Quality of Experience Prediction for HTTP YouTube Traffic on Orange Mobile Network

Data Science Project

MSc DANI – Telecom SudParis

Prepared by:

Nourhan ZIDAN

16.12.2025

1. Introduction

This project analyses the performance of different machine learning models to predict users' Quality of Experience (QoE) for HTTP YouTube content on mobile networks. This is done on a dataset containing 22 QoE Influence Factors (QoE IFs) and subjective Mean Opinion Scores (MOS) given by users after viewing the content. The goal is to understand which QoE parameters most strongly influence the perceived QoE and to build models that can accurately predict QoE levels from network measurements.

2. Dataset overview

The dataset was built from a crowdsourcing campaign test done by Orange with size 23x1560, where 1560 samples covering 22 Quality of Experience Impact Factors (QoE IFs) were tested against the target Mean Opinion Score (MOS). The videos used are of different types/complexities. Each variable belongs to a category of QoE Influence Factors (QoE IFs) as shown below:

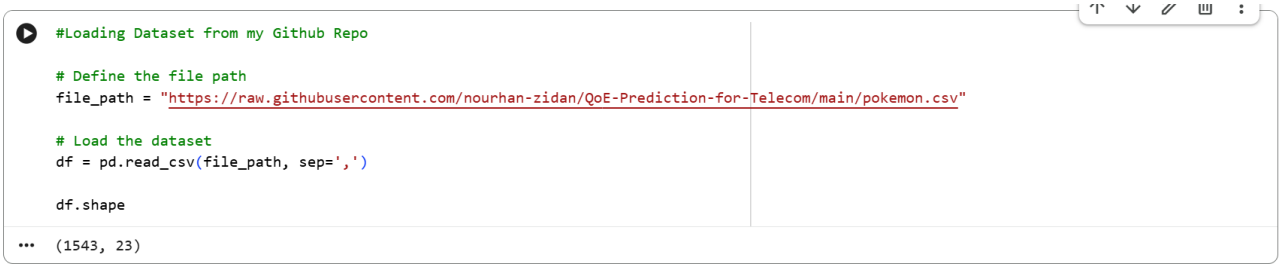
- Video parameters from VLC video player (QoA)
- Network information (QoS)
- Device characteristics (QoD)
- User's profile (QoU)
- User feedback (QoF)

MOS takes finite values from '1' to '5' with 1 being the lowest and 5 being the highest.

More information about the dataset can be found here.

3. Exploratory Data Analysis

The dataset is loaded directly into a pandas DataFrame from a CSV file named 'pokemon.csv'. To facilitate reproducibility, I added the csv file to a GitHub repository together with the code. This allows other users to view the dataset and re-run the whole experiment upon cloning the repository.

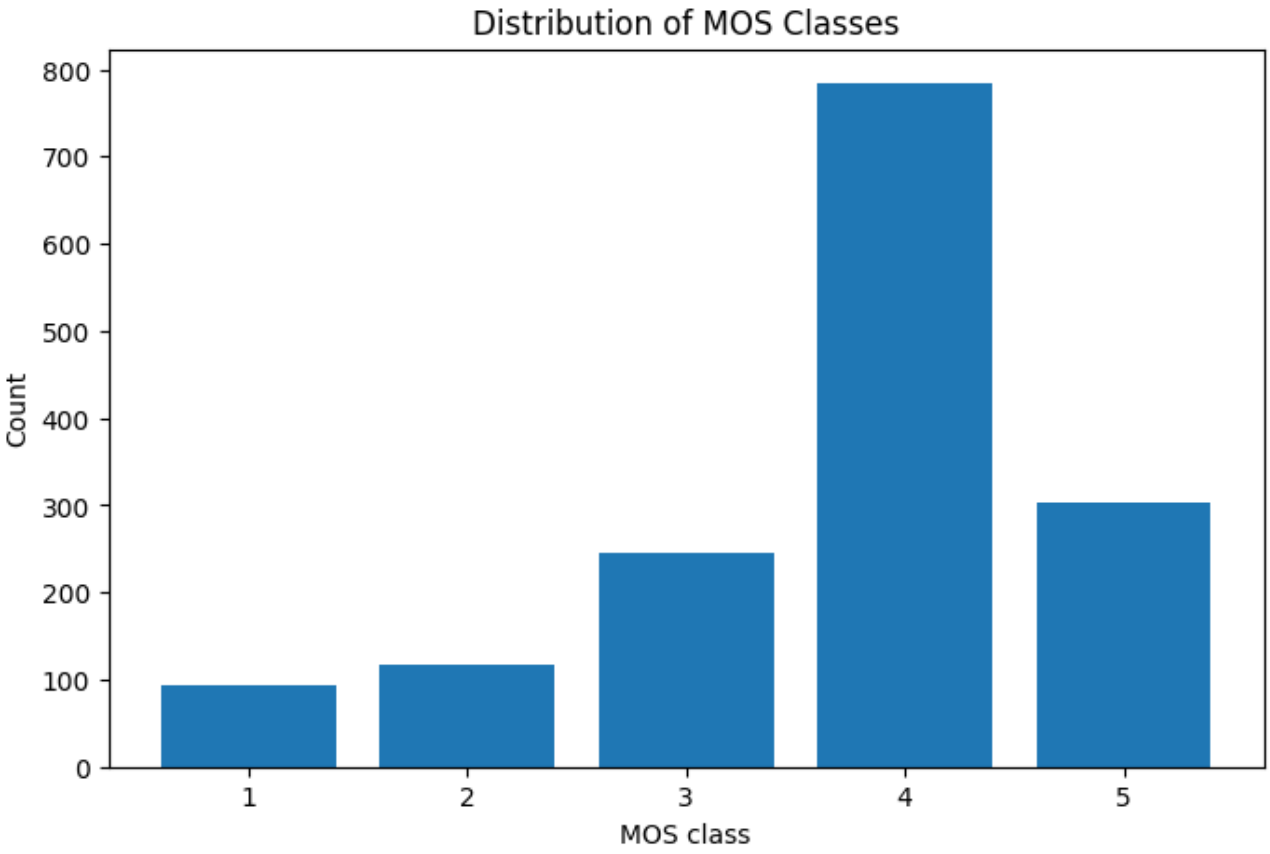


```
#Loading Dataset from my Github Repo

# Define the file path
file_path = "https://raw.githubusercontent.com/nourhan-zidan/QoE-Prediction-for-Telecom/main/pokemon.csv"

# Load the dataset
df = pd.read_csv(file_path, sep=',')

df.shape
```

```
(1543, 23)
```

The GitHub repository link can be found here.

Upon initial analysis of the dataset, I could point out some columns that were not useful to MOS prediction such as (id) and (user_id). Then I wanted to see what the distribution of the 5 MOS classes looked like which showed that there is an issue of class imbalance where class 4 was significantly dominant to other classes.



There were also two non-numerical columns (QoD_model and QoD_os-version) showing the model and the OS versions, respectively, of the android devices used by users in the experiment. Classes with very few samples like 1 and 2 might also cause the model to be unstable.

4. Preprocessing

4.1. Dropping null values

The first thing I wanted to do was drop rows with any NA cells using df.dropna() function. But surprisingly, no rows contained null values.

4.2. Dropping irrelevant columns

Based on the prior assessment of the data, I decided that the irrelevant columns (id, user_id) needed to be removed. After that, I was curious to see if the study level of the users (QoU_Ustedy).

```
df['QoU_Ustedy'].value_counts(normalize=True)

#Will drop study level since most of the data (92%) are the same
```

QoU_Ustedy	proportion
5	0.923526
4	0.052495
2	0.017498
3	0.006481

dtype: float64

```
pd.crosstab(df['QoU_Ustedy'], df['MOS'], normalize='columns')
```

QoU_Ustedy	MOS 1	2	3	4	5
2	0.010753	0.033898	0.020325	0.016582	0.013245
3	0.000000	0.008475	0.016260	0.001276	0.013245
4	0.043011	0.135593	0.077236	0.045918	0.019868
5	0.946237	0.822034	0.886179	0.936224	0.953642

I also suspected that columns like QoD_api-level, QoD_model and QoD_os-version might have little to no effect on MOS prediction. To validate that, I performed chi-square tests but contrary to my expectations, p-values were very little, suggesting that there was in fact dependency between these columns and MOS.

To conclude this step, I only dropped the columns (id, user_id and QoU_Ustedy).

4.3. Encoding Categorical Variables

Now, I move to encode the categorical columns (QoD_model, QoD_os-version) so that my model can work on them. I chose to go with One-Hot Encoding and not Label Encoding since these columns have no ordinal meaning. The number of columns after encoding is 51.

4.4. Correlation Analysis and Feature Selection

Since all my data is now numerical, I proceeded to study the dependencies between features as well as between each feature and the target MOS. I experimented with thresholds between 0.1 and 0.2, but 0.2 gave better model results. All columns with correlation_with_MOS less than 0.2 were dropped.

Shown below are the first 18 feature correlations for illustration.

```
corr = df_encoded.corr(numeric_only=True)          # full matrix
corr_with_MOS = corr["MOS"].abs()             # absolute correlation with target
print(corr_with_MOS.sort_values(ascending=False))
```

```
MOS                                     1.000000
QoF_audio                               0.840735
QoF_video                               0.689358
QoF_shift                               0.634058
QoF_begin                               0.591324
QoA_VLCframerate                        0.544164
QoA_BUFFERINGtime                       0.482378
QoA_BUFFERINGcount                      0.411176
QoA_VLCaudiorate                        0.353631
QoA_VLCaudioloss                        0.323338
QoA_VLCdropped                          0.237135
QoS_type                                0.146741
QoD_model_ARCHOS 101G9                  0.134531
QoD_os-version_4.0.4(20130118.175432)   0.134531
QoD_api-level                           0.133560
QoD_model_HTC One X+                    0.118508
QoD_os-version_4.1.1(122573.16)         0.118508
QoD_model_D5803                         0.109564
```

Final shape of the dataframe after this step is (1543, 11).

4.5. Train-Test Split

To prepare the training and testing data, I perform splitting of the dataframe with 20% of the samples reserved for testing (test_size=0.2) using random_state=1 for reproducibility and added stratify=y to ensure that the class distribution of MOS is preserved in both train and test sets and that samples from all classes are included.

4.6. Scaling Features

Finally, a Standard Scaler is fitted only on the training features (X_train) and the learned scaling (mean and standard deviation) is applied consistently to both the training and test features so that all input variables are on a comparable scale without leaking information from the test set into the scaling step.

5. Model Building and Training

In this section, I explain my thought process while experimenting with three different models and evaluating their performance.

I wanted to start by the most straight-forward approach which is the multinomial regression.

5.1. Multinomial Logistic Regression

5.1.1. Model Parameters and Initial Results

Initial model values are C (logisticregression__C) =1, solver="lbfgs" (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) and max_iter (maximum iterations) =2000.

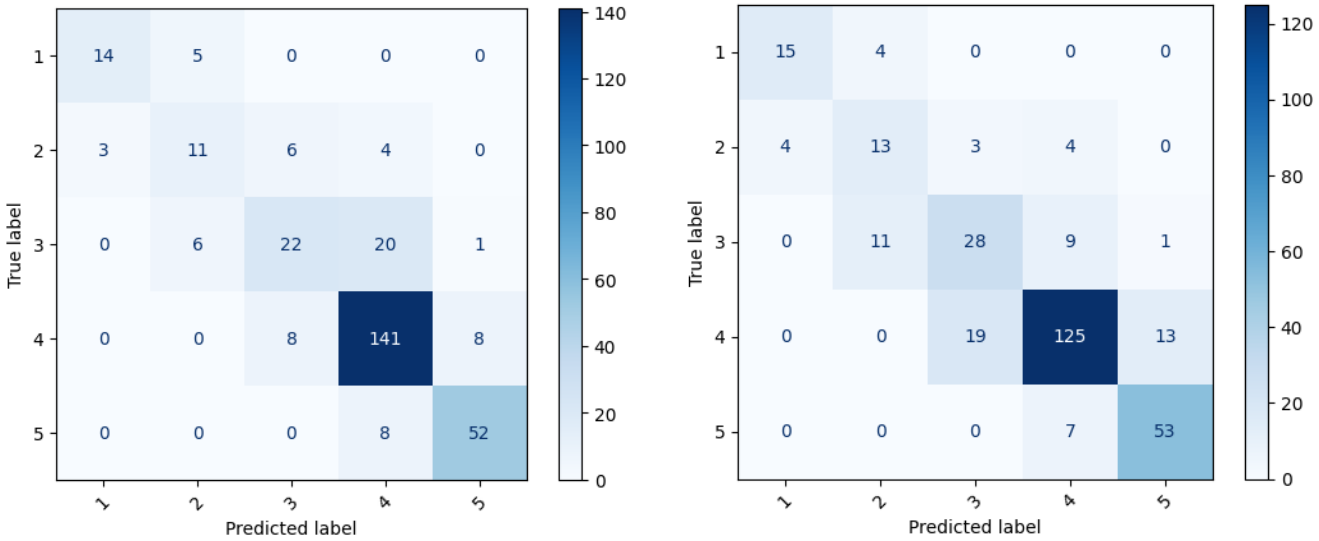
```
LogisticRegression
LogisticRegression(C=1, max_iter=2000, multi_class='multinomial')
```

Initial Accuracy is 0.777.

I suspected that the class imbalance issue is affecting the performance, so I proceeded to perform SMOTE (Synthetic Minority Oversampling Technique) which is an oversampling method that generates synthetic (artificial) samples of the minority class to help balance imbalanced datasets during model training.

What I expected was that the model will perform better after this step, but that wasn't exactly the case [Accuracy: 0.7702]. Classification improved for some classes like the minority ones but decreased for class 4.

Compared below are the confusion matrices of the model before and after SMOTE, left and right respectively.



5.1.2. Grid Search

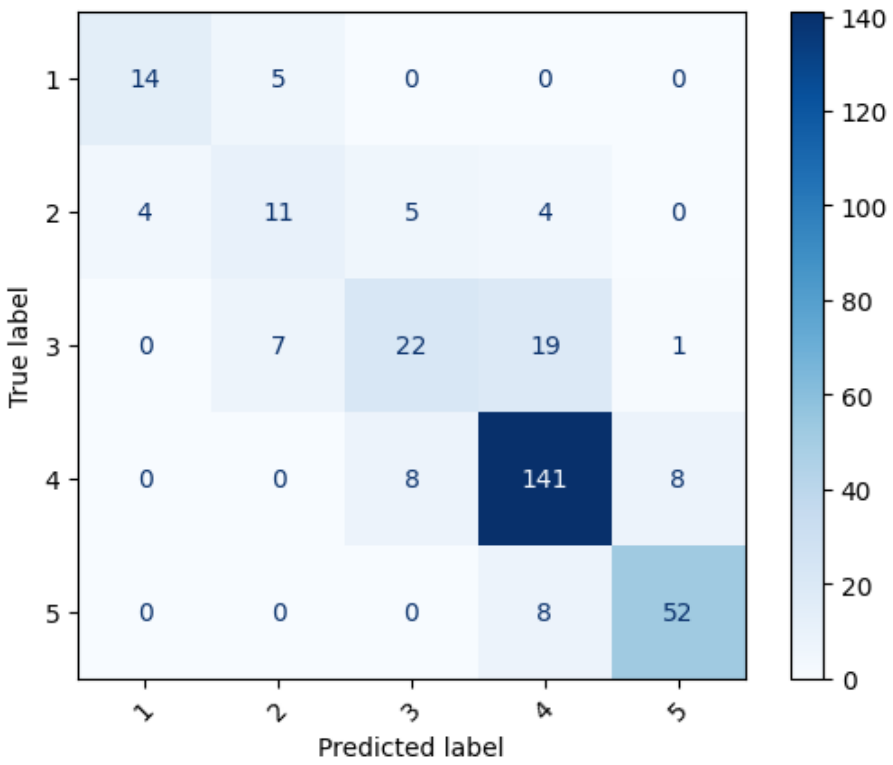
In search of the best parameters and applying cross validation on the data, I performed Grid Search with the below parameters:

- C: [1e-9, 1e-3, 0.01, 0.1, 1, 10, 100, 1000, 10000]
- K_fold= 5
- Scoring= 'f1_macro'

5.1.3. Results and analysis

The results of the best estimator are as follows:

	precision	recall	f1-score	support
1	0.78	0.74	0.76	19
2	0.48	0.46	0.47	24
3	0.63	0.45	0.52	49
4	0.82	0.90	0.86	157
5	0.85	0.87	0.86	60
accuracy			0.78	309
macro avg	0.71	0.68	0.69	309
weighted avg	0.77	0.78	0.77	309



5.2. Random Forests

5.2.1. Model Parameters and Initial Results

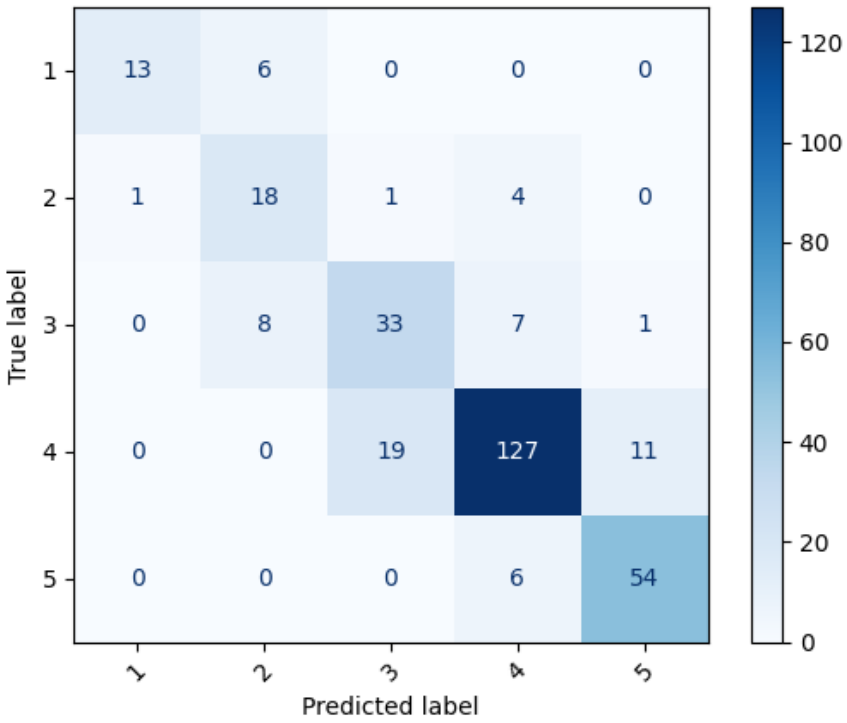
After trying multinomial logistic regression, it felt like the model may not be flexible enough to capture all the non-linear relationships between QoE Ifs and MOS. Logistic regression is great as a baseline, but it assumes mostly linear effects and can miss more complex interactions between features. So, my next model to try was Random Forests because it can handle non-linear decision boundaries and works well for multi-class problems. I also chose random forest and not a decision tree to leverage the knowledge of many trees, which helps reduce overfitting.

```
RandomForestClassifier
RandomForestClassifier(max_leaf_nodes=16, n_jobs=-1, random_state=5)
```

As a first experiment, I ran the RandomForestClassifier() with100 estimators , the criterion “Gini Impurity” and max_features $\sqrt{n}$.

Results were slightly better with accuracy= 0.79 and better overall f1-scores over all classes.

	precision	recall	f1-score	support
1	0.93	0.68	0.79	19
2	0.56	0.75	0.64	24
3	0.62	0.67	0.65	49
4	0.88	0.81	0.84	157
5	0.82	0.90	0.86	60
accuracy			0.79	309
macro avg	0.76	0.76	0.76	309
weighted avg	0.81	0.79	0.80	309



5.2.2. Grid Search

Using Grid Search, I tried the following different combinations of the parameter with 5-fold cross validation. For efficiency and runtime constraints, it was not possible to try more combinations.

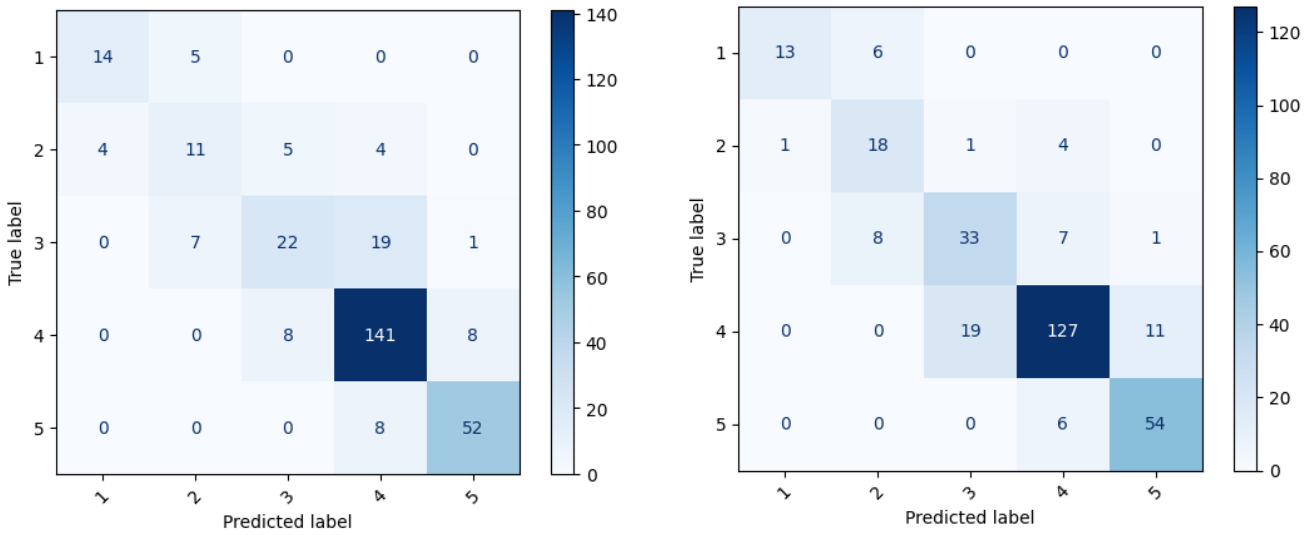
- n_estimators: [100, 200, 300],
- max_depth: [None, 10, 20],
- max_leaf_nodes: [16, 32, 64],
- min_samples_split: [2, 5, 10],
- min_samples_leaf: [1, 2, 4],
- max_features: ["sqrt", “log2”],
- criterion: ["gini", “entropy”]

The results after the grid-search are exactly as the ones shown above.

5.2.3. *Results and analysis*

Overall, Random Forests model performed better on the data than the multinomial regression model.

Below, I show the confusion matrix of Multinomial Regression (left) as compared to Random Forests (right).



5.3. XGBoost

XGBoost was a natural next step after the random forests because it is known to give better results. Random Forest builds many trees independently and averages them, which gives robustness but can leave some patterns underexplored. In contrast, XGBoost builds trees sequentially, where each new tree focuses on correcting the errors of the previous ones, allowing it to capture subtler decision boundaries and class distinctions.

Compared with Random Forest, XGBoost offers more fine-grained control over model complexity (through learning rate, number of boosting rounds, max depth, etc.), making it easier to tune for the best trade-off between bias and variance

5.3.1. *Pre-requisite: Label Encoding y-values*

For objectives like "multi:softprob" or "multi:softmax", XGBoost internally uses the label values as index positions in the predicted probability vector, so label 0 corresponds to the first column, 1 to the second, and so on. For that, I use label encoding to transform MOS class labels from [1 2 3 4 5] to [0 1 2 3 4].

5.3.2. *Model Parameters and Initial Results*

The model is set to build 300 decision trees with a maximum depth of 5, using a learning rate of 0.1 to control how strongly each tree corrects the previous ones.
The subsample and colsample_bytree parameters are both set to 0.8, meaning each tree is trained on 80% of the rows and 80% of the features, which adds randomness and helps reduce overfitting. The objective function "multi:softprob" makes the model output class probabilities for multiple classes, while the evaluation metric "mlogloss" (multi-class log loss) measures how well these probabilities match the true labels. Finally, tree_method="hist" enables a faster histogram-based tree construction algorithm, and random_state=42 fixes the random seed to make the training process reproducible.

```
xgb = XGBClassifier(
    n_estimators=300,
    max_depth=5,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    objective="multi:softprob",
    eval_metric="mlogloss",
    tree_method="hist",
    random_state=42
)
```

Test accuracy: 0.744

5.3.3. *Grid Search*

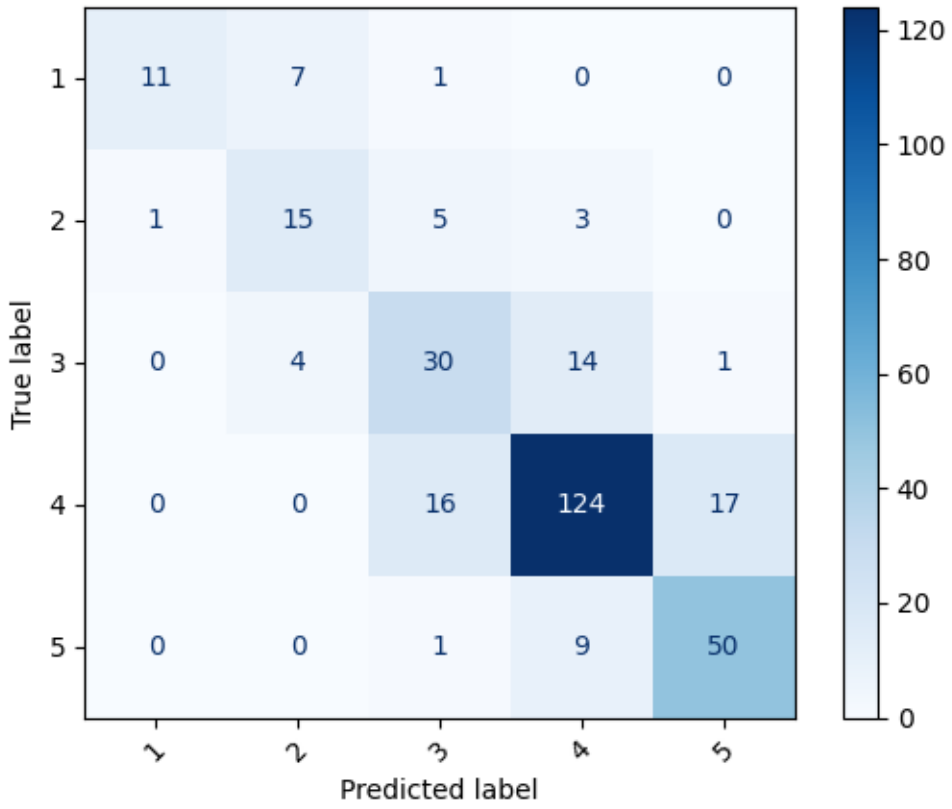
- n_estimators: [100, 300, 500],
- max_depth: [3, 5, 7],
- learning_rate: [0.01, 0.1, 0.2],
- subsample: [0.8, 1.0],
- colsample_bytree: [0.8, 1.0],
- min_child_weight: [1, 5]

5.3.4. Results and analysis

Contrary to my expectations, the overall performance and classification of the XGBoost was worse than Random Forests, with the results of XGBoost shown below.

```
print(classification_report(y_test_enc, y_pred))
```

	precision	recall	f1-score	support
0	0.92	0.58	0.71	19
1	0.58	0.62	0.60	24
2	0.57	0.61	0.59	49
3	0.83	0.79	0.81	157
4	0.74	0.83	0.78	60
accuracy			0.74	309
macro avg	0.72	0.69	0.70	309
weighted avg	0.75	0.74	0.75	309



Possible Factors for the worse performance are the small number of data samples and overfitting. With only about 1,560 samples, XGBoost's high flexibility and many tunable parameters can overfit the training data more easily than Random Forest, especially if

n_estimators is large and regularization is mild. Another reason could be that The MOS labels are inherently noisy and subjective, which favors models that are more forgiving and can average out noise better like Random Forest.

6. Conclusion

After studying the performance of the three different models: Multinomial Regression, Random Forest and XGBoost, results show a better performance by Random Forest compared to other techniques. Contrary to expectations, XGBoost showed the worse performance, even after using the best parameters obtained by grid search.

7. Future Work and Possible Enhancements

Due to time constraints of the project, only the most efficient steps were performed. However, there are plenty of steps, techniques and models that can be experimented to get better overall results. Some of those I will discuss below.

1. Analysing each feature, plot boxplot and find anomalies to drop or replace them. The anomalies might be the reason the performance is unstable.
2. Exploring ways to grow the dataset, I believe that 1560 is small for a multi-class classification problem like this one that is highly subjective (MOS).
   Example: Simple Bootstrapping
3. As the number of samples was small, I suspected that the performance of any neural network might not have been promising. However, if the size of the dataset is increased, a deep neural network might prove to be of superior performance to the techniques discussed so far in this report.
4. Trying Recursive Feature Elimination (RFE)
5. Extending hyperparameter search for Random Forest and XGBoost using a more systematic search such as randomized or Bayesian optimization.