Groupe administratif n°1 ,1bis, 2
Equipe Ghosts

Rapport de Mini-Projet

Challenge Survival

Membres :

- BORISSOV Piotr (Chef de groupe)
- CIOBANITA Emanuel-Steven
- DENIS Matthieu
- DE OLIVERA Hugo
- JERLSTRÖM Auguste

Challenge Survival :
https://codalab.lri.fr/competitions/383
Numéro dernière soumission Codalab : 8790
Vidéo YouTube :
https://www.youtube.com/watch?v=9QJ6ijvHww0&feature=youtu.be&fbclid=IwAR2VnPkBSQN-9WDF19hbZbkxB0xktYeHvtQgiwgaERofRN0tyGl4LKJa4KQ
Lien Diapositives :
https://prezi.com/view/PUsPPpDsZ4M37ZU0wU01
GitHub repository:
https://github.com/upsudghosts/ghosts

CodaLab

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir
université PARIS-SACLAY

Introduction et description du projet

Nous avons choisi le challenge Survival, qui consiste en prédire la durée de vie d'êtres humains, en fonction de plusieurs de leurs caractéristiques biologiques. Pour cela nous avons à notre disposition la base de données NHANES (National Health And Nutrition Examination Survey), qui fournit des centaines de ces caractéristiques, telles que leur âge, leur pression sanguine, leur indice de masse corporelle, ethnie, etc. Afin de nous faciliter la tâche, nous nous sommes répartis en trois groupes de deux : prédiction, preprocessing, et visualisation.

Les données à prédire sont continues (âge du décès) associées à plusieurs attributs (caractéristiques biologiques), nous allons donc utiliser une régression pour faire de la prédiction. Le défi est donc de trouver le bon type de régression, d'autant plus que nous avons une contrainte supplémentaire : chaque donnée est censurée ou non, c'est-à-dire qu'une personne censurée a quitté l'étude, et donc la valeur de son « décès » n'est pas réelle. Nous nous retrouvons donc avec deux matrices, une X contenant les données, chaque ligne représentant un individu, chaque colonne correspondant à une caractéristique. Et une matrice Y, contenant dans chaque ligne l'âge du décès de l'individu correspondant et s'il est censuré ou non (0 ou 1). Par contraintes techniques nous nous limitons aux dix attributs les plus importants par individu, et on se retrouve avec une matrice X de dimension 19297*10 et une matrice Y de dimension 19297*2.

Description des algorithmes étudiés et pseudo-code

Visualisation

Après avoir traité nos données et parcouru le modèle de prédiction, nous devrons afficher les données afin de tirer nos conclusions et de voir si notre prédiction fonctionne bien. Heureusement, Python possède de très bons outils et une bibliothèque pour nous aider à le faire. Yellowbrick est une suite d'outils de diagnostic visuel appelés "Visualiseurs" qui étendent l'API Scikit-Learn pour permettre un pilotage humain du processus de sélection du modèle.

En un mot, il combine l'apprentissage en kit avec matplotlib. De cette façon, nous pouvons produire les meilleures visualisations pour notre modèle régressif. Nous allons d'abord utiliser le graphique des résidus pour voir la différence entre la valeur observée de la variable cible (y) et la valeur prévue (ˆy), ce qui signifie l'erreur de la prédiction.

Les valeurs réelles (de l'ensemble de données) tracées en rouge, et les prédictions de notre modèle en bleu, ce qui permet de les différencier plus facilement les unes des autres. Nous pouvons également utiliser un tracé similaire qui est celui de l'erreur de prédiction qui fait essentiellement la même chose, mais en le comparant à une ligne de 45 degrés.

Preprocessing

Le preprocessing consiste à trier les données récoltées afin de les rendres plus "lisibles" et améliorer les résultats lors de la prédiction. Si jamais les données n'ont pas été passées au peigne fin, les résultats risquent d'être biaisés.

Notre tout premier "Preprocessing" a consisté à retirer toutes les lignes contenant des données censurées. Nous avions donc commencé à implémenter une classe de Preprocessing. Malheureusement, après avoir passé beaucoup de temps à l'implémenter, nous avons réalisé que cela était infaisable car nous avions besoin de retourner plusieurs tableaux (la matrice X des données et la matrice Y avec les références), ce qu'une classe de preprocessing n'autorise pas.

À la place nous avons créé une fonction drop_censored (tabX, tabY) qui prenait en argument la table des données avec la table des références et retournait deux nouveaux tableaux sans les lignes censurées.

Nous avions une première version brute du code, mais celle-ci fonctionnait mal. Cependant nous avons réussis à l'améliorer : la fonction que vous pouvez observer ci-dessous prends bien en argument deux

tableaux comme dis avant, puis nous créons l'array censored_indexes qui est composé des numéros de lignes de la table Y avec les données censurées. Puis nous enlevons ces numéros de lignes dans X puis Y avant de les retourner en résultat.

```
def drop_censored(X, Y):
    censored_indexes = np.where(Y==1)[0] #age != 0
    X_uncensored = np.delete(X, censored_indexes, axis=0)
    Y_uncensored = np.delete(Y, censored_indexes, axis=0)
    return X_uncensored, Y_uncensored
```

Figure1 : code drop_censored

Après avoir testé ce code sur différent modèles nous avons remarqué qu'il ne servait uniquement à baisser le score.

Nous manquions donc de temps pour reproduire un nouveau preprocessing et c'est ainsi que nous avons décidé de nous servir d'un preprocessing déjà codé : le PCA ou le Principal Component Analysis. Le but de ce preprocessing est de transformer les variables liées en de nouvelles variables qui n'ont pas de corrélation entre elles. Cela permet de réduire dans la majorité des cas le nombre de variables et aussi d'éviter les redondances.

. Ci-dessous vous pouvez observer le pseudo code principal pour le PCA. Le PCA consiste en deux parties : l'Exploration, puis le Scattering.

L'exploration est la partie "non aléatoire" du PCA, c'est à dire que l'on va parcourir les données et on va essayer de former des groupes. Ces nouveaux groupes de données définirons une nouvelle variable. Le scatter sert a re-répartir les données, le but étant de pouvoir continuer à créer de nouveaux groupes jusqu'à avoir le nombre de variables recherchée (il se peut que l'on ne puisse plus créer de groupe a un certain instant car les nuages de points ne le permettent pas. Le scatter "chamboule" les points et permet ainsi de recréer de nouveaux groupes).

```
Generate an initial solution Old_Config
Best_Fitness = Fitness(Old_Config)
For n = 0 to # of iterations
 Perturbation()
 If Fitness(New_Config) > Fitness(Old_Config)
    If Fitness(New_Config) > Best_Fitness
       Best_Fitness = Fitness(New_Config)
    End-If
    Old_Config = New_Config
    Exploration()
 Else
    Scaterring()
 End-If
End-For

Exploration()
 For n = 0 to # of iterations
 Small_Perturbation()
 If Fitness(New_Config) > Fitness(Old_Config)
    If Fitness(New_Config) > Best_Fitness
       Best_Fitness = Fitness(New_Config)
    End-If
    Old_Config = New_Config
 End-If
Return

Scattering()
 p_scattering = 1 - (Fitness(New_Config) / Best_Fitness)
 If p_scattering > random(0,1)
    Old_Config = Random Solution
 Else
    Exploration()
 End-If
Return
```

Figure2 : PCA pseudocode

Nous avons donc créé une classe de preprocessing en utilisant tout simplement toutes les fonctions déjà définies, puis nous avons pipeliné le preprocessing afin de faciliter l'utilisation.

```
class Preprocessing(BaseEstimator):

    def __init__(self, n_components=2):
        self.pca = PCA(n_components = n_components)

    def fit(self, X, y=None):
        return self.pca.fit(X, y)

    def fit_transform(self, X, y=None):
        return self.pca.fit_transform(X)

    def transform(self, X, y=None):
        return self.pca.transform(X)
```

Figure 3 : Preprocessing Class

Prédiction

Nous avons à notre disposition une superbe librairie de fonctions liées au machine Learning : scikit-learn-machine-learning, dont la grande majorité de nos fonctions étudiées sont issues.

Notre premier modèle était le plus simple : Linear Regression de scikit-learn, une simple régression linéaire, qui nous donnait un résultat moyen d'environ 50% de réussite. Comme nous aimons le challenge, nous avons alors décidé de chercher et de tester les autres fonctions de scikit à notre disposition : Decision Tree Regressor (max_depth = 4) nous a alors permis d'avoir le score de 73% de réussite. Mais ce n'était pas assez. Alors on a cherché dans les régressions utilisées par les anciens élèves de l'année dernière, ce qui nous a donné notre meilleur modèle : Gradient Boosting Regressor, avec lequel nous avons atteint 78%.

Le modèle GradientBoostingRegressor utilise plusieurs arbres binaires (i.e. DecisionTreeRegressor). Le premier arbre est fit sur les données de base or le deuxième et le reste ne sont pas fit sur les données de base mais sur les données résiduelles. Ainsi, le modèle final est la somme des tous les modèles.

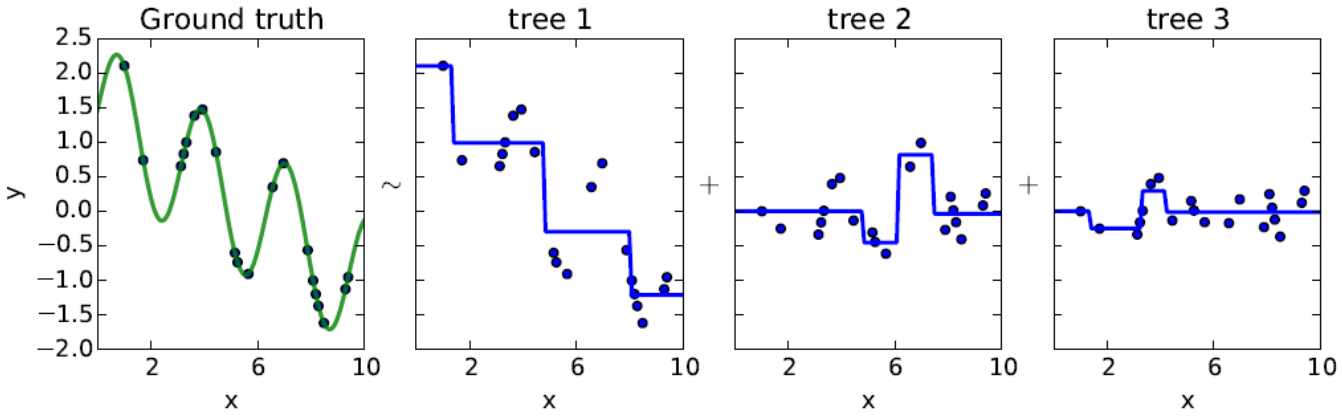


Figure4 : Graphique de comment chaque arbre est entrainé

Afin de réduire le overfitting et optimiser le modèle, il est de coutume de bien paramétrer les hyperparamètres du modèle. Pour cela on peut appliquer l'algorithme suivant :

1. Set `n_estimators` as high as possible (eg. 3000)
2. Tune hyperparameters via grid search.

```
from sklearn.grid_search import GridSearchCV
param_grid = {'learning_rate': [0.1, 0.05, 0.02, 0.01],
              'max_depth': [4, 6],
              'min_samples_leaf': [3, 5, 9, 17],
              'max_features': [1.0, 0.3, 0.1]}
est = GradientBoostingRegressor(n_estimators=3000)
gs_cv = GridSearchCV(est, param_grid).fit(X, y)
# best hyperparameter setting
gs_cv.best_params_
```

Figure5 : Tuning Hyperparamètres Algorithm

3. Finally, set `n_estimators` even higher and tune `learning_rate`.

L'algorithme utilise une méthode appelé Least Squares ou encore nommée Means Square Error qui mesure la moyenne de la différence quadratique entre les prédictions et les observations réelles. Toutes les valeurs calculées qui sont très éloignées des valeurs réelles (i.e. les labels) sont fortement pénalisées par rapport aux prédictions qui sont le moins déviantes.

$$MSE = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}$$

Figure 6 : Means Square Formula

Résultats obtenus dans le défi

Pour évaluer quel modèle est le meilleur, nous utiliserons la technique de validation croisée, plus précisément la validation croisée. Cette technique consiste à diviser l'ensemble original en k échantillons, puis l'un des k échantillons sera sélectionné comme ensemble de validation et l'autre ensemble k-1 sera l'ensemble d'entraînement pour notre modèle.

Le score de performance est calculé grâce à concordance index, puis l'opération est

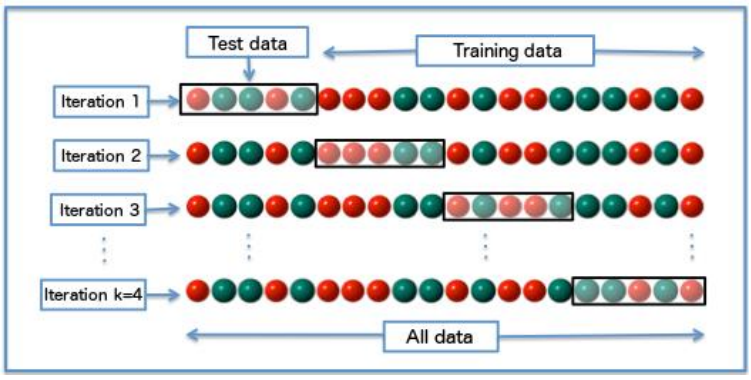


Figure 7 : Cross Validation Principle

répétée en sélectionnant un autre échantillon de validation parmi les k-1 échantillons qui n'ont pas encore été utilisés pour valider le modèle.

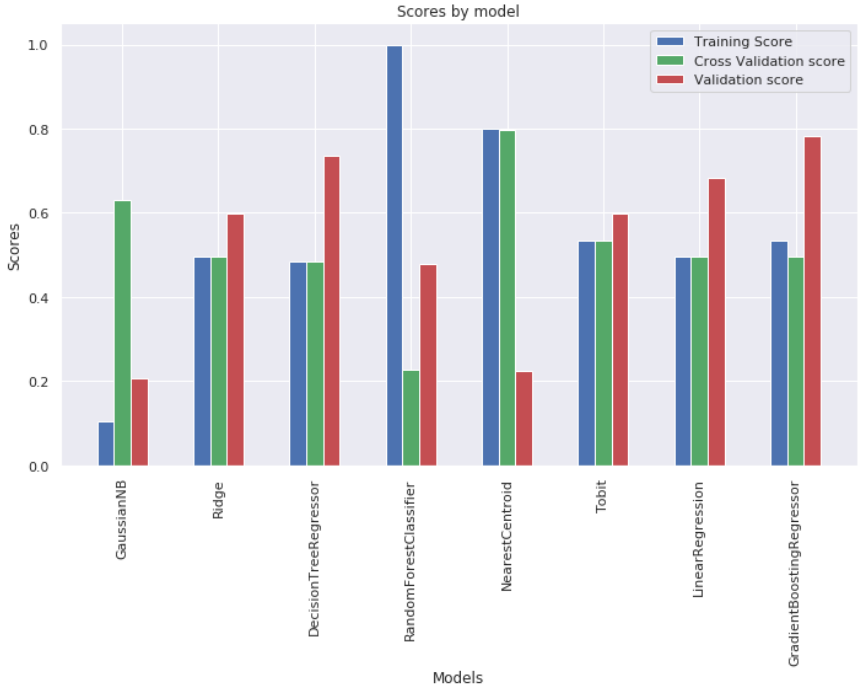


Figure 8 : Diagramme avec le score de chaque modèle

Nous le voyons sur les diagrammes en barres que Gradient Boosting Regressor paraît ne pas très bien faire des prédictions sur les données d'entraînement et de cross-validation. Cependant, sur les données de validation il s'avère le meilleur. Les hyperparamètres utilise pour notre modèle sont :

- Learning_rate = 0.1
- Max_depth = 4
- N_estimators=100
- Subsamples=1.0,
- Min_samples_split=2
- Min_samples_leaf=1

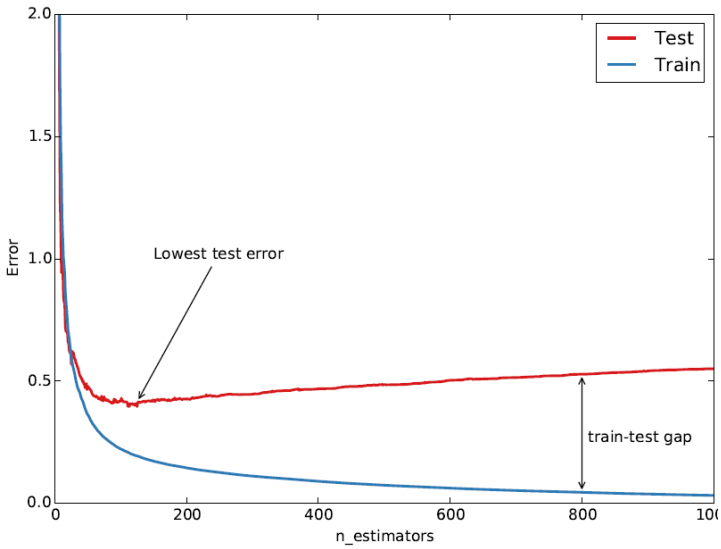
D'après la *Figure 8*, on peut remarquer qu'il y a un modèle en particulier qui nous donne a la fois des bonnes et mauvaises performances. Ce modèle est le Random Forest Classifier mais version regressor. Il nous donne 100% de réussite au training alors qu'avec cross-validation et test il est perd beaucoup en performances. Cela peut s'expliquer par le fait qu'il apprend sur un type de donnes et qu'il est super bon mais lorsqu'il rencontre des nouveaux types de données il ne sait pas comment faire. Cela s'appelle du surapprentissage. Alors que les autres modèles sont pas loin de la moyenne lors de la cross validation et du train.

Grace â la plateforme Codalab, on a pu évaluer les prédictions de chaque model qu'on a implémenté jusqu'à maintenant. Comme on peut le voir dans la figure au-dessus, on remarque que certains algorithmes implémentés font du surapprentissage sur les données d'entrainement (par exemple le RandomForestClassifier aussi le DecisionTreeRegressor peut faire si on augmente le max_depth). Ainsi, avec ces algorithmes on a une performance médiocre par rapport à ceux qui ont une performance normale sur les données d'entrainement et sur Cross-Validation. Par conséquent, le meilleur algorithme implémenté est celui de GradientBoostingRegressor qui nous donne environ 78.3% de prédictions correctes au début mais après on a eu 76.66%.

8790	Ghosts	Development Phase	Apr 28 2019		0.7801
8746	SURVIVERS	Development Phase	Apr 26 2019	version finale 2 ajout du README	0.7800

Figure 9 : Tableau des soumissions publiques

```
test_score = np.empty(len(est.estimators_))
for i, pred in enumerate(est.staged_predict(X_test)):
    test_score[i] = est.loss_(y_test, pred)
plt.plot(np.arange(n_estimators) + 1, test_score, label='Test')
plt.plot(np.arange(n_estimators) + 1, est.train_score_, label='Train')
```



Slow learning by shrinking tree predictions with 0 < learning_rate <= 1

Lower learning_rate requires higher n_estimators

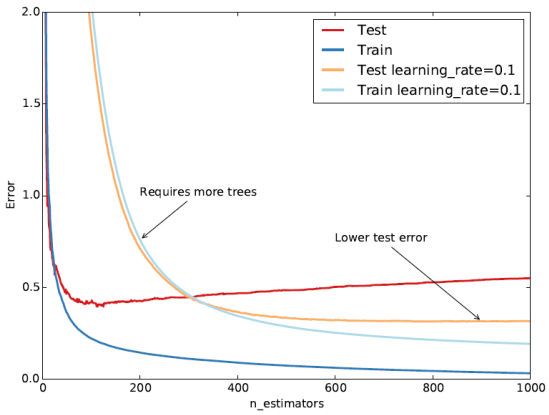


Figure 10 : Graphes de réduction du sur-apprentissage avec les hyperparamètres

Les deux graphiques présentés dans la Figure 10 nous montre le taux d'erreur que le modèle fait sur les données 'Test', 'Train' et comment on a fait pour baisser l'erreur. La courbe rouge représente le taux d'erreur faite sur les données 'Test' et la courbe bleu représente le taux d'erreur faite sur les données 'Train'. L'espace entre les deux courbes s'appelle l'overfitting qui se produit lorsque le modèle il sait très apprendre sur les données d'entrainement alors qu'au test il est mauvais. Sur le graphique de droite on vous présente une manière de réduire ce taux d'erreur. Une manière de le faire est de jouer avec le taux d'apprentissage du modèle, c'est-à-dire de diminuer ou augmenter un peu le taux d'apprentissage. Pour notre modèle et pour notre défi, la façon optimale est de diminuer le taux d'apprentissage a 0.1. Cela nous revient avec des conséquences, par exemple plus d'arbres de décision a utiliser et plus de temps a entrainer mais cela nous réduit taux d'erreur de 50% pour les données test et fait que le surapprentissage est moins important.

Discussion et conclusion

Nous avons donc réussi à résoudre le problème de prédiction, c'est-à-dire prédire des âges de décès de personnes à partir d'informations sur leur corps avec une précision au-dessus des 70%. Le modèle choisit n'est pas parfait c'est pourquoi il a des avantages et désavantages, les voici :

Avantages	Désavantages
Supporte différentes fonctions de perte	Nécessite un paramétrage soigneux
Détecte automatiquement les interactions (non linéaires) entre les caractéristiques.	Lent à entrainer mais a rapide à prédire

Figure10 : Tableau avantages/désavantages du modèle

Le choix du modèle, peut paraître non intuitif or ce modèle s'avère très bon avec les données que nous on a et une fois entraine il fait des prédictions en temps records par rapports aux autres modèles. (Cf. Figure 11).

	Train time [s]	Test time [ms]	MAE
Mean	-	-	0.4635
Ridge	0.006	0.11	0.2756
SVR	28.0	2000.00	0.1888
RF	26.3	605.00	0.1620
GBRT	192.0	439.00	**0.1438**

Figure 11 : Tableau de temps de quelques modèles

Afin de maximiser nos prédictions on voudrait envisager d'implémenter un algorithme qui prend en compte les données censurées et avec les données du preprocessing qu'on cherche les meilleurs hyperparamètres pour notre cas.

Par ce projet, nous avons appris à utiliser correctement des packages sur python, les régressions, et qu'on pouvait prédire beaucoup de choses tant que l'on possède les données adéquates en amont. Si l'on devait donner un conseil aux étudiants de l'an prochain, ils devraient rechercher par eux-mêmes les fonctions nécessaires pour leur projet, internet nous le permet, et donne de très bons résultats

Bonus

Dataset	Num. Examples	Num. Features	Sparsity	Has categorical variables	Has missing data
Training	19 297	10	72 281/(19 297 x 10) = 0.3745	Yes	No (but censored resembles missing)
Validation	2 413	10	9060/(2413 x 10) = 0.3754	Yes	No (but censored resembles missing)
Test	2 412	10	9052/(2412 x 10) = 0.3752	Yes	No (but censored resembles missing)

Figure 12 : Tableau 1 : Statistiques sur les données

Method	NaiveBayes or Gaussian Classifier	Linear regression or SVM	Decision Tree	Random Forest	Nearest Neighbors	GradientBoostingRegressor
Training	0.1063	0.4970	0.4914	1.0000	0.7991	0.5195
CV	0.63	0.50	0.49	0.22	0.80	0.50
Valid(action)	0.208448403569	0.683989487031	0.736217383794	0.478958368593	0.224931346799	0.783217626851

Figure 13 : Tableau 2 : Résultats préliminaires

Cross-validation : La cross-validation est une méthode qui permet d'estimer la fiabilité d'un modèle. Il existe plusieurs méthodes de cross-validation mais elles sont toutes basées sur une technique d'échantillonnage. Par exemple, dans notre cas nous avons divisé l'ensemble des données en un nombre k d'échantillons puis sélectionner un échantillon qui nous sert d'élément de validation et les k-1 échantillons restants constitue l'ensemble d'apprentissage pour notre modèle. Le score de performance est calculé puis l'opération est répétée k fois jusqu'à avoir utilisé chaque échantillon comme élément de validation. Nous pouvons à la fin calculer la moyenne afin d'avoir la fiabilité de notre modèle et le comparer à un autre modèle.

Sur-apprentissage : On parle de sur-apprentissage (Overfitting) lorsqu'un modèle prédictif est trop spécialisé, trop précis. En effet, lors de la phase d'apprentissage si les données sont trop précises, le modèle s'adaptera très bien à ces données, trop bien, et il capturera toutes les variations ainsi que les erreurs. Par conséquent il aura du mal à généraliser ce qu'il a appris sur des données qu'il n'a jamais vues.

La métrique :

Afin d'évaluer notre modelé, il est nécessaire d'avoir une métrique. Ainsi, la métrique qu'on utilise est l'indice de concordance. L'indice de concordance permet d'évaluer à quel point notre modèle est performant en faisant un rapport entre le temps de survie sur la probabilité de survie. Plus la valeur est haute, plus notre modèle est performant.

Voici le lien du code (line 31 - 50) :

https://github.com/upsudghosts/ghosts/blob/master/starting_kit/scoring_program/my_metric.py

Bibliographie

- Harald Steck and Balaji Krishnapuram and Cary Dehing-oberije and Philippe Lambin and Raykar, Vikas C. *On Ranking in Survival Analysis: Bounds on the Concordance* Index. In J. C. Platt and D. Koller and Y. Singer and S. T. Roweis, Advances in Neural Information Processing Systems 20, 1209-1216. Curran Associates, Inc. 2008
- Tobit Model : https://github.com/jamesdj/tobit (Consulte le 27/02/2019)
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html (Consulté le 23/03/2019)
- Optimizing Hyperparameters **:** https://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting (Consulte le 26/03/2019)
- Bibliothèque scikit avec les modèles :www.scikit-yb.org (Consulté le 20/02/2019)
- **Pseudo-code k-nearest neighbor algorithm:** https://www.researchgate.net/figure/Pseudocode-for-KNN-classification_fig7_260397165 (Consulte le
- **Dr. Saed Sayad** web-site: Machine Learning Decision Trees**:** https://www.saedsayad.com/decision_tree_reg.htm (Consulte 1/04/2019)
- **Yellowbrick: Machine Learning Visualization :** https://www.scikit-yb.org/en/latest/ (Consulte le 17/02/2019)
- Article *« Machine learning: an introduction to mean squared error and regression lines »* by **Moshe Binieli** https://medium.freecodecamp.org/machine-learning-mean-squared-error-regression-line-c7dde9a26b93

Sommaire