中南大學
CENTRAL SOUTH UNIVERSITY

统计学习研究生课程论文

题　　目：Stroke prediction based on SMOTE

学生姓名：魏莹莹　202112118

学　　院：数学与统计学院

专业班级：应用统计 2001 班

2021 年 6 月

Stroke prediction based on SMOTE

1. Introduction

Stroke is the 2nd leading cause of death globally according to the World Health Organization (WHO), responsible for approximately 11% of total deaths. A stroke occurs when the blood supply to part of your brain is interrupted or reduced, preventing brain tissue from getting oxygen and nutrients. Brain cells begin to die in minutes. A stroke is a medical emergency, and prompt treatment is crucial. Early action can reduce brain damage and other complications.

In this report, we try to predict whether a patient is likely to get stroke based on the input parameters like gender, age, various diseases, and smoking status. Our top priority in this health problem is to identify patients with a stroke.

2. Data exploration

The dataset is downloaded from https://www.kaggle.com/fedesoriano/stroke-prediction-dataset, with 5110 rows and 12 columns. Each row in the data provides relavant information about the patient including:

- id: unique identifier.
- gender: "Male", "Female" or "Other".
- age: age of the patient.
- hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension.
- heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease.
- ever_married: "No" or "Yes".
- work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed".
- Residence_type: "Rural" or "Urban".
- avg_glucose_level: average glucose level in blood.
- bmi: body mass index.

- smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown" where "Unknown" means that the information is unavailable for this patient.
- stroke: 1 if the patient had a stroke or 0 if not.

There are 3.93% missing data of BMI variable and we impute them with the mean of BMI. Then we plot the histograms of the categorical variables as shown in Figure 1. The response variable "Stroke" from the bottom right is highly imbalanced, with 4861 subjects without any stroke while only 249 subjects with a stroke. There is one subject with gender as other and will be deleted for further convenience.

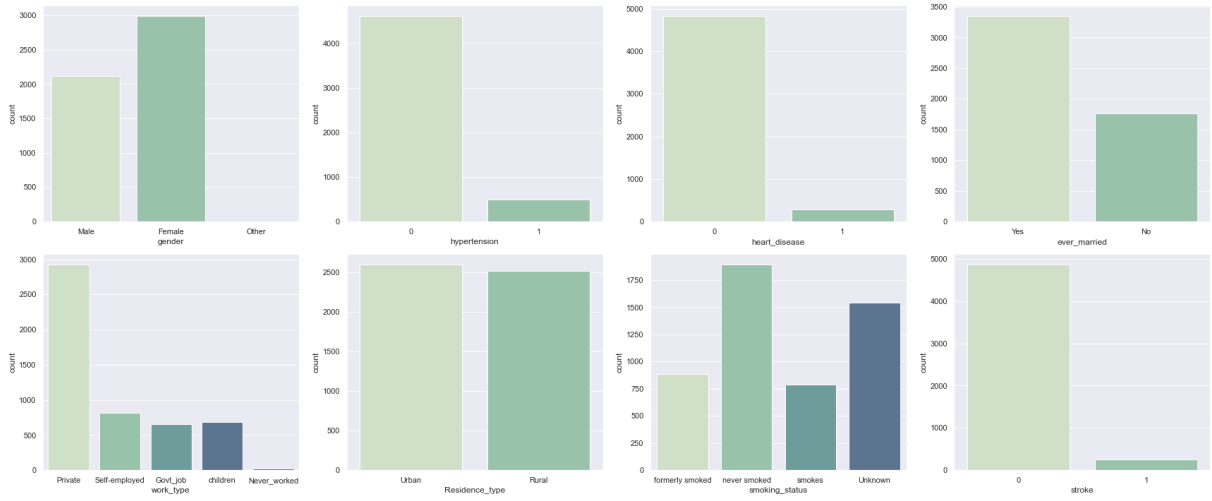

Figure 1. Distribution of categorical variables

The distribution density plot of numerical variables is shown in Figure 2. Only the bmi is unimodel. There is a small peak for age and average glucose level. And none of the distribution of these three variables can be seen as approximately normal.

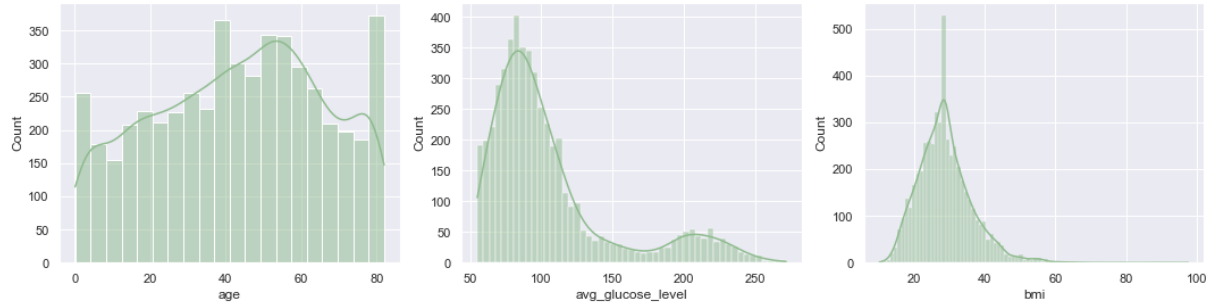

Figure 2. Distribution of numerical variables

Then dummy variables are created for categorical variables and the correlation matrix is shown in Figure 3. It seems like work type as focusing on the family and children is correlated with the age, bmi and marital status of the subject, which makes sense for demographical and sociological reasons.

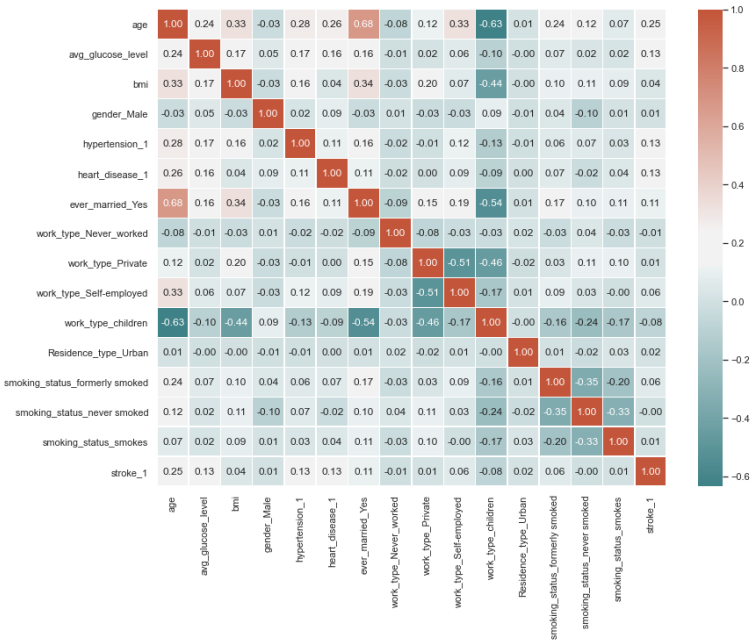

Figure 3. The correlation matrix

3. Model selection

We fit logistic regression as a benchmark model and compare it with KNN classifier and Random Forest. 30% of the data is randomly chosen as the test set and models are fit based on 70% of the data left.

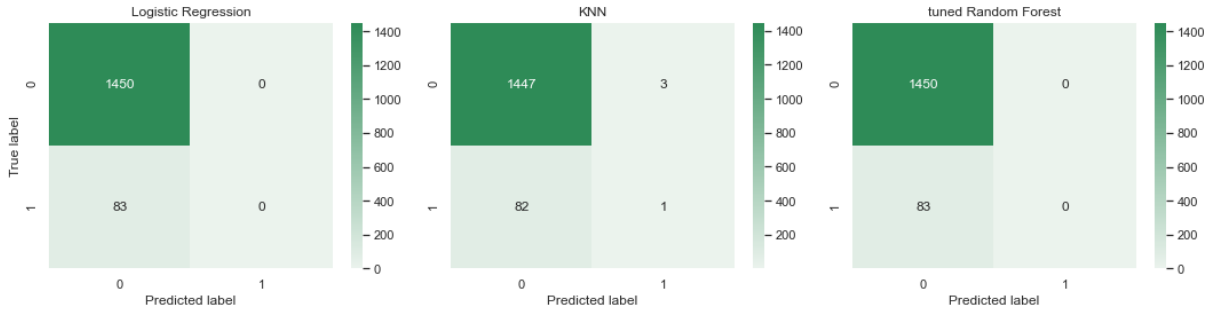

Figure 4. The confusion matrices of the three models

Before applying SMOTE, we fit logistic regression, KNN and random forest tuned based on grid search with cross validation, the confusion matrices on the test set of the three models are shown in Figure 4. Even though all three models get a high accuracy as around 0.95, they tend to predict all the subjects in the test set to be free of stroke, since the respondent variable stroke is so imbalanced that the non-stroke subjects would not make a huge difference on the outcome.

To deal with this issue, we introduce Synthetic Minority Oversampling Technique, or SMOTE for short, which is just oversampling the examples in the minority class.

This can be achieved by simply duplicating examples from the minority class in the training dataset prior to fitting a model. This can balance the class distribution but does not provide any additional information to the model. SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line. Since the original paper on SMOTE(Nitesh Chawla, et al. 2002) suggested combining SMOTE with random undersampling of the majority class. So for our model, we first oversample the minority class to have 10 percent the number of examples of the majority class, then use random undersampling to reduce the number of examples in the majority class to have twice the number of minority class.

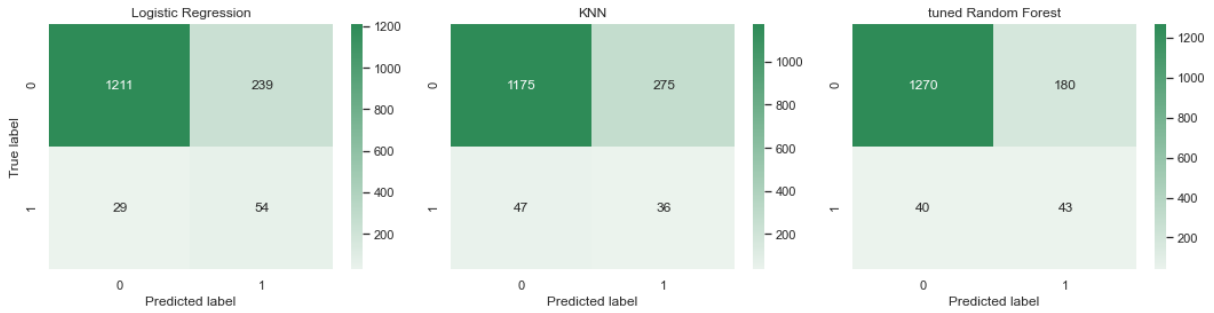

Figure 5. The confusion matrices of the three models after applying SMOTE

Table 1. Classification Report

	Logistic Regression			KNN			Random Forest			
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	support
0	0.98	0.84	0.90	0.96	0.81	0.88	0.97	0.88	0.92	1450
1	0.18	0.65	0.29	0.12	0.43	0.18	0.19	0.52	0.28	80
Accuracy			0.83			0.79			0.86	1533
macro avg	0.58	0.74	0.59	0.54	0.62	0.53	0.58	0.70	0.60	1533
weighted avg	0.93	0.83	0.87	0.92	0.79	0.84	0.93	0.86	0.89	1533

After introducing SMOTE to the training set, we fit logistic regression, KNN and random forest classifier. Randomized search with cross validation is applied and the best parameters are found. The confusion matrices and classification reports on the test set of the three models are shown in Figure 5 and Table 1. As we can observe from the confusion matrices, all three models do better on the test set with the minority class of the variable stroke. 54 out of 83 subjects with strokes are correctly classified by logistic regression, which is the best result of the three models with the highest recall of the minority class 1 as 0.65. However, logistic regression compromises its accuracy

on the majority class a lot for better prediction on the minority class. By considering the F1-score, Random Forest is the best model for over-all performance.

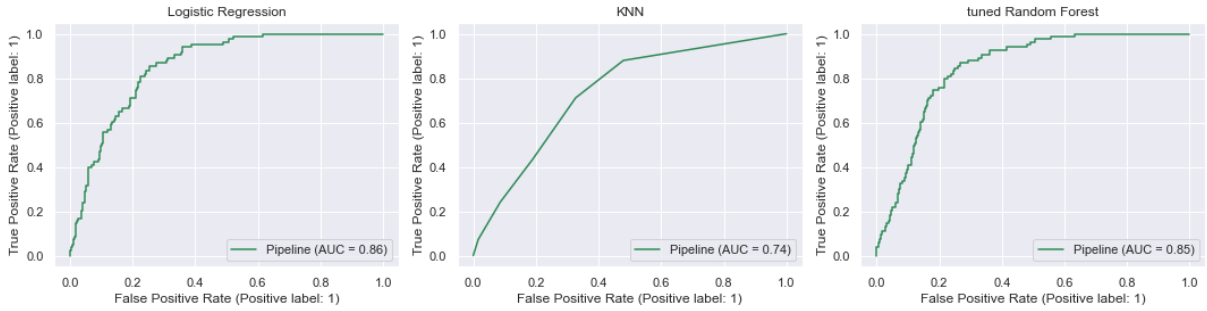

Figure 6. The ROC curves of the three models after applying SMOTE

From the ROC curves of the three models shown in Figure 6, we can observe that the KNN model is not good at this classification task, with an AUC value as only 0.74, far less than the other too models. With the AUC values as 0.86 and 0.85 for logistic regression and random forest model respectively, these two models are both reasonable to predict the stroke of subjects.

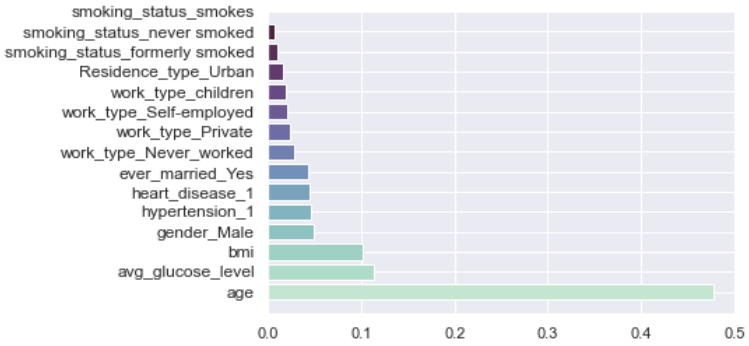

Figure 7. The importance of every feature in Random Forest with SMOTE

Feature importances are computed as the mean and standard deviation of accumulation of the impurity decrease within each tree in python and are shown in Figure 7. We observe that, as expected, the three first features are found important. We observe that the age of the subject is the most important feature with its importance as about 0.48. The average glucose level and bmi of the subjects are also found important when predicting stroke.

To improve the performance of our models, we can try other proportions of oversampling and undersampling and choose the best one based on cross validation. However, it may result in over fitting and weaken the model's ability to generalize. We can also try different techniques of balancing the data such as Borderline SMOTE, SVM SMOTE, ADASYN, etc.

Python Code

```
#Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
sns.set(style="white")
sns.set(style="darkgrid", color_codes=True)
import warnings
warnings.filterwarnings("ignore")

# import the data
df = pd.read_csv('healthcare-dataset-stroke-data.csv')
df.head()
print(df.shape)
df.info()

# explore the data
# drop the first column "id"
data = df.drop('id', axis=1)
print(data.shape)

# check the missing values
round(df.isnull().sum()/df.shape[0]*100,2) #3.93% of BMI is missing

# impute the missing data of bmi with its mean
data.bmi.fillna(np.mean(data.bmi), inplace = True)

cat_data = [x for x in data.columns if
            data[x].dtype == "object" or data[x].dtype =="int64"]
num_data = [y for y in data.columns if
            data[y].dtype != "object" and data[y].dtype != "int64"]

from collections import Counter
for i in cat_data:
    print(i," : ", Counter(data[i]), '\n')

f, ax = plt.subplots(2,4,figsize=(24,10), sharey=False)
```

```
for i in range(len(cat_data)):
    sns.countplot(x=cat_data[i], data=data, ax=ax[i//4][i%4],
                  palette=sns.cubehelix_palette(6,start=0.7,rot=-.75))
plt.tight_layout()
plt.show()

# delete the observation with gender being "other"
data = data[data.gender!="Other"]
print(data.shape)

# create dummy variables for the categorical variables
data_dummies = pd.get_dummies(data, columns=cat_data, drop_first=True)
print(data_dummies.shape)
data_dummies.head()

import matplotlib.pyplot as plt
import matplotlib
# check the distribution of the numerical variables
f, axes = plt.subplots(1,len(num_data), figsize=(15,4), sharex=False)
for i in range(len(num_data)):
    sns.histplot(x=num_data[i], data=data, ax=axes[i], kde=True,
                 color = "darkseagreen")
plt.tight_layout()
plt.show()

# Plot the correlation matrix
cmap = sns.diverging_palette(200,20,sep=20,as_cmap=True)
corr_matrix = data_dummies.corr()
fig, ax = plt.subplots(figsize=(13, 10))
ax = sns.heatmap(corr_matrix, annot=True, linewidths=0.5,
                 fmt=".2f", cmap=cmap)


# fit the models
from imblearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score,
                                    RepeatedStratifiedKFold
from sklearn.metrics import classification_report,roc_auc_score,
```

```
                                        roc_curve
from sklearn.metrics import plot_roc_curve,confusion_matrix

# Split data into train and test sets
np.random.seed(1)
X = data_dummies.drop("stroke_1", axis = 1)
y = data_dummies["stroke_1"]

from sklearn.model_selection import train_test_split,cross_val_predict
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
len(y_train)

def plot_conf_mat(y_test_pred, y_pred):
    """
    Plots a nice looking confusion matrix using Seaborn's heatmap()
    """
    sns.heatmap(confusion_matrix(y_test, y_pred), #,normalize='true'
                annot=True,cbar=True, fmt='g',cmap=sns.color_palette('
                                                    Blues'))
    plt.xlabel("Predicted label")
    plt.ylabel("True label")


##### fitting the models without SMOTE
# logistic regression
from sklearn.linear_model import LogisticRegression
# define pipeline
steps = [("scaler", StandardScaler()),
         ('model', LogisticRegression())]
logReg_pip = Pipeline(steps=steps)
# evaluate pipeline
y_train_pred_lr = cross_val_predict(logReg_pip,X_train,y_train,cv=10)
print(classification_report(y_train, y_train_pred_lr))
# Plot ROC curve and calculate and calculate AUC metric
logReg_pip.fit(X_train, y_train)
plot_roc_curve(logReg_pip, X_test, y_test)
y_test_pred_lr=logReg_pip.predict(X_test)
print(classification_report(y_test, y_test_pred_lr))
```

```
plot_conf_mat(y_test, y_test_pred_lr)

## KNN
from sklearn.neighbors import KNeighborsClassifier
steps = [("scaler", StandardScaler()),
         ('model', KNeighborsClassifier())]
KNN_pip = Pipeline(steps=steps)

y_train_pred_knn = cross_val_predict(KNN_pip, X_train, y_train, cv = 10)
print(classification_report(y_train, y_train_pred_knn))
KNN_pip.fit(X_train, y_train)
plot_roc_curve(KNN_pip, X_test, y_test)
y_test_pred_knn=KNN_pip.predict(X_test)
plot_conf_mat(y_test, y_test_pred_knn)
print(classification_report(y_test, y_test_pred_knn))

## Random Forest
from sklearn.ensemble import RandomForestClassifier
steps = [("scaler", StandardScaler()),
         ('rfc', RandomForestClassifier(random_state=1))]
RF_pip = Pipeline(steps=steps)

y_train_pred_rf = cross_val_predict(RF_pip, X_train, y_train, cv = 10)
print(classification_report(y_train, y_train_pred_rf))
RF_pip.fit(X_train, y_train)
plot_roc_curve(RF_pip, X_test, y_test)
y_test_pred_rf=RF_pip.predict(X_test)
plot_conf_mat(y_test, y_test_pred_rf)
print(classification_report(y_test, y_test_pred_rf))

# tuning the hyperparameters
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
params={
    'rfc__max_depth': [10, 20],
    'rfc__n_estimators': [50, 100, 200, 400],
    'rfc__max_features': ['auto', 'sqrt'],
    'rfc__min_samples_leaf': [2,4,8],
    'rfc__min_samples_split': [10,20]
}
```

```
rf_grid = GridSearchCV(RF_pip, params, cv=3,n_jobs=-1,scoring="f1")
rf_grid.fit(X_train, y_train)
print("Best Parameters for Model:  ",rf_grid.best_params_)

## Best Parameters for Model:
##  {'rfc__max_depth': 10, 'rfc__max_features': 'auto',
##   'rfc__min_samples_leaf': 2, 'rfc__min_samples_split': 10,
##   'rfc__n_estimators': 50}

steps = [("scaler", StandardScaler()),
         ('rfb', RandomForestClassifier(random_state=1,
         max_features= 'auto', min_samples_leaf= 2,
         n_estimators= 50,min_samples_split= 10))]

RFB_pip = Pipeline(steps)
y_pred_rfb = cross_val_predict(RFB_pip, X_train, y_train, cv = 10)
print(classification_report(y_train, y_pred_rfb))

RFB_pip.fit(X_train, y_train)
plot_roc_curve(RFB_pip, X_test, y_test)
y_test_pred_rfb=RFB_pip.predict(X_test)
print(classification_report(y_test, y_test_pred_rfb))

plot_conf_mat(y_test, y_test_pred_rfb)

# plot the confusion matrix
confcmap = sns.light_palette("seagreen",as_cmap=True)
f, ax = plt.subplots(1,3,figsize=(15,4))
sns.heatmap(confusion_matrix(y_test, y_test_pred_lr), ax=ax[0],
            annot=True,cbar=True, fmt='g',cmap=confcmap)
ax[0].set_title("Logistic Regression")
sns.heatmap(confusion_matrix(y_test, y_test_pred_knn), ax=ax[1],
            annot=True,cbar=True, fmt='g',cmap=confcmap)
ax[1].set_title("KNN")
sns.heatmap(confusion_matrix(y_test, y_test_pred_rfb), ax=ax[2],
            annot=True,cbar=True, fmt='g',cmap=confcmap)
ax[2].set_title("tuned Random Forest")
ax[0].set_xlabel("Predicted label")
ax[1].set_xlabel("Predicted label")
```

```
ax[2].set_xlabel("Predicted label")
ax[0].set_ylabel("True label")
plt.tight_layout()
plt.show()


## applying SMOTE
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
over = SMOTE(sampling_strategy=0.1, random_state=1)
under = RandomUnderSampler(sampling_strategy=0.5, random_state=1)

steps = [("scaler", StandardScaler()), ('over', over), ('under', under),
         ('model', LogisticRegression())]
logRegS_pip = Pipeline(steps=steps)
# evaluate pipeline
y_train_pred_lrs = cross_val_predict(logRegS_pip, X_train, y_train, cv =
                                     10)
print(classification_report(y_train, y_train_pred_lrs))
# Plot ROC curve and calculate and calculate AUC metric
logRegS_pip.fit(X_train, y_train)
plot_roc_curve(logRegS_pip, X_test, y_test)
y_test_pred_lrs=logRegS_pip.predict(X_test)
plot_conf_mat(y_test, y_test_pred_lrs)
print(classification_report(y_test, y_test_pred_lrs))

# borderline-SMOTE for imbalanced dataset
from imblearn.over_sampling import BorderlineSMOTE
from imblearn.over_sampling import SVMSMOTE
from imblearn.over_sampling import ADASYN

oversampleB = BorderlineSMOTE(random_state=1)
X_trainb, y_trainb = oversampleB.fit_resample(X_train, y_train)
oversampleS = SVMSMOTE(random_state=1)
oversampleA = ADASYN(random_state=1)

steps = [("scaler", StandardScaler()),('over', over), ('under', under),
         ('model', LogisticRegression())]
logRegBS_pip = Pipeline(steps=steps)
```

```
# evaluate pipeline
y_train_pred_lrBS=cross_val_predict(logRegBS_pip,X_train,y_train,cv = 10
                                    )
print(classification_report(y_train, y_train_pred_lrBS))
# Plot ROC curve and calculate and calculate AUC metric
logRegBS_pip.fit(X_train, y_train)
plot_roc_curve(logRegBS_pip, X_test, y_test)
y_test_pred_lrBS=logRegBS_pip.predict(X_test)
plot_conf_mat(y_test, y_test_pred_lrBS)
print(classification_report(y_test, y_test_pred_lrBS))


## KNN
steps = [("scaler", StandardScaler()),('over', over), ('under', under),
         ('model', KNeighborsClassifier())]
KNNS_pip = Pipeline(steps=steps)

y_train_pred_KNNS=cross_val_predict(KNNS_pip,X_train,y_train,cv = 10)
print(classification_report(y_train, y_train_pred_KNNS))
KNNS_pip.fit(X_train, y_train)
plot_roc_curve(KNNS_pip, X_test, y_test)
y_test_pred_KNNS=KNNS_pip.predict(X_test)
plot_conf_mat(y_test, y_test_pred_KNNS)
print(classification_report(y_test, y_test_pred_KNNS))


## Random Forest
steps = [("scaler", StandardScaler()),
         ('over', over), ('under', under),
         ('rfc', RandomForestClassifier(random_state=1))]
RFS_pip = Pipeline(steps=steps)

y_train_pred_rfS = cross_val_predict(RFS_pip, X_train, y_train, cv = 10)
print(classification_report(y_train, y_train_pred_rfS))
RFS_pip.fit(X_train, y_train)
plot_roc_curve(RFS_pip, X_test, y_test)
y_test_pred_rfS=RFS_pip.predict(X_test)
plot_conf_mat(y_test, y_test_pred_rfS)
print(classification_report(y_test, y_test_pred_rfS))
```

```
params={
    'rfc__max_depth': [10, 20],
    'rfc__n_estimators': [50, 100, 200, 400],
    'rfc__max_features': ['auto', 'sqrt'],
    'rfc__min_samples_leaf': [2,4,8],
    'rfc__min_samples_split': [10,20]
}

# ## Grid search
# RFS_grid = RandomizedSearchCV(RFS_pip, params, cv=3,n_jobs=-1,scoring
                                    ="f1")
# RFS_grid.fit(X_train, y_train)
# print("Best Parameters for Model:  ",RF_grid.best_params_)

RFS_Random = RandomizedSearchCV(RFS_pip, params, cv=3,n_jobs=-1,
                                scoring="f1", random_state=1)
RFS_Random.fit(X_train, y_train)
print("Best Parameters for Model:  ",RFS_Random.best_params_)

# Best Parameters for Model:   {'rfc__n_estimators': 100,
#                               'rfc__min_samples_split': 10,
#                               'rfc__min_samples_leaf': 8,
#                               'rfc__max_features': 'auto',
#                               'rfc__max_depth': 20}

steps = [("scaler", StandardScaler()),
         ('over', over), ('under', under),
         ('rfb', RandomForestClassifier(random_state=1,
         max_features= 'auto',min_samples_leaf= 8,
         n_estimators= 100, min_samples_split=10,
         max_depth=20))]
RFBS_pip = Pipeline(steps)

y_pred_RFBS = cross_val_predict(RFBS_pip, X_train, y_train, cv = 10)
print(classification_report(y_train, y_pred_RFBS))
RFBS_pip.fit(X_train, y_train)
plot_roc_curve(RFBS_pip, X_test, y_test)
```

```
y_test_pred_RFBS=RFBS_pip.predict(X_test)
print(classification_report(y_test, y_test_pred_RFBS))

plot_conf_mat(y_test, y_test_pred_RFBS)

## importance of variables
importances = RFBS_pip.steps[3][1].feature_importances_
indices = np.argsort(importances)[::-1]
for f in range(X_train.shape[1]-1):
    print("%2d) %-*s %f" % (f, 30, X_train.columns[indices[f]],
                                        importances[indices[f]]))

for i in range(X_train.shape[1]-1):
    plt.barh(i,importances[indices[i]],align='center',
             color=sns.cubehelix_palette(len(indices), start = 1, rot =
                                                    -.75)[i])
    plt.yticks(np.arange(X_train.shape[1]),X_train.columns, fontsize=12)
plt.show()

# plot the ROC curve
confcmap = sns.light_palette("seagreen",as_cmap=True)
f, ax = plt.subplots(1,3,figsize=(15,4))
plot_roc_curve(logRegS_pip, X_test, y_test, ax=ax[0], color="seagreen")
ax[0].set_title("Logistic Regression")
plot_roc_curve(KNNS_pip, X_test, y_test, ax=ax[1], color="seagreen")
ax[1].set_title("KNN")
plot_roc_curve(RFBS_pip, X_test, y_test, ax=ax[2], color="seagreen")
ax[2].set_title("tuned Random Forest")
plt.tight_layout()
plt.show()

# plot the confusion matrix
confcmap = sns.light_palette("seagreen",as_cmap=True)
f, ax = plt.subplots(1,3,figsize=(15,4))
sns.heatmap(confusion_matrix(y_test, y_test_pred_lrs), ax=ax[0],
            annot=True,cbar=True, fmt='g',cmap=confcmap)
ax[0].set_title("Logistic Regression")
sns.heatmap(confusion_matrix(y_test, y_test_pred_KNNS), ax=ax[1],
            annot=True,cbar=True, fmt='g',cmap=confcmap)
```

```
ax[1].set_title("KNN")
sns.heatmap(confusion_matrix(y_test, y_test_pred_RFBS), ax=ax[2],
            annot=True,cbar=True, fmt='g',cmap=confcmap)
ax[2].set_title("tuned Random Forest")
ax[0].set_xlabel("Predicted label")
ax[1].set_xlabel("Predicted label")
ax[2].set_xlabel("Predicted label")
ax[0].set_ylabel("True label")
plt.tight_layout()
plt.show()
```