Institut National des Sciences Appliquées
de Rennes

STLime
Rapport de Pré-Étude

Adrien Buridant
Morgane Cam
Antoine Chaffin
Yohan Couanon
Isabelle Guillou
Tangi Mendès
Lisa Relion
Taha Yassine

Responsables de projet :
Laurence Rozé (INSA, INRIA, IRISA)
Maël Guillemé (ENERGIENCY)

Septembre 2018 - Mai 2019
INSA de Rennes

Table des matières

Résumé

Le projet présenté dans ce rapport s'inscrit dans le cadre de la formation en cycle ingénieur du département Informatique à l'INSA de Rennes. Il est réalisé par un groupe de huit étudiants de deuxième année du département informatique et s'étend sur l'ensemble de l'année scolaire.

Le but de ce projet est d'implémenter un interpréteur de classifieur de séries temporelles, c'est-à-dire un outil capable d'expliquer une décision prise par un classifieur. Cette explication doit être claire et relativement simple pour qu'elle soit comprise par un utilisateur quelconque.

Pour vulgariser cet objectif, on peut prendre l'exemple d'un classifieur qui place des individus dans des classes "chiens", "chats" et "humain". Un utilisateur sans compétence technique particulière souhaiterait savoir sur quels critères le classifieur a décidé de se baser pour placer un chat dans la classe "chat". Le classifieur doit alors être en mesure d'expliquer simplement son choix mais de manière suffisamment précise pour convaincre l'utilisateur de la justesse de sa décision. Ce principe sera appliqué aux séries temporelles.

Ce premier rapport concerne la phase de pré-étude du projet. Cette phase a pour but de définir le projet dans son ensemble. Elle consiste en l'exploration préalable permettant la synthèse claire des objectifs du projet ainsi qu'une première vision des travaux à réaliser. Elle passe bien entendu par une étude de l'existant et du contexte du projet.

Introduction

Ce projet est réalisé dans le cadre d'un travail d'ingénierie de groupe sur une réalisation technique de taille conséquente au cours du septième et du huitième semestres du département informatique de l'INSA de Rennes.

Ce projet s'ancre dans le domaine de la classification, qui est un processus de prédiction de classe pour des données. Ces classes sont aussi appelées étiquettes. La classification appartient à la catégorie de l'apprentissage supervisé où l'on possède un ensemble d'apprentissage où les étiquettes sont connues. Le but de ce projet est d'appliquer la classification sur des séries temporelles, qui sont des suites de valeurs numériques continues représentant l'évolution d'une quantité spécifique au cours du temps. Elles sont généralement représentées sous forme de courbes.

Le principe général du projet peut se décomposer en deux grandes étapes. La première étape consistera à apprendre et sauvegarder un classifieur de séries temporelles. Ce classifieur va ainsi permettre de pouvoir retourner la classe d'une série temporelle qui sera passée en entrée. Cette étape est assez succincte car elle consistera à utiliser des algorithmes déjà implémentés dans des bibliothèques existantes. Enfin, la deuxième étape consistera à adapter un algorithme d'interprétabilité de classification, conçu pour des exemples sous forme de valeurs, à la classification de séries temporelles. Ainsi, cet algorithme basé sur l'algorithme LIME permettra d'expliquer les prédictions du classifieur de séries temporelles. Le prototype final qui regroupera les deux étapes s'appellera STLime. L'adaptation de l'algorithme LIME s'appellera LIMEShape.

Dans ce rapport de pré-étude, l'idée générale du projet STLime sera décrite en deux parties. Une première sur la classification de séries temporelles et une seconde sur l'interprétabilité d'un classifieur de séries temporelles, dans le but de faciliter la compréhension de ce projet. Enfin, les objectifs clairs du projet seront expliqués.

1 Notions et principe général du projet STLime

Le projet STLime a pour but d'apprendre un classifieur et d'adapter un algorithme d'interprétation de classification (LIME) pour des séries temporelles. Pour pouvoir expliquer STLime, il est nécessaire de comprendre les notions suivantes (qui seront détaillées dans la suite du rapport) : classifieur, apprentissage supervisé, algorithmes de classification, boîte noire, boîte blanche, interprétabilité et séries temporelles.

1.1 Classifieur

Un problème de classification consiste à associer des classes à des objets. L'exemple suivant illustre cette notion : on dispose d'un ensemble d'images représentant un chat ou un poisson. Le problème de classification est pour une image donnée de trouver sa classe, c'est-à-dire si elle représente un chat ou un poisson (cf. Figure 1).

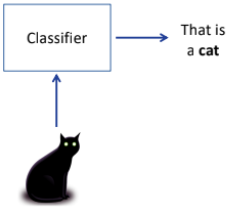

FIGURE 1 – Exemple d'un classifieur de l'image d'un chat

1.2 Apprentissage supervisé d'un classifieur

Le rôle d'un algorithme d'apprentissage est d'apprendre un classifieur. Pour pouvoir le faire, ce dernier a besoin d'informations. Celles-ci vont être fournies sous la forme d'un ensemble d'apprentissage annoté : ensemble d'objets dont on connaît déjà la classe. Comme les données pour l'apprentissage sont déjà catégorisées, on parle d'apprentissage supervisé. En effet, pour que le classifieur puisse prédire, il va d'abord falloir lui donner un jeu d'exemples afin qu'il puisse s'entraîner et s'améliorer dessus.

Pour illustrer ce concept d'apprentissage, l'exemple précédent des images représentant un chat ou un poisson est de nouveau tout indiqué. On dispose de données d'entrée déjà étiquetées. Il s'agit de photos avec leur étiquette (poisson ou chat). Le classifieur va s'entraîner avec ces données pour différencier un chat d'un poisson : c'est la phase d'entraînement. Suite à cette phase, si une photo non étiquetée est passée en entrée du classifieur, ce dernier va être capable de donner l'étiquette de la photo, c'est-à-dire de déterminer si la photo représente un poisson ou un chat (cf. Figure 2).

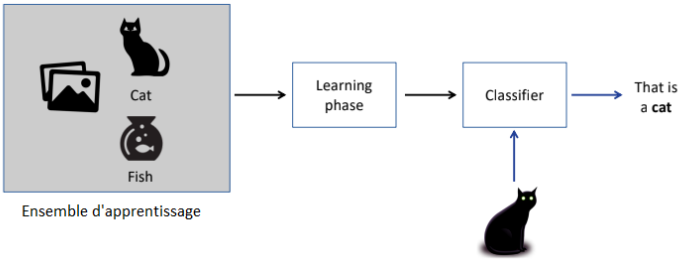

FIGURE 2 – Exemple de l'apprentissage du classifieur des images de chats ou poissons

Il existe de nombreux algorithmes d'apprentissage supervisé : les machines à vecteurs de support (Support Vector Machine ou SVM), les arbres de décision, les forêts d'arbres décisionnels (Random Forest Classifier) et les réseaux de neurones convolutifs (Convolutional Neural Networks ou CNN).

1.3 Boîte noire / Boîte blanche

Certains classifieurs de machine learning peuvent être extrêmement complexes. Un classifieur est interprétable si la justification de ses réponses peut être facilement comprise par un être humain, sinon c'est une boîte noire. L'avantage des boîtes noires est leur précision mais elles restent peu compréhensibles. Quant aux boîtes blanches, c'est-à-dire les classifieurs interprétables, c'est plutôt l'inverse : elles sont compréhensibles mais souvent moins précises. Concernant les boîtes blanches, on peut citer par exemple les arbres de décisions (cf. Figure 4), ou les règles de classification comme les règles If-then (une règle prédit une classe à partir des attributs prédictifs), les règles m-of-n (retourne vrai si au moins M d'un ensemble de N caractéristiques spécifiées sont présentes dans un exemple), etc. Parmi les différents types de boîtes noires, on peut citer les réseaux de neurones (cf. Figure 3), les machines à vecteurs de support, et d'autres ensembles de méthodes comme l'algorithme Random Forest (cf section 2.2.4).

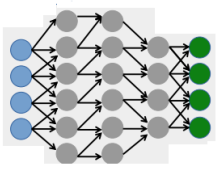

FIGURE 3 – Réseau neuronal : boîte noire

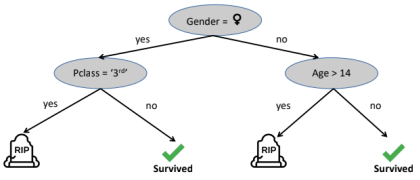

FIGURE 4 – Arbre de décision : boîte blanche

1.4 Interprétabilité

Au vu du dilemme présenté précédemment, pour combiner l'efficacité et l'interprétabilité, de nombreux travaux ont été réalisés afin d'expliquer les boîtes noires. Ils consistent à concevoir une couche d'interprétation entre le classifieur et l'utilisateur humain, comme le montre la Figure 5.

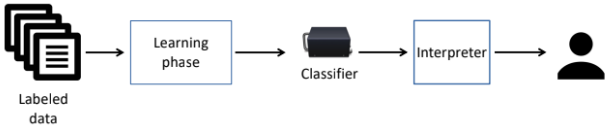

FIGURE 5 – Étape d'interprétation entre le classifieur et l'utilisateur humain

En effet, même s'il est impossible de savoir exactement ce qu'il se passe dans les boîtes noires, on peut interpréter le résultat du classifieur et essayer d'expliquer son choix et donc de créer de la confiance en son résultat. L'explication doit donc être compréhensible par l'utilisateur final.

Pour faire de l'interprétabilité sur les classifieurs boîtes noires, différentes techniques ont été étudiées afin d'établir un classifieur interprétable pour l'utilisateur. Ces méthodes se classent dans l'une des deux grandes familles suivantes :

— méthode d'explication globale,
— méthode d'explication locale.

L'explication globale revenant d'une certaine manière à du reverse engineering sur l'ensemble du classifieur (cf. Figure 6).

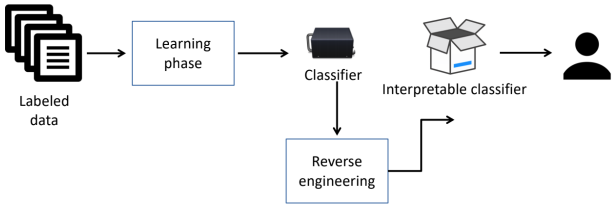

FIGURE 6 – Explication globale : reverse engineering

L'explication locale ne va pas chercher à faire du reverse engineering sur le modèle dans sa globalité, mais simplement à construire un modèle simple et compréhensible autour d'un exemple à expliquer. Sur la figure 7, il faut comprendre que le classifieur est construit à partir de toute la base d'apprentissage, tandis que le "local reverse engineering" est construit pour un exemple précis, par exemple l'image du chat.

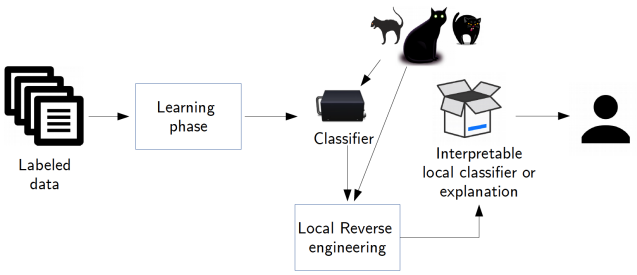

FIGURE 7 – Explication locale : local reverse engineering

Par exemple, dans le cas de la détection d'un objet ou d'un animal dans une image, il est possible d'extraire certaines parties de l'image qui ont permis ce choix. Ainsi pour l'exemple précédent concernant le chat et le poisson, l'interpréteur pourrait sortir l'image de la queue du chat pour montrer à l'utilisateur quel élément de la photo l'a aidé à faire son choix .

Pour le projet STLime, une méthode d'explication locale sera utilisée : LIME (**L**ocal **I**ntrepretable **M**odel-agnostic **E**xplanation). Cet algorithme est agnostic, c'est-à-dire qu'il peut être utilisé sur n'importe quel type de classifieur. Le projet s'appuyant entièrement sur cet algorithme, ce dernier est décrit en détail dans la section 3.

1.5 Séries temporelles

Une série temporelle est un ensemble de valeurs prises à intervalle de temps régulier. On peut donc voir une série temporelle comme un tableau de valeurs, souvent représentée sous forme de courbe sur un graphique. Les séries temporelles permettent, entre autres, de représenter l'évolution d'une variable numérique dans le temps, comme l'évolution de la température dans une journée (ou une semaine, un mois, une année), la consommation d'énergie d'un foyer tout au long de l'année ou encore les pulsations cardiaques d'une personne (électrocardiogramme), etc. Leur étude a de nombreuses applications dans le domaine médical, de la physique ou encore de l'énergie. Par exemple, l'entreprise rennaise Energiency récolte, grâce à des capteurs présents dans des usines, des séries temporelles correspondant à la consommation énergétique de ces dernières. Elle peut ainsi traiter ces données par la suite, et donner des conseils à ses utilisateurs.

Un exemple concret pour illustrer cette notion est celui d'un électrocardiogramme. Un électrocardiogramme affiche les pulsations du coeur d'un patient au cours du temps, c'est donc une série temporelle. Le médecin étudie cette mesure pour déterminer si le patient est malade ou non et, si oui, de quoi ce dernier souffre. Le médecin va alors se comporter comme un classifieur et donc chercher à "ranger" dans une classe cette série temporelle qu'est l'électrocardiogramme, les classes correspondant aux différentes maladies cardiaques connues.

La Figure 8 ci-dessous représente deux classes de séries temporelles. Chaque courbe est une série temporelle et représente un patient. Ces séries temporelles sont toutes deux étiquetées : les bleues correspondent à un patient en bonne santé, les rouges à un patient victime d'un infarctus.

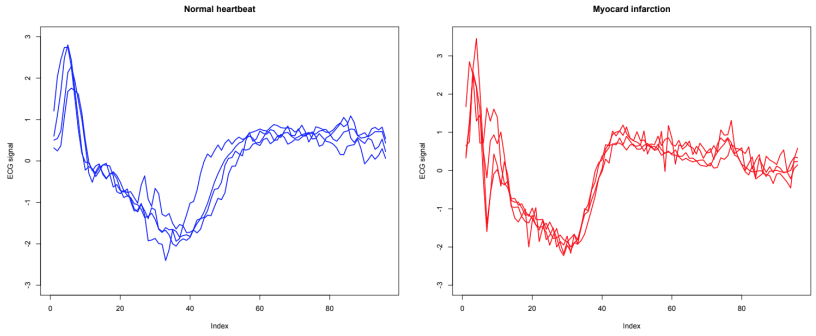

FIGURE 8 – Séries Temporelles

Pour replacer cette notion dans son contexte, les séries temporelles (qui seront mentionnées sous l'acronyme ST dans la suite de ce rapport) se situent au coeur de ce projet. Le but de l'étude est en effet d'implémenter un interpréteur de classifieur de séries temporelles.

1.6 STLime - Principe général

L'objectif de ce projet de 4ème année est d'adapter l'algorithme LIME pour l'interprétation de classifieurs de séries temporelles. Une première étape, très rapide, consiste à apprendre un classifieur de séries temporelles à partir d'une base d'apprentissage. Une deuxième étape, qui exige une quantité de travail importante, consiste à adapter LIME pour justifier la classification d'une série temporelle grâce à la présence de formes dans la série temporelle de départ. La Figure 9 ci-dessous illustre ce principe.

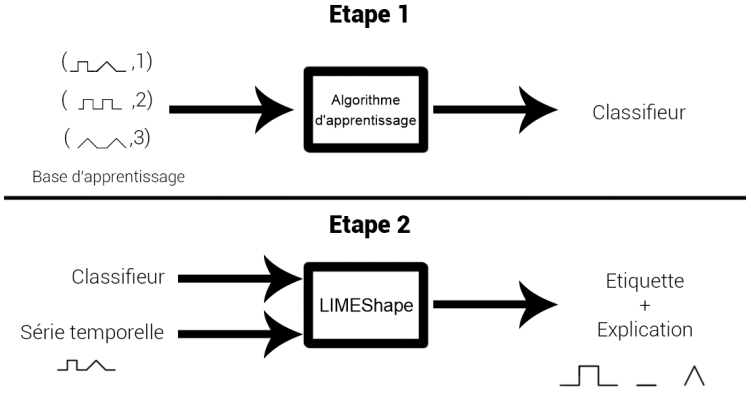

FIGURE 9 – Fonctionnement général de STLime

1.7 Mise en oeuvre

Dans le monde du machine learning, deux grands langages de programmation sont utilisés : Python et R. Tous les deux implémentent de nombreux algorithmes, R étant légèrement plus orienté statistique et Python étant un langage de programmation plus large. Dans le cadre de ce projet, le choix s'est effectué de lui-même : en effet, l'algorithme LIME est implémenté en Python et les différents algorithmes de classification sont présents dans des bibliothèques Python (Scikit-learn). De plus, ce langage est particulièrement adapté à la manipulation et au traitement des ST, au travers d'un certain nombre de bibliothèques. Python s'est donc avéré être le langage à utiliser pour cette étude.

Ce langage est multi-plateforme et est l'un des langages les plus utilisés pour les problèmes de classification, de Big Data et d'intelligence artificielle. Python est typé dynamiquement et nécessite donc une grande qualité de code pour être compréhensible et réutilisable. Il s'agit d'un langage interprété, et n'a donc pas besoin de compilation pour fonctionner ce qui le rend relativement facile d'utilisation.

Outre ses performances, Python est pratique d'utilisation de part son gestionnaire de paquet : PiP. Grâce à ce dernier, une simple commande suffit à installer une bibliothèque qui sera directement utilisable.

L'utilisation de Jupyter permet la création et l'utilisation de notebooks. Les notebooks sont des programmes interactifs dans lesquels on peut ajouter du code, du texte et des images. Ils permettent notamment de décrire pas à pas et de manière précise le fonctionnement d'un programme, d'un algorithme ou d'une bibliothèque. Ils peuvent aussi être utilisés pour faire du débogage (cf. Figure 18).

FIGURE 10 – Logo de Python

2 Classification de séries temporelles

STLime va consister à utiliser LIME pour interpréter des classifieurs de séries temporelles. Deux classifieurs vont être utilisés : 1NN-DTW et Random Forest couplé à Transform Shapelet. Concernant l'interprétation, LIMEShape indiquera les formes présentes dans la série temporelle (sous séries temporelles) qui ont le plus contribué à la classification de la ST. Cette section expliquera donc le classifieur 1NN-DTW, la notion de shapelet, l'algorithme Transform Shapelet et Random Forest.

2.1 1NN-DTW

Un algorithme qui peut être utilisé pour effectuer une classification est celui des K-NN (**K**-**N**earest **N**eighbors). Cet algorithme d'apprentissage supervisé utilise un jeu de données d'entraînement étiqueté, dans le cas du projet, il s'agira d'un ensemble de séries temporelles, certaines représentant par exemple des individus malades et d'autres, des individus sains. En donnant une ST à classifier, l'algorithme calcule la distance de cette série aux séries d'entraînement. La distance calculée peut être la distance euclidienne, la similarité cosinus, etc. Une fois toutes les distances calculées, on peut choisir les séries les plus proches de celle à classifier, l'étiquetage de ces dernières est alors pris en compte et la ST est classifiée selon la classe majoritaire des K plus proches voisins. Dans le cas de l'algorithme 1-NN, le procédé est le même, mais seule la série la plus proche est conservée et son étiquette est alors extraite.

Cependant, dans le cas des séries temporelles, certaines courbes peuvent être similaires mais décalées temporellement ou déformées. Elles seront donc considérées comme éloignées par la distance euclidienne alors qu'elles pourraient être classées de façon similaire. Pour compenser cela, l'algorithme de DTW (**D**ynamic **T**ime **W**arping) est utilisé (cf. Figure 11). Cet algorithme a pour objectif de calculer la correspondance optimale entre deux séquences tout en respectant certaines règles :

- chaque valeur de la première série doit être associée à au moins une valeur de la seconde et vice versa,
- la première valeur d'une série doit être associée à la première valeur de l'autre (mais peut être aussi associée à une autre),
- la dernière valeur d'une série doit être associée à la dernière valeur de l'autre (mais peut être aussi associée à une autre),
- l'association des indices de la première série à ceux de la deuxième doit être strictement croissante. Soient k, j des indices de la première série avec k $>$ j et m, n des indices de la seconde série avec m$>$n, alors l'association

$$k \iff n \tag{1}$$

implique que j ne peut pas être associé à m.

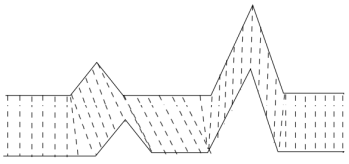

FIGURE 11 – Correspondance de deux séries temporelles utilisant le DTW

2.2 Random Forest - TransformShapelet

Pour pouvoir expliquer le deuxième classifieur, nous allons définir la notion de shapelet puis expliquer l'algorithme TransformShapelet et Random Forest.

2.2.1 Notion de shapelets

Une shapelet peut être assimilée à une sous série temporelle. Cette sous série a la particularité d'avoir une forme singulière et reconnaissable. Ainsi, parmi les séries temporelles d'entraînement, il existe des sous-ensembles qui ne sont présents que dans les séries temporelles d'une classe et pas dans les autres et inversement. Ce sont des shapelets. La présence ou l'absence d'une ou plusieurs shapelets dans la série temporelle qu'il faut classifier pourra donc être déterminante dans le choix du classifieur.

2.2.2 Shapelets utiles à la classification

Une shapelet est utile pour la classification si elle permet, par sa seule présence (ou absence), de bien distinguer les séries temporelles de classes différentes.

L'exemple ci dessous illustre ce concept : plusieurs séries temporelles étiquetées dans 2 classes distinctes et une shapelet (cf. Figure 13). Chaque série temporelle est représentée par une courbe (cf. Figure 12).

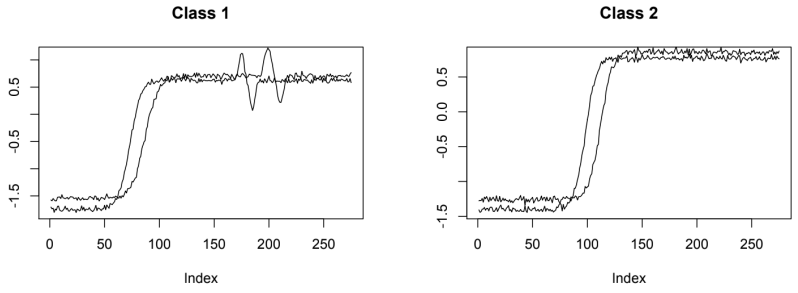

FIGURE 12 – Exemples de 2 courbes de ST

FIGURE 13 – Exemple de Shapelet

L'objectif est de déterminer si la shapelet S1 est représentative d'une classe de séries temporelles, autrement dit si elle est utile pour la classification. Ainsi, il faut calculer les distances entre la shapelet et toutes les sous sections (de même taille que la shapelet) d'une série temporelle que l'on cherche à étiqueter. La distance minimale mesurée servira à déterminer si la shapelet est utile ou non. Pour que la shapelet S1 soit caractéristique d'une classe, il faut que :

— les distances minimales entre S1 et les séries temporelles de la classe soient suffisamment faibles (il faut donc définir un seuil),
— les distances minimales entre S1 et les séries temporelles des autres classes soient élevées (toujours par rapport au seuil défini).

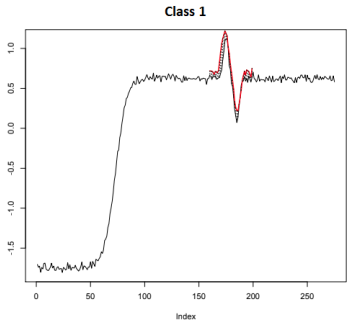

FIGURE 14 – Distance ST 1 - Shapelet

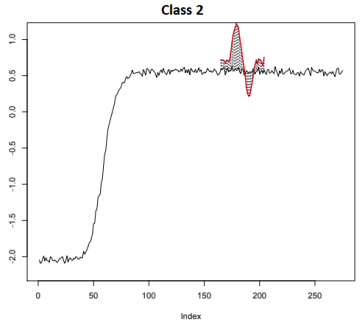

FIGURE 15 – Distance ST 2 - Shapelet

Dans la Figure 14, la distance entre S1 et la ST de la classe 1 est faible contrairement à la classe 2 dans la Figure 15. Si la même opération est ré-itérée sur l'ensemble des séries temporelles et que ce résultat est valable pour l'ensemble, alors la shapelet peut être considérée comme représentative de la classe 1.

La distance minimale entre une shapelet et une série temporelle peut être représentée sur un graphique de façon linéaire. Par exemple, avec deux classes ainsi que deux shapelets, si le but est de savoir si ces shapelets sont utiles pour différencier les deux classes, alors il suffit d'appliquer le calcul de distance entre les shapelets et les séries temporelles de chaque classe. Le graphique ci-dessous (cf. Figure 16) est alors obtenu. Chaque distance est représentée par un point rouge pour une classe et par un point bleu pour l'autre.

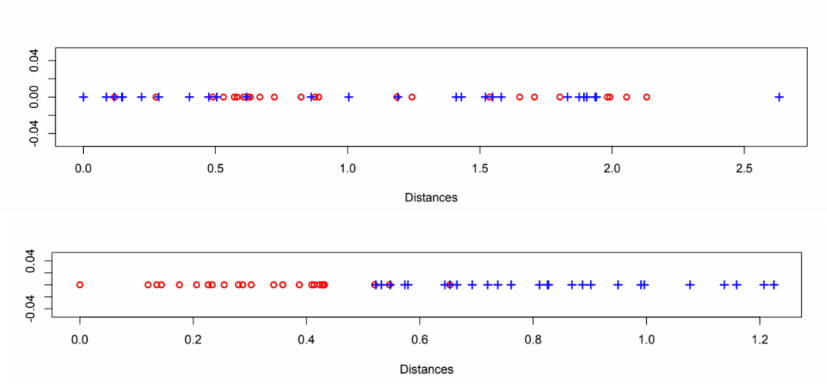

FIGURE 16 – Graphiques représentant les distances entre une shapelet et les ST

La première shapelet ne différencie pas suffisamment bien la classe rouge de la classe bleue. En effet les distances sont complètement mélangées. La seconde permet, elle, de bien séparer les deux classes. Il est même possible d'en déduire un seuil. La deuxième shapelet est donc bien meilleure et utile à la classification.

2.2.3 TransformShapelet

Pour utiliser la notion de shapelet avec un classifieur, plusieurs méthodes sont possibles. Dans le cadre de ce projet, la méthode Shapelet Transform sera utilisée. Elle consiste à :

- extraire les K meilleurs shapelets, qui nous seront données,
- transformer chaque ST en un vecteur V=(D1,D2,...,DK) de taille K où Di est la distance minimale entre la ST et la ième shapelet extraite,
- entraîner un classifieur sur ces vecteurs (ici Random Forest).

2.2.4 Random Forest

Dans ce projet l'algorithme Random Forest sera utilisé pour classifier des ST. Il prendra comme données d'entrée des vecteurs issus de l'algorithme TransformShapelet. C'est un algorithme d'apprentissage supervisé de type boîte noire. Random Forest crée un ensemble d'arbres de décision appelé forêt. Le principe de l'algorithme est donc d'apprendre plusieurs arbres de décision pour obtenir une prédiction plus précise et plus stable. Pour classifier un nouvel exemple, une stratégie est construite pour étiqueter l'exemple à partir de l'ensemble des décisions de chaque arbre. La prise de décision finale se fait donc à partir des décisions individuelles des arbres. Par exemple, l'algorithme peut attribuer l'exemple à la classe majoritaire. Le caractère aléatoire de ce modèle repose sur deux points :

— à chaque étape, l'algorithme choisit des échantillons aléatoires d'individus comme ensemble d'apprentissage puis des arbres sont créés,
— la construction d'un arbre se fait sur un sous-ensemble de variables tirées aléatoirement.

La Figure 17 ci-dessous illustre les différentes étapes de l'algorithme. La première étape consiste à diviser des données étiquetées en échantillons, ensuite tous les arbres de la forêt ont une phase d'apprentissage, puis une décision est prise en fonction des choix des arbres.

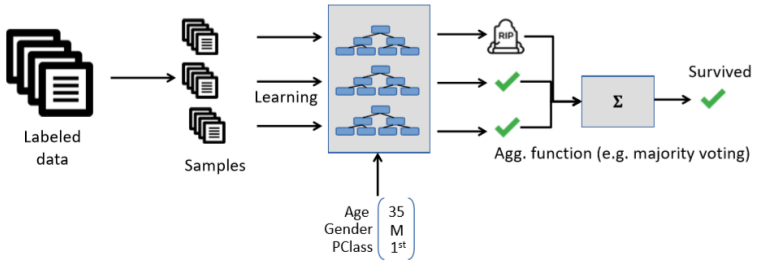

FIGURE 17 – Principe de l'algorithme Random Forest

L'un des grands avantages de ce classifieur est qu'il peut être utilisé à la fois pour les problèmes de classification et de régression. La principale limitation de Random Forest est qu'un grand nombre d'arbres peut ralentir et rendre l'algorithme inefficace pour les prévisions en temps réel. En général, ces algorithmes sont rapides à apprendre, mais assez lents pour créer des prévisions une fois qu'ils sont entraînés. Une prévision plus précise nécessite plus d'arbres, ce qui entraîne un modèle plus lent.

2.3 Bibliothèques : NumPy/Pandas/Tslearn/Scikit-learn

Pour modéliser les séries temporelles et les shapelets en Python et faire des opérations dessus, nous avons besoins de bibliothèques.

2.3.1 NumPy

NumPy est une bibliothèque qui fournit une interface pour stocker et effectuer des opérations sur des données, ainsi que diverses implémentations de tableaux. Les données que nous allons traiter pouvant être considérées comme des tableaux de nombres, NumPy va nous permettre de créer des tableaux multidimensionnels à partir de nos propres données et de les manipuler ensuite. Cette bibliothèque est principalement utilisée pour sa performance de calculs, notamment pour les calculs matriciels.

2.3.2 Pandas

La bibliothèque Pandas permet de manipuler des tableaux de données. Elle possède pour cela deux structures principales : les séries temporelles et les DataFrames. Là où l'on ne pouvait se référer à un élément d'un tableau NumPy que par sa position dans le tableau, chaque élément d'une série ou d'un DataFrame peut avoir un indice explicitement désigné par l'utilisateur.

Pandas permet de créer des séries temporelles à partir d'un tableau de valeurs. On peut alors créer des séries de int, de float, mais aussi de type venant de NumPy.

$$ex : dtype = numpy.int8 \tag{2}$$

Une fois la série créée, nous pouvons réaliser diverses actions en utilisant la notion d'index :

$$ex : pandas.Series([1, 2, 5, 7], index = [' a', ' b', ' c', ' d'] \tag{3}$$

accéder à la valeur d'un individu (cf. (3)), sommer les valeurs, les multiplier, etc. Pandas permet aussi de donner un nom à une série temporelle, demander la valeur des attributs de cette série, avoir le numéro d'ordre ou l'index de ses attributs, savoir quelles valeurs sont NaN et lesquelles ne le sont pas, créer une sous-série, etc.

Pandas permet également de travailler sur des DataFrames. On peut choisir l'ordre des colonnes lorsqu'on crée un DataFrame, le ré-indexer pour changer l'ordre des lignes et/ou des colonnes, trier selon les étiquettes ou les valeurs, ajouter ou supprimer des colonnes, avoir le nom ou la valeur des colonnes/lignes, modifier ces valeurs, accéder à des sous-ensembles du DataFrame avec le nom ou le numéro des colonnes, des lignes ou selon une condition, etc. Un DataFrame et une série, ou deux DataFrames, qu'ils aient les mêmes colonnes ou non, peuvent également être concaténés.

$$pandas.concat([df1, df2], join =' outer') \tag{4}$$

Le second paramètre est par défaut 'outer' puisqu'on ne veut pas avoir de valeurs NaN (cas où les DataFrames n'ont pas les mêmes index au niveau des colonnes). Cela revient à faire une union des valeurs des DataFrames. Si on choisit 'inner', on réalise une intersection.

$$pandas.merge(df1, df2, how =' outer', left\_on = [' C'], right\_on = [' D']), \tag{5}$$

On peut également réaliser une jointure entre DataFrames : on utilise par défaut 'outer' pour n'avoir que les lignes à valeurs positives, pour avoir toutes les lignes résultantes de la jointure dont celles contenant des valeurs NaN, on utilisera 'inner'. Si la jointure se fait sur les colonnes, on utilisera la fonction merge alors que si elle se fait sur les index on utilisera join :

$$df1.join(df2). \tag{6}$$

Avec Pandas, nous pouvons également manipuler des fichiers. Nous pouvons lire des DataFrames à partir de fichiers ou écrire des DataFrames dans des fichiers.

$$df = pandas.read\Gamma_csv(' f.csv', names = [' col1'], header = 0, dtype = ' col1' = typeCol1) \tag{7}$$

En créant le DataFrame en lisant un fichier, on peut imposer des noms aux colonnes (paramètre names). S'il y a un header dans le fichier que l'on lit, il faut ajouter header=0 en paramètre pour ne pas prendre en compte les noms de colonnes du header. On peut également préciser le type de certaines colonnes (paramètre dtype). On peut lire seulement le début d'un tableau en sélectionnant un nombre de lignes à lire (paramètre nrows) ce qui est utile pour le debugging.

$$df.to\_csv(' f.csv'). \tag{8}$$

Cet appel écrit le DataFrame dans le fichier f.csv. Si l'on veut écrire non pas dans un fichier csv mais sur la sortie standard, il faut alors utiliser sys.stdout comme paramètre.

Par défaut les paramètres principaux sont configurés de cette manière : le fichier utilise la virgule comme séparateur par défaut (sep=',' ou delimiter=','), contient un header (header = 0 lorsque l'on en veut un, None dans le cas contraire) et des cases vides comme valeurs NaN (na_rep = '').

2.3.3 Tslearn

Tslearn est une bibliothèque proposant une implémentation des algorithmes d'apprentissage spécialisés pour les séries temporelles. Pour Tslearn, une série temporelle est un tableau à une dimension : *(Température, pression, etc)*, chacune de ces valeurs étant séparée par le même intervalle de temps. Un ensemble de ST est donc un tableau à 2 dimensions, que l'on peut voir comme une liste de tableaux à une dimension. Cette représentation des séries temporelles permet de les afficher directement via Excel (au format csv). Il suffit donc de donner un tableau à la fonction to_time_series_dataset et on obtient une série temporelle exploitable avec Tslearn. Les traitements pour la classification des séries temporelles dérivent de ceux de Scikit-learn, a savoir :

— KMeans avec le module tslearn.clustering,
— KNeighbors avec le module tslearn.neighbors.

Elle possède aussi des méthodes de traitements et de construction pour les séries temporelles, notamment la création d'une série temporelle nulle, une fonction permettant de connaître la taille de la série temporelle, etc.

2.3.4 Scikit-learn

Scikit-learn est une bibliothèque dédiée au data mining et à l'analyse de données. Elle comporte de très nombreux algorithmes de Machine Learning. Elle comprend un grand nombre d'algorithmes de classification comme les arbres de décision et les forêts aléatoires. La fonction RandomForestClassifier permet de créer une Random Forest en spécifiant le nombre d'arbres de notre forêt. Il est possible d'ajouter d'autres paramètres tels que la profondeur maximale de l'arbre, le critère de mesure de la qualité de la division, etc. Pour appliquer la forêt sur les données d'entraînement il faut utiliser la fonction fit sur notre classifieur en lui précisant l'ensemble voulu. On peut alors obtenir les prédictions du classifieur sur un ensemble de test avec la fonction predict qui retourne un tableau des classes prédites. L'utilisation de Scikit-learn implique souvent celle de la bibliothèque Matplotlib pour afficher les résultats obtenus.

Afin de mieux appréhender les fonctions proposées par cette bibliothèque, un Notebook Jupiter a été créé (cf. Figure 18). Il s'agit d'un exemple d'utilisation de l'algorithme Random Forest présent dans Scikit-learn.

Importation des librairies.

```
import pandas as pd
import os
import sklearn.metrics
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
```

Chargement de la base de données :

```
os.chdir("C:\\Users\\lisa\\Documents\\Cours\\Projet IST\\Python")
AH_data = pd.read_csv("tree_addhealth.csv")
data_clean = AH_data.dropna()
```

Construction du classifieur :

Choix de l'attribut à prédire et des attributs à prendre en compte. Ici, l'attribut à prédire est "TREG1" qui prend 0 ou 1 comme valeur. Répartition des individus entre les ensembles d'entraînement et de tests. Construction du modèle sur le jeu de données d'entraînement avec une forêt de 25 arbres.

```
predictors = data_clean[['BIO_SEX','HISPANIC','WHITE','BLACK','NAMERICAN','ASIAN','age','ALCEVR1']]
targets = data_clean.TREG1
pred_train, pred_test, tar_train, tar_test = train_test_split(predictors, targets, test_size=.4)
pred_train.shape
pred_test.shape
tar_train.shape
tar_test.shape
classifier = RandomForestClassifier(n_estimators=25)
classifier = classifier.fit(pred_train,tar_train)
```

Utilisation du classifieur :

Affichage de l'importance relative de chaque attribut. Prédiction la classe d'un individu : ici le premier est rangé dans la classe 0 et le deuxième dans la classe 1.

```
print(classifier.feature_importances_)
print(classifier.predict([[2,0,1,0,0,0,15.509589,0,0,0,0,0,0,13,29,1,1,6,26,3,0,21.3,1,12]]))
print(classifier.predict([[1,0,1,0,0,0,15.093151,1,4,1,0,0,1,25,50,13,0,2,14,1,0,23.7,9,13]]))
```

```
[0.01475071 0.0080988  0.01652607 0.02241781 0.00452313 0.0032066
 0.08301043 0.05143089 0.05827232 0.09382339 0.0136514  0.01304875
 0.01984046 0.06881833 0.06230181 0.04393547 0.00952593 0.08129905
 0.07488663 0.07877014 0.0065869  0.07234981 0.0563833  0.04254187]
[0.]
[1.]
```

FIGURE 18 – Extrait du Notebook Jupiter sur l'utilisation des Random Forest

3 Interprétabilité d'un classifieur de Séries Temporelles

Dans cette partie, il est nécessaire de définir ce qu'est une explication avant d'expliquer le fonctionnement de l'algorithme LIME .

3.1 Qu'est ce qu'une explication ?

L'objectif de LIME est d'expliciter quelle classe un classifieur a choisi et de le justifier. La justification met en avant les parties de l'exemple qui ont beaucoup influencé la classification. C'est un algorithme model-agnostic, c'est-à-dire que celui-ci peut-être utilisé sur n'importe quel type de classifieur (Random Forest, Support Vector Machine, etc.).

3.2 LIME

Une explication va donner à l'utilisateur des caractéristiques de l'objet de départ qui ont fortement contribué à la classification de celui-ci.

FIGURE 19 – Exemple d'explication pour la classe "présence d'un chat" sur une image

Sur la Figure 19, les pixels que le classifieur a jugé pertinents pour classer cette image dans la classe "présence d'un chat" apparaissent en vert. Les pixels apparaissant en rouge sont ceux qui ont pesé négativement sur le classement de l'image dans ladite classe.

Une explication peut être décrite comme étant une approximation locale du modèle. Un modèle étant généralement complexe dans sa globalité, expliquer une sous partie comportant l'élément dont on souhaite expliquer la classification est plus abordable. Une explication peut-être obtenue en modifiant légèrement l'objet d'entrée et en regardant si le nouvel objet est toujours attribué à la même classe. Par exemple, pour un texte, on peut supprimer un mot et si le texte n'a plus la même classe cela signifie que le mot est important. En effectuant ce travail (classifier un élément proche) un certain nombre de fois sur des points relativement proches de l'objet à expliquer, il va être possible d'apprendre un nouveau classifieur simple qui s'approche du modèle autour de l'exemple que l'on cherche à expliquer (un classifieur linéaire par exemple).

3.3 Fonctionnement de LIME

Le fonctionnement de LIME est décrit par la figure 20. L'exemple de départ à expliquer est appelé x. Une première étape consiste à représenter x avec des attributs simplifiés compréhensibles par un utilisateur quelconque. Un exemple très simple consiste à représenter un texte par la présence/absence de mots donnés par un dictionnaire. Une deuxième étape consiste à construire un ensemble de voisins de x appelé $\{z^{'}\}$. L'étape suivante est caractérisée par la construction d'un nouveau classifieur simple pour l'ensemble $z^{'}$. Les attributs simplifiés utilisés à la première étape qui ont été importants dans la classification sont mis en avant et indiqués à l'utilisateur à la fin de l'algorithme. Chacune de ces étapes vont être détaillées par la suite.

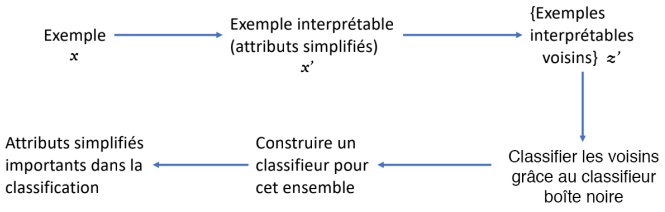

FIGURE 20 – fonctionnement général de LIME

3.3.1 Simplification des attributs

La première étape se fait en passant de l'exemple que l'on souhaite expliquer à sa version simplifiée qui consiste à le découper en un ensemble d'éléments simples présents ou absents. Par exemple pour un texte, on pourra le découper en un vecteur de taille égale au nombre de mots présents dans le dictionnaire et à chaque mot on associe un '0' s'il est absent dans le texte et un '1' s'il est présent. La Figure 21 décrit la simplification de x.

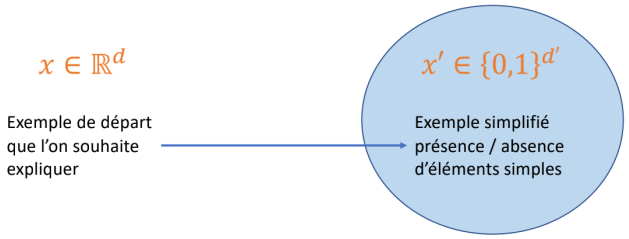

FIGURE 21 – Simplification de x

L'exemple d'une image est visible dans la Figure 22, l'image est divisée en superpixels (parties similaires), et la présence de ce superpixel est décrite par un 1 (présent) ou un 0 (absent). Pour cet exemple, le vecteur de taille n associé à l'image de droite serait $(1, 1, ..., 1)$.

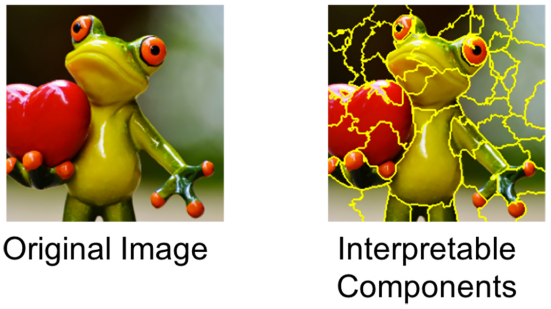

FIGURE 22 – Division de l'image de base en superpixels

Le principe est le même pour le traitement d'un texte. Si le dictionnaire de départ est {"le", "chat", "est", "mort"} alors :
— Le chat est mort → (1,1,1,1)
— chat mort → (0,1,0,1)

3.3.2 Génération et classification des voisins

Pour générer les voisins, l'exemple simplifié est perturbé en retirant aléatoirement des attributs. De ce fait, ces nouveaux exemples se retrouveront dans le voisinage de l'exemple de départ. Une fois les voisins générés, il faut les repasser dans le classifieur pour qu'ils puissent être classifiés à leur tour. Cependant ces voisins sont dans un format simplifié que le classifieur ne peut plus traiter, il faut donc les reconvertir dans le format d'entrée du classifieur. Une fonction $h_x : \{0,1\}^d \to \mathbb{R}^d$ qui connaît l'exemple de départ est alors utilisée. Un exemple sur la Figure 23 est présenté dans lequel les superpixels supprimés sont grisés.

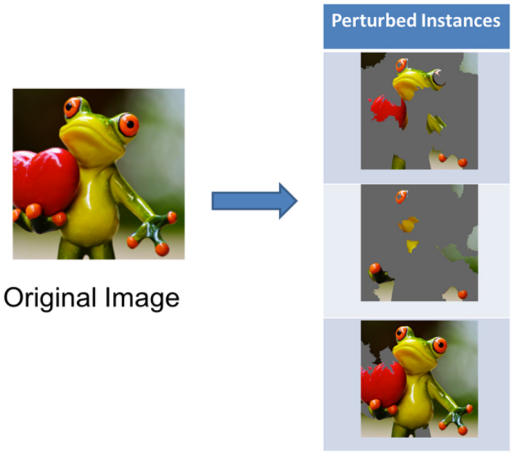

FIGURE 23 – Exemples de voisins générés

3.3.3 Construction et choix du nouveau classifieur

Soient les notations suivantes :
- G : ensemble de tous les modèles explicatifs (par exemple l'ensemble de tous les modèles linéaires sur $\mathbb{R}^{d'}$)
- f : classifieur BW utilisé
- π_x : une mesure de proximité (elle va permettre d'accorder un poids plus important aux voisins les plus proches de x)
- $\mathcal{L}$: une mesure fidélité de qui va permettre de mesurer à quel point g est fidèle à f sur l'ensemble des voisins
- Ω : une mesure de complexité de g (par exemple, plus g intègre des variables plus elle est complexe et difficile à expliquer à l'utilisateur)

Construire un nouveau classifieur revient à trouver le modèle linéaire g qui minimise $\mathcal{L}(f, g, \pi_x) + \Omega(g)$, c'est-à-dire qui minimise à la fois la précision et la complexité. Cette étape consiste à choisir le classifieur linéaire qui combine le mieux interprétabilité et fidélité. Pour cela on utilise cette formule mathématique qui combine ces 2 aspects :

$$argmin_{g \in G}\, \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

Sur LIME, $\mathcal{L}$, π_x et Ω sont définis comme suit (avec $G = \{g \in G / g(z') = w_g z'\}$) :
- $\mathcal{L}(f, g, \pi_x) = \Sigma_{(z,z') \in Z} \pi_x(z)(f(z) - g(z'))^2$
- $\pi_x(z) = e^{-D(x,z)^2/\sigma^2}$ avec σ = écrat type des voisins + x
  - Texte : D = distance cosinus
  - Image : D = distance euclidienne
- Ω
  - Texte : $\Omega = \begin{cases} \infty & \text{si le nombre de poids non nuls est plus grand qu'un K fixé} \\ 0 & \text{sinon} \end{cases}$
  - Image : Idem mais pour le nombre de superpixel gardés de l'image initiale

Trouver la solution exacte d'une telle formule est combinatoirement non réalisable, par contre il existe des méthodes d'approximation :
- régression linéaire : minimise $\sum_{x \in X} ||\beta x - y||_2$. Une telle solution vise à minimiser la précision $\mathcal{L}$ mais ne prend pas en compte la complexité Ω.
- Lasso : minimise $\sum_{x \in X} ||\beta x - y||_2 + \lambda ||\beta||_1$. Une telle solution vise à minimiser $\mathcal{L}$ et prend en compte la complexité grâce au second membre de l'addition. λ est un paramètre, lorsque qu'il augmente le nombre d'attributs de la solution diminue.
- K-Lasso : plusieurs Lasso successifs sont effectués en augmentant λ afin de diminuer le nombre d'attributs tant que celui-ci est strictement supérieur à K. Une telle solution permet d'avoir à la fois un modèle fidèle et simple.

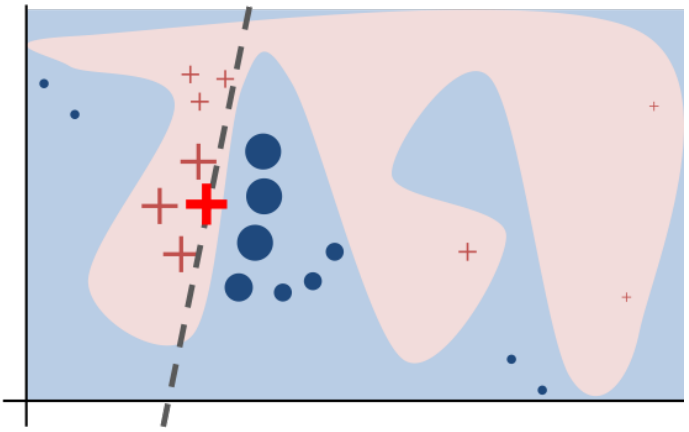

FIGURE 24 – Nouveau classifieur choisi

Un exemple d'un nouveau classifieur simple est présenté dans la Figure 24. Le modèle général, composé de 2 classes, est représenté par le fond de couleur (bleu et rose), c'est un modèle évidemment non-linéaire. La croix rouge mise en évidence est l'élément à expliquer. Les voisins proches sont générés et classés en fonction de leur similarité avec l'élément de base. Un modèle simple peut alors être appris en fonction de la classification précédemment effectuée (souvent un modèle linéaire, ici représentée par la ligne en pointillé).

3.3.4 Bilan

Le fonctionnement global de LIME revient à prendre une instance et à la classifier à l'aide d'un classifieur boîte noire dans un premier temps (exemple d'une image de grenouille dans la Figure 25). Une fois l'instance classifiée, celle-ci est transformée en un vecteur de '0' (absence d'un attribut) ou de '1' (présence d'un attribut) puis perturbée en supprimant aléatoirement des attributs ce qui permet de générer des voisins. Ces voisins ont à leur tour besoin d'être classifiés, ils sont donc reconvertis dans leur ensemble de départ en utilisant une fonction h_x puis classifiés à l'aide du classifieur de départ. En fonction du résultat du classifieur boîte noire, le meilleur classifieur linéaire interprétable sur l'espace simplifié est construit en utilisant une méthode d'approximation. Une fois cette étape passée, ce classifieur est utilisé pour expliquer le résultat du classifieur boîte noire de départ en fournissant le poids des attributs simplifiés qui ont le plus contribué à ce résultat.

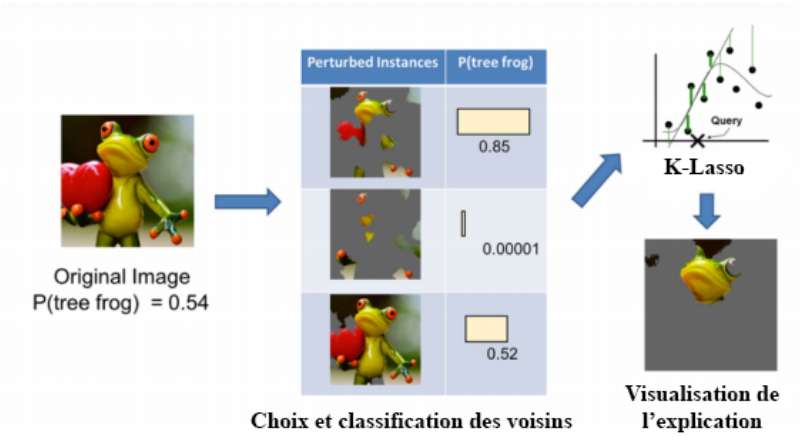

FIGURE 25 – Fonctionnement global de LIME

4 Objectifs du projet

Après avoir expliqué ce qu'est un classifieur, une série temporelle, l'algorithme LIME qui permet une interprétation locale, il est maintenant possible de présenter les objectifs du projet.

4.1 Construction et sauvegarde de classifieurs de séries temporelles

L'objectif de cette partie est de construire, entraîner et sauvegarder un classifieur de séries temporelles. Deux algorithmes vont être développés :

— 1NN-DTW,
— Random Forest avec TransformShapelet.

Cette partie sera relativement simple. En effet, les algorithmes étant présents dans Scikit-learn, il suffira de les utiliser.

4.2 Développement de LIMEShape

L'objectif de cette deuxième partie est de développer l'algorithme LIMEShape. LIME-Shape va prendre en entrée un classifieur de séries temporelles et une série temporelle et va retourner une classification associée à une explication. LIMEShape s'appuie entièrement sur l'algorithme LIME. Les différences sont la définition des attributs simplifiés, la visualisation d'une explication, la définition de la fonction h. Cette fonction permet de reconstruire une série temporelle à partir de la série temporelle de départ et d'un vecteur représentant les éléments de la série temporelle ayant été supprimés.

4.3 Interface graphique

Pour rendre STLime compréhensible par l'utilisateur, une interface graphique sera implémentée. Les principales fenêtres sont :

— la fenêtre "Classifieur" (cf. Figure 26) permettra à l'utilisateur de construire un classifieur et de le sauvegarder. Il pourra charger un classifieur déjà sauvegardé s'il le souhaite.
— la fenêtre "ST" (cf. Figure 27) permettra de charger une série temporelle et de la visualiser sur un graphique.
— la fenêtre "Shapelet" (cf. Figure 28) permettra de charger une shapelet et une série temporelle. Il sera alors possible d'afficher et de visualiser, par exemple, la plus petite distance entre la shapelet et la série temporelle.
— la fenêtre "LIME" (cf. Figure 29) permettra d'avoir l'explication de la classification des séries temporelles. Il faudra rentrer plusieurs paramètres puis l'explication sera affichée.

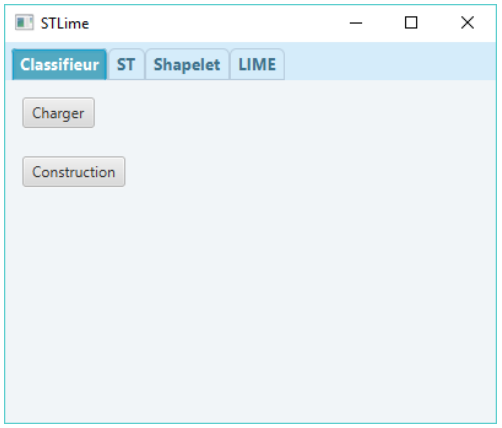

FIGURE 26 – Premier onglet : Classifieur

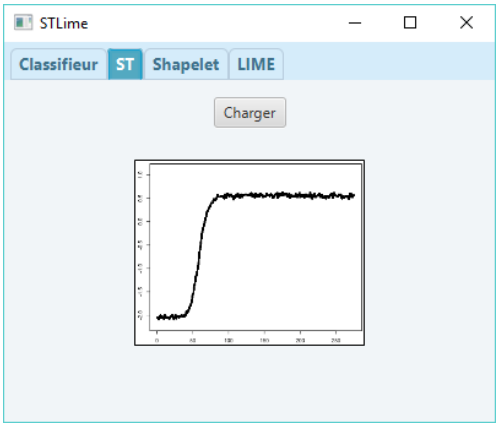

FIGURE 27 – Deuxième onglet : Séries temporelles

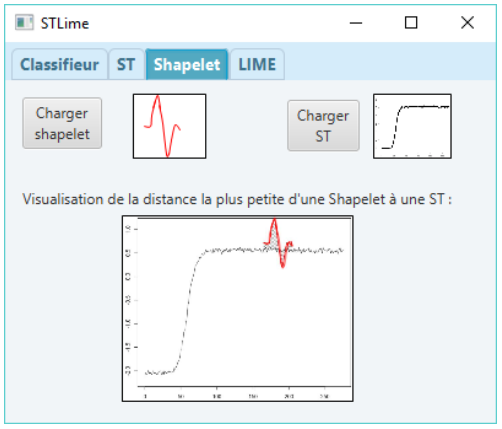

FIGURE 28 – Troisième onglet : Shapelet

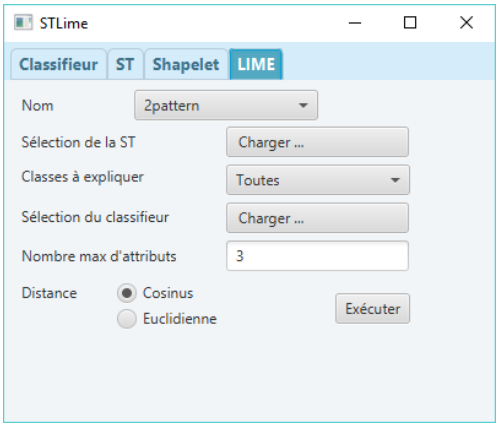

FIGURE 29 – Quatrième onglet : Algorithme de LIME

Pour créer cette interface graphique, plusieurs possibilités sont à envisager. En effet, il existe des bibliothèques Python permettant de créer des interfaces graphiques multi-plateforme. La plus connue est la bibliothèque Tkinter mais il est également possible d'utiliser Qt. Qt est un framework multi-plateforme développé en C++ qui permet notamment de créer des interfaces graphiques utilisateurs (GUI). Mais des bindings existent pour utiliser Qt dans d'autre langages que le C++. C'est le cas de Qt for Python (Pyside2) qui permet d'utiliser Qt avec Python. Lors de l'implémentation de l'interface, il faudra donc choisir entre Tkinter et Qt.

Pour tout ce qui concerne la création de graphiques, la bibliothèque Python Matplotlib est la plus appropriée. Cette bibliothèque permet de tracer et visualiser des données sous forme de graphiques. Elle sera utile pour afficher les ST.

4.4 Travail en groupe : Outils collaboratifs

Réaliser ce projet à huit est une expérience enrichissante sur la conception du travail en équipe. En effet, ce projet s'ancre dans nos premières expériences de collaboration pour un projet sur un aussi long terme. Construire un projet ensemble favorise l'échange et donc l'émergence de nouvelles idées. Ce projet permet d'alterner travail individuel et collectif. Les résolutions de chaque problème se font au niveau de chaque membre d'abord, puis au niveau du groupe. Ainsi après confrontation, il est possible de définir en commun de nouvelles orientations et la construction progressive des connaissances de chacun est favorisée. À tous ces points positifs s'ajoute le fait que ce travail collaboratif apporte un effet de motivation. En effet, une bonne ambiance règne au sein de notre groupe, permettant ainsi de définir clairement les objectifs.

Pour favoriser ce travail en équipe, des outils ont permis de se répartir les tâches au mieux. Ainsi, un Trello a été créé, qui est un outil de gestion de projet en ligne, inspiré de la méthode Kanban de Toyota. Cet outil est basé sur une organisation du projet en planches listant des cartes, qui représentent des tâches. Les cartes sont assignables à des utilisateurs et sont mobiles d'une planche à l'autre, traduisant leur avancement. Chacun peut ainsi constater l'avancement de chaque membre du groupe au sein du projet. La création d'un Google Drive a également permis de réunir tous les documents que chaque membre du groupe utilise pour les partager et les comptes rendus de chaque semaine sont tous présents dans ce dossier. Enfin, l'utilisation de Git permettra le versionnage de notre travail. Cela pourra non seulement permettre de toujours avoir une version fonctionnelle des parties des différents codes mais aussi de partager efficacement le code sur GitHub.

FIGURE 30 – Logos de Git, Google Drive et Trello

4.5 Planification

En annexe se trouve le diagramme de planification en date du 25 octobre 2018.

Maintenant que la phase de pré-étude est terminée, la phase de spécification va débuter. Cette phase va se dérouler en parallèle de la phase de planification. Ces phases termineront la première partie du projet. La seconde partie suivra avec les phases de conception, de réalisation et de tests.

Conclusion

La phase de pré-étude, première phase du projet se termine. Elle nous a permis de comprendre toutes les notions utiles pour ce projet : classifieurs boîtes noires, boîtes blanches, séries temporelles, algorithmes d'interprétation locale de classifieurs (LIME). Une fois ces notions bien assimilées, il est possible de formuler très clairement les objectifs de ce projet : adapter l'algorithme LIME afin de pouvoir l'utiliser pour n'importe quel classifieur de ST et pouvoir expliquer la classe attribuée à une ST.

Cette phase de pré-étude a également permis de dégager une première structure de ce projet. La première étape va consister à construire des classifieurs de séries temporelles à partir d'une base d'apprentissage. Cette étape sera réalisée avec Random Forest et TransformShapelet ou 1NN-DTW. La seconde étape, utilisera LIME pour justifier la classification d'une série temporelle grâce à la présence de formes dans la série temporelle de départ.

Les actions de cette première phase ont été réalisées en équipe. Cependant, chaque membre du groupe en charge de ce projet a acquis et va continuer d'acquérir durant les prochaines phases des connaissances de manière autonome. Ce qui est un aspect primordial dans la réalisation d'un projet ingénieur. Ces compétences seront des apports en matière de technique et gestion de projet.

La prochaine phase du projet est la phase de spécification. Il faudra décrire en détail les spécifications fonctionnelles du projet ainsi qu'une première idée de son architecture logicielle générale.

Annexe

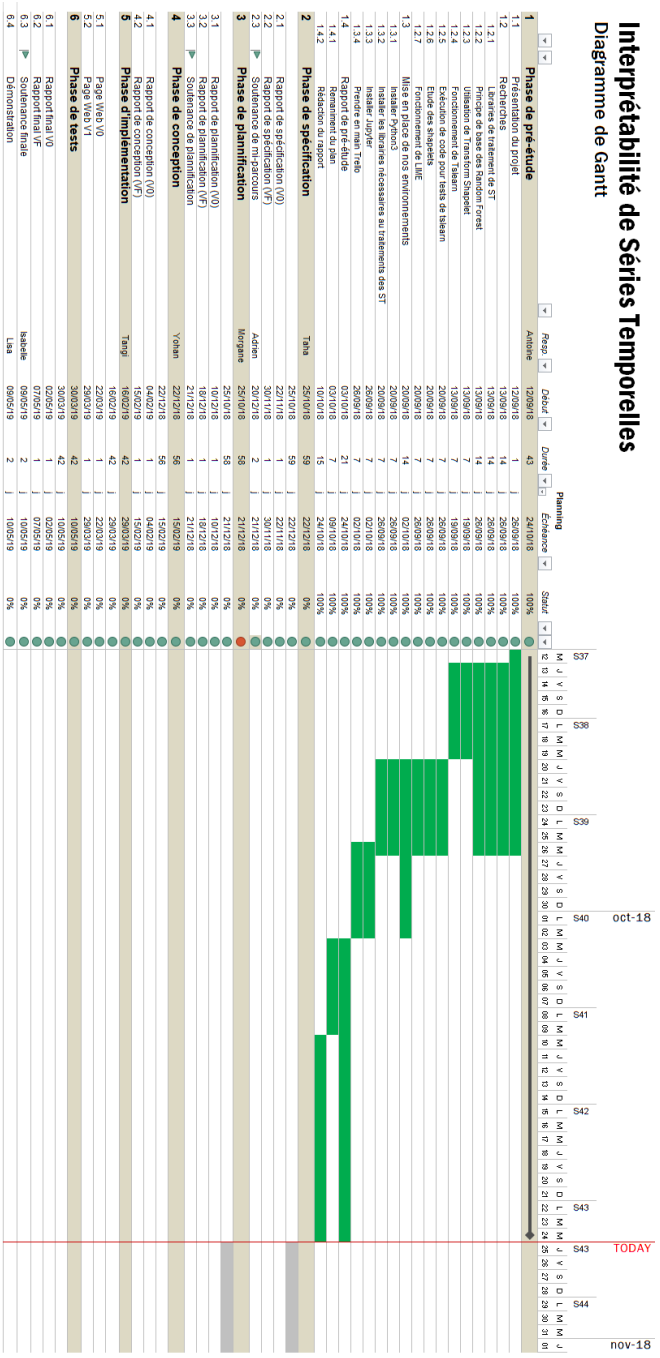

FIGURE 31 – Diagramme de planification du 25-10-2018