



INSTITUT NATIONAL DES SCIENCES APPLIQUÉES
DE RENNES

TSExplanation Rapport de conception

Adrien BURIDANT
Yohan COUANON
Isabelle GUILLOU
Tangi MENDÈS
Lisa RELION

Responsables de projet :
Laurence ROZÉ (INSA, INRIA, IRISA)
Maël GUILLEMÉ (ENERGIENCY)

ENERGIENCY

 UMR IRISA

Septembre 2018 - Mai 2019
INSA de Rennes

Table des matières

Introduction	2
1 Apprentissage et Sauvegarde du classifieur	3
1.1 Classes pour l'apprentissage et la sauvegarde	3
1.1.1 Fonctions de la classe <i>ImportTS</i>	3
1.1.2 Fonctions de la classe <i>LearningClassifier</i>	3
1.2 Déroulement	4
2 LIME	5
2.1 Adaptation des classes de LIME aux séries temporelles	5
2.1.1 <i>TSDomainMapper</i>	6
2.1.2 <i>IndexedTS</i>	6
2.1.3 <i>TSEExplainer</i>	7
2.2 Fonctionnement de LIMEShape	7
2.2.1 Découpage de la série temporelle en sous-séries temporelles	8
2.2.2 Génération des voisins	8
2.2.3 Explication	9
3 Interface graphique	10
3.1 Différentes classes	10
3.2 Classe App	10
3.3 Classe UI_Explanation	12
Conclusion	14
Bibliographie	15
Annexe	16

Introduction

La phase de pré-étude, première phase du projet, 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). Ces notions étant éclaircies, il a par la suite été 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 séries temporelles et de pouvoir expliquer le choix de la classe attribuée à une série temporelle. Dans la suite de ce rapport, le terme "série temporelle" sera désigné par l'acronyme "ST".

Comme il a été expliqué dans le rapport de spécification, la première étape va consister à construire des classifieurs de séries temporelles à partir d'une base d'apprentissage. Cette étape pourra être réalisée avec deux algorithmes : Learning Shapelet [6] ou 1NN-DTW. La seconde étape utilisera LIME pour justifier la classification d'une série temporelle grâce à la présence ou absence de formes particulières (sous-séries temporelles) dans la série temporelle de départ. Enfin, une interface graphique viendra s'ajouter à l'outil afin de faciliter son utilisation.

Dans ce rapport seront décrites la conception et l'architecture logicielle de l'outil TSEExplanation. Les différentes parties du code à réaliser seront présentées et expliquées, des diagrammes UML de classe et de séquence viendront s'ajouter aux explications techniques.

Dans un premier temps, les détails de la conception d'un classifieur de séries temporelles seront abordés. L'objectif sera de revenir sur les classes et les fonctions à implémenter pour réaliser l'apprentissage de la sauvegarde d'un tel classifieur. Dans un second temps, l'adaptation de l'algorithme LIME au cas des séries temporelles sera décrite en détail. Cette explication présentera les ajouts qui vont être faits à l'algorithme LIME. Le but est de pouvoir traiter des séries temporelles et non des images ou des textes. Enfin, le développement de l'interface graphique de l'outil final sera détaillé. Il s'agira de présenter comment l'interface graphique, indispensable à l'utilisation de l'outil TSEExplanation, va être réalisée sur le plan technique.

La Figure 1 rappelle le fonctionnement de l'outil TSEExplanation.

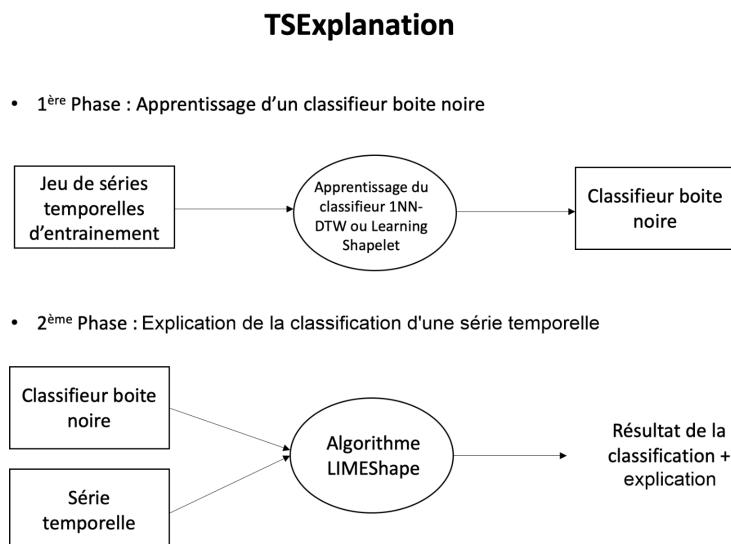


FIGURE 1 – Illustration du fonctionnement de TSEExplanation

À titre d'information, toutes les classes et méthodes de l'outil TSEExplanation présentées dans la suite de ce rapport n'existent pas encore et sont à implémenter.

1 Apprentissage et Sauvegarde du classifieur

TSExplanation se décompose en plusieurs modules. Le premier module sert à l'entraînement du classifieur. Ce module a pour mission d'importer des séries temporelles étiquetées, d'entraîner le classifieur avec ces séries et de pouvoir sauvegarder ce dernier. Pour rappel, l'utilisateur aura le choix entre deux algorithmes de classification : 1NN-DTW et Learning Shapelet[1][6][5].

1.1 Classes pour l'apprentissage et la sauvegarde

Ce module comporte deux classes. L'une est consacrée à l'importation des séries, l'autre à l'entraînement et à la sauvegarde. Cette partie du logiciel utilise des fonctions de la bibliothèque Tslearn [3]. Cette bibliothèque fournit des outils de machine learning pour l'analyse de séries temporelles. Le diagramme UML de ces deux classes se trouve sur la Figure 2.

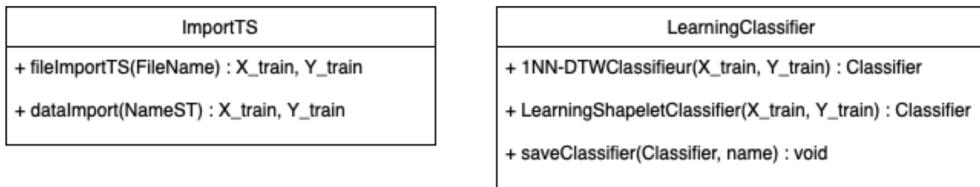


FIGURE 2 – Diagramme UML

1.1.1 Fonctions de la classe *ImportTS*

La classe *ImportTS* contiendra les fonctions suivantes :

- La fonction *fileImportTS(FileName)* permet d'importer des séries temporelles d'entraînement se trouvant dans le fichier *FileName*. Ces séries temporelles se retrouveront stockées en deux tableaux *X_train* et *Y_train*. *X_train* est un tableau à deux dimensions. La première correspond au numéro de la ST et la deuxième aux valeurs. Par exemple *X_train[5]* donne toutes les valeurs de la 5ème série temporelle d'entraînement, *X_train[5][70]* donne la 70ème valeur de la 5ème série temporelle. Pour l'entraînement du classifieur, il faut ajouter un tableau à une dimension, à savoir *Y_train*, qui contient les classes auxquelles appartiennent les ST. Par exemple si la 5ème ST est de la classe 1, *Y_train[5]* sera égal à 1. La fonction *fileImportTS(FileName)* renvoie ces deux tableaux.
- La fonction *dataImport(TSName)* effectue le même travail que la fonction *fileImportTS(FileName)*. La seule différence est que les ST importées ne proviennent pas d'un fichier mais de la base de données UEA UCR Time Series Classification Repository.

1.1.2 Fonctions de la classe *LearningClassifier*

La classe *LearningClassifier* contiendra les fonctions suivantes :

- La fonction *1NN-DTWClassifier(X_train, Y_train, name)* permet d'apprendre un classifieur avec l'algorithme *1NN-DTW* sur des séries temporelles passées en paramètre et d'appeler la fonction de sauvegarde.

- La fonction `learningShapeletClassifier(X_train, Y_train, name)` permet d'apprendre un classifieur avec l'algorithme *LearningShapelet* sur des séries temporelles passées en paramètre et d'appeler la fonction de sauvegarde.
- La fonction `saveClassifier(Classifier, name)` permet de sauvegarder le classifieur dans un fichier nommé *name* afin de pouvoir le récupérer plus tard.

1.2 Déroulement

Pour expliciter le déroulement de ce premier module, un diagramme de séquence illustrant l'apprentissage d'un classifieur 1NN-DTW se trouve sur la Figure 3.

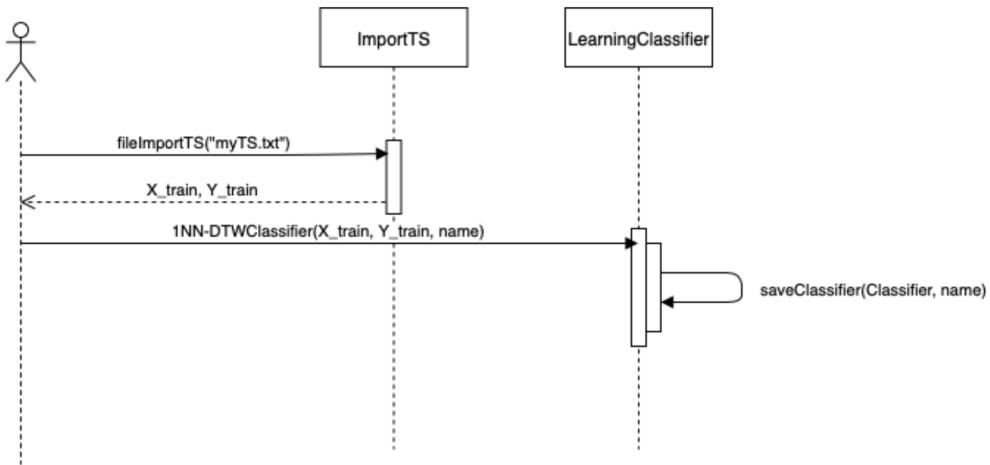


FIGURE 3 – Diagramme de séquence simulant l'apprentissage et la sauvegarde d'un classifieur

L'utilisateur sélectionne ses paramètres sur le logiciel (nom du fichier contenant les séries temporelles, type du classifieur, nom du fichier de sauvegarde). Une fois qu'il aura choisi ses paramètres, les fonctions des classes *ImportTS* et *LearningClassifier* seront appelées.

Dans un premier temps, la fonction d'importation des séries temporelles sera appelée, ici `fileImportTS(File Name)`, et retournera les deux tableaux contenant les ST (`X_train` et `Y_train`).

Dans un second temps, la fonction d'apprentissage du classifieur sera appelée à son tour, ici `1NN-DTWClassifier(X_train, Y_train, name)`, avec en paramètre les deux tableaux précédemment obtenus. Cette fonction va apprendre le classifieur puis va appeler la fonction de sauvegarde `saveClassifier(Classifier, name)`. Le classifieur sera alors sauvegardé dans un fichier.

Une fois ces étapes terminées, l'utilisateur aura à sa disposition, dans le répertoire souhaité, le fichier de sauvegarde du classifieur. En effet, au vu du temps d'apprentissage important, il est impensable que le classifieur soit réappris à chaque utilisation. C'est pourquoi il est important de sauvegarder le classifieur dans un fichier. Celui-ci sera alors chargé pour la partie explication des prédictions du classifieur (LIMEShape). Le détail de la conception de LIMEShape (2ème module) se trouve à la suite de ce rapport.

2 LIME

Le projet TSExplanation adapte un algorithme d'explication de classification (LIME) pour des exemples de types de séries temporelles, et donc pour des classificateurs de séries temporelles. LIMEShape, l'algorithme LIME [13], [7] qui va être adapté aux séries temporelles, indiquera les sous-séries temporelles présentes dans la série temporelle qui ont le plus contribué à sa classification. Cette partie va donc illustrer la 2ème phase de la Figure 1.

2.1 Adaptation des classes de LIME aux séries temporelles

Les modules *lime_text* et *lime_image* qui s'adaptent à des domaines spécifiques (texte et image) [4], [2] ont été détaillés dans le rapport de spécification. Le but pour TSExplanation est d'adapter de la même manière un module avec des fonctions dédiées pour le domaine des séries temporelles. L'algorithme LIMEShape va nécessiter différentes fonctions pour faire son cheminement. Il faut dans un premier temps découper la série temporelle en segments de sous-séries temporelles. Puis les voisins sont générés en modifiant des sous-séries temporelles. Grâce à la classification de ces voisins, un modèle linéaire approximant le classifieur autour de la série temporelle de départ est construit. Ce modèle permet de retourner une explication pour comprendre la classification de la série temporelle de départ.

Ainsi, trois classes vont donc être spécifiquement implémentées pour les séries temporelles (Figure 4) :

- *TSDomainMapper*,
- *IndexedTS*,
- *TSEExplainer*.

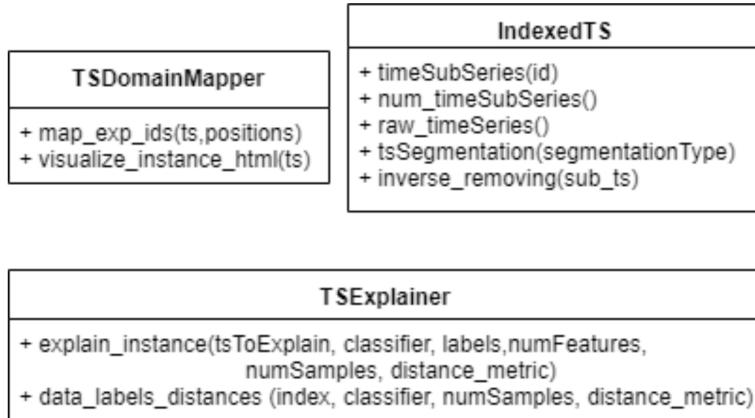


FIGURE 4 – Diagramme de classes de LIMEShape

Les classes *Lime_base* et *Explanation* déjà présentes dans le code de LIME peuvent, quant à elles, être utilisées telles quelles :

- la classe *Lime_base* contient des fonctions abstraites permettant l'apprentissage local d'un modèle linéaire construit à partir de données perturbées $(z, f(z))$ où z est un voisin obtenu par modification de l'exemple initial x . La principale fonction qui sera utilisée est :
- *explain_instance_with_data(self, neighborhood_data, neighborhood_labels, distances, label, num_features, feature_selection='auto', model_regressor=None)* qui retourne une explication à partir de données perturbées.
- la classe *Explanation* comporte différentes fonctions qui permettent de retourner les explications sous diverses formes (notebook si l'on souhaite ensuite intégrer l'explication dans un notebook, HTML si l'on veut la lire via un navigateur, etc.). Les fonctions présentes dans la classe *Explanation* sont :

- `available_labels(self)` renvoie la liste des étiquettes de classification pour lesquelles nous avons des explications,
- `as_list(self, label=1, **kwargs)` retourne l'explication sous forme d'une liste,
- `as_map(self)` renvoie une map d'explications,
- `as_pyplot_figure(self, label=1, **kwargs)` retourne l'explication sous forme d'un graphique,
- `show_in_notebook(self, labels=None, predict_proba=True, show_predicted_value=True, **kwargs)` affiche les explications HTML dans un notebook,
- `save_to_file(self, file_path, labels=None, predict_proba=True, show_predicted_value=True, **kwargs)` sauvegarde l'explication HTML dans un fichier.

2.1.1 *TSDomainMapper*

La classe *TSDomainMapper* qui sera implémentée contient les fonctions suivantes :

- `map_exp_ids(st, positions)` retourne la liste des sous-séries temporelles extraites de la ST de départ avec leurs positions (si l'on met le booléen `positions` à True) et leur poids respectif (poids normalisé calculé à l'aide de la fonction suivante). On peut ainsi connaître la contribution de chaque sous-série temporelle dans la classification de la série temporelle de départ.

$$poidsTheoriqueAttributI = \frac{poidsCalcAttributI}{\sum_{J=0}^N (poidsCalcAttributJ)} \quad (1)$$

Cette normalisation est une adaptation de Lime.

- `visualize_instance_html(st)` permet de mettre en avant les sous-séries de la ST de départ (obtenues grâce à la fonction `raw_timeSeries()`) en fonction de leur contribution à la classification de la ST (obtenue grâce à `map_exp_id()`). Plus une sous-série aura contribué de manière positive à la classification, plus elle sera mise en avant avec une couleur tirant vers le vert. Plus elle aura contribué de manière négative, plus elle aura une couleur tirant vers le rouge (Figure 12). Chaque sous-série sera également séparée par un segment vertical afin de visualiser précisément les bornes de chaque sous-série.

2.1.2 *IndexedTS*

La classe *IndexedTS* qui sera implémentée contient les fonctions suivantes :

- `timeSubSeries(id)` permet de retourner la sous-série temporelle ayant comme identifiant l'id renseigné en paramètre. Il s'agit du "getter" d'une sous-série temporelle,
- `num_timeSubSeries()` permet de connaître le nombre de sous-séries temporelles présentent dans une ST. Cette fonction va dépendre de la découpe en pré-traitement de la série temporelle. Il y a en effet deux possibilités de découpe d'une ST : une découpe à intervalle régulier, ou une découpe "intelligente" qui cherche s'il y a plusieurs motifs identiques dans la ST et, si tel est le cas, les considère comme des sous-séries temporelles et les prend en compte dans la découpe. La méthode de découpe sera désignée par l'attribut `segmentationType`,
- `raw_timeSeries()` retourne la série temporelle de départ,
- `timeSeries_position()` permet de savoir combien de fois une sous-série temporelle en particulier est présente dans la ST de départ et d'obtenir les positions de cette sous-série temporelle,

- *inverse_removing(sub_st)* prend une liste de sous-séries en paramètre et génère des voisins en remplaçant dans la série temporelle ces sous-séries par soit un segment constant, un segment nul, ou une sous-série présente dans une série tirée aléatoirement dans une base de données (Figure 5).

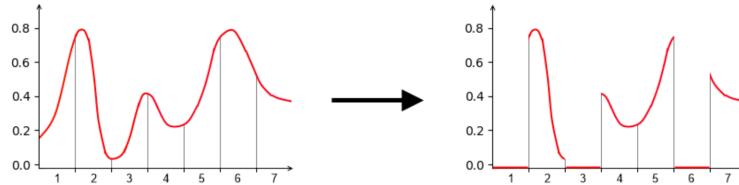


FIGURE 5 – Voisin généré avec le choix de remplacement par segments nuls

2.1.3 *TSEExplainer*

Cette classe contient la fonction *explain_instance(tsToExplain, classifier, label, numFeatures, numSamples, distance_metric)* :

- *tsToExplain* correspond à la série temporelle que l'on cherche à expliquer,
- *classifier* désigne le classifieur utilisé,
- *label* est la classe à expliquer,
- *numFeatures* correspond au nombre maximum d'attributs à utiliser dans l'explication,
- *numSamples* est la taille du voisinage à apprendre le modèle linéaire,
- *distance_metric* correspond à la distance métrique à utiliser pour la pondération de l'échantillon (par défaut la similarité cosinus).

Cette fonction va permettre de retourner une explication (*Explanation object*) détaillant comment notre série temporelle a été classée en fonction de l'étude des voisins générés avec la fonction *inverse_removing()*. La fonction *data_labels_distances(index, classifier, numSamples, distance_metric)* permet de générer aléatoirement les voisins.

Il est ensuite possible d'utiliser *as_pyplot_figure()* ou bien *show_in_notebook()* par exemple, pour traiter l'explication grâce aux méthodes présentes dans *Explanation*. Elles permettent de retourner les explications sous diverses formes exploitables.

2.2 Fonctionnement de LIMEShape

Comme dit précédemment, LIMEShape est un algorithme qui indiquera à l'utilisateur les sous-séries temporelles qui ont contribué à la classification de la série temporelle initiale. Pour arriver à un tel résultat, l'algorithme suivra des étapes spécifiques :

1. Découpage de la série temporelle en sous-séries temporelles,
2. Génération des voisins,
3. Génération de l'explication.

La Figure 14 présentée en annexe représente le diagramme de séquence résumant le fonctionnement de l'algorithme LIMEShape, ce diagramme étant détaillé par la suite.

L'utilisateur commence par faire appel à la fonction *explain_instance* en lui fournissant plusieurs paramètres tels que :

- la série-temporelle à expliquer (un tableau à 1 dimension),
- le classifieur utilisé,
- le nombre de sous-séries de la série principale à manipuler (utile pour la génération des voisins),
- le nombre de voisins à générer,
- la méthode de calcul de distance entre les voisins et la série principale.

L'algorithme va alors traiter la série temporelle fournie en fonction des différents paramètres, en suivant les 3 étapes citées ci-dessus.

2.2.1 Découpage de la série temporelle en sous-séries temporelles



FIGURE 6 – Diagramme de séquence : Découpage de la série temporelle en sous-séries temporelles

La sous-série est segmentée suivant une règle spécifiée en paramètre `segmentationType`, la liste de sous-séries est stockée dans une variable (`segmentation_list`). La partie du diagramme de séquence associée est disponible à la figure 6. Cette liste de sous-séries temporelles servira de base à la création des voisins.

2.2.2 Génération des voisins

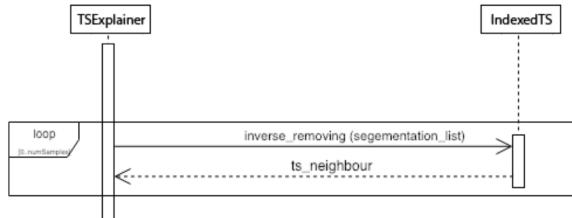


FIGURE 7 – Diagramme de séquence : Génération des voisins

En utilisant la série segmentée à l'étape précédente, `num_samples` voisins sont alors créés. Pour la création d'un voisin, un nombre aléatoire de sous-séries sont modifiées en utilisant une méthode que l'utilisateur aura préalablement choisie. (Figure 7) Cette méthode de modification peut être :

- mettre la sous-série temporelle à 0,
- garder la sous-série temporelle constante, à hauteur moyenne entre la dernière valeur de la sous-série précédente et de la première valeur de la sous-série suivante,
- sélectionner, parmi une base de sous-série temporelle, une sous-série aléatoire.

Le Figure 8 présente les différentes méthodes de modification possibles.

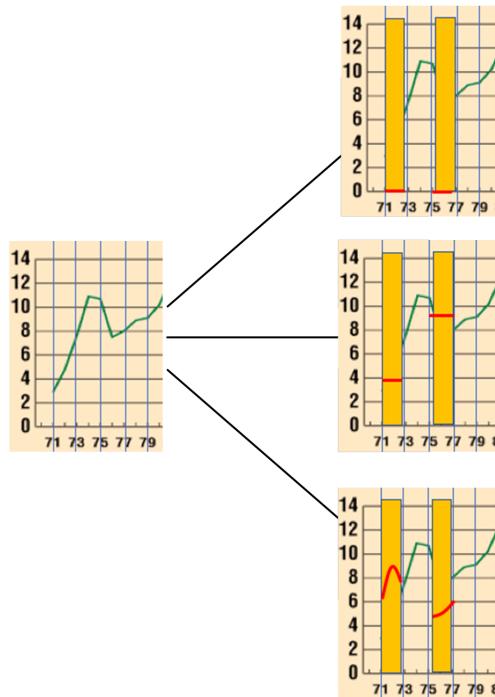


FIGURE 8 – Exemples de modification de sous-séries temporelles

2.2.3 Explication

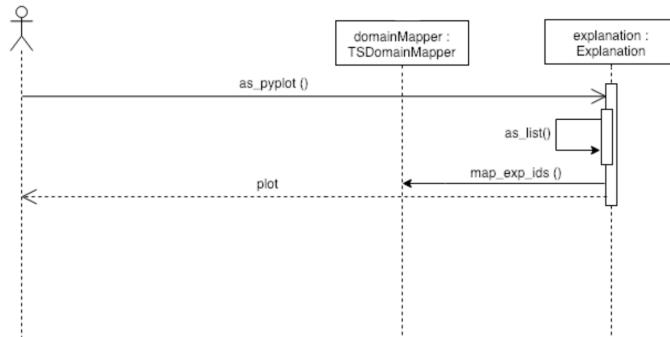


FIGURE 9 – Diagramme de séquence : Explication

Ces voisins créés sont classés. Un modèle linéaire simplifiant localement la classification autour de la série temporelle est défini. Ce modèle permet de mettre en avant les *num_features* sous-séries temporelles qui ont un impact conséquent sur la classification des voisins, et donc significatif de la série temporelle principale. L'utilisateur peut alors choisir d'afficher le résultat sous plusieurs formes possibles. Dans la Figure 9, l'utilisateur demande une visualisation via un graphique (*as_pyplot()*).

3 Interface graphique

Pour créer l'interface graphique de l'application TSEExplanation, nous allons utiliser la bibliothèque PyQt5 [10] qui permet de lier Python et Qt [12]. L'outil QtDesigner, qui est inclus lors de l'installation de PyQt5, permettra de mettre en forme l'interface. Dans un premier temps, un diagramme de classe UML permettra de visualiser l'ensemble des classes et fonctions qui concernent l'interface graphique. Ensuite les attributs et les fonctions des classes seront détaillés.

3.1 Différentes classes

L'interface graphique de l'application TSEExplanation sera composée de 2 classes. La première modélise la fenêtre principale de l'application, permettant d'accéder à nos quatre fonctionnalités. La seconde permet d'afficher l'explication de la classification d'une série temporelle, elle est appelée par la première suite à l'exécution de l'algorithme de LIMEShape. Le diagramme de classe UML de la Figure 10 illustre ces classes et leur relation.

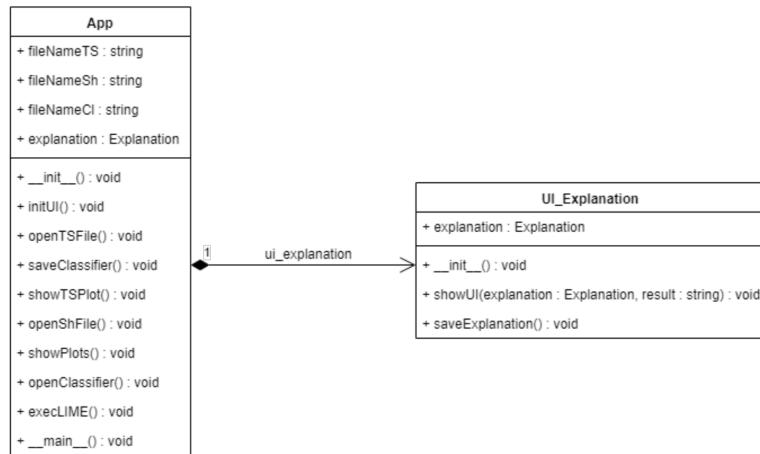


FIGURE 10 – Diagramme de classe UML de l'interface graphique

3.2 Classe App

La classe `App` correspond à la fenêtre principale de l'application TSEExplanation. Elle possède quatre attributs qui seront utilisés à travers les différentes fonctionnalités de l'application. Les attributs `fileNameTS`, `fileNameSh` et `fileNameCl` sont des string destinés à contenir les chemins des fichiers qui contiennent respectivement les séries temporelles, les shapelet et le classifieur. Ces attributs seront remplis au fur et à mesure grâce au choix de l'utilisateur. L'attribut `explanation` est un objet de type `Explanation` qui sera généré par l'algorithme de LIMEShape (partie 2 : LIMEShape), il contiendra l'explication de la classification d'une série temporelle donnée.

La fonction `__main__()` permet le lancement de l'application TSEExplanation [11]. La fonction `__init__()` permet l'initialisation de la fenêtre. Après avoir défini les éléments principaux tels que sa taille et son titre, elle appelle la fonction `initUI()` qui va créer et initialiser tous les composants de la fenêtre.

La structure principale de la fenêtre est un `QTabWidget` qui est composé de quatre onglets. Le premier onglet concerne la construction et la sauvegarde d'un classifieur. Il comporte des `QLabel` pour les zones de texte, des `QComboBox` lorsque l'utilisateur doit effectuer un choix et un `QPushButton` pour sauvegarder le classifieur. Des `QFileDialog` sont appellés lorsque l'utilisateur doit spécifier un chemin de fichier, ce qui ouvre un explorateur de fichiers. Le second onglet concerne l'affichage d'une série temporelle sous forme de

graphique. On y retrouve un *QPushButton* qui appelle un *QFileDialog*, l'utilisateur doit alors rentrer l'indice de la ST dans un *QLineEdit*. L'autre bouton permet alors l'affichage d'un graphique généré par la bibliothèque Matplotlib [9] dans un *FigureCanvas*. Le troisième onglet concerne l'affichage de la distance entre une shapelet et une ST. On y retrouve les mêmes types de composants que dans les onglets précédents. Enfin, le quatrième onglet concerne l'explication de la classification d'une série temporelle. Ici des *QRadioButton* permettront à l'utilisateur de choisir le type de distance à utiliser. La Figure 11 résume la répartition des différents composants.

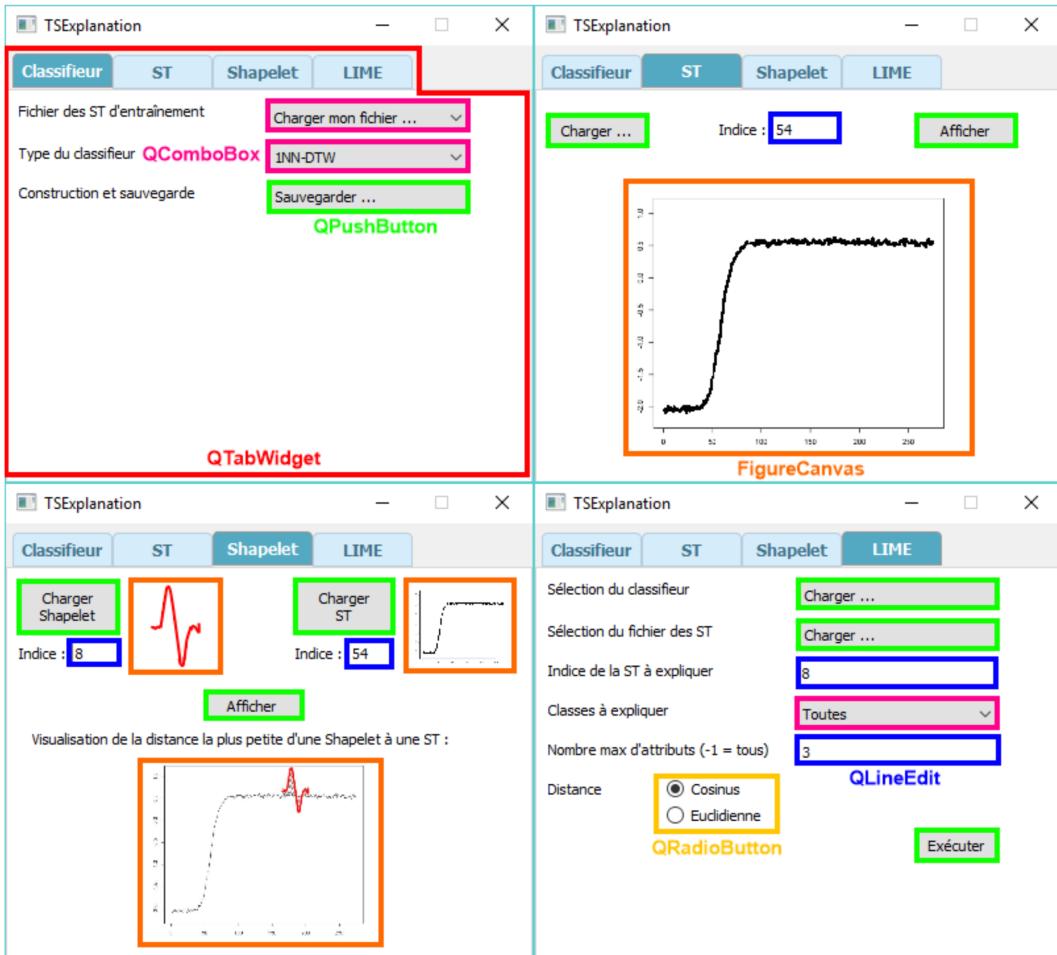


FIGURE 11 – Répartition des composants dans la fenêtre

Un certain nombre de fonctions devront également être implémentées afin de lier l'interface graphique au reste du projet :

- *openTSFile()* ouvre un explorateur de fichiers [8] qui permet à l'utilisateur de sélectionner le fichier contenant la ST qu'il veut utiliser. Le nom du fichier est alors stocké dans l'attribut *fileNameTS*. Cette fonction est utilisée dans les quatre onglets de l'application,
- *saveClassifier()* permet de construire puis de sauvegarder un classifieur lorsque l'utilisateur est dans le premier onglet, elle utilise les fonctions évoquées dans la première partie de ce rapport,
- *showTSPlot()* permet d'afficher le graphique d'une série temporelle, elle est utilisée dans le second onglet,
- *openShFile()* ouvre un explorateur de fichiers qui permet à l'utilisateur de sélectionner le fichier contenant la shapelet qu'il veut utiliser dans le second onglet de l'application. Le nom du fichier est alors stocké dans l'attribut *fileNameSh*,

- `showPlots()` permet d'afficher les trois graphiques du troisième onglet. Elle fait appelle à la fonction `showTSPlot()` pour l'affichage de la série temporelle, et elle affiche également la shapelet et la distance entre les deux,
- `openClassifier()` permet de récupérer le classifieur précédemment enregistré. Dans le dernier onglet de l'application il s'agit de la première étape afin de pouvoir appeler l'algorithme de LIMEShape,
- `execLIME()` lance l'algorithme de LIMEShape. Une fois l'explication de la classification générée ainsi que le résultat de la classification récupéré, une instance de la classe `UI_Explanation` est créée et la fenêtre affichant l'explication s'ouvre.

3.3 Classe `UI_Explanation`

La classe `UI_Explanation` correspond à la fenêtre affichant l'explication de la classification de la série temporelle. Elle est appelée par la fenêtre principale à la fin de l'exécution de l'algorithme de LIMEShape. Elle est composée d'un attribut de type `Explanation` et de deux fonctions.

La fonction `__init__()` permet l'initialisation de la fenêtre ainsi que de ses composants. Dans cette interface il y a un `QLabel` qui permettra l'affichage du résultat de la classification, et un autre pour l'affichage des poids. Au centre il y aura un `FigureCanvas` qui contiendra le graphique de l'explication. Enfin, un `QPushButton` permettra la sauvegarde de l'explication. La Figure 12 résume la répartition des différents composants.

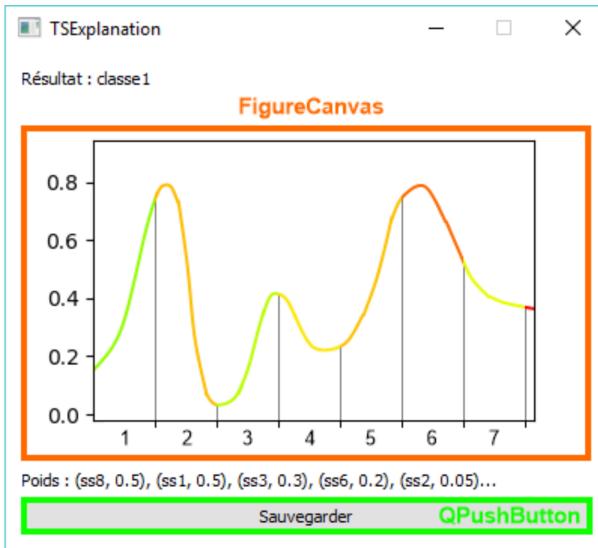


FIGURE 12 – Répartition des composants dans la fenêtre

La fonction `showUI(explanation, result)` récupère les informations stockées dans l'objet de type `Explanation` passé en paramètre avant d'afficher la fenêtre. Le paramètre `result` contient le résultat de la classification, qui sera affiché en haut de la fenêtre. L'attribut `explanation` sera initialisé avec l'objet passé en paramètre puis, afin de générer les différents éléments à afficher, plusieurs fonctions de cet attribut seront appelées. La fonction `as_pyplot_figure()` permettra de récupérer le graphique, les couples (sous-série, poids) seront obtenus avec `as_map()`.

La fonction `saveExplanation()` ouvre un explorateur de fichiers pour que l'utilisateur spécifie le chemin où il veut sauvegarder le fichier html de l'explication générée. Un fois le chemin obtenu, c'est la fonction `save_to_file()` de l'attribut `explanation` qui sera appelée.

La Figure 13 permet de récapituler l'interaction des différents modules :

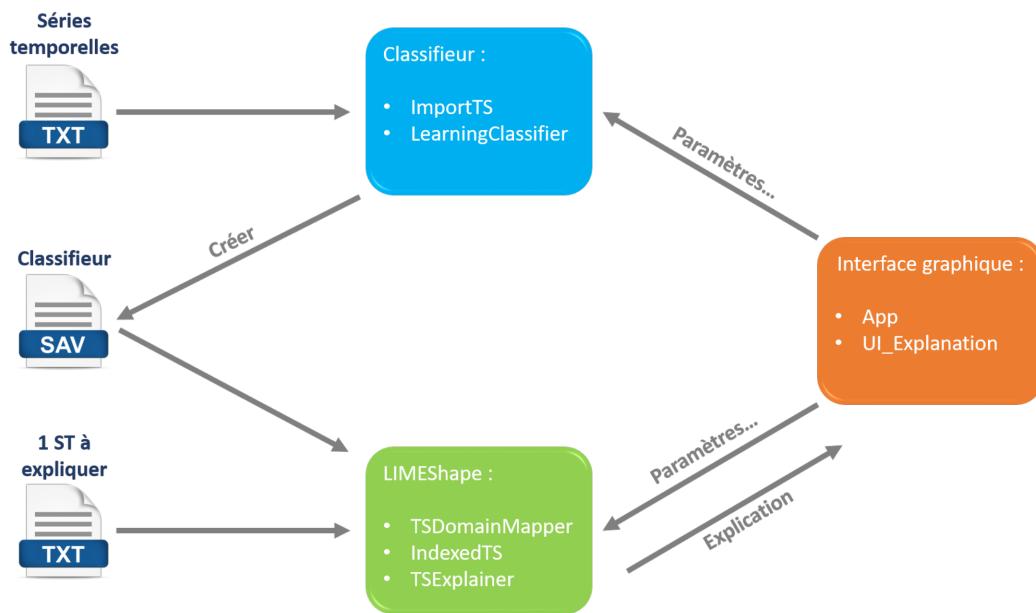


FIGURE 13 – Liaisons entre les différents modules de TSExplanation

Conclusion

Après avoir compris les notions indispensables à la réalisation de ce projet lors de la phase de pré-étude, et après avoir étudié l'aspect fonctionnel de l'outil à réaliser au cours de la phase de spécification, la conception de l'outil TSExplanation a pu être enclenchée.

Trois points principaux avaient été abordés lors de la phase de spécification : l'apprentissage d'un classifieur à partir de ST, la construction d'une explication pour un tel classifieur et enfin l'implémentation d'une interface graphique pour l'outil final. Ces trois points majeurs ont été analysés sur le plan technique.

Dans un premier temps, le code à générer pour implémenter l'apprentissage et la sauvegarde d'un classifieur de ST a été présenté en détail, au travers des deux classes ImportTS et LearningClassifier. Ces deux classes et leurs méthodes permettront donc de construire un classifieur de ST qui fournira la classification que l'on cherche à expliquer.

Dans un second temps, le code Python des modules *lime_text*, *lime_image* et *lime.explanation*, qui avait été étudié lors de la phase de spécification, a permis de dégager les classes qui allaient devoir être créées (*TSDomainMapper*, *IndexedTS* et *TSEExplainer*). Les trois classes ainsi créées seront donc dédiées au traitement des ST, au même titre que les modules *lime_text* et *lime_image* pour le traitement de textes et d'images.

Enfin, la création d'une interface graphique a été initiée. Les outils PyQt et QtDesigner avaient été étudiés lors de la phase de spécification, ce qui a permis de se pencher plus en détail sur l'interface en elle-même et de concevoir une maquette proche de ce qui va être réalisé. Les classes et fonctions à implémenter ont été intégrées à la conception de cette interface graphique et modélisées dans des diagrammes UML, afin d'avoir une vision globale du fonctionnement de l'interface.

A l'issue de cette phase de conception, les différents éléments de l'outil TSEExplanation, leur fonctionnement et leurs méthodes, ont été définis et agencés entre eux. En effet, les fonctionnalités de l'outil ont été clairement définies. Les différentes classes incluant les méthodes nécessaires à l'implémentation et au bon fonctionnement de l'algorithme LIMEShape ont été modélisées. Enfin, la réalisation, avec la bibliothèque PyQt5, des différents composants de l'interface graphique a été enclenchée.

- [1] Anthony BAGNALL et al. "The Great Time Series Classification Bake Off : A Review and Experimental Evaluation of Recently Proposed Algorithms". In : (2016).
- [2] *Documentation de LIME*. <https://lime-ml.readthedocs.io/en/latest/lime.html>.
- [3] *Documentation TSLearn*. <https://tslearn.readthedocs.io/en/latest/>.
- [4] *GitHub de LIME*. <https://github.com/marcotcr/lime>.
- [5] Josif GRABOCKA et al. "Learning Time-series Shapelets". In : *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA : ACM, 2014, p. 392–401. ISBN : 978-1-4503-2956-9. DOI : 10.1145/2623330.2623613. URL : <http://doi.acm.org/10.1145/2623330.2623613>.
- [6] Jason LINES et al. "A Shapelet Transform for Time Series Classification". In : *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '12. Beijing, China : ACM, 2012, p. 289–297. ISBN : 978-1-4503-1462-6. DOI : 10.1145/2339530.2339579. URL : <http://doi.acm.org/10.1145/2339530.2339579>.
- [7] Scott M LUNDBERG et Su-In LEE. "A Unified Approach to Interpreting Model Predictions". In : *Advances in Neural Information Processing Systems 30*. Sous la dir. d'I. GUYON et al. Curran Associates, Inc., 2017, p. 4765–4774. URL : <http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf>.
- [8] *PyQt5 file dialog*. <https://pythonspot.com/pyqt5-file-dialog/>.
- [9] *PyQt5 Matplotlib*. <https://pythonspot.com/pyqt5-matplotlib/>.
- [10] *PyQt5 Reference Guide*. <http://pyqt.sourceforge.net/Docs/PyQt5/index.html>.
- [11] *Python PyQt5 Tutorial – Example and Applications*. <https://data-flair.training/blogs/python-pyqt5-tutorial/>.
- [12] *QT Documentation*. <http://doc.qt.io/qt-5/index.html>.
- [13] Marco Tulio RIBEIRO, Sameer SINGH et Carlos GUESTRIN. ""Why Should I Trust You ?" : Explaining the Predictions of Any Classifier". In : *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA : ACM, 2016, p. 1135–1144. ISBN : 978-1-4503-4232-2. DOI : 10.1145/2939672.2939778. URL : <http://doi.acm.org/10.1145/2939672.2939778>.

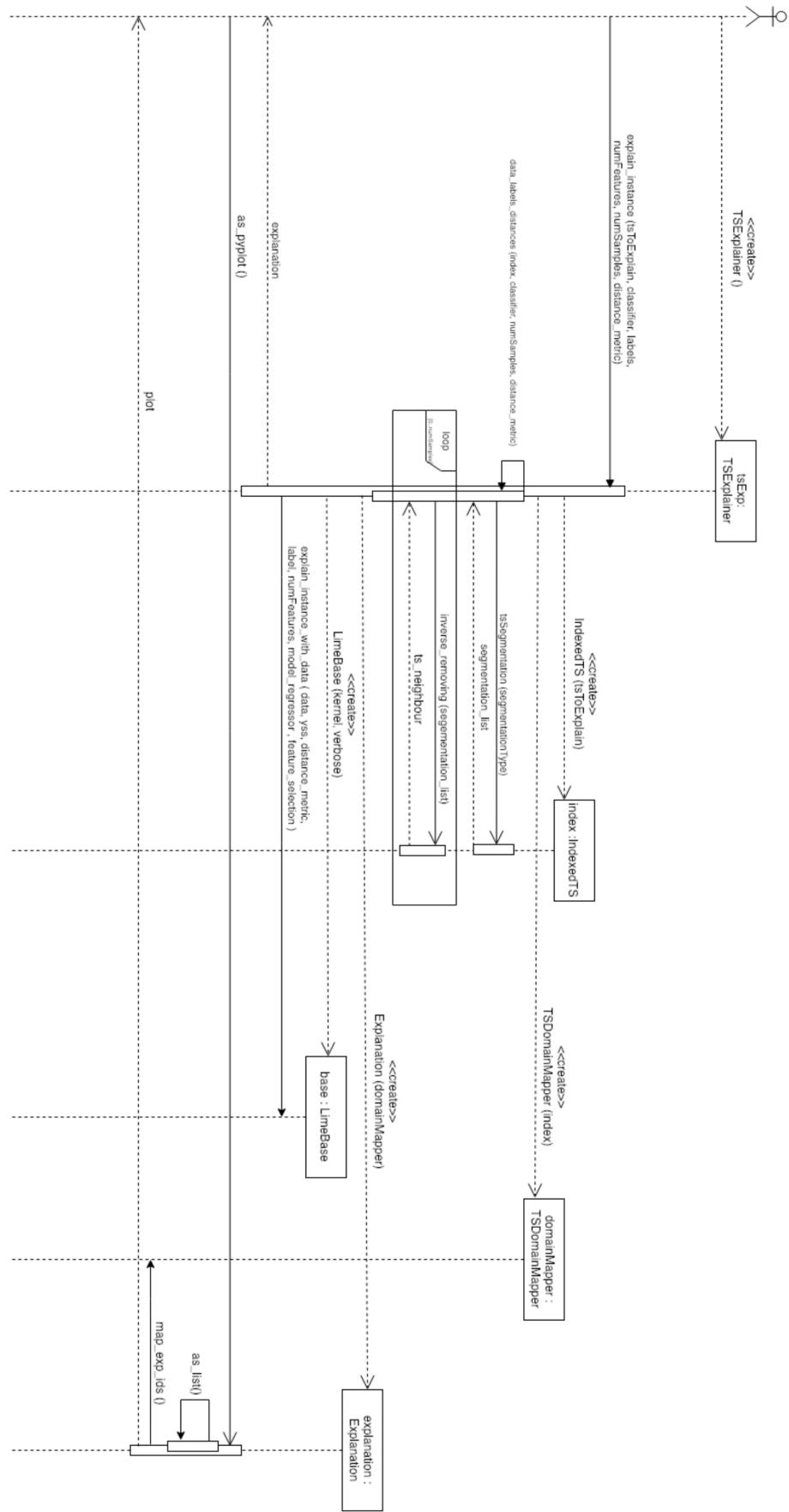


FIGURE 14 – Diagramme de séquence