Institut National des Sciences Appliquées
de Rennes

TSExplanation
Rapport de Spécification

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

Introduction

La phase de pré-étude, première phase du projet, 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). Ces notions étant éclaircies, il a 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 (ST) et pouvoir expliquer la classe attribuée à une ST.

Une première structure de ce projet a été dégagée (cf. Figure 2). 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 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.

Pour pouvoir poursuivre la lecture de ce rapport, il est nécessaire de souligner le changement de nom du projet. En effet, dans le rapport de pré-étude, ce dernier est présenté sous le nom de "STLime". Ce nom présentait certains inconvénients notamment la présence du mot "Lime" qui renvoie trop directement à l'algorithme LIME et porte donc à confusion quant au lien entre LIME et ce projet. Cet ancien nom était également trop proche de celui de l'algorithme LIMEShape (une nouvelle fois par la présence du mot "Lime"), ce qui pouvait amener l'utilisateur à confondre les deux.
Pour que ce projet se distingue d'avantage de l'algorithme LIME (qui n'est finalement qu'un outil au sein du projet), le nom "TSExplanation" a été choisi. Ce nouveau nom illustre mieux les objectifs de ce projet.

Dans ce rapport de spécification seront décrites les spécifications fonctionnelles du projet en détail, ainsi qu'une première idée de son architecture logicielle générale. Une première partie va concerner l'apprentissage boîte noire, à travers les classifieurs 1NN et Learning sous-séries temporelles. Dans un second temps, plusieurs modules de LIME permettant d'interpréter des classifieurs de séries temporelles seront présentés. Enfin, une première version de l'interface graphique de TSExplanation sera proposée.

La figure ci-dessous rappelle le déroulement du logiciel TSExplanation

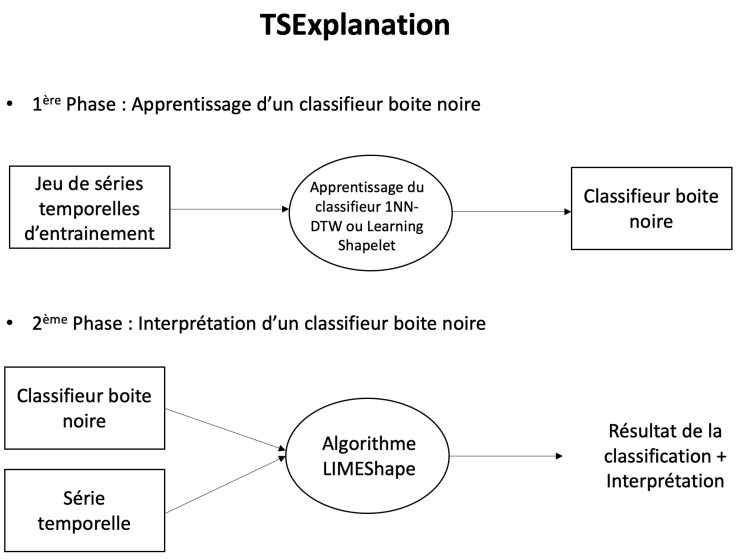

FIGURE 1 – Illustration du fonctionnement de TSExplanation

1 Apprentissage Boîte noire

La première partie de ce projet consiste à entraîner un classifieur sur des séries temporelles. Dans le cadre de notre logiciel, l'utilisateur aura le choix entre deux classifieurs : 1NN-DTW et Learning Shapelet. Le premier a été expliqué lors du précèdent rapport.

Concernant le deuxième, nous avons décidé de passer de Tranform Shapelet couplé à Random Forest (mentionné dans le rapport de pré-étude) à Learning Shapelet. En effet, Transform Shapelet est beaucoup plus intéressant d'un point de vue interprétabilité car il s'appuie sur des sous-séries temporelles de la base de données sans les transformer. Alors que Transform Shapelet est plus efficace mais modifie les sous-séries temporelles qui ne sont plus des sous-parties des exemples de départ et ne sont donc plus potentiellement interprétables par un expert du domaine. Finalement, comme l'algorithme ici à vocation d'être un BW, il n'est pas nécessaire d'avoir cet aspect d'interprétabilité, l'efficacité est beaucoup plus importante.

Dans cette partie du logiciel TSExplanation, l'utilisateur pourra alors entraîner l'un de ces deux classifieurs avec les séries temporelles de son choix et pourra l'enregistrer afin de l'utiliser pour la seconde partie, à savoir l'interprétabilité avec LIMEShape.

Cette partie du logiciel utilise de nombreuses fonctions de la bibliothèque Tslearn. Cette bibliothèque fournit des outils de machine-learning pour l'analyse de séries temporelles.

1.1 Importation des séries temporelles

Pour importer des séries temporelles afin d'entraîner le classifieur, l'utilisateur aura le choix entre l'importation par fichier texte ou l'importation via la base de séries temporelles du site UEA/UCR.

1.1.1 Par fichier

La première possibilité est d'importer des séries temporelles via un fichier texte. L'utilisateur rentrera en entrée le fichier texte contenant les séries temporelles. Le logiciel le transformera en séries temporelles utilisables pour entraîner le classifieur. Le fichier texte devra suivre un format bien précis. Chaque ligne représente une série temporelle. Dans chaque ligne, on délimite avec un espace les observations et on utilise un point pour représenter les décimales. Pour cette importation, on pourra utiliser le module nommé "utils" de la bibliothèque Tslearn.

Dans ce module, la fonction load_timeseries_txt() permettra de transformer le texte contenant les séries temporelles en un format utilisable par Tslearn pour entraîner le classifieur.

Le format de la série sera alors le suivant : un tableau à 2 dimensions contenant les valeurs des séries temporelles que l'on va appeler X_train. Par exemple X_train[5] donne 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, il faut en plus un tableau à une dimension qui contient les classes auquel appartient les séries, on va l'appeler Y_train. Par exemple si la 5ème série est de la classe 1, Y_train[5] sera égal à 1.

1.1.2 Via la base UCR/UCE de séries temporelles

Si l'utilisateur ne possède pas de séries temporelles, il peut en choisir une dans la base du site The UEA & UCR Time Series Classification Repository (Anthony Bagnall, Jason Lines, William Vickers and Eamonn Keogh, disponible à l'adresse : www.timeseriesclassification.com). Ce site propose de nombreux ensembles de séries temporelles (TwoPatterns, TwoLeadECG).

Le module data_set de Tslearn permet d'avoir la liste des séries temporelles disponibles, et permet de les télécharger et de les importer directement. On utilisera ce module dans notre logiciel. L'utilisateur n'aura qu'à entrer le nom de la série temporelle et le processus d'importation se fera automatiquement dans le bon format.

```
from tslearn.datasets import UCR_UEA_datasets
X_train, y_train, X_test, y_test =
UCR_UEA_datasets().load_dataset("TwoPatterns")
```

Dans le code ci-dessus, la fonction remplit directement les tableaux X_train, y_train, X_test, y_test comme dans l'importation par fichier. Les tableaux X_test et y_test sont du même format que les tableaux X_train et Y_train. Ces séries temporelles permettront de tester les classifieurs et permettront de voir leur précision.

Une fois les séries temporelles importées, l'utilisateur aura le choix d'entraîner deux classifieurs sur ces séries. Il s'agit de 1NN-DTW et de Learning Shapelet.

1.2 1NN-DTW

Le premier classifieur disponible pour l'utilisateur est le classifieur 1NN-DTW.

1.2.1 Apprentissage

Pour entrainer un classifieur K-NN, il faut tout d'abord importer les bibliothèques nécessaires. Pour ce faire, on importe KNeighborsTimeSeriesClassifier, ainsi que UCR_UEA_datasets afin d'avoir les jeux de données d'entraînement.

```
from tslearn.neighbors import KNeighborsTimeSeriesClassifier
from tslearn.datasets import UCR_UEA_datasets
X_train, y_train, X_test, y_test =
UCR_UEA_datasets().load_dataset("TwoPatterns")
```

Il faut ensuite créer le classifieur. Dans notre cas, on mettra le nombre de voisin à un pour réaliser un 1NN et on utilisera la metric DTW.

On peut ensuite l'entraîner à l'aide d'un jeu de données en lui donnant des séries temporelles ainsi que leurs étiquettes.

```
knn1_clf = KNeighborsTimeSeriesClassifier(n_neighbors=1, metric="dtw")
knn1_clf.fit(X_train, y_train)
```

1.2.2 Sauvegarde

Une fois notre classifieur entraîné, on peut le sauvegarder en utilisant la sérialisation. Il existe deux outils permettant de réaliser cette action : pickle qui est un outil natif de python, ainsi que joblib, un outil de Sklearn. Dans les deux cas, la démarche pour sauvegarder est relativement similaire, à ceci près que pickle prend un fichier binaire ouvert en argument contrairement à joblib qui prend juste un nom de fichier.

```
#version utilisant pickle
import pickle
pickle.dump(knn1_clf,open('pickle.sav','wb'))
#la fonction open de python prend le nom du fichier
#son mode d'ouverture (ici w pour write)
#et le type de fichier, ici b pour binary.
-------------------------------------------------
#version utilisant joblib
from sklearn.externals import joblib
joblib.dump(knn1_clf,'joblib.sav')
```

Il est à noter que l'extension du fichier est sans importance sur le processus. La différence entre pickle et joblib réside dans l'optimisation de l'utilisation de données NumPy, ce qui est très utile pour des algorithmes qui ont besoin de stocker tout le jeu de données comme c'est le cas ici : pour réaliser un 1NN, on doit stocker toutes les séries temporelles et leurs étiquettes.

Pour pouvoir réutiliser le classifieur par la suite, il suffira de créer un objet classifieur en utilisant les méthodes load des modules.

```
classifier = pickle.load(open('pickle.sav','rb'))
-------------------------------------------------
classifier = joblib.load('joblib.save')
```

1.3 Learning Shapelet

Le second classifieur disponible pour notre logiciel est le classifieur Learning Shapelet.

1.3.1 Apprentissage

Comme avec 1NN-DTW, Tslearn propose d'entraîner ce classifieur avec des séries temporelles. Cela se fait en trois étapes. La première est de créer un dictionnaire permettant de savoir pour chaque taille de sous-séries temporelles (clé), le nombre de sous-séries temporelles équivalent à générer. La seconde étape prend ce dictionnaire et construit le modèle du classifieur. On peut lui passer de nombreux paramètres afin de le spécifier. La dernière étape consiste à entraîner le classifieur avec les séries temporelles d'entraînement. Le code ci-dessous montre l'utilisation des méthode de Tslearn afin d'y parvenir.

```
from tslearn.shapelets import ShapeletModel,
grabocka_params_to_shapelet_size_dict
from keras.optimizers import Adagrad

-----------------------------------------------

shapelet_sizes = grabocka_params_to_shapelet_size_dict(
                            n_ts=X_train.shape[0],
                            ts_sz=X_train.shape[1],
                            n_classes=len(set(y_train)),
                            l=0.1,
                            r=2)
"""
n_ts is the number of timeseries in the dataset
ts_sz the length of these timeseries
n_class is the number of different classes
l is the fraction of the total length of a time serie used to
create base shapelet
r is the number of different length of shapelet to use
"""
-----------------------------------------------

shp_clf = ShapeletModel(n_shapelets_per_size=shapelet_sizes,
                    optimizer=Adagrad(lr=.1),
                    weight_regularizer=.01,
                    max_iter=50,
                    verbose_level=0)
"""
shapelets_per_size (dict)
max_iter (int (default: 1000)) - Number of training epochs.
batch_size (int (default:256)) - Batch size to be used.
verbose_level ({0, 1, 2} (default: 2)) - keras verbose level.
optimizer - keras optimizer to use for training.
weight_regularizer - keras regularizer to use for training the
classification (softmax) layer.
"""
-----------------------------------------------
shp_clf.fit(X_train, y_train)
```

1.3.2 Sauvegarde

Pour sauvegarder ce classifieur, pickle ou joblib ne sont pas appropriés : les méthodes dump ne fonctionnent pas car elles n'ont pas été prévues pour ce type d'objet. Pour l'instant, il est possible de sauvegarder les poids du modèle que nous avons créé.

```
shp_clf.model.save_weights('weights.h5')
```

Cependant, cette méthode n'est actuellement pas satisfaisante car nous ne pouvons pas charger le classifieur à partir de ces données seules, il faut d'abord le ré-entraîner. L'importation des poids permet ensuite d'être sûr d'obtenir exactement le même classifieur mais n'est pas adéquat dans ce projet puisque nous voulons sauvegarder le classifieur afin de ne pas avoir à le ré-entraîner.

```
shp_clf1 = ShapeletModel(
                        shapelets_per_size=shapelet_sizes,
                        optimizer=Adagrad(lr=.3),
                        weight_regularizer=.002,
                        max_iter=2,
                        verbose_level=0)
shp_clf1.fit(X_train, y_train)
shp_clf1.model.load_weights('weights.h5')
```

Une méthode de sauvegarde est actuellement en développement par quelqu'un d'extérieur à TsLearn. Il a ajouté deux fonctions au classifieur : save et load_model. Seulement, ces fonctions ne sont actuellement pas encore utilisables aussi simplement que les fonctions de joblib ou pickle. En effet, elles ne sauvegardent et ne chargent que les modèles keras, qui permettent de donner les probabilités d'appartenance à une classe, mais ne contient pas les labels associés aux classes, il faut donc aussi sauvegarder/charger le "label_binarizer" pour que les prédictions du modèle donnent la bonne étiquette, pour cela, on peut utiliser pickle. On utilisera la fonction inverse_transform du label_binarizer sur le résultat de la prédiction afin d'obtenir la classification correspondante aux probabilités obtenues.

```
""" Pour la sauvegarde """
shp_clf.save("model.sav")
pickle.dump(shp_clf.label_binarizer, open('label.sav','wb'))
-------------------------
""" Pour le chargement """
shp_test = shapelets.load_model("model.sav")
y_probs = shp_test.predict(X_test)
label_b = pickle.load(open('label.sav','rb'))
y_pred = label_b.inverse_transform(y_probs)
```

Une fois que l'utilisateur a entraîné et sauvegardé un classifieur, il pourra demander à connaître l'interprétabilité du classifieur grâce à la deuxième partie de notre logiciel : LIMEShape.

2 LIME

Le projet TSExplanation adapte un algorithme d'interprétation de classification (LIME) pour des exemples de types de séries temporelles, et donc pour des classifieurs de séries temporelles. Pour combiner l'efficacité et l'interprétabilité, l'objectif est d'expliquer les boîtes noires en concevant une couche d'interprétation entre le classifieur et l'utilisateur humain. Pour faire de l'interprétabilité sur les classifieurs boîtes noires, différentes techniques existent. Pour le projet TSExplanation, une méthode d'explication locale sera utilisée : LIME (Local Intrepretable Model-agnostic Explanation). LIMEShape, l'algorithme LIME qui va être adapté aux séries temporelles, indiquera les sous-séries temporelles présentes dans la série temporelle qui ont le plus contribué à la classification de la ST. Pour expliquer ce qui va être fait, il est nécessaire de comprendre le code de LIME, c'est pourquoi dans cette section le code de LIME est détaillé.

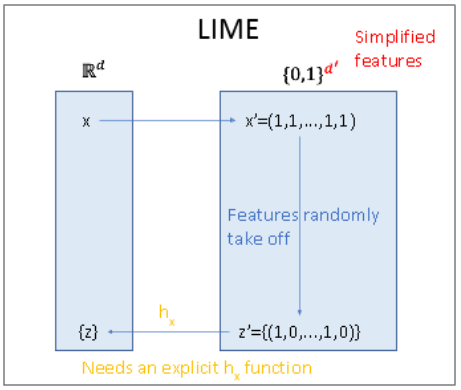

FIGURE 2 – Illustration du fonctionnement de LIME

2.1 Explication du coeur du code

Afin de bien comprendre le coeur du code de LIME, il faut s'intéresser tout d'abord à deux modules : *lime_base* et *explanation*. Ces classes sont générales et permettent d'obtenir une explication générale. Les modules qui seront expliqués par la suite s'appuient sur ces classes pour spécifier certains domaines.

2.1.1 lime_base

La classe *lime_base* contient des fonctions abstraites pour l'apprentissage local d'un modèle linéaire construit à partir de données perturbées *(z,f(z))* où z est un voisin obtenu par perturbation de l'exemple initial x. Cette classe possède un attribut interne *kernel_fn* ainsi que 4 fonctions :

- *kernel_fn* correspond à la fonction transformant un tableau de distances en un tableau de valeurs de proximité.
- *explain_instance_with_data(self, neighborhood_data, neighborhood_labels, distances, label, num_features, feature_selection'auto', model_regressorNone)* retourne une explication à partir des données perturbées.
  - *neighborhood_data* : tableau 2D des données perturbées.
  - *neighborhood_labels* : classes correspondantes des données perturbées.
  - *distances* : distances entres les voisins et l'exemple original.
  - *label* : classe à expliquer.
  - *num_features* : nombre maximum d'attributs à utiliser dans l'explication.
  - *feature_selection* : méthodes à utiliser pour la sélection d'attributs.
  - *model_regressor* : régression linéaire à utiliser. Par défaut utilise '*Ridge* fourni par la bibliothèque *Sklearn*.

- *generate_lars_path(weighted_data, weighted_labels)* permet d'appliquer l'algorithme LARS (algorithme de sélection d'attributs d'un modèle linéaire) sur les différents attributs dans le cas où la méthode de sélection utilisée est *lasso*.
  - *weighted_data* : correspond aux données transformées par *kernel_fn*.
  - *weighted_labels* : correspond aux classes transformées par *kernel_fn*.
- *forward_selection(self, data, labels, weights, num_features)* permet d'appliquer la méthode de sélection ascendante d'attributs (*forward_selection*).
  - *data* : correspond à *neighborhood_data* dans *explain_instance_with_data*.
  - *labels* : correspond à *neighborhood_labels* dans *explain_instance_with_data*.
  - *weights* : correspond à *distances* dans *explain_instance_with_data* une fois transformées par *kernel_fn*.
  - *num_features* : correspond à *num_features* dans *explain_instance_with_data*.
- *feature_selection(self, data, labels, weights, num_features, method)* permet de choisir et d'appliquer l'une des différentes méthodes de sélection d'attributs du modèle disponibles (*forward_selection*, *highest_weights*, *lasso*).
  - *data* : correspond à *neighborhood_data* dans *explain_instance_with_data*.
  - *labels* : correspond à *neighborhood_labels* dans *explain_instance_with_data*.
  - *weights* : correspond à *distances* dans *explain_instance_with_data* une fois transformées par *kernel_fn*.
  - *num_features* : correspond à *num_features* dans *explain_instance_with_data*.
  - *method* : méthode de sélection d'attributs à utiliser.

L'adaptation du code de LIME pour le domaine des séries temporelles va pouvoir s'appuyer sur le module *lime_base* qui est le coeur du code de LIME. Il sera possible d'implémenter les fonctions abstraites nécessaires.

2.1.2 explanation

Le module *explanation* est une classe d'explication avec des fonctions de visualisation. Il comporte 2 classes : *DomainMapper* et *Explanation*.

class DomainMapper(object)

La classe *DomainMapper(object)* est utile pour mapper les explications à un domaine spécifique. Une explication est, quelque soit le domaine d'application, un ensemble de couples (numéro d'attribut, poids). Retourner une telle explication à un utilisateur ne serait pas utile et serait incompréhensible, l'utilisateur n'ayant strictement aucune idée de ce qu'est l'attribut numéro i. Si on prend l'exemple de textes, le DomainMapper transforme l'explication en un ensemble de couples (mot, poids) indiquant exactement le mot et son poids, chose totalement compréhensible par l'utilisateur. De même pour les images, ce sera un ensemble de couples (image, poids) où l'image présentée mettra en avant les superpixels dont on mesure le poids.

L'idée est donc d'avoir une classe DomainMapper pour chaque domaine (texte, tableaux, images, etc.), afin d'avoir une classe d'explication générale et des classes séparées pour les spécificités et la visualisation de chaque domaine.

Ainsi, les fonctions présentes dans *DomainMapper* existent aussi dans le module *lime_text* en étant plus spécifiques et adaptées. Les fonctions de *DomainMapper* sont les suivantes :

- *map_exp_ids(self, exp, **kwargs)* permet de mapper les identifiants d'entités en noms concrets, exp correspondant à la liste de couples (id, poids) et **kwargs à des paramètres supplémentaires optionnels.

— *visualize_instance_html(self, exp, label, div_name, exp_object_name, **kwargs)* produit du code JavaScript pour la visualisation de l'instance à mettre en avant, en se basant sur la liste de couples (id, poids) (textitexp) et d'autres options optionnels (textit**kwargs). Le paramètre textitlabel correspond à l'instance à visualiser, textitdiv_name et textitexp_object_name sont des options servant à la génération du code JavaScript.

De nombreux nouveaux domaines possèdent une classe qui hérite de DomainMapper. Ils peuvent alors implémenter et adapter ces 2 fonctions.

class Explanation(object)

La classe *Explanation(object)* est également une des classes sur lesquelles nous allons nous appuyer pour implémenter nos fonctions de traitement de séries temporelles. Elle est la seconde classe présente dans le module *explanation*. Elle 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.).

— *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.

Maintenant que les différentes fonctions de *lime_base* et *explanation* ont été expliquées, il est possible de s'intéresser aux modules *lime_text* et *lime_image*. Ces deux classes sont plus spécifiques aux domaines texte et image et permettent de s'adapter aux spécificités et à la visualisation de leur domaine.

2.2 Explication du module lime_text

Ce module de Lime permet de travailler sur des textes, fournis sous forme de chaînes de caractères. Il permet d'analyser, de traiter, de visualiser des chaînes de caractères de plusieurs manières différentes, grâce à trois classes et pas moins de huit fonctions détaillées dans la suite de ce rapport.

2.2.1 class TextDomainMapper(explanation.DomainMapper)

Cette classe permet de construire un DomainMapper spécifique aux chaînes de caractères à partir de l'explication que l'on entre en paramètre. Cette classe contient deux fonctions :

— *map_exp_ids(self, exp, positions=False)* permet d'obtenir une liste de couples (mot_positions, poids) si position=true, ou (mot, poids) si position=false. En effet, le paramètre *exp* est une explication constituée d'ensembles de couples (identifiantMot,poids) que l'on souhaite étudier, et *position* indique si l'on souhaite retourner la position des mots dans la chaîne de caractères que l'on étudie.
— *visualize_instance_html(self, exp, lab, div_name, exp_obj_name, txt=True, opacity=True)* permet la visualisation de notre chaîne de caractères avec certains mots choisis mis en avant pour la visualisation.

Exemple de retour de *map_exp_ids(self, exp, positions=True)* : cf. Figure 3.

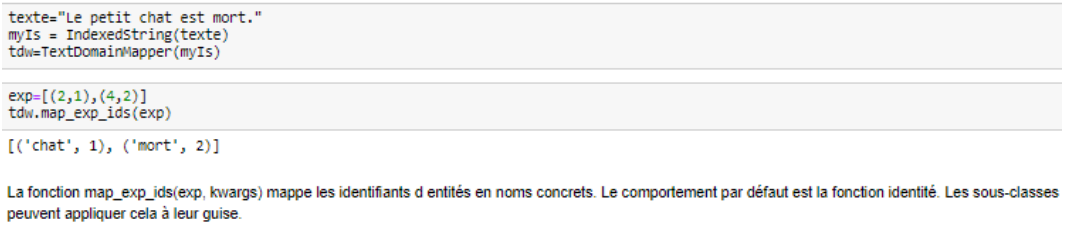

FIGURE 3 – Exemple d'utilisation de TextDomainMapper

2.2.2 *IndexedString et IndexedCharacters*

class IndexedString(object)

La classe *IndexedString(object)* permet d'analyser des chaînes de caractères de différentes manières. Cette classe contient les fonctions suivantes :

- *raw_string(self)* permet de récupérer la chaîne de caractères originale brute. Il est nécessaire de sauvegarder l'état initial de la chaîne de caractères. Cela est particulièrement utile pour que l'on puisse comparer la chaîne originale avec ses voisins générés à l'aide de la fonction *inverse_removing()* (expliquée ci-dessous).
- *num_words(self)* retourne le nombre de mots présents dans le document.
- *word(self, id)* permet d'obtenir le mot de notre chaîne de caractères se trouvant à la position *id* renseignée en paramètre.
- *string_position(self, id)* retourne une array numpy d'indices correspondant aux positions du premier caractère de chaque occurrence du mot à la position *id*.
- *inverse_removing(self, words_to_remove)* retourne une string correspondant à la sous-chaîne de notre chaîne de caractères de départ, à laquelle ont été supprimé les mots entrés en paramètre de la fonction sous forme de liste d'ids.

Ci-dessous, un exemple d'utilisation de *IndexedString* : cf. Figure 4 et Figure 5.

Création d'un IndexedString

```
texte="Le petit chat est mort. Il était vraiment super audacieux. La mort était inévitable."
print(texte)
```

```
Le petit chat est mort. Il était vraiment super audacieux. La mort était inévitable.
```

```
myIs=IndexedString(texte)
```

raw_string contient la classe originale

```
print(myIs.raw_string())
```

```
Le petit chat est mort. Il était vraiment super audacieux. La mort était inévitable.
```

num_words

permet de compter le nombre de mots différents du texte

```
print(myIs.num_words())
```

```
12
```

FIGURE 4 – Exemple d'utilisation de IndexedString

word

permet de retrouver un mot à partir de son indice

```
print(myIs.word(0))
print(myIs.word(1))
print(myIs.word(2))
print(myIs.word(9))
```

```
Le
petit
chat
audacieux
```

string_position(indice)

Permet de retourner toutes les positions du mot étant à l'indice fourni en paramètre
Attention les positions retournée sont en nombre de caractères.
Par exemple le mot "mort" d'indice 4 dans la liste des mots est aux positions 18 et 62 en terme de nombre de caractères

```
print(myIs.string_position(0))
print(myIs.string_position(4))
```

```
[0]
[18 62]
```

inverse_removing

Permet de retirer les mots dont les indices sont fournis dans la liste en paramètre

```
print(myIs.inverse_removing([6,8]))
```

```
Le petit chat est mort. Il  vraiment  audacieux. La mort  inévitable.
```

FIGURE 5 – Exemple d'utilisation de IndexedString (suite)

class IndexedCharacters(object)

La classe *IndexedCharacters(object)*, quant à elle, permet d'analyser des caractères et non pas des chaînes de caractères en tant que telle. Ci-dessous, un exemple d'utilisation de *IndexedCharacters* : cf. Figure 6 et Figure 7.

```
texte="Le petit chat court. Il aime le chien. Le chien aboie."
myIc=IndexedCharacters(texte)
```

num_words

retourne le nombre de caractères différents utilisés dans le texte.

```
print(myIc.raw_string())
print(myIc.num_words())
```

```
Le petit chat court. Il aime le chien. Le chien aboie.
18
```

word

retourne le caractere correspondant à l'indice

```
res=""
for i in range(18):
    res+=myIc.word(i)
print(res)
```

```
Le ptichaour.Ilmnb
```

FIGURE 6 – Exemple d'utilisation de IndexedCharacters

string_position

Retrouve toutes les positions d'un charactere

```
print(myIc.word(0))
print(myIc.string_position(0))
print(myIc.string_position(1))
```

```
L
[ 0 39]
[ 1  4 27 30 35 40 45 52]
```

```
print(myIc.inverse_removing([1]))
```

```
L ptit chat court. Il aim l chin. L chin aboi.
```

FIGURE 7 – Exemple d'utilisation de IndexedCharacters (suite)

2.2.3 *class LimeTextExplainer(object)*

Cette classe permet d'expliquer les classifieurs de texte, une explication étant un modèle linéaire qui peut être utilisé localement pour approximer notre classifieur boite noire. Le but est de calculer la distance de notre mot avec ses voisins et de retenir les voisins les plus proches. Cette classe ne contient qu'une seule fonction : *explain_instance(self, text_instance, classifier_fn, labels=(1,), top_labels=None, num_features=10, num_samples=5000, distance_metric='cosine', model_regressor=None)*. Elle génère des explications pour une prédiction. En effet, une fois des données voisines de notre chaîne originale générées, un apprentissage est effectué. Cette fonction permet donc de mieux comprendre les prédictions de classe faites par un classifieur.

2.3 Explication du module *lime_image*

Comme son nom l'indique, le module *lime_image* permet de travailler sur des données au format image, et d'ainsi générer une explication visuelle grâce à ses 2 classes.

2.3.1 *ImageExplanation*

En plus de la fonction d'initialisation, cette classe possède une fonction : *get_image_and_mask(self, label, positive_only=True, hide_rest=False, num_features=5, min_weight=0)*.

Cette fonction renvoie un résultat à deux composantes. La première correspond à l'image de départ et la seconde correspond à un masque qui pourra être utilisé plus tard par une fonction de skimage. L'image est représentée par un tableau numpy à 3 dimensions : 2 dimensions représentant la grille de pixels avec une dimension supplémentaire correspondant à un tableau de 3 éléments (rouge, vert, bleu). Le masque est lui un tableau numpy à 2 dimensions.
Ce masque dépend des paramètres de la fonction :

- *label* : classe à expliquer.
- *positive_only (boolean)* : si il est à True, ne seront pris en compte uniquement les superpixels contribuant positivement à la prédiction de ce label en particulier. Sinon, c'est le paramètre *num_features* qui déterminera le nombre de superpixels à utiliser.
- *hide_rest (boolean)* : si il est à True, les superpixels ne contribuant pas à l'explication seront grisés.
- *num_features (entier)* : nombre de superpixels à inclure dans l'explication
- *min_weight (réel)* : poids minimum pour considérer un superpixel comme étant "utile" à l'explication

2.3.2 *LimeImageExplainer*

Cette classe sert à expliquer la prédiction de la classification d'une image à l'aide de ses deux fonctions :

- *explain_instance(self, ...)* génère le voisinage en modifiant aléatoirement l'image et permet l'apprentissage local d'un modèle linéaire permettant d'expliquer chacune des classifications. Pour cela, plusieurs paramètres peuvent être transmis à la fonction :
  - image : image (tableau numpy 3D)
  - *classifier_fn* : fonction permettant de classifier une image et retournant la probalité d'appartenir à chaque classe.
  - labels : les classes à expliquer
  - *hide_color* : si None, le superpixel à cacher sera de la couleur la plus présente dans celui-ci. Sinon il sera de la couleur indiquée
  - *top_labels (entier)* : si ce n'est pas None, la fonction ignore le paramètre 'labels' et produit une explication pour les k labels ayant la plus grande probablité de prédiction (k est ici le paramètre)
  - *num_features (entier)* : nombre maximum d'attributs présents dans l'explication
  - *num_samples (entier)* : nombre de voisins à générer
  - *segmention_fn* : algorithme de segmentation permettant de découpter l'image en superpixels (par défaut *quickshift*)
  - *distance_metric* : méthode de mesure de distance utilisée pour les poids
  - *model_regressor* : regressor de Sklearn à utiliser pour l'explication (le regressor par défaut est Ridge regression)
  - *random_seed (entier)* : entier utilisé pour la segmentation (si None, un entier aléatoire entre 0 et 1000 sera utilisé)

2.3.3 Exemple d'utilisation

Comme pour toute utilisation de LIME, il faut commencer par entraîner un algorithme de classification, ce qui aura pour effet de générer un classifieur qu'on va interpréter. Dans notre cas, ce sera un classifieur d'image. Ce classifieur doit pouvoir prendre en entrée un tableau *numpy* et doit rendre en sortie les probabilités des prédictions. Appelons le *predict_fn* dans notre exemple. Une fois notre classifieur prêt, on commence par initialiser *LimeImageExplainer*, la classe qui se chargera de construire une explication :

```
explainer = lime_image.LimeImageExplainer()
```

L'étape d'après est la plus importante, c'est elle qui permet d'exécuter l'algorithme de LIME grâce à la fonction *explain_instance*. On passe à cette dernière tous les paramètres dont elle a besoin (supposons dans notre cas que *image* représente l'image à expliquer (cf. Figure 8) sous forme de tableau numpy 3D) :

```
explanation = explainer.explain_instance(image, predict_fn,
            top_labels=5, hide_color=0, num_samples=1000)
```

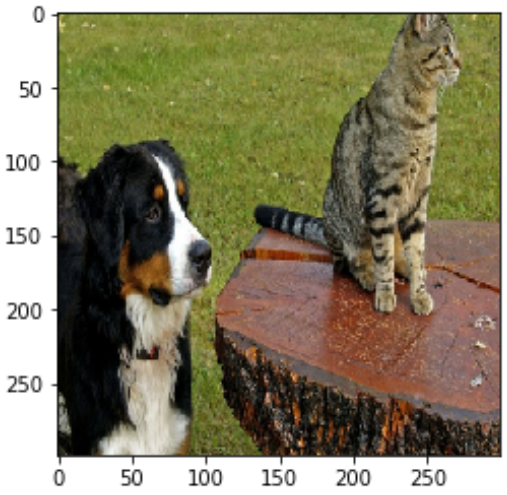

FIGURE 8 – Image à interpréter

Cette fonction commence par segmenter l'image en la découpant en superpixels en utilisant un algorithme de segmentation (par défaut *quickshift*). Ensuite une image de d'arrière-plan est générée en fonction du paramètre *hide_color*. Toutes ces données sont passées après à la fonction *data_labels* qui se charge de générer les images voisines ainsi que leurs prédictions sous forme de couple *(data, labels)*, les superpixels "désactivés" de ces images étant remplacés par l'image d'arrière-plan. Ensuite, toutes ces informations sont passées à la fonction *explain_instance_with_data* qui se charge de trouver le modèle linéaire qui minimise complexité et précision. Ses valeurs de retour sont utilisées pour construire un instance de la classe *ImageExplanation* qui permet de visualiser une explication. Une fois cette explication construite, elle est retournée à l'utilisateur. Ce dernier peut ensuite la visualiser en utilisant la fonction *get_image_and_mask* qui prend en entrée différents paramètres d'affichage. Dans notre cas, les parties de l'image ayant contribué positivement à la classification sont affichés en vert, tandis que ceux qui ont contribué négativement sont de couleur rouge :

```
temp, mask = explanation.get_image_and_mask(240, positive_only=False,
                num_features=10, hide_rest=False)
```

Le couple *(temp, mask)* permet de visualiser l'image en utilisant par exemple la bibliothèque *Matplotlib* :

```
plt.imshow(mark_boundaries(temp / 2 + 0.5, mask))
```

Le résultat final est interprétable par un humain (cf. Figure 9).

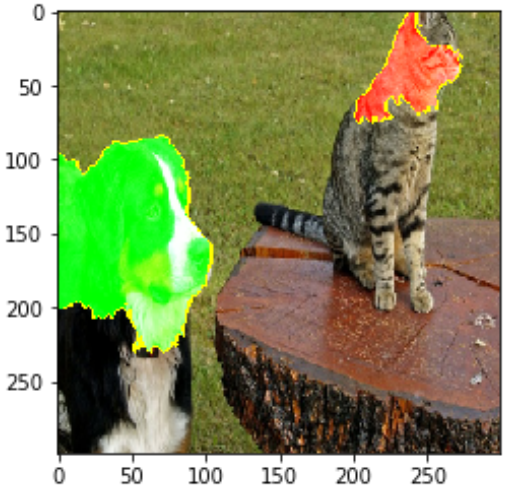

FIGURE 9 – Image une fois interprétée

2.4 Adaptation des classes aux séries temporelles

Après avoir détaillé les modules *lime_text* et *lime_image* qui s'adaptent à des domaines spécifiques, 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 tout un cheminement. Il faudra dans un premier temps découper la série temporelle en segments de sous-séries temporelles. Puis les voisins seront générés en supprimant des sous séries-temporelles. Grâce à la classification de ces voisins, un modèle linéaire approximant le classifieur autour de la ST de départ sera construit. Ce modèle permet de retourner une explication pour comprendre la classification de la ST de départ.

2.4.1 *TimeSeriesDomainMapper*

En reprenant la même structure que la classe *TextDomainMapper*, nous allons implémenter notre classe *TimeSeriesDomainMapper* qui contiendra les fonctions suivantes :

— *map_exp_ids(st,positions)* retournera la liste des sous-séries temporelles dans la ST 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 pourra ainsi connaître la contribution de chaque sous-série temporelle dans la classification de la ST de départ.

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

Cette normalisation est une adaptation de Lime.

— *visualize_instance_html(st)* permettra de mettre en avant les sous-séries de la ST de départ (obtenue 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 mis 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. Chaque sous-série sera également séparé par un segment vertical afin de bien visualiser les bornes de chaque sous-série (cf. Figure 20 à la fin de la partie 3 concernant l'interface graphique).

2.4.2 *IndexedTimeSeries*

IndexedString et *IndexedCharacters* permettent de manipuler les chaînes de caractères de manière complète, c'est pourquoi, nous allons créer notre propre classe *IndexedTimeSeries* afin de pouvoir manipuler nos séries temporelles de manière aussi large que les chaînes de caractères. Pour cela, nous implémenterons ces fonctions :

- *timeSubSeries(id)* permettra 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()* permettra d'avoir le nombre de sous-séries temporelles dans une ST. Cette fonction va dépendre de la découpe en pré-traitement de la ST. Il y aura en effet deux possibilités de découpe d'une ST : une découpe à intervalle régulier, ou une découpe "intelligente" qui recherchera s'il y a plusieurs motifs identiques dans la ST et si c'est le cas, elle les découpera de la même manière.
- *raw_timeSeries()* retournera notre ST de départ.
- *timeSeries_position* permettra de savoir combien de fois une certaine sous-série est présente dans notre ST et d'obtenir les positions de cette dernière.
- *inverse_removing(sub_st)* prendra une liste de sous-séries en paramètre et générera des voisins en remplaçant dans la ST ces sous-séries par soit un segment constant, un segment nul, ou une sous-série présente dans une base de données (cf. Figure 10).

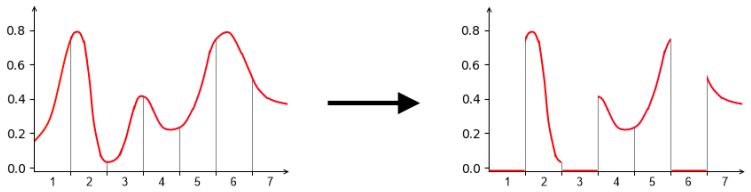

FIGURE 10 – Voisin généré avec le choix de remplacement par segments constants

2.4.3 *TimeSeriesExplainer*

Cette classe ne contiendra que la fonction *explain_instance()*. Cette fonction va permettre de retourner une explication (Explanation object) détaillant comment on a classé notre ST en fonction de l'étude des voisins générés de base avec *inverse_removing()*. On va ensuite devoir traiter l'explication avec les fonctions de Explanation pour pouvoir retourner les explications sous diverses formes exploitables. Il sera pour cela possible d'utiliser *as_pyplot_figure()* ou bien *show_in_notebook()* par exemple.

3 Interface graphique

Pour créer l'interface graphique de l'application TSExplanation, plusieurs possibilités ont été envisagées. Dans le précédent rapport, les bibliothèques PySide et Tkinter avaient été évoquées car elles étaient très répandues. Cependant en poursuivant les recherches, la bibliothèque PyQt a semblé être un outil très performant et facile d'utilisation. PySide et PyQt permettent toutes deux de lier Python et Qt, leur différence majeure concerne leur licence : PyQt doit être utilisée dans des projets open source contrairement à PySide. Le choix final s'est pourtant porté sur PyQt5, la dernière version de PyQt, car cette bibliothèque permet d'utiliser QtDesigner, un utilitaire graphique de création d'interfaces qui permet de construire les différentes fenêtres graphiquement puis de générer le code Python permettant l'affichage des fenêtres. De plus PyQt possède d'avantage de widgets que Tkinter, la documentation est également plus importante.

Dans un premier temps, un diagramme de cas d'utilisation permettra d'illustrer les différentes fonctions de l'application. Ensuite les deux fonctions primaires seront présentées, puis nos deux principales fonctions seront détaillées. Enfin, leur utilisation par ligne de commande sera expliquée.

3.1 Cas d'utilisation

L'application TSExplanation regroupe 4 principales fonctionnalités. La première concerne l'entraînement et la sauvegarde d'un classifieur, la seconde permet d'afficher une série temporelle. La troisième fonctionnalité est l'affichage de la distance entre une série temporelle et une sous-série temporelle donnée. Enfin il est possible d'afficher l'explication de la classification d'une ST. Le diagramme UML de cas d'utilisation ci-dessous illustre ces 4 possibilités (cf. Figure 11).

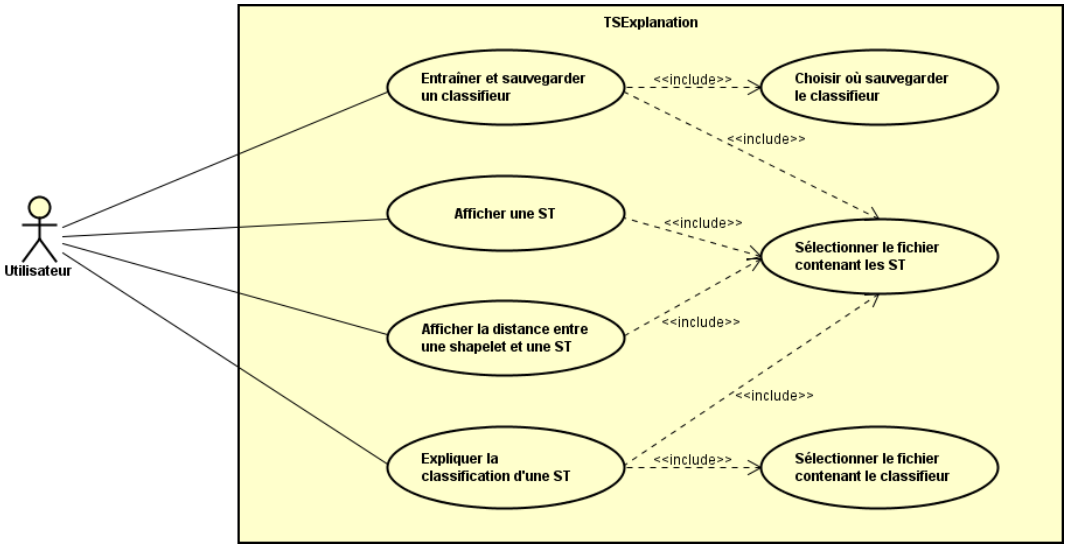

FIGURE 11 – Diagramme UML de cas d'utilisation

3.2 Fonctions primaires

L'onglet ST de l'application a pour fonction l'affichage d'une série temporelle contenue dans un fichier TXT (cf. Figure 12). Le bouton "Charger ..." permet d'ouvrir un explorateur de fichiers. Un fois le fichier de l'ensemble des ST sélectionné, il faut renseigner l'indice de la ST voulue puis le bouton "Afficher" fait apparaître le graphique associé.

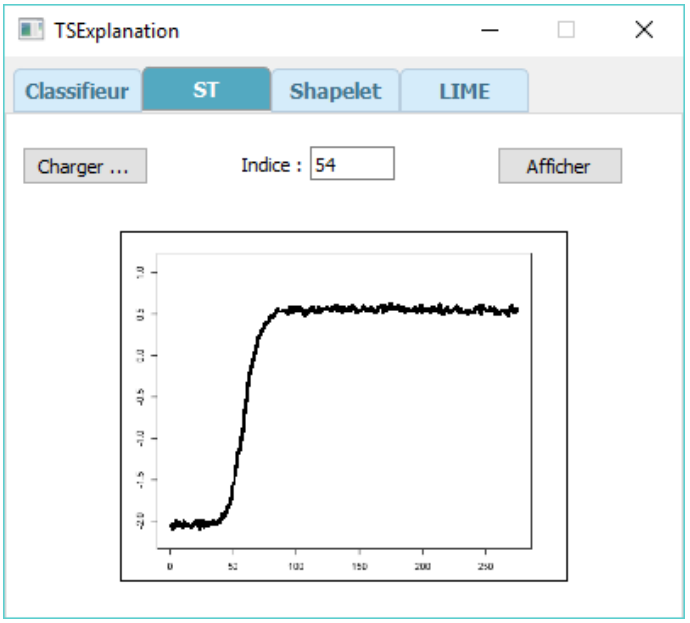

FIGURE 12 – Maquette - Onglet ST

L'onglet shapelet propose à l'utilisateur de choisir une shapelet et une série temporelle afin d'afficher graphiquement la distance entre les deux. (cf. Figure 13). Le bouton "Charger Shapelet" permet d'ouvrir un explorateur de fichiers puis de sélectionner le fichier TXT qui contient un ensemble de shapelets. Un fois le fichier sélectionné, il faut renseigner l'indice de la shapelet voulue. Il faut reproduire la même opération pour le choix de la série temporelle. Enfin, le bouton "Afficher" fait apparaître le graphique associé.

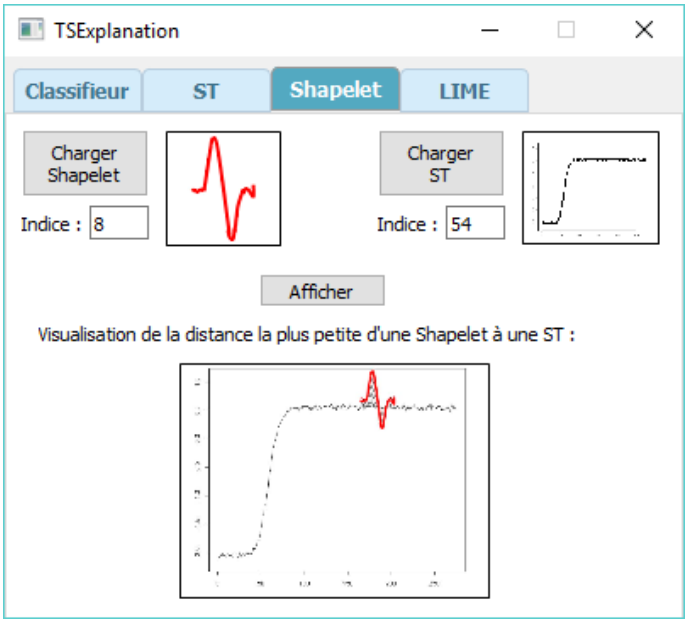

FIGURE 13 – Maquette - Onglet shapelet

3.3 Entraînement et sauvegarde d'un classifieur

A travers l'onglet "Classifieur" de l'application TSExplanation, l'utilisateur va pouvoir construire et sauvegarder un classifieur, qu'il pourra ensuite utiliser dans l'onglet LIME s'il le désire. L'onglet "Classifieur" (cf. Figure 14) propose des paramètres simples pour choisir le type de classifieur à construire.

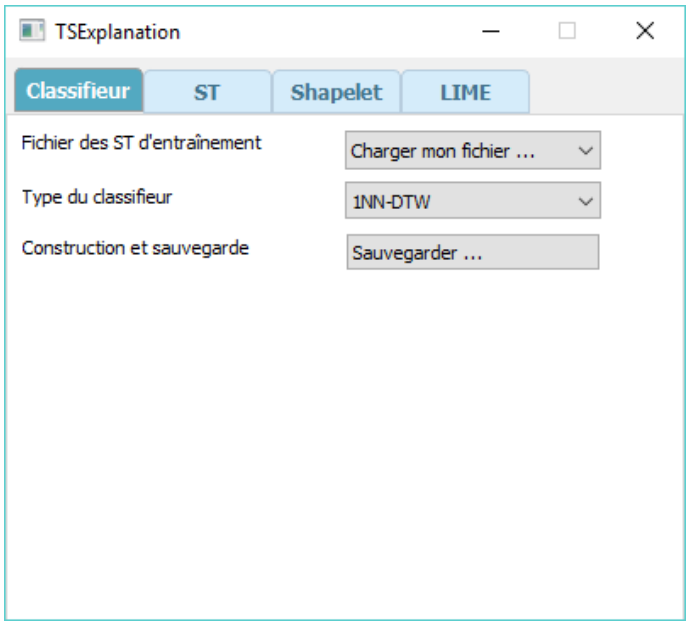

FIGURE 14 – Maquette - Onglet classifieur

La première étape consiste à choisir le fichier contenant les exemples de ST qui permettront l'apprentissage du classifieur. L'utilisateur a le choix de sélectionner dans la liste déroulante un fichier de la base de ST proposée (provenant du site TimeSeriesClassification.com) ou d'utiliser son propre fichier. Dans ce cas cela ouvrira un explorateur de fichiers où il suffira de naviguer et de sélectionner le fichier voulu, qui devra être sous le format TXT (cf. Figure 15).

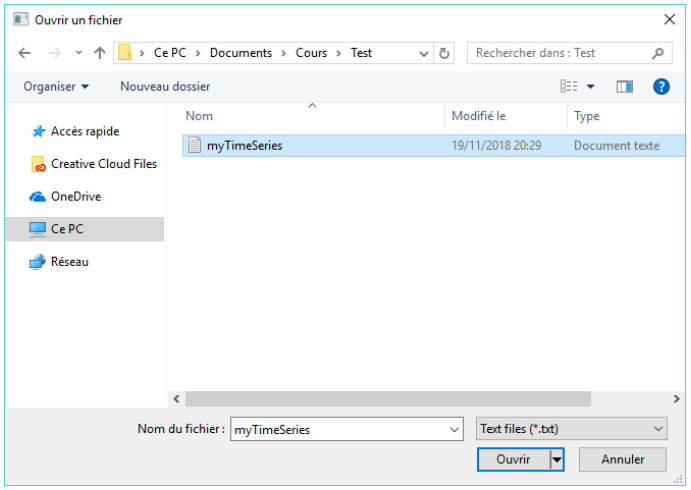

FIGURE 15 – Maquette - Choix du fichier des exemples

L'étape suivante est la sélection du type de classifieur : 1NN-DTW ou Learning Shapelet. Pour finir, il reste l'étape de l'entraînement et la sauvegarde de ce classifieur. Le bouton "Sauvegarder..." permet d'ouvrir un explorateur de fichiers, l'utilisateur peut alors choisir le chemin et le nom de sauvegarde du classifieur, le fichier aura l'extension *.sav* (cf. Figure 16).

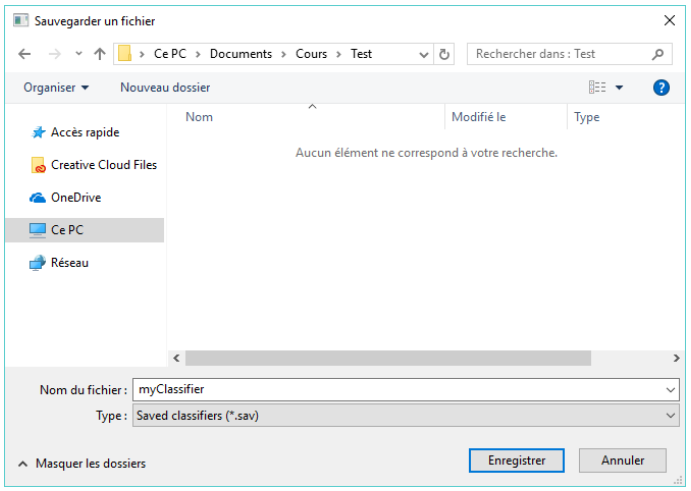

FIGURE 16 – Maquette - Sauvegarde du classifieur

3.4 Explication de la classification d'une série temporelle

L'onglet "LIME" représente la dernière partie de l'application TSExplanation, il contient la fonction principale de notre outil : l'interprétabilité de séries temporelles. A travers de nombreux paramètres, l'utilisateur peut obtenir l'explication de la classification d'une ST donnée (cf. Figure 17).

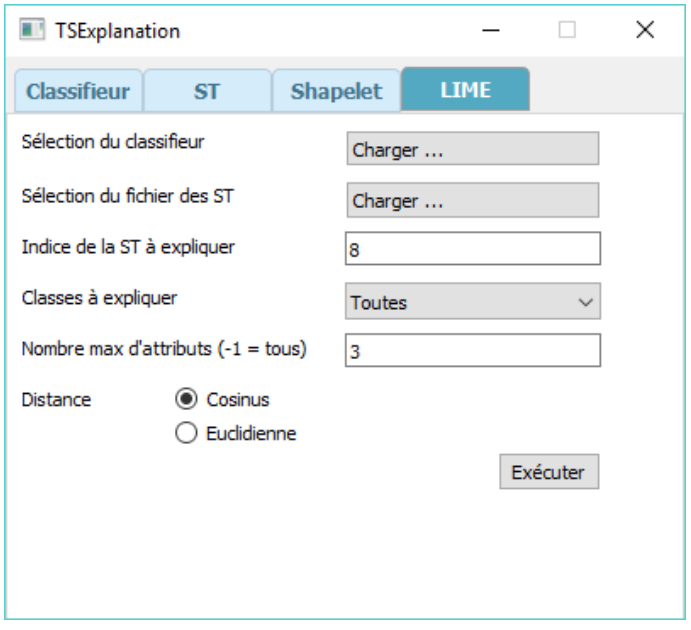

FIGURE 17 – Maquette - Onglet LIME

Dans un premier temps, l'utilisateur doit sélectionner le classifieur à utiliser. Le bouton associé ouvre un explorateur de fichiers, il faut choisir le fichier avec l'extension *.sav* contenant le classifieur voulu (cf. Figure 18).

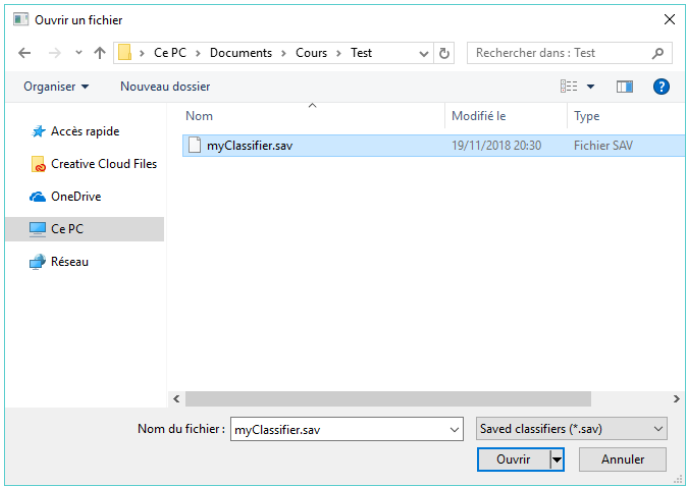

FIGURE 18 – Maquette - Sélection du classifieur

L'étape suivante est la sélection du fichier contenant les séries temporelles. Une nouvelle fois, le bouton associé ouvre un explorateur de fichiers qui permet la sélection du fichier TXT souhaité (cf. Figure 19).

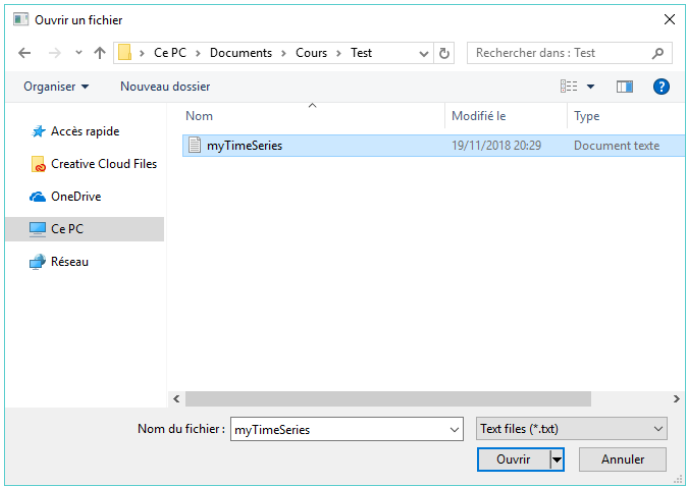

FIGURE 19 – Maquette - Sélection des ST

L'utilisateur doit ensuite spécifier l'indice de la ST qu'il veut traiter dans le fichier TXT, ainsi que la ou les classes à expliquer. Il faut aussi indiquer le nombre maximum d'attributs et la distance à utiliser. Enfin, le bouton "Exécuter" permet d'afficher l'explication correspondante calculée avec l'algorithme LIMEShape (cf. Figure 20). Le graphique sera créé par la classe *TimeSeriesDomainMapper* (cf.partie 2.4.1 - TimeSeriesDomainMapper). Les différentes sous-séries temporelles qui composent la ST seront colorées d'après leur poids.

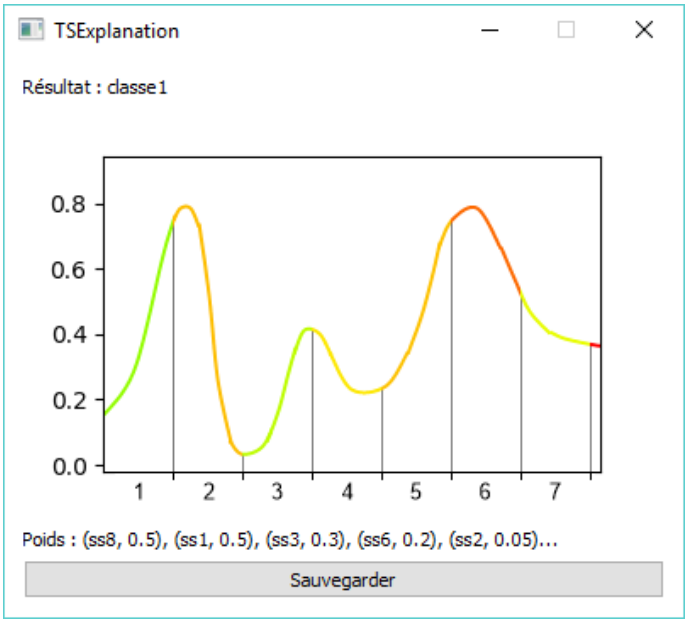

FIGURE 20 – Maquette - Affichage de l'explication

3.5 Exécution par ligne de commande

En plus de notre application TSExplanation, il nous a été demandé de pouvoir exécuter certaines de nos fonctionnalités via des lignes de commandes dans un terminal. En effet, dans le cadre de tests effectués sur de grandes quantités de données, il est préférable de pouvoir utiliser les fonctions dans des scripts. Ainsi, deux de nos fonctionnalités initiales seront disponibles sous cette forme :

— entraînement et sauvegarde d'un classifieur,
— explication de la classification d'une ST.

Python est un langage qui permet le lancement de programmes en ligne de commande avec des arguments séparés par des espaces pour spécifier diverses options. Les arguments de la ligne de commande sont stockés dans la liste *sys.argv*, il est donc simple de les récupérer dans le programme.

3.5.1 Entraînement et sauvegarde d'un classifieur

La ligne de commande permettant l'entraînement et la sauvegarde d'un classifieur permettra d'enregistrer le classifieur dans un fichier avec l'extension *.sav*. Elle sera de la forme suivante :

python SaveClassifier.py [tsfile] [classifier] [classifierfile]

Toutes les options devront être renseignées, un fichier README sera alors créé afin d'expliquer précisément la signification de chacune des options :

— tsfile : chemin du fichier TXT contenant les ST pour l'apprentissage
— classifier : type du classifieur (learning_shapelet ou 1nn_dtw)
— classifierfile : chemin du fichier avec l'extension *.sav* où sera sauvegardé le classifieur

3.5.2 Explication de la classification d'une série temporelle

La ligne de commande permettant l'explication de la classification d'une ST permettra d'enregistrer un fichier HTML qui illustrera l'explication. Elle sera de la forme suivante :

python TSExplanation.py [classifierfile] [tsfile] [tsindex] [nbclasses] [class1] ... [classn] [nbmaxattr] [distance] [explanationfile]

Toutes les options devront être renseignées sauf la liste des classes dans le cas où nbclasses = -1, dans ce cas toutes les classes seront expliquées. Un fichier README sera créé afin d'expliquer précisément la signification de chacune :

- classifierfile : chemin du fichier avec l'extension *.sav* contenant le classifieur
- tsfile : chemin du fichier TXT contenant les ST
- tsindex : indice de la ST à traiter (entier)
- nbclasses : nombre de classes à expliquer (entier), si la valeur est -1 cela signifie que toutes les classes sont à expliquer donc il ne faut pas les détailler
- class1 ... classn : détail des classes à expliquer (autant que nbclasses)
- nbmaxattr : nombre maximum d'attributs (entier)
- distance : distance à utiliser (cosinus ou euclidienne)
- explanationfile : chemin du fichier HTML où sera sauvegardée l'explication

Conclusion

Après avoir compris les notions indispensables à la réalisation de ce projet lors de la phase de pré-étude, il a été possible de se pencher sur l'aspect fonctionnel de l'outil à réaliser.

Trois points principaux ont été abordés lors de cette 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.

Dans un premier temps, l'importation de ST depuis différentes sources et l'apprentissage de classifieurs à partir de ces ST ont été spécifiés. Il est également nécessaire de pouvoir enregistrer le classifieur ainsi appris si l'on veut être en mesure de le ré-utiliser pour générer une explication.

Dans un second temps, le code Python de l'algorithme LIME a été étudié en détail, notamment au niveau des modules lime_text, lime_image et lime.explanation. Le but de cette étude a été de comprendre les méthodes de ces modules afin de définir lesquelles pourraient être adaptées au ST pour l'implémentation de LIMEShape. Il sera nécessaire de coder un certain nombre de méthodes "à la main" pour adapter LIME aux ST, toutes les méthodes des modules ne pouvant pas être adaptées. Les classes "DomainMapper" et "Explanation" semblent particulièrement intéressantes pour afficher l'explication d'un classifieur de ST. Les modules lime_text et lime_explanation vont, quand à eux, servir d'appui et d'exemples pour implémenter un module spécifique au domaine des ST.

Enfin, la création d'une interface graphique a été initiée. Les outils utilisés pour implémenter cette interface sont PyQt5 et QtDesigner. La conception de cette interface graphique permet, en plus de fournir une interface à ce projet, d'éclaircir certains points quant à l'utilisation de l'outil final. Lors de cette étape, il a fallu prendre en main PyQt5 et son outil QtDesigner et commencer à coder la gestion d'événements en Python (ouverture d'une série temporelle pour la visualiser en cliquant sur un bouton, etc).

A l'issue de cette phase de spécification, l'architecture du projet commence à faire surface. En effet, le fonctionnement et les composants clés de l'algorithme LIME deviennent plus clairs, ce qui est indispensable pour adapter LIME aux ST. De plus, le rendu final du projet, c'est-à-dire l'outil qui sera développé, commence à prendre forme, notamment au travers de l'interface graphique dont la conception a d'ors et déjà été initiée. De plus, la réalisation de ces trois tâches a mené à la prise en main de nombreuses bibliothèques Python et à l'étude d'un code existant : LIME.

La prochaine phase du projet est la phase de planification. Il faudra se pencher sur les aspects techniques et les différentes étapes de la conception du projet, définir le travail de développement à réaliser et planifier l'enchaînement des tâches qui mènera à la réalisation finale de l'outil TSExplanation.