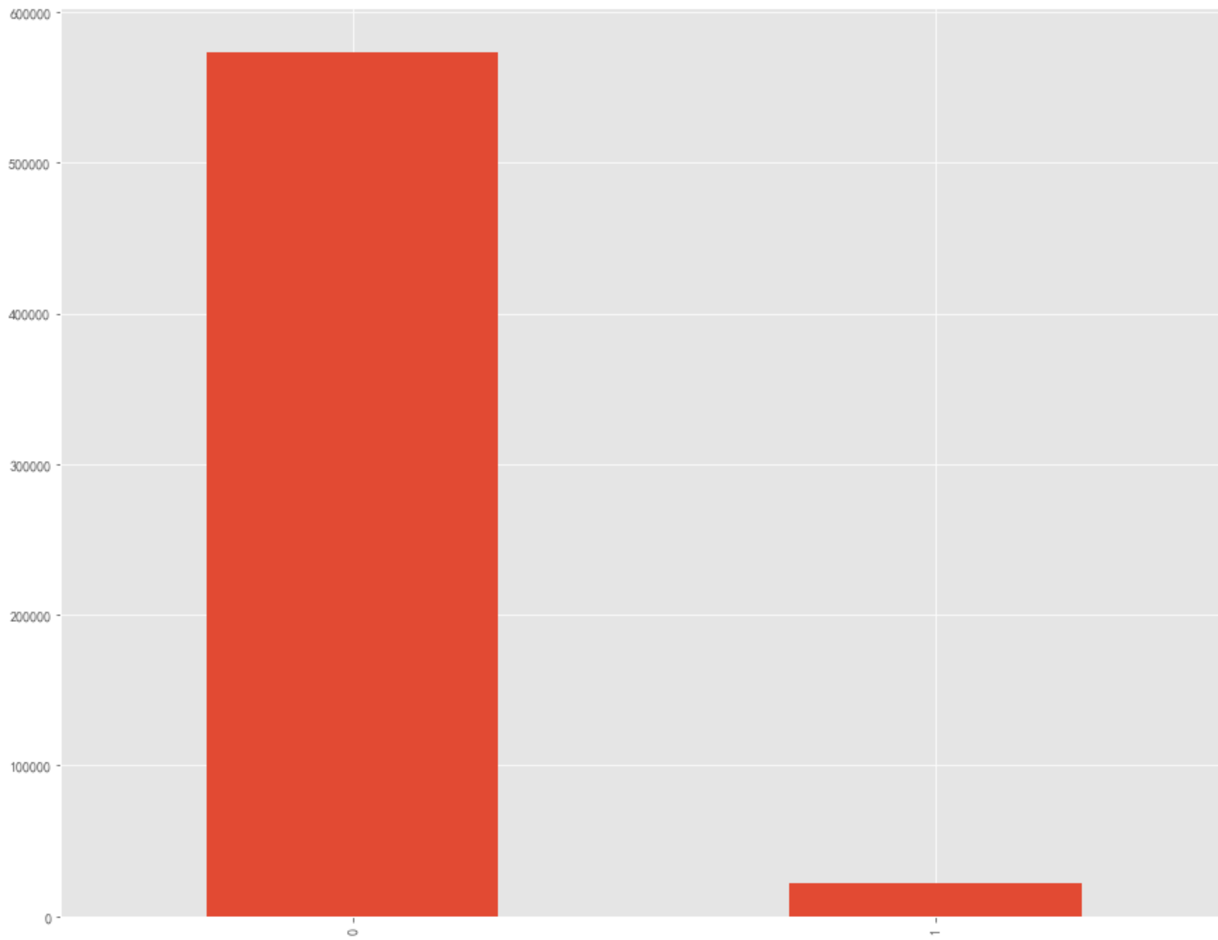
In [1]:
```
import numpy as np
import pandas as pd

from collections import Counter

import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (15,12)
plt.style.use('ggplot')

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False #用来正常显示负号

from sklearn.metrics import roc_curve, roc_auc_score
```

In [2]:
```
train = pd.read_csv('data/train.csv')

target_count = train.target.value_counts()
print('不发起索赔:', target_count[0])
print('发起索赔:', target_count[1])
print('比例:', round(target_count[0] / target_count[1], 2), ': 1')

target_count.plot(kind='bar');
```

```
不发起索赔: 573518
发起索赔: 21694
比例: 26.44 : 1
```



In [3]:
```
# 直接run
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# 移除id和target
features = train.columns[2:]

X = train[features]
y = train['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra

model = LogisticRegression(solver = 'liblinear')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 96.34%
```

In [4]:
```
model.fit(X_train[['ps_calc_02']], y_train)
y_pred = model.predict(X_test[['ps_calc_02']])

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

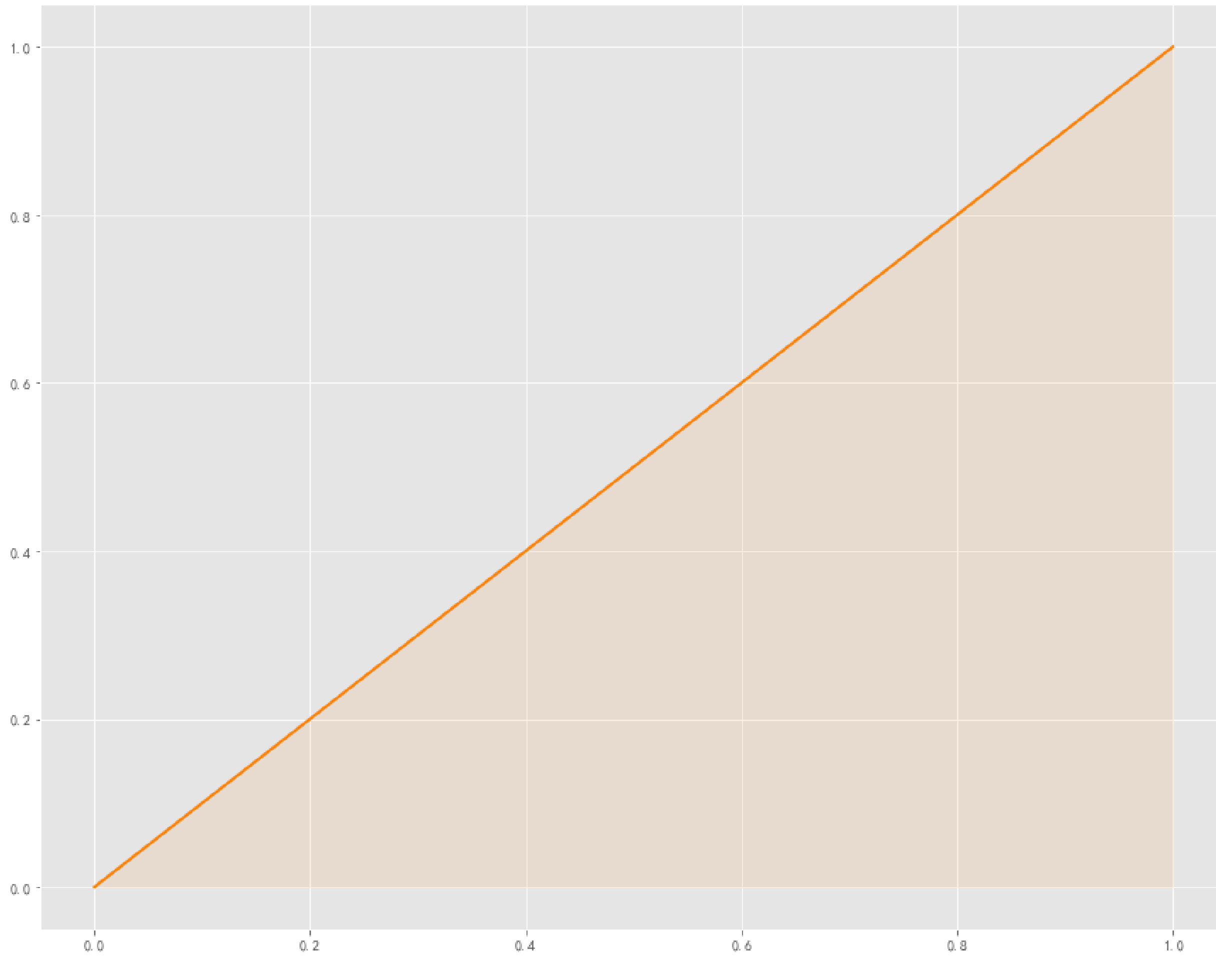
```
Accuracy: 96.34%
```

In [5]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98    114686
           1       0.00      0.00      0.00      4357

    accuracy                           0.96    119043
   macro avg       0.48      0.50      0.49    119043
weighted avg       0.93      0.96      0.95    119043
```

In [6]:
```
aucroc = roc_auc_score(y_test, y_pred)
fpr, tpr, t = roc_curve(y_test, y_pred)
fig, ax = plt.subplots(nrows=1,ncols=1, sharey=True)

ax.plot([0]+fpr.tolist(), [0]+tpr.tolist(), lw = 2, color = '#fe8006')
ax.fill_between([0]+fpr.tolist(), [0]+tpr.tolist(), color = '#fe8006', alph
```

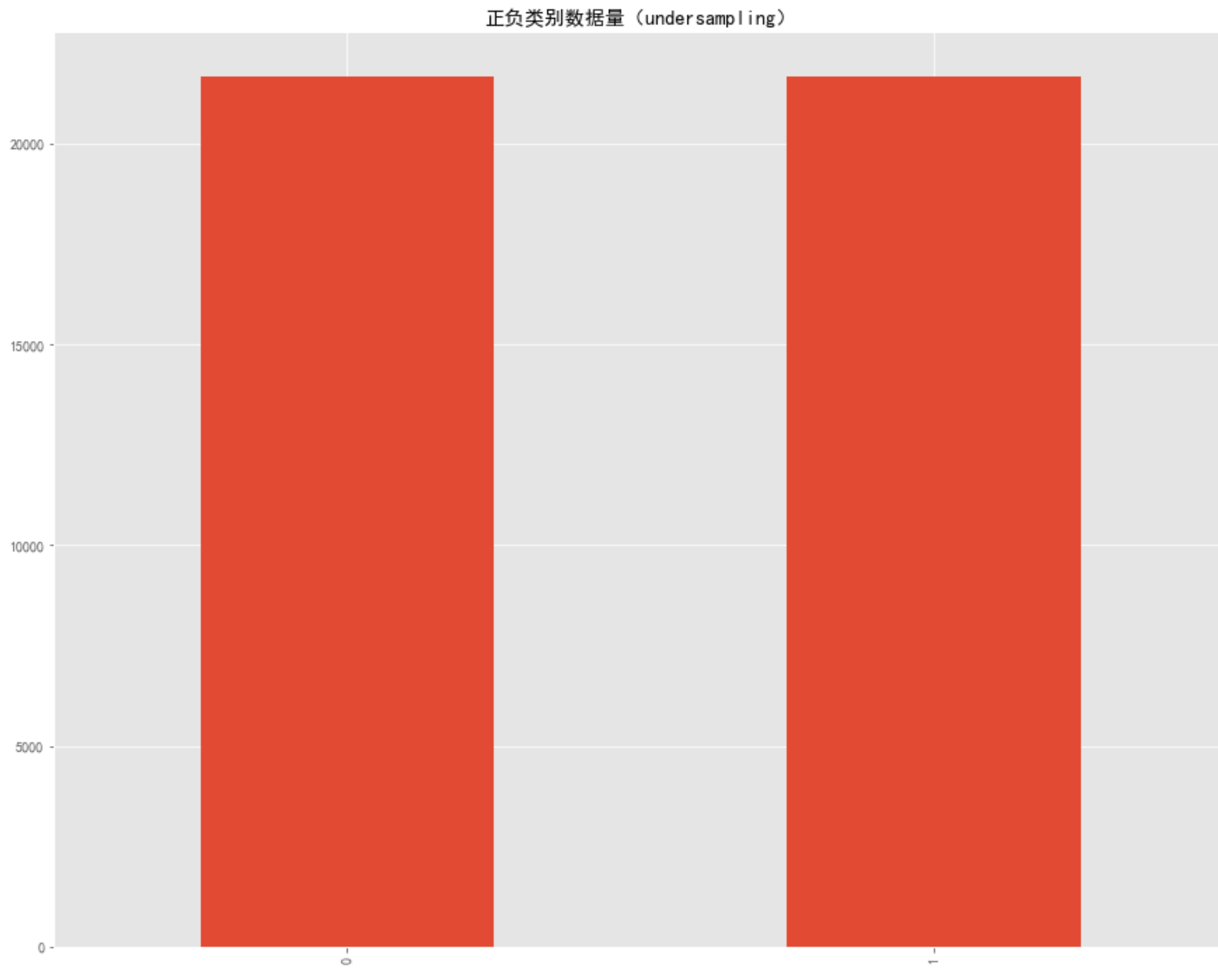
Out[6]:
```
<matplotlib.collections.PolyCollection at 0x2393c2301c8>
```



In [7]:
```
# Class count
count_class_0, count_class_1 = train.target.value_counts()

# Divide by class
df_class_0 = train[train['target'] == 0]
df_class_1 = train[train['target'] == 1]
```

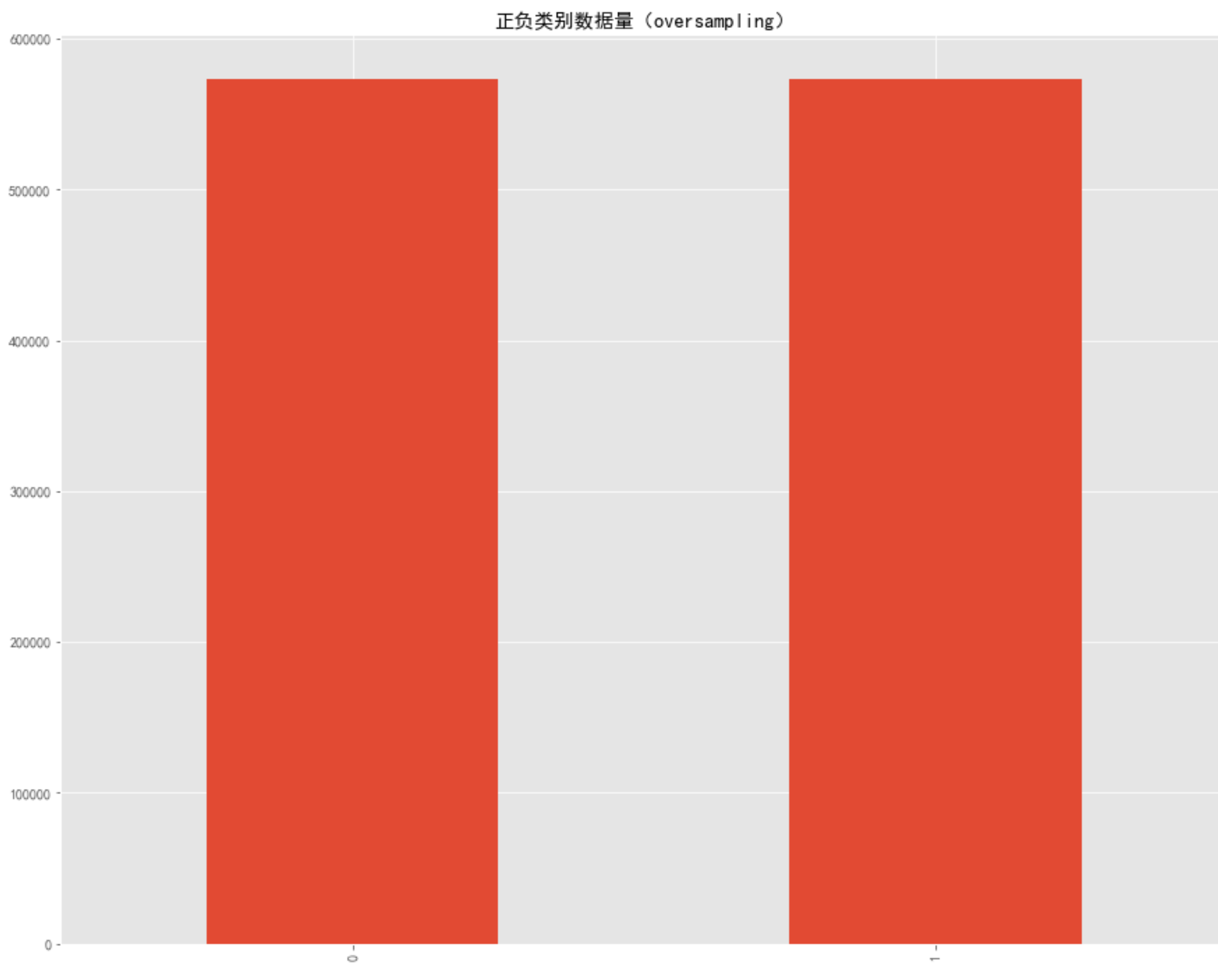
In [8]:
```
df_class_0_undersampling = df_class_0.sample(count_class_1)
df_undersampling = pd.concat([df_class_0_undersampling, df_class_1], axis=0

print('Random under-sampling:')
print(df_undersampling.target.value_counts())

df_undersampling.target.value_counts().plot(kind='bar', title='正负类别数据量
```

```
Random under-sampling:
0    21694
1    21694
Name: target, dtype: int64
```



In [9]:
```
df_class_1_oversampling = df_class_1.sample(count_class_0, replace=True)
df_oversampling = pd.concat([df_class_0, df_class_1_oversampling], axis=0)

print('Random over-sampling:')
print(df_oversampling.target.value_counts())

df_oversampling.target.value_counts().plot(kind='bar', title='正负类别数据量 (
```

```
Random over-sampling:
0    573518
1    573518
Name: target, dtype: int64
```



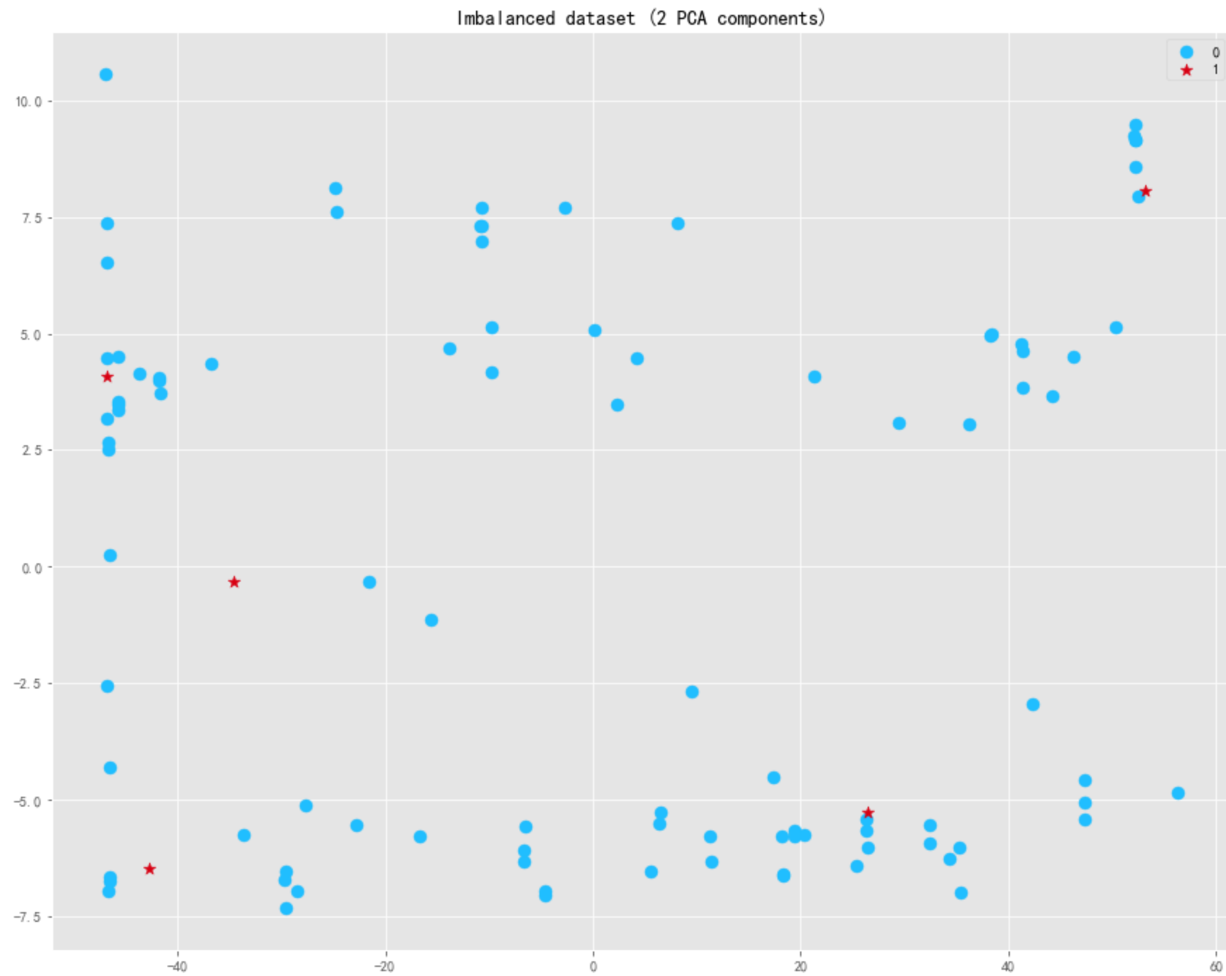
In [11]:
```
import imblearn
```

为了可视化，我们取出train里面100个样本。

In [12]:
```
df_demo = train.copy().sample(100,random_state = 0)

features = df_demo.columns[2:]

X = df_demo[features]
y = df_demo['target']
```

In [13]:
```
def plot_2d_space(X, y, label='Classes'):
    colors = ['#20beff', '#d80012']
    markers = ['o', '*']
    for l, c, m in zip(np.unique(y), colors, markers):
        plt.scatter(
            X[y==l, 0],
            X[y==l, 1],
            c=c, label=l, marker=m,
            s = 80
        )
    plt.title(label)
    plt.legend(loc='upper right')
    plt.show()
```

In [14]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X = pca.fit_transform(X)

plot_2d_space(X, y, 'Imbalanced dataset (2 PCA components)')
```



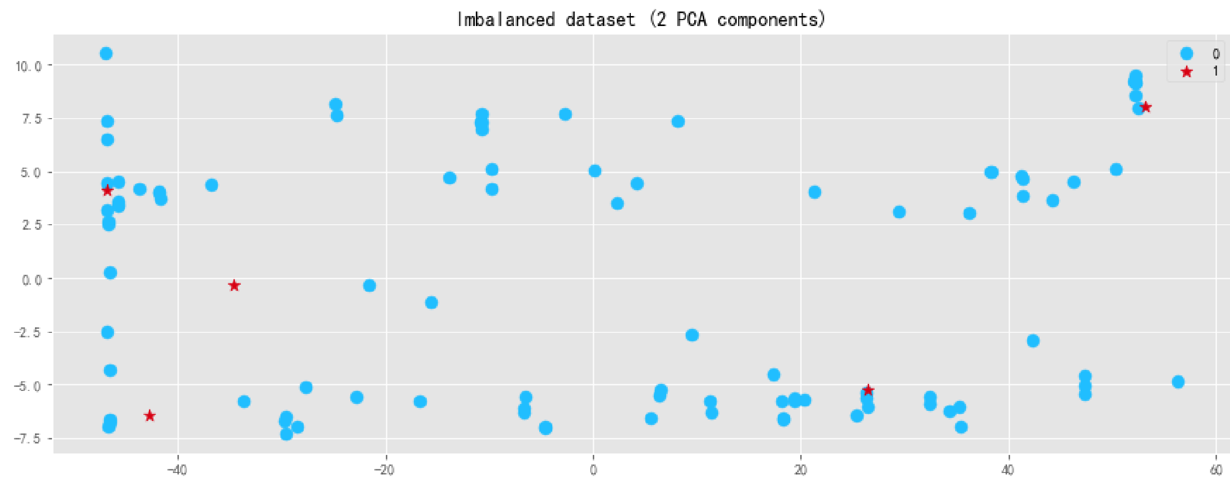
Under-sampling

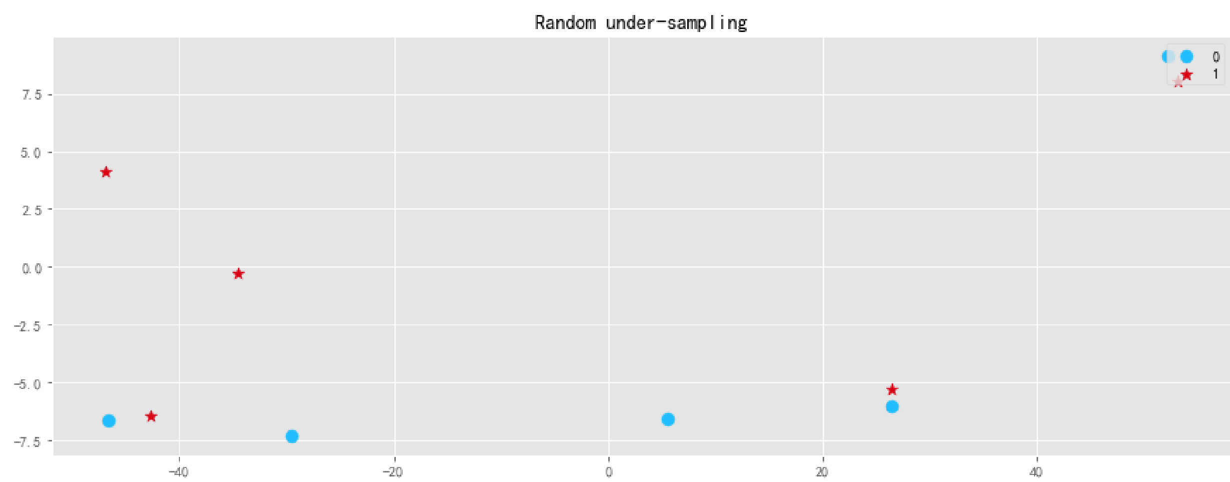
- RandomUnderSampler
- TomekLinks
- EditedNearestNeighbours
- RepeatedEditedNearestNeighbours
- AllKNN
- ClusterCentroids

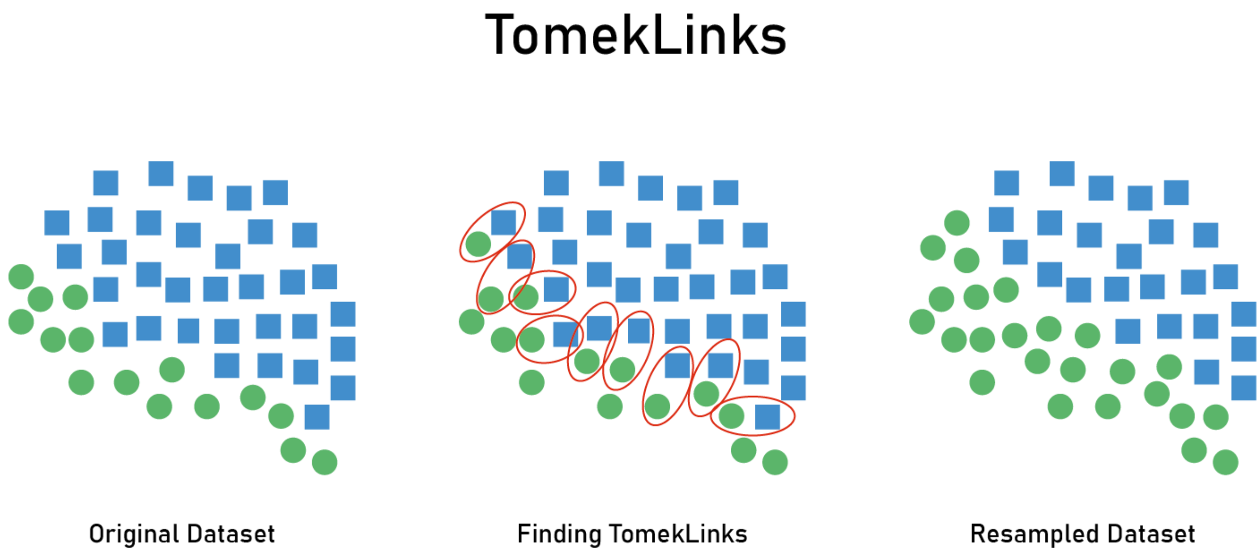
RandomUnderSampler

In [15]:
```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=2022)
X_rus, y_rus = rus.fit_resample(X, y)
print(sorted(Counter(y).items()))
print(sorted(Counter(y_rus).items()))

plt.subplot(2,1,1)
plot_2d_space(X, y, 'Imbalanced dataset (2 PCA components)')
plt.subplot(2,1,2)
plot_2d_space(X_rus, y_rus, 'Random under-sampling')
```

```
[(0, 95), (1, 5)]
[(0, 5), (1, 5)]
```






TomekLinks
Original Dataset
Finding TomekLinks
Resampled Dataset

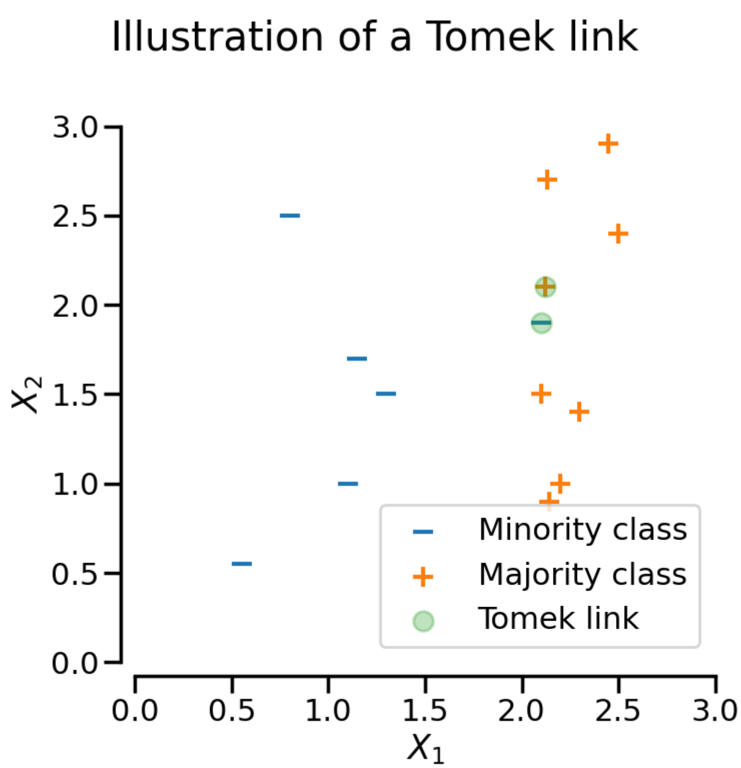
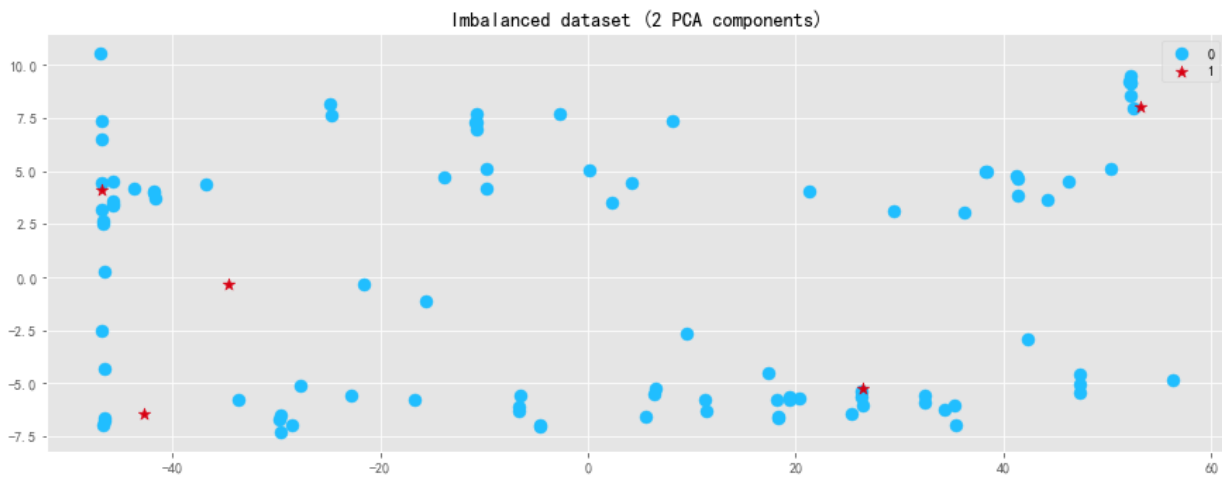
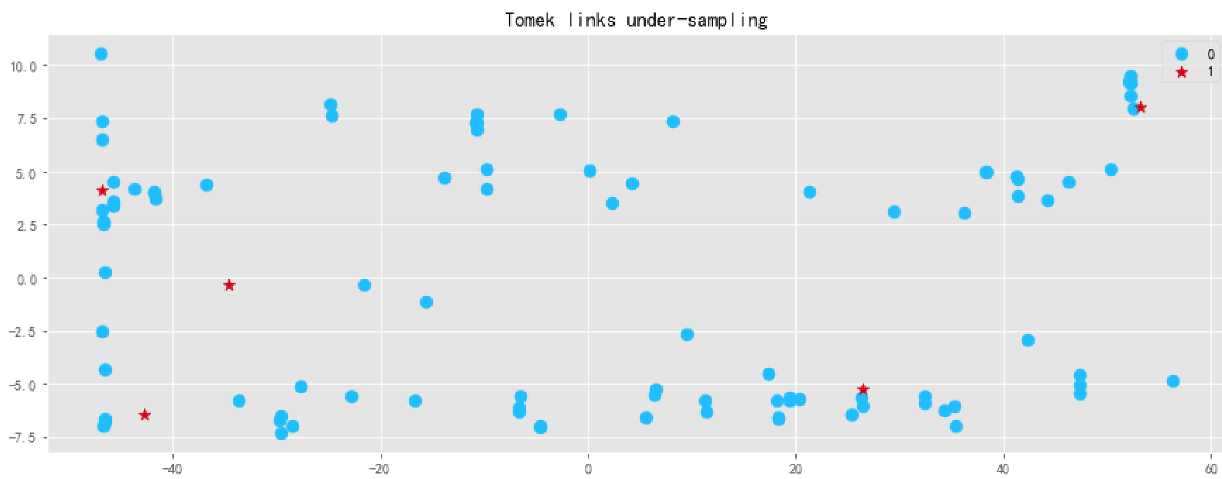


Illustration of a Tomek link
3.0
2.5
2.0
1.5
1.0
0.5
0.0
X_2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
X_1
Minority class
Majority class
Tomek link

In [16]:
```
from imblearn.under_sampling import TomekLinks


tl = TomekLinks(sampling_strategy = 'auto')
X_tl, y_tl = tl.fit_resample(X, y)
print(sorted(Counter(y).items()))
print(sorted(Counter(y_tl).items()))

plt.subplot(2,1,1)
plot_2d_space(X, y, 'Imbalanced dataset (2 PCA components)')
plt.subplot(2,1,2)
plot_2d_space(X_tl, y_tl, 'Tomek links under-sampling')
```

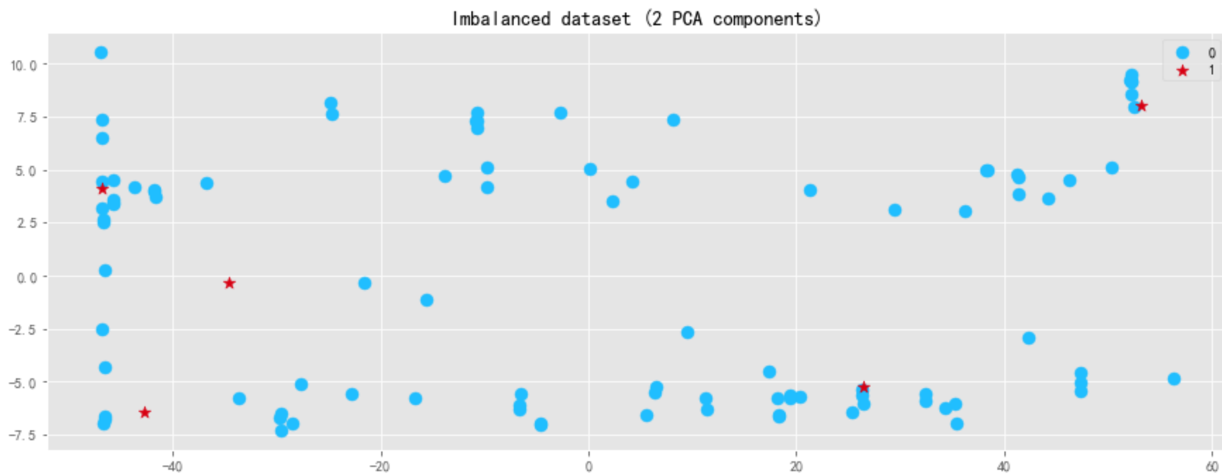
```
[(0, 95), (1, 5)]
[(0, 93), (1, 5)]
```

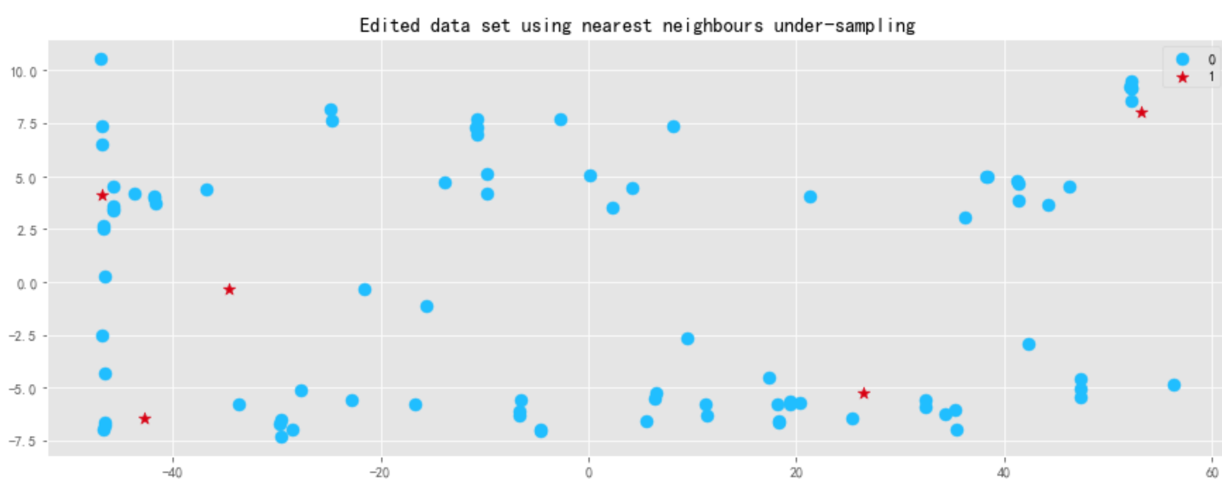




In [17]:
```
from imblearn.under_sampling import EditedNearestNeighbours


enn = EditedNearestNeighbours(sampling_strategy = 'auto',n_neighbors = 3)
X_enn, y_enn = enn.fit_resample(X, y)
print(sorted(Counter(y).items()))
print(sorted(Counter(y_enn).items()))

plt.subplot(2,1,1)
plot_2d_space(X, y, 'Imbalanced dataset (2 PCA components)')
plt.subplot(2,1,2)
plot_2d_space(X_enn, y_enn, 'Edited data set using nearest neighbours under
```

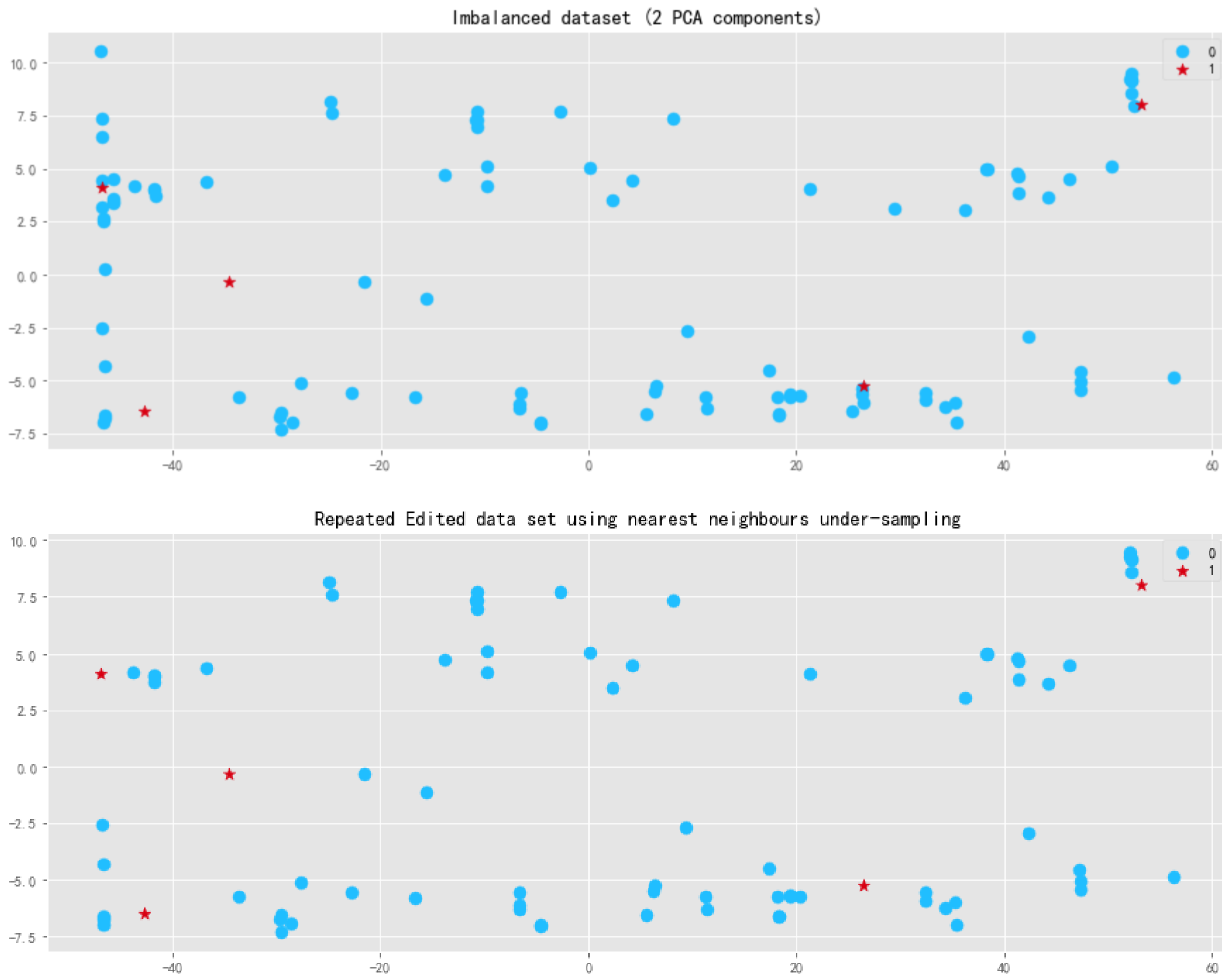
```
[(0, 95), (1, 5)]
[(0, 87), (1, 5)]
```





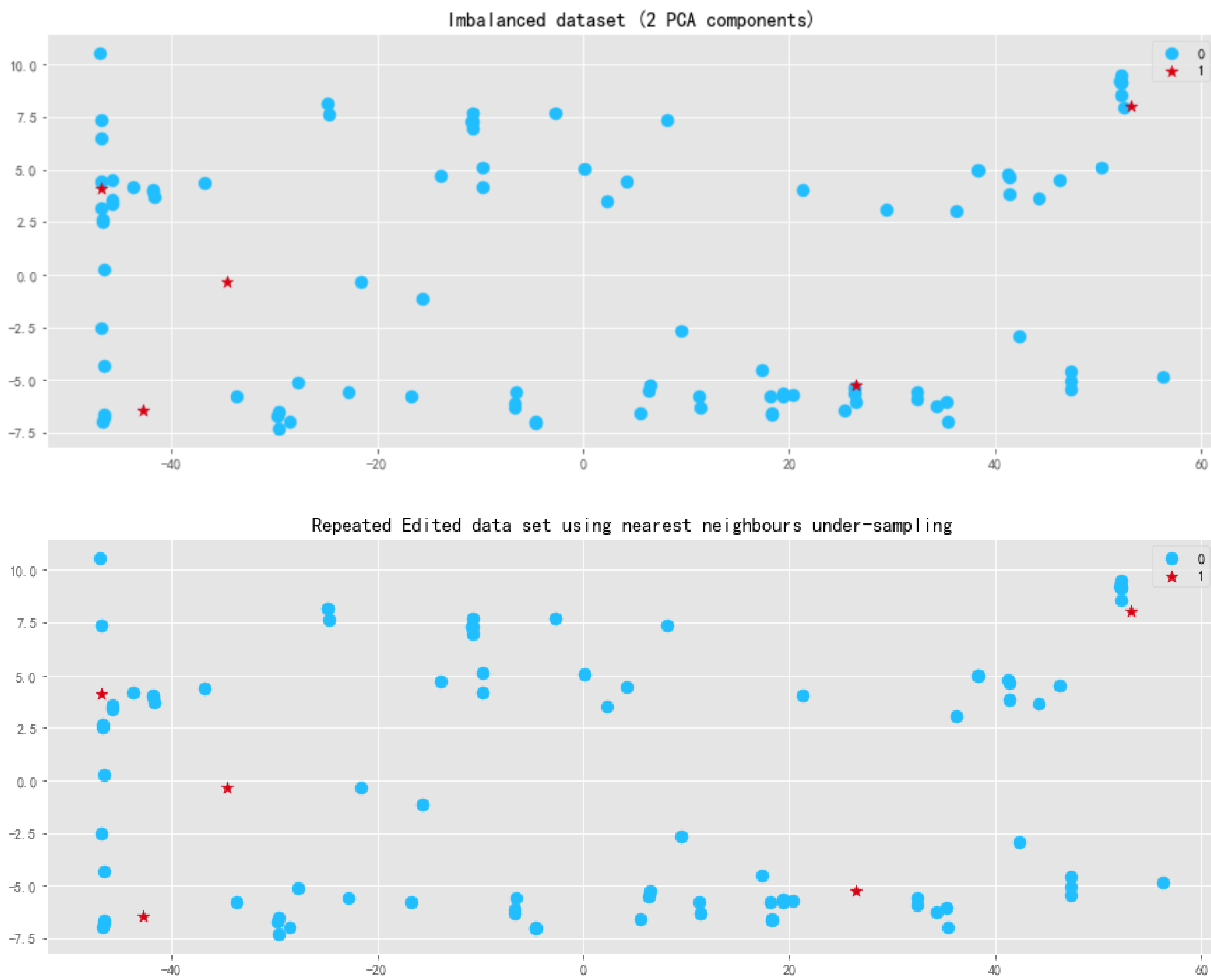
In [18]:
```
from imblearn.under_sampling import RepeatedEditedNearestNeighbours


renn = RepeatedEditedNearestNeighbours(sampling_strategy = 'auto',n_neighbo
X_renn, y_renn = renn.fit_resample(X, y)
print(sorted(Counter(y).items()))
print(sorted(Counter(y_renn).items()))

plt.subplot(2,1,1)
plot_2d_space(X, y, 'Imbalanced dataset (2 PCA components)')
plt.subplot(2,1,2)
plot_2d_space(X_renn, y_renn, 'Repeated Edited data set using nearest neigh
```

```
[(0, 95), (1, 5)]
[(0, 76), (1, 5)]
```



In [19]:
```
from imblearn.under_sampling import AllKNN

allknn = AllKNN(sampling_strategy = 'auto',n_neighbors = 3)
X_allknn, y_allknn = allknn.fit_resample(X, y)
print(sorted(Counter(y).items()))
print(sorted(Counter(y_allknn).items()))

plt.subplot(2,1,1)
plot_2d_space(X, y, 'Imbalanced dataset (2 PCA components)')
plt.subplot(2,1,2)
plot_2d_space(X_allknn, y_allknn, 'Repeated Edited data set using nearest n
```

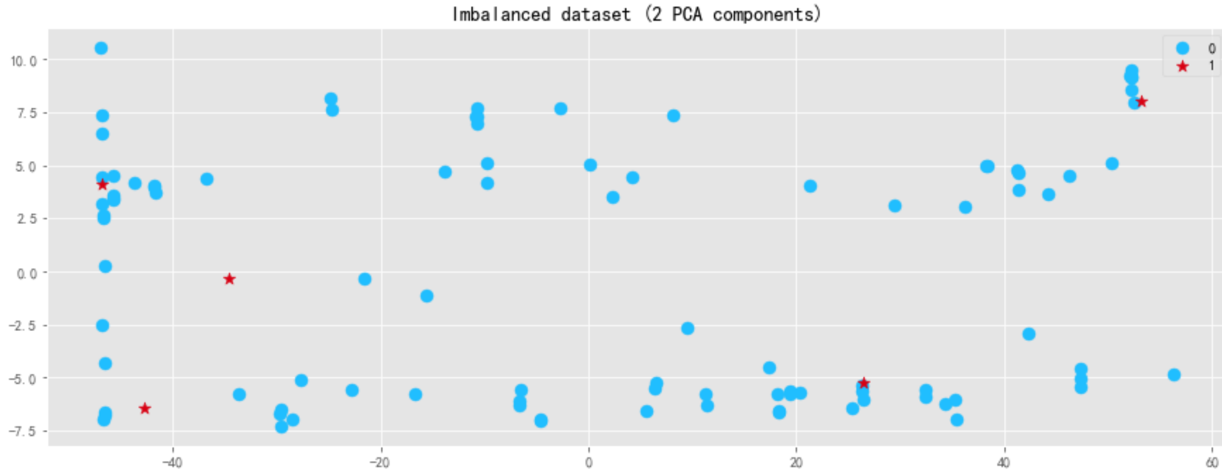
```
[(0, 95), (1, 5)]
[(0, 84), (1, 5)]
```



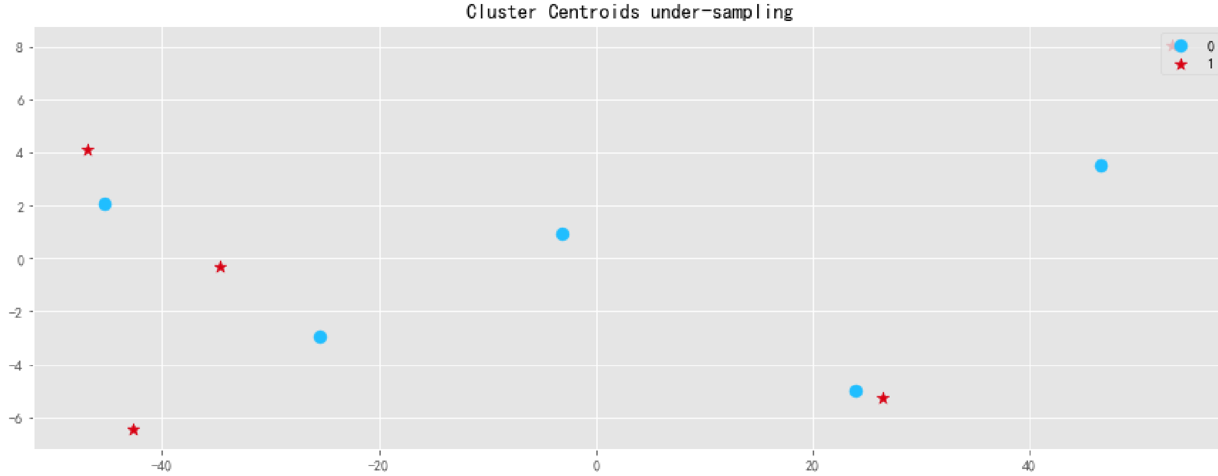
In [20]:
```
from imblearn.under_sampling import ClusterCentroids

cc = ClusterCentroids()
X_cc, y_cc = cc.fit_resample(X, y)
print(sorted(Counter(y).items()))
print(sorted(Counter(y_cc).items()))

plt.subplot(2,1,1)
plot_2d_space(X, y, 'Imbalanced dataset (2 PCA components)')
plt.subplot(2,1,2)
plot_2d_space(X_cc, y_cc, 'Cluster Centroids under-sampling')
```

```
[(0, 95), (1, 5)]
[(0, 5), (1, 5)]
```

Imbalanced dataset (2 PCA components)



Cluster Centroids under-sampling



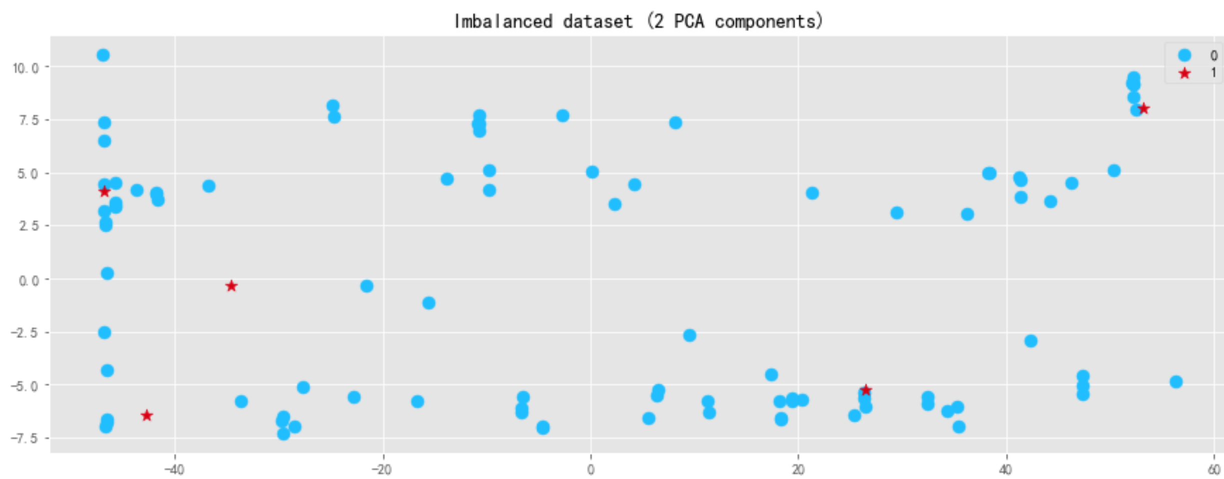
Over-sampling

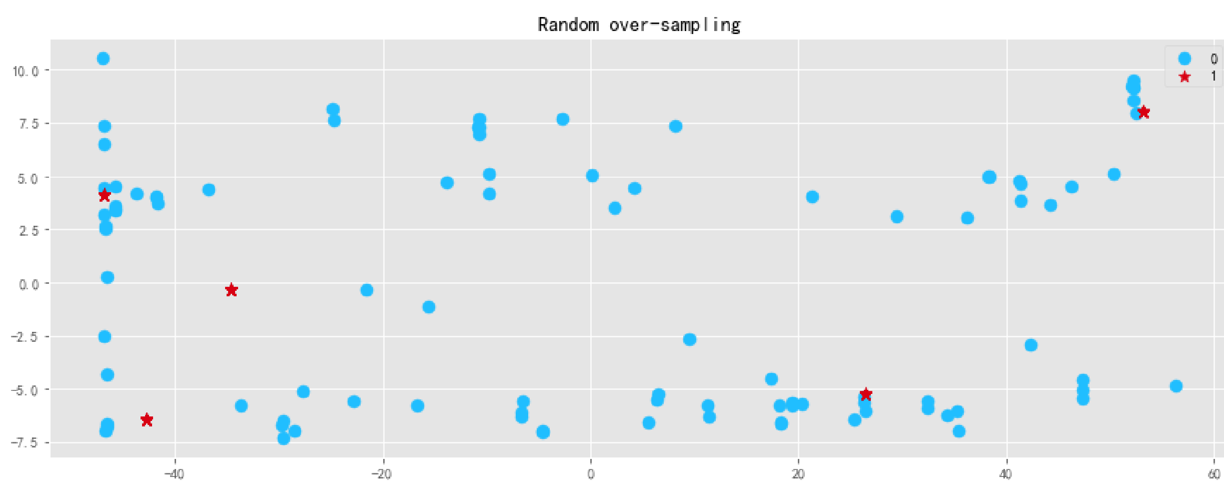
- RandomOverSampler
- SMOTE
- ADASYN

RandomOverSampler

In [21]:
```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler()
X_ros, y_ros = ros.fit_resample(X, y)

print(X_ros.shape[0] - X.shape[0], 'new random picked points')
print(sorted(Counter(y).items()))
print(sorted(Counter(y_ros).items()))

plt.subplot(2,1,1)
plot_2d_space(X, y, 'Imbalanced dataset (2 PCA components)')
plt.subplot(2,1,2)
plot_2d_space(X_ros, y_ros, 'Random over-sampling')
```

```
90 new random picked points
[(0, 95), (1, 5)]
[(0, 95), (1, 95)]
```



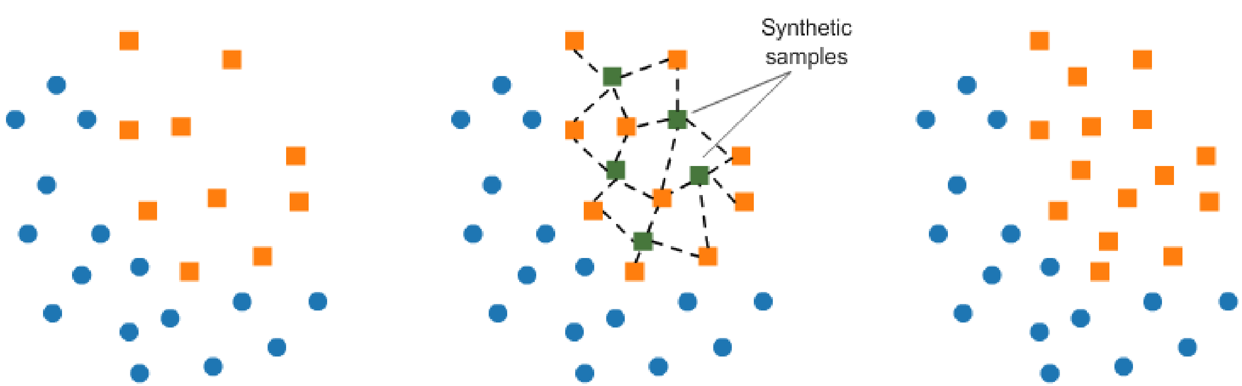


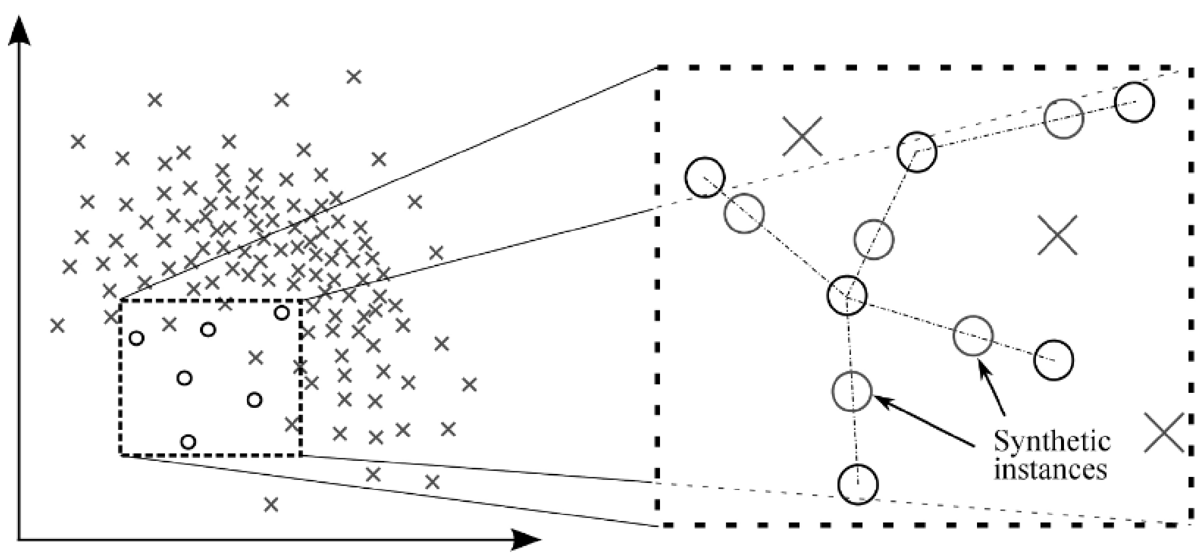
SMOTE

SMOTE（Synthetic Minority Oversampling TEchnique）由基于已经存在的少数类的合成元素组成。它从少数类中随机挑选一个点并计算该点的 k 最近邻。合成点被添加到所选点与其相邻点之间。

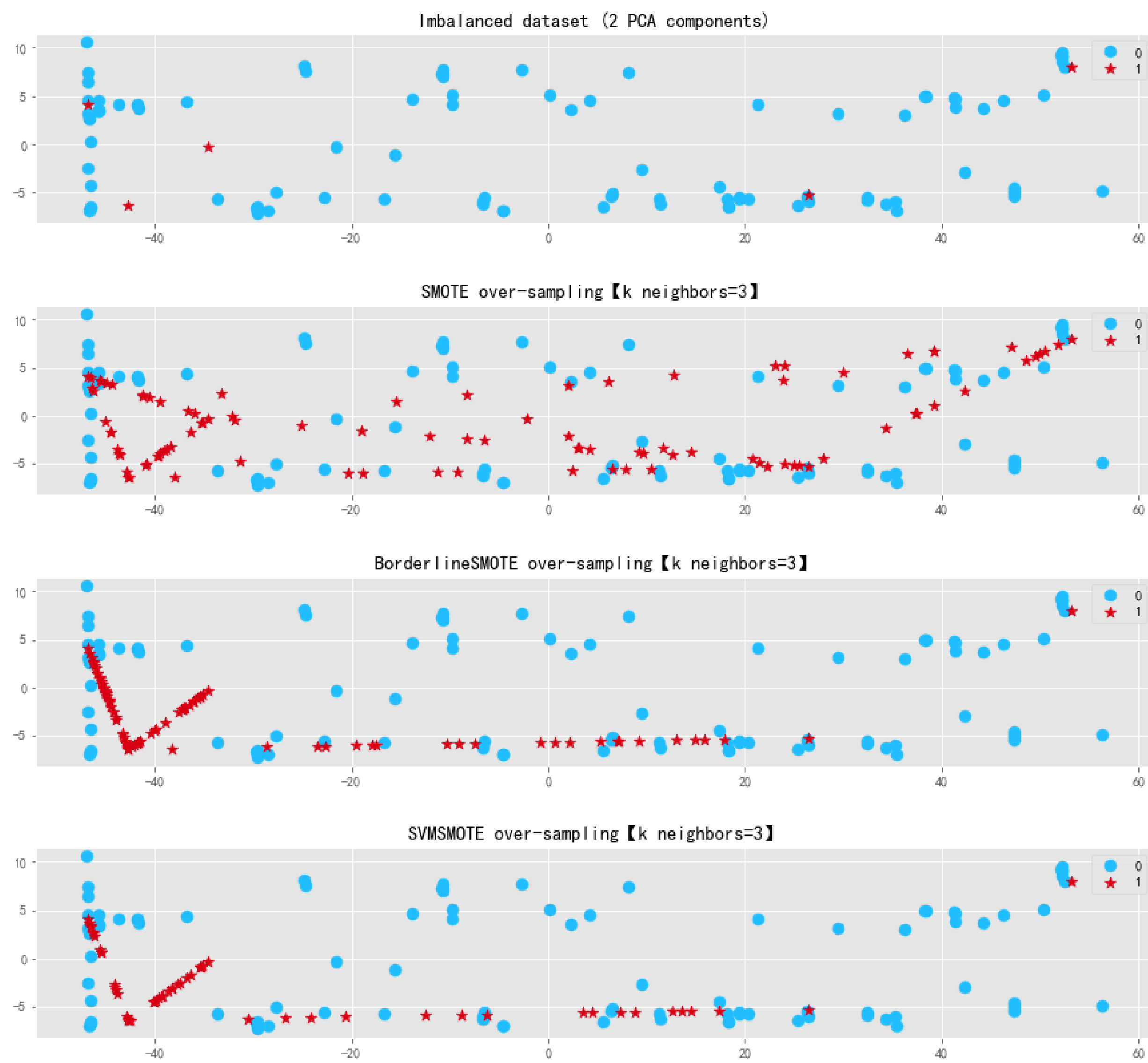
SMOTE first selects a minority class instance a at random and finds its k nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b.

The combination of SMOTE and under-sampling performs better than plain under-sampling.



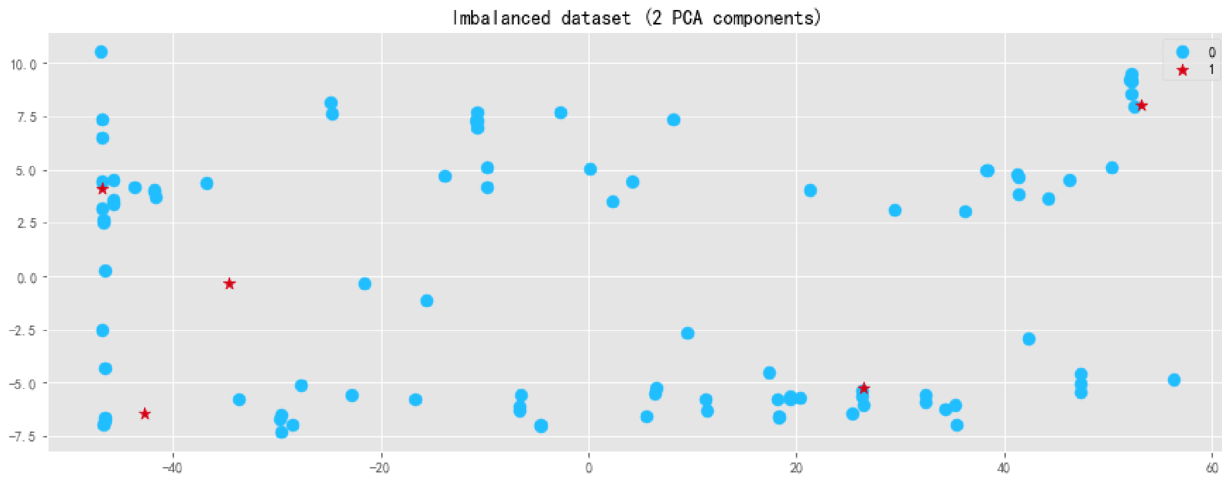


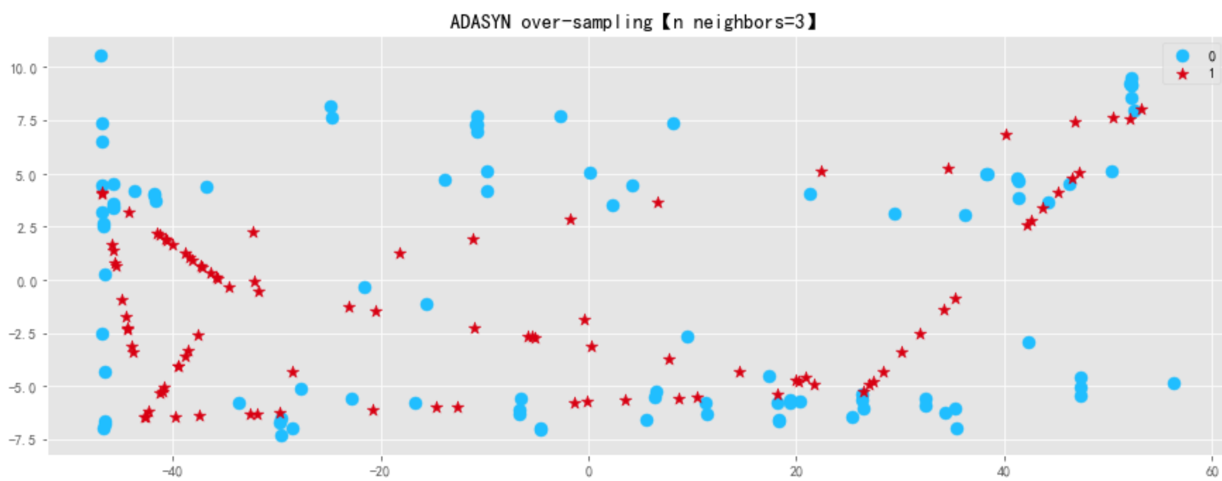
In [22]:
```
from imblearn.over_sampling import SMOTE,BorderlineSMOTE,SVMSMOTE,KMeansSMO

smote = SMOTE(k_neighbors=3)
X_sm, y_sm = smote.fit_resample(X, y)

smote_border = BorderlineSMOTE(k_neighbors=3)
X_smbl, y_smbl = smote_border.fit_resample(X, y)

smote_svm = SVMSMOTE(k_neighbors=3)
X_smsvm, y_smsvm = smote_svm.fit_resample(X, y)


plt.subplot(4,1,1)
plot_2d_space(X, y, 'Imbalanced dataset (2 PCA components)')
plt.subplot(4,1,2)
plot_2d_space(X_sm, y_sm, 'SMOTE over-sampling【k neighbors=3】')
plt.subplot(4,1,3)
plot_2d_space(X_smbl, y_smbl, 'BorderlineSMOTE over-sampling【k neighbors=3】
plt.subplot(4,1,4)
plot_2d_space(X_smsvm, y_smsvm, 'SVMSMOTE over-sampling【k neighbors=3】')
```



In [23]:
```
from imblearn.over_sampling import ADASYN

ada = ADASYN(n_neighbors = 3)
X_ada, y_ada = ada.fit_resample(X, y)

print(sorted(Counter(y).items()))
print(sorted(Counter(y_ada).items()))

plt.subplot(2,1,1)
plot_2d_space(X, y, 'Imbalanced dataset (2 PCA components)')
plt.subplot(2,1,2)
plot_2d_space(X_ada, y_ada, 'ADASYN over-sampling【n neighbors=3】')
```

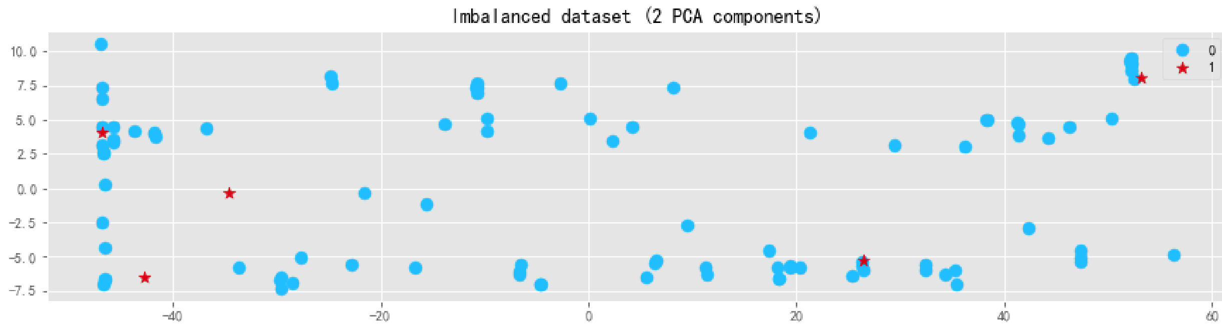
```
[(0, 95), (1, 5)]
[(0, 95), (1, 95)]
```

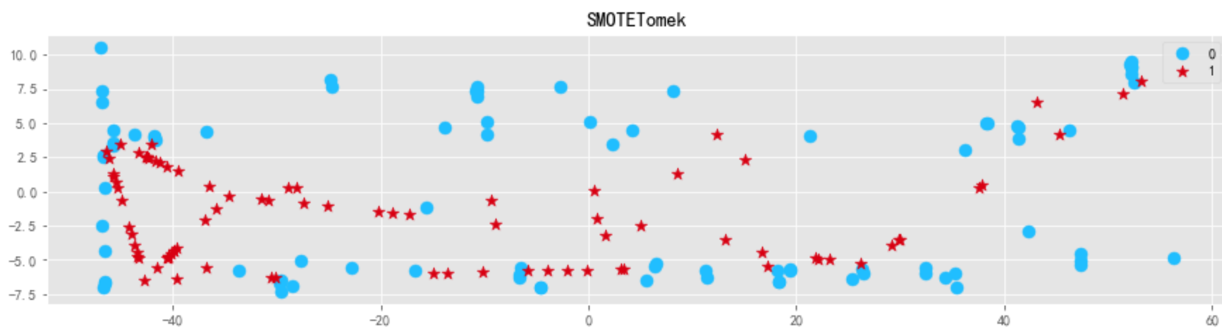


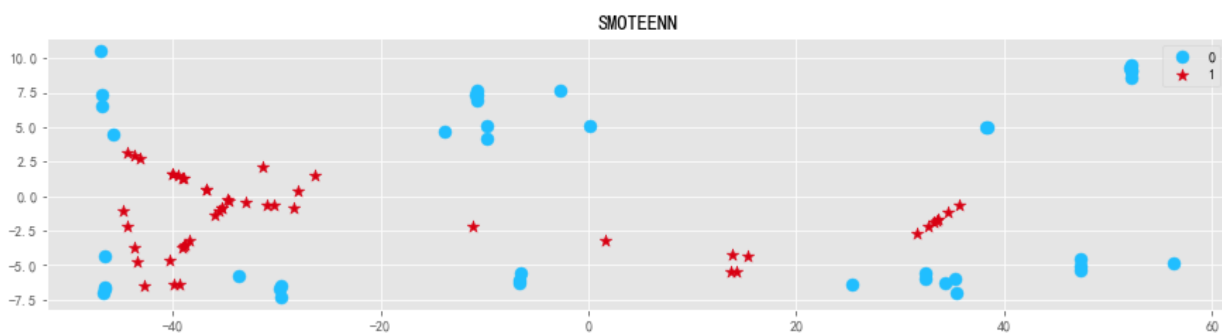


Combination of over- and under-sampling

In [24]:
```
from imblearn.combine import SMOTETomek,SMOTEENN
smote_tomek = SMOTETomek(random_state=0,smote = SMOTE(k_neighbors=3))
X_smtom, y_smtom = smote_tomek.fit_resample(X, y)

smote_enn = SMOTEENN(random_state=0,smote = SMOTE(k_neighbors=3))
X_smen, y_smen = smote_enn.fit_resample(X, y)

plt.subplot(3,1,1)
plot_2d_space(X, y, 'Imbalanced dataset (2 PCA components)')
plt.subplot(3,1,2)
plot_2d_space(X_smtom, y_smtom, 'SMOTETomek')
plt.subplot(3,1,3)
plot_2d_space(X_smen, y_smen, 'SMOTEENN')
```







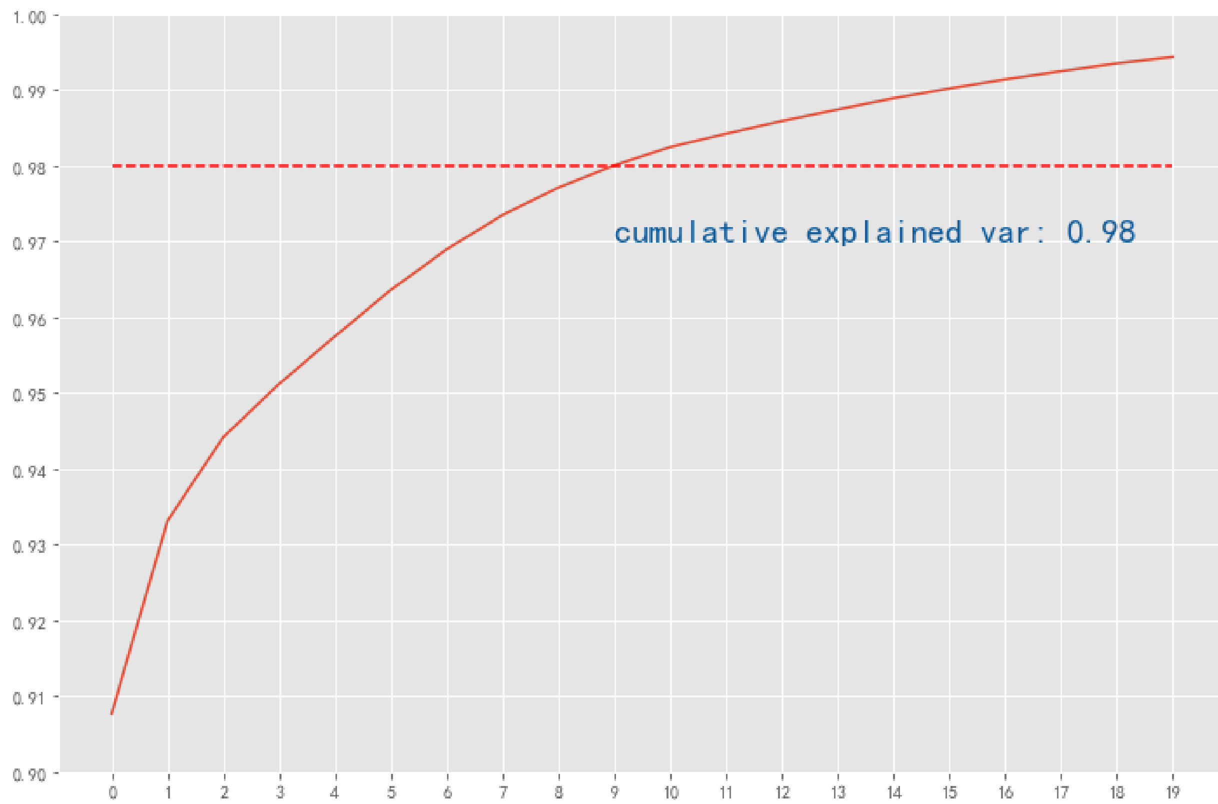
In [16]:

```
n_comp = 20
print('\nPCA执行中...')
pca = PCA(n_components=n_comp, random_state=1001)

features = train.columns[2:]
X_ = train[features]
y_ = train['target']

X_train, X_test, y_train, y_test = train_test_split(X_, y_, test_size=0.2,


X_pca = pca.fit_transform(X_train)
print('Total Explained variance: %.4f' % pca.explained_variance_ratio_.sum(
```

```
PCA执行中...
Total Explained variance: 0.9944
```

In [17]:
```
plt.figure(figsize = [12,8])

pd.Series(pca.explained_variance_ratio_).cumsum().plot()

plt.plot(range(n_comp),[0.98]*20, 'r--')
plt.xticks(ticks = range(n_comp))
plt.yticks(ticks = np.linspace(0.9,1,11))
plt.text(9,0.97,'cumulative explained var: 0.98',fontsize = 20, color = '#1
```

Out[17]:
```
Text(9, 0.97, 'cumulative explained var: 0.98')
```



所以，我们需要取出能够解释98%的variance的最少的n个principle component，那么n是多少呢？

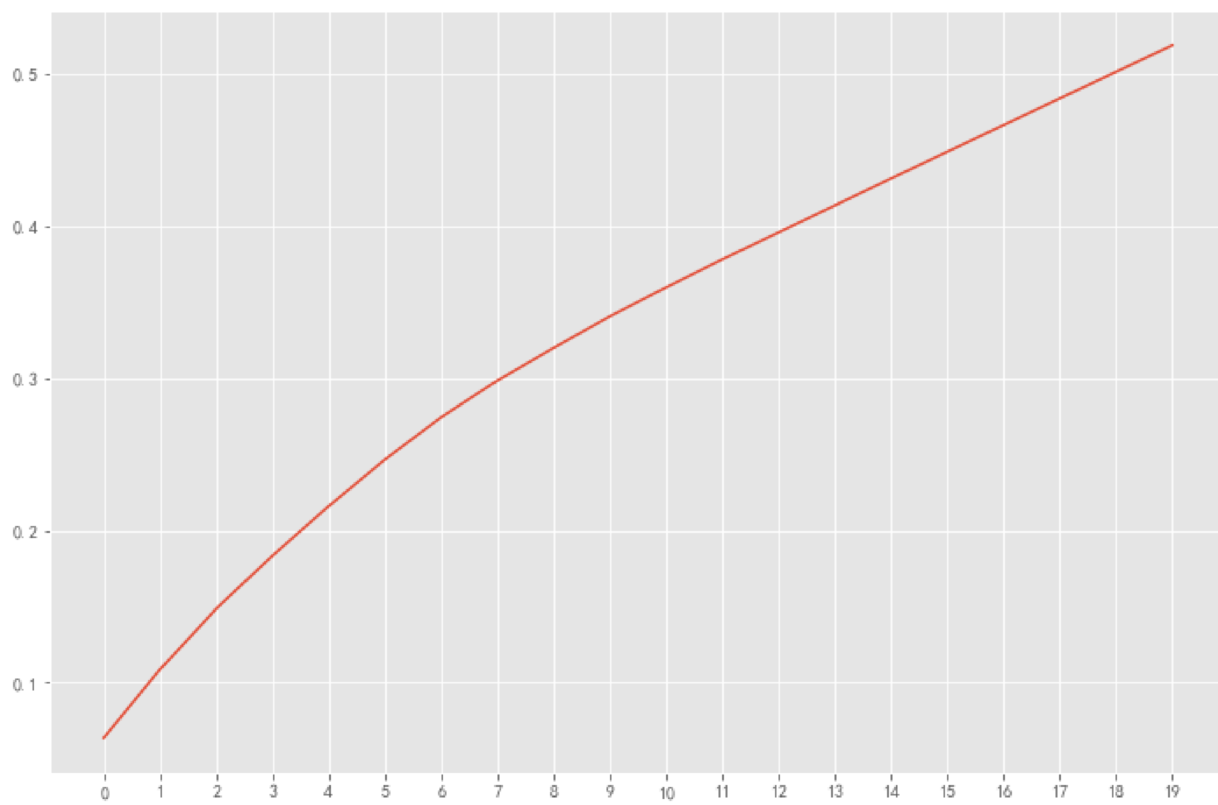
In [18]:
```
n = 9
```

In [19]:
```
pca = PCA(n_components= n, svd_solver='full', random_state=1001)
X_pca = pca.fit_transform(X_train)
```

但是，这样做真的对吗？要知道pca对于特征是需要归一化的，但是之前没有归一化，pca对于量纲的影响是非常敏感的！

In [20]:
```
n_comp = 20
print('\nPCA执行中...')
pca = PCA(n_components=n_comp, random_state=1001)

from sklearn.preprocessing import StandardScaler

features = train.columns[2:]
X_ = train[features]
y_ = train['target']
sc = StandardScaler()
X_scaled = sc.fit_transform(X_)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_, test_size


X_pca = pca.fit_transform(X_train)
print('Total Explained variance: %.4f' % pca.explained_variance_ratio_.sum(

plt.figure(figsize = [12,8])

pd.Series(pca.explained_variance_ratio_).cumsum().plot()

plt.xticks(ticks = range(n_comp))
plt.show()
```

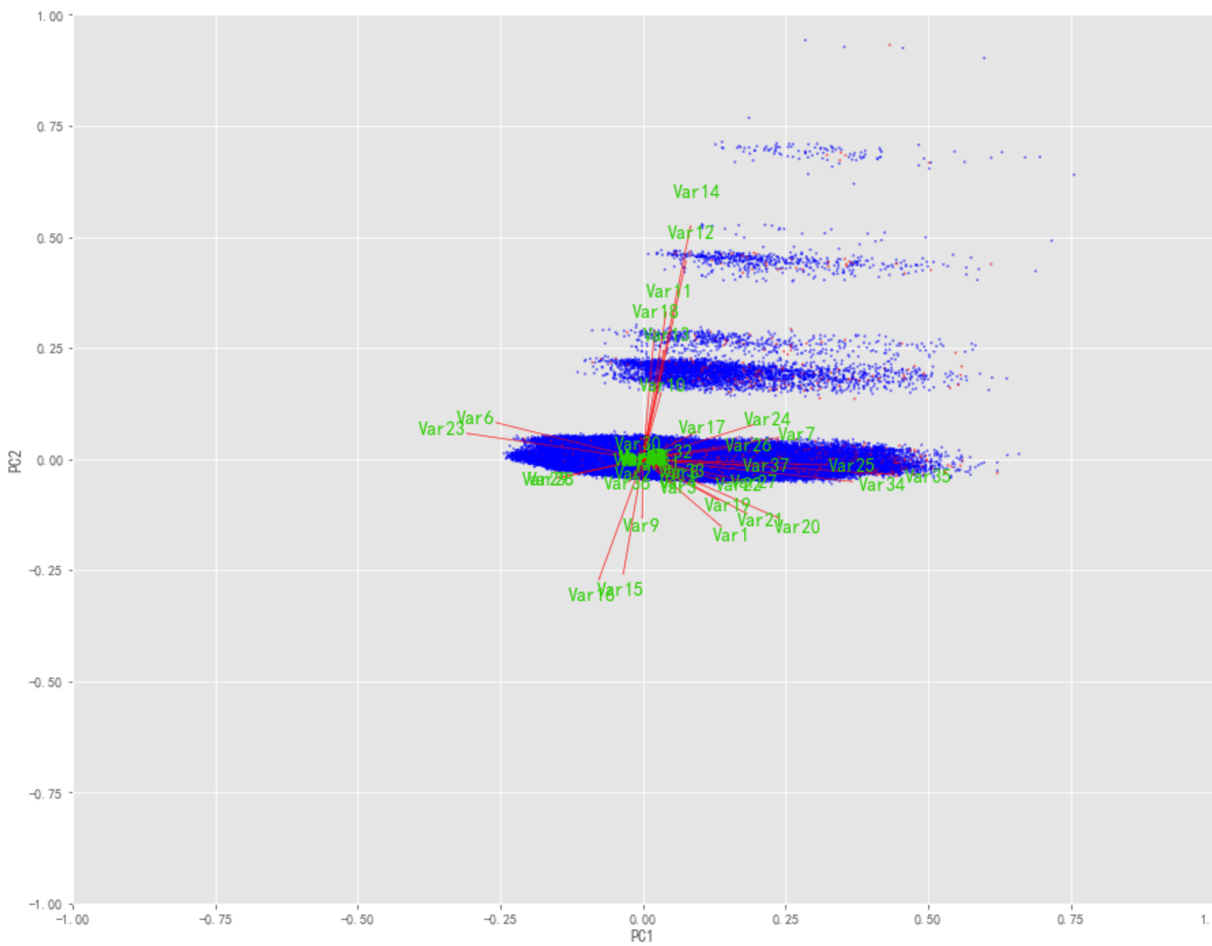
```
PCA执行中...
Total Explained variance: 0.5188
```



所以其实，我们是不存在所谓的主成分的，从这个角度去走就已经出现了错误，发现降维的过程中，没有发现明显的主成分。

In [21]:
```
pca = PCA(n_components= 2, svd_solver='full', random_state=1001)
X_pca = pca.fit_transform(X_train)
```

In [22]:
```
def myplot(score,coeff,labels=None):
    xs = score[:,0]
    ys = score[:,1]
    n = coeff.shape[0]
    scalex = 1.0/(xs.max() - xs.min())
    scaley = 1.0/(ys.max() - ys.min())
    colors = {1:'red', 0:'blue'}
    plt.scatter(xs * scalex,ys * scaley, c= y_train.apply(lambda x: colors[
    for i in range(n):
        plt.arrow(0, 0, coeff[i,0], coeff[i,1],color = 'r',alpha = 0.5)
        if labels is None:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), c
        else:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, labels[i], color
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    plt.xlabel("PC{}".format(1))
    plt.ylabel("PC{}".format(2))

myplot(X_pca[:,0:2],np.transpose(pca.components_[0:2, :]))
plt.show()
```



In [ ]:
```
from imblearn.combine import SMOTEENN  # random_state = 0
from imblearn.over_sampling import SMOTE

smote_enn = SMOTEENN(random_state=0,smote = SMOTE(k_neighbors=3))
%time X_smen, y_smen = smote_enn.fit_resample(X_pca, y_train)
```