1 Introduction [15 points]

- Group members: Amitesh Pandey, Jafarbek Arifdjanov
- Kaggle team name: Jafar and Amitesh
- Ranking on the private leaderboard: 8
- AUC score on the private leaderboard: 0.85695
- Colab link: See Link
- Piazza link: See Link
- Division of labor: Amitesh coded the models, tailored the dataset, wrote sections (2) and (3) of the report. Jafar validated and tested the models, and wrote all remaining sections of the report. Debugging and model theorising was a joint activity, so was fleshing out the report details. Approximately equal contribution from both overall, just leveraging disparate skillsets.

2 Overview [15 points]

Models and techniques attempted

In our project, we have used data imputation, outlier removal, data encoding, feature selection, and ensemble machine learning. We have also attempted (but not eventually implemented) polynomial feature generation of data and also training of neural networks in Keras. Our final model is an majority vote ensemble of Light Gradient Boost, eXtreme Gradient Boost, Random Forest and Categorical Boost models. We have used hyperopt, optuna and StratifiedKFold to optimize the hyperparameters of all of the base estimators and used iterative (epoch) training to validate their performance.

Work timeline

We completed our full model over a 5-day interval. Here is a summary of the 5 days:

- Day 1: Importing the data. Cleaning it by removing NaNs and non-feature columns. Plotting the distributions of all the features and trying out feature transformations.
- Day 2: Feature transformation/normalization continued, also plotting correlations between all features and target. Testing sanity models like logistic and SVM.
- Day 3: Accuracy of ≈ 0.72 achieved with simple ML models and hygenic data. Observed that people are scoring ≥ 0.8 on the test set on Kaggle, proceeded with more advanced models like RandomForest and achieved an accuracy of 0.76 after hypertuning parameters
- Day 4: Advanced models like RandomForest continued with a StratifiedKFold method instead of searching on a Grid. Tested neural networks (3 layers, 1 BatchNorm, 2 dropout, 30 hidden units), XGBoost, CatBoost and LightGBM classification models. Advanced methods, specially CatBoost, shined because of categorical nature of time_signature, key, and mode.
- Day 5: Best accuracy achieved on Day 4 was ≈ 0.82. A final attempt to make it to the top of the public leaderboard by ensembling all hypertuned models **along with** OneHot encoding features and selection. Final accuracy on Day 5: ≈ 0.862 (public leaderboard).

Unsuccessful Models and Techniques

- XGBoost research based "categorical=True" parameter. XGBoost recently introduced a special parameter for categorical features. Although inconvenient to code up, we found manual One Hot encoding to be superior in performance to native XGBoost categorical handling.
- GridSearchCV for hyperparameter tuning. The problem with this method is that it leads to an overfit model on the exact traing_test_split used. Instead, we used cross_validate with KFold to tune our hyperparameters for the Random Forest model.
- Polynomial Features: We attempted to generate extra features from the data using the PolynomialFeatures library. This was unsuccessful because the number of features rose exponentially which causes noisy gradients and are specially problematic on smaller datasets like ours.
- StackingClassifier: Led to a decrease in performance. We opted for voting style stacking instead.

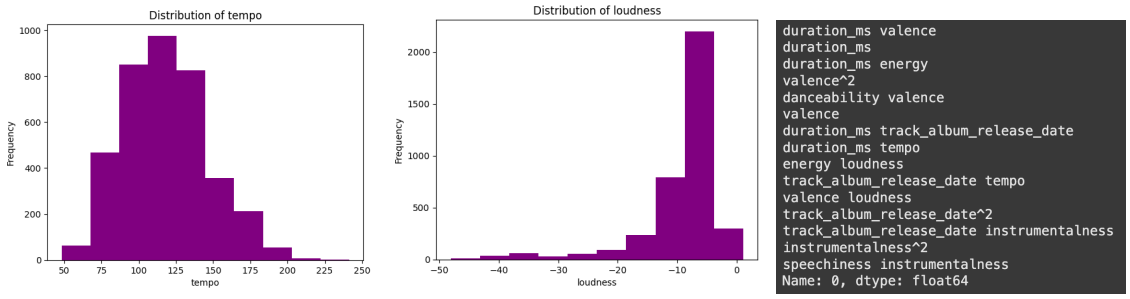


Figure 1: Skew in data and PolynomialFeatures attempt

3 Approach [20 points]

Data exploration, processing and manipulation

First, we removed NaNs. We plotted distributions of all features present in the data. This allowed us to notice that time_signature, key, and mode are categorical features. Next, we also realised through a correlation plot of the features that a lot of features (like danceability and energy) are highly correlated. This made us come to the conclusion that we will at least have to go beyond linear models in our classification approach.

We also noticed that some features had extremely high values (duration_ms), while others had negative values (loudness) and then a few were between 0 and 1. This made us realize that it would benificial to the model if we normalized all our features between 0 and 1. At this point went on with training initially.

As mentioned in the previous page, on Day 4, we realised that it might be benficial to the model if we provided it direct polynomial signals. We use SKLearn.PolynomialFeatures to generate quadratic features from all our numeric features. This allowed us to create a larger correlation plot and pick k best features. Above are images highlighting the skew and correlationalness of this dataset.

As previously mentioned, although the idea of having more features through PolynomialFeatures is useful in linear models, we realised as we proceeded to more advanced models that these additional features only add to the noise in the dataset. Additionally, it's hard to control for the correlation between, say "loudness^2" and "loudness speechness", because of this, we dropped polynomial features.

Details of models and techniques

Feature engineering: PolynomialFeatures and stats.zscore for outlier removal. Pandas: overall framework.
Model fitting: SKLearn for RandomForest LightGBM, xgboost for XGB, and catboost for CatBoostClassifier.
Model tuning: HyperOPT for XGBoost, Optuna for CatBoost, and SKLearn.KFold for RandomForest, LGBM.
Model ensemble: A (soft) majority voting classifier with SKLearn.VotingClassifier was implemented.

4 Model Selection [20 points]

As discussed in the previous page, the non linearity and intercorrelated nature of our data made it imperative for us to move on from linear models to more advanced techniques. We started with SVMs and RandomForests but eventually implemented all of the popular advanced techniques. Below is a summary of the process for our hyperparameter searches on the grids of all of these models.

Selecting RandomForest Hyperparameters

We did a StratifiedKFold cross validation hyperparameter search for the RandomForest model. The results of our hyperparameter search are visible in a tabular format in the notebook. We picked log_loss as our error metric, owing to its prevalence in binary classificaiton tasks

Selecting CatBoost Hyperparameters

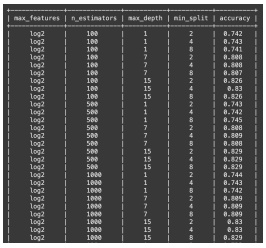
We followed a tutorial from the developers of CatBoost (at Yandex) to conduct Bayesian (previous information backed) optimization of the hyperparameters of the CatBoost model using HyperOPT. We unconventionally picked the negative of ROC_AUC as our error metric.

Selecting XGBoost Hyperparameters

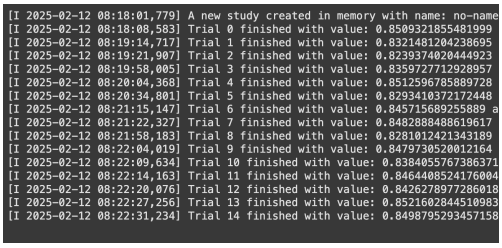
We followed a tutorial from the developers of XGBoost to conduct Bayesian (previous information backed) optimization of the hyperparameters of the XGBoost model using Optuna. We unconventionally picked the negative of ROC AUC as our error metric.

Selecting Ensemble Method

We mainly explored majority voting and stratified methods of combining our three models using SKLearn. We realised through trial and error that since the variance in our performance through individual models is not high, and the difference in performances of a single model across different batches is high, majority voting is a good idea to *even out* the differences. We used vote=soft as prescribed in the documentation.



max_features	n_estimators	max_depth	min_split	accuracy
log2	100	1	2	0.742
log2	100	1	4	0.743
log2	100	1	8	0.741
log2	100	7	2	0.808
log2	100	7	4	0.808
log2	100	7	8	0.807
log2	100	15	2	0.826
log2	100	15	4	0.83
log2	100	15	8	0.826
log2	500	1	2	0.743
log2	500	1	4	0.742
log2	500	1	8	0.745
log2	500	7	2	0.808
log2	500	7	4	0.809
log2	500	7	8	0.808
log2	500	15	2	0.829
log2	500	15	4	0.829
log2	500	15	8	0.829
log2	1000	1	2	0.744
log2	1000	1	4	0.743
log2	1000	1	8	0.742
log2	1000	7	2	0.809
log2	1000	7	4	0.809
log2	1000	7	8	0.809
log2	1000	15	2	0.83
log2	1000	15	4	0.83
log2	1000	15	8	0.829



```
[I 2025-02-12 08:18:01,779] A new study created in memory with name: no-name
[I 2025-02-12 08:18:08,583] Trial 0 finished with value: 0.8509321855481999
[I 2025-02-12 08:19:14,717] Trial 1 finished with value: 0.8321481204238695
[I 2025-02-12 08:19:21,907] Trial 2 finished with value: 0.8239374020444923
[I 2025-02-12 08:19:58,005] Trial 3 finished with value: 0.8359727712928957
[I 2025-02-12 08:20:04,368] Trial 4 finished with value: 0.8512596785889728
[I 2025-02-12 08:20:34,801] Trial 5 finished with value: 0.8293410372172448
[I 2025-02-12 08:21:15,147] Trial 6 finished with value: 0.845715689255889 a
[I 2025-02-12 08:21:22,327] Trial 7 finished with value: 0.8482888488619617
[I 2025-02-12 08:21:58,183] Trial 8 finished with value: 0.8281012421343189
[I 2025-02-12 08:22:04,019] Trial 9 finished with value: 0.8479730520012164
[I 2025-02-12 08:22:09,634] Trial 10 finished with value: 0.8384055767386371
[I 2025-02-12 08:22:14,163] Trial 11 finished with value: 0.8464408524176004
[I 2025-02-12 08:22:20,076] Trial 12 finished with value: 0.8426278977286018
[I 2025-02-12 08:22:27,256] Trial 13 finished with value: 0.8521602844510983
[I 2025-02-12 08:22:31,234] Trial 14 finished with value: 0.8498795293457158
```

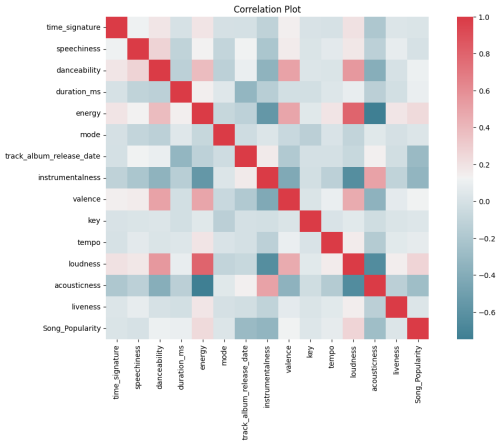
Above are our trials for hyperparameter tuning the RandomForest and CatBoost models. For detailed results of tuning XGBoost, see colab. We also tried all possible combinations of the models while picking them for the ensemble and finalised on the combination with the best out of sample performance. Final hyperparameters for all models used are as follows:

- RandomForest: (n_estimators=500, min_samples_split=4, max_features='sqrt', max_depth=15)
- CatBoost: (learning_rate=0.072, depth= 5, l2_leaf_reg=1.149, boosting_type='Plain') **[[Not used]]**
- XGBoost: (learning_rate=0.02, max_depth=10, n_estimators=260, subsample=0.815)

5 Conclusion [20 points]

Insights

To answer the question of which features are most important, we will use a correlation plot. See:



(a) Correlation Plot

	Song_Popularity
speechiness	-0.000413
time_signature	0.018423
mode	0.023820
liveness	0.026517
key	0.031733
tempo	0.073465
duration_ms	0.093976
danceability	0.112126
valence	0.120970
energy	0.237209
loudness	0.265010
acousticness	-0.270044
track_album_release_date	-0.303713
instrumentalness	-0.333993
Song_Popularity	1.000000

(b) Sorted Features coefficient

Figure 2: Feature Importance Tests

On sorting this plot with each feature's absolute correlation to Song_Popularity, we get the above list.

1. **10 Most Important Features**:
   'danceability', 'duration_ms', 'energy', 'mode', 'track_album_release_date', 'instrumentalness', 'valence', 'key', 'tempo', 'loudness', 'acousticness', 'liveness'

2. **Positively Influential Features (Positive coefficient)**:
   'danceability', 'duration_ms', 'energy', 'mode', 'valence', 'key', 'tempo', 'loudness', 'liveness'

3. **Negatively Influential Features (Negative coefficient)**:
   'instrumentalness', 'track_album_release_date', 'acousticness', 'speechiness'

The main insight was that at the heart of learning is data. Hypertuning the models, combining different models and more advanced techniques may result in some performance boosts but largely, the performance of a learning model lies in the quality of the data that's provided to it and whether signals that are intended to be learned exist in the data or not.

Challenges

We could have had a more systematic pipeline for feature engineering and selection, to make this process less time consuming and the code less clunky. We could have attempted more ensemble techniques and tuned our hyperparameters in a more standardised way. We could have been more intelligent in looking for different options when stuck at a performance level, instead of spam submitting on Kaggle.

6 Extra Credit

- We use ROC_AUC as our error metric because a large majority of songs are, in fact, not popular. Say if this percentage is 70%, then a model is able to obtain a high binary classification accuracy by simply predicting 0s at all times. This is why ROC_AUC is preferred with unbalanced classes in classification.
- Since our final models use tree based methods in an ensemble, we can parallelize both the ensemble and then the tree models separately as well. XGBoost and Random Forests are popular architectures to parallelize and libraries to do so already exist. This is because their performance typically grows with the number of estimators used, so many datasets would benefit from a large number of estimators and their simulatenous root nodes architecture, which makes quick computation important.
- One idea is that at times, even if a song is not "sonically popular" it still becomes popular because of the band/artist behind it. I suspect there is a strong relation between the artists that go into creating a song, the label that produces it, and the song's popularity. Some kind of sentiment analysis on a song's artist name/label of the artist could be benificial.

7 LLM Usage

LLMs were minimally used. Most of the debugging was done through reading documentation and StackOverFlow. The sparse instances in which LLMs were used in this project cannot precisely be pinpointed because the only times they were referred to was when the browser's in-built LLM decided to respond to a query on a search engine. They were also not used in report preparation.