EMSBD 6 - Apprentissage supervisé

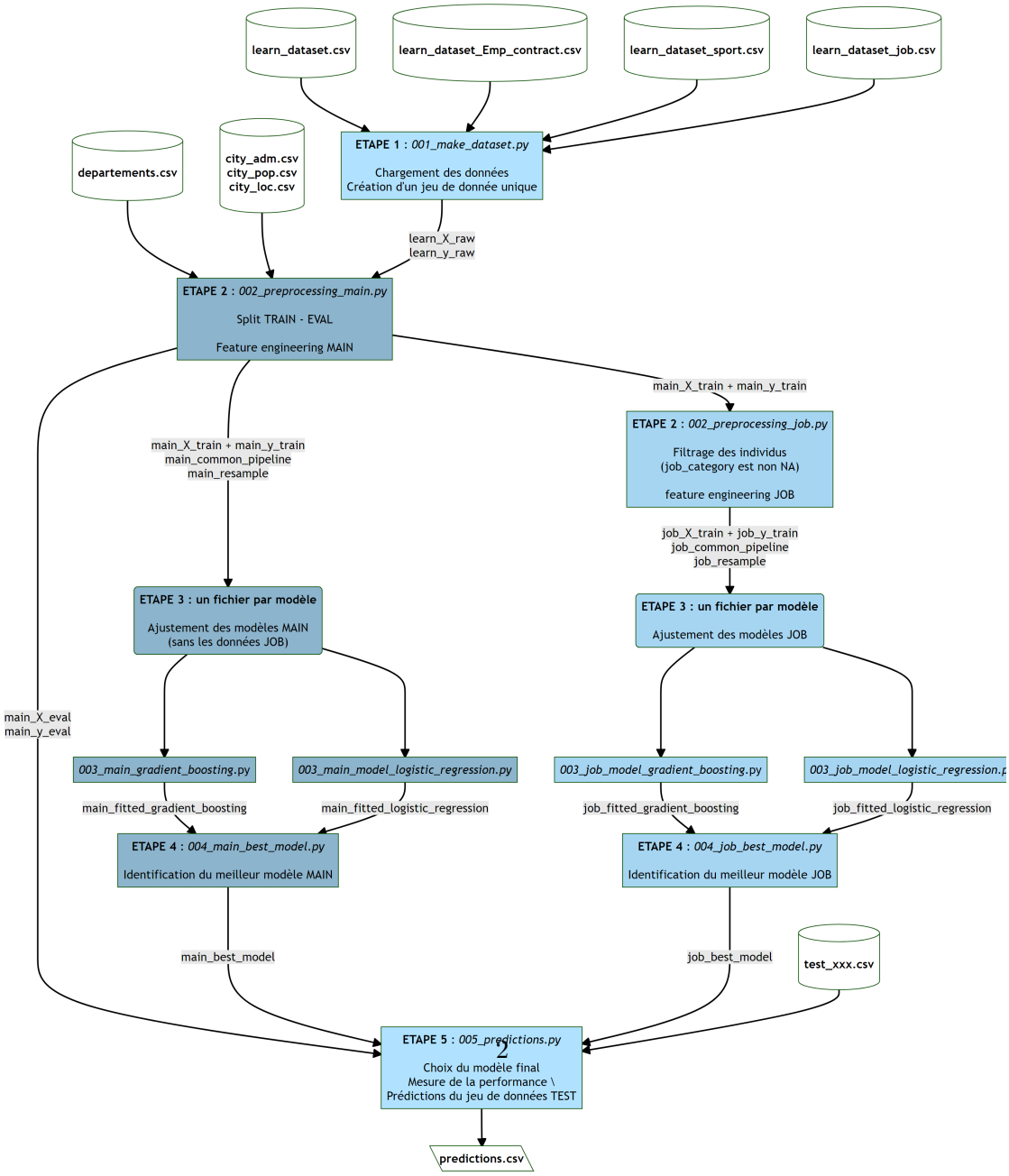
projet 3

Bruno KUBECZKA

15 avril 2024

Le **projet 3** est un projet de **classification binaire** où il s'agit de prédire pour chaque individu une **catégorie B ou G**.

Sommaire


learn_dataset.csv
learn_dataset_Emp_contract.csv
learn_dataset_sport.csv
learn_dataset_job.csv
departements.csv
city_adm.csv
city_pop.csv
city_loc.csv
ETAPE 1 : 001_make_dataset.py
Chargement des données
Création d'un jeu de donnée unique
learn_X_raw
learn_y_raw
ETAPE 2 : 002_preprocessing_main.py
Split TRAIN - EVAL
Feature engineering MAIN
main_X_train + main_y_train
ETAPE 2 : 002_preprocessing_job.py
Filtrage des individus
(job_category est non NA)
feature engineering JOB
main_X_train + main_y_train
main_common_pipeline
main_resample
job_X_train + job_y_train
job_common_pipeline
job_resample
ETAPE 3 : un fichier par modèle
Ajustement des modèles MAIN
(sans les données JOB)
ETAPE 3 : un fichier par modèle
Ajustement des modèles JOB
main_X_eval
main_y_eval
003_main_gradient_boosting.py
003_main_model_logistic_regression.py
003_job_model_gradient_boosting.py
003_job_model_logistic_regression.p
main_fitted_gradient_boosting
main_fitted_logistic_regression
job_fitted_gradient_boosting
job_fitted_logistic_regression
ETAPE 4 : 004_main_best_model.py
Identification du meilleur modèle MAIN
ETAPE 4 : 004_job_best_model.py
Identification du meilleur modèle JOB
main_best_model
job_best_model
test_xxx.csv
ETAPE 5 : 005_predictions.py
Choix du modèle final
Mesure de la performance \
Prédictions du jeu de données TEST
predictions.csv

1 Projet

1.1 Démarche

Construction du jeu de données d'apprentissage

Les fichiers **learn_dataset.csv**, **learn_dataset_Emp_contract.csv**, **learn_dataset_sport.csv**, **learn_dataset_job.csv** sont unifiés dans un **dataframe unique** par une jointure sur l'identifiant de l'individu `Id` (cf.*fichier 001_make_dataset.py*)

Le jeu de données unifié est scindé en 2 jeux de données selon un tirage aléatoire effectué en respect de la distribution des cibles y (cf. fichier *002_preprocessing_main.py*).

On en retire :

- un jeu de données **X et y TRAIN** : 80% des données du jeu de données unifié, tirées aléatoirement, vont être utilisées pour l'apprentissage des modèles
- un jeu de données **X et y EVAL** : les 20% restants sont conservés pour mesurer la performance du modèle final (cf. *fichier 005_predictions.py*)

On note que le jeu de données *learn_job* est un sous-ensemble du dataset principal : les variables associées présentent un grand nombre de valeurs manquantes qu'il peut être compliqué d'imputer.

Plutôt que d'imputer ces données manquantes issues de *learn_job*, on prend le parti de travailler le jeu de données sous **2 formes** :

- un **jeu de données MAIN** : ensemble des individus, sans les colonnes amenées par le jeu de données Job (cf. fichier *001_preprocessing_main.py*)
- un **jeu de données JOB**, composé uniquement des individus présents dans le jeu de données Job, en conservant l'ensemble des autres colonnes (cf. fichier *002_preprocessing_job.py*)

Feature Engineering

Pour chacun des jeux de données MAIN et JOB, un pipeline de traitement des variables est créé (cf. fichiers *002_preprocessing_main.py* et *002_preprocessing_job.py*) avec pour vocation

- de traiter les colonnes une par une par des `ColumnTransformer`
  - créant de nouvelles variables (feature engineering) (cf. Section 1.2 - Feature Engineering)
  - imputant les valeurs manquantes (cf. Section 1.2 - Feature Engineering)
  - supprimant la variable initiale si besoin

- de supprimer les colonnes non pertinentes
  - la colonne Id
  - toutes les colonnes issues de Job dans le cas du traitement de MAIN

Apprentissage

L'apprentissage, que ce soit sur le jeu de données MAIN (privées des colonnes JOB) ou sur le jeu de données JOB (uniquement les individus disposant d'une entrée job_category) respecte les mêmes règles

- 2 modèles sont utilisés : une **régression logistique** et un **Gradient Boosting**
- Chaque entraînement de modèle est agencé de la façon suivante:
  - Construction d'un **pipeline** par la concaténation
    - d'une section *feature_engineering*, copie du pipeline de construction des features (cf. fichiers *002_preprocessing_main.py* et *002_preprocessing_job.py)*
    - d'une section *preprocessing* de traitement des features numériques et catégorielles spécifique au modèle considéré (cf.précisions ci-dessous)
    - d'une section *estimator*, instance du modèle considéré (GradientBoostingClassifier ou LogisticRegression)
  - **Entraînement du pipeline** effectué par un `GridSearchCV` :
    - selon une **plage d'hyper-paramètres** adaptée au modèle
    - stratégie de **rééchantillonnage en 5 blocs**
    - critère de scoring **accuracy** (`scoring="accuracy"`)
    - ajustement final automatique du modèle avec les hyper-paramètres qui ont donné le meilleur score (`refit=True`)

Le **modèle de Gradient Boosting** complète le pipeline de la façon suivante

- toutes les **variables numériques** sont **maintenues en l'état** (`passthrough`)
- toutes les **variables catégorielles** sont **maintenues et transformées par un** `OneHotEncoder`, sans suppression de modalités (`drop=None`, le modèle n'étant pas sujet à des problèmes de colinéarité), en ignorant les modalités inconnues (`handle_unknown='ignore'`, toutes les colonnes de modalités passeront à 0)

Le **modèle de régression logistique** complète le pipeline de la façon suivante

- Les colonnes entraînant des colinéarités sont supprimées (cf. Section 1.2 - Feature Engineering)

- toutes les **variables numériques** sont **normalisées** (`StandardScaler`)
- toutes les **variables catégorielles** sont **maintenues et transformées par un** `OneHotEncoder`, en supprimant la 1ère modalité (`drop="first",` le modèle étant sujet à des problèmes de colinéarité) et en ignorant les modalités inconnues (`handle_unknown='ignore'`, toutes les colonnes de modalités passeront à 0)

Choix des meilleurs modèles MAIN et JOB

Une fois les 4 modèles entraînés, les **meilleurs modèles JOB et MAIN** sont identifiés en se basant sur le **score moyen obtenu après validation croisée en 5 blocs** (fichiers *004_main_best_model.py* et *004_job_best_model.py*).

Choix du modèle final et performance sur jeu de données d'évaluation

cf .fichier *005_predictions.py*

En fonction des scores des meilleurs modèles MAIN et JOB, on choisit le meilleur modèle final (modèle MAIN seul ou combinaison MAIN+JOB) (cf. Section 4.1 - Choix du modèle final).

C'est ce modèle final dont la performance est mesurée sur le jeu de données d'évaluation.

Prédiction du jeu de données de test

Les données *test_xxx.csv* sont prédites par le modèle final identifié (cf .fichier *005_predictions.py*)

1.2 Feature engineering

Le feature engineering intégré dans les pipelines des modèles est implémenté dans les fichiers *toolbox_feature_engineering.py* et *toolbox_feature_engineering_job.py*.

On y retrouve les mécanismes d'enrichissement et d'imputation des valeurs manquantes sous forme de `ColumnTransformer`.

Dans les grandes lignes, les principes directeurs ont été

- `insee_code` a été exploitée pour introduire des **informations géographiques** telles que la taille, le nombre d'habitants, le département et la région de la commune de l'individu, ainsi que la distance de la commune à la "grande ville" la plus proche (capitale, préfecture, sous-préfecture selon les coordonnées de la commune)
- Les variables catégorielles composées de données hiérarchiques (`ACTIVITY_TYPE`, `HOUSEHOLD`, `OCCUPATION_42`) ont été supprimées au profit de plusieurs features représentant chaque niveau hiérarchique (par ex. `ACTIVITY_TYPE` devient `ACTIVITY_L1_` et `ACTIVITY_L2_`) :
  - Les arbres de décision des modèles Gradient Boosting ont ainsi à disposition toutes les granularités possibles pour optimiser leur choix.

  - Pour les modèles de régression logistique, seul le niveau hiérarchique le plus détaillé est maintenu par le pipeline (on évite ainsi les colinéarités)

- Les variables numériques sont maintenues en l'état et adaptées en fonction du modèle (notamment normalisées dans les modèles de régression logistique)
- `Club` introduisant un grand nombre de données manquantes a été exploitée pour créer une variable booléenne `SPORTIF_` sans données manquantes
- Une fois les jeux de données MAIN et JOB traités, les données manquantes se résument à quelques unités. Une imputation simple a été adoptée :
  - par la catégorie majoritaire pour les variables catégorielles
  - par la valeur médiane pour les variables numériques

  La performance du modèle final étant satisfaisante, il n'a pas été nécessaire de travailler des imputations plus complexes (basées par ex. sur des corrélations entre covariables)

1.3 Prévention des fuites de données

Pour **prévenir les fuites de données** pendant la phase d'apprentissage MAIN et JOB, on a pris plusieurs mesures :

- Classiquement, le jeu de données d'apprentissage unifié est scindé en un **jeu de données TRAIN** dédié à l'apprentissage des modèles, et un **jeu de données EVAL** dédié à la mesure de la performance du modèle final.
- Les modèles JOB et MAIN ont été entraînés par le **même jeu de données TRAIN** parce qu'évalués par le même jeu de données EVAL lors de la mesure de performance du modèle final.
- Tous les traitements des features sont **inclus dans les pipelines** afin qu'ils soient réappliqués à chaque itération de la validation croisée, sur les sous-ensembles d'apprentissage uniquement.

2 Apprentissage des modèles MAIN

2.1 Gradient boosting sur jeu de données MAIN

Après apprentissage par GridSearchCV avec une méthode de rééchantillonnage à 5 blocs (cf. fichier *003_main_model_gradient_boosting.py*)

Grille d'hyper-paramètres

```
{'estimator__n_estimators': range(100, 500, 100),
 'estimator__learning_rate': [0.2, 0.3, 0.4, 0.5],
 'estimator__max_depth': [2, 3, 4]}
```

Meilleurs hyper-paramètres

```
{'estimator__learning_rate': 0.3,
 'estimator__max_depth': 4,
 'estimator__n_estimators': 400}
```

Qualité de la généralisation

clés	valeurs
split0_test_score	0.846231
split1_test_score	0.853857
split2_test_score	0.860483
split3_test_score	0.851731
split4_test_score	0.849337
mean_test_score	0.852328
std_test_score	0.00480229

Le modèle propose un **score moyen de 85.23% de bonnes prédictions** sur les jeux de données de validation

Les **scores par bloc** sont **homogènes** et présentent un **faible écart-type** : le modèle présente une **bonne aptitude à la généralisation**.

2.2 Régression logistique sur jeu de données MAIN

Après apprentissage par GridSearchCV avec une méthode de rééchantillonnage à 5 blocs (cf. fichier *003_main_model_logistic_regression.py*)

Grille d'hyper-paramètres

```
{'estimator__penalty': ['l2'],
 'estimator__solver': ['sag', 'lbfgs', 'newton-cholesky'],
 'estimator__C': [0.01, 0.1, 1, 10.0, 100.0, 1000.0, 10000.0, 100000.0]}
```

Meilleurs hyper-paramètres

```
{'estimator__C': 100.0, 'estimator__penalty': 'l2', 'estimator__solver': 'sag'}
```

Qualité de la généralisation

clés	valeurs
split0_test_score	0.830604
split1_test_score	0.832104
split2_test_score	0.83898
split3_test_score	0.831104
split4_test_score	0.829832
mean_test_score	0.832525
std_test_score	0.00331068

Le modèle propose un **score moyen de 83.25% de bonnes prédictions** sur les jeux de données de validation

Les **scores par bloc** sont **homogènes** et présentent un **faible écart-type** : le modèle présente une **bonne aptitude à la généralisation**.

2.3 Choix du meilleur modèle MAIN

modèles	scores moyens
Régression Logistique	0.832525
Gradient Boosting	0.852328

Le meilleur modèle sur le jeu de données MAIN (sans données Job) est **Gradient Boosting** avec un **score moyen de 85.23%** de bonnes prédictions.

3 Apprentissage des modèles JOB

3.1 Gradient boosting sur jeu de données JOB

Après apprentissage par GridSearchCV avec une méthode de rééchantillonnage à 5 blocs (cf. fichier *003_job_model_gradient_boosting.py*)

Grille d'hyper-paramètres

```
{'estimator__n_estimators': range(100, 500, 100),
 'estimator__learning_rate': [0.2, 0.3, 0.4, 0.5],
 'estimator__max_depth': [2, 3, 4]}
```

Meilleurs hyper-paramètres

```
{'estimator__learning_rate': 0.5,
 'estimator__max_depth': 3,
 'estimator__n_estimators': 400}
```

Qualité de la généralisation

clés	valeurs
split0_test_score	0.872928
split1_test_score	0.871628
split2_test_score	0.877478
split3_test_score	0.880403
split4_test_score	0.878739
mean_test_score	0.876235
std_test_score	0.00338653

Le modèle propose un **score moyen de 87.62% de bonnes prédictions** sur les jeux de données de validation.

Les **scores par bloc** sont **homogènes** et présentent un **faible écart-type** : le modèle présente une **bonne aptitude à la généralisation**.

3.2 Régression logistique sur jeu de données JOB

Après apprentissage par GridSearchCV avec une méthode de rééchantillonnage à 5 blocs (cf. fichier *003_job_model_logistic_regression.py*)

Grille d'hyper-paramètres

```
{'estimator__penalty': ['l2'],
 'estimator__solver': ['sag', 'lbfgs', 'newton-cholesky'],
 'estimator__C': [0.01, 0.1, 1, 10.0, 100.0, 1000.0, 10000.0, 100000.0]}
```

Meilleurs hyper-paramètres

```
{'estimator__C': 1, 'estimator__penalty': 'l2', 'estimator__solver': 'sag'}
```

Qualité de la généralisation

clés	valeurs
split0_test_score	0.869028
split1_test_score	0.853429
split2_test_score	0.864478
split3_test_score	0.862853
split4_test_score	0.849155
mean_test_score	0.859789
std_test_score	0.00735237

Le modèle propose un **score moyen de 85.98% de bonnes prédictions** sur les jeux de données de validation.

Les **scores par bloc** sont **homogènes** et présentent un **faible écart-type** : le modèle présente une **bonne aptitude à la généralisation**.

3.3 Choix du meilleur modèle JOB

modèles	scores moyens
Régression Logistique	0.859789
Gradient Boosting	0.876235

Le meilleur modèle sur le jeu de données JOB est **Gradient Boosting** avec un **score moyen de 87.62%** de bonnes prédictions.

4 Modèle final

4.1 Choix du modèle final

Après identification des meilleurs modèles sur les jeux de données MAIN et JOB, il s'agit de définir la stratégie de prédictions.

modèles	scores moyens
Meilleur modèle sur données MAIN	0.852328

modèles	scores moyens
Meilleur modèle sur données JOB	0.876235

Le modèle sur jeu de données JOB uniquement présente une meilleure performance lorsqu'il s'agit de prédire les individus ayant un Job (`job_category` non `NA`).

Choix du modèle final

Le **modèle final** sera donc une **combinaison MAIN+JOB**:

- Tous les individus sont prédits avec le meilleur modèle MAIN
- La prédiction des individus disposant d'un job (`job_category` non nul) est remplacée par la prédiction du meilleur modèle JOB.

4.2 Mesure de la performance

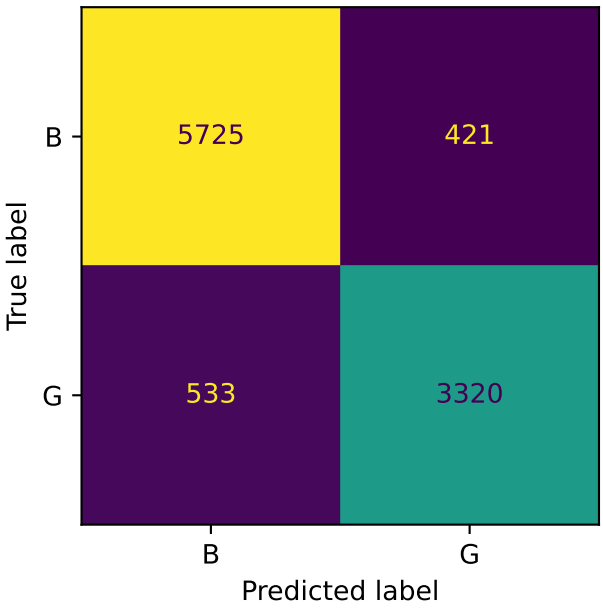
La mesure de la **performance du modèle combiné MAIN + JOB** est effectuée sur le **jeu de données X et y EVAL**.

Performance du modèle final

La performance attendue pour le **modèle combiné MAIN+JOB** est de **90.46%** de bonnes prédictions

4.3 Analyse de la classification

Matrice de confusion sur le jeu de données d'évaluation



Analyse de la classification

	precision	recall	f1-score	support
B	0.91	0.93	0.92	6146
G	0.89	0.86	0.87	3853
accuracy			0.90	9999
macro avg	0.90	0.90	0.90	9999
weighted avg	0.90	0.90	0.90	9999

La matrice de confusion sur le jeu de données d'évaluation confirme globalement l'efficacité de la combinaison de modèles MAIN+JOB avec un **taux global de bonnes prédictions accuracy = (TP+TN)/(nb observations)** de **90%.**

La **sensibilité/recall = TP / (TP+FN)** donne la capacité du modèle à détecter les positifs parmi l'ensemble des vrais positifs.

- Recall sur classe B = 93%
- Recall sur classe G = 86%
- Le modèle est **plus efficace à bien détecter les individus de classe B** que les individus de classe G.

La **precision = TP / (TP+FP)** donne la pertinence de la prédiction du modèle quand il détecte une valeur positive. Autrement dit, il mesure la confiance qui peut être donnée au modèle lorsqu'il fait une prédiction positive.

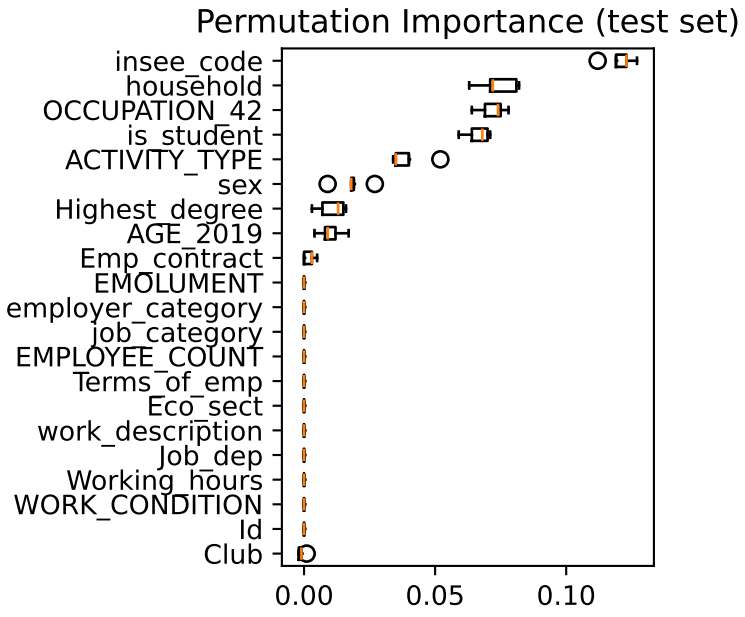
- Précision sur classe B = 91%
- Précision sur classe G = 89%
- Le **modèle est équilibré** : qu'il détecte une classe B ou G, il ne se trompe qu'une fois sur 10.

A noter que la "signification métier" de la variable cible est inconnue. De ce fait, aucune optimisation de la précision ou de la spécificité des classes a été opérée. Seul le score Accuracy a été considéré dans l'optimisation de la performance du modèle.

4.4 Importance des features

On applique aux 2 meilleurs modèles une technique de **mesure d'importance des features par permutation**.

Importance des features dans le modèle MAIN (sans les colonnes Job)

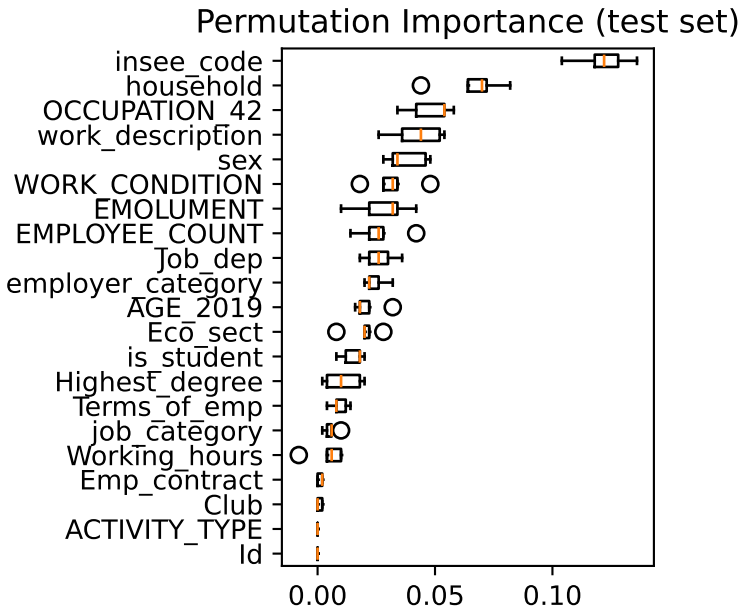


Parmi les features initiales avant feature engineering, les plus influentes sont

- `insee_code`, qui a permis d'introduire les **notions géographiques** (taille de la commune, distance avec la plus proche grande ville, département, région, nombre d'habitants)
  - cf. fonction `insee_code_fe` dans fichier *toolbox_feature_engineering.py*
- `household`, la composition du foyer (foyer célibataire, famille sans enfant, famille monoparentale)
  - cf. fonction `household_fe` dans fichier *toolbox_feature_engineering.py*
- `OCCUPATION_42`, catégorie socio-professionnelle de l'individu
  - cf. fonction `OCCUPATION_42_fe` dans fichier *toolbox_feature_engineering.py*
- `is_student`, qualité d'étudiant de l'individu
  - maintenu en état booléen dans le modèle
- `ACTIVITY_TYPE`, statut professionnel courant de l'individu (employé, chômeur, retraité, inactif)

Le sexe, le niveau d'étude, l'âge, la nature du contrat de travail et la condition sportive ont une influence modérée ou nulle dans le modèle.

Importance des features dans le modèle JOB



Parmi les features initiales importantes dans le modèle JOB, on retrouve les features `insee_code`, `household` et `OCCUPATION_42`.

On note cependant que la feature `ACTIVITY_TYPE` perd de son influence, au profit des features issues du jeu de données Job affinant les informations relatives au travail de l'individu:

- `WORK_CONDITION`, qualifiant le contrat de travail en "temps partiel"/"temps complet"
  - cf. fonction `work_condition_fe` dans fichier *toolbox_feature_engineering_job.py*
- `EMOLUMENT`, le salaire de l'individu
  - cf. fonction `emolument_fe` dans fichier *toolbox_feature_engineering_job.py*
- `EMPLOYEE_COUNT`, codification de la taille de l'établissement employant l'individu
  - cf. fonction `employee_count_fe` dans fichier *toolbox_feature_engineering.py*

Il est intéressant de noter que la feature `sex` à influence modérée dans le modèle MAIN prend de l'importance dans le modèle JOB.