A Predictive Model to Identify AMI

Oct 9, 2018

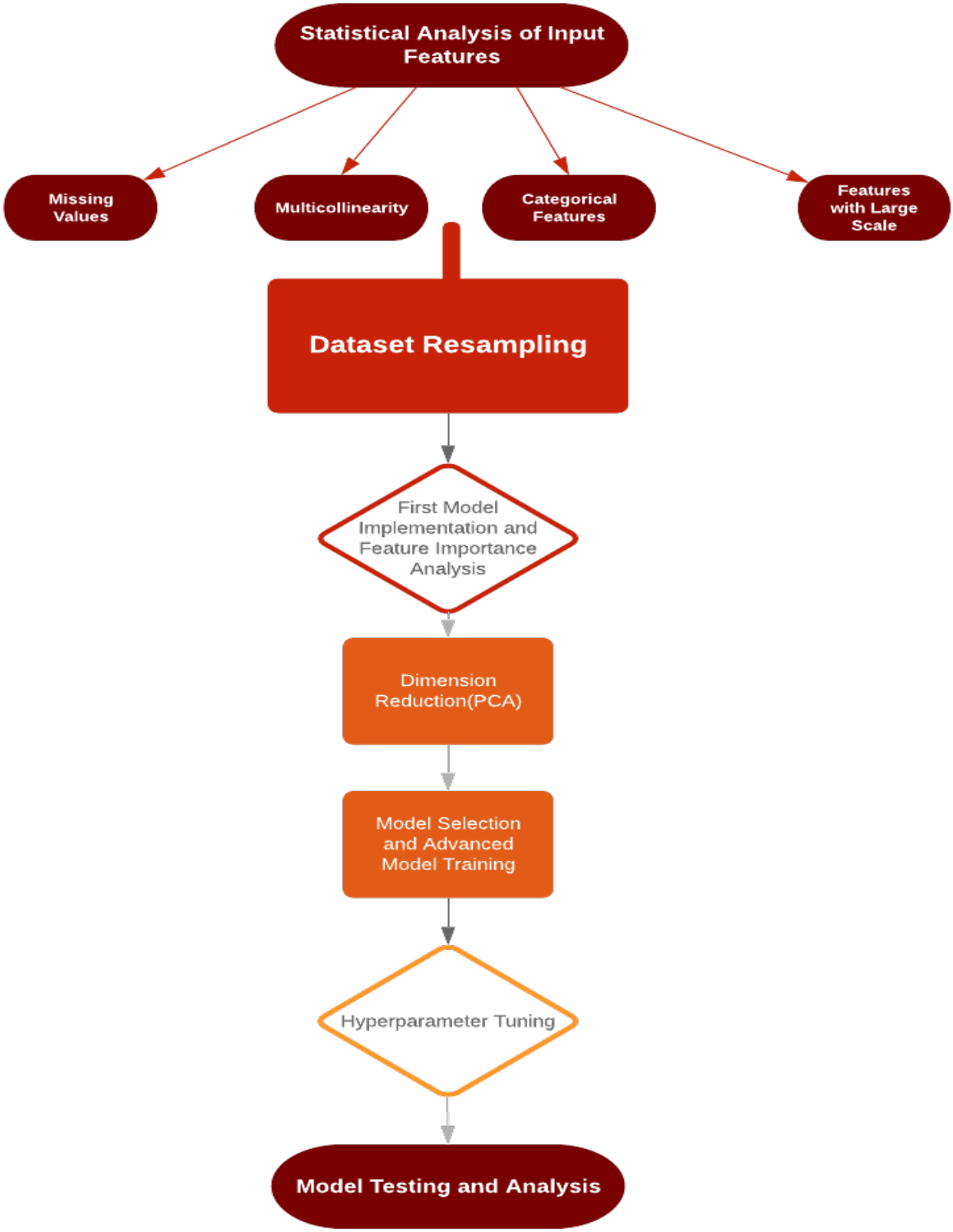
Introduction

Heart attack, the No.1 cause of deaths in the US, not only incurs a large amount of cost directly or indirectly every year, but also greatly debilitates patients and brings disastrous influence on their family. If we could know in advance, from the data provided by health insurance company, who is likely to have Acute Myocardial Infarctions (AMI) in the near future and the potential reasons, then the healthcare company could provide specific suggestions and help to their customers to avoid the potential occurrence of heart attack. Not only will the customers benefit from it, but the insurance company also saves a great amount of healthcare expenditure. It is certainly a win-win situation. So, the goal of our project is to accurately predict, for customers enrolled in the MA/MAPD plan, who is most likely to have AMI in the near future.

We are given all kinds of data of members continuously enrolled in an MA/MAPD plan from Jan. 1, 2016 to Mar. 31, 2017, and our problems are how to extract important features and construct a supervised learning model to predict whether they will have AMI in the next 3 months. To provide the optimal solution, we first perfromed the data preprocessing and feature engineering, then conducted the model selection and parameter tuning. Finally, we came up with the ideal model with the best result in the evaluation metric. Using the model, we can confidently predict who is likely to develop AMI in the future and actions he or she should take to avoid it.

Methodology

Flow Chart:



Data Preparation:

The original dataset contains the information of 100,000 people each with 448 columns of features including customers' id, age, insurance type, and past medical history, etc. Among all features, some are unclear or obviously irrelevant to our data problem, so we manually drop those columns at first. After that, we find many highly interrelated features, for example, the income and the net worth of a person are highly correlated with correlation 0.69, so we keep only net worth for further processing. For every categorical feature, we convert it into the one-hot vector by assigning several dummy variables for it. In terms of missing values, some features only have around 100 missing values, and we just drop these data. But many features have more than 10,000 missing values and dropping them will cause a certain level of information loss. So we impute them using KNN method. To be specific, by for each missing point, we compare its n nearest neighbors and take the average value of target column of those neighbors.

When we check the distribution of the target labels, we discover that the dataset is significantly imbalanced because the number of people who develop AMI is just about 2% of the whole sample. So if we use the original dataset for training, most models will just regard the minority label as the noise and ignore it. By predicting everyone as the majority class (class 0), even the simplest model can easily achieve 98% accuracy. But the result is meaningless and the AUC score is poor. (AUC stands for area under the Receiver Operating Characteristic Curve and is equivalent to the probability that a randomly chosen positive example is ranked higher than a randomly chosen negative example.) Then, handling the imbalance in the target label becomes crucial for this problem. We decide to handle the imbalance by oversampling the minority class because we do not want to further lose information by undersampling the majority class. (We have already lost some information when we removed the features.) But we cannot just randomly oversample the minority class because it will easily trigger overfitting. To be specific, when we randomly oversample a class heavily, it is likely that the train, validate, and test sets all contain same points from the class since each point in the minority class gets copied so many times. In this situation, the test set is meaningless. So to make sure we do not just clone points, we use Smote + ENN combined method to tackle the imbalance data. The Smote method first locates n neighbors for a point A and synthesize m new points in total, each at a randomly picked position on the line connecting A and those m neighbors. Then, ENN method will check the n nearest neighbors of each point and remove it when its neighbors are predominantly from other class to reducing overfitting caused by oversampling. After the process, the ratio of minority class became 0.247. Then we use the standard scaler to scale the data and get it ready for training.

Model Selection and Training:

After the preprocessing, we move into the model training and testing part. The first step is to split the dataset. As it is a huge dataset, we just split them randomly into 3 sets: training set (0.64), validating set (0.16), and testing set (0.2). Since we need to provide the most relevant features for AMI prediction, the tree-based method is most appropriate due to its high interpretability. But to handle its shortcomings: easily-overfitting, we use an advanced bagging method, Random Forest to train the model and print out the most important features. The top ten features are:

- CV_CHF: Member diagnosed with congestive heart failure in 2017
- CV_HDZ: Member diagnosed with other heart diseases in 2017
- RECON_MA_RISK_SCORE_NBR: Medical Risk Score calculated by CMS
- RX_THER_32_YR2016: Number of prescriptions related to ANTIANGINAL AGENTS in 2016
- CV_CAD: Member diagnosed with coronary artery disease in 2017
- CON_VISIT_11_Q01: Number of visits or admissions related to CHRONIC KIDNEY DISEASE in Q1 2016
- CV_SNS: Member diagnosed with cardiovascular signs and symptoms in 2017
- Diab_Complications: Member diagnosed with diabetes complications in 2017
- CV_CIR: Member diagnosed with the circulatory disease in 2017
- RX_THER_85_YR2016: Number of prescriptions related to HEMATOLOGICAL AGENTS - MISC. in 2016

Before the model selection phase, to optimize the running time and compare different models more efficiently, we use Principal Component Analysis to reduce the dimension of data. PCA is a classical approach to achieve dimension reduction. It uses orthogonal transformation to transfer original data into linear independent (uncorrelated) data points. The basic idea is to choose linearly independent principal components or features so that data points have the highest possible variance, since higher variance indicates more significance of the component. To keep 80% of the information in our data, we reduce the dimension to 203. Then, we use different algorithms such as Support Vector Machine, Gradient Boosting Decision Tree, and Logistic Regression to compare model performance by using the validation set. We find out that the model with the best accuracy and AUC is the gradient boosting method. Gradient boosting decision tree is a typical ensemble method which applies the greedy algorithm to minimize the residual tree by tree. The n-th tree is designed to predict the residual term from the weighted sum of prediction of all the n-1 trees. The final prediction is the weighted sum of all simple tree models. Then by comparing three different popular GBDT methods: Gradient Boosting Classifier, XGBoost, and LightGBM, we get that the LightGBM model has the highest computation speed and slightly better AUC. LightGBM uses histogram subtraction and bins to represent continuous values to optimize the running speed and

memory use. After deciding the ideal model, we take Grid Search method to tune the model and look for the best hyper-parameters including the max of the tree depth, number of estimators used, learning rate, subsample rate. GridSearchCV basically searches for the best parameters in the model exhaustively in the given parameter space to optimize the model performance. Besides, regulation factors should also be added to prevent the overfitting of the model by implementing L1 and L2 regularization. Then, we plug the parameter from Grid Search in the LightGBM, feed the training data to it, and finally test its performance on the test set that it has never encountered before. The result is ideal as we get over 99% both in accuracy and in AUC score.

Discussion

There are several major causes for heart attack, including but not restricted to Coronary Artery Disease, Hypertension, Diabetes etc. and we will give customer corresponding advice and recommendations according to the analysis result of their personal situation according to our model.

Generally speaking, we will recommend customers to talk to their doctor about personal risk factors and how to make lifestyle changes to reduce the chances. For example, it is better if they get their cholesterol, blood pressure, and blood sugar checked frequently. If they have high cholesterol, high blood pressure, or diabetes, it's important to pay attention to those conditions and manage at an early stage. Also, customers are recommended to quit smoking, limit their alcohol, stay physically active and incorporate regular exercise into a daily routine since exercise will help with weight control and will therefore lower cholesterol level. Speaking of eating habit, they may be recommended to favor foods that are low in trans fats, saturated fats, simple sugars, and sodium. They should eat whole foods and avoid trans fats, which can be found in many baked goods and deep-fried and processed foods. Getting adequate fiber-rich foods, such as fruits and veggies, whole grains, and beans helps as well. Further to this, limiting salt intake and aim to have two servings of fatty fish, such as salmon, tuna, or mackerel, a week because they're high in heart-healthy omega-3 fats. Moreover, for some customers whose working environment is intense, it would be important for them to learn effective ways to manage their stress. For instance, keeping a positive attitude, being assertive instead of aggressive, asserting feelings, opinions, or beliefs instead of becoming angry, defensive, or passive. It could be a good idea to learn and practice relaxation techniques: try meditation, yoga, or tai-chi for stress management.

When a new customer enrolls in the MA/MAPD plan, the company could input the data of this person after preprocessing to the model and predict whether the customer has a high risk for AMI in the near future. For those who are likely to have AMI, the company may suggest an additional plan that costs more and contains special care to help customers take precautions against AMI. Even if the customers do not accept the new insurance plan, the company may still recommend them for a more proper habit or diet. On the one hand, the company increases the income by selling extra products. On the other hand, the company cuts cost by reducing the number of

patients.

Conclusion

The solution to the business problem could be evaluated from two different angles after one-year implementation. First, the company could check whether the total cost incurred by AMI treatment has shrunk. Second, for those who were predicted to have AMI at the beginning, the company could compute the real incidence of AMI respectively for customers with the extra plan and without. By comparing the result, they could conclude in whether the additional plan was effective.

Code Documentation:

1: Preprocessing:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
```

1.1: Import Raw Data

```
# import raw data
df = pd.read_csv("TAMU_FINAL_DATASET_2018.csv")
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
df.head(10)
```

1.2 Checking Missing Value

```
# checking missing value
df.shape[0] - df.count()
```

Partial Result:

```
ID                                      0
AGE                                     0
```

```
SEX_CD                          30
AMI_FLAG                         0
ESRD_IND                        32
HOSPICE_IND                     32
ORIG_REAS_ENTITLE_CD            30
RECON_MA_RISK_SCORE_NBR          0
RECON_RX_RISK_SCORE_NBR          0
PCP_ASSIGNMENT                 157
DUAL                            30
INSTITUTIONAL                   30
LIS                             30
MCO_HLVL_PLAN_CD                33
MCO_PROD_TYPE_CD                33
CON_VISIT_04_Q01                 0
CON_VISIT_04_Q02                 0
CON_VISIT_04_Q03                 0
CON_VISIT_04_Q04                 0
CON_VISIT_21_Q01                 0
CON_VISIT_21_Q02                 0
CON_VISIT_21_Q03                 0
CON_VISIT_03_Q02                 0
CON_VISIT_03_Q04                 0
CON_VISIT_05_Q02                 0
CON_VISIT_05_Q04                 0
CON_VISIT_09_Q02                 0
CON_VISIT_10_Q02                 0
CON_VISIT_18_Q02                 0
CON_VISIT_19_Q04                 0
CON_VISIT_23_Q02                 0
CON_VISIT_24_Q02                 0
CON_VISIT_30_Q01                 0
CON_VISIT_30_Q02                 0
CON_VISIT_30_Q04                 0
```

1.3: Get a Statistical Description of the Dataset

```
# get a statistical description of the dataset
df.describe()
```

Partial Result:

ID	AGE	AMI_FLAG	ORIG_REAS_ENTITLE_CD	RECON_MA_RISK_SCORE_NBR	RECON_RX_RISK_SCORE_NBR	CON_VISIT_04_Q01	CO
count	100000.000000	100000.000000	100000.000000	99970.000000	100000.000000	100000.000000	10(
mean	50000.500000	72.770440	0.027260	0.289367	1.138959	1.028531	0.4
std	28867.657797	9.715514	0.162841	0.455188	0.946879	0.758254	1.3
min	1.000000	40.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	25000.750000	68.000000	0.000000	0.000000	0.522000	0.570000	0.0
50%	50000.500000	73.000000	0.000000	0.000000	0.862000	0.891000	0.0
75%	75000.250000	79.000000	0.000000	1.000000	1.414000	1.306000	0.0
max	100000.000000	95.000000	1.000000	3.000000			

1.4: Drop Some Missing Values (Small number of missing value)

```
df.dropna(subset=['MCO_HLVL_PLAN_CD'], inplace = True)
df.dropna(subset=['SEX_CD'], inplace = True)
df.dropna(subset=['PCP_ASSIGNMENT'], inplace = True)
df.dropna(subset=['HOSPICE_IND'], inplace = True)
```

1.5: Investigate the Relationship Between Certain Features

```
df[['LIS','Home_value','Est_income','Pct_above_poverty_line','Pct_above_povert
y_line','Est_Net_worth']].corr()
```

	Home_value	Est_income	Pct_above_poverty_line	Pct_above_poverty_line	Est_Net_worth
Home_value	1.000000	0.581651	0.307422	0.307422	0.638142
Est_income	0.581651	1.000000	0.356041	0.356041	0.695535
Pct_above_poverty_line	0.307422	0.356041	1.000000	1.000000	0.388183
Pct_above_poverty_line	0.307422	0.356041	1.000000	1.000000	0.388183
Est_Net_worth	0.638142	0.695535	0.388183	0.388183	1.000000

1.6: Drop Obviously Irrelevant Columns and Columns With High Correlation With Each Other

```
df.drop(columns=
['PCP_ASSIGNMENT','College','LIS','Home_value','Pct_below_poverty_line','Est_i
ncome','Pct_above_poverty_line','ID'],inplace=True)
```

1.7: Convert Catagorical Columns Into Numerical Ones

```
for feature in df.columns:
    if isinstance(df[feature][3],str)==True: #Char Columns
        df=pd.get_dummies(df,columns=[feature],prefix='{}'.format(feature))


df=pd.get_dummies(df,columns=
['ORIG_REAS_ENTITLE_CD'],prefix='ORIG_REAS_ENTITLE_CD')
```

1.8: Normalize Some Columns Using Mean and Std

```
# a glance of the dataset
AA=df.describe()
AA=AA.transpose()
AA[AA['mean']>1]
```

	count	mean	std	min	25%	50%	75%	max
AGE	99839.0	72.781548	9.700022	40.0	68.0000	73.000	79.000	95.000
RECON_MA_RISK_SCORE_NBR	99839.0	1.139303	0.946939	0.0	0.5225	0.863	1.414	15.120
RECON_RX_RISK_SCORE_NBR	99839.0	1.028865	0.758133	0.0	0.5700	0.891	1.306	15.704
POT_VISIT_11_Q01	99839.0	2.471319	3.289834	0.0	0.0000	2.000	3.000	74.000
POT_VISIT_11_Q02	99839.0	2.553952	3.378494	0.0	0.0000	2.000	4.000	73.000
POT_VISIT_11_Q03	99839.0	2.526247	3.435006	0.0	0.0000	2.000	3.000	86.000
POT_VISIT_11_Q04	99839.0	2.496419	3.279913	0.0	0.0000	2.000	3.000	82.000
POT_VISIT_22_Q02	99839.0	1.226304	3.353112	0.0	0.0000	0.000	1.000	111.000
POT_VISIT_22_Q03	99839.0	1.235018	3.522450	0.0	0.0000	0.000	1.000	169.000
POT_VISIT_22_Q04	99839.0	1.228388	3.432794	0.0	0.0000	0.000	1.000	103.000
POT_VISIT_22_Q01	99839.0	1.160258	3.258435	0.0	0.0000	0.000	1.000	114.000
RX_THER_36_YR2016	99839.0	2.962490	3.870506	0.0	0.0000	2.000	4.000	71.000
RX_THER_44_YR2016	99839.0	1.034055	3.719626	0.0	0.0000	0.000	0.000	95.000
RX_THER_58_YR2016	99839.0	1.902393	4.249788	0.0	0.0000	0.000	2.000	87.000
RX_THER_65_YR2016	99839.0	2.101964	4.781319	0.0	0.0000	0.000	1.000	89.000
RX_THER_37_YR2016	99839.0	1.325634	2.910430	0.0	0.0000	0.000	1.000	52.000
RX_THER_39_YR2016	99839.0	2.663208	3.519186	0.0	0.0000	1.000	4.000	77.000
RX_THER_49_YR2016	99839.0	1.512746	3.100443	0.0	0.0000	0.000	2.000	53.000
RX_THER_28_YR2016	99839.0	1.017568	2.548064	0.0	0.0000	0.000	0.000	59.000
RX_THER_27_YR2016	99839.0	2.023458	4.778523	0.0	0.0000	0.000	1.000	121.000
RX_THER_34_YR2016	99839.0	1.307044	2.677798	0.0	0.0000	0.000	1.000	52.000
RX_THER_72_YR2016	99839.0	1.281143	3.578794	0.0	0.0000	0.000	0.000	74.000
RX_THER_33_YR2016	99839.0	1.649876	2.974836	0.0	0.0000	0.000	3.000	66.000
Education_level	87683.0	3.792286	0.873933	0.0	3.0000	4.000	4.000	8.000
Length_residence	87683.0	16.718292	19.281282	0.0	5.0000	13.000	19.000	99.000
Est_BMI_decile	87683.0	4.284696	2.608054	0.0	2.0000	4.000	6.000	9.000
Num_person_household	87683.0	2.424392	1.723475	1.0	1.0000	2.000	3.000	17.000
Decile_struggle_Med_lang	74670.0	5.949136	2.737100	0.0	4.0000	7.000	8.000	9.000
Est_Net_worth	87683.0	231750.966550	292006.641220	-2500.0	2500.0000	125000.000	312500.000	1000000.(
Index_Health_ins_engage	87683.0	2.444602	2.198213	0.0	1.0000	2.000	4.000	9.000
Index_Health_ins_influence	87683.0	3.119134	2.390179	0.0	1.0000	3.000	5.000	9.000
Population_density_centile_ST	87683.0	53.301507	27.856814	0.0	30.0000	54.000	78.000	99.000
Population_density_centile_US	87683.0	59.037407	25.702157	0.0	38.0000	61.000	82.000	99.000

```
# Normalization
for i in
['Est_Net_worth','AGE','Population_density_centile_ST','Population_density_cen
tile_US']:
    df[i]=(df[i]-df[i].mean())/df[i].std()
```

1.9: Fill NaN Values Using KNN k=6

```
filter_feature = ['AMI_FLAG'] # filter useless feature
features = []
for x in df.columns:
    if x not in filter_feature:
        features.append(x)
train_data_x = df[features]
train_data_y = df['AMI_FLAG']

from fancyimpute import KNN
train_data_x = pd.DataFrame(KNN(k=6).fit_transform(train_data_x),
columns=features)
```

```
# make sure all missing values are gone
df.shape[0] - df.count()
```

1.10: Deal with Multicollinearity by Dropping One Column of Each Kind of Dummy We Created

```
df.drop(columns=['SEX_CD_F','ESRD_IND_N', 'HOSPICE_IND_N',
'DUAL_N','INSTITUTIONAL_N','MCO_HLVL_PLAN_CD_MA','Diab_Type_Diabetes
Unspeci','Dwelling_Type_A','ORIG_REAS_ENTITLE_CD_0.0'],inplace=True)
```

1.11: Now we finish the preprossessing step and save data

```
df.to_csv('Fancy_Humana_data.csv')
```

2: Dealing with Imbalanced Dataset

2.1:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.metrics import classification_report
from imblearn.combine import SMOTEENN
%matplotlib inline
```

```
df = pd.read_csv("/Users/yuqidai/Downloads/Fancy_Humana_data.csv")
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
df.head()
```

2.2: Calculate the percentage of those don't have AMI

```
(99839-df[df['AMI_FLAG']==1].count()['AGE'])/99839
```

2.3: Detailed ratio of both classes

```
Number_count1=df[df['AMI_FLAG']==1].count()[0]
Number_count0=df[df['AMI_FLAG']==0].count()[0]
print('Class 0:', Number_count0)
print('Class 1:', Number_count1)
print('Proportion:', round(Number_count0 / Number_count1, 2), ': 1')
```

Note: We can see that the data is very imbalanced, if we use it for training, the model will regard minority class as noice and just predict everyone as label 0.

2.4: Exclude the target label

```
df_feature = df.drop(['AMI_FLAG'], axis=1)
```

2.5: Scale the data

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df_feature)
scaled_data = scaler.transform(df_feature)
```

2.6: Use Smote+ENN method to oversample the minority label (preventing the overfitting)

```
smn = SMOTEENN(k=5,ratio=0.3,m=5,kind_smote='borderline1')
X_resampled, y_resampled = sm.fit_sample(X, y)
```

Ratio of class 0 after oversampling

```
y_resampled[y_resampled==1].shape[0]/y_resampled.shape[0]
```

Result:

```
0.2474046435234363
```

2.7: Organize and save data processed

```
finalfeature=pd.DataFrame(X_resampled)
finallabel=pd.DataFrame(y_resampled)
#Save the data ready for training
finalfeature.to_csv('super_fancy_feature.csv')
finallabel.to_csv('super_fancy_label.csv')
```

3: Analyze feature importance

- get the feature variables

```
df_feature = pd.read_csv("/Users/nicoledu/Downloads/super_fancy_feature.csv")
```

- get corresponding labels

```
column_names = ['ID','AMI_FLAG']
df_label = pd.read_csv("/Users/nicoledu/Downloads/super_fancy_label.csv",
header = None, names = column_names).shift(-1)
df_label = df_label[:-1]
X = df_feature
y = df_label
```

Use Random Forest to train the data(Starting from a simple model as a benchmark

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, max_depth=2, random_state=0)
clf.fit(X,y)
```

Get the most important features

```
feature_importance = clf.feature_importances_
import numpy as np
for i in range(1, 21):
    print(X.columns[np.argsort(feature_importance)[-i]])
```

Output:

```
CV_CHF
CV_HDZ
RECON_MA_RISK_SCORE_NBR
RX_THER_32_YR2016
CV_CAD
CON_VISIT_11_Q01
CV_SNS
Diab_Complications
CV_CIR
RX_THER_85_YR2016
CV_CER
POT_VISIT_21_Q04
CON_VISIT_11_Q02
RECON_RX_RISK_SCORE_NBR
RES_FAIL
CON_VISIT_02_Q01
AGE
CON_VISIT_11_Q03
```

```
ESRD_IND_Y
CV_PVD
```

Note: We use Random Forest to analyze feature importance, but hte validation result is not good enough. Also, other algorithms such as SVM are applied as well and the performance of **lightgbm** is the most efficient.

4: Algorithm Training:

4.1: Import relevant libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import xgboost as xgb
import lightgbm as lgb
from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.metrics import classification_report
from imblearn.combine import SMOTEENN
from imblearn.under_sampling import TomekLinks

%matplotlib inline
```

4.2: Calculate the percentage after resampling

```
total_number = df_label.shape[0]
(total_number-df_label[df_label['AMI_FLAG']==1].count()[0])/total_number
```

result:

```
0.7525953564765637
```

4.3: Group description

```
Number_count1=df_label[df_label['AMI_FLAG']==1].count()[0]
Number_count0=df_label[df_label['AMI_FLAG']==0].count()[0]
print('Class 0:', Number_count0)
print('Class 1:', Number_count1)
print('Proportion:', round(Number_count0 / Number_count1, 2), ': 1')
```

result:

```
Class 0: 84311
Class 1: 27716
Proportion: 3.04 : 1
```

4.4: PCA

```
# scale the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df_feature)
scaled_data = scaler.transform(df_feature)

# PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=0.8)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)
print(scaled_data.shape)
print(x_pca.shape)

X = x_pca
y = df_label['AMI_FLAG']
```

Result:

```
(112027, 452)
(112027, 203)
```

4.5: Split the data into three parts

```
X_train1, X_test, y_train1, y_test = train_test_split(X, y, test_size=0.2,
random_state=1)

X_train, X_val, y_train, y_val = train_test_split(X_train1, y_train1,
test_size=0.2, random_state=1)
```

4.6: LightGBM using random parameters(first attempt)

```
modelLGB=lgb.LGBMClassifier(max_depth=8,learning_rate=0.05,n_estimators=150)
modelLGB.fit(X_train,y_train)

print('Accuracy: {}'.format(modelLGB.score(X_val, y_val)))
print('AUC Score: {}'.format(metrics.roc_auc_score(y_val,
modelLGB.predict_proba(X_val)[:,1])))
predictions=modelLGB.predict(X_val)
print(classification_report(y_val,predictions))
```

Result:

```
Accuracy: 0.9677545327754533
AUC Score: 0.9932240919628577
             precision    recall  f1-score   support

        0.0       0.98      0.98      0.98     13430
        1.0       0.94      0.93      0.94      4495

avg / total       0.97      0.97      0.97     17925
```

4.7: Tuning using Grid Search

```
from sklearn.model_selection import GridSearchCV

params = {'boosting_type': 'gbdt',
          'max_depth' : 8,
          'objective': 'binary',
          'nthread': 3, # Updated from nthread
          'num_leaves': 70,
```

```
          'learning_rate': 0.05,
          'max_bin': 300,
          'subsample_for_bin': 200,
          'subsample': 1,
          'subsample_freq': 1,
          'colsample_bytree': 0.8,
          'min_split_gain': 0.5,
          'min_child_weight': 1,
          'min_child_samples': 5,
          'scale_pos_weight': 1,
          'num_class' : 1,
          'metric' : 'binary_error'}

# Create parameters to search

gridParams = {
    'learning_rate': [0.5],
    'max_depth' : [6,7,8],
    'n_estimators': [140],
    'random_state' : [501], # Updated from 'seed'
    'colsample_bytree' : [0.65, 0.66],
    'subsample' : [0.7,0.75],
    'reg_alpha' : [1,1.2],
    'reg_lambda' : [1,1.2,1.4],
    }

mdl = lgb.LGBMClassifier(boosting_type= 'gbdt',
          objective = 'binary',
          n_jobs = 3, # Updated from 'nthread'
          silent = True,
          max_depth = params['max_depth'],
          max_bin = params['max_bin'],
          subsample_for_bin = params['subsample_for_bin'],
          subsample = params['subsample'],
          subsample_freq = params['subsample_freq'],
          min_split_gain = params['min_split_gain'],
          min_child_weight = params['min_child_weight'],
          min_child_samples = params['min_child_samples'],
          scale_pos_weight = params['scale_pos_weight'])

# To view the default model params:

mdl.get_params().keys()

# Create the grid
```

```
grid = GridSearchCV(mdl, gridParams,
                    verbose=0,
                    cv=4,
                    n_jobs=2)

# Run the grid

grid.fit(X_train, y_train)

# Print the best parameters found

print(grid.best_params_)
print(grid.best_score_)
```

Output:

```
{'colsample_bytree': 0.65, 'learning_rate': 0.5, 'max_depth': 6,
'n_estimators': 140, 'random_state': 501, 'reg_alpha': 1.2, 'reg_lambda': 1.2,
'subsample': 0.75}
0.9846713903146619
```

4.8: Use the parameters from above Grid Search

```
modelLGB=lgb.LGBMClassifier(max_depth=6,colsample_bytree=0.65,
learning_rate=0.5,n_estimators=140,reg_lambda=1.2,subsample=0.75,
reg_alpha=1.2)
modelLGB.fit(X_train,y_train)

print('Accuracy: {}'.format(modelLGB.score(X_val, y_val)))
print('AUC Score: {}'.format(metrics.roc_auc_score(y_val,
modelLGB.predict_proba(X_val)[:,1])))
predictions=modelLGB.predict(X_val)
print(classification_report(y_val,predictions))
```

Output:

```
Accuracy: 0.9913528591352859
AUC Score: 0.9991808885027379
             precision    recall  f1-score   support

        0.0       1.00      0.99      0.99     13430
        1.0       0.98      0.99      0.98      4495

avg / total       0.99      0.99      0.99     17925
```