1 Cuprate superconducting materials above liquid nitrogen temperature from machine learning

1.1 Exploratory analysis of data ¶

Contents

Contents

In [2]:

```
import os
import sys
import time
import matplotlib
import pathlib
import graphviz
import itertools
import pycaret
import pyforest
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import seaborn as sns
#from sklearn import *
import seaborn as sns
from sklearn import tree
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
font2 = {'family' : 'Times New Roman',
    'weight' : 'normal',
    'size' : 20,
                }
def plot_roc_curve(fprs,tprs):
    plt.figure(figsize=(8,6),dpi=80)
    plt.plot(fprs,tprs)
```

```
42      plt.plot([0,1],linestyle='--')
43      plt.xticks(fontsize=13)
44      plt.yticks(fontsize=13)
45      font2 = {'family' : 'Times New Roman',
46      'weight' : 'normal',
47      'size' : 20,
48                  }
49      plt.ylabel('TP rate',font2)
50      plt.xlabel('FP rate',font2)
51      plt.title('ROC',font2)
52      plt.savefig('ROC.jpg', dpi=300)
53      plt.show()
54
55  def plot_cnf_matirx(cnf_matrix,description):
56      class_names = [0,1]
57      fig = plt.gcf( )
58      fig.set_size_inches(15.5, 10.5)
59      matplotlib.rcParams['font.sans-serif']=['SimHei']   # 用黑体显示中文
60      matplotlib.rcParams['axes.unicode_minus']=False
61      fig,ax = plt.subplots()
62      tick_marks = np.arange(len(class_names))
63      plt.xticks(tick_marks,class_names)
64      plt.yticks(tick_marks,class_names)
65      sns.heatmap(pd.DataFrame(cnf_matrix), annot = True, cmap = 'OrRd',fmt = 'g')
66      ax.xaxis.set_label_position('top')
67      plt.tight_layout()
68      plt.title(description, y = 1.1,fontsize=16)
69      font2 = {'family' : 'Times New Roman',
70      'weight' : 'normal',
71      'size' : 20,
72                  }
73      plt.ylabel('True0/1',font2)
74      plt.xlabel('Pred0/1',font2)
75      fig = plt.gcf( )
76      fig.set_size_inches(5.5, 4.5)
77      plt.savefig('cnf_matirx.jpg', dpi=300)
78      plt.show()
79
80  def plot_confusion_matrix(cm, classes,
81                            title='Confusion matrix',
82                            cmap=plt.cm.Blues):
83
```

```
84      plt.imshow(cm, interpolation='nearest', cmap=cmap)
85      plt.title(title)
86      plt.colorbar()
87      tick_marks = np.arange(len(classes))
88      plt.xticks(tick_marks, classes, rotation=0)
89      plt.yticks(tick_marks, classes)
90
91      thresh = cm.max() / 2.
92      for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
93          plt.text(j, i, cm[i, j],
94                   horizontalalignment="center",
95                   color="white" if cm[i, j] > thresh else "black")
96
97      plt.tight_layout()
98      plt.ylabel('True label')
99      plt.xlabel('Predicted label')
```

In [3]:

```
1  data = pd.read_csv("./data.csv")
2  S1 = pd.read_csv("./S1.csv")
3  S2 = pd.read_csv("./S2.csv")
4  df =S1
```

In [4]:
```
1 data.describe()
```

Out[4]:

	Tc
count	12340.000000
mean	23.503293
std	28.565083
min	0.000500
25%	3.400000
50%	10.000000
75%	33.000000
max	143.000000

Contents

In [5]:
```
1 df
```

Out[5]:

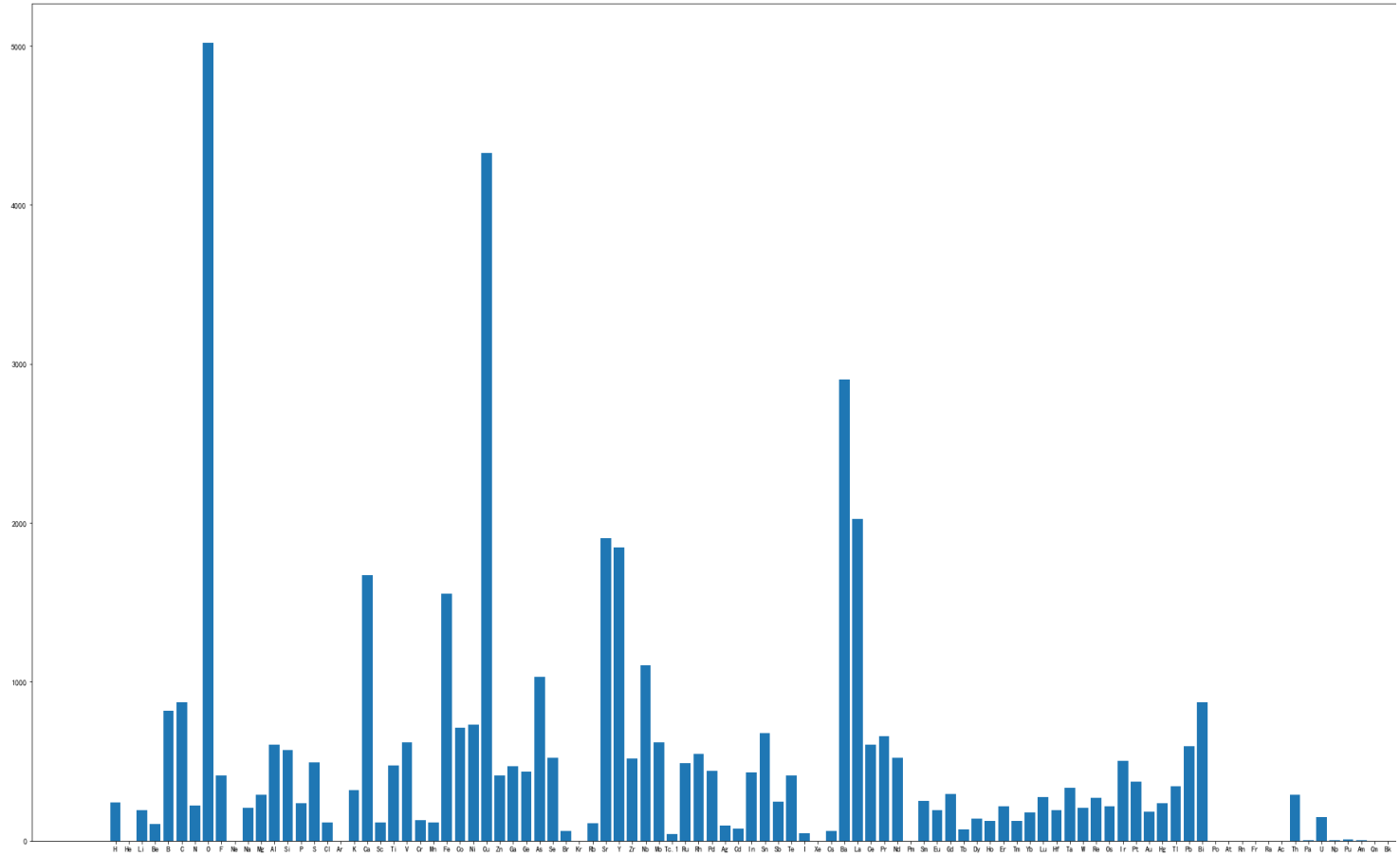
	name	Tc	H	He	Li	Be	B	C	N	O	...	Mt	Ds	Rg	Cn	Nh
0	Zr74.1Ti3.9Ni22	3.650	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
1	Zr70.2Ti7.8Ni22	3.320	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
2	Zr66.3Ti11.7Ni22	3.560	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
3	Zr62.4Ti15.6Ni22	3.215	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
4	Zr55Cu30Al10Ni5	1.350	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12335	Ag0.05Rh0.04Ti0.91	1.950	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
12336	Ag0.03Tl0.97	2.670	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
12337	Ag0.035Cd0.01Sn0.955	3.650	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
12338	Ag0.005Zn0.995	0.763	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
12339	Ag0.002Al0.998	1.128	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	

12340 rows × 120 columns

In [8]:

```
distribution = []
elem = []
for i  in  df.columns[2:]  :
    ca = (df[i]==0).sum()
    realele = 12340-ca
    distribution.append(realele)
    elem.append(i)
fengfu = pd.DataFrame(distribution)
elem = pd.DataFrame(elem)
elem["elem"]=elem
elem["distribution"]=distribution
x = elem["elem"]
y = elem["distribution"]
#seaborn.barplot(data=fengfu)
matplotlib.rcParams['font.sans-serif']=['SimHei']
matplotlib.rcParams['axes.unicode_minus']=False
plt.figure(figsize=(40,20))
plt.bar(x,y)
plt.savefig('distribution.jpg', dpi=300)
plt.show()
```

Contents

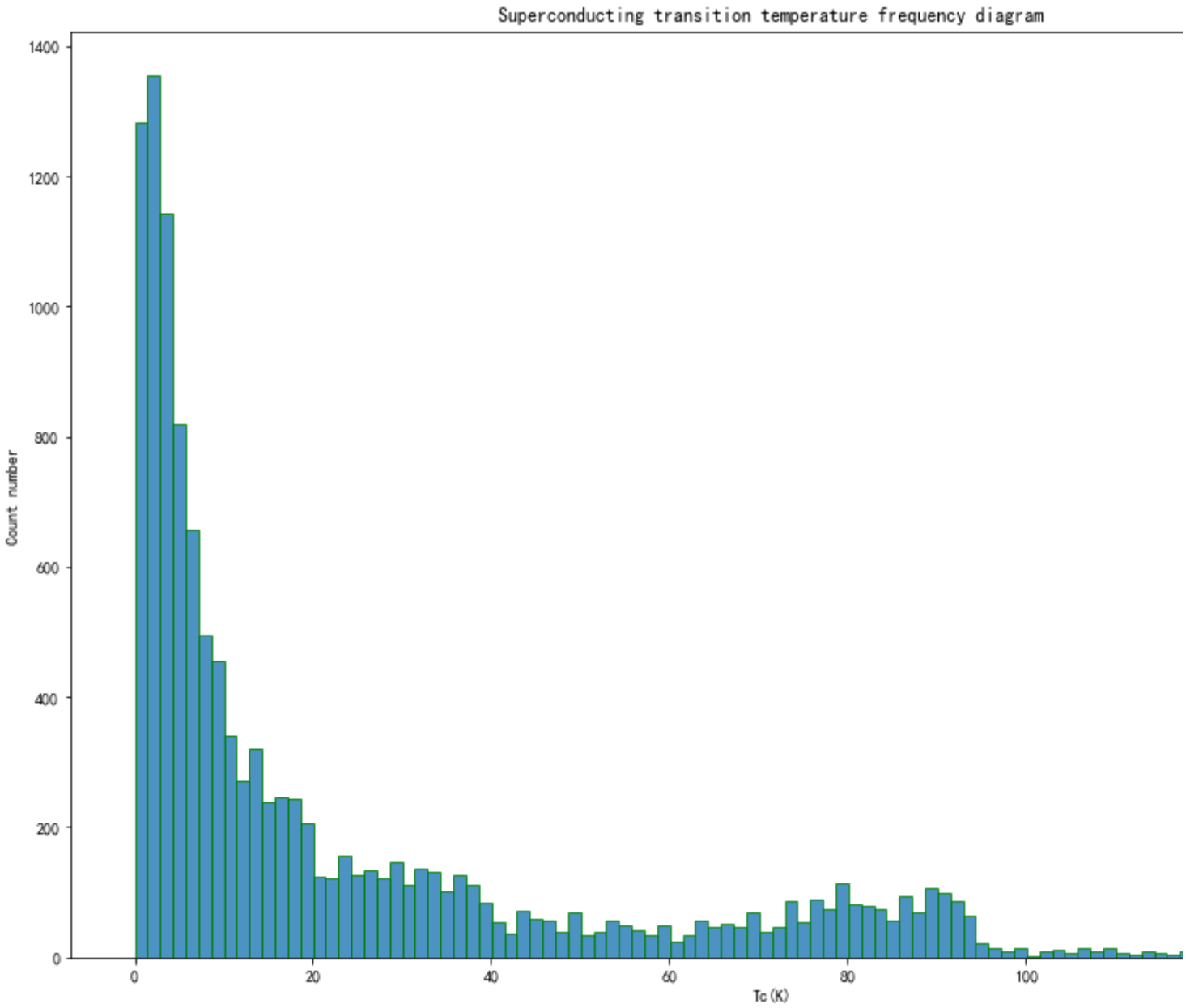


Contents

In [9]:

```
#Superconducting transition temperature frequency diagram of all materials
fig = plt.gcf( )
fig.set_size_inches(15.5, 10.5)
matplotlib.rcParams['font.sans-serif']=['SimHei']
matplotlib.rcParams['axes.unicode_minus']=False
plt.hist(data["Tc"], bins=100, color=None,edgecolor="green", alpha=0.8)
plt.title('Superconducting transition temperature frequency diagram')
plt.xlabel("Tc(K)")
plt.ylabel("Count number")
plt.show()
```

Contents



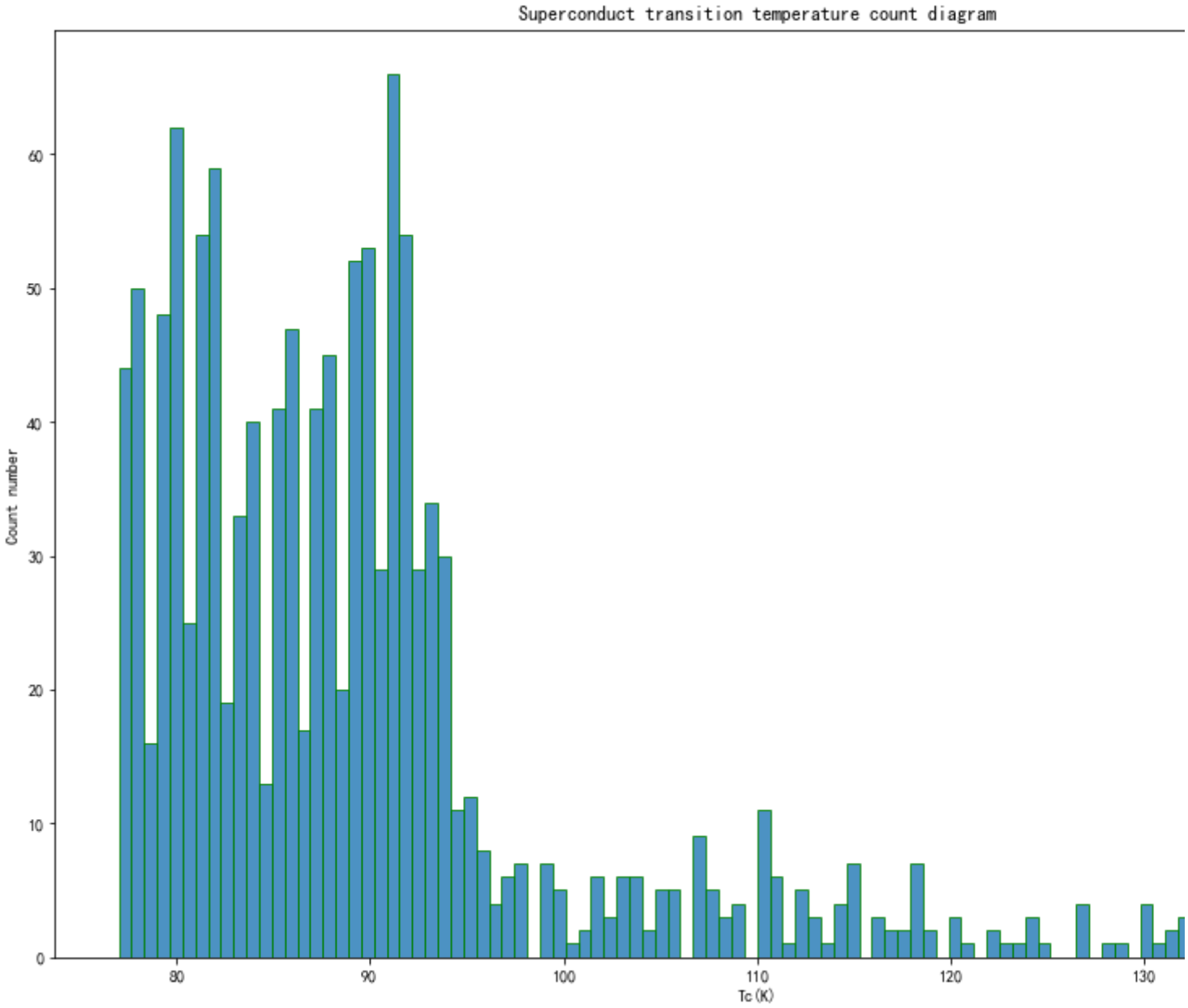
In [10]:

```
#Superconducting transition temperature frequency diagram of materiasl which Tc > 77K
fig = plt.gcf( )
fig.set_size_inches(15.5, 10.5)
dataTc = data[data["Tc"]>=77]
dataTc
matplotlib.rcParams['font.sans-serif']=['SimHei']
matplotlib.rcParams['axes.unicode_minus']=False
#plt.hist(dataTc, bins=100,log=False, color=None)
plt.hist(dataTc["Tc"], bins=100, normed=0,  edgecolor="green", alpha=0.8)
plt.xlabel("Tc(K)")
plt.ylabel("Count number")
plt.title('Superconduct transition temperature count diagram')
```

```
D:\py soft\conda\lib\site-packages\ipykernel_launcher.py:9: MatplotlibDeprecationWarning:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' instead.
  if __name__ == '__main__':
```

Out[10]:

```
Text(0.5, 1.0, 'Superconduct transition temperature count diagram')
```



1.2 Descriptors populate the data overview

1.2.1 The descriptor is generated by AFS1 and AFS2 software

In [11]:
```
S1 = pd.read_csv("./S1.csv")
S2 = pd.read_csv("./S2.csv")
```

In [12]:
```
S1
```

Out[12]:

	name	Tc	H	He	Li	Be	B	C	N	O	...	Mt	Ds	Rg	Cn	Nh
0	Zr74.1Ti3.9Ni22	3.650	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
1	Zr70.2Ti7.8Ni22	3.320	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
2	Zr66.3Ti11.7Ni22	3.560	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
3	Zr62.4Ti15.6Ni22	3.215	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
4	Zr55Cu30Al10Ni5	1.350	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12335	Ag0.05Rh0.04Ti0.91	1.950	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
12336	Ag0.03Tl0.97	2.670	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
12337	Ag0.035Cd0.01Sn0.955	3.650	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
12338	Ag0.005Zn0.995	0.763	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	
12339	Ag0.002Al0.998	1.128	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	

12340 rows × 120 columns

Contents

In [13]:
```
1 S2
```

Out[13]:

	name	Tc	Cell_Min_Number	Cell_Max_Number	Cell_range_Number	C
0	Hg0.66Pb0.34Ba2Ca1.98Cu2.9O8.4	143.000000	1.71253	6.87961	5.16708	
1	Hg0.75Ba2.07Ca2.07Cu3.11O8.208	135.800000	2.55429	7.15202	4.59773	
2	Hg0.75Ba2.07Ca2.07Cu3.11O8.187	135.400000	2.55761	7.16130	4.60369	
3	Hg1Ba2Ca2Cu3O8.29	135.000000	2.45549	6.87538	4.41989	
4	Hg1Ba2Ca2Cu3O8.27	135.000000	2.45851	6.88384	4.42532	
...	...	...	...	...	...	
12335	Ag0.9Ga0.1	0.006500	3.10000	42.30000	39.20000	
12336	Pt1	0.004157	78.00000	78.00000	0.00000	
12337	La1.92Sr0.08Cu0.99Ni0.01O4	0.001000	0.04000	15.63430	15.59430	
12338	Gd0.9Pr0.95Ce0.15Cu1O4	0.001000	1.24286	8.22857	6.98571	
12339	Au0.978In0.022	0.000500	1.07800	77.26200	76.18400	

12340 rows × 122 columns

1.3 Classification model training

1.3.1 Use pycaret to make a rough assessment of multimodels

Contents

In [14]:

```
CLS1 = pd.read_csv("./CLS1.csv")
CLS2 = pd.read_csv("./CLS2.csv")
dataset = CLS1
dataset
```

Out[14]:

	name	Tc	H	He	Li	Be	B	C	N	O	...	Mt	Ds	R
0	Hg0.66Pb0.34Ba2Ca1.98Cu2.9O8.4	0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.515971	...	0	0	
1	Hg0.75Ba2.07Ca2.07Cu3.11O8.208	0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.506417	...	0	0	
2	Hg0.75Ba2.07Ca2.07Cu3.11O8.187	0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.505776	...	0	0	
3	Hg1Ba2Ca2Cu3O8.29	0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.508901	...	0	0	
4	Hg1Ba2Ca2Cu3O8.27	0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.508297	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12335	Ag0.9Ga0.1	1	0.0	0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0	0	
12336	Pt1	1	0.0	0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0	0	
12337	La1.92Sr0.08Cu0.99Ni0.01O4	1	0.0	0	0.0	0.0	0.0	0.0	0.0	0.571429	...	0	0	
12338	Gd0.9Pr0.95Ce0.15Cu1O4	1	0.0	0	0.0	0.0	0.0	0.0	0.0	0.571429	...	0	0	
12339	Au0.978In0.022	1	0.0	0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0	0	

12340 rows × 120 columns

Contents

In [15]:

```
data = dataset.sample(frac=0.8, random_state=2020).reset_index(drop=True)
data_unseen = dataset.drop(data.index).reset_index(drop=True)

print('Data for Modeling: ' + str(data.shape))
print('Unseen Data For Predictions: ' + str(data_unseen.shape))
from pycaret.classification import *
exp_clf101 = setup(data = data, target = 'Tc', session_id=123)
```

6	Categorical Features	37
7	Ordinal Features	False
8	High Cardinality Features	False
9	High Cardinality Method	None
10	Sampled Data	(9872, 120)
11	Transformed Train Set	(6910, 9991)
12	Transformed Test Set	(2962, 9991)
13	Numeric Imputer	mean
14	Categorical Imputer	constant
15	Normalize	False
16	Normalize Method	None
17	Transformation	False
18	Transformation Method	None

In [ ]:

```
compare_models()    ##Run to get model ranking
```

In [ ]:

```
dataset = CLS2
data = dataset.sample(frac=0.8, random_state=2020).reset_index(drop=True)
data_unseen = dataset.drop(data.index).reset_index(drop=True)

print('Data for Modeling: ' + str(data.shape))
print('Unseen Data For Predictions: ' + str(data_unseen.shape))
from pycaret.classification import *
exp_clf101 = setup(data = data, target = 'Tc', session_id=123)
compare_models()
```

1.3.2 Select the appropriate model for classification (RFC,KNN,DT,ADC)

1.3.3 RFC

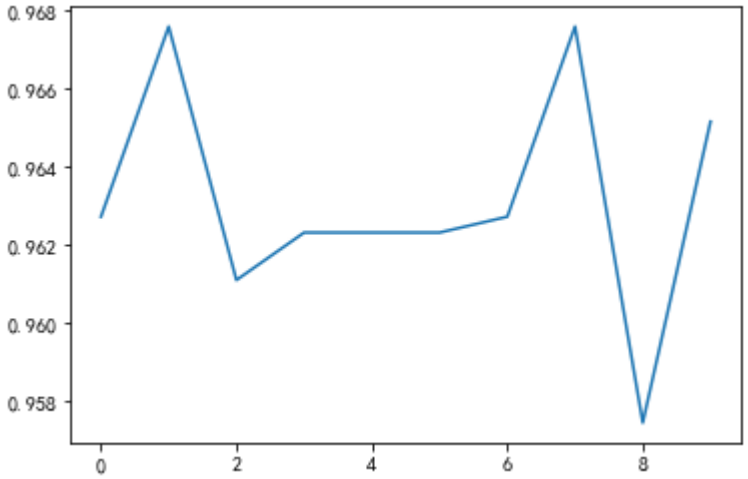
Contents

Contents

In [17]:

```
# RFC
data = CLS1
X = data.iloc[:,2:].values
y = data.iloc[:,1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2019)
rfc = RandomForestClassifier(n_estimators = 200)
rfc.fit(X_train,y_train)
print("RFCscore",rfc.score(X_test, y_test.astype('int')))
cv = ShuffleSplit(n_splits=10, test_size=.2)
scores = cross_val_score(rfc, X, y, cv=cv)
print(scores)
print(scores.mean())
ll = np.linspace(0,9,10)
ll_x = ll
ll_y = scores
plt.plot(ll_x,ll_y)
plt.show( )
```

```
RFCscore 0.9688006482982172
[0.96272285 0.96758509 0.96110211 0.96231767 0.96231767 0.96231767
 0.96272285 0.96758509 0.95745543 0.96515397]
0.9631280388978931
```



1.3.4 KNN

In [18]:
```
# KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
param_search = [
    {
        "weights":["uniform"],
        "n_neighbors":[i for i in range(1,11)]
    },
    {
        "weights":["distance"],
        "n_neighbors":[i for i in range(1,11)],
        "p":[i for i in range(1,6)]
    }
]
knn_clf = KNeighborsClassifier()
```

In [19]:
```
grid_search = GridSearchCV(knn_clf, param_search)
```

In [21]:
```
%%time
grid_search.fit(X_train, y_train)
grid_search.best_estimator_
```

```
Wall time: 24min 44s
```

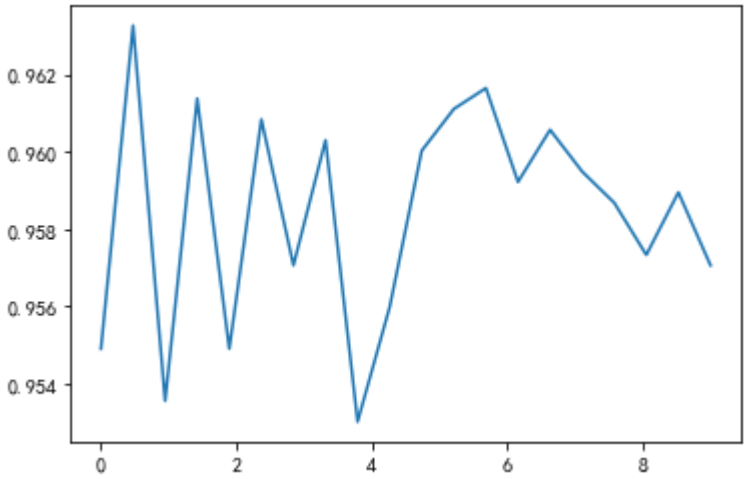
Out[21]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=6, p=2,
                     weights='distance')
```

Contents

In [22]:

```
# KNN
X = data.iloc[ : ,2:].values
y = data.iloc[ : ,1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2020)
from sklearn.neighbors import KNeighborsClassifier
reg = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                           metric_params=None, n_jobs=None, n_neighbors=6, p=2,
                           weights='distance')
reg.fit(X_train, y_train.astype('int'))
print(reg.score(X_test, y_test.astype('int')))

from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
cv = ShuffleSplit(n_splits=20, test_size=.3)
scores = cross_val_score(reg, X, y, cv=cv)
print(scores)
print(scores.mean())
ll = np.linspace(0,9,20)
ll_x = ll
ll_y = scores
plt.plot(ll_x,ll_y)
plt.show( )

```

```
0.9554294975688817
[0.95488925 0.9632631  0.95353863 0.96137223 0.95488925 0.96083198
 0.95705024 0.96029173 0.95299838 0.95596975 0.96002161 0.96110211
 0.96164236 0.95921124 0.96056186 0.95948136 0.95867099 0.95732037
 0.95894111 0.95705024]
0.9584548892490545
```



Contents

1.3.5 DT

In [24]:

```
# DT
data = CLS1
X = data.iloc[:,2:].values
y = data.iloc[:,1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2022)
data
```

Out[24]:

	name	Tc	H	He	Li	Be	B	C	N	O	F	Ne	Na	M
0	Hg0.66Pb0.34Ba2Ca1.98Cu2.9O8.4	0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.515971	0.0	0	0.0	
1	Hg0.75Ba2.07Ca2.07Cu3.11O8.208	0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.506417	0.0	0	0.0	
2	Hg0.75Ba2.07Ca2.07Cu3.11O8.187	0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.505776	0.0	0	0.0	
3	Hg1Ba2Ca2Cu3O8.29	0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.508901	0.0	0	0.0	
4	Hg1Ba2Ca2Cu3O8.27	0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.508297	0.0	0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12335	Ag0.9Ga0.1	1	0.0	0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0	0.0	
12336	Pt1	1	0.0	0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0	0.0	
12337	La1.92Sr0.08Cu0.99Ni0.01O4	1	0.0	0	0.0	0.0	0.0	0.0	0.0	0.571429	0.0	0	0.0	
12338	Gd0.9Pr0.95Ce0.15Cu1O4	1	0.0	0	0.0	0.0	0.0	0.0	0.0	0.571429	0.0	0	0.0	
12339	Au0.978In0.022	1	0.0	0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0	0.0	

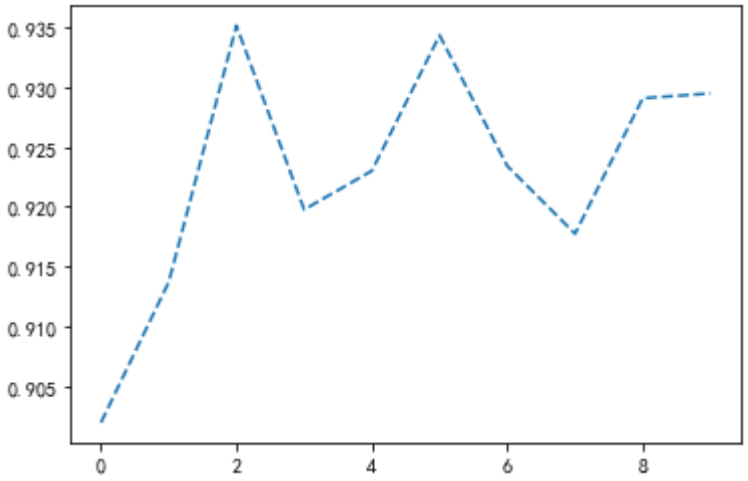
12340 rows × 120 columns

Contents

In [25]:

```
clf = tree.DecisionTreeClassifier(max_depth=7,max_features=30,criterion="entropy") #初始化树模型
clf = clf.fit(X, y)                                    #实例化训练集
score = clf.score(X, y)                                  #返回预测的准确度
print("DT",score)
cv = ShuffleSplit(n_splits=10, test_size=.2)
scores = cross_val_score(clf, X, y, cv=cv)
print(scores)
print(scores.mean())
ll = np.linspace(0,9,10)
ll_x = ll
ll_y = scores
plt.plot(ll_x,ll_y,"--")
plt.show( )
```

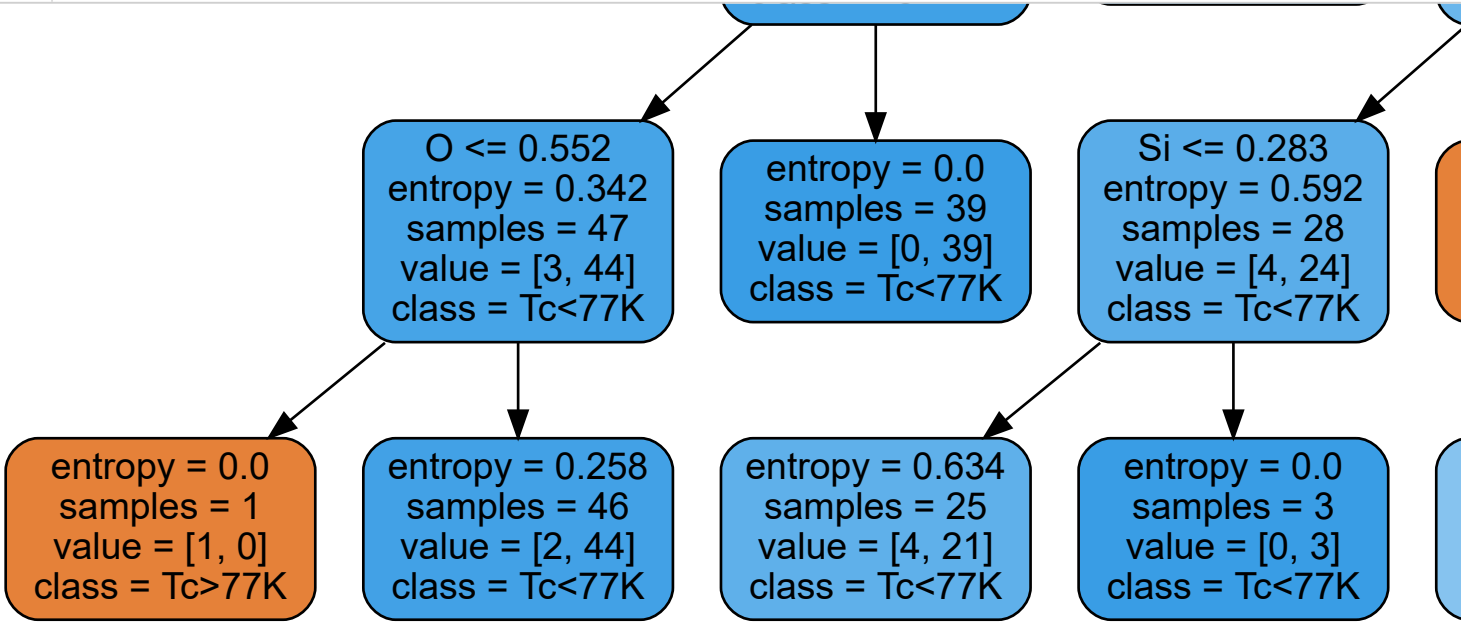
```
DT 0.9289303079416532
[0.90194489 0.9136953  0.93517018 0.9197731  0.92301459 0.93435981
 0.92341977 0.91774716 0.92909238 0.92949757]
0.9227714748784441
```



1.3.5.1 Visualization of decision trees

In [26]:

```
feature_name = data.iloc[:,2:].columns
import graphviz
dot_data = tree.export_graphviz(clf
                                ,out_file=None
                                ,feature_names= feature_name
                                ,class_names=["Tc>77K","Tc<77K"]
                                ,filled=True
                                ,rounded=True
                               )
graph = graphviz.Source(dot_data)
graph
```



In [ ]:

```
# graph.render(filename="tree")    ##save DT graph as a pdf
```

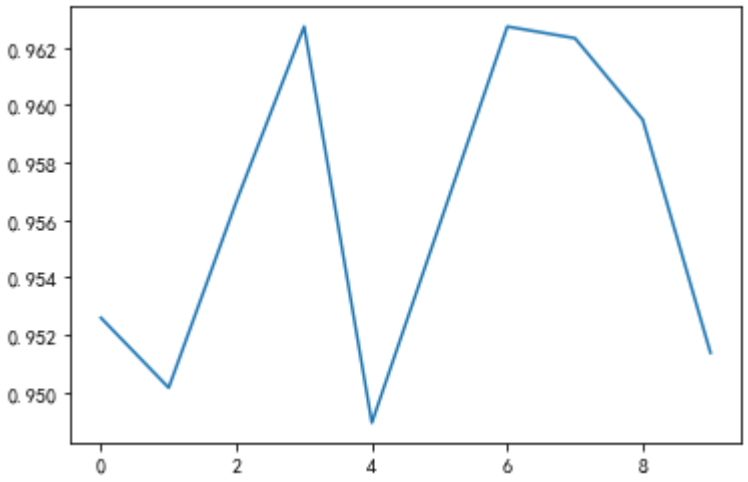
1.3.6 ADC

Contents

In [27]:

```
# Ada
from sklearn.ensemble import AdaBoostClassifier
abc = AdaBoostClassifier(n_estimators = 900)
data = CLS1
X = data.iloc[:,2:].values
y = data.iloc[:,1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2019)
abc.fit(X_train,y_train)
print("ABCscore",abc.score(X_test, y_test.astype('int')))
cv = ShuffleSplit(n_splits=10, test_size=.2)
scores = cross_val_score(abc, X, y, cv=cv)
print(scores)
print(scores.mean())
ll = np.linspace(0,9,10)
ll_x = ll
ll_y = scores
plt.plot(ll_x,ll_y)
plt.show( )
```

```
ABCscore 0.9639384116693679
[0.95259319 0.95016207 0.95664506 0.96272285 0.94894652 0.95583468
 0.96272285 0.96231767 0.95948136 0.95137763]
0.9562803889789302
```



1.3.7 RFC : Analysis of multiple achievement indicators

In [28]:
```
# RFC
data = CLS1
X = data.iloc[:,2:].values
y = data.iloc[:,1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2019)
rfc = RandomForestClassifier(n_estimators = 200)
rfc.fit(X_train,y_train)
rfc.score(X_test,y_test)
```

Out[28]: 0.9679902755267423

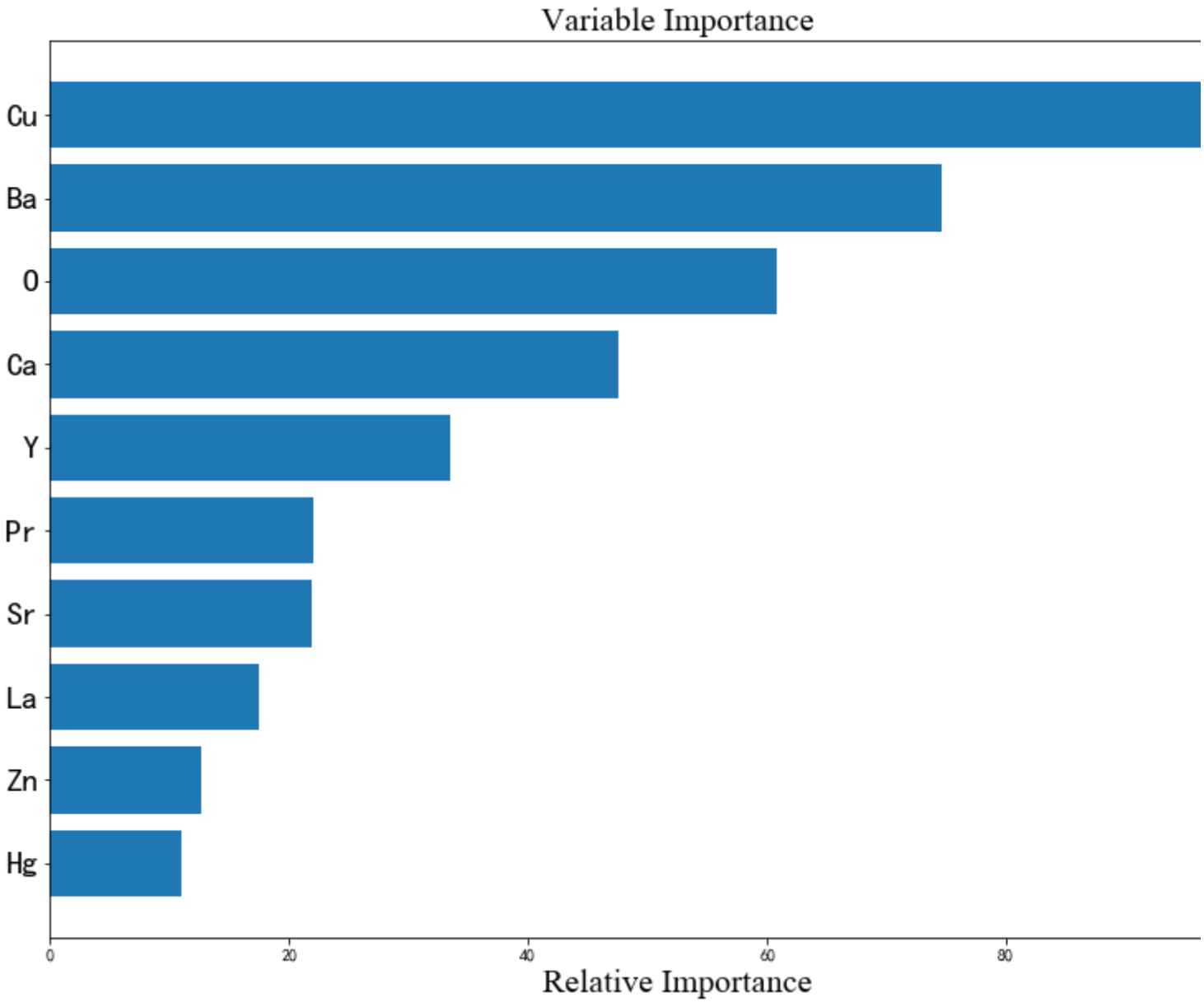
In [29]:
```
[*zip(feature_name,clf.feature_importances_)]
```

```
('Tb', 0.0),
('Dy', 0.0),
('Ho', 0.0),
('Er', 0.0),
('Tm', 0.0),
('Yb', 0.0),
('Lu', 0.0),
('Hf', 0.0),
('Ta', 0.0),
('W', 0.0),
('Re', 0.0014127133231240914),
('Os', 0.0),
('Ir', 0.0),
('Pt', 0.0),
('Au', 0.0),
('Hg', 0.022343358875741987),
('Tl', 0.02037171072073104),
('Pb', 0.0),
('Bi', 0.0),
```

In [30]:

```
clf = rfc
feature_importance = clf.feature_importances_
# make importances relative to max importance
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.subplot(1, 2, 2)
plt.barh(pos[-10:], feature_importance[sorted_idx[-10:]], align='center')
plt.yticks(pos[-10:], data.iloc[:,2:].columns[sorted_idx[-10:]],fontsize=20)
plt.xlabel('Relative Importance',font2)
plt.title('Variable Importance',font2)
fig = plt.gcf( )
fig.set_size_inches(30,10)
#plt.savefig('Variable Importance',dpi=300,bbox_inches="tight")
plt.show()
```

Contents



1.3.7.1 F1 score

In [31]:
```
from sklearn.metrics import f1_score
y_pred = rfc.predict(X_test)
f1_score(y_test,y_pred)
```

Out[31]: 0.9823305748154776

1.3.7.2 confusion matrix

In [32]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

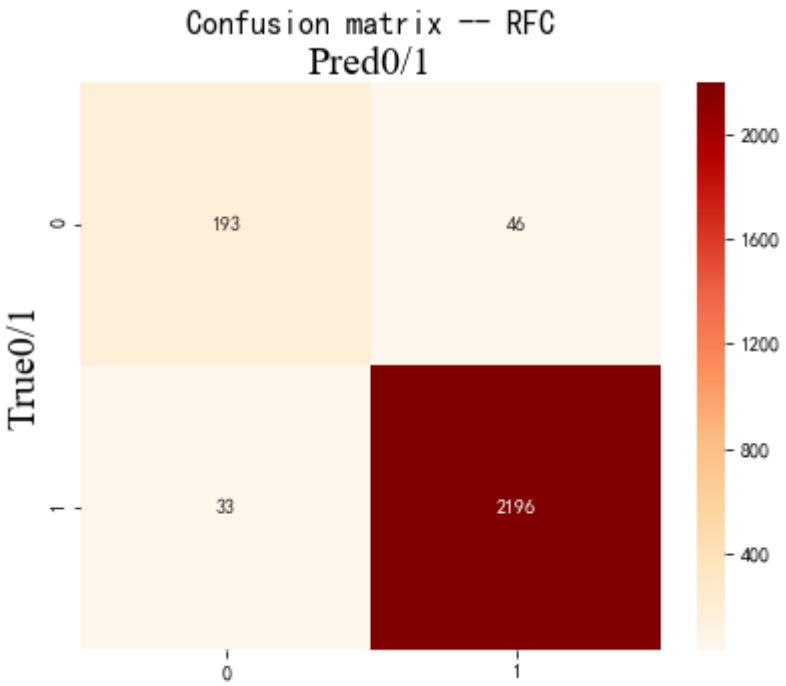
```
              precision    recall  f1-score   support

           0       0.85      0.81      0.83       239
           1       0.98      0.99      0.98      2229

    accuracy                           0.97      2468
   macro avg       0.92      0.90      0.91      2468
weighted avg       0.97      0.97      0.97      2468
```

In [33]:
```
from sklearn.metrics import confusion_matrix
cnf_matrix = confusion_matrix(y_test,y_pred)
cnf_matrix
```

Out[33]:
```
array([[ 193,   46],
       [  33, 2196]], dtype=int64)
```

In [34]:
```
1 plot_cnf_matirx(cnf_matrix,'Confusion matrix -- RFC')
```

<Figure size 1116x756 with 0 Axes>



1.3.7.3 Accuracy and call-back curves

In [35]:

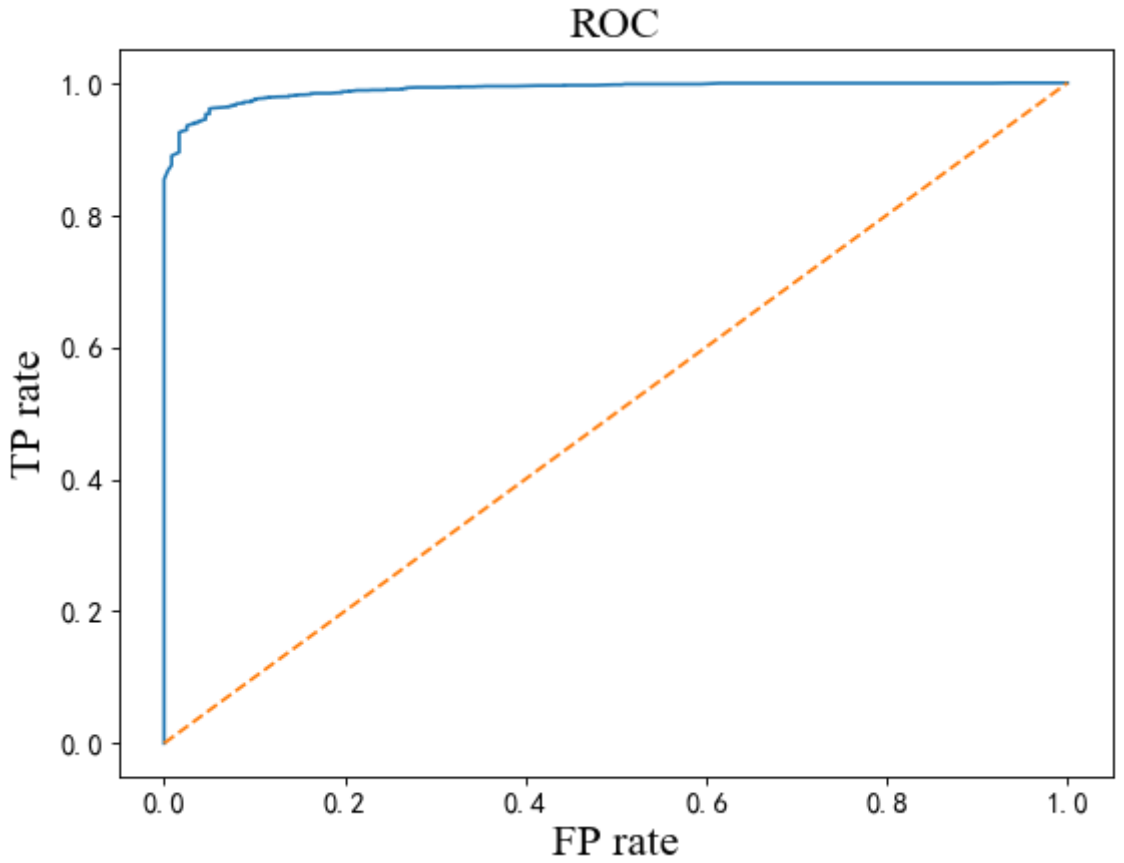
```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_auc_score
rf_clf = rfc
y_probabilities_rf = rf_clf.predict_proba(X_test)[:,1]

roc_auc_score(y_test,y_probabilities_rf)
```

Out[35]: 0.9901591985448568

In [36]:

```
from sklearn.metrics import roc_curve
fprs4,tprs4,thresholds4 = roc_curve(y_test,y_probabilities_rf)
fig = plt.gcf( )
fig.set_size_inches(15.5, 10.5)

plot_roc_curve(fprs4,tprs4)

#plt.savefig('ROC.jpg', dpi=300)
```

```
<Figure size 1116x756 with 0 Axes>
```



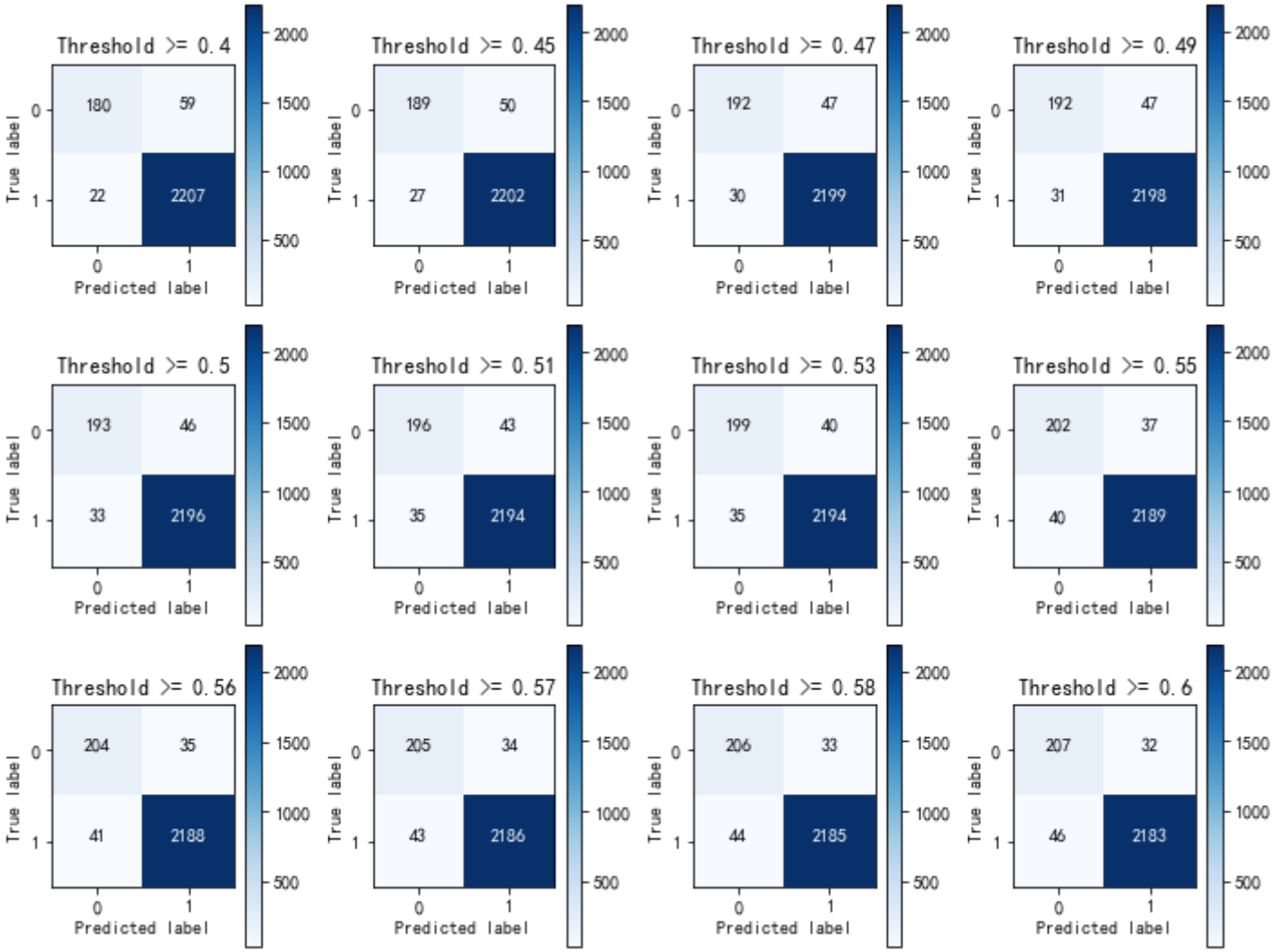
1.3.7.4 The threshold of model

Contents

In [37]:

```
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=0)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

thresholds = [0.4,0.45,0.47,0.49,0.5,0.51,0.53,0.55,0.56,0.57,0.58,0.60]
plt.figure(figsize=(10,10))
m = 1
for i in thresholds:
    y_test_predictions_high_recall = y_probabilities_rf > i

    plt.subplot(4,4,m)
    m += 1

    cnf_matrix = confusion_matrix(y_test,y_test_predictions_high_recall)
    np.set_printoptions(precision=2)

 #  print (i,"Recall:{}".format(cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1])))
 #   acc = rfc.score(X_test,y_test)
 #   print(acc)

    class_names = [0,1]
    plot_confusion_matrix(cnf_matrix
                          , classes=class_names
                          , title='Threshold >= %s'%i)
```

```
42  #  plt.savefig("before", dpi=300)
```



1.3.7.5 Cost sensitive learning

Bayes formula： P(A|B)=P(B|A)*P(A)/P(B)

Contents

In [38]:
```
len(data[data["Tc"]==0])
```

Out[38]: 1233

In [39]:
```
len(data[data["Tc"]==1])
```

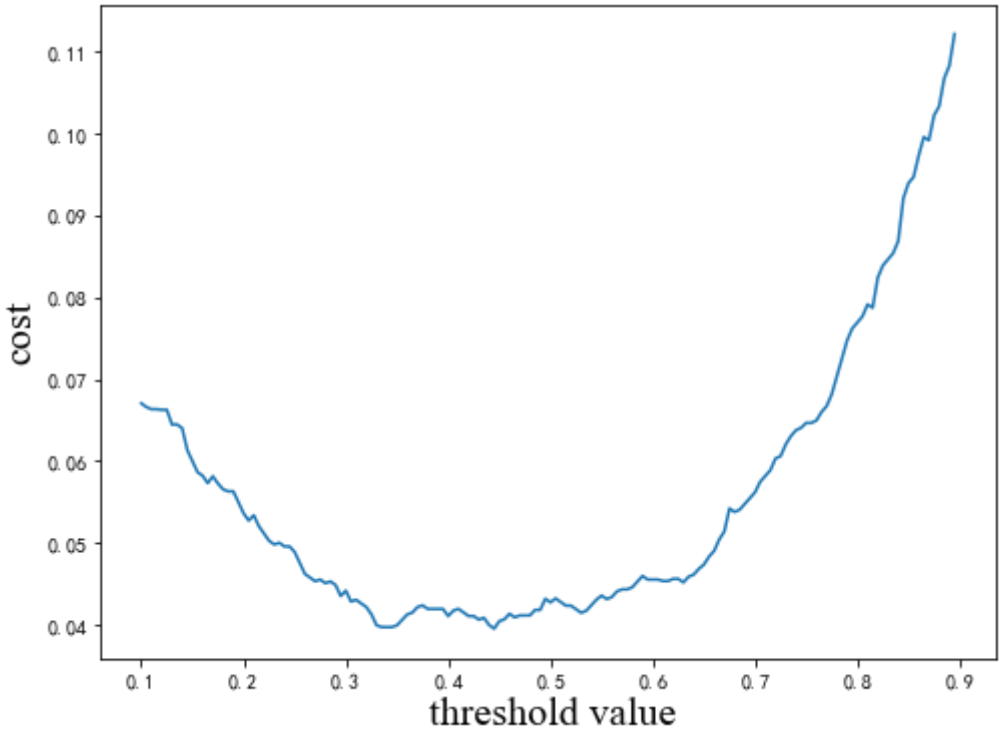
Out[39]: 11107

In [43]:
```
log = []
for i in range(100,900,5):
#    print(i/1000)
    log.append(i/1000)
```

In [48]:
```
mg = []
thresholds = log
m = 1
for i in thresholds:
    y_test_predictions_high_recall = y_probabilities_rf > i
    cnf_matrix = confusion_matrix(y_test,y_test_predictions_high_recall)
    np.set_printoptions(precision=2)
#    print(cnf_matrix)
    C01 = cnf_matrix[0][1]
    C10 = cnf_matrix[1][0]
    C00 = cnf_matrix[0][0]
    C11 = cnf_matrix[1][1]
    C =C01+C00+C10+C11
    cost = (1*(C01)/((C11+C10)))+((1.5*C10)/((C11+C01)))
#    cost = 1
#    recall =cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1])
#    acc = (C00+C11)/C
#    COST = (1/cost)*recall*acc
    COST=cost
#    print(COST)
    mg.append(COST)
```

In [49]:

```
mingan = np.array(mingan)
log = np.array(log)
plt.plot(log,mg)
font2 = {'family' : 'Times New Roman',
'weight' : 'normal',
'size' : 20,
}
plt.ylabel('cost',font2)
plt.xlabel(' threshold value',font2)
fig = plt.gcf( )
fig.set_size_inches(8, 6)
#plt.figure(figsize=(8,6))
plt.savefig('cost.jpg', dpi=300)
plt.show( )
```

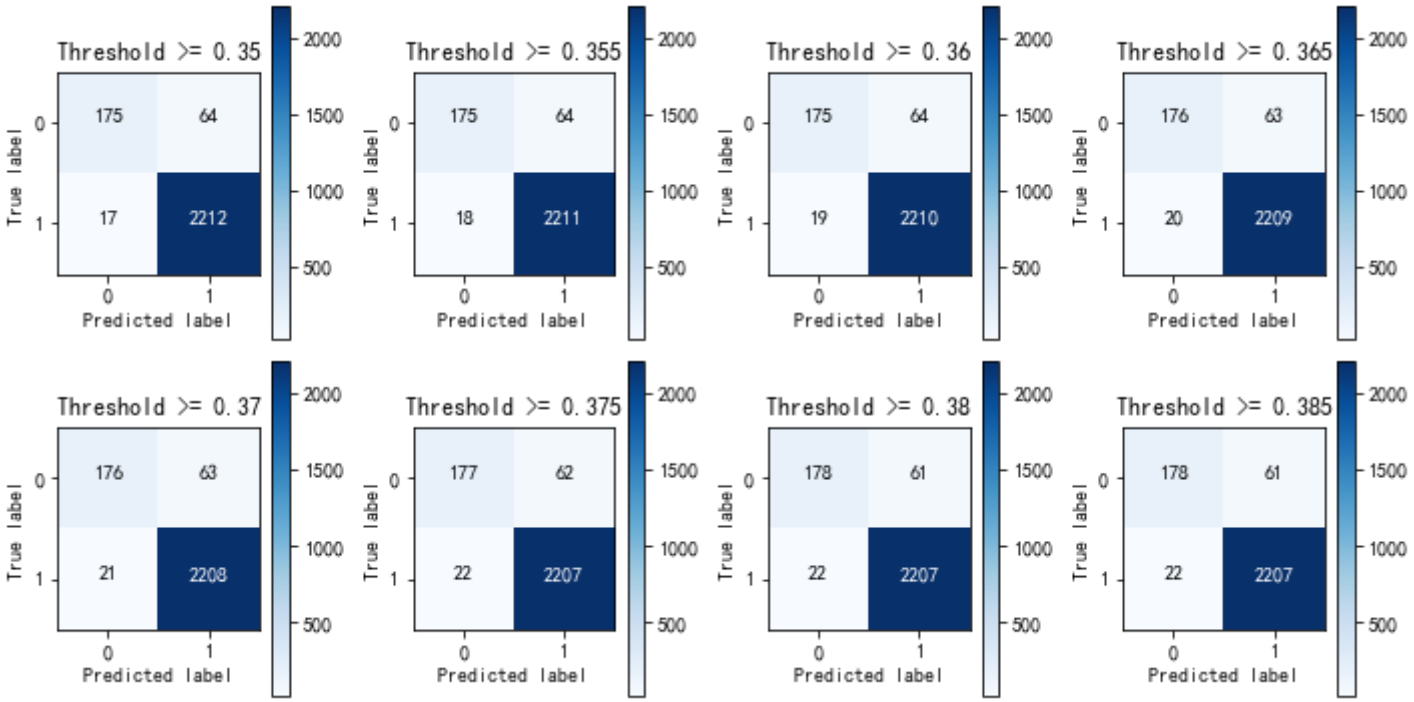


Contents

Contents

In [50]:

```
thresholds = [0.350,0.355,0.360,0.365,0.370,0.375,0.380,0.385]
plt.figure(figsize=(10,10))

m = 1
for i in thresholds:
    y_test_predictions_high_recall = y_probabilities_rf > i

    plt.subplot(4,4,m)
    m += 1

    cnf_matrix = confusion_matrix(y_test,y_test_predictions_high_recall)
    np.set_printoptions(precision=2)

#    print (i,"Recall:{}".format(cnf_matrix[0,0]/(cnf_matrix[0,1]+cnf_matrix[0,0])))
 #   acc = rfc.score(X_test,y_test)
 #   print(acc)

    class_names = [0,1]
    plot_confusion_matrix(cnf_matrix
                          , classes=class_names
                          , title='Threshold >= %s'%i)
#    plt.savefig("laser", dpi=300)

```



1.4 Train the deep neural network

1.4.1 Use S1, S2 descriptors for training

In [51]:
```
1 df = S1
```

Contents

In [73]:

```
## The basic framework of neural network
X = df.iloc[:,2:].values
Y = df.iloc[:,1].values
X_train,X_test,y_train,y_test =train_test_split(X,Y,test_size=0.25)

sc = StandardScaler()
X_transform = sc.fit_transform(X_train)
sc2 = StandardScaler()
# y scale to [0,1]

mms = MinMaxScaler()
Y_transform = mms.fit_transform(y_train.reshape(-1, 1) )

class ProgressBar(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    self.draw_progress_bar(epoch + 1, EPOCHS)

  def draw_progress_bar(self, cur, total, bar_len=50):
    cur_len = int(cur / total * bar_len)
    sys.stdout.write("\r")
    sys.stdout.write("[{:<{}}] {}/{}".format("=" * cur_len, bar_len, cur, total))
    sys.stdout.flush()

def plot_history(history):
  hist = pd.DataFrame(history.history)
  hist['epoch'] = history.epoch
  plt.figure()
  plt.xlabel('epoch')
  plt.ylabel('metric - MSE')
  plt.plot(hist['epoch'], hist['mse'], label='tr')
  plt.plot(hist['epoch'], hist['val_mse'], label = 'va')
  plt.grid(True)
  plt.legend()

  plt.figure()
  plt.xlabel('epoch')
  plt.ylabel('metric - MAE')
  plt.plot(hist['epoch'], hist['mae'], label='tr')
  plt.plot(hist['epoch'], hist['val_mae'], label = 'va')
  plt.grid(True)
  plt.legend()
```

Contents


```
42
43  logdir = '.\callbacks'
44  output_model_file = os.path.join(logdir,"study.h5")
45  tensorboard =  keras.callbacks.TensorBoard(logdir)
46  early_stop = keras.callbacks.EarlyStopping(monitor='val_loss',min_delta=1e-5,patience=50)
47
48  callbacks=[ProgressBar()
49             ,early_stop]
50
51  EPOCHS = 50000
52
53  model = tf.keras.Sequential()
54  model.add(tf.keras.layers.Flatten(input_shape=(X.shape[1],)) )
55
56  model.add(tf.keras.layers.Dense(128, activation="relu") )
57  model.add(tf.keras.layers.Dense(256, activation="relu") )
58  model.add(tf.keras.layers.Dense(128, activation="relu") )
59  model.add(tf.keras.layers.Dense(128, activation="relu") )
60  model.add(tf.keras.layers.Dense(1, activation="relu") )
61  model.compile(optimizer=tf.keras.optimizers.Adam(0.0001)
62              ,loss = 'mse'
63              ,metrics=[['mae', 'mse']]
64              )
```

In [74]:
```
1 model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_9 (Flatten)          (None, 118)               0
_________________________________________________________________
dense_39 (Dense)             (None, 128)               15232
_________________________________________________________________
dense_40 (Dense)             (None, 256)               33024
_________________________________________________________________
dense_41 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_42 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_43 (Dense)             (None, 1)                 129
=================================================================
Total params: 97,793
Trainable params: 97,793
Non-trainable params: 0
_________________________________________________________________
```

Contents

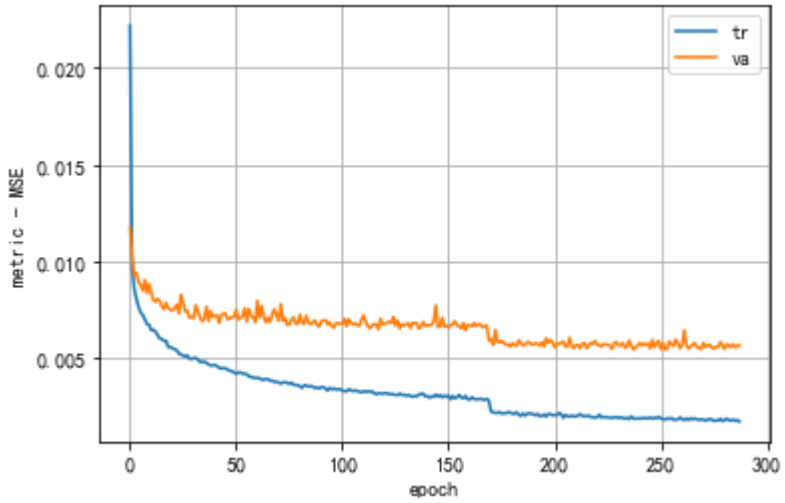
In [75]:

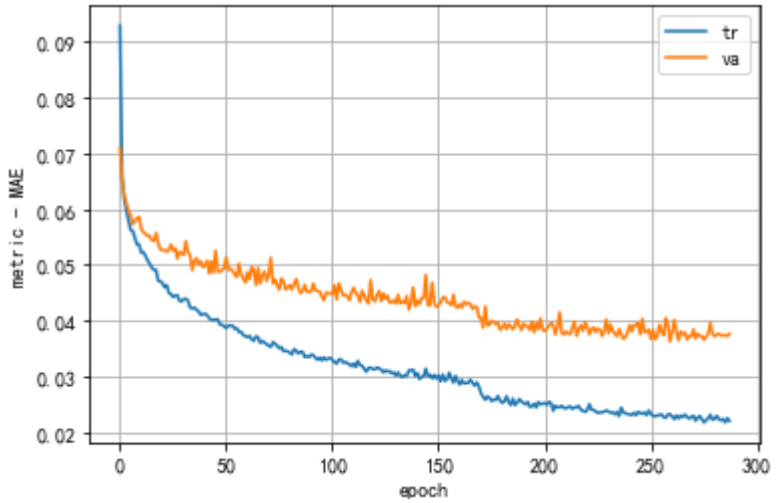
```
1 history = model.fit(
2   X_transform,Y_transform,
3   epochs=EPOCHS, validation_split = 0.2, verbose=1,
4   batch_size= 32, validation_freq= 1 ,callbacks=callbacks)
```

```
ae: 0.0375 - val_mse: 0.0057
Epoch 283/50000
7404/7404 [==============================] - 0s 66us/sample - loss: 0.0018 - mae: 0.0222 - mse: 0.0018 -
ae: 0.0377 - val_mse: 0.0056
Epoch 284/50000
7404/7404 [==============================] - 0s 67us/sample - loss: 0.0017 - mae: 0.0222 - mse: 0.0017 -
ae: 0.0373 - val_mse: 0.0055
Epoch 285/50000
7404/7404 [==============================] - 1s 68us/sample - loss: 0.0018 - mae: 0.0223 - mse: 0.0018 -
ae: 0.0374 - val_mse: 0.0057
Epoch 286/50000
7404/7404 [==============================] - 1s 71us/sample - loss: 0.0017 - mae: 0.0217 - mse: 0.0017 -
ae: 0.0374 - val_mse: 0.0055
Epoch 287/50000
7404/7404 [==============================] - 1s 68us/sample - loss: 0.0018 - mae: 0.0225 - mse: 0.0018 -
ae: 0.0372 - val_mse: 0.0056
Epoch 288/50000
7404/7404 [==============================] - 1s 72us/sample - loss: 0.0017 - mae: 0.0220 - mse: 0.0017 -
ae: 0.0376 - val_mse: 0.0056
```

Contents

In [76]:
```
1 plot_history(history)
```





In [77]:

```
y_pred = model.predict(sc.transform(X_test))
y_pred= mms.inverse_transform(y_pred)
y_test = y_test.reshape(-1,1)
mae = y_pred-y_test
print("mae",np.abs(mae).reshape(1,-1).mean())

from matplotlib import pyplot as plt
plt.scatter(y_test,y_pred)
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
print("R2",r2_score(y_test,y_pred))
plt.title('DNN')
fig = plt.gcf( )
fig.set_size_inches(5.5, 10.5)

ll = np.linspace(0,130,1200)
ll_x = ll
ll_y = ll
plt.plot(ll_x,ll_y)
plt.show( )
```

```
mae 5.567640322635964
R2 0.8715308740832991
```

1.4.2 Parameter optimization (using additional library optunaas for a longer time)

```
import pyforest
import warnings
warnings.filterwarnings("ignore")
import pathlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import os


df = pd.read_csv("S1.csv")

print(df.shape)

from sklearn.model_selection import train_test_split
X = df.iloc[:,2:].values
Y = df.iloc[:,1].values


X_train,X_test,y_train,y_test =train_test_split(X,Y,test_size=0.2)


from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_transform = sc.fit_transform(X_train)
sc2 = StandardScaler()
# y scale to [0,1]
from sklearn.preprocessing import MinMaxScaler
#mms = MinMaxScaler()
#Y_transform = mms.fit_transform(y_train.reshape(-1, 1) )
Y_transform = y_train

Y_transform.shape

```

Contents


```
39
40  import sys
41  class ProgressBar(keras.callbacks.Callback):
42    def on_epoch_end(self, epoch, logs):
43
44        self.draw_progress_bar(epoch + 1, EPOCHS)
45
46    def draw_progress_bar(self, cur, total, bar_len=50):
47        cur_len = int(cur / total * bar_len)
48        sys.stdout.write("\r")
49        sys.stdout.write("[{:<{}}] {}/{}".format("=" * cur_len, bar_len, cur, total))
50        sys.stdout.flush()
51
52  early_stop = keras.callbacks.EarlyStopping(monitor='val_loss',min_delta=1e-6,patience=50)
53  callbacks=[ProgressBar() ,early_stop]
54
55  EPOCHS = 10000
56
57
58  def objective(trial):
59        kwargs = {}
60        jilu = []
61        # Categorical parameter
62        optimizer = trial.suggest_categorical('optimizer', ["RMSprop", "Adam", "SGD"])
63        if optimizer == "RMSprop":
64              kwargs["learning_rate"] = trial.suggest_float(
65                    "rmsprop_learning_rate", 1e-5, 1e-1, log=True
66              )
67              kwargs["decay"] = trial.suggest_float("rmsprop_decay", 0.85, 0.99)
68              kwargs["momentum"] = trial.suggest_float("rmsprop_momentum", 1e-5, 1e-1, log=True)
69        elif optimizer == "Adam":
70              kwargs["learning_rate"] = trial.suggest_float("adam_learning_rate", 1e-5, 1e-1, log=True)
71        elif optimizer == "SGD":
72              kwargs["learning_rate"] = trial.suggest_float(
73                    "sgd_opt_learning_rate", 1e-5, 1e-1, log=True
74              )
75              kwargs["momentum"] = trial.suggest_float("sgd_opt_momentum", 1e-5, 1e-1, log=True)
76
77        # Int parameter
78        sj_n = [1, 2, 3, 4, 5, 6, 7,8,9,10,11,12,13,14,15,16]
79        sj_n[0] = trial.suggest_int('sj_n[0]', 32, 256)
80        sj_n[1] = trial.suggest_int('sj_n[1]', 32, 256)
```

Contents


```
    sj_n[2] = trial.suggest_int('sj_n[2]', 32, 256)
    sj_n[3] = trial.suggest_int('sj_n[3]', 32, 256)
    sj_n[4] = trial.suggest_int('sj_n[4]', 32, 256)
    sj_n[5] = trial.suggest_int('sj_n[5]', 32, 256)
    sj_n[6] = trial.suggest_int('sj_n[6]', 32, 256)
    sj_n[7] = trial.suggest_int('sj_n[7]', 32, 256)
    sj_n[8] = trial.suggest_int('sj_n[8]', 32, 256)
    sj_n[9] = trial.suggest_int('sj_n[9]', 32, 256)
    sj_n[10] = trial.suggest_int('sj_n[10]', 32, 256)
    sj_n[11] = trial.suggest_int('sj_n[11]', 32, 256)
    sj_n[12] = trial.suggest_int('sj_n[12]', 32, 256)
    sj_n[13] = trial.suggest_int('sj_n[13]', 32, 256)
    sj_n[14] = trial.suggest_int('sj_n[14]', 32, 256)
    sj_n[15] = trial.suggest_int('sj_n[15]', 32, 256)
```

```
    # Uniform parameter
    dropout_rate = trial.suggest_uniform('dropout_rate', 0.0, 0.9)

    # Loguniform parameter
    #     learning_rate = trial.suggest_loguniform('learning_rate', 1e-5, 1e-2)

    num_layers = trial.suggest_int('num_layers', 3, 15)
    model = tf.keras.Sequential()
    model.add(layers.Flatten(input_shape=(X.shape[1],)))
    test_n = [1, 2, 3, 4, 5, 6, 7,8,9,10,11,12,13,14,15,16]
    test_n[0] = trial.suggest_int('test_n[0]', 1, 2)
    test_n[1] = trial.suggest_int('test_n[1]', 1, 2)
    test_n[2] = trial.suggest_int('test_n[2]', 1, 2)
    test_n[3] = trial.suggest_int('test_n[3]', 1, 2)
    test_n[4] = trial.suggest_int('test_n[4]', 1, 2)
    test_n[5] = trial.suggest_int('test_n[5]', 1, 2)
    test_n[6] = trial.suggest_int('test_n[6]', 1, 2)
    test_n[7] = trial.suggest_int('test_n[7]', 1, 2)
    test_n[8] = trial.suggest_int('test_n[8]', 1, 2)
    test_n[9] = trial.suggest_int('test_n[9]', 1, 2)
    test_n[10] = trial.suggest_int('test_n[10]', 1, 2)
    test_n[11] = trial.suggest_int('test_n[11]', 1, 2)
```

Contents


```
    test_n[12] = trial.suggest_int('test_n[12]', 1, 2)
    test_n[13] = trial.suggest_int('test_n[13]', 1, 2)
    test_n[14] = trial.suggest_int('test_n[14]', 1, 2)
    test_n[15] = trial.suggest_int('test_n[15]', 1, 2)


    for i in range(num_layers):
          model.add(layers.Dense(sj_n[i], activation='relu'))
          if test_n[i] == 1:
                model.add(layers.BatchNormalization())
                model.add(layers.Dropout(dropout_rate))

    model.add(layers.Dense(1, activation='relu'))

    model.compile(optimizer=optimizer, loss='mse', metrics=[['mae', 'mse']])
    history = model.fit(X_transform, Y_transform, epochs=EPOCHS, validation_split=0.2, verbose=0, b
                                  validation_freq=1, callbacks=callbacks)
    y_pred = model.predict(sc.transform(X_test))
    global y_test

    y_test = y_test.reshape(-1, 1)

    mae = y_pred - y_test


    import numpy as np
    from sklearn.metrics import r2_score
    global r2
    r2 = r2_score(y_test, y_pred)

    MAE = np.abs(mae).reshape(1, -1).mean()
    import time

    ts = str(time.time())
    saved_model_path = "./saved999_models/{}".format(str(r2)) + "R2" +"|||||||MAE"+ str(
          np.abs(mae).reshape(1, -1).mean()) + "TM" + ts
    if r2 >= 0.91:
          tf.keras.experimental.export_saved_model(model, saved_model_path)
          model.save('optmodel.h5')

    print("R2",r2)
    print("mae",str(np.abs(mae).reshape(1, -1).mean()))
```

```
    return r2


import optuna
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=10000)
print(study.best_params)
print(study.best_value)
```

1.4.3 The trained model, take DNN under S1 descriptor as an example

In [78]:

```
save_model = tf.keras.experimental.load_from_saved_model("./S1model")
#save_model.predict(X).shape
save_model.summary()
```

```
Model: "sequential_418"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_418 (Flatten)        (None, 118)               0
_________________________________________________________________
dense_2105 (Dense)           (None, 251)               29869
_________________________________________________________________
dense_2106 (Dense)           (None, 153)               38556
_________________________________________________________________
dense_2107 (Dense)           (None, 239)               36806
_________________________________________________________________
dense_2108 (Dense)           (None, 124)               29760
_________________________________________________________________
dense_2109 (Dense)           (None, 1)                 125
=================================================================
Total params: 135,116
Trainable params: 135,116
Non-trainable params: 0
_________________________________________________________________
```

1.4.4 Residual analysis

Contents

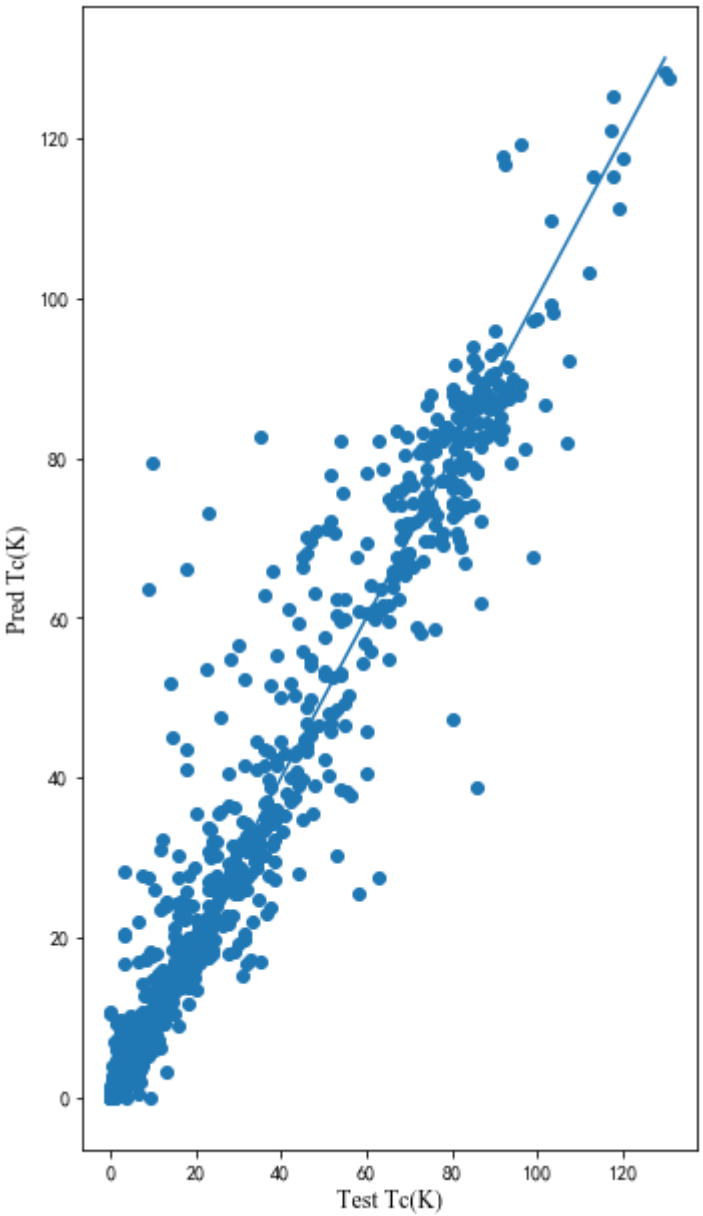
In [79]:

```
## take S2model for a example
model = tf.keras.experimental.load_from_saved_model("./S2model")
df =S2
from sklearn.model_selection import train_test_split
X = df.iloc[:,2:].values
Y = df.iloc[:,1].values
X_train,X_test,y_train,y_test =train_test_split(X,Y,test_size=0.1)


from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_transform = sc.fit_transform(X_train)
sc2 = StandardScaler()

from sklearn.preprocessing import MinMaxScaler

y_pred = model.predict(sc.transform(X_test))

from matplotlib import pyplot as plt
plt.scatter(y_test,y_pred)
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
print(r2_score(y_test,y_pred))
from sklearn.metrics import mean_squared_error, mean_absolute_error


np.sqrt(mean_squared_error(y_test,y_pred))


print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))


mean_absolute_error(y_test,y_pred)

print("MAE",mean_absolute_error(y_test,y_pred))

font2 = {'family' : 'Times New Roman',
    'weight' : 'normal',
    'size' : 12,
                }
#plt.title('Deeplearning')
```

```
fig = plt.gcf( )
fig.set_size_inches(5.5, 10.5)
plt.xlabel("Test Tc(K)",font2)

plt.ylabel("Pred Tc(K)",font2)

ll = np.linspace(0,130,1200)
ll_x = ll
ll_y = ll
plt.plot(ll_x,ll_y)
#plt.savefig('DNN回归.jpg', dpi=300)
plt.show( )
```

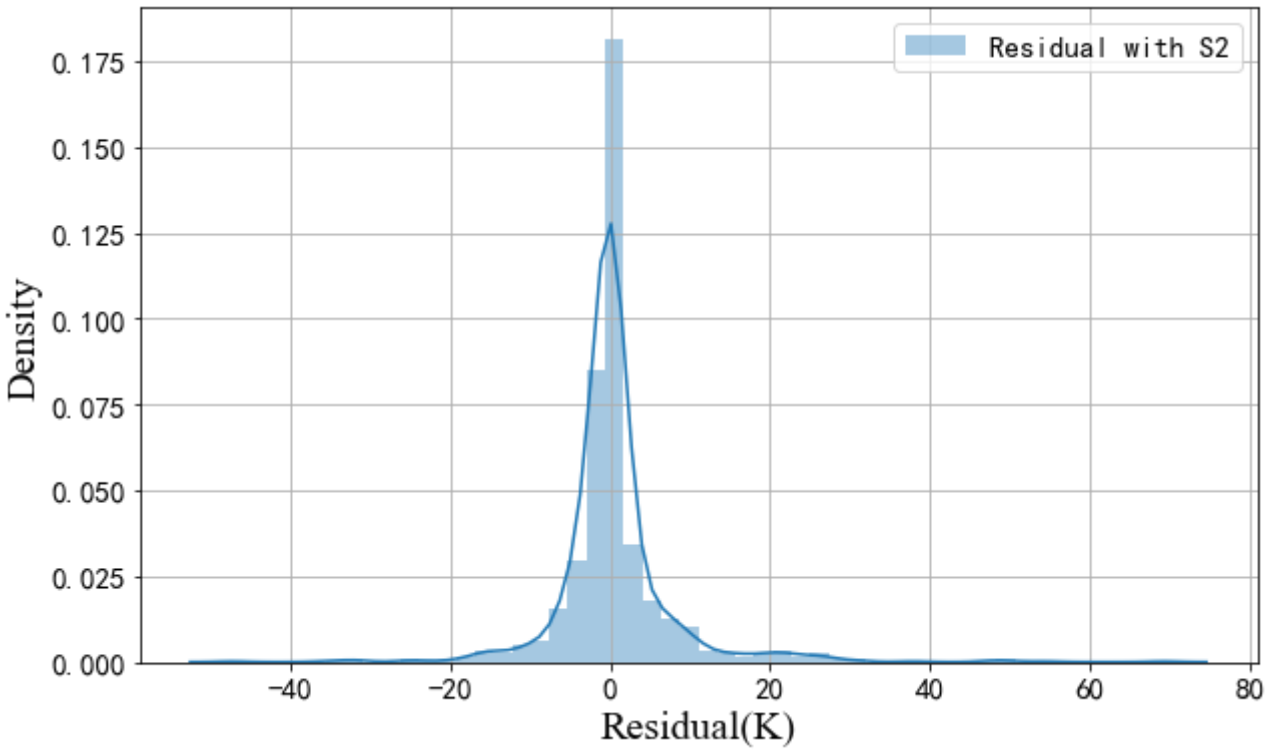
```
0.9361781128924711
RMSE 7.345803863368909
MAE 3.731645200334191
```

Contents



In [80]:

```
error = y_pred.T - y_test.T
error = pd.DataFrame(error)
fig = plt.figure(figsize = (10,6))
#ax1 = fig.add_subplot(2,1,1)  # 创建子图1
#ax1.scatter(s.index, s.values)
#plt.grid()
# 绘制数据分布图
font3 = {'family' : 'Times New Roman',
    'weight' : 'normal',
    'size' : 20,
                }
ax = fig.add_subplot(1,1,1)  # 创建子图2
#s.hist(bins=30,alpha = 0.5,ax = ax)
sns.distplot(error,ax = ax,bins = 50)
plt.legend(["Residual with S2"],fontsize =15)
plt.xlabel('Residual(K)',font3)
plt.ylabel('Density',font3)
ax.tick_params(axis='y',labelsize=15)
ax.tick_params(axis='x',labelsize=15)
#plt.title('Pred of VH')
#plt.savefig('误差分析.jpg', dpi=300)
plt.grid()
#plt.savefig("S2 residual", dpi=300,bbox_inches ="tight")
```



1.5 Virtual high throughput Settings

1.5.1 Virtual high-throughput sample generation (the following code is virtual sample distribution in HgCaBaPbCuO compounds)

```
### It should be noted that the sample distribution generated by virtual high flux conforms to the d
of ###    training data as much as possible
vh = []
import numpy
import pandas as pd
x = numpy.arange(1,300,6)
x= x/1000
x = list(x)
print(x)
for i in x:
```

```
    print(i)
    vh = []
    data =[ {'Hg':[[0,0.26,0.01]]}, {'Pb':[[0.09, 0.34, 0.01]]}, {'Ba':[[i, i, i]]}, {'Ca':[[0, 0.2
0.152]]}, {'Cu':[[0.1,0.6,0.01]]}, {'O':[[1,1,1]] }]
    def findall(pos,sum,outcome):
        if pos==len(data)-1:
            if 1/sum >=0.7 and 1/sum<=2.4:
                outcome+="O"
                print(outcome)
                vh.append(outcome)
        #     <--------------------------------output
        else:
                for (key,val) in data[pos].items():
                    for vl in val:
                        st = vl[0]
                        if st==0:
                            st=st+vl[2]
                        en = vl[1]
                        bu = vl[2]
                        while(st<=en):
                            findall(pos+1,sum+round(st,6), outcome+str(key)+str(round(s
                            st+=bu
findall(0, 0,'')

test=pd.DataFrame(vh)
print(test)
test.to_csv('./'+str(i)+'Ba.csv',encoding='gbk',index=0,header=0)
```

1.5.2 DNN prediction of virtual samples

```
see it in DNN-VH.ipynb
```

1.6 Deep neural network combined with manifold learning

In [83]:

```python
import tensorflow as tf
saved_model_path = "path_to\modell"
new_model = tf.keras.experimental.load_from_saved_model(saved_model_path)
new_model.summary()
model = new_model
a = []
for i in model.get_weights():

    a.append(i[0])

from tensorflow.keras import Sequential
model2 = tf.keras.Sequential()
print(len(model.layers))
print(len(model.layers[:-1]))
for layer in model.layers[:-1]:
    model2.add(layer)

model2.summary()
d = []
for i in model2.get_weights():
    d.append(i[0])
print("-----------------")
```

```
Model: "sequential_418"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_418 (Flatten)        (None, 118)               0
_________________________________________________________________
dense_2105 (Dense)           (None, 251)               29869
_________________________________________________________________
dense_2106 (Dense)           (None, 153)               38556
_________________________________________________________________
dense_2107 (Dense)           (None, 239)               36806
_________________________________________________________________
dense_2108 (Dense)           (None, 124)               29760
_________________________________________________________________
dense_2109 (Dense)           (None, 1)                 125
=================================================================
Total params: 135,116
```

```
Trainable params: 135,116
Non-trainable params: 0
_________________________________________________________________
6
5
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_418 (Flatten)        (None, 118)               0
_________________________________________________________________
dense_2105 (Dense)           (None, 251)               29869
_________________________________________________________________
dense_2106 (Dense)           (None, 153)               38556
_________________________________________________________________
dense_2107 (Dense)           (None, 239)               36806
_________________________________________________________________
dense_2108 (Dense)           (None, 124)               29760
=================================================================
Total params: 134,991
Trainable params: 134,991
Non-trainable params: 0
_________________________________________________________________
_________________
```

Contents

In [84]:

```
df = pd.read_csv("S1.csv")
df
```

Out[84]:

	name	Tc	H	He	Li	Be	B	C	N	O	F	Ne	Na	Mg	Al	Si
0	Zr74.1Ti3.9Ni22	3.650	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
1	Zr70.2Ti7.8Ni22	3.320	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
2	Zr66.3Ti11.7Ni22	3.560	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
3	Zr62.4Ti15.6Ni22	3.215	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
4	Zr55Cu30Al10Ni5	1.350	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.100	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12335	Ag0.05Rh0.04Ti0.91	1.950	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
12336	Ag0.03Tl0.97	2.670	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
12337	Ag0.035Cd0.01Sn0.955	3.650	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
12338	Ag0.005Zn0.995	0.763	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
12339	Ag0.002Al0.998	1.128	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.998	0

12340 rows × 120 columns

In [86]:

```
y_pred = model2.predict(sc.transform(X))
len(y_pred)
```

Out[86]: 12340

In [87]:

```
y_pred = pd.DataFrame(y_pred)
newdf = pd.concat([df,y_pred],axis=1)
newdf
```

Out[87]:

	name	Tc	H	He	Li	Be	B	C	N	O	F	Ne	Na	Mg	Al	Si
0	Zr74.1Ti3.9Ni22	3.650	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
1	Zr70.2Ti7.8Ni22	3.320	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
2	Zr66.3Ti11.7Ni22	3.560	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
3	Zr62.4Ti15.6Ni22	3.215	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
4	Zr55Cu30Al10Ni5	1.350	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.100	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12335	Ag0.05Rh0.04Ti0.91	1.950	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
12336	Ag0.03Tl0.97	2.670	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
12337	Ag0.035Cd0.01Sn0.955	3.650	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
12338	Ag0.005Zn0.995	0.763	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.000	0
12339	Ag0.002Al0.998	1.128	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0.0	0.0	0.998	0

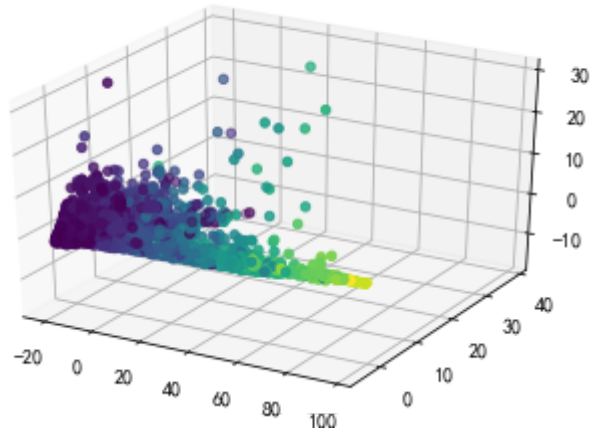
12340 rows × 244 columns

1.6.1 PCA

Contents

In [88]:

```
from sklearn.decomposition import PCA
ax = plt.axes(projection='3d')
x = y_pred
pca = PCA(n_components=3)
pca.fit(x)
x_pca = pca.transform(x)
newdf = np.array(newdf)
close = newdf[:,1]
volume = newdf[:,1]
import seaborn as sns
from matplotlib import pyplot as plt
plt.figure(figsize=(20,10))
#plt.scatter(x_pca[:, 0], x_pca[:, 1], alpha=0.2 ,c =close)
ax.scatter3D(x_pca[:, 0], x_pca[:, 1], x_pca[:, 2], c=close)
plt.show()
plt.show()
```



<Figure size 1440x720 with 0 Axes>

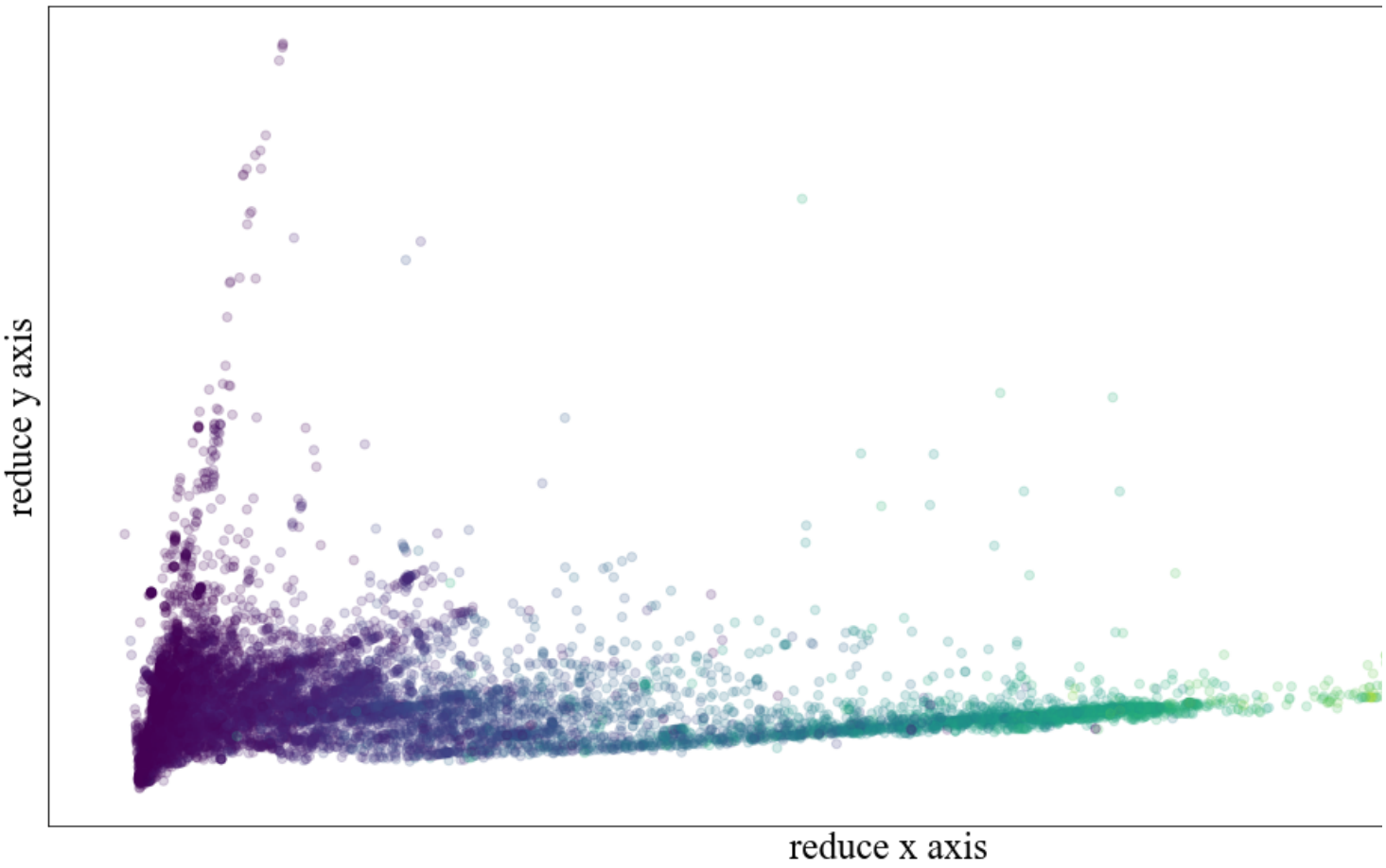
Contents

Contents

In [121]:

```
font4 = {'family' : 'Times New Roman',
    'weight' : 'normal',
    'size' : 25,
                }
from sklearn.decomposition import PCA
#ax = plt.axes(projection='3d')
x = y_pred
pca = PCA(n_components=2)
pca.fit(x)
x_pca = pca.transform(x)
newdf = np.array(newdf)
close = newdf[:,1]
volume = newdf[:,1]
plt.figure(figsize=(20,10))
plt.scatter(x_pca[:, 0], x_pca[:, 1], alpha=0.2 ,c =close,label="PCA")
#ax.scatter3D(x_pca[:, 0], x_pca[:, 1], x_pca[:, 2], c=close)
plt.xlabel('reduce x axis',font4)
plt.ylabel('reduce y axis',font4)
plt.legend(loc='upper right',fontsize= 25)
plt.xticks([])
plt.yticks([])
plt.savefig("reallPCA.png",dpi=300,bbox_inches="tight")
plt.show()
```

Contents




```
for i in range(2,120,1):
    newdf = np.array(newdf)
    close = newdf[:,i]
    print("elem Number",i-1)

    import seaborn as sns
    from matplotlib import pyplot as plt
    plt.figure(figsize=(10,5))
    plt.scatter(x_pca[:, 0], x_pca[:, 1], alpha=0.2 ,c =close )
    plt.show()
```

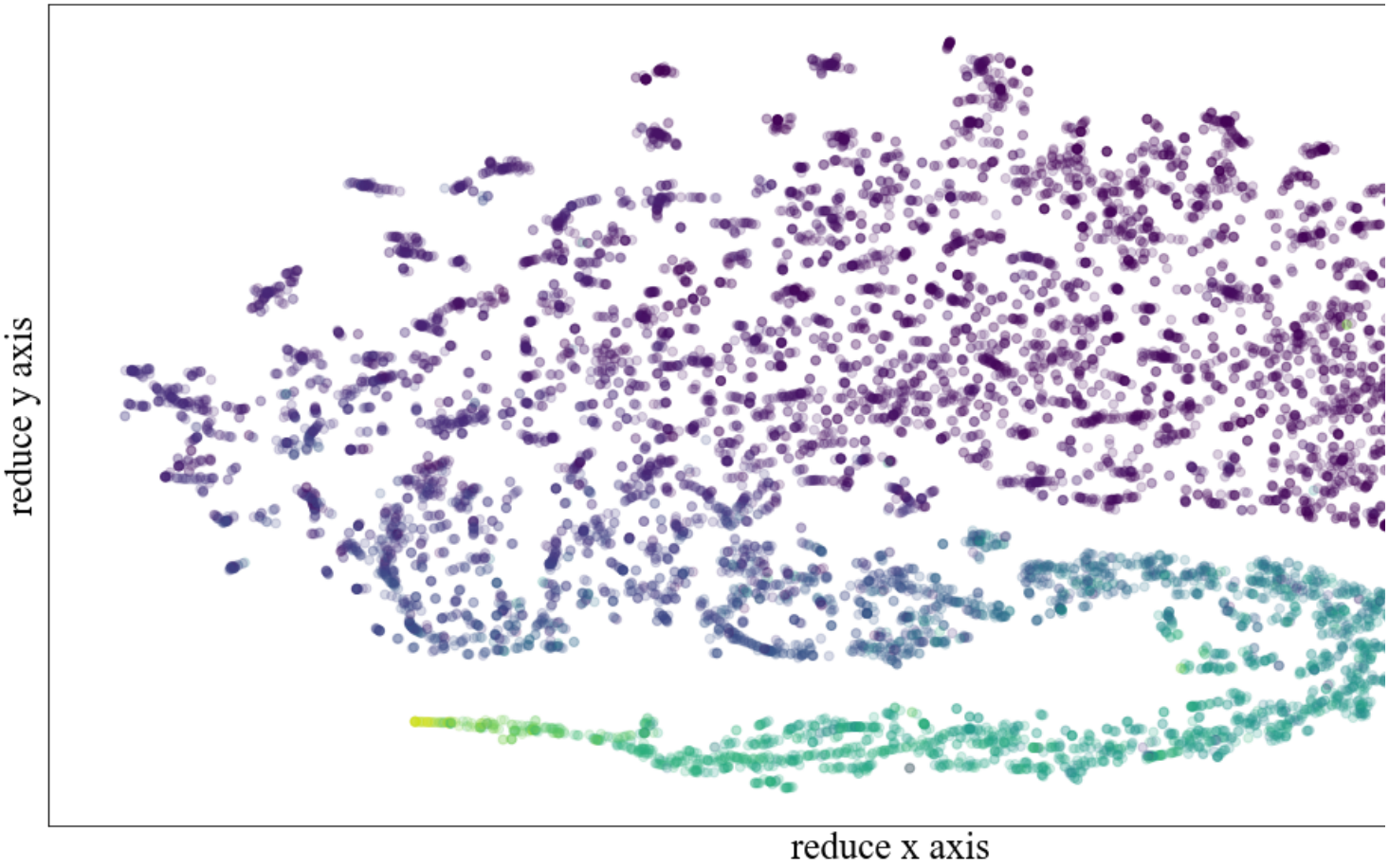
1.6.2 T-SNE

Contents

Contents

In [127]:

```
from sklearn.manifold import TSNE
#ax = plt.axes(projection='3d')
x = y_pred
X_embedded = TSNE(n_components=2).fit_transform(x)
#X_embedded
plt.figure(figsize=(20,10))
plt.scatter(X_embedded[:, 0], X_embedded[:, 1], alpha=0.2 ,c =close,label = "t-SNE")
#ax.scatter3D(X_embedded[:, 0], X_embedded[:, 1], X_embedded[:, 2], c=close)
plt.xlabel('reduce x axis',font4)
plt.ylabel('reduce y axis',font4)
plt.legend(loc='upper right',fontsize= 25)
plt.xticks([])
plt.yticks([])
plt.savefig("reallT-SNE.png",dpi=300,bbox_inches="tight")
plt.show()
plt.show()
```

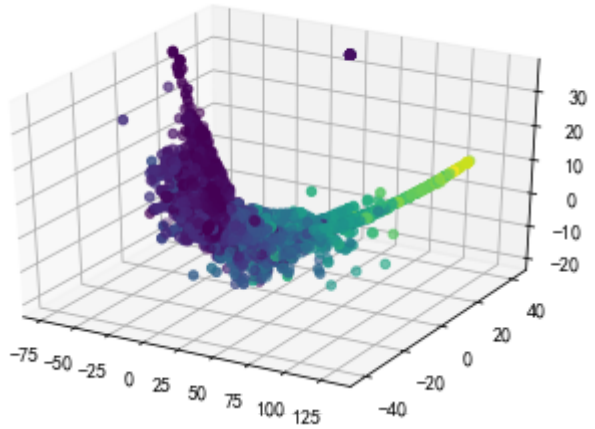


1.6.3 Isomap

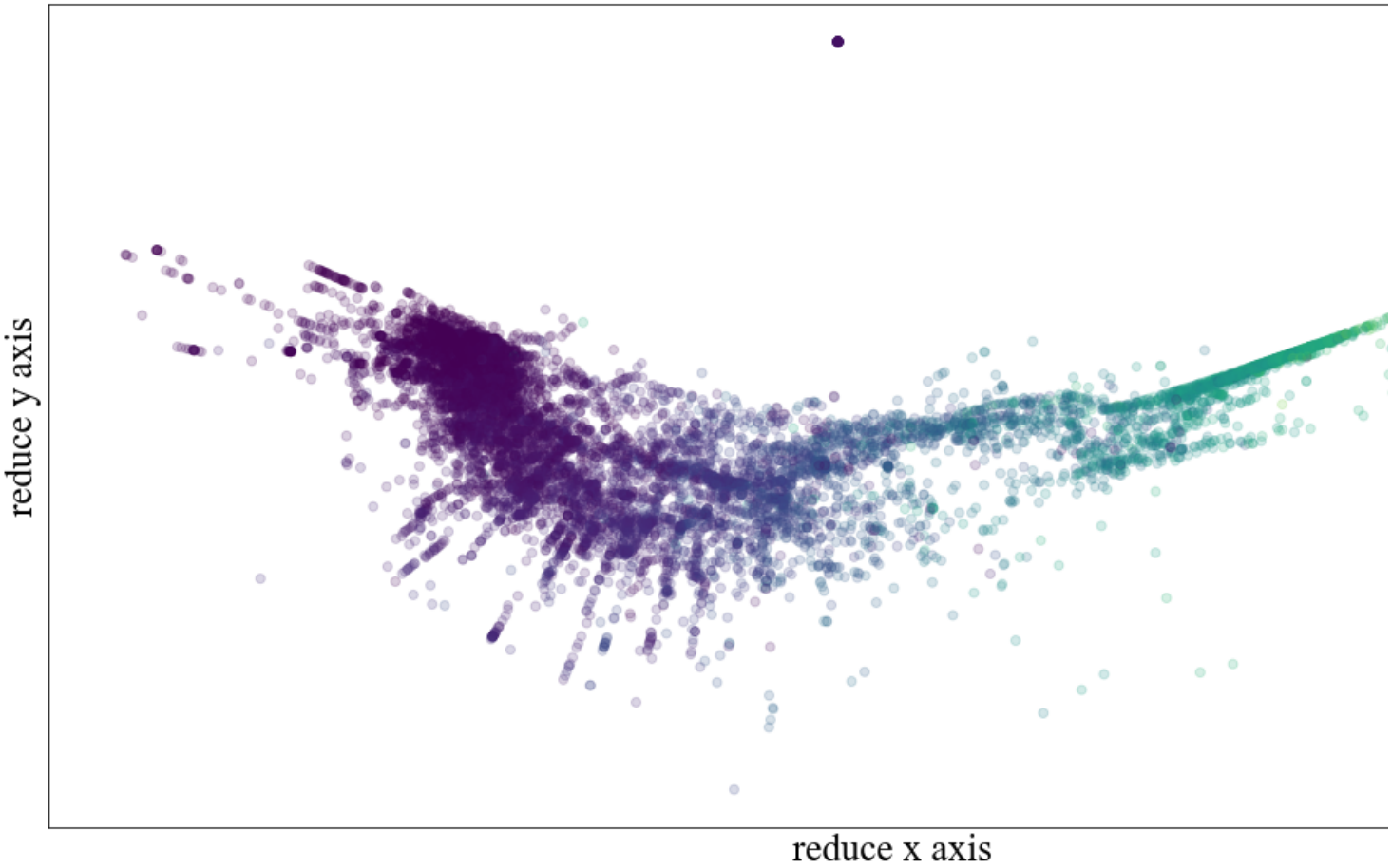
Contents

In [129]:

```
from sklearn import manifold
ax = plt.axes(projection='3d')
x = y_pred
X_embedded =  manifold.Isomap(n_components=3).fit_transform(x)
import seaborn as sns
from matplotlib import pyplot as plt
plt.figure(figsize=(20,10))
plt.scatter(X_embedded[:, 0], X_embedded[:, 1], alpha=0.2 ,c =close,label="Isomap")
ax.scatter3D(X_embedded[:, 0], X_embedded[:, 1], X_embedded[:, 2], c=close)
plt.xlabel('reduce x axis',font4)
plt.ylabel('reduce y axis',font4)
plt.legend(loc='upper right',fontsize= 25)
plt.xticks([])
plt.yticks([])
plt.savefig("reallIsomap.png",dpi=300,bbox_inches="tight")
plt.show()
```



Contents

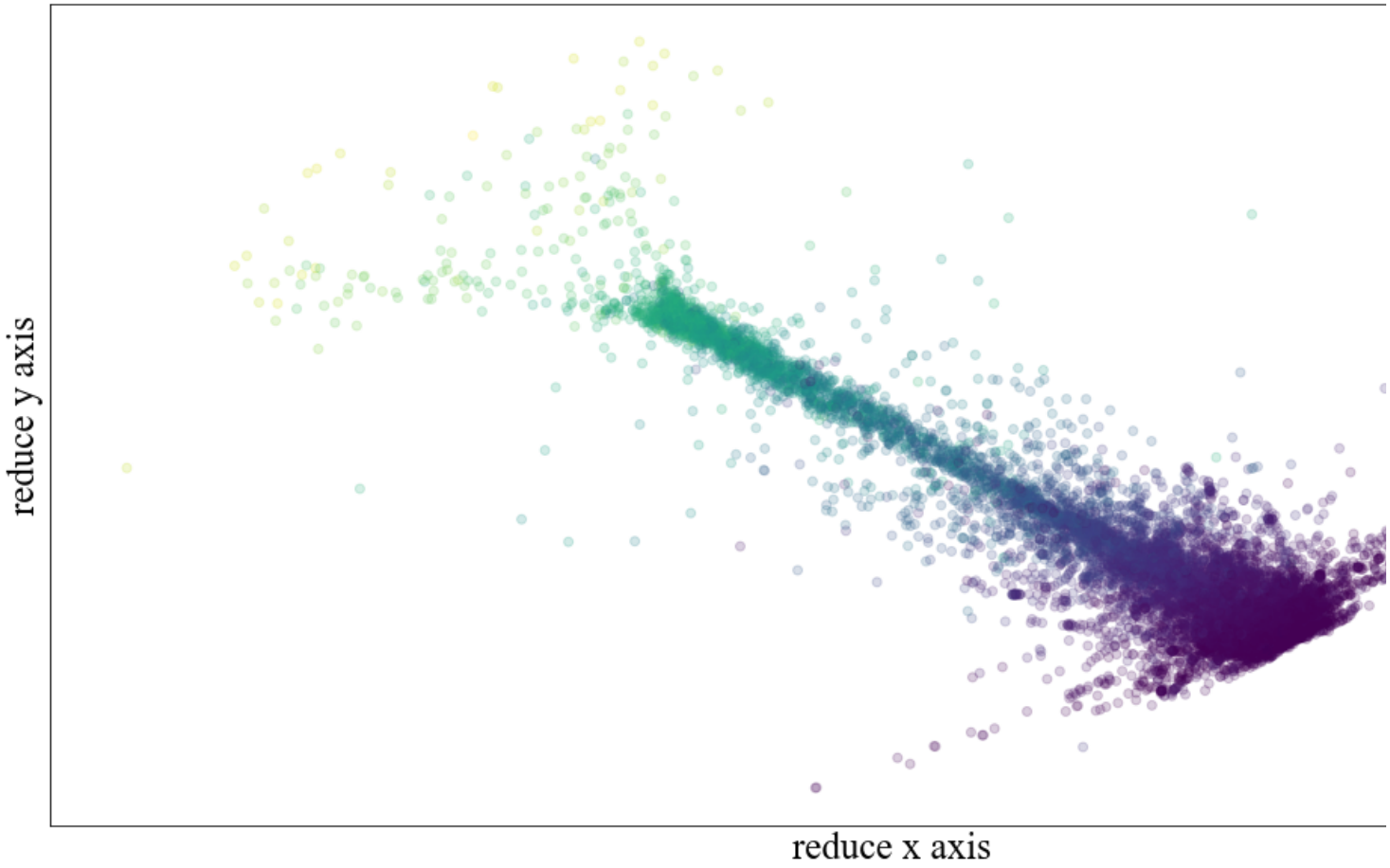


1.6.4 MDS

Contents

In [130]:

```
from sklearn import manifold

x = y_pred
X_embedded =  manifold.MDS(n_components = 2, max_iter=100, n_init=1).fit_transform(x)
plt.figure(figsize=(20,10))
plt.scatter(X_embedded[:, 0], X_embedded[:, 1], alpha=0.2 ,c =close,label="MDS")
plt.xlabel('reduce x axis',font4)
plt.ylabel('reduce y axis',font4)
plt.legend(loc='upper right',fontsize= 25)
plt.xticks([])
plt.yticks([])
plt.savefig("reallMDS.png",dpi=300,bbox_inches="tight")
plt.show()
```



END

Contents