In [1]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import mglearn
%matplotlib inline
import seaborn as sns
import platform
from matplotlib import font_manager , rc

if platform.system() == 'Darwin':
  rc('font' , family = 'AppleGothic')
elif platform.system() == 'Windows':
  path = 'C:/Windows/Fonts/malgun.ttf'
  font_name = font_manager.FontProperties(fname = path).get_name()
  rc('font' , family = font_name)
else:
  print('모름')
plt.rcParams['axes.unicode_minus'] = False
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import accuracy_score , precision_score , recall_score , roc_auc_score , f1_score , confusion_matrix , roc_curve
```

executed in 1.67s, finished 12:18:34 2023-10-27

# 1 문제 정의

- 로지스틱회귀(Logistic Regression)를 이용한 타이타닉 생존자 예측
- 목표 : 타이타닉 승객 데이터셋을 이용하여 생존 여부 예측

In [2]:
```
titanic = pd.read_csv('titanic.csv')
titanic
```

executed in 27ms, finished 12:21:03 2023-10-27

Out[2]:

| | Unnamed: 0 | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Embarked | Survived |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | S | 0 |
| **1** | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | C | 1 |
| **2** | 2 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | S | 1 |
| **3** | 3 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | 0 | 113803 | S | 1 |
| **4** | 4 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | 0 | 373450 | S | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **884** | 884 | 2 | Montvila, Rev. Juozas | male | 27.0 | 0 | 0 | 211536 | S | 0 |
| **885** | 885 | 1 | Graham, Miss. Margaret Edith | female | 19.0 | 0 | 0 | 112053 | S | 1 |
| **886** | 886 | 3 | Johnston, Miss. Catherine Helen "Carrie" | female | 28.0 | 1 | 2 | W./C. 6607 | S | 0 |
| **887** | 887 | 1 | Behr, Mr. Karl Howell | male | 26.0 | 0 | 0 | 111369 | C | 1 |
| **888** | 888 | 3 | Dooley, Mr. Patrick | male | 32.0 | 0 | 0 | 370376 | Q | 0 |

889 rows × 10 columns

In [3]:
```
titanic.columns
```

executed in 17ms, finished 12:21:38 2023-10-27

Out[3]:
```
Index(['Unnamed: 0', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch',
       'Ticket', 'Embarked', 'Survived'],
      dtype='object')
```

In [24]:
```
data = titanic[['Pclass', 'Age', 'SibSp', 'Parch']].to_numpy()
target = titanic.iloc[:,-1].to_numpy()
```

executed in 5ms, finished 12:31:31 2023-10-27

In [6]:
```
titanic[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked']].describe()
```

executed in 114ms, finished 12:23:24 2023-10-27

Out[6]:

| | Pclass | Age | SibSp | Parch |
|---|---|---|---|---|
| **count** | 889.000000 | 889.000000 | 889.000000 | 889.000000 |
| **mean** | 2.311586 | 29.315152 | 0.524184 | 0.382452 |
| **std** | 0.834700 | 12.984932 | 1.103705 | 0.806761 |
| **min** | 1.000000 | 0.420000 | 0.000000 | 0.000000 |
| **25%** | 2.000000 | 22.000000 | 0.000000 | 0.000000 |
| **50%** | 3.000000 | 28.000000 | 0.000000 | 0.000000 |
| **75%** | 3.000000 | 35.000000 | 1.000000 | 0.000000 |
| **max** | 3.000000 | 80.000000 | 8.000000 | 6.000000 |

In [7]:

```
titanic[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked']].corr()
```

executed in 15ms, finished 12:24:12 2023-10-27

Out[7]:

| | Pclass | Age | SibSp | Parch |
|---|---|---|---|---|
| **Pclass** | 1.000000 | -0.336512 | 0.081656 | 0.016824 |
| **Age** | -0.336512 | 1.000000 | -0.232543 | -0.171485 |
| **SibSp** | 0.081656 | -0.232543 | 1.000000 | 0.414542 |
| **Parch** | 0.016824 | -0.171485 | 0.414542 | 1.000000 |

In [16]:

```
titanic.iloc[:,-1].value_counts()
```

executed in 20ms, finished 12:28:15 2023-10-27

Out[16]:

```
0    549
1    340
Name: Survived, dtype: int64
```

In [9]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
kn = KNeighborsClassifier()
lr = LogisticRegression()
dt = DecisionTreeClassifier()
```

executed in 26ms, finished 12:26:54 2023-10-27

In [10]:

```
from sklearn.model_selection import train_test_split
```

executed in 5ms, finished 12:27:10 2023-10-27

In [25]:

```
train_input , test_input , train_target , test_target = train_test_split(data , target , stratify = target , test_size = 0.3)
```

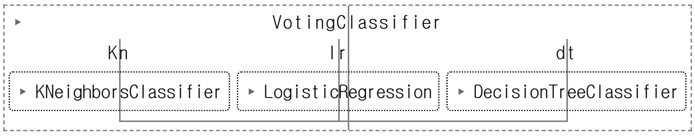
executed in 13ms, finished 12:31:37 2023-10-27

In [26]:

```
from sklearn.ensemble import VotingClassifier
vc = VotingClassifier(estimators = [('Kn' , kn),('lr' , lr),('dt' , dt)] ,voting = 'soft')
vc.fit(train_input , train_target)
```

executed in 54ms, finished 12:31:39 2023-10-27

Out[26]:



- 죽었다고 예측했을 때 살아있는 것이 큰 오점이므로 , 정밀도를 예측해보자

In [56]:

```
pred = vc.predict(test_input)
```

executed in 20ms, finished 12:47:31 2023-10-27

In [33]:

```
precision = precision_score(test_target , pred)
accuracy = accuracy_score(test_target , pred)
```

executed in 13ms, finished 12:39:32 2023-10-27

In [30]:

```
print('보팅 분류기의 정밀도 : ' , precision)
```

executed in 10ms, finished 12:36:12 2023-10-27

```
보팅 분류기의 정밀도 :  0.6436781609195402
```

In [49]:

```
model = [kn , lr , dt]
name = [vc.__class__.__name__]
pre = [precision]
acc = [accuracy]
print('보팅 분류기의 정밀도 : ' , precision , ' 보팅 분류기의 정확도 : ' , accuracy)
for i in model:
    i.fit(train_input , train_target)
    pred1 = i.predict(test_input)
    precision1 = precision_score(test_target , pred1)
    model_name = i.__class__.__name__
    accuracy1 = accuracy_score(test_target , pred1)
    name.append(model_name)
    pre.append(precision1)
    acc.append(accuracy1)
    print(f'{model_name}의 정밀도 : {precision1} , 정확도 : {accuracy1}')
```

executed in 42ms, finished 12:45:57 2023-10-27

```
보팅 분류기의 정밀도 :  0.6436781609195402  보팅 분류기의 정확도 :  0.7116104868913857
KNeighborsClassifier의 정밀도 : 0.6438356164383562 , 정확도 : 0.6966292134831461
LogisticRegression의 정밀도 : 0.6323529411764706 , 정확도 : 0.6853932584269663
DecisionTreeClassifier의 정밀도 : 0.68 , 정확도 : 0.7191011235955056
```

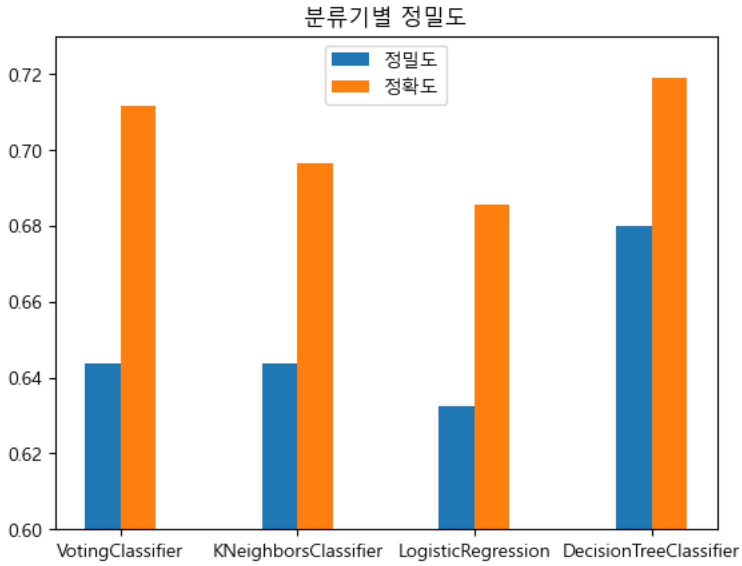
- 모든 분류기가 대체로 정확도가 낮다.

In [55]:
```
bar_width = 0.2
plt.bar(x = np.arange(4) - bar_width/2 , width = 0.2 ,  height = pre , label = '정밀도')
plt.bar(x = np.arange(4) + bar_width/2 , width = 0.2 , height = acc , label = '정확도')
plt.ylim(0.6 , 0.73)
plt.xticks(np.arange(4) , name)
plt.legend(loc = 'upper center')

plt.title('분류기별 정밀도')
```

executed in 161ms, finished 12:46:51 2023-10-27

Out[55]:
```
Text(0.5, 1.0, '분류기별 정밀도')
```



- 정밀도가 높다 = 죽었다고 예측했는데 살아있는 경우가 적다 => 손실을 최소화할 수 있는 방법?