1 Differences between original proposal and ACTS

We can summarize the differences between this implementation of ACTS and the one proposed in the paper as:

- Loop, scope of the code
- Automatic addition of certain instances to the labelled data is not present here
- Pattern creation and assigning

1.1 Loop, scope of the code

For this first aspect, we would like to remark that this implementation of ACTS is built to be used as a query strategy inside va builder.

The code in the paper describes a full active learning loop. Starting from a few labelled instances, calculate uncertainty and utility, then query, etc.. Then repeat until you have enough labels.

This code does not deal with the active learning loop, but only performs a query given a certain set of labelled and unlabelled data.

So given DL, X, L, Li as in documentation, returns the most useful instances.

1.2 Automatic addition of certain instances

In the paper it is said that if the uncertainty of an instance is equal to 0, then this is added to the labelled data with the predicted label, since there's no uncertainty on it.

This feature is not present here. This is because of the particular structure of the code.

ACTS is implemented as a query strategy, so it does not interact with the active learning manager, and thus it cannot add a certain instance to the labelled dataset.

1.3 Pattern creation and assigning

Another difference is in how the patterns (or shapelets) are found. And how each instance is assigned to a pattern.

In the original paper, the method to find patterns in the dataset is the one described in Time series shapelets: a new primitive for Data Mining.

This is basically a brute force algorithm. For each possible subsequence in the dataset, the algorithms tries to determine the optimal split point and information gain.

After having checked all the possible candidates, it selects the best one.

Despite using early abandon strategies, this algorithm takes a long time.

When the patterns have been found, a binary tree is constructed. To assign an instance to a pattern it is necessary to run down the tree. As shown in the paper above (figure 13).

Inside the acts paper, the logic is the following:

- Start from a small number of labelled instances. At first, each instance is considered as a pattern.
- Once other labelled instances arrive, they are assigned to a pattern.
- If instances of different labels are assigned to the same pattern, apply pattern splitting to that particular pattern

This is not the procedure applied in the code. We used a different approach, that has the same logic and should be equivalent.

To retrieve the patterns, we don't use a brute force algorithm, but a rather more recent algorithm, described in Learning Time Series shapelets.

This is an iterative algorithm that generates some *candidate patterns*, meaning a series of patterns that might be helpful in discriminating the classes in the dataset.

So the logic that we used is the following:

- Compute pattern candidates
- Calculate for each labelled instance the shapelet transform: vector of distances between the instance and each pattern candidate.
- Run a decision tree on these shapelet transform to find the most important patterns.

The decision tree has the same exact function of the binary tree of the original algorithm.

The only difference is in how the shapelets are computed.

Also, in our case, since the search for pattern candidates is less prohibitive in time, we do not perform initialization and pattern splitting strategy described earlier.

Instead, we look for the best patterns in all the dataset at each call of the function.

That is, we don't perform the sequence of operations described in the first list of this subsection, but rather:

- When new labelled instances arrive, we add them to the pool of labelled instances
- Delete previous patterns
- Look for new patterns using all the available data

When training the decision tree, not all pattern candidates are used. Some of them might not be used by the tree. They are removed in "drop_empty_pattern".