Final project presentation – Group 10

Applied Machine Learning 2021

Georgia Anyfantaki,
Jun Jia,
Rebecca Schmieg &
Valerii Novikov

UNIVERSITY OF COPENHAGEN

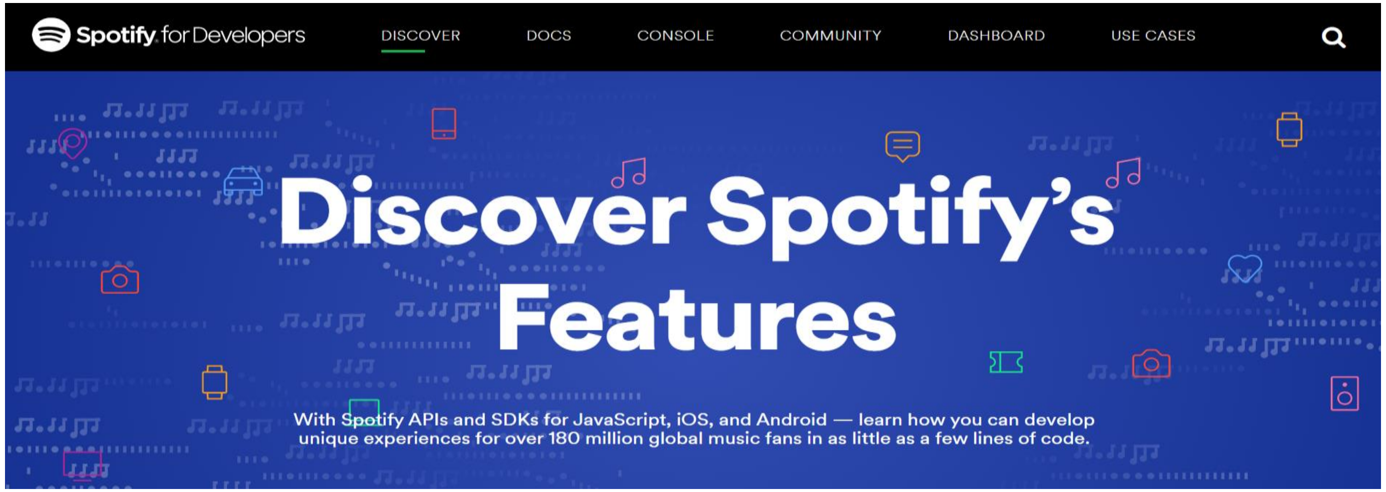
"

Everyone contributed in the programming, discussion and presentation preparation

Group 10 – Contribution statement

Introduction

- Spotify data set with ~600.000 entries
- From Yamac Eren Ay on Kaggle
- Two main goals:
  - Predict popularity (Regression & Time series)
  - Predict release year



https://developer.spotify.com/discover/#audio-features-analysis

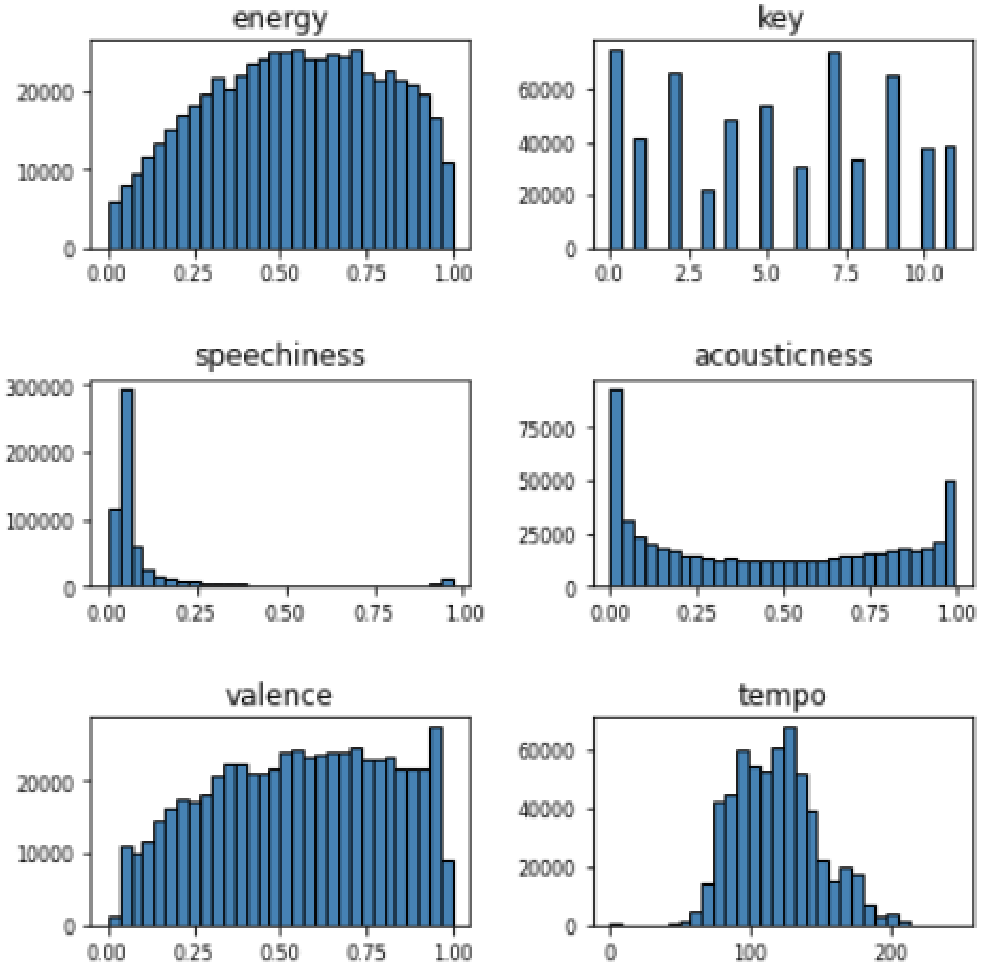
https://images.complex.com/complex/images/c_fill,dpr_auto,f_auto,q_90,w_1400/fl_lossy,pg_1/mqlimq5ifprz3klcoxpt/spotify-logo

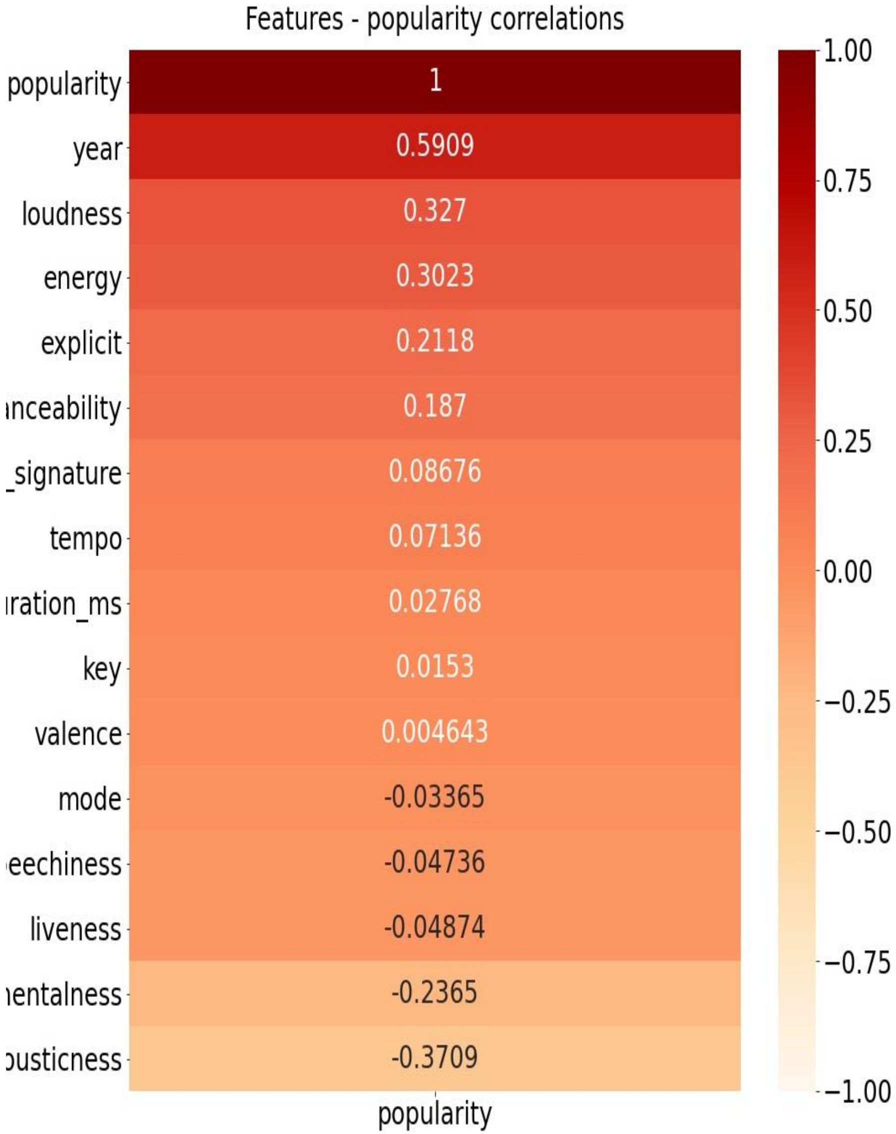
Dataset and preprocessing - Inspection

- Track: name & ID
- Artist: name & ID
- Release year
- 15 musical Features

	duration_ms	explicit	artists	artists_std	artists_minmax	danceability	energy	year	loudness	mode	speechiness	acousticness	liveness	valence
0	126903	0	['Uli']	['Uli']	['Uli']	0.645	0.4450	1922	-13.338	1	0.4510	0.674	0.151	0.127
1	98200	0	['Fernando Pessoa']	['Fernando Pessoa']	['Fernando Pessoa']	0.695	0.2630	1922	-22.136	1	0.9570	0.797	0.148	0.655
2	181640	0	['Ignacio Corsini']	['Ignacio Corsini']	['Ignacio Corsini']	0.434	0.1770	1922	-21.180	1	0.0512	0.994	0.212	0.457
3	176907	0	['Ignacio Corsini']	['Ignacio Corsini']	['Ignacio Corsini']	0.321	0.0946	1922	-27.961	1	0.0504	0.995	0.104	0.397
4	163080	0	['Dick Haymes']	['Dick Haymes']	['Dick Haymes']	0.402	0.1580	1922	-16.900	0	0.0390	0.989	0.311	0.196

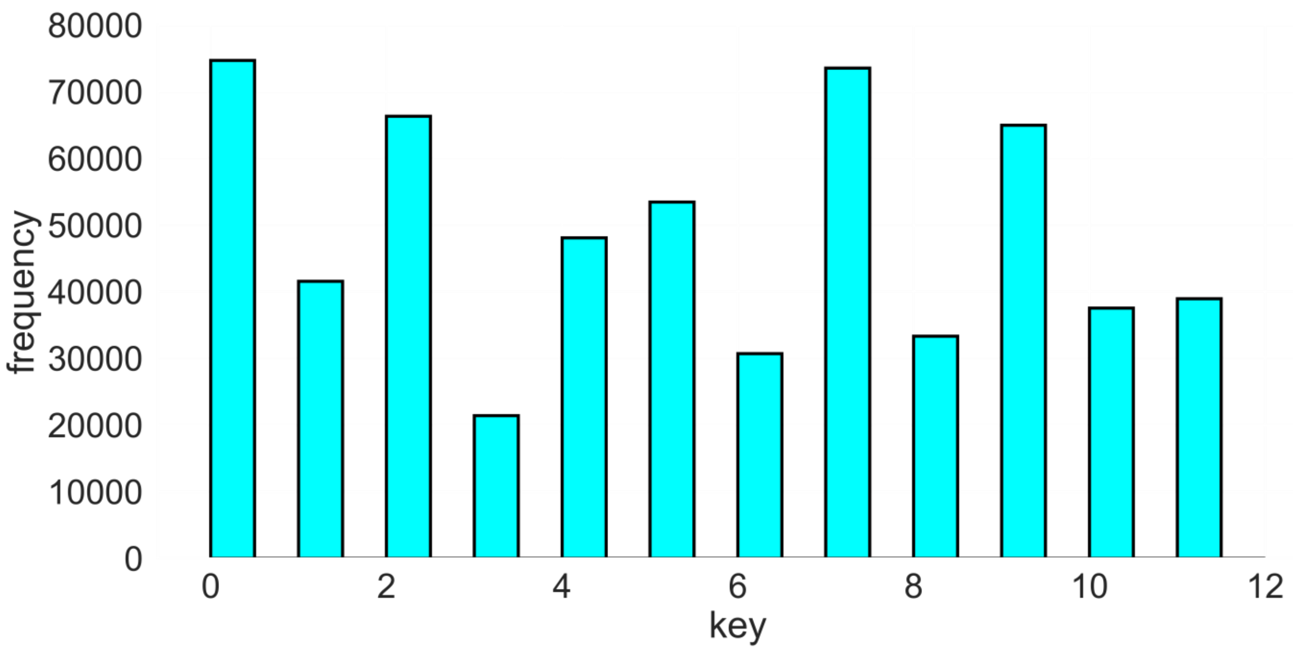
Dataset and preprocessing - Inspection





Dataset and preprocessing - Transformation

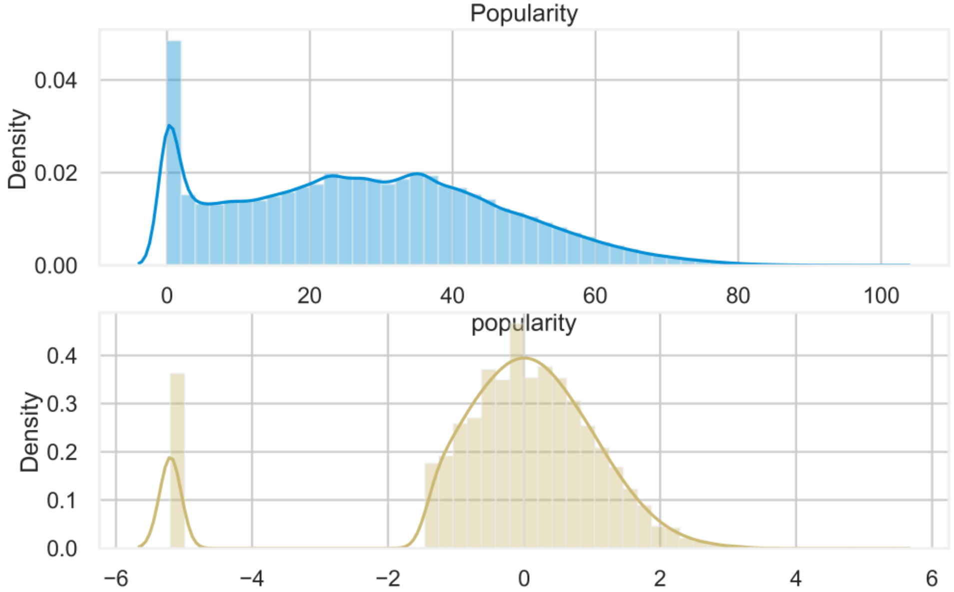
- OneHotEncoder for categorical data
- Categorical & Scaling transformation
- Transformation Popularity Truth:
  - Artist:
    - replaced by mean popularity of artist
    - Single: replace by mean all tracks
  - Apply transformation to train and test based on train set only!
- After transformations:
  - # Features: ~30

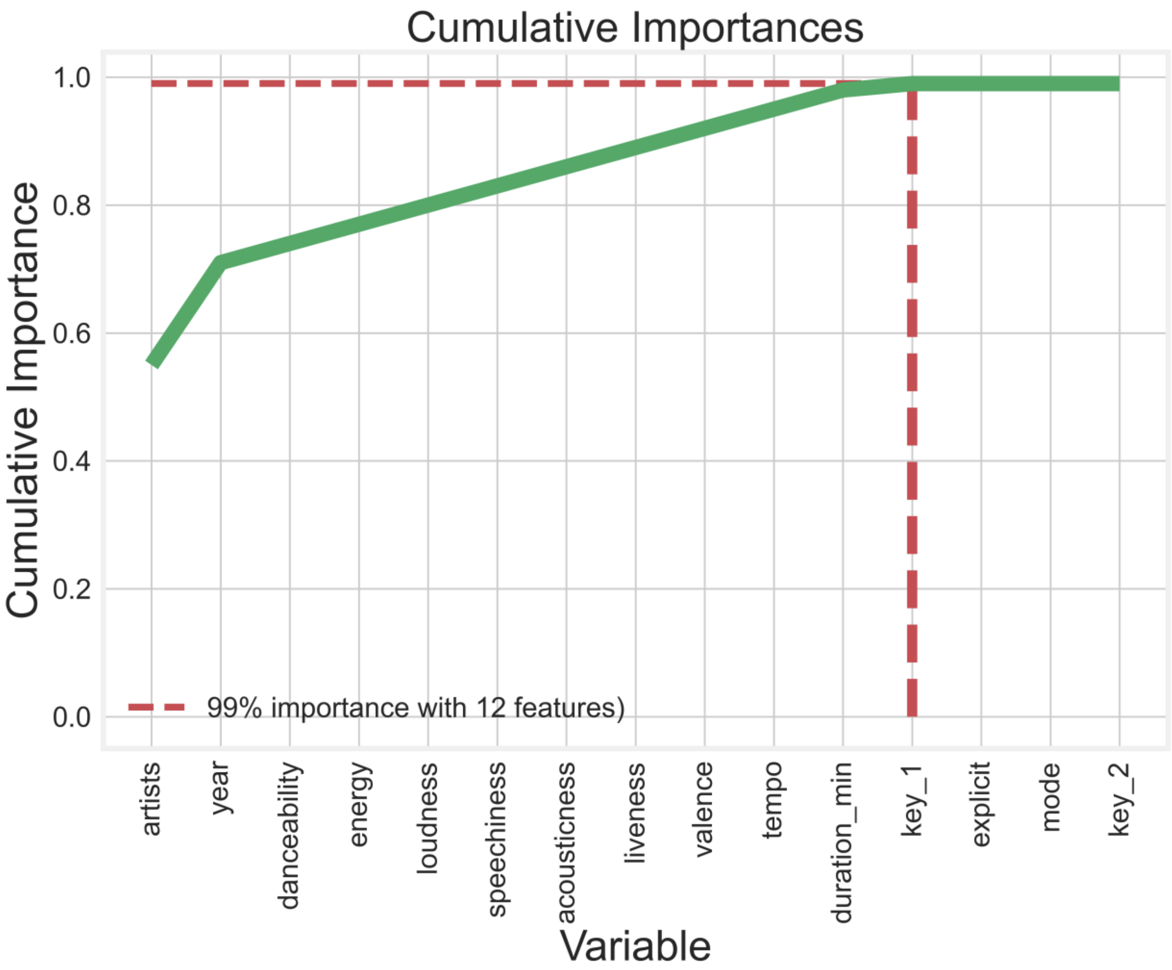


Predict popularity of a track

General approach:

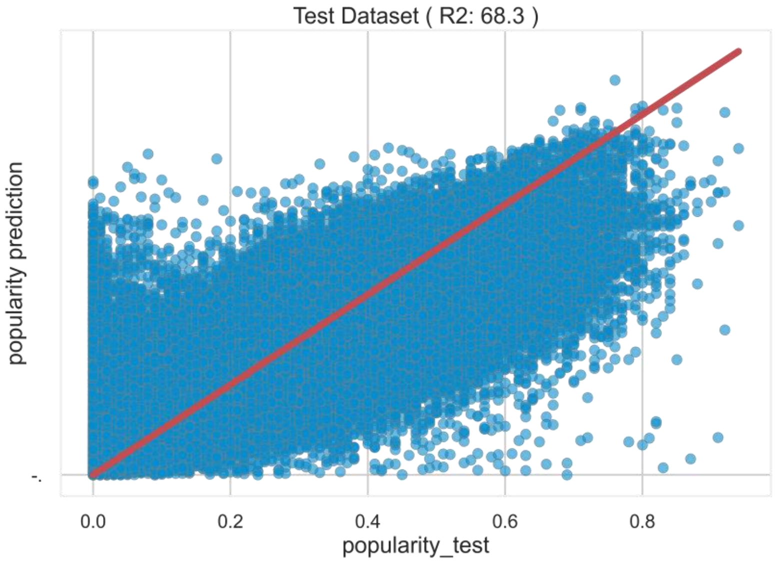
- Clean & transform data
- Test algorithm
- Test feature importance
- Hyperparameter optimization

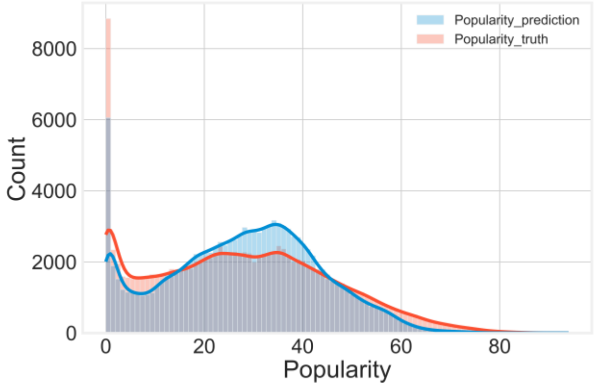


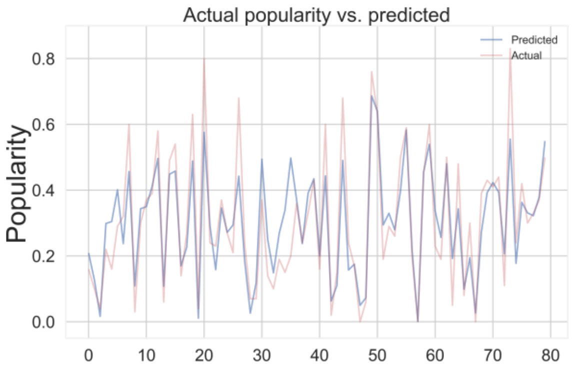


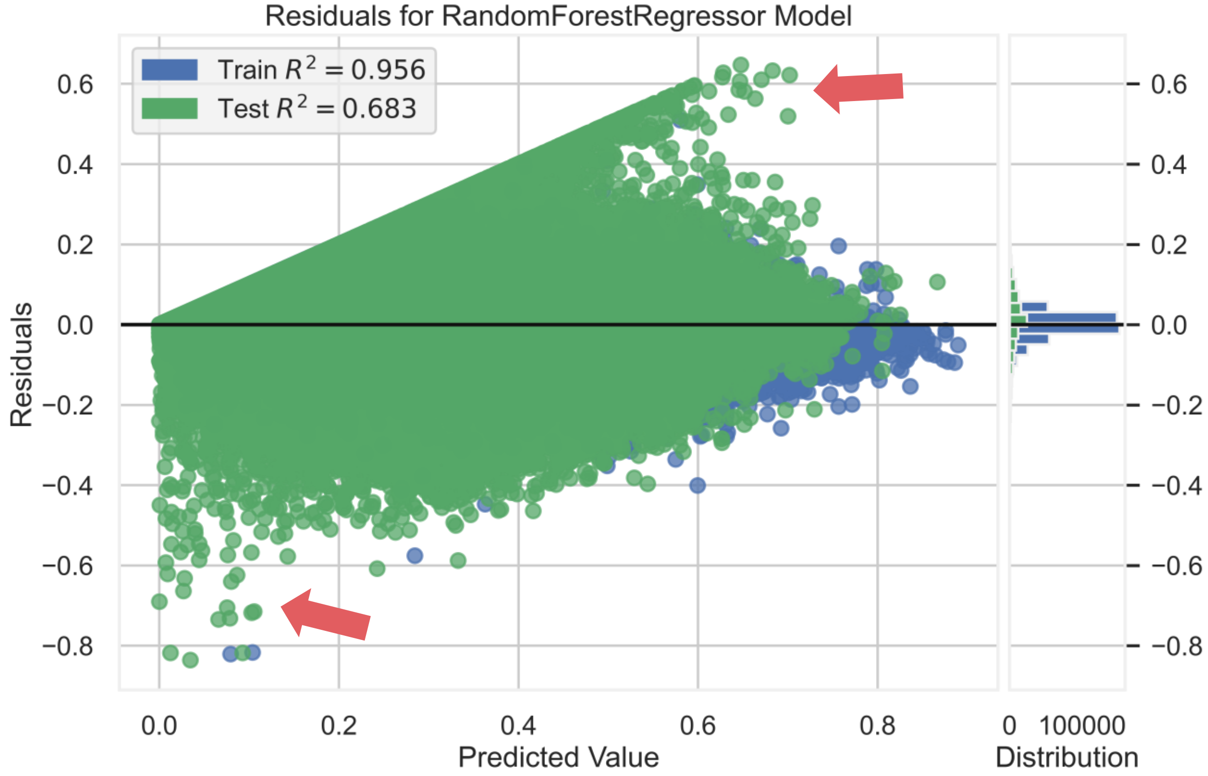
Predict popularity of track
Example of 1 algorithm

- RandomForestRegressor:
  - Feature reduction_12 features only(30 total):
    - R2 lose 0.2% but speed up 18%
  - Hyperparameter Optimization (CV+Randomized Search):
    - N_estimators= 205, min_samples_split= 2, min_samples_leaf= 1
    - max_features= 'auto', max_depth= 98, bootstrap= True
  - Scores(test):
    - R2 score: 68.3 %
    - MAE: 7.4/100









Predict popularity of track – Overview

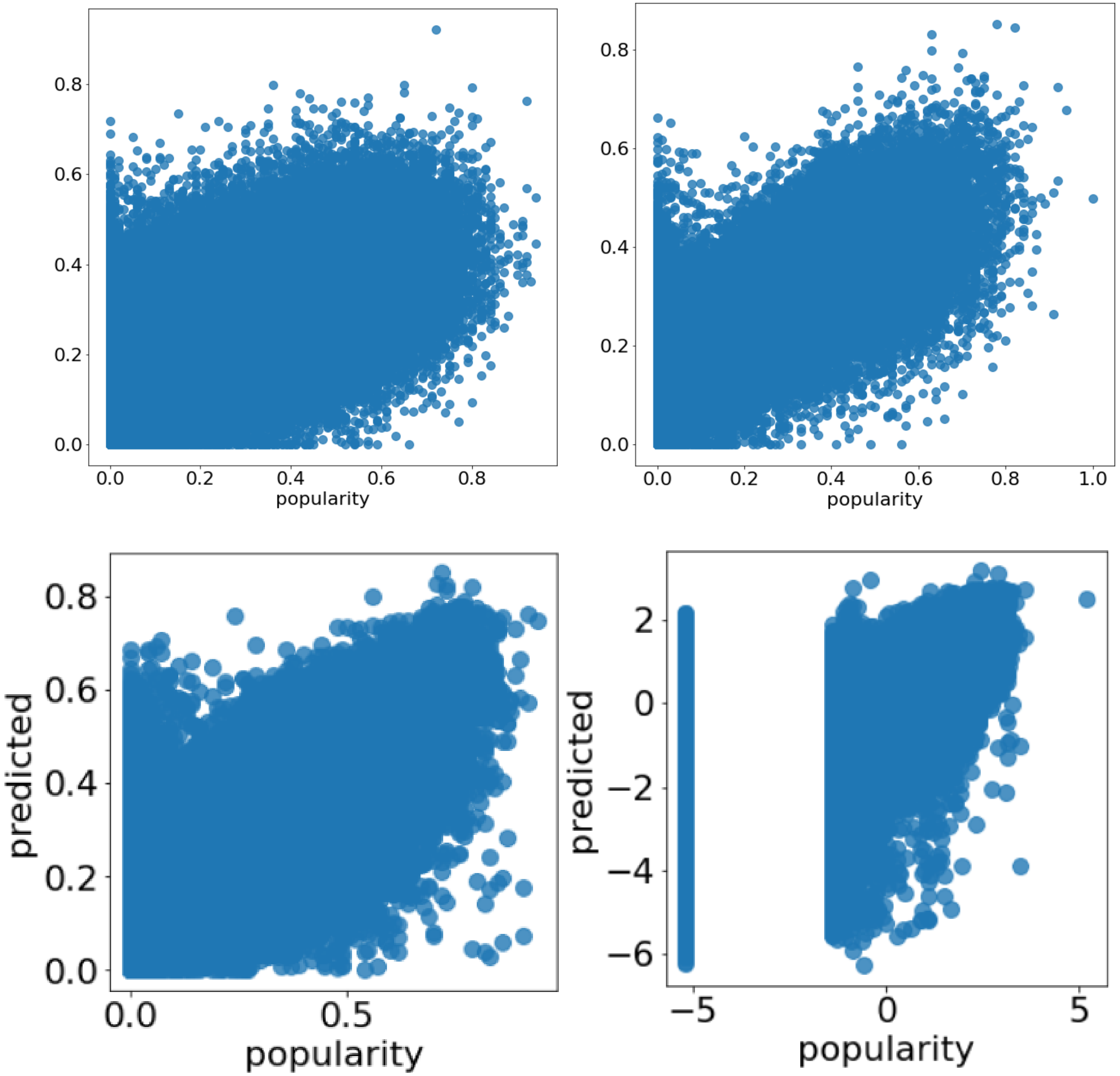
Algorithm	R2 score [%]	MAE	# Features	Optimization?
XGBRegressor	85.5 (Train) 73.4 (Test)	0.40 (Train) 0.50 (Test)	16	Yes, Randomized Search with CV
Decision tree	75.5 (Train) 73.3 (Test)	0.54(Test)	16	No
Scikit Learn RandomForestRegressor*	95.6 (Train)* 68.3 (Test)*	0.03 (Train)* 0.07 (Test)*	12	Yes, Randomized search with CV
MLP	73.0(Train) 70.7 (Test)	0.52 (Train) 0.55 (Test)	all	Yes, GridSearchCV and RandomizedSearchCV
LGBMRegressor (VN)	80.1 (Train) 74.4(Test)	0.46 (Train) 0.51 (Test)	all	No, manually
LGBMRegressor	82.1 (Train) 74.4 (Test)	0.45 (Train) 0.51 (Test)	all	Yes - RandomizedSearch with CV
Kerasregresssor			15	

** in this model popularity was scaled differently*

Similar performance of all algorithms -> probably reached information limit

Predict popularity of track – Impact of Transformation Example: XGBoost

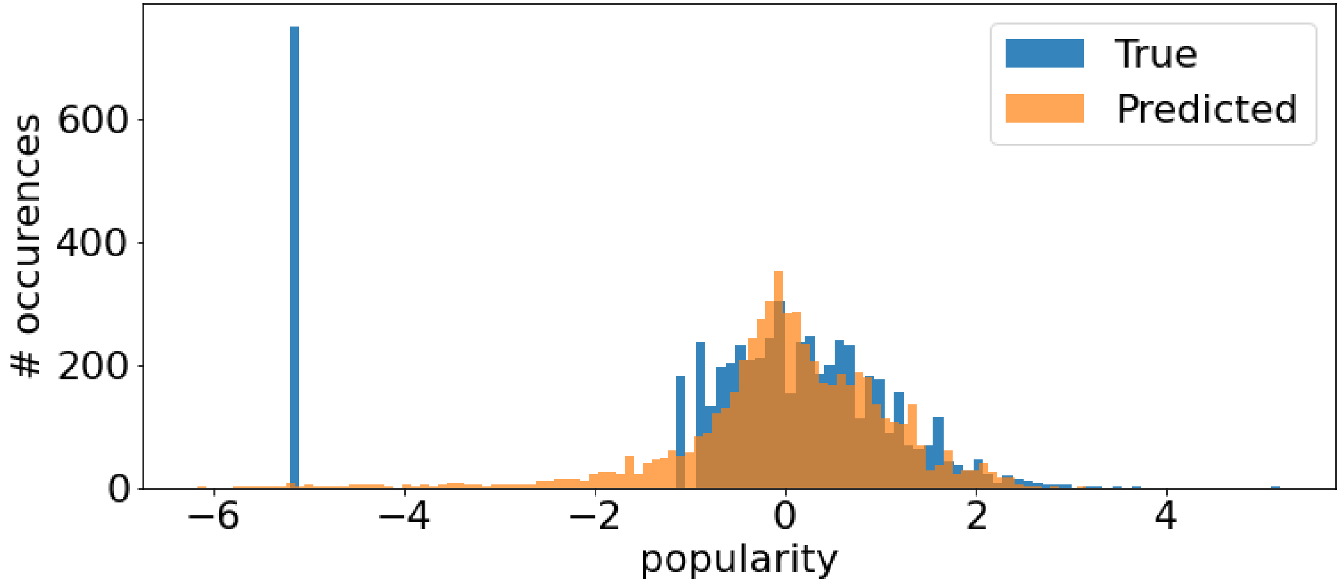
- Consider only musical features:
  - R2: 45.0 %
- Add artist by ID (>100.000 classes)
  - R2: 54.6 %
- Represent artist by mean popularity (+std, + range) + OHE + Scaling:
  - R2: 67.7 %
- Quantile Transform Popularity:
  - R2: 73.4

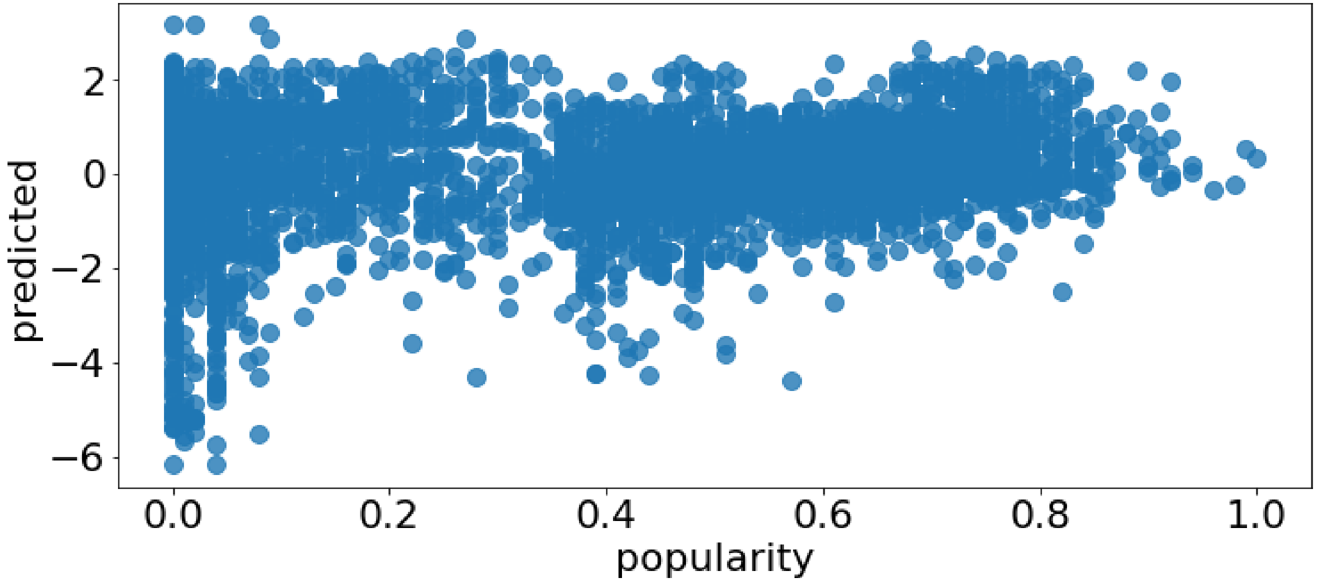


Predict popularity of track – Future predictions Example: XGBoost

Goal: Predict 1 year into the future:

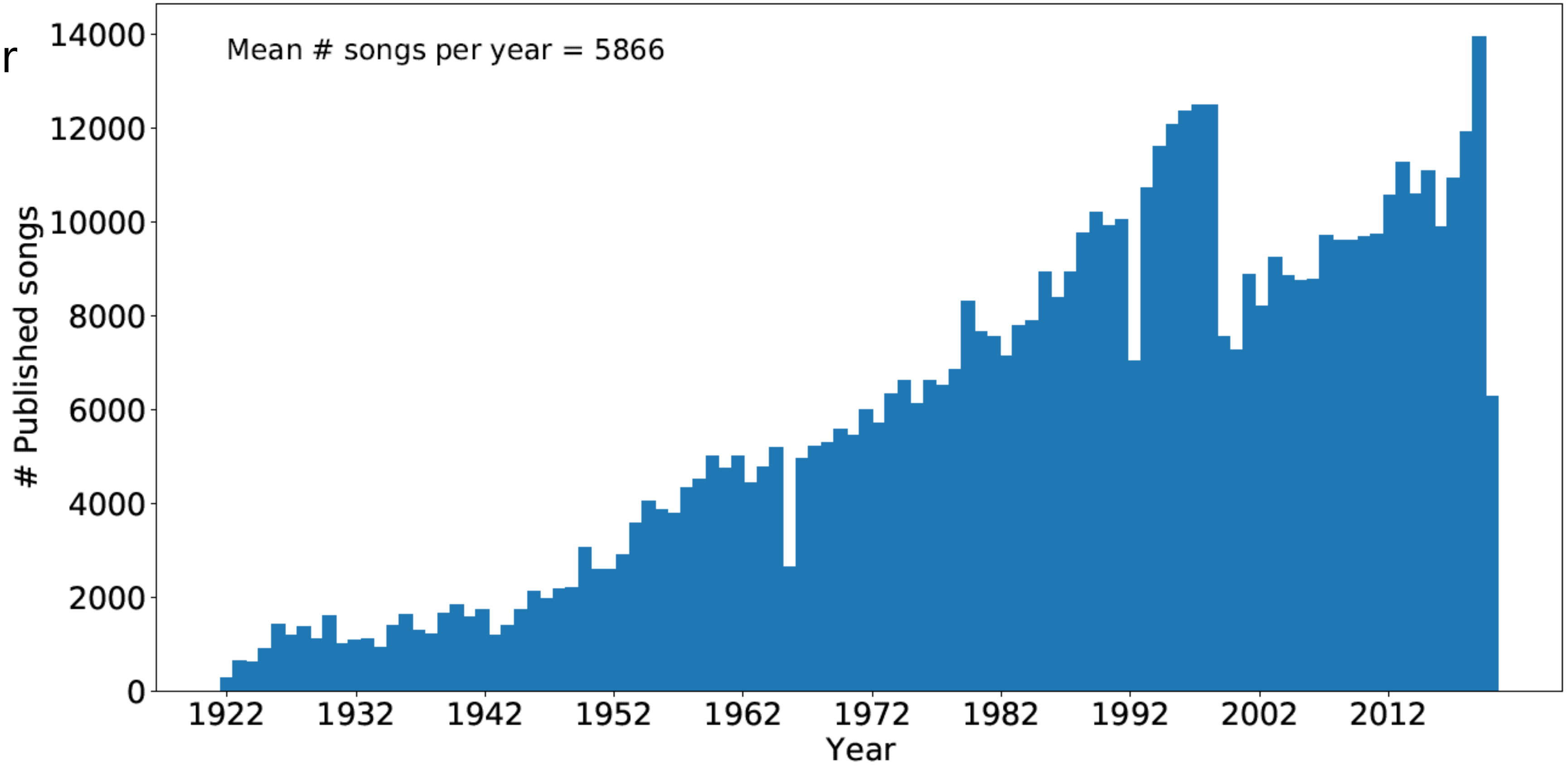
- XGBRegressor algorithm from general prediction
- Train on 2010-2020 data
- Predict: 2021 data
- Result:
  - MAE: 0.39 train <-> 0.40 general
  - MAE: 1.45 test <-> 0.50 general





Predict release year of track

- Highly unbalanced
- Resampling to balance data
  - Random Oversamper
  - Random Undersampler
  - SMOTE
  - “Bootstrapping”

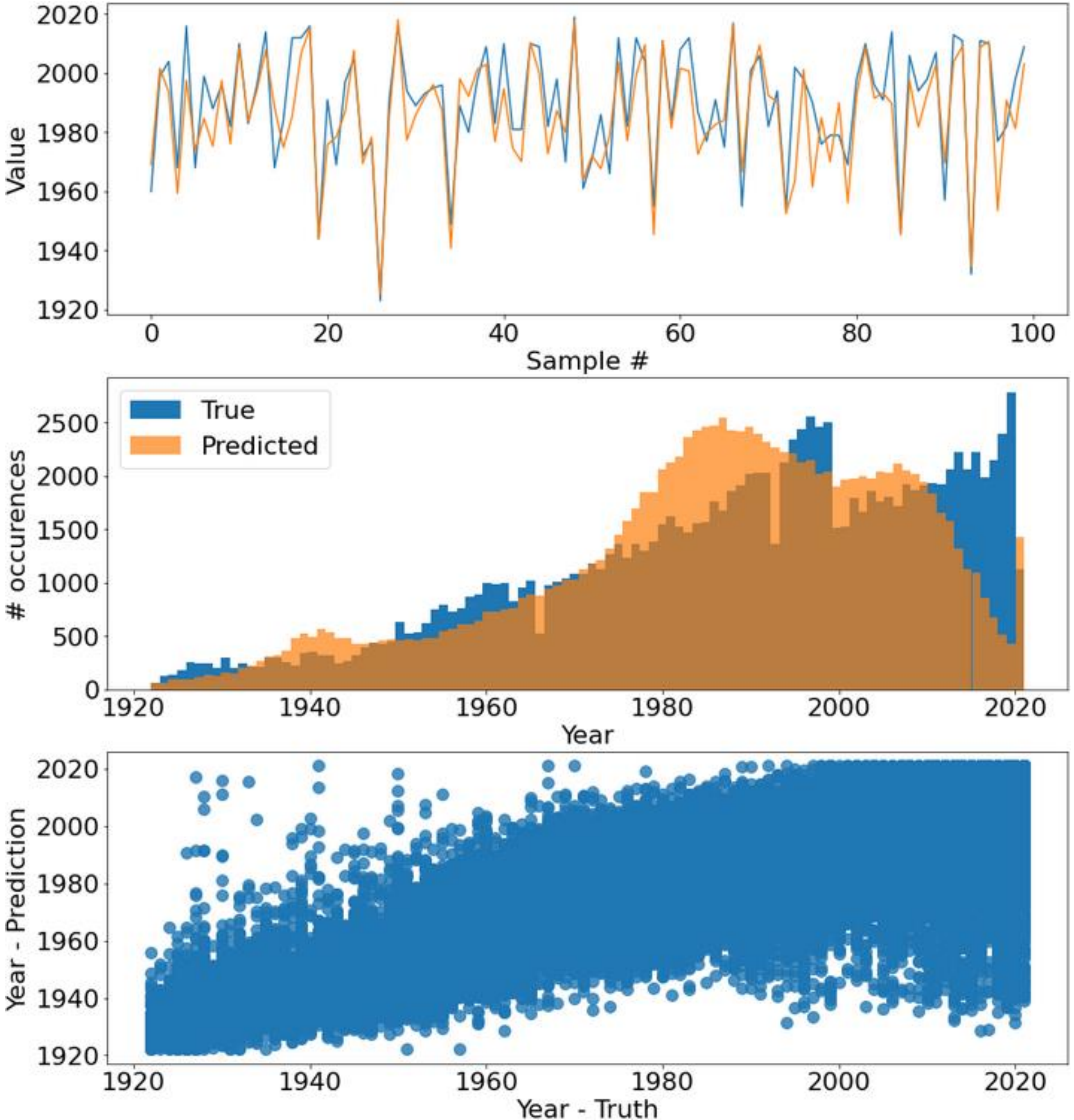


Predict release year of track

Algorithm	R2 score	MAE [years]	# Features	Optimization?	Comment
XGBRegressor	**92.4 % (train)** **74.3 % (test)**	**5.7** **8.2**	**16**	**Yes, RandomizedSearch with CrossValidation**	**"Bootstrapping", 2000 / year**
KerasRegressor	81.0 % (train) 68.8 % (test)	8.9 8.8	16	Yes, RandomizedSearch with CrossValidation	"Bootstrapping", 2000 / year
Decision tree	70.5 % (train) 66.7 % (test)	9.6			
Random forest	96.4% (train) 73.0% (test) 72.8% (test)	8.6	30 30 11		
LightGBM	92.1 % 72.1 % (test)	4.71 8.77	all	Optimized using GridSearchCV	Resampled using SMOTE, but it didn't help at all
MLP	62.3 % 60.9 % (test)	10.56 10.71	all	Manually	
Algorithm	**Accuracy**	**MAE**	**# Features**	**Optimization?**	**Comment**
LightGBM Classification (20 y)	65.7 (train) 60.5 (test)			GridSearchCV	
XGBClassifier (10 y)	49 % (train) 41 % (test)		22	Yes, RandomSearchCV	Resampled, no artist transform or OHE

Predict release year of track - Evaluation

- Best performance observed for:
  - XGBoostRegressor
  - Resampled training data (2000 / year)
  - Hyperparamter optimization
  - Scores:
    - R2 score: 74.3 %
    - MAE: 8.2 years
  - Feature reduction – 16 features only:
    - **'explicit'**, 'energy', 'artists_std', **'artists'**, 'duration_ms', **'popularity'**, 'artists_minmax', 'loudness', 'danceability', 'mode', 'speechiness', **'acousticness'**, 'liveness', 'tempo', 'valence', 'key_1'



15/06/2021

Conclusions / Summary:

- Successfully implemented algorithms for release year and popularity predictions
- Main performance improvement:
  - Both: Replace artist by mean popularity
  - Popularity: Final quantile transform “normal”
  - Release year: Resampling with replacement (“Bootstrapping”)
- For us: Tree-based > NN, Guess:
  - optimization (Layers, neurons + Hyperparameter optimization not optimal)
- Popularity and release year prediction similar in performance
- Future popularity prediction: Performs worse (expected)

Appendix Final project – Group 10

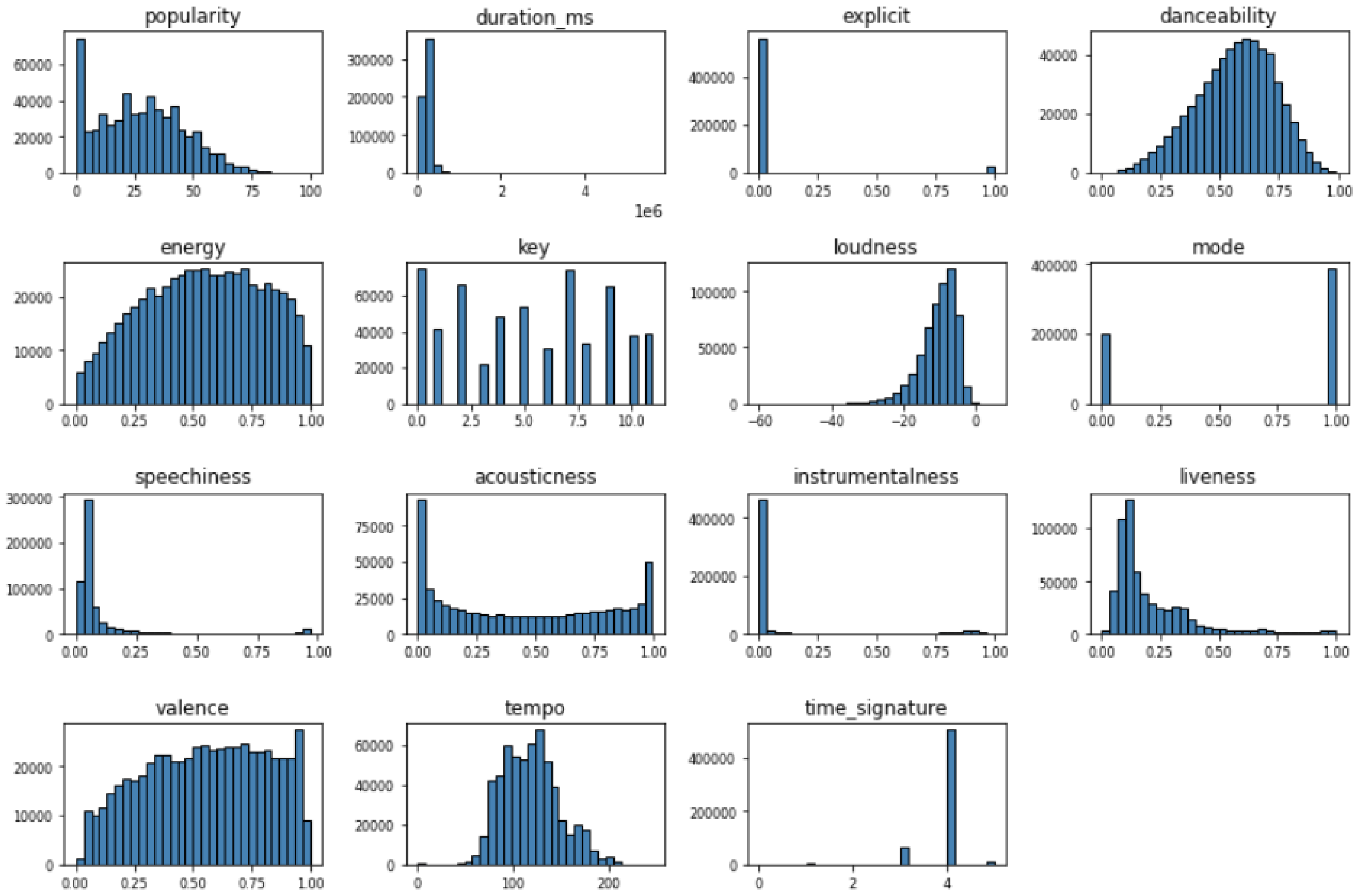
Georgia Anyfantaki,
Jun Jia,
Rebecca Schmieg &
Valerii Novikov

"

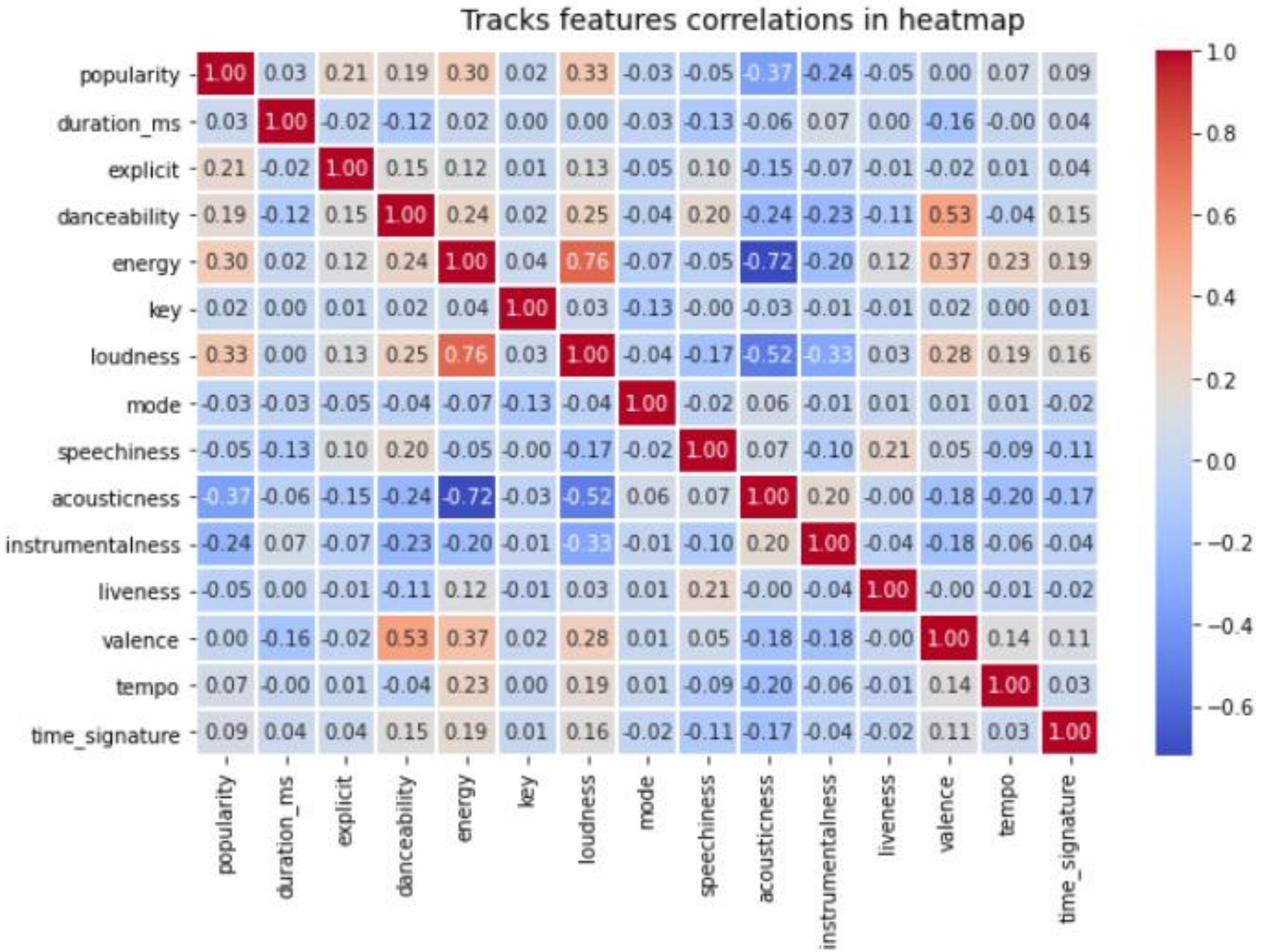
Everyone contributed in the programming, discussion and presentation preparation

Group 10 – Contribution statement

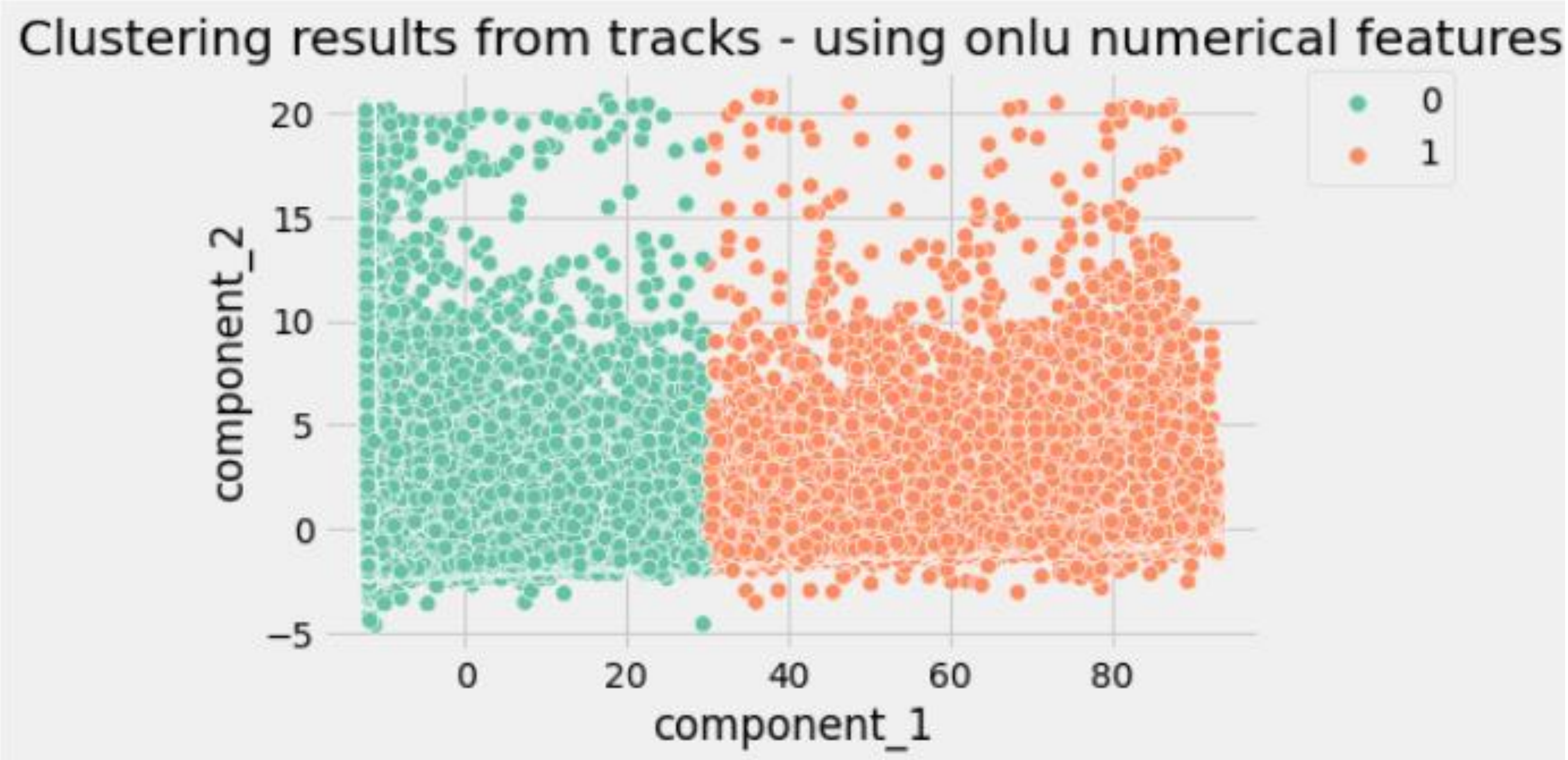
Visualization and inspection of the features – Histograms of all numerical features



Visualization and inspection of the features – Heat map of Pearson's correlations between all numerical features

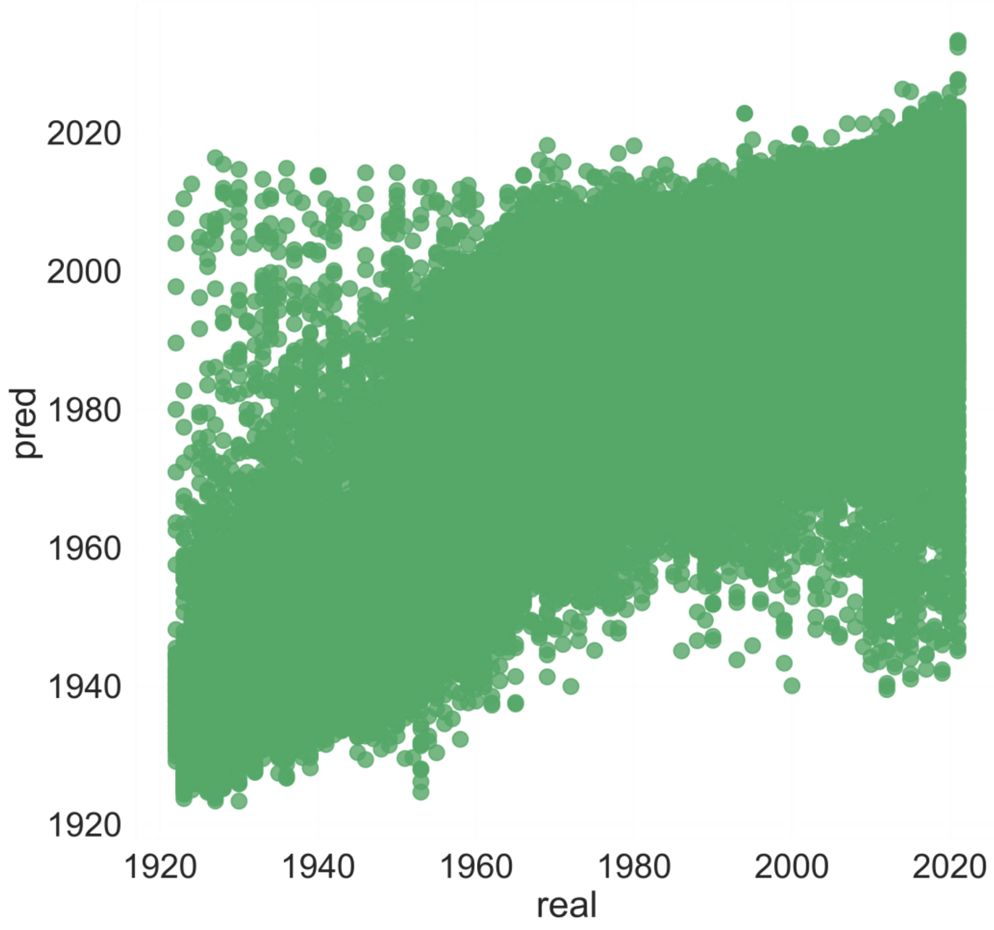


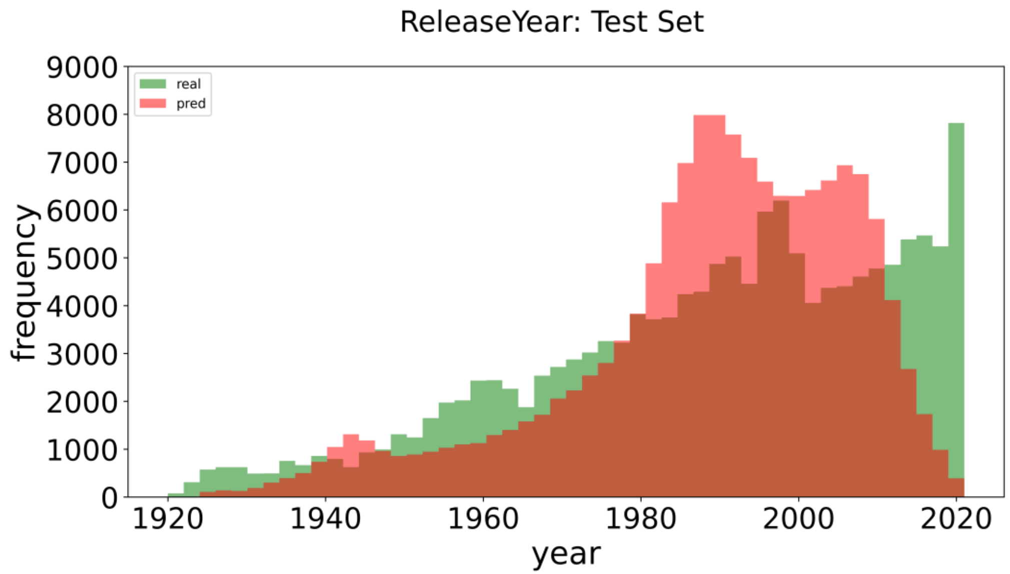
Visualization and inspection of the features – PCA and Clustering

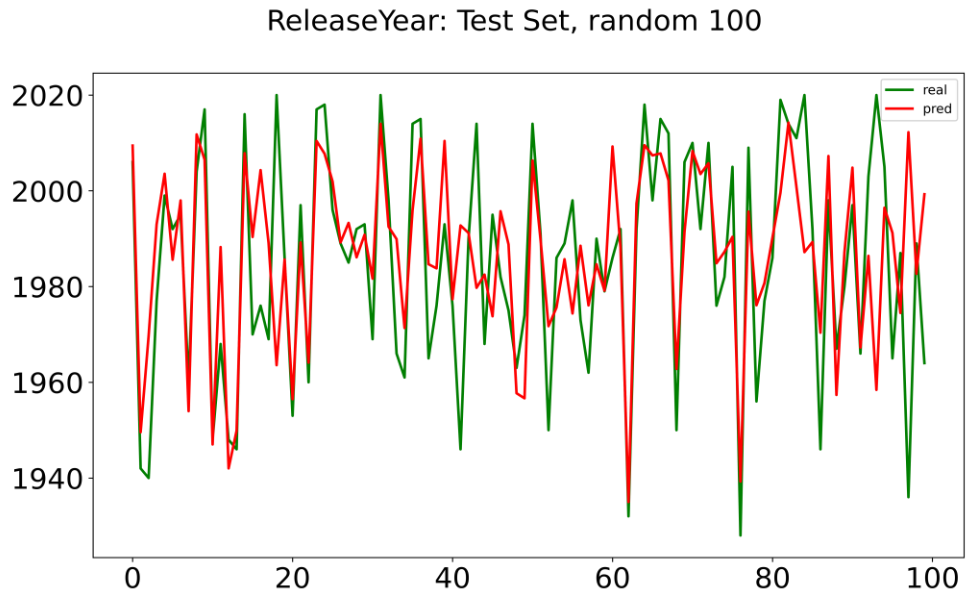


Clustering was performed after PCA (n_components =2) using kmeans - Did not really work that well!

Release year prediction – LGBMRegressor (VN)







Release year prediction – LGBMRegressor (VN)

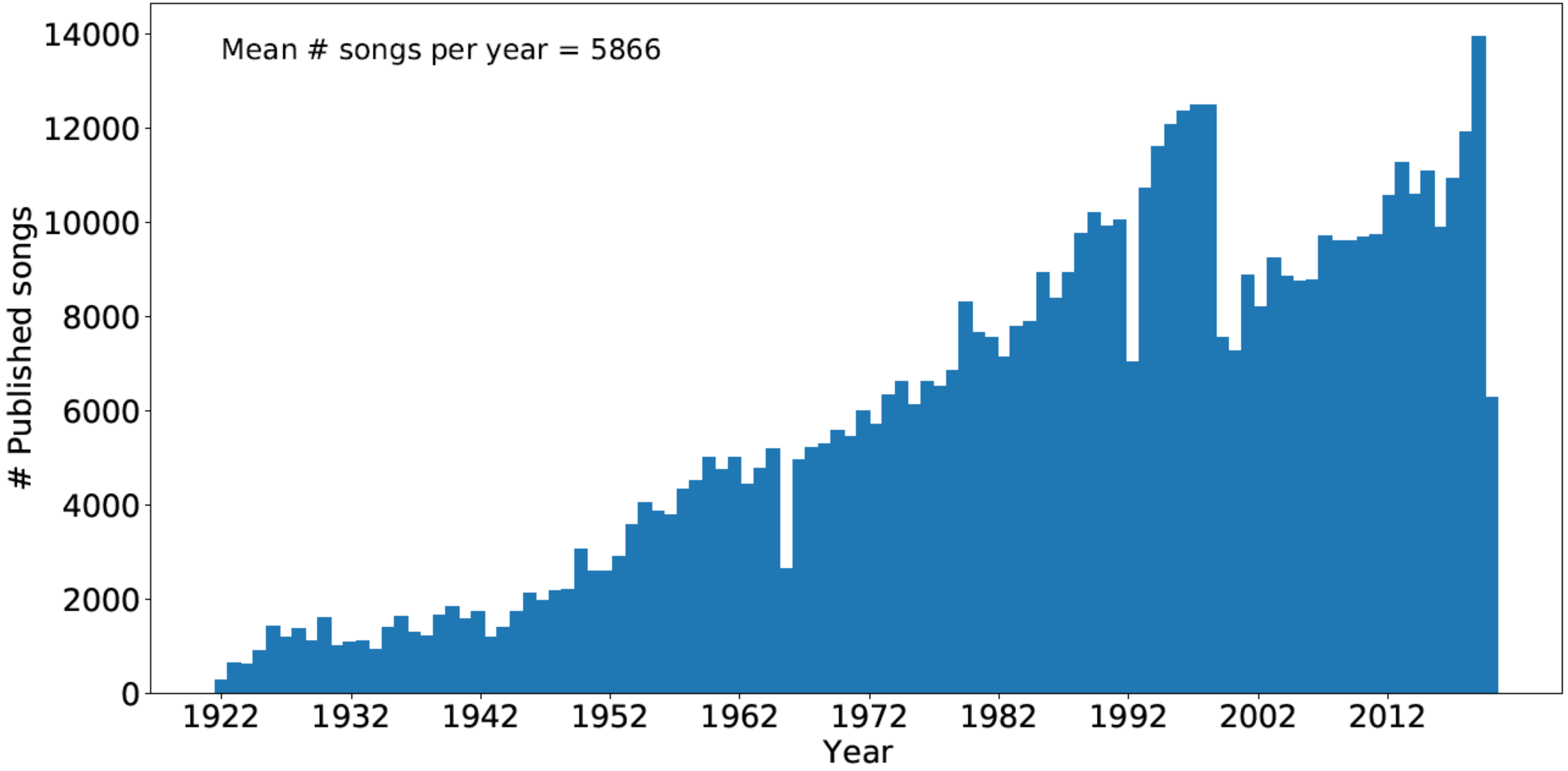
```python
# GRID Search with lightgbm

def algorithm_pipeline(X_train_data, X_test_data, y_train_data, y_test_data,
                       model, param_grid, cv=3, scoring_fit='neg_mean_squared_error',
                       do_probabilities = False):
    gs = GridSearchCV(
        estimator=model,
        param_grid=param_grid,
        cv=cv,
        n_jobs=-1,
        scoring=scoring_fit,
        verbose=2
    )
    fitted_model = gs.fit(X_train_data, y_train_data)

    if do_probabilities:
      pred = fitted_model.predict_proba(X_test_data)
    else:
      pred = fitted_model.predict(X_test_data)

    return fitted_model, pred
```

```python
from sklearn.model_selection import GridSearchCV

model = lgb.LGBMRegressor(n_jobs=-1)
param_grid = {
           'objective': ['regression'],
                         'boosting_type': ['gbdt'],
          'metric': ['rmsle'],
                        # metric= 'mean_squared_error',
                         'learning_rate': [0.015,0.03],
          'num_leaves': [150,400,650],
          'max_depth': [25,100,250],
          'n_estimators': [100,500,2000],
                'num_boost_round': [300,600]
}
model, pred = algorithm_pipeline(X_train_reg_0, X_test_reg_0, y_train_reg, y_test_reg, model,
                                 param_grid, cv=3, scoring_fit='r2')

print(model.best_score_)
print(model.best_params_)
```

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   49.7s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  5.5min
[Parallel(n_jobs=-1)]: Done 324 out of 324 | elapsed: 12.5min finished

[LightGBM] [Warning] num_iterations is set=600, num_boost_round=600 will be ignored. Current value: n
um_iterations=600
0.6693836717273411
{'boosting_type': 'gbdt', 'learning_rate': 0.015, 'max_depth': 25, 'metric': 'rmsle', 'n_estimators':
100, 'num_boost_round': 600, 'num_leaves': 150, 'objective': 'regression'}
```

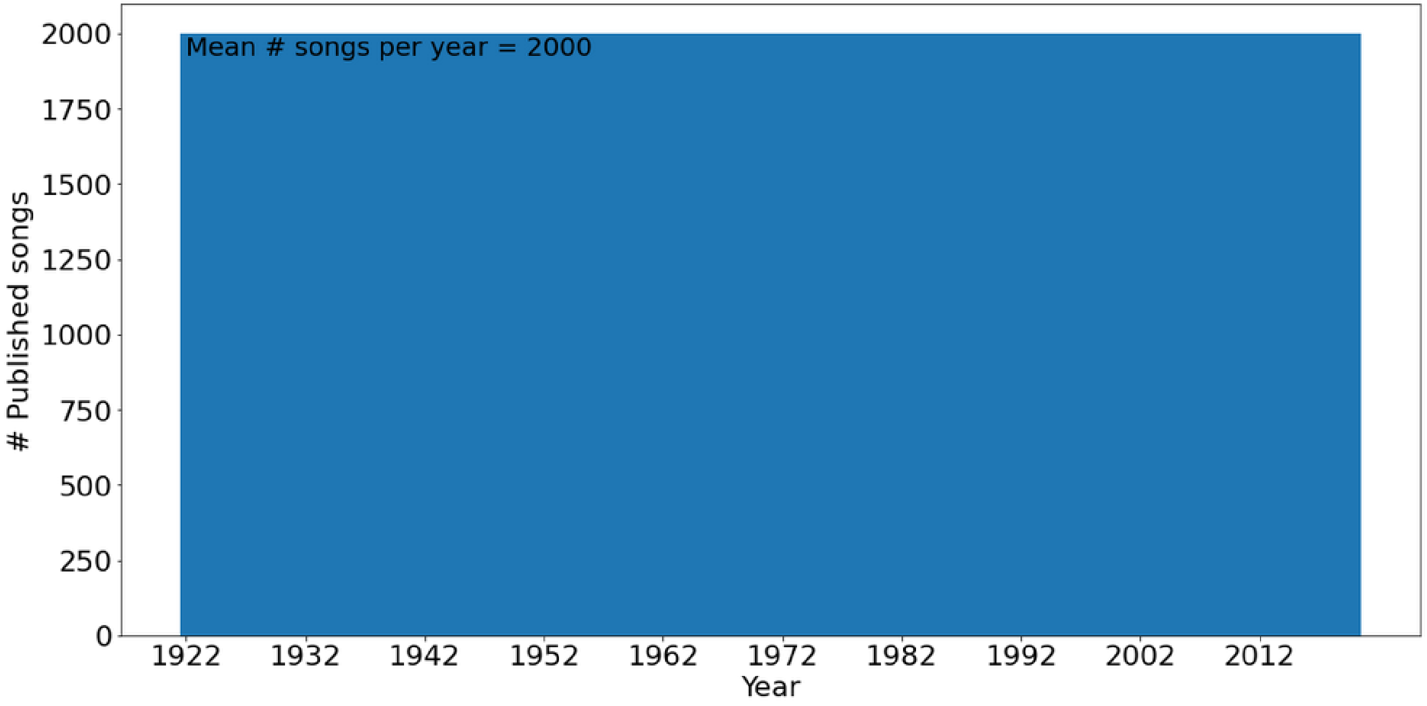
Release year prediction – Approach 3

- Problem: Highly imbalanced data:



Release year prediction – Approach 3

- Problem: Highly imbalanced data
- Try balancing data with random oversampling / SMOTE / random undersampling:
  - XGB and NN did not work well
- Try balancing by picking samples with replacement:
  - 2000 samples per year (3-4x for lowest year, ⋯)



Only last method proved to be working well

Release year prediction – Approach 3
Keras Classifier

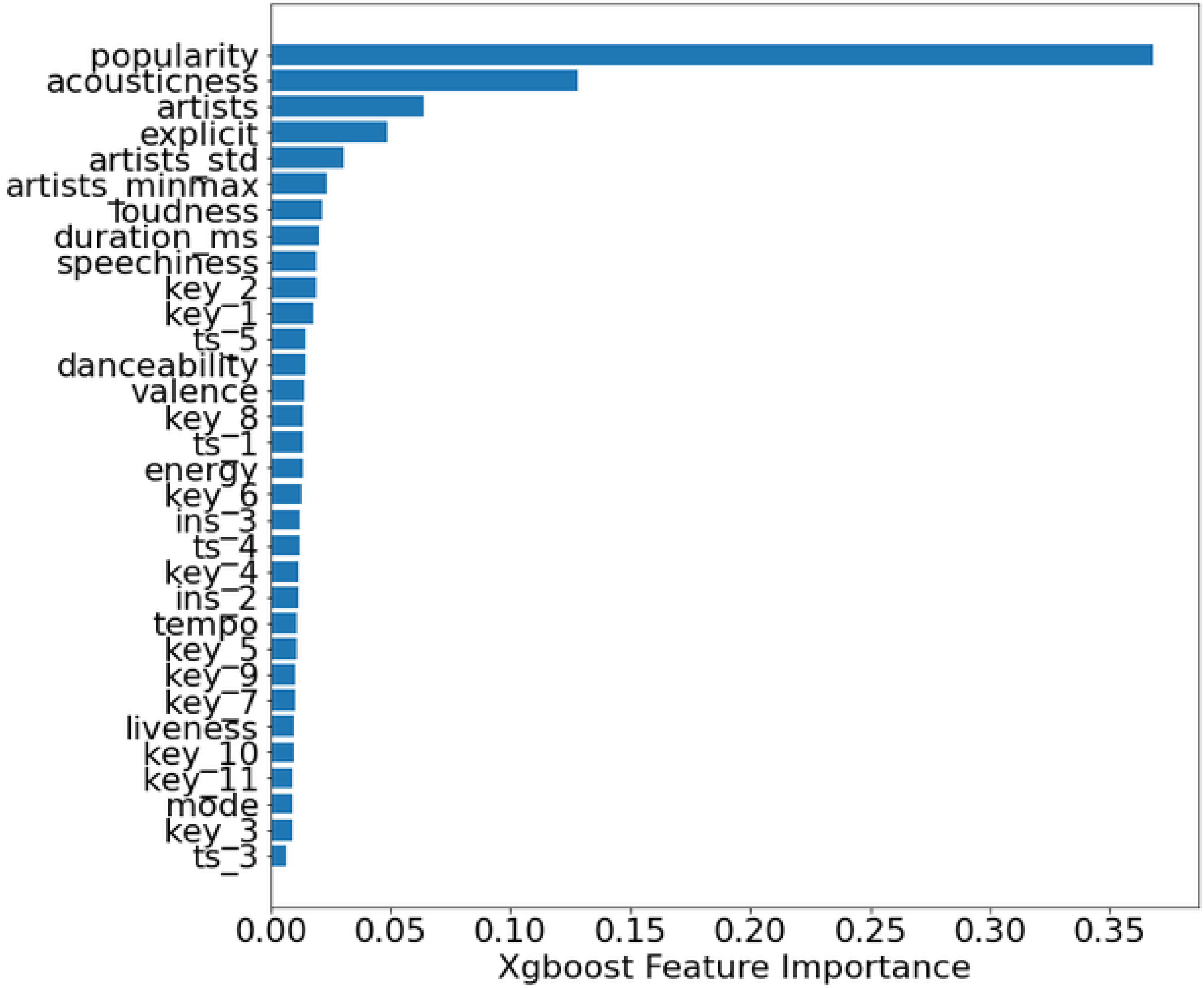
- Using Keras Classifier with multi-class output
- 100 possible outputs performs bad -> cluster years into decades and hence 10 categories
- Best performance:
  - R2 score on test: 50 %
  - Accuracy: 0.39

This seemed to not be a good way of handling this task. Switching to **Regression** instead.

These results did not incorporate the artist transformances.

Release year prediction – Approach 3 XGBRegressor

- Clean data (remove duplicates, remove single outlier at 1900,⋯)
- Add features based on artist mean, std and (max-min) popularity of training set
- Regression with XGB
  - Reduce features based off feature importance -> best choice seem to be keeping 16 parameters
  - R2: 88.2 (train) and 72.2 (test)
  - MAE (in years): 7.18 (train) and 8.65 (test)



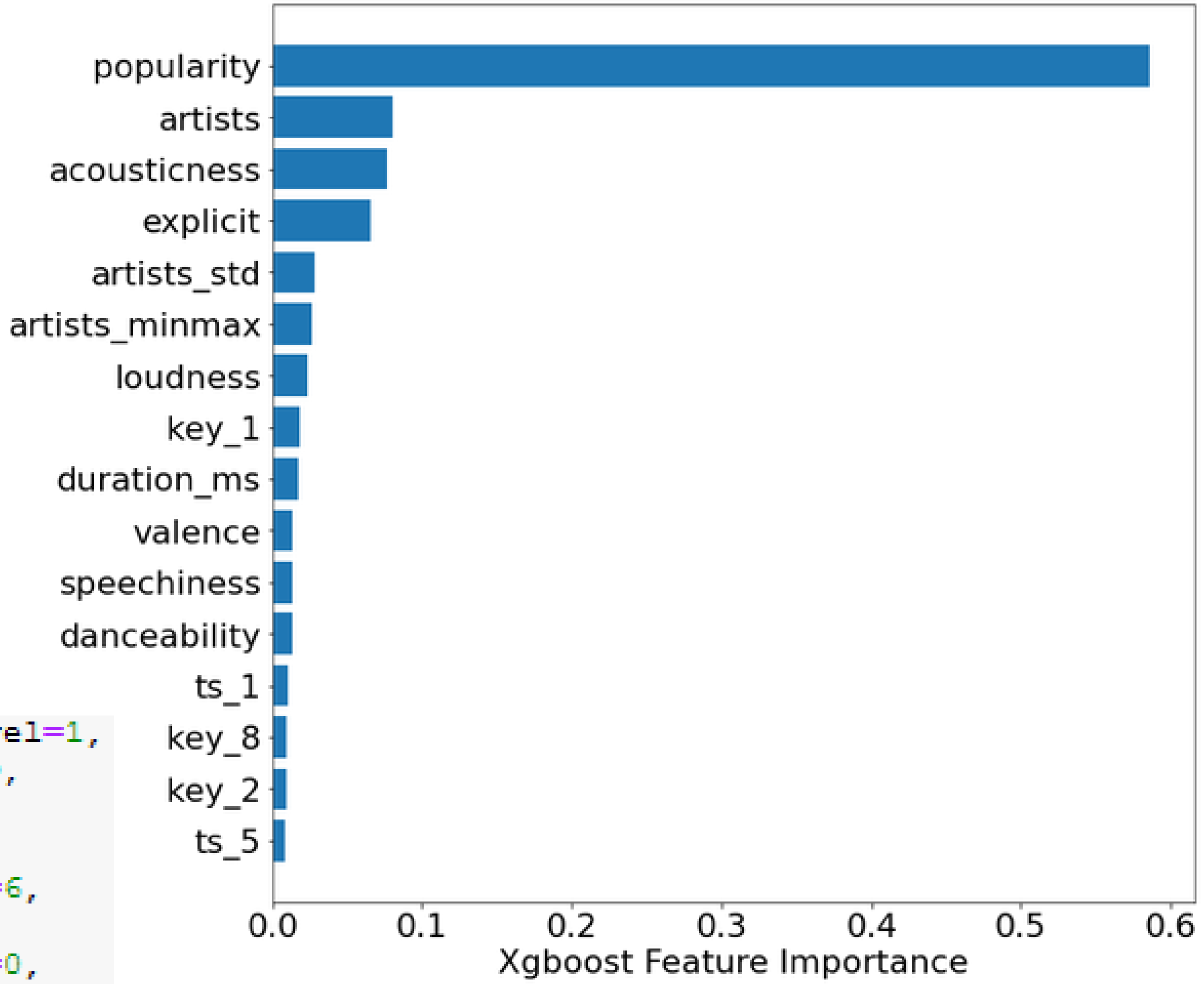
Release year prediction – Approach 3 Improvements for XGBRegressor

- Rerun with 16 features only:

  'artists_std', 'explicit', 'artists_minmax', 'artists', 'duration_ms', 'energy', 'popularity', 'loudness', 'danceability', 'mode', 'speechiness', 'acousticness', 'liveness', 'valence', 'tempo', 'key_1'

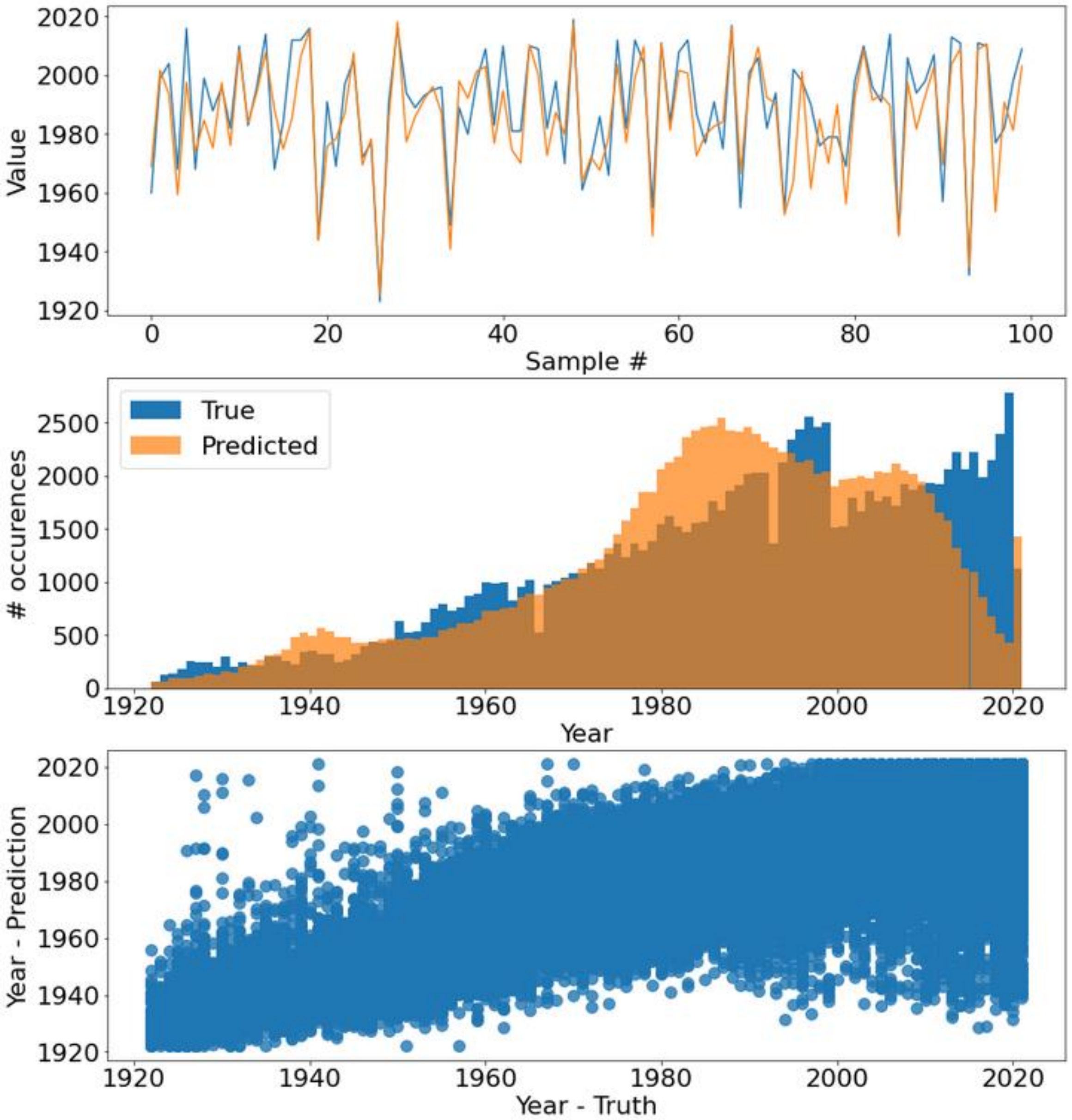
- Hyperparameter optimization:
  - 100 different fits + CV
  - Best parameters:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
colsample_bynode=1, colsample_bytree=1, eta=0.4170423469764199,
eval_metric='mae', gamma=0.3997113330682822, gpu_id=-1,
importance_type='gain', interaction_constraints='',
learning_rate=0.4170423469764199, max_delta_step=0, max_depth=6,
min_child_weight=4, monotone_constraints='()',
n_estimators=346, n_jobs=8, num_parallel_tree=1, random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
subsample=0.7878054415851274, tree_method='exact',
validate_parameters=1, verbosity=2)
```



Release year prediction – Approach 3
Final result for XGBRegressor

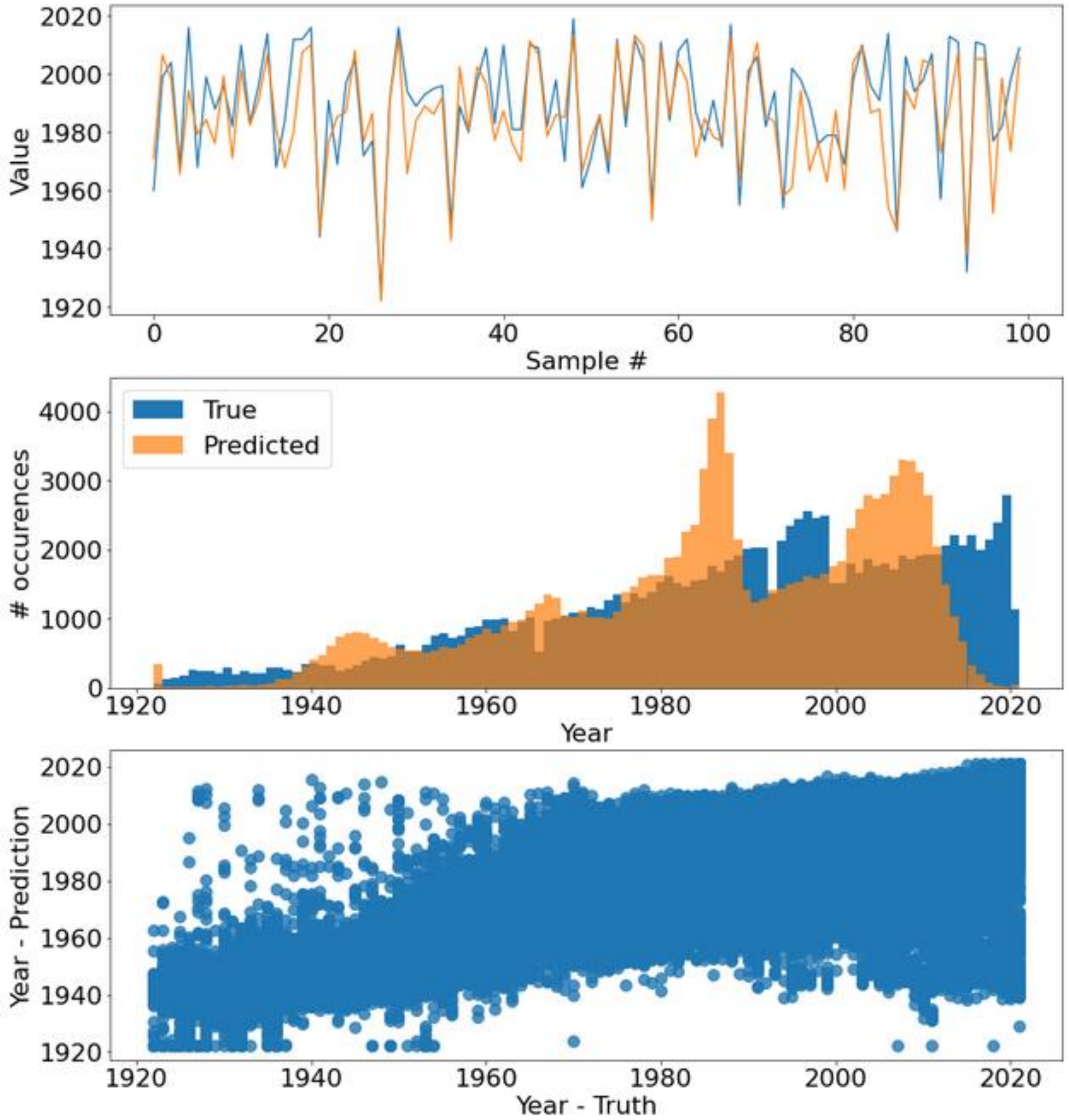
- Best performance:
  - Train set:
    - R2 score: 92.4 %
    - MAE: 5.68 [years]
  - Test set:
    - R2 score: 74.3 %
    - MAE: 8.24 [years]



This algorithm (optimized and data processing) turned out to be the best performing one for the release year prediction

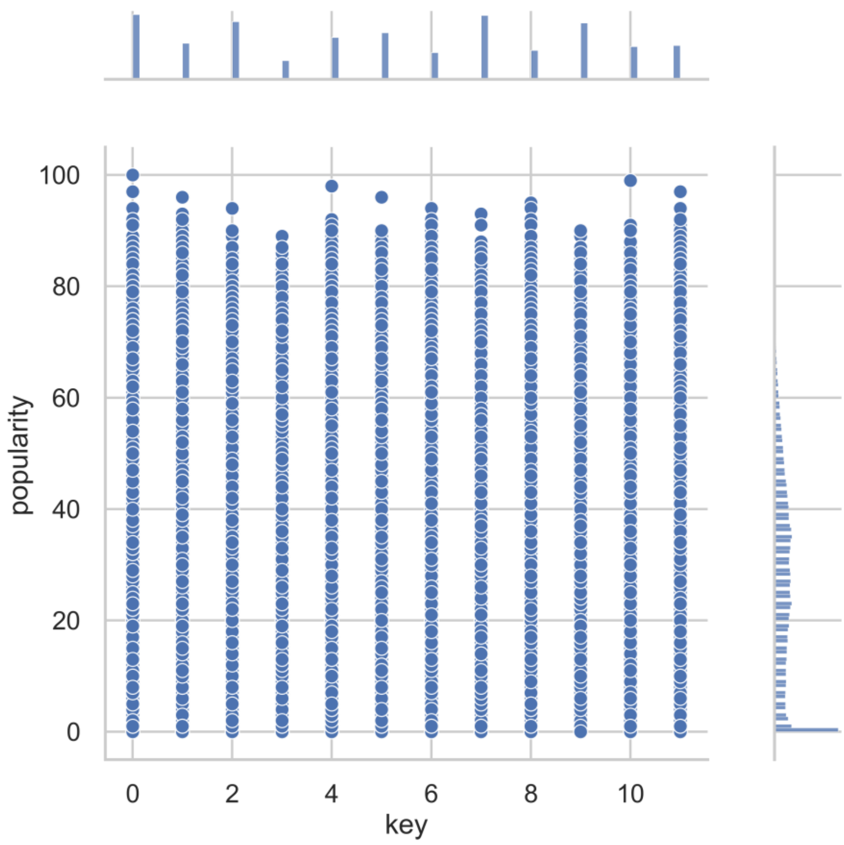
Release year prediction – Approach 3
Final result for KerasRegressor

- Based on knowledge from XGBRegressor, improving KerasRegressor as well
- Using 16 best features
- Hyperparameter optimization
- Best performance:
  - Train set:
    - R2 score: 81.0 %
    - MAE: 8.8 [years]
  - Test set:
    - R2 score: 68.8 %
    - MAE: 8.9 [years]
  - **Not as good as the XGBRegressor performance!**



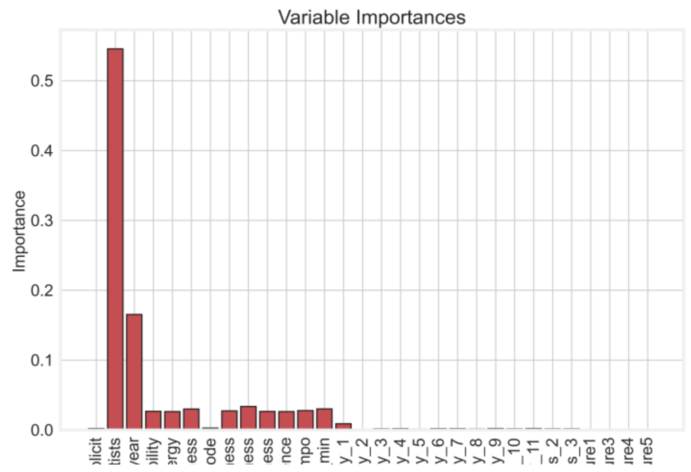
Predict popularity of a track

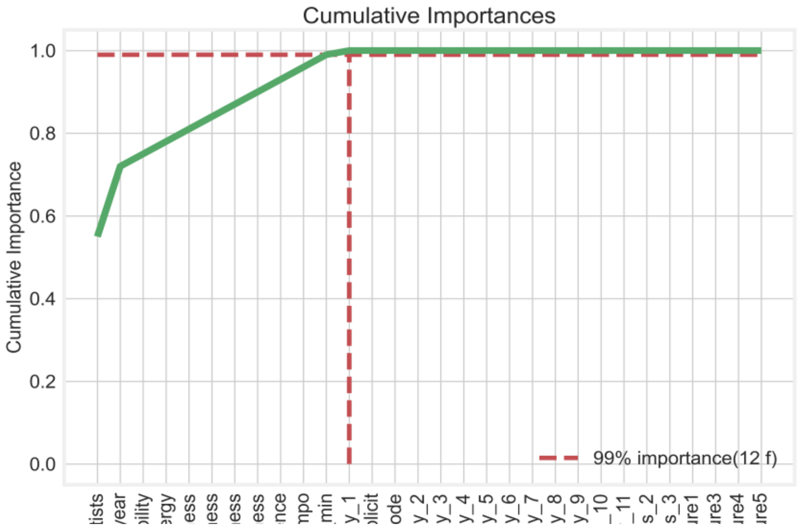
Discrete scattering plot encourages us to apply one hot encoding method on some features




```
data_track.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 586672 entries, 0 to 586671
Data columns (total 20 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   id                586672 non-null  object
 1   name              586601 non-null  object
 2   popularity        586672 non-null  int64
 3   duration_ms       586672 non-null  int64
 4   explicit          586672 non-null  int64
 5   artists           586672 non-null  object
 6   id_artists        586672 non-null  object
 7   release_date      586672 non-null  object
 8   danceability      586672 non-null  float64
 9   energy            586672 non-null  float64
 10  key               586672 non-null  int64
 11  loudness          586672 non-null  float64
 12  mode              586672 non-null  int64
 13  speechiness       586672 non-null  float64
 14  acousticness      586672 non-null  float64
 15  instrumentalness  586672 non-null  float64
 16  liveness          586672 non-null  float64
 17  valence           586672 non-null  float64
 18  tempo             586672 non-null  float64
 19  time_signature    586672 non-null  int64
dtypes: float64(9), int64(6), object(5)
memory usage: 89.5+ MB
```



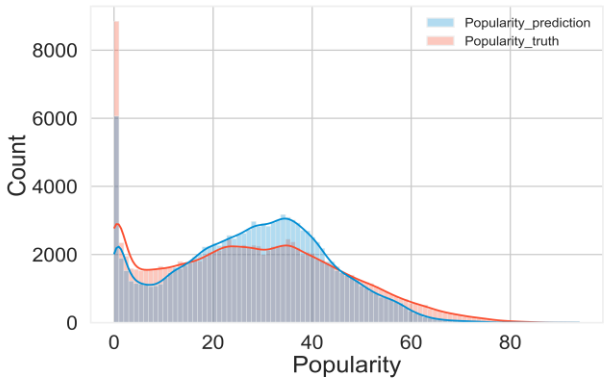


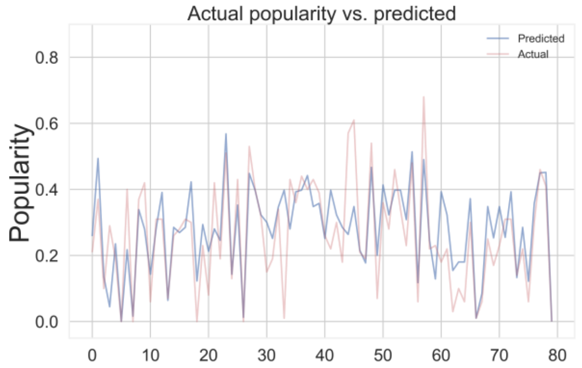
Feature importances

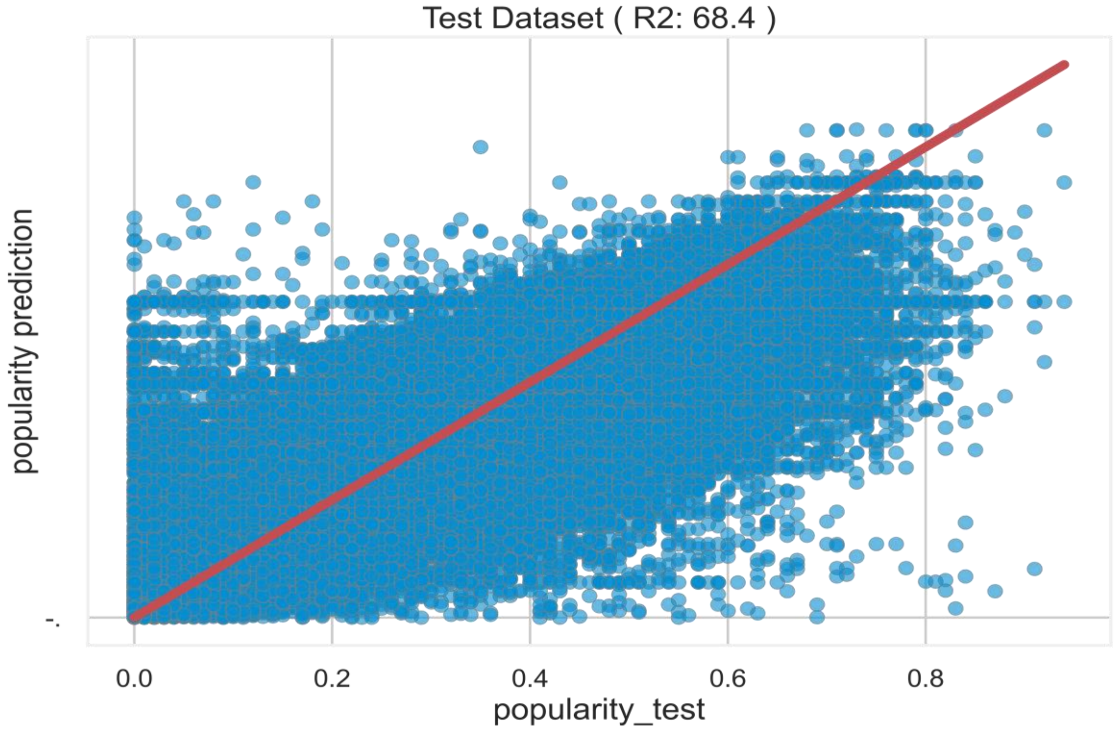
Popularity prediction – Decision tree

- Decision Tree Regressor:
  - Scores(test):
    - R2 score: 68.4 %
    - MAE: 7.7/100
  - Feature Reduction (None 30):
  - Hyperparameter Optimization (Grid Search):
    - cv, n_jobs, verbose, max_depth, max_features, min_sample_split, min_samples_leaf

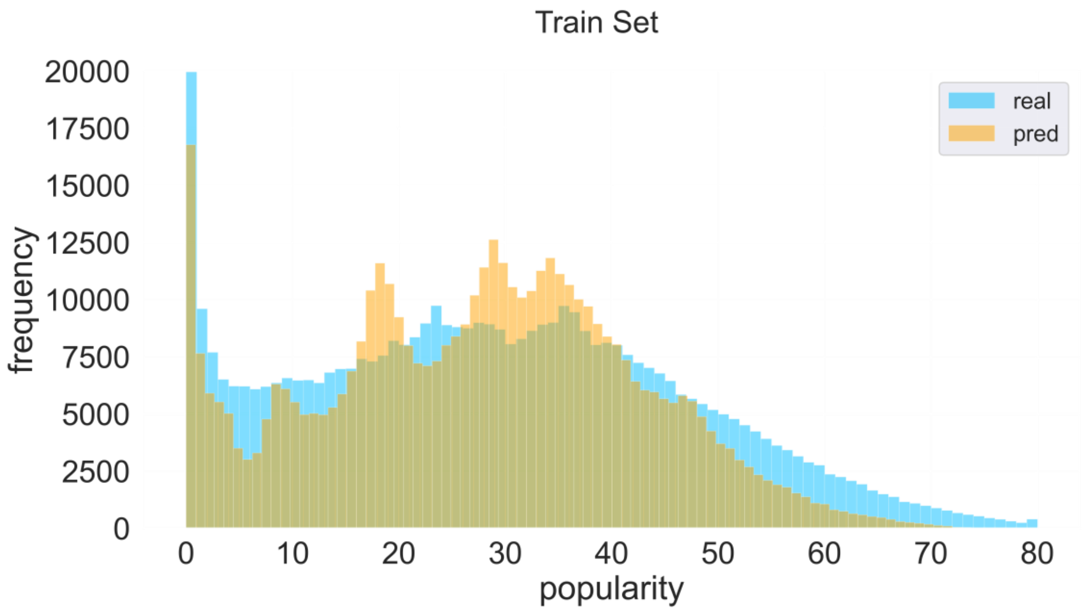
Almost the same precision as the Random forest regresion results

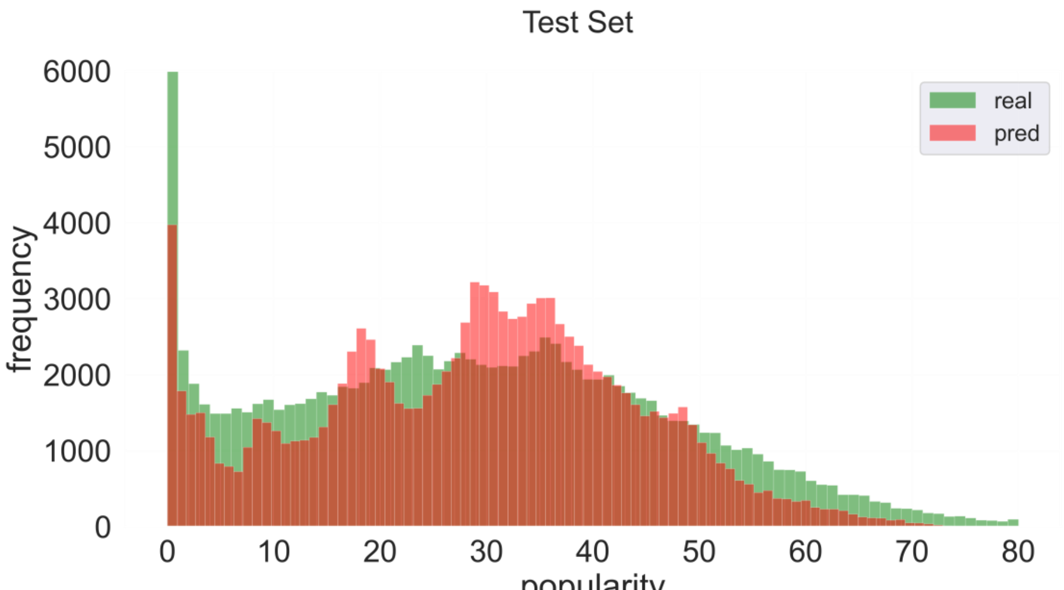


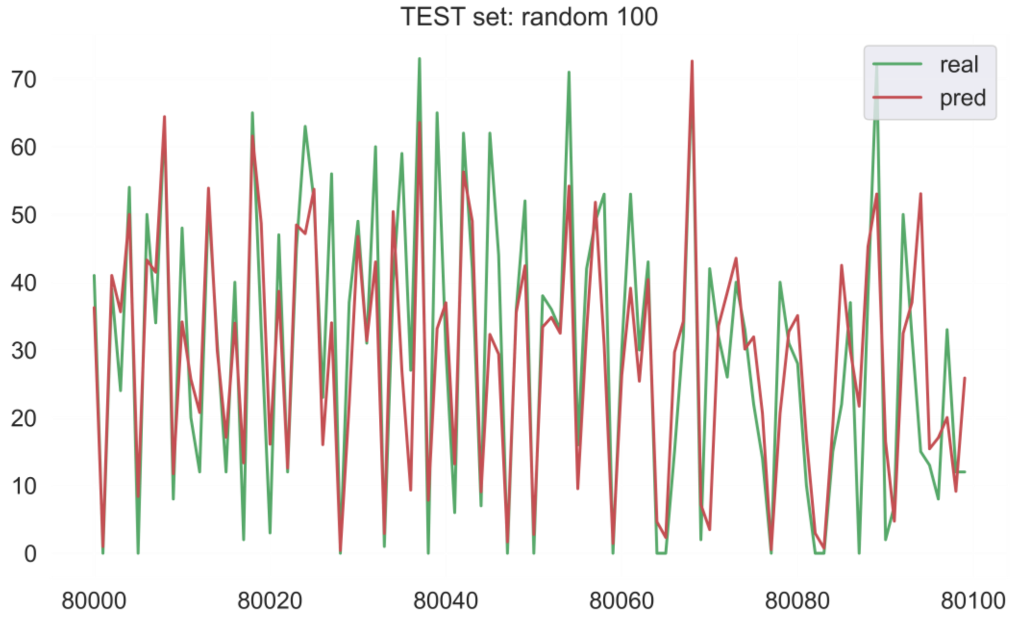


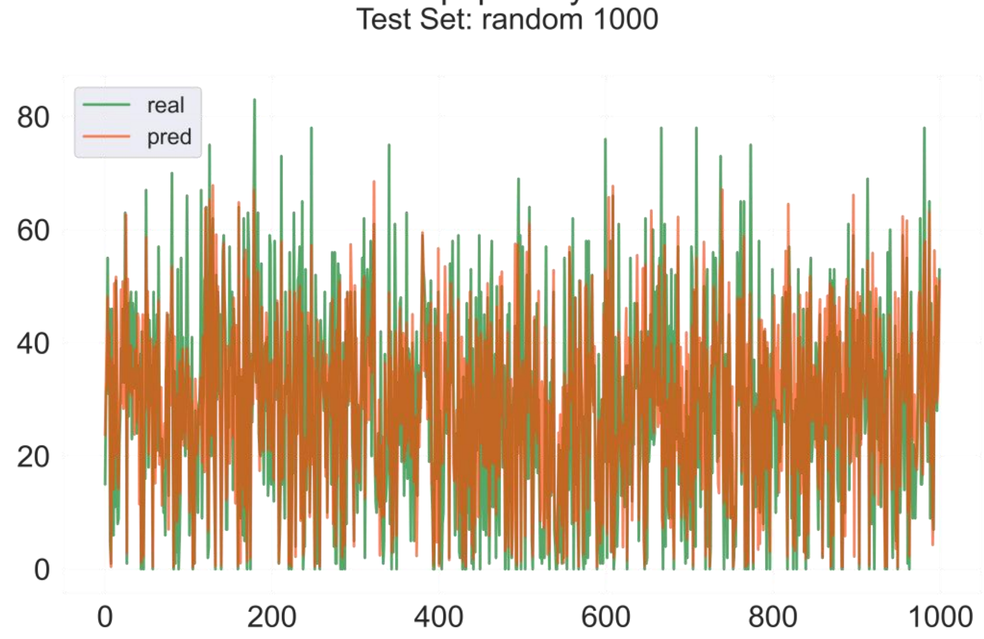


VN: Popularity prediction – MultiLayer Perceptron

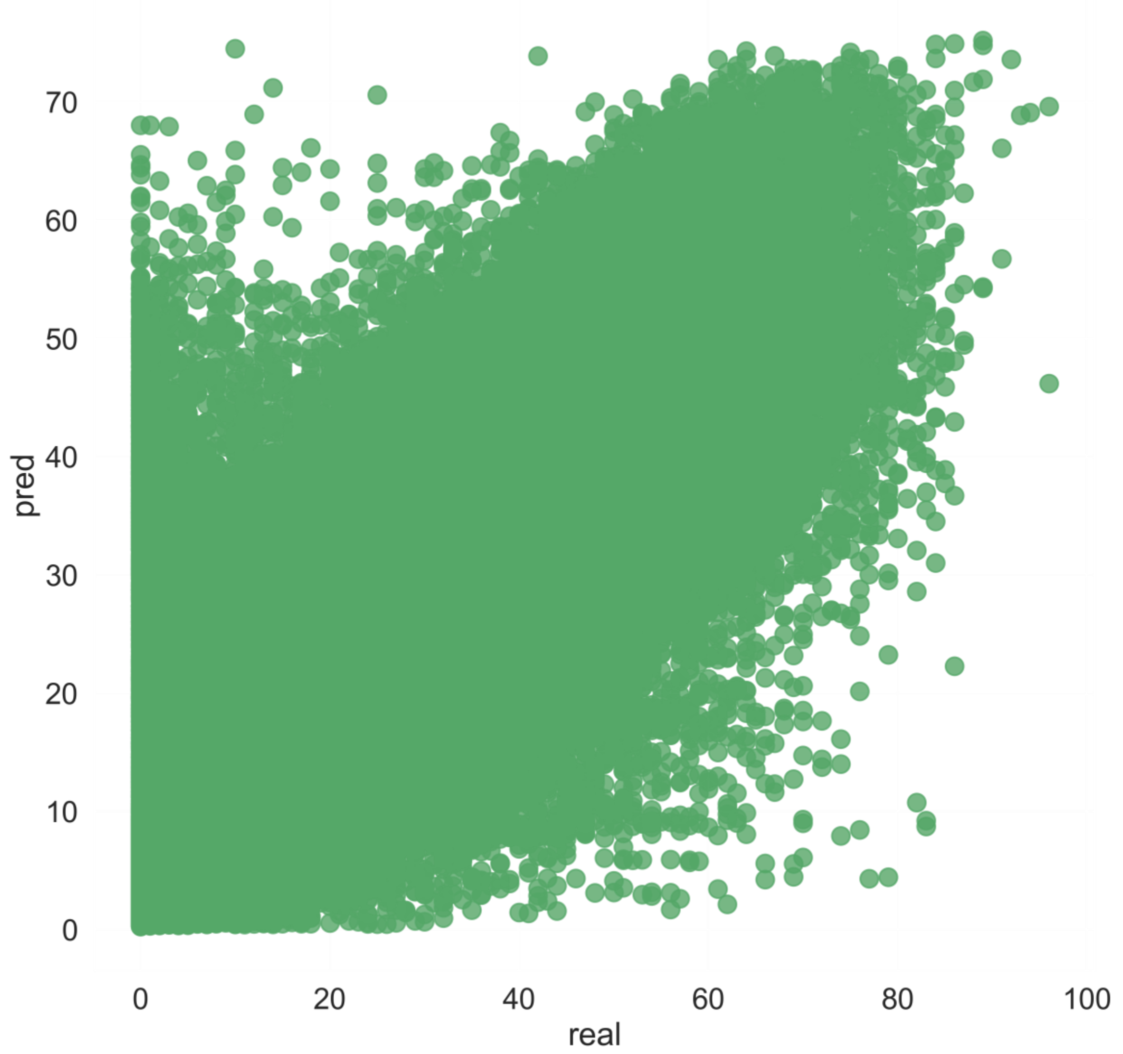








VN: Popularity prediction – MultiLayer Perceptron



1. OneHotEncoder for categorical variables

2. Artist string column: replaced by mean popularity of artist

3. MinMaxScaler

4. HPO:
- GridSearchCV for optimization of layer structure only
- RandomizedSearch for other optimization of other HP ('max_iter', 'batch_size')

5. No feature selection

VN: Popularity prediction – MultiLayer Perceptron - HPO

```
# initially I used GridSearch for 'hidden_layer_sizes' optimization


# solver='adam'-  slower
# solver='lbfgs' -faster, but doesn't converge
from scipy.stats import randint, poisson

import time

start = time.time()



model = MLPRegressor(activation='logistic', solver='adam' , learning_rate='adaptive', learning_rate_init=0.015, random_state=42,
param_grid = {'hidden_layer_sizes': [(5,5,5),(5,5),(10,5,1),(15,15,15),(30,30,30),(15,10),(50,50,50,50,50),(25,25,25,25)],
              'batch_size':[30]
}

model, pred = search_pipeline(X_train_1, X_test_1, y_train, y_test, model,
                              param_grid, cv=3, scoring_fit='r2',
                              search_mode = 'GridSearchCV', n_iterations = 0)

print(model.best_score_)
print(model.best_params_)

end = time.time()
print(end - start)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  22 out of  24 | elapsed:  5.5min remaining:   30.1s
[Parallel(n_jobs=-1)]: Done  24 out of  24 | elapsed:  6.1min finished
0.5354726164058927
{'batch_size': 30, 'hidden_layer_sizes': (5, 5)}
428.51941990852356
```

In [142]:

```
# then I applied RandomizedSearch to optimize 'learning_rate_init', 'batch_size' and 'max_iter'
# (when sizes of hidden layers were taken from GridSearch)


# solver='adam'-  slower
# solver='lbfgs' -faster, but doesn't converge
from scipy.stats import randint, poisson

import time

start = time.time()



model = MLPRegressor(activation='logistic',solver='adam', hidden_layer_sizes=(5,5),
                     learning_rate='adaptive', random_state=42,
                   # batch_size=30,
                     learning_rate_init=0.015
                    )

param_grid = {#'learning_rate_init': randint(0.01,0.05),
              'batch_size':randint(5, 40),
              'max_iter': randint(200,500)
}

model, pred = search_pipeline(X_train_1, X_test_1, y_train, y_test, model,
                              param_grid, cv=3, scoring_fit='r2',
                              search_mode = 'RandomizedSearchCV', n_iterations = 20)

print(model.best_score_)
print(model.best_params_)

end = time.time()
print(end - start)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  7.4min
[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed: 17.4min finished
0.5383573370386555
{'batch_size': 18, 'max_iter': 249}
1093.506673336029
```

GridSearchCV

RandomizedSearchCV

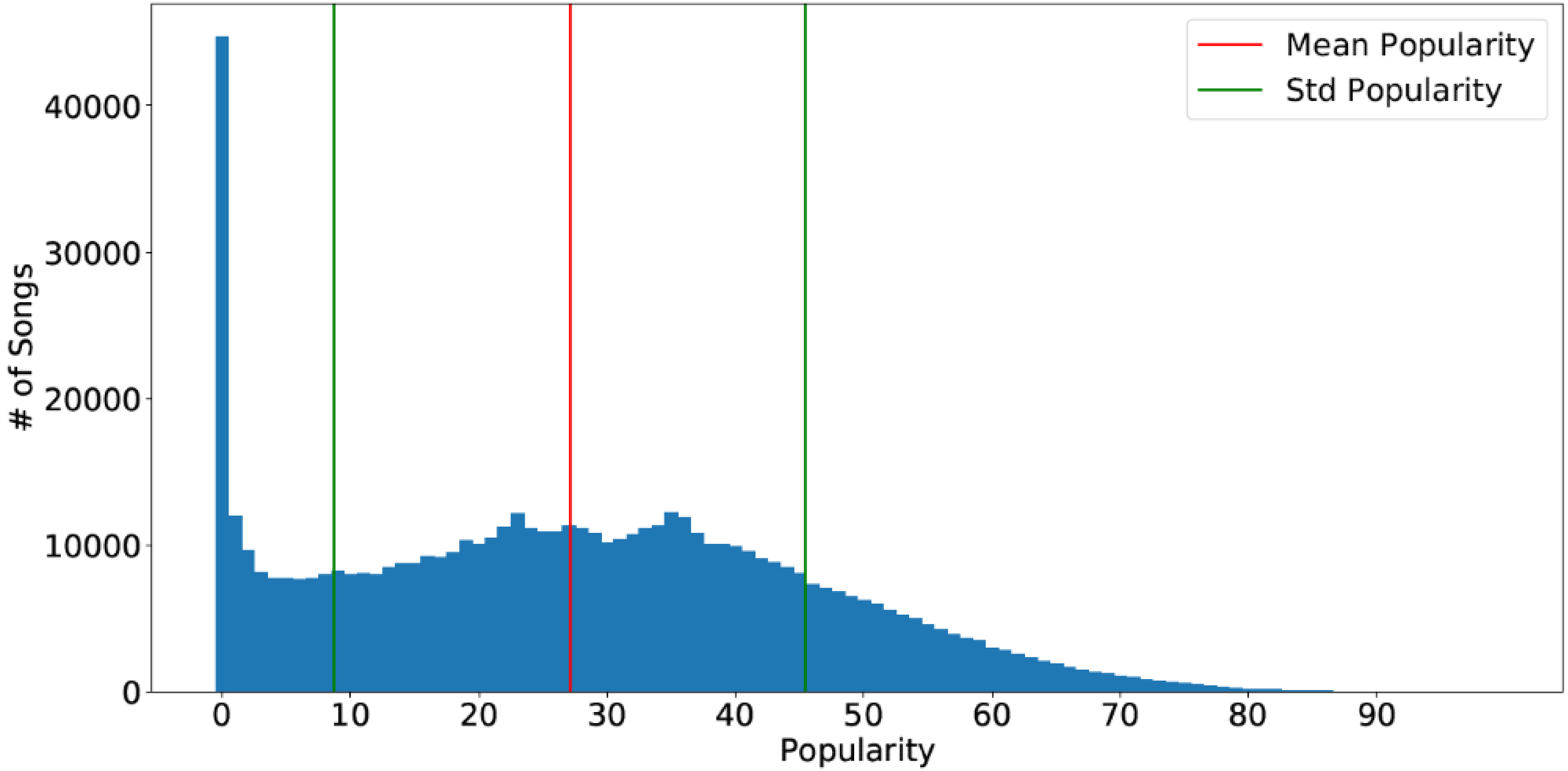
cv=3

VN: Popularity prediction – MLP (and LGBM) - results

Algorytm	R2_score	MAE
MLP, no transformation for target variable	**68.8 (Train)** **64.6 (Test)**	**7.4 (Train)** **7.8 (Test)**
MLP, target variable transformed	**73.01 (Train)** **70.68 (Test)**	**0.52* (Train)** **0.55* (Test)**
LGBM, no transformation for target variable	**74.4 (Train)** **67.6 (Test)**	**6.65 (Train)** **7.50 (Test)**
LGBM, target variable transformed	**80.07 (Train)** **74.43 (Test)**	**0.46* (Train)** **0.51* (Test)**

Popularity prediction – Approach 3

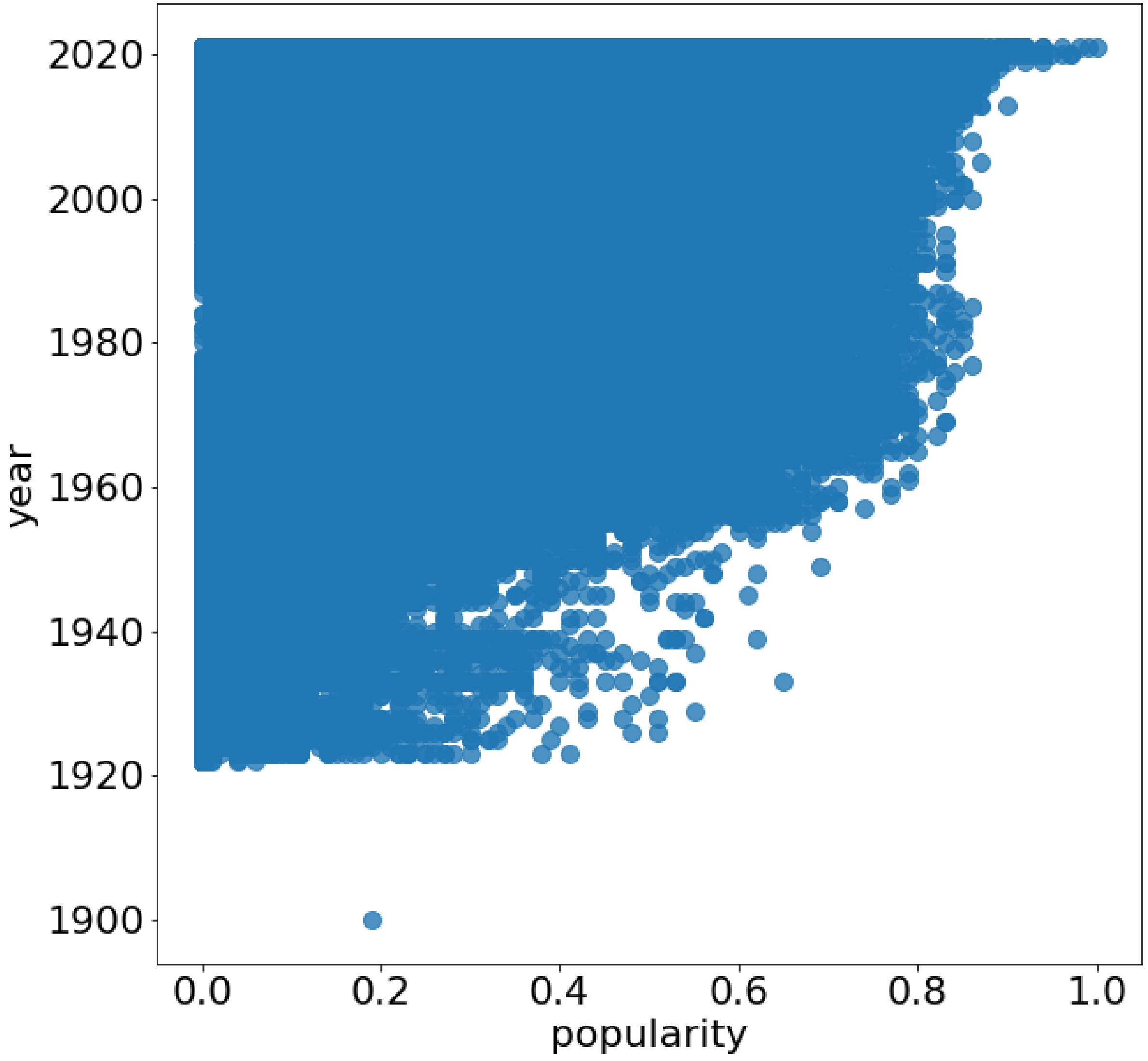
Popularity distribution songs:



Popularity prediction – Approach 3

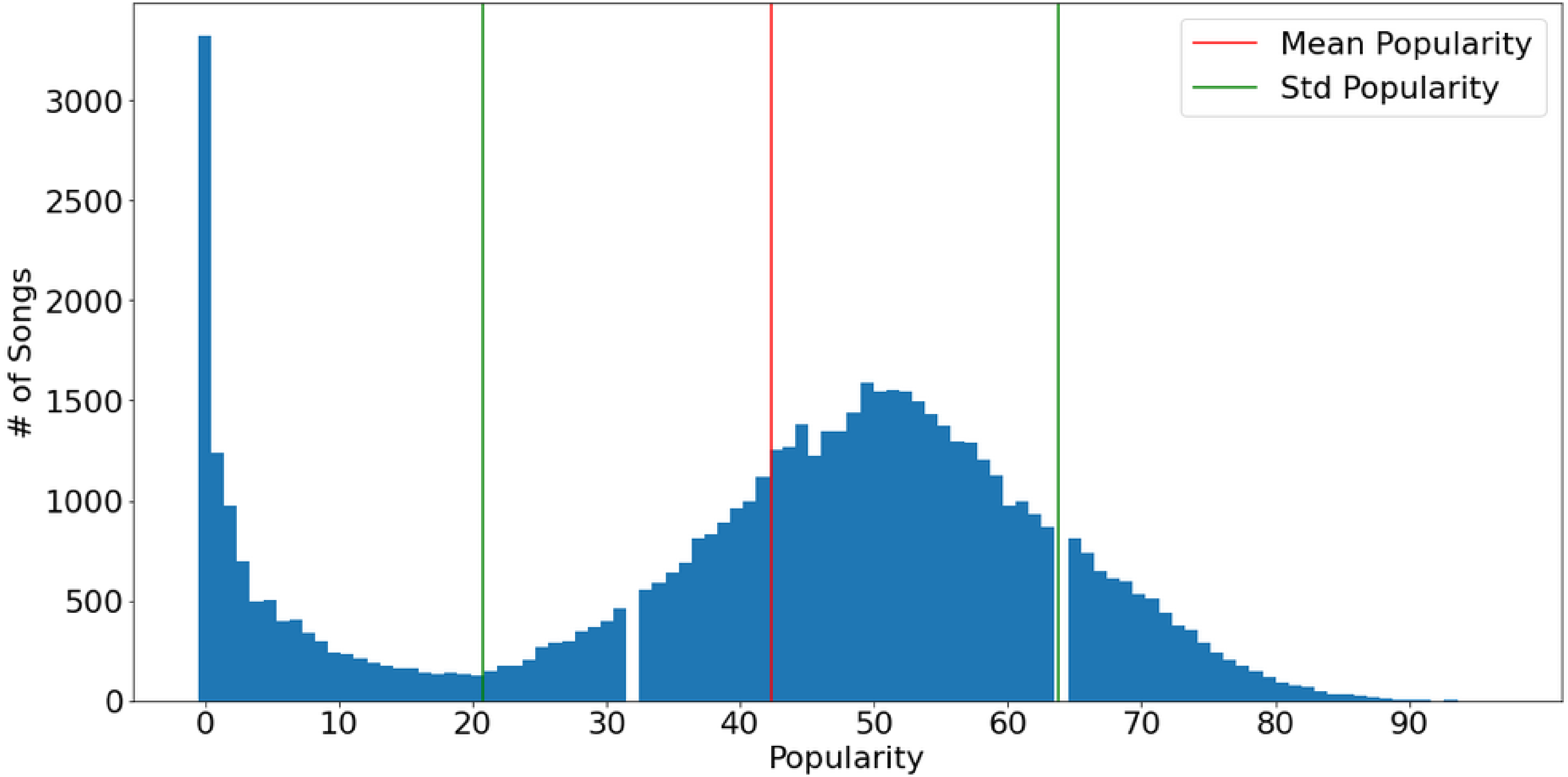
Popularity distribution per year:

- Old songs: lower popularity (average and range)
- Newer song: increasing range of popularity
- Single point at 1900 -> ignored (outlier)



Popularity prediction – Approach 3

Popularity distribution songs 2016-2021:

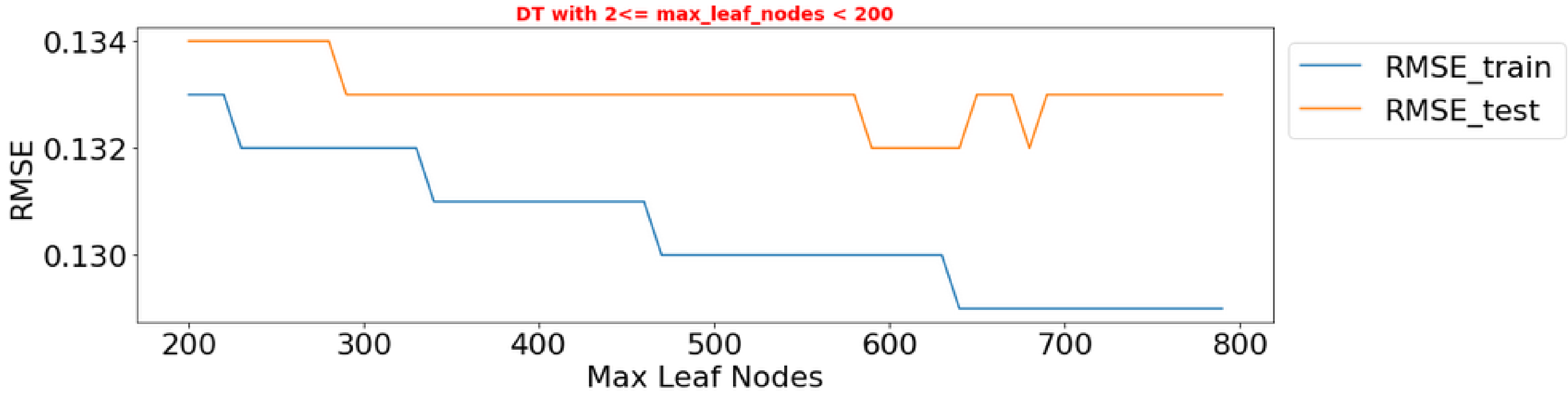


Popularity prediction – Approach 3

Popularity distribution:

- Throughout all years big portion is unpopular
- Spread of popularity wider and towards higher mean popularity value

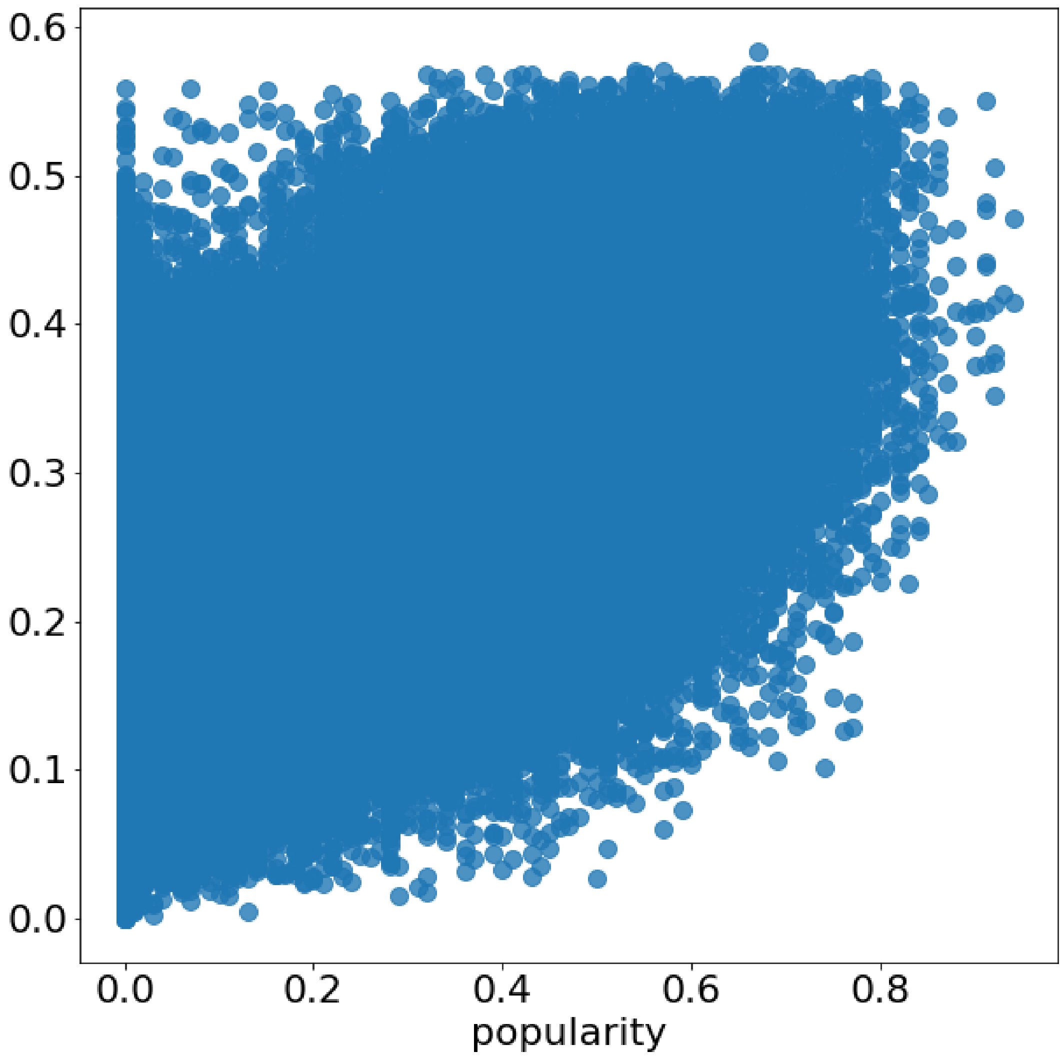
Initial model with DecisionTreeClassifier, optimum is 13.3 % RMSE

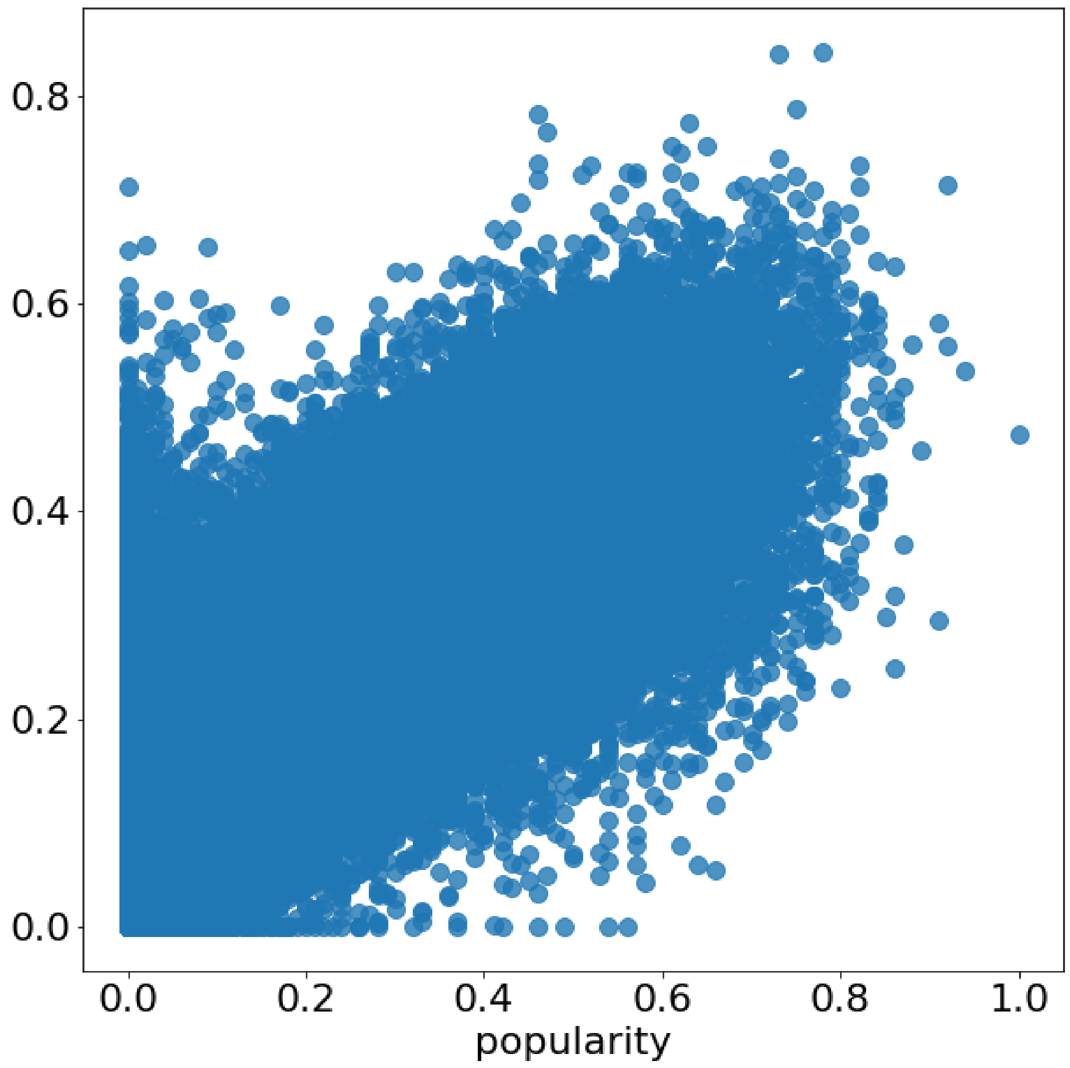


Popularity prediction – Approach 3

- Regression using XGBoost (reg:squarederror)
- Feature selection -> Best model with all 16 **original features** included:
  'duration_ms', 'explicit', 'danceability', 'energy', 'key', 'popularity', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'time_signature','artist_category'
- Hyperparameter optimization: RandomizedSearch with inbuilt CV:
  - Quantile transform: uniform – <u>normal</u>
  - XGB parameters: param_dist = {'n_estimators': stats.randint(2, 200),
    'learning_rate': stats.uniform(0.01, 0.6),
    'subsample': stats.uniform(0.3, 1.9),
    'max_depth': [3, 4, 5, 6, 7, 9, 12, 15],
    'min_child_weight': [1, 2, 3, 4, 5, 6],
    'gamma': stats.uniform(0, 1)

Popularity prediction – Approach 3 – Comparing initial and improved model



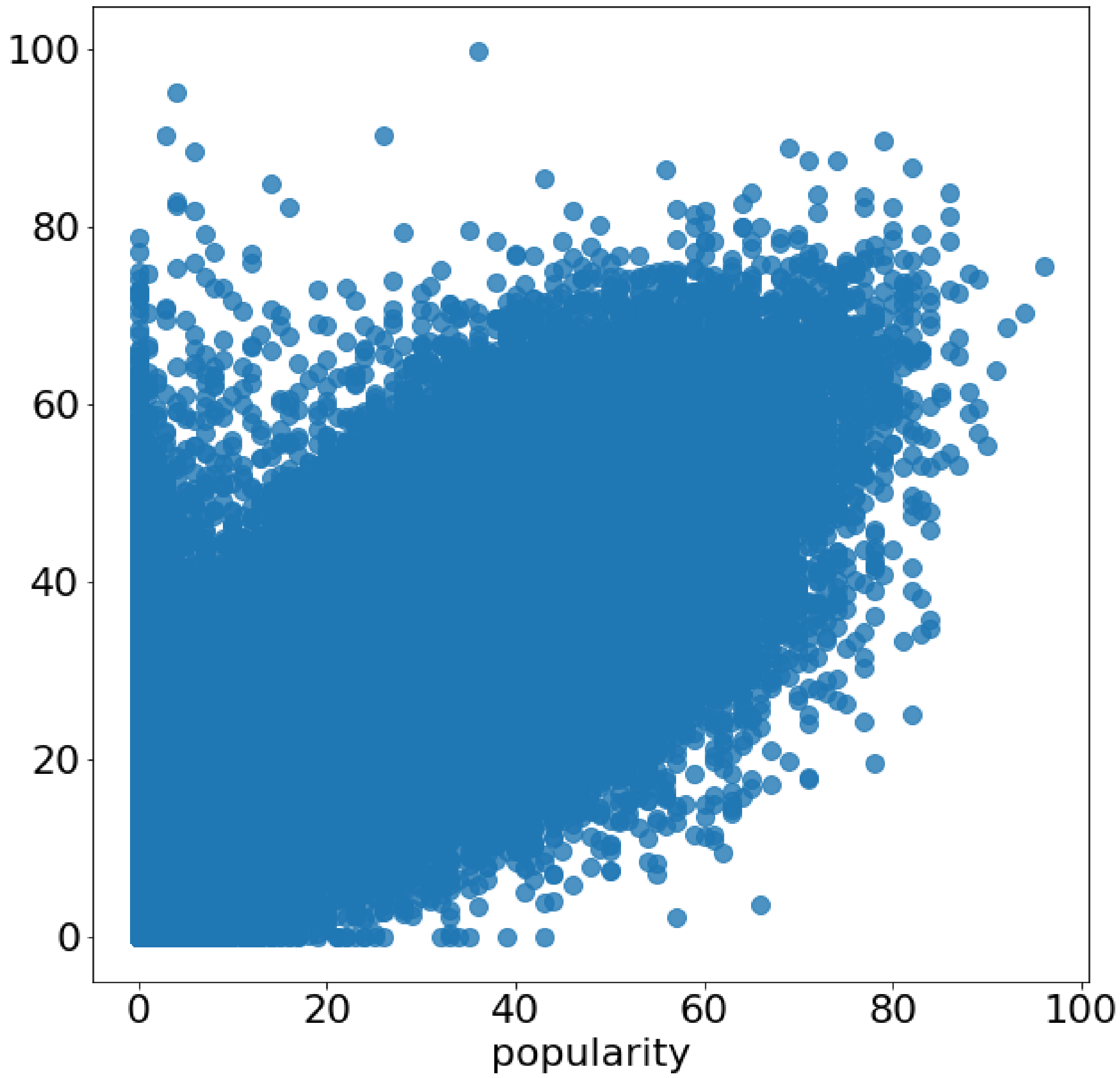


Popularity prediction – Approach 3 – Improved model

Incorporate artist (ID):

- R2 improves to 54.6
- RSME 12.3 %

Artist has a big impact, but replacing it by it's ID as category not best solution.



Popularity prediction – Approach 3 – Improved model

Comparison with rescaling popularity data using random oversampler:

- No improvement
- Best result lead only to RSME of 13.4 %
- R2 score: 46.0 %

This is roughly 10 percent worse than before

Popularity prediction – Approach 3 – Improved model

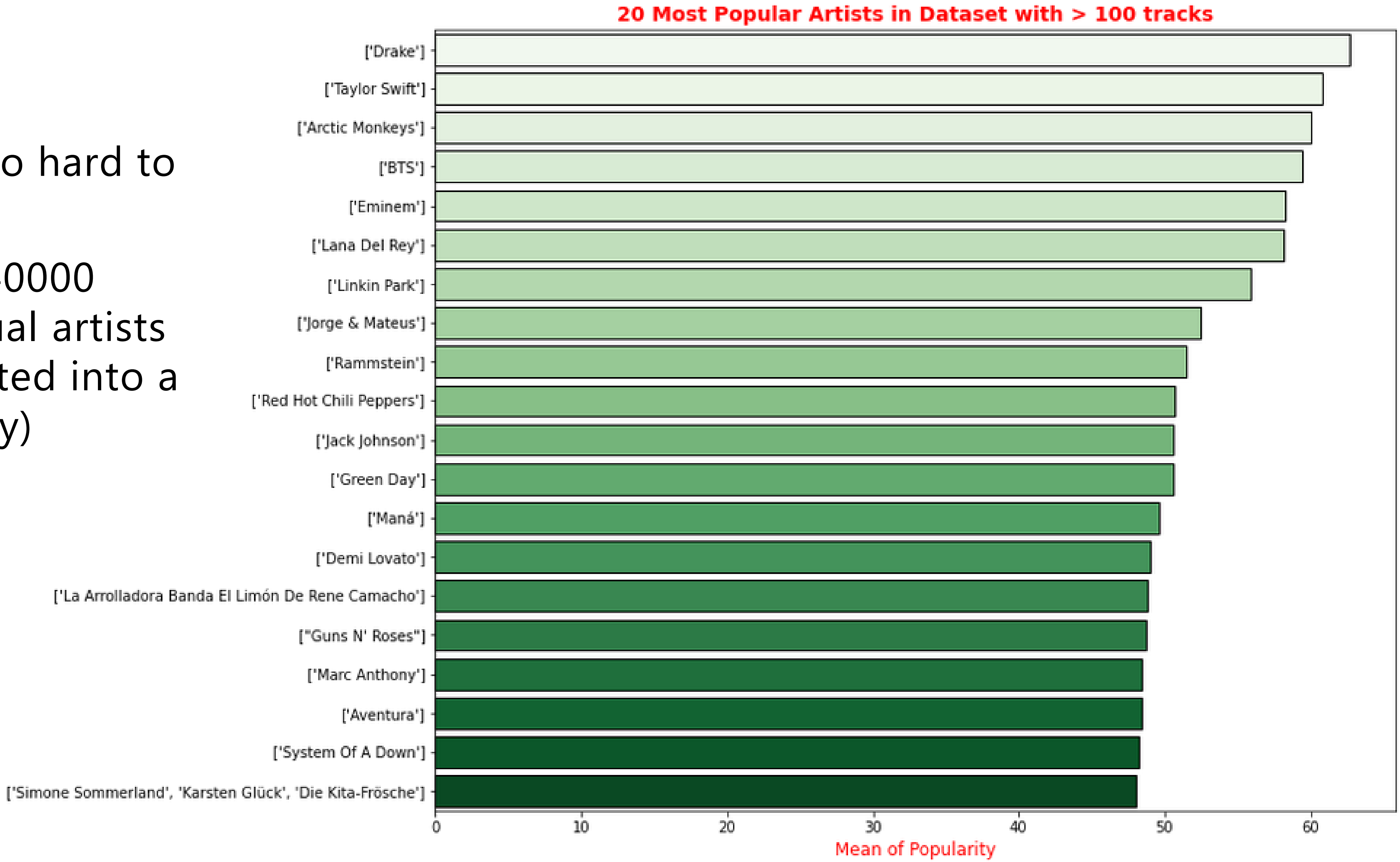
- Best settings:

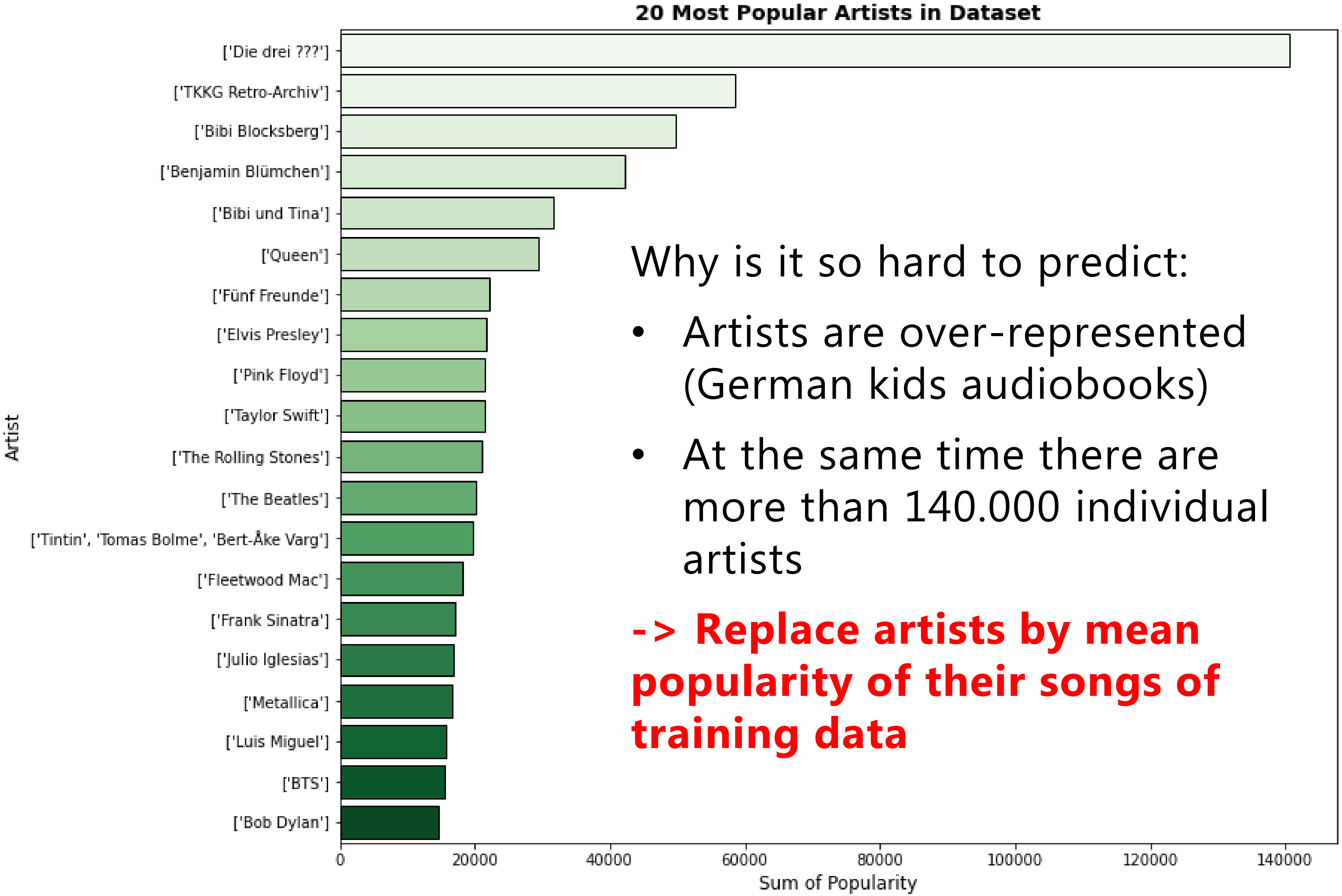
```
In [158]: clf.best_estimator_
          clf.best_params_

Out[158]: {'gamma': 0.1128482654196935,
           'learning_rate': 0.30062892449261447,
           'max_depth': 12,
           'min_child_weight': 2,
           'n_estimators': 192,
           'subsample': 0.403178262355943}
```

- RSME at optimum: 12.28 %, R2 score: 54.9 %
- Without rescaling samples and using a normal distribution for the quantile transform

Why is it so hard to predict:

- Over 140000 individual artists (converted into a category)





Why is it so hard to predict:

- Artists are over-represented (German kids audiobooks)
- At the same time there are more than 140.000 individual artists

-> Replace artists by mean popularity of their songs of training data

Popularity prediction– Approach 3 – Introduce transformations

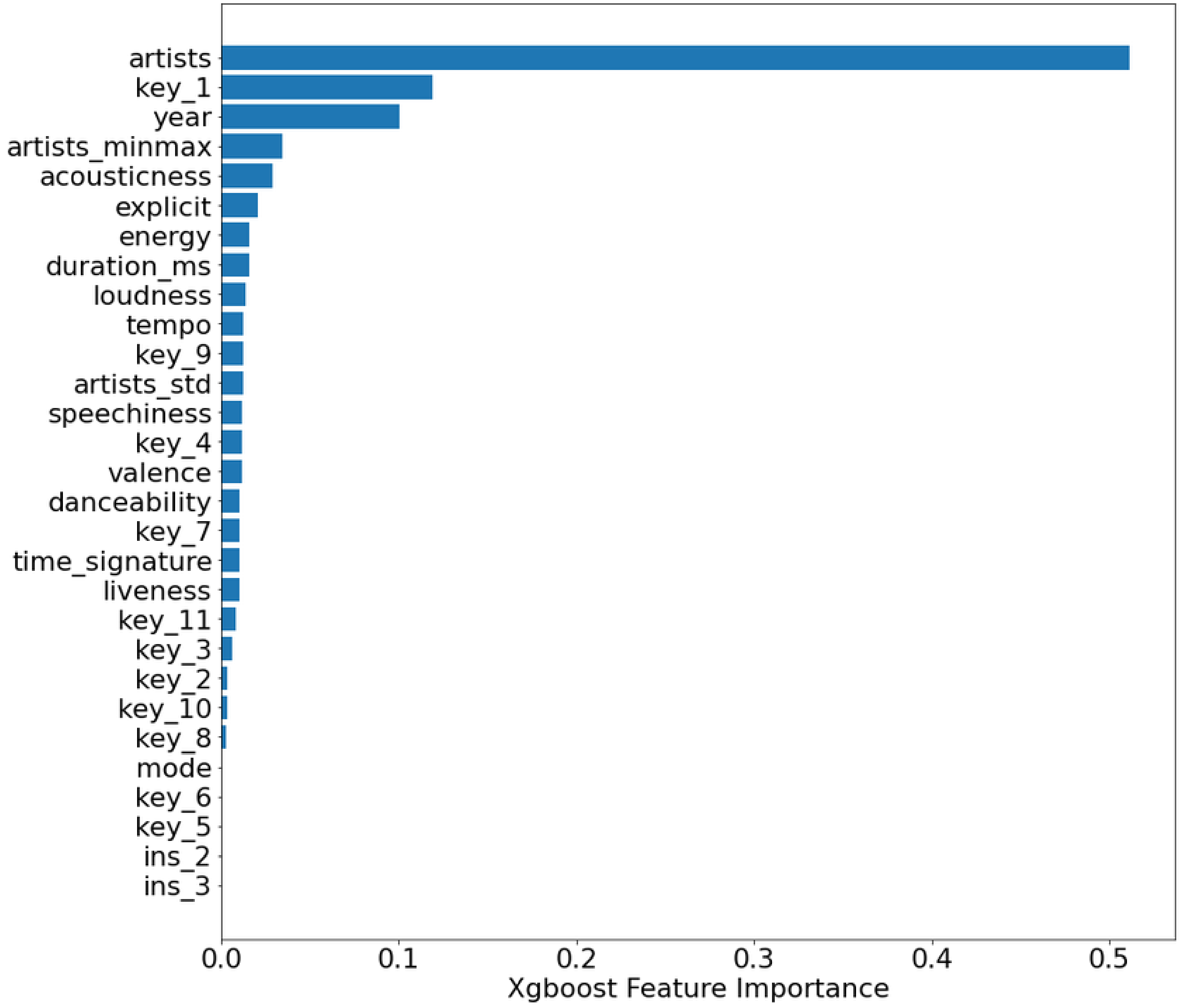
- Introduce 3 new features from popularity of training dataset:
  - Artist's mean popularity of all his/her/their tracks
  - Artist's mean popularity standard deviation
  - Artist's range of popularity among tracks (max-min)
  - If only exists once in train: Replace with value of total value of train set
  - Fit to train, transform train and test set

```
def fit (self, X, y):
    self.artists_df =  y.groupby(X.artists).agg(['mean', 'count'])
    self.artists_df.loc['unknown'] = [y.mean(), 1]
    self.artists_df.loc[self.artists_df['count'] <= self.MinCnt, 'mean'] = y.mean()
    self.artists_df.loc[self.artists_df['count'] >= self.MaxCnt, 'mean'] = 0
    self.artists_std_df =  y.groupby(X.artists_std).agg(['std', 'count'])
    self.artists_std_df.loc['unknown'] = [y.std(), 1]
    self.artists_std_df.loc[self.artists_std_df['count'] <= self.MinCnt, 'std'] = y.std()
    self.artists_std_df.loc[self.artists_std_df['count'] >= self.MaxCnt, 'std'] = 0
    self.artists_minmax_df =  y.groupby(X.artists_minmax).agg(['max', 'count'])
    self.artists_minmax_df.loc['unknown'] = [y.max()-y.min(), 1]
    self.artists_minmax_df.loc[self.artists_minmax_df['count'] <= self.MinCnt, 'max'] = y.max()-y.min()
    self.artists_minmax_df.loc[self.artists_minmax_df['count'] >= self.MaxCnt, 'max'] = 0
    return self
```

Popularity prediction– Approach 3 – Introduce transformations

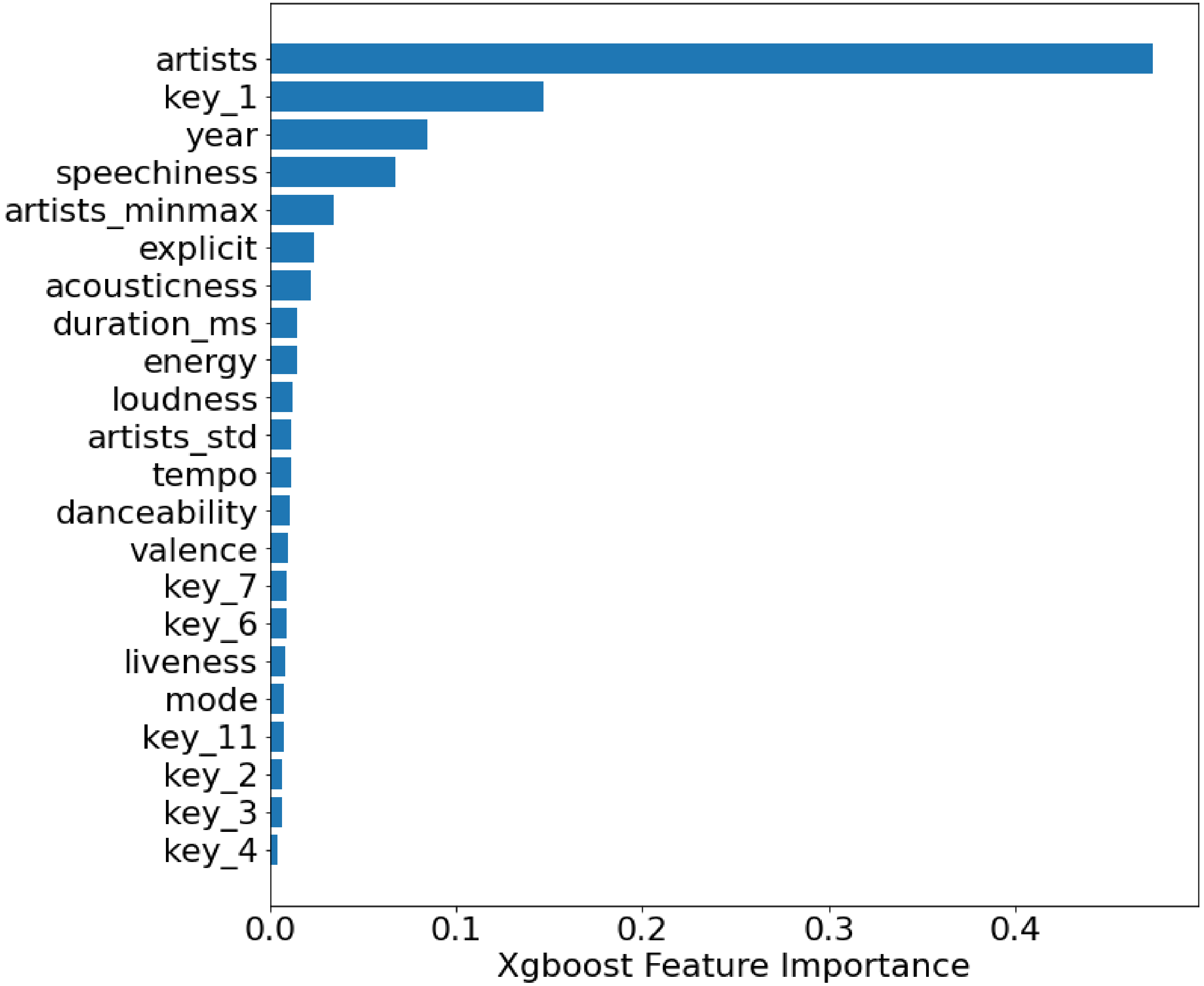
- Adding OneHotEncoder for:
  - Instrumentalness
  - Key of song
  - Time signature of song
- Feature Importance:
  - Designed features from popularity very important
  - Best performance initially when not removing any features
- Further parameter tweaking + enhanced transformations + feature reduction (actually improves performance now!)

Rerun optimization with reduced features



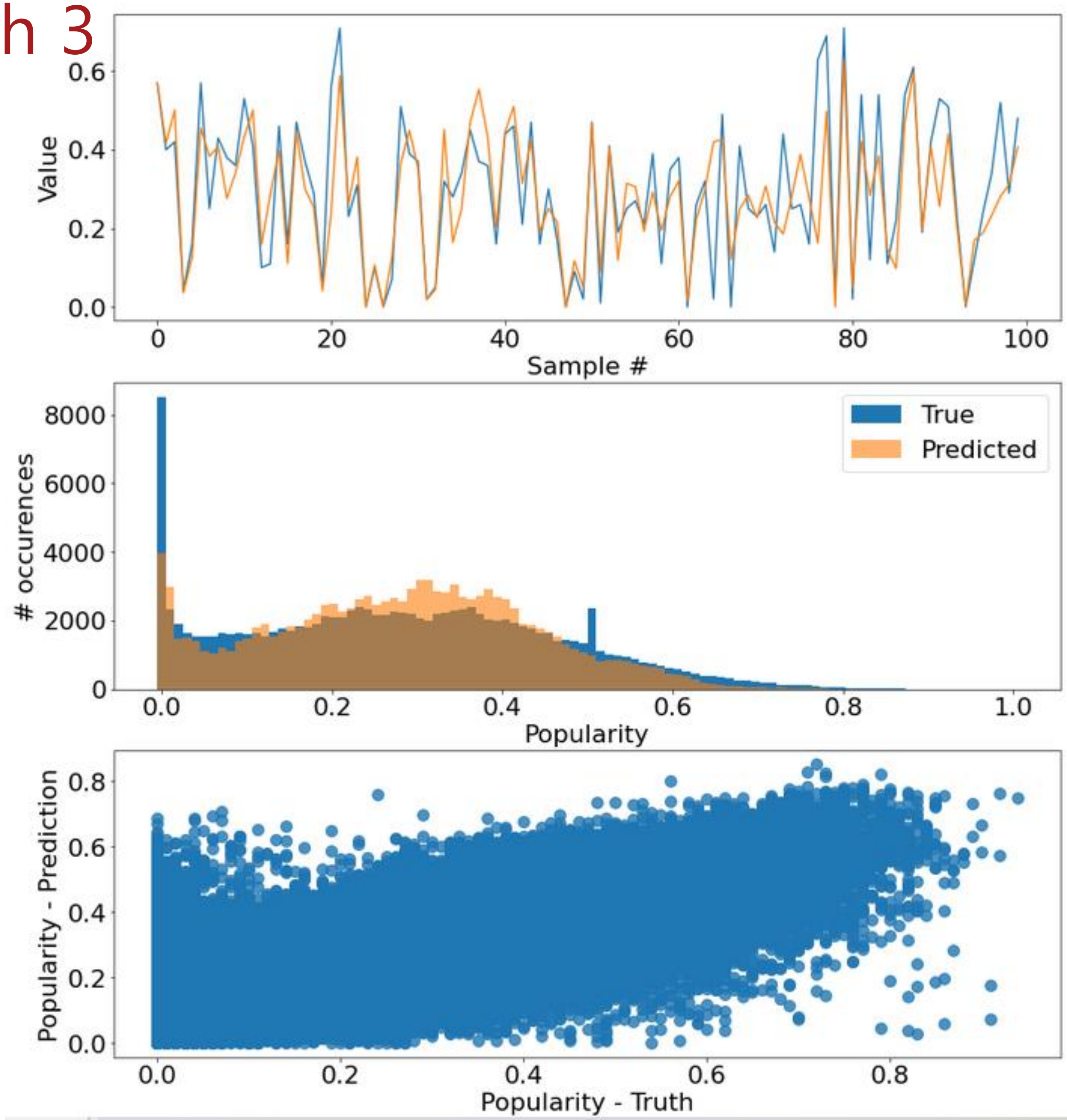
Popularity prediction – Approach 3 – Final model with XGB

- Further parameter tweaking
- Reduce number of features to 16
- Optimized usage of transformations & OneHotEncoder
- **Best result:**
  - Train: 73.8 % r2 score, MAE: 0.067, RSME = 0.094
  - Test: 67.9 % r2 score, MAE: 0.073, RSME = 0.104



Popularity prediction – Approach 3 – Final model with XGB

- Further parameter tweaking
- Reduce number of features to 16
- Optimized usage of transformations & OneHotEncoder
- **Best result:**
  - Train: 73.8 % r2 score, MAE: 0.067, RSME = 0.094
  - Test: 67.9 % r2 score, MAE: 0.073, RSME = 0.104



Popularity prediction – Approach 3 – Final model with XGB

- **Best result:**
  - Train: 73.8 % r2 score, MAE: 0.067, RSME = 0.094
  - Test: 67.9 % r2 score, MAE: 0.073, RSME = 0.104

```
xgb_finopt = XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, eta=0.276612531511493,
             eval_metric='mae', gamma=0.5165040579559896, gpu_id=-1,
             importance_type='gain', interaction_constraints='', max_delta_step=0, max_depth=18,
             min_child_weight=1, monotone_constraints='()',
             n_estimators=500, n_jobs=8, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
             subsample=0.80698455864976704, tree_method='exact',
             validate_parameters=1, verbosity=2)
```

Popularity prediction – Approach 3 – Final model XGB: Cross-check resampling again

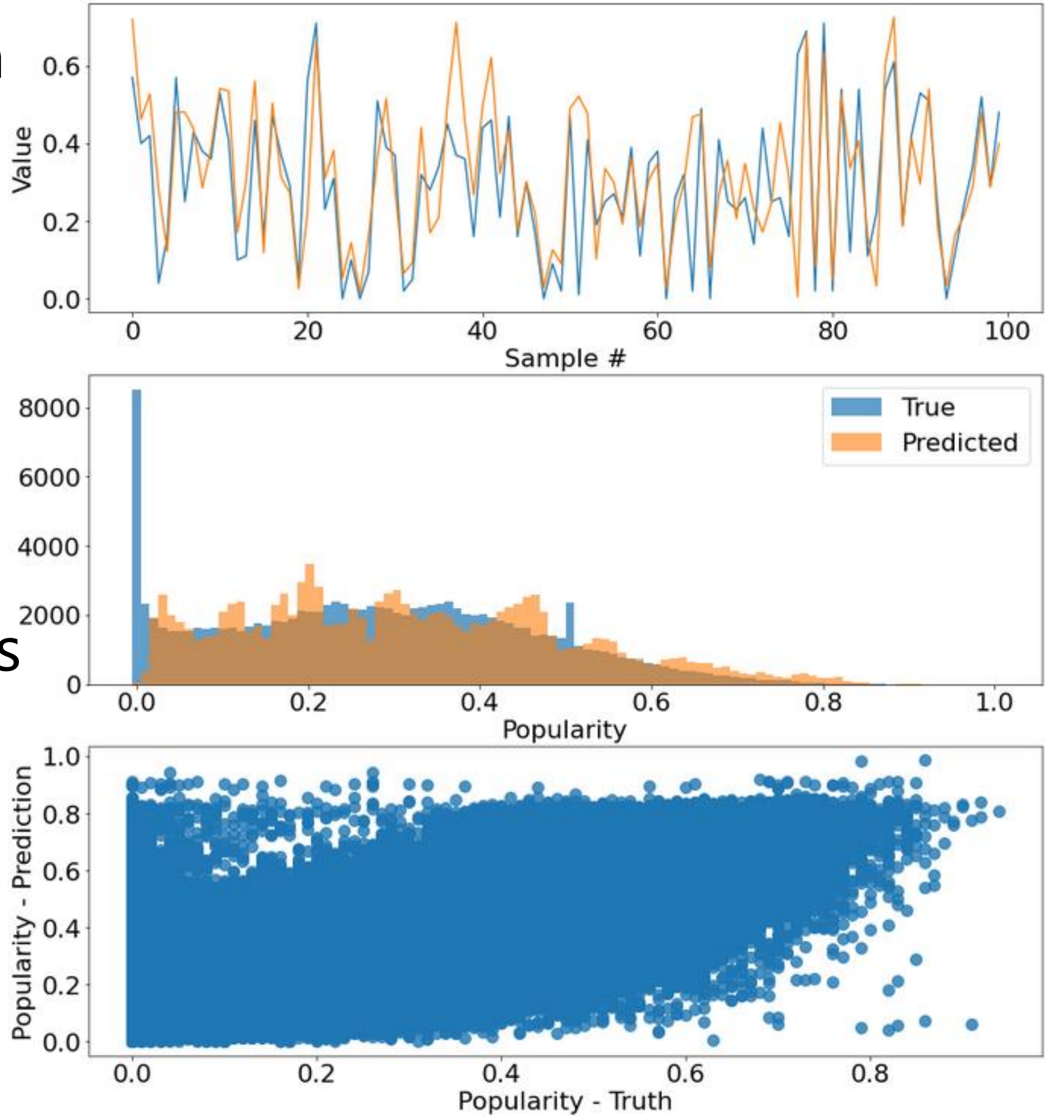
Cross-check if Resampling for popularity makes with all transformations and OHE difference:

- For release year: resampling improves performance
- Popularity: unbalanced, but also expected, therefore unclear if improvement from balancing expected
- Doing randomsampling with 2000 samples per popularity value
- Rerunning Hyperparameter optimization

Popularity prediction – Approach 3 – Final model XGB: Cross-check resampling again

Cross-check: Resampling for popularity with all transformations and OHE:

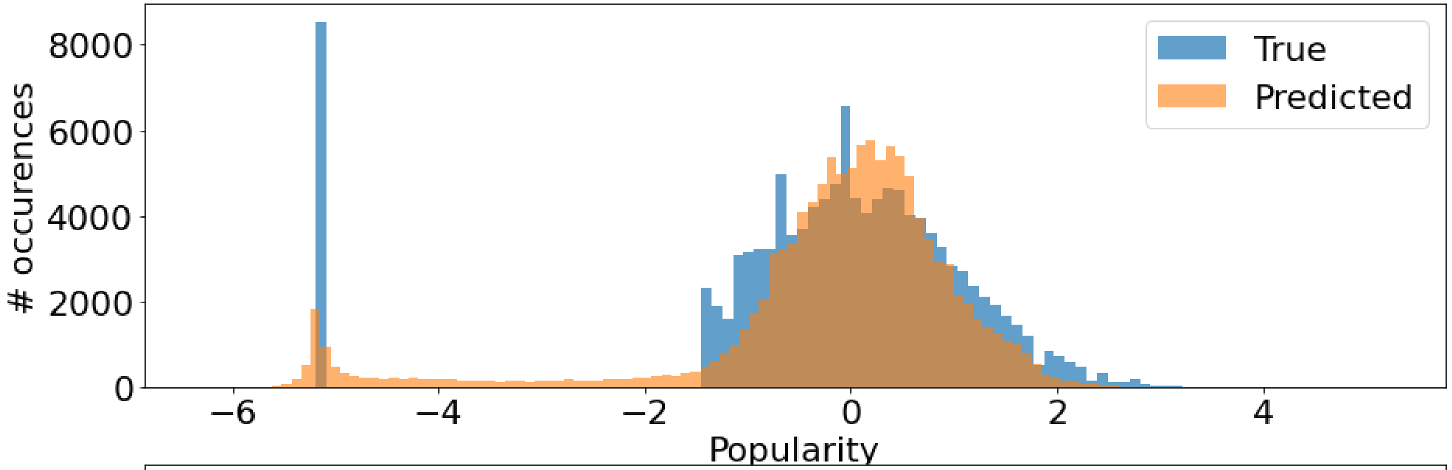
- Hyperparemeter optimized performance:
  - Train: R2 = 85 %, MAE = 0.080
  - Test: R2 = 47 %, MAE = 0.095
- From histogram: Model now cannot predict the majority of songs being in 0
- Performs worse, but R2 on train set improves
- Confirms that only release year needs to be rescaled, but not popularity

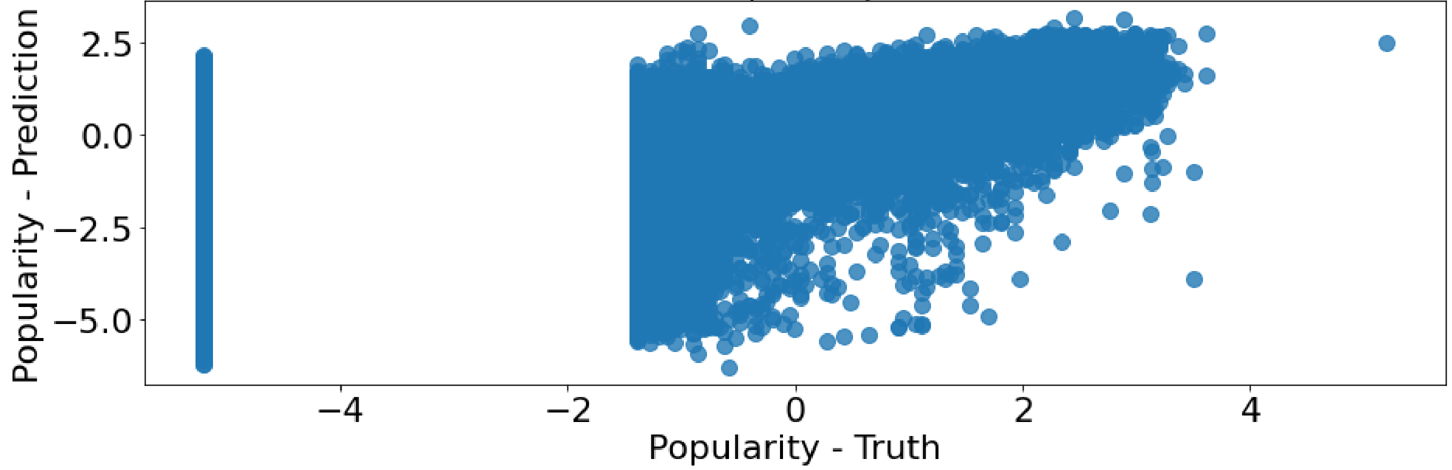


Popularity prediction – Approach 3 – Final model XGB: **LastMinuteUpdate**

Quantile normal transform for popularity

- Hyperparemeter optimized performance:
  - Train: R2 = 85.5 %, MAE = 0.40
  - Test: R2 = 73.4 %, MAE = 0.50
- From histogram: Model now cannot predict the majority of songs being in 0



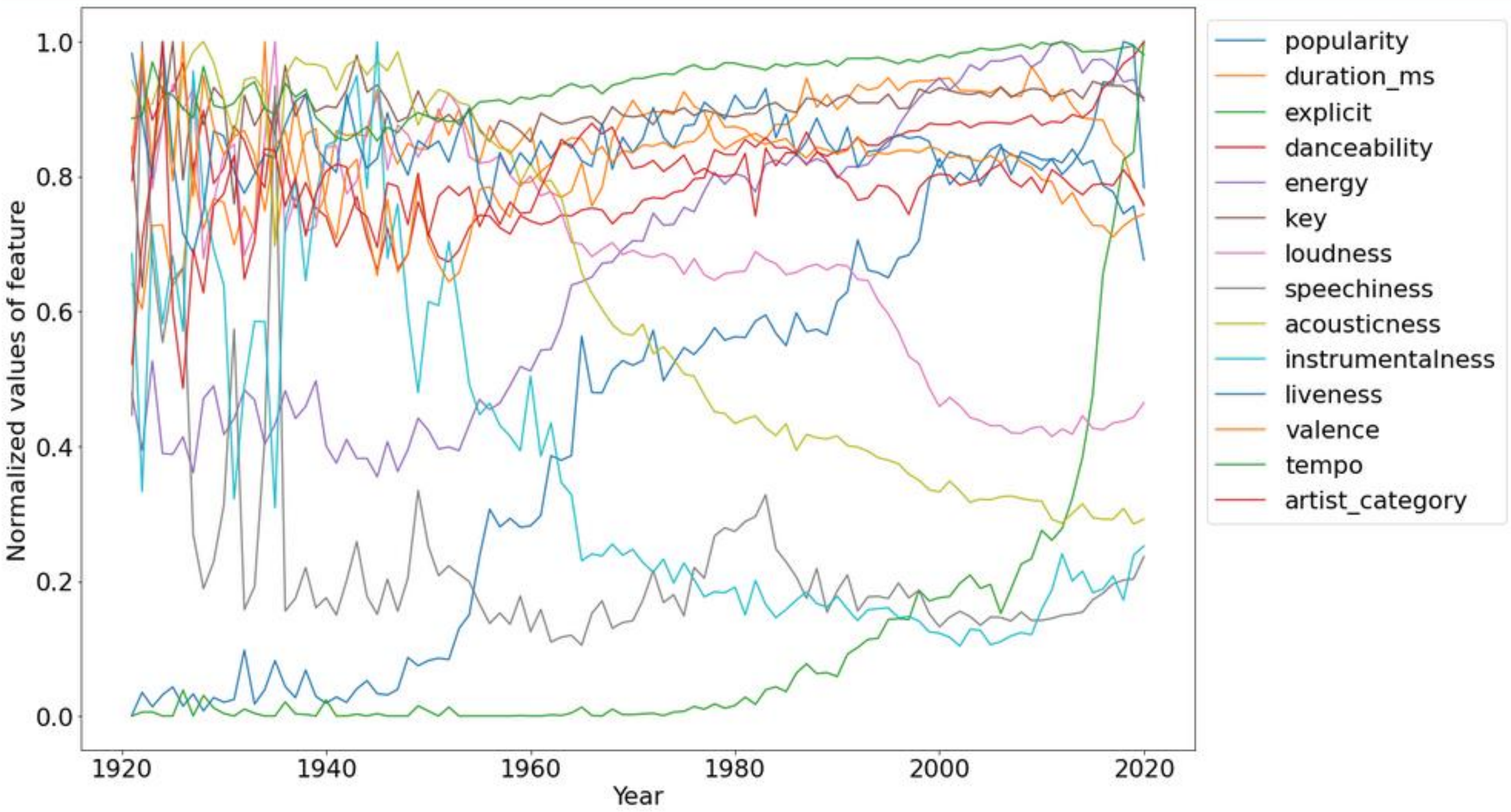


Popularity prediction – Approach 3 – Why is it so hard to predict

- Look into spotify api and look into how popularity is determined:
  - Via algorithm
  - Click based
  - Also time-dependent
  - Undisclosed information
- Number of clicks -> Should greatly improve our predictions, but we don't have that information in our dataset
- With this low number in features, despite the big data set it is not possible to determine popularity with high precision
- If we would have more musical features and maybe also how and when clicks, we could re-engineer popularity algorithm probably and predict better

Future popularity prediction – Approach 3

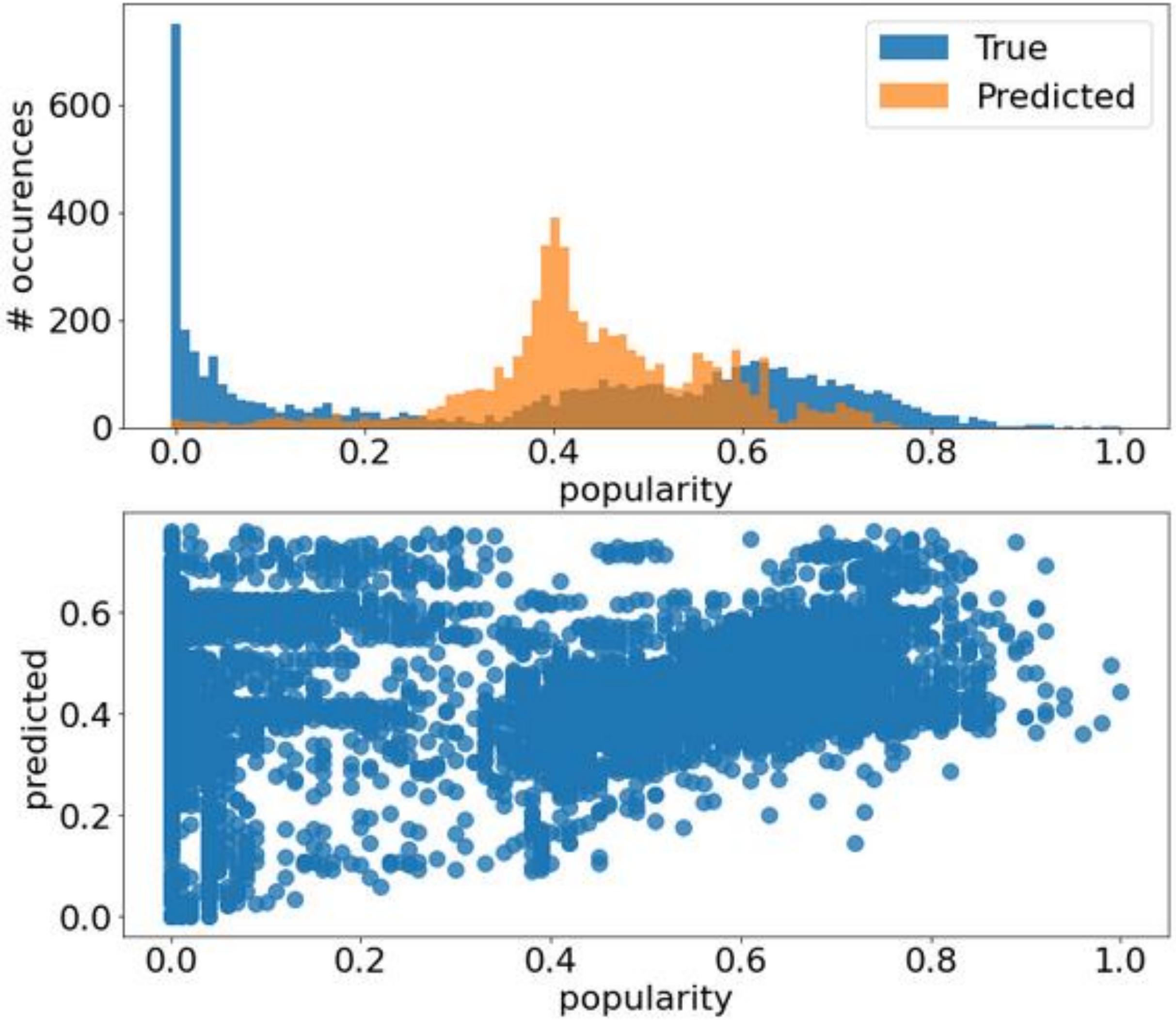
- Check how different parameters change over time (mean per year):



Future popularity prediction – Approach 3

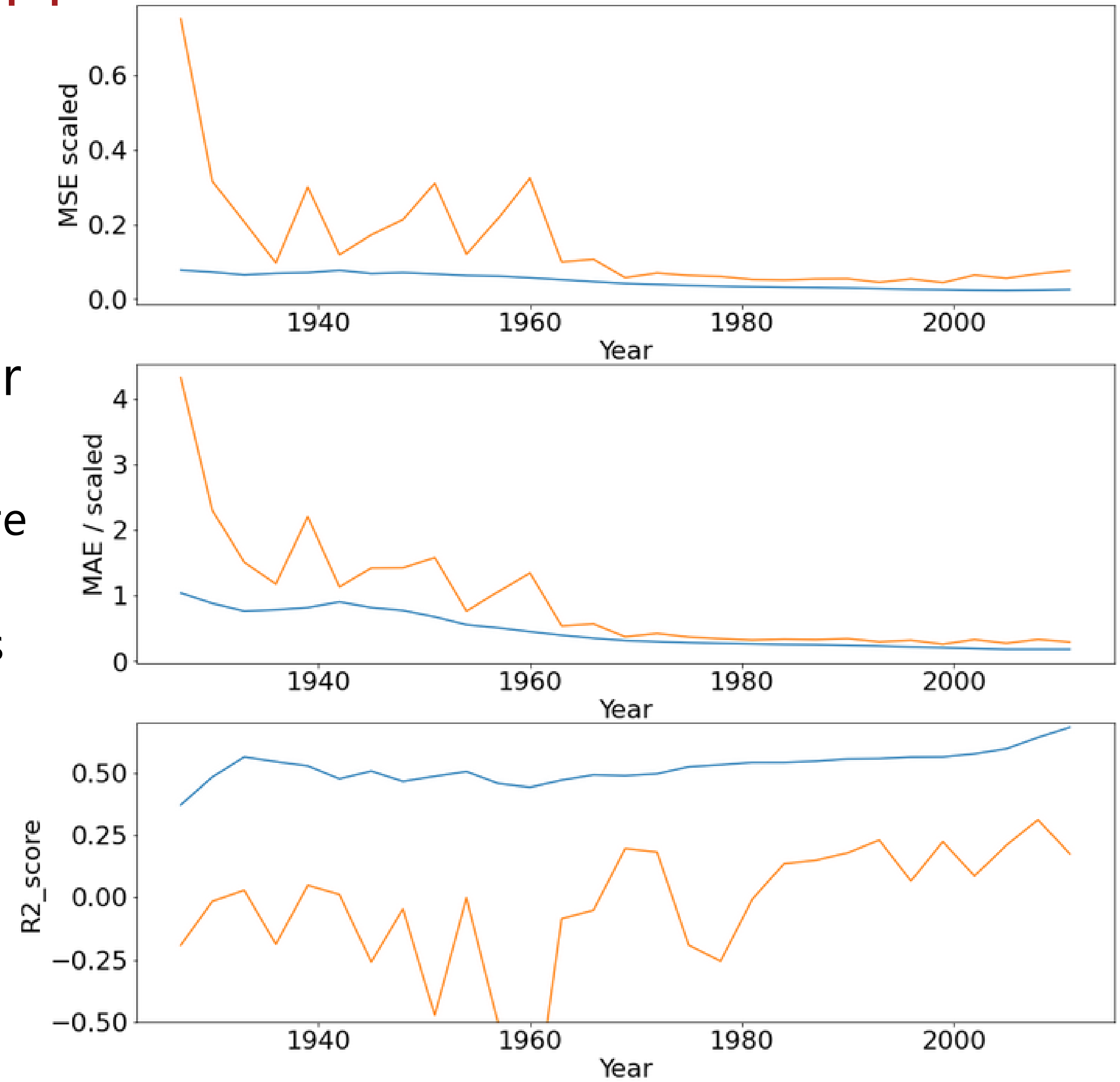
Goal: Predict 1 year into the future:

- XGBRegressor algorithm from general prediction
- Train on 2010-2020 data
- Predict: 2021 data
- Result:
  - MAE: 0.072 train <-> 0.067 general
  - MAE: 0.24 test <-> 0.073 general



Future popularity prediction – Approach 3

- Future prediction for different slices:
  - 10 years train, 1 year test
  - Steps of 3 years
- Performance improves towards newer data
  - Due to more variance in features for more recent music
  - More training data for more recent times



Popularity prediction – LightGBM model

Created two models, one that minimizes the *mean_squared_error* and one that minimizes the *mean_absolute_error*

```
# LightGBM prediction

# Define model

hyper_params = {
    'task': 'train',
    'boosting_type': 'gbdt',
    'objective': 'regression',
    'metric': ['mean_squared_error'],
    'learning_rate': 0.005,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.7,
    'bagging_freq': 10,
    'verbose': 0,
    "max_depth": 8,
    "num_leaves": 128,
    "max_bin": 512,
    "num_iterations": 10000,
    "n_estimators": 1000,
  }

gbm = lgb.LGBMRegressor(**hyper_params)
```

```
# Fit model to training data and time to see how long it takes

t1 = time.time()

model_mse = gbm.fit(X_train, y_train,
        eval_set=[(X_test, y_test)],
        eval_metric='mean_squared_error',
        early_stopping_rounds=10)

t2 = time.time()

print("Running time of LightGBM regression fit is (in seconds):", t2-t1)
```

```
# Same model, only change loss function from MSE to MAE

hyper_params2 = {
    'task': 'train',
    'boosting_type': 'gbdt',
    'objective': 'regression',
    'metric': ['mean_absolute_error'],
    'learning_rate': 0.005,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.7,
    'bagging_freq': 10,
    'verbose': 0,
    "max_depth": 8,
    "num_leaves": 128,
    "max_bin": 512,
    "num_iterations": 10000,
    "n_estimators": 1000,
  }

gbm2 = lgb.LGBMRegressor(**hyper_params2)
```

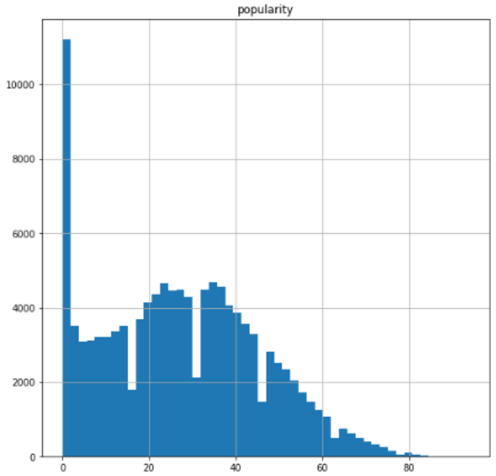
```
# Fit model to training data and time to see how long it takes

t1 = time.time()

model_mae = gbm2.fit(X_train, y_train,
        eval_set=[(X_test, y_test)],
        eval_metric='mean_absolute_error',
        early_stopping_rounds=10)

t2 = time.time()

print("Running time of LightGBM regression fit is (in seconds):", t2-t1)
```

Artists feature was not used for training in the beginning

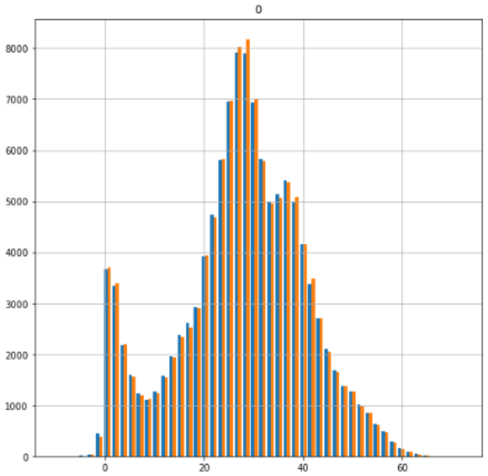
```
training_variables = ['duration_ms', 'explicit','danceability',  'energy', 'key',  'loudness', 'mode',
                      'speechiness', 'acousticness', 'instrumentalness',  'liveness', 'valence',
                      'tempo', 'time_signature', 'year']
```

Popularity prediction – LightGBM model

1. RobustScaler() - scale only training variables



True popularity



Predicted popularity distributions by the two models

Popularity here ranges from 0 to (almost) 100

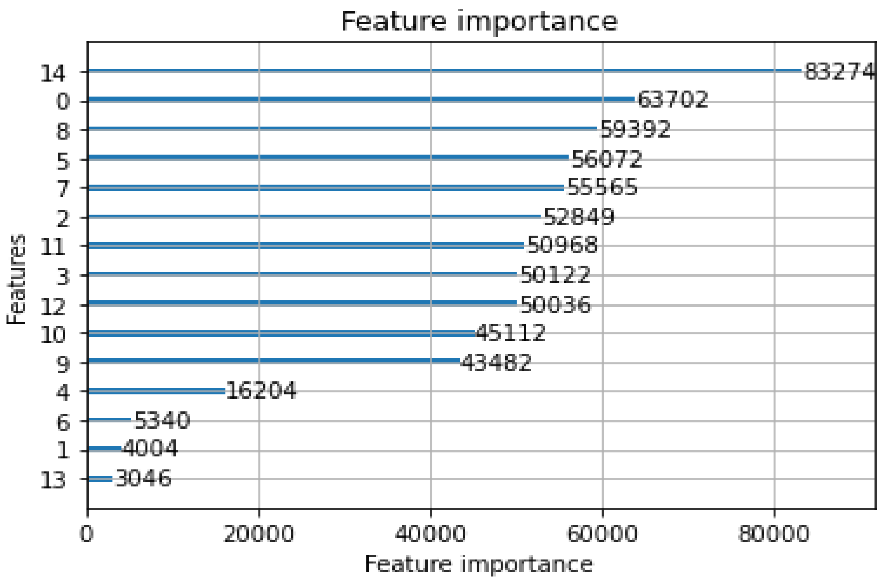
	MSE loss model	MAE loss model
MSE	163.32	164.53
RMSE	12.78	12.82
MAE	9.607	9.65
R2	0.51191739	0.508

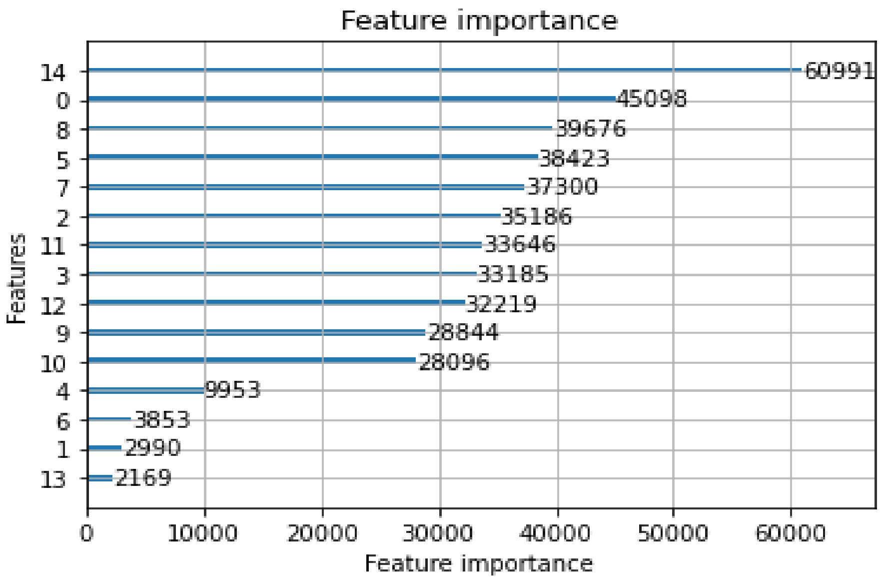
Both models have essentially the same performance as can be seen from the table of metrics and the distributions

Popularity prediction – LightGBM model Feature Importance

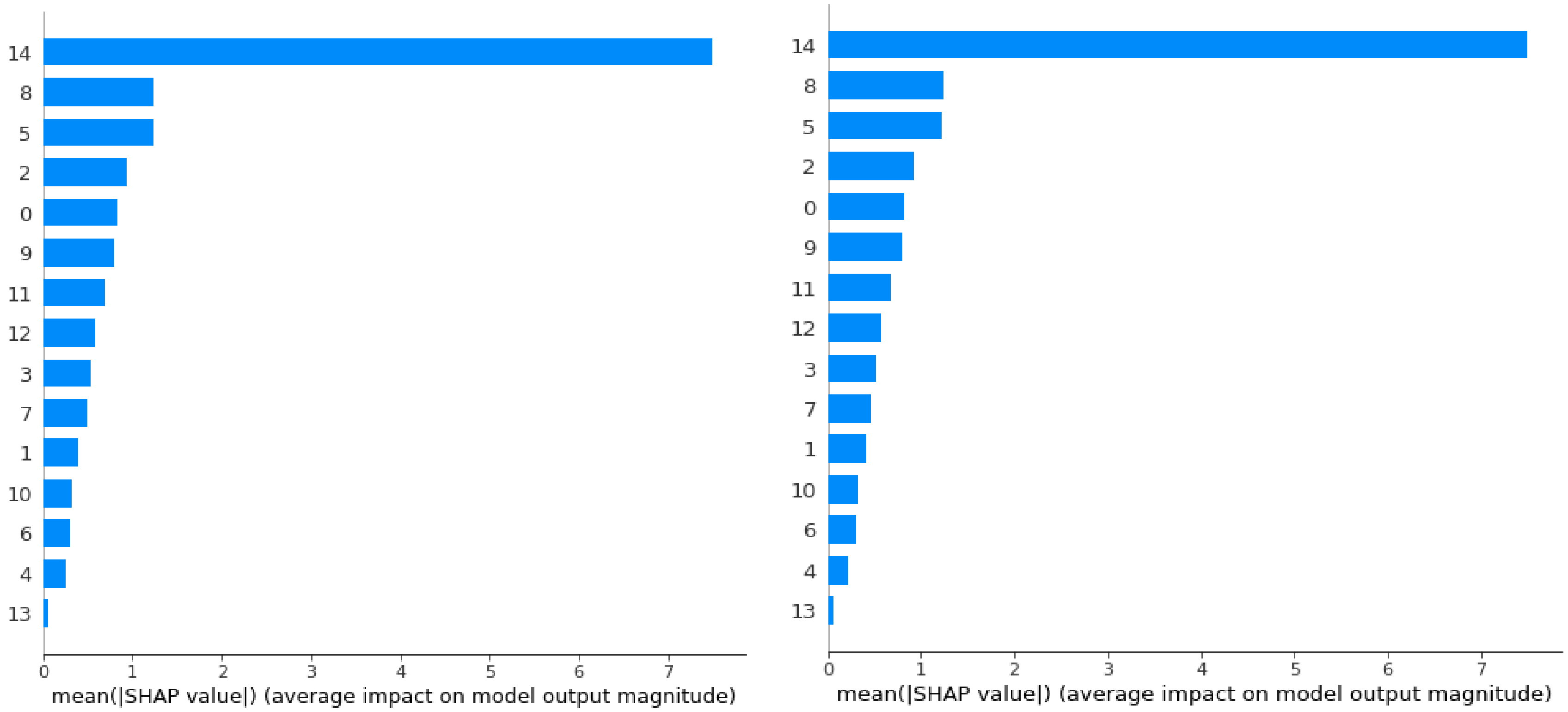
Built – in feature importance by LightGBM

The two models agree on the most and least important features, but not entirely in the intermediate ones





Popularity prediction – LightGBM model & Shap values



Shap values of the two models

Left: MSE model
Right: MAE model

Shap values of the two models are identical.

Popularity prediction – LIghtGBM model with selected features

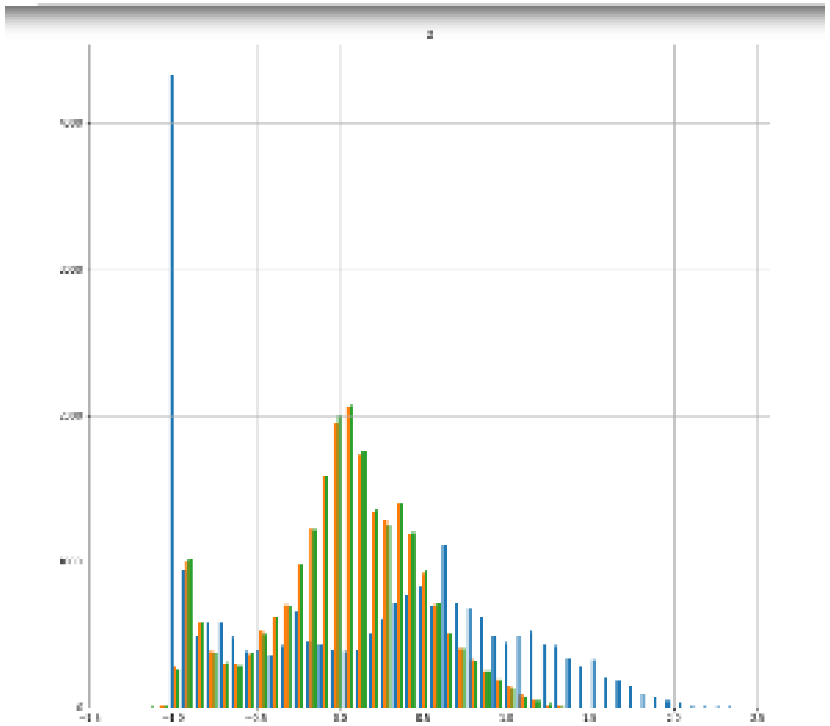
```
less_variables = ['duration_ms', 'danceability',  'energy', 'loudness', 'speechiness',
                 'acousticness', 'instrumentalness',  'liveness', 'valence', 'tempo', 'year']
```

	MSE model	MAE model
MSE	165.81	167.33
RMSE	12.88	12.94
MAE	9.69	9.74
R2	0.5045	0.4999

So after droping the 4 least important features, the model's performance is slightly worse but slightly faster.

Popularity prediction – LightGBM model

2. RobustScaler for both training and target variable



Blue distribution: Scaled true popularity

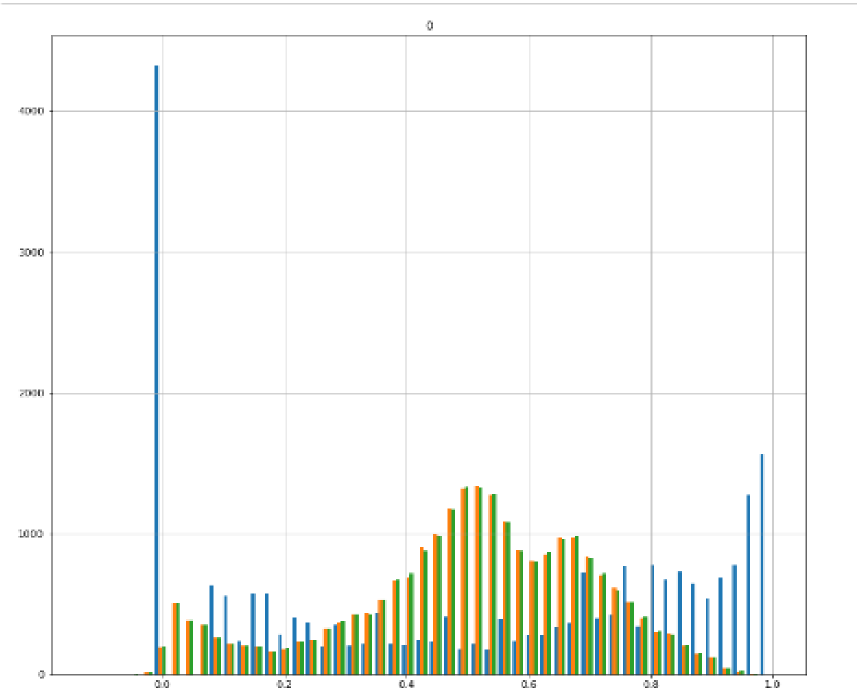
Orange&Green distributions: MSE and MAE predicted popularity distributions

	MSE model	MAE model
MSE	0.208318	0.20986
RMSE	0.4564	0.4581
MAE	0.343115	0.344489
R2	0.51191739	0.508303

Both models seem to perform slightly better when we scale target variable too

Popularity prediction – LightGBM model

3. Scale both training and target variables with QuantileTransformer (output = uniform)



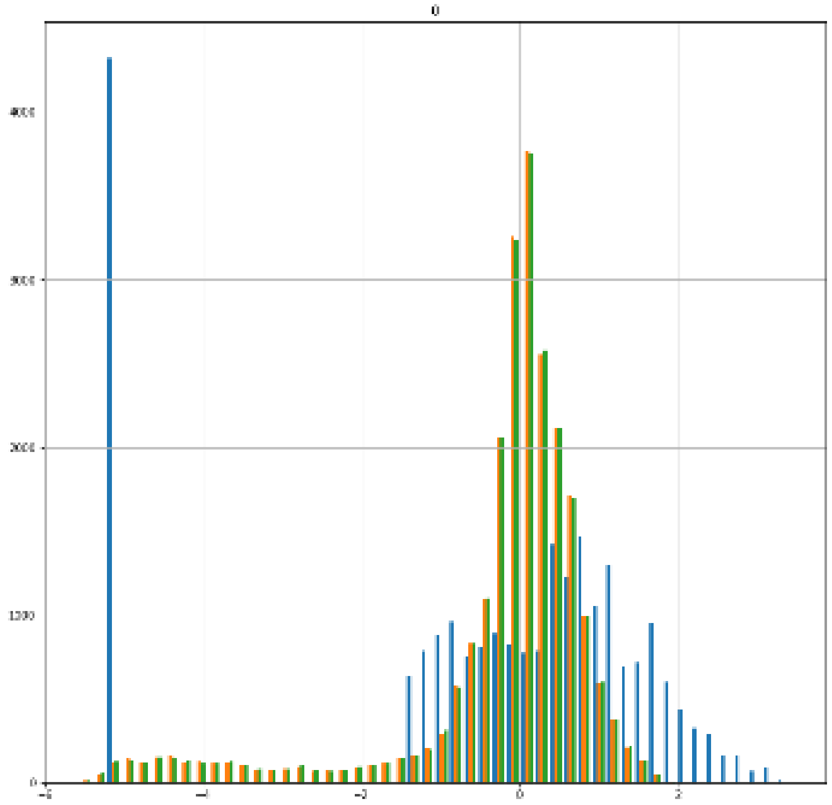
Blue distribution: True scaled popularity

Orange & Green: Predicted popularities

	MSE model	MAE model
MSE	0.0405068	0.0404341
RMSE	0.201263	0.20108235
MAE	0.15631	0.15613294
R2	0.526595	0.527445

Popularity prediction – LightGBM model

4. Scale both training and target variables with QuantileTransformer (output = normal)



Blue distribution: True scaled popularity

Orange & Green: Predicted popularities

	MSE model	MAE model
MSE	1.0147753	1.01038
RMSE	1.00736	1.00517
MAE	0.6642	0.662656
R2	0.6215677	0.623206

Scaling the data with QuantileTransformer normal seems to improve both models' performance by a lot

Popularity prediction – LightGBM model Final

- Scaler : QuantileTransformer (output distribution = normal) applied both to training and target variables
- Training variables: ALL, as reducing them reduced the performance

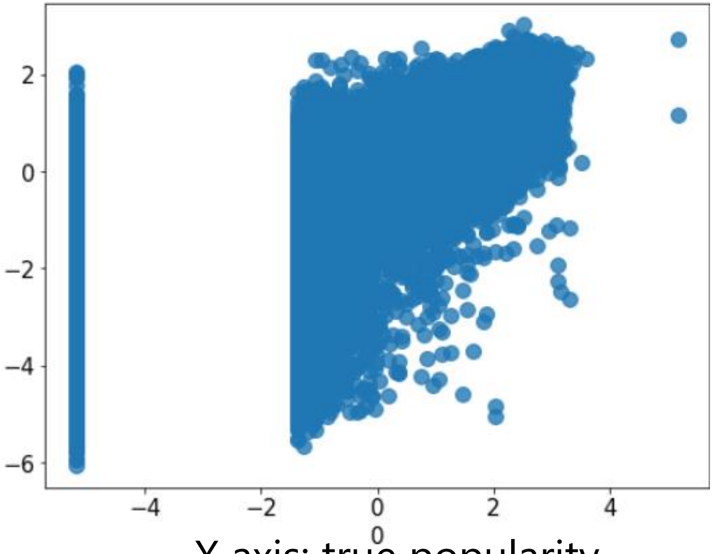
  Including artists features and OneHotEncoding of 'key', 'time_signature' and 'instrumentalness'

- Model was Hyperparameter optimised by RandomizedSearchCV

```
best_hyper_params = {
    'task': 'train',
    'boosting_type': 'gbdt',
    'objective': 'regression',
    'metric': ['mean_squared_error'],
    'learning_rate': 0.018182496720710064,
    'feature_fraction': 0.6508884729488529,
    'bagging_fraction': 0.9699098521619943,
    'bagging_freq': 19,
    'verbose': 0,
    'max_depth': 12,
    'l1': 0.08324426408004218,
    'l2': 0.021233911067827616 ,
    "max_bin": 320,
    "num_iterations": 50000,
    "n_estimators": 1000,
    'min_data_in_leaf':127,
    'min_sum_hessian_in_leaf': 0.5247564316322378,
    'num_leaves': 238

   }
```

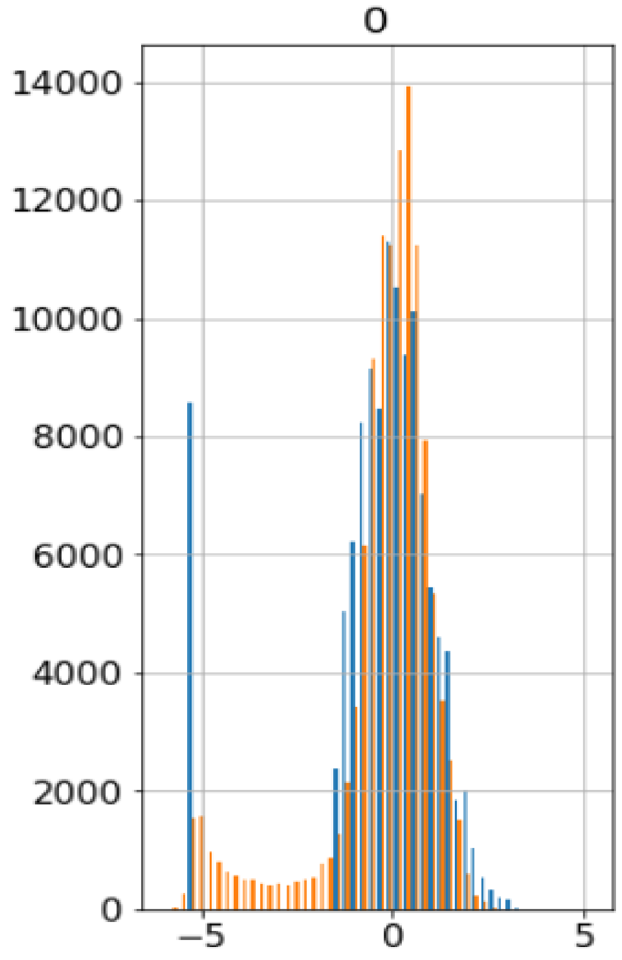
Popularity prediction – LightGBM model Final - Results

	Train	Test
MSE	0.46	0.68
RMSE	0.68	0.82
MAE	0.42	0.51
R2 SCORE	0.83	0.74



X axis: true popularity
Y axis: predicted

It is evident from the scatter plot on the left and the distributions on the right that the algorith has a hard time predicting the popularity when it is very low and it assigns a random value to it.



Blue: true
Orange: predicted

"

Thank you, we really learned a lot on this extensive project

Slide 69 – Haha ☺