SD - Final Project Report

Shamkhal Guliyev, Yigit Yumus

1st January, 2025

Introduction

We were given a list of songs with a lot of attributes from the MTCFeatures package. Each song is a record that is stored as an array of tunes. Each tune has respective values in different columns, including *phrase_end*. It is a boolean value that indicates if the tune is the end of the song. Our goal was to train different models to predict the target column *phrase_end* for each tune.

Task 1: Choosing the right attributes

To predict *phrase_end*, we first needed to determine which attributes affect the target attribute most. We determined that **correlation analysis** is the most suitable method for this purpose. However, it accepts only individual numerical values. There were two issues with the dataset in this regard:

1. The dataset stored tunes as arrays for each song.
2. Values were not numerical.

Therefore, we extracted and stored each tune as a single record by applying the *explosion method* to the dataset. Then, we converted the possible values to numerical ones (for example, True $\rightarrow$ 1, False $\rightarrow$ 0). After performing the correlation analysis, we constructed a graph of attributes that correlate the most with the target.

We applied a threshold of 0.2 to choose the best attributes. So, we got *13 attributes* that were used for predictions.

Task 2: Training Models

Solving the problem with NaN values

Before training the models on the dataset, we had to make sure that the dataset was fully prepared for training. There was one major problem: **NaN values** were scattered everywhere in the dataset. Some attributes had many, while others contained a small portion of them. We decided to implement three different strategies to solve this problem:

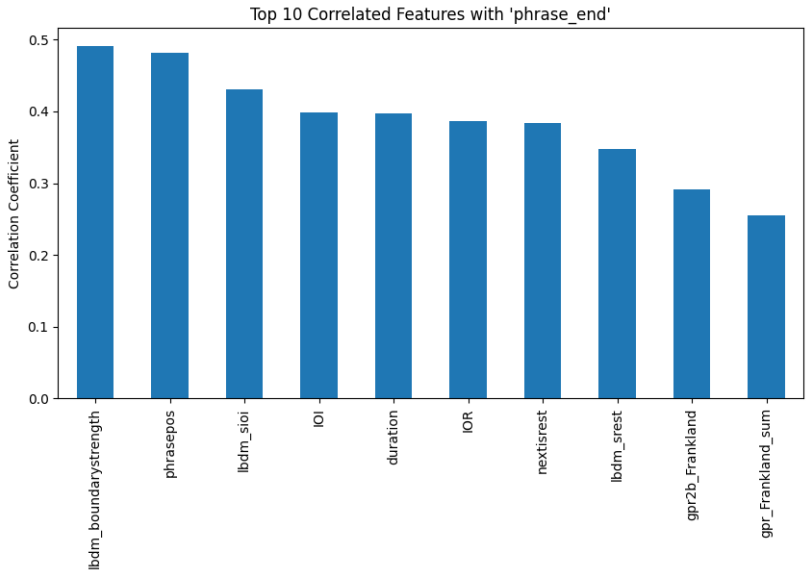

Figure 1: *Histogram of the top 10 features correlated with the target*

1. **Columns with more than 90% NaN values:** These attributes would not have significantly affected the predictions since the majority of the values were NaN. Therefore, we confidently removed these columns.

2. **Columns with a moderate number of NaN values:** We used *imputation*, which fills NaN values with certain values obtained using a strategy such as *mean, median,* or *mode*. Specifically, we chose **mean imputation**, meaning that for a single column, the average of its values will be computed, and then all NaN values will be replaced by this value. This helped to retain the relationship between values.

3. **Columns with relatively few NaN values:** Since these columns were critical for predictions, the best method was to *remove the records* where these columns contained NaN values.

Training the models and making predictions

We used four models to make predictions on the target variable:

- Logistic Regression
- Decision Tree
- Random Forest
- Neural Networks

Initially, the accuracy of the Logistic Regression was 100%, which meant that the model was likely **overfitting**. According to the graph, correlation coefficients were not as high to cause the overfitting. That is why we looked at the **feature importance coefficients:**

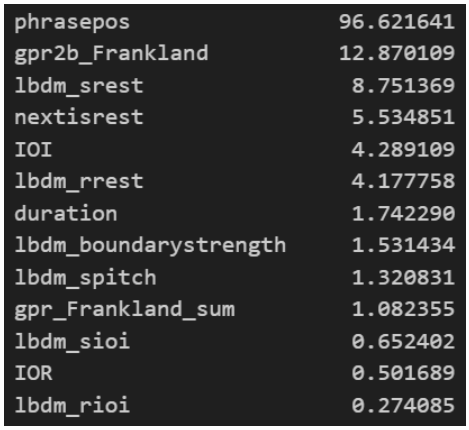

Figure 2: *List of importance coefficients of the Logistic Regression*

We noticed that the *phrasepos* attribute has a very high coefficient compared to others. This means that the model heavily relies on it to make predictions, not accounting for the input from other features. So, the model is not generalizing but rather making predictions based on the *phrasepos* attribute. So, we removed it from the dataset and trained the model again. The accuracy dropped to 83.98%, indicating that the overfitting problem was solved. To fully make sure, we checked the feature importance coefficients again.

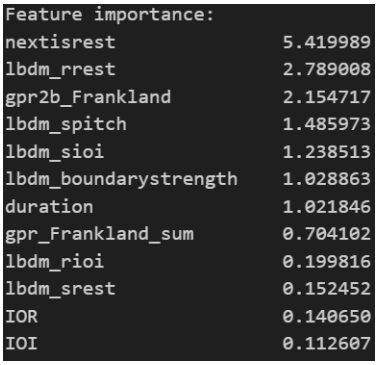

Figure 3: *List of importance coefficients of the Logistic Regression after removing the 'phrasepos' attribute*

Now the importance is evenly distributed among all features, meaning that the model is doing a good job in the generalization.

The results of other models were also quite high. To make sure that there was no overfitting for every model, we did the following:

- Compared the training and test accuracy
- Performed a cross-validation
- Checked the feature importance (if it is evenly distributed among all features)

Results

Accuracy results:

- Logistic Regression: 83.98%
- Decision Tree: 94.34%
- Random Forest: 94.49%
- Neural Networks: 94.37%

In addition to these results, we tested models with different attributes (different thresholds) and test sizes to optimize the results.

Conclusion

In conclusion, we aimed to predict whether a tune was the ending of a song. First, we had to remake the dataset so that each tune was stored as a record, which allowed us to use models on it. Then, we had to prepare data by choosing the attributes that contribute the most to making the predictions more accurate. Afterward, we had to deal with NaN values that were scattered everywhere using different strategies.

We used four prediction models to make predictions. We had to play with hyperparameters of models to reduce the computation time while keeping the accuracy of predictions at a high level. In addition, we tested the models with different attributes and thresholds to optimize their performance. **Random Forest** achieved the highest accuracy. On the other hand, **Logistic Regression** trains the fastest, although it has the lowest score among others (84.85%).