COMM055 Coursework:

Beanie Intelligence Analysis Report

University of Surrey

Word Count: 1518

Contents

1 Introduction 3

2 Data sets 3
2.1 Spotify API Data Set 3
2.1.1 Funky Hypothesis 3

3 Hypotheses 3
3.1 Spotify API Data Set 3
3.1.1 Funky Hypothesis 3

4 Data Preparation 3
4.1 Spotify API Data Set 3
4.1.1 Funky Hypothesis 3

5 Modelling 6
5.1 Spotify API Data Set 6
5.1.1 Funky Hypothesis 6

6 Results and Evaluation 7
6.1 Spotify API Data Set 7
6.1.1 Funky Hypothesis 7

7 Conclusion and Further Work 12
7.1 Spotify API Data Set 12
7.1.1 Funky Hypothesis 12

8 Appendix 14
8.1 Appendix 1: Funky Hypothesis 14

1 Introduction

This document presents the Data Mining process over two data sets with special emphasis on model development, evaluation and comparisons. Several hypotheses are discussed with their corresponding data pre-processing and modelling phases.

2 Data sets

2.1 Spotify API Data Set

2.1.1 Funky Hypothesis

Using the Spotify Data Mining tool our team created [2], I downloaded the top 200+ Funk songs [1], with the addition of an extra 13 that are mentioned as "Honorable mentions". The reason I decided to choose this list of songs was to eliminate as much personal bias as I could from the dataset.

3 Hypotheses

3.1 Spotify API Data Set

3.1.1 Funky Hypothesis

Using a few of the features listed in [3], some of the most promising features seemed to be "*energy*", "*time_signature*", "*key*", "*mode*", "*popularity*", "*valence*" and "*tempo*". With that in mind, I set out to predict the "*danceability*" label of a given song. The creation of the label is described in section 4.1.1.

4 Data Preparation

4.1 Spotify API Data Set

4.1.1 Funky Hypothesis

First, I checked the correlation and relationships between the different variables. I disregarded all the variables that were calculated from each other (i.e. had 1.00 linear correlation). The only correlations that are greater than 0.25 and less than 1.00 are the following:

- "*loudness*" and "*energy*" ≈ 0.594
- "*danceability*" and "*valence*" ≈ 0.404
- "*speechiness*" and "*liveness*" ≈ 0.271

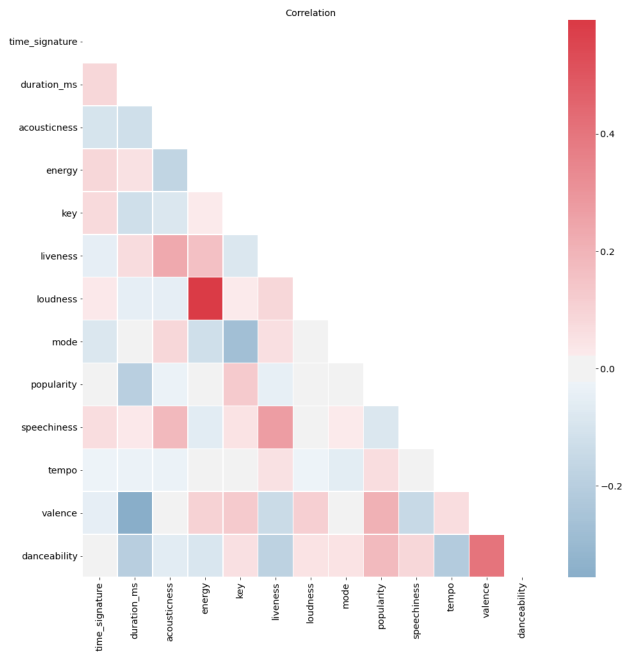


Figure 1: Correlation between selected features.

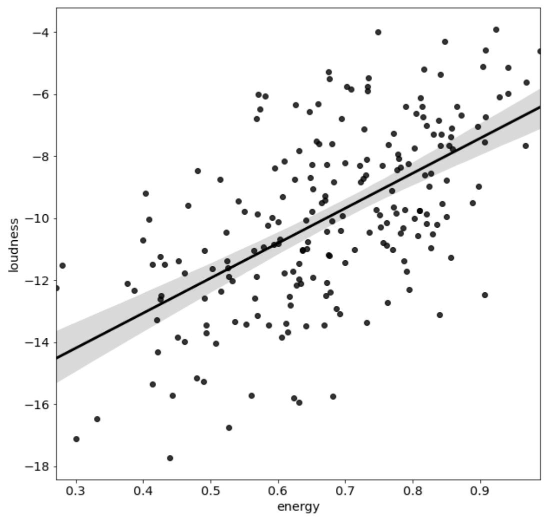


Figure 2: Correlation between energy and loudness.

Even though there is no apparent correlation with the "*danceability*" feature, other than "*valence*", which indicates how happy or sad a song is, I decided to proceed with my hypothesis.

Using the numerical "*danceability*" feature provided via the Spotify Analyzer, I created a "*danceability_labels*" feature that has 4 classes:

- Class 0: Not danceable, $\forall$ "*danceability*" $\in [0.0, 0.5)$, with 14 songs.
- Class 1: Somewhat danceable, $\forall$ "*danceability*" $\in [0.5, 0.7)$, with 65 songs.
- Class 2: Very danceable, $\forall$ "*danceability*" $\in [0.7, 0.8)$, with 63 songs.
- Class 3: Extremely danceable, $\forall$ "*danceability*" $\in [0.8, 1.0)$, with 71 songs.

Furthermore I looked into the skewness of each attribute, or the measure of the asymmetry of the distribution of each of the selected features. This is better represented in figures (See section 8.1).

Column	Skew
time_signature	-10.246
duration_ms	3.658
acousticness	1.108
energy	-0.307
key	-0.147
liveness	1.905
loudness	-0.175
mode	-0.336
popularity	-0.188
speechiness	2.225
tempo	1.647
valence	-1.311
danceability	-0.773

Table 1: Skewness of attributes.

The results of checking for the skewness of each parameter shows that "*time_signature*" or how many beats are in each bar of a song, was the most skewed one. Which also validates why it has almost 0 correlation with any of the other features. Similarly the skewness of "*duration_ms*" tells us that the song lengths vary widely from song to song. With that and the correlation plot (see figure 1) in mind, I decided to drop these two features from the exploration, instead of removing the skewness from them. This decision was taken, due to the limited size of the dataset.

5 Modelling

5.1 Spotify API Data Set

5.1.1 Funky Hypothesis

After doing some research on which models would be best to run, I decided on the following:

- Multi-Class Support Vector Machine (SVM):
  - Even though SVM models are inherently 2-class classifiers. The current class model I am working with has 4 classes, which can be binarized. Then a Multi-Class SVM can be used to solves the problem as it forms multiples of two classes. [5]

- Multi-Class Decision Tree:
  - The Decision Tree classifier, is a classic approach for Multi-Class classification.
- Naive Bayes, KNN and the Bagging classifier are all also Multi-Class classifiers that I decided to look into as a batch.

6 Results and Evaluation

6.1 Spotify API Data Set

6.1.1 Funky Hypothesis

As discussed in section 5.1.1, 3 separate model "batches" were run to predict the "*danceability_label*" of a funk song.

Initially the Multi-Class SVM Classifier was ran, as following:

```
OneVsRestClassifier(svm.SVC(
                kernel='poly', degree=2, probability=True, tol=1e-6,
                    random_state=self.random_seed))
```

The OneVsRestClassifier and the pre-processing to ensure the feature columns are fed in binarized assures that the model will run as intended.

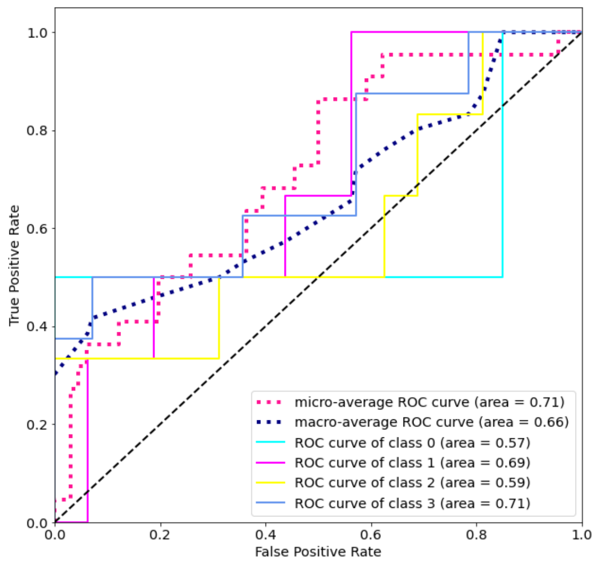


Figure 3: The ROC curve for the different class prediction accuracy that SVM produced.

In evaluating this model, we can observe that even though the averages for the area under the curve, indicate a well performant model, it is not the case. Looking at the area under the curve for each individual class, we can see that in class 0 with the least number of songs, and class 3 the model did not perform as well.

The Multi-Class Decision Tree Classifier was ran with the default settings (see [4]). Pre-processing ensued, to ensure the feature columns are binarized such that the model will run as intended.

Classes	**precision**	**recall**	**f1-score**	**support**
0	0.00	0.00	0.00	2
1	0.33	0.33	0.33	6
2	0.27	0.50	0.35	6
3	0.60	0.38	0.46	8
accuracy			0.36	22
macro avg	0.30	0.30	0.29	22
weighted avg	0.38	0.36	0.35	22

Table 2: Confusion Matrix for Decision Tree

In evaluating this model, we can observe that due to the 0.1 % split for the test data, there were only two songs for class 0, which where miss classified, similarly for all other classes, the sample of songs should not make that much difference in the overall accuracy and in this case could be attributed to the random seed the model was run with, since the model accuracy is only 36 %.

With the following results in mind I decided to run a k-fold cross validation algorithm on a batch of models (including the Decision Tree), to test if this would smooth out the bias of the models and reveal their true accuracy. As mentioned in section 5.1.1, I decided to run the "*Naive Bayes - Gaussian*", "*KNN*", "*Bagging*" and "*Decision Tree*" classifiers.

The settings for the following models were as such:

```
extra_tree = ExtraTreeClassifier(random_state=self.random_seed)
models = [('Naive Bayes', OneVsRestClassifier(GaussianNB())), ('KNN',
    OneVsRestClassifier(KNeighborsClassifier(n_neighbors=len(
            X.columns)+1, metric='euclidean', n_jobs=-1, weights='
                distance'))),
        ('Bagging', OneVsRestClassifier(
            BaggingClassifier(extra_tree, random_state=self.random_seed
                )))),
            ('Decision Tree', OneVsRestClassifier(
                DecisionTreeClassifier()))]
```

The following models where run with k-fold cross validation of $k = 10$ and evaluated with ROC curves.

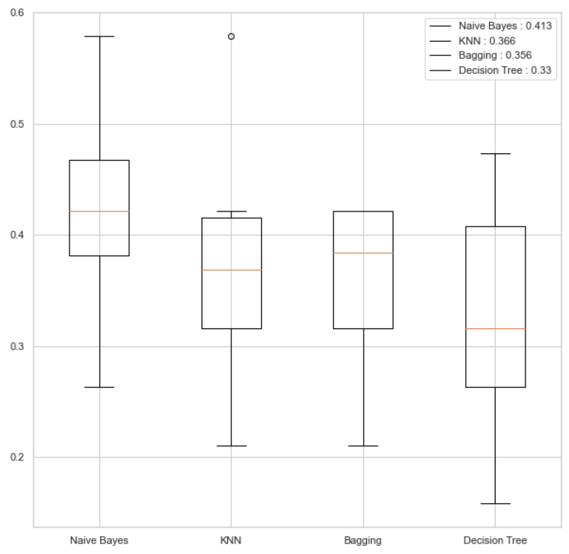
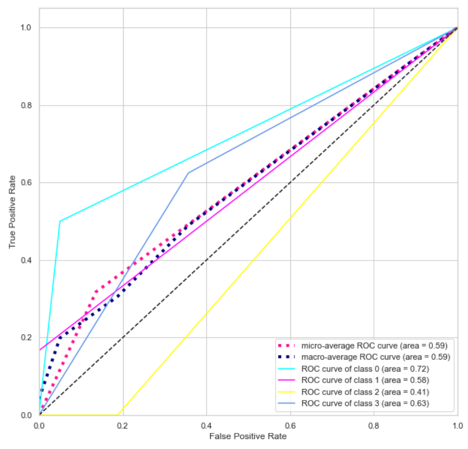
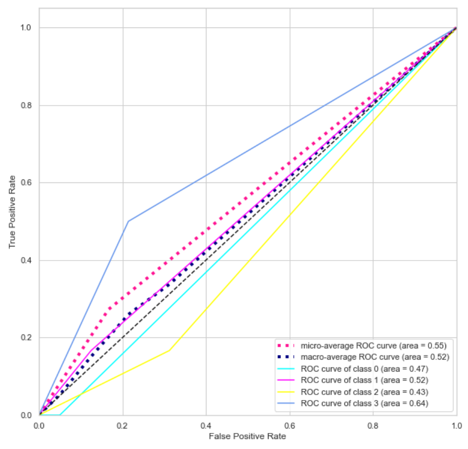


Figure 4: The average model accuracy after 10-fold cross validation.

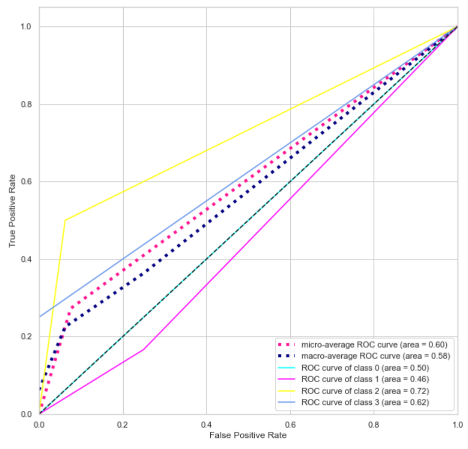
Looking at the cross validation results it seems that the "*Naive Bayes*" Gaussian Classifier is the most accurate out of all the classifiers with the least amount of variance between each k-fold validation. It seems that the "*Decision Tree*" Classifier is the worst performant one of them all with the highest variance between each k-fold validation.



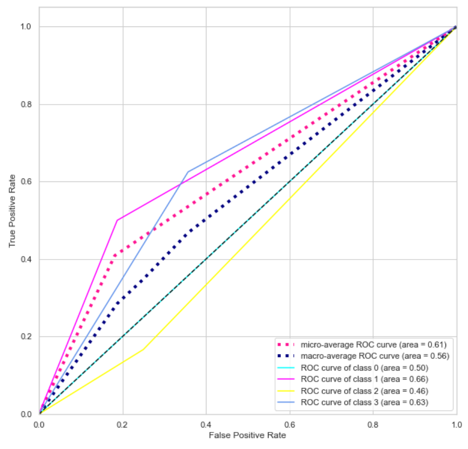
(a) Naive Bayes (Gaussian) Classifier



(b) KNN Classifier



(c) Bagging Classifier



(d) Decision Tree Classifier

Figure 5: Put your caption here

In order to evaluate each one of the model's performance more thoroughly we must observe the ROC curve for it (see figure 5). It seems that all three "*Naive Bayes*", "*KNN*" and "*Bagging*" Classifiers, performed really badly in predicting class 3, this is also reflected in the AUC value for that class. In contrast the "*Bagging*" Classifier, seems to have a lot better performance at predicting class 3, although it falls short in predicting class 0, which is as should be expected, given that class 0 only has 14 data points before the train, test split. Even after the stratified split for each class and the 10-fold cross validation though, it seems that the "*Bagging*" Classifier needs more data to improve in accuracy.

They all seem to generalise relatively well, when looking at their average AUC, even if their performance is not great.

7 Conclusion and Further Work

7.1 Spotify API Data Set

7.1.1 Funky Hypothesis

In conclusion, this hypothesis has been a very surface exploration of what could possibly be a relatively good way in classifying whether or not a song is dance-able and to what degree. The models seemed to generalize well under k-fold cross validation, although did not have the best performance in fitting the solution space with the features selected.

There are multiple things that could be done to improve the predictive performance of any of the models used. One of them could be to download spotify's recommended playlists for Funk music, concatenate them and use them as a dataset. The idea would be that with more data, the Classifiers would be able to find any underlying patterns in the music and other features would be more strongly correlated to "*danceability*". A short-coming/bias with the approach described above is that it is assumed that the spotify playlists have 0 bias and have made sure to include only funk songs in these playlists (which is not the case).

One more way that classification could have been made easier, would be to use a deep learning neural network with an encoder-decoder system to populate a feature space with the features of the current dataset. Then use one of my own playlists with funk music to predict its danceability. In this scenario, the short-coming/bias of the neural network would come from the feature space that it has mapped initially, which is the current dataset, which is again not very big.

Last but not least, it should be noted, that even the current dataset used in the exploration can have user bias via the user that create the list [1]. One way around that would be to conduct some sort of survey or ask a few funk musicians to select which songs should be included in a funk-playlist of all time. Since Funk as a genre utilizes elements of jazz and disco, it is quite tough to pinpoint which songs are "purely-funk". Via asking multiple musicians, or simply multiple people that listen to such music, I could have taken the intersection of their playlists as the dataset for this investigation. It would still carry bias, but it would be a more scientific way of removing as much of it as possible.

References

[1] Jeff B. 200 greatest funk songs. "`https://digitaldreamdoor.com/pages/best_rb-funk.html`".

[2] beanieintelligence. spotify_data_mining. "`https://github.com/beanie-intelligence/spotify_data_mining`", Mar 2020.

[3] Spotify for Developers. Get audio features for a track. "`developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/`".

[4] Scikit. 1.12. multiclass and multilabel algorithms. "`scikit-learn.org/stable/modules/multiclass.html.`".

[5] Shuyi Zhang, Bin Guo, Anlan Dong, Jing He, Ziping Xu, and Song Xi Chen. What is a multi-class svm method? *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2017.

8 Appendix

8.1 Appendix 1: Funky Hypothesis

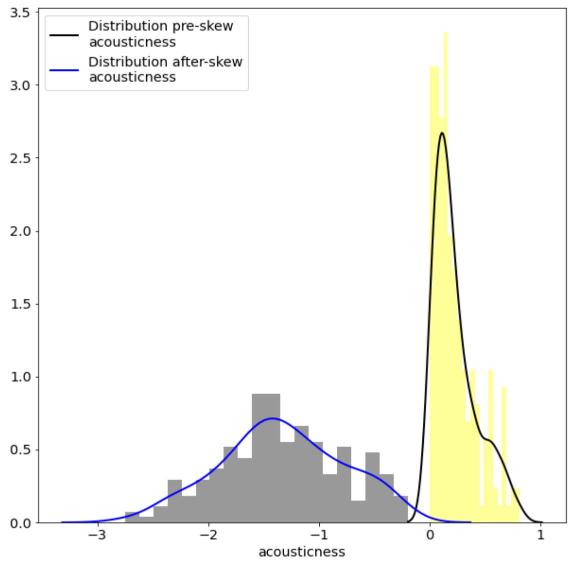


Figure 6: Funk-distribution-hist-acousticness-before-after-skew

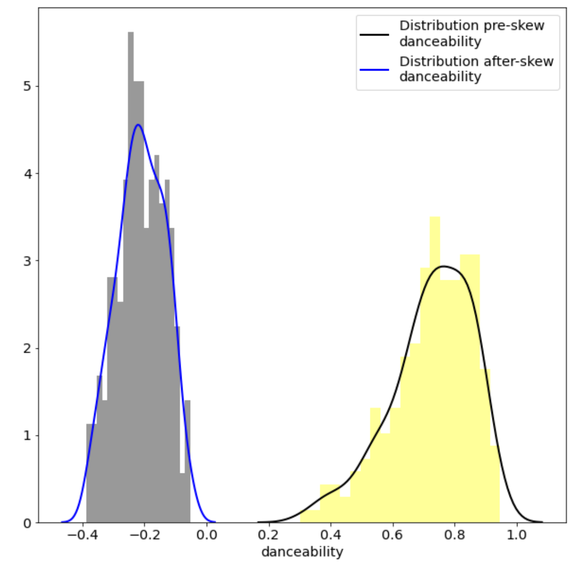


Figure 7: Funk-distribution-hist-danceability-before-after-skew

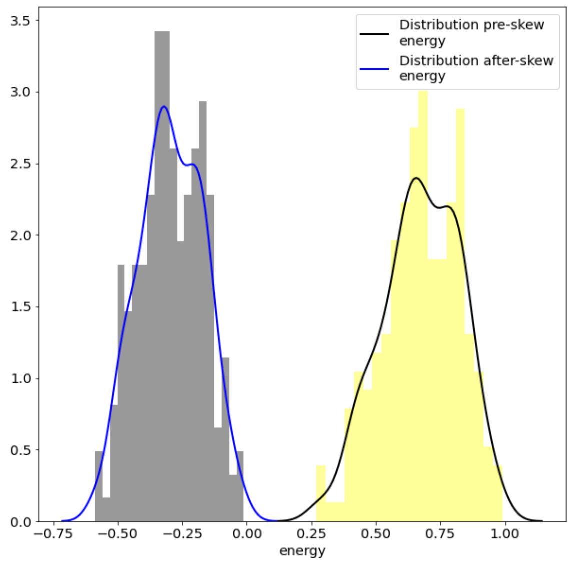


Figure 8: Funk-distribution-hist-energy-before-after-skew

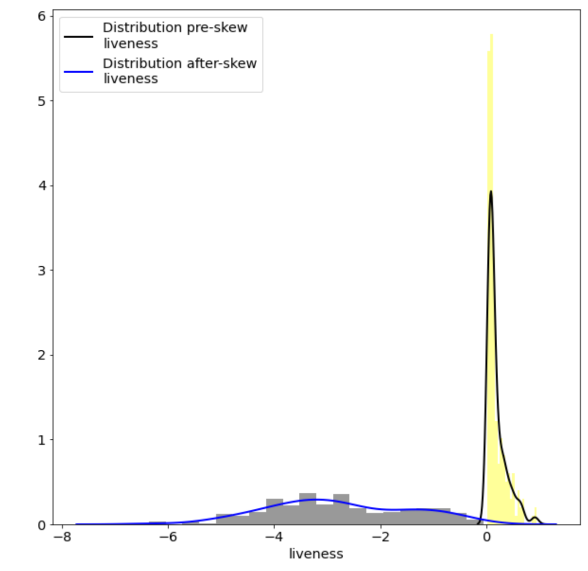


Figure 9: Funk-distribution-hist-liveness-before-after-skew

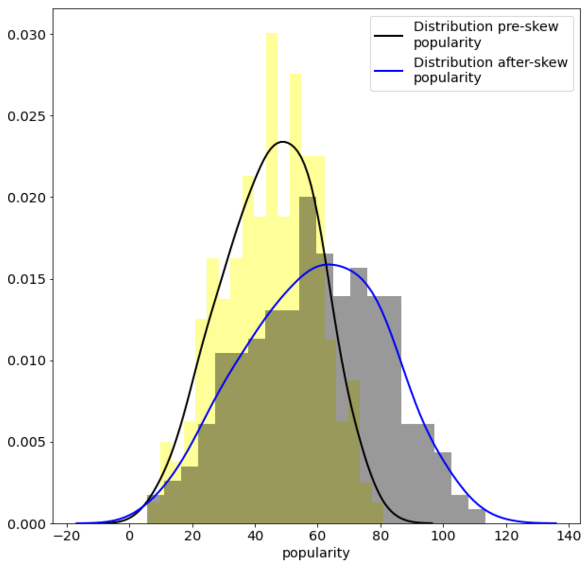


Figure 10: Funk-distribution-hist-popularity-before-after-skew

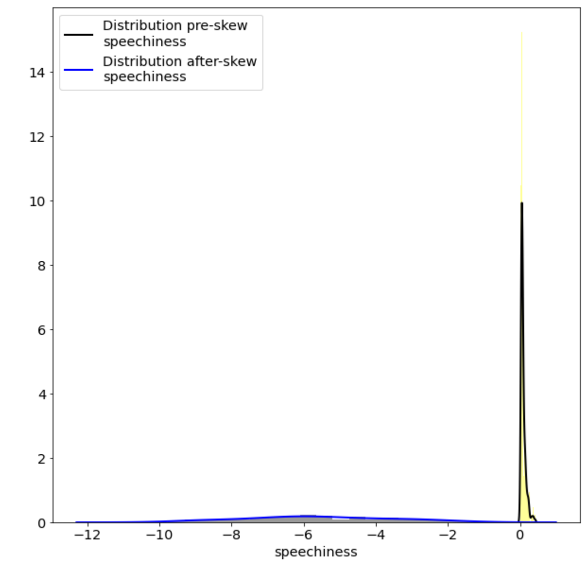


Figure 11: Funk-distribution-hist-speechiness-before-after-skew

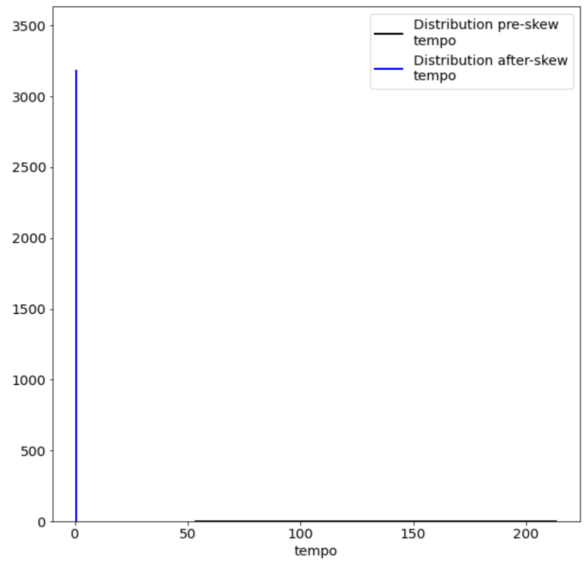


Figure 12: Funk-distribution-hist-tempo-before-after-skew

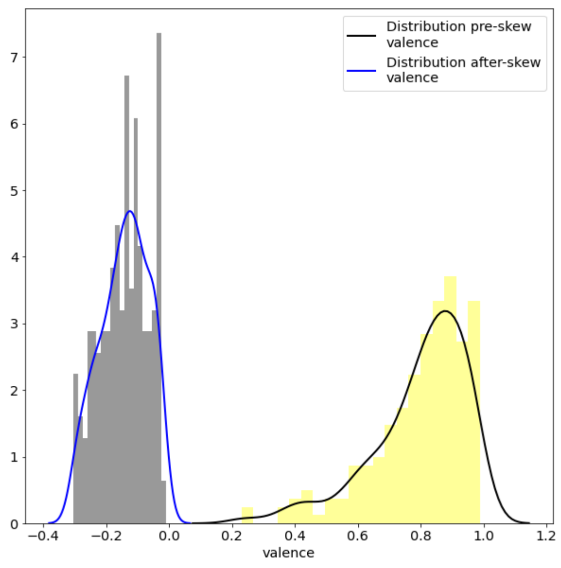


Figure 13: Funk-distribution-hist-valence-before-after-skew