Machine Learning and Spectrograms

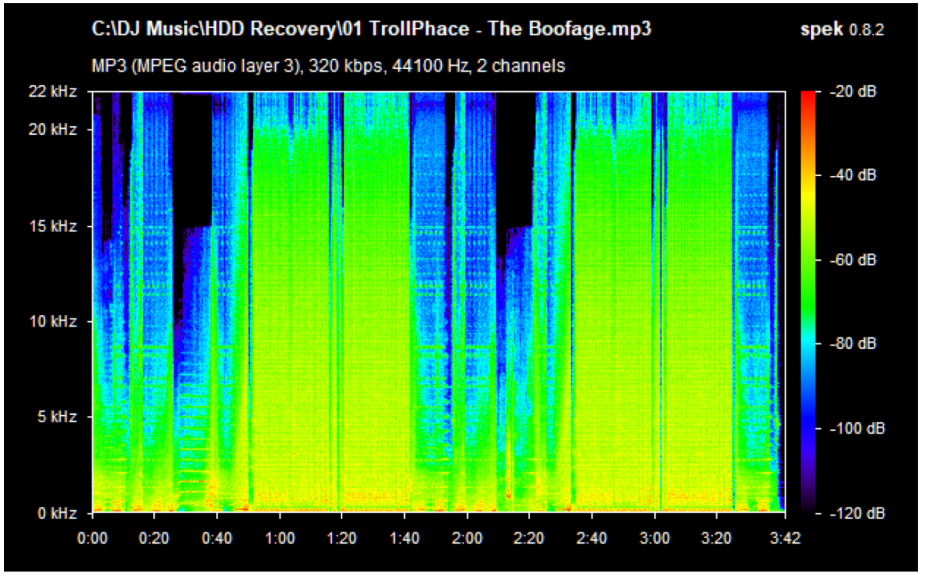
Tom Slesinger and Nathan Duffy

Overview

1. Research Question
2. Getting Data
3. Labelling Data
4. ML
    a. Feature Engineering (x2)
    b. Slightly Better ML
5. Website
6. Findings
7. Limitations
8. Next Steps

Research Question

Is it possible to classify the quality of a song based off of its audio-visual spectrogram?



Getting the Data

We wrote a Python script that used subprocess and pyautogui to streamline the creating of spectrograms.

```
#Loop through all files in directory
for root, dir, files, in os.walk(filepath):
    for file in files:
        #Find File Path of Song (need to not hard code)
        song_filepath =  filepath +'\\' + str(file)
        print('Processing: ', song_filepath)

        #Call Spek with that filepath

        subprocess.Popen([r'C:\Program Files (x86)\Spek\Spek.exe', song_filepath])

        #subprocess.Popen([r'C:\Users\spitf_000\Downloads\spek-0.8.2\Spek\spek.exe', fp])


        #Implementing adaptive wait time based on file size
        file_size = os.stat(fp).st_size
        print(file_size)
        wait_time = (file_size/1000000)/2
        time.sleep(wait_time)

        #Setting SAFE pyautogui
        #Can tune once the rest is stable
        pyautogui.FAIL_SAFE = True
        pyautogui.PAUSE = 0.5

        #Key Presses to Save each file
        pyautogui.hotkey('ctrl', 's')
        pyautogui.press('enter')
        pyautogui.hotkey('alt', 'f4')
```

Labelling Data

We crowdsourced labelling data by having our class label roughly 100 pictures and labelling the rest of the 800 by ourselves.

Machine Learning

The machine learning portion of this project involved a few steps. We first had to get the data from the google sheet into a pandas dataframe.

```
In [2]:
1 drive_and_label = pd.read_excel('Spectogram_Data_v1.xlsx')
2 drive_and_label.head()
```

Out[2]:

	Spectogram	Meta_Label	New_Label
0	https://drive.google.com/uc?export=download&id...	320kpbs	192 kbps (LAME)
1	https://drive.google.com/uc?export=download&id...	320 kbps	320 kbps (LAME)
2	https://drive.google.com/uc?export=download&id...	320 kbps	128 kbps (CBR)
3	https://drive.google.com/uc?export=download&id...	128 kbps	128 kpbs (AAC)
4	https://drive.google.com/uc?export=download&id...	320 kbps	128 kbps (LAME)

Machine Learning

In [ ]:

```
d = {}
#Loop through each Google Drive link
for idx, row in drive_and_label.iterrows():

    #Creating a 2-D Array with just pixel values
    #print ('Finding feature array...')
    im = imread(row['Spectogram'])
    im_crop = crop_center(im, 505, 295)
    features = im_crop.flatten()
    #print('... array found.')

    #Adding features to dictionary
    d[idx] = features
```

In [26]:

```
df = pd.DataFrame.from_dict(d)
df = df.T
df.insert(0, 'label', drive_and_label['New_Label'])
df.tail()

df.to_csv('SCORE.csv', sep = ',')
```

Machine Learning

The sci-kit-image library has a function called imread. This encodes each pixel into a list with three values [R, G, B]. We can flatten this 3-D matrix to get a list of RGB values that correspond to a label.

This have us a sparse matrix of (1000, 446926) but we could reduce the dataset to (1000, ~8000). This is still quite a lot of data.

K-Fold Cross Validation

We were able to use K-fold cross validation to compare many different algorithms but... didn't get anything higher than the dummy.

In [13]:

```
# prepare configuration for cross validation test harness
seed = 123
# prepare models
models = []
models.append(('LR', LogisticRegression()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('TREE', DecisionTreeClassifier()))
models.append(('SVM', SVC()))

# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scori
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

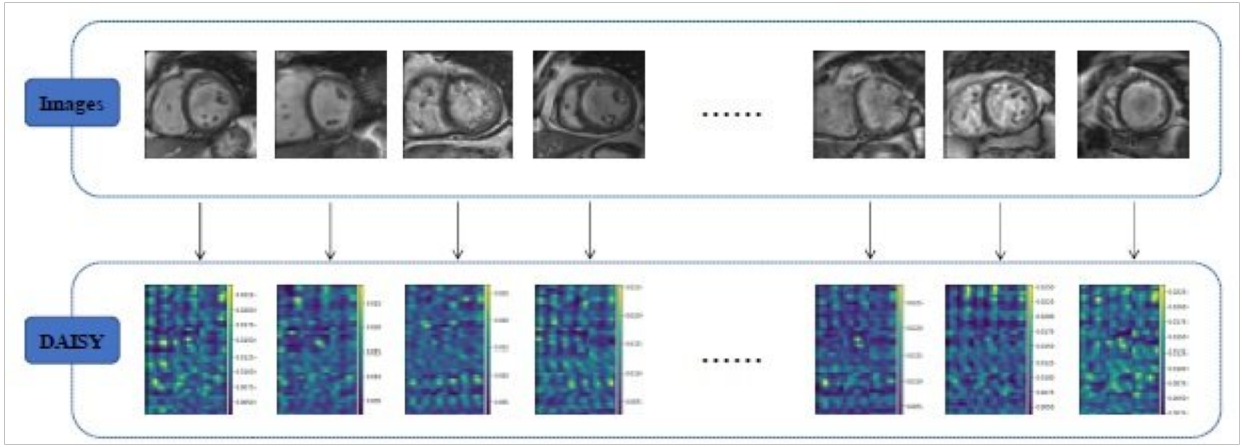
Attempting to improve accuracy

Since the decision tree was roughly higher we decided to to change hyperparameters of a decision tree (depth and max leafs) but didn’t get any improvements.

This means that the data in its current format was unlearnable.

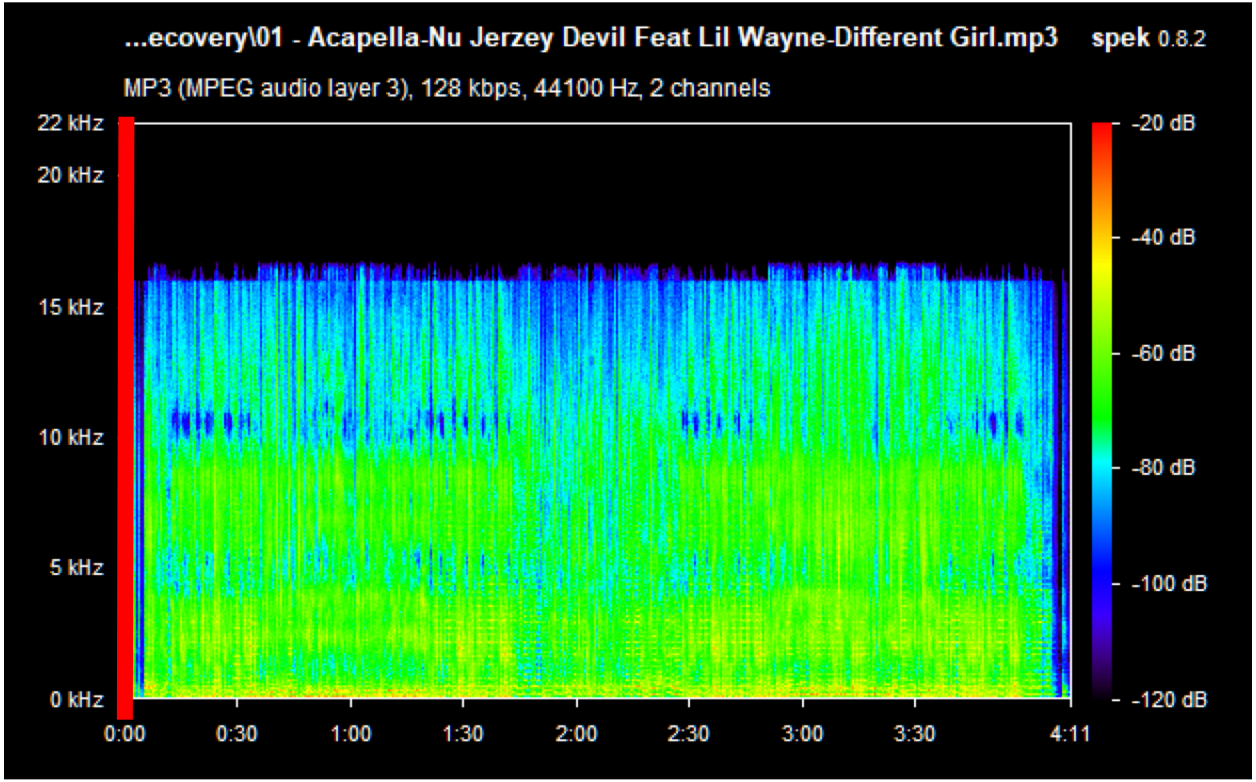
DAISY features?

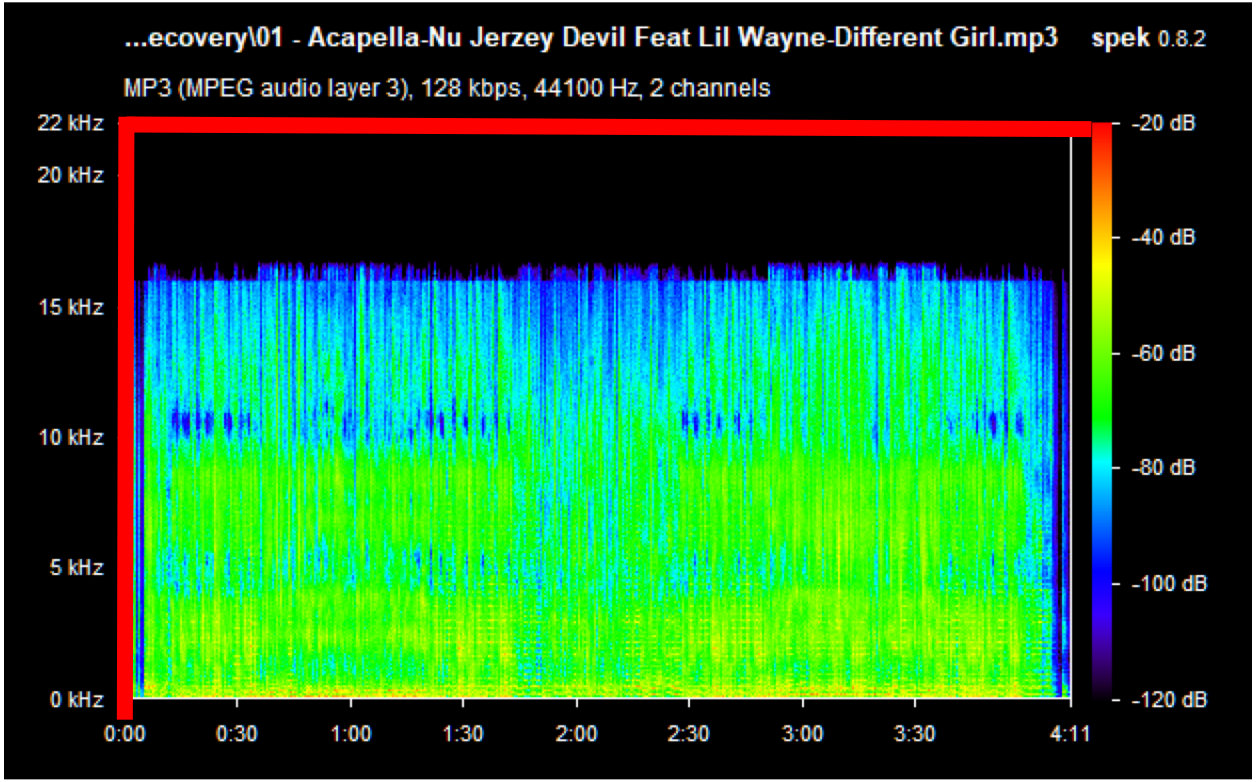
We thought we could use daisy features which extracts descriptors from an image but it created roughly 1 million features per image which gave memory errors :(

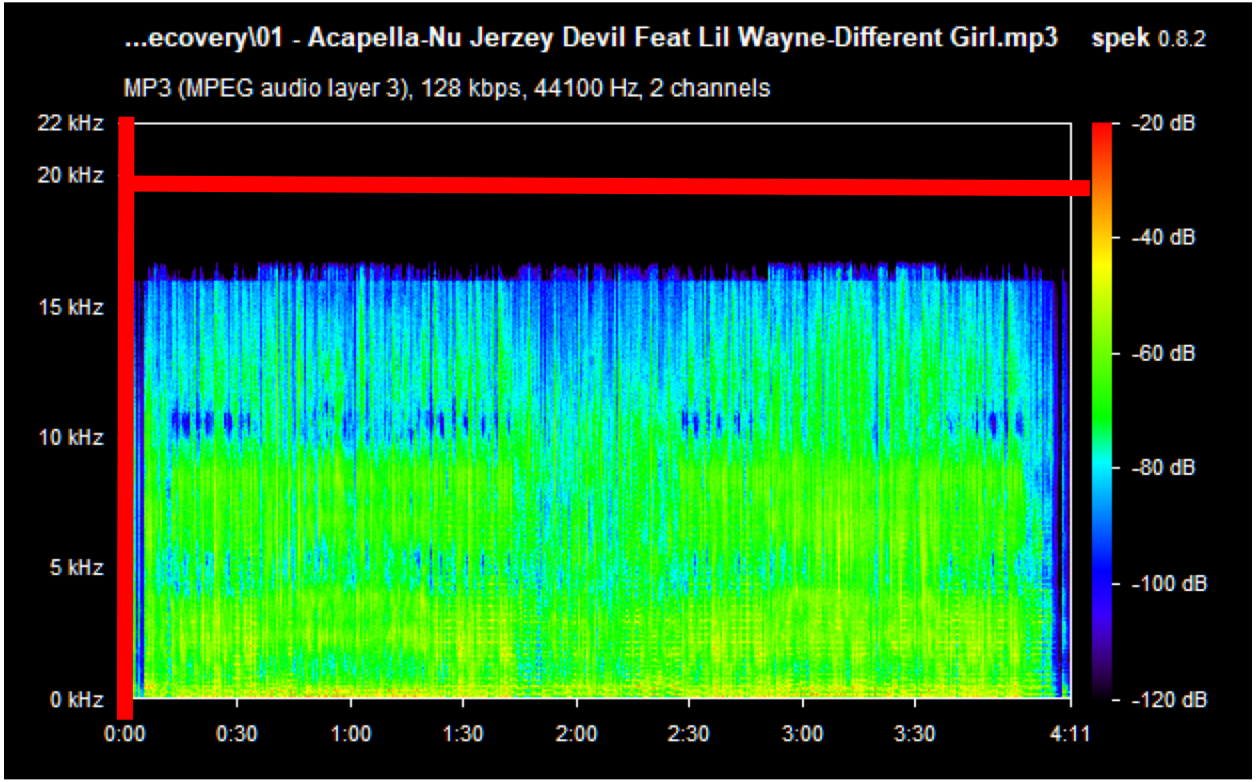


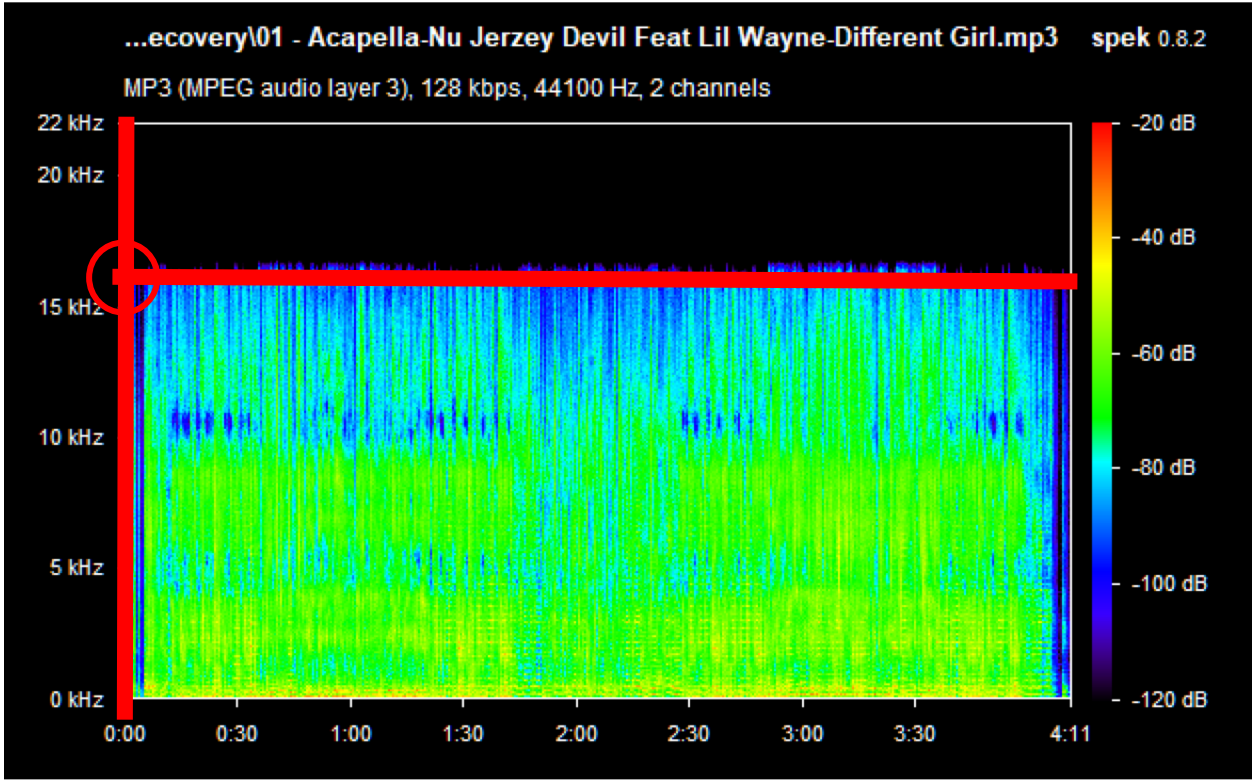
Feature Engineering pt. 2

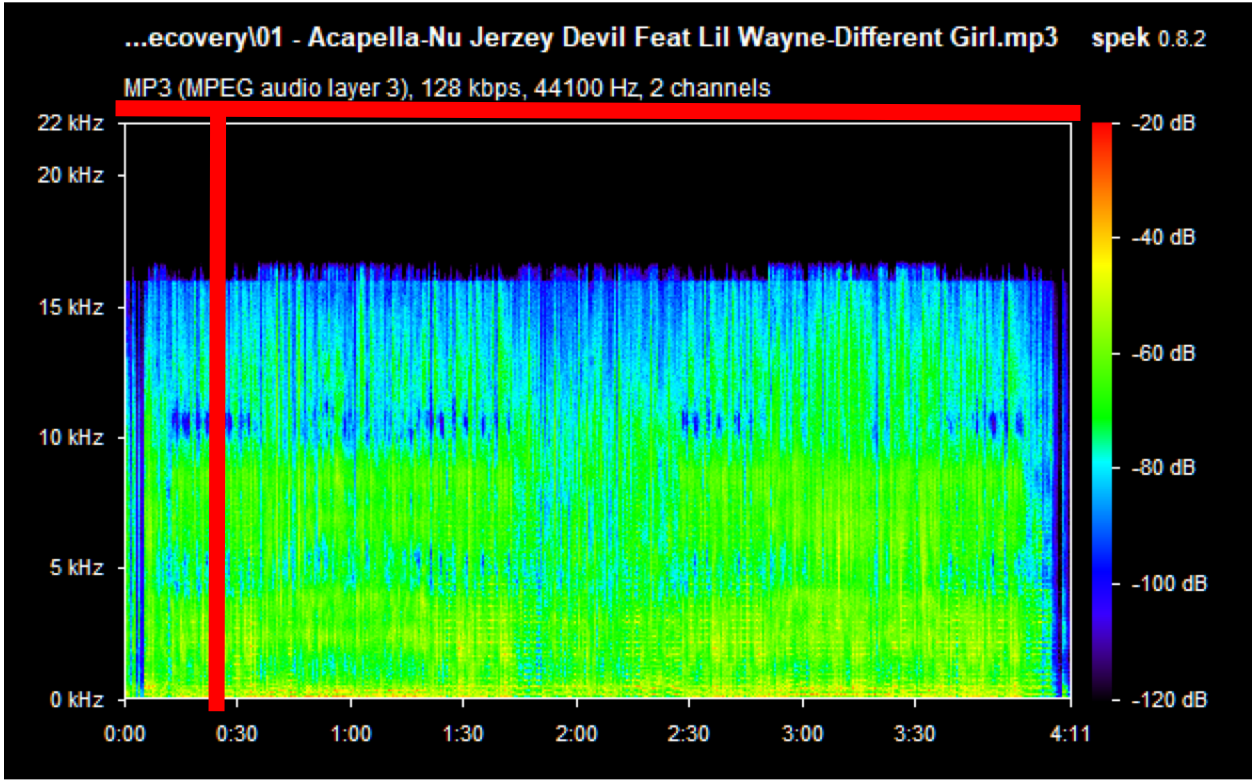
After meeting with professor Paul, he had a great idea for feature engineering. Instead of using simple RGB values we could write some code to find the shelf by seeing where the black pixels end.


...ecovery\01 - Acapella-Nu Jerzey Devil Feat Lil Wayne-Different Girl.mp3
spek 0.8.2
MP3 (MPEG audio layer 3), 128 kbps, 44100 Hz, 2 channels
22 kHz
20 kHz
15 kHz
10 kHz
5 kHz
0 kHz
-20 dB
-40 dB
-60 dB
-80 dB
-100 dB
-120 dB
0:00
0:30
1:00
1:30
2:00
2:30
3:00
3:30
4:11


...ecovery\01 - Acapella-Nu Jerzey Devil Feat Lil Wayne-Different Girl.mp3
spek 0.8.2
MP3 (MPEG audio layer 3), 128 kbps, 44100 Hz, 2 channels
22 kHz
20 kHz
15 kHz
10 kHz
5 kHz
0 kHz
-20 dB
-40 dB
-60 dB
-80 dB
-100 dB
-120 dB
0:00
0:30
1:00
1:30
2:00
2:30
3:00
3:30
4:11


...ecovery\01 - Acapella-Nu Jerzey Devil Feat Lil Wayne-Different Girl.mp3
spek 0.8.2
MP3 (MPEG audio layer 3), 128 kbps, 44100 Hz, 2 channels
22 kHz
20 kHz
15 kHz
10 kHz
5 kHz
0 kHz
-20 dB
-40 dB
-60 dB
-80 dB
-100 dB
-120 dB
0:00
0:30
1:00
1:30
2:00
2:30
3:00
3:30
4:11


...ecovery\01 - Acapella-Nu Jerzey Devil Feat Lil Wayne-Different Girl.mp3
spek 0.8.2
MP3 (MPEG audio layer 3), 128 kbps, 44100 Hz, 2 channels
22 kHz
20 kHz
15 kHz
10 kHz
5 kHz
0 kHz
-20 dB
-40 dB
-60 dB
-80 dB
-100 dB
-120 dB
0:00
0:30
1:00
1:30
2:00
2:30
3:00
3:30
4:11


...ecovery\01 - Acapella-Nu Jerzey Devil Feat Lil Wayne-Different Girl.mp3
spek 0.8.2
MP3 (MPEG audio layer 3), 128 kbps, 44100 Hz, 2 channels
22 kHz
20 kHz
15 kHz
10 kHz
5 kHz
0 kHz
-20 dB
-40 dB
-60 dB
-80 dB
-100 dB
-120 dB
0:00
0:30
1:00
1:30
2:00
2:30
3:00
3:30
4:11

Feature Engineering Pt. 2

This would allow us to get a value for the shelf of spectrogram and hopefully improve accuracy.

In [6]:

```
d = {}

for idx, row in drive_and_label.iterrows():
    #Find picture from GD link
    im = imread(row['Spectogram'], as_grey = True)

    #Crop Picture to get rid of top section
    #50 pixels is the standard header
    im = im[49:]

    #Find Bounds of image (to give baseline)
    y_bound, x_bound = im.shape

    #Looping across the x-axis (predefined resolution)
    score = []
    temp = 0
    for ii in range(0, y_bound):
        #If the sum of the pixels across the row is
        if sum(im[ii]) != 0:
            temp += 1
            score.append(temp)
        #If the sum is 0 we give a bad score
        elif sum(im[ii]) == 0:
            temp -= 1
            score.append(temp)

    d[idx] = score
```

Out[25]:

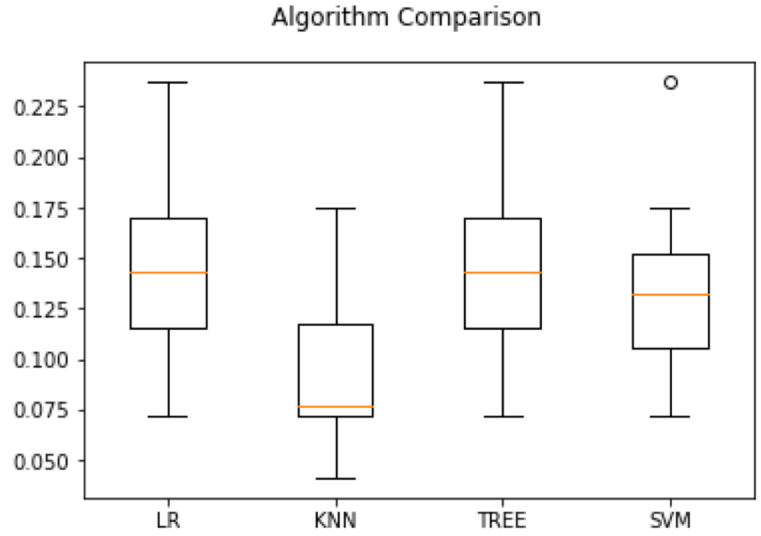
	0	1	2	3	4	5	6	7	8	9	...	330	331	332	333	334	335	336	337	338	33
0	192 kbps (LAME)	-1	-2	-3	-4	-5	-6	-7	-6	-5	...	278	277	276	275	274	273	272	271	270	26
1	320 kbps (LAME)	-1	-2	-3	-4	-5	-6	-7	-6	-5	...	278	277	276	275	274	273	272	271	270	26
2	128 kbps (CBR)	-1	-2	-3	-4	-5	-6	-7	-6	-5	...	278	277	276	275	274	273	272	271	270	26
3	128 kpbs (AAC)	-1	-2	-3	-4	-5	-6	-7	-6	-5	...	278	277	276	275	274	273	272	271	270	26
4	128 kbps (LAME)	-1	-2	-3	-4	-5	-6	-7	-6	-5	...	278	277	276	275	274	273	272	271	270	26

5 rows × 340 columns

K-Fold Cross Validation (Again)

Dummy accuracy was the same with RGB features.

```
LR: 0.145971 (0.045442)
KNN: 0.093467 (0.037930)
TREE: 0.145971 (0.045442)
SVM: 0.136693 (0.043746)
```



New features help?

The new features helped by about 2-3 percent on average. Which still means that our algorithm is only slightly better than guessing.

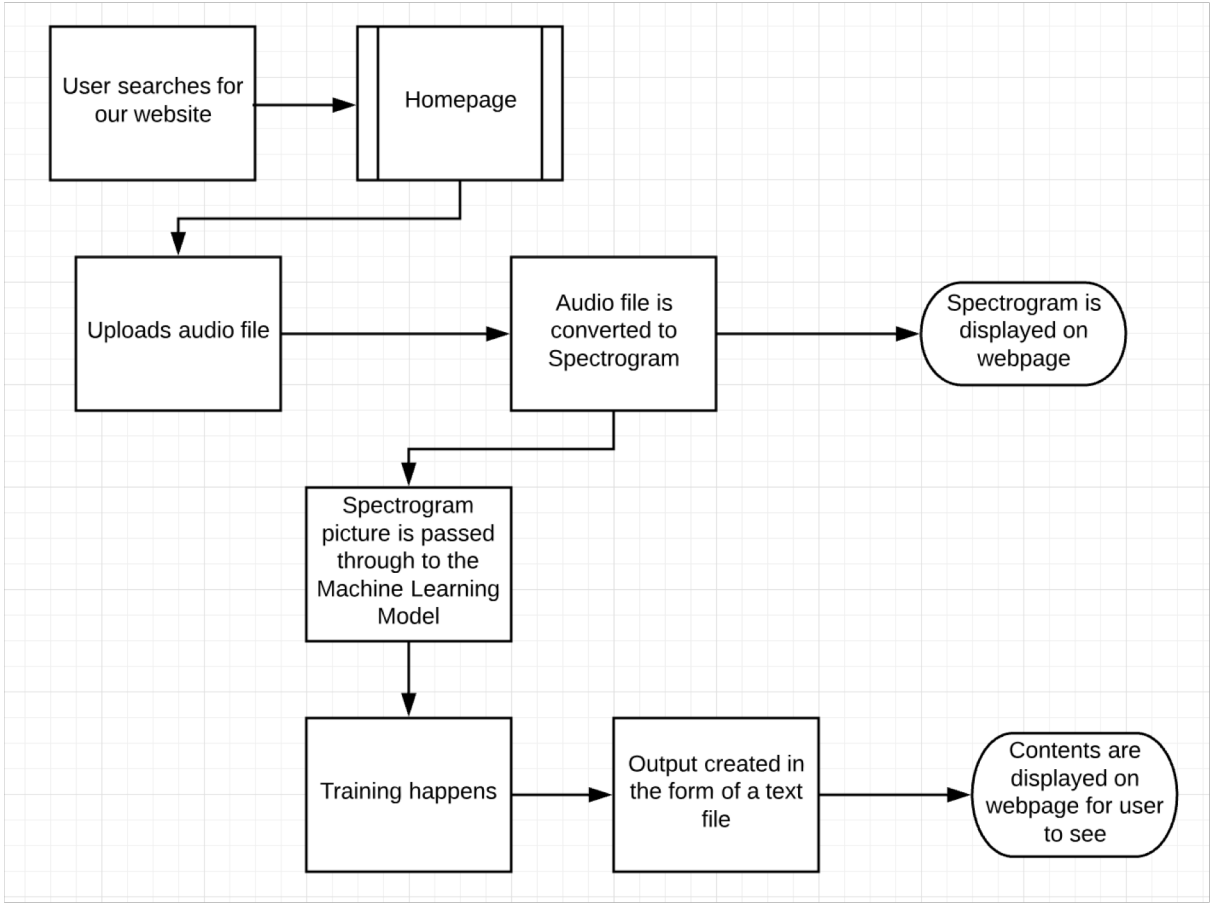
ML Findings

We found that, with our features, it is not possible to classify with great certainty. This accuracy can be improved

Implementing a Website

- After creating our initial machine learning model, it only makes sense to have this publically available online.
- Musicians have a need to verify song quality, and our model will help them along that process.
- This also allows us to collect and train new instances of song data, which will hopefully increase our accuracy in the future.

WorkFlow

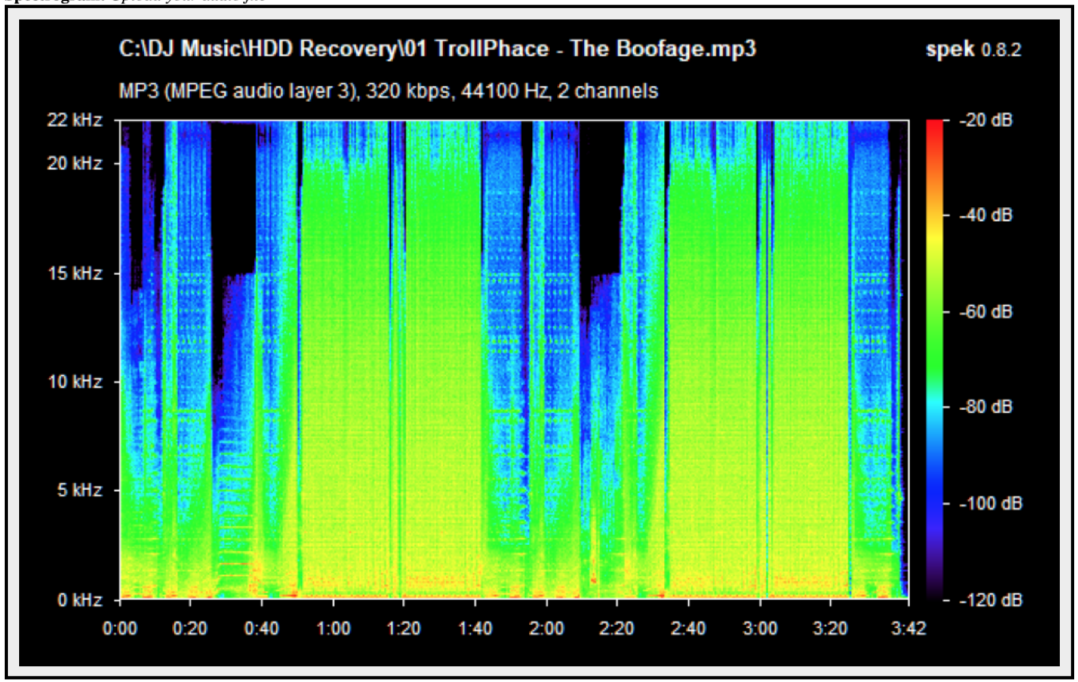

User searches for our website
Homepage
Uploads audio file
Audio file is converted to Spectrogram
Spectrogram is displayed on webpage
Spectrogram picture is passed through to the Machine Learning Model
Training happens
Output created in the form of a text file
Contents are displayed on webpage for user to see

Prototype

Audio Verification

Choose File **output_file.jpeg**

Spectrogram: *Upload your audio file*



Output:

Process

- Javascript -->
- Child Nodes: .execfile() -->
- Ajax: client side process to create asynchronous web applications. -->
- Flask: module run through python which eliminates the need to communicate across programming languages. (much simpler)

Flask

```
#Imports
from flask import Flask, request, render_template
import pickle
import numpy as np
import ML as pkg


app = Flask(__name__)

@app.route("/")
def home():
    return render_template('home.html')

@app.route("/",methods=['POST','GET'])
def get_result():
    if request.method=='POST':
        result = request.form
        return(result)



if __name__ == '__main__':
    app.debug = True
    app.run()
```

Python Scripts

We converted the Jupyter Notebook files into python scripts to allow the website to communicate with the machine learning algorithm.

The script takes a filepath of the spectrogram and returns a .txt file with the output and the confidence.

```python
#Imports
import pandas as pd
import numpy as np
from skimage.io import imread
import skimage
import matplotlib
import numpy as np
import matplotlib.pyplot as plt


from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
scalar = StandardScaler()

from sklearn.tree import DecisionTreeClassifier




#Function takes a filepath of an image
#converts to features for ML gives an overall idea of how much black is in the pic
def spek_to_feature(filepath):

    im = imread(filepath, as_gray = True)

    #Crop Picture to get rid of top section
    #50 pixels is the standard header
    im = im[49:]

    #Find Bounds of image (to give baseline)
    y_bound, x_bound = im.shape

    #Looping across the x-axis (predefined resolution)
    score = []
    temp = 0

    for ii in range(0, y_bound):
        #If the sum of the pixels across the row is
        if sum(im[ii]) != 0:
            temp += 1
            score.append(temp)
        #If the sum is 0 we give a bad score
        elif sum(im[ii]) == 0:
            temp -= 1
            score.append(temp)

    return np.asarray(score)
```

```
def main():

    print('Please input the filepath of the spectogram.')
    #filepath = input()

    #Finding Features of new Spectogram

    features = spek_to_feature(str(filepath))


    #Training the Algorithm
    df = pd.read_csv('SCORE.csv', header = None)

    #Finding and converting raw data
    Y = df.iloc[:, 0].values.astype(str)
    X = df.iloc[:, 1:].values.astype(float)
    scalar.fit(X)
    X = scalar.transform(X)

    #Finding and converting test data
    Y_test = df.iloc[800:, 0].values
    X_test = df.iloc[800:, 1:].values.astype(float)

    scalar.fit(X_test)
    X_test = scalar.transform(X_test)


    classifier = DecisionTreeClassifier(random_state = 123)

    classifier.fit(X,Y)

    #Outputing the prediction and probability
    output = str(classifier.predict([features]))
    proba = np.amax(classifier.predict_proba([features]))

    with open('output.txt', 'w') as f:
        print(output, file = f)
        print(proba, file = f)
```

The script isn’t perfect

A big problem with this script is that we are retraining and fitting the algorithm each time you want to classify a song.

This could be solved by using pickle to handle python objects and keep the model trained and just call .fit

Limitations

Some of the limitations of this study are that some pictures could be labeled wrong, and that there’s not enough training data.

Future Steps

- Once the website is up and running, we would ideally like to implement a database to house the data collected on these songs.
- Once this is created we can retrain the algorithm every roughly 10-20 songs with more data. (This will probably still need human supervision for a while)
- Keep improving accuracy