선진국 분류하기

201300938 최영조

1. 분석 배경과 목적

    ⑴ 분석 배경

    ⑵ 분석 목적

    ⑶ 분석 방법

2. 자료 수집

    ⑴ 스크래핑

    ⑵ 데이터병합

3. 예측 모델

    ⑴ 빈값 처리

    ⑵ 머신러닝과 parameter

    ⑶ threshold

4. 결론과 논의

□ 자료 출처

1. 분석 배경과 목적

(1) 분석 배경

○ 브루나이처럼 1인당 GDP가 매우 높음에도 정치와 석유를 제외한 인프라는 후진국 수준인 나라도 있고 중국처럼 매우 강력하지만 선진국이라 얘기하기 어려운 나라들도 있다. 이처럼 선진국으로 분류되기 위해서는 많은 요소가 고려된다.

○ 머신러닝을 활용해 여러 지표를 종합적으로 판단해 선진국인지 아닌지를 분류할 수 있다.

○ 전 세계 200개 가까운 나라 중 선진국으로 분류되는 비율은 1/4도 되지 않기 때문에 정분류율/전체의 비율을 따지는 정확도 외에 예측모델의 다른 성과측정 지표가 필요하다.

(2) 분석 목적

○ recall과 precision을 최적화해 선진국을 분류해내는 비용적인 측면과 위험성 측면을 가장 줄일 수 있다.

○ 선진국이 아닌 국가의 경우 각 나라에 맞는 발전 과제를 생각해 볼 수 있다.

(3) 분석 방법

○ 신뢰할 수 있는 조사기관의 자료를 웹상에서 스크래핑한다. ('자료수집'에서 더 상술한다.)

○ 수집하는 자료들은 다음과 같다.

- 1인당 GDP, 문해율, 교육지수, 민주주의지수, 언론자유지수, 평균수명, 부패지수

○ 웹페이지의 모든 항목 간 국가명 표기 차이가 있기 때문에 병합 이전에 일치시키도록 한다.

○ 스크래핑한 데이터는 하나의 DataFrame으로 합쳐 시각화한다.

○ 모든 조사가 같은 나라들을 대상으로 한 것이 아니기 때문에 포함되지 않는 국가들이 있다. 이런 국가들의 값은 상관관계가 큰 다른 지표를 구간화하여 같은 구간 내의 국가들의 평균값으로 대치하기로 한다.

○ 성과가 높은 머신러닝 모델을 찾고 Grid Search를 이용해 좋은 parameter를 찾는다. 해당 parameter에서 recall과 precision에 따른 곡선을 그리고 최적의 threshold를 찾아낸다.

○ 수집하는데 시간이 오래 걸릴 수 있으니 엑셀 파일로 저장을 병행하며 진행한다.

2. 자료 수집

(1) 스크래핑

자료 수집에서 사용된 라이브러리는 다음과 같다.

```
import numpy as np
import pandas as pd
```

```
import requests
from bs4 import BeautifulSoup
```

웹페이지 접근을 위해 requests, html 스크랩을 위해 BeautifulSoup를 이용하였으며 자료 저장을 위해 numpy와 pandas를 임포트하였다.

모든 지표는 위키항목에서 비슷한 방식을 통해 스크랩하였다. 세부적인 차이는 약간씩 있으나 중복되는 내용이 많기 때문에 한가지 지표에 대해서만 설명하도록 한다.

유엔 회원국

다음은 위키피디아의 유엔 회원국 페이지에 기재된 유엔 회원국들의 목록이다.

나라 이름 순서 목록 [편집]

국명	가입 일자
가나	1957년 3월 8일
가봉	1960년 9월 20일
가이아나	1966년 9월 20일
감비아[1]	1965년 9월 21일
과테말라	1945년 11월 21일
그레나다	1974년 9월 17일
그리스	1945년 10월 25일
기니	1958년 12월 12일
기니비사우	1974년 9월 17일
나미비아	1990년 4월 23일
나우루	1999년 9월 14일
나이지리아	1960년 10월 7일
남수단	2011년 7월 14일

```
url = 'https://ko.wikipedia.org/wiki/유엔_회원국'
```

```
page = requests.get(url)
soup = BeautifulSoup(page.text, 'html.parser')
```

해당 페이지에서 html을 스크랩한다.

```
document1 = soup.find_all('td')
len(document1)
```

```
723
```

```
document1[0].text
```

```
' 가나'
```

```
document1[1].text
```

```
'1957년 3월 8일\n'
```

```
document1[2].text
```

```
' 가봉'
```

```
document1[3].text
```

```
'1960년 9월 20일\n'
```

```
document1[4].text.strip()
```

```
'가이아나'
```

```
document1[6].text.strip()
```

```
'감비아[1]'
```

원하는 정보는 'td' 태그에 있기 때문에 document1이라는 변수에 td 태그를 포함한 모든 정보를 저장한다. document1의 홀수 번호는 국가명이 나오고 짝수 번호는 유엔 가입 연월일이 나오기 때문에 알맞게 반복문을 사용한다.

```
un_list = []
for i in range(193):
    # 중간 띄어쓰기 생략
    nation =  document1[2*i].text.replace(' ','')
    if nation[-1] == ']':
        temp_list = nation.split('[')
        nation = temp_list[0]
    un_list.append(nation)

print(len(un_list))
print(un_list[:8])
```

```
193
['가나', '가봉', '가이아나', '감비아', '과테말라', '그레나다', '그리스', '기니']
```

723개의 길이지만 유엔 회원국은 193개 뿐이니 for문을 알맞게 돌려준다. 중간에 가봉[1]과 같은 표기가 고쳐질 수 있도록 수정을 같이 해준다. 얻은 결과를 pandas의 Series로 저장한 결과는 다음과 같다.

```
0          가나
1          가봉
2        가이아나
3         감비아
4        과테말라
         ...
188      프랑스
189       피지
190      핀란드
191      필리핀
192      헝가리
Length: 193, dtype: ob
```

이와 같은 과정을 1인당 GDP, 문해율, 교육지수, 민주주의지수, 언론자유지수, 평균수명, 부패지수에 대해서 수행한다. 위키의 항목에 따라 집계된 국가 개수가 193개 이상이기도, 미만이기도 하기 때문에 for문을 사용하기 보다는 마지막 국가를 확인한 뒤 while문을 사용했다. IMF가 분류한 선진국의 경우 39개밖에 되지 않기 때문에 CIA에서 추가한 8개 국가를 더했다.

예시)

```
nation = []
gdp = []
info = gdp_doc.find_all('td')

i = 1
name = 'aa'
while name != '남수단':
    # 중간 띄어쓰기 생략
    name = info[(3*i)-1].text.replace(' ','')
    money = info[3*i].text[:-1].replace(',','')
    nation.append(name)
    gdp.append(int(money))
    i+=1

print(len(nation),len(gdp))
```

```
191 191
```

un Series에는 표기 오류가 많으니 수정을 거쳤다.

```
un[20:30]
```

```
20          대한민국
21           덴마크
22       도미니카공화국
23        도미니카연방
24     독일동독과서독참조
25          동티모르
26           라오스
27         라이베리아
28      라트비아소련참조
29    러시아[6]소련참조
Name: 0, dtype: object
```

```
def repair(x):
    x1 = x.replace('소련참조','')
    x2 = x1.replace('동독과서독참조','')
    x3 = x2.replace('유고슬라비아참조','')
    x4 = x3.replace('중화민국참조','')
    if x4[-1]== ']':
        x4 = x4[:-3]
    return x4
```

```
un = un.map(repair)
```

```
repair_dict = {'말레이시아말라야연방참조':'말레이시아','슬로바키아체코슬로바키아참조':'슬로바키아',
               '시리아[23]아랍연합공화국참조':'시리아','싱가포르말라야연방참조':'싱가포르','예멘북예멘과남예멘참조':'예멘',
              '이집트아랍연합공화국참조':'이집트','체코체코슬로바키아참조':'체코','탄자니아[39]탕가니카와잔지바르참조':'탄자니아'}
def repair2(x):
    for k,v in repair_dict.items():
        if x == k:
            x = v
    if x[-1]=='[':
        x = x[:-1]
    return x

un = un.map(repair2)
```

un 회원국은 아니지만 대만, 홍콩, 코소보의 경우 집계된 자료가 많으므로 기준 국가에 추가해 사용하도록 한다.

```
# 홍콩과 대만, 코소보는 집계된 수치가 많으니 un회원국은 아니지만 사용하기로 한다.
un[193] = '중화민국'
un[194] = '홍콩'
un[195] = '코소보'
```

(2) 데이터병합

merge 함수를 이용하기 위해서는 기준이 되는 컬럼들의 표기가 일치해야한다. 자료가 아예 집계가 되지 않아 일치하지 않는 경우도 있지만 똑같은 나라에 대해 그저 표기 차이로 인해 일치하지 않는 경우도 많기 때문에 다음과 같은 반복문을 사용해 표기를 일치시키도록 한다.

```
data_list = [developed2,corruption,demo,education,gdp,life,literacy,press]
data_str_list = ['developed2','corruption','demo','education','gdp','life','literacy','press']
```

```
# 각 목록별 un 시리즈의 국가명과 일치하지 않는것들 체크

for i in range(len(data_list)):
    data = data_list[i]
    data_name = data_str_list[i]
    cond = (un.isin(data['국가']))
    repair = un[cond==False].values
    cond2 = (data['국가'].isin(un))
    repair2 = data[cond2==False]['국가'].values
    if len(repair) < 35:
        print('{}의 수정해야할 국가들은 {}개입니다. :₩n'.format(data_name,len(repair)),
              'un에만 있는 표기',repair,'₩n₩n {}에만 있는 표기'.format(data_name),repair2)
    else:
        print('{}의 수정해야할 국가는 너무 많으니 따로 파일을 열람하시오.'.format(data_name))
    print('₩n')
```

```
education의 수정해야할 국가들은 14개입니다. :
 un에만 있는 표기 ['나우루' '마셜제도' '모나코' '북마케도니아' '산마리노' '소말리아' '에스와티니' '조선민주주의인민공화국' '중화인민공화국'
 '콩고민주공화국' '태국' '투발루' '중화민국' '코소보']

 education에만 있는 표기 ['마케도니아' '팔레스타인' '타이' '중국' '스와질란드']


gdp의 수정해야할 국가들은 9개입니다. :
 un에만 있는 표기 ['리히텐슈타인' '모나코' '소말리아' '시리아' '안도라' '조선민주주의인민공화국' '중화인민공화국' '쿠바' '중화민국']

 gdp에만 있는 표기 ['마카오' '푸에르토리코' '타이완' '중국']
```

un에만 있는 표기와 각 지표들에만 있는 표기를 짝지어서 수정해주면 된다. 만약 짝이 없는 경우는 집계가 되지 않은 것이기 때문에 뒤에서 빈 값 처리를 해주면 된다.

- corruption에는 수정할 것 없음
- demo에는 중국,대만
- education에는 타이,중국,스와질란드,마케도니아
- gdp에는 중국,타이완
- life에는 프랑스(본토),타이,중국,마케도니아,콩고,스와질란드,타이완
- literacy에는 수정할 것 없음
- press에는 코소보[c],이스라엘[d],세르비아[c],몬테네그로[c],중국

각 지표마다 다음과 같은 함수를 만들어 표기를 일치시키는 작업을 진행한다. 중복되는 과정이므로 나머지 지표들은 그림을 따로 첨부하지 않는다.

```
# education

def rep_edu(x):
    if x == '타이':
        x = '태국'
    if x == '중국':
        x = '중화인민공화국'
    if x == '스와질란드':
        x = '에스와티니'
    if x == '마케도니아':
        x = '북마케도니아'
    return x

education['국가'] = education['국가'].map(rep_edu)
```

모든 과정을 거치고 국가 컬럼에 맞게 병합(merge)을 한 뒤 선진국 여부 컬럼의 빈 값을 0으로 채워 넣으면 다음과 같은 데이터프레임이 만들어진다.

	국가	선진국여부	청렴도	민주주의지수	교육지수	gdp	평균수명	문해율	언론자유지수
0	가나	0.0	41.0	6.63	0.551	2223.0	61.03	71.5	20.81
1	가봉	0.0	31.0	3.61	0.618	8112.0	63.65	89.0	35.60
2	가이아나	0.0	37.0	6.15	0.568	5252.0	66.24	91.8	26.63
3	감비아	0.0	37.0	4.33	0.358	755.0	59.83	51.1	31.35
4	과테말라	0.0	27.0	5.26	0.508	4617.0	71.47	75.9	35.94
...	...	...	...	...	...	...	...	...	...
192	필리핀	0.0	36.0	6.64	0.637	3294.0	67.99	95.4	43.91
193	헝가리	0.0	46.0	6.63	0.834	17463.0	74.98	99.0	30.44
194	중화민국	1.0	NaN	7.73	NaN	24878.0	81.18	96.1	24.98
195	홍콩	1.0	NaN	6.02	0.822	49334.0	81.78	93.5	29.65
196	코소보	0.0	37.0	NaN	NaN	4442.0	74.15	91.9	29.68

197 rows × 9 columns

3. 예측 모델

(1) 빈 값 처리

지표별 빈 값 개수를 확인하기 전에 국가별로 빈 값의 개수를 확인한다.

```
# 빈값이 너무 많은 국가는 제거
cond = (df.isnull().sum(axis=1)>=1)
s1 = df[cond == True]['국가']
s2 = df.isnull().sum(axis=1)
```

```
s3 = s2[s2>=1]
```

```
df1 = pd.DataFrame(s1,columns=['국가'])
df2 = pd.DataFrame(s3,columns=['빈값의 개수'])
```

```
null_df = pd.merge(df1,df2,left_index=True,right_index=True)
null_df['빈값의 개수'].value_counts().sort_index()
```

```
1    17
2    14
3     5
4     7
5     1
6     3
Name: 빈값의 개수, dtype: int64
```

7개의 지표 중 1~4개의 빈 값을 가진 국가들은 개수가 많지만 5~6개의 빈 값을 가진 국가들은 그 개수도 적고 자료로서의 가치도 크지 않으니 완전히 제거하도록 한다.

```
# 빈값이 너무 많은데 4개 밖에 되지 않으니 제거하도록 한다.
null_df[null_df['빈값의 개수']>=5]
```

	국가	빈값의 개수
10	나우루	6
39	마셜제도	5
44	모나코	6
178	투발루	6

```
df.drop([10,39,44,178],axis=0,inplace=True)
```

```
df.reset_index(drop=True,inplace=True)
```

남은 국가들 중에는 많으면 4개까지의 빈 값을 가진 국가들이 있다. 이 빈 값을 채우기 위해 상관계수가 높은 다른 지표를 살펴 힌트를 얻도록 한다.
(다만 선진국 여부라는 항목은 레이블로 사용할 것이기 때문에 고려하지 않기로 한다.)

```
df.corr()
```

	선진국여부	청렴도	민주주의지수	교육지수	gdp	평균수명	문해율	언론자유지수
선진국여부	1.000000	0.724403	0.653464	0.614227	0.784802	0.582733	0.375101	-0.497759
청렴도	0.724403	1.000000	0.754702	0.692898	0.799814	0.685937	0.492228	-0.609598
민주주의지수	0.653464	0.754702	1.000000	0.634811	0.584570	0.609721	0.434454	-0.796713
교육지수	0.614227	0.692898	0.634811	1.000000	0.647391	0.814684	0.843460	-0.401827
gdp	0.784802	0.799814	0.584570	0.647391	1.000000	0.648601	0.459014	-0.421734
평균수명	0.582733	0.685937	0.609721	0.814684	0.648601	1.000000	0.716786	-0.301231
문해율	0.375101	0.492228	0.434454	0.843460	0.459014	0.716786	1.000000	-0.164576
언론자유지수	-0.497759	-0.609598	-0.796713	-0.401827	-0.421734	-0.301231	-0.164576	1.000000

- 청렴도는 민주주의지수/ 민주주의지수는 언론자유지수/ 교육지수는 문해율/ gdp는 청렴도/ 평균수명은 교육지수/ 문해율은 교육지수/ 언론자유지수는 민주주의지수와의 상관계수가 두드러진다.

상관계수가 높은 지표에 따라 그룹을 만들어 속한 그룹 내의 평균값으로 빈 값을 대치시키도록 한다. 다만 범주형 지표가 없고 모두 연속형 지표들이기 때문에 구간을 만들어 범주형 자료 취급을 하도록 한다.

```
# 문해율

cat_edu = pd.qcut(df['교육지수'],6)
cat_edu2 = pd.Categorical(cat_edu)
edu_interval = cat_edu2.codes
print("교육지수의 구간:₩n ",cat_edu2.categories)
```

```
교육지수의 구간:
  IntervalIndex([(0.205, 0.445], (0.445, 0.551], (0.551, 0.659], (0.659, 0.725], (0.725, 0.818], (0.818, 0.939]],
                closed='right',
                dtype='interval[float64]')
```

```
df['임시_교육'] = edu_interval
df['문해율'].fillna(df.groupby('임시_교육')['문해율'].transform('mean'),inplace=True)
```

위와 같은 과정을 다른 모든 컬럼들에도 적용한다.
(과정이 중복되므로 생략)

```
df.isnull().sum()
```

```
국가          0
선진국여부       0
청렴도         0
민주주의지수      0
교육지수        0
gdp         0
평균수명        0
문해율         0
언론자유지수      0
dtype: int64
```

모든 대치 후 빈 값이 없는 상태로 만들었으니 머신러닝을 적용하도록 한다.

(2) 머신러닝과 parameter

머신러닝을 적용하는 부분에서는 선진국 여부보다는 음성, 양성이라는 표현을 쓰도록 한다.
시각화를 위해 matplotlib를 임포트하고 데이터와 레이블을 분리하도록 한다.
1차적으로 정확도 측정을 위해 cross_val_score 역시 임포트한다.

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# 레이블과 데이터 분리

y = df['선진국여부']
X = df.drop(['국가','선진국여부'],axis=1)
```

```
from sklearn.model_selection import cross_val_score
```

모든 과정을 캡쳐하지는 않았지만 다음 그림에서 tree는 decision tree, svc는 support vector machine, knn은 kneighbors, log는 logistic regression(solver='liblinear') ,mlp는 신경망(solver='lbfgs')이다.

```
# binary classification은 기본적으로 Stratified Kfold 사용
cross_val_score(tree,X,y,cv=5)
```

```
array([1.        , 1.        , 1.        , 0.97368421, 0.94736842])
```

```
# 아래 예측모델들은 StandardScaler 사용
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
cross_val_score(svc,X_scaled,y,cv=5)
```

```
array([1.        , 0.94871795, 0.94871795, 0.89473684, 1.        ])
```

```
cross_val_score(knn,X_scaled,y,cv=5)
```

```
array([0.97435897, 0.97435897, 0.94871795, 0.92105263, 0.94736842])
```

```
cross_val_score(log,X_scaled,y,cv=5)
```

```
array([1.        , 0.97435897, 0.94871795, 0.89473684, 1.        ])
```

```
cross_val_score(mlp,X_scaled,y,cv=5)
```

```
array([1.        , 0.97435897, 0.94871795, 0.86842105, 0.94736842])
```

정확도 측면에서는 대부분의 분류기가 매우 좋은 성과를 보여준다.

```
y.value_counts()
```

```
0    154
1     39
Name: 선진국여부, dtype: int64
```

그러나 양성 샘플의 개수가 전체의 약 1/4 가량 밖에 안되기 때문에 분류기가 극단적으로 모든 샘플을 음성으로 분류해도 0.797 가량의 정확도가 나온다. 그렇기 때문에 정확도 외에 다른 성과측정 지표를 구해보도록 한다.

precision과 recall, FPR 등을 측정하기 위해 train set과 test set을 분리하도록 한다. 다만 샘플 들의 개수가 적기 때문에 절반 대 절반의 비율로 나누도록 한다.

```
from sklearn.model_selection import train_test_split

# 샘플의 개수가 적으므로 test_set의 크기를 키우도록 하겠음
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.5,random_state=1234)
```

로지스틱 회귀를 사용해 성과를 측정할 것이기 때문에 데이터의 스케일을 standardization을 통해 조정하도록 한다.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

우선 정확도 측면에서 최적의 parameter를 찾아 그 모델을 토대로 다른 지표들을 측정하도록 한다.

```
params = {'C':[0.01,0.1,1,10,100],'solver':['lbfgs','liblinear']}
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(LogisticRegression(),params,cv=5,return_train_score=True)
grid_search.fit(X_scaled,y)
pd.DataFrame(grid_search.cv_results_).sort_values(by='rank_test_score').head()
```

ne	param_C	param_solver	params	split0_test_score	split1_test_score	split2_test_score	...	mean_test_score	std_test_score	rank_test_score	split0_train_s
99	0.1	liblinear	{'C': 0.1, 'solver': 'liblinear'}	1.0	1.000000	0.974359	...	0.973819	0.040769	1	0.96
03	1	liblinear	{'C': 1, 'solver': 'liblinear'}	1.0	0.974359	0.948718	...	0.963563	0.039317	2	0.98
90	10	lbfgs	{'C': 10, 'solver': 'lbfgs'}	1.0	0.974359	0.974359	...	0.963428	0.035778	3	0.98
09	10	liblinear	{'C': 10, 'solver': 'liblinear'}	1.0	0.974359	0.974359	...	0.963428	0.035778	3	0.98
89	1	lbfgs	{'C': 1, 'solver': 'lbfgs'}	1.0	0.974359	0.948718	...	0.958300	0.035681	5	0.96

가장 좋은 parameter는 C = 0.1, solver = liblinear일 때이므로 이 parameter으로 먼저 다른 성과들을 측정하도록 한다.

```
log = LogisticRegression(C=0.1,solver='liblinear')
```

```
log.fit(X_train_scaled,y_train)
y_pred_scaler = log.predict(X_test_scaled)
```

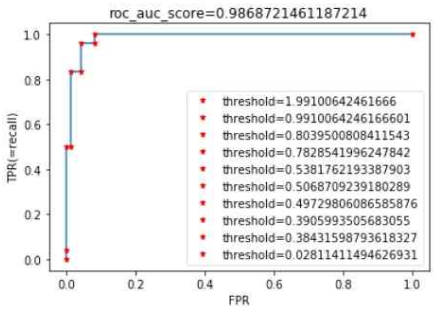
```
print(classification_report(y_test,y_pred_scaler))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        73
           1       0.88      0.88      0.88        24

    accuracy                           0.94        97
   macro avg       0.92      0.92      0.92        97
weighted avg       0.94      0.94      0.94        97
```

accuracy는 괜찮게 나오지만 precision과 recall이 낮으니 개선시킬 수 있는 방향을 찾아야한다.

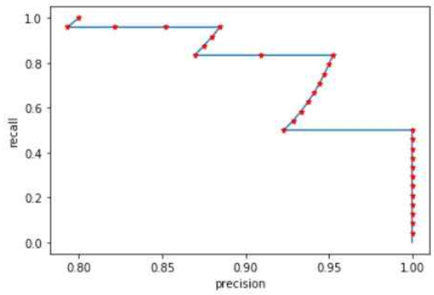
```
fpr, tpr, thresholds = roc_curve(y_test,log.predict_proba(X_test_scaled)[:,1])
auc = roc_auc_score(y_test,log.predict_proba(X_test_scaled)[:,1])
plt.figure()
plt.plot(fpr,tpr)
plt.xlabel("FPR")
plt.ylabel("TPR(=recall)")

# threshold에 따라서
for i,v in enumerate(list(thresholds)):
    plt.plot(fpr[i],tpr[i],'r*',label='threshold={}'.format(v),markersize=5)
plt.title('roc_auc_score={}'.format(auc))
plt.legend()
```

```
<matplotlib.legend.Legend at 0x2247634ba88>
```



auc 값 역시 상당히 괜찮게 나오기 때문에 precision과 recall에 관한 곡선을 그려 적당한 threshold를 찾도록 한다.

```python
from sklearn.metrics import precision_recall_curve
precision,recall,thresholds = precision_recall_curve(y_test,log.predict_proba(X_test_scaled)[:,1])

plt.figure()
plt.plot(precision,recall)
plt.xlabel("precision")
plt.ylabel("recall")

# threshold에 따라서
for i,v in enumerate(list(thresholds)):
    plt.plot(precision[i],recall[i],'r*',markersize=5)
```



precision이 대략 0.95 부근에서 recall 값 역시 괜찮아 보이므로 이 부근의 threshold를 찾도록 한다.

```python
for k in range(0,15):
    i = np.argmin(np.abs(precision-0.945-k/100))
    print('precision : {}, recall : {}, threshold: {}'.format(precision[i],recall[i],thresholds[i]))
```

```
precision : 0.9444444444444444, recall : 0.7083333333333334, threshold: 0.6068233242171948
precision : 0.9523809523809523, recall : 0.8333333333333334, threshold: 0.5381762193387903
precision : 0.9523809523809523, recall : 0.8333333333333334, threshold: 0.5381762193387903
precision : 0.9523809523809523, recall : 0.8333333333333334, threshold: 0.5381762193387903
precision : 1.0, recall : 0.5, threshold: 0.8039500808411543
precision : 1.0, recall : 0.5, threshold: 0.8039500808411543
precision : 1.0, recall : 0.5, threshold: 0.8039500808411543
precision : 1.0, recall : 0.5, threshold: 0.8039500808411543
precision : 1.0, recall : 0.5, threshold: 0.8039500808411543
precision : 1.0, recall : 0.5, threshold: 0.8039500808411543
precision : 1.0, recall : 0.5, threshold: 0.8039500808411543
precision : 1.0, recall : 0.5, threshold: 0.8039500808411543
precision : 1.0, recall : 0.5, threshold: 0.8039500808411543
precision : 1.0, recall : 0.5, threshold: 0.8039500808411543
precision : 1.0, recall : 0.5, threshold: 0.8039500808411543
```

threshold가 0.54 정도이면 precision과 recall 값이 괜찮게 나온다. 단지 더 만족스러운 recall 값을 얻기 위해서 parameter를 바꾼 모델로 같은 과정을 반복하도록 한다.

solver를 기본값인 lbfgs로 두면 이때 정확도 측면에서 최적의 C 값은 10이다.
(위의 GridSearch 결과 참고)

```python
log = LogisticRegression(C=10)
```

```python
log.fit(X_train_scaled,y_train)
y_pred_scaler = log.predict(X_test_scaled)
```

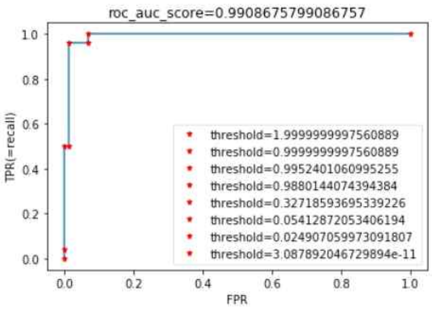
```python
print(classification_report(y_test,y_pred_scaler))
```

```
              precision    recall  f1-score   support

           0       0.94      0.99      0.96        73
           1       0.95      0.79      0.86        24

    accuracy                           0.94        97
   macro avg       0.94      0.89      0.91        97
weighted avg       0.94      0.94      0.94        97
```

precision과 accuracy가 상당히 높지만 recall이 낮기 때문에 역시 적당한 threshold를 찾도록 한다.

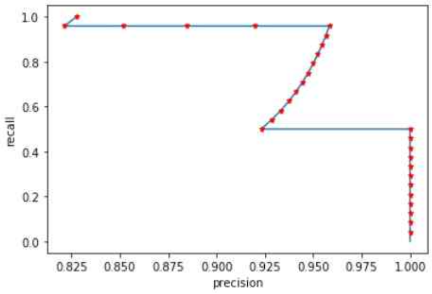
```
fpr, tpr, thresholds = roc_curve(y_test,log.predict_proba(X_test_scaled)[:,1])
auc = roc_auc_score(y_test,log.predict_proba(X_test_scaled)[:,1])
plt.figure()
plt.plot(fpr,tpr)
plt.xlabel("FPR")
plt.ylabel("TPR(=recall)")

# threshold에 따라서
for i,v in enumerate(list(thresholds)):
    plt.plot(fpr[i],tpr[i],'r*',label='threshold={}'.format(v),markersize=5)
plt.title('roc_auc_score={}'.format(auc))
plt.legend()
```

```
<matplotlib.legend.Legend at 0x224764c2b08>
```



auc 값 역시 매우 높게 나온다.

```
precision,recall,thresholds = precision_recall_curve(y_test,log.predict_proba(X_test_scaled)[:,1])

plt.figure()
plt.plot(precision,recall)
plt.xlabel("precision")
plt.ylabel("recall")

# threshold에 따라서
for i,v in enumerate(list(thresholds)):
    plt.plot(precision[i],recall[i],'r*',markersize=5)
```



precision이 대략 0.95 부근에서 recall 값 역시 높아 보이기 때문에 이 부근에서 threshold를 찾도록 한다.

```
for k in range(0,15):
    i = np.argmin(np.abs(precision-0.945-k/100))
    print('precision : {}, recall : {}, threshold: {}'.format(precision[i],recall[i],thresholds[i]))
```

```
precision : 0.9444444444444444, recall : 0.7083333333333334, threshold: 0.8227574941522426
precision : 0.9545454545454546, recall : 0.875, threshold: 0.4098948021210839
precision : 0.9583333333333334, recall : 0.9583333333333334, threshold: 0.32718593695339226
precision : 0.9583333333333334, recall : 0.9583333333333334, threshold: 0.32718593695339226
precision : 1.0, recall : 0.5, threshold: 0.9952401060995255
precision : 1.0, recall : 0.5, threshold: 0.9952401060995255
precision : 1.0, recall : 0.5, threshold: 0.9952401060995255
precision : 1.0, recall : 0.5, threshold: 0.9952401060995255
precision : 1.0, recall : 0.5, threshold: 0.9952401060995255
precision : 1.0, recall : 0.5, threshold: 0.9952401060995255
precision : 1.0, recall : 0.5, threshold: 0.9952401060995255
precision : 1.0, recall : 0.5, threshold: 0.9952401060995255
precision : 1.0, recall : 0.5, threshold: 0.9952401060995255
precision : 1.0, recall : 0.5, threshold: 0.9952401060995255
precision : 1.0, recall : 0.5, threshold: 0.9952401060995255
```

threshold가 0.33 정도일 때 precision과 recall이 모두 매우 높은 수치임을 확인할 수 있다.
비록 정확도 측면에서는 solver='liblinear'일 때 더 높은 값을 갖지만 recall과 precision의 측면에서는 solver='lbfgs'일 때 parameter 조정을 통해 더 좋은 성과를 얻어낼 수 있다.

이 threshold에서의 정확도 역시 다시 측정해보기로 한다.

```
real_pred_df = pd.DataFrame(y_test)
real_pred_df['예측값'] = log.predict_proba(X_test_scaled)[:,1]>0.33
```

```
def convert_bool(x):
    if x == False:
        x=0
    else:
        x=1
    return x
real_pred_df['예측값'] = real_pred_df['예측값'].map(convert_bool)
```

```
real_pred_df[15:25]
```

	선진국여부	예측값
58	0	0
29	0	0
51	1	1
6	1	1
136	0	0
180	0	0
5	0	0

```
real_pred_df.sum(axis=1).value_counts()
```

```
0    72
2    22
1     3
dtype: int64
```

분류가 옳게 되었으면 실제값과 예측값이 모두 0이거나 모두 1로써 그 합이 0 혹은 2가 나와야한다. 실제값과 예측값에 대해 DataFrame을 만들어 가로축으로의 합을 구하고 원소별로 개수를 셌을 때 오분류는 단 3개만 관측된다.

```
94/97
```

```
0.9690721649484536
```

test set 97개 중에 94개가 옳게 예측된 것이니 정확도는 0.97이 나온다.

```
# 이 모델이 판단을 내릴 때 가장 영향을 많이 미치는 변수
pd.DataFrame(list(log.coef_),columns=list(X.columns))
```

	청렴도	민주주의지수	교육지수	gdp	평균수명	문해율	언론자유지수
0	-0.533966	1.284279	2.246487	3.548757	1.773667	0.103054	-1.808646

이 모델이 판단을 내릴 때는 gdp, 교육지수, 언론자유지수, 평균수명, 민주주의지수, 청렴도, 문해율 순으로 영향을 미친다.

4. 결론과 논의

Logistic Regression의 parameter가 solver='lbfgs'이고 C=10일 때 threshold를 0.33으로 잡으면 정확도 : 0.9691, precision : 0.95833, recall : 0.95833의 성과가 나온다.

이 모델에서 gdp, 교육지수, 언론자유지수, 평균수명, 민주주의지수, 청렴도, 문해율 순으로 클래스 분류에 중요도를 보인다.

변형시켜서 더 좋은 성과를 얻은 threshold는 predict_proba 즉 예측 확률에 관한 것이다.
기본 값일 때는 1번에 속할 확률이 0.5보다 클 때 1번 클래스로 분류하는데 그 임계치(threshold)를 0.33으로 낮추게 되면 0번에 속할 확률이 0.76이고 1번에 속할 확률이 0.34만 되더라도 1번 클래스로 분류하게 된다. 그럼에도 오히려 여러 수치들이 개선되는 현상이 생긴 것은 양성과 음성의 개수가 맞지 않는 불균형 data set이기 때문에 그런 것이 아닌가 추측된다.

다음은 이 모델이 최종적으로 오분류한 국가와 선진국 여부에 따른 각 지표의 평균값이다.

```
# 해당 모델이 잘못 분류한 케이스
df.iloc[y_test[real_pred_df.sum(axis=1)==1].index]
```

	국가	선진국여부	청렴도	민주주의지수	교육지수	gdp	평균수명	문해율	언론자유지수
27	라트비아	1	58.000000	7.490000	0.835	18172.0	73.950000	99.8	19.53
106	안도라	1	46.076923	6.412069	0.718	15956.5	74.164839	100.0	24.63
151	카타르	0	62.000000	3.190000	0.698	69688.0	78.880000	96.3	42.51

```
df.groupby('선진국여부').mean()
```

선진국여부	청렴도	민주주의지수	교육지수	gdp	평균수명	문해율	언론자유지수
0	36.760221	4.713732	0.584754	6705.213822	68.121165	81.258156	38.961710
1	67.473866	8.080886	0.845749	43307.153846	80.287531	98.396325	20.426304

IMF 기준 선진국이 아닌 카타르는 민주주의지수와 언론자유지수가 매우 좋지 않지만 모델에 가장 큰 영향을 미치는 1인당 gdp가 매우 높아 양성으로 분류된 듯하다. 라트비아와 안도라는 IMF 기준 선진국이지만 두드러지는 수치가 없어 절대 다수인 음성으로 분류된 듯하다. threhold를 더 낮췄으면 라트비아와 안도라가 선진국으로 분류될 수 있었겠지만 카타르처럼 어느 하나가 두드러진 국가들 역시 선진국으로 오분류되었을 가능성이 크다. threshold가 0.5에서 0.33으로 낮아짐에 따라 라트비아, 안도라처럼 두드러진 것이 없는 국가들 중 음성에서 양성으로 옮겨갔을 것이라 추측할 수 있다.

실제로 아래와 같이 threshold가 0.5일 때는 양성으로 분류한 것 중 실제 양성 19개, 음성 1개였으나 (precision은 처음부터 꽤 좋았음) threshold를 0.33으로 낮추니 양성으로 분류한 것 중 실제 양성 22개, 음성 1개로써 몇몇 국가들이 옮겨갔음을 확인할 수 있다.

```
df.iloc[y_test[log.predict(X_test_scaled)==1].index]['선진국여부'].value_counts()
```

```
1    19
0     1
Name: 선진국여부, dtype: int64
```

```
df.iloc[y_test[real_pred_df['예측값']==1].index]['선진국여부'].value_counts()
```

```
1    22
0     1
Name: 선진국여부, dtype: int64
```

사실 이 데이터셋은 원래부터 레이블과 개별 지표간의 상관계수가 매우 높은 상태였다. 또 선진국들과 후진국 들 간의 차이는 사실 극단적인 경우가 많다. 그렇기 때문에 위의 안도라, 라트비아처럼 극단적으로 좋은 지표들을 갖지는 않았지만 선진국으로 분류되어있거나, 카타르처럼 선진국으로 분류되어있지 않지만 몇몇 지표들에서는 굉장히 뚜렷한 수치를 갖는 국가들을 어떻게 분류해내느냐가 핵심이었다. threshold를 조정함으로써 라트비아, 안도라와 같이 약간 애매한 위치에 있는 국가들을 양성으로 이동시켰기 때문에 유의미한 변화를 주었다고 판단할 수 있다.

□ 자료 출처

모든 지표는 위키피디아에서 얻어올 수 있다. 단, 위키백과 내에서도 출처가 확실한 경우만을 사용한다. 아래 지표들은 위키백과에서 인용한 1차 출처에 대해서만 표시하였다.

- 기준 국가들은 UN 가입국으로 한다.
- 선진국 여부는 국제통화기금에서 2016년 분류한 기준을 따른다.
  (http://www.imf.org/external/pubs/ft/weo/2016/01/pdf/text.pdf)
- 1인당 GDP : 국제통화기금(2019년)
- 평균수명 : UN(2018년)
- 문해율 : CIA 월드팩트북(2013년)
- 언론자유 : RSF(2019년)
- 민주주의지수 : The Economist(2017년)
- 부패지수 : 국제투명성기구(TI)(2018년)
- 교육 지수 : UNITED NATIONS DEVELOPMENT PROGRAMME(2016년)