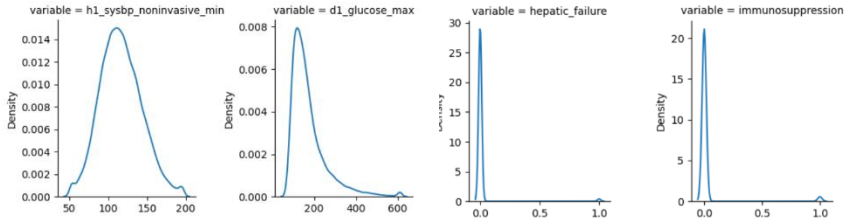
Data mining assignment2

- Data pre-preprocessing

1. Dataset analysis:

#	Column	Non-Null Count	Dtype
0	encounter_id	44939 non-null	int64
1	patient_id	44939 non-null	int64
2	hospital_id	44939 non-null	int64
3	age	42874 non-null	float64
4	bmi	43288 non-null	float64
5	elective_surgery	44939 non-null	int64
6	ethnicity	44239 non-null	object
7	gender	44926 non-null	object
8	height	44308 non-null	float64
9	icu_admit_source	44888 non-null	object
10	icu_id	44939 non-null	int64
11	icu_stay_type	44939 non-null	object
12	icu_type	44939 non-null	object
27	temp_apache	42956 non-null	float64
28	ventilated_apache	44593 non-null	float64
29	d1_diasbp_max	44867 non-null	float64
30	d1_diasbp_min	44867 non-null	float64
31	d1_diasbp_noninvasive_max	44413 non-null	float64
32	d1_diasbp_noninvasive_min	44413 non-null	float64
33	d1_heartrate_max	44874 non-null	float64
34	d1_heartrate_min	44874 non-null	float64
35	d1_mbp_max	44845 non-null	float64
36	d1_mbp_min	44845 non-null	float64
37	d1_mbp_noninvasive_max	44227 non-null	float64
38	d1_mbp_noninvasive_min	44227 non-null	float64
39	d1_resprate_max	44754 non-null	float64
40	d1_resprate_min	44754 non-null	float64
41	d1_spo2_max	44785 non-null	float64
42	d1_spo2_min	44785 non-null	float64

First, I checked about the features of the dataset, discover that there are some types of features of majority, **categorical, continuous, and binary**. Below is frequency distribution of continuous and binary data.



2. Data cleaning:

```
df = df.drop(columns=['encounter_id', 'patient_id', 'hospital_id', 'icu_id'])
df_test = df_test.drop(columns=['encounter_id', 'patient_id', 'hospital_id', 'icu_id'])
df.head()
```

Secondly, I deleted data mentioned above, since they are just some ID, provide no information about patient's health condition, give huge noise while doing ML process.

3. Data transformation:

```
    elif mode=='one_hot':
        non_numeric = df.select_dtypes(exclude=['number']).columns
        df_non_numeric = df[non_numeric]

        df_non_numeric = pd.get_dummies(df_non_numeric)
        return df_non_numeric

df_non_numeric = encoding(df, mode='one_hot')
df_test_nan = encoding(df_test, mode='one_hot')
df_test.head()
```

Then, I decided to do **one-hot encoding**, to **convert categorical data into binary.**

4. Data imputation:

```
d1_potassium_max            4748
d1_potassium_min            4748
h1_mbp_noninvasive_min      4495
h1_mbp_noninvasive_max      4495
apache_4a_icu_death_prob    3891
                            ...
icu_stay_type                  0
icu_type                       0
pre_icu_los_days               0
patient_id                     0
encounter_id                   0
Length: 83, dtype: int64
```

```python
def fill0(df):
    print(df.isnull().sum(axis = 0).sort_values(ascending = False))

    #threshold = int(0.95*df.shape[0])
    #df = df.dropna(axis=1, thresh=threshold)

    for column in df.columns:
        df[column].fillna(df[column].mode()[0], inplace=True)

    print(df.isnull().sum(axis = 0).sort_values(ascending = False))
```

I checked about the number of 0s in those column having missing values, there's no column that loss data too much, so I didn't prune any column, but just fill these NaN with **mode** of the column.

5. Data imbalance handling:

```python
print((Y == 0).mean() * 100)
```

```
has_died    91.370524
dtype: float64
```

```python
df = df.astype(float)
smote = SMOTE(random_state=42)
df_resampled, Y_resampled = smote.fit_resample(df, Y)
```

There is obvious data imbalance, for not to lose any important information from those data with result 0, my method to handle it is **oversampling**. And to prevent overfitting because of repeatedly calculate same data, I used **SMOTE** method to synthesis data.

● Classification Methods:

Below are my functions used for data classification:

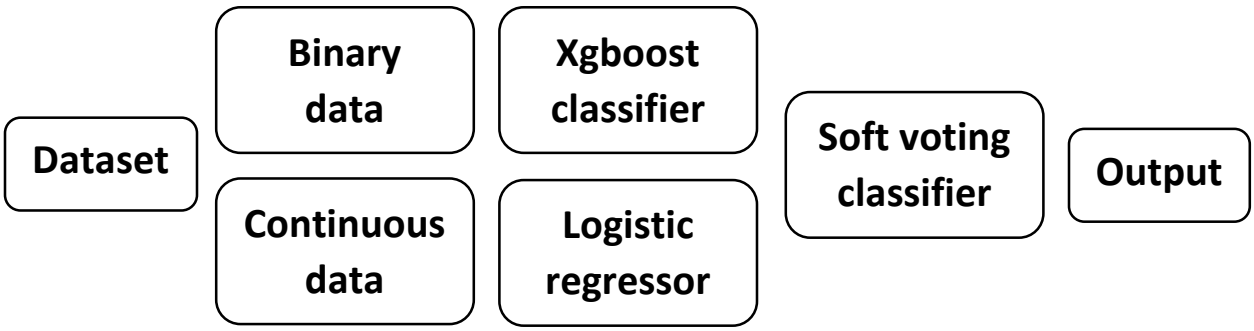
```python
from sklearn.ensemble import VotingClassifier

from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import f1_score, confusion_matrix, ConfusionMatrixDisplay, roc_auc_score, roc_curve

from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
```

Firstly, I split the resampled data into two part: **data with binary values** (including one-hot encoded categorical data) and **those with continuous values**.

```python
df_binary, df_continuous = bin_cont_split(df_resampled)
df_test_binary, df_test_cont = bin_cont_split(df_test)
```

Then I threw these two part into **XGBClassifier** and **LogisticRegression**, and using grid search with cross validation to find their best hyper-parameter respectively.

After, these two model is put into a **soft voting classifier**, below is the structure:

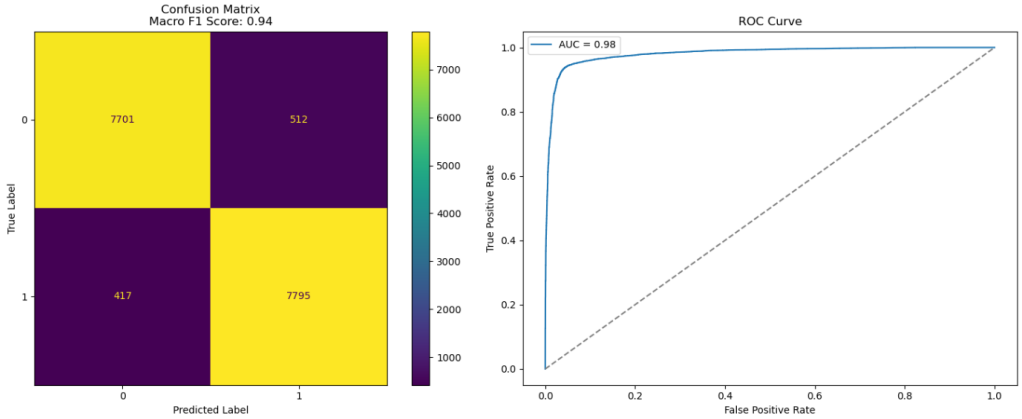


Lastly, we will use cross validation, to be careful, despite over sampling, **I always use stratified k-fold to cut it into 5 slices**. Because of model complexity, **I don't use cross_val_score directly**. Train two base model first, then train the voting classifier. A base model doesn't use unseen features for prediction, and voting classifier doesn't re-fit the base model if it is pre-fitted, which makes this kind of multi model structure achievable.

```
for i, (train_index, test_index) in enumerate(skf.split(df_resampled, Y_resampled)):
    print('Fold:', i+1)
    bin_train_fold, cont_train_fold = df_binary.iloc[train_index], df_continuous.iloc[train_index]

    X_train_fold, X_test_fold = df_resampled.iloc[train_index], df_resampled.iloc[test_index]
    y_train_fold, y_test_fold = Y_resampled.iloc[train_index], Y_resampled.iloc[test_index]

    print('\tBase model fitting...')
    xgb_clf.fit(bin_train_fold, y_train_fold)
    lr_gre.fit(cont_train_fold, y_train_fold)

    print('\tVoting classifier fitting...')
    vot_clf.fit(X_train_fold, y_train_fold)

    print('\tVisualizing...')
    # Call the function for each fold and pass the corresponding subplot axes
    visualize(vot_clf, X_test_fold, y_test_fold, axes[i, 0], axes[i, 1])
```

- # Results & Analysis:

Below is the visualization of validation result, I use **stratified-k-fold** on **original dataset** (not oversampled), which will be shown every fold.



The high macro f1-score and AUROC is maintained in another 4 Fold.

Below is my top20 features for classification (**one-hot encoding backtracked**):

```
'gender',
'apache_3j_diagnosis',
'apache_3j_bodysystem',
'apache_post_operative',
'apache_2_bodysystem',
'h1_mbp_min',
'icu_admit_source',
'icu_stay_type',
'h1_sysbp_noninvasive_max',
'ethnicity',
'age',
'heart_rate_apache',
'd1_sysbp_max',
'd1_diasbp_noninvasive_min',
'd1_sysbp_noninvasive_max',
'h1_heartrate_max',
'h1_spo2_min',
'd1_sysbp_noninvasive_min',
'd1_heartrate_min',
'd1_diasbp_max',
```