German Credit

Rapport pour le cours

« Apprentissage, réseaux de neurones et

modèles graphiques » (RCP 209)

PHELIZOT Yvan

Juillet 2017

Table des matières

Introduction

Le « credit scoring » est un outil financier d'aide à la décision permettant d'évaluer la « solvabilité » d'un emprunteur. Cette pratique s'applique aussi bien aux particuliers, aux entreprises qu'à des portefeuilles de gestion d'actif.

En fonction de la réponse, il est possible de déterminer le risque associé à l'emprunteur qu'il rembourse ou pas le crédit.

L'évaluation du risque de crédit repose énormément sur le « Big Data ». Elle combine de nombreux facteurs pour déterminer de manière la plus précise possible le résultat.

Le but de ce rapport est de présenter la méthodologie mise en place pour déterminer ce risque financier en se basant sur le jeu de données « German Credit Data ». À partir des attributs données, il va être nécessaire de donner une réponse sur le fait d'accorder ou pas un crédit. Il s'agit donc d'un problème de classification constitué de deux classes : « crédit accordé » et « crédit non accordé ».

Il est important de commencer par une pré-analyse du jeu fourni afin de déterminer ces caractéristiques et sa qualité (données manquantes, insuffisantes, …). Cela permettra notamment de réduire les dimensions du problème en supprimant les attributs non significatifs.

À partir du jeu de données traitée, différentes familles d'outils seront utilisées. Pour chacune d'elles, plusieurs modèles seront créés. Les meilleurs modèles obtenus seront ensuite améliorés pour obtenir le meilleur résultat possible.

Les développements sont réalisés en **Python** avec les librairies **Numpy**, **scikit-learn**, **Panda** (qui facilite la manipulation des données), **Matplotlib** et **Seaborn** pour la représentation graphique.

Les différentes sources sont disponibles sur : **https://github.com/cotonne/rcp209**

I. Exploration des données

I.1. Présentation du jeu de données

Les données utilisées sont issues du jeu de données « German Credit », librement accessible depuis le site « UCI Machine Learning Repository » (https://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)). Deux fichiers sont proposés.

Le premier fichier est constitué de 1000 entrées. C'est un jeu assez faible en comparaison de volumes de données plus conséquents auxquels on peut faire face (plusieurs Téraoctets de données par exemple).

Un exemple d'entrée :

A11 6 A34 A43 1169 A65 A75 4 A93 A101 4 A121 67 A143 A152 2 A173 1 A192 A201 1

Chaque entrée est composée de 21 attributs différents, aussi bien catégoriels que numériques (discrète et continue).

Le second fichier est dérivé du premier, les données catégorielles ayant été encodées sous forme de nombre. Les attributs sont décrits dans le fichier « german.doc ».

Les 20 premiers attributs peuvent être décomposés en plusieurs familles :

- Données relatives au prêt (durée, montant, but, …)
- Solvabilité de l'emprunteur (Patrimoine, présence de garant, ...)
- Informations sur l'emprunteur (age, travailleur étranger, situation professionnelle, …)

Plusieurs attributs sont discrétisés (par exemple, l'attribut n°6 « Savings account/bonds »). Il aurait pu être intéressant de disposer de la valeur précise de ces informations, quitte à effectuer une discrétisation peut-être plus adaptée par la suite

Le dernier attribut correspond à la décision prise par la banque d'attribuer ou non le crédit. On comptabilise 700 accords et 300 refus.

Nous pouvons remarquer que certaines données couramment utilisées ne sont pas présentes (par exemple, le taux d'endettement ou le salaire).

I.2. Analyse univariée des données

L'analyse univariée va nous permettre de comprendre les données que nous avons et leur "forme". Cela nous aidera à détecter des informations insuffisamment représentées ou, au contraire, sur-représentées.

I.2.1. Analyse statistique

Attributs numériques

Le tableau suivant présente les propriétés statistiques des attributs numériques :

	Duration in month	*Credit amount*	*Installment rate*	*Present residence rate*	*Age*	*Number of existing credits*	*Nb of liable people*
Min	4	250	1	1	19	1	1
<= 10%	9	1066.2	1	1	23	1	1
Moyenne	20.9	3271.26	2.97	2.85	35.55	1.41	1.16
Médian	18	2319.5	3	3	33	1	1
Écart-type	12.05	2821.32	1.12	1.1	11.37	0.58	0.36
<= 90%	36	6306.7	4	4	52	2	2
Max	72	18424	4	4	75	4	2

Nous pouvons en en déduire que l'emprunteur moyen :

- veut un crédit d'une durée de 1 à 3 ans pour 1066 à 6306 DM (soit de 545€ à 3224€)
- est actif entre 23 ans et 52 ans
- qu'il a déjà un ou deux crédits
- qu'il a au moins un garant

Il pourrait être intéressant de comparer ces informations avec des données plus globales, comme des données démographiques. Par exemple, l'age moyen en Allemagne est actuellement de 44 ans. En comparaison de l'âge de l'échantillon, il n'est pas forcément représentatif de la population allemande.

Attributs catégoriels

Les données sont représentées sous forme de tables de contingences. Ces données nous serviront pour les calculs suivants(χ^2, distribution relative, ...)

La représentation graphique de ces résultats sous graphique de distribution relative est présentée en annexes.

I.2.2. Représentation des données

Représentation fréquentielle

La représentation fréquentielle donne une vision de la manière dont les données sont réparties sur leur ensemble de définition.

Nous voyons ainsi distinctement que l'âge des demandeurs est compris entre 25 et 45 ans et qu'il diminue rapidement.

De même, pour le montant demandé, on voit clairement qu'il est en grande majorité inférieur à 4000 DM et qu'il y a peu de demandes pour des valeurs supérieures.

I.3. Réduction des dimensions

Un des problèmes récurrents dans le domaine du **Machine Learning** est le problème de dimensionnalité (*Curse of Dimensionality*). Quand le nombre de dimensions commence à être élevé, c'est-à-dire quand le nombre de variables du jeu de test et d'apprentissage est important, les données risquent d'être « éparpillées » dans le domaine, rendant difficile la fiabilité de la prédiction. Une solution est de fournir un volume important de données en entrée. Cependant, nous n'avons pas accès à plus de données que celles du fichier d'entrée.

Une autre possibilité est de réduire les attributs des données d'entrée. C'est l'approche que nous prendrons.

I.3.1. Distribution relative

Une première méthode simple et graphique de voir l'influence des variables d'entrées sur le résultat est l'affichage sous forme d'un graphique de distribution relative.

Les graphiques des distributions sont présentés en annexe.

Certaines variables ont une influence sur l'acceptation du crédit (le volume relatif change en fonction de la valeur de l'attribut) :

- Status of checking account
- Credit history
- Purpose
- Savings account
- Guarantors
- Property
- Personal status and sex
- Duration in month
- Age
- Credit amount

I.3.2. Analyse en composantes principales

L'analyse en composantes principales est une technique permettant de reprojeter les données selon des axes qui expliquent au mieux la variance. Cette transformation aide à simplifier les données en entrée en offrant des surfaces de séparation des ensembles plus claires.

Avant de traiter les données via une analyse en composantes principales, ces dernières doivent standardiser. Pour ce faire, nous utiliserons la classe « **Criminaliser** » de scikit-learn. Nous utilisons la classe « **PCA** » pour effectuer l'analyse.

Le graphique suivant présente la variance expliquée par composante ainsi que la variance cumulée :

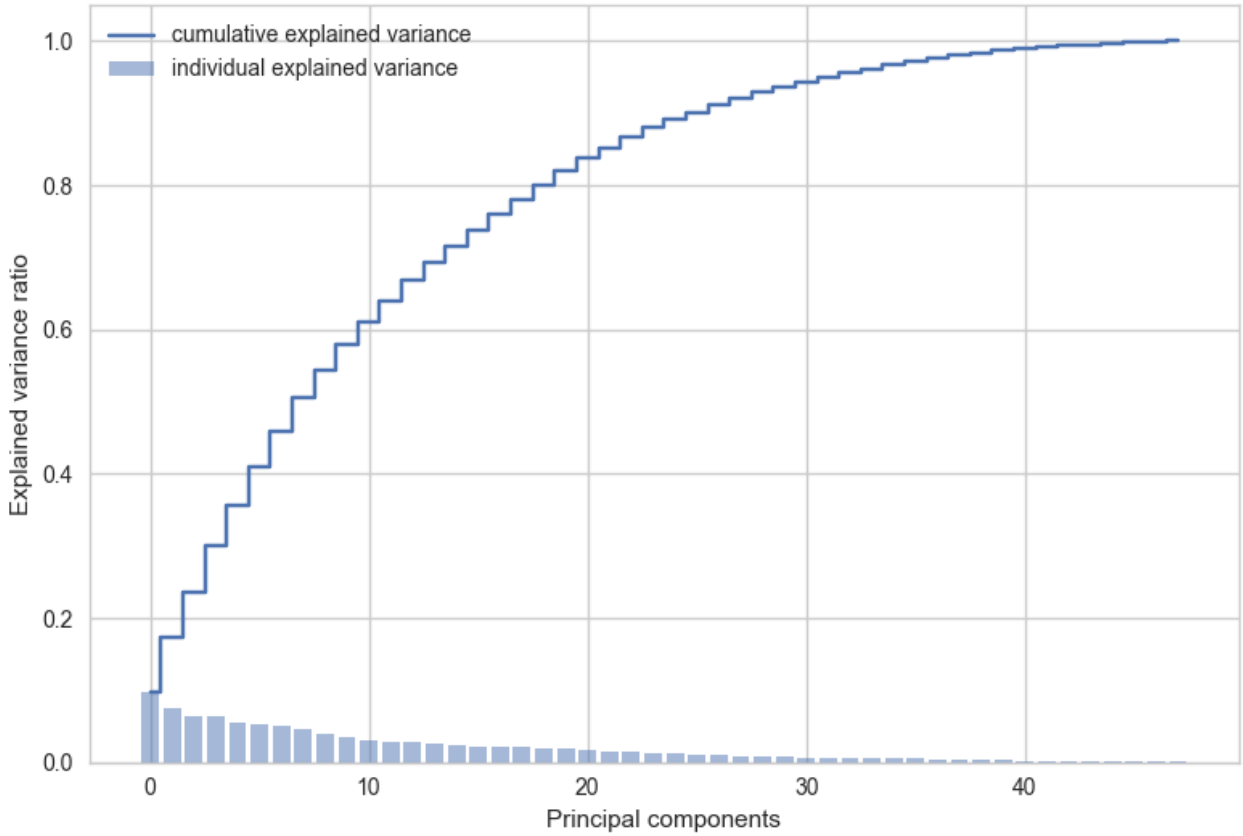


Illustration 1: Graphique cumulée des variances expliquées par les composants de l'ACP

On constate que la majorité des composants est nécessaire pour obtenir une variance expliquée suffisante (>95%). Il ne paraît pas intéressant de faire une projection des données via une analyse en composantes principales.

En conservant les deux premiers composantes qui expliquent moins de 20 % de la variance, nous obtenons le résultat suivant :

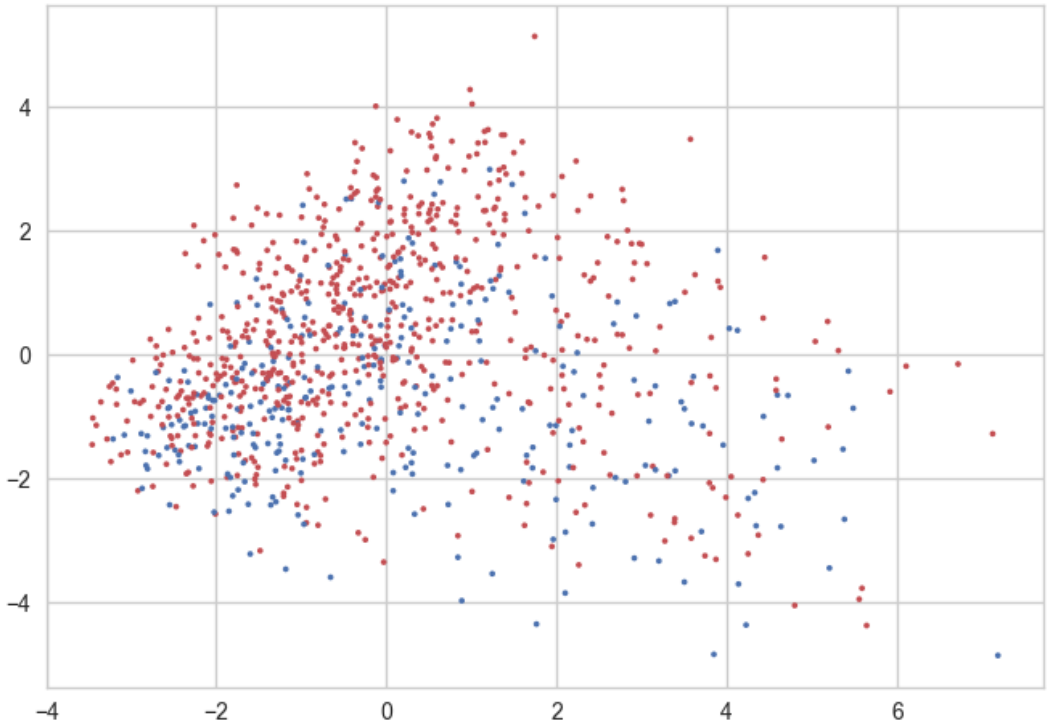


Illustration 2: Projection des entrées sur les deux principales composantes (en rouge, les acceptations de crédit)

Nous observons que, effectivement, la répartition n'est pas homogène.

I.3.3. Test indépendance du χ^2

Le test du χ^2 sur les différents paramètres va nous aider à déterminer les paramètres ayant une influence sur l'accord du prêt.

Il est maintenant nécessaire de vérifier les conditions de validité du test. Il faut que toutes les classes ont une valeur non nulle et que 80% des classes doivent avoir une valeur supérieur ou égale à 5, ce qui est le cas.

Nous posons les hypothèses suivantes :

- H0 : L'attribut « X » et la valeur de sortie « Credit Score » sont indépendants
- H : L'attribut « X » et la valeur de sortie « Crédit Score » ne sont pas indépendants

Nous calculons les valeurs-p. L'hypothèse « H0 » sera rejetée si $p \leq 0.05$ avec un risque d'erreur égale à 5 %. Le tableau suivant synthétise les différentes résultats. En rouge, les attributs qui ont une influence significative sur le résultat.

Features	**Status of checking account**	**Credit history**	**Purpose**	**Savings account**	**Present employment since**
p-values	**< 0.01**	**< 0.01**	0.94	**< 0.01**	**< 0.01**

Features	**Personal status and sex**	**Guarantors**	**Property**	**Other installment plans**	**Housing**
p-values	0.13	0.31	**< 0.01**	0.06	0.73
Features	**Job**	**Telephone**	**Foreigner**	**Duration in month**	**Credit amount**
p-values	0.62	0.37	**0.01**	**0.01**	0.06
Features	**Installment rate**	**Present residence since**	**Age**	**Number of existing credits**	**Number of liable people**
p-values	0.14	0.95	0.28	0.44	0.93

Les valeurs non rouge peuvent être retirées du jeu de données car elles n'ont pas une influence significative sur l'acceptation par la banque du prêt. Nous constatons aussi que ces résultats confirment l'analyse issue de l'observation des distributions relatives.

I.3.4. t-SNE

Nous pouvons utiliser un outil d'apprentissage non-supervisé non-linéaire. Scikit-learn fournit une classe « **TNSE** » permettant d'effectuer une « t-Stochastic Neighbor Embedding ». C'est généralement une technique efficace de réduction de dimension servant à transformer les données dans un espace de dimension 2 ou 3. Cette technique s'appuie sur la distribution statistique de la distance entre pairs afin les entrées proches aient une probabilité élevée. Ensuite, il cherche à minimiser la divergence de Kullback-Leibler entre les distributions par rapport à leur position sur la carte. Après exécution, nous obtenons le résultat suivant :

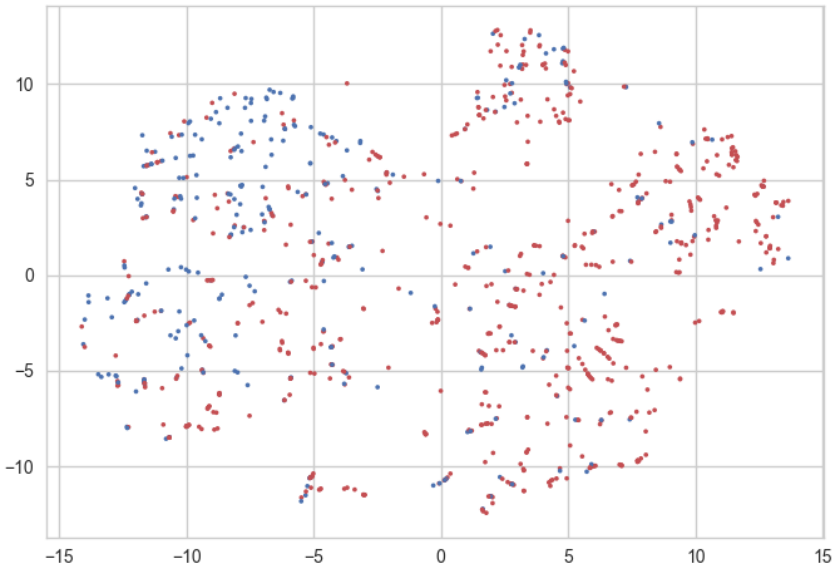


Illustration 3: Projection des données après t-SNE

Nous pouvons voir que les données sont un peu séparées. Globalement les réponses négatives se trouvent sur la gauche, en bleu, les réponses positives en rouge. Cependant, les données ne présentent pas de répartition significative.

I.3.5. Analyse bivariée des données

Il est aussi possible de calculer la corrélation entre les données. Par exemple, des données trop fortement corrélées entre elles n'apportent pas plus d'informations. Il suffit de conserver une des variables tout en ayant une variance équivalente.

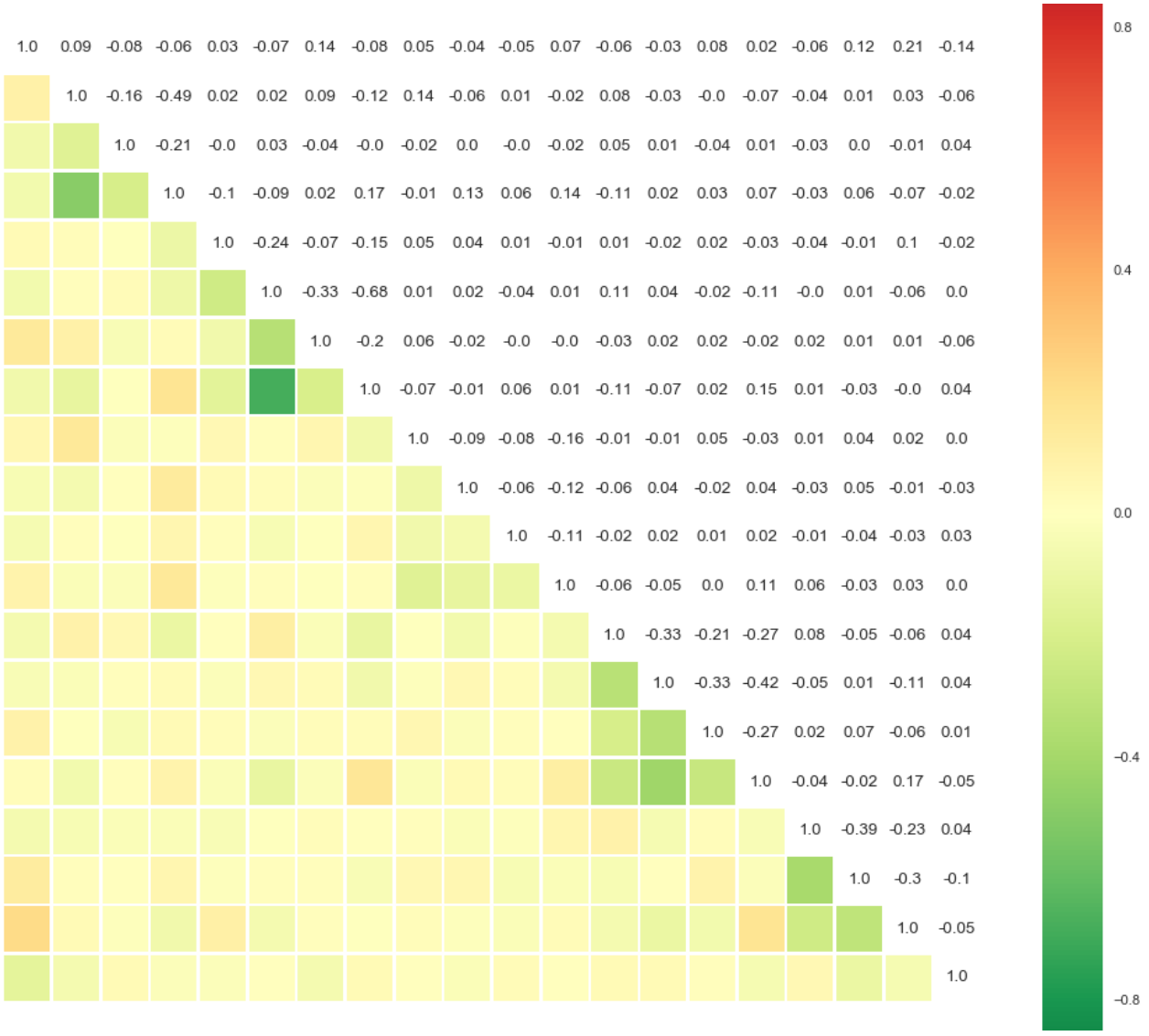


Illustration 4: Matrice croisée des correlations

Le graphique nous montre une "carte de chaleur" des corrélations. Il a été déterminé en conservant uniquement les variables significatives telles que déterminées via le χ^2. Les données catégorielles sont ensuite transformées via le "one-hot encoder".

La corrélation la plus importante est de 0.68 pour 'Credit history_A33' et 'Credit history_A34'. Nous supposons que cette forte corrélation peut être due au hasard.

Nous conservons donc les attributs tels quels.

II. Présentation de la méthode

II.1. Traitement des données

Nous avons vu que certaines attributs avaient une influence importante. La taille du jeu de données est très faible. Au vu de la taille du jeu de test, nous pouvons tenter de prédire les données en conservant tout ou partie des attributs les plus significatifs afin de constater l'impact du sous-ensemble d'attributs sur la prédiction.

Il est possible, dans certains cas, d'augmenter le jeu de données de données. Par exemple, nous pourrions créer des entrées supplémentaires dans les données du jeu. Cette technique est très efficace dans le domaine de l'audio ou de l'image. Cependant, dans notre cas, elle semble difficile à mettre en œuvre.

Il pourrait être possible de dériver de nouvelles informations à partir du jeu de données existantes. Par exemple, dans le cas « Personal status and sex », nous pourrions séparer le sexe du statut. Cependant, toutes les combinaisons (statuts, sexe) ne sont pas représentées, ce qui ne permet pas de faire apparaître la notion de statuts et l'attribut n'est pas significatif.

Avant de fournir les données au modèle, nous normaliserons celles-ci afin de faire converger plus rapidement l'estimateur vers le résultat, d'équilibrer l'importance de chaque variable et pour avoir une meilleure réponse. Les données discrètes sont transformées en données continues avec LabelEncoder. Toutes les données ont été normalisées avec **MinMaxScaler**.

II.2. Choix des familles

Il n'existe pas a priori de méthode déterminant le choix d'accorder un emprunt à une personne. Pour déterminer le meilleur classificateur, il est nécessaire de tester différentes familles. Nous nous intéresserons aux familles vu pendant le cours « RCP209 » :

- Classificateur linéaire
- SVM
- Arbre de décision
- Réseau de neurones

Le jeu de test sera séparé en deux selon un rapport 70/30. 70 % du jeu servira de base d'apprentissage et 30 % servira de base de test/validation pour évaluer la capacité de généralisation. Les éléments composant le jeu de test seront tirés de manière aléatoire.

Pour chaque famille, nous testerons via une recherche exhaustive avec validation croisée un ensemble de modèles avec un certain nombre d'hyper-paramètres. Nous chercherons à affiner pour obtenir à la fois une bonne erreur d'apprentissage et de généralisation, mais aussi que ces deux valeurs soient proches pour éviter le sur-apprentissage. Nous comparerons ensuite les différents modèles de chaque famille entre eux. Celui qui présente la meilleure généralisation sera conservée.

III. Analyse linéaire discriminante

Un premier outil, simple et efficace, que nous pouvons utiliser est l'analyse linéaire discriminante. Il s'agit de déterminer le meilleur droite/hyperplan capable d'approximer au mieux la relation entre les variables d'entrée et de sortie selon une relation linéaire.

Nous utiliserons la classe **LinearDiscriminantAnalysis** de scikit-learn. Le seul hyper-paramètre intéressant que possède cette classe est le type de solver (décomposition en valeurs singulières, moindre carrée et vecteur propre)

Après exécution, les résultats avec toutes les variables :

- Scale Score apprentissage = 77.25%
- Scale Score test = 77.00%

Avec les variables filtrées :

- Scale Score apprentissage = 76.12%
- Scale Score test = 77.00%

Dans les deux cas, le meilleur solveur est celui des vecteurs propres (eigen).

Le score pour l'erreur et pour l'apprentissage sont proches, indiquant que l'analyse discriminante a bien généralisée.

IV. SVM

Le SVM, pour Support Vector Machine (Machine à Support Vecteur), est un outil efficace pour classifier les résultats. Il s'appuie sur la recherche d'un hyperplan maximisant la marge séparant les deux ensembles.

IV.1. SVM linéaire

Le premier modèle de cette famille est le SVM linéaire. Il semble peu vraisemblable que l'ensemble des données soit facilement séparable. C'est pourquoi nous introduirons le paramètre C, variable d'équilibrage. Il offre la possibilité d'accepter un nombre d'erreurs dans la marge. Il peut amener à une meilleure généralisation. Nous nous servirons du composant « **LinearSVC »** pour ce test.

Les paramètres explorés pendant la recherche seront :

- C : paramètre de généralisation. Plus C est grand, plus on accepte d'erreurs, plus C est petit, plus la marge est étroite. Nous allons faire varier C de 10^n avec $n \in \{-5, 5\}$

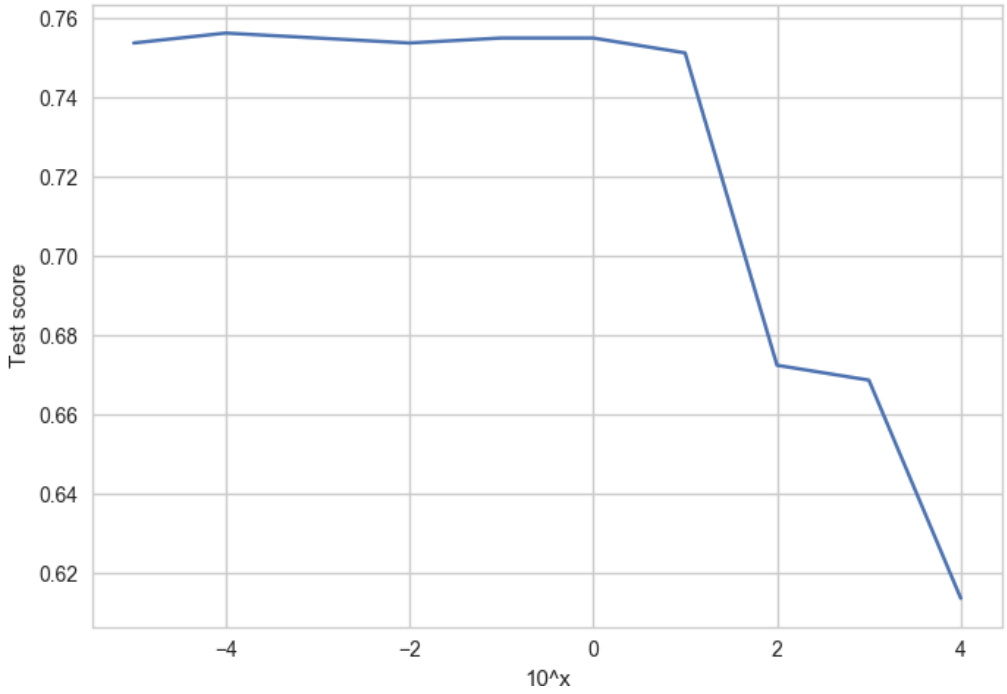
On obtient le résultat suivant :

- Score apprentissage = 77.62%
- Score test = 77.50%

Avec le sous-ensemble :

- No Score apprentissage = 77.00%
- No Score test = 78.50%

L'impact du paramètre de généralisation C sur le score est le suivant :



Au-delà de 1, le modèle ne répond pas correctement, il n'arrive plus à généraliser correctement. Nous conservons le résultat obtenu.

IV.2. Kernel SVM

Il est possible de reprojeter les données dans espace différent, appelé « espace de Hilbert ». Cela nous permet de faire apparaître une séparation linéaire dans cette espace via une transformation non-linéaire.

Pour ce faire, nous prendrons **SVC** de scikit-learn. La recherche des hyper-paramètres sera fait dans le champ suivant :

- 'C' : paramètre de généralisation. C variera de 10^n avec $n \in \{-5, 5\}$
- 'kernel': Noyau utilisée pour la projection. Nous testerons les noyaux linéaires, polynomiaux, gaussien et sigmoïde.
- 'gamma': hyper-paramètre de certains noyaux. Il variera de 10^n avec $n \in \{-5, 5\}$

Avec toutes les variables explicatives, le score du meilleur modèle à la suite de la recherche est :

- No Score apprentissage = 77.12%
- No Score test = 78.00%

Avec le sous-ensemble, le score du meilleur modèle à la suite de la recherche est :

- No Score apprentissage = 77.00%
- No Score test = 78.50%

V. Arbre

Dans le cas des arbres, il n'est pas nécessaire de repartir de données normalisées. Nous reprendrons le jeu de départ inchangé.

V.1. Arbre simple

Les arbres sont des outils simples à comprendre. Ils sont constitués de nœuds représentant des décisions. Chaque décision emmène vers un autre nœud jusqu'à atteindre une feuille fournissant une réponse.

Le composant **DecisionTreeClassifier** sert de base. Scikit-learn utilise une version optimisée de l'algorithme CART. Avec la configuration de base, nous avons comme résultat :

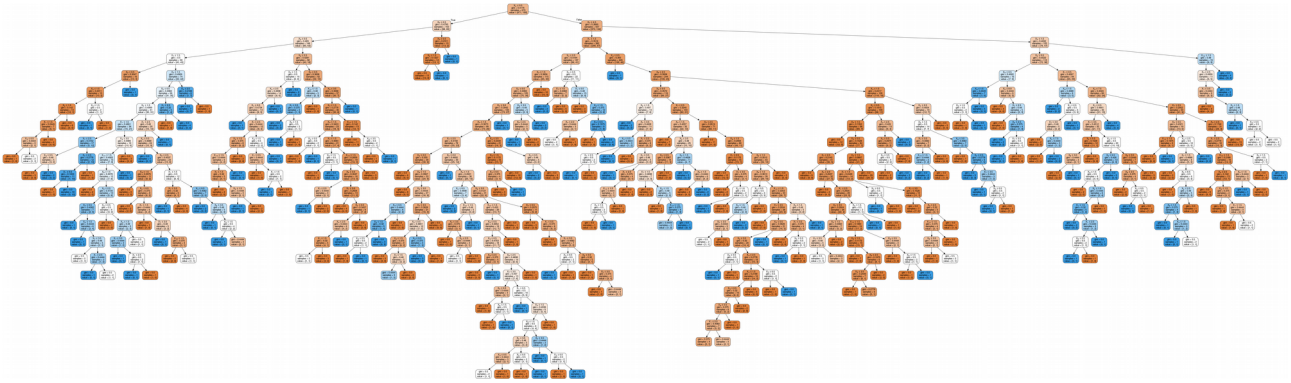
- Score apprentissage = 100.00 %
- Score test = 71.21 %

Avec le sous-ensemble d'attributs :

- Score apprentissage = 97.00 %
- Score test = 68.00 %

Le score d'apprentissage est de 1. Cela indique de l'arbre a été capable d'apprendre l'ensemble du jeu d'apprentissage. Le score d'apprentissage est très différent du score de test. On peut en déduire que l'arbre a appris les particularités de l'ensemble d'apprentissage et n'a pas forcément bien généralisé.

Le schéma suivant représente l'arbre calculé:



On peut constater qu'il est très large et profond. Il va être nécessaire d'élaguer l'arbre afin qui acquière de meilleures capacités de généralisation

On effectue alors une recherche des meilleurs hyper-paramètres par validation croisée selon les paramètres suivants:

- 'max_depth' : hauteur de l'arbre, de 1 à 10.
- 'min_samples_leaf' : nombre minimal d'entrée par nœud, de 1 à 10
- 'min_samples_split': le nombre d'entrées minimal pour créer un nouveau nœud, de 2 à 10

Après exécution, les résultats suivants sont obtenus :

- Score apprentissage = 70.29%
- Score test = 69.39%

Avec le sous-ensemble, nous obtenons :

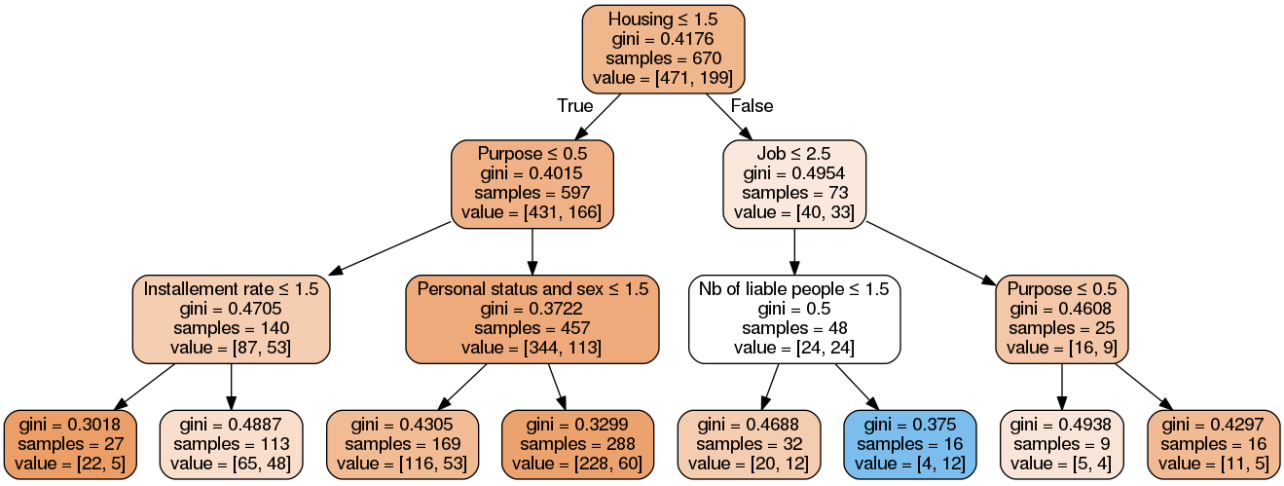
- Score apprentissage = 75.00 %
- Score test = 77.00 %

Maintenant, les deux scores sont très proches, ce qui indique que l'arbre trouvé "généralise" plutôt bien. De plus, le résultat obtenu avec moins d'attributs est bien supérieur au score précédent, mais aussi au score avec tous les attributs. Nous pouvons en constater que les arbres sont vite perturbés par le jeu d'apprentissage fourni en entrée.

Les paramètres du modèle sont les suivants :

- 'max_depth': 6
- 'max_features': 'auto'
- 'min_samples_leaf': 3
- 'min_samples_split': 6

Au vu des paramètres du meilleur estimateur, l'arbre s'est raccourci en longueur et en largeur. De même, nous pouvons visualiser l'arbre résultat.



Avec l’ensemble des attributs, le score obtenu est assez faible (70%). Cela montre que l’arbre est perturbé par des informations de faible valeur.

V.2. Extra-trees

Nous avons vu deux possibilités pour améliorer la performance d’un arbre seul : les RandomForests et le Bagging :

- Bagging : Il s’agit de créer un outil de décision basé sur l’agrégation des réponses de multiples estimateurs faibles. Chaque estimateur est créé sur un tirage avec remise de l’ensemble des données d’entrée.
- Random Forest : le principe est de créer plusieurs estimateurs en sélectionnant un sous-ensemble aléatoire des attributs sur le jeu de données d’entrée(Feature sampling). Chaque estimateur s’appuie sur ces attributs pour l’apprentissage et la prédiction.

Les différents estimateurs issus du bagging ne sont pas indépendants. En effet, l’ensemble d’apprentissage est partagé. Le feature sampling permet de résoudre en partie ce problème.

Il existe d’autres outils introduisant plus d’aléatoire dans le modèle. Nous utiliserons les **ExtraTreesClassifier** qui ajoutent un niveau supplémentaire d’aléatoire :

- Random Split : la séparation est définie de manière aléatoire entre les attributs. Celui qui obtient le meilleur résultat au sens du gain d’information est sélectionné pour créer le nœud test.

Avec la configuration par défaut pour un extra-tree dans Scikit-learn, nous obtenons les résultats suivants :

- Score apprentissage = 100 %
- Score test = 72.72 %

Avec le sous-ensemble de variables :

- Score apprentissage = 97.62 %
- Score test = 76.50 %

Avec les valeurs par défaut, le résultat est déjà bon. Pour déterminer le meilleur des extra-trees, nous faisons varier un ensemble d’hyper-paramètres :

    - 'max_depth' (hauteur max de l’arbre) : de 1 à 12 par pas de 2
    - 'min_samples_leaf' (nombre minimal de feuilles par nœud): range(1, 10, 2)
    - 'min_samples_split': range(2, 10, 2)
    - 'n_estimators' (nombre d’estimateurs) : de 50 à 450 par pas de 50

Nous obtenons le résultat suivant :

- Score apprentissage = 98.65 %

- Score test = 74.84 %

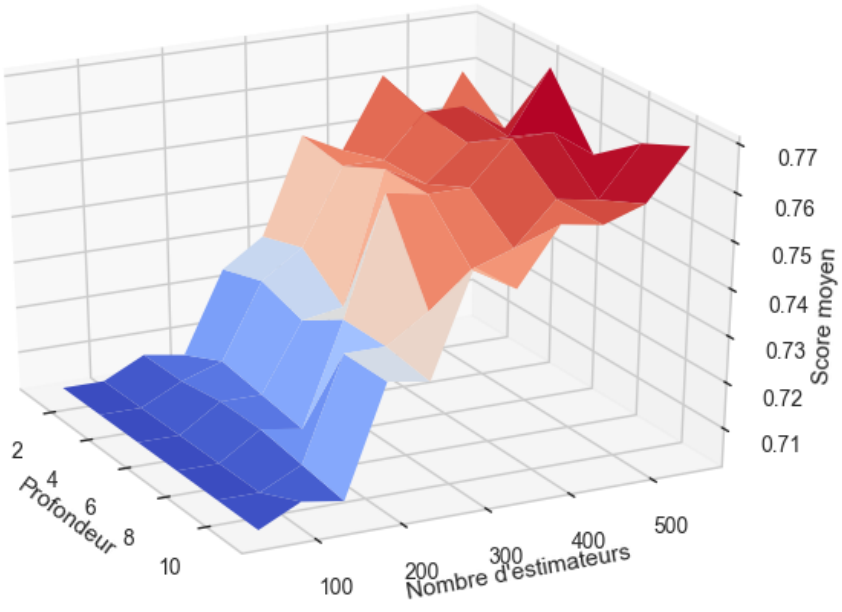
Avec le sous-ensemble :

- Score apprentissage = 83.50 %
- Score test = 76.50 %

Le résultat est légèrement supérieur pour l'erreur d'apprentissage et de test. Avec le sous-ensemble de données, on peut constater une meilleur généralisation. Le meilleur estimateur a les paramètres suivants :

- 'max_depth': 11
- 'min_samples_leaf': 1
- 'min_samples_split': 2
- 'n_estimators': 350

Le nombre d'estimateurs est plus élevé et la profondeur plus importante comparé à l'arbre. Nous pouvons visualiser l'influence de la profondeur et du nombre d'estimateurs sur le score final. Nous obtenons le graphique suivant :



Avec un faible nombre d'estimateurs, la profondeur de l'arbre à un impact faible sur le résultat. Cependant, au-delà de 200 estimateurs, nous constatons que la profondeur peut faire varier de quelques points le score du modèle.

V.3. Boosting

Le *boosting* construit plusieurs arbres en série. Chaque arbre prend en entrée les erreurs de son prédécesseur et il essaye d'apporter une réponse améliorée. Pour construire ce modèle, nous nous appuyons sur l'algorithme Adaboost (pour *Adaptative Boosting*) présent dans la librairie scikit-learn.

Nous utilisons le composant **AdaBoostClassifier**. En utilisant le paramétrage par défaut, nous obtenons le résultat suivant :

- Score apprentissage = 99.70 %
- Score test = 71.51 %

Avec le sous-ensemble :

- Score apprentissage = 97.37 %
- Score test = 72.50 %

Ce score est assez faible au vu des autres techniques utilisées précédemment. Nous effectuons donc une recherche des meilleurs hyper-paramètres. La grille de recherche est la suivante :

- 'n_estimators' : nombre d'estimateurs, de 10 à 400, par pas de 20
- 'base_estimator': AdaBoost fait la somme des différentes réponses de plusieurs estimateurs faibles. Nous utiliserons des arbres simples comme celui utilisé dans le paragraphe « Arbre Simple ». Nous faisons varier la hauteur de ces arbres

Nous obtenons :

- Score apprentissage = 77.46 %
- Score test = 74.84 %

Avec le sous-ensemble :

- Score apprentissage = 73.75 %
- Score test = 74.50 %

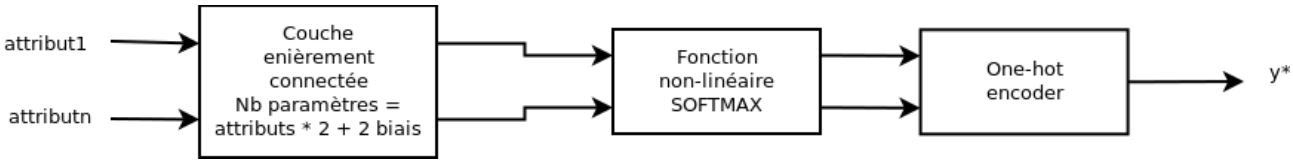
Avec la recherche, nous constatons, de même, de meilleures fonctions de prédiction comparées à un arbre seul.

VI. Réseaux de neurones

VI.1. Réseaux de neurones à une couche

Les réseaux de neurones sont un outil puissant dans la mallette du « Data Scientist ». L'idée est de reproduire la manière dont un cerveau fonctionne. Celui-ci est constitué d'une multitude de couches de neurones. Chaque neurone est un élément simple effectuant une agrégation des entrées pondérées suivies par une partie non-linéaire.

Pour le premier modèle, nous créons une couche constituée de 50 neurones en entrée pour les 48 variables d'entrée ré-encodées. Le réseau utilisé est le suivant :



La phase d'apprentissage s'appuie sur la **backprogragation**, qui fait redescendre étape après étape la différence constatée entre les données en sortie et celles en entrée des neurones. Nous effectuons 1000 passes.

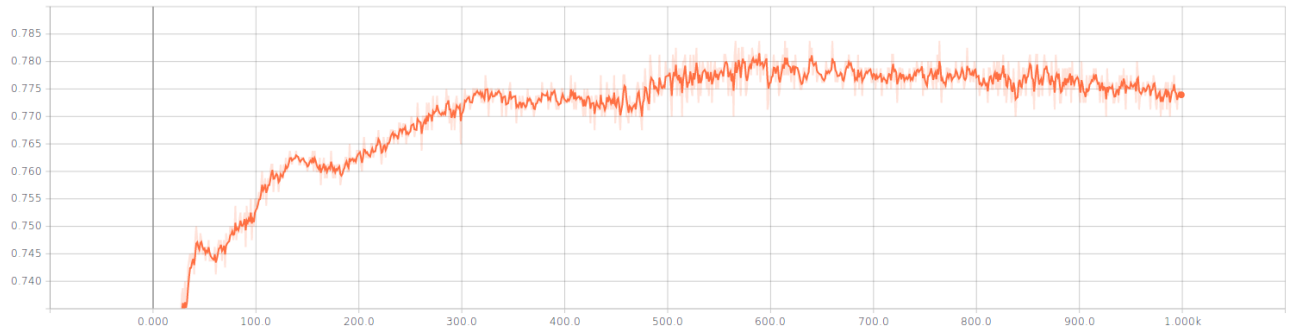
L'implémentation est faite via **Keras**.

- Apprentissage : 77.50%
- Test : 79.50%

Avec le sous-ensemble :

- Apprentissage : 75.25%
- Test : 78.50%

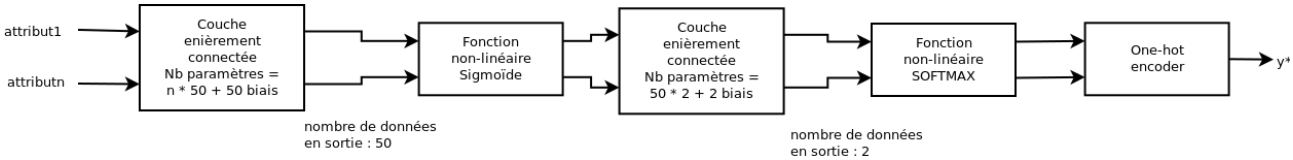
Il est possible de visualiser la courbe d'évolution du score obtenu :



La courbe atteint à pic au alentour de 600 passes. Au-delà, la fonction de prédiction ne généralise pas correctement.

VI.2. Réseau de neurones à une couche cachée

Nous avons testé un modèle basé sur un réseau de neurones constituée d'une couche simple. Nous pouvons ajouter facilement une couche cachée. Cette couche peut permettre de mieux généraliser, avec un temps d'apprentissage plus long. Nous utilisons le réseau suivant :



Au bout de 1000 passes, nous avons :

- Apprentissage : 77.88%
- Test : 79.00%

Avec le sous-ensemble :

- Apprentissage : 74.75%
- Test : 78.50%

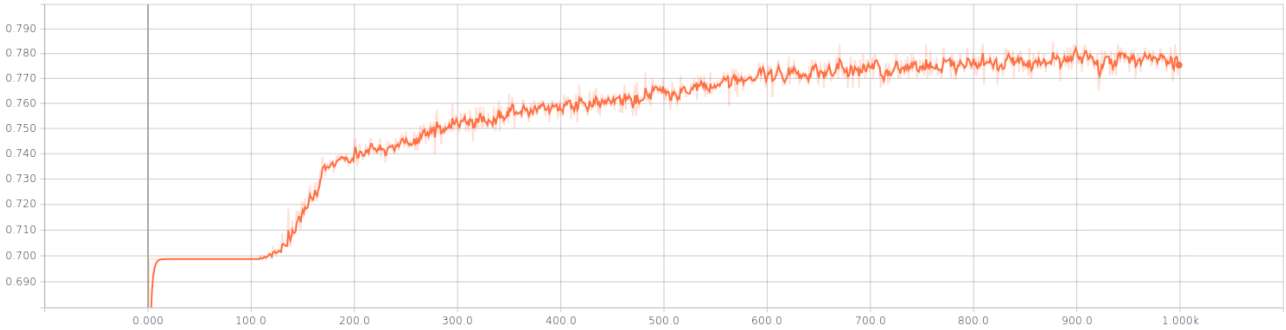
La fonction sigmoïde n'est pas forcément la plus adaptée. Bien qu'elle soit proche du comportement observé pour des neurones réels, la fonction ReLU peut être utilisée à la place en espérant de meilleurs performances. Au bout de 1000 passes, nous avons :

- Apprentissage : 99.25%
- Test : 76.00%

Avec le sous-ensemble :

- Apprentissage : 83.25%
- Test : 76.00%

Nous conservons donc les résultats avec la fonction sigmoïde. Le résultat est proche d'un réseau de neurones sans couche cachée. Observons l'évolution du score en fonction du nombre de passes :



Un plateau est observé au début, indiquant que le réseau n'arrive pas à « apprendre » plus du jeu de données. Après 150 passes, le score augmente jusqu'à l'arrivée à un pic aux alentours de la 800è passe.

VII. Choix du meilleur estimateur

VII.1. Synthèse

Nous pouvons synthétiser les différents résultats dans le tableau suivant sous la forme « résultat apprentissage/test » :

	ALD	**SVM**	**KSVM**	**Arbre seul**	**Extra-trees**	**Adaboost**	**NN 1 couche**	**NN 2 couches**
Tous les attributs	77.25 / 77.00	77.62 / 77.50	77.12 / 78.00	70.29 / 69.39	98.65 / 74.84	77.48 / 74.84	77.50 / 79.50	77.88 / 79.00
Attributs filtrés	76.12 / 77.00	77.00 / 78.50	77.00 / 78.50	75.00 / 77.00	83.55 / 76.50	73.75 / 74.5	75.25 / 78.50	74.75 / 78.50

VII.2. Choix du meilleur estimateur

Au vu de l'efficacité des choix des attributs, nous ne retiendrons que ces résultats.

Les différentes réponses sont assez proches. Même si le meilleur résultat en test vient du réseau de neurones à une couche, l'analyse linéaire discriminante et le SVM offrent des résultats tout à fait satisfaisants. La capacité (au sens de la dimension VC) de l'ALD comparée à celle du réseau de neurones est bien inférieure. En tenant compte de l'inégalité suivante :

$$R < R_{emp} + \sqrt{\left(\frac{h\left(\ln\left(\frac{2n}{h}\right)+1\right)-\ln\left(\frac{\alpha}{4}\right)}{n}\right)}$$ avec h, la dimension VC

Nous pouvons préférer l'ALD ou SVM dans la mesure où nous sommes sûrs que le risque espéré sera bien maîtrisé par rapport à celui d'un réseau de neurones à une couche.

VII.3. Analyse

Le résultat est au maximum de 80% environ de précision à la prédiction. Nous pouvons supposer qu'une partie de la variance ne s'explique pas par les paramètres fournis. Il manque des informations pour pouvoir trouver un meilleur résultat.

Comme indiqué lors de l'étude du jeu de données, des informations comme le salaire de l'emprunteur sont sans doute prises en compte lors de l'accord du crédit.

Une autre information manquante est la représentativité du jeu de données fourni : est-ce que les décisions prises sont homogènes entre les conseillers?. En effet, si nous voulons pouvoir utiliser plus largement le modèle de prédiction choisi, il est nécessaire d'avoir un jeu représentatif. Il est ainsi difficile de se prononcer sur la possibilité de généraliser le modèle trouvé.

Nous remarquons la présence plus importante de réponses positives que négatives. Nous supposons que cela vient du fait que les personnes ont forcément tendance à déposer une demande si elles sont sûrs de pouvoir obtenir une réponse négative. Par exemple, on peut voir que tous les demandeurs ont au moins une personne se portant garant pour eux, ce qui n'est pas forcément le cas de la plupart des personnes.

Conclusion

Le jeu de données “Credit Card Savings” est un jeu de données connu du monde de la “Data Science”. Lors de notre étude, nous avons suivi le parcours classique pour effectuer une analyse du jeu de données. Nous avons cherché à trouver le meilleur classificateur en développant des outils grâce à des librairies puissantes du monde de la “Data Science”.

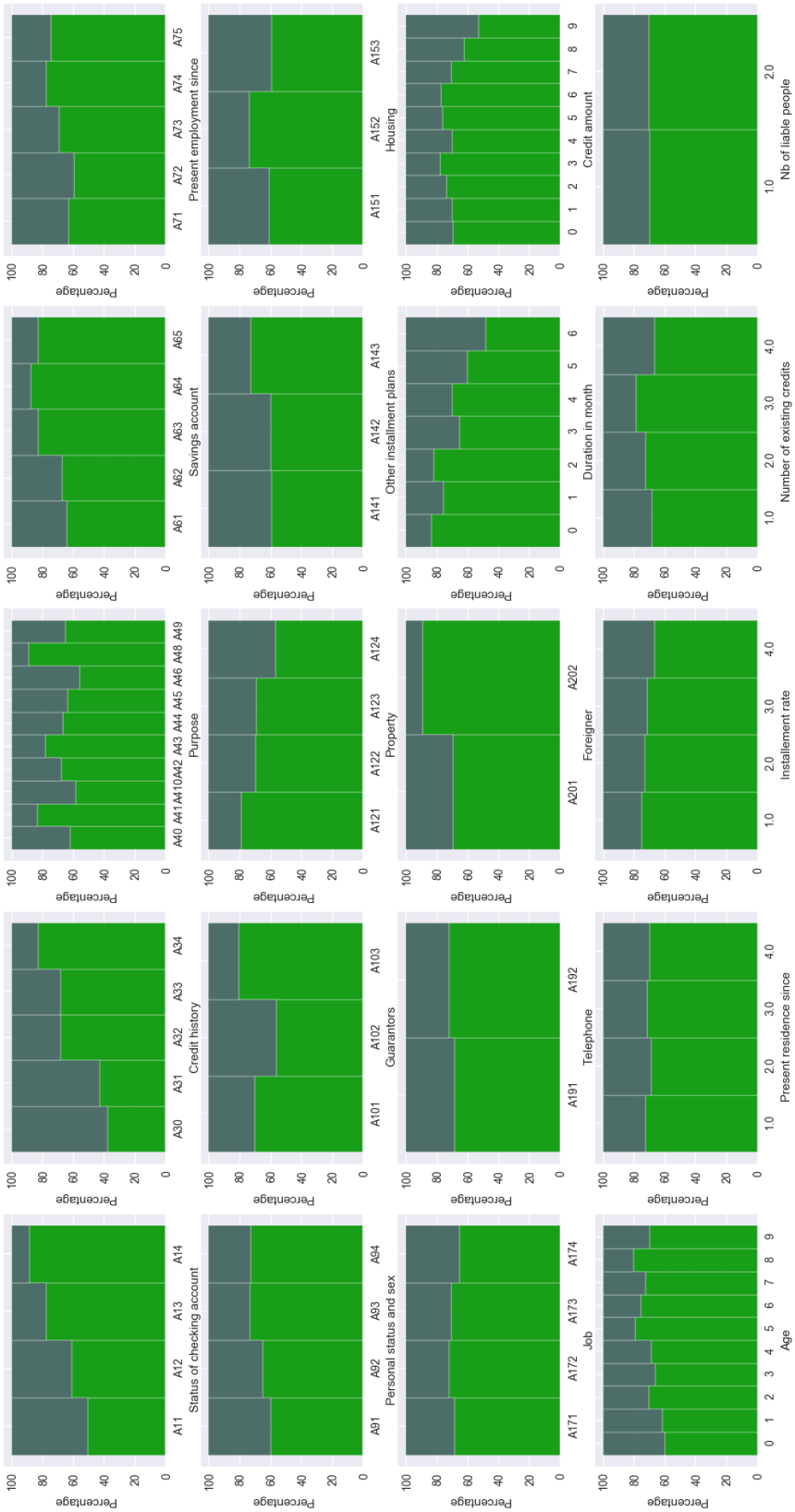
Grâce à ce modèle, nous pouvons imaginer de le déployer comme un outil d’aide à la décision pour des conseillers bancaires afin de les guider dans le choix d’accorder un crédit aux demandeurs.

Nous faisons face à plusieurs limites de cet outil:

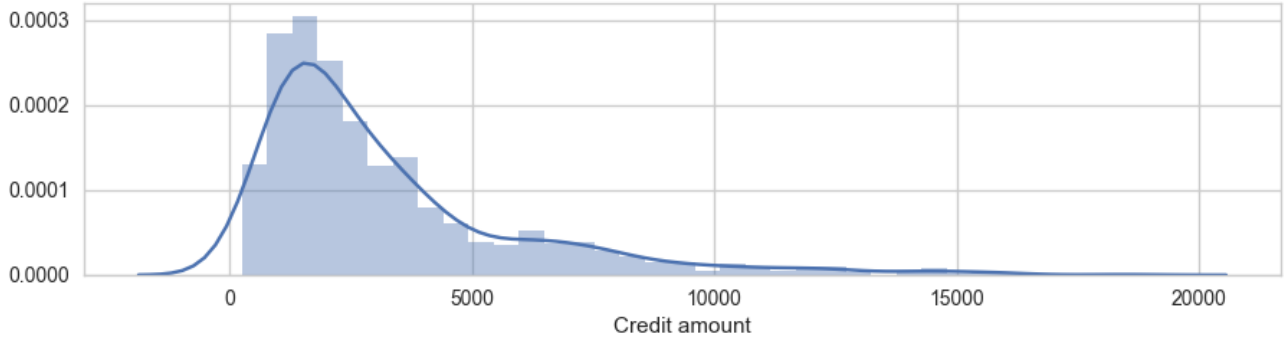
- Il n’est pas simple de fournir une explication rationnelle aux choix pris par le modèle. Le refus pourrait être vu comme discriminatoire par rapport au demandeur. Afin de se prémunir d’un risque de procès, il serait important de s’assurer de l’absence de prise en compte de paramètre comme le sexe par exemple.
- La décision prise par le modèle est correcte 3 fois sur 4. Il est nécessaire de se demander si celui-ci est aussi performant qu’un humain.
- Le modèle trouvé pourrait induire en erreur le conseiller. Le modèle semble se tromper autant sur les réponses positives que négatives. Pour limiter le coût en cas d’accord de crédit à une personne non solvable, il est nécessaire de conserver le jugement d’un expert bancaire qu’est le conseiller.

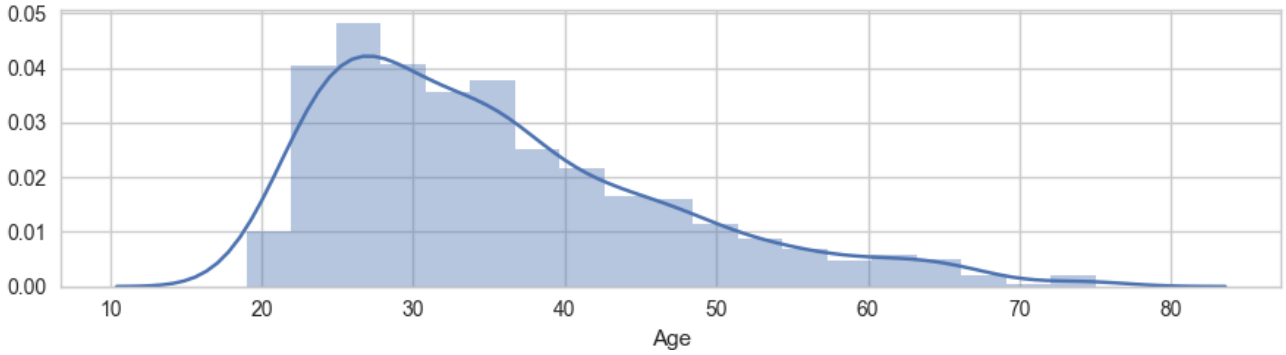
Annexes

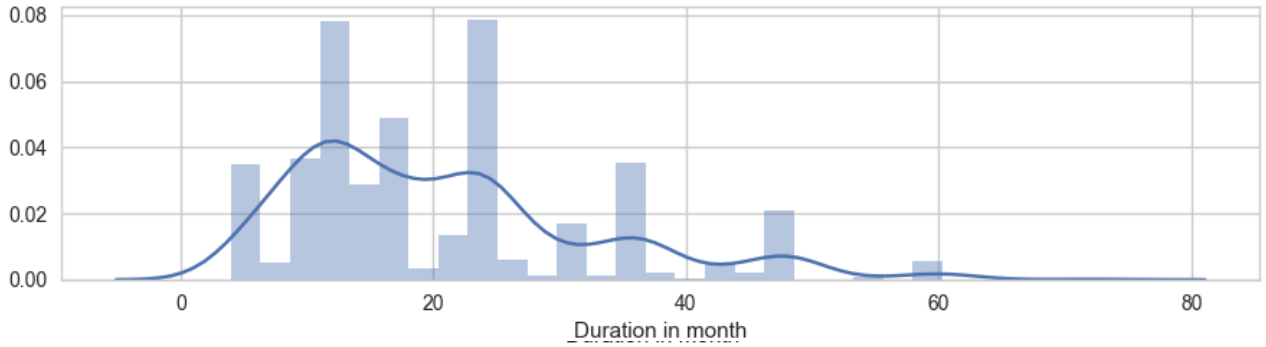
Distribution relative

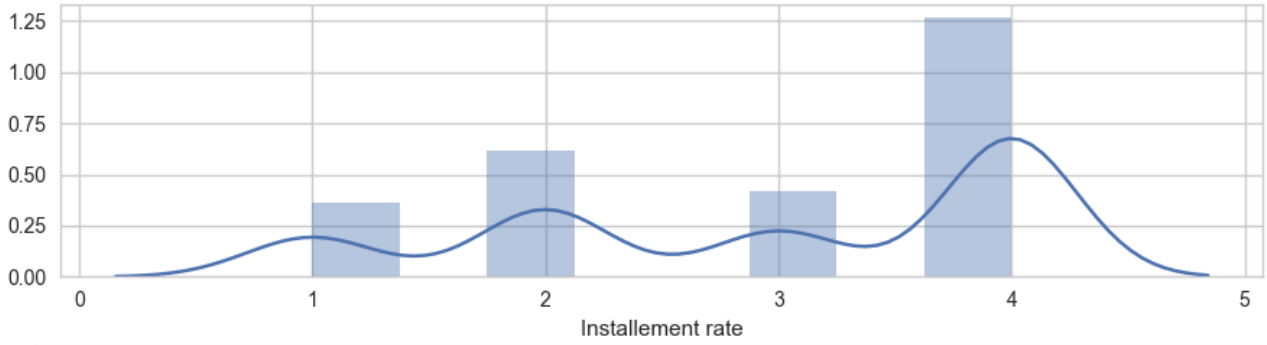


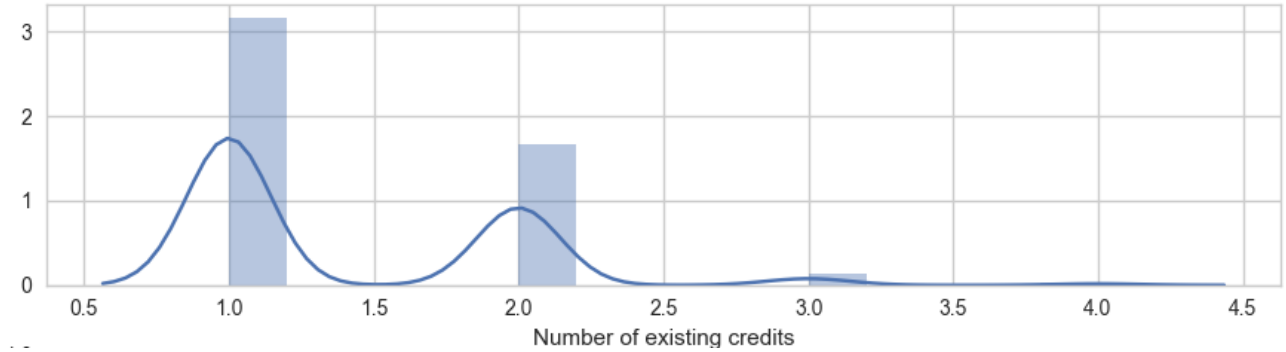
Distribution

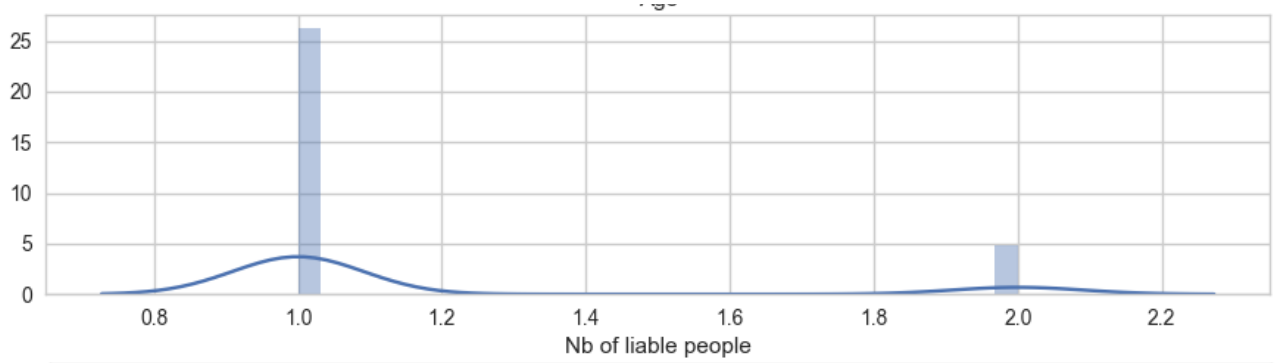


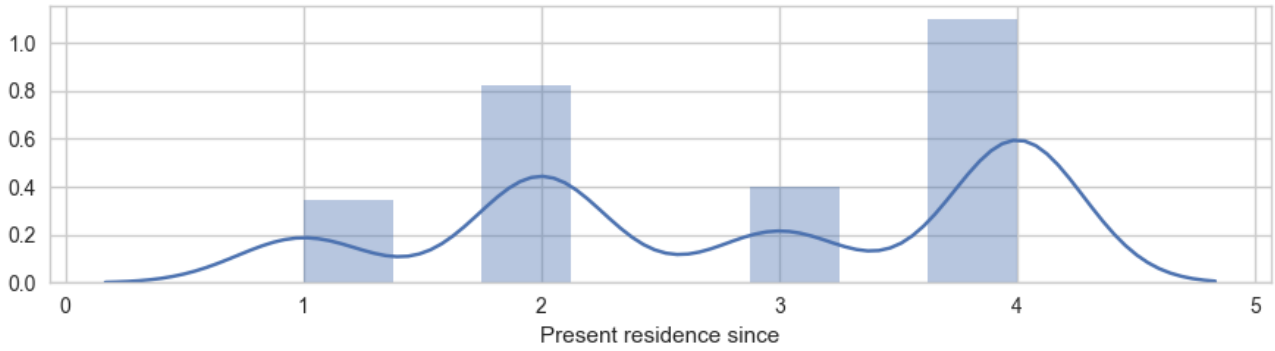












Tables de contingences

Credit	A11	A12	A13	A14	Total
1	139	164	49	348	700
2	135	105	14	46	300
Total	274	269	63	394	1000

Status of checking account

Credit	A30	A31	A32	A33	A34	Total
1	15	21	361	60	243	700
2	25	28	169	28	50	300
Total	40	49	530	88	293	1000

Credit history

	A40	A41	A410	A42	A43	A44	A45	A46	A48	A49	Total
1	145	86	7	123	218	8	14	28	8	63	700
2	89	17	5	58	62	4	8	22	1	34	300
Total	234	103	12	181	280	12	22	50	9	97	1000

Purpose

	A61	A62	A63	A64	A65	Total
1	386	69	52	42	151	700
2	217	34	11	6	32	300
Total	603	103	63	48	183	1000

Savings account

	A71	A72	A73	A74	A75	Total
1	39	102	235	135	189	700
2	23	70	104	39	64	300
Total	62	172	339	174	253	1000

Present employment since

	A91	A92	A93	A94	Total
1	30	201	402	67	700
2	20	109	146	25	300
Total	50	310	548	92	1000

Personal status and sex

	A101	A102	A103	Total
1	635	23	42	700
2	272	18	10	300
Total	907	41	52	1000

Guarantors

	A121	A122	A123	A124	Total
1	222	161	230	87	700
2	60	71	102	67	300
Total	282	232	332	154	1000

Property

	A141	A142	A143	Total
1	82	28	590	700
2	57	19	224	300
Total	139	47	814	1000

Other installment plans

	A151	A152	A153	Total
1	109	527	64	700
2	70	186	44	300
Total	179	713	108	1000

Housing

	A171	A172	A173	A174	Total
1	15	144	444	97	700
2	7	56	186	51	300
Total	22	200	630	148	1000

Job

	A191	A192	Total
1	409	291	700
2	187	113	300
Total	596	404	1000

Telephone

	A201	A202	Total
1	667	33	700
2	296	4	300
Total	963	37	1000

Foreigner