In [1]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import f1_score,precision_score,recall_score,roc_auc_score,accuracy_sco
re,roc_curve,confusion_matrix
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost.sklearn import XGBClassifier
import lightgbm as lgb
from sklearn.preprocessing import MinMaxScaler
import os
import pandas as pd
import lightgbm as lgb
from sklearn.preprocessing import Imputer
import math
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
%matplotlib inline
from sklearn.preprocessing import Imputer
from sklearn import preprocessing
import numpy as np
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29:
DeprecationWarning: numpy.core.umath_tests is an internal NumPy module and should
 not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

In [2]:

```
f = 'pocd.csv'
pocd = pd.read_csv(f, encoding = 'gb18030')

pocd.describe()
```

Out[2]:

	Sex	Age	Location	Tumor size	Grade	Sta
count	1330.000000	1330.000000	1330.000000	1330.000000	1330.000000	1330.000(
mean	1.680451	57.630827	2.022556	4.651053	1.830827	2.418045
std	0.466477	11.934602	0.889417	2.608218	0.375046	0.757986
min	1.000000	19.000000	1.000000	0.500000	1.000000	1.000000
25%	1.000000	50.000000	1.000000	3.000000	2.000000	2.000000
50%	2.000000	59.000000	2.000000	4.000000	2.000000	3.000000
75%	2.000000	66.000000	3.000000	6.000000	2.000000	3.000000
max	2.000000	89.000000	3.000000	30.000000	2.000000	3.000000

In [3]:

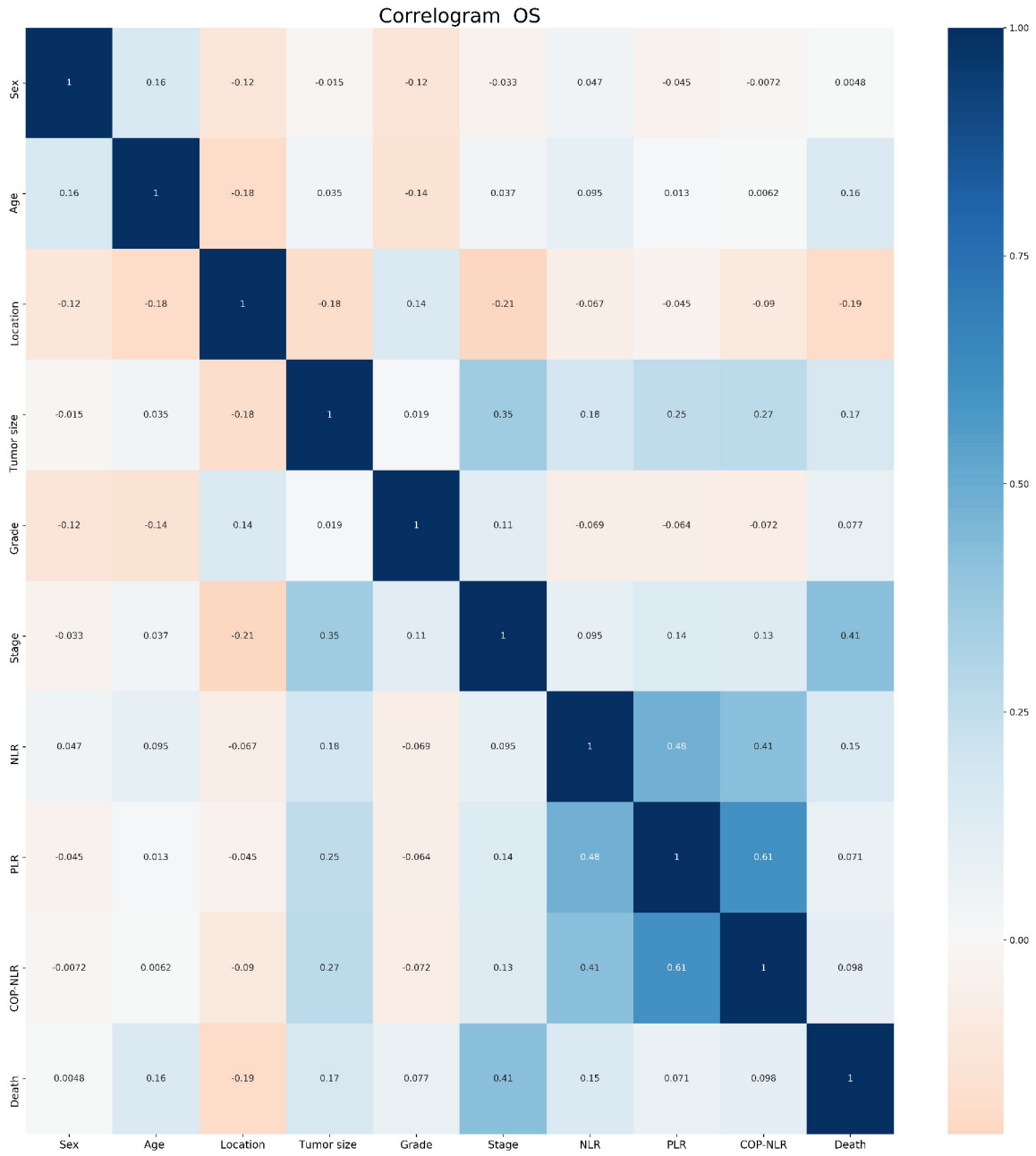
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

plt.figure(figsize=(22,23), dpi= 600)
sns.heatmap(pocd.corr(), xticklabels=pocd.corr().columns, yticklabels=pocd.corr().columns,
cmap='RdBu', center=0, annot=True)


plt.title('Correlogram OS', fontsize=22)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
```

Out[3]:

```
(array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5]),
 <a list of 10 Text yticklabel objects>)
```



In [4]:

```
X= pocd.iloc[:, 0:-1]
y = pocd.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_state=2018,stratif
y=y)
print(X_train)
print(y_train)
```

```
      Sex  Age   Location  Tumor size   Grade  Stage       NLR         PLR  \
1003    1    59         2          8.0      2      3  1.307692   72.527473
180     2    46         3          1.0      2      1  1.500000  106.666667
1096    1    51         3          8.0      2      3  1.280488  119.512195
1050    2    38         2          1.0      2      3  1.205279   70.381232
155     2    57         3          1.0      2      1  1.068421   69.473684
...   ...   ...       ...          ...    ...    ...       ...         ...
1064    2    74         2          6.0      2      3  3.133333  130.666667
879     2    57         1         11.0      2      3  3.208861  232.911392
1022    2    69         2          2.0      2      3  1.004348  111.304348
1211    2    43         3          5.0      2      3  2.742105  214.210526
970     1    59         2          6.0      2      3  2.058824  272.764706

      COP-NLR
1003        0
180         0
1096        0
1050        0
155         0
...       ...
1064        1
879         2
1022        0
1211        1
970         1

[1064 rows x 9 columns]
      Death
1003      1
180       0
1096      1
1050      0
155       0
...     ...
1064      1
879       1
1022      1
1211      1
970       0

[1064 rows x 1 columns]
```

In [5]:

```
xy_train = pd.concat([X_train, y_train], axis=1)
xy_test = pd.concat([X_test, y_test], axis=1)

f_xy_train = os.path.splitext(f)[0] + '_train.csv'
f_xy_test = os.path.splitext(f)[0] + '_test.csv'
xy_train.to_csv(f_xy_train, index=False, encoding = 'gb18030')
xy_test.to_csv(f_xy_test, index=False, encoding = 'gb18030')

xy_train = pd.read_csv(f_xy_train, encoding = 'gb18030')
xy_test = pd.read_csv(f_xy_test, encoding = 'gb18030')
```

In [6]:

```
X_train = xy_train.iloc[:, 0:-1]
y_train = xy_train.iloc[:, -1:]
X_test = xy_test.iloc[:, 0:-1]
y_test = xy_test.iloc[:, -1:]
```

In [7]:

```
import numpy as np

X_train = np.array(X_train)

X_test = np.array(X_test)
y_train = np.array(y_train)

ya, yb = y_train.shape
y_train = y_train.reshape(ya,)
y_test = np.array(y_test)
ya, yb = y_test.shape
y_test = y_test.reshape(ya,)
```

In [8]:

```
from sklearn.preprocessing import MinMaxScaler
ss = MinMaxScaler()
xy_train = ss.fit_transform(xy_train, y_train)
xy_test = ss.transform(xy_test)
print(xy_train )
```

```
[[0.         0.57142857 0.5        ... 0.05973563 0.         1.        ]
 [1.         0.38571429 1.         ... 0.09140097 0.         0.        ]
 [0.         0.45714286 1.         ... 0.10331566 0.         1.        ]
 ...
 [1.         0.71428571 0.5        ... 0.09570258 0.         1.        ]
 [1.         0.34285714 1.         ... 0.19115179 0.5        1.        ]
 [0.         0.57142857 0.5        ... 0.24546292 0.5        0.        ]]
```

In [9]:

```
xy_train.shape
```

Out[9]:

(1064, 10)

In [11]:

```
lr = LogisticRegression(penalty='l2',tol=0.0001,C=0.1,fit_intercept=True,intercept_scaling=1,class_weight=None,max_iter=100,multi_class='ovr',verbose=0,warm_start=False,n_jobs=1)  # 逻辑回归模型
lr.fit(X_train, y_train)
lr_y_proba=lr.predict_proba(X_train)
lr_y_pre=lr.predict(X_train)
lr_score = lr.score(X_train, y_train)
lr_accuracy_score=accuracy_score(y_train,lr_y_pre)
lr_preci_score=precision_score(y_train,lr_y_pre)
lr_recall_score=recall_score(y_train,lr_y_pre)
lr_f1_score=f1_score(y_train,lr_y_pre)
lr_auc=roc_auc_score(y_train,lr_y_proba[:,1])



print('lr_accuracy_score:%.3f,lr_preci_score:%.3f,lr_recall_score:%.3f,lr_f1_score:%.3f,lr_auc:%.3f'
      %(lr_accuracy_score,lr_preci_score,lr_recall_score,lr_f1_score,lr_auc))
```

```
lr_accuracy_score:0.657,lr_preci_score:0.575,lr_recall_score:0.496,lr_f1_score:0.533,lr_auc:0.737
```

In [12]:

```
tr = DecisionTreeClassifier(splitter='best', max_depth=5, min_samples_split=80,min_samples_leaf=65, min_weight_fraction_leaf=0.01, max_features=None, random_state=None, max_leaf_nodes=None,class_weight=None, presort=False)  # 决策树模型
tr.fit(X_train, y_train)
tr_y_pre=tr.predict(X_train)
tr_y_proba=tr.predict_proba(X_train)
tr_score = tr.score(X_train, y_train)
tr_accuracy_score=accuracy_score(y_train,tr_y_pre)
tr_preci_score=precision_score(y_train,tr_y_pre)
tr_recall_score=recall_score(y_train,tr_y_pre)
tr_f1_score=f1_score(y_train,tr_y_pre)
tr_auc=roc_auc_score(y_train,tr_y_proba[:,1])
print('tr_accuracy_score:%.3f,tr_preci_score:%.3f,tr_recall_score:%.3f,tr_f1_score:%.3f,tr_auc:%.3f'
     %(tr_accuracy_score,tr_preci_score,tr_recall_score,tr_f1_score,tr_auc))
```

```
tr_accuracy_score:0.698,tr_preci_score:0.659,tr_recall_score:0.484,tr_f1_score:0.558,tr_auc:0.767
```

In [13]:

```
forest=RandomForestClassifier(n_estimators=500,max_features = "auto",min_samples_leaf = 5 ,n_jobs = 100,random_state =1) #  随机森林 forest.fit(X_train,y_train)
forest.fit(X_train,y_train)
forest_y_pre=forest.predict(X_train)
forest_y_proba=forest.predict_proba(X_train)
forest_accuracy_score=accuracy_score(y_train,forest_y_pre)
forest_preci_score=precision_score(y_train,forest_y_pre)
forest_recall_score=recall_score(y_train,forest_y_pre)
forest_f1_score=f1_score(y_train,forest_y_pre)
forest_auc=roc_auc_score(y_train,forest_y_proba[:,1])
print('forest_accuracy_score:%.3f,forest_preci_score:%.3f,forest_recall_score:%.3f,forest_f1_score:%.3f,forest_auc:%.3f'
      %
(forest_accuracy_score,forest_preci_score,forest_recall_score,forest_f1_score,forest_auc))
```

```
forest_accuracy_score:0.885,forest_preci_score:0.878,forest_recall_score:0.823,forest_f1_score:0.850,forest_auc:0.943
```

In [14]:

```
Gbdt=GradientBoostingClassifier(learning_rate=0.2,n_estimators=60,max_depth=2, max_features='auto', min_samples_split=20,min_samples_leaf=3,random_state =1)#CBDT
Gbdt.fit(X_train,y_train)
Gbdt_y_pre=Gbdt.predict(X_train)
Gbdt_y_proba=Gbdt.predict_proba(X_train)
Gbdt_accuracy_score=accuracy_score(y_train,Gbdt_y_pre)
Gbdt_preci_score=precision_score(y_train,Gbdt_y_pre)
Gbdt_recall_score=recall_score(y_train,Gbdt_y_pre)
Gbdt_f1_score=f1_score(y_train,Gbdt_y_pre)
Gbdt_auc=roc_auc_score(y_train,Gbdt_y_proba[:,1])
print('Gbdt_accuracy_score:%.3f,Gbdt_preci_score:%.3f,Gbdt_recall_score:%.3f,Gbdt_f1_score:%.3f,Gbdt_auc:%.3f'
      %(Gbdt_accuracy_score,Gbdt_preci_score,Gbdt_recall_score,Gbdt_f1_score,Gbdt_auc))
```

```
Gbdt_accuracy_score:0.756,Gbdt_preci_score:0.675,Gbdt_recall_score:0.730,Gbdt_f1_score:0.702,Gbdt_auc:0.846
```

In [18]:

```
gbm=lgb.LGBMClassifier(learning_rate=0.08, n_estimators=180, lambda_l1=0.2 ,lambda_l2= 10 ,max_d
epth=2, bagging_fraction = 0.8,feature_fraction = 0.6) #lgb
gbm.fit(X_train,y_train)
gbm_y_pre=gbm.predict(X_train)
gbm_y_proba=gbm.predict_proba(X_train)
gbm_accuracy_score=accuracy_score(y_train,gbm_y_pre)
gbm_preci_score=precision_score(y_train,gbm_y_pre)
gbm_recall_score=recall_score(y_train,gbm_y_pre)
gbm_f1_score=f1_score(y_train,gbm_y_pre)
gbm_auc=roc_auc_score(y_train,gbm_y_proba[:,1])
print('gbm_accuracy_score:%.3f,gbm_preci_score: %.3f,gbm_recall_score:%.3f,gbm_f1_score:%.3f,g
bm_auc:%.3f'
      %(gbm_accuracy_score,gbm_preci_score,gbm_recall_score,gbm_f1_score,gbm_auc))
```

```
gbm_accuracy_score:0.725,gbm_preci_score: 0.641,gbm_recall_score:0.685,gbm_f1_scor
e:0.662,gbm_auc:0.807
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: Dep
recationWarning: The truth value of an empty array is ambiguous. Returning False,
 but in future this will result in an error. Use `array.size > 0` to check that an
array is not empty.
  if diff:
```

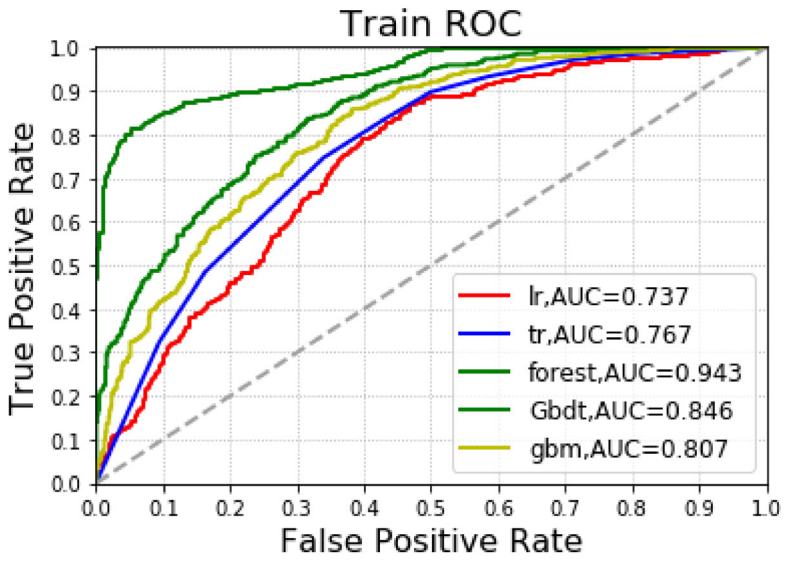
In [20]:

```
#机器学习算法图集合 plt.figure(figsize=(8, 6), facecolor='w')
lr_fpr,lr_tpr,lr_threasholds=roc_curve(y_train,lr_y_proba[:,1]) # 计算ROC的值,lr_threasholds为阈值
tr_fpr,tr_tpr,tr_threasholds=roc_curve(y_train,tr_y_proba[:,1]) # 计算ROC的值,lr_threasholds为阈值

forest_fpr,forest_tpr,forest_threasholds=roc_curve(y_train,forest_y_proba[:,1]) # 计算ROC的值,svm_threasholds为阈值
Gbdt_fpr,Gbdt_tpr,Gbdt_threasholds=roc_curve(y_train,Gbdt_y_proba[:,1]) # 计算ROC的值,svm_threasholds为阈值
#Xgbc_fpr,Xgbc_tpr,Xgbc_threasholds=roc_curve(y_train,Xgbc_y_pre) # 计算ROC的值,svm_threasholds为阈值
gbm_fpr,gbm_tpr,gbm_threasholds=roc_curve(y_train,gbm_y_proba[:,1]) # 计算ROC的值,svm_threasholds为阈值

plt.plot(lr_fpr,lr_tpr,c='r',lw=2,label=u'lr,AUC=%.3f'% lr_auc)
plt.plot(tr_fpr,tr_tpr,c='b',lw=2,label=u'tr,AUC=%.3f' % tr_auc)
plt.plot(forest_fpr,forest_tpr,c='g',lw=2,label=u'forest,AUC=%.3f' % forest_auc)
plt.plot(Gbdt_fpr,Gbdt_tpr,c='g',lw=2,label=u'Gbdt,AUC=%.3f' % Gbdt_auc)
#plt.plot(Xgbc_fpr,Xgbc_tpr,c='g',lw=2,label=u'Xgbc,AUC=%.3f' % svm_auc)
plt.plot(gbm_fpr,gbm_tpr,c='y',lw=2,label=u'gbm,AUC=%.3f' % gbm_auc)

plt.plot((0,1),(0,1),c='#a0a0a0',lw=2,ls='--')
plt.xlim(-0.001, 1.001)
plt.ylim(-0.001, 1.001)
plt.xticks(np.arange(0, 1.1, 0.1))
plt.yticks(np.arange(0, 1.1, 0.1))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.grid(b=True, ls=':')
plt.legend(loc='lower right', fancybox=True, framealpha=0.8, fontsize=12)
plt.title(u'Train ROC', fontsize=18)
fig = plt.figure(figsize=(8, 15), dpi= 600)
plt.show()
```



<Figure size 4800x9000 with 0 Axes>

In [21]:

```
lr = LogisticRegression(tol=0.000001,C=1,max_iter=1000,n_jobs=-1)
lr.fit(X_train, y_train)
lr_y_proba=lr.predict_proba(X_test)
lr_y_pre=lr.predict(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:1228:
 UserWarning: 'n_jobs' > 1 does not have any effect when 'solver' is set to 'libli
near'. Got 'n_jobs' = -1.
  " = {}.".format(self.n_jobs))
```

In [22]:

```
lr_score = lr.score(X_test, y_test)
lr_accuracy_score=accuracy_score(y_test,lr_y_pre)
lr_preci_score=precision_score(y_test,lr_y_pre)
lr_recall_score=recall_score(y_test,lr_y_pre)
lr_f1_score=f1_score(y_test,lr_y_pre)
lr_auc=roc_auc_score(y_test,lr_y_proba[:,1])
print('lr_accuracy_score:%.3f,lr_preci_score:%.3f,lr_recall_score:%.3f,lr_f1_score:%.3f,lr_au
c:%.3f'
      %(lr_accuracy_score,lr_preci_score,lr_recall_score,lr_f1_score,lr_auc))
```

```
lr_accuracy_score:0.763,lr_preci_score:0.719,lr_recall_score:0.657,lr_f1_score:0.6
87,lr_auc:0.830
```

In [23]:

```
tr = DecisionTreeClassifier(splitter='best', max_depth=5, min_samples_split=80,min_samples_leaf
=65, min_weight_fraction_leaf=0.01, max_features=None, random_state=None,
max_leaf_nodes=None,class_weight=None, presort=False)  # 决策树模型
tr.fit(X_train, y_train)
tr_y_pre=tr.predict(X_test)
tr_y_proba=tr.predict_proba(X_test)
```

In [24]:

```
tr_score = tr.score(X_test, y_test)
tr_accuracy_score=accuracy_score(y_test,tr_y_pre)
tr_preci_score=precision_score(y_test,tr_y_pre)
tr_recall_score=recall_score(y_test,tr_y_pre)
tr_f1_score=f1_score(y_test,tr_y_pre)
tr_auc=roc_auc_score(y_test,tr_y_proba[:,1])
print('tr_accuracy_score:%.3f,tr_preci_score:%.3f,tr_recall_score:%.3f,tr_f1_score:%.3f,tr_au
c:%.3f'
     %(tr_accuracy_score,tr_preci_score,tr_recall_score,tr_f1_score,tr_auc))
```

```
tr_accuracy_score:0.684,tr_preci_score:0.644,tr_recall_score:0.448,tr_f1_score:0.5
28,tr_auc:0.765
```

In [25]:

```
forest=RandomForestClassifier(n_estimators=500,max_features = "auto",min_samples_leaf = 5 ,rando
m_state =41) #  随机森林
forest.fit(X_train,y_train)
forest_y_pre=forest.predict(X_test)
forest_y_proba=forest.predict_proba(X_test)
```

In [26]:

```
forest_accuracy_score=accuracy_score(y_test,forest_y_pre)
forest_preci_score=precision_score(y_test,forest_y_pre)
forest_recall_score=recall_score(y_test,forest_y_pre)
forest_f1_score=f1_score(y_test,forest_y_pre)
forest_auc=roc_auc_score(y_test,forest_y_proba[:,1])
print('forest_accuracy_score:%.3f,forest_preci_score:%.3f,forest_recall_score:%.3f,forest_f1_s
core:%.3f,forest_auc:%.3f'
      %
(forest_accuracy_score,forest_preci_score,forest_recall_score,forest_f1_score,forest_auc))
```

```
forest_accuracy_score:0.680,forest_preci_score:0.596,forest_recall_score:0.590,for
est_f1_score:0.593,forest_auc:0.781
```

In [27]:

```
Gbdt=GradientBoostingClassifier(learning_rate=0.2,n_estimators=60,max_depth=2, max_features='aut
o', min_samples_split=20,min_samples_leaf=3,random_state =1)#CBDT
Gbdt.fit(X_train,y_train)
Gbdt_y_pre=Gbdt.predict(X_test)
Gbdt_y_proba=Gbdt.predict_proba(X_test)
```

In [28]:

```
Gbdt_accuracy_score=accuracy_score(y_test,Gbdt_y_pre)
Gbdt_preci_score=precision_score(y_test,Gbdt_y_pre)
Gbdt_recall_score=recall_score(y_test,Gbdt_y_pre)
Gbdt_f1_score=f1_score(y_test,Gbdt_y_pre)
Gbdt_auc=roc_auc_score(y_test,Gbdt_y_proba[:,1])
print('Gbdt_accuracy_score:%.3f,Gbdt_preci_score:%.3f,Gbdt_recall_score:%.3f,Gbdt_f1_score:%.3
f,Gbdt_auc:%.3f'
      %(Gbdt_accuracy_score,Gbdt_preci_score,Gbdt_recall_score,Gbdt_f1_score,Gbdt_auc))
```

```
Gbdt_accuracy_score:0.718,Gbdt_preci_score:0.642,Gbdt_recall_score:0.648,Gbdt_f1_s
core:0.645,Gbdt_auc:0.802
```

In [31]:

```
gbm=lgb.LGBMClassifier(learning_rate=0.08, n_estimators=180, lambda_l1=0.2 ,lambda_l2= 10 ,max_d
epth=2, bagging_fraction = 0.8,feature_fraction = 0.6)  #lgb
gbm.fit(X_train,y_train)
gbm_y_pre=gbm.predict(X_test)
gbm_y_proba=gbm.predict_proba(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: Dep
recationWarning: The truth value of an empty array is ambiguous. Returning False,
 but in future this will result in an error. Use `array.size > 0` to check that an
array is not empty.
  if diff:
```

In [32]:

```
gbm_accuracy_score=accuracy_score(y_test,gbm_y_pre)
gbm_preci_score=precision_score(y_test,gbm_y_pre)
gbm_recall_score=recall_score(y_test,gbm_y_pre)
gbm_f1_score=f1_score(y_test,gbm_y_pre)
gbm_auc=roc_auc_score(y_test,gbm_y_proba[:,1])
print('gbm_accuracy_score:%.3f,gbm_preci_score: %.3f,gbm_recall_score:%.3f,gbm_f1_score:%.3f,g
bm_auc:%.3f'
      %(gbm_accuracy_score,gbm_preci_score,gbm_recall_score,gbm_f1_score,gbm_auc))
```

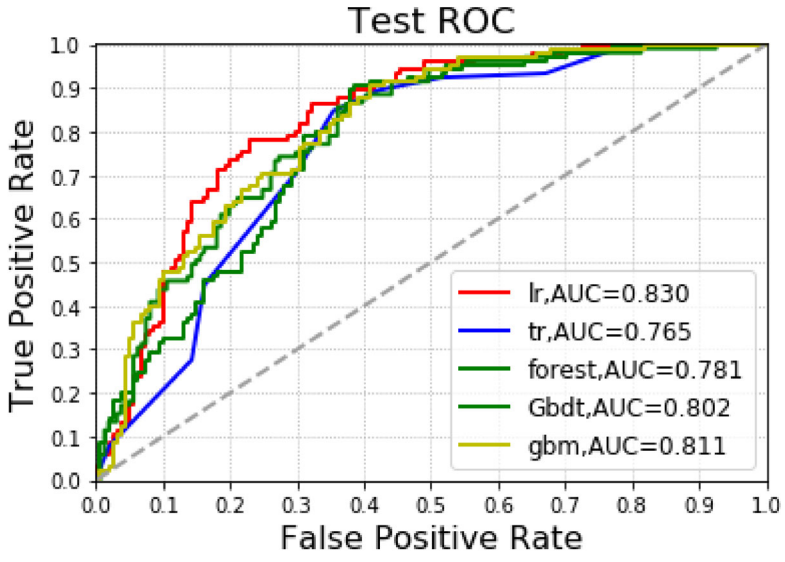
```
gbm_accuracy_score:0.733,gbm_preci_score: 0.660,gbm_recall_score:0.667,gbm_f1_scor
e:0.664,gbm_auc:0.811
```

In [34]:

```
tr_fpr,tr_tpr,tr_threasholds=roc_curve(y_test,tr_y_proba[:,1])
lr_fpr,lr_tpr,lr_threasholds=roc_curve(y_test,lr_y_proba[:,1])
forest_fpr,forest_tpr,forest_threasholds=roc_curve(y_test,forest_y_proba[:,1])
Gbdt_fpr,Gbdt_tpr,Gbdt_threasholds=roc_curve(y_test,Gbdt_y_proba[:,1])
gbm_fpr,gbm_tpr,gbm_threasholds=roc_curve(y_test,gbm_y_proba[:,1])

plt.plot(lr_fpr,lr_tpr,c='r',lw=2,label=u'lr,AUC=%.3f'% lr_auc)
plt.plot(tr_fpr,tr_tpr,c='b',lw=2,label=u'tr,AUC=%.3f' % tr_auc)
plt.plot(forest_fpr,forest_tpr,c='g',lw=2,label=u'forest,AUC=%.3f' % forest_auc)
plt.plot(Gbdt_fpr,Gbdt_tpr,c='g',lw=2,label=u'Gbdt,AUC=%.3f' % Gbdt_auc)
plt.plot(gbm_fpr,gbm_tpr,c='y',lw=2,label=u'gbm,AUC=%.3f' % gbm_auc)

plt.plot((0,1),(0,1),c='#a0a0a0',lw=2,ls='--')
plt.xlim(-0.001, 1.001)
plt.ylim(-0.001, 1.001)
plt.xticks(np.arange(0, 1.1, 0.1))
plt.yticks(np.arange(0, 1.1, 0.1))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.grid(b=True, ls=':')
plt.legend(loc='lower right', fancybox=True, framealpha=0.8, fontsize=12)
plt.title(u'Test ROC', fontsize=18)
fig = plt.figure(figsize=(8, 15), dpi= 600)
plt.show()
```



<Figure size 4800x9000 with 0 Axes>

In [35]:

```
import numpy as np


cols = pocd.iloc[:, 0:-1].columns
feature_importance = gbm.feature_importances_
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5


fig = plt.figure(figsize=(2, 3), dpi= 300)


plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, cols[sorted_idx], fontsize=5)
plt.xlabel('Relative Importance', fontsize=6)
plt.title('gbm Variable Importance', fontsize=5)
plt.show();
```