Good In Tech

Les outils responsables de l'IA

Une approche critique par la pratique

Institut Mines-Télécom Business School

TELECOM Paris TELECOM SudParis INSTITUT POLYTECHNIQUE DE PARIS

FONDATION DU RISQUE Louis Bachelier

SciencesPo ÉCOLE DU MANAGEMENT ET DE L'INNOVATION

Auteurs

Guillaume Peltier, chargé de mission pour la Chaire Good In Tech

Coordonné par **Jean-Marie John-Mathews**, Data scientist et membre de la Chaire Good In Tech

Résumé

L’intelligence artificielle est l’objet de controverses diverses et variées ces dernières années : discrimination, atteinte à la vie personnelle, faille de sécurité, etc. De nombreuses communautés académiques en *Machine Learning* travaillent à rendre ces algorithmes plus responsables par la création de méthode de débiaisement, d’anonymisation, etc. Dans ce rapport nous cherchons à mettre à l’épreuve les « outils de l’IA responsable ». Quelles sont leurs limites opérationnelles ?

Pour cela, nous construisons un algorithme de prédiction de défaut bancaire et simulons trois types de dysfonctionnements éthiques : discrimination au genre, atteinte à la vie personnelle et faille de sécurité. En appliquant les méthodes de l’IA responsable sur ces dysfonctionnements, nous avons pu montrer un certain nombre de difficultés opérationnelles : outils peu matures, outils efficaces uniquement dans des cas très précis, coût de calcul, etc.

L’application *a posteriori* de méthodes techniques de correction possède donc des limites. Il est nécessaire que l’approche responsable de l’IA transparaisse dans les choix organisationnels, politiques et techniques dès la phase de conception des algorithmes. Corriger les incidents de l’IA suppose d’aborder les dysfonctionnements de l’IA d’une manière holistique.

Contents

2 Travaux préliminaires : construction d'un algorithme de prédiction du risque de défaut

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt

from sklearn import model_selection
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from copy import deepcopy

import utility as uti

import seaborn as sns
sns.set_theme(style='darkgrid')

plt.rcParams['figure.dpi'] = 100
```

[2]:
```
seed = 2021
```

2.1 Présentation de la base german credit scoring / cas d'usage (prédiction risque de défaut)

Pour étudier les différentes catégories d'éthique de l'intelligence artificielle définies plus haut, nous utilisons une version nettoyée de la base de données : German Credit Risk. Cette base de données est mise à disposition par le professeur Hofmann et contient 1000 entrées chacune composée de 22 variables. Chaque entrée représente une personne qui a contracté un emprunt à une banque. Chaque personne est alors placée dans la classe "0" si elle a remboursé son crédit ou dans la classe "1" si elle n'a pas réussi à le rembourser. Cette classification correspond à la variable "default".

[3]:
```
credit = pd.read_csv("data/german_credit_prepared.csv",
                     sep=",",
                     engine="python")
credit.iloc[0]
```

[3]:
```
default                                                          0
account_check_status                                       < 0 DM
duration_in_month                                                6
credit_history           critical account/ other credits existing (not …
purpose                                        domestic appliances
credit_amount                                                 1169
savings                                 unknown/ no savings account
present_emp_since                                    .. >= 7 years
installment_as_income_perc                                       4
sex                                                           male
```

```
personal_status                                                 single
other_debtors                                                     none
present_res_since                                                    4
property                                                   real estate
age                                                                 67
other_installment_plans                                           none
housing                                                            own
credits_this_bank                                                    2
job                                          skilled employee / official
people_under_maintenance                                             1
telephone                      yes, registered under the customers name
foreign_worker                                                     yes
Name: 0, dtype: object
```

2.2 Création du modèle

On va créer un modèle qui décidera s'il faut accorder un prêt à un nouvel individu, le modèle prédisant si ce dernier arrivera à rembourser ce prêt ou non. On cherche ainsi à créer un classifieur qui étant donné les caractéristiques d'une personne la place dans la classe 0 s'il faut lui accorder un prêt (car il y a de grandes chances qu'elle rembourse ce prêt) et dans la classe 1 sinon. L'idée est par la suite d'appliquer les différentes méthodes d'éthique au modèle ainsi créé. On choisit ici une régression logistique car c'est un modèle très utilisé dans les banques à la fois pour sa simplicité et son efficacité. De plus, utiliser une régression logistique illustre la pertinence des problématiques éthiques pour les modèles les plus simples tels que la régression logistique.

[4]:
```
y = credit.default
X = credit.drop(columns=["default"])

# class the variable between categorical and ordinal
cat_variables = [col for col in X.columns if credit[col].dtype==object]
ord_variables = [col for col in X.columns if credit[col].dtype==int]

preprocess = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(drop='first'), cat_variables),
        ('ord', StandardScaler(), ord_variables)
    ])

model = Pipeline(
        [
            ('prepro', preprocess),
            ('logreg', LogisticRegression())
        ]
)
```

[5]:
```
split = model_selection.train_test_split(X, y,
                                         test_size=0.20,
```

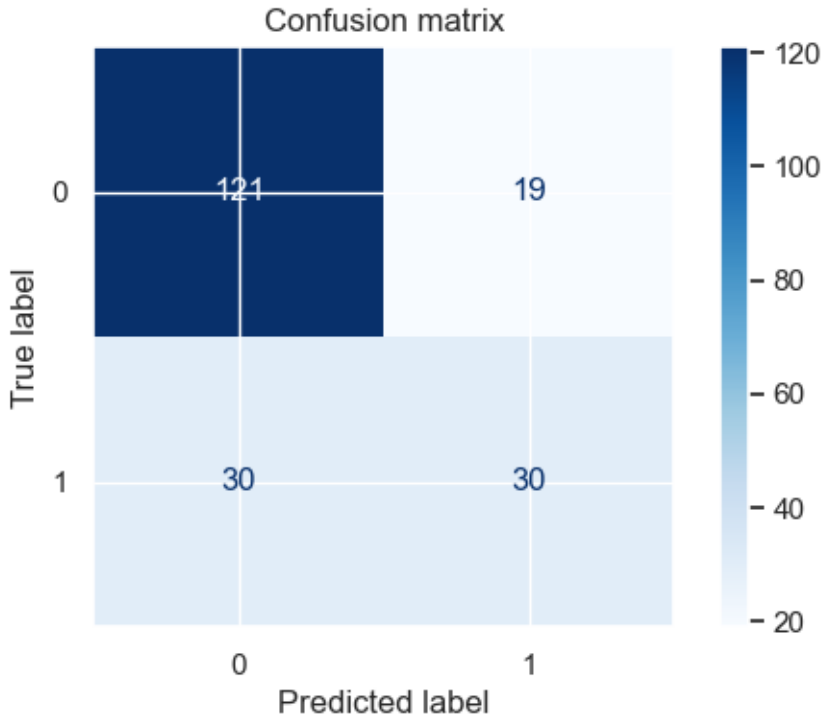
```
                                            random_state=seed)
X_train, X_test, y_train, y_test = split

logreg = deepcopy(model.fit(X_train, y_train))
```

2.3 Présentation de la performance du modèle

Les performances du modèles sont mesurées par les quatre notions suivantes :

- *Sensitivity* : la fraction de labels 1 prédits correctement, ici la fraction de personnes ayant fait défaut qui sont prédit comme incapable de rembourser sur la totalité des personnes prédites comme incapables de rembourser.
- *Specificity* : la fraction de labels 0 prédits correctement, ici la fraction de personnes ayant remboursé leur prêt et qui sont prédit comme capable de rembourser, sur la totalité des personnes qui sont prédit comme capables de rembourser.
- *Precision* : la fraction de label prédit comme 1 correct, ici la fraction de personnes prédites comme incapable de rembourser et qui ont fait défaut sur la totalité des personnes ayant fait défaut.
- *Accuracy* : la fraction de label correct sur la totalité des personnes, ici la fraction de personnes donc le label est correct sur la totalité des personnes.

[6]:
```
uti.mesure_clas(logreg, X_test, y_test)
```




```
Sensitivity  Specificity   Precision     Accuracy
```

```
0.86          0.50          0.61          0.76
```

Les valeurs obtenues pour chacune des mesures nous permettent de dire que le modèle entraîné fait de bonnes prédictions. On va utiliser ce modèle pour les trois notebooks illustrant respectivement les notions de robustness, fairness et de privacy.

3 Outils pour la lutte contre les discriminations algorithmiques

[1]:
```
%%capture

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from tqdm import tqdm

from sklearn.metrics import plot_confusion_matrix
from aif360.datasets import BinaryLabelDataset
from aif360.metrics import ClassificationMetric
from aif360.algorithms.preprocessing import Reweighing
from aif360.algorithms.postprocessing import CalibratedEqOddsPostprocessing

plt.rcParams['figure.dpi'] = 100
```

[2]:
```
%%capture
%run ./Introduction.ipynb
```

[3]:
```
NEW = False
```

3.1 Présentation de la notion d'équité

3.1.1 Définition de la *fairness*

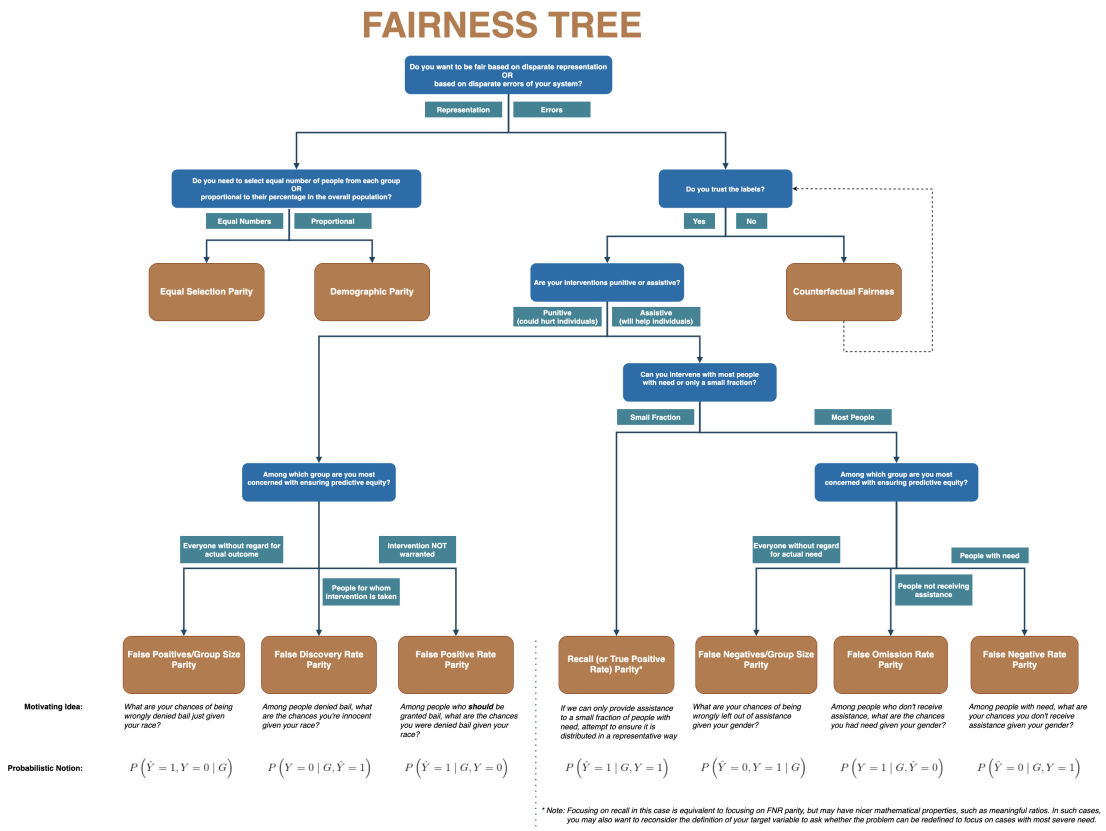
On appelle biais algorithmique une erreur systématique et reproductible d'un système informatique qui conduit les sorties de ce système à favoriser un groupe d'individus plutôt qu'un autre. Cette notion est déjà abordée par Joseph Weizenbaum en 1967 qui décrit les programmes informatiques de l'époque comme des outils mimiquant le raisonnement humain pour résoudre des problèmes. Il dit dans *Computer Power and Human Reason* que les règles suivies par les programmes internalisent la vision du monde de leur créateur, ce qui inclut leurs biais et leurs attentes. Cela concerne également les algorithmes qui apprennent par eux-même puisque, s'il existe une base de données, celle-ci est confectionnée par des hommes, s'il existe des règles (ex apprentissage par renforcement), ces dernières ont été fixées par les programmeurs. Les algorithmes d'apprentissage sont donc eux aussi concernés par les biais algorithmiques.

Le principe de *fairness* en apprentissage statistique correspond aux différentes manières de corriger ou de mitiger ce problème de biais algorithmique. S'il est difficile de donner une définition universelle de la *fairness*, de manière générale on considère qu'un résultat est équitable s'il est indépendant d'un certain jeu de variables appelées *variables sensibles*. Pour une personne les variables sensibles correspondent classiquement aux attributs comme le sexe, l'ethnicité, l'orientation sexuelle, et tout autre caractéristique sur laquelle la discrimination est interdite.

Bien que le problème de biais algorithmique soient connus depuis les années 70, les recherches en matière de *fairness* sont très récentes. Les publications les plus importantes datent ainsi seulement de 2018.

3.1.2 Définition mathématique

Bien que la définition littéraire de *fairness* soit intuitive, il est nécessaire de faire des hypothèses pour mettre en place une définition mathématique. Il existe ainsi différentes définitions de la *fairness* en fonction des hypothèses retenues. De manière générale, ces définitions visent à minimiser la déviation (ou le biais statistique) d'une ou plusieurs *métriques de parité*, cela pour tous les individus ou tous les groupes d'individus. La définition varie en fonction de la métrique retenue et il existe un grand nombre de métriques de parités comme on peut le voir sur (liste). Certains outils existent pour choisir une métrique plutôt qu'une autre en fonction du rôle qu'aura le modèle statistique. Par exemple *Aquiteas*, qui est un ensemble d'outils de *fairness* développés par le centre *data science and public policy* de l'université de Chicago, propose l'arbre de décision suivant :



Nous allons maintenant donner trois définitions de la *fairness* en utilisant trois métriques distinctes. On utilise dans ce qui suit les notations suivantes :

- Y : la caractéristique binaire de sortie
- X : le vecteur de caractéristique d'entrée sans la caractéristique sensible
- S : la caractéristique sensible
- $\hat{Y}$: la prédiction de la caractéristique de sortie

Définition 1 : *equalized odds*

On dit que $\hat{Y}$ satisfait l'égalité des chances par rapport à S et Y si :

$$P(\hat{Y}=1 \mid S=0, Y=y) = P(\hat{Y}=1 \mid S=1, Y=y); \quad y=0,1$$

Autrement dit, $\hat{Y}$ est indépendant de S conditionnellement à Y.

Définition 2 : *demographic parity*

On dit que $\hat{Y}$ satisfait la parité démographique par rapport à S si :

$$P(\hat{Y} = 1 \mid S = 0) = P(\hat{Y} = 1 \mid S = 1)$$

Autrement dit, $\hat{Y}$ est indépendant de S.

Définition 3 : *counterfactual fairness*

On dit que $\hat{Y}$ satisfait l'équité contrefactuelle par rapport à S et X si :

$$P(\hat{Y} = 1 \mid S = 0, X = x) = P(\hat{Y} = 1 \mid S = 1, X = x)$$

Autrement dit, $\hat{Y}$ est indépendant de S conditionnellement à X.

L'égalité des chance et la parité démographique sont des métriques de parité de groupe (*group fairness*), on cherche à avoir le même traitement pour des groupes de personnes définies par leurs variables sensibles S. Alors que l'équité contrefactuelle est une métrique de parité individuelle (*individual fairness*), on cherche à fournir un traitement similaire à des personnes similaires peu importe leurs variables sensibles.

Il est tout à fait possible d'avoir une fusion entre *group fairness* et *individual fairness*, produisant alors un *subgroup fairness*.

3.1.3 Application de la *fairness*

Les problèmes de biais algorithmiques peuvent survenir des données ou du modèle. Pour cela, on catégorise les méthodes d'implémentation des solutions de *fairness* en trois parties :

- pre-processing : ces méthodes qui ne dépendent pas du modèle, tendent à mitiger les problèmes de *fairness* en modifiant les données elles-même. Il s'agit souvent de transformer une ou plusieurs caractéristiques ou de changer la représentation des groupes.
- in-processing : ces méthodes incorporent dans le processus d'entraînement les métriques que l'on cherche à minimiser.

- post-processing : ces méthodes n'ont pas besoin d'un accès au modèle d'entraînement et traitent le problème de *fairness* directement sur les prédictions et non sur la base de données d'entraînement comme le font les méthodes in-processing.

Dans la suite, nous allons illustrer les différents biais qui peuvent exister dans notre modèle de prédiction en prenant comme variable sensible le sexe de l'individu. Nous verrons ensuite l'implémentation de méthodes *pre-processing*, *post-processing* et *in-processing* pour résoudre ces biais.

3.2 Mise en situation et implémentations

3.2.1 Utilisation de la matrice de confusion

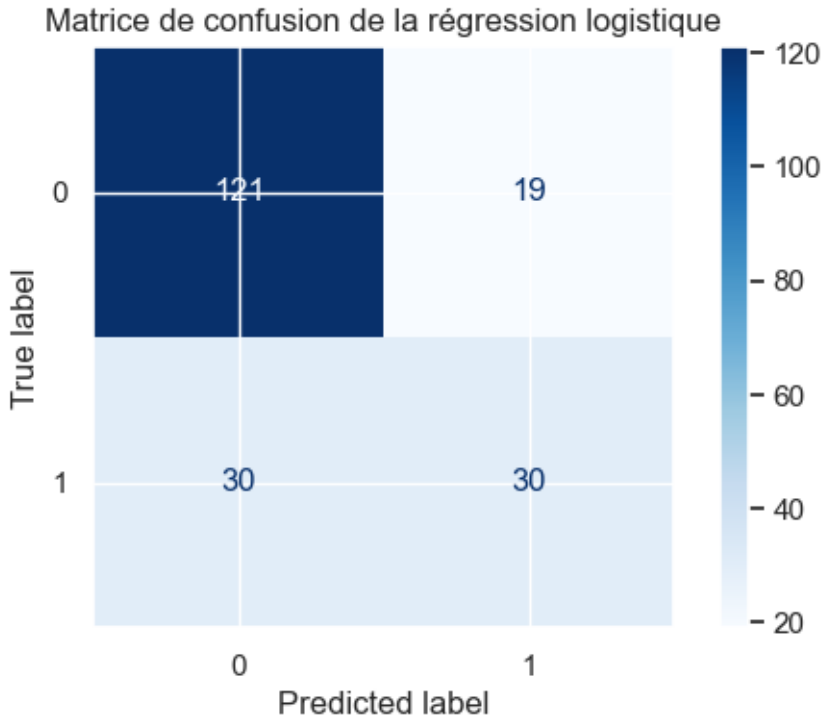
La majorité des mesures de partialité sont basées sur la matrice de confusion. Elle comporte quatre classes :

- Vrai positif (VP) : la vraie classe est 1 et la classe prédite est 1
- Faux positif (FP) : la vraie classe est 0 et la classe prédite est 1

- Vrai négatif (VN) : la vraie classe est 0 et la classe prédite est 0
- Faux négatif (FN) : la vraie classe est 1 et la classe prédite est 0

Les métriques de partialité sont calculées à partir des quatre valeurs associées à ces quatre classes pour des groupes de personnes partageants des caractéristiques communes (ex: sex, ethnicité).

[4]:
```
title = 'Matrice de confusion de la régression logistique'
disp = plot_confusion_matrix(logreg,
                             X_test,
                             y_test,
                             cmap=plt.cm.Blues)
ax = disp.ax_.set_title(title)
plt.show()
```



Dans cette représentation on a :

- Vrai positif : bas à droite
- Faux positif : haut à droite
- Vrai négatif : haut à gauche
- Faut négatif : bas à droite

3.2.2 Étude de cas

Prenons le cas de figure suivant : la banque allemande utilise le modèle de régression logistique pour accepter ou refuser les prêts de ses clients et souhaite vérifier si son modèle traite les hommes

et les femmes de la même manière. Dans ce cas, la variable sensible est donc `sex`. Avant de choisir la métrique à optimiser à l'aide de l'arbre de décision ci-dessus, nous allons examiner comment la librairie *AI Fairness 360*, qui permet un calcul efficace des différentes métriques de *fairness*, fonctionne.

Mise en place On utilise *AIF360* de la manière suivante :

- On crée un dataframe `results` qui contient la sortie Y, la prédiction $\hat{Y}$ et la variable sensible encodée en binaire, cela pour chaque entrée de la base de données de test.
- On construit deux objets de type BinaryLabelDataset, un pour les prédictions (qui prend en argument un tableau contenant la prédiction $\hat{Y}$ et la variable sensible pour chaque entrée de la base de données de test) et un pour les sorties (les arguments étant la sortie Y et la variable sensible)
- On utilise ensuite l'objet *ClassificationMetric* qui prend en entrée les deux objets de type *BinaryLabelDataset* construits ci-dessus. On doit également préciser à cette métrique quel groupe est privilégié et quel groupe ne l'est pas. Dans notre cas, on considère que les privilégiés sont les hommes et les discriminés sont les femmes. Cet objet permet de calculer un très grand nombre de métriques. Ces dernières sont énoncées à l'adresse (liste).

[5]:
```
def model2fair(modelf, X_test, y_test, col=X.columns):
    y_test_df = pd.Series(y_test, index=X_test.index, name='default')
    y_pred_df = pd.Series(modelf.predict(X_test[col]),
                          index=X_test.index, name='default')

    sex_bin = X_test.sex.map({'male': 0, 'female': 1})

    results_test = pd.concat([y_test_df, sex_bin], axis=1)
    results_pred = pd.concat([y_pred_df, sex_bin], axis=1)

    bld_test = BinaryLabelDataset(df=results_test,
                                  label_names=['default'],
                                  protected_attribute_names=['sex'])

    bld_pred = BinaryLabelDataset(df=results_pred,
                                  label_names=['default'],
                                  protected_attribute_names=['sex'])

    fairness_metrics = ClassificationMetric(bld_test, bld_pred,
                                            unprivileged_groups=[{'sex': 1}],
                                            privileged_groups=[{'sex': 0}])
    return fairness_metrics, (bld_test, bld_pred)
```

[6]:
```
fairness_metrics, (bld_test, bld_pred) = model2fair(logreg, X_test, y_test)
```

On vient de construire l'objet `fairness_metrics`, il nous reste à choisir quelle métrique nous allons optimiser.

Choix de la mesure En suivant l'arbre d'*Aequitas* un des chemins possible est le suivant: **Voulez-vous être équitable sur la base d'une représentation hétérogène ou sur la base**

d'erreurs hétérogènes de votre système ?

Dans notre cas, on cherche à traiter les personnes indépendement de leur sexe, on veut ainsi être équitable sur la base d'erreurs hétérogène.

Faites-vous confiance aux labels ?

Nous répondons oui à cette question car la variable `sex` est difficilement falsifiable.

Votre intervention est-elle punitive ou positive ?

Si le résultat de la prédiction vaut 1, alors le prêt est refusé. Pour cela, l'intervention est punitive.

Dans quel groupe êtes-vous le plus soucieux d'assurer l'équité ?

La réponse dépend de la position dans laquelle on est : celle de l'emprunteur ou celle du banquier. Un emprunteur potentiel pouvant rembourser ne souhaite pas être placé dans la classe 1 et cherche ainsi maximiser l'équité de la *False Positive Rate*. Le banquier souhaite que les personnes dont le prêt a été refusé alors qu'elles peuvent rembourser soient équitablement réparties selon le sexe. Il cherche ainsi à maximiser l'équité de *False Discovery Rate*.

La dernière question nous montre que la définition d'équité adoptée est directement reliée à l'intérêt de celui qui fait ce choix d'équité. Nous traiterons les deux points de vue (celui du banquier et celui de l'emprunteur) dans la suite.

Nous allons maintenant calculer les deux métriques *False Positive Rate* et *False Discovery Rate* propres à notre modèle et notre base de données.

False Positive Rate Ratio Pour s'assurer de l'équité de la *False Positive Rate* on se concentre sur les "faux positifs" i.e. les personnes prédites comme incapable de rembourser sachant qu'elles l'étaient.

Le ratio étudié est défini de la façon suivante :

$$\frac{\mathbb{P}(\hat{Y} = 1 \mid D = \text{unprivileged}, Y = 0)}{\mathbb{P}(\hat{Y} = 1 \mid D = \text{privileged}, Y = 0)}$$

Le numérateur correspond à la probabilité d'être prédit comme incapable de rembourser alors qu'on en était capable et que l'on était une femme lorsque le dénominateur est la probabilité d'être prédit comme incapable de rembourser alors qu'on en était capable et que l'on était un homme. Une valeur de 1 pour ce ratio signifie qu'une femme capable de rembourser son prêt a autant de chance de voir son prêt refusé que si elle était un homme.

[7]:
```
fprr = fairness_metrics.false_positive_rate_ratio()
print(f'False Positive Rate Ratio : {fprr:.2f}')
```

```
False Positive Rate Ratio : 2.78
```

Pour notre modèle, on voit que le ratio est de 2,78. Ainsi une femme capable de rembourser sont prêt a 2,78 fois plus de chance de se faire refuser un prêt qu'un homme capable de rembourser. Notre modèle n'est pas équitable du tout.

False Discovery Rate Ratio De la même manière, on s'assure de l'équité de la *False Discovery Rate* en utilisant le ratio suivant qui correspond à la probabilité de pouvoir rembourser son prêt alors que celui-ci a été refusé et que l'on est une femme sur la probabilité de pouvoir rembourser son prêt alors que celui-ci a été refusé et que l'on est un homme.

$$\frac{\mathbb{P}(Y = 0 \mid D = \text{unprivileged}, \hat{Y} = 1)}{\mathbb{P}(Y = 0 \mid D = \text{privileged}, \hat{Y} = 1)}$$

Une valeur de 1 assure l'équité homme/femme sur les individus capables de rembourser un prêt qui a été refusé.

[8]:
```
fpdr = fairness_metrics.false_discovery_rate_ratio()
print(f'False Discovery Rate Ratio : {fpdr:.2f}')
```

```
False Discovery Rate Ratio : 1.16
```

Selon cette mesure, une femme dont le prêt a été refusé a 1,16 plus de chances de rembourser son prêt qu'un homme dont le prêt a été refusé. Le modèle est beaucoup plus équitable selon cette métrique. On voit à travers cet exemple que l'équité d'un modèle dépend fortement de la métrique utilisée et de l'intérêt de celui souhaitant mesurer cette équité.

Selon les deux métriques, le sexe de l'individu qui demande un prêt joue un rôle sur la possibilité que ce prêt soit accordé. Or, tout comme l'ethnicité ou la religion, il est interdit en France de discriminer selon cette variable. Nous allons maintenant explorer différentes méthodes pour rétablir l'équité.

3.2.3 Retrouver l'équité

Une première méthode naïve est de retirer la variable sensible du jeu de données et d'entraîner le modèle sans qu'il n'ait accès à cette variable. Voyons ce que cela donne sur notre base de données.

[9]:
```
# create a new model that does not take `sex` variable
col_nosex = [column for column in X.columns if column != ["sex"]]
X_nosex = X[col_nosex]

# class the variable between categorical and ordinal
cat_variables = [col for col in X_nosex.columns if credit[col].dtype==object]
ord_variables = [col for col in X_nosex.columns if credit[col].dtype==int]

preprocess_nosex = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(drop='first'), cat_variables),
        ('ord', StandardScaler(), ord_variables)
    ])

model_nosex = Pipeline(
        [
            ('prepro', preprocess_nosex),
            ('logreg', LogisticRegression())
        ]
)
```

```
# new model that train without `sex` variable
logreg_nosex = deepcopy(model_nosex.fit(X_train[col_nosex], y_train))

# new metrics
fairness_metrics_nosex, _ = model2fair(logreg_nosex, X_test, y_test, col_nosex)
```

[10]:
```
fprr = fairness_metrics_nosex.false_positive_rate_ratio()
print(f'False Positive Rate Ratio : {fprr:.2f}')
```

```
False Positive Rate Ratio : 2.78
```

[11]:
```
fpdr = fairness_metrics_nosex.false_discovery_rate_ratio()
print(f'False Discovery Rate Ratio : {fpdr:.2f}')
```

```
False Discovery Rate Ratio : 1.16
```

On remarque que les deux ratios n'ont quasiment pas été modifiés alors que la variable **sex** a été retirée de la base de données d'entraînement. Une explication à cela est que les autres variables contiennent les informations suffisantes pour détecter le sexe de l'individu. La situation maritale est une variable donnant beaucoup d'informations sur le sexe de l'individu. On voit à travers cet exemple que retirer la variable sensible ne suffit pas à retirer l'information de cette variable sensible. On doit ainsi utiliser des méthodes plus complexes qui sont proposées dans la libraire *AIF60*.

Présentation des trois grandes familles Les méthodes d'aborder les problèmes d'équité se découpe en trois familles qui correspondent à trois moments de la conception d'un modèle.

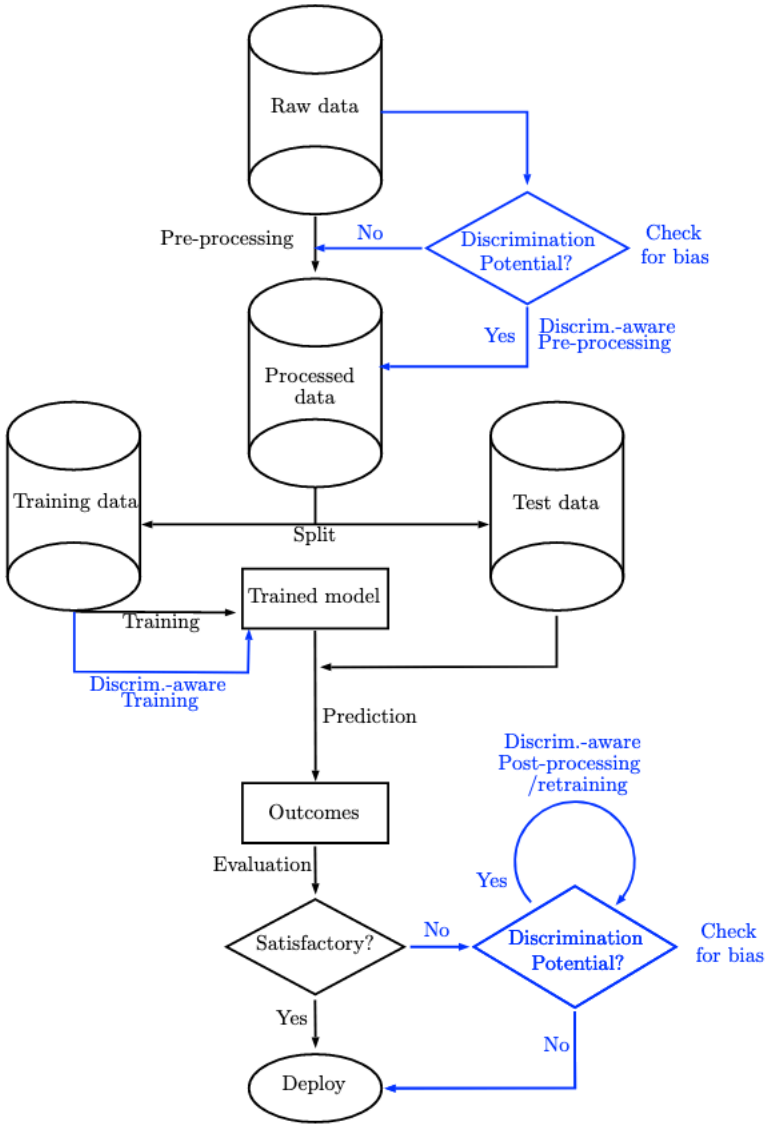


FIGURE 1 A general ML pipeline that shows opportunities for bias detection and mitigation in pre-, in-, and postprocessing stages (marked in blue).

Pre-Processing Les méthodes pre-processing sont des méthodes agnostic visent à traité les problèmes d'équité avant même que les données soit utilisées pour entraîner un modèle en transformant une ou plusieurs variables.

In-Processing Ces méthodes implémentent une ou plusieurs mesures directement dans le processus d'entraînement. Ces méthodes sont souvent assez généraliste dans leur énoncé mais leur implémentation dépend du modèle choisi.

Post-processing Ces méthodes sont comme les méthodes pre processing n'ont pas besoin d'avoir accès au modèle pour les mettre en place. Mais au lieu de se concentrer sur les données en amont, ces méthodes traitent les problèmes d'équité sur les prédictions du modèle.

Implémentation de la méthode

Reweight La méthode utilisée ci-dessous est une méthode de *pre-processing*. Le principe de cette méthode est le suivant. On affecte un poids à chaque entrée de la base de données pour s'assurer que lors de la phase d'entraînement on tire de façon équitable les individus parmis les quatres groupes suivant : les femmes capables de rembourser leur prêt, les hommes capables de rembourser leur prêt, les femmes incapables de rembourser leur prêt, les hommes incapables de rembourser leur prêt. Cette méthodes permets par exemple de corriger le fait qu'un base de données contiennent plus d'hommes que de femmes. Appliquons cette méthode et voyons comment sont affectés les ratios False Positive Rate et False Discovery Rate.

[12]:
```
def reweight_weight(y_train):
    rw = Reweighing(unprivileged_groups=[{'sex': 1}],
                    privileged_groups=[{'sex': 0}])

    sex_bin_df = pd.DataFrame(X_train.sex.map({'male': 0, 'female': 1}),
                              columns=['sex'])
    bld_train = BinaryLabelDataset(df=pd.concat([sex_bin_df, y_train], axis=1),
                                                    label_names=['default'],
                                                ␣
 ↪protected_attribute_names=['sex'])
    bld_train = rw.fit_transform(bld_train)

    return bld_train.instance_weights.ravel()
```

[13]:
```
w_train = reweight_weight(y_train)
logreg_rw = deepcopy(model.fit(X_train, y_train, logreg__sample_weight =␣
 ↪w_train))

fairness_metrics_rw, _ = model2fair(logreg_rw, X_test, y_test)
```

[14]:
```
fprr_rw = fairness_metrics_rw.false_positive_rate_ratio()
print(f'False Positive Rate Ratio : {fprr_rw:.2f}')
print(f'Gain : {(fprr - fprr_rw)/fprr:.2f}')
```

```
False Positive Rate Ratio : 1.88
Gain : 0.32
```

[15]:
```
fpdr_rw = fairness_metrics_rw.false_discovery_rate_ratio()
print(f'False Discovery Rate Ratio : {fpdr_rw:.2f}')
print(f'Gain : {(fpdr - fpdr_rw)/fpdr:.2f}')
```

```
False Discovery Rate Ratio : 1.04
Gain : 0.11
```

On observe une nette amélioration des ratios après implémentation de cette méthode. Tirer équitablement les individus a réduit les inégalités selon la *false positive rate* de 30% et les inégalités selon la *false discovery rate* de 10%.

Pour évaluer l'efficacité de la méthode, on crée des bases de données biaisées de deux façons différentes :

- la demande de prêt est facilitée pour les hommes (on change les 1 en 0 pour une fraction de la population masculine)
- la demande de prêt est rendue plus difficile pour les femmes (on change les 0 en 1 pour une fraction de la population féminine)

On observe ensuite la manière dont évolue la performance du modèle et les métriques d'équité avant et après l'application du *reweighting*.

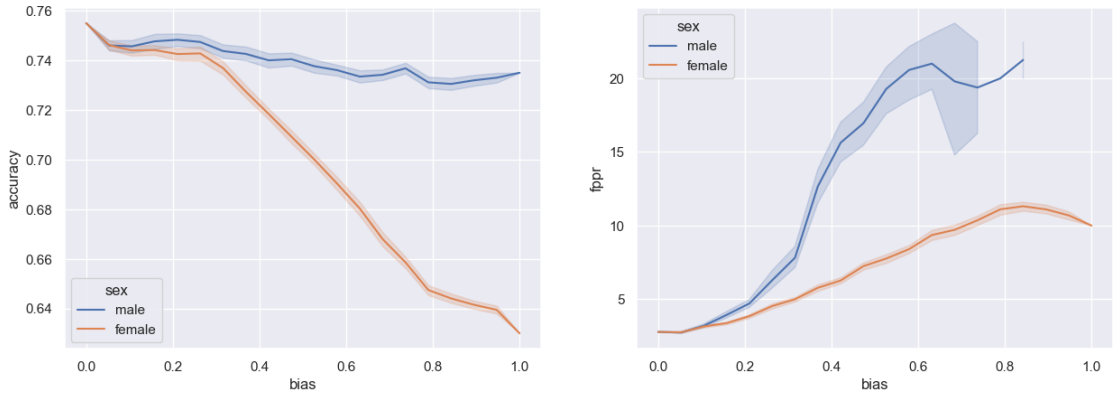
[16]:
```
def bias_train(frac, sex):
    if sex == 'male':
        label = 1
    else:
        label = 0

    y_train_subset = y_train[X_train.sex == sex]
    index_subset_label = y_train_subset[y_train_subset == label].index

    size=int(frac*len(index_subset_label))
    change_idx = np.random.choice(index_subset_label, size=size, replace=False)

    y_train_mod = deepcopy(y_train)
    y_train_mod[change_idx] = 1 - label

    return y_train_mod
```

[17]:
```
columns = ['sex', 'bias', 'fppr', 'fpdr', 'fppr_rw',
           'fpdr_rw', 'accuracy', 'accuracy_rw']
n_test = 100

if NEW:
    df_fair = pd.DataFrame(columns=columns)

    for sex in ['male', 'female']:
        for frac in tqdm(np.linspace(0, 1, 20)):
            for _ in range(n_test):
                y_train_mod = bias_train(frac, sex)

                logreg_bias = model.fit(X_train, y_train_mod)
                fairness_bias, _ = model2fair(logreg_bias, X_test, y_test)

                w_train_mod = reweight_weight(y_train_mod)
                logreg_bias_rw = model.fit(X_train, y_train_mod,
                                           logreg__sample_weight = w_train_mod)

                fairness_bias_rw, _ = model2fair(logreg_bias_rw, X_test, y_test)
```

```
                res = {'sex': sex,
                       'bias': frac,
                       'fppr': fairness_bias.false_positive_rate_ratio(),
                       'fpdr': fairness_bias.false_discovery_rate_ratio(),
                       'fppr_rw': fairness_bias_rw.false_positive_rate_ratio(),
                       'fpdr_rw': fairness_bias_rw.false_discovery_rate_ratio(),
                       'accuracy': fairness_bias.accuracy(),
                       'accuracy_rw': fairness_bias_rw.accuracy()}

                df_fair = df_fair.append(res, ignore_index=True)

                np.savez_compressed(f'data/fairness/res_reweight.npz',
                        df_fair=df_fair)
else:
    file = np.load('data/fairness/res_reweight.npz', allow_pickle=True)
    df_fair = pd.DataFrame(file['df_fair'], columns=columns)

df_fair['accuracy_diff'] = df_fair['accuracy'] - df_fair['accuracy_rw']
df_fair['fppr_gain'] = (df_fair['fppr'] - df_fair['fppr_rw']) / df_fair['fppr']
df_fair['fpdr_gain'] = (df_fair['fpdr'] - df_fair['fpdr_rw']) / df_fair['fpdr']
```

Sans *reweighting* On regarde dans un premier temps l'évolution des mesures d'accuracy et du ratio *False Positive Rate Ratio* en fonction de la fraction de la population (homme ou femme) que l'on a modifié pour discriminer les femmes, cela sans avoir implémenté la méthode de *reweighting*. On trace en orange l'évolution des mesures en fonction de la fraction des femmes dont la classification passe de 0 à 1 (un prêt remboursé devient non remboursé). Les courbes bleues correspondent à l'évolution des mesures en fonction de la fraction des hommes dont la classification passe de 1 à 0 (un prêt non remboursé devient remboursé).

[18]:
```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

sns.lineplot(x='bias', y='accuracy', data=df_fair, hue='sex', ax=axes[0])
sns.lineplot(x='bias', y='fppr', data=df_fair, hue='sex', ax=axes[1])

plt.show()
```



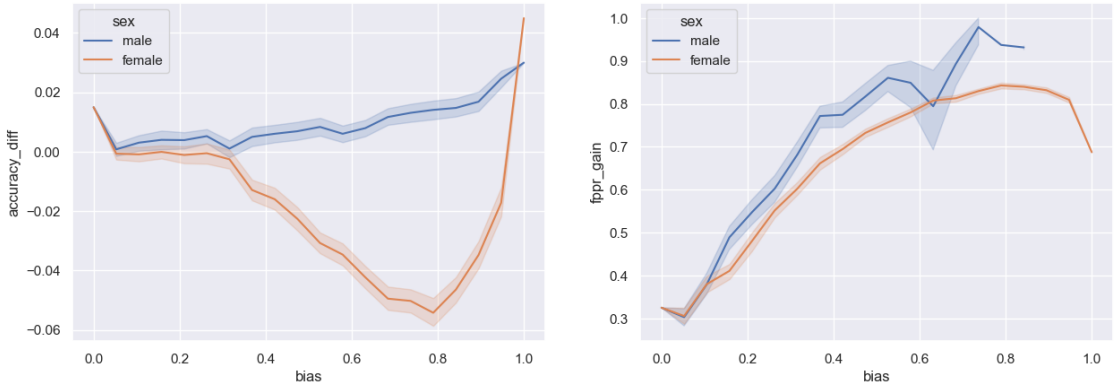
Le graphique de gauche ci-dessus représente l'évolution des performances du modèle en fonction du biais de la base de données sur laquelle il est entraîné. Il nous montre que la discrimination avantageant directement les hommes n'a que très peu d'impacte sur les performances du modèle par rapport à la discrimination désaventageant directement les femmes. Cela vient sûrement du déséquilibre existant entre le nombre de prêts remboursés et le nombre de prêts non remboursés dans la base de données d'entraînement (il y a beaucoup plus de prêts remboursés que de prêts non remboursés). Ainsi, modifier les sorties des femmes remboursant le prêt a un impact plus important que modifier les sorties des hommes ne remboursant pas.

Le graphique de droite ci-dessus représente l'évolution de la première métrique d'équité *False Positive Rate Ratio* en fonction du biais. On voit que ce ratio est très impacté par la discrimination qui avantage les hommes. On arrive même à un maximum où les femmes capables de rembourser ont plus de vingt fois plus de chance de se faire refuser qu'un homme capable de rembourser. De plus, ce graphique permet de vérifier le fait que le ratio *False Positive Rate Ratio* permet de quantifier la discrimisation dans une base de données. On voit grâce à ce graphique que les métriques d'équité sont utiles pour mesurer les inégalités intrinsèques à nos bases de données biaisées.

On peut maintenant observer les effets du *reweighting* sur les modèles en fonction du biais des bases d'entraînement.

Avec *reweighting* On implémente la méthode *reweighting* puis on compare l'accuracy et le ratio *False Positive Rate Ratio* avec ceux obtenus sans l'implémentation de la méthode *reweighting* afin d'observer l'efficacité de cette méthode.

[19]:
```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

sns.lineplot(x='bias', y='accuracy_diff', data=df_fair, hue='sex', ax=axes[0])
sns.lineplot(x='bias', y='fppr_gain', data=df_fair, hue='sex', ax=axes[1])

plt.show()
```



Le graphique de gauche représente la différence entre la performance du modèle avec *reweighting* et la performance du modèle sans *reweighting* toujours en fonction du biais (homme ou femme) appliqué à la base de données. Le graphique de droite représente le gain du ratio *False Positive Rate Ratio* grâce au *reweighting* (gain calculé en faisant la différence entre la le ratio du modèle avec *reweighting* et le ratio du modèle sans *reweighting*) en fonction du biais appliqué à la base de données.

Le graphique de gauche montre que si le biais n'est pas très important, le *reweighting* n'affecte pas les performances du modèle, cela pour les deux discriminations. Par contre, lorsque la discrimination est faîte sur les femmes directement et que le biais est important, on a une forte perte de performance induite par le *reweighting*. Cette perte s'ajoute au fait que les modèles s'entraînant sur des bases de données discriminant grandement les femmes ont déjà des performances faibles (cf graphique précédant). Ne pas avoir une base de données équitable peut donc avoir un impact important sur les performances du modèle même après l'application de méthodes d'équité.

Le graphique de droite montre l'évolution du gain d'équité après l'application de la méthode *reweighting*. On remarque que l'on a bien une augmentation du gain lorsque l'on augmente le biais et que le type de discrimination n'influence que très peu cette augmentation.

3.3 Critiques et conclusion

3.3.1 Les librairies d'équité ne sont pas assez matures

Contrairement au deux autres notions, les librairies d'équité sont plus nombreuses :

- *What-If-Tool* développé par Google depuis 2019
- *AI Fairness 360* développé par IBM depuis 2018
- *Fair-learn* développé par Microsoft depuis 2019
- *Aequitas* développé par *Center for Data Science and Public Policy* (Université de Chicago) depuis 2018

De plus, les librairies ne sont pas uniquement des libraires d'implémentation de méthodes.par exemple, *What-If-Tool* et *Aequitas* proposent des outils de visualisation des impacts de l'équité. Il n'est plus question uniquement de mettre en place des solutions mais également de sensibiliser sur les risques potentiels. Les méthodes pour mitiger ces risques ne sont cette fois-ci pas autant poussées sur les modèles d'apprentissage profonds et par exemple il n'existe pas de méthodes de

débiaisage pour les modèles de vision ou de langages, deux secteurs où le risque de discrimination de certains groupes pourraient avoir de grandes conséquences. Pour ce qui est des métriques, des librairies comme *AI Fairness 360* propose des outils très simples pour les calculer. Mais n'étant pas encore utiliser de manière importante les objets qui sont demandés d'utiliser pour faciliter le calcul des métriques ne sont pas très bien détaillés. Les échanges sur les sites comme *stack overflow* montre le manque de retour, on peut voir qu'il y a des dizaines de questions sur l'outil de Google mais encore très peu pour les autres libraires. Il est nécessaire de continuer de simplifier les outils et de créer des visualisations des enjeux. Les librairies d'équités peuvent ainsi être un exemple pour les librairies des autres enjeux car elles sont plus matures. Contrairement aux deux autres, de nouvelles méthodes pour des modèles d'apprentissage profond attendent d'être développées. On peut également attendre un plus grand nombre d'échanges sur les sites d'entraide, échanges qui se feront lorsque le nombre d'utilisateurs de tels outils augmentera. Pour en arriver là il est important de démocratiser et de normaliser des outils comme *What-If-Tool* pour que son utilisation ne soit pas cantonné à des modèles implémentés avec la librairie d'apprentissage de Google *Tensor Flow*.

3.3.2 Les librairies pour l'équité ne sont pas assez adaptées aux industriels

Grâce aux outils de visualisation et le nombre important de méthodes développées pour des modèles plus simples, les librairies d'équité sont les librairies les plus adaptées aux industriels en comparaison à celles de privacy et robustness. Néanmoins le nombre de métriques et de méthodes de débiaisage différentes rend difficile le choix de celle la plus pertinente. Comme on ne peut pas optimiser tous les critères d'équité il est très important de choisir la métrique permettant l'optimisation des critères que l'on souhaite prioriser, cette métrique dépendant du problème étudié. Pour aider à ce choix, des documents comme l'arbre ci-dessous existent. Malgré tout, cela reste une version simplifiée des choix possibles qui nécessite de la part du développeur de connaître très précisément le problème étudié pour le ranger dans l'une des classes proposées par l'arbre. Ce choix de classe est une tâche compliquée pour ce dernier.
Il est aussi important de souligner que la majorité des méthodes implémentées se base sur l'existence de deux choses: une variable sensible accessible et des groupes avantagés/discriminés. Or l'existence de cette variable n'est pas une hypothèse tenable car, de plus en plus, les réglementations comme la RGPD empêchent le stockage de la plupart des variables que l'on pourrait considérer comme sensibles. Par ailleurs, ne pas connaitre le sexe ou l'ethinicté d'une personne ne veut pas dire que le modèle ne sera pas disciminant et malgré le retrait des variables dites sensibles, le modèle peut rester inégalitaire. Il est donc nécessaire de trouver une solution à la mesure de l'équité tout en respectant les données des utilisateurs. Afin de rendre l'utilisation de ces méthodes encore plus populaire, ces dernières doivent faire partie des outils de autoML, les implémentations se faisant in-processing (durant l'entraînement). Continuer à produire des documents de communication permettrait également de sensibiliser les utilisateurs car la compréhension des métriques d'équité est fondamentale pour comprendre les enjeux d'un modèle.

3.3.3 Il existe un arbitrage entre performance et équité

Peu importe le moment où on implémente les méthodes d'équité (pre, in, post-processing) celles-ci auront nécessairement un impact sur les performances du modèle. L'implémentation des différentes méthodes sur l'exemple nous le montre avec une biasse de l'*accuracy* lorsque l'on rajoute une méthode. Il existe donc un compromis entre la précision du modèle et son équité. De même, tout ajout de méthodes implique des temps de calcul plus long pour la majorité des méthodes existantes. Mais ces compromis ne sont pas uniquement entre deux notions distinctes, comme nous le montre

[1], il existe un compromis intrinsèque à la notion d'équité que l'on cherche à obtenir. Pour illustrer ce dernier point, nous allons prendre l'exemple du papier ci-dessus : en cherchant un modèle pour déterminer la probabilité d'un individu de contracter une maladie. Le papier démontre que, pour la base de donnée considérée, l'une de ces trois propriétés sera vraie pour n'importe quel modèle utilisé :

- les estimations de probabilité du test sont systématiquement faussées vers le haut ou vers le bas pour au moins un sexe
- le test attribue une estimation du risque moyen plus élevée aux personnes en bonne santé (non porteuses) d'un sexe que de l'autre
- le test attribue une estimation du risque moyen plus élevée aux porteurs de la maladie d'un sexe par rapport à l'autre.

Comprendre les arbitrages afin de faire les choix judicieux est donc très important pour une bonne implémentation des modèles. On voit ainsi le besoin d'exemples comme celui ci-dessus dans des secteurs diversifiés et la nécessité de supports pour aider les industries à comprendre les enjeux et les arbitrages nécessaires.

4 Outils pour la protection des données personnelles

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import diffprivlib.models as diff

from tqdm import tqdm

from copy import deepcopy
from itertools import combinations, product
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import FunctionTransformer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import confusion_matrix, classification_report

import utility as uti

import seaborn as sns
sns.set_theme(style='darkgrid')

plt.rcParams['figure.dpi'] = 100
```

[2]:
```
%%capture
%run ./Introduction.ipynb
```

[3]:
```
seed = 1234
NEW = False
```

4.1 Présentation des enjeux de la protection des données personnelles

Avec la part grandissante de services dépendant de méthodes d'apprentissage statistique, il devient indispensable de se poser la question de la protection des données utilisées par ces méthodes. Les deux raisons suivantes sont responsables du fait que les modèles statistiques sont considérés comme des actifs. La première raison est qu'il est coûteux de concevoir un modèle soi-même. Un modèle de Google composé de 340 millions de paramètres coûte plus de soixante mille dollars à entraîner. Il faut également prendre en compte le temps nécessaire pour choisir le modèle et trouver la bonne architecture ainsi que les bonnes méthodes d'implémentation. De plus, l'efficacité d'un modèle statistique repose grandement sur la qualité des données qu'il reçoit en entrée. Trouver, nettoyer et annoter une base de données est une autre des tâches coûteuses liée à la création d'un modèle. La deuxième raison concerne les enjeux de sécurité et de confidentialité liés au modèle. La base de données peut contenir des informations sensibles ou encore l'utilisateur du modèle peut vouloir s'assurer de l'utilisation des informations qu'il fournit.

La protection des données dans la sphère de l'apprentissage statistique correspond à des méthodes pour se prémunir contre une utilisation adverse des outils d'apprentissage dans le but de soutirer des informations personnelles ou confidentielles. Le but est de s'assurer qu'un modèle ne divulgue pas plus d'informations que celles pour lesquelles il a été conçu. Les différentes attaques de protection des données sont les suivantes :

- Inférence d'appartenance : cette attaque cherche à déterminer si une entrée particulière fait partie de la base d'entraînement du modèle cible.
- Inférence de propriété : cette attaque cherche à inférer des propriétés de la base de données d'entraînement du modèle cible.
- Copie de modèle : cette attaque cherche à récupérer des informations plus ou moins précises sur un modèle utilisé en boîte noire. Lors de l'utilisation d'un modèle en boîte noire, les éléments constituant le modèle ne sont pas accessibles à l'utilisateur, en particulier les paramètres du modèle ne sont pas connus de l'utilisateur.

Les deux méthodes principales de défense sont :

- la cryptographie : cette défense consiste à crypter les données utilisées par le modèle qui prend alors en entrée des données cryptées. L'utilisateur crypte ainsi ses données personnelles avant de les envoyer à un modèle détenu par une personne extérieure.
- la confidentialité différentiable : cette méthode englobe différents outils permettant d'obtenir des garanties mathématiques sur la protection des données contre différentes attaques.

Nous traitons uniquement la deuxième méthode dans cet article. Les attaques mentionnées ci-dessus peuvent avoir un impact important sur les entreprises proposant des modèles d'apprentissage statistiques. L'inférence d'appartenance et de propriétés sont des attaques sur les bases de données qui impliquent soit des fuites d'information importantes soit un non-respect du traitement des données personnelles. Les copies de modèle peuvent quant à elles permettre à des concurrents de réduire leur écart technologique sans avoir à supporter les coûts associés au développement du modèle. Il est donc important de comprendre ces attaques et leurs limites mais également de connaître les différentes méthodes pour se prémunir contre celles-ci.

Dans cette étude de cas, nous implémentons deux attaques et une défense sur la même base de données utilisée jusqu'à présent. La première attaque est une copie de modèle lorsque la deuxième est une inférence de propriété qui consiste à retrouver la proportion de femme dans la base d'entraînement d'un modèle. On implémente alors une méthode de confidentialité différentiable pour contrer cette deuxième attaque.

4.2 Étude de cas et implémentation

4.2.1 Model extraction

Prenons le cas de figure suivant: un nouvel acteur bancaire veut se lancer dans les prêts mais plutôt que de payer une équipe pour labelliser les demandes il souhaite utiliser un modèle de la banque allemande.

Le but de cette attaque est de récupérer les paramètres du modèle afin d'en avoir une copie. On suppose les deux choses suivantes pour que l'attaque soit possible:

- on connaît l'architecture du modèle : régression logistique.
- on peut utiliser le modèle cible sans contraintes (mais comme une boîte noire)

Afin de récupérer les paramètres du modèle cible, on commence par créer un modèle ayant la même architecture que le modèle cible. On va ensuite constituer une base de données à l'aide du modèle cible. Pour cela, on génère aléatoirement des entrées selon deux méthodes que l'on comparera ensuite et on labellise ces données en utilisant les sorties retournées par le modèle cible sur ces entrées.

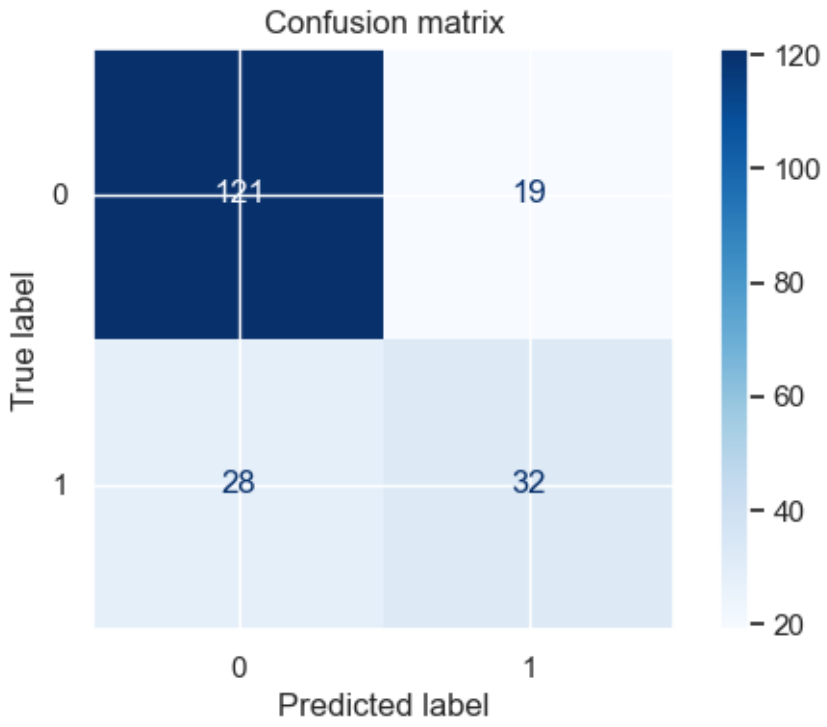
Le nouveau modèle est ensuite entraîné avec la nouvelle base de données ainsi constituée. Dans ce qui suit, le modèle `logreg` correspond au modèle secret qui a été entraîné sur la base de données de départ. C'est ce modèle que nous tentons de copier.

[4]:
```
model_pirate = deepcopy(model)
model_secret = deepcopy(logreg)
```

Pour commencer, supposons que l'attaquant ait accès à une base de données d'entrées tirées selon la même distribution que les entrées utilisées pour entraîner le modèle secret. Puisque l'on a accès en boîte noire au modèle secret, on peut obtenir les sorties prédites par le modèle secret sur ces entrées et constituer notre propre base de données d'entraînement. On peut alors entraîner notre modèle sur cette base d'entraînement et regarder si notre modèle ressemble au modèle secret.

[5]:
```
def new_entries(n):
    '''Retrun n new entries based on possible values and frequency of X␣
 ↪dataset'''
    list_entries = []
    for col in X.columns:
        frequence = (X[col].value_counts(normalize=True)).to_numpy()
        entries = np.random.choice(X[col].unique(), n, p=frequence)
        list_entries.append(entries)

    return pd.DataFrame(np.array(list_entries).transpose(), columns=X.columns)
```

[6]:
```
n = len(X_train)
X_train_pirate = new_entries(n)
y_train_pirate = model_secret.predict(X_train_pirate)

np.random.seed(seed)
model_pirate.fit(X_train_pirate, y_train_pirate)
uti.mesure_clas_list([model_pirate, model_secret], X_test, y_test, ['pirate',␣
 ↪'secret'])
```



	Sensitivity	Specificity	Precision	Accuracy
pirate	0.86	0.53	0.63	0.77
secret	0.86	0.50	0.61	0.76

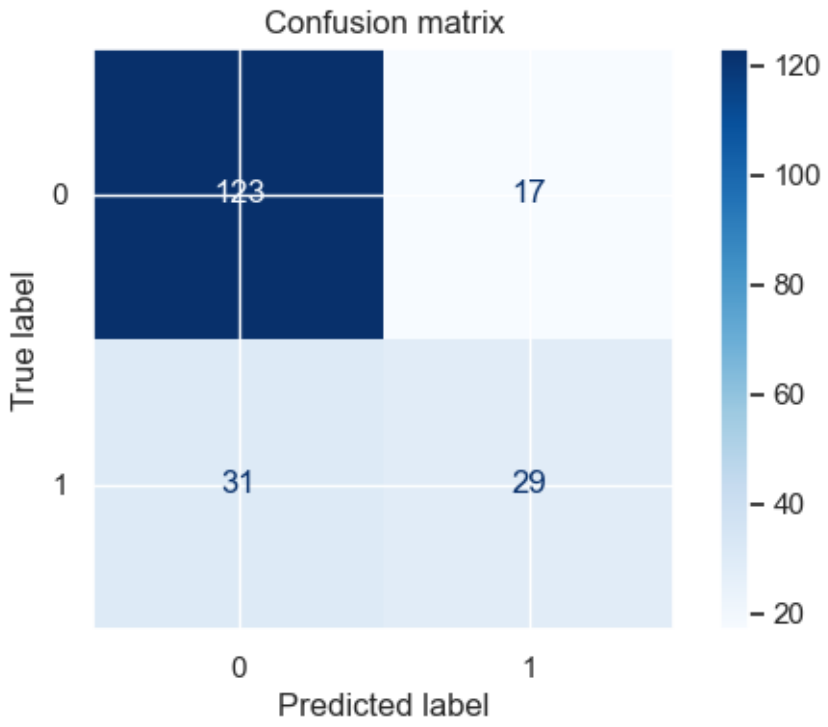
On remarque que le modèle extrait a des performances sur l'ensemble de test qui sont proches des performances du modèle cible. Toutefois, on remarque qu'il a tendance à l'augmenter le nombre de personnes prédites comme capable de rembourser alors qu'elles ne le sont pas.

Nous retirons maintenant l'hypothèse d'avoir accès à une base de données similaire à celle utilisée pour l'entraînement du modèle secret. On va créer notre base d'entraînement en tirant aléatoirement les variables des entrées et en utilisant le modèle secret en boîte noire pour prédire la sortie associée à chaque entrée créée.

[7]:
```
def new_entries_unif(n):
    '''Retrun n new entries based on possible values uniformaly'''
    list_entries = []
    for col in X.columns:
        entries = np.random.choice(X[col].unique(), n)
        list_entries.append(entries)

    return pd.DataFrame(np.array(list_entries).transpose(), columns=X.columns)
```

[8]:
```
n = len(X_train)
```

```
X_train_pirate_unif = new_entries_unif(n)
y_train_pirate_unif = model_secret.predict(X_train_pirate_unif)

np.random.seed(seed)
model_pirate.fit(X_train_pirate_unif, y_train_pirate_unif)
uti.mesure_clas_list([model_pirate, model_secret], X_test, y_test, ['pirate',␣
↪'secret'])
```



	Sensitivity	Specificity	Precision	Accuracy
pirate	0.88	0.48	0.63	0.76
secret	0.86	0.50	0.61	0.76

Le nouveau modèle appris a des performances semblables au modèle secret une fois encore. Jusqu'ici, pour comparer deux modèles nous avons comparé les métriques de performance de ces derniers, c'est la méthode la plus classique. Une deuxième façon de quantifier la distance entre deux modèles et de regarder leurs comportements respectifs sur des entrées identiques afin de voir si les modèles se trompent sur les mêmes entrées. La première métrique (celle que nous avons utilisée) est globale tandis que la deuxième est une métrique locale. On note `precision` la mesure locale décrite ci-dessus et on utilise la métrique de performance "accuracy" pour avoir une mesure globale.

[9]:
```
y_pirate = model_pirate.predict(X_test)
y_secret = model_secret.predict(X_test)
```

```
# Pourcentage of same prediction
precision = (y_secret == y_pirate).astype(int).sum()/len(y_secret)

# Difference of accuracy betwenne secret and pirate prediction
accu_diff = accuracy_score(y_test, y_secret) - accuracy_score(y_test, y_pirate)

print(f"local : {precision:.3f}, global : {accu_diff:.3f}")
```

```
local : 0.945, global : -0.005
```

Les résultats obtenus montrent que l'on a créé un modèle qui est très proche globalement et localement du modèle secret. L'évasion de modèle a ainsi très bien fonctionné.

4.2.2 Créer un modèle qui surapprend

Afin de réaliser une attaque d'inférence de propriété, il est nécessaire d'attaquer un modèle ayant surappris. L'objectif de l'attaque est de récupérer des informations (des propriétés) de la base de données d'entraînement du modèle cible. Pour cela, on va exploiter le surapprentissage et tenter d'exploiter ce que le modèle a appris sur la base de données d'entraînement. Dans un premier temps, nous créons un modèle ayant surappris sur notre base de données. Il est difficile de faire surapprendre une régression logistique qui est un modèle simple en prolongeant seulement l'entraînement. Une façon de faciliter le surapprentissage de cette dernière est de coupler les variables de la façon suivante.

On suppose qu'une entrée dans la base de données d'origine est de la forme :

$$x_1, x_2, ..., x_n$$

avec x_i la variable de l'individu pour la i-ème catégorie.

On crée avec cette entrée la nouvelle entrée suivante pour le modèle que l'on souhaite faire surapprendre :

$$x_1, x_2, ..., x_n, (x_i x_j)_{i,j \in C}$$

avec C l'ensemble des variables catégorielles. On augmente ainsi grandement la dimension de notre modèle ce qui facilitera le surapprentissage. Nous créons dans un premier temps la base de données augmentée à partir de notre base de donnée initiale.

[10]:
```
# We need to compute evey possibility of variables to normalise coefficient␣
 ↪order
# in the regression as a lot of possibility will not be present every time
all_variables = cat_variables + [f'{i}_{j}' for i, j in
                                                 combinations(cat_variables, 2)]
all_values = [X[col].unique() for col in cat_variables]
all_combinaisons = list(combinations(all_values, 2))

categorie_values = [[a + b for a,b in list(product(*x))] for x in␣
 ↪all_combinaisons]

def add_relations(df_orig):
    '''Takes a dataframe and add the relation define above'''
```

```
    df = df_orig.copy()
    for i, j in combinations(cat_variables, 2):
        df[f'{i}_{j}'] = df[i] + df[j]
    return df
```

On crée ensuite le modèle prenant en entrée les entrées de la base de donnée augmentée et on l'entraîne sur cette dernière. Le modèle est nommé `model_overfit`.

[11]:
```
preprocessor_comb = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(categories=all_values+categorie_values,
                              drop='first',
                              sparse=False), all_variables),
        ('ord', StandardScaler(), ord_variables)
    ])

model_overfit = Pipeline(
    steps=[
        ('relations', FunctionTransformer(add_relations)),
        ('preprocessor', preprocessor_comb),
        ('logreg', LogisticRegression(max_iter=200))
    ]
)
```

Afin de vérifier si notre modèle a bien surappris, on peut regarder ses performances sur l'ensemble d'entraînement et l'ensemble de test. Un modèle qui surapprend aura d'excellentes performances sur l'ensemble d'entraînement et des performances moins bonnes sur l'ensemble de test. En effet,un model qui a surappris va connaître parfaitement les données de la base d'entraînement mais généralisera très mal sur les données de test. [IMAGE ?] On peut également comparer ce modèle avec notre modèle initial pour observer l'évolution.

[12]:
```
#%timeit model.fit(X_train, y_train)
#print(f"accuracy test : {model.score(X_test, y_test):.3f}, ", end='')
#print(f"accuracy train : {model.score(X_train, y_train):.3f}")

print("""22.6 ms ± 1.21 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
accuracy test : 0.755, accuracy train : 0.785""")
```

```
22.6 ms ± 1.21 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
accuracy test : 0.755, accuracy train : 0.785
```

[13]:
```
#model_overfit.fit(X_train, y_train)
#print(f"accuracy test : {model_over.score(X_test, y_test):.3f}, ", end='')
#print(f"accuracy train : {model_over.score(X_train, y_train):.3f}")

print("""212 ms ± 13.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
accuray test : 0.725, accuracy train : 0.958""")
```

```
212 ms ± 13.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
accuray test : 0.725, accuracy train : 0.958
```

Si notre modèle initial a des scores similaires sur l'ensemble de test et l'ensemble d'entraînement, le modèle `model_overfit` a de meilleurs scores sur la base d'entraînement et de moins bons scores sur la base de test, il a bien surappris. Ce surapprentissage implique que des informations concernant la base d'entraînement ont été apprises par le modèle et sont accessibles par quelqu'un qui attaquerait ce modèle surentraîné comme nous l'illustrons dans la partie suivante.

4.2.3 Proprety inference

On se place dans le cas particulier suivant: la banque allemande ne rend pas publique la proportion de femmes utilisées pour entraîner leur modèle. Un groupe d'individus pense que cette banque se trompe plus régulièrement sur les femmes que sur les hommes et veut prouver que celle-ci n'a pas utilisé un jeu de données équilibré homme/femme pour entraîné leur modèle.

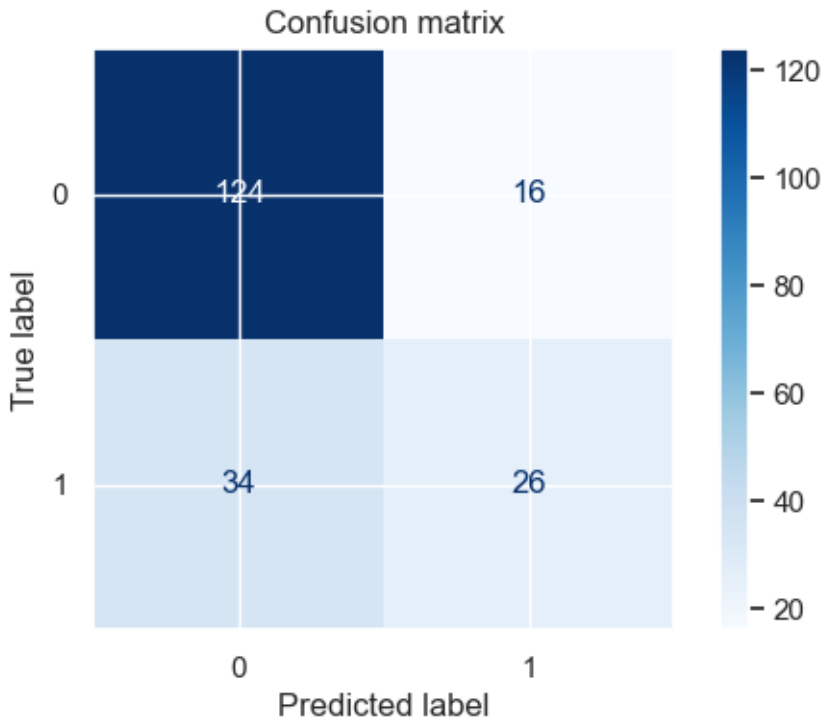
L'objectif de l'attaque est ainsi de récupérer la proportion de femmes dans la base de données d'entraînement en ayant simplement accès en boîte noire au modèle de la banque.

Afin de simuler le modèle de la banque allemande, on crée un modèle entraîné sur une base de données composée d'une faible proportion de femmes. Ce modèle sera le modèle secret dont on cherchera à déterminer la proportion de femme utilisée dans la base d'entraînement.

On commence pour cela par créer une base de données biaisée avec une faible proportion de femmes, puis on entraîne un nouveau modèle sur cette base de données biaisée.

[14]:
```python
def gen_entries_woman(n, frac_woman=False):
    '''Creates a database with the same distribution as the original for every
       variables except for the sex variable which distribution is set to␣
 ↪frac'''

    list_entries = []
    for col in X.columns:
        if col == 'sex':
            entries = np.random.choice(['female', 'male'], n,
                                       p=[frac_woman, 1-frac_woman])
        else:
            distribution = (X[col].value_counts(normalize=True)).to_numpy()
            entries = np.random.choice(X[col].unique(), n, p=distribution)
        list_entries.append(entries)

    return pd.DataFrame(np.array(list_entries).transpose(), columns=X.columns)
```

[15]:
```python
woman_frac = 0.3

X_bias = gen_entries_woman(len(X_train), woman_frac)
y_bias = logreg.predict(X_bias)

#np.random.seed(seed)
model_secret = deepcopy(model_overfit.fit(X_bias, y_bias))
uti.mesure_clas_list([model_secret, logreg], X_test, y_test, ['bais', 'base'])
```



	Sensitivity	Specificity	Precision	Accuracy
bais	0.89	0.43	0.62	0.75
base	0.86	0.50	0.61	0.76

Nous venons de créer notre modèle secret qui a été entraîné avec une proportion de femmes de 30 %. On veut maintenant savoir si nous sommes capables de retrouver cette proportion en ayant accès à ce modèle en boîte noire uniquement.

Pour procéder à l'inférence de propriété on propose la méthode suivante basée sur [lien ?]. Dans un premier temps, on cherche à déterminer si la base de données d'entraînement de `model_secret` contient plus d'hommes ou plus de femmes.

On commence par générer 150 bases de données dont la fraction de femmes suit une loi normale centrée en 0.4 et d'écart type 0.05. Dans ces bases de données, il y a plus d'hommes que de femmes (en moyenne 60% d'hommes). On génère ensuite 150 bases de données dont la fraction de femmes suit une loi normale centrée en 0.6 et d'écart type 0.05. Dans ces bases de données, il y a plus de femmes que de d'hommes (en moyenne 40% d'hommes).

Ensuite pour chaque base de donnée créée, on crée un modèle (de régression logistique) que l'on entraîne sur la base de données correspondante (soit 300 modèles différents en tout).

On crée maintenant un méta modèle `meta_model` qui est une régression logistique qui prend en entrée les coefficients d'un modèle et le place dans la classe 0 ou la classe 1. La classe 0 correspond à un modèle s'étant entraîné sur une base de données ayant moins de filles et la classe 1 à un modèle s'étant entraîné sur une base de données ayant plus de filles.

On entraîne ainsi notre méta modèle sur les 300 modèles précédemment créés.

On donne enfin le modèle `model_secret` à notre méta modèle et on observe la sortie de celui-ci. Si ce dernier le place dans la classe 0 alors il suppose que le modèle secret s'est entraîné avec moins d'exemples de femmes et s'il le place dans la classe 1 alors il suppose que le modèle secret s'est entraîné avec plus d'exemples de femmes.

L'implémentation de ces étapes est faites dans les lignes de codes suivantes.

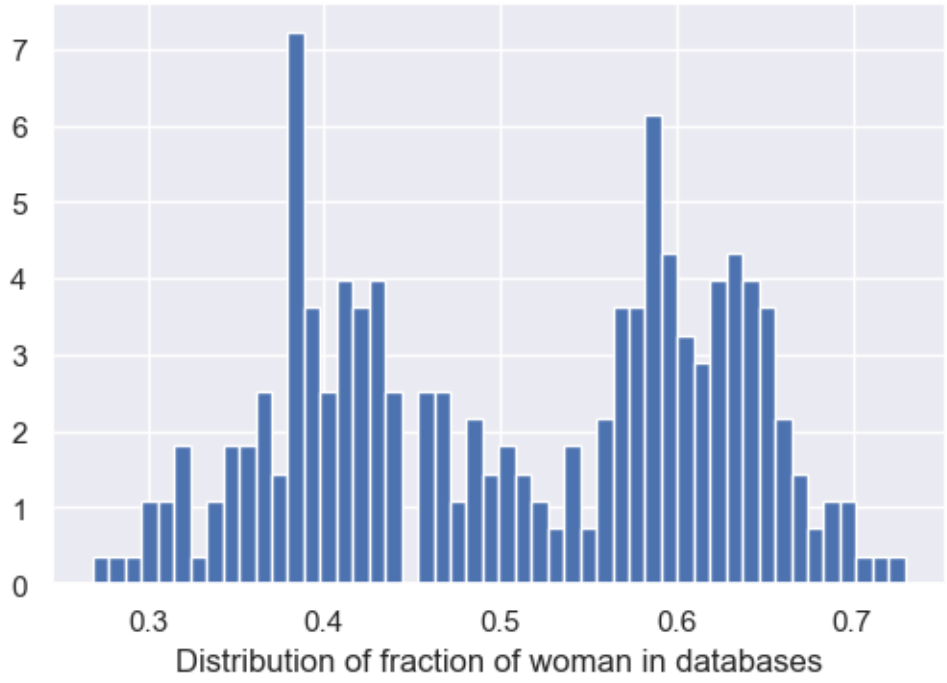
[16]:
```python
def gen_models(architecture, n_models, n_entries, unif=False):

    np.random.seed(seed)
    dist_woman = []

    for _ in range(n_models):
        mean = np.random.choice([0.4, 0.6])
        dist = np.clip(np.random.normal(mean, scale=0.05), 0, 1)
        dist_woman.append(dist)
    dist_woman = np.array(dist_woman)

    list_X = [gen_entries_woman(n_entries, dist) for dist in tqdm(dist_woman)]

    flatten_list = pd.DataFrame(np.array(list_X).reshape(-1, 21),
                                columns=X.columns)
    list_y = logreg.predict(flatten_list).reshape(n_models, -1)

    list_models = [deepcopy(model_archi.fit(X, y))
                                        for X, y in tqdm(zip(list_X, list_y))]

    return list_models, dist_woman
```

[17]:
```python
# set the number of database and the number of entries in each
n_models = 300
n_entries = len(X_train)

# chose if the distribution of all variables except 'sex' is choosen with
# the distribution of the secret database or uniformaly
unif = False

# model architechture : model, model_overfit
model_archi = model_overfit

# names for the file
name = '_overfit' if model_archi == model_overfit else ''
dist_name = '_unif' if unif else '_prop'


if NEW:
    list_models, dist_woman = gen_models(model_archi, n_models, n_entries, unif)
    meta_X = [model['logreg'].coef_[0] for model in list_models]
```

```
    np.savez_compressed(f'data/privacy/data_{name}{dist_name}.npz',
                        meta_X=meta_X,
                        dist_woman=dist_woman)
else:
    file = np.load(f'data/privacy/data_{name}{dist_name}.npz',␣
 ↪allow_pickle=True)
    meta_X = file['meta_X']
    dist_woman = file['dist_woman']

meta_y = (dist_woman > 0.5).astype('int')
```

Visualisation de la répartition de la proportion d'hommes dans les bases de données

[18]:
```
plt.hist(dist_woman, density=True, bins=50)
plt.xlabel('Distribution of fraction of woman in databases')
plt.show()
```



On peut visualiser sur ce graphique que la proportion de femmes dans les bases de données créées sont bien distribuées autour de 0.4 et de 0.6. On a ensuite créé un modèle par base de données et l'avons entraîné sur celle-ci. On va maintenant entraîner notre méta modèle à l'aide des 300 modèles générés.

Entraînement du méta-modèle

[19]:
```
split = model_selection.train_test_split(meta_X,
                                         meta_y,
                                         test_size=0.20)
X_train_meta, X_test_meta, y_train_meta, y_test_meta = split

meta_model = LogisticRegression(max_iter=200)

np.random.seed(seed)
meta_model.fit(X_train_meta, y_train_meta)

pred = meta_model.predict([model_secret['logreg'].coef_[0]])[0]

print("(0 : majority of man, 1 : majority of woman)")
print(f"The model predictied : {pred}")
```

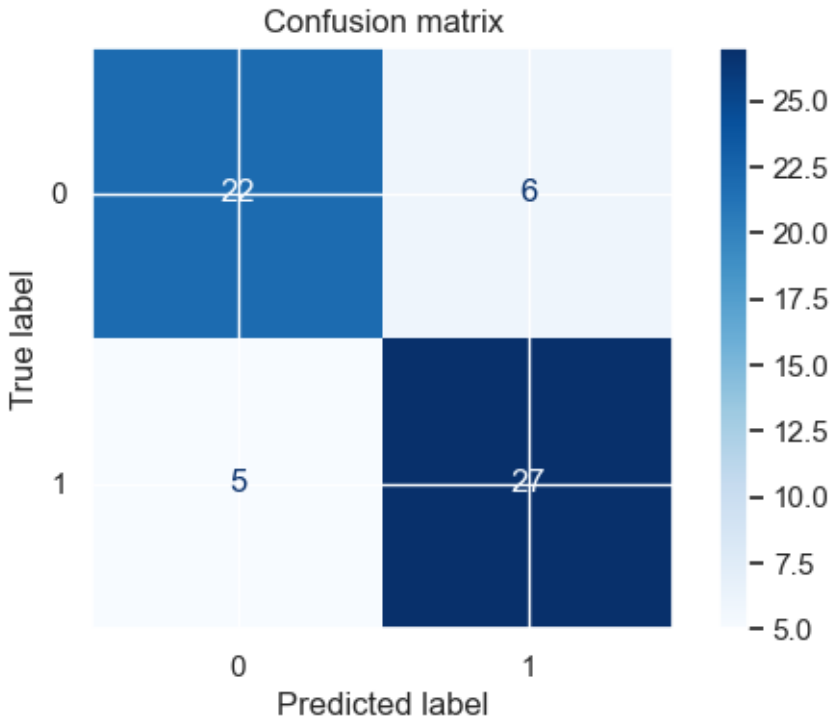
```
(0 : majority of man, 1 : majority of woman)
The model predictied : 0
```

On rappelle que notre méta modèle qui prend en entrée les coefficient d'un modèle et prédit si ce modèle a été entraîné sur une base avec une majorité de femmes ou non.

Le modèle secret a correctement prédit le fait que la base de données sur laquelle on a entraîné notre modèle est constituée de plus d'hommes que de femmes.

Pour mesurer la qualité de la méthode que l'on vient d'implémenter, on utilise notre base de test pour le méta modèle.

[20]:
```
uti.mesure_clas(meta_model, X_test_meta, y_test_meta)
```



```
Sensitivity  Specificity   Precision     Accuracy
    0.79         0.84         0.82         0.82
```

Les métriques de performance montrent que le méta modèle prédit de façon juste si un modèle s'est entraîné sur une base de données constituée plus d'hommes que de femmes. Nous allons passer à un niveau au-dessus et tenter de prédire la proportion de femmes présente dans la base d'entraînement utilisée pour entraîner un modèle. Pour se faire, nous allons changer notre méta modèle et utiliser une régression linéaire plutôt qu'une régression logistique. La variable cible n'est plus binaire mais continue et correspond à la fraction de femme dans la base d'entraînement utilisée pour entraîner le modèle d'entrée.

Régression linéaire pour trouver la proportion de femmes Pour mesurer la performance de notre méta modèle linéaire on fait le calcul suivant pour les éléments de la base de test:

$$|y_{true} - y_{pred}| * 100$$

avec y_{true} la fraction de femme de la base d'entraînement, y_{pred} la prédiction du méta modèle. On moyenne ensuite sur l'ensemble des éléments de la base de test pour avoir le pourcentage d'erreurs moyen.

[21]:
```
split = model_selection.train_test_split(meta_X,
                                         dist_woman,
                                         test_size=0.20,
                                         random_state=seed)
X_train_meta, X_test_meta, y_train_meta, y_test_meta = split

meta_model_linear = LinearRegression()
np.random.seed(seed)
meta_model_linear.fit(X_train_meta, y_train_meta)

y_pred_meta = meta_model_linear.predict(X_test_meta)
err = (np.abs(y_test_meta - y_pred_meta) * 100).mean()

print(f'Mean pourcentage error: {err:.2f}')
```

```
Mean pourcentage error: 7.61
```

Les métriques montrent que l'on a un méta modèle capable de retrouver assez précisément la fraction de femmes utilisées pour entraîner un modèle avec une erreur moyenne de %. Cette erreur est faible si l'on garde en tête le fait que l'on a seulement utilisé les coefficients du modèle pour retrouver cette information qui est propre à la base de données.

Regardons maintenant la prédiction du méta modèle sur le modèle secret.

[22]:
```
predict = meta_model_linear.predict(model_secret['logreg'].coef_)
true = woman_frac

print(f'Real       : {true}')
```

```
print(f'Predictied : {predict[0]:.2f}')
```

```
Real       : 0.3
Predictied : 0.44
```

On a une prédiction qui est proche de la proportion de femme présente dans la base de données d'entraînement. On est ainsi satisfait de notre attaque.

Cette partie montre que lorsque les données ne sont pas protégées, la récupération de modèle ou l'inférence de propriété sont possibles. Les méthodes de défense contre la copie de modèles sont majoritairement des méthodes non techniques comme la limitation d'utilisation ou une implémentation en API qui permet de contrôler plus précisément les demandes. Pour contrer l'inférence de propriétés, il existe des méthodes que l'on peut implémenter lors de l'entraînement pour compliquer les attaques.

4.2.4 Mitiger les dégâts

La méthode la plus populaire pour mitiger les attaques d'inférence est la confidentialité différentielle (*differental privacy*). L'idée fondamentale de cette méthode est l'introduction d'aléatoire. On introduit à un moment de la conception d'un modèle (soit dans les données, soit lors de l'entraînement) un bruit qui soit indépendant des données du modèle. Le but est de conserver une sortie proche de la sortie du modèle sans bruit tout en s'assurant que l'on ne puisse pas soutirer des informations sur les données du modèle à partir de la sortie du modèle bruité.

L'implémentation de cette méthode dépend du modèle statistique choisi. Si la confidentialité différentielle pour une régression logistique et un réseau de neurones n'est pas la même, de façon générale la confidentialité différentielle dépend principalement d'un paramètre : ϵ appelé le *budget de confidentialité*. La confidentialité différentiable autorise la fuite d'une certaine quantité d'information, quantité contrôlée par le paramètre ϵ. Plus la valeur de ce paramètre est faible, plus le modèle sera protégé.

Il existe plusieurs librairies implémentant des outils de confidentialité différentielles. Ces dernières sont listées dans la dernière partie. Dans notre cas, on utilise la librairie *diffprivlib* (IBM) qui permet d'implémenter ces méthodes avec des modèles scikit-learn. L'implémentation consiste à remplacer le modèle de régression logistique dans notre modèle d'origine par la régression logistique de la librairie *diffprivlib*.

Pour implémenter cette méthode, on a besoin de fixer deux paramètres :

- ϵ : le budget de confidentialité
- δ : une borne de régularisation

On ne s'occupera pas du deuxième paramètre dans cette implémentation en le laissant sa valeur par défaut.

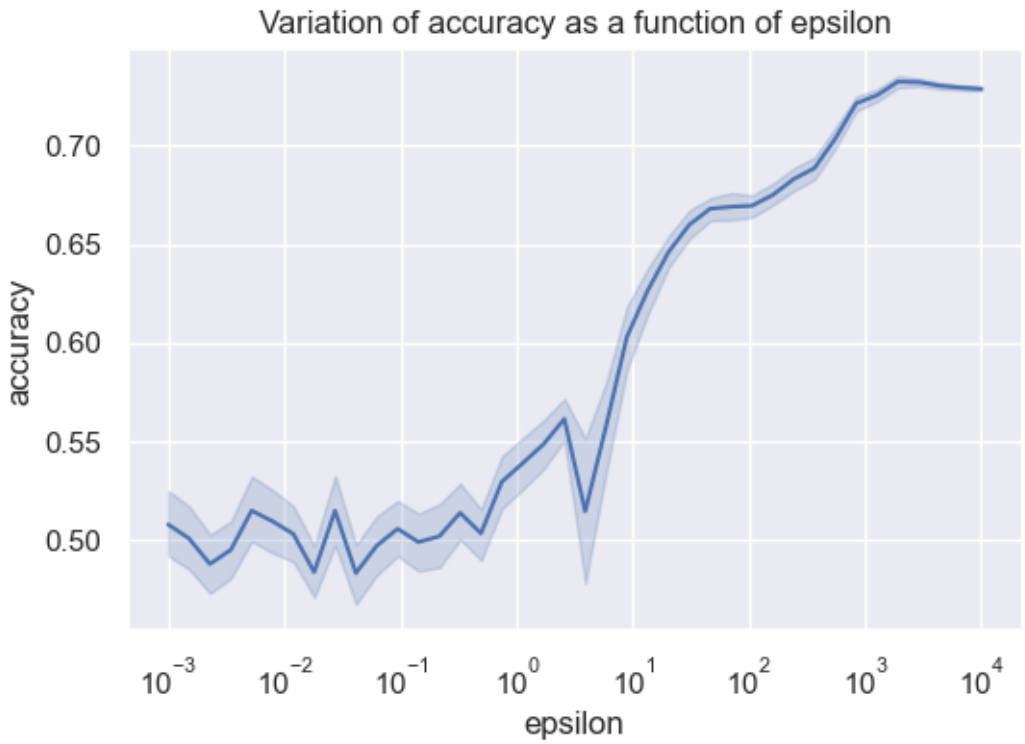
[23]:
```
epsilon = 10
data_norm = 10

model_overfit_diff = Pipeline(
    steps=[
        ('relations', FunctionTransformer(add_relations)),
```

```
        ('preprocessor', preprocessor_comb),
        # The only difference with model_over
        ('logreg', diff.LogisticRegression(max_iter=200, data_norm=data_norm))
    ]
)
```

Dans un premier temps, on souhaite vérifier l'implémentation de la confidentialité différentielle. Pour cela, on vérifie que lorsque l'on augmente le paramètre ϵ, on diminue la protection du modèle et ainsi on tend vers une performance proche de la performance du modèle initiale lorsque celui-ci n'était pas protégé. On souhaite également observer que diminuer le paramètre ϵ augmente la protection du modèle et fait décroître ses performances.

[24]:
```
if NEW:
    results = pd.DataFrame()
    for epsilon in tqdm(np.logspace(-3, 4, 40)):
        for _ in range(100):
            model_overfit_diff.set_params(logreg__epsilon=epsilon)
            model_overfit_diff.fit(X_train, y_train)
            accuracy = model_overfit_diff.score(X_test, y_test)

            results = results.append({'accuracy': accuracy, 'epsilon':epsilon},
                                     ignore_index=True)

    np.savez_compressed(f'data/privacy/diffpriv.npz', results=results)
else:
    file = np.load(f'data/privacy/diffpriv.npz', allow_pickle=True)
    results = pd.DataFrame(file['results'], columns=['accuracy', 'epsilon'])
```

[25]:
```
grid = sns.lineplot(x='epsilon', y='accuracy', data=results)
grid.set(xscale="log")
plt.title('Variation of accuracy as a function of epsilon')
plt.show()
```



Pour quantifier les performances du modèle, on choisi d'étudier la performance de ce dernier (*accuracy). Pour cela, On a tracé l'évolution de la performance du modèle en fonction du paramètre* epsilon*. Le graphe obtenu ci-dessus illustre correctement la relation entre le paramètre* epsilon* et la performance et comme on s'y attendait, relâcher la contrainte de sécurité (i.e. diminuer *epsilon*) entraîne une augmentation des performances du modèle. On note également que lorsque le budget est trop faible, le modèle devient inutilisable en atteignant une performance de 0.5 qui la valeur de la performance d'un modèle de classification binaire prédisant aléatoirement.

Pour tester l'efficacité de cette méthode de protection, on utilise l'attaque d'inférence de propriété créée plus haut pour détecter s'il y a plus d'hommes que de femmes dans la base de données d'entraînement et on regarde l'efficacité de cette attaque sur notre modèle protégé.

Pour voir l'impact du budget ϵ sur la protection du modèle, on génère 300 modèles en augmentant progressivement le budget ϵ et on calcule le pourcentage de modèles que notre méta modèle est capable de prédire correctement. On rappelle que ce méta modèle cherche à prédire si le modèle donné en entrée s'est entraîné sur une base de données constituée de plus de femmes ou bien l'inverse.

On s'attend à ce que plus le budget est faible, plus le modèle est protégé et plus le méta modèle a du mal à distinguer entre les deux classes.

[26]:
```
n_models = 300
n_entries = len(X_train)
```

```
# chose if the distribution of all variables except 'sex' is choosen with
# the distribution of the secret database or uniformaly
unif = False

dist_name = '_unif' if unif else '_prop'

# setting epsilon as 'inf' is the same as having no differential privacy
model_overfit_diff.set_params(logreg__epsilon=float('inf'))
model_archi = model_overfit_diff

if NEW:
    list_models, dist_woman = gen_models(model_archi, n_models, n_entries, unif)

    meta_X_train = [model['logreg'].coef_[0] for model in list_models]
    meta_y_train = (dist_woman > 0.5).astype('int')

    np.savez_compressed(f'data/privacy/data_diff{dist_name}.npz',
                        meta_X_train=meta_X_train,
                        meta_y_train=meta_y_train)
else:
    file = np.load(f'data/privacy/data_diff{dist_name}.npz', allow_pickle=True)
    meta_X_train = file['meta_X_train']
    meta_y_train = file['meta_y_train']
```

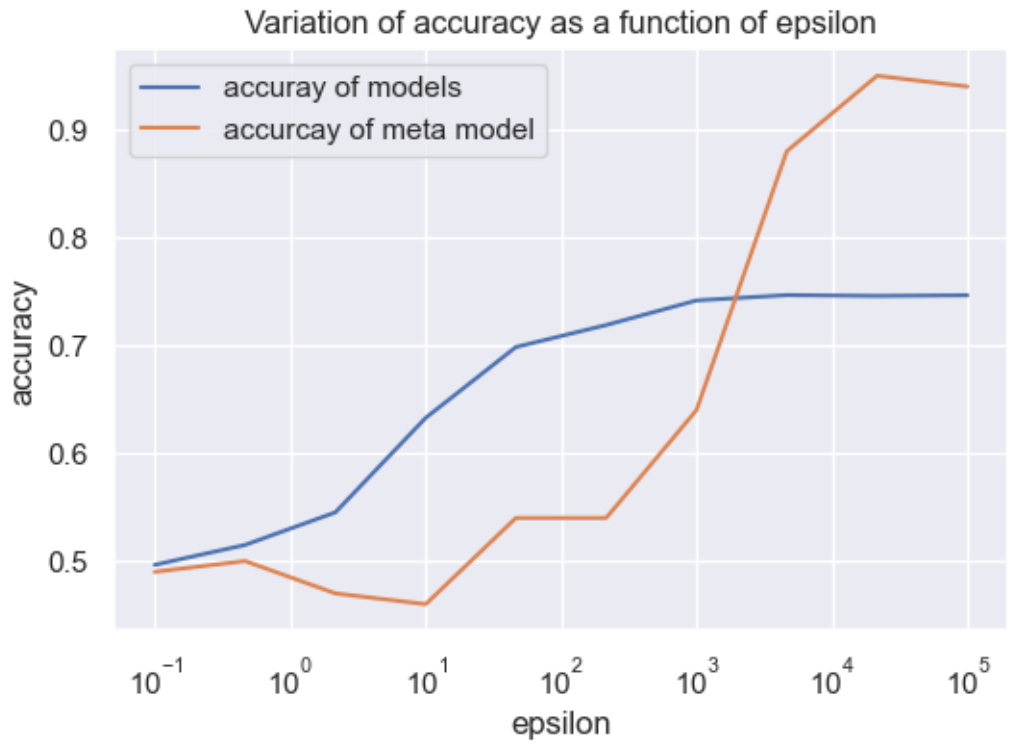
[27]:
```
meta_model = LogisticRegression(max_iter=200)
meta_model.fit(meta_X_train, meta_y_train)
```

[27]:
```
LogisticRegression(max_iter=200)
```

[28]:
```
n_models = 100
n_entries = len(X_train)

columns=['epsilon', 'accuracy', 'sensitivity',
         'specificity', 'precision']

model_archi = model_overfit_diff

if NEW:
    list_score, list_meta_score = [], []
    for epsilon in np.logspace(-1, 5, 10):
        model_archi.set_params(logreg__epsilon=epsilon)
        list_models, dist_woman = gen_models(model_archi, n_models, n_entries)

        meta_X_test = [model['logreg'].coef_[0] for model in list_models]
        meta_y_test = (dist_woman > 0.5).astype('int')

        list_meta_score.append(meta_model.score(meta_X_test, meta_y_test))
```

```
        list_score.append([model.score(y_test, model.predict(X_test))
                                            for model in list_models])

    np.savez_compressed(f'data/privacy/data_diff{dist_name}_test.npz',
                        list_meta_score=list_meta_score,
                        list_socre=list_score)
else:
    file = np.load(f'data/privacy/data_diff{dist_name}_test.npz',␣
↪allow_pickle=True)
    list_score = file['list_socre']
    list_meta_score = file['list_meta_score']
```

[29]:
```
plt.plot(np.logspace(-1, 5, 10),
         np.array(list_score).mean(axis=1),
         label='accuray of models')
plt.plot(np.logspace(-1, 5, 10),
         np.array(list_meta_score),
         label='accurcay of meta model')

plt.xscale('log')
plt.title('Variation of accuracy as a function of epsilon')
plt.xlabel('epsilon')
plt.ylabel('accuracy')
plt.legend()

plt.show()
```



Ce graphique représente l'évolution de la performance du méta modèle (en orange) et l'évolution de la moyenne des performances des modèles sous-jacents (en bleu) en fonction du paramètre ϵ. Ces deux courbes montrent que plus le paramètre ϵ augmente, plus la performance des modèles et du méta modèle sont bonnes. On retrouve pour la courbe bleu le même comportement que le graphique précédent avec la performance des modèles qui tendent vers la performance d'un modèle sans confidentialité différentielle lorsqu'ϵ tend vers l'infini. La courbe orange montre l'existence d'un saut dans les performances du méta modèle pour une valeur d'ϵ égale à 1000. Enfin, les deux courbes illustrent le fait que lorsque le paramètre de budget est trop petit, les modèles et le méta modèle se comportent comme des modèles de classifications binaires aléatoires avec une performance d'environ 0.5.

Ces courbes mettent en lumière l'existence d'un compromis nécessaire entre les performances d'un modèle et sa capacité à résister à une attaque d'inférence de propriété. En effet, plus nos modèles accordant les prêts sont efficaces (i.e. plus la courbe bleue est haute), plus le méta modèle détectant s'il y a plus d'hommes que de femmes dans la base d'entraînement est précis (i.e. plus la courbe orange est haute). Comme les deux courbes n'ont pas la même allure on peut choisir un ϵ qui maximise la différence entre la courbe de performance des modèles et la courbe de performance du méta modèle. Cette valeur d'ϵ correspond à un modèle relativement performant et assez robuste aux attaques. Dans notre cas, ce point est atteint pour ϵ proche de 100.

4.3 Critiques et conclusion

Nous avons pu voir dans la partie précédente que le fait de négliger l'aspect protection des données lors de l'entraînement d'un modèle laissait la possibilité à une personne mal intentionnée de

récupérer les paramètres de ce modèle ou d'avoir accès à des informations sur la base de données utilisée pour l'entraînement de ce dernier. Des solutions telles que la confidentialité différentiable impliquent un apprentissage dit "agrégé" qui permet de protéger les données d'entraînement. L'utilisation de ces solutions reste délicate et les raisons à cela sont données dans les paragraphes suivants.

4.3.1 Le manque de ressource rend la prise en main des librairies difficile.

On constate qu'il y a trois librairies principales pour protéger la *privacy* :

- Diffprivlib : développée par IBM depuis 2019
- OpenDP : développée par OpenDP (Harvard) depuis 2019
- Opacus : développée par Facebook depuis 2020
- TensorFlow Privacy : dévoloppée par Google depuis 2019

Elles ont en commun d'être très récentes, elles sont donc plus difficiles à mettre en œuvre à cause du manque d'exemples et de retour utilisateur. Par exemple, la librairie *Diffprivlib* a une documentation claire et précise mais qui ne contient pas d'exemple d'implémentation. Ces exemples sont présents dans un autre dossier dans le code du projet et restent assez peu nombreux. À cela s'ajoute le manque d'échange sur les plateforme d'entraide comme *stack overflow*. Pour *Diffprivlib* et *Opacus* on ne trouve que deux questions sans réponses et pour *OpenDP*, aucune question n'est présente. Ces échanges sont très importants pour le développement de ces libraires car ils proposent des exemples de cas réels et facilitent la compréhension des implémentations. La démocratisation de ces outils est donc très importante pour une adoption des outils. Pour espérer que cela soit le cas, il faut évidemment continuer de développer ces librairies, mais surtout d'encourager le maximum de personnes avec des connaissances techniques à utiliser les outils et de proposer un retour. La création d'exemples pertinents présents dans la documentation viendra de cet échange entre développeur et utilisateur. De plus, ces échanges permettent bien souvent d'avoir des avis sur les avantages et les inconvénients des différentes méthodes existantes. Les trois librairies ne contiennent d'ailleurs aucun document de comparaison. Tous ces problèmes impliquent qu'aujourd'hui si un individu s'intéresse à la protection des données dans des modèles statistiques il doit s'instruire de manière importante sur le sujet et avoir une idée très claire de ce qu'il cherche avant d'implémenter les méthodes. Encourager des méta analyses pourrait dans ce sens abaisser la barrière d'entrée en permettant de créer des intuitions sur les avantages et les inconvénient des méthodes de la protection des données. Ce manque de ressource est évidemment un frein pour les individus mais pour les industriels ces librairies sont d'autant plus difficile à mettre en place qu'ils ne sont pas les cibles de celles-ci.

4.3.2 Le public visé par ces librairies ne permet pas une mise en œuvre efficace en entreprise.

Les trois librairies citées plus haut ont des méthodes et de la documentation d'une grande précision, mais elles ont l'inconvénient d'avoir été écrites pour des développeurs et des scientifiques et non pas des industriels. Comme dit plus haut, les concepteurs de ces librairies partent du principe que l'utilisateur à une bonne connaissance des dernières recherches en apprentissage statistique. Or, avant d'être un problème académique, la protection de la vie privée est un problème opérationnel. Les méthodes utiles pour des problèmes concrets sont perdues parmi les dizaines d'implémentations des dernières publications dans le domaine. Par exemple, *Opacus* ne s'implémente que sur des modèles codés en *PyTorch*, la librairie d'apprentissage profond de Facebook, et ne propose que

trois exemples sur des réseaux complexes et récents (image classifier, text classifier, LSTM). Ces exemples sont intéressants d'un point de vue technique mais ne prennent pas en compte l'écart qu'il existe entre la réalité industrielle et le monde académique. Dans la majorité des cas les modèles utilisés sont soit relativement simples comme des régressions ou des arbres de décisions soit les modèles proviennent d'outils qui proposent des modèles complexes prêt à l'emploi. Pour une adoption rapide des méthodes de protection des données, il faut se concentrer sur la clarté et la simplicité des modèles utilisés en industries ou inclure ces méthodes dans les modèles pré-entraînés. Ces changements sont possibles si les industriels s'approprient le sujet. Tout comme l'exemple de *Hugging face* qui a démocratisé les modèles complexes de traitement du langage en s'adressant aux entreprises, on peut très bien imaginer qu'une entreprise s'empare du sujet. Il faut également poursuivre la production de communication non scientifique pour susciter une prise de conscience.

4.3.3 Il existe un arbitrage fort entre performance et privacy

La dernière remarque sur cette partie est que l'utilisation de la privacy a un impact négatif sur le temps d'entraînement du modèle ou bien sur les performances de celui-ci. Protéger notre modèle implique donc de faire des concessions et comme nous allons le voir, il est possible de choisir les concessions que l'on souhaite faire. L'ajout de calcul à tout moment de l'implémentation d'un modèle entraîne un temps d'entraînement ou d'exécution plus long. Il faut donc choisir entre la mise en place d'un grand nombre de sécurité ou une utilisation rapide du modèle, cela dépendra de l'importance que l'on porte à la protection des données du modèle. Mais le temps n'est pas la seule contrainte, des méthodes de protection des données ont une influence sur les performances du modèle. Par exemple, le *differential privacy* permet de contrôler l'impacte de la méthode sur la performance d'un modèle. Un des paramètres de cette méthode est justement la variation maximale de l'*accuracy* que l'on est prêt à tolérer lors de l'ajout ou du retrait d'un élément de la base d'entraînement. Ce contrôle ne s'applique que pour la métrique d'*accuracy* mais il existe un grand nombre d'autres métriques permettant de rendre compte de la performance d'un modèle. Pour celles-ci, l'expérimentation est encore la meilleure solution. Il est intéressant de noter que des concessions sont également nécessaires entre les trois notions présentées dans ce rapport. Par exemple [arxiv.org/pdf/1905.12101.pdf] Bagdasaryan & Shmatikov (2019) démontre que l'impact de l'*accuracy* entraînait des conséquences non symétriques sur tous les groupes d'entrées. Ainsi les groupes sous représentés étaient plus touchés par la perte d'*accuracy*. De même, l'obfuscation de l'architecture ou des résultats qui permettra une meilleure contrôle sur la sécurité des données mais empêche d'implémenter des outils d'explications de décision. C'est ainsi que la protection des données d'un modèle peut se faire au détriment de la transparence. On a donc besoin de documents non techniques et de communication lorsque l'on explique les enjeux de la notion de protection des données. Comprendre les différentes options possibles et surtout rendre claire que la mise en place des outils implique des compromis. Des comparaisons des différentes méthodes et de leur impact sur les métriques devraient être présentes dans les exemples de librairies.

5 Outils pour la sécurité algorithmique

[1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from tqdm import tqdm
from itertools import product
from copy import deepcopy

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report

import utility as uti

import seaborn as sns
sns.set_theme(style='darkgrid')

plt.rcParams['figure.dpi'] = 100
```

[2]:
```
%%capture
%run ./Introduction.ipynb
```

[3]:
```
seed = 2021   # Keep results consistent through testing
NEW = False   # True : recreate all data (can be very long), False : read data␣
 ↪from disk
```

5.1 Présentation de la notion de *robustness*

Le concept de stabilité statistique (*robustness*) est une notion importante en statistique et en économétrie qui existait déjà lorsque les modèles d'apprentissage machines sont apparus. Il est défini de la manière suivante :

Un modèle est robuste s'il n'est pas affecté par la présence de valeurs aberrantes ou de déviation des hypothèses sur les données de départ.

Cette définition a été contrainte d'évoluer afin de s'adapter à la nouvelle manière de mettre les modèles à jour. Si auparavant les modèles étaient entraînés au sein d'un laboratoire avec un grand contrôle des données d'entraînement, aujourd'hui les modèles se mettent à jour constamment et de façon automatisée. Cette automatisation induit une perte de contrôle stricte des données et une nouvelle vulnérabilité qui rendent nécessaire l'ajout d'un nouveau rôle pour la stabilité.

En apprentissage statistique, un modèle est dit robuste s'il n'est pas affecté par des données qui ne sont pas similaires à celles sur lesquelles le modèle s'est entraîné.

Dans la plupart des cas, ces données sont spécifiquement construites par une entité malveillante dans le but de grandement impacter les résultats du modèle de départ. Cette nouvelle notion de stabilité algorithmique est pour cela nommée *adverserial robustness* (*adverserial* dans le sens conflictuel).

Il existe trois catégorisations des attaques d'*adverserial robustness* qui répondent chacune à une question :

- Qu'est-ce qui est attaqué ?
- Comment est-il attaqué ?
- Pourquoi est-il attaqué ?

Tout comme les attaques de *privacy*, les attaques d'*adverserial robustness* sont catégorisés dans un premier temps entre les attaques sur modèle boîte blanche et sur modèle boîte noire. Un modèle boîte blanche est un modèle dont on peut prévoir le fonctionnement interne car on connaît toutes ces caractéristiques des éléments qui le composent. En apprentissage statistique cela correspond à un modèle où l'on a accès aux données d'entraînement et de test, à la structure, aux coefficients et toute autre information permettant de reproduire le modèle à l'identique. Au contraire, le modèle boite noire est un modèle pour lequel on a aucune visibilité sur le fonctionnement interne. Dans le cas du modèle boîte noire, on a ainsi seulement accès aux entrées et sorties du modèle. En apprentissage statistique le modèle de boîte noire correspond à la majorité des modèles commerciaux qui fonctionnent sous forme d'API (*Application Programming Interface*). L'utilisateur de ces modèles envoie des données au modèle en question et ne récupère que la sortie. Les attaques *adverserial robustness* contre des modèles boîtes blanches sont très faciles à mettre en œuvre et ces modèles sont très difficilement protégeables. Cette raison motive l'existence des modèles en boîte noire qui sont des modèles pouvant espérer être robustes aux attaques adversaires.

5.1.1 Qu'est-ce qui est attaqué ?

Il existe différents types d'attaque selon la partie du modèle qui est ciblée. Les attaques d'évasion (*evasion attacks*) qui ajustent les données de test, les attaques d'empoisonnement (*poisining attacks*) qui ajustent les données d'entraînement, et les attaques d'exploration (*exploration attacks*) qui questionnent le modèle boite noire pour faire de la rétro-ingénierie, sont les trois grandes familles d'attaque.

5.1.2 Comment est-il attaqué ?

Les façons selon lesquelles le modèle est attaqué dépendent grandement de la quantité et du type d'information dont l'adversaire dispose.

Avant l'entraînement : Selon s'il peut modifier la base d'entraînement ou uniquement ajouter de nouvelles données à celle-ci, l'adversaire pourra effectuer une *data modification* ou une *data injection*. Si en revanche il a accès au modèle, il pourra corrompre directement la logique de celui-ci (*logic corruption*).

Après l'entraînement : Deux types d'attaque existent :

- *Adaptative attacks* : Ces attaques consistent à modifier les entrées en multipliant les requêtes jusqu'à ce que la sortie convienne. On cherche alors à obtenir une sortie spécifique ou à créer des entrées sur lesquels le modèle se trompe.
- *Non-adaptative attacks* : La modification des entrées se fait grâce à une connaissance préalable d'informations sur les données d'entraînement dans le même but que pour les attaques adaptatives.

5.1.3 Pourquoi est-il attaqué ?

La raison d'une attaque appartient à l'une des quatre grandes familles suivante :

- Réduction de la qualité générale de prédiction d'un modèle
- Engendrer une erreur de classification sur toutes les entrées
- Engendrer une erreur de classification sur toute une classe d'entrée
- Engendrer une erreur de classification sur une entrée précise

Dans la suite nous allons mettre en oeuvre deux attaques :

- une évasion contre un modèle boîte noire de type attaque adaptative ayant pour but d'engendrer une erreur de classification sur une entrée précise.
- un empoisonnement contre un modèle boîte noire de type *data injection* ayant pour but d'engendrer une erreur de classification sur une entrée précise.

5.2 Description générale des différentes attaques et implémentation

Le cas de figure est le suivant, on suppose que notre banque utilise le modèle de régression logistique développé dans l'article d'introduction. Celui-ci utilise une base d'entraînement de 800 personnes et les classe en fonction de leur capacité de remboursement (0 : capable de rembourser, 1 : incapable de rembourser). Dans cette optique, on peut tout à fait concevoir qu'un individu apprenant que son prêt a été refusé par le modèle va chercher à contourner celui-ci. Nous présentons ici deux cas de figure.

5.2.1 Évasion

On se place dans une situation où l'individu est en mesure de questionner le modèle sans contrainte. On peut imaginer un problème de sécurité qui cause la divulgation du modèle, l'utilisation par la banque d'un modèle open source ou l'utilisation d'un modèle annexe que l'on aurait calibré grâce à la méthode de *model extraction* (cf. l'article *privacy*). Le type d'attaque utilisé est une attaque adaptative donc on va chercher les variables les plus falsifiables et les modifier jusqu'à ce que l'individu refusé soit accepté.

En observant les différentes variables on remarque que très peu d'entre elles peuvent être modifiées sans que cela soit vérifiable. La cause à cela est qu'il s'agit d'une base de données pour crédit, aussi les informations du client sont examinées avec précaution. On peut néanmoins supposer que les deux variables suivantes peuvent être falsifiées sans difficulté.

- purpose : correspond à la finalité du prêt
- personal_status : la condition de vie en couple

L'attaque commence par choisir une personne dont le prêt a été refusé pour jouer le rôle d'adversaire :

[4]:
```
idx_default = np.where(logreg.predict(X_test))[0]

np.random.seed(seed)
adv_ind = X_test.iloc[np.random.choice(idx_default)]
adv_ind
```

[4]:
```
account_check_status                                          < 0 DM
duration_in_month                                                 24
credit_history                          existing credits paid back duly till now
purpose                                                      car (new)
credit_amount                                                   1207
savings                                                  … < 100 DM
present_emp_since                                        … < 1 year
installment_as_income_perc                                         4
sex                                                           female
personal_status                                              divorced
other_debtors                                                   none
present_res_since                                                  4
property                      if not A121 : building society savings agreeme…
age                                                               24
other_installment_plans                                         none
housing                                                         rent
credits_this_bank                                                  1
job                                              skilled employee / official
people_under_maintenance                                           1
telephone                                                       none
foreign_worker                                                   yes
Name: 504, dtype: object
```

Notre individu ci-dessus est une femme étrangère divorcée de 24 ans avec une personne à sa charge qui n'a pas remboursé toutes ces dettes. Elle se voit refuser son crédit pour une nouvelle voiture et cherche donc à modifier les variables difficilement vérifiables (ie `purpose` et `personal_status`) afin d'avoir une chance que son prêt soit accepté. Pour cela, elle utilise une méthode exhaustive en créant toutes les combinaisons possibles avec les variables difficilement vérifiables et les faire tester par le modèle dans l'espoir que l'une d'entre elle lui permette que son prêt soit accepté.

Le modèle de régression logistique attribue un score dans l'intervalle [0,1] à chaque individu puis le place dans l'une des deux classes prêt accordé / prêt refusé en fonction du score obtenu. Aussi un score supérieur à 0.5 placera l'individu dans la catégorie 1 : prêt refusé, tandis qu'un score inférieur ou égal à 0.5 placera l'individu dans la catégorie 0 : prêt accepté.

[5]:
```
# variables that are difficults to verify
var_columns = ['purpose', 'personal_status']

# variables that are hard to falsify (all the other)
fix_columns = [x for x in X.columns if x not in var_columns]

# catesian product of all possibilities from variables that can change
all_values = [X[col].unique() for col in var_columns]
prod = list(product(*all_values))

# all possible entries that can be generated
all_adv_ind = pd.DataFrame([adv_ind[fix_columns].tolist() + list(p) for p in␣
 ↪prod],
```

```
                      columns=fix_columns + var_columns)
all_adv_ind = all_adv_ind[X.columns]

# predict the score for all entries
all_adv_ind['proba'] = logreg.predict_proba(all_adv_ind)[:, 1]
```

[6]:
```
# take only the entries that gets a 0
valid_adv_ind = all_adv_ind[all_adv_ind.proba < 0.5]

# calculate the number of variables that were changed
nb_diff = []
for i in range(len(valid_adv_ind)):
    val_diff = (valid_adv_ind.iloc[i][var_columns] == adv_ind[var_columns])
    nb_diff += [val_diff.value_counts()[False]]

valid_adv_ind = valid_adv_ind.assign(diff = nb_diff)

# take the least change entrie that have the biggest impact
valid_adv_ind.sort_values(by=['diff', 'proba'], inplace=True)
valid_adv_ind.head(3)
```

[6]:
```
   account_check_status  duration_in_month  \
12               < 0 DM                 24
14               < 0 DM                 24

                            credit_history     purpose  credit_amount  \
12  existing credits paid back duly till now  car (used)           1207
14  existing credits paid back duly till now  car (used)           1207

         savings present_emp_since  installment_as_income_perc     sex  \
12  … < 100 DM      … < 1 year                             4  female
14  … < 100 DM      … < 1 year                             4  female

   personal_status  … age  other_installment_plans housing  \
12          single  …  24                    none    rent
14         married  …  24                    none    rent

    credits_this_bank                          job people_under_maintenance  \
12                  1  skilled employee / official                         1
14                  1  skilled employee / official                         1

    telephone foreign_worker     proba diff
12       none            yes  0.387855    2
14       none            yes  0.471957    2

[2 rows x 23 columns]
```

On voit ainsi que modifier les variables `purpose` et `personal_status` suffit pour passer en dessous

d'un score de 0.5 faisant ainsi changer la classe de notre individu de 1 à 0 (le prêt est maintenant accepté). Voyons la combinaison de ces variables ayant permis ce changement de classe.

[7]:
```
def print_diff(old_comb, new_comb):
    '''Print the difference between two entries'''
    diff_comb = (new_comb != old_comb)
    diff_pd = pd.DataFrame([old_comb[diff_comb],
                            new_comb[diff_comb]], index=['old', 'new']).
 ↪transpose()

    if diff_pd.empty: raise ValueError

    # juste for styling
    max_ind = max([len(x) for x in diff_pd.index])
    max_old = max([len(x) for x in diff_pd.old])
    max_new = max([len(x) for x in diff_pd.new])

    for i in new_comb[diff_comb].index :
        print('{:<{}} : {:<{}} --> {:<{}}'.format(i, max_ind,
                                                  diff_pd.old[i], max_old,
                                                  diff_pd.new[i], max_new))
```

[8]:
```
try:
    print_diff(adv_ind[var_columns], valid_adv_ind.iloc[0][var_columns])
except ValueError:
    print("Aucun changement")
```

```
purpose         : car (new) --> car (used)
personal_status : divorced  --> single
```

Pour passer à un score de moins de 0.5, il suffit à notre individu de falsifier la raison de son prêt en disant que c'est pour une voiture d'occasion et de dire qu'elle n'a jamais été mariée.

Maintenant que nous avons fait l'étude de cas sur un individu en particulier, nous allons voir comment ce raisonnement se généralise à notre population de test. On applique la même méthode pour chaque individu qui se voit refuser son crédit et on calcule la probabilité minimale atteignable pour chacun d'entre eux.

[9]:
```
def gen_all_ind(df, var_columns):
    # variables that do not change
    fix_columns = [x for x in df.columns if x not in var_columns]

    # catesian product of all possibilities from variables that can change
    all_values = [X[col].unique() for col in var_columns]
    prod = list(product(*all_values))

    all_inds = []
    for idx, ind in tqdm(df.iterrows()):
        for p in prod:
```

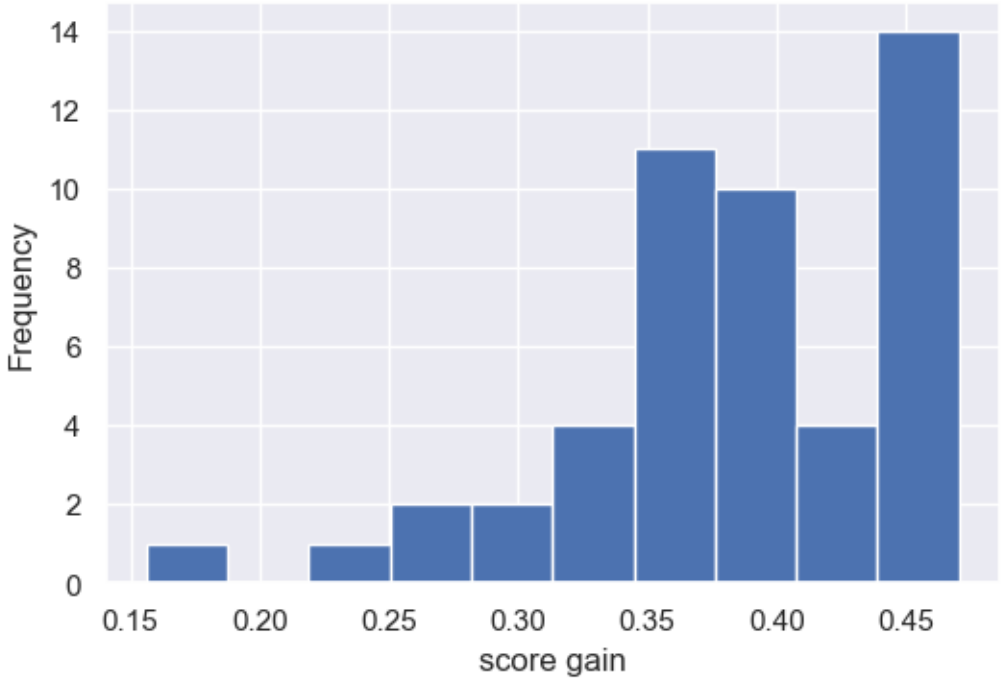
```
            all_inds += [list(ind[fix_columns])
                         + list(p)
                         + [idx]]

    column_names = fix_columns + var_columns + ['index_old']
    return pd.DataFrame(all_inds, columns=column_names)
```

[10]:
```
X_default = X_test.iloc[idx_default]
X_default = X_default.assign(old_proba = logreg.predict_proba(X_default)[:, 1])

if NEW:
    old_var = [X_default[name].rename('old_'+name) for name in var_columns]
    X_default = pd.concat([X_default, pd.DataFrame(old_var).transpose()],␣
 ↪axis=1)

    all_adv_inds = gen_all_ind(X_default, var_columns)

    all_adv_inds['proba'] = logreg.predict_proba(all_adv_inds[X.columns])[:, 1]

    np.savez_compressed('data/privacy/all_adv_inds.npz',
                        data=np.array(all_adv_inds),
                       columns=all_adv_inds.columns)
else:
    file = np.load('data/privacy/all_adv_inds.npz', allow_pickle=True)
    all_adv_inds = pd.DataFrame(file['data'], columns=file['columns'])


# fonction that takes the minimum proba of a group of entries
def min_proba(df):
    return df.sort_values('proba').head(1).proba

proba_min = all_adv_inds.groupby('index_old').apply(lambda df: min_proba(df))
X_default = X_default.assign(proba_min = proba_min.droplevel(1))

print('Default label : {}'.format(len(X_default)))
print('Change to no-default : {}'.format(len(X_default[X_default.proba_min < 0.
 ↪5])))
```

```
Default label : 49
Change to no-default : 46
```

Sur les 49 personnes labellisées comme incapable de rembourser dans la base de données de test, l'évasion à permis de changer la labellisation pour 46 d'entre eux. Autrement dit, 94% des personnes dans la base de test dont le prêt s'est vu refusé peuvent se faire accepter en changeant seulement la raison du prêt et/ou leur statut de relation. Ceci est très problématique du point de vue de la banque. Voyons maintenant la distribution du gain de score obtenu par l'évasion.

[11]:
```
(X_default.old_proba - X_default.proba_min).plot.hist()
plt.ylabel('Frequency')
plt.xlabel('score gain')
plt.show()
```



On voit sur ce graphique qu'une majorité d'individus voient leur score diminuer d'au moins 0.35 lorsqu'ils changent les deux paramètres falsifiables. Notre modèle n'est ainsi pas robuste car il ne prend pas en compte que l'agent peut être malveillant.

On remarque tout de même qu'une trentaine de personnes ne sont pas capables de modifier le label attribué (1 : le prêt est refusé) en falsifiant ces deux variables. La partie suivante va démontrer que si l'agent est capable de rajouter des éléments dans la base d'entraînement (*data augmentation*) alors tous ces individus peuvent voir leur prêt se faire accepter.

Poisoning On se place cette fois dans une situation un peu moins probable où l'adversaire peut ajouter des éléments dans la base de données d'entraînement. On peut néanmoins imaginer un complice dans le département informatique qui aurait accès à la base et qui voudrait assurer un prêt pour l'adversaire. On ne suppose pas ici de modification des éléments existants (ce qui serait une attaque par *data modification*) mais bien juste l'ajout d'éléments à la base de données (*data injection*).

Nous allons voir qu'avec un petit nombre d'ajouts on peut faire passer un individu de 1 (défaut) à 0, cette situation étant particulièrement intéressante pour les individus pour qui l'évasion n'a pas été suffisante. Commençons par choisir un individu au hasard parmi les personnes refusées après la tentative d'évasion.

[12]:
```
X_poison = X_default[X_default.proba_min >= 0.5]
idx_poison = X_poison.index

np.random.seed(seed)
adv_ind = X.loc[[np.random.choice(idx_poison)]]
adv_ind.iloc[0]
```

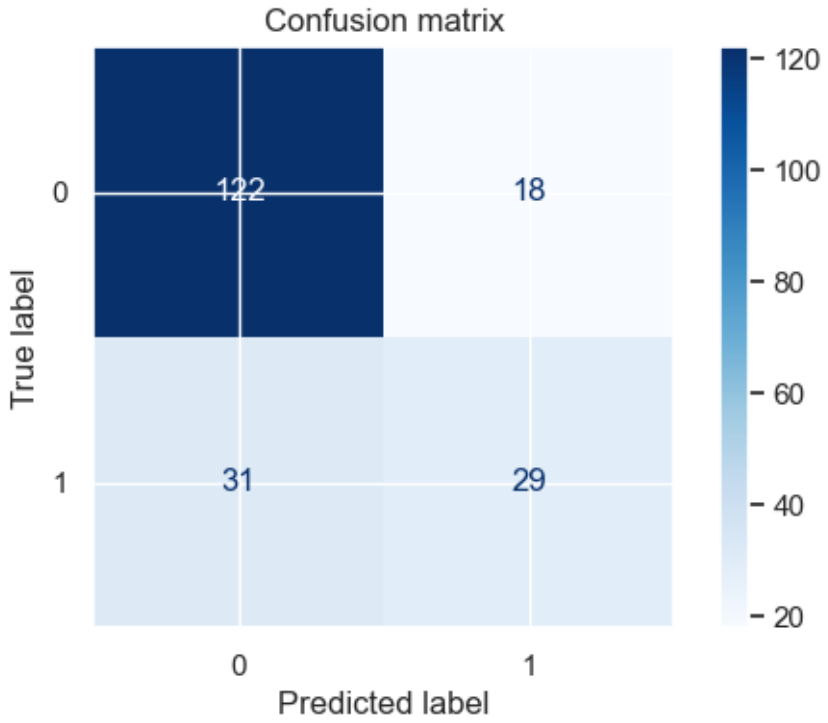
[12]:
```
account_check_status                                    0 <= … < 200 DM
duration_in_month                                                    36
credit_history                          delay in paying off in the past
purpose                                                      car (new)
credit_amount                                                     7432
savings                                                  … < 100 DM
present_emp_since                                    1 <= … < 4 years
installment_as_income_perc                                          2
sex                                                            female
personal_status                                               divorced
other_debtors                                                     none
present_res_since                                                    2
property                  if not A121 : building society savings agreeme…
age                                                                 54
other_installment_plans                                           none
housing                                                           rent
credits_this_bank                                                    1
job                                        skilled employee / official
people_under_maintenance                                             1
telephone                                                         none
foreign_worker                                                     yes
Name: 815, dtype: object
```

Notre individu (815) est cette fois-ci une femme de 54 ans, seul, qui a eu du mal à payer ses crédits passés. C'est une travailleuse étrangère qui demande un peu plus de 7400 DM pour une voiture neuve. Les attaques d'évasion ne sont pas suffisantes pour qu'un prêt soit accordé à cette personne. Dans la suite, nous travaillons avec les données originelles de cet individu (les attaques d'évasions ont été annulées) afin de voir si le *poisoning* seul suffit à renverser la tendance.

Pour cette attaque, on ajoute 5 fois la ligne de l'individu 815 avec comme label 0 dans les données d'entraînement du modèle. Cela consiste en réalité à rentrer 5 fois le profil de l'individu en stipulant que le prêt a été accordé toutes ces fois afin de biaiser le modèle. On entraîne alors le modèle sur cette nouvelle base de données et on regarde les changements obtenus concernant la prédiction du nouveau modèle ainsi que l'impact que ce changement a eu sur les différentes métriques. Le but est alors de diminuer le score de l'individu 714 sans pour autant modifier en profondeur le modèle.

[13]:
```
# Number of time the line will be added to the training set
n = 5

# Adding the lines to the training set
X_train_pois = X_train.append([adv_ind]*n, ignore_index=True)
```

```
y_train_pois = pd.Series(list(y_train) + [0]*n)

# Seeing metrics for the new model train on the poison training set
logreg_pois = model.fit(X_train_pois, y_train_pois)
uti.mesure_clas_list([logreg_pois, model], X_test, y_test, ['poison', 'base'])
```



	Sensitivity	Specificity	Precision	Accuracy
poison	0.87	0.48	0.62	0.76
base	0.87	0.48	0.62	0.76

[14]:
```
old_score = logreg.predict_proba(adv_ind)[0,1]
new_score = logreg_pois.predict_proba(adv_ind)[0,1]
print(f'{old_score:.3f} --> {new_score:.3f}')
```

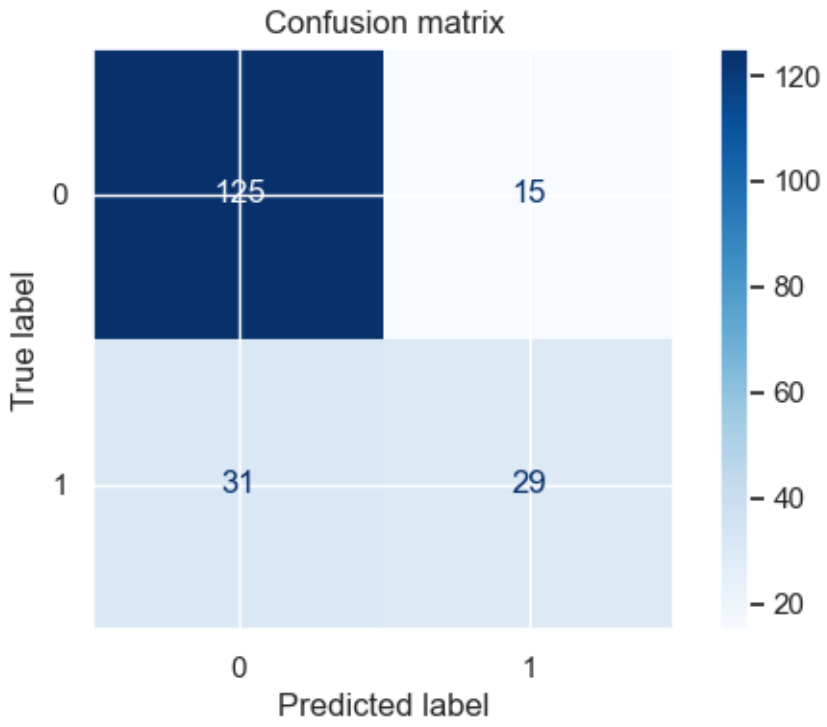
```
0.858 --> 0.634
```

On a une nette amélioration du score de l'individu 815 et comme le montre la matrice de confusion qui compare les labels associés au modèle initial à ceux associés au modèle entraîné sur la base de donnée modifiée, la classification des autres individus a été peu impactée. Malgré nos efforts, l'ajout de 5 entrées n'est pas suffisant pour obtenir le changement de label concernant l'individu 815. Dans ce qui suit, on va chercher le nombre minimum d'ajouts permettant ce changement de classe.

[15]:
```python
def min_pois(ind, echo=False, X_train=X_train, y_train=y_train):
    n = 0
    X_train_pois = X_train
    y_train_pois = y_train

    # if ind is a pd.Series we change it to a pd.Dataframe
    if isinstance(ind, pd.Series):
        ind = ind.to_frame().transpose()

    logreg_pois = model.fit(X_train_pois, y_train_pois)

    # Adding one by one the same entry until the score is less or equal that 0.5
    while logreg_pois.predict_proba(ind)[0,1] > 0.5:
        n += 1
        X_train_pois = X_train_pois.append([ind], ignore_index=True)
        y_train_pois = pd.Series(list(y_train_pois) + [0])

        logreg_pois = model.fit(X_train_pois, y_train_pois)

    if echo:
        old_score = logreg.predict_proba(ind)[0,1]
        new_score = logreg_pois.predict_proba(ind)[0,1]

        print(f'{n:>2} : {old_score:.3f} --> {new_score:.3f}')
        uti.mesure_clas_list([logreg_pois, logreg], X_test, y_test,
                             ['poison', 'base'])

    return n
```

[16]:
```python
_ = min_pois(adv_ind, echo=True)
```

```
10 : 0.858 --> 0.479
```

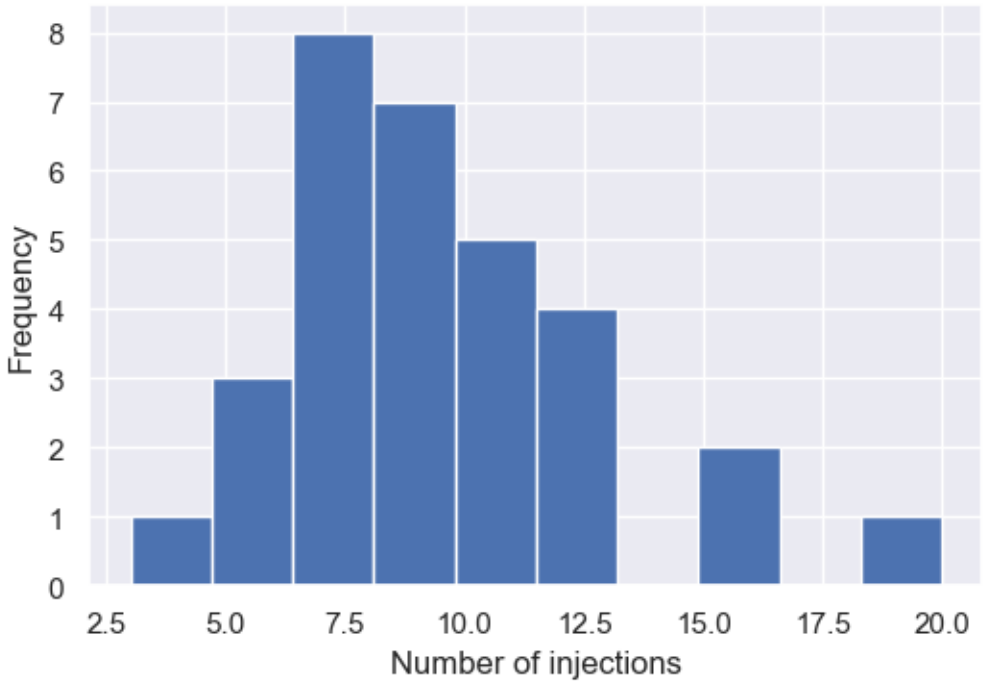


	Sensitivity	Specificity	Precision	Accuracy
poison	0.89	0.48	0.66	0.77
base	0.86	0.50	0.61	0.76

Pour passer de la classe 1 à la classe 0, l'individu 815 a besoin de rajouter 10 fois son entrée associée au label 0. On remarque par ailleurs que faire ces 10 ajouts n'a pas un impact significatif sur les différentes mesures y compris sur les prédictions associées aux autres individus. On peut ainsi considérer que l'attaque est valide.

Nous allons maintenant examiner le nombre d'ajouts nécessaires à l'obtention du prêt concernant les autres individus pour qui l'évasion n'est pas suffisante.

[17]:
```
if NEW:
    list_min = X_poison.apply(lambda x: min_pois(x[X.columns]), axis=1)
    np.savez_compressed('data/robustness/list_min.npz', data=list_min)
else:
    list_min = np.load('data/robustness/list_min.npz')['data']
    list_min = pd.Series(list_min)

list_min.plot.hist()
plt.xlabel('Number of injections')
plt.show()
```



Le graphe ci-dessus montre que l'on a une répartition assez étendue allant de 3 à 19 injections nécessaires pour changer la prédiction.

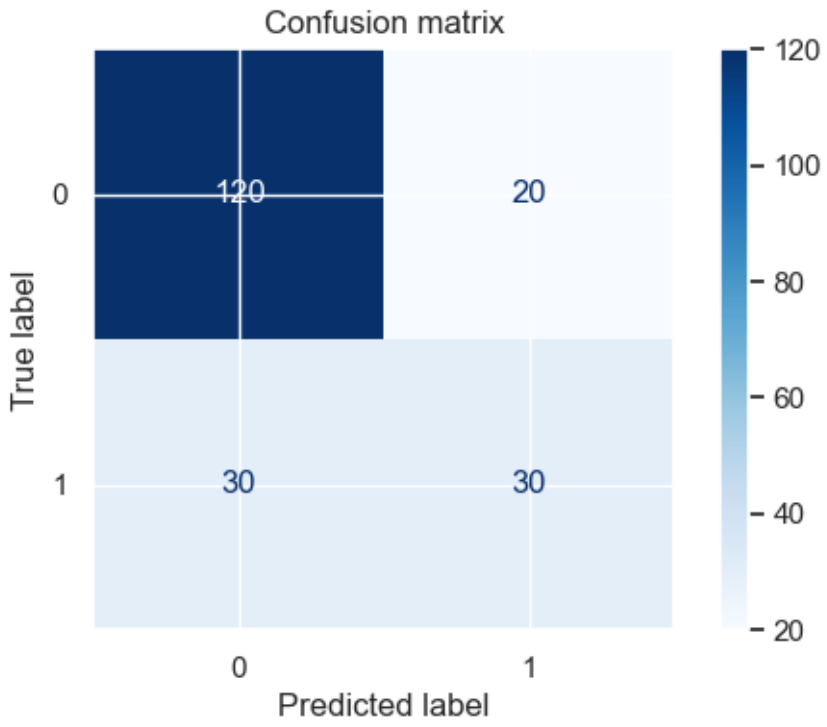
Sur la base d'entraînement qui contient 800 entrées il peut être difficile de faire passer inaperçu une dizaine de copies de la même entrée. Rien n'empêche ce dernier de coupler les attaques, il peut alors dans un premier temps effectuer une attaque d'évasion afin de faire diminuer son score puis dans un second temps produire une attaque de type poisoning afin que son score passe en dessous de 0,5. Cela permettra notamment d'utiliser moins d'injections et ainsi que l'attaque soit moins repérable.

[18]:
```
def min_pois_best_comb(adv_ind, echo=False):

    # if ind is a pd.Series we change it to a pd.Dataframe
    if isinstance(adv_ind, pd.Series):
        adv_ind = adv_ind.to_frame().transpose()

    adv_ind_all_comb = all_adv_inds[all_adv_inds.index_old == adv_ind.index[0]]
    adv_ind_best_comb = adv_ind_all_comb.sort_values('proba').head(1)[X.columns]

    n = 0
    X_train_pois = X_train
    y_train_pois = y_train

    logreg_pois = model.fit(X_train_pois, y_train_pois)
```

```
    while logreg_pois.predict_proba(adv_ind_best_comb)[0,1] > 0.5:
        n += 1
        X_train_pois = X_train_pois.append([adv_ind_best_comb],␣
↪ignore_index=True)
        y_train_pois = pd.Series(list(y_train_pois) + [0])

        logreg_pois = model.fit(X_train_pois, y_train_pois)

    if echo:
        try:
            print_diff(adv_ind.iloc[0], adv_ind_best_comb.iloc[0])
        except ValueError:
            print("Aucun changement")

        old_score = logreg.predict_proba(adv_ind_best_comb)[0,1]
        new_score = logreg_pois.predict_proba(adv_ind_best_comb)[0,1]

        print(f'{n:>2} : {old_score:.3f} --> {new_score:.3f}')
        uti.mesure_clas_list([logreg_pois, logreg], X_test, y_test,
                             ['poison', 'base'])
    return n
```

[19]:
```
_ = min_pois_best_comb(adv_ind, True)
```

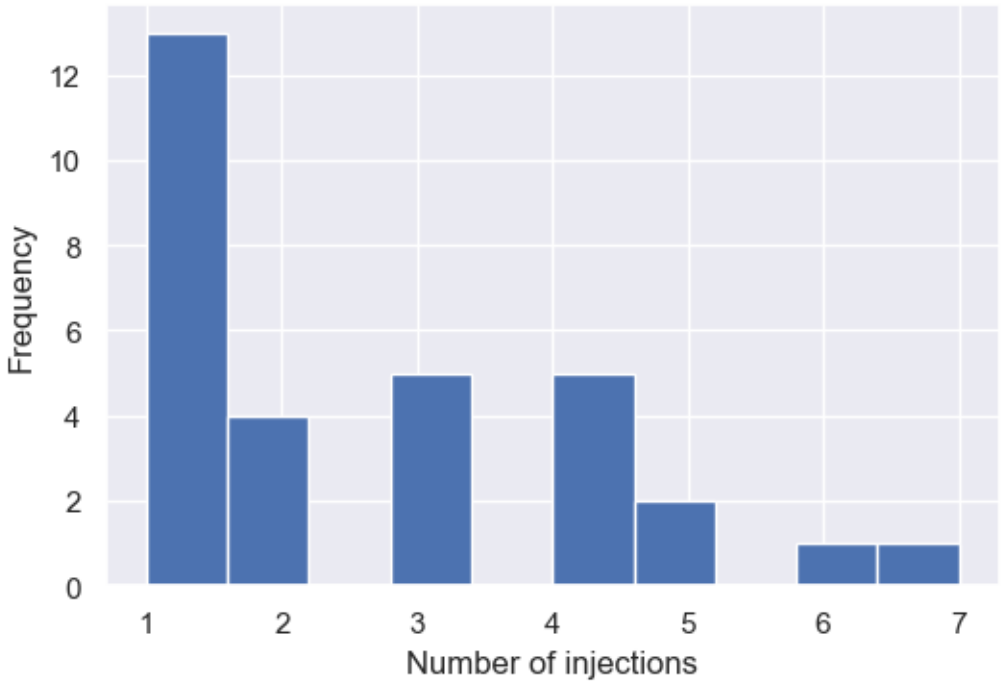
```
account_check_status : 0 <= … < 200 DM          --> no checking account
job                  : skilled employee / official --> management/ self-
employed/ highly qualified employee/ officer
 1 : 0.550 --> 0.496
```



	Sensitivity	Specificity	Precision	Accuracy
poison	0.86	0.50	0.60	0.75
base	0.86	0.50	0.61	0.76

Ce couplage d'attaque permet que l'individu 815 soit labellisé 0 avec seulement 1 ajout dans la base de données. Le nombre d'ajouts est ainsi nettement inférieur aux 10 ajouts nécessaires lorsque seule l'attaque *poisoning* est utilisée. On peut maintenant se pencher sur le cas des autres individus nécessitant une attaque de type *poisoning* (l'attaque d'évasion étant insuffisante).

[20]:
```
if NEW:
    list_min_best_comb = X_poison.apply(lambda x: min_pois_best_comb(x[X.
 ↪columns]), axis=1)
    np.savez_compressed('data/robustness/list_min_best_comb.npz',␣
 ↪data=list_min_best_comb)
else:
    list_min_best_comb = np.load('data/robustness/list_min_best_comb.
 ↪npz')['data']
    list_min_best_comb = pd.Series(list_min_best_comb)

list_min_best_comb.plot.hist()
plt.xlabel('Number of injections')
plt.show()
```



On observe alors que pour l'individu le plus difficile à aider, seulement 7 ajouts à la base d'entraînement sont nécessaires pour changer sa labellisation.

Cette étude de cas montre que si l'on ne se préoccupe pas de la *robustness* d'un modèle, les prédictions de celui-ci peuvent être aisément modifiées par un adversaire extérieur.

La faiblesse principale du modèle étudié est qu'il ne prend pas en compte la possibilité que l'agent puisse être malveillant et falsifie son entrée. Dans notre cas, la régression logistique pondère trop grandement des variables qui peuvent être facilement modifiées. Une solution possible est de retirer les variables falsifiables. Seulement, cette solution implique que les variables puissent être classées de façon binaire dans les classes falsifiables / non falsifiables ce qui n'est pas souvent le cas. De plus, retirer un trop grand nombre d'informations réduit la capacité de prédiction du modèle. Des solutions plus fines existent et sont présentées dans la partie suivante.

5.3 Mitiger les dégâts

Pour se prémunir des attaques de *robustness* on cherche à rendre notre modèle plus robuste.

Il existe trois grandes familles de méthodes de défense :

- ***Adverserial training*** : La notion d'adversaire est introduite dans le processus d'entraînement. Il existe plusieurs manières de procéder : la première consiste à ajouter des perturbations plus ou moins importantes aux données selon leur risque d'être falsifiées. Une deuxième idée est de changer l'algorithme d'entraînement en minimisant la fonction perte sur le voisinage des entrées. D'autres techniques comme l'apprentissage par groupe, l'entraînement utilisant un modèle adversaire, ma méthode ME-Net, ou encore l'entraînement

conscient des erreurs de classification ont également pour objectif de rendre un modèle robuste.

- **Utilisation de régulation** : Cette méthode consiste à limiter l'impact des perturbations des données d'entraînement sur les sorties du modèles. On utilise pour cela des bornes comme contraintes de notre fonction perte. D'autres techniques modifient le gradient lui-même pour améliorer la robustness d'un modèle (ex : Parseval, DeepDefense, TRADES).
- **Défense certifié** : Contrairement aux deux autres méthodes de défense, celle-ci propose des garanties statistiques sur la *robustness* du modèle.

Il existe très peu de librairies python différentes permettant de mettre en place ces défenses. Les principales sont :

- *Adverserial Robustness Toolbox* (ART) : développée par IBM en 2019, elle vise à évaluer et défendre contre l'évasion, l'empoisonnement, l'extraction et l'exploration.
- *AugLy* : développer par Baidu en 2020 a les mêmes objectifs.

Nous utiliserons par la suite l'ART d'IBM.

5.3.1 Présentation des différentes solutions

Contrer l'évasion On suppose que les hypothèses qui ont permis l'attaque restent inchangées. (Celles-ci seront remises en cause dans la dernière partie.)

On cherche à éviter qu'un agent puissent prendre avantage de la falsification de son entrée, pour cela nous allons utiliser une méthode d'*Adverserial training*.

Le principe de cet entraînement est de rendre notre modèle robuste à des perturbations exercées sur les entrées. En reprenant les notations de la partie *Généralisation de l'attaque*, l'entraînement du modèle sur un couple (x, y) consiste à minimiser la fonction suivante :

$$\min_{\theta \in \Theta} \mathbb{E}_\delta \ell(h_\theta(x + \delta), y)$$

On rappelle que Δ correspond à l'ensemble des perturbations permises, ℓ la fonction de perte, h la fonction de notre modèle, θ les paramètres du modèle, x l'entrée et y la classe de l'entrée. On cherche ainsi à minimiser l'espérance de la fonction perte sur l'entrée bruitée $x + \delta$.

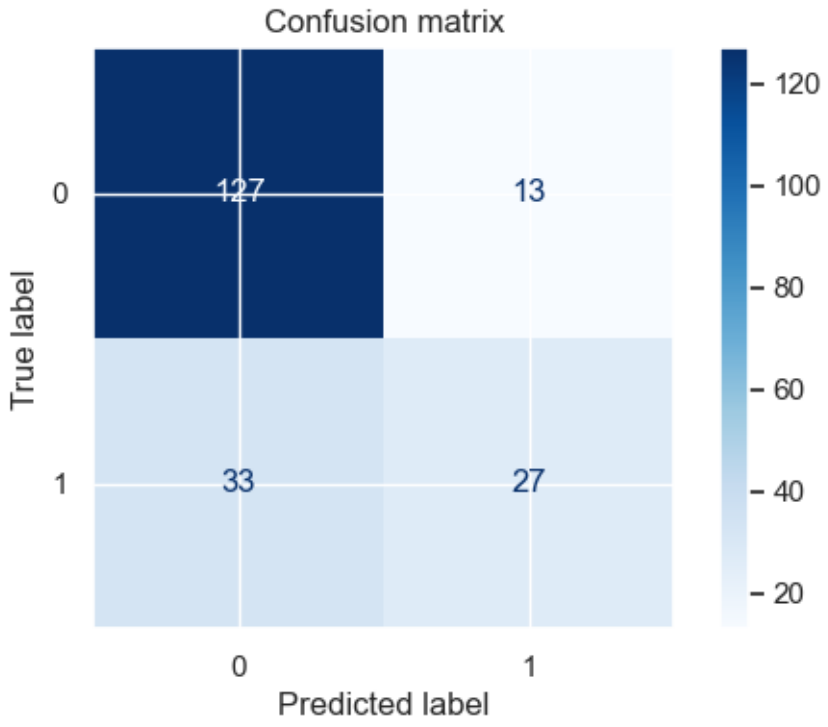
En pratique, on va approximer cette espérance pour entraîner le modèle en modifiant directement notre base de données. Si on avait accès à la méthode d'entraînement, on aurait pu implémenter la méthode de tirage aléatoire au moment où l'entrée est sélectionnée mais comme ce n'est pas le cas, nous implémentons une autre méthode qui consiste à modifier la base de données, méthode qui ne sera pas détaillée dans ce qui suit.

[21]:
```
var_columns = ['account_check_status', 'job']
type_col = {'high': ['purpose', 'personal_status', 'account_check_status',␣
 ↪'job']}

alt_inds = gen_all_ind(pd.concat([X_train, y_train], axis=1), var_columns)

# Calcul de l'indicatrice
same = []
for i, group in list(alt_inds.groupby('index_old')):
    same.append(pd.concat([group[col] == X_train.loc[i][col]
```

```
                            for col in var_columns], axis=1))
same = pd.concat(same, axis=0)
```

```
800it [00:24, 32.25it/s]
```

[22]:
```
# Choice the coefficient w_i and give it to each columns
coef = {'high': 0.4}
col_coef = {col: coef[key] for key in type_col.keys() for col in type_col[key]}

# Calculate the weigth for every entries
weight_list = ([(col_coef[col]/X[col].unique().size +
                 same[col]*(1 - col_coef[col])) for col in var_columns])
weight = weight_list[0]
for w in weight_list[1:]:
    weight *= w

logreg_robust = model.fit(alt_inds[X.columns], alt_inds.default,␣
↪logreg__sample_weight=weight)
uti.mesure_clas_list([logreg_robust, logreg], X_test, y_test, ['robust',␣
↪'base'])
```



	Sensitivity	Specificity	Precision	Accuracy
robust	0.91	0.45	0.68	0.77
base	0.86	0.50	0.61	0.76

La matrice de confusion et les différentes métriques montrent que le nouveau modèle a des caractéristiques très proches du modèle initial. Nous cherchons à voir s'il est plus robuste que le modèle initial. Pour cela, on va calculer le nombre de personnes qui peuvent modifier leur classe (de 1 vers 0) en falsifiant les variables définies comme facilement falsifiables.

[23]:
```
# Calculate the old probability without falsification and keep those that␣
↪predict 1
old_proba = pd.Series(logreg_robust.predict_proba(X_test)[:, 1],
                      name='old_proba',
                      index=X_test.index)
X_default = pd.concat([X_test, old_proba], axis=1)[old_proba > 0.5]

# Generate all possible falsification and calculate probability for each
alt_inds_default = gen_all_ind(X_default, var_columns)
alt_inds_default['proba'] = logreg_robust.predict_proba(alt_inds_default[X.
↪columns])[:, 1]

# For each individual get the minimum probability attainable with falsification
proba_min = alt_inds_default.groupby('index_old').apply(lambda df:␣
↪min_proba(df))
X_default = X_default.assign(proba_min = proba_min.droplevel(1))

print('Default label : {}'.format(len(X_default)))
print('Change to no-default : {}'.format(len(X_default[X_default.proba_min < 0.
↪5])))
```

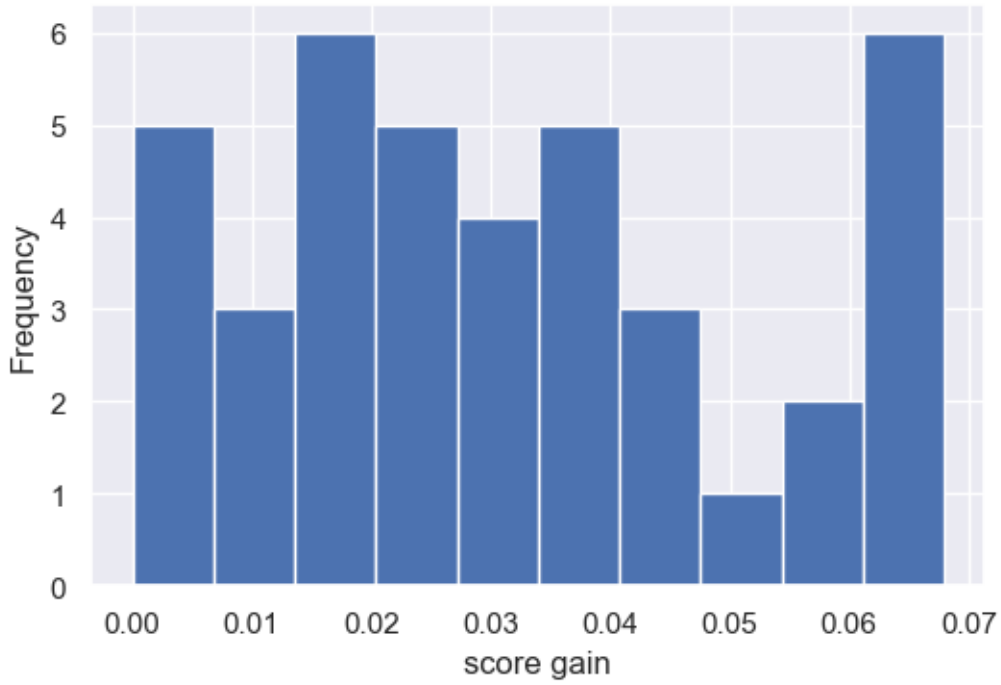
```
40it [00:01, 30.48it/s]
```

```
Default label : 40
Change to no-default : 7
```

En associant une pondération de 0.4 à nos variables falsifiables et de 0 aux variables non falsifiables, on obtient après entraînement du modèle que seulement 7 individus peuvent changer de classe sur 40. En comparaison, le modèle initial autorisait 44 personnes sur 49 à modifier leur classe. On a donc un modèle bien plus robuste que le modèle initial.

Afin de rentrer dans les détails, on peut regarder l'impact d'attaques par évasion sur le score des individus.

[24]:
```
(X_default.old_proba - X_default.proba_min).plot.hist()
plt.ylabel('Frequency')
plt.xlabel('score gain')
plt.show()
```



Pour le modèle classique, on avait en moyenne une baisse de 0.4 sur le score d'un individu après une attaque par évasion. Dans le cas du nouveau modèle cette moyenne est divisée par 10 et le score des individus varie très peu lors de ce type d'attaque. Les variables falsifiables ont ainsi beaucoup moins d'impact sur la prédiction du modèle et ce dernier est devenu robuste contre les attaques d'évasion.

On peut maintenant étudier l'influence du coefficient de pondération associé (que l'on a précédemment fixé à 0.4) aux variables falsifiables sur la robustness du modèle.

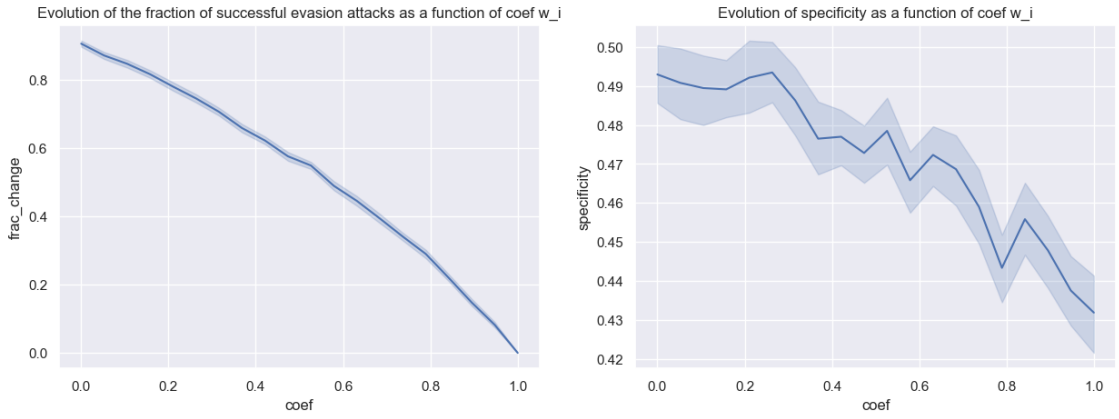
[25]:
```
def gen_reg_robust(proba=0.4, inds=alt_inds, X=X):
    # The coefficient w_i
    coef = {'high': proba}
    col_coef = {col: coef[key] for key in type_col.keys() for col in␣
 ↪type_col[key]}

    # Calculate the weigth for every entries
    weight_list = ([(col_coef[col]/X[col].unique().size +
                     same.loc[sub_alt_inds.index][col]*(1 - col_coef[col]))
                                                        for col in var_columns])

    weight = weight_list[0]
    for w in weight_list[1:]:
        weight *= w

    logreg_robust = model.fit(inds[X.columns], inds.default,
```

```
                                logreg__sample_weight=weight)

    # Keeping metrics on the model to see how the coefficient affects the␣
↪classification
    y_pred = logreg_robust.predict(X_test)
    metrics = classification_report(y_test,
                                    y_pred,
                                    output_dict=True)

    # Calculate the old probability without falsification and keep the 1s
    old_proba = pd.Series(logreg_robust.predict_proba(X_test)[:, 1],
                          name='old_proba',
                          index=X_test.index)
    X_default = pd.concat([X_test, old_proba], axis=1)[old_proba > 0.5]

    # Generate all possible falsification and calculate probability for each
    alt_inds_default = gen_all_ind(X_default, var_columns)
    alt_inds_default['proba'] = logreg_robust.predict_proba(alt_inds_default[X.
↪columns])[:, 1]

    # For each individual get the minimum probability attainable with␣
↪falsification
    proba_min = alt_inds_default.groupby('index_old').apply(lambda df:␣
↪min_proba(df))
    X_default = X_default.assign(proba_min = proba_min.droplevel(1))

    return metrics, len(X_default), len(X_default[X_default.proba_min < 0.5])
```

[26]:
```
n_same = 100
columns=['coef', 'n_ones', 'n_change',
         'accuracy', 'sensitivity',
         'specificity', 'precision']

if NEW:
    df_res = pd.DataFrame(columns=columns)

    for proba in np.linspace(0, 1, 20):
        for _ in range(n_same):
            idx_ind = np.random.choice(alt_inds.index_old.unique(), 600,
                                       replace=False)
            idx = alt_inds.apply(lambda x: x.index_old in idx_ind, axis=1)
            sub_alt_inds = alt_inds[idx]
            try:
                metrics, n_ones, n_change = gen_reg_robust(proba, sub_alt_inds,
                                                           X.loc[idx_ind])
                res = {'coef': proba,
                       'n_ones': n_ones,
```

```
                        'n_change': n_change,
                        'accuracy': metrics['accuracy'],
                        'sensitivity': metrics['0']['recall'],
                        'specificity': metrics['1']['recall'],
                        'precision': metrics['1']['precision']}

                df_res = df_res.append(res, ignore_index=True)
            except ValueError:
                print('error')

    np.savez_compressed(f'data/robustness/data_res.npz', df_res=df_res)

else:
    file = np.load(f'data/robustness/data_res.npz', allow_pickle=True)
    df_res = pd.DataFrame(file['df_res'], columns=columns)
```

[27]:
```
df_res['frac_change'] = df_res['n_change']/df_res['n_ones']

fig, axes = plt.subplots(1, 2, figsize=(15, 5))

sns.lineplot(x='coef', y='frac_change', data=df_res, ax=axes[0])
sns.lineplot(x='coef', y='specificity', data=df_res, ax=axes[1])

axes[0].set_title('Evolution of the fraction of successful evasion attacks as a␣
↪function of coef w_i')
axes[1].set_title('Evolution of specificity as a function of coef w_i')

plt.show()
```



Le premier graphique représente l'évolution de la fraction de la population de la classe 1 (dont le prêt est non accepté) capable de changer de classe à l'aide d'une attaque de type évasion en fonction du coefficient de pondération associé aux variables falsifiables. On remarque que lorsque

ce coefficient est de 0, on retrouve le modèle initial pour lequel 44 personnes sur 49 pouvaient changer de classe. De plus, plus le coefficient de pondération est grand, moins il y a de personnes capables de changer de classe à l'aide d'une attaque de type évasion. Lorsque le coefficient de pondération vaut 1, le modèle fait abstraction aux variables falsifiables et le nombre de personnes capables de changer de classe est nul.

Le deuxième graphique illustre l'évolution de la métrique `specificity` en fonction du coefficient de pondération. Cette métrique est l'une des mesures utilisée pour calculer les performances du modèle étudié. Plus précisément, la `specificity` correspond à la fraction de labels 0 prédits correctement, dans notre cas la fraction de personnes ayant remboursé leur prêt et qui sont prédit comme capable de rembourser, sur la totalité des personnes qui sont prédit comme capables de rembourser. Ce deuxième graphique illustre ainsi le fait que les performances du modèle diminuent avec l'augmentation du coefficient de pondération.

Le premier graphique démontre ainsi l'efficacité de la méthode visant à contrer les attaques d'évasion lorsque que le second illustre la décroissance de performances liée à l'utilisation de cette méthode.

Il y a alors un compromis à faire entre performance et robustesse.

5.4 Critiques et conlusion

5.4.1 Les libraires pour la stabilité des modèles statistiques ne sont pas assez matures.

Les différentes implémentations des attaques et des méthodes de robustness (cf partie I et II) ont permis de constater certaines lacunes des différentes librairies existantes. Aucune des méthodes mises en œuvre dans la deuxième partie ne proviennent des librairies et ce pour deux raisons. La première est pédagogique, voir le détail de l'implémentation permet de comprendre les idées de celle-ci. La deuxième est le manque de méthodes correspondant à mon problème, un modèle de régression logistique avec des variables catégorielles. Cela peut s'expliquer par la présence de seulement deux librairies sur la stabilité des modèles : *Adverserial Robustness Toolbox* (ART), développé par IBM en 2019 et *AdvBox* développé par Baidu en 2020. Avec une majorité des exemples et de la documentation de *AdvBox* en chinois. Pour ce qui est des exemples de ART ils sont nombreux pour les différentes attaques mais plus rares pour les méthodes de défenses. À cela s'ajoute le manque d'échange sur les sites comme *stack overflow* avec seulement trois questions pour ART et aucune pour *AdvBox*. Pour espérer une adoption des méthodes de stabilité, il faut continuer le développement des librairies existantes. ART, la librairie plus avancée, rassemble dans un seul répertoire tout ce qu'il faut pour attaquer et défendre plusieurs aspects d'un modèle. Un très grand nombre d'outils, sans une méthode pour choisir les plus pertinentes, risque de décourager l'implémentation. Lorsque le besoin est clairement identifié et que l'on sait quelle méthode on va utiliser et que celle-ci est implémentée, il est plutôt facile de la mettre en œuvre. Pour pallier ces problèmes et continuer de faciliter la tâche des développeurs, il faut continuer à investir du temps et de l'argent dans les outils pour couvrir le plus de cas possible mais en se concentrant sur les méthodes d'explication et de choix des modèles. Pour choisir au mieux les modèles on peut investir dans des méta analyses ou bien des études de cas dans les exemples des librairies qui soulignent les avantages et inconvénients des méthodes.

5.4.2 Les libraires de stabilité sont à jour et bien documentés mais ne sont pas prête pour des cas industrielles.

On retrouve les inconvénients des secteurs de recherche naissants qui n'ont pas encore quitté le monde académique. Les documentations sont pensées pour des développeurs ayant déjà une idée précise des enjeux, des méthodes et des solutions de la stabilité. Contrairement à la plupart des libraires utilisées dans ce rapport, l'ART contient des exemples d'implémentation très méthodiques et bien détaillés. Mais ils ne sont qu'une quinzaine (sur plus d'une centaine de méthodes) et ces exemples ne proposent pas d'explication sur les raisons du choix des méthodes présentées. On note tout de même que la stabilité est une problématique qui se visualise aisément dans certains cas comme la vision et il existe des outils comme ArtDemo qui permettent de comprendre visuellement les enjeux. Mais comme pour la *privacy* ces visualisations et la plupart des méthodes se concentrent sur des modèles avancés et complexes. La majorité des modèles utilisés en entreprise dont les enjeux de la stabilité sont importants, comme les modèles de pré-tris pour embauches, sont soit des modèles simples, soit des modèles plus complexes mais directement implémentés par des outils d'autoML. Pour une adoption des méthodes et les rendre accessibles aux entreprises il faut des implémentations directement dans ces outils et se concentrer sur des solutions pour des modèles simples. L'impact d'une méthode sera d'autant plus importante qu'elle peut être mise en place facilement dans le monde réel. Il serait idéale de voir une appropriation industrielle du sujet avec la mise en place de produits simples, ceci couplé à une législation sur la notion de stabilité lorsque les modèles statistiques sont implémentés dans des domaines sensibles. Comme toujours la production de communication non scientifique pousse également à la discussion et à sensibiliser les industriels et les utilisateurs sur le sujet.

5.4.3 Il existe un arbitrage important entre performance et stabilité

On a vu que la stabilité algorithmique requiert l'implémentation de méthodes autour du modèle choisie et qu'un modèle classique n'est pas robuste face à des exemples adverses. L'ajout de ces méthodes est donc nécessaire mais impactent les performances du modèle. Dans notre exemple de régression logistique l'augmentation de la stabilité n'a pas engendré une baisse drastique de l'*accuracy* mais des publications comme [3] montrent qu'il existe une tension entre ces deux variables. De plus, l'ajout de méthodes augmente le temps d'entraînement et d'utilisation d'un modèle. Dans notre exemple, le temps d'entraînement du modèle sans stabilité est de 50ms et 38 des 40 personnes qui ont refusé peuvent modifier la prédiction. Lorsque l'on implémente notre méthode le temps d'entraînement passe à 6s mais cette fois-ci seulement 7 des 40 personnes on pu modifier leur prédiction. L'inconvénient est qu'il n'y a pas de document non scientifique qui explique ces arbitrages et aide à la prise de décision. L'idéal serait des tableaux de comparaison entre les méthodes pour comprendre les différences entre chaque méthode. Cela passe également par la prise en main des outils et une adoption par les développeurs pour avoir un retour sur ce qui est important et ce sur quoi on peut faire des compromis avec la performance. On peut également imaginer que dans les communications scientifiques ces questions de compromis soient abordées plus régulièrement.

6 Bibliographie

6.1 Introduction

[1] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science.

[2] Jessica Morley and Luciano Floridi and Libby Kinsey and Anat Elhalal (2019). From What to How. An Overview of AI Ethics Tools, Methods and Research to Translate Principles into Practices. CoRR, abs/1905.06876.

[3] Jessica Morley and Anat Elhalal and Francesca Garcia and Libby Kinsey and Jakob Mökander and Luciano Floridi (2021). Ethics as a service: a pragmatic operationalisation of AI Ethics. CoRR, abs/2102.09364.

[4] S. Majumdar. Fairness, Explainability, Privacy, and Robustness for Trustworthy Algorithmic Decision Making, In Big Data Analytics in Chemoinformatics and Bioinformatics, published by Elsevier

6.2 Fairness

[5] Jon M. Kleinberg and Sendhil Mullainathan and Manish Raghavan (2016). Inherent Trade-Offs in the Fair Determination of Risk Scores. CoRR, abs/1609.05807.

[6] Mitchell, S., Potash, E., Barocas, S., D'Amour, A., & Lum, K. (2021). Algorithmic Fairness: Choices, Assumptions, and Definitions. Annual Review of Statistics and Its Application, 8(1), 141-163.

[7] Sina Fazelpour and Zachary C. Lipton (2020). Algorithmic Fairness from a Non-ideal Perspective. CoRR, abs/2001.09773.

[8] Rachel K. E. Bellamy and Kuntal Dey and al. (2018). AI Fairness 360: An Extensible Toolkit for Detecting, Understanding, and Mitigating Unwanted Algorithmic Bias. CoRR, abs/1810.01943.

[9] Kamiran, F., Calders, T. (2012) Data preprocessing techniques for classification without discrimination. https://doi.org/10.1007/s10115-011-0463-8

6.3 Privacy

[10] Emiliano De Cristofaro. (2020). An Overview of Privacy in Machine Learning.

[11] Mohammad Al-Rubaie and J. Morris Chang (2018). Privacy Preserving Machine Learning: Threats and Solutions. CoRR, abs/1804.11238.

[12] Naoise Holohan and Stefano Braghin and P'ol Mac Aonghusa and Killian Levacher (2019). Diffprivlib: The IBM Differential Privacy Library. CoRR, abs/1907.02444.

6.4 Robustness

[13] Muhammad Shafique and Mahum Naseer and Theocharis Theocharides and Christos Kyrkou and Onur Mutlu and Lois Orosa and Jungwook Choi (2021). Robust Machine Learning Systems: Challenges, Current Trends, Perspectives, and the Road Ahead. CoRR, abs/2101.02559.

[14] Matthew Jagielski and Alina Oprea and Battista Biggio and Chang Liu and Cristina Nita-Rotaru and Bo Li (2018). Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. CoRR, abs/1804.00308.

[15] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, & Aleksander Madry. (2019). Robustness May Be at Odds with Accuracy.

[16] Anirban Chakraborty and Manaar Alam and Vishal Dey and Anupam Chattopadhyay and Debdeep Mukhopadhyay (2018). Adversarial Attacks and Defences: A Survey. CoRR, abs/1810.00069.