Group: Zeru Zhou's Group

Name: Zeru Zhou

Email: zeruzhou9@gmail.com

Country: United States

University: University of Southern California

Specialization: Data Science

Github Repo:

Problem Description

In this project, I need to build binery classification machine learning models to predict if the bank clients will renew the term deposit or not in order to make corresponding strategies to maintain clients.

Dataset Information

The dataset contains 3 parts: Client information, compaign information and social/economical context.

There are total 41188 instances and 20columns.

The target response is whether the bank client will renew the term deposit.

Library importing

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.linear_model import RidgeClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from imblearn.pipeline import Pipeline
#from sklearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, precision_score,
```

```
from sklearn.model_selection import train_test_split
import warnings
from sklearn.exceptions import ConvergenceWarning
warnings.filterwarnings("ignore", category=ConvergenceWarning)
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import RFECV
```

Data uploading

In [2]:
```
my_df = pd.read_csv('../data/bank-additional/bank-additional-full.csv', sep=';')
```

In [3]:
```
my_df.shape
```

Out[3]:
```
(41188, 21)
```

In [4]:
```
my_df.head()
```

Out[4]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	cam
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	

5 rows × 21 columns

In [5]:
```
my_df['y'].value_counts()
```

Out[5]:
```
no     36548
yes     4640
Name: y, dtype: int64
```

In [6]:
```
my_df.isnull().any()
```

Out[6]:
```
age               False
job               False
marital           False
education         False
default           False
housing           False
loan              False
contact           False
month             False
day_of_week       False
duration          False
campaign          False
pdays             False
previous          False
poutcome          False
emp.var.rate      False
cons.price.idx    False
cons.conf.idx     False
euribor3m         False
nr.employed       False
y                 False
dtype: bool
```

In [7]:
```
sub = []
for i in my_df.isnull().any().keys():
    if my_df.isnull().any()[i] == False:
        sub.append(i)
sub
```

Out[7]:
```
['age',
 'job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'day_of_week',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'poutcome',
 'emp.var.rate',
 'cons.price.idx',
 'cons.conf.idx',
 'euribor3m',
 'nr.employed',
 'y']
```

As we can see, this is an imbalanced dataset and it has NA values. We need to clean the dataset before moving to any analysis and model building.

Data Cleaning and imputation

Drop duplicated rows

In [8]:
```
my_df.duplicated(subset= sub).value_counts()
```

Out[8]:
```
False    41176
True        12
dtype: int64
```

In [9]:
```
my_df = my_df.drop_duplicates()
my_df.shape
```

Out[9]:
```
(41176, 21)
```

Impute missing values

First, find categorical variables

In [78]:
```
my_df.dtypes
```

Out[78]:
```
age                     int64
job                    object
marital                object
education              object
default                object
balance               float64
```

```
housing             object
loan                object
contact             object
day                float64
month               object
duration             int64
campaign             int64
pdays                int64
previous             int64
poutcome            object
y                   object
day_of_week         object
emp.var.rate       float64
cons.price.idx     float64
cons.conf.idx      float64
euribor3m          float64
nr.employed        float64
dtype: object
```

In [80]:
```python
my_dict = {}
for n,i in enumerate(my_df.columns):
    my_dict[i] = my_df.dtypes[n]
type_df = pd.DataFrame(my_dict, index=['dtype'])
type_df
```

Out[80]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	durati
dtype	int64	object	object	object	object	float64	object	object	object	float64	object	int

In [86]:
```python
type_df = type_df.T
type_df
```

Out[86]:

	dtype
age	int64
job	object
marital	object
education	object
default	object
balance	float64
housing	object
loan	object
contact	object
day	float64
month	object
duration	int64
campaign	int64
pdays	int64
previous	int64
poutcome	object
y	object
day_of_week	object

emp.var.rate	float64
cons.price.idx	float64
cons.conf.idx	float64
euribor3m	float64
nr.employed	float64

In [90]:
```
col = type_df.loc[type_df['dtype'] == 'object']
categorical = list(col.index)
categorical.remove('y')
categorical
```

Out[90]:
```
['job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'poutcome',
 'day_of_week']
```

In [91]:
```
df = pd.get_dummies(my_df, columns=categorical)
```

In [93]:
```
df.head()
```

Out[93]:

	age	balance	day	duration	campaign	pdays	previous	y	emp.var.rate	cons.price.idx	cons.conf.id
0	58	2143.0	5.0	261	1	-1	0	no	NaN	NaN	NaN
1	44	29.0	5.0	151	1	-1	0	no	NaN	NaN	NaN
2	33	2.0	5.0	76	1	-1	0	no	NaN	NaN	NaN
3	47	1506.0	5.0	92	1	-1	0	no	NaN	NaN	NaN
4	33	1.0	5.0	198	1	-1	0	no	NaN	NaN	NaN

In [94]:
```
x,y = df.drop(columns=['y']), df['y']
```

First, we can try simple imputer like filling the value with mean

In [95]:
```
x
```

Out[95]:

	age	balance	day	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.ic
0	58	2143.0	5.0	261	1	-1	0	NaN	NaN	Na
1	44	29.0	5.0	151	1	-1	0	NaN	NaN	Na
2	33	2.0	5.0	76	1	-1	0	NaN	NaN	Na
3	47	1506.0	5.0	92	1	-1	0	NaN	NaN	Na
4	33	1.0	5.0	198	1	-1	0	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	...	
41183	73	NaN	NaN	334	1	999	0	-1.1	94.767	-50
41184	46	NaN	NaN	383	1	999	0	-1.1	94.767	-50

41185	56	NaN	NaN	189	2	999	0	-1.1	94.767	-50
41186	44	NaN	NaN	442	1	999	0	-1.1	94.767	-50
41187	74	NaN	NaN	239	3	999	1	-1.1	94.767	-50

86387 rows × 73 columns

In [104…

```
x = x.drop(columns=['duration'])
```

In [105…

```
x_simple = SimpleImputer(strategy='mean').fit_transform(x)
pd.DataFrame(x_simple).head()
```

Out[105]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	58.0	2143.0	5.0	1.0	-1.0	0.0	0.081922	93.57572	-40.502863	3.621293	5167.03487	0.0	0.0	0.0
1	44.0	29.0	5.0	1.0	-1.0	0.0	0.081922	93.57572	-40.502863	3.621293	5167.03487	0.0	0.0	0.0
2	33.0	2.0	5.0	1.0	-1.0	0.0	0.081922	93.57572	-40.502863	3.621293	5167.03487	0.0	0.0	1.0
3	47.0	1506.0	5.0	1.0	-1.0	0.0	0.081922	93.57572	-40.502863	3.621293	5167.03487	0.0	1.0	0.0
4	33.0	1.0	5.0	1.0	-1.0	0.0	0.081922	93.57572	-40.502863	3.621293	5167.03487	0.0	0.0	0.0

Since this is high-dimensional dataset with almost 100k instances, using simple imputer may lead to huge error. So we try iterative imputer using round robin algorithm

In [134…

```
x_iter = IterativeImputer(n_nearest_features=50).fit_transform(x)
x_iter_df = pd.DataFrame(x_iter)
```

In [108…

```
x_iter_df.head()
```

Out[108]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	58.0	2143.0	5.0	1.0	-1.0	0.0	-0.001759	93.460615	-37.686446	3.690600	5166.507093	0.0	0.0
1	44.0	29.0	5.0	1.0	-1.0	0.0	0.013415	93.471578	-38.045953	3.665590	5165.584233	0.0	0.0
2	33.0	2.0	5.0	1.0	-1.0	0.0	-0.012448	93.451250	-38.446562	3.637786	5166.385384	0.0	0.0
3	47.0	1506.0	5.0	1.0	-1.0	0.0	0.008979	93.502296	-38.150717	3.642176	5163.373385	0.0	1.0
4	33.0	1.0	5.0	1.0	-1.0	0.0	-0.024682	93.495518	-37.526338	3.675858	5162.429888	0.0	0.0

After filling all the NAs, the next step is to eliminate some outliers

In [109…

```
x_iter_df.describe()
```

Out[109]:

	0	1	2	3	4	5	
count	86387.000000	86387.000000	86387.000000	86387.000000	86387.000000	86387.000000	86387.0
mean	40.501314	1068.949277	14.197018	2.670437	479.792504	0.386181	0.0
std	10.534612	2267.890700	6.665170	2.947981	483.824356	1.713173	1.1
min	17.000000	-8019.000000	1.000000	1.000000	-1.000000	0.000000	-3.4
25%	32.000000	163.000000	10.445861	1.000000	-1.000000	0.000000	-0.2
50%	39.000000	584.000000	13.730860	2.000000	246.000000	0.000000	0.0

75%	48.000000	1211.586290	17.166562	3.000000	999.000000	0.000000	1.1
max	98.000000	102127.000000	36.740440	63.000000	999.000000	275.000000	10.8

In [126…
```
y
```

Out[126]:
```
0        no
1        no
2        no
3        no
4        no
       ...
41183    yes
41184     no
41185     no
41186    yes
41187     no
Name: y, Length: 86387, dtype: object
```

In [135…
```
x_iter_df['y'] = y.values
```

In [17]:
```
my_list = []
for key, value in my_df.dtypes.items():
    if value == 'object':
        my_list.append(key)
my_list.remove('y')
my_list
```

Out[17]:
```
['job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'day_of_week',
 'poutcome']
```

In [18]:
```
my_df = pd.get_dummies(my_df, columns=my_list)
my_df
```

Out[18]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m
0	56	261	1	999	0	1.1	93.994	-36.4	4.857
1	57	149	1	999	0	1.1	93.994	-36.4	4.857
2	37	226	1	999	0	1.1	93.994	-36.4	4.857
3	40	151	1	999	0	1.1	93.994	-36.4	4.857
4	56	307	1	999	0	1.1	93.994	-36.4	4.857
...	...	...	...	...	...	...	...	...	...
41181	37	281	1	999	0	-1.1	94.767	-50.8	1.028
41182	29	112	1	9	1	-1.1	94.767	-50.8	1.028
41184	46	383	1	999	0	-1.1	94.767	-50.8	1.028
41185	56	189	2	999	0	-1.1	94.767	-50.8	1.028
41186	44	442	1	999	0	-1.1	94.767	-50.8	1.028

40428 rows × 64 columns

In [19]:
```
for i in my_df.columns:
    if i == 'y':
        continue
    q_low = my_df[i].quantile(0.01)
    q_high = my_df[i].quantile(0.99)
    my_df = my_df.loc[(my_df[i] <= q_high) & (my_df[i] >= q_low)]
```

In [20]:
```
my_df.shape
```

Out[20]:
```
(36103, 64)
```

In [30]:
```
my_df = my_df.drop(columns=['duration'])
my_df.shape
```

Out[30]:
```
(36103, 63)
```

Here above, For each feature, I only dropped very extreme values that not belong to the central 99%. It is also feasible to use q1, q3, and 1.5 IQR to detect outliers but I don't want that much data lose.

Model Building

For each model, we'll build pipeline for feature selection and hyperparameter tuning, using cross validation. Also SMOTE is included since this is imbalanced dataset.

Split the dataset into train/test

In [31]:
```
x,y = my_df.drop(columns=['y']), my_df['y']
```

In [32]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=42
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[32]:
```
((32492, 62), (3611, 62), (32492,), (3611,))
```

Logistic Regression

For Logistic Regression, we can perform feature selection by using l1 peanlization, it is a shrink method by reducing the coefficients of irrelavent variables to 0. We can also use other methods but this is the most convinent way. We do feature selection together with hyperparameter tuning.

In [33]:
```
log = LogisticRegression(penalty='l1', solver='saga', max_iter=300)
```

In [34]:
```
scale = MinMaxScaler()
```

In [35]:
```
imbalance = SMOTE()
```

In [36]:
```
param = {'model__C': np.logspace(-3, 3, num=50)}
```

In [37]:
```
x_train_scaled = scale.fit_transform(x_train)
```

In [38]:
```
pipe = Pipeline(steps=[('smote', imbalance), ('model', log)])
```

In [39]:
```
clf_log = GridSearchCV(pipe, param, n_jobs=-1).fit(x_train_scaled, y_train)
```

```
/Users/zhouzeru/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:3
50: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/Users/zhouzeru/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:3
50: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/Users/zhouzeru/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:3
50: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/Users/zhouzeru/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:3
50: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/Users/zhouzeru/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:3
50: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/Users/zhouzeru/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:3
50: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/Users/zhouzeru/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:3
50: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/Users/zhouzeru/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:3
50: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/Users/zhouzeru/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:3
50: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
```

In [40]:
```
x_test_scaled = scale.fit_transform(x_test)
```

In [41]:
```
y_pred = clf_log.predict(x_test_scaled)
```

In [42]:
```
acc = accuracy_score(y_test, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.7864857380227084
```

In [43]:
```
pre = precision_score(y_test, y_pred,  pos_label='no')
print(f'The precision score is {pre}')
```

```
The precision score is 0.9589578872234118
```

In [44]:
```
rec = recall_score(y_test, y_pred,  pos_label='no')
print(f'The recall score is {rec}')
```

```
The recall score is 0.8037690696978762
```

In [45]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[2687  656]
 [ 115  153]]
```

Try RFE instead of l1-peanlty

In [46]:
```
log = LogisticRegression(max_iter=5000, penalty='none')
```

In [47]:
```
feature, target = imbalance.fit_resample(x_train, y_train)
```

In [48]:
```
feature_scaled = scale.fit_transform(feature)
```

In [49]:
```
clf_log_rfe = RFECV(log, cv=5, n_jobs=-1).fit(feature_scaled, target)
```

In [50]:
```
y_pred = clf_log_rfe.predict(x_test_scaled)
```

In [51]:
```
acc = accuracy_score(y_test, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.9277208529493215
```

In [52]:
```
pre = precision_score(y_test, y_pred,  pos_label='no')
print(f'The precision score is {pre}')
```

```
The precision score is 0.9331084879145587
```

In [53]:
```
rec = recall_score(y_test, y_pred,  pos_label='no')
print(f'The recall score is {rec}')
```

```
The recall score is 0.9931199521387974
```

In [54]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[3320   23]
 [ 238   30]]
```

Without class imbalance treatment

In [55]:
```
log = LogisticRegression(max_iter=5000)
```

In [56]:
```
param = {'C': np.logspace(-3, 3, num=50)}
```

In [57]:
```
clf_log_1 = GridSearchCV(log, param, n_jobs=-1).fit(x_train_scaled, y_train)
```

```
/Users/zhouzeru/opt/anaconda3/lib/python3.9/site-packages/joblib/externals/loky/process_
executor.py:702: UserWarning: A worker stopped while some jobs were given to the executo
r. This can be caused by a too short worker timeout or by a memory leak.
  warnings.warn(
```

In [58]:
```
y_pred = clf_log_1.predict(x_test_scaled)
```

In [59]:
```
acc = accuracy_score(y_test, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.9304901689282747
```

In [60]:
```
pre = precision_score(y_test, y_pred,  pos_label='no')
print(f'The precision score is {pre}')
```

```
The precision score is 0.9342696629213483
```

In [61]:
```
rec = recall_score(y_test, y_pred,  pos_label='no')
print(f'The recall score is {rec}')
```

```
The recall score is 0.9949147472330242
```

In [62]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[3326   17]
 [ 234   34]]
```

K-Neighbor

In [63]:
```
knn = KNeighborsClassifier(weights='uniform')
```

In [64]:
```
params = {'n_neighbors':[int(x) for x in np.linspace(1,20,num=20)]}
```

In [65]:
```
clf_knn = GridSearchCV(knn, params, n_jobs=-1).fit(feature_scaled, target)
```

In [66]:
```
y_pred = clf_knn.predict(x_test_scaled)
```

In [67]:
```
acc = accuracy_score(y_test, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.910828025477707
```

In [68]:
```
pre = precision_score(y_test, y_pred,  pos_label='no')
print(f'The precision score is {pre}')
```

```
The precision score is 0.934176487496407
```

In [69]:
```
rec = recall_score(y_test, y_pred,  pos_label='no')
print(f'The recall score is {rec}')
```

```
The recall score is 0.9721806760394855
```

In [70]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[3250   93]
 [ 229   39]]
```

Ensemble Tree

In [71]:
```
rdf = RandomForestClassifier()
```

In [72]:
```
params = {'ccp_alpha': np.logspace(-3, 3, num=20)}
```

In [73]:
```
clf_rdf = GridSearchCV(rdf, params, n_jobs=-1).fit(feature_scaled, target)
```

In [74]:
```
y_pred = clf_rdf.predict(x_test_scaled)
```

In [75]:
```
acc = accuracy_score(y_test, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.8759346441428967
```

In [76]:
```
pre = precision_score(y_test, y_pred,  pos_label='no')
print(f'The precision score is {pre}')
```

```
The precision score is 0.9570571518787496
```

In [77]:
```
rec = recall_score(y_test, y_pred,  pos_label='no')
print(f'The recall score is {rec}')
```

```
The recall score is 0.9066706551002094
```

In [78]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[3031  312]
 [ 136  132]]
```

Boosting Tree

In [79]:
```
xgb = XGBClassifier(eta = 0.01, objective = 'binary:logistic')
```

In [80]:
```
params = {'reg_alpha': np.logspace(-3, 3, 10)}
```

In [81]:
```
def change(num):
    if num == 'no':
        return 0
    else:
        return 1
```

In [82]:
```
target_num = [change(i) for i in target]
```

In [83]:
```
clf_xgb = GridSearchCV(xgb, params, n_jobs=-1).fit(feature_scaled, target_num)
```

In [84]:
```
y_pred = clf_xgb.predict(x_test_scaled)
```

In [85]:
```
y_test_num = [change(i) for i in y_test]
```

In [86]:
```
acc = accuracy_score(y_test_num, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.8726114649681529
```

In [87]:
```
pre = precision_score(y_test_num, y_pred)
print(f'The precision score is {pre}')
```

```
The precision score is 0.2818181818181818
```

In [88]:
```
rec = recall_score(y_test_num, y_pred)
print(f'The recall score is {rec}')
```

```
The recall score is 0.4626865671641791
```

In [89]:
```
cm = confusion_matrix(y_test_num, y_pred)
print(cm)
```

```
[[3027  316]
 [ 144  124]]
```

SVM

In [92]:
```
svc = SVC()
```

In [98]:
```
params = {'C': np.logspace(-3, 2, 10)}
```

In [99]:
```
clf_svm = GridSearchCV(svc, params, n_jobs=-1).fit(feature_scaled, target)
```

```
/Users/zhouzeru/opt/anaconda3/lib/python3.9/site-packages/joblib/externals/loky/process_
executor.py:702: UserWarning: A worker stopped while some jobs were given to the executo
r. This can be caused by a too short worker timeout or by a memory leak.
  warnings.warn(
```

In [100…
```
y_pred = clf_svm.predict(x_test_scaled)
```

In [101…
```
acc = accuracy_score(y_test, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.9293824425366934
```

In [102…
```
pre = precision_score(y_test, y_pred,  pos_label='no')
print(f'The precision score is {pre}')
```

```
The precision score is 0.9356659142212189
```

In [103…
```
rec = recall_score(y_test, y_pred,  pos_label='no')
print(f'The recall score is {rec}')
```

```
The recall score is 0.9919234220759796
```

In [104…
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[3316   27]
 [ 228   40]]
```

As results above, I've trialed several ML models with feature engineering & hyperparameter tuning. Logistic regression had the best performance in contrast to SVM, Bayesian model, and ensemble tree models.

According to the models above and the metrics we used, logistic regression with recursive feature elimination and support vector machine with RBF kernel had the greatest accuracy while considerable precision/recall score. When converting these ML metrics into business metrics, this filtering model that select potential clients could significantly improve our targeting efficiency. Reducing the number of targeted clients while still convincing a lot of them to subscribe, the targeting efficiency could be boosted.