Institut Mines Télécom Atlantique

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Rapport de stage :

Identification automatique de l'apnée du sommeil par signal ECG

Juillet 2018 - Août 2018

Réalisé par :
HAFID Fayçal

Encadré par :
RADI Malika

Organisme d'accueil :

Centre de Développement et des Technologies Avancées (CDTA)
Baba Hassen - Alger, Algérie

C.D.T.A

Remerciements

Je tiens à remercier vivement toute l'équipe du CDTA qui m'a si bien accueilli et qui a considérablement facilité mon insertion en leur compagnie.
Je remercie particulièrement Mme RADI Malika qui a été une encadrante très disponible et toujours à l'écoute, qui a su me transmettre ses connaissances pratiques en Data Mining et me guider à la rédaction du présent rapport.
Je ne saurais oublier de mentionner Mme GHANEM Khalida et Mr OUADAH Noureddine pour leur aide à l'effectuation de ce stage et pour leur disponibilité.
Enfin, je remercie mon ami Kaci Islem MAKOUR pour les échanges d'informations pertinentes qu'on a eues durant ce stage, ainsi que ma famille pour leur soutient permanent.

Résumé

L'identification automatique de l'apnée du sommeil est importante afin d'alléger le travail des cliniciens pour l'analyse d'un grand volume de données et l'accélération du diagnostic. En effet, lors d'une apnée le rythme cardiaque ralentit à son début et s'accélère à la fin, ce qui affecte sa variabilité. Le travail effectué est l'utilisation d'une seule dérivation du signal cardiaque afin de diagnostiquer une apnée du sommeil et comparer la performance de plusieurs algorithmes de classification (SVM , Naive Bayes, kNN, Extreme Machine Learning, Arbres de Décision, et Ensemble Learning) et de plusieurs choix de features.

Les algorithmes sont testés sur une base de donnée de 35 patients préalablement annotée par minute d'enregistrement. Les paramètres qui déterminent une bonne classification sont extraits du rythme cardiaque et de l'étude de sa variabilité HRV (Heart Rate Variability).

Introduction

L'apnée du sommeil est un trouble durant lequel la personne atteinte cesse involontairement de respirer pendant son sommeil. Une apnée peut durer de dix secondes jusqu'à deux minutes, selon la sévérité. Cet arrêt de la respiration fait baisser le niveau d'oxygène dans le sang et inhibe un cumul de dioxyde de carbone. Lorsque le niveau d'oxygène dans le sang atteint un seuil critique, le corps envoie le signal au cerveau de se réveiller pour le forcer à reprendre la respiration.

Le diagnostic de l'apnée du sommeil relève d'un processus pouvant être très gourmand en temps et en ressources financières, laissant un grand nombre de personnes dans le monde non diagnostiquées et donc non traitées. Aujourd'hui, la méthode standard pour le diagnostic de ce trouble est l'usage d'une polysomnographie dans le laboratoire de sommeil d'un hôpital. Une polysomnographie est une étude compréhensive du sommeil se faisant à l'aide de plusieurs mesures incluant une électroencéphalographie (EEG), une électro-oculographie (EOG), une électromyographie (EMG), une électrocardiographie (ECG) et la saturation en oxygène du sang. La plupart de ces signaux se mesurent à l'aide de capteurs et d'électrodes et requièrent des câblages qui restreignent les mouvements du patient durant son sommeil. Ensuite, les données doivent être analysées par un médecin spécialisé pour établir un diagnostic correct et entamer un traitement adapté. Le coût annuel pour une polysomnographie varie entre 4000$ et 6000$ par patient.

Le *Data Mining* (fouille des données) désigne l'analyse de données depuis différentes perspectives et le fait de transformer ces données en informations utiles, en établissant des relations entre les données ou en repérant des patterns**.** Ce sous-domaine de l'informatique qui croise le Machine Learning et l'Intelligence Artificielle permet d'établir une analyse prédictive et une classification. Ses possibilités et ses avantages en font un outil pertinent pour la détection des anomalies respiratoires qui occurrent chez les patients atteints de l'apnée du sommeil.

Durant ce stage, l'objectif est d'automatiser le diagnostic de l'apnée du sommeil uniquement à partir du signal électrocardiogramme (ECG) en construisant différents modèles de classification classiques du Data Mining et faire une analyse comparative de leurs performances et des différentes approches essayées. Les données d'ECG utilisées sont extraites de la base de données médicales PhysioNet, une source fiable largement utilisée par un grand nombre de chercheurs à travers le monde. Le travail a été réalisé sur MATLAB 2014.

Les informations citées dans cette section sont extraites de la référence [1]

Plan du rapport

1. Apnée du sommeil

L'apnée du sommeil est un trouble du sommeil identifié par des anomalies respiratoires ou une cessation complète de la respiration pendant de longues périodes. Par conséquence, le niveau d'oxygène dans le sang du patient commence à chuter. Si le niveau d'oxygène atteint un seuil critique, le cerveau du patient le force à se réveiller pour restaurer un niveau d'oxygène normal. Ce supplice nocturne cause au patient un manque de sommeil et il est estimé que 70-80% des personnes atteintes par l'apnée du sommeil n'en sont pas diagnostiquées et restent donc non traitées. Le non traitement de l'apnée peut être la cause principale d'un développement du diabète, de maladies cardiaques et de dépression.
L'*apnée du sommeil obstructive* (ASO) est perçue comme l'apnée du sommeil la plus répandue comme elle constitue 84% des cas d'apnée diagnostiqués. L'ASO se caractérise par une obstruction partielle ou complète des voies respiratoires, conduisant à la réduction ou l'arrêt du passage de l'oxygène vers les poumons. Lorsque les voies respiratoires supérieures sont bloquées, les poumons tentent de forcer l'air à travers le diaphragme et les canaux d'air résultant en ronflements.
Les ronflements représentent donc le principal symptôme lié à l'apnée du sommeil obstructive.

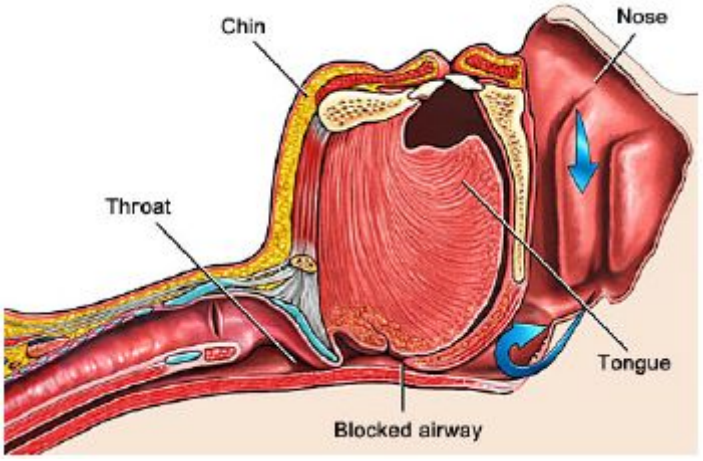


Figure 1 [1] : le diaphragme qui bloque les voies respiratoires

Les informations citées dans cette section sont extraites de la référence [1].

2. Data Mining

Le *Data Mining*, ou fouille de données, est un terme désignant un processus calculatoire qui permet de retrouver des *patterns* (modèles) pertinents et intéressants pour un groupe de données. Ces patterns sont souvents distingués comme étant soit des modèles prédictifs ou des modèles descriptifs. Le *Data Mining* joue un rôle important quand le nombre de données est trop grand pour pouvoir être traité par un être-humain à la main.

2.1 Méthodes de Data Mining

On distingue deux **méthodes** de *Data Mining*, la classification et la régression.

2.1.1 Classification

La classification est une méthode qui permet de répartir les objets d'un groupe de données en classes ou catégories en se basant sur les similarités entre les objets d'une même classe. La procédure consiste à utiliser un grand groupe de données pour entraîner un modèle, appelé *Classifier* (classificateur) qui fera un plan pour relier les caractéristiques des objets avec leur classe respective. Le modèle entraîné sert alors à classifier de nouvelles données selon leurs caractéristiques et à prédire une classe pour chacun d'entre eux.

2.1.2 Régression

La régression est un concept plus large que la classification car les méthodes de régression permettent aussi de faire de la classification. Tandis qu'un classificateur permet de déterminer la nature d'un objet, un régresseur permet de prédire ou d'estimer une sortie en se basant sur des probabilités et des statistiques sur les relations entre les données.

2.2 Types de modèles

La plupart des méthodes de Data Mining utilise un modèle. Le choix du modèle d'apprentissage dépend de l'objectif voulu et chacun possède des caractéristiques différentes.

2.2.1 Modèles prédictifs

Les modèles prédictifs sont les modèles qui sont capables de prédire une sortie à partir de statistiques et d'attributs de données. Les modèles utilisés dans notre étude (kNN, SVM, Naive Bayes…) sont des modèles prédictifs, la sortie qu'on cherche à prédire étant l'absence ou la présence d'apnée à partir d'attributs qu'on va extraire des données en notre possession.

2.2.2 Modèles descriptifs

Les modèles descriptifs utilisent généralement les données après les avoir traitées pour générer des relations entre elles et des patterns. Un modèle descriptif permet de résumer les similarités et les différences parmi les données qui lui sont fournies. Les méthodes de *clustering* produisent de bons modèles descriptifs car ils permettent de regrouper entre eux des objets qui n'ont aucun lien trivial.

2.3 Learning (Apprentissage)

Le résultat de chaque type de modèle possède lui aussi différentes propriétés d'apprentissage et sera subdivisé en deux types d'apprentissage détaillés ci-dessous :

2.3.1 Supervised Learning (Apprentissage supervisé)

Les méthodes qui utilisent de l'apprentissage supervisé ont généralement besoin d'un jeu de données (*dataset*) avec lequel elles seront entraînées. L'entraînement d'un modèle demande comme entrée le jeu de données comme attributs, chacun appairé avec la classe qui lui correspond. La méthode d'apprentissage supervisé produit alors un modèle capable de classifier les nouvelles instances qui lui seront fournies en entrée.
La qualité (en terme de précision) du modèle produit est directement lié à la qualité du jeu de données qui ont servi à l'entraînement, en matière d'équilibre, de diversité, quantité, etc... Une haute qualité de données d'entraînement conduit généralement à de meilleures performances.
Parmi les problèmes qu'on peut rencontrer avec les méthodes d'apprentissage supervisé, on retrouve l'*overfitting* : fournir trop d'attributs/features d'entraînement qui fait que le modèle sera trop conditionné aux modèles d'entraînement et ne sera pas capable de généraliser en dehors de ces données, et/ou lui fournir des données non équilibrées. Par exemple dans notre étude, si on entraîne un modèle en lui donnant des données avec que de l'apnée, sa performance sera probablement mauvaise car il ne saura pas à quoi ressemblent les attributs d'une observation qui ne présentera pas d'apnée.
De même, avoir plus d'attributs/features que d'observations dans un jeu de données mènera à un modèle possédant de mauvaises propriétés de prédiction.

2.3.2 Unsupervised Learning (Apprentissage non supervisé)

Contrairement à l'apprentissage supervisé, le non supervisé ne prend pas en compte l'appartenance à une classe (*unlabeled data*). Au lieu de donner des exemples d'attributs classifiés (accompagnés de la classe qui leur correspond) pour dire à l'algorithme quel objet est correct et lequel ne l'est pas, l'algorithme essaie par lui-même de regrouper des objets ensemble à partir de leurs attributs. Les réseaux de neurones et l'algorithme *K-means* sont des méthodes classiques d'apprentissage non supervisé.

2.4 Types d'attributs/features

Il y a deux principaux types de données qu'on peut rencontrer, même s'ils peuvent parfois être confondus.

2.4.1 Attributs numériques

Les attributs numériques peuvent prendre n'importe quel ensemble de valeurs numériques, fussent-elle entières ou réelles, discrètes ou continues. L'âge ou la masse d'un individu sont des exemples d'attributs numériques.

2.4.2 Attributs catégoriques

Les attributs catégoriques sont un ensemble de valeurs qui ne sont pas forcément sous la forme d'un nombre, par exemple "true" ou "false", "male" ou "female", … La plupart des attributs catégoriques peuvent aussi être transformés et représentés sous la forme d'attributs numériques tout en gardant leurs propriétés catégoriques (par exemple les sexes possibles pour une personne peuvent être codés en 0 et 1).
Que les données soient numériques ou catégoriques, on peut généralement les représenter sous la forme d'une matrice.

Les informations citées dans les sections 2.1 à 2.4 sont extraites de la référence [1]

2.5 Classificateurs

2.5.1 k Nearest Neighbors

Le classificateur kNN, pour *k Nearest Neighbors* (les k plus proches voisins) est l'un des modèles les plus simples mais qui donne des résultats tout aussi bons que ceux des méthodes plus sophistiquées. Ce classificateur prédit la classe d'un objet à partir de la classe majoritaire des k objets les plus proches de lui parmi les données. kNN est dit un apprenant paresseux, car il ne procède pas à une phase d'entraînement : en effet, il ne fait que stocker les données dites d'entraînement et calcule une distance pour chaque nouvel objet à tester.
La valeur choisie du paramètre k (nombre des plus proches voisins dont on prendra la classe majoritaire) est cruciale à la performance et diffère d'un jeu de données à un autre. En effet, la Figure 2 montre un cas où le choix de k=3 et le choix de k=5 conduiraient chacun à un résultat différent (objet rouge pour le premier choix, objet bleu pour le second).

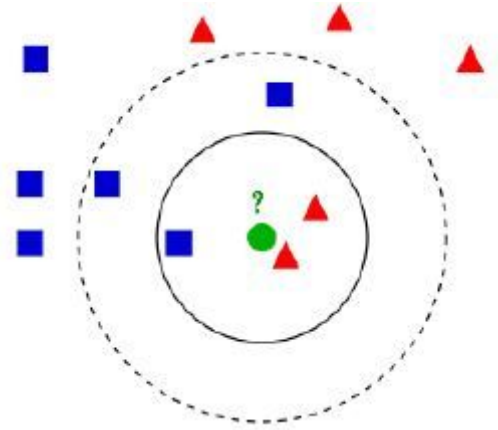


Figure 2 : illustration soulignant l'impact du choix du paramètre k [5]

Afin d'éviter une égalité de classes (deux classes majoritaires), on prend k comme étant impair (respectivement pair) si les classes sont d'un nombre pair (respectivement impair).
De même, plusieurs distances peuvent être envisagées, à savoir les distances euclidienne, *cityblock*, *chebychev*…

Les informations citées dans cette section sont extraites de la référence [1]

2.5.2 Support Vectors Machine

Les machines à vecteurs de support sont l'une des méthodes les plus utilisées en pratiques. Se basant sur des techniques poussées en mathématiques, il a été prouvé qu'on peut linéairement séparer un jeu de données par un hyperplan avec une certaine marge de tolérance. On parle alors de *hard margin* (marge dure) et de *soft margin* (marge souple). Lorsqu'on choisit d'employer une marge dure, les objets doivent obligatoirement être d'un côté où de l'autre de la marge (voir Figure 3) contrairement à la marge souple qui tolère à une minorité d'objets d'empiéter la frontière définie par la marge (voir Figure 4).

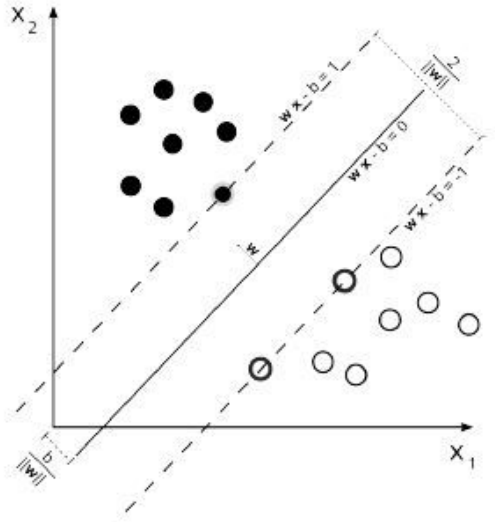


Figure 3 : Classification par marge dure, aucun objet se trouvant à l'intérieur de la marge représentée en traits discontinus [6]

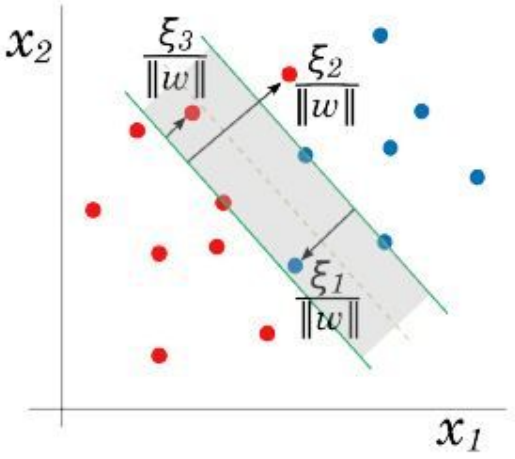


Figure 4 : Classification par marge souple, présence de minorités à l'intérieur de la marge représenté en fond coloré [7]

Évidemment, l'emploi d'une marge dure conduit à une plus grande précision en terme de séparation en deux (ou plus) classes différentes car elle est plus rigoureuse et tolère moins d'erreurs, mais en pratique il n'est pas souvent possible de séparer un jeu de données à l'aide d'une marge dure. Le problème consiste alors à optimiser les paramètres de l'algorithme pour décider combien d'objets peuvent être autorisés à franchir les frontières pour obtenir l'hyperplan avec les meilleurs performances.
Ce qui est désigné comme étant un vecteur de support (Support Vector) se définit comme étant les deux objets les plus proches de la marge.
Beaucoup de problématiques ne peuvent être *linéairement* séparées, on y remédie en augmentant la dimension de l'espace (par exemple passage de la deuxième dimension 2D à la troisième dimension 3D, voir Figure 5). Cette transformation se fait grâce à des fonctions ou méthodes kernel.

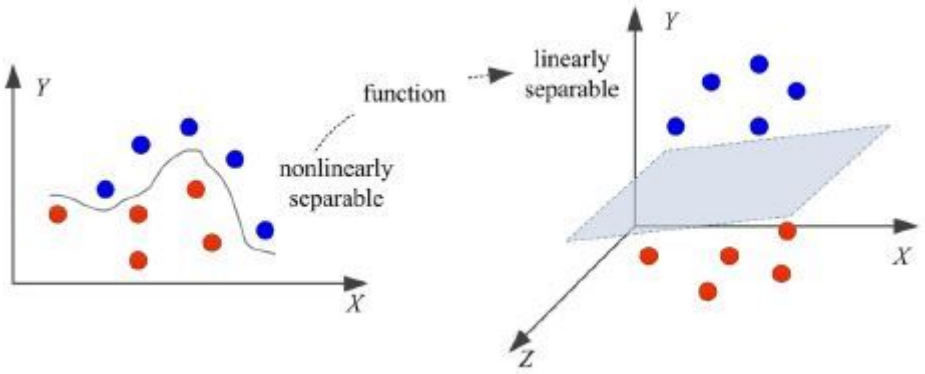


Figure 5 : passage de la 2D à la 3D pour des données qui n'étaient pas linéairement séparables [8]

Les informations citées dans cette section sont extraites de la référence [1]

2.5.3 Naive Bayes

Le classificateur de *Naive Bayes* se base sur le théorème de Bayes, un classique de la théorie des probabilités conditionnelles. De ce fait, cette méthode est conçue pour fonctionner sous l'hypothèse qu'au sein de chaque classe, les features sont indépendantes l'une de l'autre. C'est une hypothèse forte qui tend à fournir de bons résultats en pratique même lorsqu'elle n'est pas vérifiée, plus particulièrement dans les applications de traitement de texte.
La méthode est justement dite " naïve " car elle suppose les features comme étant indépendantes même si elles ne le sont pas.
Son avantage est la rapidité de son exécution car le calcul de probabilité n'est pas gourmand en ressources, et elle se démarque par le fait qu'elle fournit généralement les meilleurs résultat quand le nombre de features est relativement grand par rapport au nombre d'observations.
Lors de la phase d'entraînement, Naive Bayes tente d'estimer les paramètres d'une distribution pour une probabilité conditionnelle. Le choix de cette distribution est très décisif sur l'efficacité de la performance, les plus utilisées en pratique sont la distribution normale (gaussienne) et la Kernel distribution (estimation par noyau).

Les informations citées dans cette section sont extraites de la documentation officielle de MATLAB.

2.5.4 Arbres de décision

Un arbre binaire de classification (arbre de décision binaire pour la classification) est un arbre dont les feuilles (derniers noeuds, qui ne possèdent pas de fils) sont les classes prédites. Tous les autres noeuds représentent une expression logique et chacun se subdivise donc en deux branches, une branche qui vérifie l'expression (données pour lesquelles l'expression vaut *true*) et une seconde qui ne la vérifie pas (données pour lesquelles l'expression vaut *false*).
Une fois l'arbre construit, il va servir de modèle aux données de test. Chacune de ces dernières va parcourir l'arbre de haut en bas en suivant les branches selon les conditions logiques des noeuds parcourus, jusqu'à atteindre la feuille qui indiquera la classe qui lui est attribuée.

Pour un même jeu de données, il existe beaucoup d'arbres possibles, il s'agit alors de trouver l'arbre qui fournit les meilleures prédictions.
Pour ce fait, l'objectif à chaque noeud est de trouver la condition avec la plus grande entropie. En d'autres mots : qui permet de séparer les données en deux parties qui soient les plus équilibrées possibles en terme de taille. L'entropie est maximale si la condition sépare les données en deux parties égales, et elle est minimale si la condition est soit vérifiée par toutes les données, ou le contraire.
En pratique, pour chaque niveau de l'arbre (on procède de haut en bas), on examine toutes les séparations possibles pour chaque feature. Séquentiellement, on prend la condition qui offre la meilleure séparation à chaque niveau.
On devine rapidement que plus on a de features, plus on risque d'obtenir un arbre complexe et pas forcément plus performant, il faut donc faire attention à sélectionner des features prédictives afin de ne pas tomber dans le problème de l'*overfitting*.
Pour réduire la complexité de l'algorithme sans impacter ses performances, on a recours à deux techniques à la fois : l'imposition de critères d'arrêt et l'usage du *pruning* (action de tailler un arbre/des branches).
Le pruning est une technique très adaptée pour contrer l'overfitting et consiste à retirer des parties de l'arbre qui ne contribuent pas à la performance globale. Par exemple si à partir d'un certain noeud, l'unanimité des feuilles descendantes sont de la même classe, on peut retirer tous ses noeuds fils et les remplacer par des feuilles de la classe.
Similairement, l'imposition de critères d'arrêt permet de réduire la complexité par exemple en éliminant les descendants d'un noeud dont les descendants sont en grande majorité les observations d'une même classe, il revient alors à sélectionner le nombre minimal d'observations d'une même classe pour décider de s'arrêter à faire des séparations ou pas.

Les informations citées dans cette section sont extraites de la référence [1] et de la documentation officielle de MATLAB.

2.5.5 Extreme Learning Machine

Les *Extreme Learning Machines* (ELM, machines à apprentissage extrême) sont un réseau de neurones à propagation avant, pouvant être à une seule couche cachée ou à plusieurs, dans lequel les paramètres des neurones cachés sont sélectionnés aléatoirement et n'ont donc pas besoin d'être ajustés.
Les pondérations de sortie des noeuds cachés sont généralement appris en une seule itération, ce qui revient à apprendre un modèle linéaire (l'implémentation la plus simple utilise la méthode des moindres carrés). Les ELM sont prouvées pour avoir de bonnes capacités de généralisation et d'être plus rapide que les réseaux de neurones à propagation arrière.
Différentes fonctions continues peuvent être utilisées au sein des neurones. Ces fonctions utilisent des paramètres a et b qui sont justement choisis aléatoirement par l'algorithme, les plus utilisées sont:
Fonction sigmoïdale = $\frac{1}{1+exp(-(ax+b))}$, fonction de Fourier = $sin(ax+b)$, fonction de Hardlimit = 1 si a.x-b ≥ 0, 0 sinon.

Les informations citées dans cette section sont extraites de la référence [9].

2.5.6 Ensemble Learning

L'*ensemble learning* est un paradigme du *machine learning* selon lequel plusieurs classificateurs (*base learners*) à la fois sont entraînés pour résoudre le même problème. Contrairement aux approches classiques du machine learning où une seule hypothèse est apprise du jeu de données, les méthodes ensemblistes essaient de construire un ensemble d'hypothèses et les combinent pour les utiliser.
Les base learners sont généralement construit à partir du jeu de données à l'aide de classificateurs classiques comme les arbres de décision. La plupart des méthodes ensemblistes utilisent un même type de base learners (par exemple n'utiliser que des arbres) : on parle d'homogénéité. Mais certaines méthodes utilisent aussi des classificateurs différents, on parle alors d'hétérogénéité.
La procédure générale pour la construction d'un classificateur ensembliste se divise en deux étapes. Premièrement, on produit un certain nombre de base learners soit en parallèle, soit d'une manière séquentielle (cas dans lequel la génération d'un base learner a une influence sur la génération des suivants). Deuxièmement, on combine les classificateurs ensemble; la méthode la plus répandue étant de les faire voter pour une classe après les avoir pondérés.
On distingue plusieurs types d'approches pour une méthode ensembliste, ceux qui seront traitées ci-dessous sont la méthode du *Boosting* et la méthode du *Bagging*.

2.5.6.1 Boosting

Le Boosting est une famille d'algorithmes qui consiste à créer des base learners faibles (weak learners) et les renforcer/booster à travers des itérations. L'algorithme de Boosting le plus connu est AdaBoost qui construit des weak learners d'une manière séquentielle. Le premier classificateur produit tente de classifier des observations dont les pondérations sont égales. Ensuite, un autre classificateur est produit avec la même méthode que le premier mais en adaptant les pondérations en augmentant celles qui ont été mal classifiée et en diminuant celles qui ont été correctement classifiée. On réitère l'opération jusqu'à avoir autant de classificateur qu'on souhaite. A la fin, la prédiction se fait par vote des différents classificateurs, pondérés par un poids qui dépend de leur erreur d'apprentissage :

$$f(x) = \sum_{t=1}^{T} \alpha_t . h_t(x)$$

$f(x)$ est la prédiction finale de l'observation x.
$h_t(x)$ est la prédiction de l'observation x donnée par le classificateur *t.*
α_t est le poids associé à h_t. Pour AdaBoost, on prend $\alpha_t = \frac{1}{2} log(\frac{1-\varepsilon_t}{\varepsilon_t})$ et ε_t est l'erreur de classification de h_t.

2.5.6.2 Bagging

Le mot Bagging est une abréviation de Bootstrap Aggregation. Pour bagger un classificateur comme un arbre de décision par exemple, on produit plusieurs répliques bootstrap du jeu de données et on y crée un arbre. Une réplique bootstrap est un nouveau jeu de donnée de produit à partir du premier en le sous-échantillonnant : on fait des tirages aléatoires avec remise d'observations du jeu de donnée jusqu'à atteindre la même taille que le jeu de données original. On se retrouve alors avec le même nombre d'observations que le jeu de données original, mais certaines de ces observations y apparaissent et d'autres non. La probabilité qu'une observation apparaisse au moins une fois vaut 0.632. Pour chaque

réplique on produit par exemple un arbre. La prédiction finale se fait par vote de ces arbres là en prenant la classe majoritaire.

Les informations citées dans cette section sont extraites des références [2], [3], [4] et de la documentation officielle de MATLAB.

2.6 Calcul des performances

La précision d'un classificateur, donnée en % est calculée comme suit : on classifie les données de test, on obtient alors un vecteur contenant les classes prédites. On compare alors ce vecteur avec les classes déjà connues des données de test. La précision est égale au nombre de classes correctement prédites, divisées par la taille des données de test.
Pour avoir une idée plus détaillée sur la qualité d'un classificateur, on peut avoir recours à une matrice de confusion sous la forme suivante :

		Classe réelle	
		0	**1**
Classe prédite	**0**	**Vrai négatif**	**Faux négatif**
	1	**Faux positif**	**Vrai positif**

Vrai négatif : indique une bonne prédiction d'absence d'apnée.
Vrai positif : indique une bonne prédiction de présence d'apnée.
Faux positif : indique une fausse prédiction de présence d'apnée (fausse alarme).
Faux négatif : indique une fausse prédiction d'absence d'apnée.

3. Analyse du problème

Le *Data Mining* est au fait une partie d'un processus plus grand appelé KDD pour *Knowledge Discovery in Databases* (découverte d'informations dans les bases de données), qui se déroule en neuf étapes :
1- **Planifier et comprendre le domaine d'application :** cela revient à se documenter sur l'apnée du sommeil.
2- **Sélectionner des données et créer un ensemble d'attributs/features sur lesquelles la découverte sera faite :** étape dont les sections *4. Données utilisées* et *6. Extraction de features* font l'objet.
3- **Effectuer un traitement initial sur les données pour l'élimination ou la réduction de bruits :** étape dont la section *5. Moyenne glissante* fait l'objet.
4- **Transformation des données et réduction de la complexité :** non nécessaire pour notre étude car les données utilisées sont équilibrées et diversifiées sans pour autant être trop nombreuses.
5- **Choisir une méthode de *Data Mining* qui correspond aux caractéristiques du problème :** on cherche au fait à faire le diagnostic de l'apnée du sommeil à partir de données déjà diagnostiquées, en d'autres mots on souhaite être capable de classifier chaque nouvelle donnée en "apnée" et "pas d'apnée"

en entraînant un modèle à partir de données classifiées à l'avance. On utilisera donc une méthode de **classification supervisée** qui nous produira un modèle prédictif (revoir les sections *2.1.1 Classification*, *2.2.1 Modèles prédictifs* et *2.3.1 Supervised Learning (Apprentissage supervisé)*).
6- **Choisir un algorithme pertinent pour le domaine d'application :** la section *7. Implémentation* décrit justement l'usage de différents algorithmes (dont les concepts sont expliqués dans la section *5. Classificateurs*) pour conclure lequel est le plus pertinent pour cette application.
7- **Implémenter l'algorithme choisi :** le toolbox "*Statistics Toolbox*" de MATLAB a été utilisé, la plupart des algorithmes de cette étude y étant déjà implémentés. Comme ces derniers nécessitent un très grand bagage mathématique pour être implémentés à la main d'une manière efficace, seul l'algorithme k Nearest Neighbors a été codé à la main. Les résultats obtenus avec ce dernier et ceux obtenus avec l'implémentation offerte par MATLAB sont très similaires.
8- **Evaluer les performances et les informations obtenues :** la section *8. Analyse des résultats* en fait l'objet.
9- **Intégration et incorporation des résultats dans d'autres parties du système :** étape optionnelle qui sort du cadre du présent stage, discutée dans la section *9. Conclusion et perspectives*.

Les informations citées dans cette section sont extraites de la référence [1]

4. Données utilisées

La base de données utilisée, « Apnea-ECG Database », est composée de signaux ECG de 35 patients atteints de différents degrés d'apnée du sommeil. Chaque signal est l'enregistrement d'une nuit de sommeil échantillonné à 100Hz et annoté (présence ou pas d'apnée) pour chaque minute, c'est-à-dire pour chaque 6000 échantillons. Les données des patients sont équilibrées en durée (entre 428 minutes et 530 minutes d'enregistrement, soit entre 7 et 9 heures de sommeil) et diversifiées en pourcentage de présence d'apnée (de 0% à 95%). Des détails concernant la longueur de chaque enregistrement, le nombre de minutes d'apnées annotées et le pourcentage d'apnée sont fournis en annexe (voir l'annexe *11.1 Détails sur les données utilisées*).

Ci-dessous une représentation en rouge d'un signal ECG annoté avec une présence d'apnée, et en bleu un second signal ECG annoté avec l'absence d'apnée, les deux issues d'un même enregistrement d'un seul patient et représentées à la même échelle, tous deux sont d'une durée de 10 secondes. En comptant le nombre de battements (QRS), on retrouve 10 battements dans le cas avec apnée et 11 battements dans le cas sans apnée. Cela représente une différence dans le rythme cardiaque de 6 battements par minute.

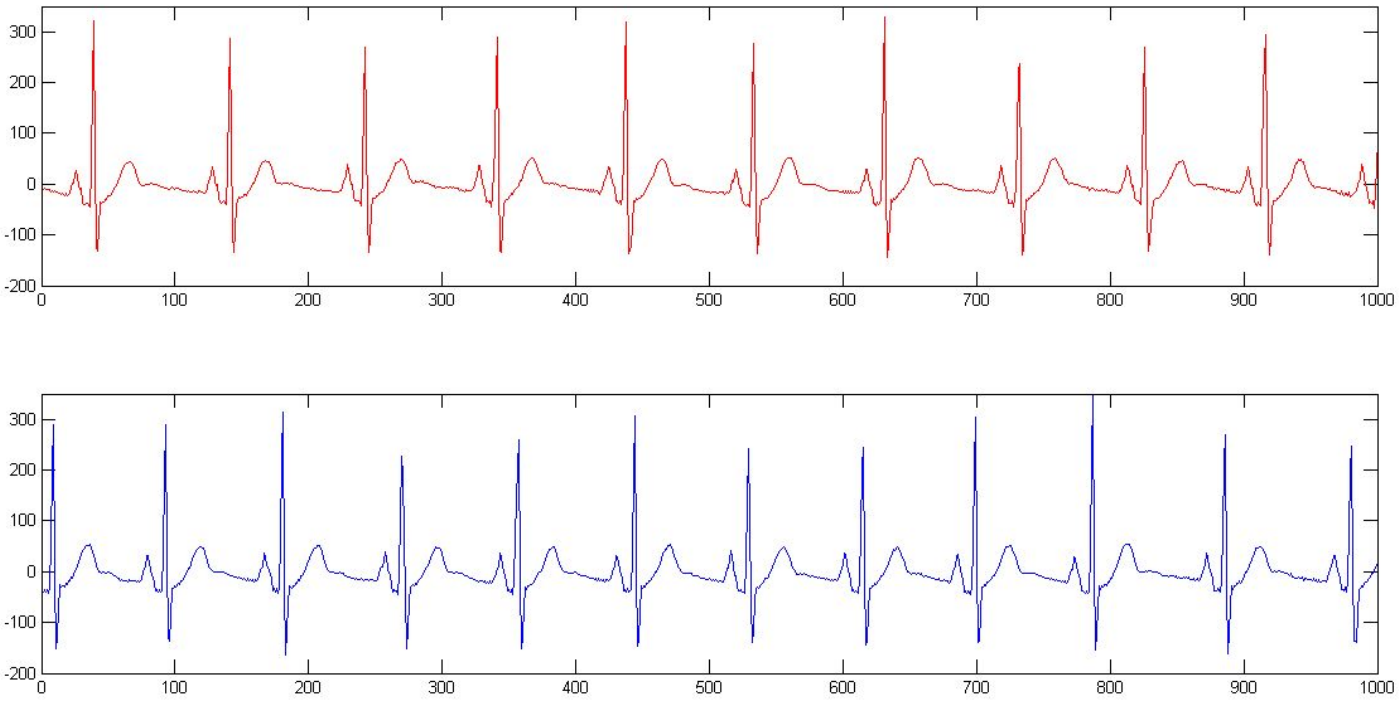


Figure 6 : aperçu de 10 secondes de deux signaux ECG
En rouge : un ECG avec apnée du sommeil, en bleu : un ECG sans apnée du sommeil

5. Moyenne glissante

Comme nous pouvons le voir, le signal ECG présente des fluctuations qui pourraient influencer le calcul de paramètres qui nous serviront de *features*. Pour cela, nous produisons à partir du signal une moyenne glissante, qui permet de supprimer les fluctuations transitoires (les pics qui sont très courts dans le temps et qui ont une grande amplitude) en faisant glisser une fenêtre (un court intervalle) de taille impaire de préférence. Le centre de cette fenêtre prendra comme nouvelle valeur la moyenne des valeurs l'entourant dans la fenêtre. Dans notre étude, nous prenons 5 échantillons comme étant la taille de notre fenêtre.
La figure ci-dessous montre une différence entre le signal original (en bleu) et sa version filtrée (en rouge). On a donc les grands pics qui sont beaucoup atténués (de 300 à 100 environ et de -130 à -80 environ) et les autres parties du signal qui le sont à peine. Le but est de lisser le phénomène sans en modifier la forme pour noyer les valeurs extrêmes.

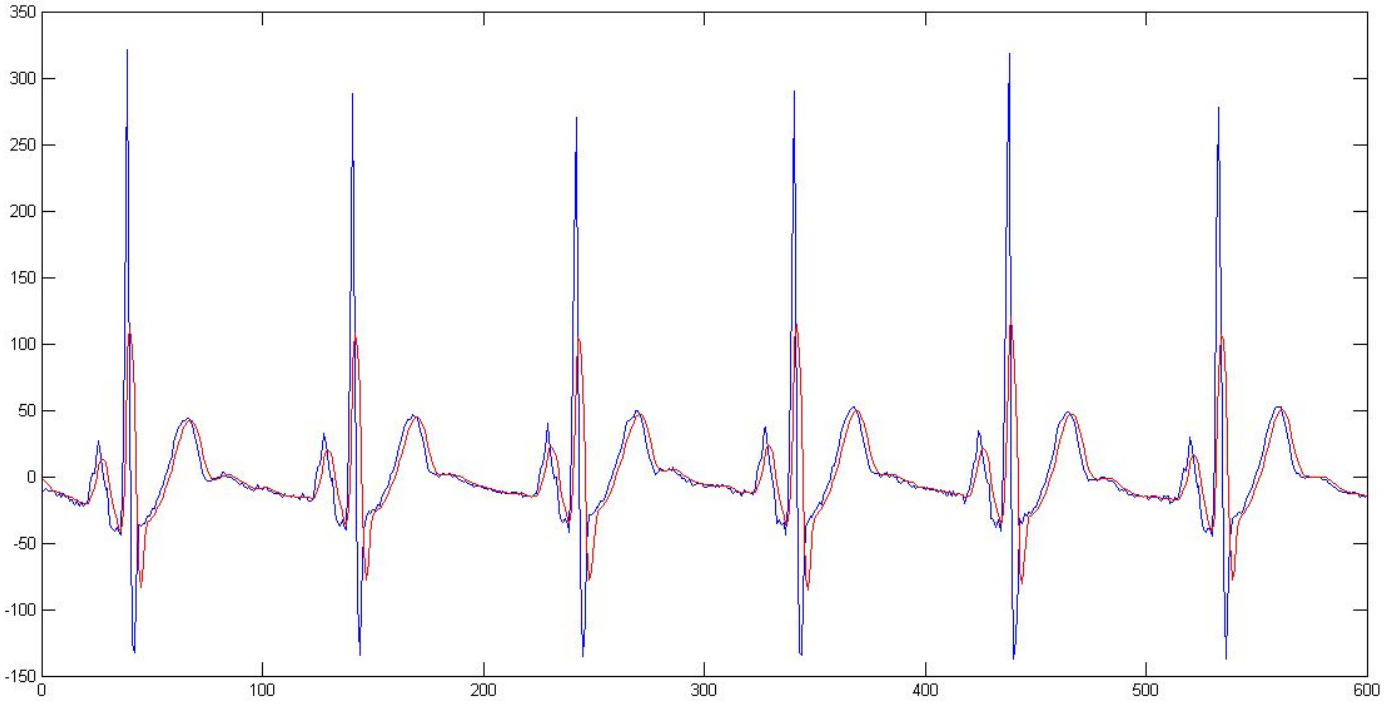


Figure 7 : Superposition d'un signal ECG brut (en bleu) et de sa version filtrée par une moyenne glissante (en rouge)

6. Extraction de features

Le signal est alors prêt à être exploité par l'extraction de features comme suit : on filtre le signal par la moyenne glissante puis pour chaque 6000 échantillons (une minute d'enregistrement), on calcule les features qu'on souhaite avoir, elles seront reliées à l'annotation correspondante (l'annotation vaut 1 pour indiquer l'absence d'apnée, et vaut 8 pour indiquer la présence d'apnée).

La taille d'un enregistrement n'étant généralement pas un multiple de 6000, on tronque au dernier multiple de 6000 qui soit immédiatement inférieur à la taille de l'enregistrement. Ayant autour de 500 observations, une de plus ou de moins représenterait 0.2% de données en plus ou en moins. Nous pouvons donc admettre que cela n'aura pas d'influence sur les performances. On obtient finalement une matrice dont chaque ligne représente une observation, et chaque colonne représente une feature.

On mélange l'ordre des lignes (opération de *shuffling*) pour avoir des données équilibrées, de sorte à ce que deux observations successives n'appartiennent généralement pas au même patient.

On normalise chaque feature (chaque colonne) par rapport à sa valeur maximale de sorte à avoir des features de valeurs < 1. La normalisation est une étape très importante qui a un grand impact sur les performances. En effet, si on omet de normaliser, nous obtiendrons des features qui risquent de beaucoup varier par rapport à d'autres. Les avoir dans le même ordre de grandeur permet d'avoir une vue plus équilibrée et d'obtenir de meilleures performances.

Nous obtenons 16290 observations qu'on doit séparer en training set (observations qui serviront à construire un modèle de classification) et en testing set (observations qui permettent de tester le modèle en comparant la classification donnée par ce dernier avec l'annotation correspondante). Pour ce faire, on choisit de prendre 80% des observations pour l'entraînement et 20% pour le test.

On obtient alors 13032 observations pour l'entraînement, et 3258 observations pour le test. Notre étude va alors comparer les performances obtenues par plusieurs classificateurs classiques pour plusieurs types de features différentes :

- ***Etape 1 :*** dans un premier temps, on construira notre modèle selon 2 features temporels : l'espérance et la variance.
- ***Etape 2 :*** ensuite, aux deux précédentes on ajoute deux autres features (on en aura donc 4) : espérance, variance, skewness et kurtosis.
- ***Etape 3 :*** on procédera à l'extraction de features fréquentiels, qui seront au nombre de 23.
- ***Etape 4 :*** on fait fusionner les 23 features de l'étape 3 avec les 4 features de l'étape 2, on en aura alors 27.
- ***Etape 5 :*** essai de méthodes heuristiques.

Pour chacune de ces méthodes, on testera différents classificateurs qui ont été expliqués dans la section *2.5 Classificateurs* :

- kNN (*k Nearest Neighbors* - Les k plus proches voisins).
- SVM (*Support Vectors Machine* - Machine à vecteurs de support).
- *Naive Bayes* (Classification naïve bayésienne).
- *Extreme Machine Learning.*
- *Decision Trees* (arbres de décision).
- *Ensemble Learning* (méthodes ensemblistes) qui se subdivisent en méthodes de *Boosting* et en méthodes de *Bagging*.

L'ensemble des features utilisés est :
Features temporelles :

Moyenne	Variance	Skewness	Kurtosis
$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$	$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N} (x_i - \mu)^2$	$\gamma = E((\frac{X-\mu}{\sigma})^3)$	$\beta = E((\frac{X-\mu}{\sigma})^4)$

Features de HRV temporelles [10] : **maximum**, **minimum**, **moyenne**, **médiane** (valeur qui permet de couper les données en deux parties égales),
SDNN (déviation standard de l'intervalle NN qui renseigne sur la variabilité globale)

$$\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(\mathrm{RR}_i - \overline{\mathrm{RR}}\right)^2}$$

RR_i représente le temps entre le pic R_i et le pic R_{i+1}, tandis que $\overline{RR}$ représente l'intervalle RR moyen.
SDANN (déviation standard de la moyenne des intervalles R-R des segments de 5 min sur toute la période d'enregistrement, qui exprime la variabilité globale des cycles de 5 min, c'est-à-dire la variabilité à long terme), **meanHR** (rythme cardiaque moyen), **sdHR** (déviation standard du rythme cardiaque), **HRVTi** (index triangulaire de la variabilité du rythme cardiaque), **TINN** (interpolation triangulaire de la distribution discrète des intervalles RR).

Features de HRV fréquentielles par la méthode de Welch : [10]

- **peakHF**, **peakLF**, **peakVLF** en Hz : respectivement la fréquence du pic en haute fréquence, en basse fréquence et en très basse fréquence.
- **aHF**, **aLF**, **aVLF** en ms^2 : puissances correspondantes à la surface sous la courbe dans les bandes de fréquences : haute fréquence, basse fréquence, très basse fréquence.

- **pHF**, **pLF**, **pVLF** en % : respectivement aHF/aTotal, aLF/aTotal et aLVF/aTotal.
- **nHF**, **nLF** en % : surfaces normalisées (nHF = aHF/(aLF+aHF) et nLF = aLF/(aLF+aHF)).
- **PSD** en ms^2/Hz : densité spectrale de puissance.
- **F** en Hz : vecteur de fréquences.

7. Implémentations

Les performances sont résumés d'une manière générale sous la forme de la précision en %.
En annexe sont données les matrices de confusion des principaux classificateurs.

7.1 Par 2 features temporelles

Pour cette première étape, les features choisies sont simples : l'espérance et la variance de chaque 6000 échantillons des données nous fournit une observation. Ci-dessous un tableau qui détaille les résultats en les citant du plus performant au moins performant :

Méthode	Performance (précision en %)	Observations
Bagging (TreeBagger)	90.72	Performance obtenue avec 52 arbres de décision. La courbe des performances en fonction du nombre d'arbres est donnée dans la *Figure 8*
Arbre de décision	88.45	Un aperçu de l'arbre de décision est donné en annexe. Les paramètres par défaut de MATLAB ont été utilisés
kNN	86.84	Cette performance a été obtenue en utilisant la distance *cityblock* et avec k=3. D'autres distances et d'autres valeurs de k ont donné de moins bonnes performances. La courbe des performances en fonction de k est donnée dans la *Figure 9*
Boosting (RobustBoost)	83.40	Les méthodes ensemblistes du type Boosting ont été implémentés avec 600 arbres de décision. Il n'y a pas eu de recherche exhaustive du meilleur nombre d'arbres car il a été remarqué que la performance y était proportionnelle, il a donc suffit de prendre un nombre assez grand pour avoir une bonne performance et assez petit pour ne pas avoir un temps d'exécution trop grand
Boosting (LogitBoost)	82.87	
Boosting (AdaBoostM1)	79.05	
Boosting (Subspace)	71.89	
Naive Bayes	70.56	Il a fallu utiliser une distribution Kernel pour aboutir à de bons résultats. En effet, l'usage de la loi normale conduit à de mauvais résultats (<50%)

Extreme Machine Learning	62.93	Performance obtenue pour 72 neurones cachés
SVM	62.87	Performance obtenue en utilisant une fonction Kernel RBF (Radial Basis Function)

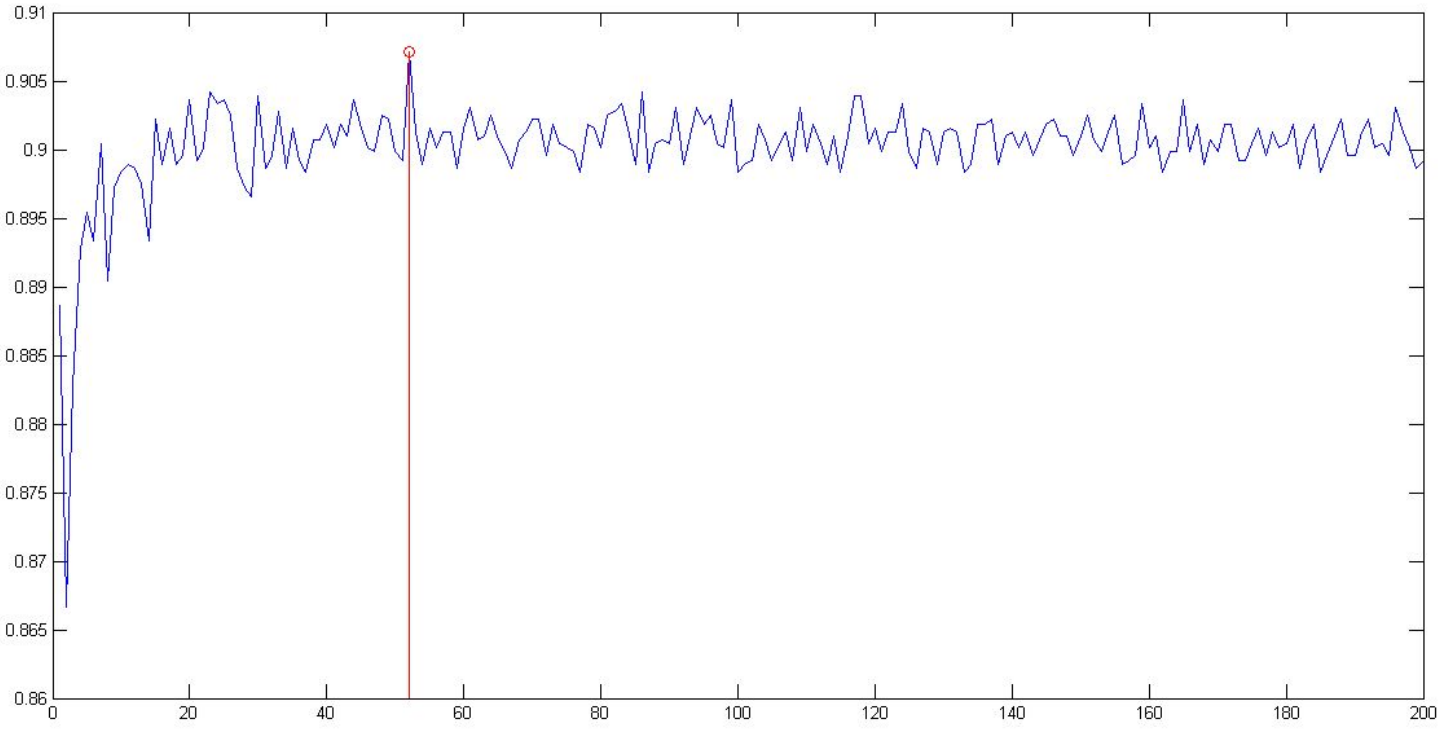


Figure 8 : évolution de la précision du TreeBagger en fonction du nombre d'arbres

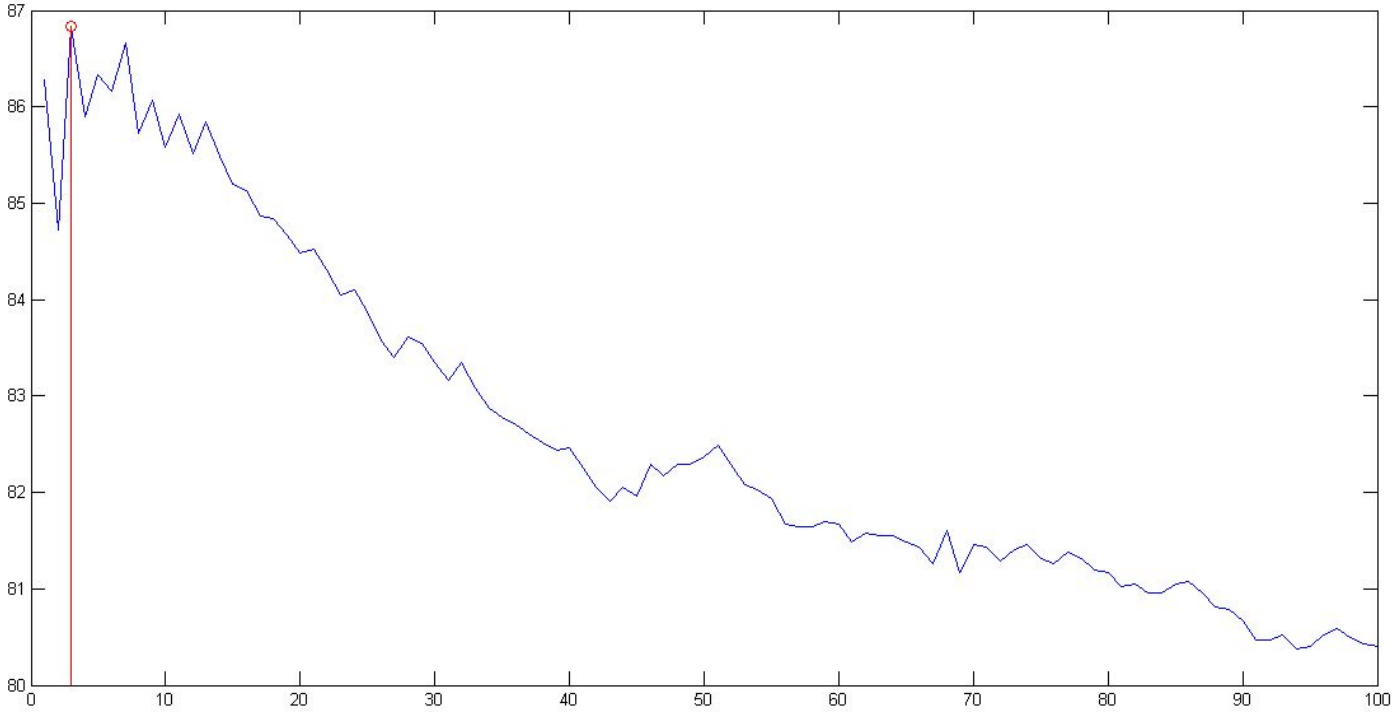


Figure 9 : évolution de la précision de kNN en fonction du paramètre k

7.2 Par 4 features temporelles

On garde les features précédentes et on y ajoute deux autres : skewness et kurtosis.

Méthode	Performance (précision en %)	Observations
Bagging (TreeBagger)	95.74	Performance obtenue avec 19 arbres
kNN	93.97	Performance obtenue pour k=1 et une distance CityBlock
Arbre de décision	92.89	Paramètres par défaut de MATLAB
Boosting (RobustBoost)	84.43	Les méthodes ensemblistes du type Boosting ont été implémentés avec 600 arbres de décision. Il n'y a pas eu de recherche exhaustive du meilleur nombre d'arbres car il a été remarqué que la performance y était proportionnelle, il a donc suffit de prendre un nombre assez grand pour avoir une bonne performance et assez petit pour ne pas avoir un temps d'exécution trop grand
Boosting (LogitBoost)	83.25	
Boosting (AdaBoostM1)	79.52	
Boosting (Subspace)	75.65	
Naive Bayes	70.56	Il a fallu utiliser une distribution Kernel pour aboutir à de bons résultats. En effet, l'usage de la loi normale conduit à de mauvais résultats (<50%)
SVM	70.09	Performance obtenue en utilisant une fonction Kernel RBF (Radial Basis Function)
Extreme Machine Learning	75.42	Performance obtenue pour 1000 neurones cachés et une fonction d'activation sigmoïdale

7.3 Par 23 features fréquentielles

Les features fréquentielles utilisées sont extraites des variabilités du rythme cardiaque temporelles et fréquentielles.
Temporelles : maximum, minimum, moyenne, médiane, SDNN, SDANN, meanHR, sdHR, HRVTi, TINN
Fréquentielles : extraites grâce à la méthode de Welch, étude en très basse fréquence (entre 0 et 0.16Hz), en basse fréquence (entre 0.16 et 0.6Hz) et en haute fréquence (0.6 à 3Hz).

Méthode	Performance (précision en %)	Observations
Bagging (TreeBagger)	82.52	Performance obtenue en utilisant 69 arbres
kNN	79.62	Performance obtenue pour k=13 et une distance CityBlock
Boosting (LogitBoost)	79.29	Les méthodes ensemblistes du type Boosting ont été implémentés avec 600 arbres de décision. Il n'y a pas eu de recherche exhaustive du meilleur nombre d'arbres car il a été remarqué que la performance y était proportionnelle, il a donc suffit de prendre un nombre assez grand pour avoir une bonne performance et assez petit pour ne pas avoir un temps d'exécution trop grand
Boosting (RobustBoost)	78.89	
Boosting (AdaBoostM1)	78.26	
SVM	78.05	Performance obtenue en utilisant une fonction Kernel RBF (Radial Basis Function)
Arbre de décision	74.99	Paramètres par défaut de MATLAB
Naive Bayes	71.97	Il a fallu utiliser une distribution Kernel pour aboutir à de bons résultats. En effet, l'usage de la loi normale conduit à de mauvais résultats (<50%)
Extreme Machine Learning	81.07	Performance obtenue pour 1000 neurones cachés et une fonction d'activation sigmoïdale

7.4 Par toutes les features

Fusion des 23 features de la section 7.3 avec les 4 features de la section 7.2.

Méthode	Performance (précision en %)	Observations
Bagging (TreeBagger)	92.02	Performance obtenue en utilisant 83 arbres
Boosting (LogitBoost)	89.05	Les méthodes ensemblistes du type Boosting ont été implémentés avec 600 arbres de décision. Il n'y a pas eu de recherche exhaustive du meilleur nombre d'arbres car il a été remarqué que la performance y était proportionnelle, il a donc suffit de prendre un nombre
Boosting (RobustBoost)	89.11	

Boosting (AdaBoostM1)	86.57	assez grand pour avoir une bonne performance et assez petit pour ne pas avoir un temps d'exécution trop grand
Boosting (Subspace)	68.88	
Arbre de décision	88.66	Paramètres par défaut de MATLAB
kNN	84.85	Performance obtenue pour k=13
Extreme Machine Learning	84.40	Performance obtenue pour 926 neurones cachés et une fonction d'activation sigmoïdale
SVM	82.64	Performance obtenue en utilisant une fonction Kernel RBF (Radial Basis Function)
Naive Bayes	75.02	Il a fallu utiliser une distribution Kernel pour aboutir à de bons résultats. En effet, l'usage de la loi normale conduit à de mauvais résultats (<50%)

7.5 Heuristiques

Les heuristiques sont des règles empiriques simples basées sur l'expérience et sur l'analogie. Généralement, on n'en obtient pas la solution optimale mais une solution approchée.
L'heuristique tentée est la suivante :

- On construit trois modèles différents, ceux qui ont donné les meilleurs résultats
    - Un classificateur kNN avec 4 features temporelles
    - Un arbre de décision avec 4 features temporelles
    - Un classificateur TreeBagger avec l'ensemble des 27 features temporelles et fréquentielles
- Pour chaque observation, si kNN et l'arbre de décision donnent la même prédiction, c'est celle qui est choisie. Si chacun donne une prédiction différente, on prend celle donnée par le TreeBagger (donc la classe majoritaire des 3 classificateurs).

Cette méthode donne une précision de 95.22%, dépassant les performances des méthodes classiques tout en étant très peu gourmande en calcul.

7.6 Unsupervised Learning - Clustering

Malgré le fait que notre problème relève du supervised learning, il est possible de tenter une méthode d'unsupervised learning : le clustering, plus particulièrement l'algorithme *kmeans.*
Le clustering se fait sans supervision dans le sens où on donne les données d'entraînement à l'algorithme sans préciser leurs classes respectives, et il les regroupe en clusters (groupes) selon leurs similarités et leurs différences.
On procède donc à la divison de nos données en deux groupes, l'algorithme retourne les coordonnées des deux centres appartenant chacun à un cluster. Pour la classification, on calcule la distance (la distance CityBlock a été utilisée) entre chaque donnée de test et les deux centres. La donnée de test appartient au cluster dont elle se trouve le plus proche du centre.

Une fois qu'on l'a affectée à un cluster, on lui donne la classe majoritaire des objets dans le cluster auquel elle appartient.
Cette méthode n'a pas donné de très bons résultats que ça soit pour 2 features temporelles, 4 features temporelles, 23 features fréquentielles ou l'ensemble des 27 features. En effet, les performances sont respectivement de 65.62%, 65.80%, 65.71%, et 65.62%.

8. Analyse des résultats

Globalement, les classificateurs du TreeBagger, kNN et l'arbre de décision fournissent les meilleures performances. La meilleure performance (94%) en utilisant les méthodes classiques revient à l'algorithme kNN en utilisant comme features la moyenne, la variance, skewness et kurtosis.
La meilleure performance dans l'ensemble (95.22%) est obtenue grâce à une méthode heuristique.
Les méthodes ensemblistes se sont montrées efficaces peu importes les features utilisées, contrairement aux autres classificateurs classiques qui étaient souvent plus efficaces dans un cas plutôt que dans l'autre. Ceci est d'autant plus vrai pour le bagging que pour le boosting. Cependant, le Bagging est une méthode aléatoire, ses performances ne sont pas fixes et varient légèrement d'une exécution à l'autre.
Les méthodes SVM, Naive Bayes et ELM ont vu leur performance augmenter avec le nombre de features utilisées, et sont donc moins touchées par l'overfitting que l'arbre de décision par exemple dont au contraire, la performance diminue un peu.

9. Conclusion et perspectives

Cinq approches différentes ont permis des performances supérieures à 90% de précision dans le diagnostic de l'apnée du sommeil. On conclut alors que le Data Mining est un outils très adéquat au diagnostic de ce trouble. Le classificateur kNN avec 4 features temporelles reste le modèle le plus approprié pour sa haute performance et sa basse complexité, sans oublier la facilité de son implémentation.
A long terme, cette étude peut permettre la création d'une application Android qui reçoit des données ECG par bluetooth, permettra de les compresser et les stocker tout en les diagnostiquant pour l'apnée.

10. Références

[1] Analysis of Online Sleep Apnea Data Streams for Mobile Platforms, Steffen Lien. (Thesis submitted for the degree of Master in Informatics, University of OSLO, Autumn 2016)
[2] Schapire, R.E.: The strength of weak learnability. Machine Learning 5(2) (1990) 197–227
[3] Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to Boosting. Journal of Computer and System Sciences 55(1) (1997) 119–139
[4] Breiman, L.: Bagging predictors. Machine Learning 24(2) (1996) 123–140
[5] URL : https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm consulté le 29/08/2018
[6] URL : https://en.wikipedia.org/wiki/Support_vector_machine consulté le 29/08/2018
[7] URL : http://efavdb.com/svm-classification/ consulté le 29/08/2018
[8] URL : http://www.mdpi.com/2073-4441/7/8/4477/htm consulté le 29/08/2018
[9] URL : https://en.wikipedia.org/wiki/Extreme_learning_machine consulté le 01/09/2018
[10] Design, evaluation, and application of Heart Rate Variability Analysis Software (HRVAS)

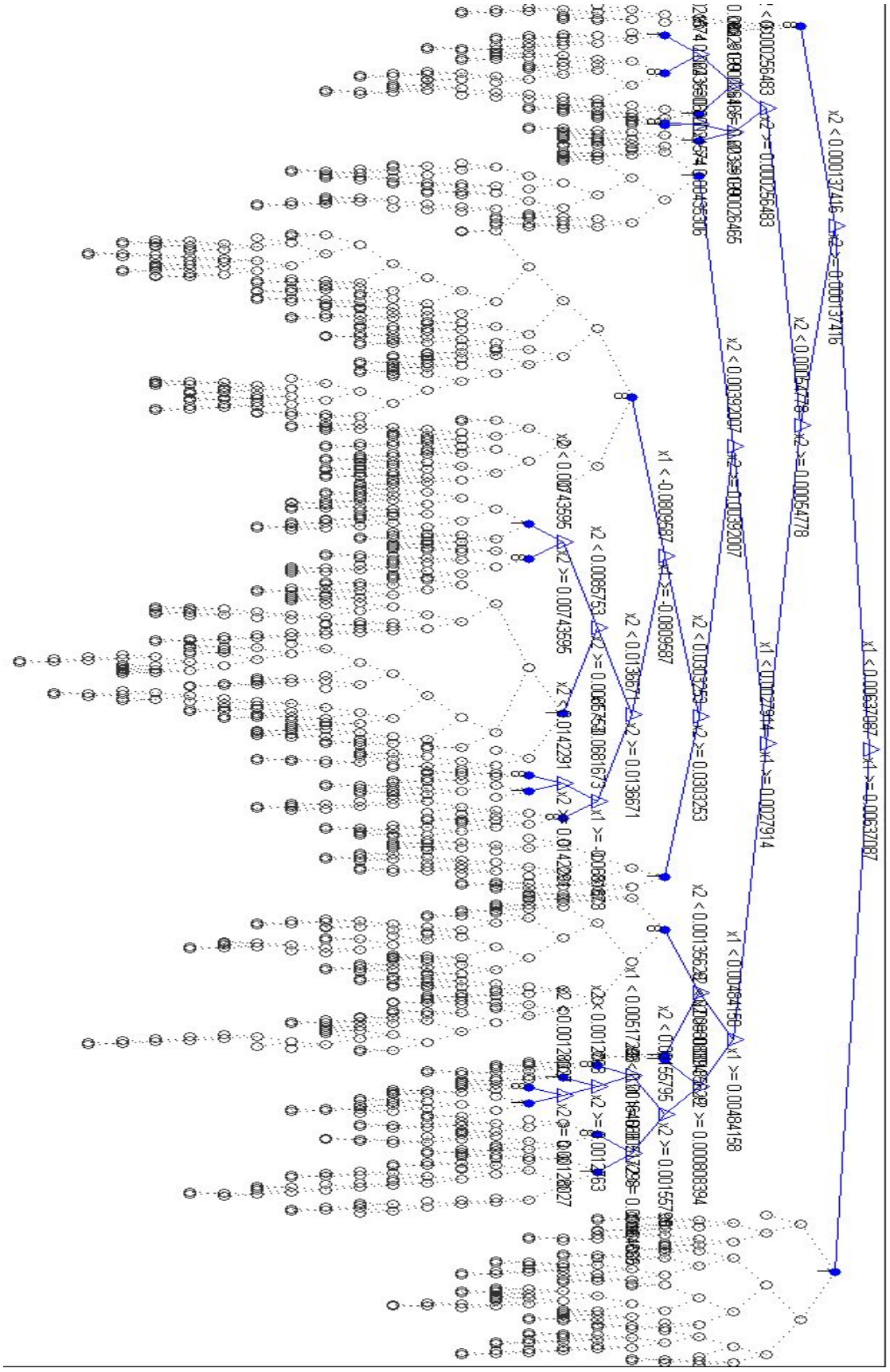
11. Annexes

11.1 Détails sur les données utilisées

Donnée	Longueur (minutes)	Minutes d'apnée	Pourcentage d'apnée
A01	492	470	95,36
A02	530	420	79,19
A03	522	246	47,08
A04	496	453	91,20
A05	453	276	60,89
A06	509	206	40,41
A07	510	322	63,12
A08	500	189	37,76
A09	498	381	76,50
A10	516	100	19,34
A11	468	222	47,43
A12	516	100	19,34
A13	494	244	49,36
A14	522	383	73,30
A15	509	368	72,25
A16	481	320	66,48
A17	484	158	32,63
A18	484	158	32,63
A19	502	205	40,83
A20	510	315	61,76
B01	486	19	3,90
B02	528	93	17,59

B03	440	73	16,57
B04	428	10	2,33
B05	432	57	13,18
C01	483	0	0
C02	501	1	0,19
C03	453	0	0
C04	481	0	0
C05	465	3	0,64
C06	467	1	0,21
C07	428	4	0,93
C08	514	0	0
C09	467	2	0,42
C10	430	1	0,23

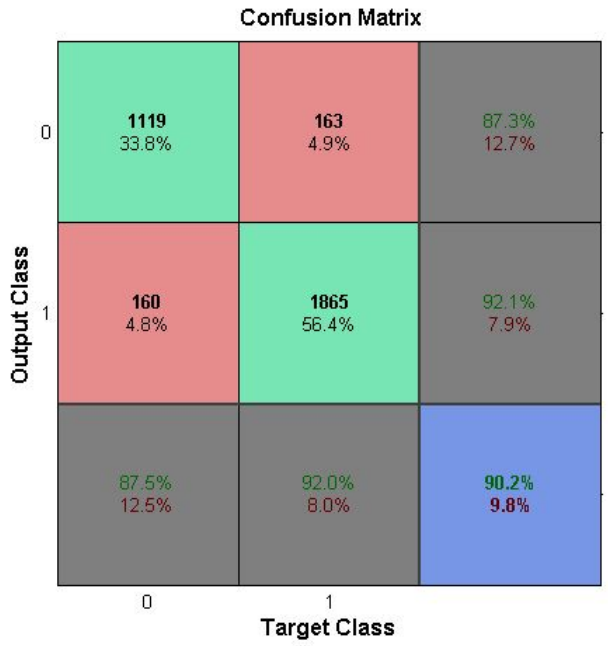
11.2 Aperçu de l’arbre de décision



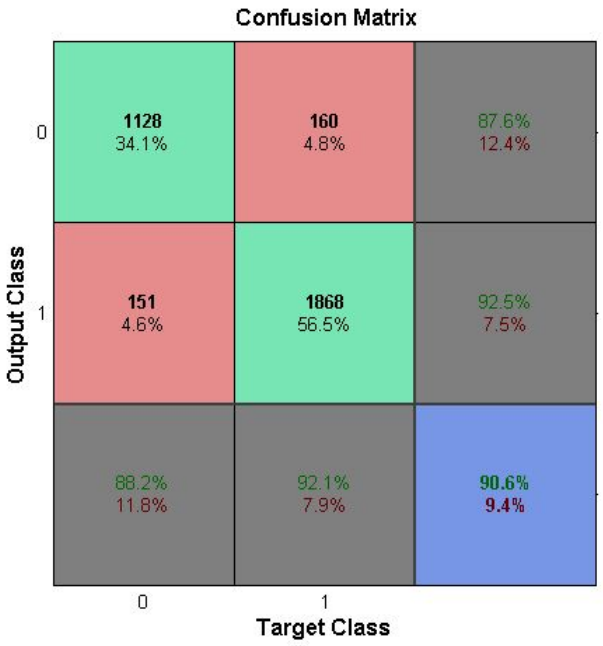
11.3 Matrices de confusion

11.3.1 Avec 2 features temporelles

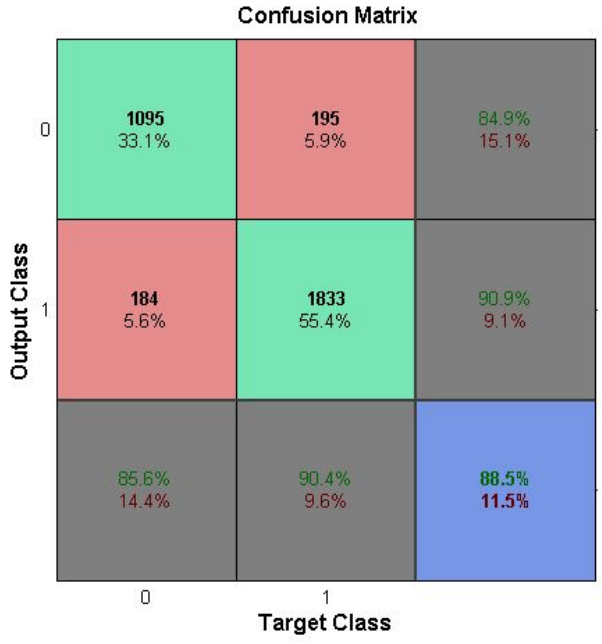
kNN



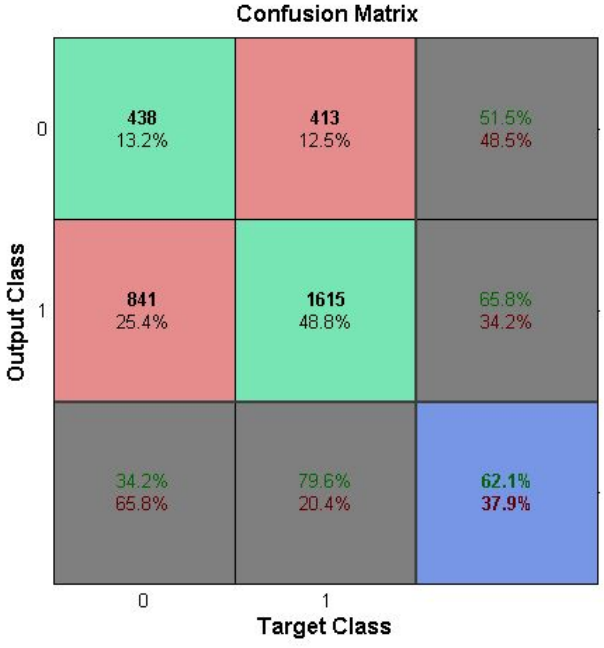
TreeBagger



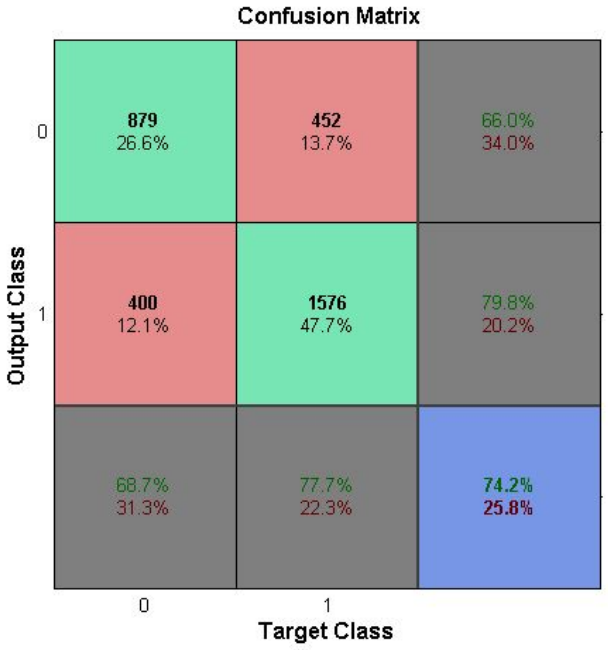
Arbre de décision



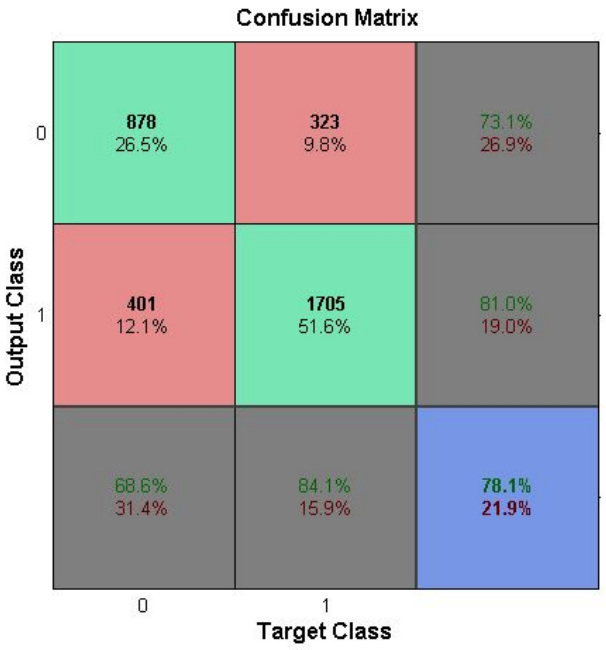
SVM



Naive Bayes



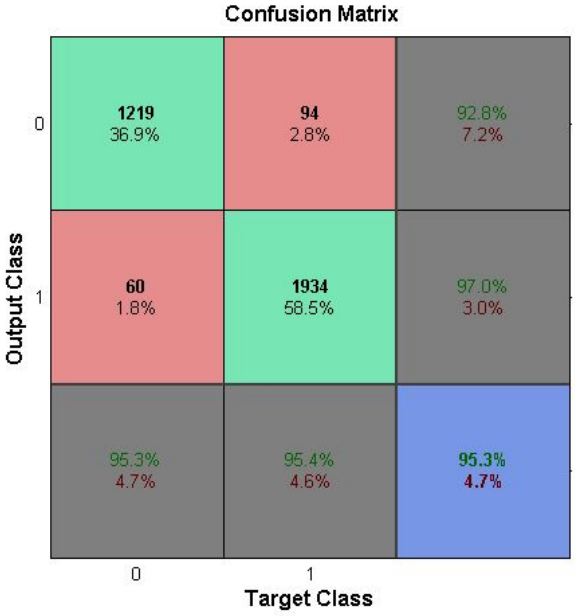
AdaBoost



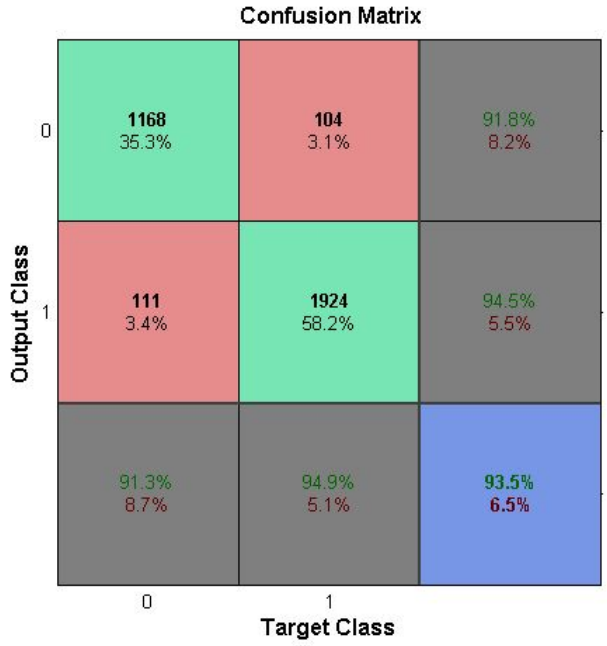
11.3.2 Avec 4 features temporelles

kNN

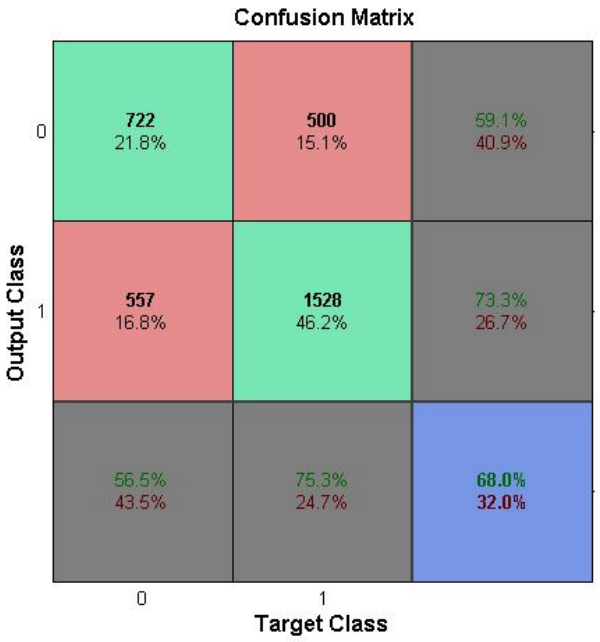
TreeBagger



Arbre de décision

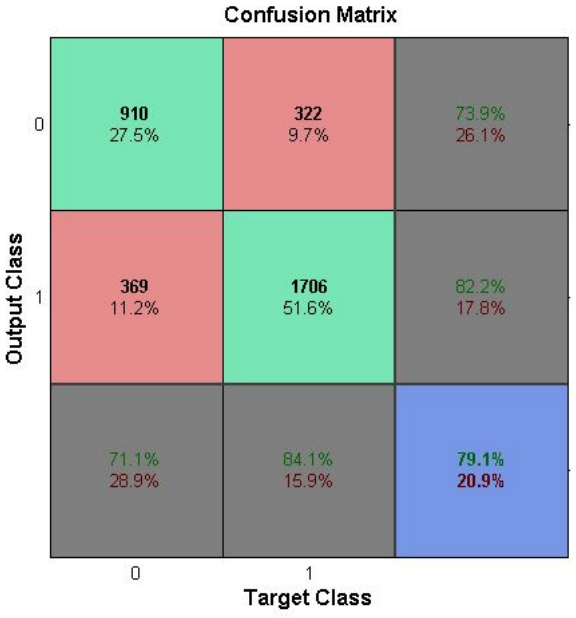


SVM



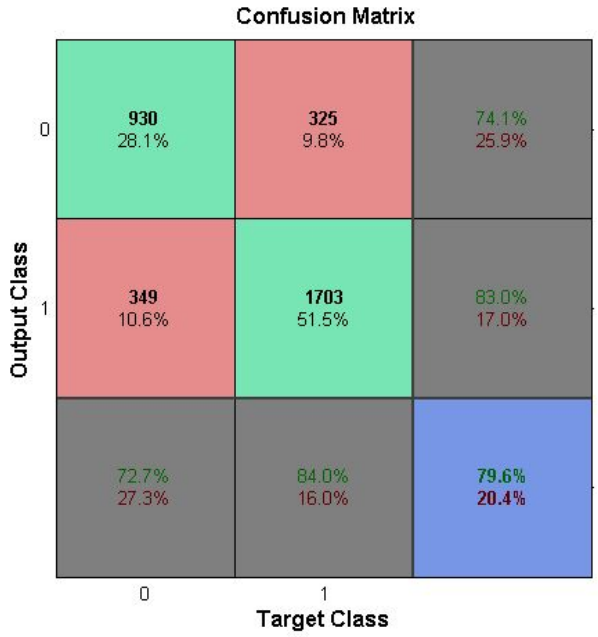
Naive Bayes

AdaBoost

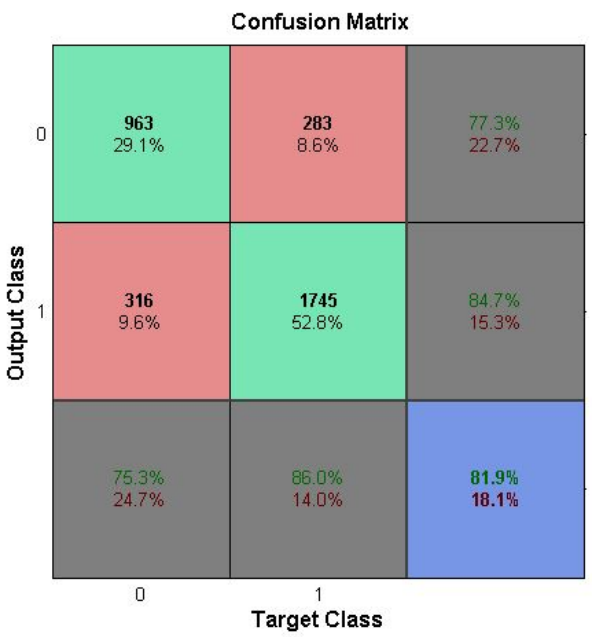


11.3.3 Avec 23 features

kNN

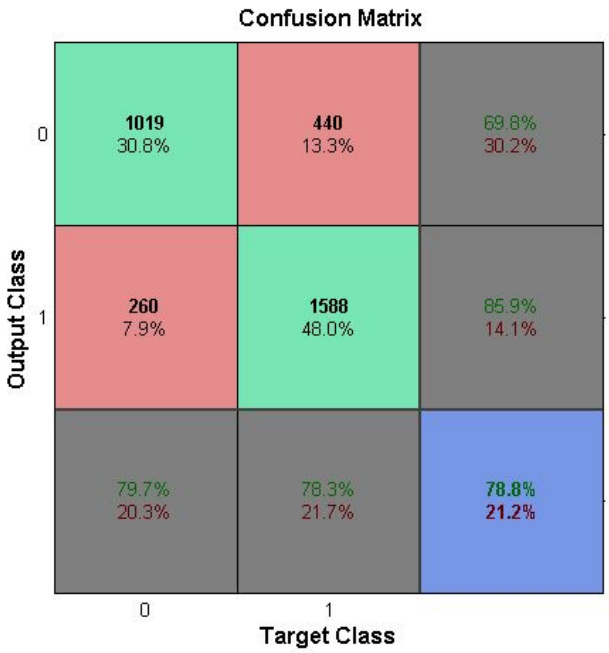


TreeBagger

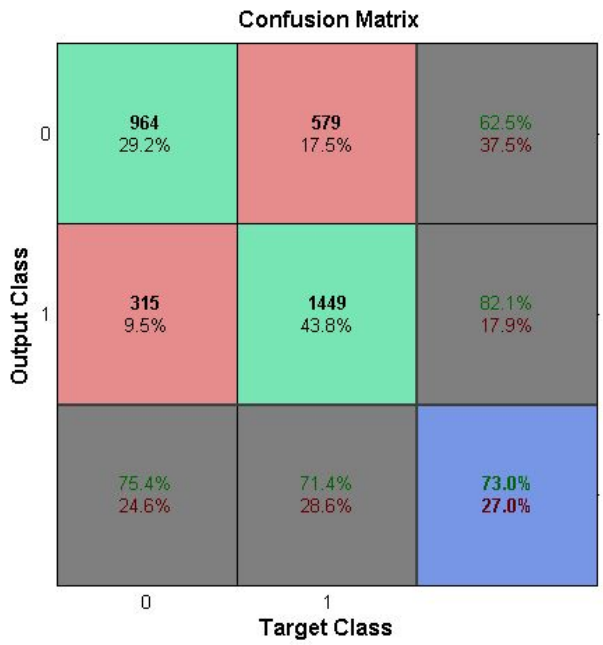


Arbre de décision

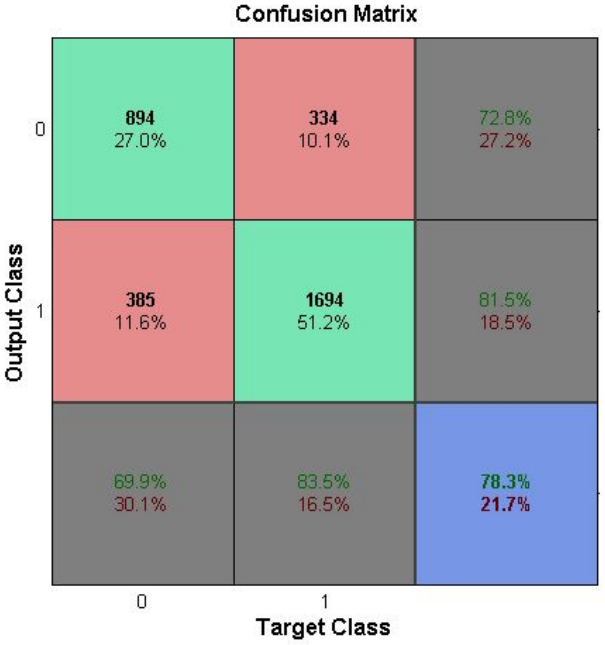
SVM



Naive Bayes



AdaBoost

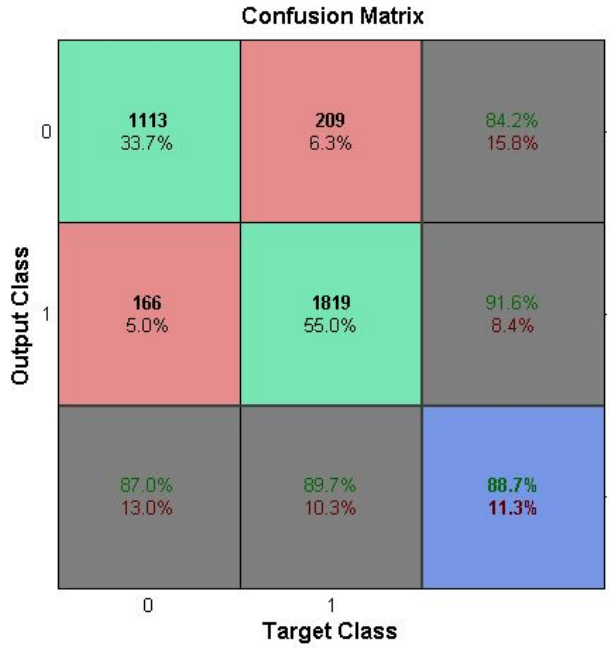


11.3.4 Avec 27 features

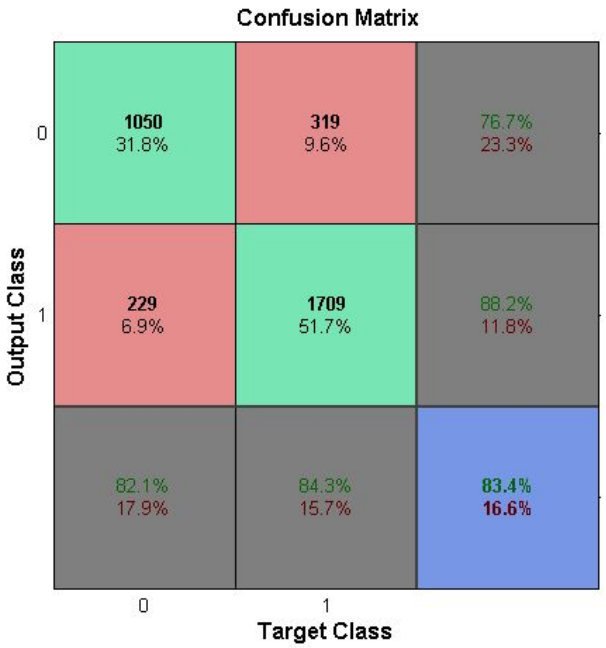
kNN

TreeBagger

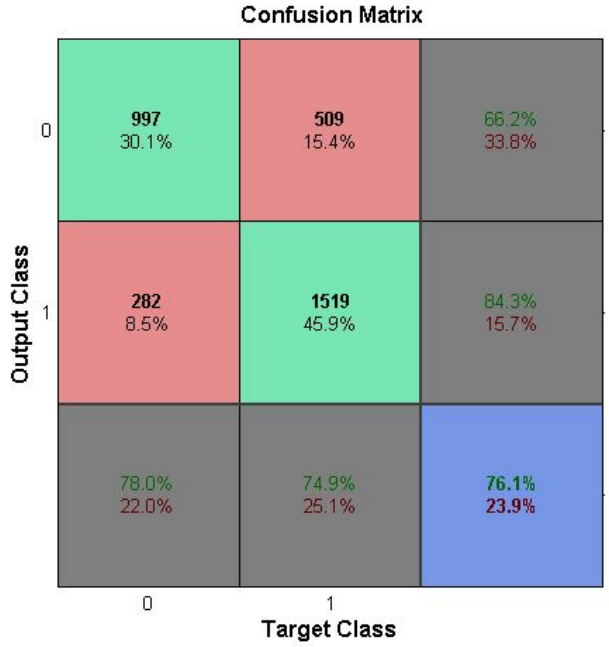
Arbre de décision



SVM



Naive Bayes



AdaBoost