Hugo Porta

Compte rendu du challenge de SD 210:

Première étape le preprocessing:

LabelEncoder:

Tout d'abord, comme il était conseillé dans l'intitutilé du challenge, j'ai voulu effectuer le préprocessing des données catégorielles. Mon but était de les trasformer en données binaires et de remplacer les dataframes catégoriels en dataframe contenant des vecteur binaires.

Une fois les données récupérer j'ai en premier labélisé la variable cible via un labelencoder car la première variable est granted.

```
df.VARIABLE_CIBLE = df.VARIABLE_CIBLE == 'GRANTED'
encbis = LabelEncoder()
df.VARIABLE_CIBLE = encbis.fit_transform(df.VARIABLE_CIBLE)
```

Puis j'ai rentré dans une liste name l'ensemble des noms de colonnes de données catégorielles et effectué le labelencoder suivant, où je concaténais le test et le train puis sur chaque colonne j'appliquais le labelencoder. Je ne sais pas pourquoi je l'avais implémenté sous forme de boucle car ensuite je l'ai utilisé directement dans un autre preprocessing.

```
df2 = df
df2_test = df_test
frames = [df2, df2_test]
result = pd.concat(frames)
for i in name:
    enc = LabelEncoder()
    enc.fit(result[i])
    df2[i] = enc.transform(df2[i])
    df2_test[i] =enc.transform(df2_test[i])
```

Imputer:

Il fallait maintenant gérer les valeur NaN, pour cela j'ai effectué un imputer qui sur les données numériques remplaçait par la valeur moyenne du dataframe considéré et sur les données anciennement catégorielles par la valeur la plus fréquente.
Dan ce cas j'avais effectué un traitement séparé sur le test et le train car contrairement au labelencoder ou au onehotencoder, ce n'est pas nécessaire. Encore une fois j'effectué une boucle sur les données, je pense que cette erreur est surtout du au fait que j'ai commencé trés tôt le challenge et que l'on a gagné en capacité et compréhension durant cette période car on va voir par la suite que je n'ai pas du tout effectué comme ça le second préprocesing que j'ai utilisé.

```
df3 = df2
df3_test = df2_test
k=0
for i in name:
    a = pd.DataFrame(df3[i])
    b = pd.DataFrame(df3_test[i])
    imputer = Imputer(strategy = "most_frequent")
    df3[i]= imputer.fit_transform(a)
    df3_test[i]= imputer.fit_transform(b)
iterator3 = df3.iteritems()
for j in iterator3:
    if j[0]!= "VARIABLE_CIBLE":
```

```
    c = pd.DataFrame(df3[j[0]])
    d = pd.DataFrame(df3_test[j[0]])
    imputer = Imputer()
    df3[j[0]]= imputer.fit_transform(c)
    df3_test[j[0]]= imputer.fit_transform(d)
```

OneHotEncoder:

Il fallait maintenant transformé les données que l'on avait labélisées en vecteur binaire pour éviter toute notion d'ordre entre les données qui n'ont pas lieu d'être. Les résultats sont en sorti des sparse matrix que l'on ne peut mettre directement dans les dataframes. De nouveau on constate la nécessité de concaténer pour avoir un OneHotEncoder qui englobe l'ensemble des données. En effet si on effectuait deux OneHotEncoder indépendants on aurait deux transformations indépendantes et donc qui ne correspondent pas.

```
df4 = df3
df4_test =df3_test
frames = [df4, df4_test]
resultdf = pd.concat(frames)
result = []
resultTest = []
for i in name:
   ohe = OneHotEncoder()
   a = pd.DataFrame(df4[i])
   b = pd.DataFrame(df4_test[i])
   c = pd.DataFrame(resultdf[i])
   ohe.fit(c)
   resultTest.append(ohe.transform(b))
   result.append(ohe.transform(a))
```

De ma liste de sparse matrix je devais encore récupérer un array de vecteur pour l'insérer dans les dataframes pour cela j'ai effectuer le code suivant avec la méthode toarray() des sparse matrix. Je ne faisait pas ça pour le 6éme élèment car c'est FIRST_CLASS qui a beaucoup de valeurs distinctes et donc des array de trop grande taille, ce qui entraine une erreur mémoire sur ce code.L'autre technique que j'aurais pu effectuer, c'est d'enregistrer dans un dataframe chaque composante des vecteurs binaires que j'obtenais ainsi par exemple pour Country si on avait 20 valeurs différentes j'aurais obtenu 20 dataframes différents avec dedans des variables binaires.

```
for i in range(0,len(result)):
   if i!=6:
      C = result[i].toarray()
      df4[name[i]] = C

for i in range(0,len(resultTest)):
   if i!=6:
      C = resultTest[i].toarray()
      df4_test[name[i]] = C
```

J'ai ensuite testé ce préprocessing via les randomforest en effectuant au préalable une sélection de features par selectkbest. Cependant du fait de la taille des vecteurs obtenus le calcul des hyperparamètres via gridsearchCV ou par des boucles de cross validation était trés long et pas trés fructueux en terme de score. En effet j'était limité à O.63 .
J'ai donc décider de partir sur un préprocessing plus simple que l'on peut voir par la suite où je labelencode seulement pour améliorer ma vitesse de calcul et ainsi favoriser le choix des hyperparamètres bien que l'on introduit une relation d'ordre. De plus j'ai aussi effectué le préprocessing directement et plus à travers des boucles. Enfin je traite les dates autrement que par un labelencoder en récupérant l'année

```
train_fname = 'train.csv'
```

```
test_fname = 'test.csv'
df = pd.read_csv(train_fname, sep=';')
df_test = pd.read_csv(test_fname, sep=';')
train_length = df.shape[0]
y = df.VARIABLE_CIBLE == 'GRANTED'
encbis = LabelEncoder()
y = encbis.fit_transform(y)
df.drop('VARIABLE_CIBLE',axis=1,inplace=True)
name =
["VOIE_DEPOT","COUNTRY","SOURCE_BEGIN_MONTH","FISRT_APP_COUNTRY","FISRT_APP_
TYPE","LANGUAGE_OF_FILLING","FIRST_CLASSE","TECHNOLOGIE_SECTOR",

"TECHNOLOGIE_FIELD","MAIN_IPC","FISRT_INV_COUNTRY","FISRT_INV_TYPE","SOURCE_ID
X_ORI","SOURCE_CITED_AGE","SOURCE_IDX_RAD"]
date = ["PRIORITY_MONTH","FILING_MONTH","PUBLICATION_MONTH",
      "BEGIN_MONTH"]
donnee = pd.concat((df,df_test))
enc = LabelEncoder()
donnee[name] = donnee[name].apply(enc.fit_transform)
for i in date:
   donnee[i] = donnee[i].str.extract(r'/([\d]{4})')
imp = Imputer()
donnee  = imp.fit_transform(donee)
X_train = donnee[:train_length]
X_test = donnee[train_length:]
```

Deuxième étape le processing:

RandomForest:

Le premier processing que j'ai effectué était via les randomforest. Avant de les runner j'effectuais un choix des features car du fait du premier processing le traitement était plutôt long. J'ai essayé deux types de sélection de features, tout d'abord via la PCA où je traçais le graphe de la variance cumulée en fonction du nombre de features puis via ce graphe je pouvais choisir le nombre de features en foncton du pourcentage souhaité.

```
X_stand = preprocessing.scale(X + 0.0)
pca3 = PCA()
pca3.fit(X_stand)
X_PCA = pca3.transform(X)
X_PCA_test = pca3.transform(X_test)

Var = (pca3.explained_variance_ratio_ * 100)
lim = Var.size
Result_PCA = [None] * (lim - 1)
Result_PCA[0] = Var[0]
for i in range(1,lim-1):
   Result_PCA[i] = Result_PCA[i-1] + Var [i]
plt.plot(Result_PCA)
print(X_PCA[:,:25].shape)
```

La seconde méthode que j'ai utilisé et qui a donné de meilleurs résultats par la suite est selectkbest.Cette méthode choisit les n features selon un type de score qui est imposé, en effet on ne pouvait pas utiliser un autre type de scoring que chi2.

```
X = df5.values
X_test = df5_test.values
```

```
y = df3["VARIABLE_CIBLE"].values
SKB = SelectKBest(chi2,k=30)
SKB.fit(X_skb,y)
X_best = SKB.transform(X)
X_best_test = SKB.transform(X_test)
```

Ensuite j'ai essayé d'optimiser les paramètres suivants: nombre d'estimateurs et profondeur maximale des arbres via des gridsearchcv ou des doubles boucles. J'ai privilégié les doubles boucles car comme le traitement était long, je pouvais surveiller l'évoulution en printant les scores à chaque étape.Néanmoins je restais bloqué à 0.63 en score. J'ai donc décider de changer mon preprocessing et de prendre un nouveau classifier.

GradientBoostingClassifier:

J'ai donc décider de choisir ce classifieur de la bibliothèque sklearn car il est réputé pour être le plus efficace. Néanmoins même en utlisant le second préprocessing qui offre un jeu de données beaucoup plus simple à traiter l'éxécution de ce classifieur est extrêment lente (environ une demi heure pour un fit et predict sur mon ordinateur muni d'un processeur intel i5). J'ai néanmoins via des doubles boucles et et cross_val_score réussi à obtenir un nombre d'estimateur et une profondeur optimals: 150 et 10. J'obtenais ainsi comme score 0.712 environ.

```
e_range = [125,150,175]
depth_range = [5,10,15]
for i in e_range:
    for j in depth_range:
        gbc = GradientBoostingClassifier(max_depth = j,n_estimators=i)
        Result = cross_val_score(gbc,X_train,y,scoring='roc_auc',cv=3).mean()
        print(i)
        print(j)
        print(Result)
```

J'aurais pu me limiter à ceci me je trouvais la méthode de processing pas assez rigoureuse car au final je n'optimisais que deux paramètres. J'ai donc décider d'utiliser xgboost qui est une bibliothèque implémentant elle aussile gradient boosting classifier mais beaucoup plus rapide dans l'éxécution.

Xgboost:

Tout d'abord comme je travail sur une machine windows et que je n'ai pas réussi à l'installer sur ce système d'exploitation, j'ai utilisé une virtual box et un système d'exploitation Linux, auquel j'ai alloué 4 GB de rame. Ma méthode sur xgboost pour optimiser les paramètres était la suivantes:

Tout d'abord pour des paramètres pris au hasard j'ai effectué le code suivant, qui permet de connaitre le nombre d'estimateurs optimal pour les paramètres pris, en effet la cross validation sur xgboost se fait pour chaque itération ainsi avec le paramètre early_stopping_round on peut s'arrêter lorsque le score commence à diminuer. On prend un num_rounds grand pour atteindre cette optimum.

```
Matrix_train  = xgb.DMatrix(X_train, label=y)
Matrix_test   = xgb.DMatrix(X_test)
param_xgb   = {'max_depth': 9, 'eta': 0.01,'objective': 'binary:logistic'
, 'subsample': 0.9, 'eval_metric': 'auc','colsample_bytree' :0.7,'seed':27,'min_child_weight' : 5}
num_rounds = 5000

CVresult =
```

```
xgb.cv(param_xgb,Matrix_train,nfold=5,num_boost_round=num_rounds,early_stopping_rounds=50)
print(CVresult)
```

Puis on va optimiser les paramètres via des doubles boucles de cross validation car je n'ai jamais réussi à faire fonctionner le XGBClassifier pour utiliser GridSearchCV de sklearn. En effet il me renvoyer toujour une nonetype error. Les couples de paramètres que l'on a optmisé un à un sont : (max_depth,min_child_weight) ; (colsample_bytree,subsample) et enfin alpha et gamma mais la valeur obtenue était celle par défaut 0.

```
Matrix_train  = xgb.DMatrix(X_train, label=y)
Matrix_test   = xgb.DMatrix(X_test)

num_rounds = 240
alpha_range = [0.002,0.003,0.004]
for j in alpha_range:
	param_xgb   = {'max_depth': 9, 'eta': 0.1, 'objective': 'binary:logistic'
, 'subsample': 0.9, 'eval_metric': 'auc','colsample_bytree' :0.7,'seed':27,'min_child_weight' :
5,'gamma':0,'alpha':j}
	CVresult = xgb.cv(param_xgb,Matrix_train,nfold=3,num_boost_round=num_rounds)
	print(j)
	print(CVresult[239:])
```

Aprés chaque optimisation de paramètres il faut de nouveau calculer le nombre d'estimateur optimal c'est pour ça que num_rounds varie. Enfin pour finir j'ai diminuer learning rate eta puis recalculer le nombre de classifier optimal j'ai ainsi obtenu 0.72 avec le script en attaché.