Regression-based age prediction model

Ana Parović and Daniele Amato
Politecnico di Torino
Student id: s344174 and s334211
s344174@studenti.polito.it and s334211@studenti.polito.it

This report represents the result of the regression-based age prediction model. For this purpose, three different models were used: random forest, linear regression with regularization techniques, and gradient boosting. The models were evaluated using root mean squared error as performance metric.

I. PROBLEM OVERVIEW

The dataset used for model development is composed of features extracted from individuals' audio notes. The features are related to either specific measures associated with the audio signal (e.g., voice pitch and energy) or to some individual's characteristics (e.g., gender and ethnicity). This dataset consists of 2933 samples provided with the target value *age* for the training of the model and 691 without it for the evaluation. In addition to the 20 present in the file, we extracted 80 more features from the original audio signals. The goal is to predict the age of the samples in the evaluation set, with the lowest possible *root mean squared error*.

II. PROPOSED APPROACH

a. Preprocessing

The first step to every model generation is preprocessing of the data. In order to check the distribution of every feature and the possible presence of outliers, we used a histogram plotting function from the *matplotlib*[1] library as shown in Figure 1.

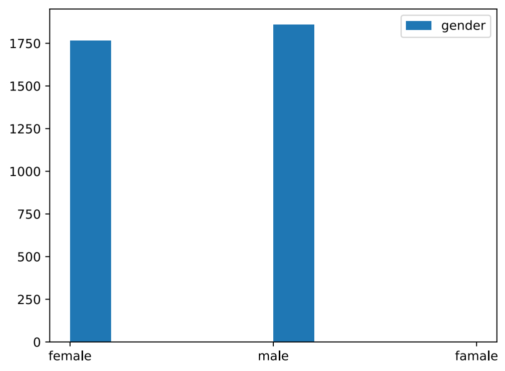

Figure 1: Gender feature histogram

Thanks to this, we noticed that in the feature `gender` appeared one sample labeled as "famale": assuming it was a typing error, we changed it to "female". We also noticed that the field `tempo` was interpreted as a string because of the square brackets, so we deleted them and converted it to float. Furthermore, the feature `sampling_rate` had the same value for each sample and therefore was removed, together with `path` that clearly gave no useful information. Then, we proceeded with converting categorical features to numerical, in particular `gender` and `ethnicity`: since they had no missing values, we used dummy encoding. After this, we exploited the *librosa*[2] library to build the spectrogram of the audio signals (Figure 2) and extract new features from it: we aggregated over time using mean and standard deviation in order to obtain the values for each frequency bean.

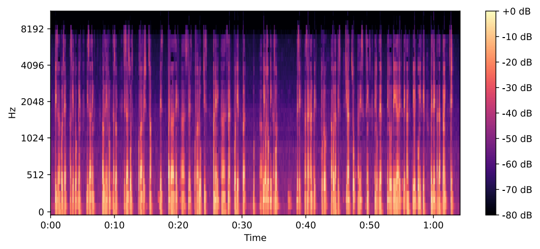

Figure 2: Spectrogram

Before choosing which regression models to use, we split the development set into training and validation, to get an idea of how they could perform on the evaluation samples prior to submitting the predictions to the platform.

b. Model selection

For the purposes of our project, we decided to use three different models, including random forest, linear regression (with Lasso and Ridge regularization) and XGBoost.

Random Forest is a model based on decision trees that combines predictions from multiple trees to reduce overfitting. It is a great choice for our problem, since it works well on datasets with a large number of features.

Linear regression is a simple model that assumes a linear relationship between input features and the target variable. It is a useful choice for its simplicity and ability to easily interpret the result.

At last, XGBoost is a powerful gradient boosting algorithm based on building decision trees, where each tree corrects the errors of the previous ones and combines their predictions minimizing the error with gradient descent. It is particularly useful for datasets that are large and diverse, which is suitable for our problem.

Additionally, we didn't apply normalization or standardization to the dataset since the selected models are not sensitive to feature scaling.

To evaluate regression models, besides the traditional training and evaluation data split, we used cross-validation for more realistic and reliable results.

c. Hyperparameters tuning

For what concerns hyperparameter tuning, we exploited the *GridSearchCV* function to test a variety of values on several hyperparameters for each of the 3 models.

Random Forest

- `n_estimators`: The number of trees in the forest.
- `min_samples_split`: The minimum number of samples required to split an internal node.

- `min_samples_leaf`: The minimum number of samples required to be at a leaf node.
- `max_features`: The number of features considered for splitting at each node.

XGBoost

- `n_estimators`: The number of boosting rounds.
- `learning_rate`: The step size for updating weights.
- `max_depth`: The maximum depth of each tree.
- `subsample`: The fraction of training samples used for each boosting round.
- `colsample_bytree`: The fraction of features sampled for each tree.

Linear Regression with Ridge and Lasso Regularization

- `alpha`: The regularization strength, which controls the trade-off between the magnitude of coefficients and the fit to the data.

III. RESULT

The hyperparameters that provided the best results in terms of our evaluation metric were:

1. **Random forest**

   - `n_estimators` = 250
   - `max_depth` = None
   - `min_samples_split` = 2
   - `min_samples_leaf` = 6
   - `max_features` = 1.0

   Root mean squared error for Random forest - 9.948

2. **Linear regression**

   - `alpha (Lasso)` = 0.01
   - `alpha (Ridge)` = 10

   Root mean squared error for Lasso linear regression - 9.980

   Root mean squared error for Ridge linear regression - 9.967

3. **XGBoost**

   - `n_estimators` = 285
   - `learning_rate` = 0.06
   - `max_depth` = 3
   - `subsample` = 0.6
   - `colsample_bytree` = 0.6

   Root mean squared error for XGBoost - 9.560

In addition, a feature importance analysis was conducted to assess the significance of individual features. An attempt was made to remove the least relevant features, but this did not result in improved performance. Consequently, the full feature set of 320 features was retained for the final analysis. The models yielded the following results:

Linear Regression demonstrated the highest RMSE, performing similarly to Random Forest. Linear Regression's limitations are likely due to its inability to model complex interactions in the dataset and its sensitivity to the high dimensionality of the feature set. While Random Forest leveraged its ensemble nature to model nonlinear relationships, its RMSE was comparable to Linear Regression, as it lacked advanced techniques to optimize error weighting and refinement.

XGBoost delivered the best performance, due to its ability to refine predictions step by step, improving accuracy with each iteration. Its design effectively balanced learning from the data while avoiding overfitting, thanks to techniques that selectively focus on the most relevant features. This made XGBoost particularly suited to handling the complexity of the dataset.

To visualize the performance of all the models by comparing the predicted values with the actual values, we used the [4]*seaborn* library, as shown in Figure 3.

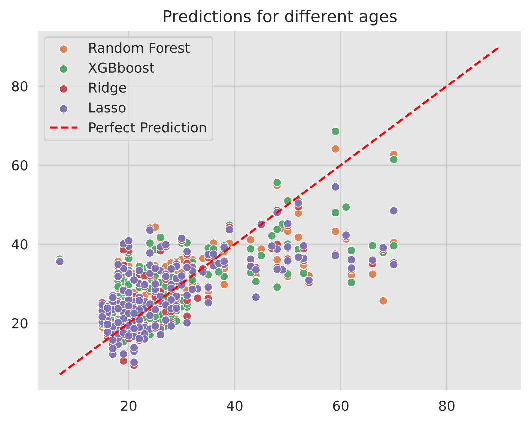

Figure 3: Comparison of different regression models

As shown in the figure, the majority of the samples have target values between 20 and 40. Within this range, the models exhibit comparable performance. However, in regions where the samples are sparse, XGBoost consistently demonstrates predictions closer to the perfect prediction line. This superior performance in less dense areas highlights why XGBoost was the most effective method overall.

IV. DISCUSSION

In the future, we could improve our model by focusing on feature selection and dimensionality reduction techniques such as PCA, t-SNE, or Recursive Feature Elimination (RFE) to identify and remove irrelevant features. Additionally, exploring other regression models like SVM or neural networks could provide better predictive accuracy. These models, though more complex, might capture non-linear relationships and complex patterns in the data. Applying standardization could also enhance model performance for these two models, which are sensitive to the scale of input data.

References

[1] The Matplotlib Development Team, *Matplotlib: Visualization with Python* - `https://matplotlib.org/stable/index.html`

[2] Brian McFee et al., *Librosa: Audio and Music Signal Processing in Python* - `https://librosa.org/doc/latest/index.html`,

[3] Tianqi Chen and Carlos Guestrin, *XGBoost Documentation* - `https://xgboost.readthedocs.io/`

[4] Juanito R. Duz, *lm_viz: A Python Library for Model Visualization* - `https://juanitorduz.github.io/lm_viz/`