Identifying Machine Gun Kelly vs. Eminem using Sound-Based Features

Eamonn Mailey

Basic Tools

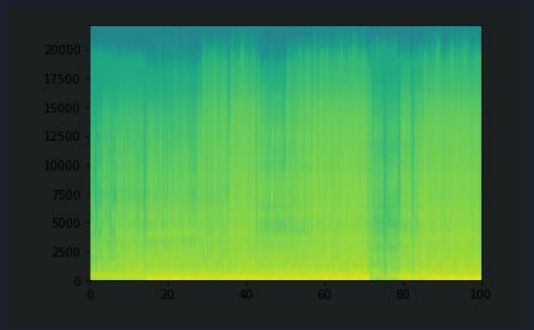
- Jupyter Notebook
- Pandas
- Sklearn
- Librosa
- Matplotlib
- Sidify
- Spotify

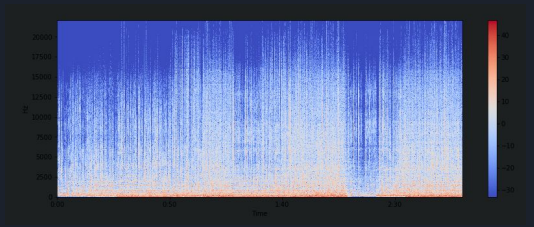
What data? Where data?

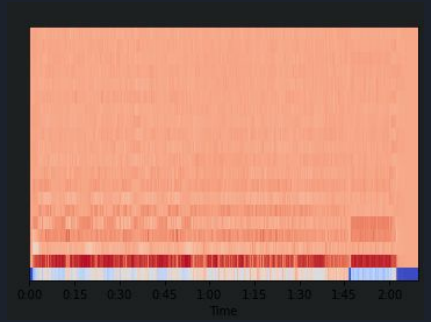
- No standard libraries
  - Time to make my own
- Spotify
  - Shoutouts to people who make playlists
- Sidify Music Converter
  - First 3 mins
- Now we have WAVs

Working With WAVs

- Librosa - audio and music processing
- What is FFT (Fast Fourier Transform)
  - Frequency Intensities
- Improving Upon FFT w/ MFCC
  - (Mel Frequency Cepstral Coefficients)
- Mel Frequency Cepstrum (MFC)
  - Encodes the power spectrum of sound
  - Fourier Transform of the logarithm of the signals spectrum
  - Successfully used in speech and speaker recognition







Cleaning + Working with MFCC data

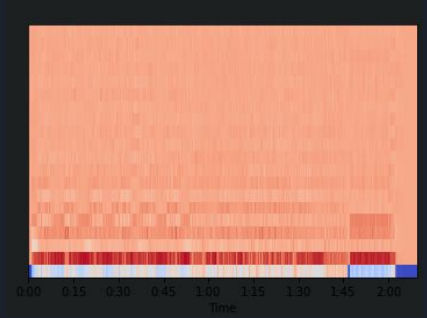
- Cache Data for time save
- Turns into 20 Features (ceps) per song
  - Taking data for 20 features for hundreds of thousands of frames for each song would be too much, so we average over time
  - I also cut most of the back end and the very beginning off of the songs when averaging, as some songs got cut off at 3 minutes due to the software I was using and also the beginning and ends of songs do not always represent the sound of the main portion

```
mfccData = librosa.feature.mfcc(data, sr=sr)
```

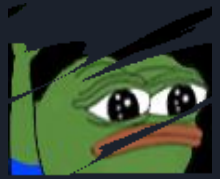
```
mfccData = np.load(MGKdir + "MFCC\\" + mfcc)
x = mfccData.T
frames, coefs = x.shape
mfccData = np.mean(x[int(frames*1/10):int(frames*6/10)], axis=0)
```

- Store in DataFrame, with Artist as 0 (MGK) or 1 (Eminem)
  -

12	13	14	15	16	17	18	19	Artist
7464	-0.804199	3.013386	-1.919305	1.993430	0.216227	0.703446	-3.328345	0
069	3.610677	3.991945	1.805106	3.064812	4.118328	2.636167	1.145486	0
2799	6.410483	0.033452	5.910886	-4.333644	7.261153	-3.836775	3.811460	0
707	4.389051	4.831897	2.433865	-0.751176	0.087663	0.649371	-1.234107	0
194	2.282683	2.395646	0.899051	1.334392	1.592976	4.875860	-2.244050	0
	...	...	...	...	...	...	...	...
729	4.226511	5.106968	5.868017	0.473829	3.552892	2.632421	1.910032	1
5142	-8.774842	2.989717	-1.696608	-7.083531	-1.045804	-2.761361	-4.646432	1
2063	-0.787569	-2.379001	-0.295823	0.474308	0.135065	-1.817821	-0.912429	1
394	9.722982	2.538850	4.165314	3.799022	-3.355212	0.485991	-3.564075	1
866	-1.890921	7.915220	-0.795118	3.071722	1.837076	2.121810	0.228534	1

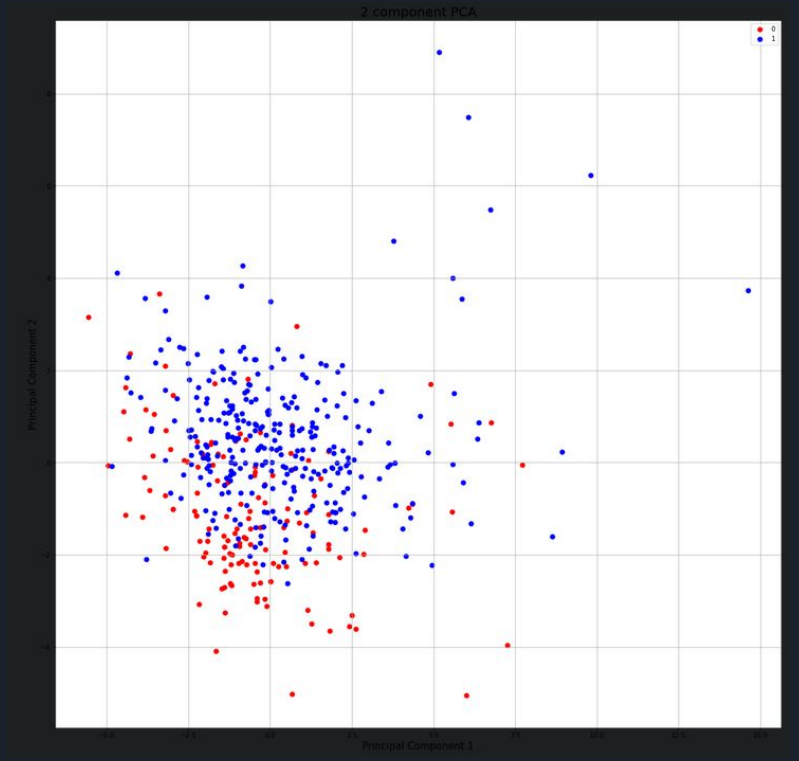


PCA Belly Flop



- Too much averaging
  - Averaging the MFCCs over time along with using StandardScaler to standardize inputs along with transforming the dimensionality to 2D took away most of the distinction →
- Goodbye to those hours

- Red = MGK
- Blue = Eminem



New Plan: Logistic Regression

- Luckily the data I had from trying PCA was very usable (before scaling)
- Binary Output Variable
- Many features (input variables)
- Used a lot in music recommendation and speech recognition software
- Seperate Inputs and Output:

```
x = totalList.loc[:,range(20)].values

y = totalList.loc[:,['Artist']].values
y = y.ravel()
```

- Train Data:

```
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=0)
```

Finding Accuracy: Correct Prediction

- Accuracy:
  - 88.32%

```
In [604]: print("Accuracy: " + str(100 * metrics.accuracy_score(y_test, y_pred)) + "%")

          Accuracy: 88.32116788321169%
```

(Hey that's pretty good!)

- That's just one seed though. Let's test a hundred random seeds:

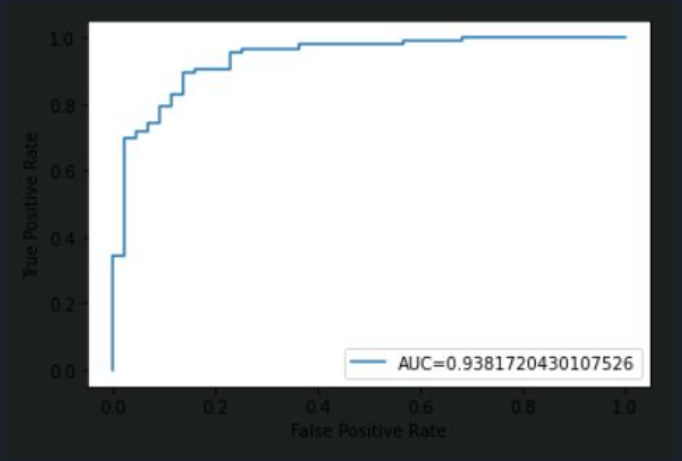
```
In [605]: accuracy = 0
          for i in range(100):
              X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=np.random)
              log_regression = LogisticRegression(max_iter=10000)
              log_regression.fit(X_train, y_train)
              y_pred = log_regression.predict(X_test)
              accuracy += 100 * metrics.accuracy_score(y_test, y_pred)
          accuracy /= 100
          accuracy

          83.6277372262774
```

- There is a much more realistic estimate of ~83.63%

Finding Accuracy: ROC Curve (Receiver Operating Characteristic)

- The ROC curve is a great measure of overall accuracy too
  - The higher the AUC (Area Under the Curve) is, the more accurate the model is at predicting outcomes
  - It displays the percentage of true positives predicted by the model as the prediction probability cutoff is lowered from 1-0:
- ROC Graph of Random Test and Training set:
  - AUC = .93817 (very good!!)



Finding Accuracy: ROC Curve (Receiver Operating Characteristic)

- This isn't entirely accurate either, so I averaged a hundred random seeds here as well:

```
In [615]:
avg = 0
for i in range(100):
    X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=np.random)
    y_pred_proba = log_regression.predict_proba(X_test)[::,1]
    fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
    auc = metrics.roc_auc_score(y_test, y_pred_proba)
    avg += auc

avg /= 100
avg

0.9246817579420434
```

- This is a more realistic estimate of .92468
  - This is very close to 1, which indicated that the model is great at predicting the correct outcome

Can’t tell if it’s MGK or Em?

Don’t flip a coin,
USE MY ALGO!

Thanks for Watching :)