Ch1_(2)

March 4, 2025

1 Feature Transformation

[1]:
```
import pandas as pd
import numpy as np
datadict = {
    'F1': np.random.rand(100),
    'F2': np.random.randint(1, 100, size=100),
    'F3': np.random.randn(100),
    'F4': np.random.uniform(0, 10, size=100),
    'F5': np.random.normal(50, 10, size=100),
    'F6': np.random.exponential( 5, size=100 ),
}
data = pd.DataFrame( datadict )
X_train = data [ :75 ]
X_test = data [ 75: ]
```

[2]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 75 entries, 0 to 74
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   F1      75 non-null     float64
 1   F2      75 non-null     int32
 2   F3      75 non-null     float64
 3   F4      75 non-null     float64
 4   F5      75 non-null     float64
 5   F6      75 non-null     float64
dtypes: float64(5), int32(1)
memory usage: 3.4 KB
```

[3]:
```
from sklearn.preprocessing import StandardScaler
scaler1 = StandardScaler()
scaler1.fit( X_train )

X_train1 = scaler1.transform( X_train )
```

```
X_test1 = scaler1.transform( X_test )

print( X_train1.mean( axis=0 ))
print( X_test1.mean( axis=0 ))

print( X_train1.std( axis=0 ))
print( X_test1.std( axis=0 ))
```

```
[ 2.98973809e-16  5.29206308e-17  2.96059473e-17 -3.25665421e-17
  1.95399252e-16 -1.70095419e-16]
[-0.03611417  0.02758654  0.06801525 -0.36748011 -0.0440568   0.35893671]
[1. 1. 1. 1. 1. 1.]
[1.2675044  0.95067374 0.98181746 0.98570533 1.07285923 1.51575508]
```

[4]:
```
from sklearn.preprocessing import MinMaxScaler
scaler2 = MinMaxScaler()
scaler2.fit( X_train )

X_train2 = scaler2.transform( X_train )
X_test2 = scaler2.transform( X_test )

print( X_train2.max( axis=0 ))
print( X_test2.max( axis=0 ))

print( X_train2.min( axis=0 ))
print( X_test2.min( axis=0 ))
```

```
[1. 1. 1. 1. 1. 1.]
[1.00039474 0.97959184 0.92928805 0.89276434 0.77309881 1.28211469]
[0. 0. 0. 0. 0. 0.]
[ 0.02419875  0.05102041  0.07300726  0.00076903  0.00537292 -0.00085909]
```

[5]:
```
datadict2 = {
    'F1': np.random.gamma(2, 2, 1000),
    'F2': np.random.normal(0, 1, 1000),
    'F3': np.random.uniform(0, 1, 1000)
}
data2 = pd.DataFrame(datadict2)
```

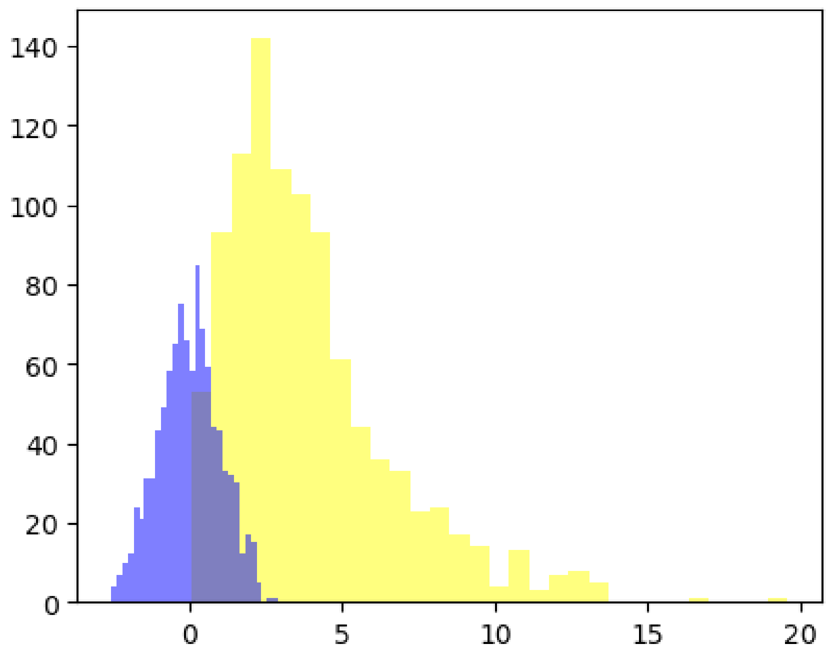
[6]:
```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer ( method ='yeo-johnson' )
pt.fit( data2 )

data2tr = pt.transform( data2 )

import matplotlib.pyplot as plt
plt.figure( figsize = (5, 4) )
```

```
plt.hist( data2[ 'F1' ], bins=30, color='yellow', alpha=0.5 )
plt.hist( data2tr[ :, 0 ], bins=30, color='blue', alpha=0.5)
```

```
[6]: (array([ 4.,  7., 10., 12., 24., 21., 31., 31., 43., 49., 58., 65., 75.,
            66., 58., 85., 69., 59., 44., 43., 33., 32., 30., 12., 17., 15.,
             5.,  0.,  1.,  1.]),
      array([-2.56731486, -2.38444342, -2.20157198, -2.01870055, -1.83582911,
            -1.65295767, -1.47008624, -1.2872148 , -1.10434336, -0.92147192,
            -0.73860049, -0.55572905, -0.37285761, -0.18998617, -0.00711474,
             0.1757567 ,  0.35862814,  0.54149957,  0.72437101,  0.90724245,
             1.09011389,  1.27298532,  1.45585676,  1.6387282 ,  1.82159963,
             2.00447107,  2.18734251,  2.37021395,  2.55308538,  2.73595682,
             2.91882826]),
      <BarContainer object of 30 artists>)
```

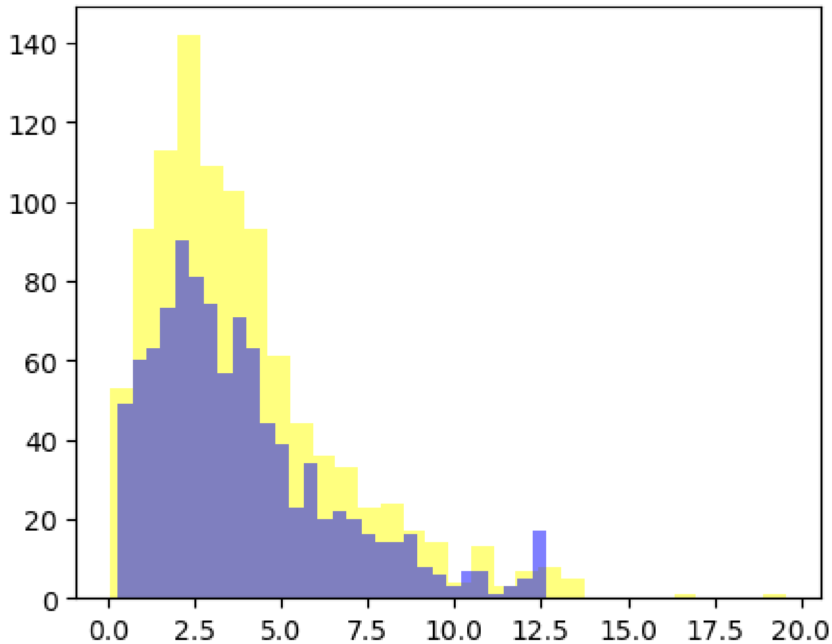



```
[7]: data2_01 = data2.quantile( 0.01 )
     data2_99 = data2.quantile( 0.99 )
     data2tr2 = data2.clip( data2_01, data2_99, axis=1 )

     plt.figure( figsize = (5, 4) )
     plt.hist( data2['F1'], bins=30, color='yellow', alpha=0.5)
     plt.hist( data2tr2['F1'], bins=30, color='blue', alpha=0.5)
```

```
[7]: (array([49., 60., 63., 73., 90., 81., 74., 57., 71., 63., 44., 39., 23.,
            34., 20., 22., 20., 16., 14., 14., 16.,  8.,  6.,  3.,  7.,  7.,
```

```
         1.,   3.,   5., 17.]),
 array([ 0.30024477,  0.71209224,  1.1239397 ,  1.53578716,  1.94763463,
         2.35948209,  2.77132955,  3.18317702,  3.59502448,  4.00687194,
         4.41871941,  4.83056687,  5.24241433,  5.6542618 ,  6.06610926,
         6.47795672,  6.88980419,  7.30165165,  7.71349911,  8.12534658,
         8.53719404,  8.9490415 ,  9.36088897,  9.77273643, 10.18458389,
        10.59643136, 11.00827882, 11.42012628, 11.83197375, 12.24382121,
        12.65566867]),
 <BarContainer object of 30 artists>)
```



[8]:
```python
bin_bdr = [0, 2.5, 5.0, 7.5, float('inf')]
F1_bin = pd.cut( data2['F1'], bin_bdr, labels=False )
F1_bin
```

[8]:
```
0      2
1      0
2      3
3      2
4      1
      ..
995    0
996    0
997    1
998    0
999    2
```

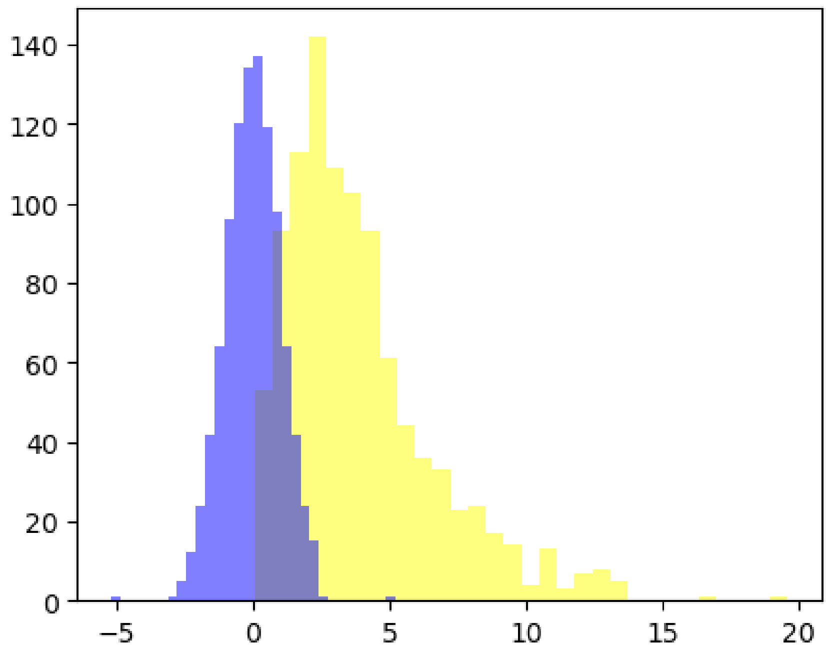
```
Name: F1, Length: 1000, dtype: int64
```

[9]:
```python
F2_rank = data2['F2'].rank()
F2_rank / data2.shape[0]
```

[9]:
```
0      0.709
1      0.870
2      0.884
3      0.142
4      0.149
       …
995    0.394
996    0.057
997    0.471
998    0.171
999    0.130
Name: F2, Length: 1000, dtype: float64
```

[10]:
```python
from sklearn.preprocessing import QuantileTransformer
qt = QuantileTransformer ( n_quantiles=100, output_distribution='normal' )
qt.fit( data2 )
data2tr3 = qt.transform( data2 )
plt.figure( figsize = (5, 4) )
plt.hist( data2['F1'], bins=30, color='yellow', alpha=0.5)
plt.hist( data2tr3[ :, 0 ], bins=30, color='blue', alpha=0.5)
```

[10]:
```
(array([  1.,   0.,   0.,   0.,   0.,   0.,   1.,   5.,  12.,  24.,  42.,
         64.,  96., 120., 134., 137., 119.,  98.,  64.,  42.,  24.,  15.,
          1.,   0.,   0.,   0.,   0.,   0.,   0.,   1.]),
 array([-5.19933758e+00, -4.85271508e+00, -4.50609257e+00, -4.15947007e+00,
        -3.81284756e+00, -3.46622506e+00, -3.11960255e+00, -2.77298004e+00,
        -2.42635754e+00, -2.07973503e+00, -1.73311253e+00, -1.38649002e+00,
        -1.03986752e+00, -6.93245011e-01, -3.46622505e-01,  4.89217555e-11,
         3.46622506e-01,  6.93245011e-01,  1.03986752e+00,  1.38649002e+00,
         1.73311253e+00,  2.07973503e+00,  2.42635754e+00,  2.77298004e+00,
         3.11960255e+00,  3.46622506e+00,  3.81284756e+00,  4.15947007e+00,
         4.50609257e+00,  4.85271508e+00,  5.19933758e+00]),
 <BarContainer object of 30 artists>)
```



[11]:
```
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

city = { 'city': ['Seoul', 'Tokyo', 'Paris', 'Paris', 'Tokyo', 'Seoul',
                  'London', 'Madrid', 'Seoul', 'Beijing', 'London', 'Paris'] }
citydf = pd.DataFrame( city )
label_encoder = LabelEncoder()
citydf[ 'city_encoded' ] = label_encoder.fit_transform( citydf ['city'] )
citydf
```

[11]:

	city	city_encoded
0	Seoul	4
1	Tokyo	5
2	Paris	3
3	Paris	3
4	Tokyo	5
5	Seoul	4
6	London	1
7	Madrid	2
8	Seoul	4
9	Beijing	0
10	London	1
11	Paris	3

[12]:
```
one_hot_encoder = OneHotEncoder( sparse_output = False )
one_hot_encoded = one_hot_encoder.fit_transform( citydf[[ 'city' ]] )
```

```
pd.DataFrame( one_hot_encoded, columns = label_encoder.classes_)
```

[12]:

```
    Beijing  London  Madrid  Paris  Seoul  Tokyo
0       0.0     0.0     0.0    0.0    1.0    0.0
1       0.0     0.0     0.0    0.0    0.0    1.0
2       0.0     0.0     0.0    1.0    0.0    0.0
3       0.0     0.0     0.0    1.0    0.0    0.0
4       0.0     0.0     0.0    0.0    0.0    1.0
5       0.0     0.0     0.0    0.0    1.0    0.0
6       0.0     1.0     0.0    0.0    0.0    0.0
7       0.0     0.0     1.0    0.0    0.0    0.0
8       0.0     0.0     0.0    0.0    1.0    0.0
9       1.0     0.0     0.0    0.0    0.0    0.0
10      0.0     1.0     0.0    0.0    0.0    0.0
11      0.0     0.0     0.0    1.0    0.0    0.0
```

[13]:

```
freq = citydf['city'].value_counts()
freq
```

[13]:

```
city
Seoul      3
Paris      3
Tokyo      2
London     2
Madrid     1
Beijing    1
Name: count, dtype: int64
```

[14]:

```
citydf['city'].map( freq )
```

[14]:

```
0     3
1     2
2     3
3     3
4     2
5     3
6     2
7     1
8     3
9     1
10    2
11    3
Name: city, dtype: int64
```

[15]:

```
tgdict = {
    'F1': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank'],
    'F2': ['Female', 'Male', 'Male', 'Male', 'Female', 'Male'],
```

```
    'Y' : [ 20, 50, 60, 80, 30, 50 ]
}
tgdf = pd.DataFrame( tgdict )
```

[16]:
```
tg_mean = tgdf.groupby('F2')['Y'].mean()
tg_mean
```

[16]:
```
F2
Female    25.0
Male      60.0
Name: Y, dtype: float64
```

[17]:
```
tgdf['F3'] = tgdf['F2'].map( tg_mean )
```

[18]:
```
# tgdf['F3'] = tgdf.groupby('F2')['Y'].transform('mean')
```

[19]:
```
tgdf
```

[19]:

	F1	F2	Y	F3
0	Alice	Female	20	25.0
1	Bob	Male	50	60.0
2	Charlie	Male	60	60.0
3	David	Male	80	60.0
4	Eve	Female	30	25.0
5	Frank	Male	50	60.0

2 Feature Selection

[20]:
```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=20,
                           n_informative=8, n_redundant=12, random_state=1)
print(X.shape, y.shape)
```

```
(1000, 20) (1000,)
```

2.1 Filter

[21]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
skb = SelectKBest( f_classif )
# ============ SelectKBest  parameter =================
# score_func
#          : f_regression, mutual_info_regression
#          : chi2, f_classif, mutual_info_classif
# k, percentile :
```

```
skbfit = skb.fit( X, y )
dfscores = pd.DataFrame( skbfit.scores_ , columns=['score'] )
dfscores.sort_values('score', ascending=False )
```

[21]:

```
         score
10  380.629147
15  253.048298
17  187.004679
13  153.136321
4   125.664978
14   99.579063
0    98.070020
9    92.021442
16   77.857707
19   66.274510
2    64.085935
6    63.110197
1    43.582535
8    13.212999
11    3.071803
3     2.181145
18    1.888992
7     1.086309
12    0.851270
5     0.331828
```

[22]:

```
skb = SelectKBest( f_classif, k=5 )
skbfit = skb.fit( X, y )
skb.get_support()
```

[22]:

```
array([False, False, False, False,  True, False, False, False, False,
       False,  True, False, False,  True, False,  True, False,  True,
       False, False])
```

[23]:

```
skb.transform(X).shape
```

[23]:

```
(1000, 5)
```

2.2 Wrapper

[24]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
model = LogisticRegression()
rfe = RFE( model, n_features_to_select = 8, verbose = 1 )
# ============ RFE   parameter =================
# estimator : coef_  feature_importances_       sklearn       .
# n_features_to_select :           . default           .
```

```
# step :              . default  1.

rfefit = rfe.fit( X, y )
```

```
Fitting estimator with 20 features.
Fitting estimator with 19 features.
Fitting estimator with 18 features.
Fitting estimator with 17 features.
Fitting estimator with 16 features.
Fitting estimator with 15 features.
Fitting estimator with 14 features.
Fitting estimator with 13 features.
Fitting estimator with 12 features.
Fitting estimator with 11 features.
Fitting estimator with 10 features.
Fitting estimator with 9 features.
```

[25]:
```
rfefit.get_support()
```

[25]:
```
array([ True,  True, False, False, False, False, False, False, False,
        True,  True, False, False, False,  True,  True,  True, False,
       False,  True])
```

[26]:
```
rfefit.transform(X).shape
```

[26]:
```
(1000, 8)
```

[27]:
```
from sklearn.feature_selection import RFECV
rfecv = RFECV( model, cv=5 )
rfecvfit = rfecv.fit(X, y)
```

[28]:
```
rfecvfit.get_support()
```

[28]:
```
array([ True,  True, False, False, False, False, False, False, False,
       False,  True, False, False, False,  True, False, False, False,
       False,  True])
```

[29]:
```
rfecvfit.transform(X).shape
```

[29]:
```
(1000, 5)
```

[30]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_selection import SequentialFeatureSelector
knn = KNeighborsClassifier( n_neighbors=3 )
sfs = SequentialFeatureSelector(knn, n_features_to_select=3)
# ============ SequentialFeatureSelector   parameter =================
# n_features_to_select :           .
```

```
# direction : 'forward'   'backward'
# scoring :     . None     estimator     score   .

sfsfit = sfs.fit(X, y)
```

[31]:
```
sfsfit.get_support()
```

[31]:
```
array([False, False, False, False,  True, False, False, False, False,
       False,  True, False, False, False,  True, False, False, False,
       False, False])
```

[32]:
```
sfsfit.transform(X).shape
```

[32]:
```
(1000, 3)
```

[33]:
```
! pip install Boruta
```

```
Collecting Boruta
  Downloading Boruta-0.4.3-py3-none-any.whl.metadata (8.8 kB)
Requirement already satisfied: numpy>=1.10.4 in
c:\users\admin\anaconda3\lib\site-packages (from Boruta) (1.26.4)
Requirement already satisfied: scikit-learn>=0.17.1 in
c:\users\admin\anaconda3\lib\site-packages (from Boruta) (1.5.1)
Requirement already satisfied: scipy>=0.17.0 in
c:\users\admin\anaconda3\lib\site-packages (from Boruta) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in
c:\users\admin\anaconda3\lib\site-packages (from scikit-learn>=0.17.1->Boruta)
(1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
c:\users\admin\anaconda3\lib\site-packages (from scikit-learn>=0.17.1->Boruta)
(3.5.0)
Downloading Boruta-0.4.3-py3-none-any.whl (57 kB)
Installing collected packages: Boruta
Successfully installed Boruta-0.4.3
```

[34]:
```
from boruta import BorutaPy
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier( random_state=123, max_depth=5 )
brtfs = BorutaPy( rf, n_estimators=7, max_iter=15, verbose=1,
                  random_state=123, alpha=0.01 )
# n_estimators :   iteration      estimator
# max_iter :    iteration
# alpha :

np.int = np.int64
np.float = np.float64
np.bool = np.bool_
```

```
brtfs.fit( X, y )
```

```
Iteration: 1 / 15
Iteration: 2 / 15
Iteration: 3 / 15
Iteration: 4 / 15
Iteration: 5 / 15
Iteration: 6 / 15
Iteration: 7 / 15
Iteration: 8 / 15
Iteration: 9 / 15
Iteration: 10 / 15
Iteration: 11 / 15
Iteration: 12 / 15
Iteration: 13 / 15
Iteration: 14 / 15


BorutaPy finished running.

Iteration:      15 / 15
Confirmed:      8
Tentative:      11
Rejected:       1
```

[34]:
```
BorutaPy(alpha=0.01,
         estimator=RandomForestClassifier(max_depth=5, n_estimators=7,
                                          random_state=RandomState(MT19937) at
0x1B59E7D7640),
         max_iter=15, n_estimators=7,
         random_state=RandomState(MT19937) at 0x1B59E7D7640, verbose=1)
```

[35]:
```
brtfs.support_
```

[35]:
```
array([False, False, False, False, False, False,  True,  True, False,
        True,  True, False, False,  True,  True,  True, False,  True,
       False, False])
```

[36]:
```
brtfs.transform(X).shape
```

[36]:
```
(1000, 8)
```

2.3 Embedded

[37]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
selector = SelectFromModel(estimator=RandomForestClassifier())
# ============ SelectFromModel   parameter =================
# threshold :           .
#                               .
#             'mean'(default), 'median', '1.25*mean'         .
# max_features :                .
# importance_getter : 'auto'    estimator   coef_   feature_importances_    .

selector.fit(X, y)
```

[37]:
```
SelectFromModel(estimator=RandomForestClassifier())
```

[38]:
```
selector.get_support()
```

[38]:
```
array([ True,  True, False, False, False, False, False, False, False,
        True,  True, False, False, False,  True,  True, False,  True,
       False, False])
```

[39]:
```
selector.transform(X).shape
```

[39]:
```
(1000, 7)
```

3 Under / Over Sampling

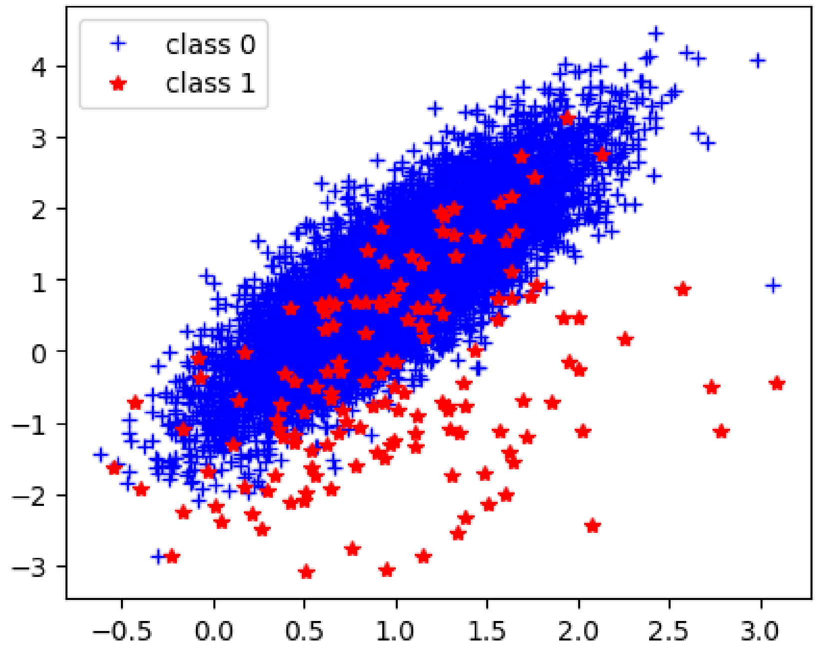
[40]:
```
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
                           n_clusters_per_class=1, weights=[0.99],␣
 ↪random_state=1)
```

[41]:
```
pd.Series(y).value_counts()
```

[41]:
```
0    9853
1     147
Name: count, dtype: int64
```

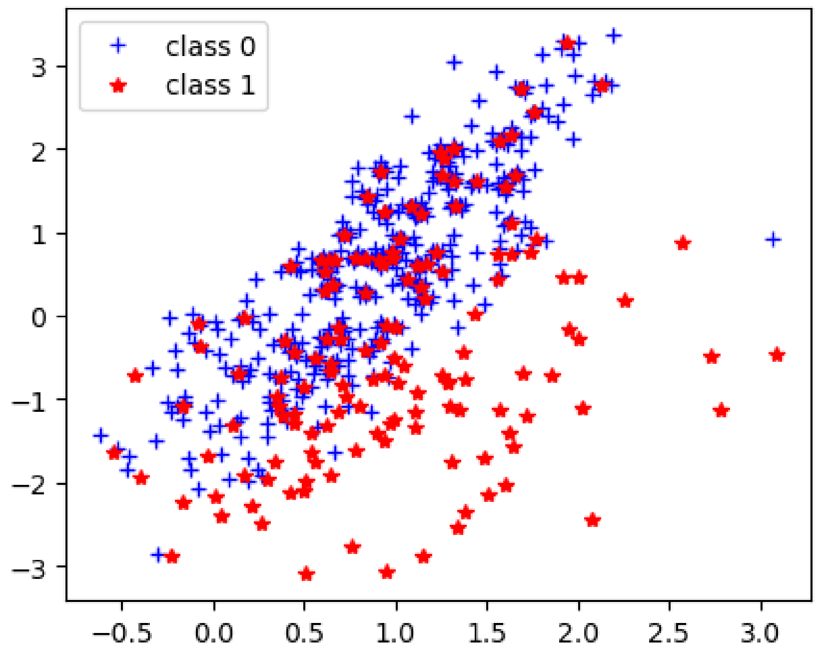
[42]:
```
plt.figure(figsize=(5, 4))
plt.plot( X[y==0, 0], X[y==0, 1], 'b+', label="class 0" )
plt.plot( X[y==1, 0], X[y==1, 1], 'r*', label="class 1" )
plt.legend()
```

[42]:
```
<matplotlib.legend.Legend at 0x1b59e06a510>
```



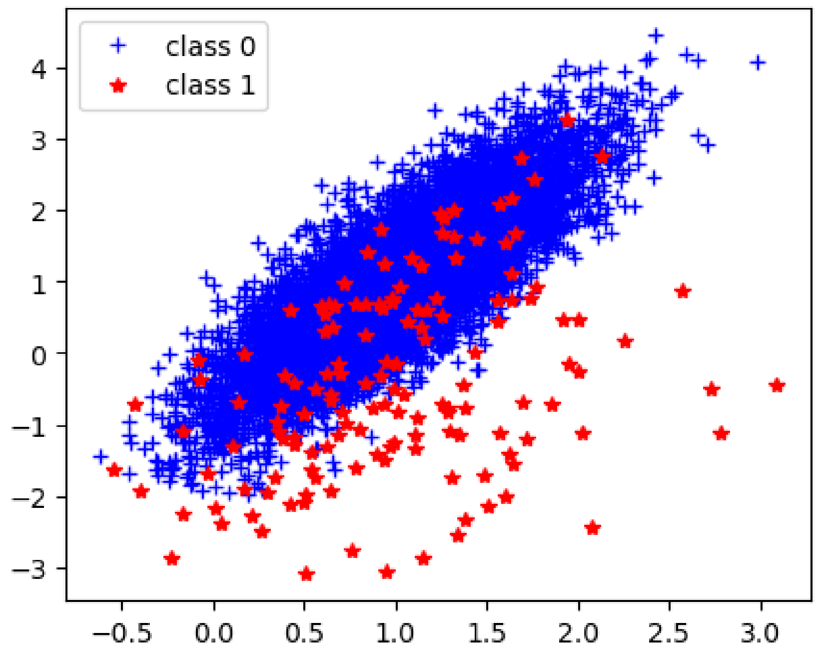
[43]:
```
from imblearn.under_sampling import CondensedNearestNeighbour
undersample1 = CondensedNearestNeighbour(n_neighbors=1)
X1, y1 = undersample1.fit_resample(X, y)
plt.figure(figsize=(5, 4))
plt.plot( X1[y1==0, 0], X1[y1==0, 1], 'b+', label="class 0" )
plt.plot( X1[y1==1, 0], X1[y1==1, 1], 'r*', label="class 1" )
plt.legend()
```

[43]: <matplotlib.legend.Legend at 0x1b59e984890>



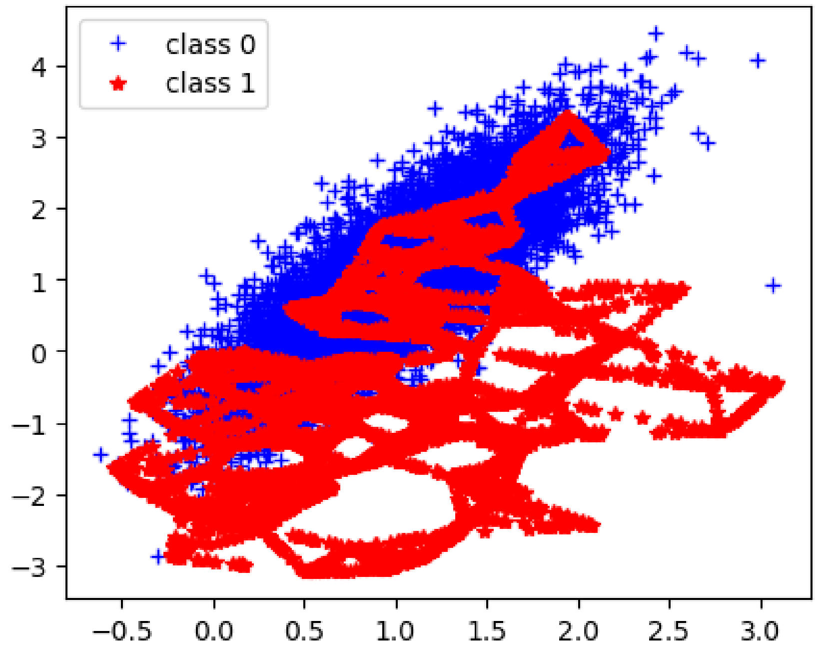
[44]:
```
from imblearn.under_sampling import TomekLinks
undersample2 = TomekLinks()
X2, y2 = undersample2.fit_resample(X, y)
plt.figure(figsize=(5, 4))
plt.plot( X2[y2==0, 0], X2[y2==0, 1], 'b+', label="class 0" )
plt.plot( X2[y2==1, 0], X2[y2==1, 1], 'r*', label="class 1" )
plt.legend()
```

[44]: <matplotlib.legend.Legend at 0x1b59fabb290>



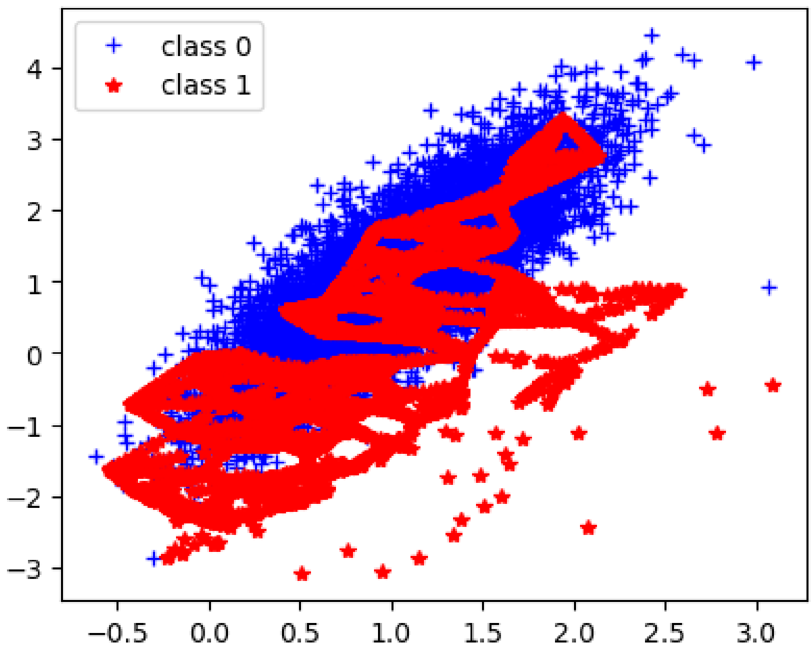
[45]:
```python
from imblearn.over_sampling import SMOTE
oversample1 = SMOTE()
OX1, Oy1 = oversample1.fit_resample(X, y)
plt.figure(figsize=(5, 4))
plt.plot( OX1[Oy1==0, 0], OX1[Oy1==0, 1], 'b+', label="class 0" )
plt.plot( OX1[Oy1==1, 0], OX1[Oy1==1, 1], 'r*', label="class 1" )
plt.legend()
```

[45]: <matplotlib.legend.Legend at 0x1b59fb5d970>



[46]:
```
from imblearn.over_sampling import ADASYN
oversample2 = ADASYN()
OX2, Oy2 = oversample2.fit_resample(X, y)
plt.figure(figsize=(5, 4))
plt.plot( OX2[Oy2==0, 0], OX2[Oy2==0, 1], 'b+', label="class 0" )
plt.plot( OX2[Oy2==1, 0], OX2[Oy2==1, 1], 'r*', label="class 1" )
plt.legend()
```

[46]:
```
<matplotlib.legend.Legend at 0x1b59fc00a10>
```


class 0
class 1
4
3
2
1
0
−1
−2
−3
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0