Rapport de projet Machine learning

Détection des chutes à l'aide des algorithmes de Machine Learning

---

Professeure : **Mme. Mathilde MOUGEOT**

Élaboré par :

Chaïma BOUGHANMI boughanmichaima9@gmail.com
Vhiny MOMBO vmombo78@gmail.com
M2 Data Science

Année universitaire: 2021 / 2022

1- Introduction

Les chutes constituent un risque majeur pour la santé et l'autonomie des personnes âgées. Des systèmes de détection des chutes rapides et fiables peuvent améliorer les chances de survivre à l'accident et de faire face à ses conséquences physiques et psychologiques. Des recherches récentes ont trouvé des solutions, qui présentent toutes d'importants inconvénients, dont l'un est l'intrusion dans la vie du patient.
Cet article[1] propose un nouveau système de surveillance pour la détection des chutes basé sur un capteur de sol sensible en matériau piézoélectrique et une approche d'apprentissage automatique.

2- Objectif et méthodologie du travail

L'objectif de ce projet est d'étudier la performance de plusieurs algorithmes de machines learning à détecter automatiquement les chutes grâce aux indicateurs fournis.
Pour ce faire nous allons suivre les étapes suivantes en utilisant la programmation Python :
a) Entraîner et tester différents classificateurs binaires de machine learnig à utiliser, à savoir :
I- Naive Bayes II- LDA III- QDA IV- Logistic regression V- KNN VI- Decison Tree VII- Random Forest VIII- Adaboost XI- Gradient boosting.
b) Comparer ces classificateurs pour détecter automatiquement les chutes.
c) Tirer des conclusions sur les performances et les modèles utilisés.

3- Description du jeu de données

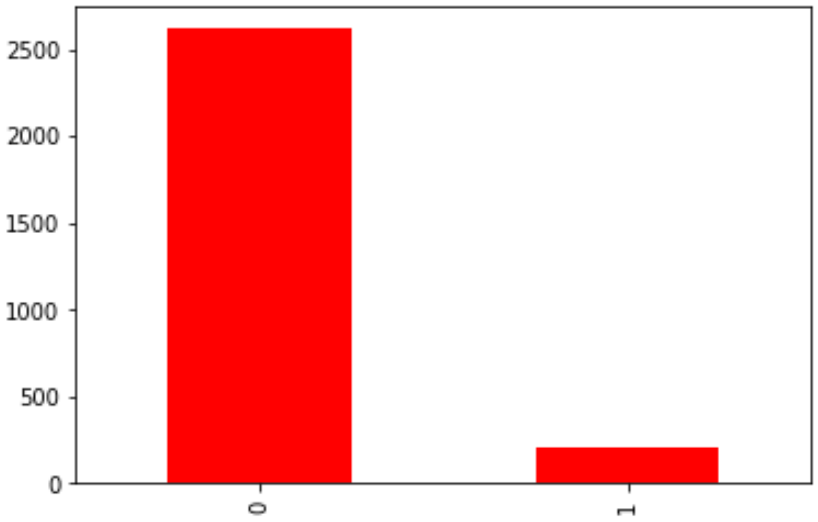
La base de données a été constituée à partir d'acquisitions de 28 volontaires simulant des chutes et d'autres comportements.
Le jeu de données 'falldataproject.csv' contient un échantillon d'observations caractérisant la marche puis la chute (ou non) de plusieurs personnes. La marche est décrite grâce à 87 indicateurs et l'indicateur de chute (ou non) est décrit par une étiquette binaire. Les variables correspondent à plusieurs indicateurs calculés sur les signaux bruts ou sur le signal dérivé ou sur l'énergie des signaux (ff signifie Fast Fourier). signaux (fft signifie Fast Fourier Transform). Pour des raisons de confidentialité, toutes les variables (sauf le label) ont été mises à l'échelle et le nom des variables a été tronqué.
Pour en avoir un apperçu, voir l'extrait ci-dessous.

	obs	raw_feat_X1	raw_feat_X2	raw_feat_X3	raw_feat_X4	raw_feat_X5	raw_feat_X6	raw_feat_X7	raw_feat_X8	raw_feat_X9
0	0	0.249744	-0.162770	0.223727	0.393904	-0.154366	0.128968	1.090661	0.913849	0.505526
1	1	0.385843	-0.660978	-0.127798	-0.205710	-0.160936	0.111606	0.171391	2.889781	0.377333
2	2	3.344528	-4.535931	0.165140	-0.228745	3.203818	3.379462	1.089901	2.097552	0.877990
3	3	3.190676	-2.884463	-1.153080	-0.698292	1.868221	2.493077	2.546198	3.817391	3.711000
4	4	2.338575	-2.699941	-0.069211	-0.025849	1.420714	2.137326	1.097388	2.101987	1.200319

5 rows × 89 columns

Nous avons 87 variables explicatives et 2821 observations. Le but étant de predire 'FALL'.

[1] https://ieeexplore.ieee.org/document/8037597



Il y a 202 falls pour 2619 marches normales. Il n'y a pas de données manquantes, mais le dataset n'est pas équilibré, c'est pourcela on a essayé d'utiliser SMOTE[2] pour équilibrer nos données. Ensuite nous avons essayé de detecter les chutes en utilisant les méthodes vues en classe.

Préparation des données
On note X les variables explicatives et Y la variables target, à expliquer qui est la variable 'FALL' du dataset. On prend pas la variable 'obs' car elle correspond juste à un index pour les données.
Les données étant déséquilibrer on a creer un fonction qui permettrait d'oversample la classe minoritaire afin d'avoir un peut plus d'exemple de ce classe. Cette fonction se base sur SMOTE qui est une excellente technique d'oversampling.

Nous avons divisé notre base de données en 75% pour l'entrainement (X_train_originel, Y_train_originel) et 25% pour le test (X_test_originel, Y_test_originel).

Dans la suite de document, on présentera les résultats obtenues sur un jeu de données de train réequilibré (voir le jupyter Notebook).

4- Les algorithmes de machines learning utilisés et comparaison des résultats

I- Naive Bayes- Classificateur naif de Bayes
Il s'agit d'un type de classification simple (« naïf ») reposant sur le théorème de Bayes, qui est lui-même un résultat de base issu de la théorie des probabilités. Le principe mathématique consiste en une forte indépendance des hypothèses simplificatrices « naïves » impliquant l'indépendance des variables.
On a calculé un erreur de cet algorithme qu'on a appélé E_{test} définie en vérifiant que y_{test} sont différentes de y_{pred} par: $E_{test} = \frac{y_{test}+y_{pred}}{y_{test}}$.

II- LDA: Analyse discriminante linéaire
Est une technique de réduction de la dimensionnalité qui est couramment utilisée pour modéliser les différences entre les groupes, c'est-à-dire pour séparer deux ou plusieurs classes. Elle applique l'approche générative pour la classification. Elle est basée sur l'hypothèse que chaque classe peut être modélisée par une distribution gaussienne et que toutes les classes partagent la même matrice de covariance.

III- QDA: Analyse discriminante quadratique
QDA est similaire à LDA mais sans l'hypothèse que les classes partagent la même matrice de covariance, c'est-à-dire que chaque classe a sa propre matrice de covariance. Dans ce cas, la frontière entre les classes est une surface quadratique au lieu d'un hyperplan.

IV- Logistic regression
Est un processus de modélisation de la probabilité d'un résultat discret en fonction d'une variable d'entrée. La

[2]https://arxiv.org/pdf/1106.1813.pdf

régression logistique la plus courante modélise un résultat binaire, c'est-à-dire quelque chose qui peut prendre deux valeurs, comme vrai/faux, oui/non, etc.

V- KNN- K plus proche voisins
L'algorithme KNN part du principe que les choses similaires sont proches les unes des autres. KNN fonctionne en trouvant les distances entre une requête et tous les exemples dans les données, en sélectionnant le nombre spécifié d'exemples (K) les plus proches de la requête, puis en votant pour l'étiquette la plus fréquente (dans le cas de la classification) ou en faisant la moyenne des étiquettes (dans le cas de la régression).

VI- Decision Tree- Arbre de décision
C'est un outil d'aide à la décision représentant un ensemble de choix sous la forme graphique d'un arbre. Les différentes décisions possibles sont situées aux extrémités des branches (les « feuilles » de l'arbre), et sont atteintes en fonction de décisions prises à chaque étape.Il fonctionne en appliquant de manière itérative des règles logiques très simples (typiquement des séparations de données par « hyperplan », généralisation d'un plan à plus de 2 dimensions), chaque règle étant choisie en fonction du résultat de la règle précédente.

VII- Random Forest- Forêt aléatoire
Il effectue un apprentissage en parallèle sur de multiples arbres de décision construits aléatoirement et entraînés sur des sous-ensembles de données différents. Le nombre idéal d'arbres, qui peut aller jusqu'à plusieurs centaines voire plus, est un paramètre important : il est très variable et dépend du problème. Concrètement, chaque arbre de la forêt aléatoire est entrainé sur un sous ensemble aléatoire de données selon le principe du 'bagging', avec un sous ensemble aléatoire de features (caractéristiques variables de données) selon le principe des « projections aléatoires ». Les prédictions sont ensuite moyennées lorsque les données sont quantitatives ou utilisés pour un vote pour des données qualitatives, dans le cas des arbres de classification.

IX- AdaBoost
C'est un méta-algorithme de boosting, il peut être utilisé en association avec de nombreux autres types d'algorithmes d'apprentissage afin d'en améliorer les performances. Les sorties des autres algorithmes (appelés classifieurs faibles) sont combinées en une somme pondérée qui représente la sortie finale du classeur boosté. AdaBoost est adaptatif dans le sens où les classeurs faibles subséquents sont ajustés en faveur des échantillons mal classés par les classeurs précédents.

X- Gradient Boosting
Le boosting de gradient est un autre type de boosting en machine learning. Il repose fortement sur la prédiction que le prochain modèle réduira les erreurs de prédiction lorsqu'il sera mélangé avec les précédents. L'idée principale est d'établir des résultats cibles pour ce prochain modèle afin de minimiser les erreurs.

Comparaison des performances des algorithmes

Pour la comparaison des algorithmes, on s'est basé sur 5 métriques, principalement : **accuracy, précision, recall, f1 et AUC(Area Under the Curve)**. Au vu de la nature du problème et du jeu de données, les métriques les plus importantes sont le recall et la précision. Elles sont très importantes, de très bons indicateurs de performance pour nos algorithmes. Car plus le recall donne une mesure de la précision avec laquelle nos modèles identifient les vrais chutes tandis que la précision (rapport des vrais positif sur tous les positifs) est plus elle est haute, moins le modèle se trompe sur ce qu'est réellement une chute.

Dans un permier temps, on a entrainé les différents algorithmes sur les données d'entrainement puis on a évalué leurs performances sur les données de test sans faire de recherche d'hyerparamètres.

Dans le tableau ci-dessous on a récapitulé les performances de tous les algorithmes qu'on a utilisés :

	gb	LDA	QDA	logreg	knn	tree	RF	Adaboost	GB
accuracy	0.967422	0.992918	0.983003	0.995751	0.991501	0.990085	0.992918	0.988669	0.991501
precision	0.716216	1.000000	0.820896	1.000000	1.000000	0.961538	1.000000	0.960784	0.962264
recall	0.963636	0.909091	1.000000	0.945455	0.890909	0.909091	0.909091	0.890909	0.927273
f1	0.821705	0.952381	0.901639	0.971963	0.942308	0.934579	0.952381	0.924528	0.944444
auc	0.965689	0.954545	0.990783	0.972727	0.945455	0.953009	0.954545	0.943918	0.962100

Cette approche n'étant pas assez robuste, il est difficile de généraliser ces modèles entrainés pour la prédiction car la division de dataset peut fortement biaiser nos modèles. Ainsi, à l'aide d'un **gridsearch**, on a réalisé une recherche des meileurs hyperparamètres (les plus importants) pour KNN, Decision Tree, Random Forest, Adaboost et Gradient Boosting, puis évaluer la performance des modèles en faisant un **k-folds** (avec k = 10) sur les données d'entrainements.
La recherche d'hyperparamètres nécessitant une métrique à maximiser. On a choisi le recall car on veut un algorithme capable de bien détecter les chutes. Aussi pour la grille de recherche de ces hyperparamètres on s'est basé sur des valeurs empiriques tout en essayant de ne pas avoir des temps de calculs trop longs. Ainsi quelques modèles finaux sont: KNN (K = 1), Adaboost (n_estimator = 81, learning_rate=0.1), Gradient Boosting (n_estimators = 105, learning_rate = 0.3).
On peut voir plus de détails dans le notebook. Ceci nous conduit aux résultats suivants de la figure 1 :

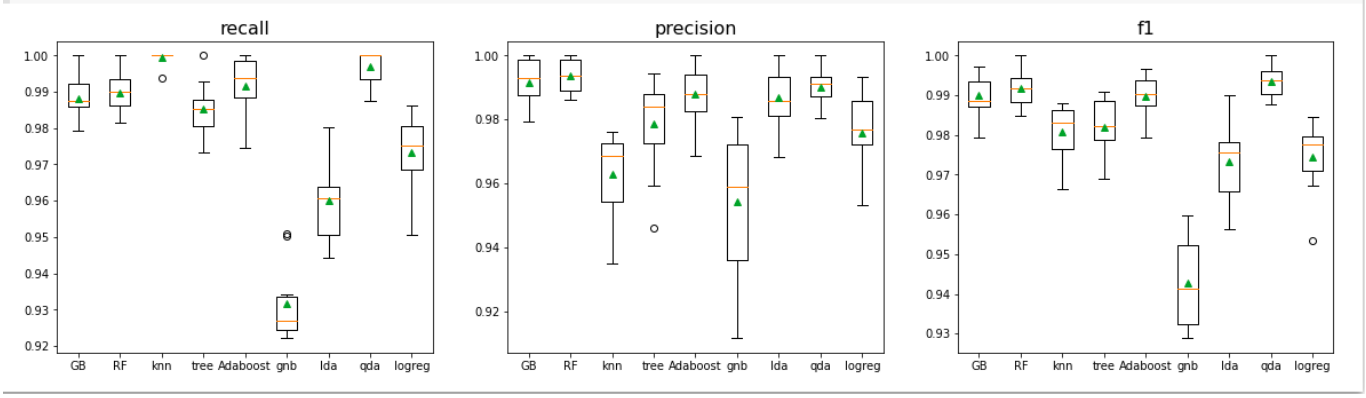


Figure 1: Boxplot des évaluations des modèles sur le jeu données test. De gauche à droite on a le recall, la precision et le F1-score. On distingue une bonne performance globale avec Random Forest et QDA

Le recall et la precision de nos algorithmes sont en moyenne supérieurs à 92% ce qui est une bonne performance. Random forest se distingue des autres par ses scores élévés et ses écarts-types très petits. On tient également à noter un recall très important pour le KNN qui aurait fait de cet algorithme un bon candidate mais au vu de la valeur moyenne et l'écart type de sa précision il n'est plus adapté. La méthode Naives Bayes semble l'algorithme le moins adapté aux problèmes, on aurait pu se dire que c'est dû aux ré-équilibrage du jeu de données éffectué ci-dessus mais après vérification on trouve qu' au contraire le ré-équilibrage améliore la performance de nos différents algorithmes (voir figure 2) .

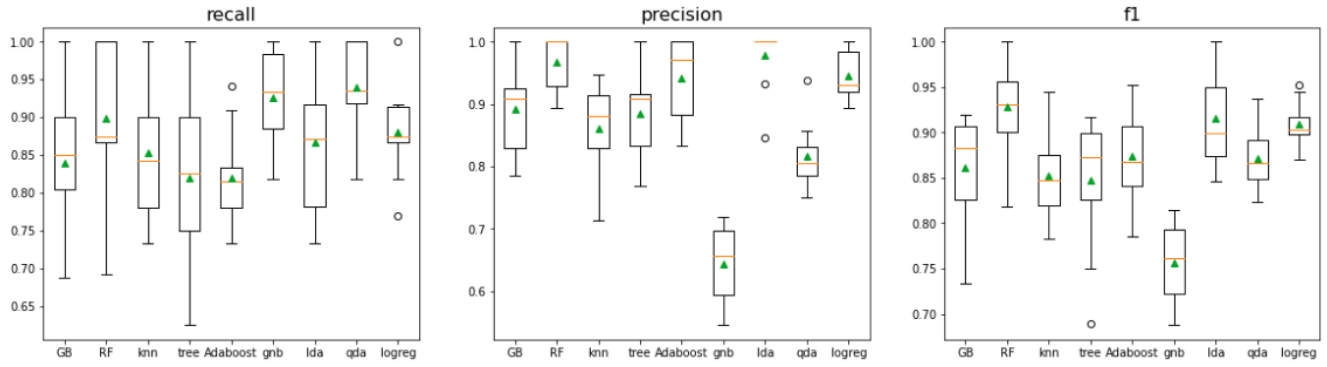


Figure 2: Récapitulatif des évaluations des modèles en faisant des k-folds sur dataset non équilibré . De gauche à droite on a : le recall, la precision et le F1-score. On voit une mauvaise performance globale par rapport à la figure 1

Enfin, on a regardé la performance des algorithmes sur le jeu de données de test (X_test_originel, Y_test_originel) qu'on n'a pas évalué en utilisant aucun modèle. Ceci est resumé dans le tableau suivant.

	GB	RF	knn	tree	Adaboost	gnb	lda	qda	logreg
accuracy	0.987252	0.990085	0.971671	0.975921	0.983003	0.973088	0.988669	0.991501	0.990085
precision	0.907407	0.960000	0.753846	0.800000	0.847458	0.742857	0.909091	0.927273	0.925926
recall	0.924528	0.905660	0.924528	0.905660	0.943396	0.981132	0.943396	0.962264	0.943396
f1	0.915888	0.932039	0.830508	0.849558	0.892857	0.845528	0.925926	0.944444	0.934579
auc	0.958436	0.951299	0.950013	0.943642	0.964807	0.976783	0.967870	0.978069	0.968635

Figure 3: Récapitulatif des évaluations des modèles sur les données de test. On retrouve la bonne performance de Random Forest et QDA

La tableau de la figure 3 nous montre que les algorithmes RF, QDA et Logistic Regression(logreg) ont de très bon f1, notamment QDA qui surclasse tous les autres algorithmes avec un recall de 96% pour une precision de 92% suivi de logreg (recall = 94%, precision = 92%) et RF(recall = 90%, precision = 96%). Sur ce de test, QDA performe très bien, c'est à quoi on pourrait s'attendre car les valeurs obtenus rentre dans les intervalles de confiance fournis par les boxplots précédents figure 1. Cependant bien qu'ayant mieux performé que Random Forest, sur le jeu de test, il reste très difficile de dire que sur ce problème, le modèle logistique correspond mieux que le Random Forest au vu de la figure 1.

5- Conclusion

L'objectif de ce projet est d'étudier la performance de plusieurs algorithmes de machines learning à détecter automatiquement les chutes grâce aux indicateurs fournis.
En regardant, les valeurs de recall et precision (par extension F1-score) qui sont les métriques les mieux adaptées au problème, il en sort que QDA et Random Forest sont assez comparables et fournissent les meilleurs résultats.