03_Modelling.ipynb

September 24, 2021

1 Initialisation

[ ]:
```
# Importations
import sys
sys.path.append('..')

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold, RepeatedStratifiedKFold
from sklearn.model_selection import cross_validate
from imblearn.pipeline import Pipeline

from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from lightgbm import LGBMClassifier
from sklearn.metrics import confusion_matrix, classification_report
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import TomekLinks
from preprocessing import preprocessor as prep
from preprocessing import preprocessor_no_scaler as prep_no_scl
from styles import *
```

[ ]:
```
# Initialisation
train = pd.read_csv('../02_data/application_train.csv')
test = pd.read_csv('../02_data/application_test.csv')

id_error_msg = lambda x: '`SK_ID_CURR` is not unic for {} set!'.format(x)
assert len(train.SK_ID_CURR.unique()) == train.shape[0], id_error_msg('train')
assert len(test.SK_ID_CURR.unique()) == test.shape[0], id_error_msg('test')
train.set_index('SK_ID_CURR', inplace=True)
test.set_index('SK_ID_CURR', inplace=True)

print('Training set dimensions :', train.shape)

cls_size = train.TARGET.value_counts()
cls_freq = train.TARGET.value_counts(normalize=True)
```

```
print(pd.DataFrame({'size': cls_size,
                    'freq': cls_freq.apply(lambda x: '%.3f' % x)}))
```

```
Training set dimensions : (307511, 121)
     size   freq
0  282686  0.919
1   24825  0.081
```

```
[ ]: train_sample = train[::10]
     print('Sampled training set dimensions :', train_sample.shape)
     cls_size = train.TARGET.value_counts()
     cls_freq = train.TARGET.value_counts(normalize=True)
     print(pd.DataFrame({'size': cls_size,
                         'freq': cls_freq.apply(lambda x: '%.3f' % x)}))
```

```
Sampled training set dimensions : (30752, 121)
     size   freq
0  282686  0.919
1   24825  0.081
```

On échantillonne le dataset en prenant 10% des points de données

```
[ ]: X, y = train.iloc[:, 1:], train.iloc[:, 0]#.values.reshape(-1,1)
     Xs, ys = train_sample.iloc[:, 1:], train_sample.iloc[:, 0]#.values.reshape(-1,1)

     X_train, X_test, y_train, y_test = train_test_split(Xs, ys, test_size=.2,
                                                         random_state=0)
     print('X_train:', X_train.shape)
     print('y_train:', y_train.shape)
     print('X_test:', X_test.shape)
     print('y_test:', y_test.shape)
```

```
X_train: (24601, 120)
y_train: (24601,)
X_test: (6151, 120)
y_test: (6151,)
```

```
[ ]: print(y_train.value_counts())
```

```
0    22659
1     1942
Name: TARGET, dtype: int64
```

2 Rééquilibrage de classes - SMOTE/Tomek

Il y a ~8% de cas de défaut dans le jeu d'entraînement contre 92% de cas sans défaut. Le déséquilibre des classes pose problème dans le cadre de la prédiction de la classe minoritaire par un algorithme de ml.

Il faut rééquilibrer les classes du jeu d'entraînement avant de sélectionner le meilleur modèle de ml

[ ]:
```
resamplr = SMOTETomek(tomek=TomekLinks(sampling_strategy='majority'))
```

2.1 Impact de SMOTE Tomek sur la répartition des classes

[ ]:
```
X_train_trans = prep.fit_transform(X_train)
print(X_train_trans.shape)
print(X_train_trans)
print(y_train.shape)
print(y_train.value_counts())
```

```
(45318, 235)
0    22659
1    22659
Name: TARGET, dtype: int64
```

Rééquilibrage exécuté en 1min environ pour un jeu d'entraînement divisé par 10.

temps d'entraînement 52s pour un jeu d'entraînement divisé par 10 avec seulement les 50 premières colonnes contre 60.5s avec toutes les colonnes.

2.2 Impact de SMOTE Tomek sur l'entraînement d'un modèle

[ ]:
```
sgd = Pipeline([('p', prep), ('m', SGDClassifier())])
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
#cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=42)
scoring = ['precision_macro','recall_macro'] #, 'accuracy']
sgd_scor = cross_validate(sgd, X_train, y_train, scoring=scoring, cv=cv)
print('Model 1\n' + line_decor)
#print('accuracy scores:', sgd_scor['test_accuracy'])
print('precision scores:', sgd_scor['test_precision_macro'])
print('recall scores:', sgd_scor['test_recall_macro'])
#print('Mean Accuracy: %.4f' % np.mean(sgd_scores['test_accuracy']))
print('Mean Precision: %.4f' % np.nanmean(sgd_scor['test_precision_macro']))
print('Mean Recall: %.4f' % np.nanmean(sgd_scor['test_recall_macro']))
```

```
Model 1
--------
precision scores: [       nan 0.46056911 0.46056911 0.46056911        nan]
recall scores: [nan 0.5 0.5 0.5 nan]
Mean Precision: 0.4606
Mean Recall: 0.5000
```

Validation croisée sans SMOTE Tomek : 8.7s avec un échantillon divisé par 10

[ ]:
```
sgd_imb = Pipeline([('p', prep), ('r', resamplr), ('m', SGDClassifier())])
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
#cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=42)
```

```
scoring = ['precision_macro','recall_macro'] #, 'accuracy']
sgd_imb_scor = cross_validate(sgd_imb, X_train, y_train, scoring=scoring, cv=5)
print('Model 1 - with imbalance handling\n' + line_decor)
#print('accuracy scores:', sgd_imb_scor['test_accuracy'])
print('precision scores:', sgd_imb_scor['test_precision_macro'])
print('recall scores:', sgd_imb_scor['test_recall_macro'])
#print('Mean Accuracy: %.4f' % np.mean(sgd_imb_scores['test_accuracy']))
print('Mean Precision: %.4f' % np.nanmean(sgd_imb_scor['test_precision_macro']))
print('Mean Recall: %.4f' % np.nanmean(sgd_imb_scor['test_recall_macro']))
```

```
Model 1 - with imbalance handling
--------
precision scores: [       nan 0.55255999 0.5584412         nan 0.55571135]
recall scores: [       nan 0.66237227 0.63354292        nan 0.67955739]
Mean Precision: 0.5556
Mean Recall: 0.6585
```

Validation croisée avec SMOTE Tomek (stratégie majoritaire) : 207.6s avec un échantillon divisé par 10

[ ]:
```
smote_unsmote_ratio = 207.6 / 8.7
print('{:.2f}'.format(smote_unsmote_ratio))
```

```
23.86
```

[ ]:
```
smote_unsmote_ratio = 186.5 / 9.6
print('{:2f}'.format(smote_unsmote_ratio))
```

```
19.427083
```

Le SMOTE Tomek multiplie par un facteur 19 à 24 le temps d'exécution du modèle

Essai d'une validation croisée sans SMOTE Tomek avec tous les points du jeu d'entraînement

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2)
print('X_train:', X_train.shape)
print('y_train:', y_train.shape)
print('X_test:', X_test.shape)
print('y_test:', y_test.shape)
```

```
X_train: (246008, 120)
y_train: (246008, 1)
X_test: (61503, 120)
y_test: (61503, 1)
```

[ ]:
```
sgd = Pipeline([('p', prep), ('m', SGDClassifier())])
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
#cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=42)
scoring = ['precision_macro','recall_macro'] #, 'accuracy']
```

```
sgd_scor = cross_validate(sgd, X_train, y_train, scoring=scoring, cv=cv)
print('Model 1\n' + line_decor)
#print('accuracy scores:', sgd_scor['test_accuracy'])
print('precision scores:', sgd_scor['test_precision_macro'])
print('recall scores:', sgd_scor['test_recall_macro'])
#print('Mean Accuracy: %.4f' % np.mean(sgd_scores['test_accuracy']))
print('Mean Precision: %.4f' % np.nanmean(sgd_scor['test_precision_macro']))
print('Mean Recall: %.4f' % np.nanmean(sgd_scor['test_recall_macro']))
```

```
Model 1
--------
precision scores: [0.45967644 0.45966627 0.45966627 0.45967562 0.45967562]
recall scores: [0.5 0.5 0.5 0.5 0.5]
Mean Precision: 0.4597
Mean Recall: 0.5000
```

Validation croisée sans SMOTE Tomek exécutée en 57.9s sur tout le jeu de données

[ ]:
```
unsampled_sampled_ratio = 57.9 / 8.7
print('{:.2f}'.format(unsampled_sampled_ratio))
```

```
6.66
```

Il faut 7 fois plus de temps pour exécuter la même chose sur 10 fois plus de données (pas parfaitement linéaire donc)

[ ]:
```
print('{:.2f}'.format(207.6 * unsampled_sampled_ratio))
```

```
1381.61
```

[ ]:
```
1381 / 60
```

[ ]:
```
23.016666666666666
```

2.3 Réduction du temps de rééquilibrage en suppprimant des colonnes

[ ]:
```
X_train_resampl_cut, y_train_resampl_cut = resamplr.fit_resample(
    X_train_trans[:,:50], y_train
    )
print(X_train_resampl_cut.shape)
print(y_train_resampl_cut.value_counts())
```

```
(45313, 50)
1    22659
0    22654
Name: TARGET, dtype: int64
```

Il faudrait 23 minutes rien que pour faire du rééquilibrage avec le jeu de données actuel. Pas souhaitable.

Il faut trouver un moyen de raccourcir le temps d'exécution du rééquilibrage.

3 Modèle 1 : SGD Classifier

[ ]:
```
model1 = Pipeline([('p', prep), ('m', SGDClassifier())])
model1.fit(X_train, y_train)
y_pred = model1.predict(X_test)
conf_mat = confusion_matrix(y_test, y_pred)
print('Model 1\n' + line_decor)
print('Score: %.4f' % model1.score(X_test, y_test))
print(line_decor + '\nConfusion matrix\n' + str(conf_mat))
print(classification_report(y_test, y_pred))
```

```
Model 1
--------
Score: 0.9190
--------
Confusion matrix
[[56522     0]
 [ 4981     0]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     56522
           1       0.00      0.00      0.00      4981

    accuracy                           0.92     61503
   macro avg       0.46      0.50      0.48     61503
weighted avg       0.84      0.92      0.88     61503
```

4 Modèle 2 : Random Forest Classifier

[ ]:
```
model2 = Pipeline([('p', prep_no_scl), ('m', RandomForestClassifier())])
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scoring = ['accuracy','precision_macro','recall_macro']
scores_model2 = cross_validate(model2, X_train, y_train, scoring=scoring, cv=cv,
                               n_jobs=-1)

print('Model 2\n' + 8 * '-')
print('Mean Accuracy: %.4f' % np.mean(scores_model2['test_accuracy']))
print('Mean Precision: %.4f' % np.mean(scores_model2['test_precision_macro']))
print('Mean Recall: %.4f' % np.mean(scores_model2['test_recall_macro']))
```

[ ]:
```
model2 = Pipeline([('p', prep_no_scl), ('m', RandomForestClassifier())])
model2.fit(X_train, y_train)
y_pred = model2.predict(X_test)
```

```
conf_mat = confusion_matrix(y_test, y_pred)
print('Model 2\n' + 8 * '-')
print('Score: %.4f' % model2.score(X_test, y_test))
print(8 * '-' + '\nConfusion matrix\n' + str(conf_mat))
print(classification_report(y_test, y_pred))
```

```
Model 1
--------
Score: 0.9185
--------
Confusion matrix
[[56485     4]
 [ 5011     3]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     56489
           1       0.43      0.00      0.00      5014

    accuracy                           0.92     61503
   macro avg       0.67      0.50      0.48     61503
weighted avg       0.88      0.92      0.88     61503
```

[ ]:
```
y_pred = model2.predict(X_test)
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)
```

```
[[56512     5]
 [ 4979     7]]
```

[ ]:
```
model2.get_params()
```

5 Modèle 3 : LightGBM

[ ]:
```
model3 = Pipeline([('p', prep), ('m', LGBMClassifier())])
model3.fit(X_train, y_train)
print('Score:', model3.score(X_test, y_test))
```

```
Score: 0.9192071931450498
```

[ ]:
```
y_pred = model3.predict(X_test)
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)
```

```
[[56447    81]
 [ 4888    87]]
```

[ ]:
```python
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     56528
           1       0.52      0.02      0.03      4975

    accuracy                           0.92     61503
   macro avg       0.72      0.51      0.50     61503
weighted avg       0.89      0.92      0.88     61503
```

[ ]:
```python
# à faire

# smote tomek
# random search precision des deux classes (privilégier light_gbm)
#
# choisir optimisation recall(classe 1)
# fonction coût : manque à gagner pour chaque treshold
# treshold = + = + precision - recall
# precision élevée = on accepte tout le monde
# recall élevée = on refuse tout le monde
# regarder crer une colonne intérêts (amt credit - good price),
# optimiser mon threshold % de ça
```