Pitch Class Detection using Machine Learning

By: Sean Dvir & Misha Levitin

Introduction:

This project will address musical pitch detection using different machine learning algorithms.

The musical note identification problem, or, equivalently, the pitch detection problem can be stated as follows:

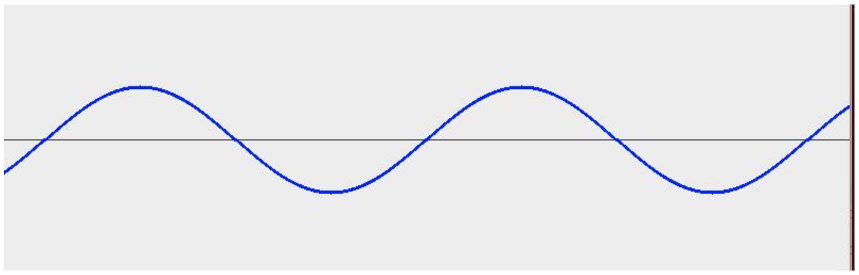
“ given an audio file, fragmented in a certain number of time frames, classify each such time frame in one of several classes of pitches.
As each pitch uniquely corresponds to one musical note, following the classification process, the musical note for each time frame of the piece will be known and consequently the musical score will be completely determined.“

In this project we will attempt to build a model which will be able to fit monophonic audio samples to the pitch classes they belong to.

We will address the problem as a classification problem, where pitch can belong to one of the 12 pitch classes - C, Db, D, Eb, E, F, Gb, G, Ab, A, Bb, B, or in solfege - Do, Re, Mi, Fa, Sol, La, Ti where Re, Mi, Sol, La, Ti has flats.

We will attempt to find the best way of finding F0 - The fundamental frequency.

The fundamental frequency is the base frequency of the waves of sound produced by an instrument.
In layman's terms: we know that audio is actually waves produced. In math and physics we tend to address only simple waves and treat them individually, sine, cosine, etc.



But in reality, a voice that is produced by any instrument is made up of a ton of different kinds of waves:

Not only that, it contains *partials*, which are the frequencies of the fundamental frequency multiplied by some constant.

So to find the real pitch of a sound we need to find a way to detect F0.

Review:

We chose to use supervised learning techniques such as Support Vector Machine, Stochastic Gradient Descent Classifier and K-Nearest Neighbours algorithms for the classification of audio samples into musical notes.

Supervised learning was chosen since our dataset is comprised of labeled data - audio files where each file is named with its corresponding pitch class.

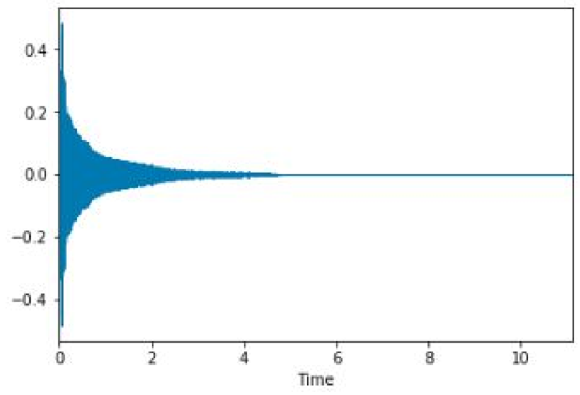
We also took inspiration from this paper [1].

Feature extraction:

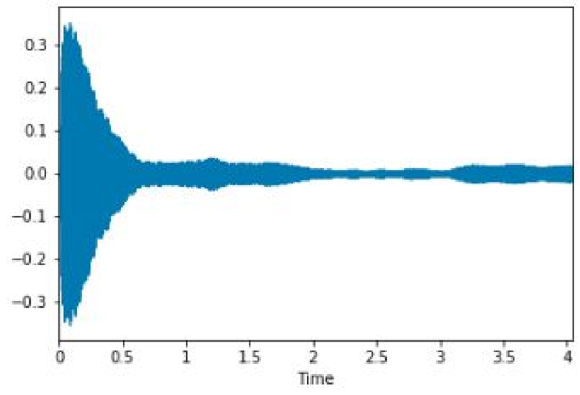
We started experimenting with different audio samples using the Librosa python library, which offers advanced methods for music information retrieval.

Plotting of audio files, the upper one represents the note A4 played on a acoustic guitar, and the other one on a piano.

```
In [47]: y_guitar, sr_guitar = librosa.load('A4_guitar.mp3')
         y_piano, sr_piano = librosa.load('A4_piano.mp3')

In [48]: S_guitar = np.abs(librosa.stft(y_guitar))
         S_piano = np.abs(librosa.stft(y_piano))

In [56]: librosa.display.waveplot(y_guitar, sr_guitar)
         # ipd.Audio(y_guitar, rate=sr_guitar)

Out[56]: <matplotlib.collections.PolyCollection at 0x7f35002d1160>
```

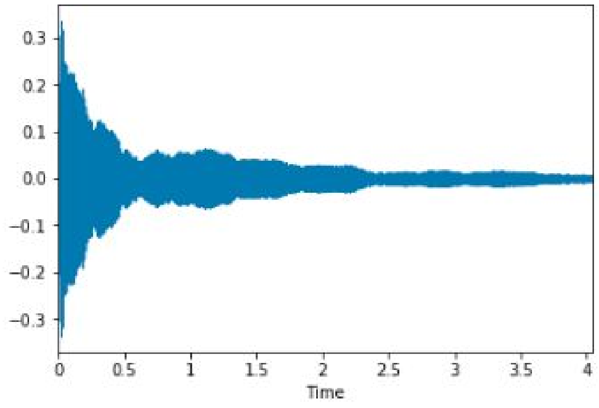


```
In [57]: librosa.display.waveplot(y_piano, sr_piano)
         # ipd.Audio(y_piano, rate=sr_piano)

Out[57]: <matplotlib.collections.PolyCollection at 0x7f350037b3c8>
```



And a plot of the note C3 played on a piano:

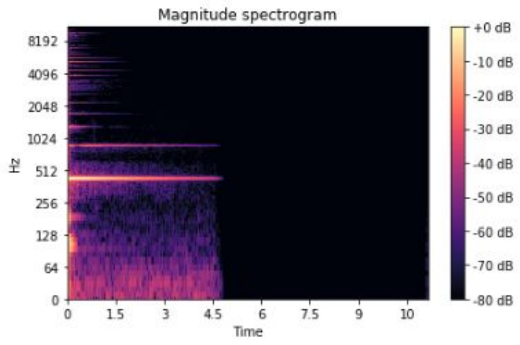
```
In [65]: librosa.display.waveplot(y_piano, sr_piano)
         # ipd.Audio(y_piano, rate=sr_piano)

Out[65]: <matplotlib.collections.PolyCollection at 0x7f35001f3b38>
```



As we can see, they are all quite similar.
Now lets see their plots after applying a short-time-fourier-transform:

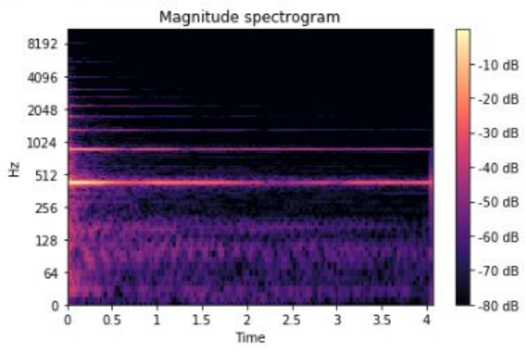
The guitar A4 note:

```
In [53]: librosa.display.specshow(librosa.amplitude_to_db(S_guitar, ref=np.max),y_axis='log', x_axis='time')
         plt.title('Magnitude spectrogram')
         plt.colorbar(format='%+2.0f dB')
         plt.tight_layout()
```



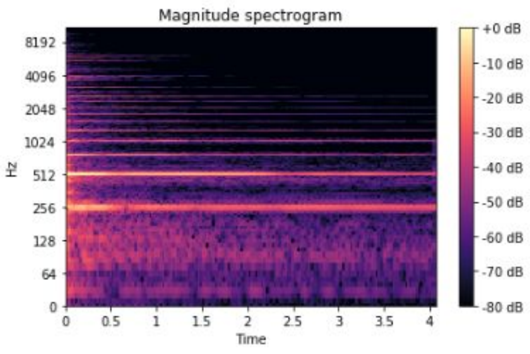
The piano A4 note:

```
In [54]: librosa.display.specshow(librosa.amplitude_to_db(S_piano, ref=np.max),y_axis='log', x_axis='time')
         plt.title('Magnitude spectrogram')
         plt.colorbar(format='%+2.0f dB')
         plt.tight_layout()
```



The piano C4 note:

```
In [66]: librosa.display.specshow(librosa.amplitude_to_db(S_piano, ref=np.max),y_axis='log', x_axis='time')
         plt.title('Magnitude spectrogram')
         plt.colorbar(format='%+2.0f dB')
         plt.tight_layout()
```



We can see that in each plot there is exactly one line that is more dominant than all the others.
That line marks the **fundamental frequency**.
All the other lines above it are it’s partials, that frequency multiplied by a constant - 1, 2, etc.

i.e - the A4 note has a frequency of 440, and we can see lines at 440, 880, 1320, etc.

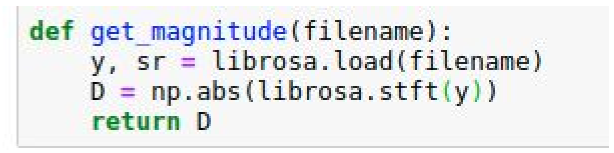
We also considered using other features that we can extract from our data, such as ZCR - zero crossing rate, MFCC — Mel Frequency Cepstral Co-efficients, and more,

but we decided to use the magnitude matrix produced from the short time fourier transform since it promised the best results.
We also decided to add the ZCR feature to the best model we will create to see if it affects the results.

Preparing the dataset:

Our dataset contains 88 piano notes, and 44 guitar notes.
We decided to use the piano collection for training and the guitar collection for testing our models.

We created the function get_magnitude which extracts the features from an audio file:

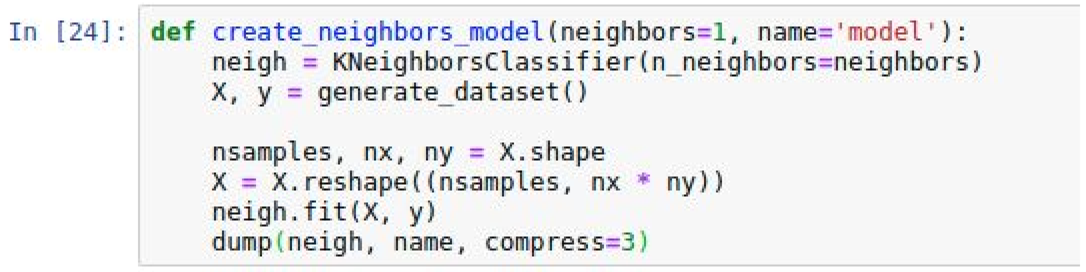


```
def get_magnitude(filename):
    y, sr = librosa.load(filename)
    D = np.abs(librosa.stft(y))
    return D
```

Which enables us to now create our features matrix and target array.

Now all that is left to do is send them to any of the machine learning classifiers.

This function for example, builds a K neighbors model and saves it to our file system for later use.



```
In [24]: def create_neighbors_model(neighbors=1, name='model'):
             neigh = KNeighborsClassifier(n_neighbors=neighbors)
             X, y = generate_dataset()

             nsamples, nx, ny = X.shape
             X = X.reshape((nsamples, nx * ny))
             neigh.fit(X, y)
             dump(neigh, name, compress=3)
```

Creating & Training the models:

We used similar functions to create 5 different models:

2 K neighbors classifiers - one where k = 1 and one where k = 3.
2 Support vector machines - one a support vector classifier and one a linear SVC.
1 Stochastic gradient descent classifier.

After we successfully created our models and trained them(which are the first machine learning models we've ever trained btw! @_@), it was time to test their performance!

But before that, we tried to guess based on our knowledge which ones will be the best and which ones...wont.

So based on the paper[1] we read, the best one was the neighbors classifier, and the worst ones were the svm classifiers. We can assume that the results will be quite similar since we use the same features, but we cant be certain because they used a different dataset and probably different parameters for the models.
We guessed that the best one will be the neighors classifier where k = 1, since our data doesn’t have any noise, and the sgd will be second.

Testing the models:

We used sklearn’s classification_report and accuracy_score for reviewing our model’s performance.

This is the meaning of each of the columns that will show in the tables:

Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as the ratio of true positives to the sum of true and false positives.

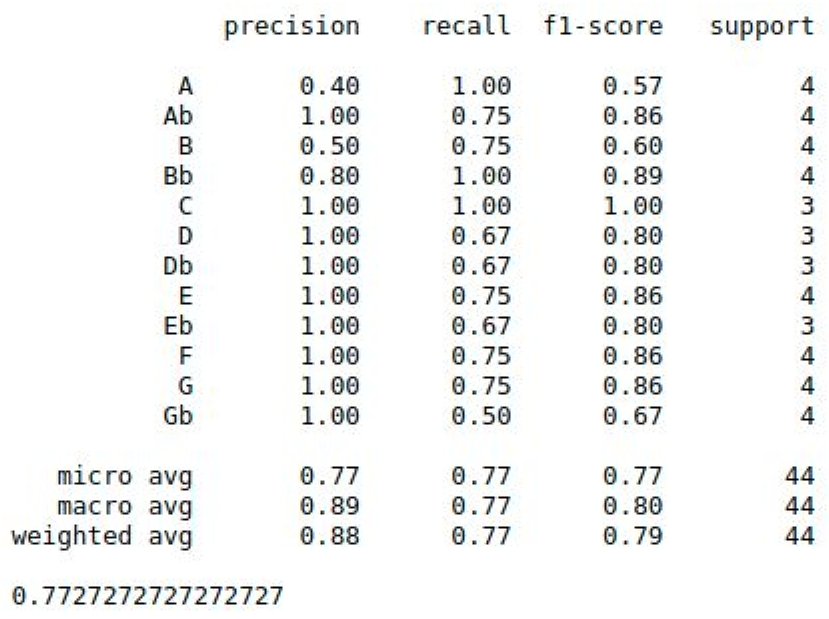
Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.

F1 Score - The relative contribution of precision and recall to the F1 score are equal, which makes it the best measure for the model’s performance.

The moment has come to test our models!

We decided to start with the SVM’s:

1.Support Vector Classifier:

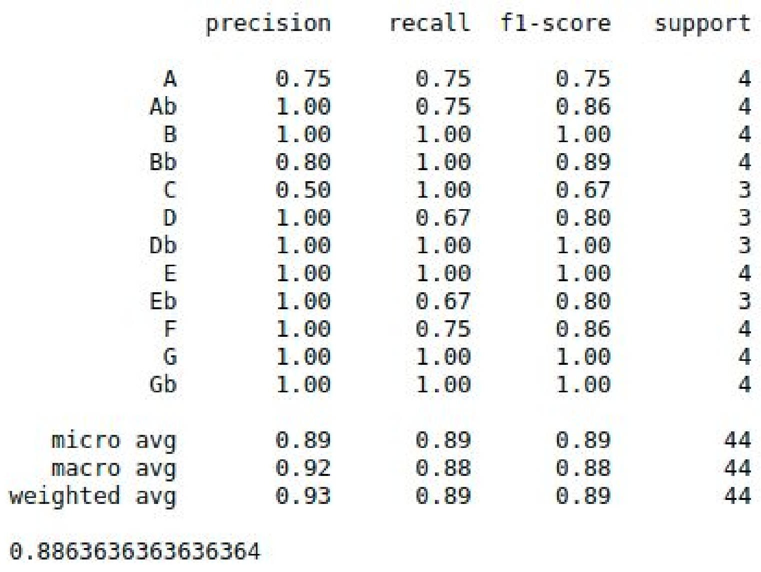


```
              precision    recall  f1-score   support

           A       0.40      1.00      0.57         4
          Ab       1.00      0.75      0.86         4
           B       0.50      0.75      0.60         4
          Bb       0.80      1.00      0.89         4
           C       1.00      1.00      1.00         3
           D       1.00      0.67      0.80         3
          Db       1.00      0.67      0.80         3
           E       1.00      0.75      0.86         4
          Eb       1.00      0.67      0.80         3
           F       1.00      0.75      0.86         4
           G       1.00      0.75      0.86         4
          Gb       1.00      0.50      0.67         4

   micro avg       0.77      0.77      0.77        44
   macro avg       0.89      0.77      0.80        44
weighted avg       0.88      0.77      0.79        44

0.7727272727272727
```

Our SVC got a total score of 77.2% accuracy, which is pretty nice for such a small data set. It’s precision was pretty good, and it got 9 perfect score out of 12, Albeit it’s recall results are pretty bad, it couldn't find the right pitch class for a big portion of the data. This gives us a poor f1-score and a poor model altogether.

2. Linear Support Vector Machine:

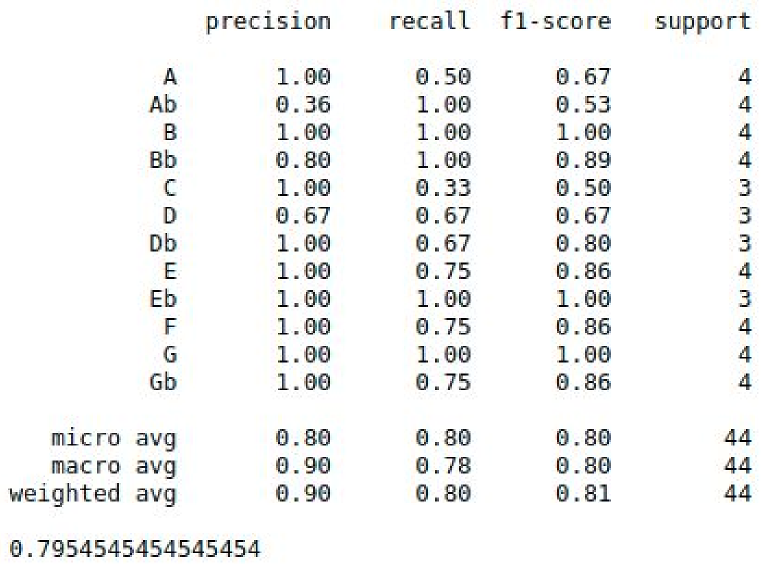
Our linear SVM performed a lot better than we expected:



```
              precision    recall  f1-score   support

           A       0.75      0.75      0.75         4
          Ab       1.00      0.75      0.86         4
           B       1.00      1.00      1.00         4
          Bb       0.80      1.00      0.89         4
           C       0.50      1.00      0.67         3
           D       1.00      0.67      0.80         3
          Db       1.00      1.00      1.00         3
           E       1.00      1.00      1.00         4
          Eb       1.00      0.67      0.80         3
           F       1.00      0.75      0.86         4
           G       1.00      1.00      1.00         4
          Gb       1.00      1.00      1.00         4

   micro avg       0.89      0.89      0.89        44
   macro avg       0.92      0.88      0.88        44
weighted avg       0.93      0.89      0.89        44

0.8863636363636364
```

It got a total score of 88.6%, which is really high and surprising considering the results we expected. Both it’s recall and prediction were good, and the overall model performed well. We assume that the reason for the good results is that the model uses a one-vs-the-rest scheme, as opposed to the regular svm which uses a one-vs-one.

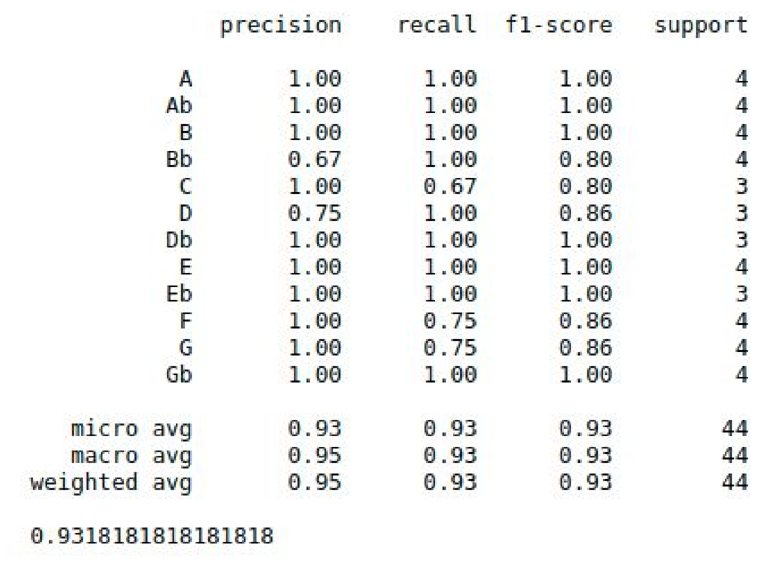
3.Stochastic Gradient Descent Classifier:



```
              precision    recall  f1-score   support

           A       1.00      0.50      0.67         4
          Ab       0.36      1.00      0.53         4
           B       1.00      1.00      1.00         4
          Bb       0.80      1.00      0.89         4
           C       1.00      0.33      0.50         3
           D       0.67      0.67      0.67         3
          Db       1.00      0.67      0.80         3
           E       1.00      0.75      0.86         4
          Eb       1.00      1.00      1.00         3
           F       1.00      0.75      0.86         4
           G       1.00      1.00      1.00         4
          Gb       1.00      0.75      0.86         4

   micro avg       0.80      0.80      0.80        44
   macro avg       0.90      0.78      0.80        44
weighted avg       0.90      0.80      0.81        44

0.7954545454545454
```

Our SGD classifier got a nice score of 79.5%. It’s precision was pretty nice but the recall results weren't good enough.

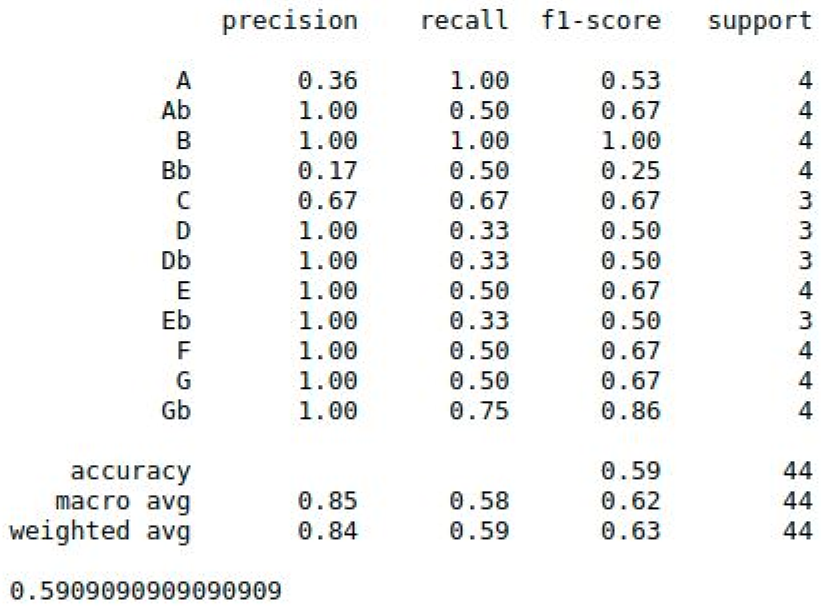
4. K = 1 Nearest Neighbors:



```
              precision    recall  f1-score   support

           A       1.00      1.00      1.00         4
          Ab       1.00      1.00      1.00         4
           B       1.00      1.00      1.00         4
          Bb       0.67      1.00      0.80         4
           C       1.00      0.67      0.80         3
           D       0.75      1.00      0.86         3
          Db       1.00      1.00      1.00         3
           E       1.00      1.00      1.00         4
          Eb       1.00      1.00      1.00         3
           F       1.00      0.75      0.86         4
           G       1.00      0.75      0.86         4
          Gb       1.00      1.00      1.00         4

   micro avg       0.93      0.93      0.93        44
   macro avg       0.95      0.93      0.93        44
weighted avg       0.95      0.93      0.93        44

0.9318181818181818
```

We just found our best model so far, with an outstanding precision and recall scores of 93%! It did a great job in every aspect. That was expected of the KNN algorithm since it’s use case fits our problem perfectly. It’s not surprising that it got such a great result when K = 1 since our samples are high quality and without any noise.

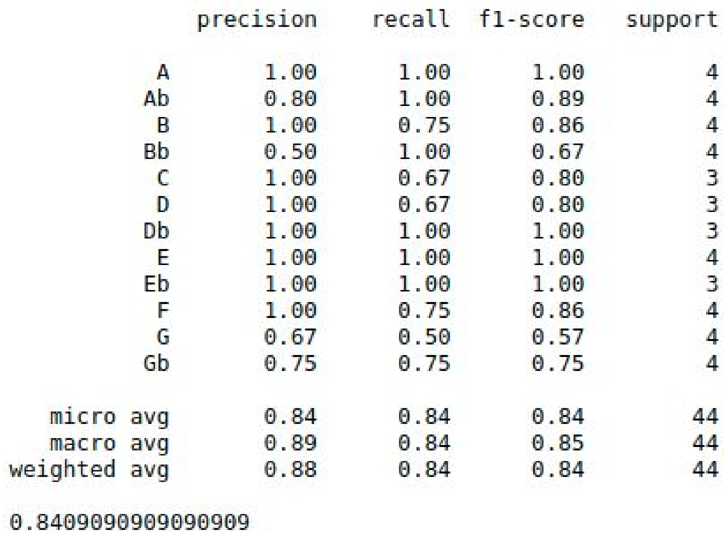
5.K = 3 Nearest Neighbor:



```
              precision    recall  f1-score   support

           A       0.36      1.00      0.53         4
          Ab       1.00      0.50      0.67         4
           B       1.00      1.00      1.00         4
          Bb       0.17      0.50      0.25         4
           C       0.67      0.67      0.67         3
           D       1.00      0.33      0.50         3
          Db       1.00      0.33      0.50         3
           E       1.00      0.50      0.67         4
          Eb       1.00      0.33      0.50         3
           F       1.00      0.50      0.67         4
           G       1.00      0.50      0.67         4
          Gb       1.00      0.75      0.86         4

    accuracy                           0.59        44
   macro avg       0.85      0.58      0.62        44
weighted avg       0.84      0.59      0.63        44

0.5909090909090909
```

And we just found our most horrible model, with a poor score of 59%. It’s not so surprising that the same algorithm that gave us the best score also gave us the worst one with a different K since it is used for problems where there is a lot of noise in the data, which is not the case.

Bonus: K = 1 Nearest Neighbor with zero crossing rate:



```
              precision    recall  f1-score   support

           A       1.00      1.00      1.00         4
          Ab       0.80      1.00      0.89         4
           B       1.00      0.75      0.86         4
          Bb       0.50      1.00      0.67         4
           C       1.00      0.67      0.80         3
           D       1.00      0.67      0.80         3
          Db       1.00      1.00      1.00         3
           E       1.00      1.00      1.00         4
          Eb       1.00      1.00      1.00         3
           F       1.00      0.75      0.86         4
           G       0.67      0.50      0.57         4
          Gb       0.75      0.75      0.75         4

   micro avg       0.84      0.84      0.84        44
   macro avg       0.89      0.84      0.85        44
weighted avg       0.88      0.84      0.84        44

0.8409090909090909
```

After seeing how good the KNN algorithm was, we decided to use it with an extra feature for the feature matrix - the zero crossing rate, which represents the times the signal crosses the X line. The results are worse than the original one, as expected, since the audio files are at different lengths.

What We Learned:

First of all we learned that we love the field of AI!
This field holds the future in its hands, and no time has ever been better to start sailing towards the endless sea of big data and machine learning, since we now have all the tools necessary for creating almost anything we can think of, the 4 pillars of AI are now freely accessible:

1. Data - Kaggle, Google Dataset Search
2. Algorithms - Arxiv, GitHub
3. Compute - Google Colab, Kaggle Kernels
4. Education - School of AI, KhanAcademy, Fast .AI

There are a lot of high level api's for every environment we wanna develop with - .NET, Node, Python and more.

Second of all, we learned how to use some of these api's(which are all very similar - create a model instance, use fit method to train it, predict method to predict, etc).

We are happy that we chose to build this project from scratch,

That helped us understand exactly how to collect data, search for features inside the data, create a training set and a test set, creating our models and training them, and then using them to predict for different data! We even recorded ourselves singing different notes and then fed our best model with the recordings, which identified them correctly.

We learned about different supervised learning algorithms:

The SVM can perform both linear and non-linear classification, and it works by representing examples as points in space and then trying to map them so that there will

be a clear gap between the examples from one class and the examples from the other ones.
The KNN decides the result class for a specific sample based on a majority vote of its k neighbours. K should be set according to the amount of noise in the data.

The SGD aims to minimize a loss function. It is similar to the Gradient descent algorithm, the only difference being that the SGD updates itself after each sample while GD updates after completing a full iteration, which means it starts approaching the minimum with the first sample it goes over and then continues to improve itself with each new sample.

Conclusions:

We have succeeded in building a well performing model for detecting the pitch class of monophonic audio, and it generated great results both on human voice and guitar recordings. We have learned a lot and thought about a ton of different ideas of where we can take this further(which hopefully you'll be able to see next year!)

The full source code for the project can be found in this repository:
https://github.com/Seanitzel/Pitch-Class-Detection-using-ML

Bibliography:

[1]
https://www.academia.edu/7560072/Musical_Pitch_Detection_Using_Machine_Learning_Algorithms - The paper we read to got hints on how to approach the problem.

[2] https://scikit-learn.org/stable/index.html - the python library we used to implement our solution.