AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

FACULTY OF ELECTRONICS, COMPUTER SCIENCE AND TELECOMMUNICATIONS

DEPARTMENT OF TELECOMMUNICATIONS

Bachelor of Science Thesis

Testowanie metod analizy zaproponowanych w zaleceniu P.1401

Testing Algorithms Proposed by Recommendation P.1401

Author: *Marcin Duplaga*
Degree programme: *Teleinformatyka*
Supervisor: *dr hab. inż. Lucjan Janowski*

Kraków, 2020

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Table of Contents

Introduction

Recently, it has been recorded a significant increase in the use of streaming services on the Internet like Video-On-Demand [1]. Therefore, it is essential to understand the factors, which determine the success or failure of various streaming services or applications. Pursuing this goal, the situation resulted in the development of methods and ways, which can determine users' perception of using those services. The field of science behind that is called Quality of Experience, which is constantly evolving and increasingly complex. Regarding QoE, the basic procedure to measure users' satisfaction of the video is made by a subjective test [2]. One way of carrying out such tests can be through questionnaires, where users' are asked to evaluate the quality of the given multimedia sequence. Afterwards, having the sequence scored by many users, it is possible to define an overall rating for a given media. An Alternative approach of measuring video quality can be done by objective evaluation, which is based on already collected data by subjective tests. There are different ways to develop objective assessment methods, which predict the user's experience. Therefore, the question arises how to evaluate the models' performance to indicate the best one of them. An obvious technique would be to check the difference between objective and subjective results for the same data set, but it is not sufficient. Consequently, the Recommendation ITU-T P.1401 was created for that reason. "It presents a framework for the statistical evaluation of objective quality algorithms regardless of the assessed media type [3]".

Accordingly, in this thesis, I designed a simulator to evaluate metrics from the ITU-T Recommendation to compare objective assessment methods against results from the subjective test. Moreover, the point of the simulation is to verify if it is possible to determine the best performing model as well as establish which parameter is best to use. The simulator is written in Python, and it is compatible with any system that supports this programming language and its necessary libraries. I presented an example of using this simulation tool by running 1000 simulations written in a compact matrix form and use it to present the ITU-T P.1401 metrics performance for various objective models.

In Chapter 1 of this thesis, I am trying to explain the basics of Quality of Experience, subjective and objective test assessments. Afterwards, Chapter 2 contains specification of this simulator and explanation of particular modules that are crucial to acquire correct results. The Chapter 3 includes tests that were taken to verify metrics performance as well as the results of those tests. In Chapter 4, I am sharing my conclusions from the tests, based on results and the theory that stands behind them.

1. The purpose of the project

This chapter is dedicated to explain the basics of QoE, different assessment methods, and ways of measuring the user's perception of a given sequence. The second subsection is about metrics and their use in objective models' comparison. The chapter ends with the simulator's goals, which I want to achieve in this work.

1.1. Foundations of Quality of Experience

An increased use of Internet streaming services in recent years [1] makes QoE an interesting field of study to explore. To better understand what exactly is the Quality of Experience, it can be described as "the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and / or enjoyment of the application or service in the light of the user's personality and current state [4]".

When measuring the user satisfaction of the given media, it could be achieved in various ways. The most common and basic method to do it is by subjective tests. Although, there are also different methods like measuring user's feelings by objective models or the data-driven approach [5]. Regarding subjective assessments, there are a few factors that are necessary to cover. The first one is human resources. It is impossible to carry out a subjective test without people who are scoring the displayed sequences. The next factors are specially prepared multimedia sequences itself and in most cases isolated conditions but that can be omitted sometimes. The typical subjective test is accomplished when every sequence in the test is rated by the users. To obtain proper results from these tests, it is recommended to have between 4 and 40 people attending the scoring process [6]. Those numbers are not accidental, the required minimum is for the statistical procedures, while having more than 40 people are unnecessary and time-consuming. The ratings scale, in which the scoring process is made can be differently defined, but in this thesis, I will focus on ACR scale (Absolute Category Rating), presented in Table 1.1. In this method, the sequences are displayed one after another and between those sequences, there is a time for the user to judge the quality of a just seen sequence. That means, all sequences are evaluated independently [6]. Those discrete values from 1 to 5 describe possible scores for a given sequence, in which 1 is the lowest score, meaning that the user cannot stand the quality and probably will not watch again. In contrary, 5 is the highest score, meaning that the user is delighted by the sequence's quality.

Score	1	2	3	4	5
Meaning	Bad	Poor	Fair	Good	Excellent

Table 1.1. Possible quality scores in ACR scale [7]

Subsequently, to interpret the obtained results from subjective tests, I will use in this thesis, so-called MOS (Mean Opinion Score). This provides a simple way to establish the overall score of the sequence. It averages all users' scores for a specific media. It could be not the best method for every assessment to indicate the overall performance, but its advantage is the simplicity as well as popularity [8].

Certainly, there are other ways to measure user's satisfaction than MOS. For instance, there is a GoB (Good or Better) method [8]. It analyzes the probability of scores above the predefined threshold by the operator. It is beneficial regarding the diversity in users' scores during assessments. Another possible method is to measure QoE by Paired Comparison, in which the scoring process is simplified because the users no longer score on a predefined scale. This time, the subjects have to evaluate the quality only by comparing two sequences against each other, which of them prefers more [9]. The disadvantage of this method is that it takes more pairs of sequences to rate by each user, hereby it involves overall more time to process the entire assessment.

In this work, I decided to go with the MOS method. Although it needs to be mentioned, it has several disadvantages, for instance, if the quality of a given sequence is on the edge between two values of ACR scale, the subject may have difficulties when rating the sequence, but overall, it is a widely used method during subjective assessment. The subjective test itself does have some flaws like the time spent on rating media by users, which in some cases can be quite a long time. Another aspect worth mentioning is that people tend to score based on their individual characteristics. When considering a movie or speech sample, there is a possibility that the used, for example, coarse words in a video can affect the user rating and so the overall performance of the sequence. That could be a problem, if the test providers are more interested in the overall quality and not the content presented itself. The last crucial factor is the financial costs of those tests.

On the other hand, the objective QoE assessments are based on already done subjective tests and the models' overall performance is evaluated against this data. Having those models designed, it can overcome some flaws of the subjective method. Some media factors taken into consideration can predict the overall score of the sequence, which can speed up the rating process and give the draft version of the sequence score. The next advantage of objective models is that the execution is done without human attention to it. It just needs the data before creating such an objective model. According to that type of tests, there were developed methods and metrics, which can evaluate the objective model performance. The Recommendation provides three algorithms that would help to determine model efficiency against subjective data. Those metrics will cover three conditions of the objective model, which is accuracy, consistency, and linearity [3].

In the next subsection, I will show how the metrics are calculated in theory and how they would be used in the simulator during the assessments.

1.2. Description of metrics mentioned in ITU-T P.1401

The first described metric would be *rmse* (root mean square error), which will indicate the accuracy of the obtained results. For consistency, it is recommended to use *outlier ratio* algorithm. Finally, to measure data linearity, it is suggested to use *Pearson correlation coefficient R*. The accuracy will point out the difference between individual scores between the subjective and objective data. This method will show how close are the ratings from the prediction models to those obtained during subjective assessment. The consistency covers the number of well-prepared data to analyze by checking how many outlier points were obtained during assessment. The last parameter, data linearity, will measure how well the predicted scores fit to those obtained during the subjective test in terms of a linear relationship between them.

In the following subsections, I will represent in detail, how the metrics are calculated, where in the simulator's code are implemented, and how they will be compared across the simulation process.

1.2.1. Root mean square error

The rmse algorithm is recommended to use for measuring the data accuracy. To calculate the rmse, it is necessary to measure the predicted error (perror). In essence, perror is the difference between the observed MOS_o (subjective data) and predicted MOS_p (objective data), expressed in equation (1.2). Afterwards, it is possible to calculate the rmse with the equation (1.1).

$$rmse = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(perror(i)^2)}, \tag{1.1}$$

where N is the number of sequences involved in a given test and $perror$ is the predicted error given by:

$$perror(i) = MOS_o(i) - MOS_p(i). \tag{1.2}$$

Those mathematical operations are included in the simulator inside the `rmse.py` module, where the value of the rmse is calculated using the `calculate_rmse()`. The confidence intervals are measured in `confidence_interval_rmse()`. Finally, there is a comparison of rmse values between different objective models coded in `compare_models()`, which call `compare_rmse()` to do it.

1.2.2. Outlier ratio

In the case of calculating consistency of the data received from objective models, the recommended algorithm is the outlier ratio. It is a measure, which calculates the number of outlier points (false scores) to the total number of ratings for the sequence N, expressed in equation (1.3). The outlier point is constructed as a perror value, which exceeds 95% confidence intervals, which indicate the false score.

$$\text{Outlier Ratio} = \frac{\text{Total Number of Outliers}}{N} \tag{1.3}$$

The evaluation of the outlier ratio algorithm is coded in the simulator inside the `outlier_ratio.py` module, where the value is calculated using the `calculate_outlier_ratio()` function. The `confidence_interval_outlier_ratio()` function measures its confidence interval. Finally, there is a comparison of outlier ratio values between different objective models in `compare_models()`, which call `compare_outlier_ratio()`.

1.2.3. Pearson correlation coefficient R

The last parameter mentioned in this section, which corresponds to linearity, is Pearson correlation coefficient R. It allows measuring a linear fit of the objective data set to the subjective one. It is given by equation (1.4), where $X(i)$ denotes the subjective MOS of i sample, whereas $Y(i)$ denotes the objective MOS of i sample.

$$R = \frac{\sum(X(i) - \overline{X}) \times (Y(i) - \overline{Y})}{\sqrt{\sum(X(i) - \overline{X})^2 \times \sum(Y(i) - \overline{Y})^2}} \tag{1.4}$$

The Pearson correlation coefficient algorithm is included in the simulator inside the `pearson.py` script, where its value and confidence intervals are calculated in `calculate_pearson_coefficient()` and `confidence_interval_pearson_coefficient` functions, respectively. The parameter evaluation ends with comparing different models' values in `compare_models()`, which call `compare_pearson_coefficient()`.

1.3. Simulator objectives

In this thesis, I will evaluate the mentioned metrics from Recommendation for different objective models against subjective data. Therefore, I will try to determine if it is possible to differentiate the objective models by using those algorithms and if so, then which of the algorithms performs the best. For

this purpose, I implemented a simulator, which will be described in detail in the next chapter. I tried to design this simulator in such a way to give the user as much control as possible. The simulator consists of a few Python scripts that realize the simulation process, including generating subjective and objective data sets, evaluating metrics, and a comparison module. Moreover, the simulator can be enhanced by providing new metric modules that may explore new opportunities in objective assessment methods. There is also a possibility of providing different objective models than those, which I will use later on. Finally, the simulator should be prepared for a real data evaluation, both subjective and objective methods.

2. Implementation

In this chapter, I mainly focus on the simulation implementation, which includes the description of the environment, where the simulation runs in subsection 2.1. Afterwards, I explain the simulation process based on just one algorithm rmse, starting from subsection 2.2.1, regarding the input data of the simulator. Following the entire simulation procedure in subsection 2.2.2. Finally, this chapter ends in subsection 2.2.3, in which I describe the output matrix of the simulator.

2.1. Programming environment and libraries

Operating system — Windows 10 Home

Hardware — Intel Core i7-8750H 2.20GHz, 16 GB RAM, SSD drive

Software — PyCharm Professional 2020.2.3

Programming language — Python3 (version 3.6.7)

Python main libraries:

- `NumPy` (version 1.19.0)
- `SciPy` (version 1.14.5)
- `compress-pickle` (version 1.2.0)

2.2. Simulator description

The simulator consists of seven Python scripts:

- `generate_data.py`
- `objective_models.py`
- `rmse.py`

- `outlier_ratio.py`
- `pearson.py`
- `metrics_comparison.py`
- `simulation.py`

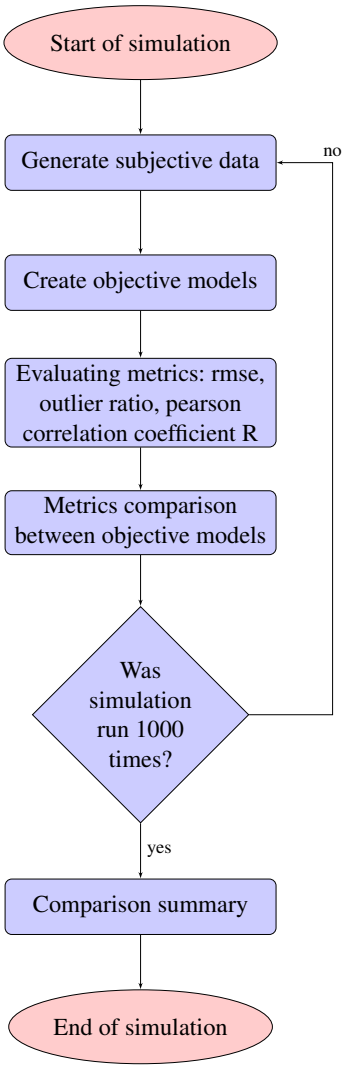

Figure 2.1. Flowchart of simulation process

In the flowchart 2.1 I present how the simulation process is carried out. It starts by running `simulation.py` where each module is coordinated and used to obtain the final result of the objective models' comparison. The entire simulation process is looped 1000 times to get statistically significant outcomes. The total duration of such simulation depends on the hardware specification. On my setup, it lasts around 6 seconds without saving results to files. Although, when running the simulation with saving results into files, it lasts around 10 seconds, which is the consequence of a good code optimization.

The optimization was based on stacking the results across multiple simulations and saving the results only once instead of saving them after each iteration. Moreover, in the beginning I was operating on `for` loops, which extended significantly the time duration. In place of those loops, I focus on doing matrices' operations that accelerate the simulation's time.

The entire simulation process is divided into three phases. The first one is preparing the input data as shown in Listing 2.1. In this phase, I simulate subjective and objective assessments, which will be analyzed. The Next step is to evaluate the algorithms from Recommendation for objective models. After having the metrics calculated, the simulation ends with the models' comparison matrix. Everything is included inside `simulation.py`, which is the main script of the simulator.

The default settings for each simulation are 100 different sequences and 30 subjects, hereby the normal distribution is used when generating subjective data. The Second setting, which will be common for simulations is how the objective models are prepared. I will take into consideration 5 models as presented in Listing 2.2. The first one is the perfect one, it predicts exactly the true quality of the sequence. Subsequently, other models are getting worse by applying some noise to them. This way, it is possible to distinguish the models in the correct order.

The next part of the simulation process is metrics evaluation. In this phase, each metric (rmse, outlier ratio, Pearson correlation coefficient R) is calculated for each objective method. Afterwards, the obtained values are compared between objective models' thus creating the comparison matrix. In this phase, a single simulation is completed, but it is not sufficient to evaluate the metrics performance. Therefore, the simulation process is repeated 1000 times. Following to the end of the simulation, the final matrix is being created in the `metrics_comparison.py` module. This output is a three-dimensional matrix containing objective models of comparison for each metric under evaluation.

Listing 2.1. Module *generate_data.py*

```
import numpy as np

number_of_voters = 30
number_of_sequences = 100
acr = [1, 2, 3, 4, 5]
psi_low, psi_high = 1.0, 5.0
delta_mean, delta_st_deviation = 0, 0.7
epsilon_low, epsilon_high = 0.3, 0.9

def uniform(low, high, size):
    return np.random.uniform(low, high, size)

def normal(mean, st_deviation, size):
    return np.random.normal(loc=mean, scale=st_deviation, size=size)

def scoring_process(psi, delta, epsilon):
    ones = np.ones(number_of_voters)
```

```
    extended_psi = np.outer(psi, ones)
    ratings = np.round(normal(mean=(extended_psi + delta), st_deviation=epsilon,
                              size=(number_of_sequences,number_of_voters)))
    ratings[ratings > acr[-1]] = acr[-1]
    ratings[ratings < acr[0]] = acr[0]
    measured_mos = np.mean(ratings, axis=1)
    return ratings, measured_mos

def generate_subjective_data():
    # sequence quality
    psi = uniform(psi_low, psi_high, number_of_sequences)
    # user imprecision
    delta = normal(delta_mean, delta_st_deviation, number_of_voters)
    # overall error
    epsilon = uniform(epsilon_low, epsilon_high, number_of_voters)
    # sequences scored by users - o_ijr
    ratings, measured_mos = scoring_process(psi, delta, epsilon)
    return psi, delta, epsilon, ratings, measured_mos
```

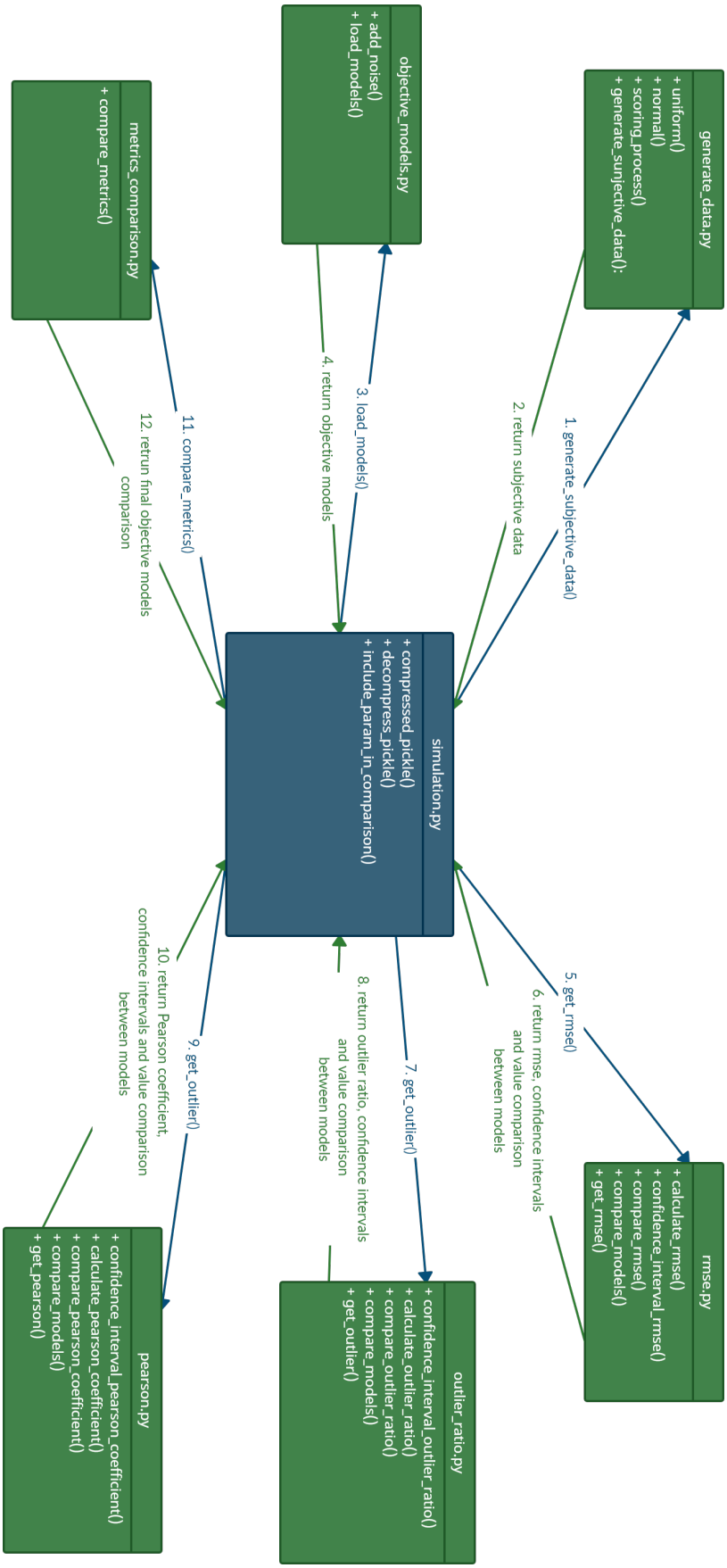

Figure 2.2. Concept map of the simulator's structure

2.2.1. Input data

At the beginning of the simulation, it is necessary to provide the input data, which will be analyzed across the simulation. In general, the input is considered as a measured MOS and predicted MOS, but in detail this is a subjective and objective data, respectively. For those two data sets, it was out of the scope of this thesis to prepare real data. Instead of it, I used a Theoretical Subject Model for simulating subjective data [10]. This model provides a method for generating realistic subjective scores and it is expressed by equation (2.1). o_{ijr} is an overall score for a given sequence, which is consisted of ψ_j — the true quality of a sequence, Δ_i — the user's imprecision in ratings, ϵ_{ijr} — the general scoring deviation [11].

$$O_{ij} = \psi_j + \Delta_i + \epsilon_{ij}, \tag{2.1}$$

where $O_{ij} \sim \mathcal{N}(\psi_j + \Delta_i, \epsilon_i)$. Afterwards, O_{ij} is rounded to U_{ij} from set $\{1, 2, 3, 4, 5\}$. It means not only rounding but also censoring is used to limit the obtained results to a specific set.

Those three factors are very intuitive because the overall score of the media can be simplified this way. The based score is the sequence quality itself, which is rated by the user and all of that is burdened with the error. The difference in user's rating can be caused by personal preferences or even tiredness after a long time of scoring different sequences. After generating overall ratings for sequences, the score is averaged to obtain the measured MOS per sequence. The implementation of generating subjective data following this algorithm is made in `generate_data.py`, presented in Listing 2.1.

The next step is to create different objective models that will be later considered in the comparison. Once more, it is out of the scope to create real objective models, for instance, models based on neural networks or machine learning. Therefore, I designed objective models that are based only on the true quality of each sequence ψ_j, which I know from the simulator setup. With typical subjective data, ψ_j is unknown, nevertheless for a simulation scenario it is reasonable to use ψ_j. It is a simple but effective way, which will help in metrics evaluation of those models against subjective data. The perfect method predicts the same values as the generated media quality ψ_j. The next models are worse than the previous one by applying more noise to the method's prediction scores. The noise is considered as a random number from a normal distribution characterized by $\mathcal{N}(0, 0.25)$. In total, I will analyze 5 different prediction models gathered in a simple array as shown in Listing 2.2.

Listing 2.2. Module *objective_models.py*

```
def add_noise(data, mean, st_deviation, size):
    model = data + normal(mean, st_deviation, size)
    model[model > acr[-1]] = acr[-1]
    model[model < acr[0]] = acr[0]
    return model
```

```
def load_models(psi, number_of_samples):
    models = []
    predicted_mos_best = psi
    models.append(predicted_mos_best)
    predicted_mos_good = add_noise(psi, 0, 0.25, number_of_samples)
    models.append(predicted_mos_good)
    predicted_mos_bad = add_noise(psi, 0, 0.5, number_of_samples)
    models.append(predicted_mos_bad)
    predicted_mos_worse = add_noise(psi, 0, 0.75, number_of_samples)
    models.append(predicted_mos_worse)
    predicted_mos_the_worst = add_noise(psi, 0, 1, number_of_samples)
    models.append(predicted_mos_the_worst)
    return models
```

An important issue to mention here is the order in which the models are appended to the `models` array. The model's index in this array will distinguish each one from another in the comparison matrix at the end of the simulation. The generation of objective methods is coded inside `objective_models.py`. The final transformation is made in the main script `simulation.py`, where a `Python array` with the models is transformed into `NumPy` object — matrix.

To summarize, I generate subjective data by using the Theoretical Subject Model. I artificially propose objective assessment methods just to be able to detect if the recommended metrics work properly and establish which of them perform better. In the next subsection, I will explain the metric evaluation process based on the rmse parameter.

2.2.2. Simulation process

Having a proper data set up, it is time to calculate parameters (rmse, outlier ratio, Pearson correlation coefficient R) for each model. This metric evaluation is called in the `simulation.py` script in the main `for` loop as shown in Listing 2.3. Although, all three metrics are calculated in different Python scripts.

Listing 2.3. Calling metrics' evaluation functions

```
rmse.get_rmse(measured_mos, predicted_mos, all_rmse, all_rmse_ci,
    ↪ all_rmse_compared_arr)
outlier.get_outlier(measured_mos, predicted_mos, ratings, all_outlier,
    ↪ all_outlier_ci, all_outlier_ratio_compared_arr)
pearson.get_pearson(measured_mos, predicted_mos, all_pearson, all_pearson_ci,
    ↪ all_pearson_compared_arr)
```

The parameters' implementation is very similar to each other, that is why in this section, only one parameter — rmse will be explained. The only difference in calculating other metrics is the mathematical operations specific for a given algorithm. To obtain the correct results and do not have any errors, the

functions shown in Listing 2.3 must be called with appropriate arguments' order and in a specific data format.

When considering the rmse algorithm, the arguments are a measured MOS, which is a `NumPy` matrix with dimensions 1×"the number of sequences considered in the test." It provides results from the subjective assessment. The next argument is an predicted MOS. It is a `NumPy` matrix again, but this time with dimensions 1×"number of sequences"×"number of objective models", which stores the results for every objective method. Furthermore, it is crucial to adjust the first two dimensions of the above matrices. Otherwise, the simulator will arise an error and the final score will not be computed. The rest of the arguments are simple arrays containing `NumPy` matrices. The `all_rmse` contains every calculated rmse in the simulation, `all_rmse_ci` contains confidence intervals for all rmse values. The `all_rmse_compared_arr` is a matrix of compared rmse values between models under evaluation.

Next operations are repeated for each metric. The flow of operations is as follows:

- call get method for the parameter (Listing 2.7)
- calculate the parameter's value (Listing 2.4)
- calculate the value's confidence interval (Listing 2.5)
- compare the values across different objective models (Listing 2.6)

The mathematical operations inside the functions are implemented in accordance with the Recommendation ITU-T P.1401 and are presented below [3]. Out of those functions, the essential is values comparison module. Based on results from it, I will decide if it helps to determine the best objective model and which of the metrics performs the best. The comparing process relies on the hypothesis that H_0 assumes there is no difference between values. Therefore, H_1 considers that the difference exists, and it is statistically significant [3]. In consideration of the code implementation, the comparison module will create a matrix, which will contain the values 1, 0 or -1. Supposing that the result of comparing two values makes the H_1 hypothesis true, then two variants can happen. One of them is that the first model considered in comparison achieved better results. In this case, the simulator assigns the value 1. Otherwise, it assigns -1 in case of worse performance than the second model. On the other hand, if H_0 is true, then the comparison results with 0 value in the matrix cell, meaning the results are statistically equal. Listing 2.6 presents the mentioned operations.

Listing 2.4. Function calculating rmse value

```
def calculate_rmse(measured_mos, predicted_mos):
    number_of_samples = measured_mos.shape[0]
    perror = measured_mos - predicted_mos
    perror_sum = np.sum((perror**2), axis=0)
    rmse = math.sqrt( perror_sum / (number_of_samples - 1))
    return rmse
```

Listing 2.5. Function calculating rmse confidence interval

```
def confidence_interval_rmse(rmse, n, number_of_samples):
    lower_bound = rmse*math.sqrt(n) / math.sqrt(st.chi2.interval(alpha=0.95, df=(
        ↪ number_of_samples-4))[1])
    upper_bound = rmse*math.sqrt(n) / math.sqrt(st.chi2.interval(alpha=0.95, df=(
        ↪ number_of_samples-4))[0])
    interval = np.vstack([lower, upper])
    return interval
```

Listing 2.6. Function comparing different rmse values

```
def compare_rmse(rmse_1, rmse_2, threshold):
    if rmse_1 > rmse_2:
        q = math.pow(rmse_1, 2) / math.pow(rmse_2, 2)
        if q >= threshold:
            return -1
        else:
            return 0
    else:
        q = math.pow(rmse_2, 2) / math.pow(rmse_1, 2)
        if q >= threshold:
            return 1
        else:
            return 0


def compare_models(rmse_param, threshold, number_of_metrics):
    compared = np.zeros((number_of_metrics, number_of_metrics))
    for x in range(0, number_of_metrics):
        for y in range(0, number_of_metrics):
            if x < y:
                comparing_result = compare_rmse(rmse_param[x], rmse_param[y],
                    ↪ threshold)
                compared[x, y] = comparing_result
    return compared
```

Listing 2.7. Function *get_rmse*, which coordinate other functions

```
def get_rmse(measured_mos, predicted_mos, all_rmse, all_rmse_ci,
    ↪ all_rmse_compare_arr):
    number_of_samples = predicted_mos.shape[1]
    number_of_metrics = predicted_mos.shape[2]

    rmse_arr = []
    rmse_ci_arr = []
    n = number_of_samples - 4
    for x in range(number_of_metrics):
        rmse_arr.append(calculateRMSE(measured_mos, predicted_mos[0, :, x]))
```

```
        rmse_ci_arr.append(confidenceIntervalRMSE(
          rmse_arr[-1], n, number_of_samples))
    rmse = np.hstack(rmse_arr)
    rmse_ci = np.hstack(rmse_ci_arr)

    rmse_threshold = st.f.ppf(0.95, (number_of_samples - 4),
      (number_of_samples - 1))
    rmseCompared = compareModels(rmse, rmse_threshold, number_of_metrics)

    all_rmse.append(rmse)
    all_rmse_ci.append(rmse_ci)
    all_rmse_compare_arr.append(rmseCompared)

    return all_rmse, all_rmse_ci, all_rmse_compare_arr
```

At this point, the simulation ends with three matrices containing the objective methods' comparison. Each of them regards a different metric. This is not the best approach to display the simulator's outcome. That is why I created the `metrics_comparison.py` module, in which the matrices are combined into one final matrix. Firstly, the metrics that will be under further evaluation are appended to a `Python list`. Afterwards, it is transformed to a `NumPy` matrix.

It is crucial to mention here that the simulation run once is not sufficient to come to the conclusion how well the metrics differentiate objective models. That is why the entire simulation should be run multiple times. In this thesis, it is considered running it 1000 times. Each time, new subjective and objective data are generated. In the next section, I present computing the final comparison matrix, which summarizes all of the 1000 simulations.

2.2.3. Output data

The matrices are originally computed in each simulation iteration as an upper triangular with possible values of 1, 0, -1. In the `metrics_comparison.py` module, presented in Listing 2.8, follows an extraction of two matrices from each comparison matrix. The first one contains only positive values and the second negatives values. Subsequently, the second matrix with negative values is transposed, and it becomes a lower triangular matrix. With this approach, divided matrices are concatenated to combine the final matrix, in which values above the main diagonal indicate in how many simulations the objective model read in line was better than the model read in columns. Contrarily, the values below the main diagonal indicate how many times the objective model read in columns was better than the model read in lines. The difference between the two models at the intersection of a column with a row and the intersection of a row with a column indicates the percentage of how many times those models were statistically equal.

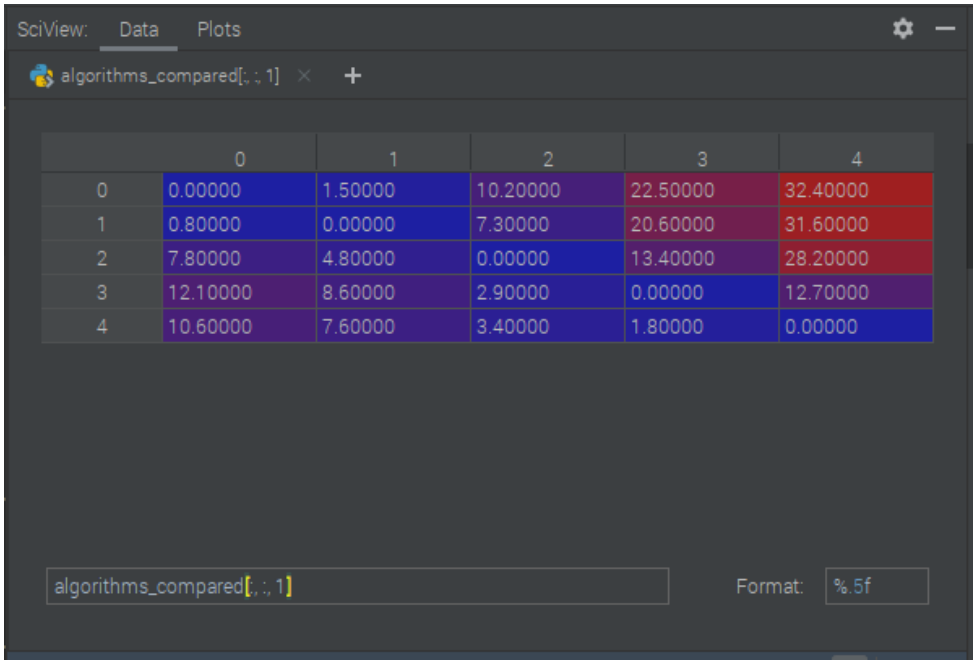

Figure 2.3. Exemplary output matrix

After each metric matrix is processed, all of them are joined into one matrix along the third dimension. This matrix is "number of objective models"×"number of objective models"×"number of metrics", which stores the performance of objective models against subjective data using recommended algorithms across the entire simulation. As mentioned, the index in the third dimension reflects the parameter index in `metrics` array, hereby it is known what parameter's summary is displayed on each matrix. For instance, the matrix presented in Figure 2.3 display `metrics_compared[:, :, 1]`. The `metrics` array order is `(rmse, outlier, pearson)`, which indicates metrics indexes. Therefore, the matrix display in Figure 2.3 is outlier ratio.

Listing 2.8. Algorithm summarizing all parameters

```
def compare_metrics(number_of_models, metrics):
    if number_of_models > 1:
        stacked_metrics = np.stack(metrics, axis=2)
        number_of_simulations = stacked_metrics.shape[-1]

        metrics_summary = []
        for matrix in metrics:
            matrix_upper = np.count_nonzero(matrix == 1, axis=2) /
                ↪ number_of_simulations * 100
            matrix_lower = np.count_nonzero(matrix == -1, axis=2) /
                ↪ number_of_simulations * 100
            matrix_lower = matrix_lower.T
            metrics_summary.append(matrix_lower + matrix_upper)
        results_matrix = np.stack(metrics_summary, axis=2)
        return results_matrix
    else:
        print("WARNING ! In order to compare objective models, there needs to be at
            ↪ least two of them")
```

3. Results

In this chapter, I present different test scenarios, where the metrics evaluate objective models under different circumstances. In essence, each test consists of three parts. The first one is the outline of the test, where I describe the steps to prepare the simulator for a specific test. Secondly, I present the results in matrices for each evaluated metric. Finally, I summarize the test by commenting the results and draw some conclusions. The initial test is about metrics evaluation against simulated real subjective data. The second test is considering users that score similarly. Afterwards, there is a test, where the simulator tries to handle a badly prepared input data set. The next test is about very accurate objective models and the attempt to see how big needs to be the difference between objective algorithms' scores to distinguish them. In the end, I carry out the test, where users can rate sequences in a continuous scale and check how this fact will impact the results.

3.1. Simulate real subjective test

In this scenario, the input data is generated to simulate a real-life subjective tests and obtain results from it. To achieve that, 100 random numbers were generated from a uniform distribution to define different quality scores for each sequence. Those numbers are generated within the discrete ACR scale 1 to 5. The uniform distribution is used because it allows receiving numbers equally distributed and not accumulated in a specific area, that is, why the sequences' quality will be diverse and not distorted by any influencing factor [12]. The next parameter of the Theoretical Subject model is user imprecision. I selected a normal distribution for this parameter because most of the surrounding us properties can be explained with this distribution. Secondly, I assumed the fact that the majority of users scores similarly. Only a few of them rate sequences differently than the others. This distribution is described by the mean and the standard deviation [12]. In this case, I assumed the mean equals to 0 because it is undesirable to influence the objective sequence score. The standard deviation is set to 0.7 to influence users' liability in the rating process. The last parameter is the overall error, both the sequences' quality and users' bias. This is defined by a uniform distribution, with the low value of 0.3 and the high of 0.9.

To summarize:

- sequence quality $\psi \sim \mathcal{U}(1, 5)$
- user imprecision $\Delta \sim \mathcal{N}(0, 0.7)$
- error $\epsilon \sim \mathcal{U}(0.3, 0.9)$

In this scenario, objective models were designed as presented in Listing 2.2 inside the `load_models` function. More precisely, objective models are built as follows best $= \psi$, good $= \psi + \mathcal{N}(0, 0.25)$, bad $= \psi + \mathcal{N}(0, 0.5)$, worse $= \psi + \mathcal{N}(0, 0.75)$, the worst $= \psi + \mathcal{N}(0, 1)$. Essentially, what is expected to happen from this test is to check if the recommended metrics mentioned in this thesis allow to indicate the best objective model. Moreover, whether the metrics arrange the rest of them in the correct order. The results are presented below in Tables 3.8, 3.2, 3.3.

Table 3.1. Summary of rmse comparison between objective models given in % across 1000 simulations in test case 3.1

model	best	good	bad	worse	the worst
best	x	99.1	100.0	100.0	100.0
good	0.0	x	99.9	100.0	100.0
bad	0.0	0.0	x	94.2	100.0
worse	0.0	0.0	0.0	x	72.7
the worst	0.0	0.0	0.0	0.0	x

Table 3.2. Summary of outlier ratio comparison between objective models given in % across 1000 simulations in test case 3.1

model	best	good	bad	worse	the worst
best	x	91.2	100.0	100.0	100.0
good	0.0	x	97.8	99.9	100.0
bad	0.0	0.0	x	73.3	98.6
worse	0.0	0.0	0.0	x	44.3
the worst	0.0	0.0	0.0	0.0	x

Table 3.3. Summary of Pearson correlation coefficient R comparison between objective models given in % across 1000 simulations in test case 3.1

model	best	good	bad	worse	the worst
best	x	100.0	100.0	100.0	100.0
good	0.0	x	99.9	100.0	100.0
bad	0.0	0.0	x	81.6	99.5
worse	0.0	0.0	0.0	x	48.9
the worst	0.0	0.0	0.0	0.0	x

The outcomes show that each metric indicates the correct order of models' performance from the best to the worst. In the case of better performing models (those characterized with zero or small noise in predicting the sequences' quality), the best performing algorithm is Pearson correlation coefficient R, which focuses on data linearity, so there is no surprise that performs better than the others. Although, regarding the comparison between worse efficient models, the verification of data linearity accomplishments are lower than the rmse algorithm. The rmse allows to distinguish the better model even 238 times more than R coefficient when comparing *worse* with the *the worst* model. In this test, it is clearly visible that the best performing algorithm is rmse followed by Pearson Correlation coefficient R and ending with the outlier ratio.

Since each run of the simulation is based on random numbers from different distributions, it is nearly impossible to restore the same values of this test. Therefore, the results described above are stored using `compressed-pickle` library in *output_data/scenario_1* on GitHub repository. The data can be decoded using the same library and be analyzed again.

3.2. Each user scores identically

In this test, I assumed the case in which all users rate identically each sequence or the difference in scores is very low. This type of test is unlikely to happen in reality, but I have wondered how metrics will proceed under those circumstances. The generation of the subjective test's data is similar to the previous scenario 3.1 with only the difference in the parameters of normal and uniform distribution. I assumed that user imprecision is almost equal to 0 and the overall error is equal to 0, speculating that both sequences' quality and users are infallible. The objective models' parameters remain the same as in scenario 3.1. The results are presented below in Tables 3.4, 3.5, 3.6.

To summarize:

- sequence quality $\psi \sim \mathcal{U}(1, 5)$
- user imprecision $\Delta \sim \mathcal{N}(0, 0.1)$
- error $\epsilon \sim \mathcal{U}(0, 0)$

Table 3.4. Summary of rmse comparison between objective models given in $\%$ across 1000 simulations in test case 3.2

model	best	good	bad	worse	the worst
best	x	100.0	100.0	100.0	100.0
good	0.0	x	99.9	100.0	100.0
bad	0.0	0.0	x	94.1	99.9
worse	0.0	0.0	0.0	x	72.6
the worst	0.0	0.0	0.0	0.0	x

Table 3.5. Summary of outlier ratio comparison between objective models given in $\%$ across 1000 simulations in test case 3.2

model	best	good	bad	worse	the worst
best	x	5.2	32.9	54.2	65.2
good	1.1	x	17.1	44.1	60.7
bad	0.2	0.0	x	11.5	23.3
worse	0.0	0.0	0.4	x	7.7
the worst	0.1	0.0	0.1	0.7	x

Table 3.6. Summary of Pearson correlation coefficient R comparison between objective models given in $\%$ across 1000 simulations in test case 3.2

model	best	good	bad	worse	the worst
best	x	99.9	100.0	100.0	100.0
good	0.0	x	99.3	100.0	100.0
bad	0.0	0.0	x	78.0	99.3
worse	0.0	0.0	0.0	x	46.2
the worst	0.0	0.0	0.0	0.0	x

The results for Pearson Correlation coefficient R and rmse are not surprising, each metric indicates the correct order for the objective models. What is interesting is that the outlier ratio is capable of pointing out a better objective model only between worse performing ones. Between the best model and the other

well performing, the difference is not that significant. Furthermore, in some simulations, the objectively worse model is indicated with better efficiency than better models. Moreover, when each user will score exactly the same (Δ equal to $\mathcal{N}(0,0)$), it is even possible that the correct order is distorted as presented in table 3.7. The reason for such a situation is that sequence quality is measured in a continuous scale. This way, it is possible to have the sequence's quality equal to, for instance, 2.5 and the user by definition cannot rate the sequence exactly 2.5. Furthermore, the user can assign the score in discrete scale 2 or 3 in this case. The outlier ratio operates on the perror value, the difference between the observed MOS and the predicted MOS. That is why, most of the time there would be a significant statistical difference when comparing two models. The results from such comparisons favor worse objective models because those models are more likely to indicate the correct sequence's quality score.

Table 3.7. Summary of outlier ratio comparison between objective models given in $\%$ across 1000 simulations in enhanced test case 3.2

model	best	good	bad	worse	the worst
best	x	0.0	0.0	0.0	0.0
good	87.6	x	0.4	0.1	0.0
bad	99.4	20.5	x	0.6	0.3
worse	99.9	32.7	6.3	x	1.4
the worst	99.8	43.3	9.1	4.0	x

3.3. Poorly prepared subjective data

Earlier tests took into consideration well-prepared data for subjective assessment. Perfect conditions are probably the case that everyone wants to achieve. In reality, when carrying out a subjective test, there are several points in which the overall score of each user can be distorted. For instance, the subject could be tired, annoyed through the day or simply the conditions are not comfortable to rate sequences. When those conditions are met, there is more likely that users will rate the sequences irregularly and the MOS could differ. That is why, in this test I try to check how well the metrics will perform evaluating objective models. Whether it is possible to indicate the best performing model against subjective data. The objective methods are designed exactly as in the previous tests.The test conditions for the simulator are described below and later on the results are presented.

To summarize:

- sequence quality $\psi \sim \mathcal{U}(1, 5)$
- user imprecision $\Delta \sim \mathcal{N}(0, 2)$
- error $\epsilon \sim \mathcal{U}(0.3, 0.9)$

Table 3.8. Summary of rmse comparison between objective models given in $\%$ across 1000 simulations in test case 3.3

model	best	good	bad	worse	the worst
best	x	8.6	89.1	99.5	100.0
good	0.0	x	58.0	97.5	99.7
bad	0.0	0.0	x	60.7	97.5
worse	0.0	0.0	0.0	x	46.8
the worst	0.0	0.0	0.0	0.0	x

Table 3.9. Summary of outlier ratio comparison between objective models given in $\%$ across 1000 simulations in test case 3.3

model	best	good	bad	worse	the worst
best	x	14.8	73.6	93.7	98.0
good	0.1	x	47.6	89.2	97.3
bad	0.0	0.1	x	46.9	87.8
worse	0.0	0.0	0.0	x	36.9
the worst	0.0	0.0	0.0	0.2	x

Table 3.10. Summary of Pearson correlation coefficient R comparison between objective models given in $\%$ across 1000 simulations in test case 3.3

model	best	good	bad	worse	the worst
best	x	100.0	100.0	100.0	100.0
good	0.0	x	99.4	100.0	100.0
bad	0.0	0.0	x	80.6	99.6
worse	0.0	0.0	0.0	x	49.8
the worst	0.0	0.0	0.0	0.0	x

The matrices show some inconsistencies in the obtained results. Both algorithms' outlier ratio and rmse present similar performance against subjective data. However, Pearson correlation coefficient R persists almost perfect results like in the test 3.1 with simulated real subjective data. I suppose this

situation is because both rmse and outlier ratio are based on the calculated perror factor. With badly prepared subjective data, it results in a significant difference between user scores and predicted scores by the objective methods. Therefore, those two algorithms are not quite enough to determine the best possible model. This confirms the statement from Recommendation ITU-T P.1401, that the subjective assessments' method should be well prepared to omit misjudgment in evaluating objective methods' accuracy [3]. The R coefficient obtained much more accurate results. This parameter is not based on error as much as the previous parameters, hereby this algorithm without problem indicates the correct order of models' performance.

3.4. Comparison of precise objective models

In this test, I want to check how significant needs to be a difference between objective models to indicate which model is better according to the evaluating metrics. In reality, it could be the case when there is a need to differentiate nearly perfectly efficient objective prediction models. To keep the test simple, a subjective data set is generated accordingly to section 3.1. The only difference is in the created objective models. This time, the difference is very low, the standard deviation of each model is increased constantly every 0.05, starting from 0.1 to 0.25. The models were designed as follows best $= \psi$, good $= \psi + \mathcal{N}(0, 0.1)$, bad $= \psi + \mathcal{N}(0, 0.15)$, worse $= \psi + \mathcal{N}(0, 0.2)$, the worst $= \psi + \mathcal{N}(0, 0.25)$ The function of generating those model is presented below on Listing 3.1.

Listing 3.1. Creating very accurate objective models

```
def load_models(psi, number_of_samples):
    models = []
    predicted_mos_best = psi
    models.append(predicted_mos_best)
    predicted_mos_good = add_noise(psi, 0, 0.1, number_of_samples)
    models.append(predicted_mos_good)
    predicted_mos_bad = add_noise(psi, 0, 0.15, number_of_samples)
    models.append(predicted_mos_bad)
    predicted_mos_worse = add_noise(psi, 0, 0.2, number_of_samples)
    models.append(predicted_mos_worse)
    predicted_mos_the_worst = add_noise(psi, 0, 0.25, number_of_samples)
    models.append(predicted_mos_the_worst)
    return models
```

To summarize:

- sequence quality $\psi \sim \mathcal{U}(1, 5)$
- user imprecision $\Delta \sim \mathcal{N}(0, 0.7)$
- error $\epsilon \sim \mathcal{U}(0.3, 0.9)$

Table 3.11. Summary of rmse comparison between objective models given in $\%$ across 1000 simulations in test case 3.4

model	best	good	bad	worse	the worst
best	x	11.4	69.6	93.2	98.7
good	0.0	x	16.5	69.9	92.4
bad	0.0	0.0	x	25.5	73.4
worse	0.0	0.0	0.0	x	26.8
the worst	0.0	0.0	0.0	0.0	x

Table 3.12. Summary of outlier ratio comparison between objective models given in $\%$ across 1000 simulations in test case 3.4

model	best	good	bad	worse	the worst
best	x	6.1	25.0	63.4	90.9
good	0.0	x	11.0	38.6	77.4
bad	0.0	0.0	x	15.5	49.8
worse	0.0	0.0	0.0	x	21.2
the worst	0.0	0.0	0.0	0.0	x

Table 3.13. Summary of Pearson correlation coefficient R comparison between objective models given in $\%$ across 1000 simulations in test case 3.4

model	best	good	bad	worse	the worst
best	x	29.8	96.7	100.0	100.0
good	0.0	x	22.0	87.7	99.9
bad	0.0	0.0	x	23.0	81.6
worse	0.0	0.0	0.0	x	20.3
the worst	0.0	0.0	0.0	0.0	x

As shown in Tables 3.11, 3.12, 3.13 each metric determined the best model correctly, however, based on just very few simulations. Objective models differing by a standard deviation of approximately 0.1 are very difficult to distinguish according to those metrics. The best performing algorithm is Pearson

correlation coefficient R, however it indicates the best model only during 298 simulations out of 1000, which is not an outstanding result. The rest of the models, where the difference in the standard deviation is higher than 0.1, are easily compared and arranged in the correct order. The worse performing algorithm is the outlier ratio, where the differences between almost all models compared are not significant.

3.5. The users score in continuous scale

Normally, users score within the discrete ACR scale as presented in Figure 3.1, having only five choices to describe their feelings about the media. In this test, I will check what happens if the user could rate sequences in a continuous scale as shown on Figure 3.2. The parameters for the Subject model remain the same as in scenario 3.1 as well as the prepared objective models. The only difference is that users' scores during subjective assessments are not rounded to a discrete scale of values $\{1, 2, 3, 4, 5\}$. The results of this test are presented below in Tables 3.14, 3.15, 3.16.

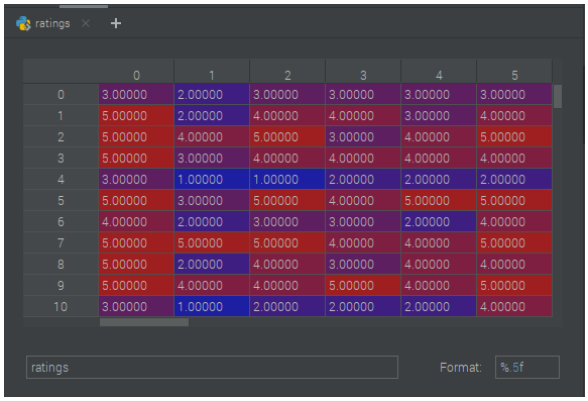

Figure 3.1. Snippet of users' individual ratings in discrete scale

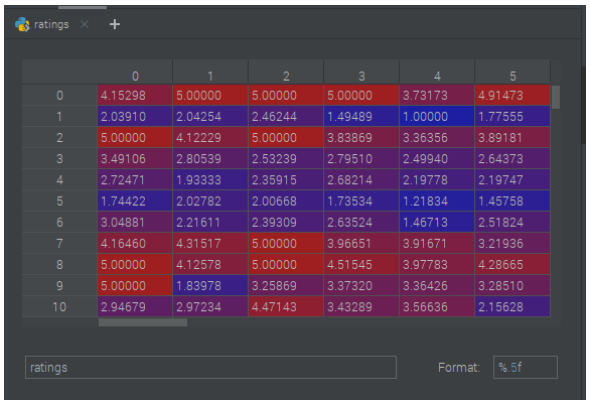

Figure 3.2. Snippet of users' individual ratings in continuous scale

To summarize:

- sequence quality $\psi \sim \mathcal{U}(1, 5)$
- user imprecision $\Delta \sim \mathcal{N}(0, 0.7)$
- error $\epsilon \sim \mathcal{U}(0.3, 0.9)$

Table 3.14. Summary of rmse comparison between objective models given in % across 1000 simulations in test case 3.5

model	best	good	bad	worse	the worst
best	x	98.4	100.0	100.0	100.0
good	0.0	x	99.4	100.0	100.0
bad	0.0	0.0	x	93.3	99.9
worse	0.0	0.0	0.0	x	70.0
the worst	0.0	0.0	0.0	0.0	x

Table 3.15. Summary of outlier ratio comparison between objective models given in % across 1000 simulations in test case 3.5

model	best	good	bad	worse	the worst
best	x	94.1	100.0	100.0	100.0
good	0.0	x	97.7	100.0	100.0
bad	0.0	0.0	x	73.1	97.8
worse	0.0	0.0	0.0	x	44.5
the worst	0.0	0.0	0.0	0.0	x

Table 3.16. Summary of Pearson correlation coefficient R comparison between objective models given in % across 1000 simulations in test case 3.5

model	best	good	bad	worse	the worst
best	x	100.0	100.0	100.0	100.0
good	0.0	x	99.7	100.0	100.0
bad	0.0	0.0	x	80.6	99.9
worse	0.0	0.0	0.0	x	46.6
the worst	0.0	0.0	0.0	0.0	x

The presented matrices show similar results as the first test about real subjective data. All metrics arranged objective models in the correct order. Once again, the rmse algorithm performs the best, followed by Pearson correlation coefficient R and outlier ratio. It is visible that there is no difference whether the user rates in a discrete 5-level scale or within it.

The results of each test are stored on GitHub repository mention in Appendix of this thesis. The path to those results is `output_data/scenario_X`, where `X` refers to the test's number in which it was listed in this chapter. For instance, the results of *"3.5. Users scoring in continuous scale"* can be found in `output_data/scenario_5` directory.

4. Summary

This is the last chapter in this thesis, in which I am extracting the conclusions of different tests from the Chapter Results in subsection 4.1. Following that in subsection 4.2, I will point out some further development directions, by which the simulator can be enhanced. Accordingly, in the future, it may discover some new findings in the topic of objective quality prediction methods.

4.1. Conclusions

The purpose of the presented thesis was to compare different algorithms comparing the objective QoE prediction models. To achieve it, I designed a simulator written in Python, which includes a comparison of different models using recommended metrics against simulated subjective data. The data considered in the simulating process were proposed as statistical models. The Theoretical Subject Model was used to receive data for the subjective assessments. On the other hand, the objective models' results were artificially created to provide information regarding their evaluating metrics from the Recommendation. The simulation results in a three-dimensional matrix containing the comparison outcome. This way, by carrying out different tests and analyzing the output matrices, I could determine which algorithm from the Recommendation performs the best. Accordingly to the executed tests, I came to conclusion that the best algorithm to measure the objective methods' efficiency is the Pearson correlation coefficient R, followed by the rmse and ending with the outlier ratio. In the case of models' evaluation against well-carried out subjective assessments, the rmse algorithm performs slightly better than the R coefficient. However, in general, the Pearson coefficient determines a better model in more accomplished simulations. The final choice, which metric to use to compare objective methods can be difficult, but in the majority of cases the Pearson correlation coefficient R is the best approach. The entire simulation was based on the theory behind statistical evaluation of QoE subjective and objective methods. It is still possible to expand this simulator by providing different data than proposed in this thesis. In the next section, I present a few possible simulation's growth directions.

4.2. Further development

In order to enhance and develop the current version of the simulator, there is an opportunity to create different than recommended metrics modules and calculate each of them for the objective models. As a consequence, there is a possibility that it will reveal that a different parameter can better indicate the model's performance and will replace the recommended metric. To add such a parameter module, it is resourceful to base on current Python scripts, for instance, `outlier_ratio.py` or `rmse.py`. Essentially, such modules should have methods that calculate the value of the parameter, its confidence intervals, and contain a comparing algorithm. All of that should be wrapped in `get_parameter` method that is called inside `simulator.py` as shown in Listing 2.3. The final step is to include this parameter in the overall comparison, which is presented below. It is important to notice at which index a new parameter is added to the `metrics` array to interpret the final matrix.

Listing 4.1. Including parameter evaluation in the final comparison

```
metrics = []
all_rmse, all_rmse_ci, all_rmse_compared, metrics = include_param_in_comparison(
    'rmse', all_rmse, all_rmse_ci, all_rmse_compared_arr, metrics)
all_outlier, all_outlier_ci, all_outlier_ratio_compared, metrics =
    ↪ include_param_in_comparison(
    'outlier', all_outlier, all_outlier_ci, all_outlier_ratio_compared_arr, metrics)
all_pearson, all_pearson_ci, all_pearson_compared, metrics =
    ↪ include_param_in_comparison(
    'pearson', all_pearson, all_pearson_ci, all_pearson_compared_arr, metrics)

metrics_compared = compare_metrics(number_of_models, metrics)
```

If the user of the simulator wants to check and test different objective models than suggested in this thesis, it is also accessible to change. The only modification is in the `objective_models.py` module and is based on appending a different model that predicts the score for each sequence considered in the simulation process. An Alternative way to accomplish this type of modification is to change the already existed `predicted_mos_arr` array in the `simulation.py` module by appending a new model, right before transforming this array into `NumPy` object.

Another possible development of this simulator is to seize the opportunity of real data evaluation. This type of analysis could show how well actual objective models perform and also use of realistic subjective tests results to prove the accomplishments of this simulator and evaluation of the mentioned metrics. In order to conduct such analysis, it is necessary to modify the number of simulation iteration. Furthermore, a removal of a few lines from `simulation.py` in main loop, which are responsible for generating subjective data, loading objective models and finally `all_factor_arr` at the end of the main loop and outside it in the same script. After those operations, the last point is to provide subjective and objective data that would be considered in the simulation process.

Bibliography

[1] *Cisco Annual Internet Report (2018–2023) White Paper*. URL: *https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html* (visited on 2021-01-03).

[2] ETSI TR 102 643 V1.0.1. *Human Factors (HF); Quality of Experience (QoE) requirements for real-time communication services*. 2009-12. URL: *https://www.etsi.org/deliver/etsi_tr/102600_102699/102643/01.00.01_60/tr_102643v010001p.pdf* (visited on 2021-01-03).

[3] ITU-T Recommendation P.1401. "Methods, metrics and procedures for statistical evaluation, qualification and comparison ofobjective quality prediction models". In: (2012).

[4] Kjell Brunnström et al. *Qualinet White Paper on Definitions of Quality of Experience*. Qualinet White Paper on Definitions of Quality of Experience Output from the fifth Qualinet meeting, Novi Sad, March 12, 2013. Mar. 2013. URL: *https://hal.archives-ouvertes.fr/hal-00977812*.

[5] Ruochen Huang, Xin Wei, Liang Zhou. "A survey of data-driven approach on multimedia QoE evaluation". In: (2018).

[6] ITU-T Recommendation P.910. "Subjective video quality assessment methods for multimedia applications". In: (2008).

[7] Q. Huynh-Thu et al. "Study of Rating Scales for Subjective Quality Assessment of High-Definition Video". In: *IEEE Transactions on Broadcasting* 57.1 (2011), pp. 1–14. DOI: *10.1109/TBC.2010.2086750*.

[8] T. Hoßfeld et al. "No silver bullet: QoE metrics, QoE fairness, and user diversity in the context of QoE management". In: *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*. 2017, pp. 1–6. DOI: *10.1109/QoMEX.2017.7965671*.

[9] J. Lee, F. De Simone, and T. Ebrahimi. "Subjective Quality Evaluation via Paired Comparison: Application to Scalable Video Coding". In: *IEEE Transactions on Multimedia* 13.5 (2011), pp. 882–893. DOI: *10.1109/TMM.2011.2157333*.

[10] Lucjan Janowski, Margaret Pinson. "The Accuracy of Subjects in a Quality Experiment: A Theoretical Subject Model". In: (2015).

[11] Lucjan Janowski, Jakub Nawała, Werner Robitza, Zhi Li, Lukáš Kasula, Krzysztof Rusek. *Notation for Subject Answer Analysis*. 2019-03-14. URL: *https://arxiv.org/abs/1903.05940* (visited on 2021-01-03).

[12] *Statistical Distributions*. URL: *http://people.stern.nyu.edu/adamodar/New_Home_Page/StatFile/statdistns.htm* (visited on 2020-12-20).

List of Tables

Appendix

I store the simulation code on my personal GitHub repository, which will be continuously maintained. The service GitHub is a version control system, which I have chosen for several reasons. The main points are that it is a common solution used worldwide. This system allows holding the entire history of modification in the project. This way, it is easy to take care of the code maintenance. Moreover, it has a collaborating module, which provides an opportunity for enhancing the current version of the simulator by the thesis-related enthusiasts in the future.

Link to the repository: engineering-thesis

Repository stores:

- source code of the simulator
- results from the presented tests in Chapter 3
- list of required libraries