Concours Crésus - Prédiction du meilleur accompagnement pour personnes endettées

A - Nettoyage

---

1 - Traitement NA

Les fonctions sont disponibles en annexe A1.

---

2 - Traitement Types

Les fonctions sont disponibles en annexe A2.

---

3 - Mega traitement

Les fonctions sont disponibles en annexe A3.

Le raisonnement est développé en commentaires dans les fonctions. Egalement, les différents essais sont disponibles dans le notebook exporté ‘nettoyage_cresus.pdf’ .

B - Entrainement

1 - ACP + KMeans

Les fonctions sont disponibles en annexe B1.

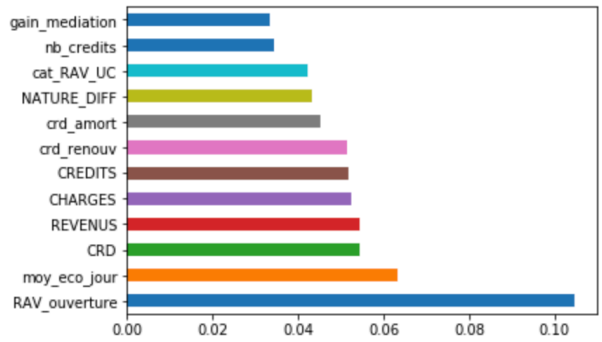
Pour explorer premièrement les données, j'ai effectué une PCA sur les données. J'ai ainsi pu les observer rapidement sur les axes les plus importants.
J'ai également essayé un algorithme de clustering (Kmeans) en essayant plusieurs nombre de groupes. Cependant, cela n'a donnée aucun résultat cohérent.

De plus, les données seraient peu interprétables. En effet, les données étant projetées sur de nouveaux axes, il faudrait explorer les axes importants pour repérer les features importantes sur lesquelles ils sont basés.

2 - Random Forest

Afin de déterminer quelles valeurs attribuer aux paramètres, j'ai utilisé des GridSearch en augmentant au fur et mesure les intervalles de valeurs (par souci de performance).

Variables	Grid Search 1	Grid Search 2	Grid Search 3
max_depth	90	90	75
max_features	5	5	8
min_samples_leaf	4	3	2
min_samples_split	12	8	12
n_estimators	100	200	300
Score Local	0.58	0.58	0.59
Score Kaggle	0.59235	0.59097	0.60156



Sur les les trois modèles RandomForest générés, les features les plus importantes restent CREDITS, CHARGES, REVENUS, CRD avec le reste à vivre (RAV) toujours en première place.

3 - Logistic regression

Une autre solution pour la classification est la régression logistique. Ayant 6 classes, nous devons utiliser la régression logistique multinomiale.

Cette méthode semblent moins performantes que les arbres de décision utilisés précédemment. Cependant, je n'ai utilisé que quelques ensembles de features.C - Conclusion

C - Interprétation

1 - Interprétation

Parmi les différents algorithmes utilisés, on retrouve souvent les mêmes composantes importantes: celles qui sont liées aux crédits et aux charges. En revanche, les données personnelles semblent beaucoup moins importantes

2 - Pistes d'amélioration

Il faudrait approfondir l'analyse des features et/ou adapter le nettoyage pour la régression logistique.

Il peut être également interessant d'essayer de prédire moins de groupes, ou due regrouper certains accompagnement.

Enfin, un dernier algorithme qui pourrait donner de bons résultat est la SVM.

Annexe A1:

```
def traitement_na(df):
    df_ = df.copy()

    df_ = df_.replace('Non Renseigne', np.NAN)
    for CRD in df_.columns.tolist():
        if 'crd_' in CRD:
            df_[CRD] = df_[CRD].fillna(0)


    # Catégoriser la PROFESSION selon la proximité des revenus
    revenus_prof = {}
    for p in df_.PROF.unique():
        REV = df_.loc[df_.PROF == p, 'REVENUS'].mean()
        try:
            revenus_prof[p] = round(REV)
        except:
            pass

    correct_PROF = df_.REVENUS.apply(lambda x: min(revenus_prof.items(), key=lambda v: abs(v[1] - int(x)))[1])
    df_['PROF'] = df_['PROF'].fillna(correct_PROF)



    # Catégoriser la cat_RAV_UC selon la proximité des Reste A Vivre Ouverture
    cat_RAV_ = {}
    for r in df_.cat_RAV_UC.unique():
        REV = df_.loc[df_.cat_RAV_UC == r, 'RAV_ouverture'].mean()
        try:
            cat_RAV_[r] = round(REV)
        except:
            pass

    correct_RAV = df_.RAV_ouverture.apply(lambda x: min(cat_RAV_.items(), key=lambda v: abs(v[1] - int(x)))[1])
    # NE MARCHE PAS ?
    #df_['cat_RAV_UC'] = df_['cat_RAV_UC'].fillna(cat_RAV_)
    df_['cat_RAV_UC'] = df_['cat_RAV_UC'].fillna('C')
    df_['RAV_UC'] = round(df_[df_.cat_RAV_UC == 'C'].RAV_UC.mean())




    for p in df_.PROF.unique():
        med = df_.loc[df_.PROF == p].age.median()
        df_.loc[df_.PROF == p, 'age'] = df_.loc[df_.PROF == p, 'age'].fillna(med)

    # Tranche d'age en fonction de l'age précedement calculé
    df_.tranche_age = df_.apply(lambda x: trancheAge(x), axis=1)


   for p in df_.PROF.unique():
       med = df_.loc[df_.PROF == p].age.median()
       df_.loc[df_.PROF == p, 'age'] = df_.loc[df_.PROF == p, 'age'].fillna(med)

   # Tranche d'age en fonction de l'age précedement calculé
   df_.tranche_age = df_.apply(lambda x: trancheAge(x), axis=1)


   # Il y a 60% de '2' dans adulte foyer
   df_.adulte_foyer = df_.adulte_foyer.fillna(2)
   # Peu important
   df_.situation = df_.situation.fillna('autre')
   # Trop de cas pssibles, malgrès une majorité (50%) d' Endettement
   df_.NATURE_DIFF = df_.NATURE_DIFF.fillna('autre')
   # Peu important
   df_.region = df_.region.fillna('inconnu')
   # Peu important
   df_.cat_moy_eco_jour = df_.cat_moy_eco_jour.fillna('inconnu')
   # Majorité
   df_.LOGEMENT = df_.LOGEMENT.fillna('locataire')
   # NA environ 0
   df_.moy_eco_jour = df_.moy_eco_jour.fillna(0)
   # Valeur majoritaire
   df_.cat_credit = df_.cat_credit.fillna('1€-499€')
   # Valeur medianne
   df_.CRD = df_.CRD.fillna(df_.CRD.median())


   # Colonnes peu exploitables
   df_ = df_.drop(columns=['IMPAYES_DEBUT','STRUCTURE PRESCRIPTRICE'])

   df_.gain_mediation = df_.gain_mediation.fillna(0).astype(int)

   df_.cat_impayes = df_.cat_impayes.fillna('Inconnu')

   #df_ = df_.dropna()

   return df_
```

```python
def trancheAge(x):
    age = round(x.age)
    if age < 25:
        return '<25ans'

    elif age >= 25 and age <= 34:
        return  '25-34ans'

    elif age >= 35 and age <= 44:
        return  '35-44ans'

    elif age >= 45 and age <= 54:
        return  '45-54ans'

    elif age >= 55 and age <= 64:
        return  '55-64ans'

    elif age >= 65 and age <= 74:
        return  '65-74ans'

    elif age >= 75:
        return '>75ans'
```

Annexe A2:

```python
def traitement_types(df):
    df_ = df.copy().replace('\\N',np.NAN)

    for c in df_.columns:
        if 'crd_' in c or c in ['CRD','IMPAYES_DEBUT','age','adulte_foyer']:
            df_[c] = pd.to_numeric(df_[c], errors='coerce')

    df_.moy_eco_jour = df_.moy_eco_jour.str.replace(',', '.', regex=False).astype(float)
    df_.RAV_UC = df_.RAV_UC.str.replace(',', '.', regex=False).astype(float)

    df_['Date'] = pd.to_datetime(df_.year.astype(str) + '-' + df_.month.astype(str), format='%Y-%m')
    df_ = df_.drop(columns=['year','month'])

    return df_
```

Annexe A3:

```python
def mega_traitement(df,dummies=False):
    df_ = df.copy()
    df_ = traitement_types(df_)
    df_ = traitement_na(df_)

    ## ?? A RETIRER ?? ##
    for cc in list(df_.dtypes[df_.dtypes == 'object'].to_dict().keys()):
        #print(cc)
        cat_dtype = pd.api.types.CategoricalDtype(categories=df_[cc].unique().tolist(), ordered=False)
        df_[cc] = df_[cc].astype(cat_dtype)
    # # # # # # # # # #

    if dummies:
        or_dummies = pd.get_dummies(df_.ORIENTATION)
        df_[df_.ORIENTATION.unique().tolist()]=or_dummies

    for cat in list(df_.dtypes[df_.dtypes == 'category'].to_dict().keys()):
        gle = LabelEncoder()
        #print(cat)
        df_[cat] = gle.fit_transform(df_[cat].astype(str))

    return df_
```

Annexe B1:

```
def myAcp(df):

    sc = StandardScaler()
    Z = sc.fit_transform(features)
    acp = PCA(svd_solver='full')
    coord = acp.fit_transform(Z)

    return pd.DataFrame(coord)
```