NTNU
Norwegian University of Science and Technology

TPK4186 - Advanced Tools for Performance Engineering
Spring 2023

Assignment 4: Construction Project

Group 11

Sipan Omar, Morten Husby Sande & Kim-Iver Brevik Blindheimsvik

1.2) Organization of code

Libraries used in our code:

- *numpy*: adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- *matplotlib*: Creates static, animated, and interactive visualizations in Python.
- *re*: Provides regular expression support. Regular expressions are a powerful language for matching text patterns.
- *sys*: This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.
- *openpyxl*: A library for reading and writing Excel files. It allows easy manipulation of Excel spreadsheets, including adding/removing data to spreadsheets, looking for data in cells, storing information in python, and more.
- *math*: Provides access to the mathematical functions defined by the C standard.

Sklearn libraries:

- *sklearn.svm*: Provides a set of efficient tools for machine learning and statistical modeling using support vector machines. It offers a variety of SVM algorithms along with tools for classification, regression, and outlier detection.
- *sklearn.model_selection*: Provides tools for cross-validation, hyperparameter tuning, and train-test splitting of data.
- *sklearn.metrics*: Provides tools to evaluate the accuracy of a sklearn model.
- *sklearn.neural_network*: Multi-layer Perceptron classifier.
- *sklearn.ensemble*: Includes averaging algorithms based on randomized decision trees.
- *sklearn.tree*: A decision tree classifier.

Assumptions:

In the *Villa.xlsx* file we have made the assumption that J.1 is listed up at the wrong place in the original spreadsheet. With an official contract we would of course consult the company to make sure that this is the case. The code provided has a fixed version of the excel file(which of course also is provided along with the original file. The original file is named *VillaOriginal.xlsx*. All further calculations are based on the fixed version of the excel file.

Structure:

The python code is divided into 8 different python files. *Main.py* is the file where all of the functions are tested and run. *Printer.py* is the file where everything that gets outputted is designed to be appealing and understandable. The printer functions are run in *Main.py*.
Loader.py, Task.py, Regression.py, Project.py, Classification.py and Statistics.py include classes and functions that are described below.

2.1) PERT Diagrams

Task 1: Design data structures to encode PERT diagrams.

In the *Task.py* file we have the functions to encode PERT diagrams in the *Task* class. Information on every task such as task- *key, description*, *duration, etc.* is defined for each task (see figure 1). There are get and set functions for most of this data. These get and set functions are used to extract data from files or do calculations (more on this below). For the duration, the *createDurationValue* function takes in either 'Minimum', 'Expected' or 'Maximum' and assigns a value to the task based on this chosen input. This value decides if the minimum, expected, or maximum duration of a task will be used for the task.

The predecessors for each task are given, but can be changed, set or added with given functions. For the successors the *addSuccessors* function uses a list of all tasks and assigns successors to a task based on this. In the *Project.py* file, the *Project* class takes in data which can be extracted from an excel file, or created by creating tasks with task data. In this class, the *addAllSuccessors* function uses the *addSuccessors* from the Task class to assign successors to all tasks in a project. Tasks are stored in the form of dictionaries so the *createListFromDict* has been created to transform that data to a list. This is necessary for some calculations later.

```
class Task:
    def __init__(self, key, description, duration, predecessorTasks=None, successorTasks=None):
        self.earlyStartDate = None
        self.lateStartDate = 0
        self.earlyCompletionDate = None
        self.lateCompletionDate = None
        self.key = key
        self.description = description
        self.duration = duration
        self.predecessorTasks = predecessorTasks if predecessorTasks else []
        self.successorTasks = successorTasks if successorTasks else []
        self.lastTask = False
        self.firstTask = False
        self.criticalTask = self.isCriticalTask()
        self.durationValue = None
```

Figure 1: The *__init__* function in the task class with all parameters

Task 2: Design a loader to load projects from Excel spreadsheets. Two examples of spreadsheets are joined: Warehouse.xlsx and Villa.xlsx.

The *ExcelLoader* function in the *Loader.py* file uses the openpyxl library to transform data from excel documents into tasks and combines them into a dictionary. Every task gets an assigned key which is the task name. *Descriptions*, *durations, predecessor tasks, etc.* are also extracted and assigned to a task. The task dictionary is then turned into a project. *ExcelLoader* finally runs the *createListFromDict* and *addAllSuccessors* from the Project class before it returns a project.

Design a printer to print projects, including the list of successors of each task, their early and late dates, and their criticality. Use this printer to test both the data structures you designed in Task 1. and your loader.

The *printProjectExcel* function takes in a project and prints the data in a structured way to a txt file.

The *printProjectSummary* function prints projects, including the list of successors of each task, their early and late dates, and their criticality. Other information such as task durations, total project duration etc is also represented with this function. (In the *Main.py* file the *printProjectSummary* for the warehouse and villa project are under Task 3 because calculations have to be done before it gives proper answers).

Task 3: Design a calculator to calculate the early and late dates of each task of a project. Test your calculator on both examples Warehouse.xlsx and Villa.xlsx.

The *calculateEarlyStartDate* and *calculateEarlyCompletionDate* uses the predecessor tasks and iterates through them to set the correct early dates for a task. The *calculateLateCompletionDate* and *calculateLateStartDate* uses *successors* and therefore different logic where it has to start from the back of the tasklist and move backwards to calculate the correct late dates for a task. With the *setCriticalTasks* function, a task gets a True or False tag based on if earlyStartDate == lateStartDate. All the functions stated above are for one task. In the Project class function with for loops go through all the tasks and assign successors, early dates, etc to the whole project. With the *calculateAll* function shown in figure 2, everything will be calculated in one function. The *value* parameter in the *calculateAll* function is as stated above, 'Minimum', 'Expected' or 'Maximum' which decides what duration will be used for the other calculations. All of the calculated values are stored in the parameters shown in figure 3. As written for Task 2, the Printer class handles the representation for this information.

```
def calculateAll(self, value):
    self.setDuration(value)
    self.calculateEarlyDates()
    self.calculateLateDates()
    self.calculateCriticalTasks()
    self.calculateProjectDuration()
```

Figure 2: Function in the *Project* class that makes all the necessary calculations for a project.

```
class Project:
    def __init__(self, tasks):
        self.tasks = tasks
        self.earlyStartDate = 0
        self.lateStartDate = 0
        self.earlyCompletionDate = 0
        self.lateCompletionDate = 0
        self.Duration = 0
        self.taskList = []
        self.status = "unchecked"
```

Figure 3: The *__init__* function in the *Project* class with all parameters

Design functions to calculate the shortest, the expected and the longest duration of a project.

The *printProjectDurations* in the Printer class takes in a Project and calculates the duration of each project based on if the 'Minimum', 'Expected' or 'Maximum' durations are used. The *printProjectDurations* works by first running the *calculateAll* function for a duration, and then the *calculateProjectDuration* to give out the duration for that project.

2.2) Machine Learning

Task 4: Draw at random a sample 1000 of values of durations, for each value of the risk factor, according to the above principle. Calculate the project duration for each project of the sample. Perform statistics on these durations (minimum, maximum, mean, standard-deviation, deciles), as well as on the numbers of successful, acceptable and failed projects.

In the *Statistics* class all of the functions to answer task 4 are present. *randomDurationsOfProjectTasks* takes in the TaskList which is already set when the villa project got imported. Every task in the list gets a copy that is then assigned a new duration value based on a risk factor and the premises given in the Assignment. The new tasks are created with the new durations and added to a new project. Calculations for the project are done with the *calculateAll* function and the new project is then returned without changing the information of the original taskList and villa project.

The *getRandomProjects* function takes in numSamples and a riskFactor and creates a list with the number of stated projects. In our case numSamples is set to 1000, so that 1000 random projects are generated and added to a list that is then returned. This list is then run in the *getProjectDurations* function that gives out the project duration for all projects in a list. This data is then returned. The *calculateStatistics* function uses this list of project durations to do calculations such as mean duration, std, etc. It also compares the different durations to the Expected duration to check if a task is successful, acceptable or a failure.

All of this information is stored in the class and called upon in the *printStatistics* and *printProjectClassification* functions in the Printer class.

Results:

```
Number of successful projects: 995
Number of acceptable projects: 5
Number of failed projects: 0
```

Results for r = 0.8

```
Number of successful projects: 797
Number of acceptable projects: 203
Number of failed projects: 0
```

Results for r = 1.0

```
Number of successful projects: 14
Number of acceptable projects: 462
Number of failed projects: 524
```

Results for r = 1.2

```
Number of successful projects: 201
Number of acceptable projects: 759
Number of failed projects: 40
```

Results for r = 1.4

```
Number of successful projects: 482
Number of acceptable projects: 365
Number of failed projects: 153
```

Results for r = 'random'

Task 5: Draw at random a sample of instances according to the above principle in order to apply classification methods. Apply at least 3 different classification methods. Use 80% of the sample as a learning set and the remaining 20% as a test set. Report and discuss the results.

The different training and test sets were split using the **train_test_split** function from *sklearn.model_selection*[1] package. See figure 4 below. For task 5 we built our classification methods with the following functions (Figure 4), similar to task 6. We imported RandomForestClassifier (RFC)[2], DecisionTreeClassifier (DTC)[3] and MLPClassifier (MLP)[4] from the sklearn library.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)
```

Figure 4: train_test_split function in action from the *sklearn.model_selection* library and Import of *RandomForestClassifier* for classifications

We converted the labels from text to integers to easier display the number of successful, acceptable and failed projects. The results are printed to both the sys.stdout and to a .csv file: *ClassificationResults.csv* found in the files sub-folder. See results below:

[1] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[3] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[4] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Results:

```
                    Successful                  Acceptable                  Failed
Successful                  95                  7                   0
Acceptable                  7                   51                  9
Failed              0                   10                  21
Accuracy:  0.835
PS C:\Users\morte\OneDrive - NTNU\NTNU\APE\Assignment 4\Aktiv> & C:/Users/morte/AppData/Local/Micr
orte/OneDrive - NTNU/NTNU/APE/Assignment 4/Aktiv/Main.py"
The gates were added
                    Successful                  Acceptable                  Failed
Successful                  117                 4                   0
Acceptable                  2                   48                  3
Failed              0                   7                   19
Accuracy:  0.92
PS C:\Users\morte\OneDrive - NTNU\NTNU\APE\Assignment 4\Aktiv> & C:/Users/morte/AppData/Local/Micr
orte/OneDrive - NTNU/NTNU/APE/Assignment 4/Aktiv/Main.py"
The gates were added
                    Successful                  Acceptable                  Failed
Successful                  79                  19                  8
Acceptable                  45                  14                  5
Failed              21                  6                   3
Accuracy:  0.48
PS C:\Users\morte\OneDrive - NTNU\NTNU\APE\Assignment 4\Aktiv>
```

As we can see from the results above, the RandomForestClassifier (Number 2) method yielded the highest accuracy, and is therefore the optimal model of the three. For these results we used a gate at K.1.

We tried at different gates. For gate H.1 the MLP Classification had these results:

```
                    Successful                  Acceptable                  Failed
Successful                  22                  85                  0
Acceptable                  14                  44                  0
Failed              5                   30                  0
Accuracy:  0.33
```

For gate N.2 the MLP Classification had these results:

```
                    Successful                  Acceptable                  Failed
Successful                  96                  6                   0
Acceptable                  71                  10                  0
Failed              15                  2                   0
Accuracy:  0.53
```

For gate D.1 the MLP Classifier had these results:

```
                    Successful                  Acceptable                  Failed
Successful                  103                 1                   0
Acceptable                  68                  0                   0
Failed              28                  0                   0
Accuracy:  0.515
```

Task 6: Draw at random a sample of instances according to the above principle in order to apply regression methods. Apply at least 3 different regression methods. Use 80% of the sample as a learning set and the remaining 20% as a test set. Report and discuss the results.

NB: Linear regression is slow to compile, but does get there.

For task 6 we used the SVR part of the *sklearn.SVM*[5] package (figure 5). Most importantly the **svr.fit()** and **svr.predict()** functionalities to process our data sets.

```
svr_rbf = SVR(kernel="rbf", C=100, gamma=0.1, epsilon=0.1)
svr_rbf.fit(X_train, y_train)
y_pred = svr_rbf.predict(X_test)
```

Figure 5: Fit model for regression, RBF (Radial Basis Function)

The different training and test sets were split using the **train_test_split** function from the *sklearn.model_selection*[6] package.

We chose to focus on RBF (Radial Basis Function), Linear and Sigmoid regression for this assignment as these were all available in the same package of sklearn.

While the samples are drawn at random, and this obviously leads to variations, the results for each of the methods stayed within the same order of error and never overlapped or came close.. We can see from the results below that the linear regression method yielded the best values for the predicted values in regards to accuracy. The results were calculated using the *sklearn.metrics*[7] package which had built-in functions to calculate **mean_squared_error** etc. The results can be seen in the *RegressionResults.csv* file, change the **kernel** parameter in main between: “rbf”, “lin” or “sig” to see for the different methods.

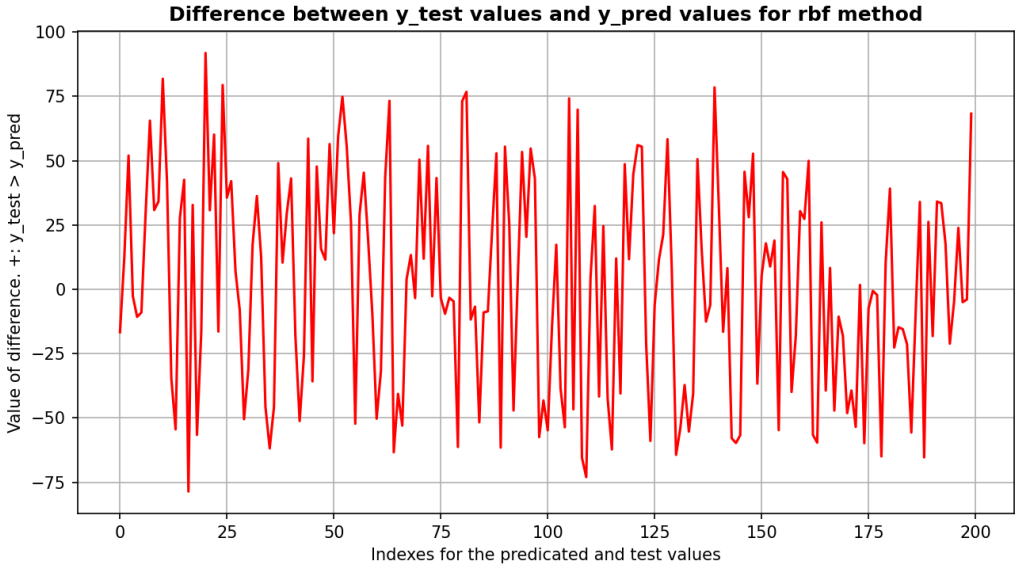
[5] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

[6] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[7] https://scikit-learn.org/stable/modules/model_evaluation.html

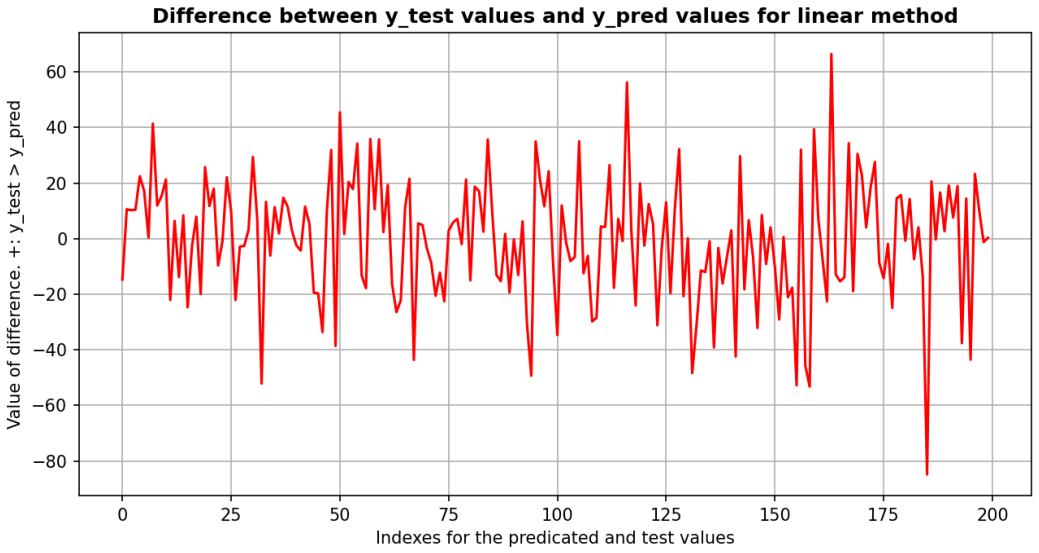
Results *(Gate set at K.1)*:

RBF:



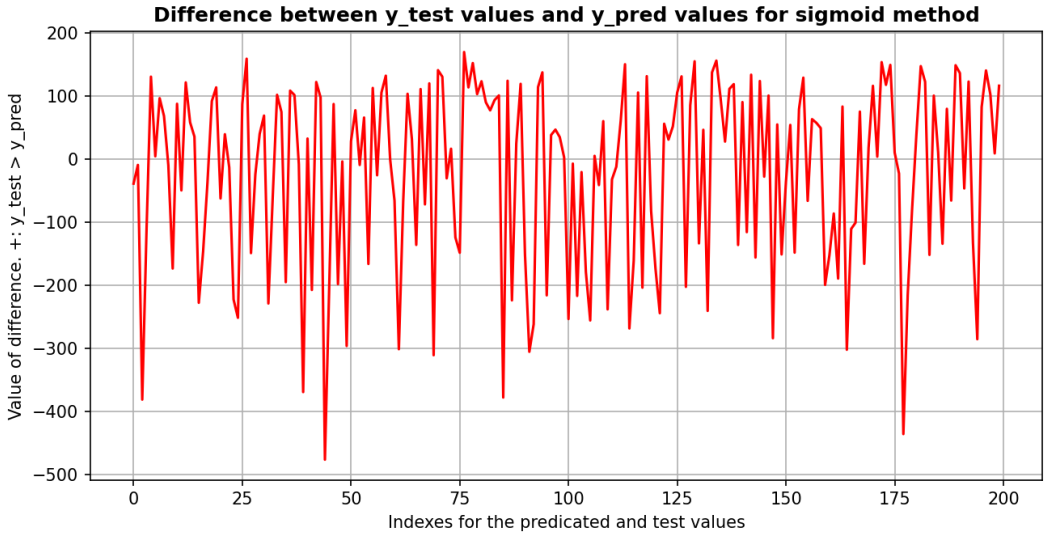

```
Mean Absolute Error:  35.3906030526556
Mean Squared Error:  41.50898272193512
R^2 score:  -0.0019649446771896617
```

Linear:




```
Mean Absolute Error:  17.214161227407207
Mean Squared Error:  22.213739627454565
R^2 score:  0.7064794551868787
```

Sigmoid:




```
Mean Absolute Error:  118.46484989465283
Mean Squared Error:  146.2972037351665
R^2 score:  -10.943302787057025
```

Results for the linear regression method with different gates:

Gate C.1:

```
MAE 17.0385
RMSE    22.2185
R^2 0.686078
```

Gate D.1:

```
MAE 17.5232
RMSE    21.8269
R^2 0.706968
```

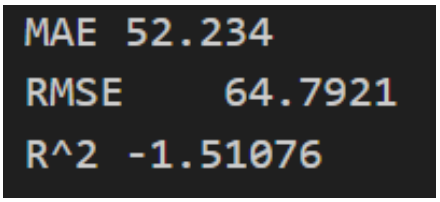
Gate H.1:

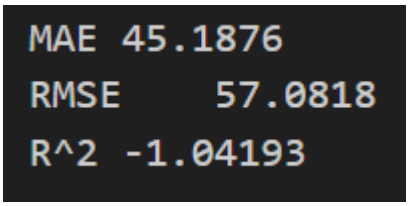
```
MAE 11.0281
RMSE    14.9098
R^2 0.869531
```

Gate M.1:

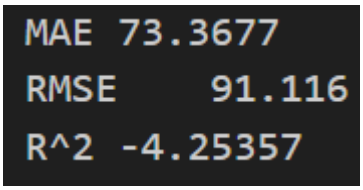
```
MAE 31.6505
RMSE    40.2268
R^2 -0.0264433
```

Gate N.1/N.2:





Gate O.4:



It looks as though the gates in the middle are the most accurate. As we pass the H.1 gate we get a higher error that seems to increase for each gate. The group figured this is because the algorithm has a lower chance to balance the duration if it gets a large error in one of the later guesses at the task duration.