV방식은 진행하지 않았습니다.
- RandomizedSearchCV
  - 앞선 grid search방식의 경우 비교할 parameter조합이 적은 경우에 좋습니다. 하지만 parameter의 조합이 많은경우 매우 오랜시간이 걸립니다. random search는 이름과 같이 가능한 모든 조합을 탐색하되, 각 반복마다 임의의 수를 대입하여 지정한 횟수만큼만 평가하여 최적의 조합을 찾아 냅니다.
- Hyperopt
  - Hyperopt는 베이지안 최적화 기법을 활용한 hyperparameter 튜닝 라이브러리로 현재 파라미터 최적화를 위해 가장 많이 사용되는 라이브러리중 하나입니다.

Hyperopt라이브러리를 통한 LGBMClassifier Hyperparameter튜닝을 위한 매개변수 범위를 지정합니다

learning rate와 n_estimator가 서로 영향을 주고, max_depth, min_child_samples, num_leaves가 서로 영향을 주기 때문에 각각 주석처리하며 분리하여 파라미터 튜닝을 진행하였습니다.

In [57]:
```
learn_r = np.arange(0.05, 0.15, 0.001)
n_est = np.arange(50, 150, dtype=int)

m_depth = np.arange(3, 20, dtype=int)
m_child_samples = np.arange(2, 100, dtype=int)
n_leaves = np.arange(2, 100, dtype=int)

space = {
        # 'learning_rate' : hp.choice('learning_rate', learn_r),
        # 'n_estimators': hp.choice('n_estimators', n_est),

        'max_depth': hp.choice('max_depth', m_depth),
        'min_child_samples' : hp.choice('min_child_samples', m_child_samples),
        'num_leaves' : hp.choice('num_leaves',n_leaves),

        }
```

In [58]:
```
def hyper_tunning(params):
        param = {
                'objective': 'binary',
                'boosting_type': 'gbdt',
                'metric' : 'binary_logloss',

                'learning_rate': 0.064,
                'n_estimators': 138,

                'max_depth': 1,
                'min_child_samples' : 10,
                'num_leaves' : 91,

                # 'learning_rate': params['learning_rate'],
                # 'n_estimators': params['n_estimators'],

                # 'max_depth': int(params['max_depth']),
                # 'min_child_samples' : int(params['min_child_samples']),
                # 'num_leaves' : int(params['num_leaves']),

        }

        lgb_clf = LGBMClassifier(**param)
        kfold = StratifiedKFold(n_splits=5)
        max_score = cross_val_score(lgb_clf, test_x_2, test_y, scoring='accuracy', cv=kfold).mean()

        return 1 - max_score

trials = Trials()
best = fmin(fn=hyper_tunning,
            space=space,
            max_evals=50,
            algo=tpe.suggest,
            trials=trials
            )

print(best)
```

```
100%|██████████| 50/50 [00:17<00:00,  2.82it/s, best loss: 0.11583333333333334]
{'max_depth': 1, 'min_child_samples': 10, 'num_leaves': 91}
```

In [60]:
```
# learn_r[14]
n_est[87]
# best
```

Out[60]: 137

hyper_tunning() 함수를 실행하고 best loss 값을 통해 튜닝 결과를 확인했을때 다음과 같은 파라미터 조합이 가장 성능이 좋았고, 튜닝전보다 정확도가 향상된 것을 확인할 수 있었습니다.

In [61]:
```
# 0.11099999999999999
# 0.10975000000000001
# 0.10899999999999999
best_param = {
    'objective': 'binary',
    'boosting_type': 'gbdt', #'dart'
    'metric' : 'binary_logloss',

    'learning_rate': 0.064,
    'n_estimators': 137,

    'max_depth': 1,
    'min_child_samples' : 10,
    'num_leaves' : 91,
}
```

In [62]:
```
final_clf = LGBMClassifier()
acc = cross_val_score(final_clf, all_x_2, y)
print(acc.mean())
```

```
0.8845000000000001
```

In [63]:
```
final_clf.fit(all_x_2, y)

test_df = pd.read_csv("./test_open.csv", header = None)
test_df.columns = [str(i) for i in range(0, test_df.shape[1])]
test_df = test_df.drop('7',axis=1)
test_df = select_feature.transform(test_df)
# test_df = test_df[sfs_x]
pred = final_clf.predict(test_df)
pred = pred + 1
pred.tofile('./32170578_김산.csv', sep='\n')
```

후기

이외에도 각단계에서 방법들을 많이 바꾸어 가며 테스트를 해보았습니다.

여러 모델들을 훈련시킨후 해당 모델들의 예측값들을 feature로 만들어 로지스틱 회귀를 하는 Stacking기법을 사용해보기도 했지만, 가장 최적의 결과를 만들어 내는것은 데이터 전처리에서 였던것 같습니다.

모델을 바꾸어가고 Stacking도 해보고, hyperparameter튜닝을 위한 optuna나 hyperopt와 같은 라이브러리도 사용하고, feature selection에서 feature개수도 바꾸어보고 여러 feature selection방식을 사용해 보았지만, 결국 91%의 정확도에서 크게 개선되는 부분이 없었습니다.

이후에 Kaggle에서의 유사한 케이스에서의 해결 방법이 있는지 구글링을 통해 알아보았는데 결국 데이터 전처리가 가장 중요하다는 말을 듣게 되었고, feature간 상관계수를 찾는 작업을 하게 되었습니다.

통계학에서 회귀분석의 전제조건중 종속변수들의 값은 서로 독립적이어야한다는 조건이 있는데 feature간 상관계수가 높을경우 이러한 전제가정을 위배하게 되고, 이는 정확도를 떨어뜨립니다. 이를 다중공신성 문제라고 합니다.

전처리 이후의 작업들도 물론 중요하지만, 전처리가 가장 중요하다는 생각을 갖게 된것 같습니다.

이러한 전처리 이후에 feature selection도 해보고, hyperparameter튜닝도 해보았지만, `train_open` 데이터 셋에서의 정확도는 오르는듯 했으나, `test_open` 데이터 셋에서의 정확도는 오히려 낮아졌습니다. 아마 overfitting의 문제인것 같은데 이러한 부분은 아직 경험이 많이 부족하고, 수십시간 쏟아 여러 파라미터들을 테스트 해보았지만, 해결하지 못했습니다.

이번 중간대체과제를 시작으로 앞으로 조금더 머신러닝에 대한 경험을 쌓아가면서 이러한 부분들을 개선해 나가고자 합니다.