Table of Contents

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from pyecharts import Pie
import missingno as msno
from sklearn import metrics as mt

from attr import *
from plot_comfusion_matrix import *
```

数据预处理

导入数据

```
df = pd.read_excel('data_cn.xlsx')
df.loc[df.label ==2,'label'] = 0
df.head()
```

	Status_of_existing_checking_account	Duration_in_month	Credit_hi
0	A11	6	A34
1	A12	48	A32
2	A14	12	A34
3	A11	42	A32
4	A11	24	A33

5 rows × 21 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
Status_of_existing_checking_account                         1000 non-nul
Duration_in_month                                           1000 non-nul
Credit_history                                              1000 non-nul
Purpose                                                     1000 non-nul
Credit_amount                                               1000 non-nul
Savings_account                                             1000 non-nul
Present_employment_since                                    1000 non-nul
Installment_rate_in_percentage_of_disposable_income         1000 non-nul
Personal_status_and_sex                                     1000 non-nul
guarantors                                                  1000 non-nul
Present_residence_since                                     1000 non-nul
Property                                                    1000 non-nul
Age                                                         1000 non-nul
Other_installment_plans                                     1000 non-nul
Housing                                                     1000 non-nul
Number_of_existing_credits_at_this_bank                     1000 non-nul
Job                                                         1000 non-nul
Number_of_people_being_liable_to_provide_maintenance_for    1000 non-nul
Telephone                                                   1000 non-nul
foreign_worker                                              1000 non-nul
label                                                       1000 non-nul
dtypes: int64(8), object(13)
memory usage: 164.1+ KB
```

将有序文本数据转换为数值数据

```
for key, value in maping.items():
    df[key].map(value)
df.head()
```

	Status_of_existing_checking_account	Duration_in_month	Credit_hi
0	A11	6	A34
1	A12	48	A32
2	A14	12	A34
3	A11	42	A32
4	A11	24	A33

5 rows × 21 columns

将无序文本数据用哑变量替换

```
from sklearn.preprocessing import OneHotEncoder
cols_orderless = df.select_dtypes(include=['object']).columns
for col in cols_orderless:
    df[col] = np.unique(df[col], return_inverse=True)[1]
    for i in range(pd.value_counts(df[col]).count()):
        df[col+'_'+str(i+1)] = OneHotEncoder().fit_transform(df[col].value
    df.drop(col, axis=1 ,inplace=True)
df.head()
```

	Duration_in_month	Credit_amount	Installment_rate_in_percentage_
0	6	1169	4
1	48	5951	2
2	12	2096	2
3	42	7882	2
4	24	4870	3

5 rows × 62 columns

数据规范化

```
from sklearn.preprocessing import scale
cols = df.drop('label', axis=1).columns
df[cols] = scale(df[cols])
df.head()
```

	Duration_in_month	Credit_amount	Installment_rate_in_percentage_
0	-1.236478	-0.745131	0.918477
1	2.248194	0.949817	-0.870183
2	-0.738668	-0.416562	-0.870183
3	1.750384	1.634247	-0.870183
4	0.256953	0.566664	0.024147

5 rows × 62 columns

特征工程

RFE筛选特征

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
X_val = df.drop('label', axis=1)
y_val = df['label']
rfe = RFE(estimator=LogisticRegression(), n_features_to_select=30).fit(X_
```

查看被筛选掉的特征

```
X_val.columns[rfe.support_ == False]
```

```
Index(['Present_residence_since',
       'Number_of_people_being_liable_to_provide_maintenance_for',
       'Status_of_existing_checking_account_3', 'Credit_history_4',
       'Purpose_3', 'Purpose_4', 'Purpose_6', 'Purpose_7', 'Purpose_10',
       'Savings_account_2', 'Savings_account_3', 'Present_employment_sinc
       'Present_employment_since_2', 'Present_employment_since_3',
       'Present_employment_since_5', 'Personal_status_and_sex_1',
       'Personal_status_and_sex_2', 'Personal_status_and_sex_4',
       'guarantors_1', 'guarantors_2', 'Property_2', 'Property_3',
       'Other_installment_plans_1', 'Other_installment_plans_2', 'Housing_
       'Housing_3', 'Job_1', 'Job_2', 'Job_3', 'Job_4', 'Telephone_2'],
      dtype='object')
```

保留未被筛选掉的特征

```
X_val = X_val.iloc[:, rfe.support_]
```

```
X_val.head()
```

	Duration_in_month	Credit_amount	Installment_rate_in_percentage_
0	-1.236478	-0.745131	0.918477
1	2.248194	0.949817	-0.870183
2	-0.738668	-0.416562	-0.870183
3	1.750384	1.634247	-0.870183

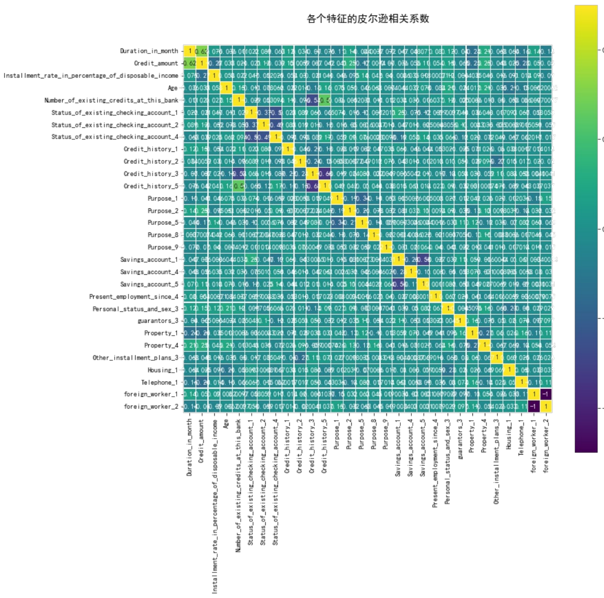
4	0.256953	0.566664	0.024147

5 rows × 30 columns

相关性分析

```
plt.figure(figsize=(12,12))
plt.title('各个特征的皮尔逊相关系数', y=1.05, size=15)
sns.heatmap(X_val.corr(),linewidths=0.1,vmax=1.0, square=True, cmap=plt.c
```

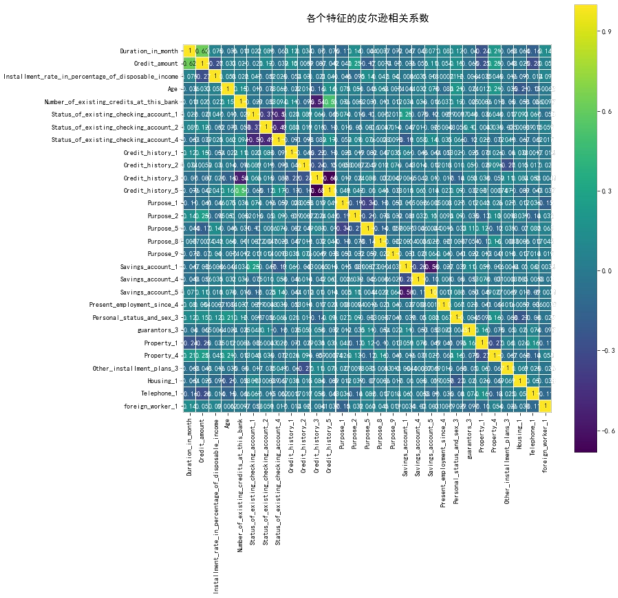
```
<matplotlib.axes._subplots.AxesSubplot at 0x25c36928ba8>
```




```
X_val.drop('foreign_worker_2', axis=1, inplace=True)
```

```
plt.figure(figsize=(12,12))
plt.title('各个特征的皮尔逊相关系数', y=1.05, size=15)
sns.heatmap(X_val.corr(),linewidths=0.1,vmax=1.0, square=True, cmap=plt.c
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x25c36c65630>
```



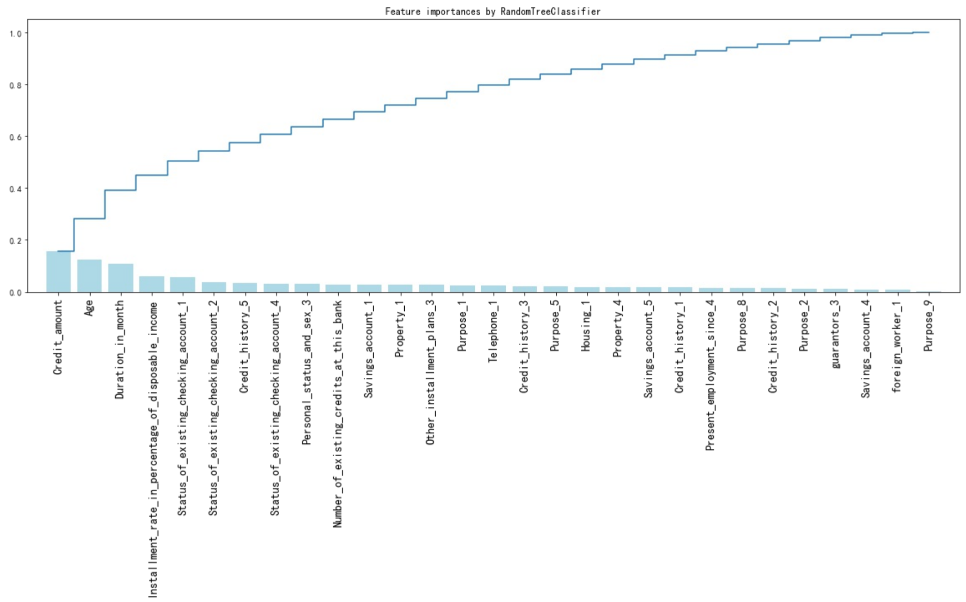
Embedded

```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators=20)   #构建分类随机森林分类器

clf.fit(X_val, y_val) #对自变量和因变量进行拟合
for i, score in enumerate(clf.feature_importances_):
    print(X_val.columns[i], '\t\t',score)
    if score < 0.01:
        print()
        print('不要的特征:', X_val.columns[i])
        print()
```

```
Duration_in_month    0.109518188865
Credit_amount    0.15750984933
Installment_rate_in_percentage_of_disposable_income    0.0589523531786
Age    0.123632471707
Number_of_existing_credits_at_this_bank    0.0289180343327
Status_of_existing_checking_account_1    0.0559697840993
Status_of_existing_checking_account_2    0.0363934048045
Status_of_existing_checking_account_4    0.0313215632037
Credit_history_1    0.0164216669871
Credit_history_2    0.0134495141705
Credit_history_3    0.0222631542981
Credit_history_5    0.0328904506053
Purpose_1    0.0256921560515
Purpose_2    0.0130205883787
Purpose_5    0.0221490882456
Purpose_8    0.0134919509788
Purpose_9    0.00320983204385

不要的特征: Purpose_9

Savings_account_1    0.0282526457623
Savings_account_4    0.00885029260625

不要的特征: Savings_account_4

Savings_account_5    0.0175097800863
Present_employment_since_4    0.0158101510065
Personal_status_and_sex_3    0.0291493110169
guarantors_3    0.0128600750858
Property_1    0.0270253720512
Property_4    0.0187056116726
Other_installment_plans_3    0.0265226746788
Housing_1    0.0192242043361
Telephone_1    0.0241806468915
foreign_worker_1    0.00710518352569

不要的特征: foreign_worker_1
```

可视化

```
## feature importances 可视化##
importances = clf.feature_importances_
feat_names = X_val.columns
indices = np.argsort(importances)[::-1]
fig = plt.figure(figsize=(20,6))
plt.title("Feature importances by RandomTreeClassifier")
plt.bar(range(len(indices)), importances[indices], color='lightblue',  al
plt.step(range(len(indices)), np.cumsum(importances[indices]), where='mic
plt.xticks(range(len(indices)), feat_names[indices], rotation='vertical',
plt.xlim([-1, len(indices)])
plt.show()
```




```
cols_filted = ['Purpose_9', 'Savings_account_4', 'foreign_worker_1']
X_val.drop(cols_filted, axis=1, inplace=True)
X_val.head()
```

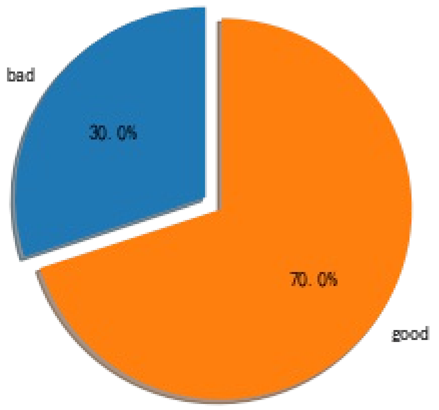
	Duration_in_month	Credit_amount	Installment_rate_in_percentage_
0	-1.236478	-0.745131	0.918477
1	2.248194	0.949817	-0.870183
2	-0.738668	-0.416562	-0.870183
3	1.750384	1.634247	-0.870183
4	0.256953	0.566664	0.024147

5 rows × 26 columns

重采样

处理数据不平衡

```
plt.pie([pd.value_counts(y_val)[0], pd.value_counts(y_val)[1]], explode=(
        shadow=True, startangle=90)
_ = plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as
```

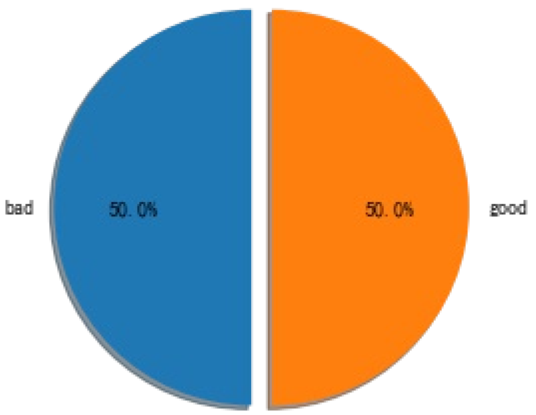


通过Logistic训练数据，移除低概率数据

```
from collections import Counter
from imblearn.under_sampling import InstanceHardnessThreshold
iht = InstanceHardnessThreshold(random_state=0, estimator=LogisticRegress
X_resampled, y_resampled = iht.fit_sample(X_val, y_val)
print(sorted(Counter(y_resampled).items()))
```

```
[(0, 300), (1, 300)]
```

```
plt.pie([len(y_resampled[y_resampled == 0]), len(y_resampled[y_resampled
       shadow=True, startangle=90)
_ = plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as
```



模型训练与评价

数据划分

```
from sklearn.model_selection import train_test_split
#X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resa
```

```
## 不重采样
X_train, X_test, y_train, y_test = train_test_split(X_val, y_val)
```

```
from imblearn.under_sampling import InstanceHardnessThreshold
from imblearn.over_sampling import SMOTE
iht = InstanceHardnessThreshold(random_state=0, estimator=LogisticRegress
#X_train, y_train = iht.fit_sample(X_train, y_train)
#X_test, y_test = iht.fit_sample(X_test, y_test)
#sm = SMOTE()
#X_test, y_test = sm.fit_sample(X_test, y_test)
```

模型训练

使用线性判别LDA分类

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
```

```
LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=Non
              solver='svd', store_covariance=False, tol=0.0001)
```

使用决策树DecisionTree分类

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=No
          max_features=None, max_leaf_nodes=None,
          min_impurity_decrease=0.0, min_impurity_split=None,
          min_samples_leaf=1, min_samples_split=2,
          min_weight_fraction_leaf=0.0, presort=False, random_state=None
          splitter='best')
```

使用伯努利朴素贝叶斯Bernoulli Naive Bayes

```
from sklearn.naive_bayes import BernoulliNB
bnb = BernoulliNB()
bnb.fit(X_train, y_train)
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

使用最邻近KNN算法

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

使用Logistics回归

```
from sklearn.linear_model import LogisticRegression
lg = LogisticRegression()
lg.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=T
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

使用SVM

```
from sklearn.svm import SVC
svc = SVC(kernel='rbf')
svc.fit(X_train, y_train)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

结果评价

```
models = dict(LDA=lda, DecisionTree=dtc, BernoulliNB=bnb, KNN=knn, Logist
```

Precision, Recall, F1-score, Accuracy

```
from sklearn import metrics
for name, model in models.items():
    print(name)
    print(metrics.classification_report(y_test, model.predict(X_test)))
    print('Accuracy:\t', metrics.accuracy_score(y_test, model.predict(X_t
    print('_________________________________________________________')
```

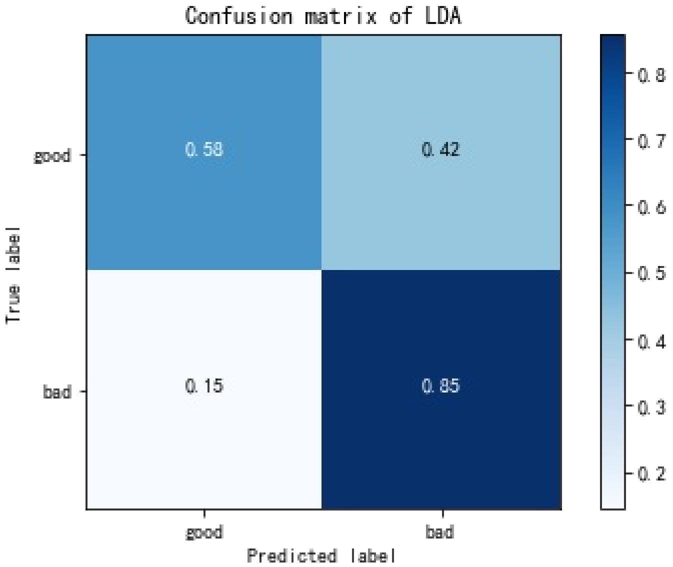
```
LDA
             precision    recall  f1-score   support

          0       0.61      0.58      0.59        71
          1       0.84      0.85      0.85       179

avg / total       0.77      0.78      0.77       250

Accuracy:	 0.776
_________________________________________________________
DecisionTree
             precision    recall  f1-score   support

          0       0.48      0.48      0.48        71
          1       0.79      0.79      0.79       179

avg / total       0.70      0.70      0.70       250

Accuracy:	 0.704
_________________________________________________________
BernoulliNB
             precision    recall  f1-score   support

          0       0.59      0.59      0.59        71
          1       0.84      0.84      0.84       179

avg / total       0.77      0.77      0.77       250

Accuracy:	 0.768
_________________________________________________________
KNN
             precision    recall  f1-score   support

          0       0.53      0.41      0.46        71
          1       0.78      0.85      0.82       179

avg / total       0.71      0.73      0.72       250

Accuracy:	 0.728
```

```
------------------------------------------------------------
LogisticRegression
             precision    recall  f1-score   support

          0       0.67      0.56      0.61        71
          1       0.84      0.89      0.86       179

avg / total       0.79      0.80      0.79       250

Accuracy:  0.796
------------------------------------------------------------
SupportVectorMachine
             precision    recall  f1-score   support

          0       0.62      0.46      0.53        71
          1       0.81      0.89      0.85       179

avg / total       0.75      0.77      0.76       250

Accuracy:  0.768
------------------------------------------------------------
```

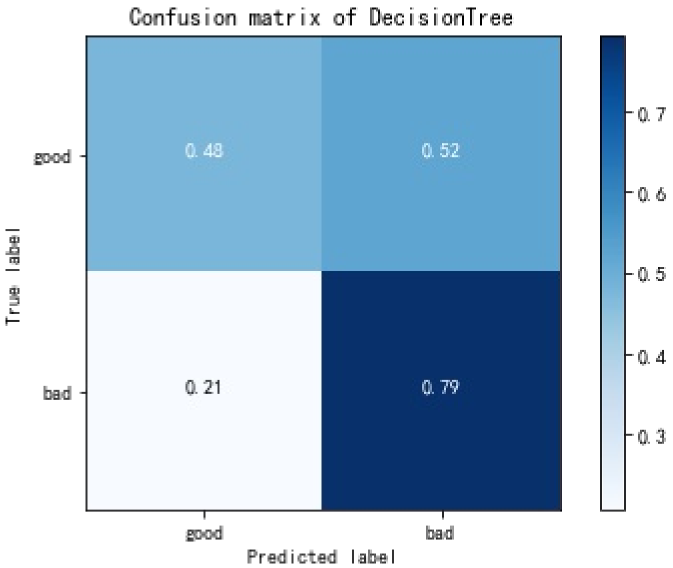
Confusion Matrix

```
from sklearn.metrics import confusion_matrix
from plot_comfusion_matrix import *

for name,model in models.items():
    cm = confusion_matrix(y_test, model.predict(X_test))
    plot_confusion_matrix(cm, classes=['good', 'bad'], normalize=True, ti
    plt.show()
    print('\n\n')
```

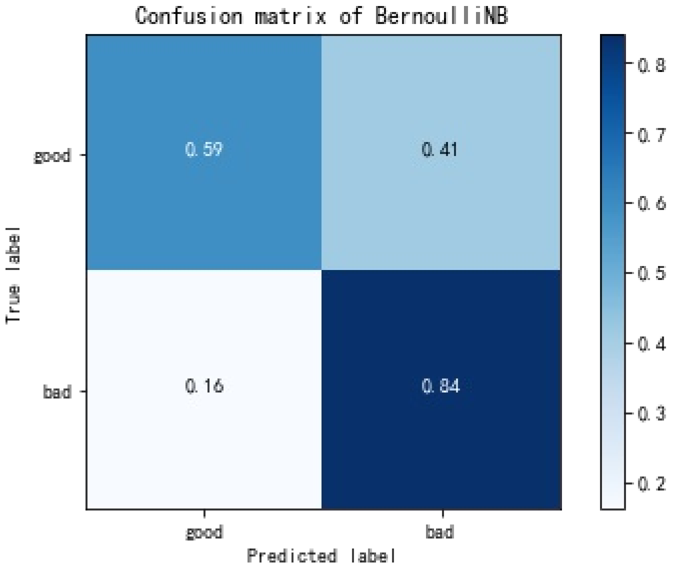
```
Normalized confusion matrix
[[ 0.57746479  0.42253521]
 [ 0.1452514   0.8547486 ]]
```



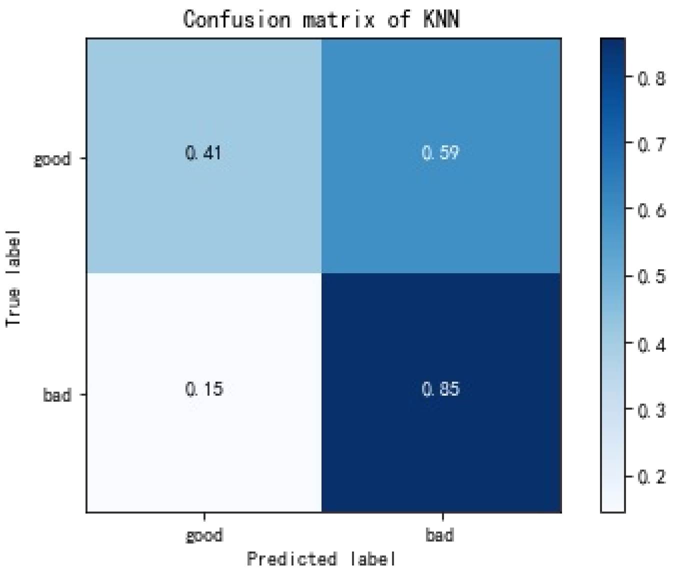

```
Normalized confusion matrix
[[ 0.47887324  0.52112676]
 [ 0.20670391  0.79329609]]
```



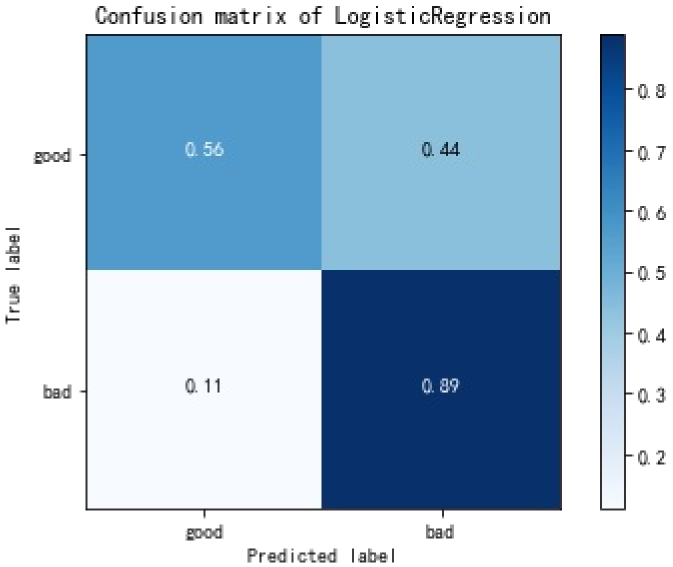

```
Normalized confusion matrix
[[ 0.5915493   0.4084507 ]
 [ 0.16201117  0.83798883]]
```



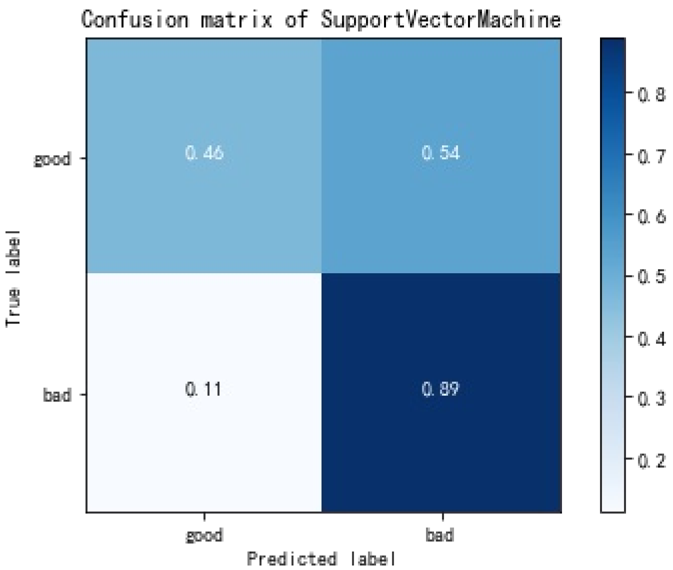

```
Normalized confusion matrix
[[ 0.4084507   0.5915493]
 [ 0.1452514   0.8547486]]
```




```
Normalized confusion matrix
[[ 0.56338028   0.43661972]
 [ 0.11173184   0.88826816]]
```




```
Normalized confusion matrix
[[ 0.46478873  0.53521127]
 [ 0.11173184  0.88826816]]
```



网络搜索和交叉验证提升模型

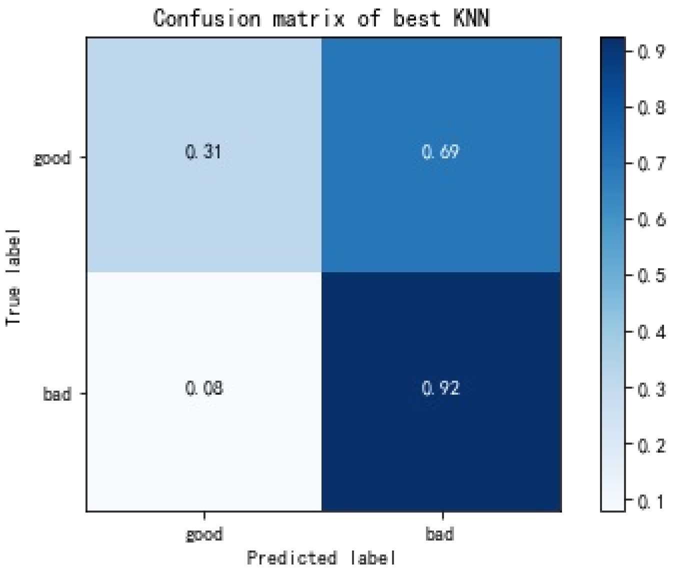
以 **KNN** 为例

```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn_ = KNeighborsClassifier()
k_range = list(range(1,20))
weight_options = ['uniform', 'distance']
kernel = ['linear', 'poly', 'rbf', 'sigmoid', 'precomputed']
algorithm = ['auto', 'ball_tree', 'kd_tree', 'brute']
param_knn = dict(n_neighbors=k_range, weights=weight_options, algorithm=a

RSCV_knn = RandomizedSearchCV(knn_, param_knn, cv=10, scoring='accuracy',
RSCV_knn.fit(X_train, y_train)
print("knn_best:%f" % metrics.accuracy_score(y_test, RSCV_knn.predict(X_t
```

```
knn_best:0.748000
```

```python
plot_confusion_matrix(confusion_matrix(y_test, RSCV_knn.predict(X_test))
                      classes=['good', 'bad'], normalize=True, title='Confus
print(metrics.classification_report(y_test, RSCV_knn.predict(X_test)))
```

```
Normalized confusion matrix
[[ 0.30985915  0.69014085]
 [ 0.07821229  0.92178771]]
             precision    recall  f1-score   support

          0       0.61      0.31      0.41        71
          1       0.77      0.92      0.84       179

avg / total       0.73      0.75      0.72       250
```



集成算法

随机森林

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=15,random_state=1)
rfc.fit(X_train, y_train)
rfc.score(X_test, y_test)
```

```
0.7439999999999999
```

极限随机树

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import cross_val_score

etc = ExtraTreesClassifier(n_estimators=1000, max_features=10,
    min_samples_split=2, random_state=0)
etc.fit(X_train, y_train)
etc.score(X_test, y_test)
```

```
0.7760000000000002
```

AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier
abc = AdaBoostClassifier(n_estimators=1000, learning_rate= 0.10)
abc.fit(X_train, y_train)
abc.score(X_test, y_test)
```

```
0.7640000000000001
```

梯度树提升（Gradient Tree Boosting）

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(n_estimators=1000, learning_rate=0.1,
    max_depth=1, random_state=0)
gbc.fit(X_train, y_train)
gbc.score(X_test, y_test)
```

```
0.7600000000000001
```

投票分类器

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier

clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()

eclf = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('gnb', c

for clf, label in zip([clf1, clf2, clf3, eclf], ['Logistic Regression', '
    scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='accurac
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std
```

```
Accuracy: 0.77 (+/- 0.02) [Logistic Regression]
Accuracy: 0.73 (+/- 0.04) [Random Forest]
Accuracy: 0.74 (+/- 0.02) [naive Bayes]
Accuracy: 0.77 (+/- 0.03) [Ensemble]
```

网格搜索下的投票分类器

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = BernoulliNB()
eclf = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('gnb', c

params = {'lr__C': [1.0, 100.0], 'rf__n_estimators': [20, 200],}

grid = GridSearchCV(estimator=eclf, param_grid=params, cv=5)
grid = grid.fit(X_train, y_train)
grid.score(X_test, y_test)
```

```
0.7880000000000003
```

模型评价

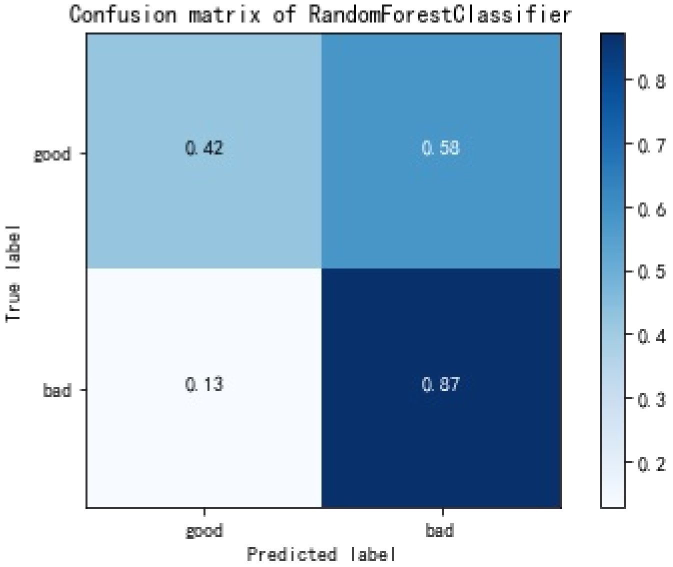
```
embad_models = dict(RandomForestClassifier=rfc, ExtraTreesClassifier=etc,
                    GradientBoostingClassifier=gbc, VotingClassifier=grid)
```

```
for name, model in embad_models.items():
    print(name)
    print(metrics.classification_report(y_test, model.predict(X_test)))
    print('Accuracy:\t', metrics.accuracy_score(y_test, model.predict(X_t
    print('________________________________________________________')
```

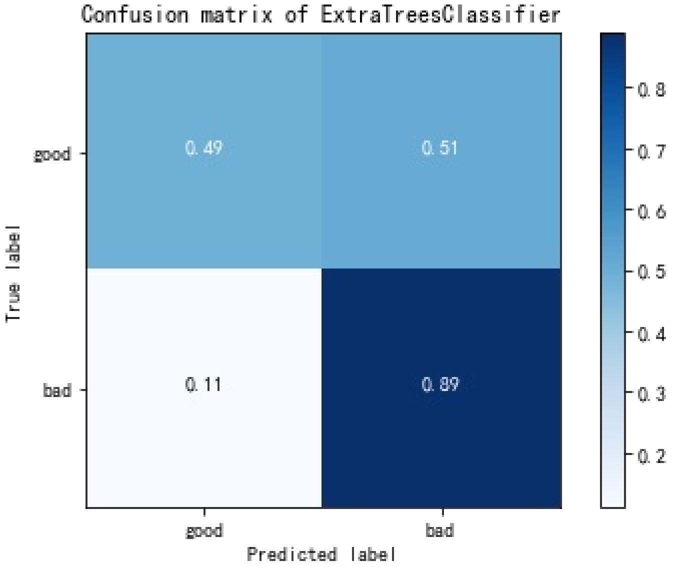
```
RandomForestClassifier
             precision    recall  f1-score   support

          0       0.57      0.42      0.48        71
          1       0.79      0.87      0.83       179

avg / total       0.73      0.74      0.73       250

Accuracy:  0.744
________________________________________________________
ExtraTreesClassifier
             precision    recall  f1-score   support

          0       0.64      0.49      0.56        71
          1       0.82      0.89      0.85       179

avg / total       0.76      0.78      0.77       250

Accuracy:  0.776
________________________________________________________
AdaBoostClassifier
             precision    recall  f1-score   support

          0       0.60      0.51      0.55        71
          1       0.82      0.87      0.84       179

avg / total       0.75      0.76      0.76       250

Accuracy:  0.764
________________________________________________________
GradientBoostingClassifier
             precision    recall  f1-score   support

          0       0.60      0.48      0.53        71
          1       0.81      0.87      0.84       179

avg / total       0.75      0.76      0.75       250

Accuracy:  0.76
________________________________________________________
VotingClassifier
             precision    recall  f1-score   support

          0       0.65      0.56      0.60        71
          1       0.84      0.88      0.86       179

avg / total       0.78      0.79      0.78       250

Accuracy:  0.788
________________________________________________________
```

```python
for name,model in embad_models.items():
    cm = confusion_matrix(y_test, model.predict(X_test))
    plot_confusion_matrix(cm, classes=['good', 'bad'], normalize=True, ti
    plt.show()
    print('\n\n')
```

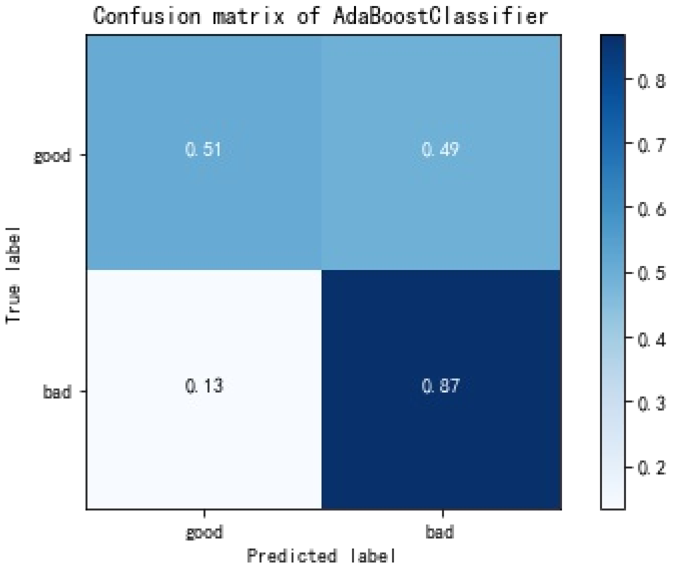
```
Normalized confusion matrix
[[ 0.42253521  0.57746479]
 [ 0.12849162  0.87150838]]
```



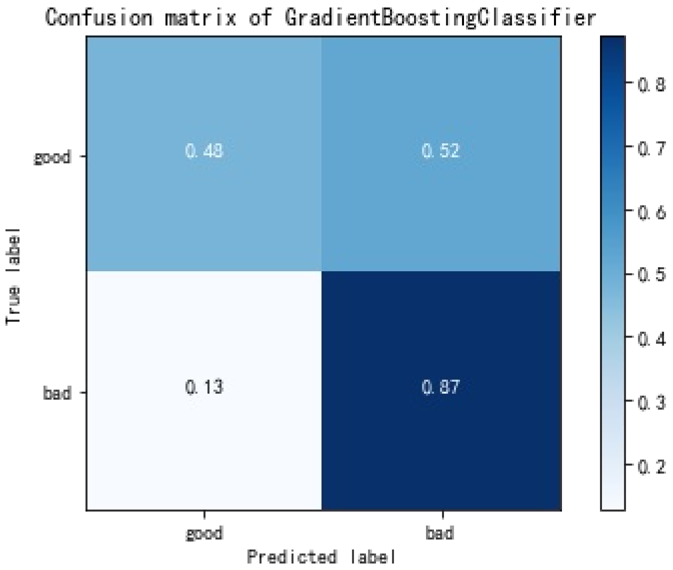

```
Normalized confusion matrix
[[ 0.49295775  0.50704225]
 [ 0.11173184  0.88826816]]
```



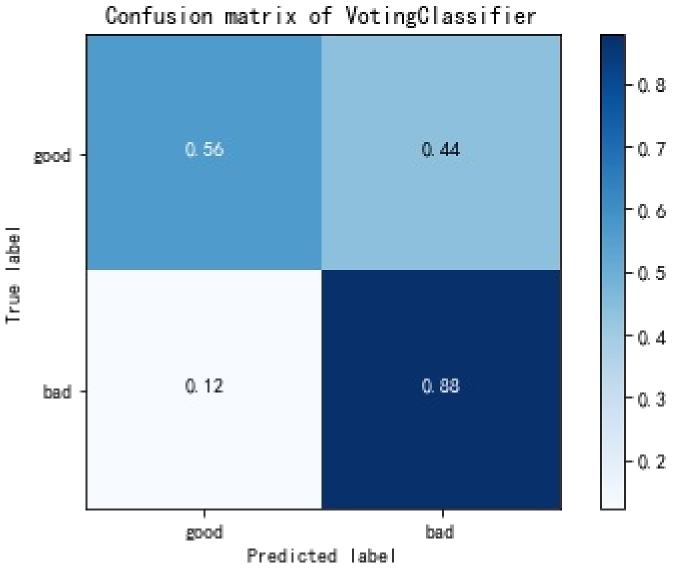

```
Normalized confusion matrix
[[ 0.50704225  0.49295775]
 [ 0.13407821  0.86592179]]
```




```
Normalized confusion matrix
[[ 0.47887324  0.52112676]
 [ 0.12849162  0.87150838]]
```




```
Normalized confusion matrix
[[ 0.56338028  0.43661972]
 [ 0.12290503  0.87709497]]
```



Confusion matrix of VotingClassifier
True label
good
bad
0.56
0.44
0.12
0.88
Predicted label
0.8
0.7
0.6
0.5
0.4
0.3
0.2