L2 MI – Mini Projet
Challenge Survival – Groupe Survivers
Chef de projet : Nourry Justine
Membres : Andriamiarisoa Ny Tsanta, Bedjou Zinedine, Dupré Camille
URL du challenge: https://codalab.lri.fr/competitions/383
Numéro de la dernière soumission Codalab : 8746
Repo GitHub du projet + Diapos
:https://github.com/survivers-info232/survivers
URL de la vidéo Youtube : https://www.youtube.com/watch?v=hiFeA47m6FQ

● Introduction :

Dans notre challenge “survival”, nous essayons de prédire la survie des patients grâce aux données sur leur régime alimentaire avec l'aide de l’apprentissage automatique. L’objectif de l'analyse de survie est de prédire le temps attendu avant qu'un événement donné se produise. Dans notre cas, nous essayons d’estimer le temps de survie des 14 000 patients à partir de l’ensemble des données NHANES. Ces données présentent 681 paramètres et ont été réalisées sur une période de 9 ans (de 1999 à 2008). Ce problème est un problème de régression.
Nous aurions voulu travailler sur le projet Hadaca car il s'agit de celui qui nous semblait le plus intéressant. Cependant le projet survival restait dans la même idée, nous aurions aimé l'approfondir plus mais par manque de personne cela n'a pas été possible.

Le projet comporte 3 parties:

- Le preprocessing, c’est-à-dire la transformation de données brutes en données utilisables par le classifieur.
- Le classifieur qui analyse les données d'entraînement ainsi que les résultats attendus sur ces données d'entraînement pour se calibrer.
- l’affichage des résultats de manière compréhensible et pertinente via une interface graphique.

● Description des parties :

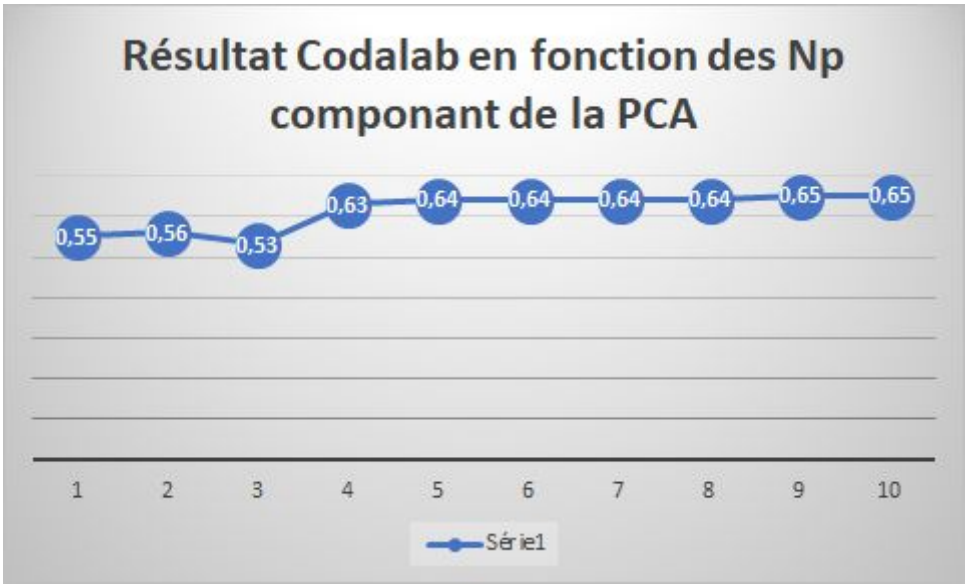
- **1) Preprocessing :** Dupré Camille et Nourry Justine

Le preprocessing consiste à préparer les données pour le classifieur, c'est-à-dire à trier, sélectionner et modifier les données afin de permettre au classifieur d'être plus efficace.
Il y a deux possibilitées :

- Améliorer les données en les pré-traitant.
- Améliorer la visibilité en diminuant le nombre de colonnes pour chaque donnée.

Dans un premier temps, nous avons voulu séparer les données censurées et non censurées (*voir figure 1*). Après avoir passé quelque temps à coder nous sommes arrivées à une impasse et nous avons donc décidé de changer d'idée.
Dans un deuxième temps, nous avons utilisé l'algorithme Principal Component Analysis (PCA) combiné à un pipeline *(voir code 1)* afin de réduire la dimension de notre ensemble de données en sélectionnant uniquement les plus pertinentes. Cet algorithme permet de rendre les données plus compréhensibles: si après avoir diminué de moitié le nombre de composantes nous obtenons un score similaire cela nous permet de comprendre quelles sont les données les plus importantes pour prédire la date de mort d'un patient.

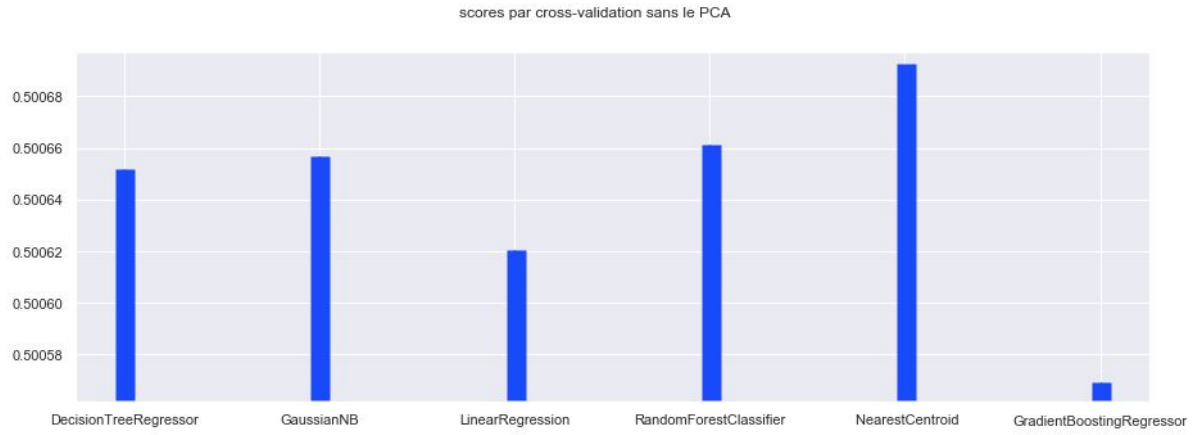
Cependant ce dernier point nous a fait rencontrer un premier problème : en effet notre score sur codalab est bas lorsque nous utilisons le pré processing (cf graphe ci-dessous). Cela s'explique par le fait qu'en diminuant le nombre de colonnes on perd de l'information (il est difficile par la suite de prévoir la date de mort d'une personne avec 2 données, sa taille et son poids par exemple).
Nous avons également un deuxième problème : lorsqu'on fait un PCA avec le nombre de features de notre matrice on obtient un score moins élevé que sans le PCA (notre score codalab avant l'ajout du pré processing était de 0,78). Pour essayer de comprendre le problème, on nous a conseillé de faire des tests avec différentes combinaisons des pre-processings PCA et Standardisation, mais nous n'avons pas eu le temps de le faire.



De plus, nous nous sommes rendues compte que la PCA n'était pas très utile car nous n'avions pas 681 colonnes comme prévu mais 10, donc en diminuant la dimension nous perdions beaucoup d'informations. Cependant, on voit qu'entre 10 et 4 composantes le score ne diminue pas énormément alors que la quantité d'informations a beaucoup diminué.

- **2) Classifieur :** Dupré Camille et Nourry Justine

Suite à ce qui a été vu dans la partie précédente, nous avons décidé de ne pas utiliser de pré processing. Nous récupérons donc les 'public data' qui étaient déjà pré-traités sans notre intervention. Nous avons ensuite comparé plusieurs algorithmes possibles de scikit-learn [1] sur les données d'entraînement. Nous avons sur le jupyter notebook créé une boucle comparant nos différentes régressions en testant les résultats par cross validation afin de déterminer laquelle est la meilleure (cf graphe ci-dessous [2], *voir code 2)*.



Par la suite, en comparant les résultats des soumissions Codalab (*voir le tableau bonus "Preliminary results"*) nous avons déterminé lequel était le plus adapté pour classer nos données.
Nous avons comparé les différentes régressions sans utiliser PCA puisque cela diminuait nos résultats. Pour cela, nous avons mis un choix pour utiliser ou non PCA *(voir code 3)*.
Ainsi nous avons choisi <GradientBoostingRegressor>, même si le but étant d'obtenir 1, notre résultat (0.7801) est très proche du but. Et grâce à cette régression, nous avons pu dépasser le score de REFERENCE0 et REFERENCE1 respectivement à 0.7672 et 0.3153.

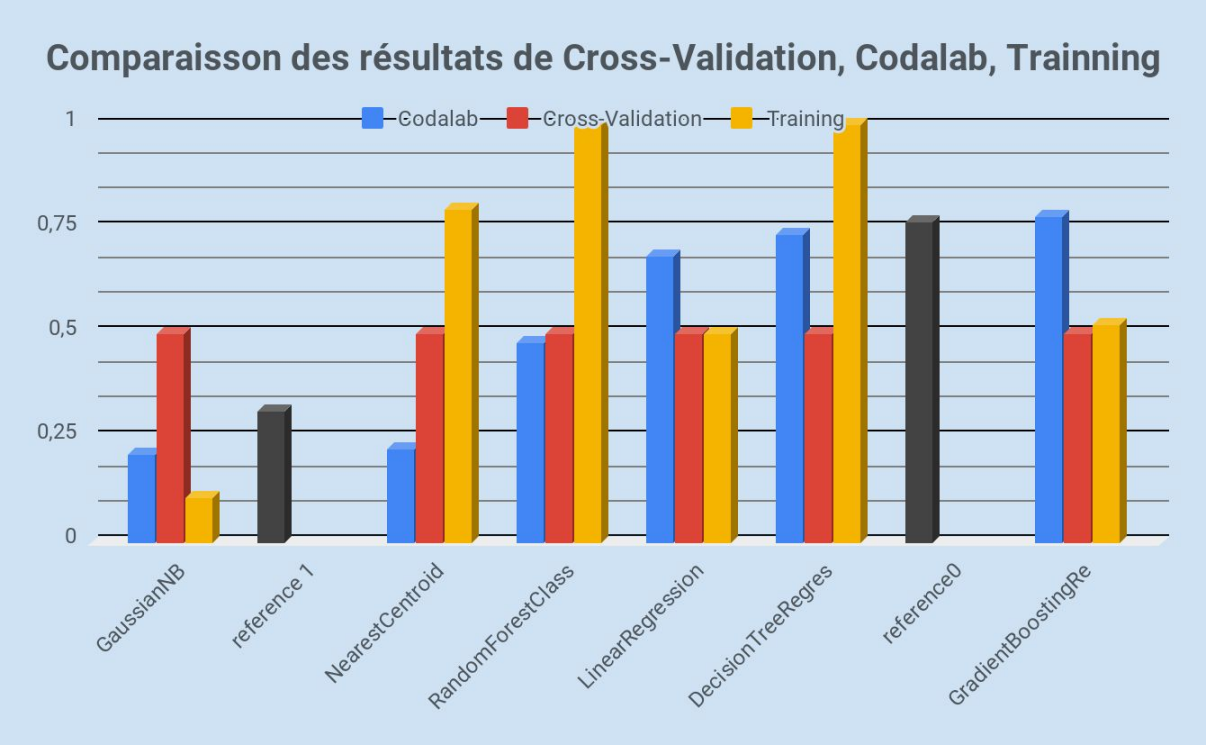
- **3) Interface graphique :** Bedjou Zinedine et Andriamiarisoa Ny Tsanta

Le but de la partie interface graphique est d'afficher les données de manière à ce qu'elles soient utilisables plus facilement et de rendre les résultats apportés par le classifieur accessibles et compréhensibles par tous.
Nous représentons ces résultats sous la forme d'histogrammes et à l'aide de régressions (*voir figure 2*) (ce qui semble le plus approprié pour notre projet).
Pour ce faire, nous utiliserons des librairies open-sources et/ou intégrées à python telles que MatPlotLib et Numpy.

- **Evolution des résultats**

Sans utiliser de pré processing, nous avons déterminé grâce aux différents résultats codalab quel algorithme est le plus adapté pour classer nos données.

Le schéma suivant indique le taux de réussit Codalab en fonction des différents algorithmes utilisés :



Comme indiqué précédemment, nous avons utilisé <GradientBoostingRegressor> comme régression.

Certaines de nos régressions ont un score nettement plus faible alors que lors de l'apprentissage celles-ci avaient de meilleurs résultats. Cela est dû au sur-apprentissage : L'intelligence artificielle a "appris par coeur" l'ensemble de tests mais sur un ensemble différent il reste encore de nombreuses erreurs.

Appendices :

Figures 1 : Pour le preprocessing

Les tableaux ci-dessous montrent le principe d'élimination des données censurées : l'algorithme prend en entrée toutes les données et ne garde que celles qui sont labellisées "non censurées".

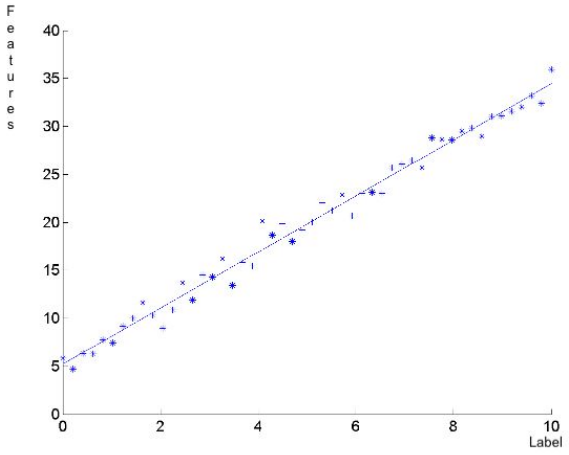
Étape 1 : Exemple de tableau des données initiales.

Bernard	….	non censurée
Jean	….	censurée
Kevin	….	censurée
Léa	….	non censurée

Étape 2 : Exemple de tableau après élimination des données censurées.

Bernard	….	non censurée
Léa	….	non censurée

Figure 2 : Régression Linéaire [3]



Exemple de Régression Linéaire : (modifié à partir d'un exemple trouvé sur internet)

On appelle droite de régression, la droite qui se « rapproche le mieux » du nuage de points, représenté par un échantillon d'apprentissage qu'on va fournir au classifieur. Plus communément appelée la régression linéaire, elle est considérée comme une méthode d'apprentissage supervisée utilisée pour prédire une variable quantitative.

- **Code 1 :** (lignes 30 - 43 et lignes 47 - 104)

  https://github.com/survivers-info232/survivers/blob/master/survivers/starting_kit/sample_code_submission/model.py

- **Code 2 :**

```
metric_name, scoring_function = get_metric()

X_train = D.data['X_train']
Y_train = D.data['Y_train']

graph_array_sans_PCA = np.empty((6,2),dtype = np.object)
nom_reg = ['DecisionTreeRegressor', 'GaussianNB', 'LinearRegression', 'RandomForestClassifier',
' NearestCentroid', 'GradientBoostingRegressor']
compteur1 = 0
for j in range (6) :
    M = model(j)
    M.fit(X_train , Y_train)
    scores = cross_val_score(M, X_train, Y_train, cv=5, scoring=make_scorer(scoring_function))
    graph_array_sans_PCA [compteur1,0] = nom_reg[compteur1]  #regression testé
    graph_array_sans_PCA [compteur1,1] = scores.mean()  #score de cross-validation
    compteur1 += 1
```

- **Code 3 :** (exemple avec la première régression)

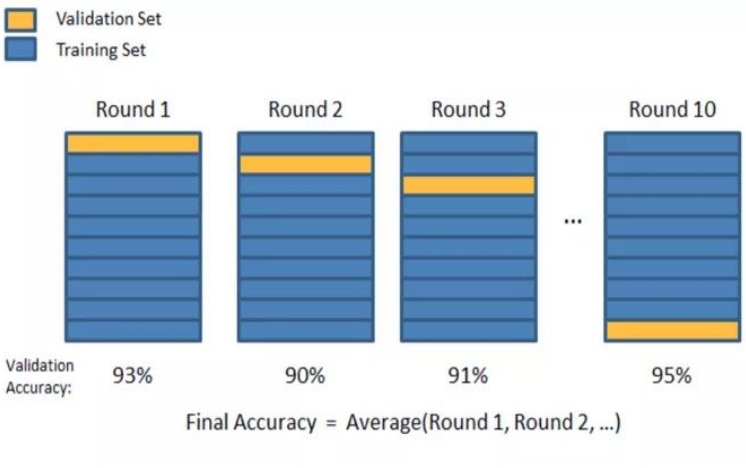
```
class model(BaseEstimator):
    baseline_clf = Pipeline([('Prepro', Preprocessor()),('DecisionTreeRegressor',
DecisionTreeRegressor(max_depth=4))])
    def __init__(self, choice =5 , n_components = 10, is_pca = False):
        '''
        This constructor is supposed to initialize data members.
        Use triple quotes for function documentation.
        '''
        self.num_train_samples=0
        self.num_feat=1
        self.num_labels=1
        self.is_trained=False

        if is_pca :
            if choice = … :
                self.baseline_clf = Pipeline([('Prepro', Preprocessor(n_components)),
('DecisionTreeRegressor', DecisionTreeRegressor(max_depth=4))])
            .
            .
            .
      else :
            if choice = … :
                self.baseline_clf = DecisionTreeRegressor()
```

- **Bonus :**

Cross validation [4]

La validation croisée (« cross-validation ») est, en apprentissage automatique, une méthode d'estimation de fiabilité d'un modèle fondé sur une technique d'échantillonnage.



1) On divise l'échantillon original en k échantillons.

2) On sélectionne un des échantillons comme ensemble de validation et les autres échantillons constitueront l'ensemble d'apprentissage.

3) On calcule le score de performance, et on répète l'opération pour que chaque sous-échantillon ait été utilisé exactement une fois comme ensemble de validation.

4) On calcule le score de performance en faisant une moyenne de tous les autres scores.

Sur-apprentissage

En statistique, le sur-apprentissage ou sur-ajustement (en anglais « *overfitting* ») est une analyse statistique qui correspond trop étroitement ou exactement à un ensemble particulier de données. Ainsi, cette analyse peut ne pas correspondre à des données supplémentaires et ne pas prévoir de manière fiable les observations futures.

Métrique

Pour évaluer les performances de notre modèle, nous avons besoin d'une métrique : l'indice de concordance. Cet indice permet de valider la capacité prédictive d'un modèle de survie.
Il correspond au rapport entre le temps et la probabilité de survie la plus élevée (temps de survie/probabilité de survie). Des valeurs élevées signifient que notre modèle prédit des probabilités de survie plus élevées pour des durées de survie plus élevées. Code (fontion custom_c_index) : https://github.com/survivers-info232/survivers/blob/master/survivers/starting_kit/scoring_program/my_metric.py

Tableau bonus : Preliminary results (taux de réussite)

Method	Naivebayes or Gaussian classifier	Linear regression or SVM	Decision Tree	Random Forest	Nearest Neighbors	GradientBoosting Regressor
Training	0.1063	0.4970	0.4493	1.0000	0.7991	0.5195
CV	0.63 (+/- 0.38)	0.50 (+/- 0.02)	0.45 (+/- 0.02)	0.22 (+/- 0.02)	0.80 (+/- 0.01)	0.49 (+/- 0.02)
Validation (codalab)	0.2084484036	0.683989487	0.6128	0.4795732662	0.2249313468	0.7799825349

On peut voir grâce au tableau et particulièrement la cross validation que toutes les régressions ne sont pas aussi performantes.
Nearest Neighbors a un très bon résultat : 0.80. Cependant il n'a que 0.22 sur Codalab. On peut en déduire qu'il a fait du sur-apprentissage (de même pour toutes les autres régressions ayant un meilleur score de cross validation que sur codalab).
En comparaison GradientBoosting Regressor obtient un score de 0.49 à la cross validation et 0.7799 sur codalab, cette régression semblait moins intéressante lors des tests de cross validation mais lorsque nous avons essayé sur Codalab nous avons remarqué que le résultat était beaucoup plus intéressant.

Tableau bonus : Statistiques des données.

Dataset	num. Example	num. Variables	Sparsity	Has categorical variables?	Has missing data?	Num. examples in each class
Training	19297	10	nb 0 / nb case de la matrice = 89673 / 231564 =0.38724931336477175	oui	Non, pas de missing data, mais 90% de censored data qui n'ont pas la date de mort du patient.	C'est une régression
validation	2413	10	9060 / 24130 = 0.375466224617	oui	Les valeurs censored n'ont pas la date de mort du patient	C'est une régression
test	2412	10	9052 / 24120 = 0.375290215589	oui	Les valeurs censored n'ont pas la date de mort du patient	C'est une régression

- **Références :**

[1] Scikit learn: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
[2] TP 10, cours de python du S3 (base du code donnant ce graphique)
[3]Régression :
https://www.science-emergence.com/Articles/Régression-linéaire-simple-avec-python/?fbclid=IwAR35uhXs4r8vKDrcWY7nATzStdw8qdRm4DGTMBoDFXBPKAd7BEUH6ePZs5U
[4] Cours de mini projet, Isabelle Guyon 2019.