Sisteme Inteligente, Seria B, Sem. 2, 2022-2023

A

Mark as done

Opened: Monday, 24 April 2023, 12:00 AM
Due: Wednesday, 31 May 2023, 11:45 PM

Please read the instructions carefully. If you fail to comply with any of the requirements, you risk that your assignment is graded with 1. For this assignment, you will work individually.

Each student must present his/her project in front of the TA during the lab hour, in week 14.

You are asked to implement in scikit-learn/Tensorflow a classifier that solves the task of predicting a conflict in software projects, in case of concurrent commits. The classifier should be selected by yourself. You should follow the methodology steps presented in the lab (End to end Machine Learning project). The dataset which will be used is available here. The features are:

1) Whether a Pull Request (PR) was created before merging the branches.

When a developer requests for the changes to be pulled in another branch, a common practice is to open a PR so that their colleagues can review the code and approve the changes if everything is ok. It is a good chance that after creating a Pull Request, many dangers hidden in the code that might have caused a conflict to disappear in the process. The value for this feature is 0 (for no PR opened) and 1 (PR was created for that merge commit).

2) The number of added and deleted lines. This might help in confirming the probability of a conflict.

3) The number of developers on each branch. It has been proven that more parallel work in the development process leads to a higher chance of having merge conflicts.

4) The duration of parallel development. The active time of development might be an indicator of complex code. The value of this feature is expressed in hours.

5) The number of simultaneously changed files. If there are no files that have been simultaneously changed in both branches, then it is impossible for a conflict to occur. An increased number of files changed in parallel raises the risk of having different lines of code in the same part of the file on the two separate branches.

6) The number of:

- added files
- deleted files
- renamed files
- copied files
- modified files

These numbers indicate the degree of complexity of changes made in both branches. The larger the modification, the higher the possibility of finding inconsistent changes through the committed files.

7) The number of commits on each branch. The number of commits is an indicator of intense activity in a certain branch. Thus, with increased activity the number of changes and inconsistencies can possibly raise. The way this number is computed is by counting how many commits have been made from the common ancestor until one of the parents of the merge commit.

8) The density of commits in the last week of development. Similar to the previous feature, a high number of commits in the time span right before the merge has happened can also indicate that many changes have been made over a short period of time. When the density is high in the previous week of the development, most times it means that there is high time pressure for the feature to be ready and for the merge to happen. Therefore, changes made in a hurry may result in bugs and inconsistencies.

9) The frequency of selected keywords in the commit messages on both branches. The list of the analyzed words:

- fix

- bug
- feature
- improve
- document
- refactor
- update
- add
- remove
- use
- delete
- change

The presence of some of these words has the potential to reveal a certain connection between the operations made in the commits, the commit messages and their indication of a merge conflict. The measurements are made by taking every commit on a branch starting from the ancestor to the parent of the merge commit, retrieving its commit message and searching for the keywords.

10) Commit message length. Besides searching for keywords, we tried to check whether the length of commit messages has an impact on the result of the merge. The main assumption is that the length of the commit might be an indicator of the code quality. Thus, when commits are well explained, we infer the code is also more thoroughly developed. Then, a lower quality of code can be the grounds for more future conflicts, based on the high probability of the code not having a proper structure or architecture.

11) Four features derived from the above:

- minimum length of the commit messages in a branch
- maximum length of the commit messages in a branch
- median length of the commit messages in a branch
- average length of the commit messages in a branch

Goals:

a) (2p) data preprocessing, inspection, visualization, scaling and encoding

b) (2p) train / test split and performance metric selection

c) (1p) attribute selection and combinations

d) (4p) model selection, evaluation on the training set and optimization; underfitting and overfitting avoidance; evaluation on the test set

Possible bonus: imputation for missing data (as discussed with your TA), pipeline processing

Deliverable:

One jupyter notebook containing all the required steps implemented in scikit learn (or Tensoflow, if you decide so) and explanations+results (code and markdown cells respectively). The notebook name should be the same as the name of the author (surname first). For example, if the author is Ioana Popescu, the notebook should be called PopescuIoana.ipynb and will be uploaded on moodle before the deadline. Your notebook must be developed in such a way that I can smoothly run it on Ubuntu Linux with no further adjustment needed.

Add submission

Submission status

Submission status	No submissions have been made yet
Grading status	Not graded
Time remaining	35 days 13 hours remaining
Last modified	-
Submission comments	Comments (0)

◄ Attendance

Jump to...

B ►