Projet NLP 3A ENSAE

Loïc Thomas

March 2024

Code disponible ici : https://github.com/Serv360/projetNLP_3A_ENSAE

1 Présentation de la tâche

Dans le cadre du cours *Machine Learning for Natural Language Processing* (SE364) dispensé en troisième année à l'Ecole nationale de la Statistique et de l'Administration économique (ENSAE), je travaille sur des données du projet Socface.

> Le projet Socface associe des archivistes, démographes et informaticiens pour analyser les documents de recensement français et en extraire des informations à très grande échelle. L'objectif est de rassembler et de traiter par reconnaissance automatique de l'écriture toutes les listes nominatives manuscrites des recensements de 1836 à 1936.
> Produites tous les cinq ans, ces listes sont organisées spatialement (commune ; quartiers, hameaux ou rues ; maisons ; ménages) et résument les informations issues du recensement, en listant chaque individu avec certaines de ses caractéristiques, comme son nom, son année de naissance ou sa profession.
> Le projet vise à tirer parti de ce matériel d'archives pour produire une base de données de tous les individus ayant vécu en France entre 1836 et 1936, qui sera utilisée pour analyser les changements sociaux sur une période de 100 ans. Un impact important de Socface sera l'accès public aux listes nominatives : elles seront mises à disposition gratuitement, permettant à quiconque de parcourir des centaines de millions d'enregistrements.[a]
>
> [a] https://socface.site.ined.fr/

Je vais m'intéresser à un des champs du recensement désigné de la façon suivante : *position dans le ménage*, *situation par rapport au chef de ménage* ou encore *link* en anglais. En particulier, un individu recensé peut être chef de ménage, *chef*, ou non. Ce champ est important, par exemple, pour regrouper les ménages afin de faire des analyses sociologiques à cette échelle, encore aujourd'hui privilégiée à l'Insee.

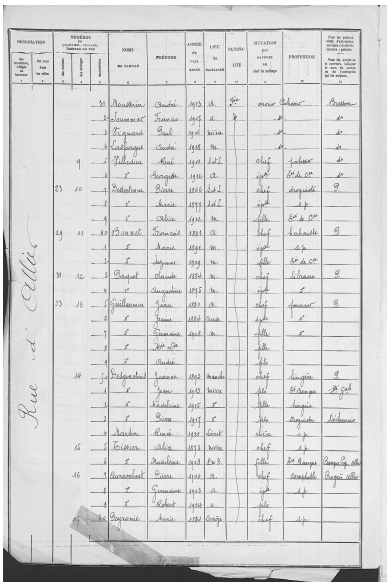
Le fait d'être *chef* n'est pas toujours renseigné, et pas toujours bien renseigné ou lisible. Dès lors, il est intéressant de savoir le prédire à partir des autres champs bien renseignés de la ligne. Par exemple, dans les données que j'utilise pour entraîner mes modèles à réaliser cette tâche, le champ *link* est vide pour 4389 individus sur 25081.

2 Présentation et description des données

Je dispose de 1218 pages de recensement contenant en tout 25081 individus[1]. Les pages sont disponibles au format Json sous la forme de longues chaînes de caractères. Les lignes sont séparées par des sauts de ligne (\n) et les champs sont séparés par des caractères spéciaux Ⓐ, Ⓑ, Ⓒ, Ⓓ, Ⓔ, Ⓕ, Ⓗ, Ⓘ, Ⓙ, Ⓚ, Ⓛ, Ⓜ, Ⓞ et Ⓟ. Ils correspondent respectivement aux champs suivants : *age*, *birth_date*, *civil_status*, *education_level*, *employer*, *firstname*, *link*, *lob*, *maiden_name*, *nationality*, *observation*, *occupation*, *surname*, *surname_household*.

La Figure 1a montre un exemple de formulaire de rencensement sur lequel une extraction des informations est réalisée avec de la computer vision pour obtenir les données textuelles. La Figure 1b présente le nombre de lignes qui sont vides pour chacun des champs. Enfin, la Figure 1c donne le nombre de valeurs uniques. Cela est très important pour le choix de la représentation numérique des champs. S'il y a un faible nombre de valeurs uniques, on peut imaginer faire du one-hot-encoding. Si le nombre de valeurs uniques est très grand, il sera pertinent de les agréger si on veut produire, à partir d'elles, des variables catégorielles.

[1] base de donnée créée par Teklia, disponible ici https://arkindex.teklia.com/element/1109b196-7224-429a-9ef6-2f128034ed51



(a) Exemple de feuille de recensement.

age	8645
birth_date	17737
civil_status	14376
education_level	25081
employer	22170
firstname	150
link	4345
lob	15845
maiden_name	25081
nationality	11767
observation	24479
occupation	8902
surname	5922
surname_household	19441

(b) Nombre de lignes vides par champ.

age	242
birth_date	150
civil_status	6
employer	1046
firstname	2125
link	935
lob	2786
nationality	61
occupation	1558
surname	7078
surname_household	3107
target	2

(c) Nombre de val. uniques par champ.

Figure 1

3 Nettoyage et création des champs numériques continus et catégoriels

3.1 Nettoyage des données

La première étape est d'obtenir un dataframe de taille 25081 rows $\times$ 14 columns. Les 14 colonnes correspondant aux 14 champs énumérés dans la partie précédente.

Ensuite, j'ai remarqué la présence de nombreux *idem* et *Idem* dans les données. Ils signifient que la valeur de cette ligne est la même que la première valeur qui n'est pas *idem* ou *Idem* dans la même colonne. L'utilisation des fonction *replace* et *ffill* de la bibliothèque pandas permet de les remplacer par les bonnes valeurs.

J'ai fait le choix de drop les colonnes entièrement vides ou quasiment suivantes : *education_level*, *maiden_name* et *observation*. Elles ne contiennent pas ou pas assez d'information pour la prédiction.

J'ai gardé seulement les lignes qui possède un *link* et j'ai éliminé les lignes pour lesquelles ce champ est vide. J'ai créé un champ, appelé *target* valant 1 si le *link* correspond à un chef de famille et 0 sinon. Pour cela, il faut bien faire attention aux différentes dénominations de chef. Elles sont disponibles dans le code, il y en a 43. En prenant seulement "chef", le modèle est forcément moins bon.

Enfin, le dataset est déséquilibré : 4637 pour 20692 lignes avec un *link* non vide. J'ai fait le choix de le rééquilibrer en enlevant au hasard $20692 - 2 * 4637$ individus qui ne sont pas chefs et en gardant donc 4637 individus non chefs. On perd ainsi beaucoup d'informations mais cela permet d'avoir un prédicteur non biaisé. Il existe des techniques pour bien choisir les individus à garder ou pour créer de nouveaux individus à partir des vrais qui rassemblent un maximum de l'information mais je n'ai pas developpé ce point-là.

Le dataset est maintenant nettoyé. Nous avons déjà à ce stade notre colonne *target* qu'on souhaite prédire et on a conservé 9274 individus.

3.2 Création des champs numériques continus et catégoriels

L'étape suivante est de créer des champs numériques pour les intégrer dans les classifieurs. J'ai fait le choix d'utiliser des classifieurs de Machine Learning classiques en créant moi-même des champs pertinents à la main. Des modèles

d'embeddings du type word2vec, GloVe, ELMo ou encore BERT qui peuvent transformer les mots en vecteurs numériques seraient tout à fait adaptés. Je n'ai pas fait ce choix-là car je crois qu'en créant un petit nombre de catégories au sein des champs les plus pertinents pour prédire le chef, on arrive à un très bon résultat. Une telle démarche permet également d'avoir une bonne compréhension du modèle.

J'ai choisi de me concentrer sur les colonnes suivantes qui présentaient pour moi le plus d'intérêt pour la tâche de prédiction : *firstname*, *civil_status*, *occupation* et *age*.

Avec le **prénom** de l'individu, il est possible d'avoir une bonne idée de son sexe. Or, les chefs de famille sont principalement des hommes dans la classification du recensement de cet époque. Par ailleurs, *firstname* est le champ le plus renseigné dans mes données, il a donc un pouvoir explicatif fort. Enfin, j'ai accès à une autre source de données qui pour chaque prénom (6947 prénoms) me donne son nombre de porteurs féminins et masculins. De là, je crée donc une variable numérique continue de la manière suivante :

$$\text{name\_value} = \arcsin\left(\frac{\text{PREVALENCE\_MALE} - \text{PREVALENCE\_FEMALE}}{\text{PREVALENCE\_MALE} + \text{PREVALENCE\_FEMALE}}\right) / \frac{\pi}{2}$$

Je donne une valeur de 0 à l'individu si son prénom n'est pas renseigné ou si la chaîne de caractères est vide. Sans la fonction arcsin, les valeurs seraient toutes collées très proches de -1 (prénoms féminins) et 1 (prénoms masculins) car la majorité des prénoms sont très genrés. L'utilisation de la fonction arcsin permet d'éclater un peu plus les valeurs. Cela a un intérêt tout d'abord pour la visualisation des variables explicatives mais aussi pour certains modèles qui prédisent mieux lorsque les données ont une distribution plus uniforme.

Le **statut civil** présente un avantage important : c'est une variable standardisée qui ne prend que 6 valeurs distinctes : *Garçon*, *Femme mariée*, *Fille*, *Homme marié*, *Veuf*, *Veuve*. Il y a un léger inconvénient toutefois, un peu plus de la moitié des lignes n'ont pas de *civil_status* (voir Figure 1b). À partir de cette colonne je crée un champ *sex_from_civil_status* qui vaut 1 si le statut civil est masculin, -1 s'il est féminin et 0 s'il est absent. Je crée également 6 dummies variables pour chacune des valeurs.

L'**occupation** est également très intéressante. Elle est globalement bien renseignée, en particulier très bien renseignée pour les chefs de famille. Le métier permet d'avoir une idée sur le sexe de l'individu et également sur son âge. En effet, pour les enfants le métier n'est souvent pas renseigné ou bien contient une des nombreuses chaînes de caractères qui indiquent que l'individu n'a pas de profession. Le nom du métier est souvent genré, il constitue donc un bon proxy du genre. J'ai choisi de créer trois champs à partir de la colonne *occupation* : *has_no_occupation*, *is_farmer*, *has_male_job*. La première permet de discriminer les femmes qui ne travaillent pas et les enfants. La seconde permet de discriminer les hommes qui travaillent dans le milieu agricole et la troisième les hommes qui travaillent. La deuxième est probablement fortement inclue dans la troisième mais je l'ai prise quand même car le monde agricole est le principal secteur d'activité à cette époque.

Enfin l'**âge** a également un pouvoir explicatif très fort. J'ai constaté que les moins de 25 ans sont rarement chefs de famille, ce qui permet d'exclure de nombreuses personnes. Le champs *age* est plutôt bien renseigné, autant que *occupation*. J'ai enlevé les valeurs non entières et non comprises entre 0 et 110 et j'ai imputé les valeurs manquantes par 25, le seuil me semble-t-il à partir duquel les hommes commencent à devenir chefs de famille (voir Figure 2). J'ai aussi créé une variable indicatrice codant si l'âge est imputé ou non.

La Figure 2 permet de visualiser le fait d'être chef de famille en fonction des 12 champs calculés (tous sauf l'indicatrice d'imputation de l'âge) à partir des quatre colonnes *firstname*, *civil_status*, *occupation* et *age*. J'ai appliqué un shift aléatoire autour de 0 et 1 pour mieux voir.

J'explique briévement pourquoi je me suis pas intéressé aux autres colonnes. Les arguments sont issus de la Figure 1b et 1c. La **date de naissance** est redondante avec l'âge et elle est renseignée beaucoup moins souvent. L'**employeur** est quasiment jamais renseigné et ce sont majoritairement des valeurs uniques, difficilement agrégeables. La **localisation** (*lob*) est rarement renseignée et apporte a priori pas tellement d'information. C'est similaire pour la **nationalité**. Le **nom de famille** est souvent renseigné mais il y a de nombreuses valeurs uniques non agrégeables (plus grand nombre parmi les champs avec 7078 valeurs uniques). Enfin, le **nom de famille du ménage** est très rarement renseigné.

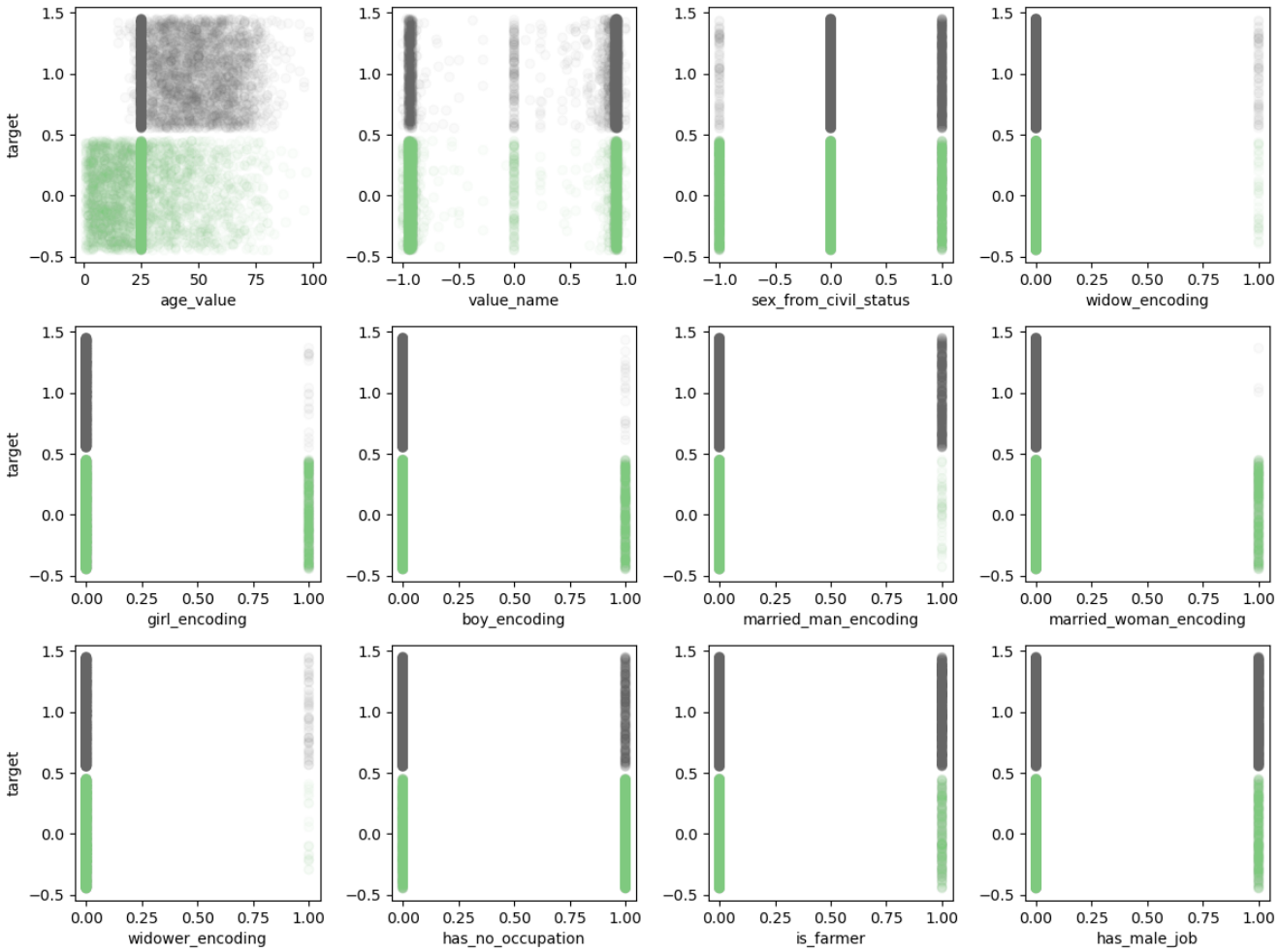


Figure 2: Être un chef de famille (1 : noir, 0 : vert) en fonction des champs calculés.

Il aurait été intéressant toutefois de regarder l'ordre dans lequel apparaissent les individus sur les feuilles de recensement. Le chef de ménage apparaît souvent en premier. Le nom de famille/nom de famille du ménage et la localisation/nationalité peuvent alors être de très bons indicateurs d'une unité de ménage qui se suit. Je n'ai pas creusé de ce côté-là. L'avantage de mon choix de variables et de mes modèles est que je peux prédire le fait d'être chef de famille sur des individus isolés. Par exemple, peut-être que certaines feuilles sont non numérotées et que leur ordre a été perdu. Dans ce cas, il peut être difficile d'utiliser la technique décrite pour détecter les unités de ménage, parmi les individus en début de feuille.

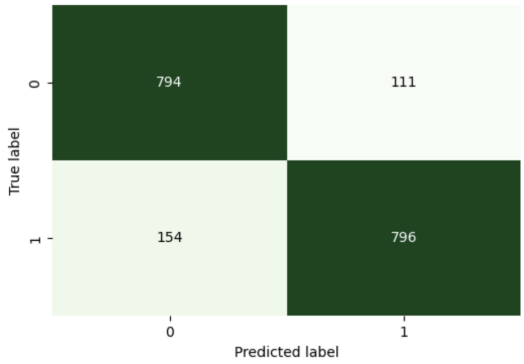
4 Comparaison des modèles

J'ai désormais un dataset contenant 13 variables explicatives et ma target. Je compare quatre types de classifieurs de sklearn : *GradientBoostingClassifier*, *RandomForestClassifier*, *GaussianProcessClassifier* avec un kernel *RBF* et *KNeighborsClassifier*.

Je fais un grid search pour les random forests et les classifieurs KNN pour trouver les meilleurs hyperparamètres (*max_depth* et *n_neighbors*). Je fais ensuite une validation croisée (3 groupes) pour le meilleur classifieur de chacun des quatre types. J'obtiens les résultats disponibles dans la Table 3a. On note en particulier que l'entraînement du GaussianProcessClassifier est particulièrement long, 3 minutes contre quelques secondes maximum pour les autres. Le classifieur KNN a le désavantage de prendre de la place en mémoire, en particulier lorsque le set d'entraînement est très grand ce qui peut être le cas pour le recensement.

Classifier	Accuracy
GaussianProcessClassifier	0.8435099
RandomForestClassifier (max_depth=10)	0.8480714
KNeighborsClassifier (n_neighbors = 20)	0.8274758
GradientBoostingClassifier	0.8482275

(a) Performances des classifieurs.



(b) Matrice de confusion (GBClassifier).

Figure 3

On voit sans surprise que le GradientBoostingClassifier a de meilleures performances que les trois autres. C'est souvent le cas, même si dans notre cas l'accuracy de tous est très proche. En particulier, on calcule la matrice de confusion sur un jeu de test pour un entraînement sur 80% des données en Figure 3b. L'accuracy est de 0.8571429. Les performances de prédiction sur les deux classes sont équilibrées.

Enfin l'analyse de l'importance, Figure 4 en Annexes, permet de voir que les variables qui contiennent le plus d'information sont *has_no_occupation*, *age_value* et *value_name*. Je m'attendais à la présence de *age_value* et *value_name* en tête. *has_no_occupation* est un peu plus étonnant mais la Figure 2 montre bien une corrélation entre *has_no_occupation* et *target*. Une matrice similaire à une matrice de confusion en apporterait la confirmation chiffrée.

5 Comment appliquer le modèle sur les données réelles.

J'ai déjà abordé cette question dans les parties précédentes. Mes transformations pour obtenir les 13 champs calculés numériques continus ou catégoriels sont applicables sur n'importe quelle ligne extraite par Computer Vision dont les informations ont été typées. La classification se fait ensuite aisément avec le Gradient Boosting Regressor. On privilégiera ce dernier car il est rapide à entraîner, il est rapide pour prédire, il a peu tendance à l'overfitting, il ne demande pas beaucoup de place en mémoire et on a obtenu grâce à lui les meilleurs résultats.

Ma pipeline permet alors de prédire pour des lignes isolées si l'individu est un chef de famille ou non.

6 Conclusion

J'obtiens des bons résultats de classification des chefs de famille grâce à la création de 13 champs calculés à partir des colonnes *firstname*, *civil_status*, *occupation* et *age* et des classifieurs *GradientBoostingClassifier*, *RandomForestClassifier*, *GaussianProcessClassifier* avec un kernel *RBF* et *KNeighborsClassifier*. Je privilégierais le *GradientBoostingClassifier* du fait de ses performances temporelles, spatiales et de précision plus grandes, détaillées en section 4 et 5.

Pour aller plus loin et améliorer les résultats, il serait possible de mieux choisir les paramètres du Gradient Boosting classifier, en particulier *max_depth*. Calculer d'autres champs pertinents est tout à fait possible, notamment encore à partir de la colonne occupation qui contient beaucoup d'information. En essayant d'autres aggrégations, il serait possible d'améliorer l'accuracy. Attention toutefois à l'overfitting si on choisit les champs en fonction de l'accuracy sur notre sous-échantillon. Utiliser l'ordre des individus dans les feuilles de recensement pourrait apporter beaucoup d'information également mais le domaine d'application est un peu différent. Enfin, l'utilisation de modèles d'embedding est également adapté à cette tâche pour produire des vecteurs numériques utilisables dans les classifieurs.

7 Annexes

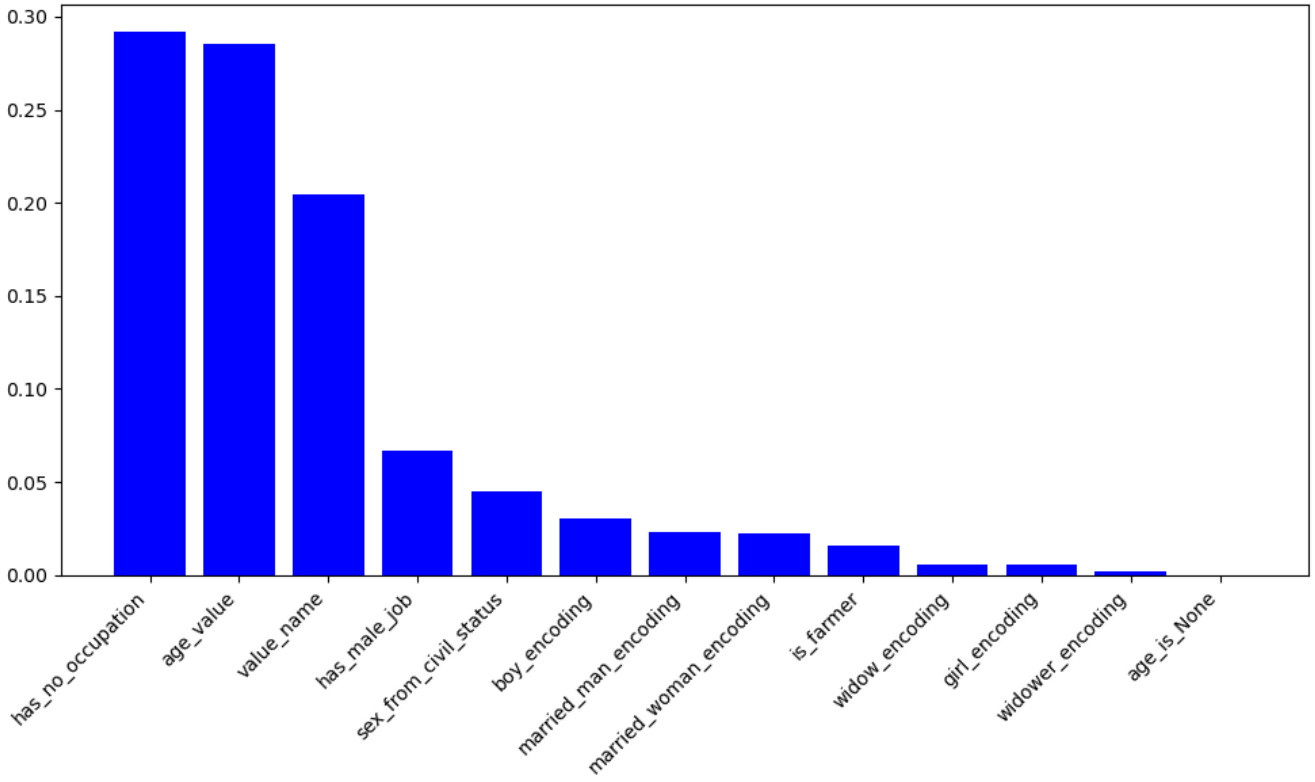


Figure 4: Importances des variables explicatives pour la Random Forest (max_depth = 10)

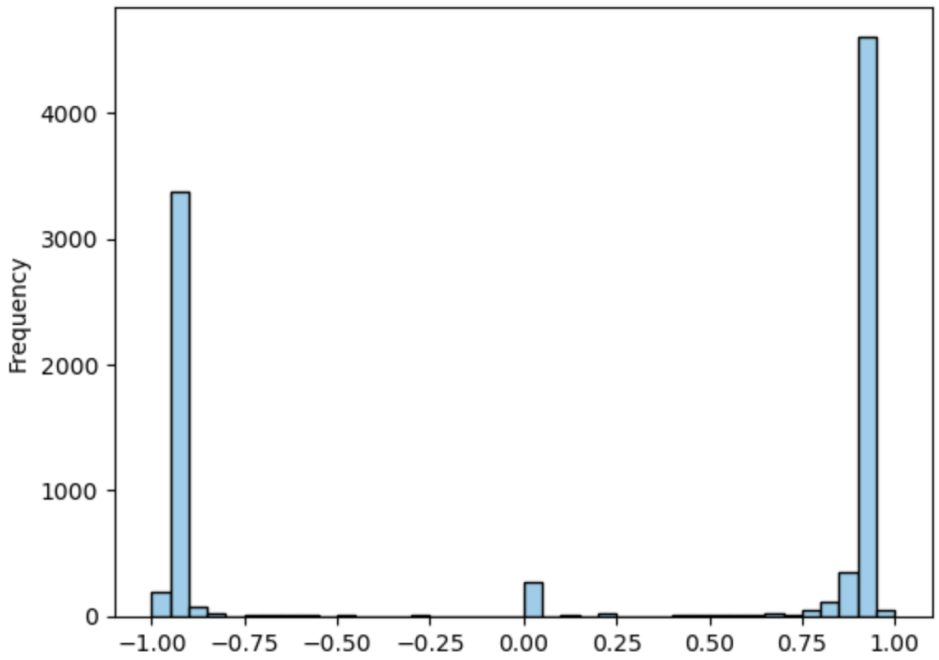


Figure 5: Distribution de la variable *value_name* obtenu à partir du champ *firstname*.