Computational intelligence project n°2 : Generating unit tests for methods

Problem :

Creating tests for every method or function can be really long and is often seen as “not interesting” by developers. Furthermore, creating tests by hand can lead to incomplete code coverage. If we could automate this process, it would be a great gain of time and a preciseness.

Data :

The dataset used in this project is “methods2test” from Microsoft. The dataset is available on a git repository (https://github.com/microsoft/methods2test). It is a collection of java methods and classes linked to their unit test cases and unit test methods coming from more than 9000 different repositories. The set is already separated into train, test and validation subset. For computational speed problems, we will not use all the data but only a small part (~1000 folders containing each between 10 and 100 classes) of it.

Machine Learning Approach :

To address the problem of automating test case generation, a clustering-based machine learning pipeline was implemented. The goal was to group semantically similar methods together and leverage these groupings to find and adapt test cases for a given input method. This approach involved the following steps:

1. Data Representation

The dataset contains pairs of Java methods and their corresponding unit test cases. To enable clustering, the source code for each method and test was

preprocessed into a numerical format suitable for machine learning. This was achieved using the following process:

TF-IDF Vectorization: Each method and test case was tokenized, and a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer was used to convert the code into a high-dimensional numerical representation. This technique captures the relative importance of tokens in the corpus, ensuring meaningful feature representation.

2. Clustering

Clustering algorithms are unsupervised machine learning techniques that group similar data points into clusters based on their feature representations. For this project:

- K-Means Algorithm: The widely-used K-Means algorithm was selected for clustering due to its simplicity and effectiveness. The methods' TF-IDF vectors were input to the K-Means model, which partitioned them into a predefined number of clusters (`n_clusters`), where each cluster ideally contained semantically similar methods.
- Cluster Assignments: After fitting the model, each method in the dataset was assigned to a cluster. These clusters represent groups of methods that are likely to have similar functionality.

3. Test Case Matching

Given an input Java method, the following steps were taken to find a suitable test case:

- The input method was vectorized using the same TF-IDF vectorizer, and its cluster was determined using the trained K-Means model.
- All test cases associated with methods in the same cluster were identified as candidates for the input method.
- To rank the candidate test cases, a cosine similarity metric was computed between the input method vector and the vectors of each candidate test. The most similar test case was selected as the best match.

Validation Results :

For now, the output only provide one test for each method which is incomplete. Furthermore, because of computation power available, there is only 100

clusters on 1000 different projects which does not give a good result and because of the nature of tests, we cannot really find the accuracy of a given model. However, for tested methods, the result was not satisfying.

Improvements :

A greater computational power could help to train the model more accurately, permitting to adjust parameters more precisely.

We could integrate the model in an app for a better user experience or even integrate as a plugin in IDEs.

An NLP model could be use in complement of this k-mean to adapt tests to the existing code. Incremental learning should also be implemented in order to extends the already existing database.