특징 선택

실무에서는 대규모의 데이터를 기반으로 분류예측 모형을 만들어야 하는 경우가 많다. 대규모의 데이터라고 하면 표본의 갯수가 많거나 아니면 독립변수 즉, 특징데이터의 종류가 많거나 혹은 이 두가지 모두인 경우가 있다. 여기에서는 특징데이터의 종류가 많은 경우에 가장 중요하다고 생각되는 특징데이터만 선택하여 특징데이터의 종류를 줄이기 위한 방법에 대해 알아본다.

In [1]:

```
%%time
from sklearn.datasets import fetch_rcv1
rcv_train = fetch_rcv1(subset="train")
rcv_test = fetch_rcv1(subset="test")
X_train = rcv_train.data
y_train = rcv_train.target
X_test = rcv_test.data
y_test = rcv_test.target

# Ont-Hot-Encoding된 라벨을 정수형으로 복원
classes = np.arange(rcv_train.target.shape[1])
y_train = y_train.dot(classes)
y_test = y_test.dot(classes)

print(X_train.shape)
```

```
(23149, 47236)
CPU times: user 7.99 s, sys: 1.14 s, total: 9.13 s
Wall time: 9.15 s
```

분산에 의한 선택

원래 예측모형에서 중요한 특징데이터란 종속데이터와의 상관관계가 크고 예측에 도움이 되는 데이터를 말한다. 하지만 상관관계 계산에 앞서 특징데이터의 값 자체가 표본에 따라 그다지 변하지 않는다면 종속데이터 예측에도 도움이 되지 않을 가능성이 높다. 따라서 표본 변화에 따른 데이터 값의 변화 즉, 분산이 기준치보다 낮은 특징데이터는 사용하지 않는 방법이 분산에 의한 선택 방법이다. 예를 들어 종속데이터와 특징데이터가 모두 0 또는 1 두가지 값만 가지는데 종속데이터는 0과 1이 균형을 이루는데 반해 특징데이터가 대부분(예를 들어 90%)의 값이 0이라면 이 특징데이터는 분류에 도움이 되지 않을 가능성이 높다.

하지만 분산에 의한 선택은 반드시 상관관계와 일치한다는 보장이 없기 때문에 신중하게 사용해야 한다.

In [2]:

```
from sklearn.feature_selection import VarianceThreshold

selector = VarianceThreshold(1e-5)
X_train_sel = selector.fit_transform(X_train)
X_test_sel = selector.transform(X_test)
X_train_sel.shape
```

Out[2]:

```
(23149, 14330)
```

In [3]:

```
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score
```

In [4]:

```
%%time
model = BernoulliNB()
model.fit(X_train, y_train)
print("train accuracy:{:5.3f}".format(accuracy_score(y_train, model.predict(X_train))))
print("test accuracy :{:5.3f}".format(accuracy_score(y_test, model.predict(X_test))))
```

```
train accuracy:0.381
test accuracy :0.324
CPU times: user 23.6 s, sys: 4.05 s, total: 27.6 s
Wall time: 23.7 s
```

In [5]:

```
%%time
model = BernoulliNB()
model.fit(X_train_sel, y_train)
print("train accuracy:{:5.3f}".format(accuracy_score(y_train, model.predict(X_train_sel))))
print("test accuracy :{:5.3f}".format(accuracy_score(y_test, model.predict(X_test_sel))))
```

```
train accuracy:0.529
test accuracy :0.441
CPU times: user 19.8 s, sys: 3.23 s, total: 23 s
Wall time: 19.9 s
```

단일 변수 선택

단일 변수 선택법은 각각의 독립변수를 하나만 사용한 예측모형의 성능을 이용하여 가장 분류성능 혹은 상관관계가 높은 변수만 선택하는 방법이다. 사이킷런 패키지의 feature_selection 서브패키지는 다음 성능지표를 제공한다.

- chi2 : 카이제곱 검정 통계값
- f_classif : 분산분석(ANOVA) F검정 통계값
- mutual_info_classif : 상호정보량(mutual information)

하지만 단일 변수의 성능이 높은 특징만 모았을 때 전체 성능이 반드시 향상된다는 보장은 없다.

feature_selection 서브패키지는 성능이 좋은 변수만 사용하는 전처리기인 `SelectKBest` 클래스도 제공한다. 사용법은 다음과 같다.

In [6]:

```
from sklearn.feature_selection import chi2, SelectKBest
```

In [7]:

```
%%time

selector1 = SelectKBest(chi2, k=14330)
X_train1 = selector1.fit_transform(X_train, y_train)
X_test1 = selector1.transform(X_test)

model = BernoulliNB()
model.fit(X_train1, y_train)
print("train accuracy:{:5.3f}".format(accuracy_score(y_train, model.predict(X_train1))))
print("test accuracy :{:5.3f}".format(accuracy_score(y_test, model.predict(X_test1))))
```

```
train accuracy:0.505
test accuracy :0.438
CPU times: user 19.8 s, sys: 4.15 s, total: 24 s
Wall time: 19.8 s
```

다른 모형을 이용한 특성 중요도 계산

특성 중요도(feature importance)를 계산할 수 있는 랜덤포레스트 등의 다른 모형을 사용하여 일단 특성을 선택하고 최종 분류는 다른 모형을 사용할 수도 있다.

In [8]:

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import ExtraTreesClassifier
```

In [9]:

```
%%time
n_sample = 10000
idx = np.random.choice(range(len(y_train)), n_sample)
model_sel = ExtraTreesClassifier(n_estimators=50).fit(X_train[idx, :], y_train[idx])
selector = SelectFromModel(model_sel, prefit=True, max_features=14330)
X_train_sel = selector.transform(X_train)
X_test_sel = selector.transform(X_test)
```

```
CPU times: user 26.7 s, sys: 710 ms, total: 27.4 s
Wall time: 26.9 s
```

In [10]:

```
%%time
model = BernoulliNB()
model.fit(X_train_sel, y_train)
print("train accuracy:{:5.3f}".format(accuracy_score(y_train, model.predict(X_train_sel))))
print("test accuracy :{:5.3f}".format(accuracy_score(y_test, model.predict(X_test_sel))))
```

```
train accuracy:0.604
test accuracy :0.491
CPU times: user 18.7 s, sys: 3.73 s, total: 22.5 s
Wall time: 19.9 s
```