NTNU

Kunnskap for en bedre verden

Assignment 4: Construction Project

TPK4186 - Advanced Tools for Performance Engineering

Done by

Group 10

Georg Hove Zimmer & Fredrik Faanes Slagsvold

Task 1 & Task 2

We have two main classes in our data structure. The first one is the Task class, which holds all the information related to an individual task. The second class is the Project class, which is essentially a collection of tasks and gates. This data structure allows us to organize and manage projects effectively.
To facilitate reading project data from an Excel sheet, we have implemented the Reader class. This class takes an Excel file as input and returns a Project object, representing the project structure described in the file. Additionally, within the Reader class, we establish connections between the predecessors and successors of each task to ensure proper task sequencing.
Our implementation has been tested and verified using two sample files: ***Villa.tsv*** and ***Warehouse.tsv***. These files serve as evidence of the functionality of our Reader class, as they demonstrate successful parsing and construction of Project objects based on the data provided in the assignment. We also have a printer class that prints the output.

Task 3 & Task 4

Our Calculator class plays a central role in performing various calculations for project management. It includes two important methods: ***calculate_early_times()*** and ***calculate_late_times()***. These methods are responsible for computing the early start date, early completion date, late start date, and late completion date for each task in the project.
To validate the correctness of these calculations, we have compared the results obtained from the Calculator class with the expected values provided in the assignment. See ***expected_time.tsv*** file for the Warehouse project. This serves as a confirmation that our calculations align with the assignment requirements.

In order to incorporate risk factors into the calculations, we have introduced additional methods: ***calculate_early_times_with_riskfactor()*** and ***calculate_late_times_with_riskfactor()***. These modified versions of the calculation methods take into account a risk factor and use random time values instead of the expected time (using the random.triangular() function).

To handle the update of task durations, we have implemented the ***update_duration_tuple()*** method. This method allows us to modify the duration of a task based on the provided risk factor, i.e: ***(a , e’, b)*** where $a \leq e' \leq b$.

To add an intermediate gate to the project, the ***add_gate_at_index()*** function in the Project class can be used. This function inserts a gate after the specified index. The ***split_project()*** function can then be employed to access all tasks upstream of the gate, splitting the project accordingly.

The ***run_project()*** function executes the project simulation and performs the necessary calculations. It involves initializing variables, reading project data, calculating expected times, and simulating the project for a specified number of iterations. In each iteration, a new project is created, an intermediate gate is added, task durations are updated, and early & late times are calculated. The project is split, and the duration and status are recorded for analysis.
After running the simulation, the ***perform_statistics()*** function provides essential statistics, including the counters for successful, acceptable, and failed projects, as well as project durations, early completion dates, and project statuses. You can run the Calculator class to see the result from Task 3 and Task 4.

Task 5 & Task 6

The tasks related to machine learning are implemented in the MachineLearning class. The code is well-documented with comments in order to provide clarifications.

For classification purposes, we have utilized three different classification methods: *DecisionTreeClassifier*, *RandomForestClassifie*r, and *Support Vector Classifier*. These methods are employed to classify the instances based on their attributes. We have generated a confusion matrix to evaluate the performance of these classifiers. Additionally, a classification report is displayed in the terminal when the code is executed, providing detailed information about the classification results.
Interpretation of the results of one run (See Machine Learning Task 5.tsv):

1. Decision Tree Classifier Report:
    - **Precision**: This metric measures the accuracy of the positive predictions. For the "Acceptable" class, the precision is 0.64, meaning that 64% of the instances predicted as "Acceptable" were correct. The precision for the "Failed" class is 0.50, and for the "Successful" class, it is 0.79.
    - **Recall**: This metric measures the ability of the classifier to identify all the positive instances. The recall for the "Acceptable" class is 0.54, for the "Failed" class it is 0.54, and for the "Successful" class it is 0.87.
    - **F1-score**: This metric is the harmonic mean of precision and recall. It provides a balanced measure of the classifier's performance. The F1-score for the "Acceptable" class is 0.58, for the "Failed" class it is 0.52, and for the "Successful" class it is 0.83.
    - **Support**: This indicates the number of instances in each class in the test set.
2. Random Forest Classifier Report:
    - Precision, recall, and F1-score are calculated in the same way as for the Decision Tree Classifier.
    - Compared to the Decision Tree Classifier, the Random Forest Classifier generally shows improved performance across all classes. The precision, recall, and F1-scores are higher, indicating better accuracy in classifying instances.
3. Support Vector Machines Classifier Report:
    - Precision, recall, and F1-score are calculated in the same way as for the Decision Tree and Random Forest Classifiers.
    - The Support Vector Machines Classifier exhibits varied performance across different classes. It has the highest precision and recall for the "Successful" class, indicating accurate predictions. However, it has lower precision and recall for the "Failed" class, suggesting more difficulty in correctly identifying instances in that class.

Overall, the Random Forest Classifier shows the highest accuracy, as indicated by the accuracy metric. It demonstrates better performance across all classes, with higher precision, recall, and F1-scores. The Decision Tree Classifier follows, while the Support Vector Machines Classifier has lower accuracy and shows more variation in performance across classes. Furthermore, the confusion matrices have been saved as PNG files (**confusionmatrix1**, **confusionmatrix2**, and **confusionmatrix3**) in the directory for further analysis. By running MachineLearning.py, you can show the result in the terminal, but there are also two tsv files with your evaluation.

Regarding regression, we have applied three regression methods: *LinearRegression*, *DecisionTreeRegressor*, and *RandomForestRegressor*. The results are reported in terms of Mean Squared Error (MSE), which is a common metric used to evaluate regression models.

Interpretation of the results of one run (See Machine Learning Task 6.tsv):

1. Linear Regression MSE: 0.5297364580832918
    - The Linear Regression model achieved an MSE of approximately 0.530.
    - The MSE measures the average squared difference between the predicted and actual values.
2. Decision Tree Regression MSE: 0.895
    - Compared to the Linear Regression model, this indicates a higher level of error in the predictions.
    - The Decision Tree model might be overfitting the training data or lacking the ability to capture the underlying patterns effectively.
3. Random Forest Regression MSE: 0.42523449999999996
    - The Random Forest Regression model achieved an MSE of approximately 0.425.
    - This value is lower than both the Linear Regression and Decision Tree Regression models' MSEs.
    - The Random Forest model is performing better in terms of prediction accuracy, as it is able to leverage the ensemble of decision trees to make more accurate predictions.

Overall, based on the MSE values, the Random Forest Regression model seems to be the most effective among the three regression methods for the given task.

We can see by testing and observing that the later we put the intermediate gate, the better the results. This makes sense as the model gets more tasks to train on and therefore a better result. We have to discuss where the best placement of this intermediate gate is. By the description of each task, we thought it was a logical splitting of the project after G.5. Then our intermediate gate is in the project to tell us that phase one is finished and we could start phase 2.

If you want to look at different results you could change the input value in add_gate(“G.5”) in run_project() in Calculater.py on line 196, to some other code (“N.2” for instance).
All the files (tsv) are run with an intermediate gate before G.5, but right now the code contains an intermediate gate before N.2. So you can see that an intermediate gate later in the project gives a better result.

NB! Running the code in the Calculator file and in the MachineLearning file takes a little bit of time.