Classification of musical genres

Final project for Applied Machine Learning 2020

Eliot, Mads and Sofus

Finding features

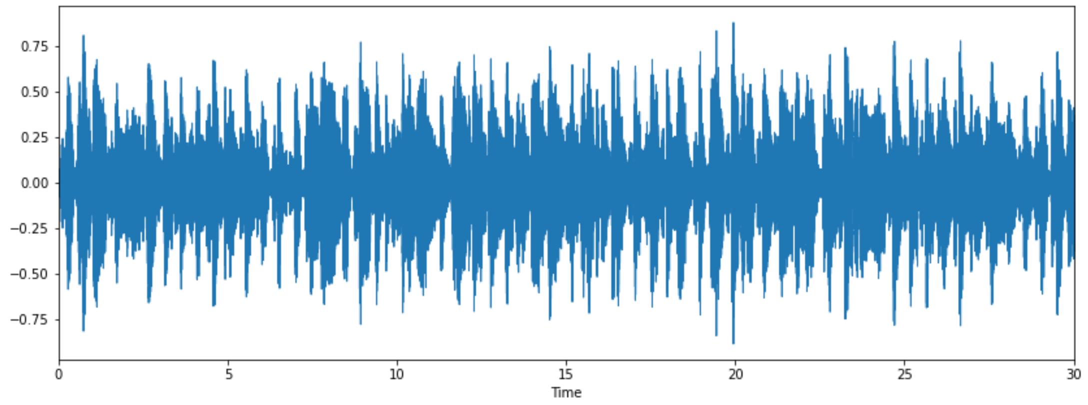
We use the GTZAN dataset [1]
- 1000 songs in 10 genres: pop, jazz, blues, classical, hiphop, metal, rock, reggae, disco, country
- Decided it was fun to find our own features and use a bunch of different approaches
- Use Timbre and Librosa python packages

train_features =

Librosa: zerocrossing, spectralrolloff, chromagramstd, chromagrammean, chroma_cens

Timbre: hardness, depth, brightness, roughness, warmth, sharpness, boominess

Additional: fourier_std, fourier_mean



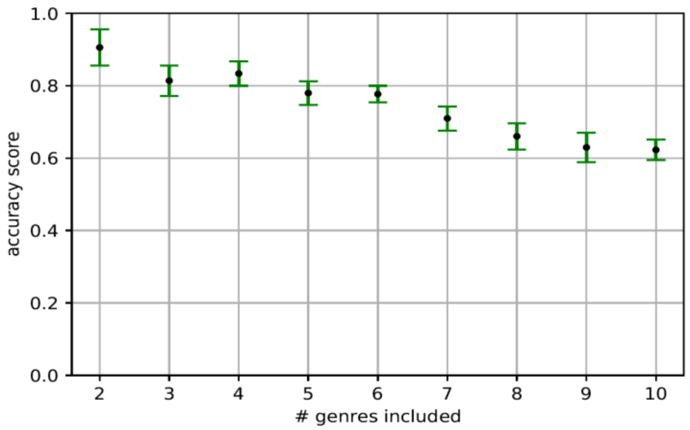
[1] https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification

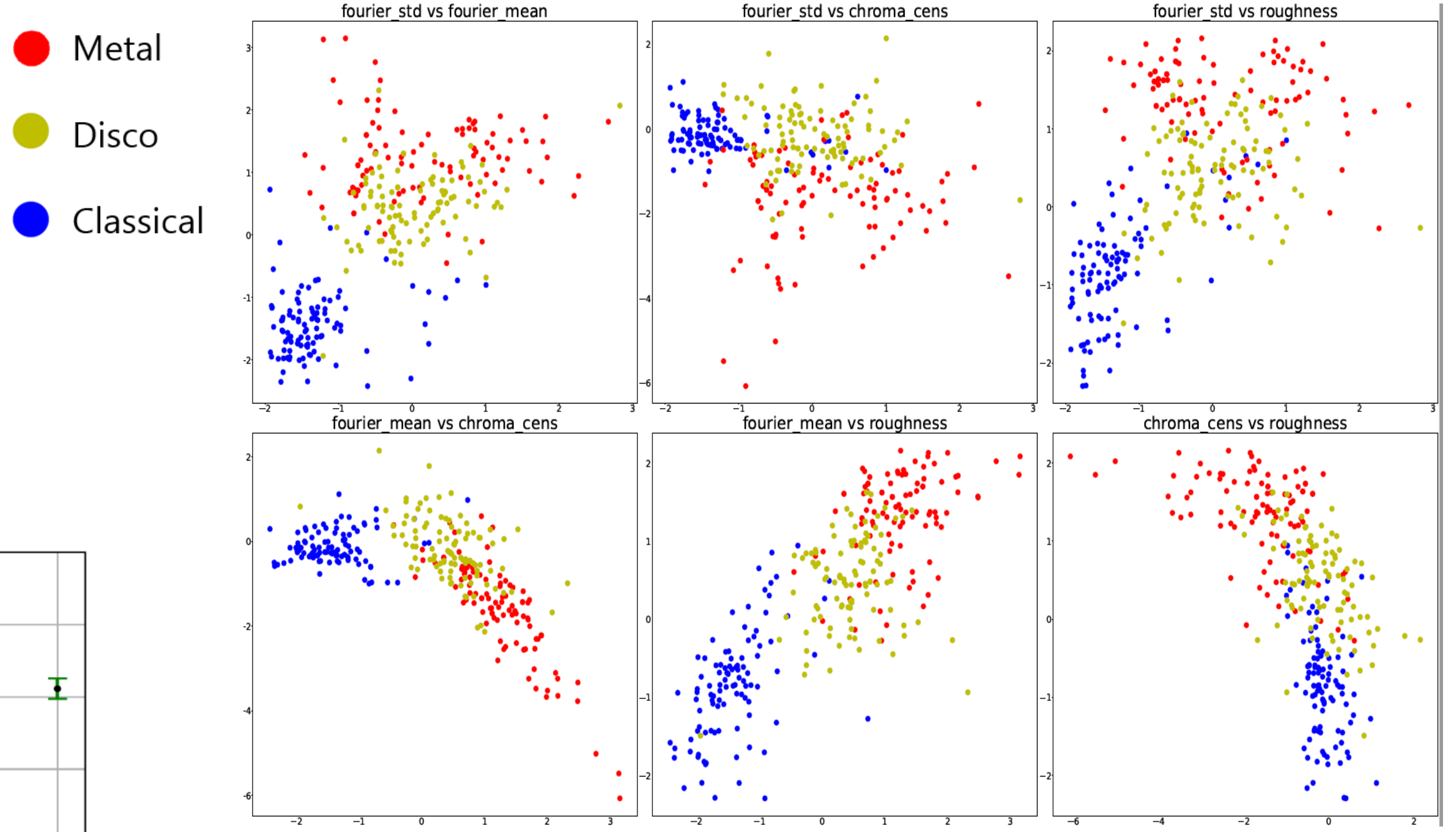
The initial approach: Gradient boosted tree and Nearest Neighbor

For three genres, kNeighbour gives an accuracy score of 0.99 +/- 0.04 with 50 fold cross validation

- This drops to 0.55+/- 0.11 for 10 genres

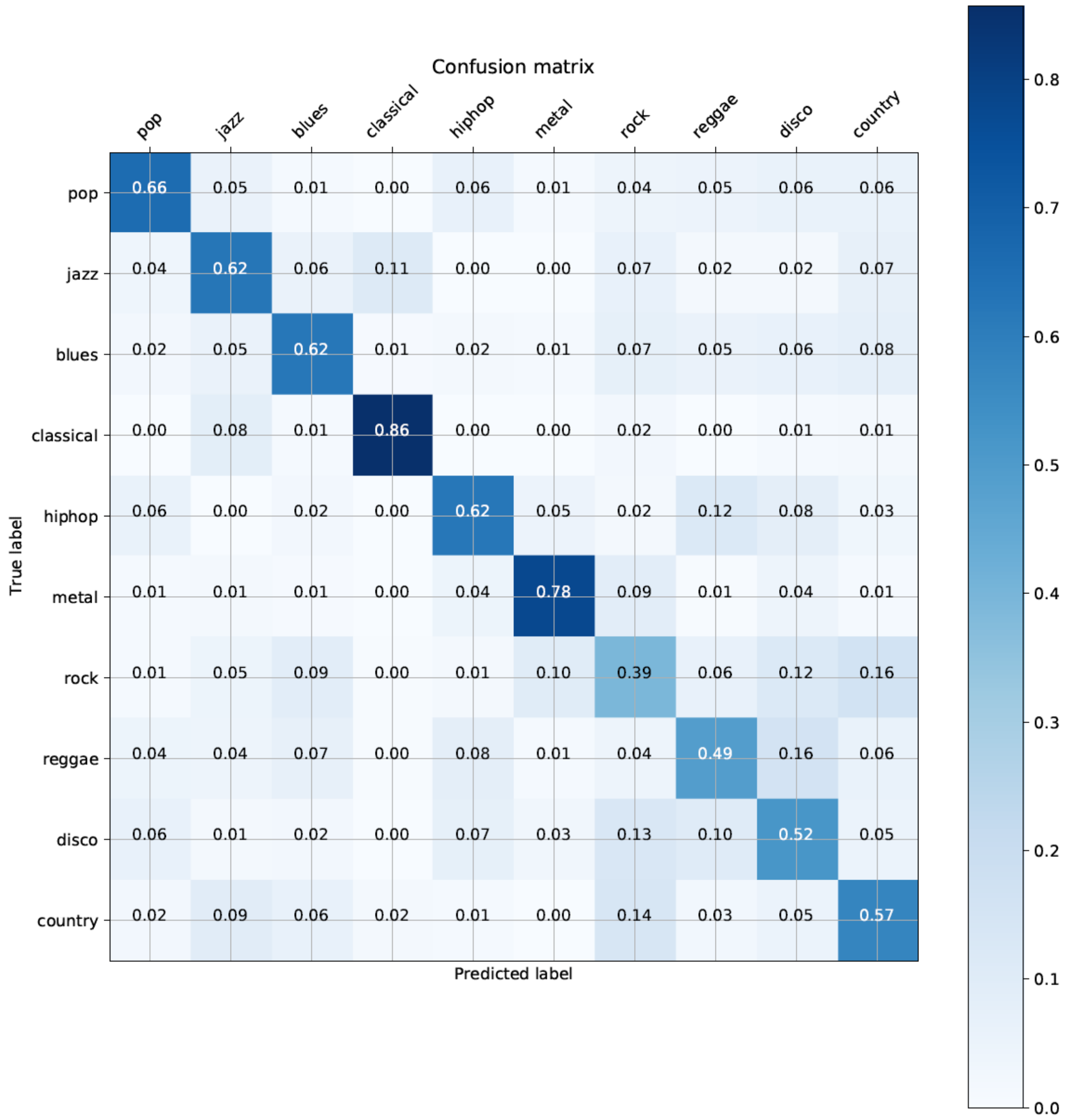
For three genres, LightGBM classifier gives an accuracy score of 0.80+/- 0.03, but holds up better for more genres.





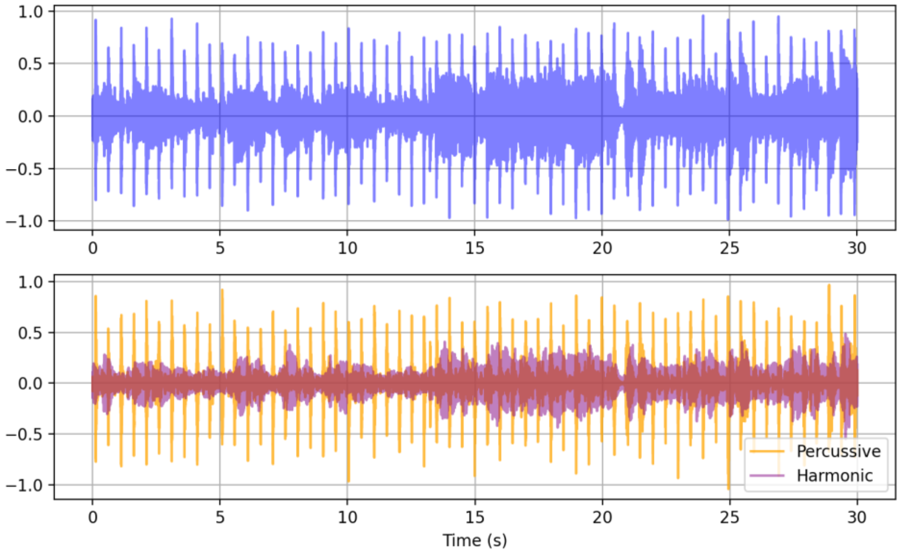
Finding errors in the prediction – what genres are hardest to predict?

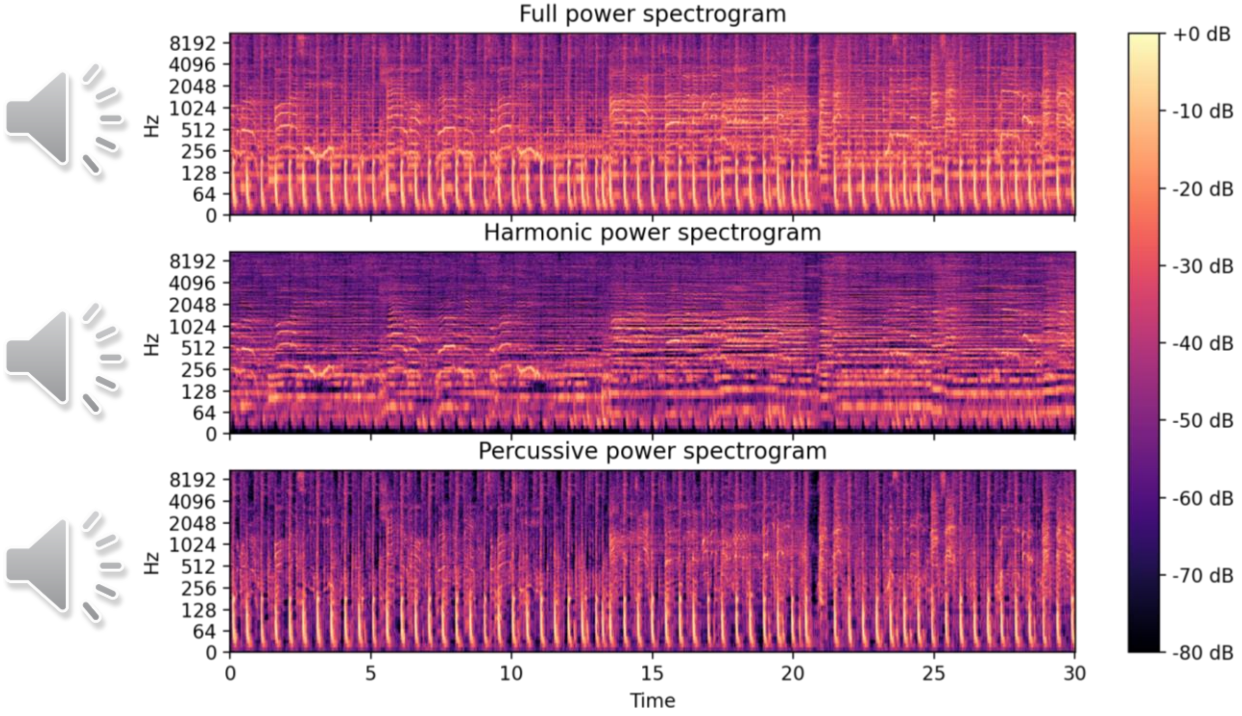
- Classical and metal are easiest to predict
- Rock is hardest, which makes sense intuitively – vaguely defined!
- Can we introduce variables that improves the guesses on the lowest scoring classes?



Harmonic Percussive Source Separation

- Median filtering of spectrogram
- Analyse melodic and rythmic features separately

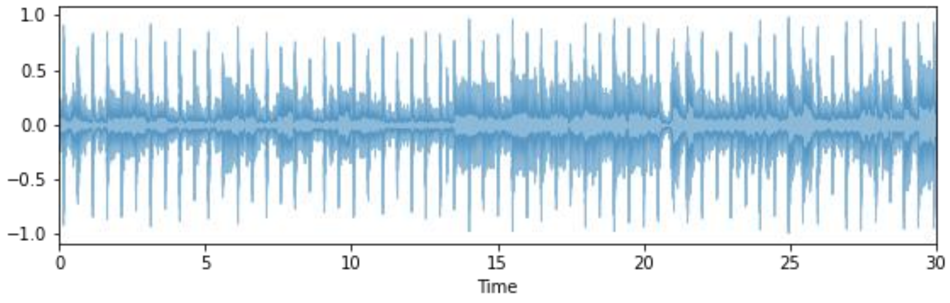


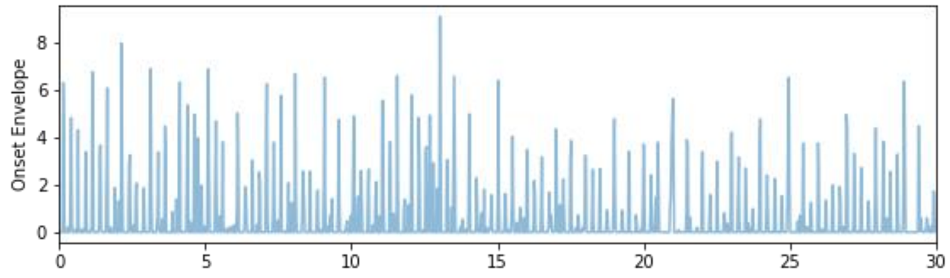


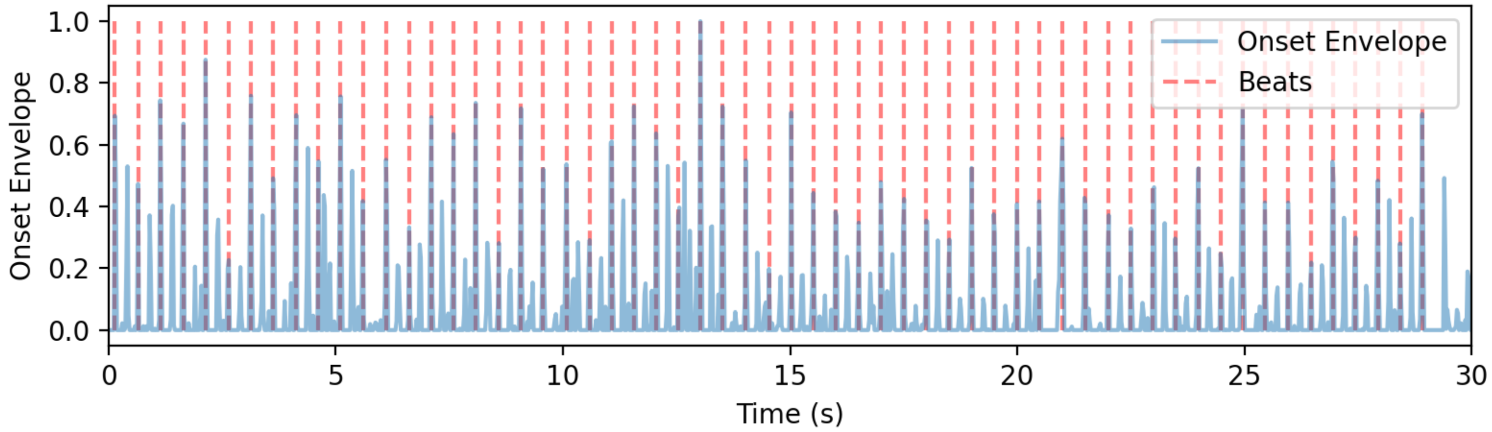
For details on HPSS, see: Derry FitzGerald Proc. of the 13th Int. Conference on Digital Audio Effects (DAFx-10) (2010)

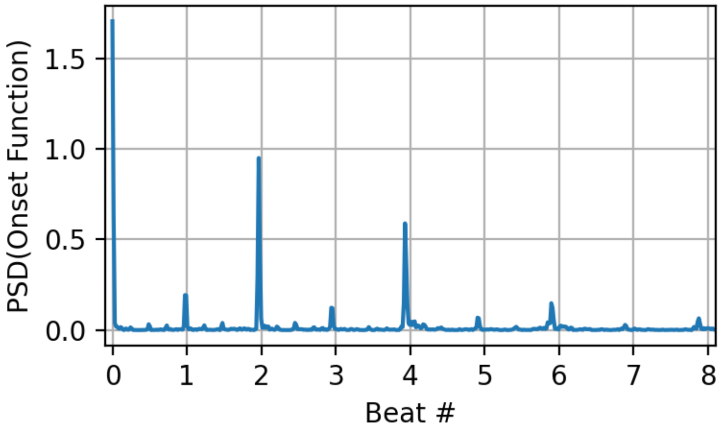
Rythmic features

- Tempo : estimate from autocorrelation
- onBeat : $\frac{\int \delta(t-t_n)\, OE(t) dt}{\int OE(t) dt}$
- Overlap : $\int PSD[OE](f)\ \mathrm{norm}(f; \mu_i, \sigma_i)\ df$
  - Early and Late
- Synco : Early - Late





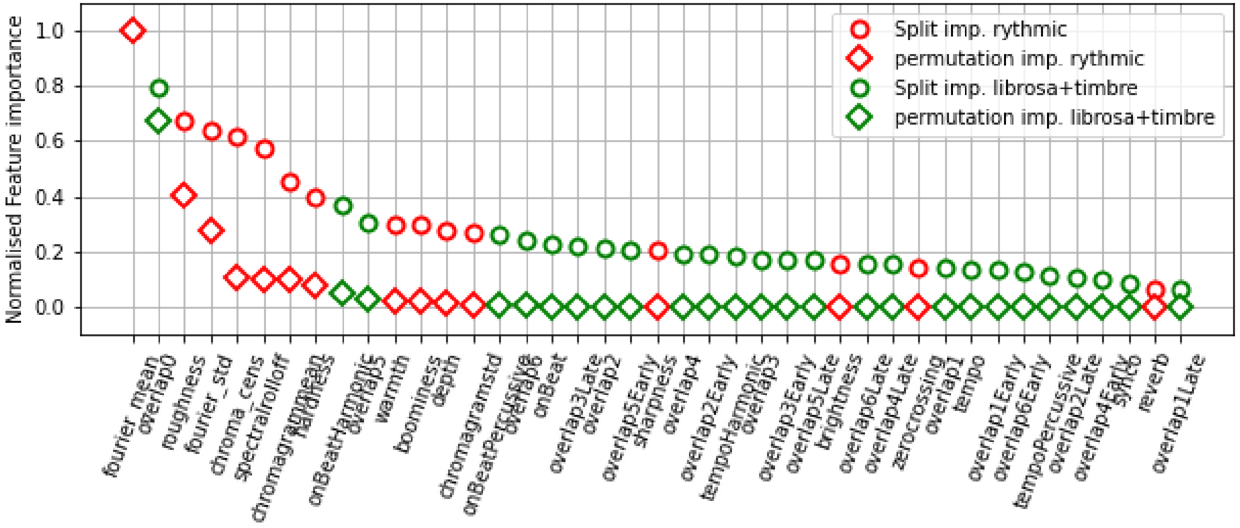


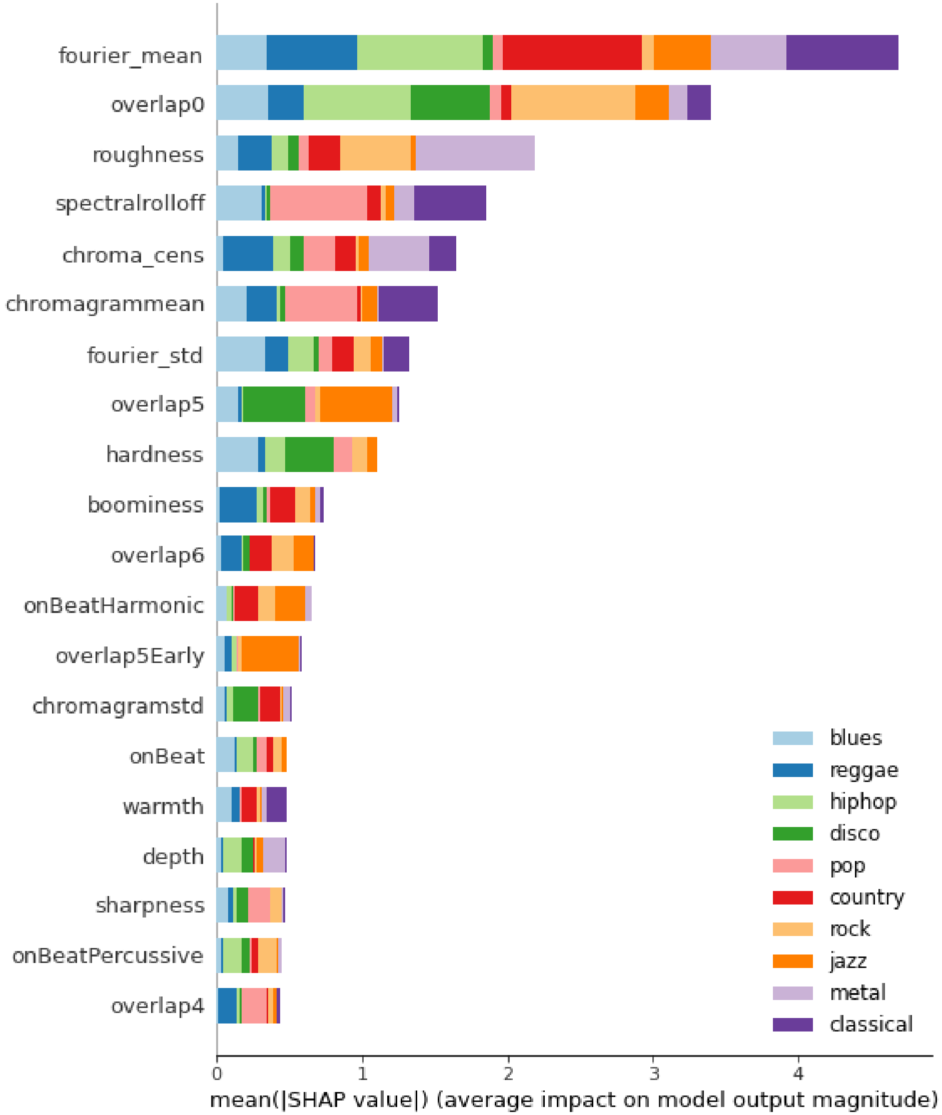


Feature Importance

- Permutation (10 repeats) and split importance
- SHAP values

··· Homemade features are competitive!



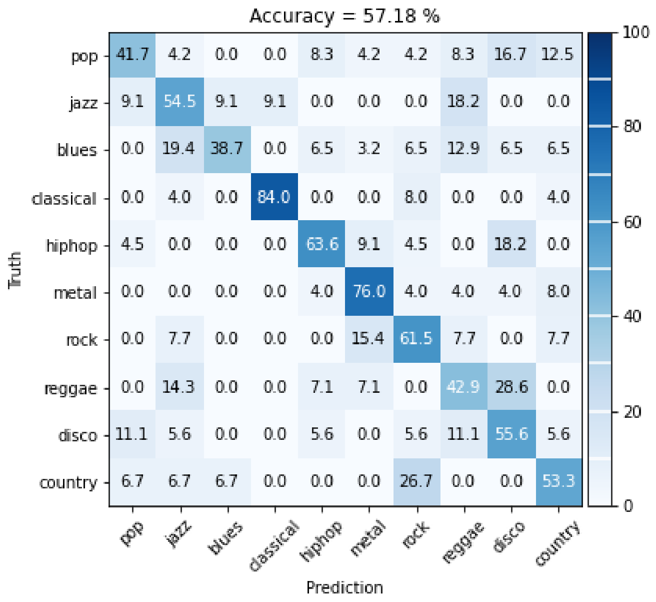


Improvement?

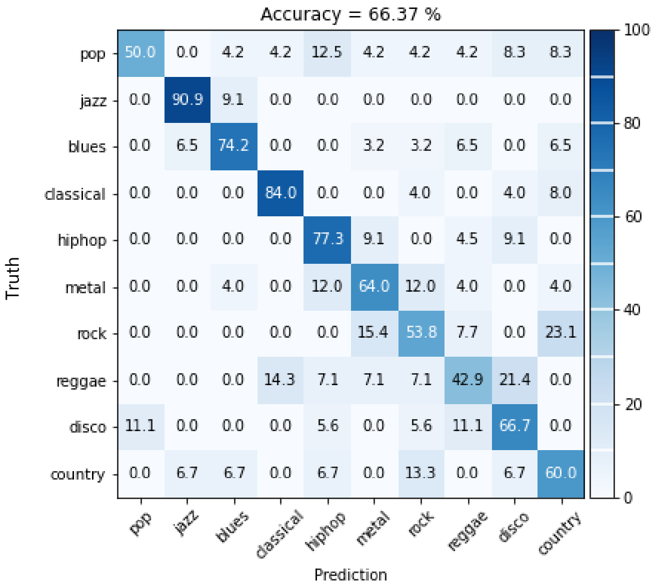
- LightGBM Classifier (log loss)
- StandardScaler, Data split Train:Val:Test =70:10:20
- Hyper parameter optimization (GridSearch with 5CV)
  - $\Gamma_{\text{learn}} = 8.14 \times 10^{-4}$, max depth $= 6$, num leaves $= 9$, $n_{\text{estimators}} = 30000$ (early stopping 1000)

Modest improvement!

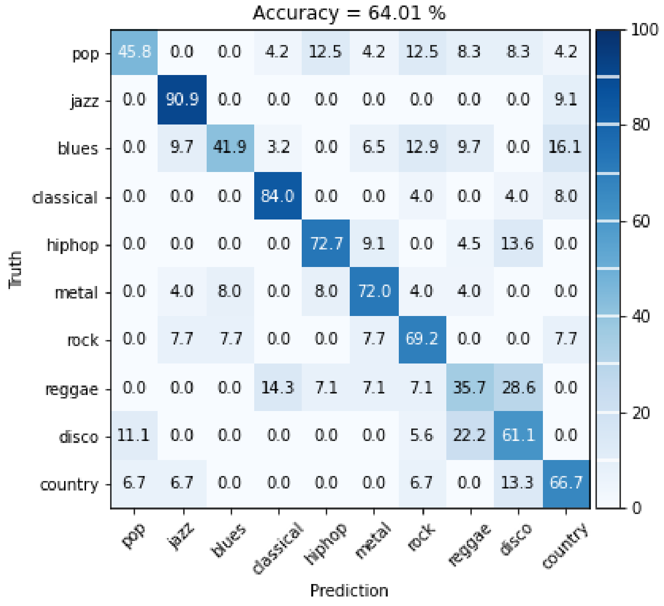
Librosa & timbre:



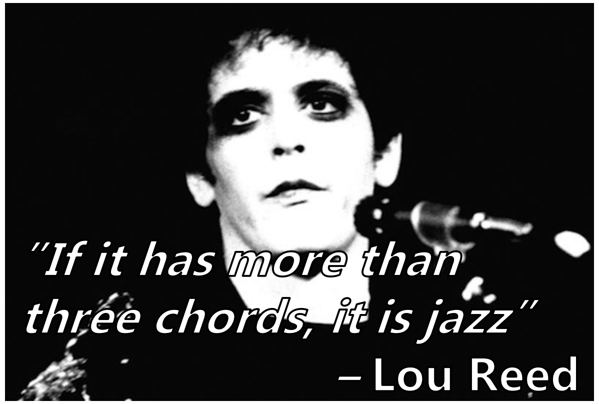
All 41 variables:

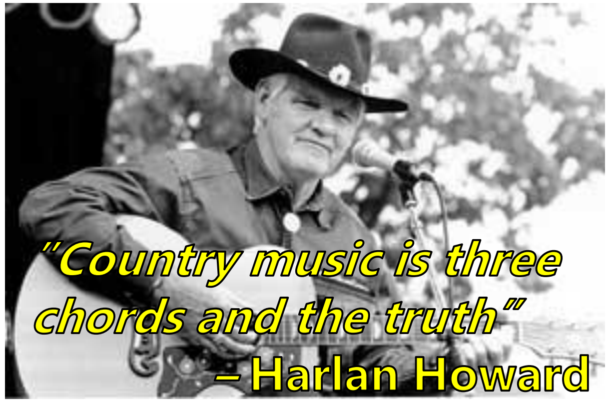


Top 10 variables:



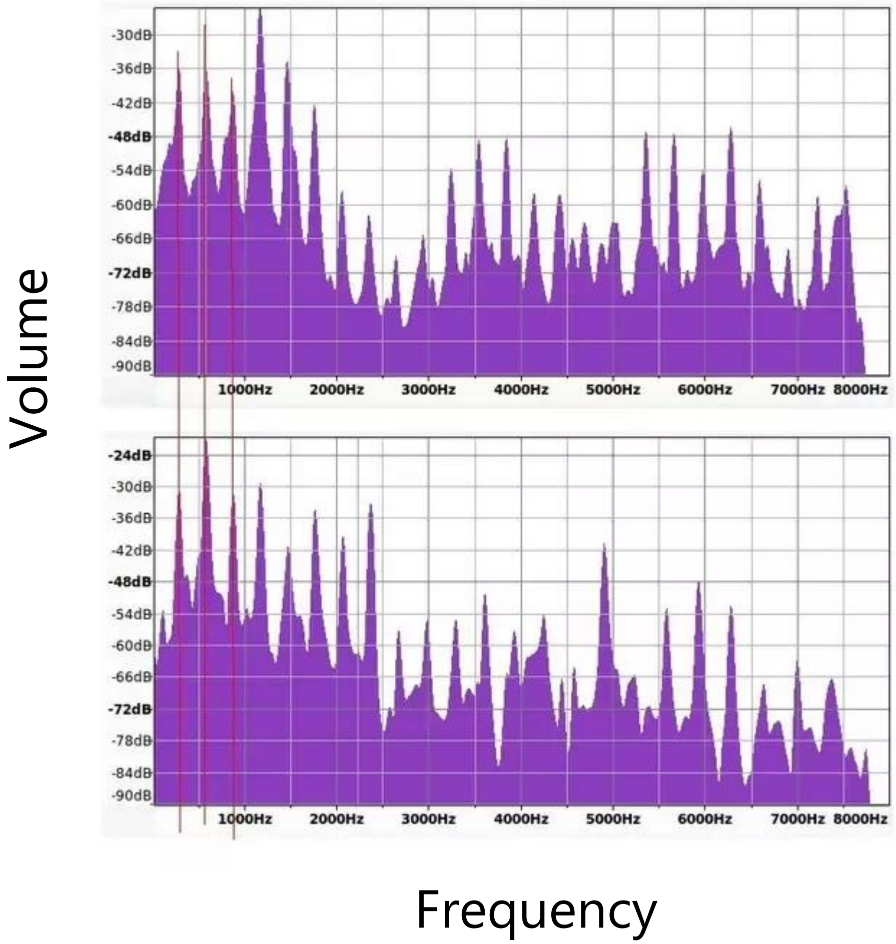
- Rythmic features improves model somewhat
- Surprising for me that "Early" & "Late" variables aren't more important
- Many variables, little data...
  - Get more data
  - Dimensionality reduction
  - Recursive feature elimination
- More advanced hyper parameter optimization
- Extract chords and their progressions from Harmonic component





Different approach: Neural network with MFCC's

If you play the same note on a guitar and a piano with the same amplitude, what makes them sound different is **timbre**



Finding features

Waveform

↓

DFT

↓

Log-Amplitude Spectrum

↓

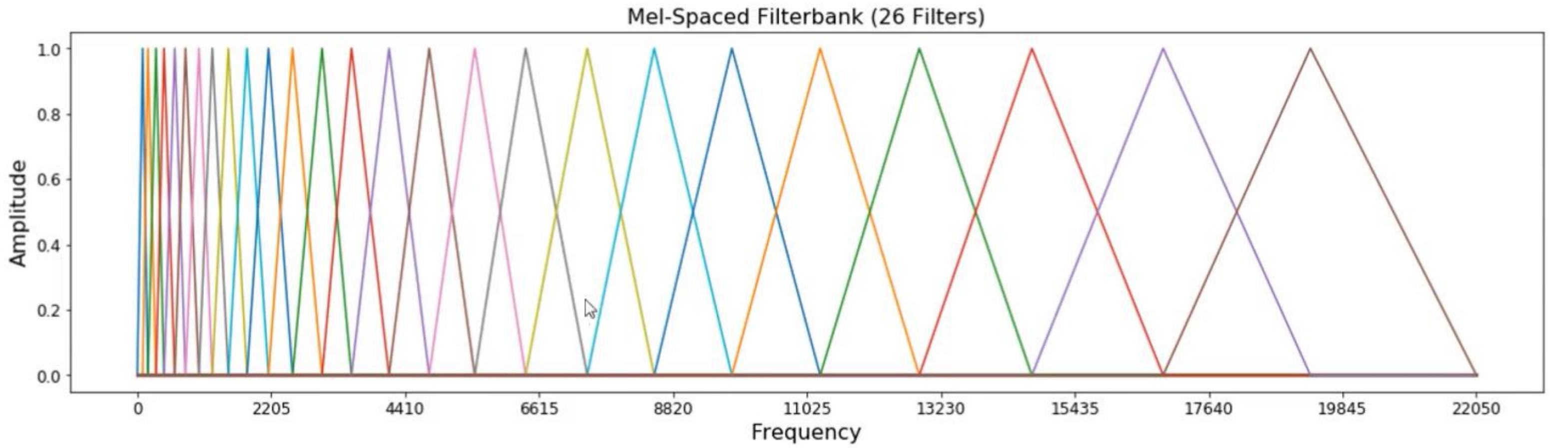
Mel-Scaling

↓

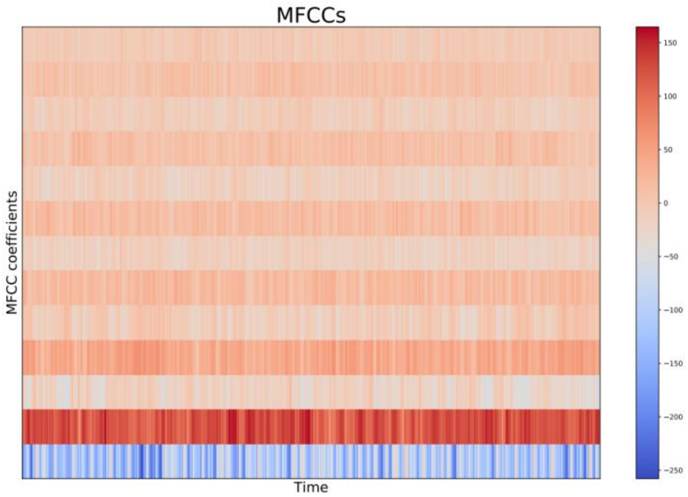
Discrete Cosine Transform

↓

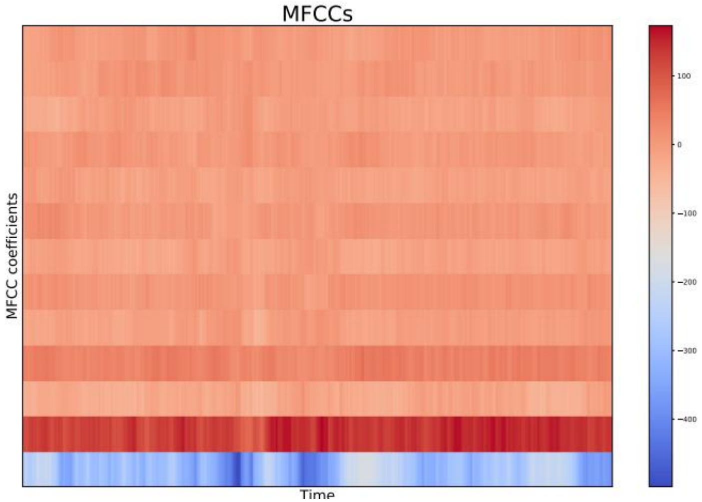
MFCC



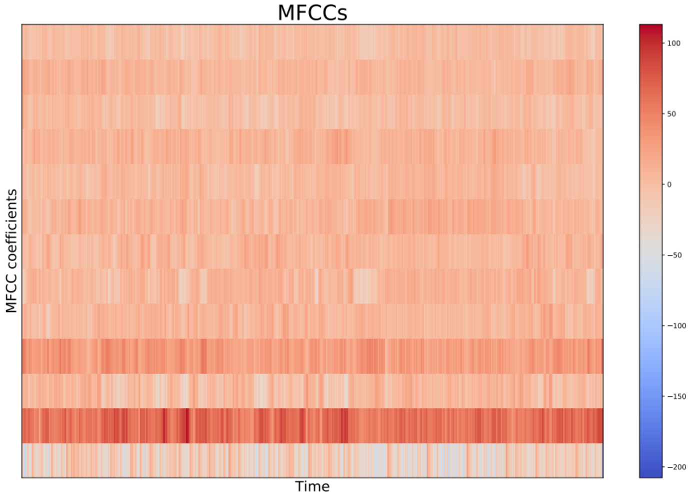
Example MFCCs



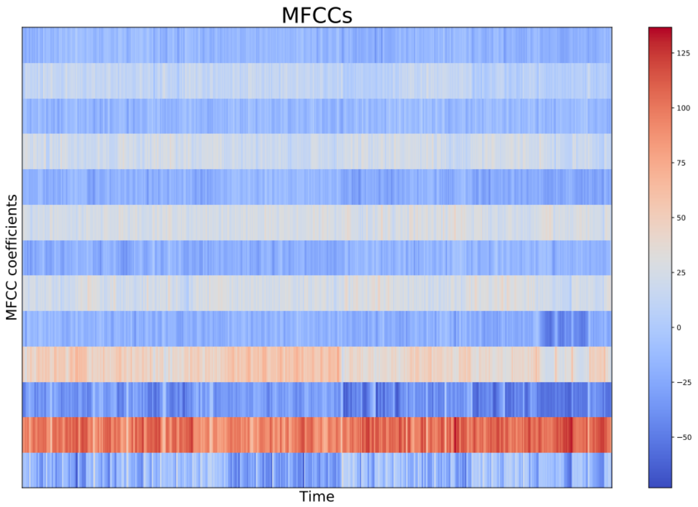
Blues



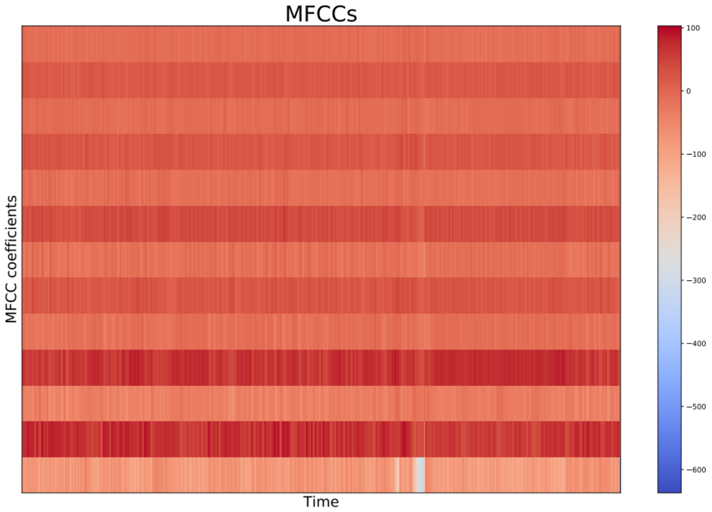
Classical



Country



Rock



Metal

Conventional Neural Network

- Used Keras.Sequential to build CNN
  - 3 x Convolution layer, max pooling, batch normalization
  - 1 Dense layer with 64 neurons
  - Dropout layer
  - Dense layer with 10 neurons

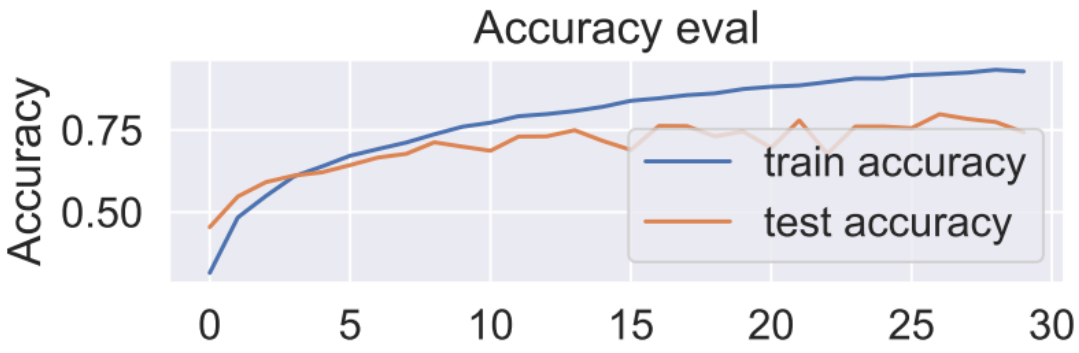
- 30 epochs used

- Takes roughly 30 minutes to train

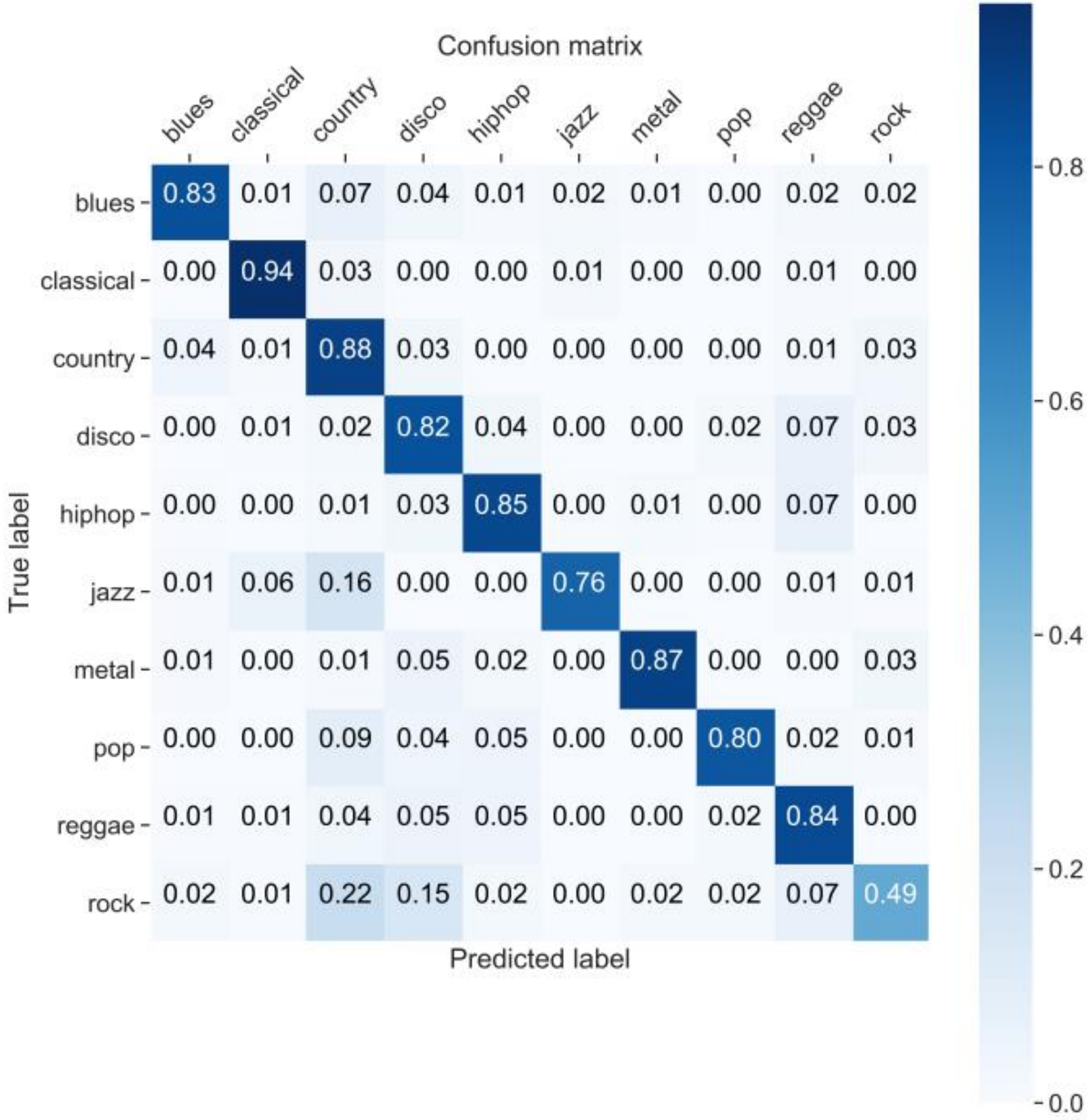
- Optimized using Keras.Tuner “RandomSearch”
  - number of convolution layers
  - number of filters in each layer

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 128, 11, 224)      2240
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 64, 6, 224)        0
_________________________________________________________________
batch_normalization_6 (Batch (None, 64, 6, 224)        896
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 62, 4, 224)        451808
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 31, 2, 224)        0
_________________________________________________________________
batch_normalization_7 (Batch (None, 31, 2, 224)        896
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 30, 1, 32)         28704
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 15, 1, 32)         0
_________________________________________________________________
batch_normalization_8 (Batch (None, 15, 1, 32)         128
_________________________________________________________________
flatten_2 (Flatten)          (None, 480)               0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                30784
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650
=================================================================
Total params: 516,106
Trainable params: 515,146
Non-trainable params: 960
_________________________________________________________________
```

Results



- Scores 80.8% on Test data
- Classical and metal are easiest to predict
- Rock is hardest, mostly mislabeled as country



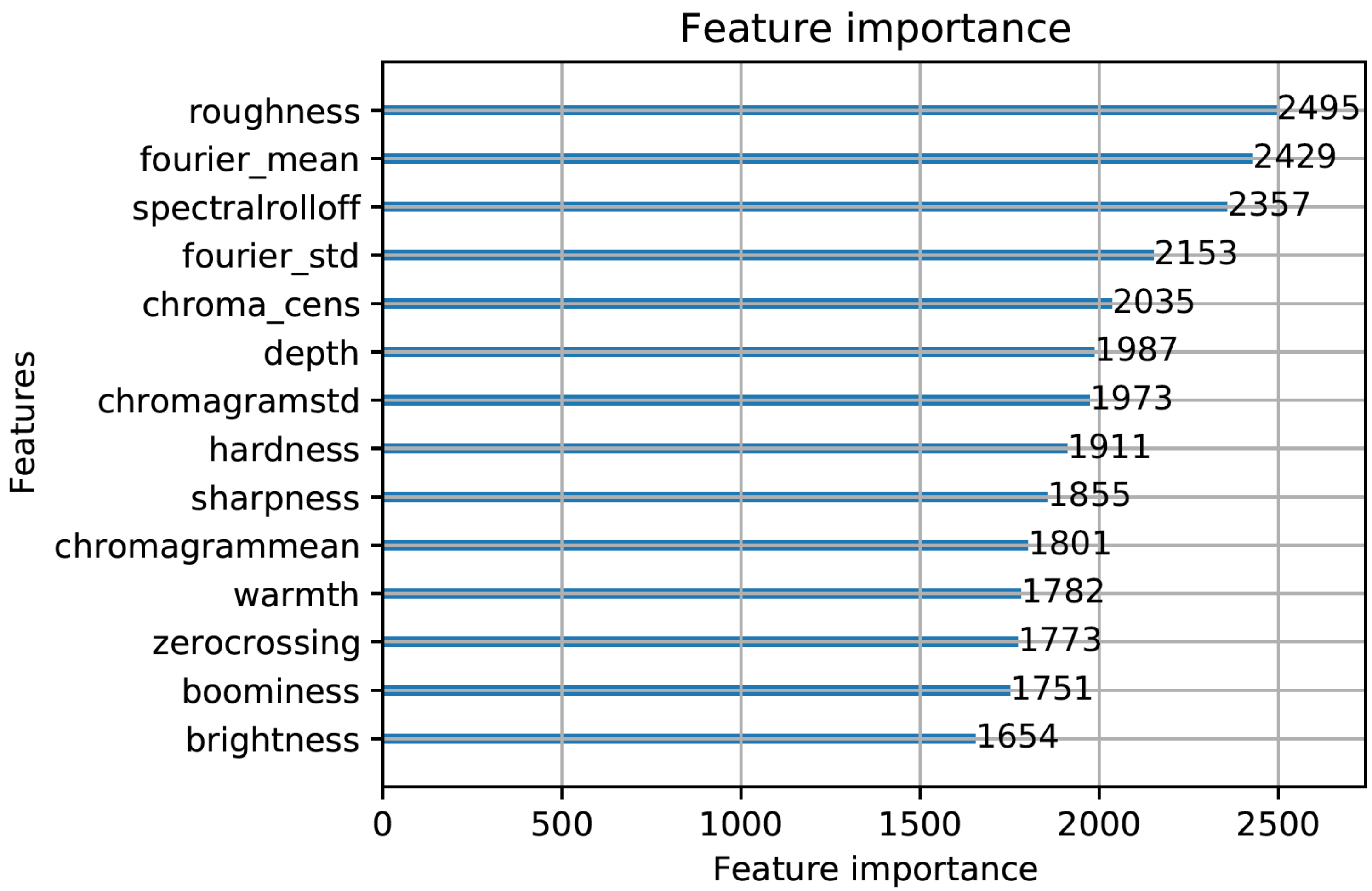
Outlook

- CNN is a good model for classifying music
- I don’t think it will get much better, since humans can’t always classify genres perfectly
- Would like to keep optimizing more hyperparameters in the CNN
- Try to extract instruments in all the tracks

Appendix

Feature importance of initial approach

The variables all contribute to the classification. Surprisingly, one of the most important variables is the mean of the fourier transform, done by hand.



Optimizing tree-based algorithm

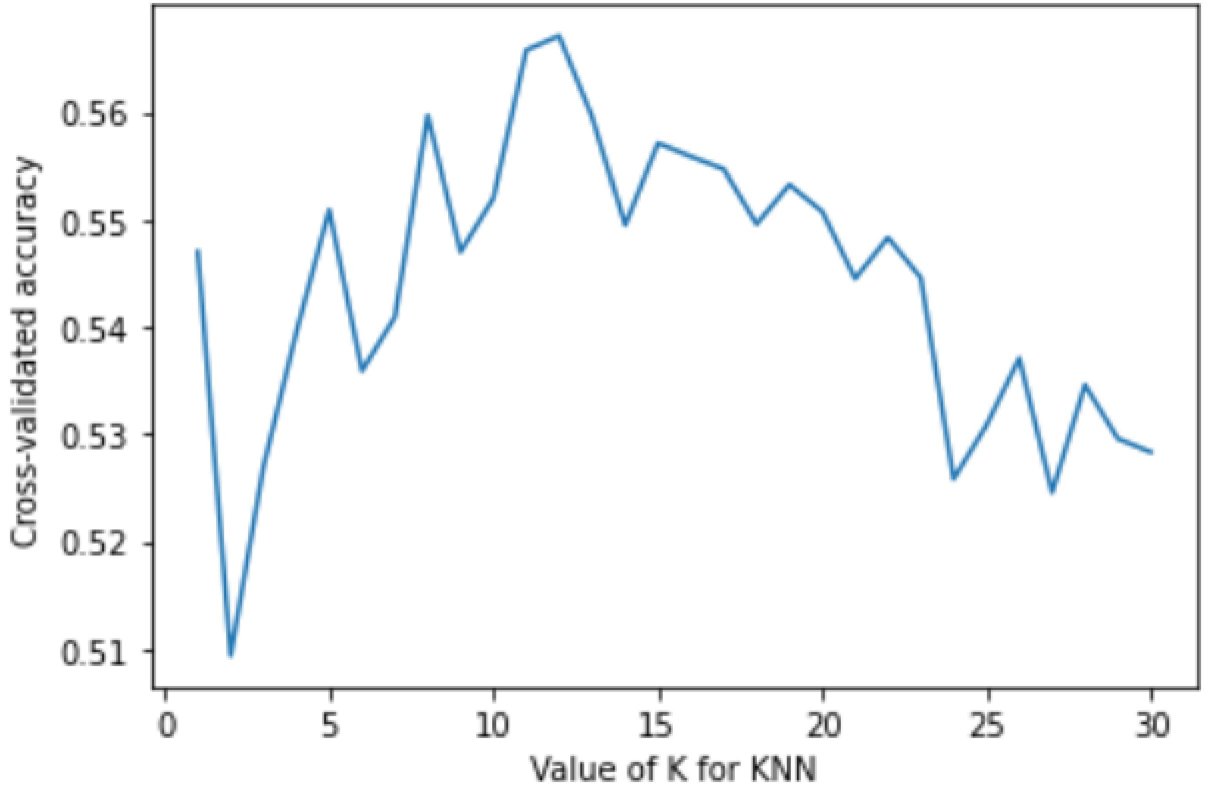
We use LightGBM for the initial attempts at efficient classification.
The optimal parameters we found to be:
n_epochs = 500
n_leafs = 10
max_depth = 10

This yields a logloss of 0.3 for the multi-classification algorithm, using LightGBM cross validation.

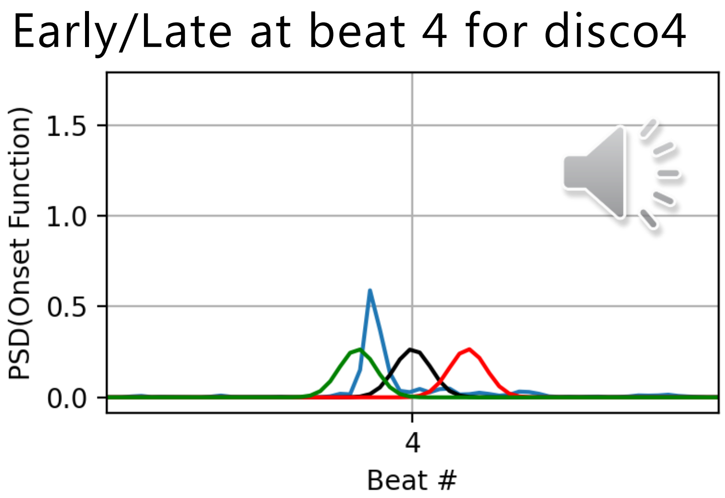
kNearestneighbor

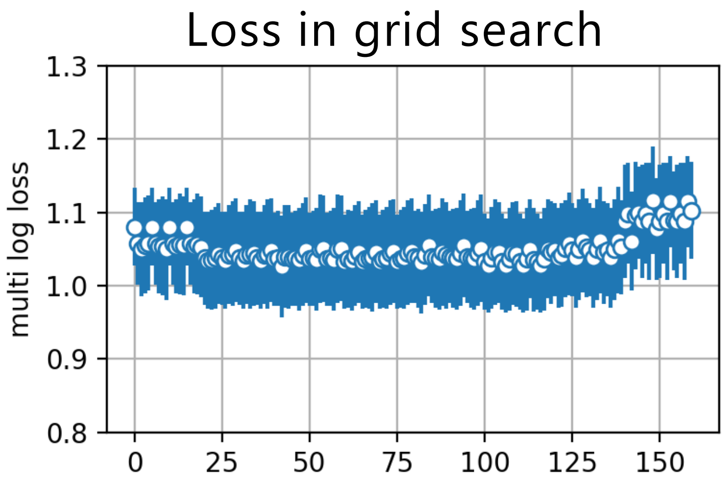
The kNearestneighbor is very straight-forward and easy to use. This is the approach that many have used on the data set.

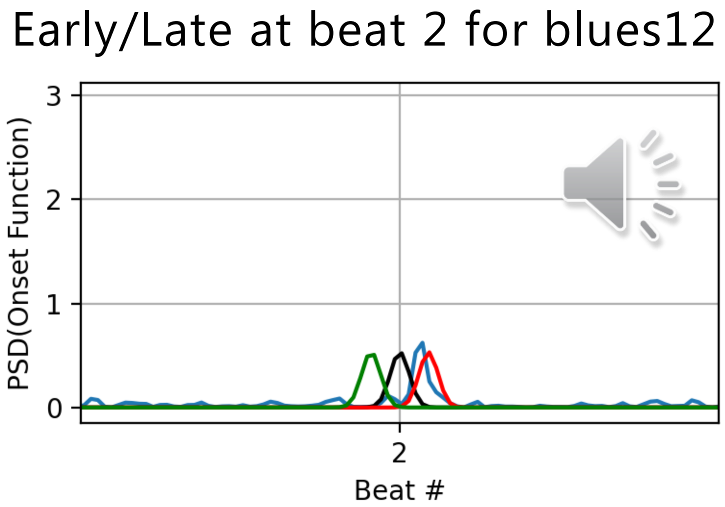
We use n_neighbors = 12, based on simple optimization.

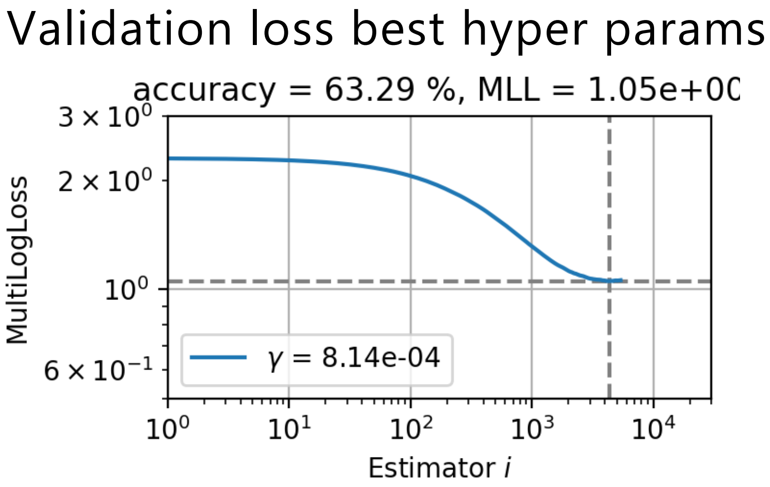


Extracted features, training

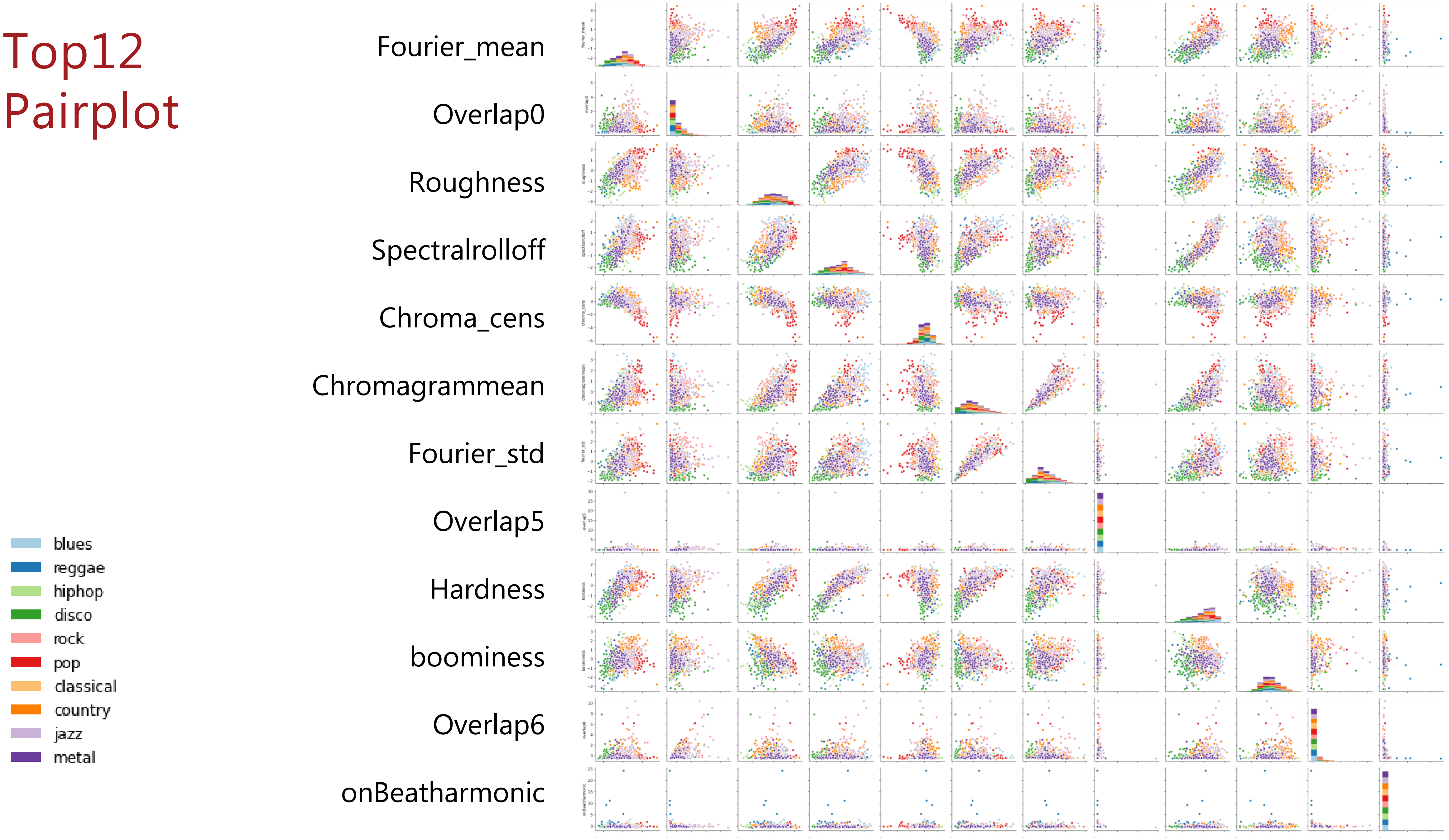








Top12 Pairplot



Preprocessing - NN

-Started with 100 samples per genre, then divided each by 5 to have more data
-Extracted MFCC's using the Librosa package
-Parameters: n_fft=2048 (window for fft in num. of samples), hop_length=512 (in num. Of samples)

```
# extract mfcc
mfcc = librosa.feature.mfcc(signal[start:finish], sample_rate, n_mfcc=num_mfcc,
mfcc = mfcc.T
```

-Saved all MFCC's, labels, and mapping to a .json file

```
# save MFCCs to json file
with open(json_path, "w") as fp:
    json.dump(data, fp, indent=4)
```

Tuning Neural Network

```
from kerastuner import RandomSearch
from kerastuner.engine.hyperparameters import HyperParameters
```

-Varied number of filters from 32 to 256 in steps of 32 for 2 Convolutional layers

```
tuner=RandomSearch(build_model,
                          objective='val_accuracy',
                          max_trials=3,
                          directory=LOG_DIR,project_name="MusicClass")
```

```
tuner.search(x=X_train,
             y=y_train,
             epochs=8,
             batch_size=64,
             validation_data=(X_test,y_test))
```

- Tried with 1-3 Convoultional layers, also with 2-3 Dense layers

Neural network details

Activation= ReLu
Max Pool window = 2 x (3,3), then (2,2)
Strides = (2,2)
Padding=same

Optimizer= Adam
Learning rate=0.0001
Loss = sparse_categorical_crossentropy
Metrics=accuracy

```
# build network topology
model = keras.Sequential()

# 1st conv layer
model.add(keras.layers.Conv2D(224, (3, 3), activation='relu', input_shape=input_shape))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# 2nd conv layer
model.add(keras.layers.Conv2D(224, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# 3rd conv layer
model.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
model.add(keras.layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# flatten output and feed it into dense layer
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(0.3))
#model.add(keras.layers.Dense(32, activation='relu'))
#model.add(keras.layers.Dropout(0.1))
# output layer
model.add(keras.layers.Dense(10, activation='softmax'))
```