Wine quality classification
PA026 report

Miroslav Mažgut, Filip Gregora

June 2025

The primary objective of this project was to predict wine quality based on various chemical compounds, including citric acid, residual sugar, and others. A secondary aim was to examine the effect of different upsampling techniques to the performance of machine learning algorithms.

1 Exixting projects

An earlier study working with the same dataset is presented in [1]. While that work focused on applying standard machine learning models to the wine quality dataset, it did not address the issue of class imbalance.

In our project, we build upon this foundation by introducing and evaluating various sampling techniques specifically designed to handle imbalanced data. Our main contribution lies in applying methods such as SMOTE, KMeans-SMOTE, and synthetic data generation to improve the representation of minority classes, thereby aiming to boost the overall performance and robustness of the models.

2 Dataset, Preprocessing and Evaluation

This chapter introduces the dataset used in the project, describes the preprocessing steps, and describes used evaluation metrics.

2.1 Dataset

In this project, two datasets were used. The first dataset contains 4 898 samples of white wine, while the second contains 1 599 samples of red wine. These datasets were combined into a single dataset, with a label which indicate whether each sample is red or white wine. As a result, we obtained a dataset with a total of 6,497 wine samples. Both of these datasets can be accessed from: https://archive.ics.uci.edu/dataset/186/wine+quality

Each sample includes the following features: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, quality, and a binary indicator specifying whether the wine is red.

The dataset contains seven distinct wine quality classes. As shown in Figure 1, the majority of instances are concentrated in the middle quality classes, while the border classes contain significantly fewer samples. We expect that the limited number of samples in the border classes negatively affects model performance. Therefore, applying upsampling techniques may help improve the performance of machine learning models. However, we expect that the two smallest classes may not have enough samples to effectively apply upsampling methods.

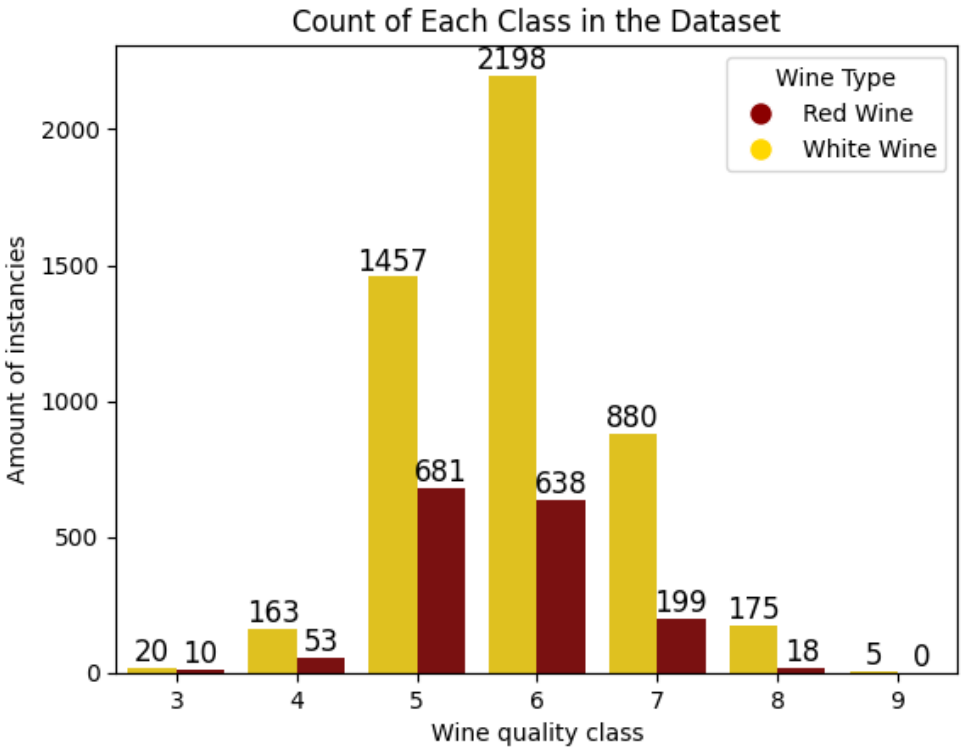

Figure 1: Plot of number of instances in each class

2.2 Preprocessing, Learning Pipeline

The first step we have done with our dataset was to separate the dataset into input features and target values. From original dataset we have removed attribute quality and store it in its own table. Subsequently, the data was split into training, validation, and test sets. The test set contains 20 % of samples, train set contains 64 % of samples and validation set contains 16 %.

The next step was to normalize the data using standardization, which transforms the data to follow a Gaussian distribution with a mean equal to 0 and a standard deviation equal to 1.

$$x' = \frac{x - \mu}{\sigma}$$

Where x' is transformed value, x is input value, μ the average value from the data, and σ is the standard deviation from the data.

After this step we applied model dependent preprocessing or directly some machine learning model. Each model or preprocessing contains different hyperparameters, which we need to optimize. For this purpose we have used Halving Grid Search, which start searching all possible combinations, but after some steps prune not promissing solutions.

After this step, we applied model specific preprocessing or directly a machine learning model. Each model or preprocessing method includes various hyperparameters, which require tuning. For this purpose, we used Halving Grid Search, an efficient optimization technique that begins with evaluating all possible hyperparameter combinations on a limited subset of resources. As the search progresses, less promising configurations are pruned, allowing more resources to be allocated for the most promising candidates.

2.3 Evaluation

For evaluation, we use multiple metrics and observe the performance of models across each of them. The first metric is the Root Mean Squared Error (RMSE), which measures the average magnitude of the prediction error and is commonly used in regression tasks. RMSE measures how far are the predicted values from the actual values.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Where n is number of samples, y_i is prediction of model for the i-th sample and $\hat{y}_i$ is correct value for the i-th sample.

The second metric used is accuracy, which represents the proportion of correctly predicted instances from the total number of predictions. Accuracy is commonly used in classification tasks, we include it in our evaluation because, despite performing a regression task, the number of quality classes is relatively small. For this reason we aim to assess how well the models can distinguish between these discrete quality levels.

The last used metric is balanced accuracy, which calculates the average in each class and average them together. This metric is useful, when we have imbalanced class distributions, because it gives equal weight to each class regardless of its frequency. The reason for using this metric is same as for accuracy, however, due to the imbalanced distribution of classes, balanced accuracy provides a more meaningful evaluation.

3 Models

After preprocessing, we trained several machine learning models and compared their performance. The selected models are Naive Bayes (NB), Regression Tree (Tree), Ridge Regression (LR), Support Vector Machine (SVM), and Neural Networks (NN). In our implementation code, we are referring to these models using abbreviations, which are provided in parentheses behind each model.

3.1 Naive Bayes (NB)

The Naive Bayes model is quite simple and usually does well with text classification tasks, especially when features are conditionally independent given the class. It applies Bayes' Theorem with the "naive" assumption that all features contribute independently to the outcome. Despite its simplicity, Naive Bayes can perform surprisingly well, particularly with large datasets and when speed is important.

3.2 Regression Tree

This model works by splitting the data into subsets based on feature values, creating a tree-like structure where each leaf represents a predicted value. It captures non-linear relationships and interactions between variables effectively. Regression Trees are easy to interpret and visualize, but they can be prone to overfitting if not properly pruned or regularized. In our project, we limited the maximum depth of the regression tree to 20 to prevent overfitting and maintain model interpretability.

3.3 Ridge Regression

The Ridge Regression is a linear model that includes L2 regularization, which adds a penalty to large coefficient values to prevent overfitting. It is particularly useful when dealing with multicollinearity, where predictor variables are highly correlated. Ridge Regression balances model complexity and fit, often leading to better generalization on unseen data compared to ordinary linear regression. In the project we set ridge alpha to 42.

3.4 SVM

The Support Vector Machine aims to find the optimal hyperplane that separates classes (or fits data, in regression) with the maximum margin. It is effective in high-dimensional spaces and can model non-linear relationships using kernel functions. SVMs are robust to overfitting, especially in cases with clear margins of separation, but can be computationally intensive with large datasets. In the implementation, we used the Radial Basis Function (RBF) kernel to capture non-linear relationships in the data.

3.5 Neural Network

This model is inspired by the structure of the human brain and consists of layers of interconnected nodes (neurons) that can learn complex patterns in data. Neural Networks are highly flexible and capable of modeling both linear and non-linear relationships. They perform well on large datasets, especially in tasks like image and speech recognition, but require careful tuning and are often computationally expensive to train.

For this neural network we chose feedforward architecture with 32 neurons in first hidden layer, 128 neurons in second, 64 neurons in third, 16 in fourth and 8 in last hidden layer. Feedforward architecture is neural network, where each neuron in each layer is connected to all neurons in previous layer. We trained the model for 500 epochs using the Adam optimizer with a learning rate of 0.0001, a batch size of 64, and one worker for data loading. We applied a dropout rate of 0.5 to prevent overfitting, and the input layer was shaped according to the number of features in the training data.

4 Upsampling

To address class imbalance, we experimented with several sampling methods and compared their approaches. The selected methods are Random Sampling, SMOTE, KMeans-SMOTE, and ChatGPT Prompting. In the following paragraphs, each method is described in detail, outlining how it works and its role in balancing the dataset.

4.1 Random Upsampling

Random Sampling selects data points randomly from the dataset without considering the class distribution. This simple method can be quick but may lead to imbalanced samples that do not represent minority classes well.

4.2 SMOTE Upsampling

SMOTE (Synthetic Minority Over-sampling Technique) balances class distribution by generating synthetic examples for the minority class. It creates new samples by interpolating between existing minority instances in feature space, helping to improve model performance on underrepresented classes.

4.3 KMeans SMOTE Upsampling

This upsampling method builds on SMOTE by first clustering the data using KMeans. It then applies SMOTE within each cluster, which helps preserve the underlying data structure and reduces the introduction of noise during oversampling.

4.4 GPT Upsampling

ChatGPT Prompting leverages a language model like ChatGPT to generate synthetic samples based on user-defined prompts. This method enables controlled data generation aligned with specific attributes or class characteristics, offering a flexible approach to augmenting datasets.

The following prompt was used together with the provided raw data: *For the uploaded dataset, do upsampling so that the number of samples is balanced for the quality attribute. The dataset contains physicochemical analysis of northern Portuguese wines. For upsampling interpret each sample as row and predict the data only based on your opinion. For upsampling use deep AI generative approach. Provide me upsampled dataset.*

The promt to ChatGPT can be found there: https://chatgpt.com/share/68053f08-2e4c-8011-a21c-1b7f9f6cacd5.

5 Used technologies, installation and startup instructions

Generally speaking, we used the Python programming language along with the environment and technologies it provides. For development, we used a combination of Jupyter notebooks (more precisely, Markdown notebooks compiled in JupyterLab into Jupyter notebooks for better versioning) together with the version control system Git, hosted on the faculty's GitLab repository.

For technologies, we used Python's libraries. For data manipulation and processing, we used Pandas and NumPy; for model construction, scikit-learn and PyTorch; and for data plotting, Matplotlib and Seaborn, together with some extra utility libraries.

5.1 Installation and Startup Instructions

To run the notebook, Python 3, JupyterLab, and the libraries listed in the `requirements.txt` file (installable via the `pip` package manager) are required.

The project includes CSV files containing both raw and upsampled data. It also consists of four Markdown files:

- `wine_quality_classification`: main classification notebook
- `wine_quality_no_borders_upsampling`: upsampling without class border constraints
- `wine_quality_upsampling`: standard upsampling implementation
- `graphs_preparation`: generates plots from recorded evaluation data in `upsampling_whole_sep.csv` saved in `figs` directory

6 Results

In the results of our experiments, we observed several interesting patterns. These results can be seen in figure 2. First, we found that the neural network model achieved the highest accuracy and balanced accuracy across all upsampling methods. In terms of RMSE, the best results were obtained by both the SVM model and the neural network model. We also noticed an inverse relationship between accuracy and RMSE. Models with higher accuracy often have worser RMSE values.

Across all models, upsampling increased balanced accuracy, which indicated improved performance on classes with few samples. But only the neural network model, with SMOTE upsampling and with KMeans SMOTE upsampling, showed improvements in both accuracy and balanced accuracy. On the other hand, upsampling generally led to higher (=worser) RMSE values, suggesting bigger mistakes in predictions. The GPT based upsampling did not show good results across any of the metrics.

We also experimented with a modified dataset, where we removed the border quality classes with the fewest samples, specifically quality 3 and quality 9. The results of this experiment are in figure 3. For consistency, we used the original test set, quality 3 and 9 included, for evaluation.

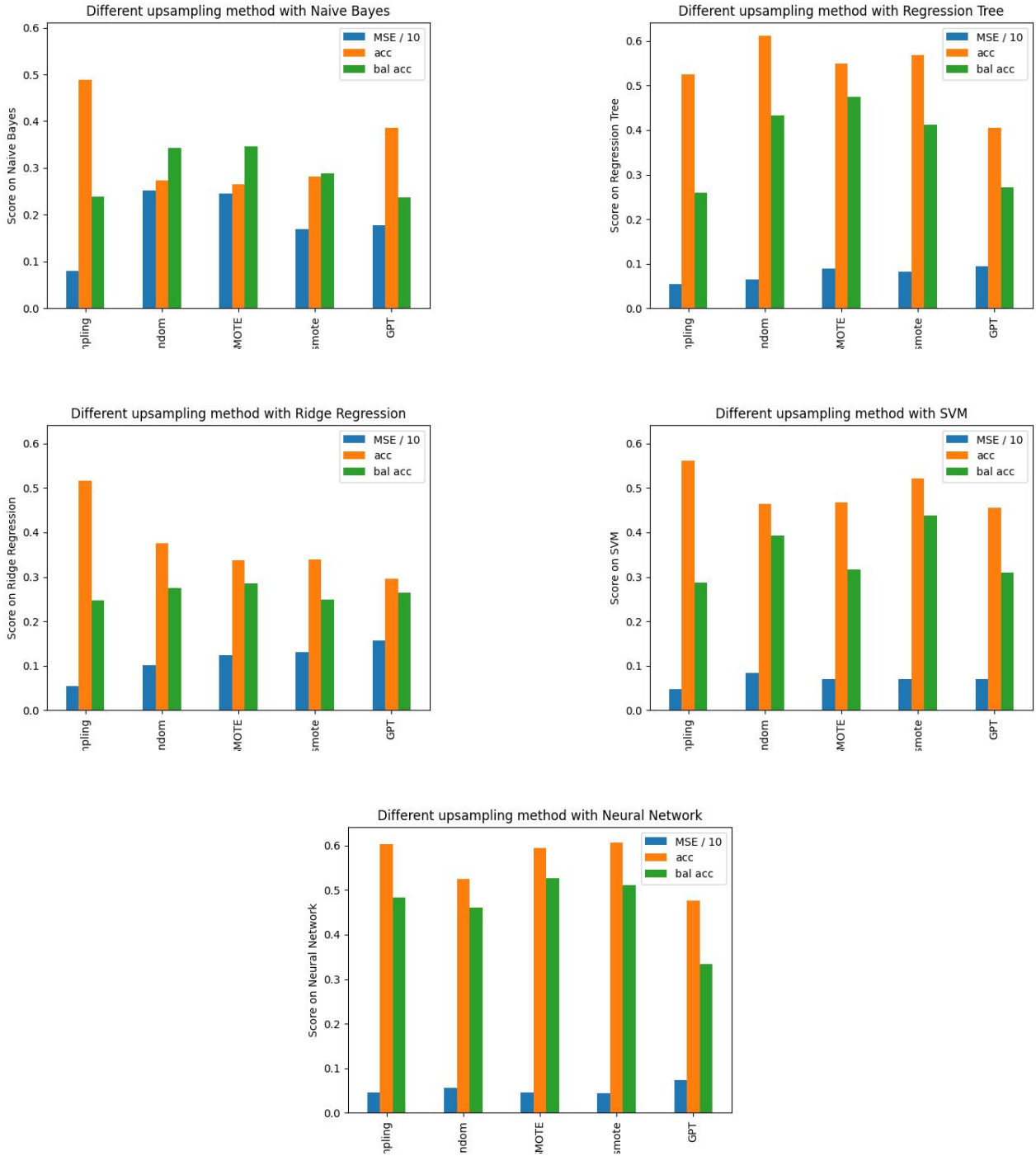

Figure 2: Plots showing evaluation metrics of sampled data on different model architectures

In the modified dataset, we observed improved performance across all models in all evaluation metrics, except for the neural network model. The neural network was an exception, where none of the upsampling methods improved its performance except GPT-based upsampling, which significantly enhanced the results.

On the other hand GPT-based upsampling did not perform well with all other models. In contrast, random upsampling, SMOTE, and KMeans-SMOTE showed similar results across all models. For more sophisticated models such as SVM and Neural Networks, KMeans-SMOTE achieved the best results, followed by SMOTE, with random upsampling performing slightly worse. The differences between these methods were small. For simpler models like Naive Bayes and Regression Tree, this order was reversed, though the variation in performance across upsampling methods remained minor.

7 Conclusion

In this project, we focused on predicting wine quality using various machine learning models and evaluating how different upsampling techniques influence their performance. We used several regression and classification metrics to measure model accuracy, robustness, and error rate.

We tested five different models: Naive Bayes, Regression Tree, Ridge Regression, SVM, and Neural Network and applied multiple upsampling techniques: Random Upsampling, SMOTE, KMeans-SMOTE, and GPT-based synthetic data generation. Our experiments showed that upsampling methods generally improved balanced accuracy across all models, especially in cases where class imbalance was significant, but achieved worser results on RMSE.

GPT-based upsampling underperformed with most models, but with neural network, when we removed small classes it significantly improved results. We also observed that models like SVM and Neural Networks benefit from advanced upsampling methods such as KMeans-SMOTE, but difference between upsampling methods is small and is worth considering if to use some advanced upsampling or to use random upsampling with significantly less work.

Overall, the project demonstrates that addressing class imbalance through upsampling can lead to better performance, especially when using more complex models, and highlights the importance of choosing the right sampling strategy for a given model and dataset.

8 AI usage

ChatGPT was used to support the writing process by assisting with grammar correction and sentence restructuring. Its main role was to improve the clarity and readability of the text. The content and ideas presented in this work are entirely our own; the AI was only used as a tool to refine language and ensure proper academic expression.

The only exception was the use of ChatGPT for generating sample data, as demonstrated in the relevant section of the report.

References

[1] Hafsa Elmrini, Aicha Bouhorma, and Khadija Rhoulami. “Wine Quality Prediction Using Machine Learning Algorithms”. In: *Journal of Computer and Communications* 9.10 (2021), pp. 73–85. DOI: 10.4236/jcc.2021.910006. URL: https://www.scirp.org/journal/paperinformation?paperid=107796.

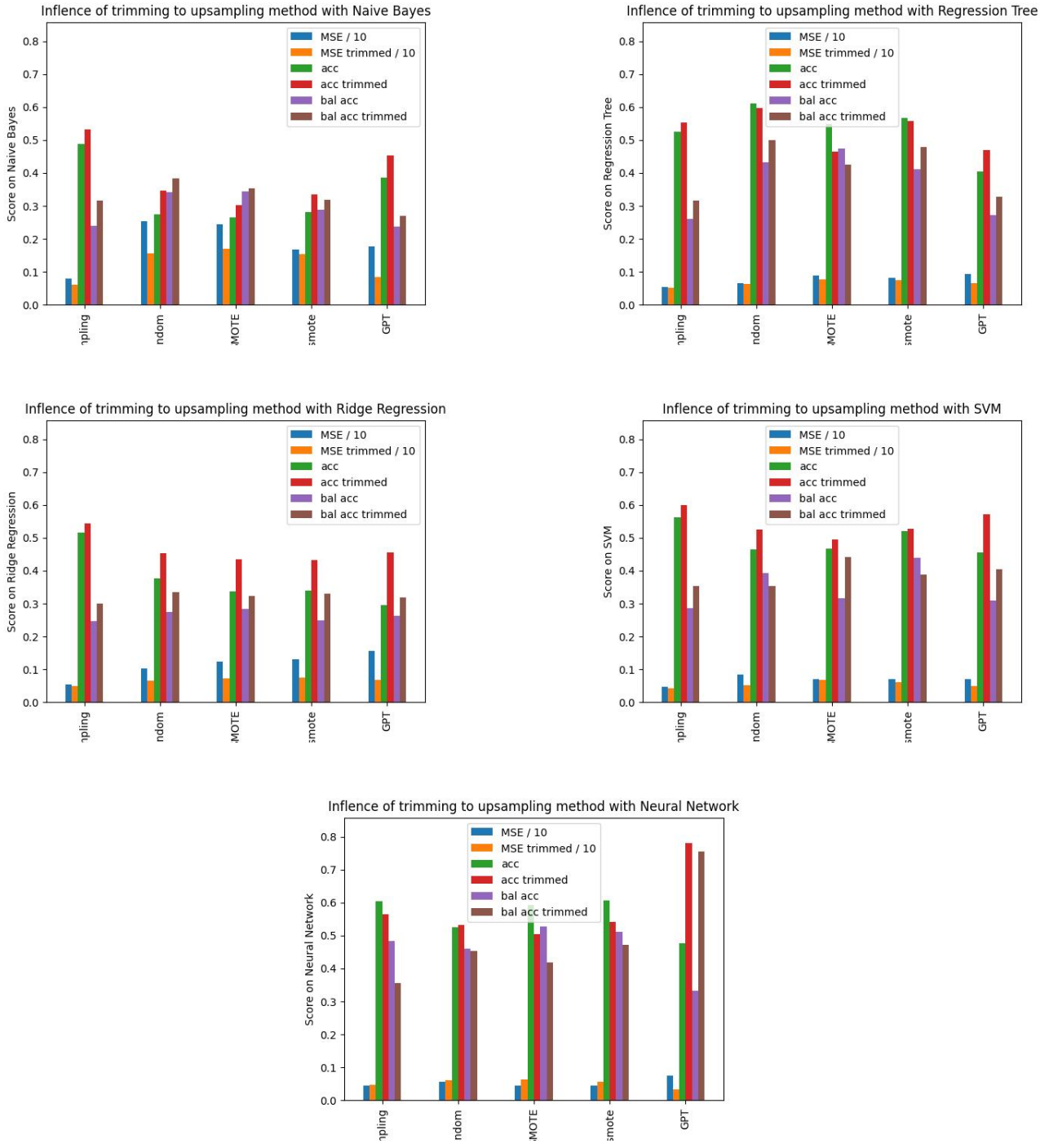

Figure 3: Plots also include trimmed version of sampled datasets on different model architectures