Table of Contents

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.svm import SVC

import matplotlib.pyplot as plt
from sklearn import metrics
from numpy import *
import time

import sys
sys.path.append('../smo/')
import smo_simple
import smo_platt_no_kernel
import smo_platt_with_kernel

sys.path.append('../metrics/')
import metrics

plt.rcParams['font.family']=['Songti SC']
plt.rcParams['axes.unicode_minus'] = False
pd.set_option('display.float_format', lambda x: '%.5f' % x)
pd.options.display.max_rows = 200
np.set_printoptions(threshold=np.inf)
np.set_printoptions(linewidth=100, suppress=True)
```

In [2]:
```
def test_smo_simple(X,y,df):
    '''简化版的smo'''
    start = time.time()
    y_pred, prob_pre, w, b, alphas, sv_idx, sv_data, sv_label = smo
_simple.predict(X,y,max_iter=200)
    end = time.time()
    print(f"time: {end - start} s")
    dff = df.drop(df.columns[[0,1]], axis=1)
    dff.insert(0, 'y_predict', y_pred)
    dff.insert(0, 'y', y)
    dff.insert(0, 'prob_predict', prob_pre)
    recall, precision = metrics.recall_and_precision(dff,metrics.me
tricStruct(-1,1))
    ks = metrics.plot_ks_curve(dff,metrics.metricStruct(-1,1))
    print(f"recall: {recall}, \nprecision: {precision}, \nks: {ks}"
)

    return recall, precision, ks,end - start, dff, y_pred, prob_pre
, w, b, alphas, sv_idx, sv_data, sv_label
```

In [3]:
```
def test_smo_no_kernel(X,y,df):
    '''smo platt版本，用了拉格朗日乘子法做软间隔，没加核函数'''
    start = time.time()
    y_pred, prob_pre, w, b, alphas, sv_idx, sv_data, sv_label = smo
_platt_no_kernel.predict(X,y, max_iter=200)
    end = time.time()
    print(f"time: {end - start} s")
    dff = df.drop(df.columns[[0,1]], axis=1)
    dff.insert(0, 'y_predict', y_pred)
    dff.insert(0, 'y', y)
    dff.insert(0, 'prob_predict', prob_pre)
    recall, precision = metrics.recall_and_precision(dff,metrics.me
tricStruct(-1,1))
    ks = metrics.plot_ks_curve(dff,metrics.metricStruct(-1,1))
    print(f"recall: {recall}, \nprecision: {precision}, \nks: {ks}"
)

    return recall, precision, ks,end - start, dff, y_pred, prob_pre
, w, b, alphas, sv_idx, sv_data, sv_label
```

In [4]:
```
def test_smo_with_kernel(X,y,df,ktype='linear', k1=0, C=0.6, toler=
0.0001, max_iter=40):
    '''smo platt版本，用了拉格朗日乘子法做软间隔，加了核函数'''
    start = time.time()
    y_pred, prob_pre, w, b, alphas, sv_idx, sv_data, sv_label = smo
_platt_with_kernel.predict(X,y,ktype, k1, C, toler, max_iter)
    end = time.time()
    print(f"time: {end - start} s")
    dff = df.drop(df.columns[[0,1]], axis=1)
    dff.insert(0, 'y_predict', y_pred)
    dff.insert(0, 'y', y)
    dff.insert(0, 'prob_predict', prob_pre)
    recall, precision = metrics.recall_and_precision(dff,metrics.me
tricStruct(-1,1))
    ks = metrics.plot_ks_curve(dff,metrics.metricStruct(-1,1))
    print(f"recall: {recall}, \nprecision: {precision}, \nks: {ks}"
)

    return recall, precision, ks,end - start, dff, y_pred, prob_pre
, w, b, alphas, sv_idx, sv_data, sv_label
```

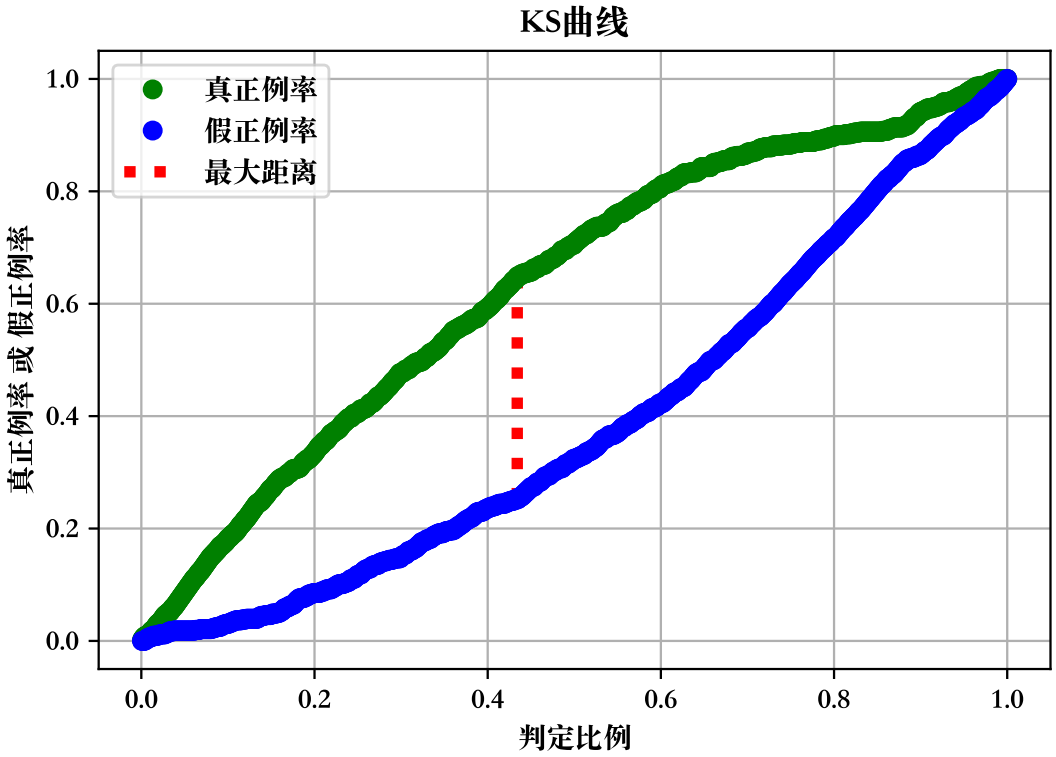
In [5]:
```
def cmp_score(dict1,type):
    score_df1 = pd.DataFrame(dict1,index=["Recall", "Precision", "KS","Time(s)"])

    # 比较得分
    score_df1.drop(["Time(s)"],axis=0).plot.bar(rot=0)
    t1 = "不同SMO算法的得分" if type==1 else "不同惩罚系数的RBF内核SMO算法得分"
    plt.title(t1)
    plt.ylim(0,1.2)
    plt.ylabel("得分")
    plt.xlabel("指标类别")

    # 比较时间
    score_df1.drop(['Recall','Precision','KS'],axis=0).plot.bar(rot=0)
    t2 = "不同SMO算法的时间(s)" if type==1 else "不同惩罚系数的RBF内核SMO算法的时间(s)"
    plt.title(t2)
    plt.ylabel("时间(s)")
    return score_df1
```

In [6]:
```
if __name__ == '__main__':
    df = pd.read_csv("../data/preprocess.csv")
    X,y = df.drop(df.columns[[0,1]], axis=1), df["Label"]
    # X,y = df[["GAGE_TOTLE_PRICE","PAYMENT_TYPE"]], df["Label"]  # 选取两个特征方便看图观察
    y = y.apply(lambda x: -1 if x==0 else x)
    t = []
    # t[0]
    t.append(test_smo_simple(X,y,df))
```

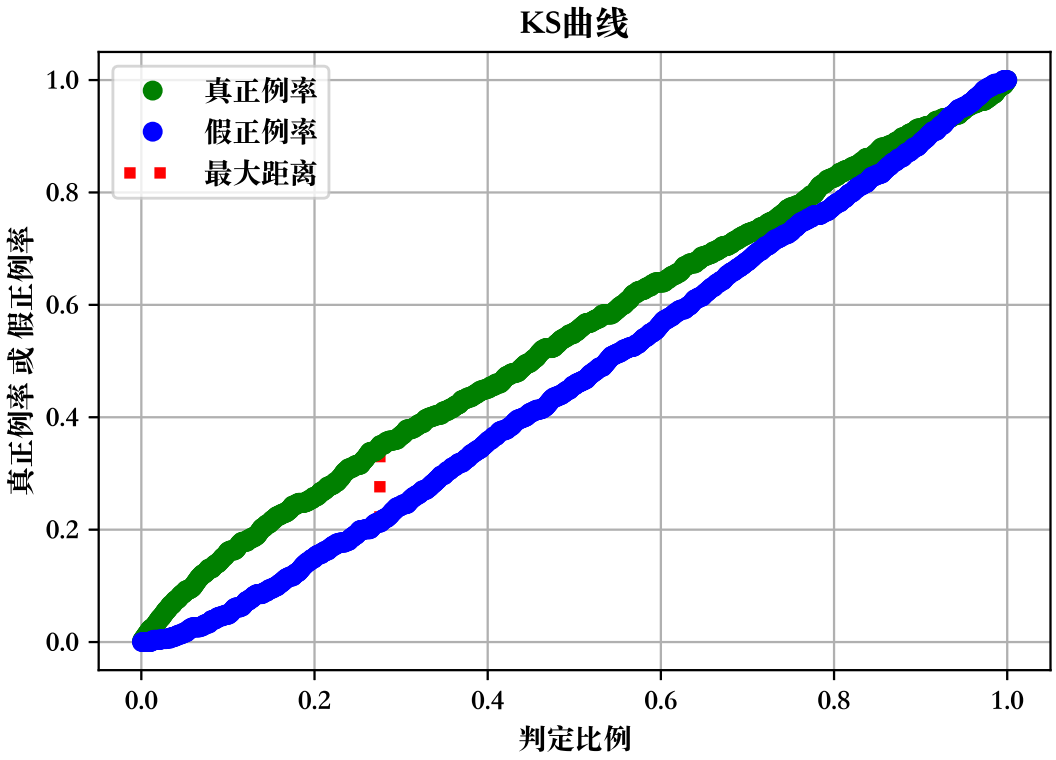
```
there are 1210 Support Vectors
time: 42.2342369556427 s
```




```
recall: 0.7307692307692307,
precision: 0.6480218281036835,
ks: 0.3972622652937614
```

In [7]:
```
    # t[1]
    t.append(test_smo_no_kernel(X,y,df))
```

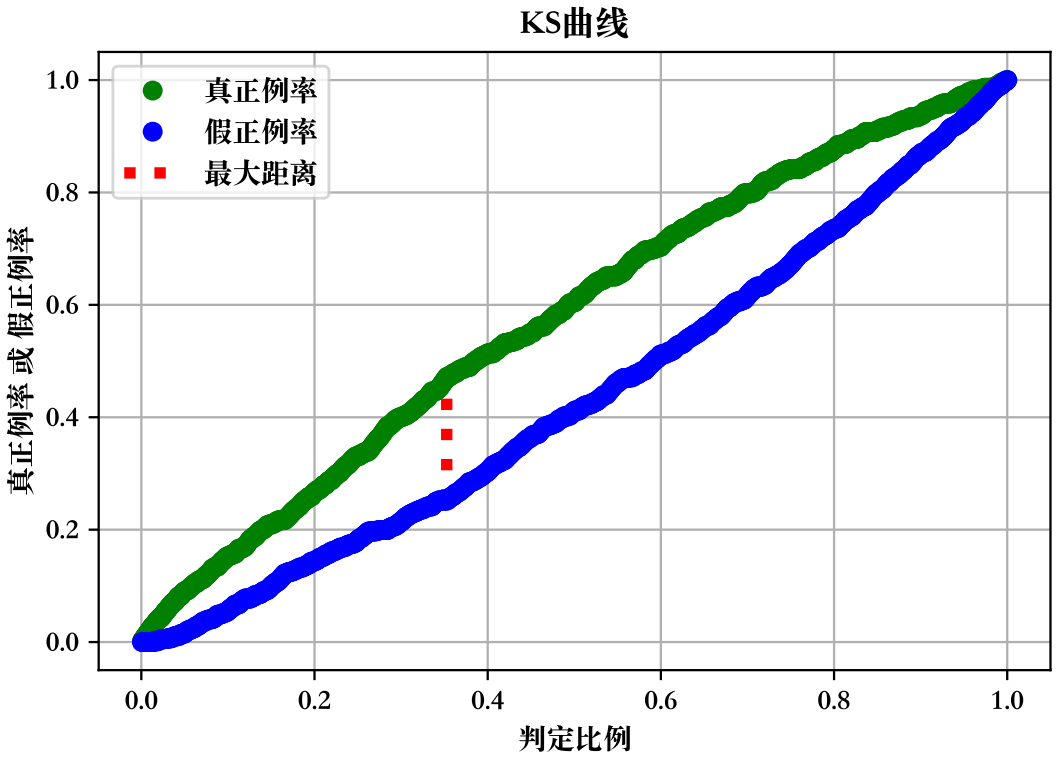
```
there are 225 Support Vectors
time: 205.69921374320984 s
```




```
recall: 0.45076923076923076,
precision: 0.5195035460992907,
ks: 0.13817080557238037
```

In [8]:
```
    # t[2]
    t.append(test_smo_with_kernel(X,y,df))
```

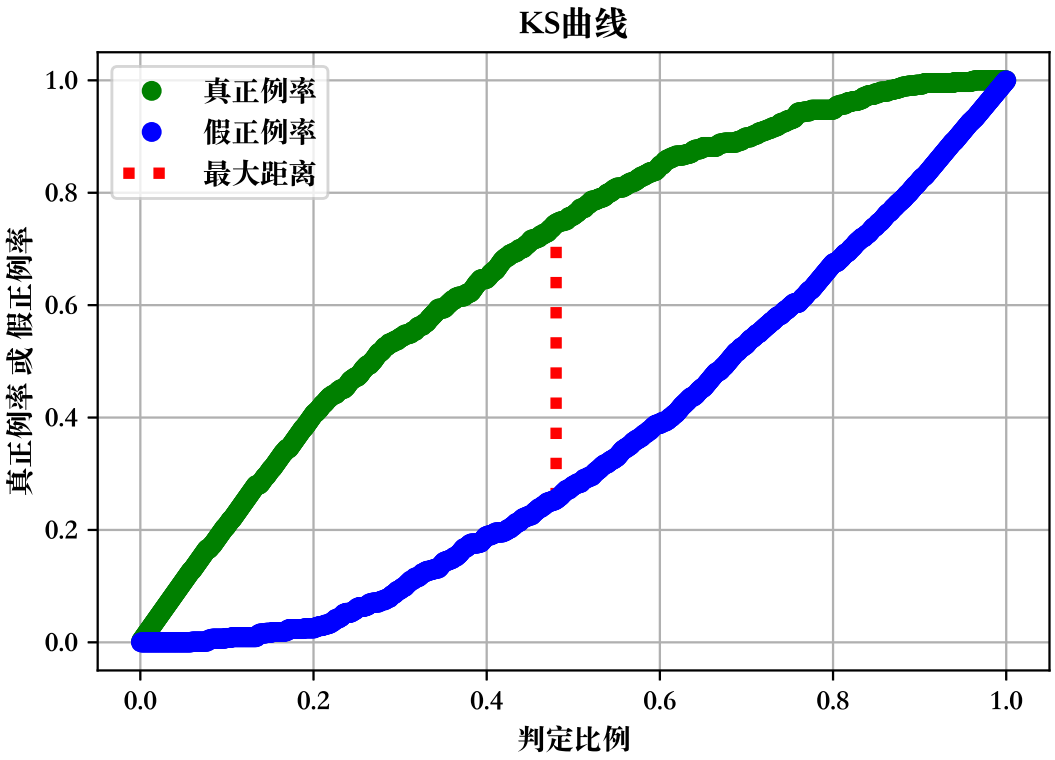
```
there are 251 Support Vectors
time: 100.15252304077148 s
```




```
recall: 0.7446153846153846,
precision: 0.5383759733036707,
ks: 0.22033918837068445
```

In [9]:
```
    # t[3]
    t.append(test_smo_with_kernel(X,y,df,"rbf",1.3,0.6,0.0001,1000)
)
```

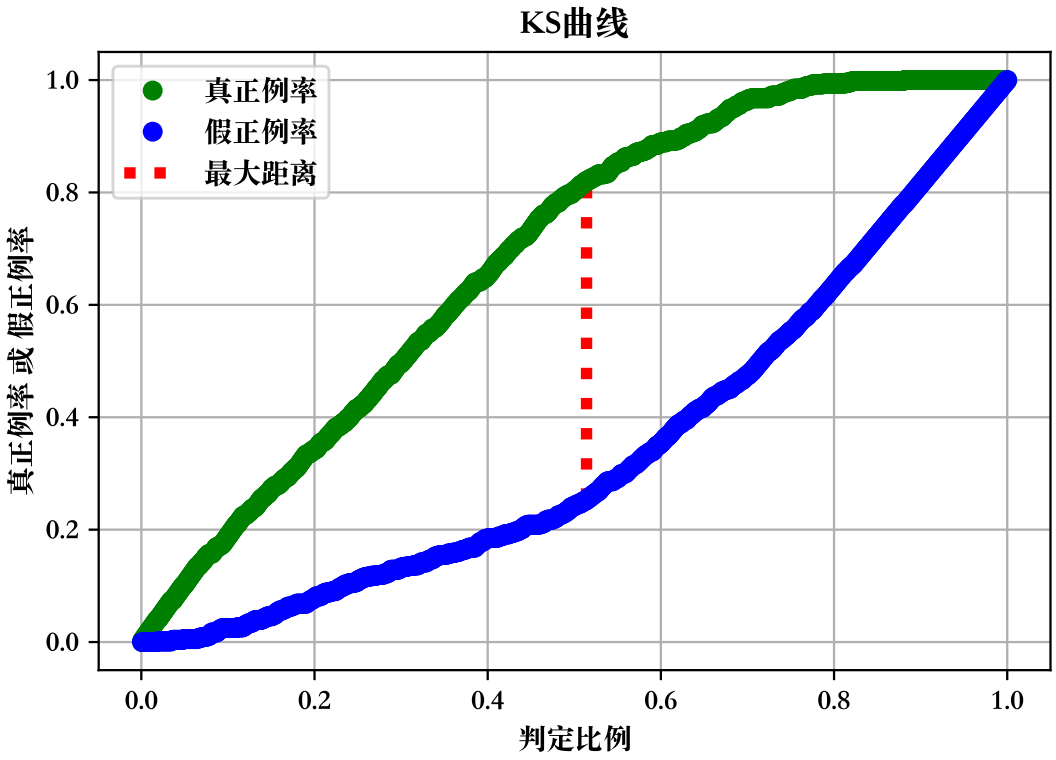
```
there are 128 Support Vectors
time: 141.7985589504242 s
```




```
recall: 0.0876923076923077,
precision: 1.0,
ks: 0.4915606703008278
```

In [10]:
```
    # t[4]
    t.append(test_smo_with_kernel(X,y,df,"rbf",1.3,6,0.0001,1000))
```

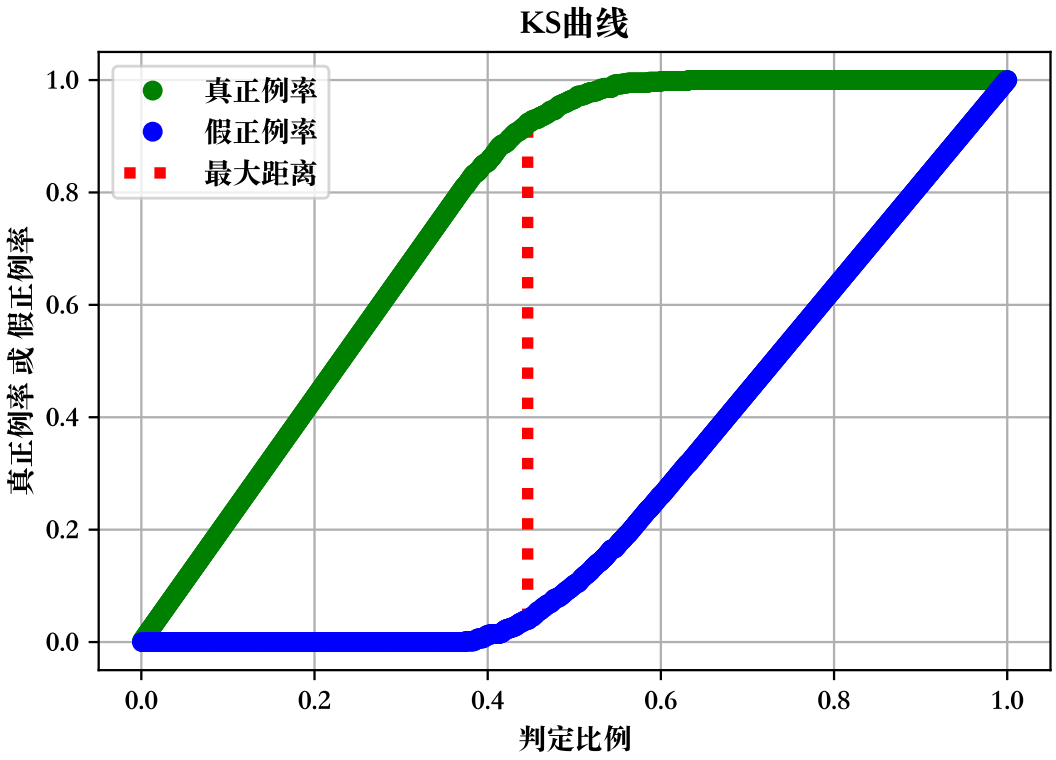
```
there are 188 Support Vectors
time: 155.33205795288086 s
```




```
recall: 0.7215384615384616,
precision: 0.7504,
ks: 0.5682576216434485
```

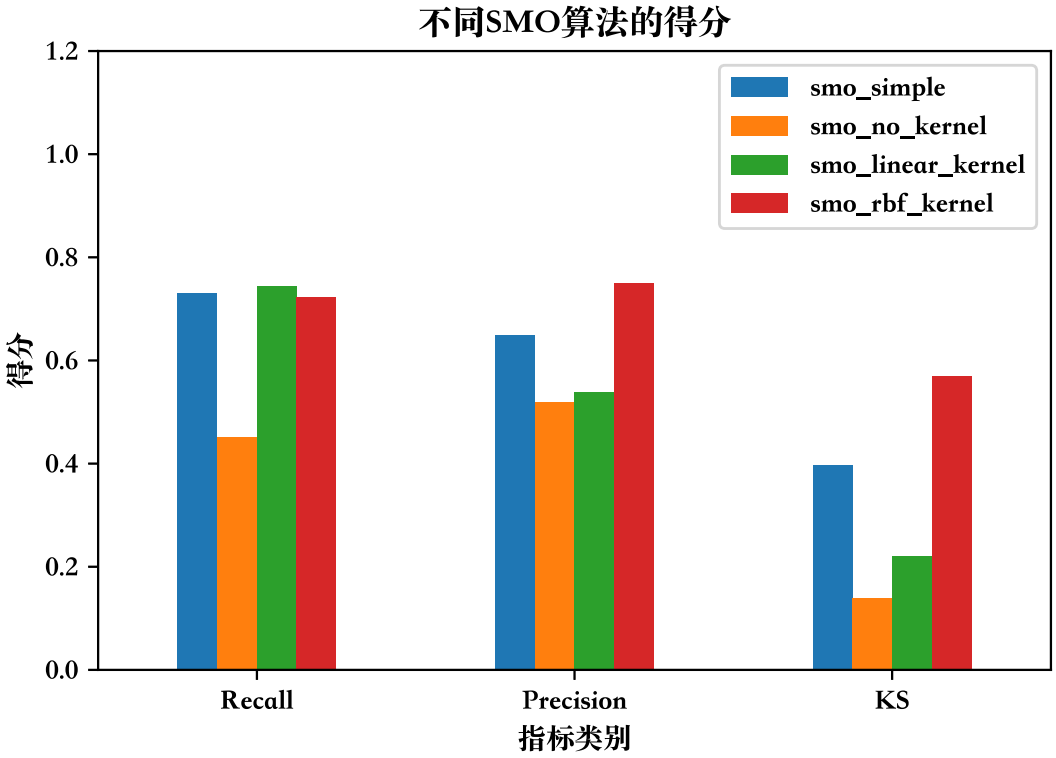
In [11]:
```
    # t[5]
    t.append(test_smo_with_kernel(X,y,df,"rbf",1.3,60,0.0001,1000))
```

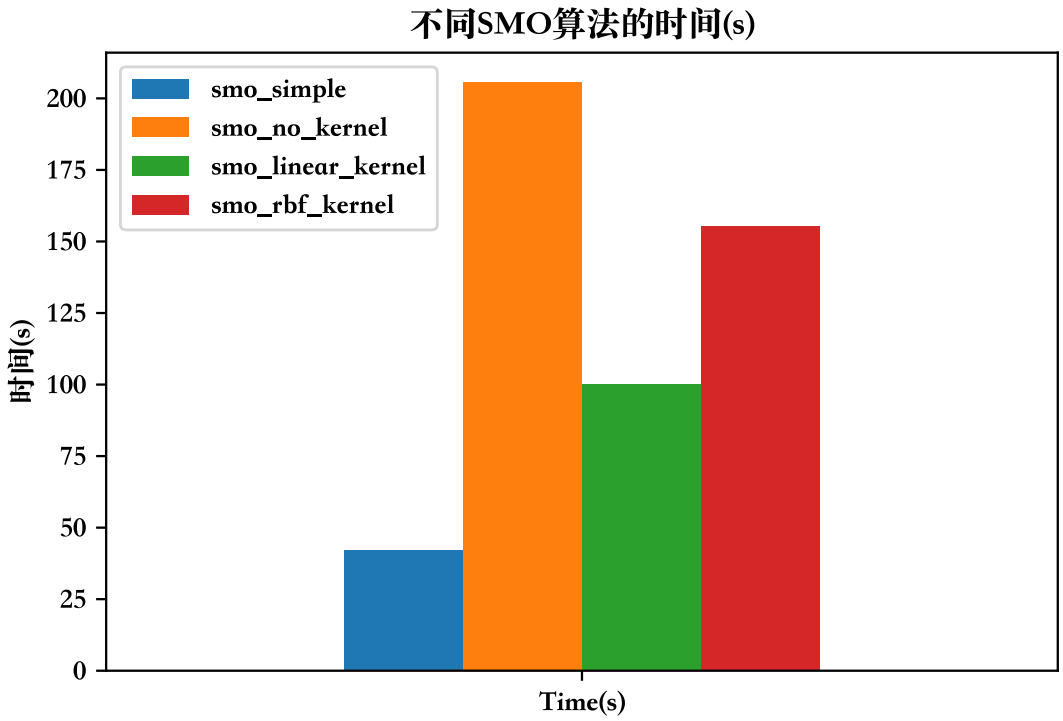
```
there are 623 Support Vectors
time: 245.54531383514404 s
```



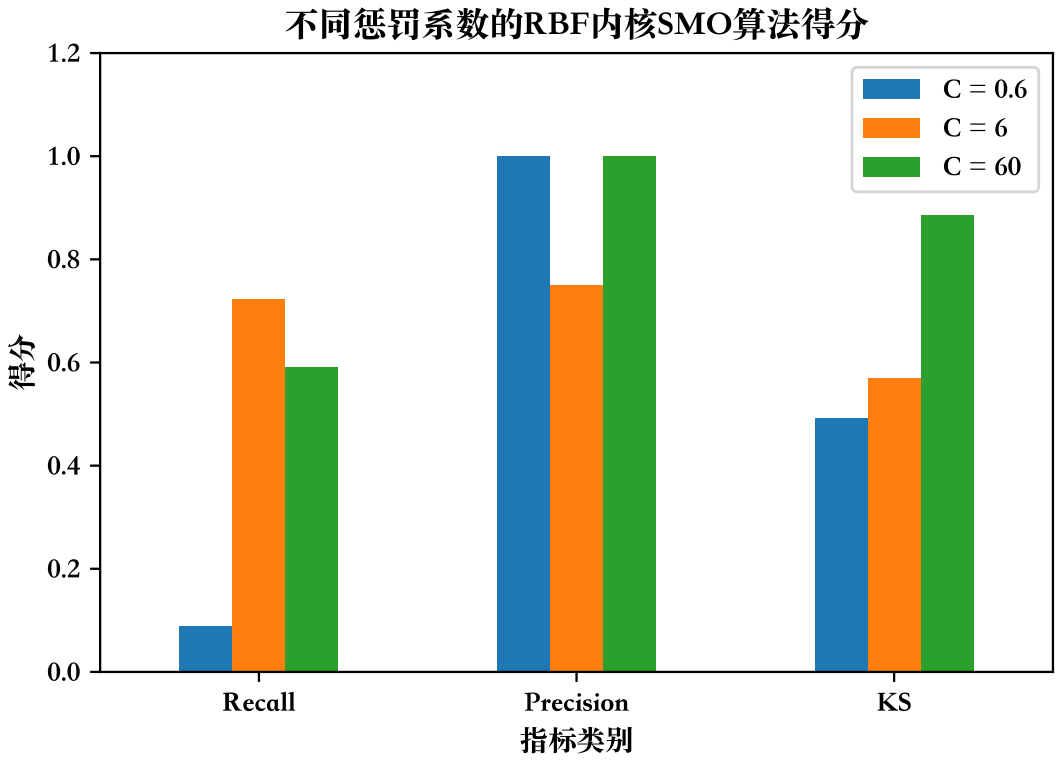

```
recall: 0.5907692307692308,
precision: 1.0,
ks: 0.8852453058752271
```

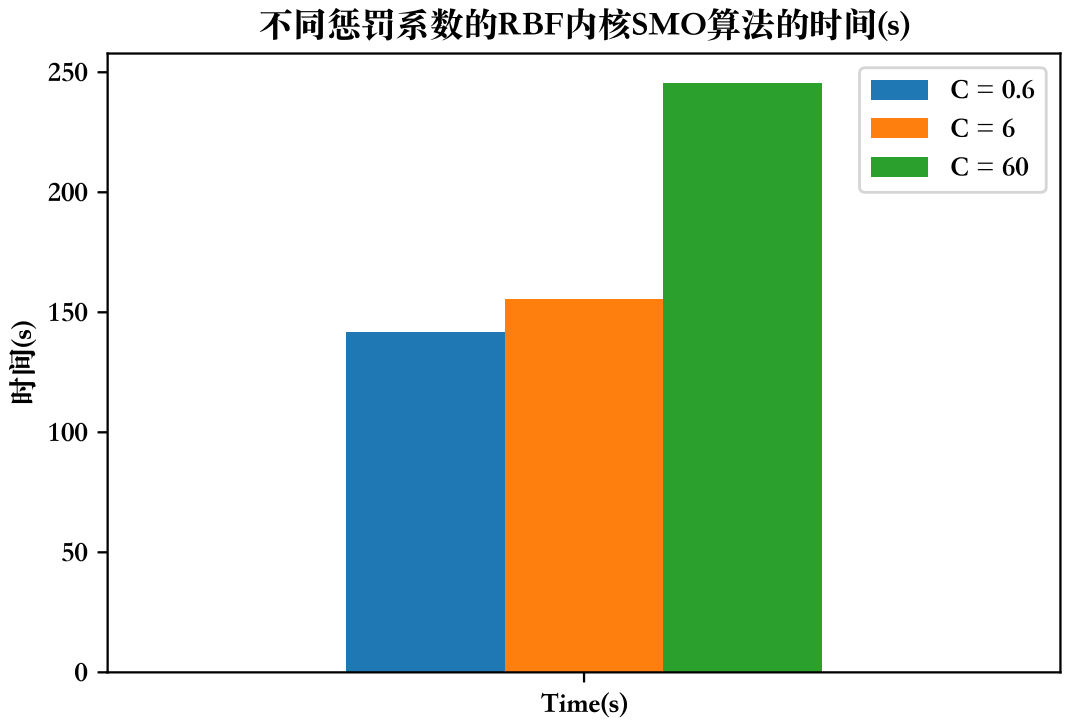
In [14]:
```
    dict1 = {"smo_simple" : t[0][0:4]
         ,"smo_no_kernel" : t[1][0:4]
         ,"smo_linear_kernel": t[2][0:4]
         ,"smo_rbf_kernel": t[4][0:4]
         }
    score_df1 = cmp_score(dict1, 1)
```





In [15]:
```
dict2 = {"C = 0.6" : t[3][0:4]
      ,"C = 6" : t[4][0:4]
      ,"C = 60": t[5][0:4]
      }
score_df2 = cmp_score(dict2, 2)
```





linear内核smo的recall特别高，但是precision和ks值过低
rbf内核smo在惩罚系数取6时各指标都表现不错
所以保存这两个结果

In [16]:
```
t[2][4].to_csv('../result/smo_linear.csv',index=False)
t[4][4].to_csv('../result/smo_rbf_6.csv',index=False)
```