MIDDLE EAST TECHNICAL UNIVERSITY NORTHERN CYPRUS CAMPUS
CNG445 Software Development with Scripting Languages

Assignment 1: A Data Analyser for the Classification Commit Messages

This assignment aims to help you practice Python basics including primitive types, variables, operators, decision making, loops, sequences, dictionaries, functions, modules, regular expressions, exception handling, command line arguments, and file processing. Your main task in this assignment is to develop a simple data analyser for commit messages and their classifications.

Overview

Software developers commonly use Git[1] to work together and keep track of their changes over time. When they complete their changes, they commit them to their Git software repository with a commit message to describe what they have changed.

The application that you will develop in this assignment will process **txt** data files in the following format. You will be provided with two input files, for example, **identities.txt** and **commits.txt**.

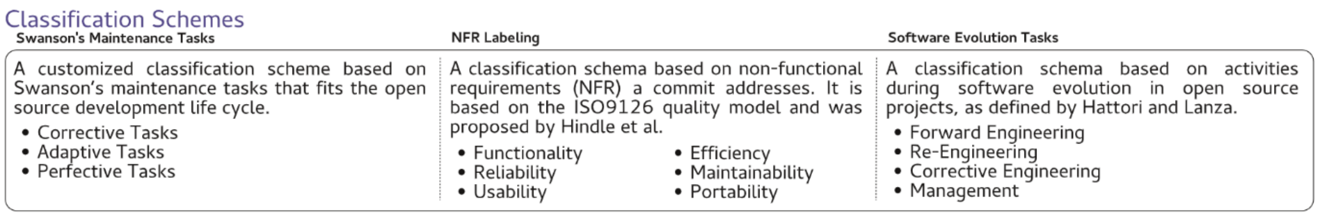
- **identities.txt** contains committer (developer) information, including their IDs, names, and email addresses.
- **commits. txt** contains commit messages, their IDs, classification features, and committer IDs. This file includes the commits from multiple repositories.

Sample data for the commits file will look like:

```
1,1,0,0,0,0,1,0,0,0,1,0,0,0,1,Add support for unowned/weak array elements
2,0,0,1,0,1,0,0,0,0,1,0,0,0,1,drivers: Mark arrays as unowned instead of weak
3,1,0,0,0,1,1,0,0,0,1,0,0,0,2,ctypresolver: signals: register default implementations
4,1,0,0,0,0,1,0,0,0,1,0,0,0,3,signals: collect default implementations
5,0,1,0,0,0,1,0,0,0,0,0,1,0,3,embedded: Fix check() for relative paths
```

Column(s)	Explanation
1	Unique Commit ID
2-14	Commit classification features: 2-4 Swanson's Maintenance Tasks, 5-10 NFR Labelling, 11-14 Software Evolution Tasks
15	Committer ID
16	Commit message

As can be seen, the data will be separated with commas, thus parsing of the data should be considered based on this information. Also, in Figure 1 given below, you find the information about the classification features in further details.



Classification Schemes

Swanson's Maintenance Tasks

A customized classification scheme based on Swanson's maintenance tasks that fits the open source development life cycle.
- Corrective Tasks
- Adaptive Tasks
- Perfective Tasks

NFR Labeling

A classification schema based on non-functional requirements (NFR) a commit addresses. It is based on the ISO9126 quality model and was proposed by Hindle et al.
- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Software Evolution Tasks

A classification schema based on activities during software evolution in open source projects, as defined by Hattori and Lanza.
- Forward Engineering
- Re-Engineering
- Corrective Engineering
- Management

Figure 1. Classification schemes with information about the features provided in the dataset.

- **Swanson's Maintenance Tasks (SwM):**
  Features: (Column 2) Adaptive Tasks, (Column 3) Corrective Tasks, (Column 4) Perfective Tasks. Commit messages can **only have one** of these features marked as "true" and others as "false", i.e., 1 and 0 respectively.

[1] https://git-scm.com/

- **NFR Labelling:**
  Features: (Column 5) Maintainability, (Column 6) Usability, (Column 7) Functionality, (Column 8) Reliability, (Column 9) Efficiency, (Column 10) Portability.
  Commit messages can **have multiple** features marked as "true". Please note that a commit may not have any NFR labelling, i.e., all the features might be "false".

- **Software Evolution Tasks (SoftEvol):**
  Features: (Column 11) Forward Engineering, (Column 12) Re-Engineering, (Column 13) Corrective Engineering, (Column 14) Management.
  Commit messages can **only have one** of these features marked as "true".

Furthermore, sample data for the identities file will look like this and will also be separated with commas:

```
1,Brosch Florian,flo.brosch@gmail.com
2,Luca Bruno,lucabru@src.gnome.org
3,Luca Bruno,luca.bruno@immobiliare.it
```

Column	Explanation
1	Unique Committer ID
2	Full Name of the Committer/Developer
3	Committer email address

Please note that, the same committer might have different accounts bind to different email addresses and with different IDs, thus you must handle such kind of cases accordingly.

Implementation Requirements
This application should receive the data file names as a **command-line argument** and then create a nested dictionary. Please note that the content of the data files can be changed but their formats will be the same.

The sample command-line execution of the program should be as follows. Please note that you need to name your program as `commitsanalyser.py` , but the names of the files for the commits and identifies can be different.

```
python commitsanalyser.py commits.txt identities.txt
```

The **nested** dictionary whose structure is shown in Figure 2 is created. The rounded boxes show the dictionaries and the rectangles in the rounded boxes show the dictionary keys. A key should be created for each committer based on their names (assuming that there is no committer with the same name and surname)**.** For each of the committers that are available in the identities file, a dictionary should be able to store the total number of each type of classification scheme that are given in the commits file. The counters for classification scheme features should be stored in the list structure.

You are not allowed to change the structure of this dictionary or keep additional information in the dictionary. However, you might use an auxiliary dictionary to store the data in the identities file.

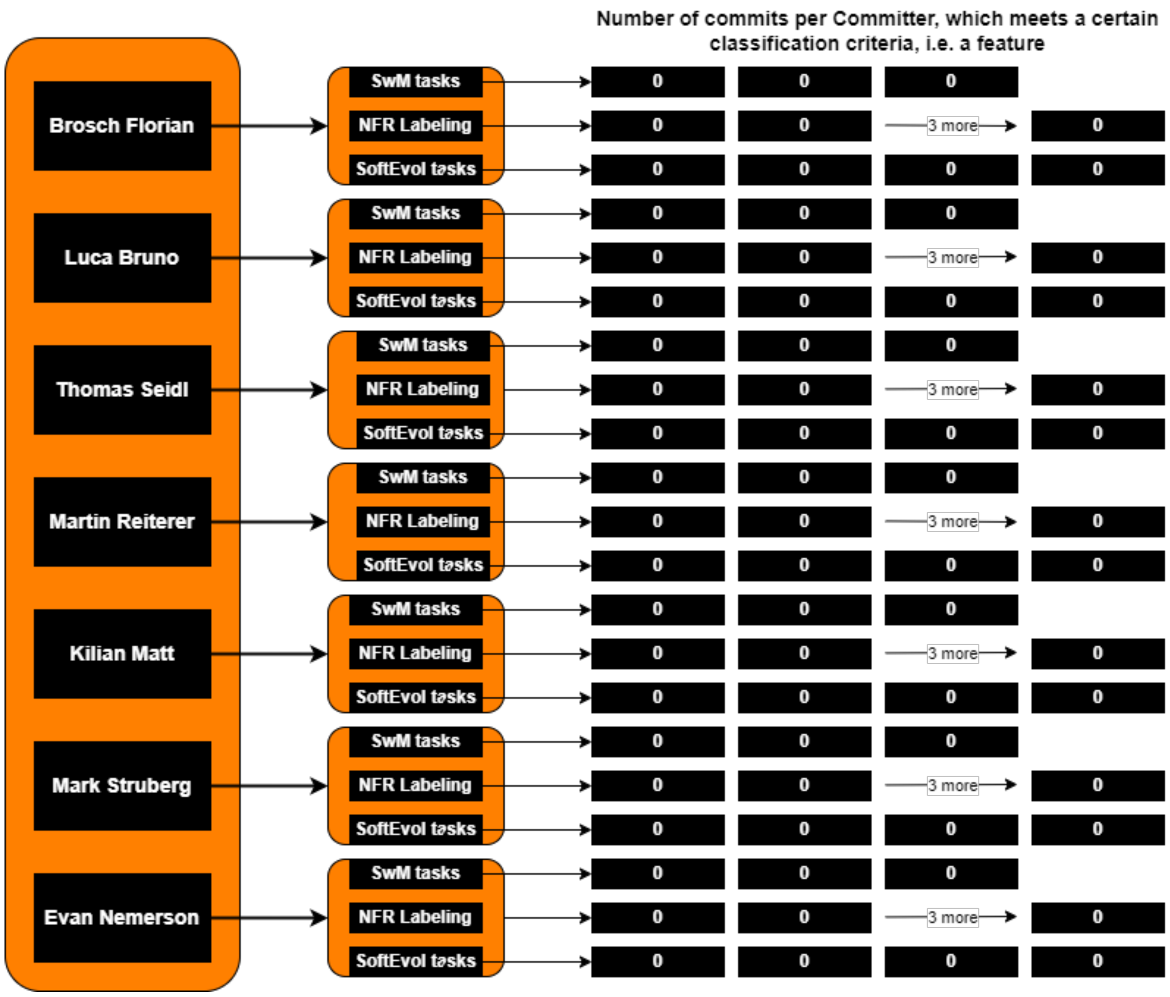


Figure 2. Required structure to keep the information about committers and classification information about their commits

Once the identities file is read and the identities information is used to create a dictionary, the dictionary is updated accordingly while the commits file is being read. The application will then show the following menu in a loop:

1. Compare the number of commits done by a particular developer for a given **classification scheme**.
2. Compare the number of commits done by all developers, which are classified with a given **feature** (for example, developer X has Y commits, developer I has J commits, and developer A has B commits for a given feature).
3. Print the developer with the maximum number of commits for a given **feature** (for example, print the developer who has the maximum number of commits with Corrective Tasks).
4. Exit

Once a menu item is selected, the application should ask for the required details.

- If the first item is selected, then the classification scheme (Swanson's Maintenance Tasks, NFR Labelling or Software Evolution tasks), and the committer (such as, Brosch Florian) should be selected. Please note that the menu should list the available classification schemes and committers, and then one classification scheme and one commit will be selected. For example, if the Swanson's Maintenance Tasks (SwM) are chosen for the developer Brosch Florian, then the application should compare and display the number of commits classified by SwM features (such as, Corrective, Adaptive, etc.).
- If the second or third item is selected, the menu should list the available classification schemes. Once the user selects one of the classification schemes (Swanson's Maintenance Tasks, NFR

Labelling or Software Evolution tasks), then the menu will display its corresponding features (such as, Corrective Tasks, Adaptive Tasks or Perfective Tasks if Swanson's Maintenance Tasks is selected as a classification scheme), and the user will select one of them.

The results for the first two menu items should be shown as a bar chart, you can find an example piece of code below about how you can create a bar chart in Python (see Figure 3). Please note that you need to install `matplotlib`.

```
import matplotlib.pyplot as plt
features = ["Corrective Tasks", "Adaptive Tasks", "Perfective Tasks"]
values = [11, 9, 13]
plt.bar(features, values)
plt.xlabel('Features')
plt.ylabel('Total Number of Commits')
plt.title('Comparison for Brosch Florian\'s Commits Classified by SwM Tasks')
plt.show()
```

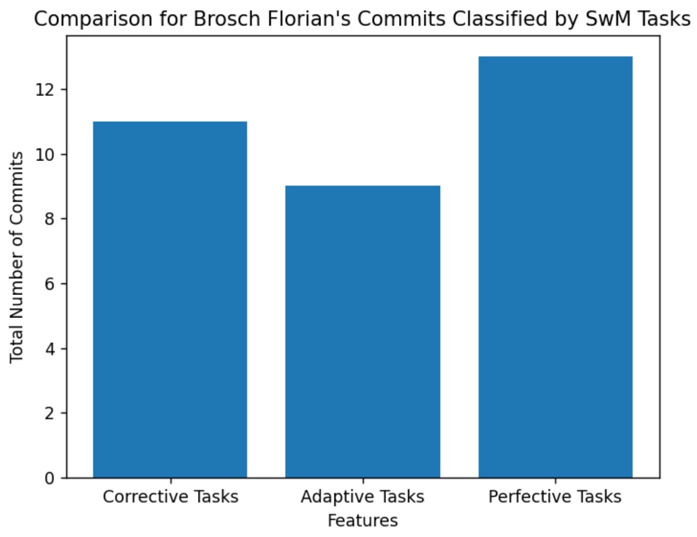


Figure 3. Bar chart for the comparison of the commits from a particular developer for SwM Tasks

Sample Data

Sample data with identities and commits are available on ODTUCLASS. These data files are from the dataset collected by Mauczka et al. (2015)[2] and modified for this assignment.

Rules

- You need to write your program by using **Python 3.x**.
- You can **only** use all built-in functions and modules (apart from matplotlib)
- You also need to create a file called ReadMe.txt which contains the following items. Please note that **if you do not submit ReadMe.txt, your submission will not be evaluated.**
  - Team members
  - Which version of Python 3.x you have used
  - Which operating system you have used
  - How you have worked as a team, especially how you have divided the tasks among the team members (who was responsible for what?), how you have communicated, how you have tested the program, etc.
- You need to put all your files into a folder which is named with your student id(s) and submit the compressed version of the folder in the **.zip** format.
- **Only one team member** should submit the assignment.
- **Code quality, modularity, efficiency, maintainability, and appropriate comments** will be part of the grading.

[2] https://github.com/flobrosch/msr-data

Grading Policy
Your assignment will be graded as follows:

Grading Item	**Mark (out of 100)**
Take command line arguments and use them to open correct data files	5
Initial dictionary creation based on the identifies file	12
Read from a file	8
Update the total number of commits for the SwM scheme	10
Update the total number of commits for the SoftEvol scheme	10
Update the total number of commits for the NFL labelling scheme	15
Compare for a specific classification scheme and a developer with a bar chart	10
Compare for a specific feature for developers with a bar chart	10
Print the developer with the maximum number of commits with a given feature	10
Manage menu operations	10

References:
Andreas Mauczka, Florian Brosch, Christian Schanes, and Thomas Grechenig. 2015. Dataset of developer-labeled commit messages. In Proceedings of the 12th Working Conference on Mining Software Repositories (MSR '15). IEEE Press, 490–493.