ChordSuggester: using LSTMs to predict musical chord sequences

Juan Luis García López

Jan 10 · 15 min read

Logo of the application

ChordSuggester is a computer-aided musical composition system. It is not intended to be a professional tool but just the result of a Master's thesis which tries to cover the whole process for a DataScience project:

- **Data Acquisition** by scraping data from ultimate-guitar.com using *Selenium and BeaufitulSoup*. This part is interesting by itself since there are no examples of clean datasets including chord songs.
- **Data cleaning and preparation,** using *Pandas* and *music21*.
- **Data analysis,** using Pandas.
- **Modelling,** using Keras for training an LSTM neural network.
- **Visualisation of the results** on a *React* Application that consumes the model using *TensorFlow.js* and shows the results using the musical JavaScript libraries *Tone.js* and *Vexflow*.

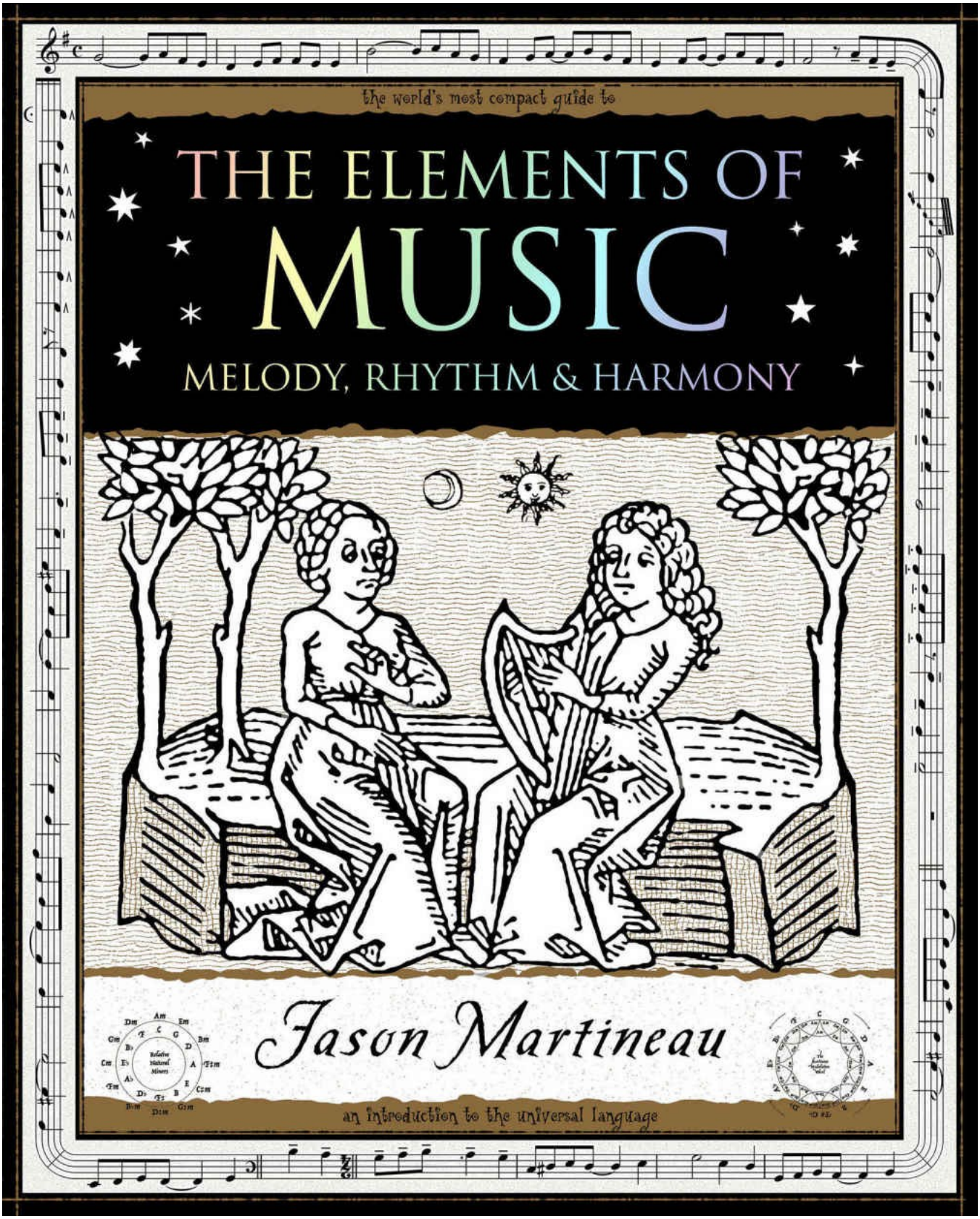
. . .

Some music theory

To fully understand the system, it is necessary to have a basic knowledge of music theory. In this article some important musical concepts will be explained.

The elements of music

Musical composition includes several aspects, such as *melody, rhythm or harmony.*



The Elements of Music: Melody, Rhythm, & Harmony: Melody, Rhythm and Harmony by Jason Martineau

- **Melody** refers to how notes are disposed *horizontally, in time, one before the other*. For example, Gregorian Chant focuses just on melody, having a repetitive rhythm and no harmony.

Gregorian Chant score: all the notes have same duration (no rhythm) and only one note sound at once.

- **Rhythm** refers to *how note durations are combined*. Even using just rhythmic resources, good compositions can be created. One example could be African percussion, which tends to be much richer than classical European music.

Sometimes, a rich rhythm is enough

- **Harmony** refers to how notes are disposed *vertically* i.e. how several ones sound *at the same time*. ***ChordSuggester* will focus on this aspect**. For example, the second movement of Beethoven's 7th symphony does not have a special melody or rhythm, but its harmony makes it one of the great masterpieces on history.

Of course, these three elements can be combined to create even more interesting compositions. A well-know example is *The Pink Panther Theme* by the great American composer *Henry Mancini:*

What is a note?

A **note** is just a sound with a given **frequency** (in reality is an aggregate of several frequencies but we can simplify the problem attending just to the frequency with more amplitude, called *fundamental frequency)*. This sound can be represented mathematically:

A note with its several frequencies. In this case, the **fundamental frequency** is the **red** one.

The English system represents musical using letters *(A,B,C,D,E,F,G) corresponding to (La,Si,Do,Re,Mi,Fa,Sol) in Latin system* .

Origin of Latin naming system. UT was replaced by DO for simplicity when singing. SI comes from the initials of Sant Joannes

Attending only to the fundamental frequency, we can univocally identify a note with a frequency:

Musical Harmonic Series

What is an interval?

The interval is the distance between two notes. This distance is measured in semitones (or half-steps in USA). The distance between adjacent notes in a piano is one semitone.

Distance between notes

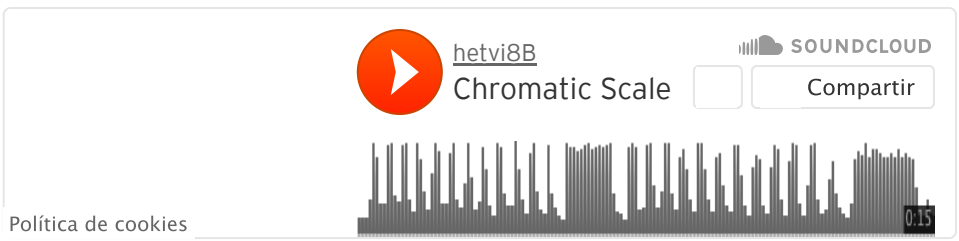
For example, the distance between D and G is 5 semitones (ST), i.e., 2 tones (T) and 1 ST.

Additionally, we can use *accidentals* to modify our notes:

- A '#', called *sharp*, increases a note in one ST.
- A ' ♭ ', called sharp, increases a note in one ST.

Note that, for example, G ♭ = F#, A ♭ = G# or even E# = F (distance between E and F is just one ST).

As a result, we have the *chromatic scale*, that contains all these 12 notes:



Chromatic scale (ascending and descending) played by a flute.

Ascending and descending chromatic scale in a sheet

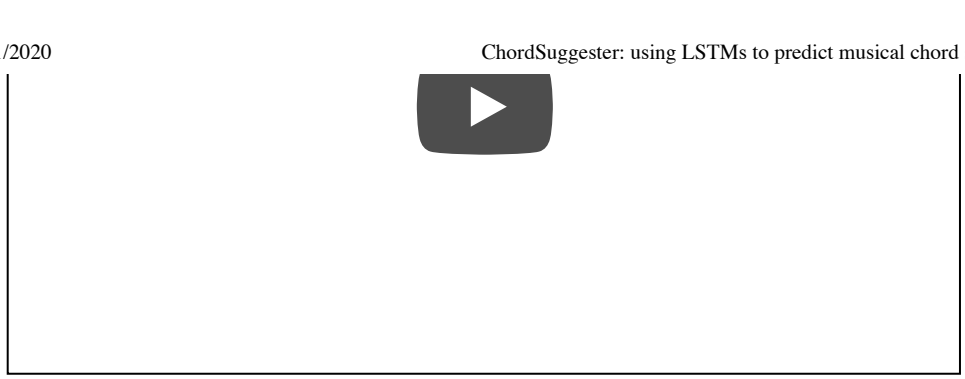
The intervals have a musical name. For example, a distance of 3 ST is called a *minor third*. Below there is a table including interval names and frequency ratios:

Interval names (0 STs = Unison, 1 ST = Minor 2nd...) and its frequency ratios

Let's attend a moment to the frequency ratios. Note that the *simplest* ratios (excluding unison and octave that are trivial) correspond to *Perfect Fifth* (3/2) and *Major Third* (5/4).

And… here comes the curious thing!! These intervals are considered the most pleasant (*consonant* is the musical term) by our ear versus other more complex (*dissonant*) as Minor Second (16/15). Therefore, our musical system is not arbitrarily chosen, it makes physical sense and even satisfies the KISS principle!

More information about that (video in Spanish):



What is a chord?

A *chord* is a set of several notes sounding at the same time. Theoretically, there is no limit on the number of notes to be included in a chord.

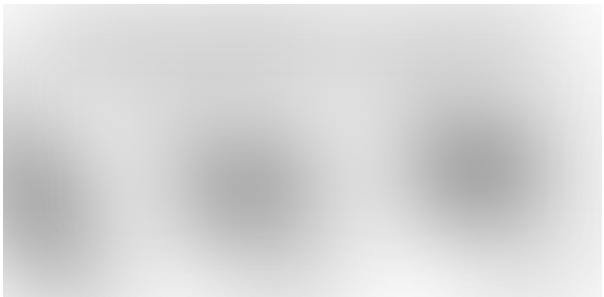
Nevertheless, it's not very common to have more than 6 notes in a chord and the more usual is to have 3 or 4.

We find several ways to represent a chord:

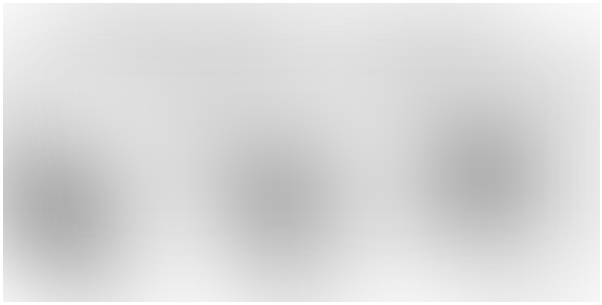
- Indicating the notes it has (we will see it in the code, where we use an array of 12 positions).
- Naming it. This is the musical way.

The name of a chord is composed by:

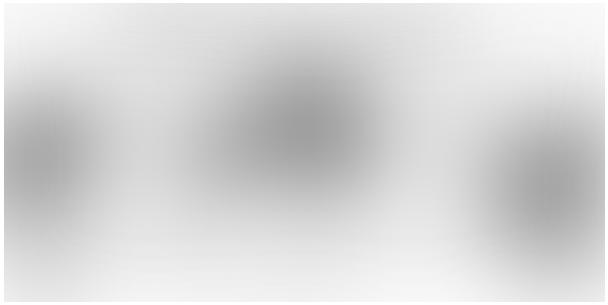
- **Root (AKA fundamental)**: is the main note. For example, could be 'B'
- **Type or Quality**: is a name that indicates the intervals from the root note. For example, the most extended chord, that is the *major* chord contains a *Major Third* (4 STs) and a *Perfect Fifth* (7 STs) from the root note. Again, **most extended = simplest.** For example, a *Cmaj* chord or simply C is composed by C, E and G notes.



Major chords: 4 and 7 semitones



Minor chords: 3 and 7 semitones



7th chords, 4, 7 and 10 semitones

- **Slash**: A **slash chord** is a chord (eg. G/B) which indicates emphasis of a bass note other than the root of the chord. When a chord is played it is typically assumed the bass will emphasise the root of the chord. Occasionally a different note is preferred and results in a chord with an alternate bass note. The inclusion of the slash note (AKA *on*) made the problem more complex, so it was ignored for the current approach.

Relation between chords: the circle of 5ths

The harmony not only consist on setting notes vertically (create a chord) but also on disposing chords horizontally (one before the other).

In order to be interesting, a composition must create *tension* and *distension*. There are tons of theory behind that, but we can summarise it by looking at this diagram called ***Circle of Fifths:***

Circle of fifths

Let's suppose that we start with the chord **C**. If we go clockwise and choose a chord (the nearest, the more normal), we will create tension: **G**. The ear will ask us to go back counterclockwise and play again a **C**. This is the simplest chord progression and, again, on of the most common. Other interesting progression is **C-F-G**, that initially creates distension in order to generate a greater tension when going to G. For that reason, there are lost of songs with an amplitude **3** inside the circle of 5ths.

More information:

. . .

Data Acquisition

It is difficult to find a clean dataset containing chords for lots of song. A good example is this one that contains *jazz* progressions extracted from Real Book. However, there is a lack of datasets with a wide range of styles.

There were a handful of options to generate this data:

- **Extracting from mp3** or similar files. This approach had two problems: one note very difficult to manage: *find mp3s*. The second one was more important: the ***clean information rate***. MP3 size is huge in comparison with the clean information to extract. Eg: a 3MB mp3 could contain only a sequence of 50 chords (MB order vs Kb order). That could make the data extraction process tedious and slow.
- **Extracting from midi** files. This approach bust the firstly chosen, because it is easy to find midi pages or even GBs of data inside midi zip compilations. Additionally, the clean information rate is much higher (KB order vs Kb). Nevertheless, very soon I discovered that this process is *easy* for certain types of music (homophonic music):

Bach Chorale: each measure is a chord

…while it is very difficult for other types (polyphonic music):

Das Wohltemperierte Clavier (J.S Bach). In each measure, notes not belonging to the chord are included

There are some tools to analyse music, but it was a non-trivial work.

- **Scraping from a chord page**: There are lots of pages providing guitar chords, such us UltimateGuitar, La Cuerda, … In this case, forgetting about the scrap process itself, the information was almost totally clean. For a song, a list of chord names were provided.

Scraping from UltimateGuitar

Finally, the third approach was taken and a scraping library was necessary to extract data from one of the most known guitar chords page: UltimateGuitar. The chosen library was BeautifulSoup.

The scraping process had several challenges:

- UltimateGuitar is **protected** against web scraping. I consider using some pay methods such the easy-to-use ScraperApi bit, finally, *the thousand-year old technique*

of rebooting the router was fairly enough. As a counterpart, some special strategies to store already-scraped data, resume the last error page and merge data was necessary.

Number 429 is like 666 for Data Scientists

- There is **no index** to get the song list from. To work around that, I analysed the url patterns when searching for different criteria. I chose *decade*, *genre* and *sub-genre*. In later iterations I removed the last criteria because using just *decade* and *genre* was faster and the generated data amount was enough.

```
https://www.ultimate-guitar.com/explore?
genres[]=14&&decade[]=2010&subgenres[]=343
```

Not only songs were scraped, but also the *genre*, sub-genre (AKA *style*) and *decade* dictionaries.

- When opening the page, a **cookie notification message** appeared preventing the desired HTML to appear. BeautifulSoup just analyses *html* so it cannot simulate event such as *clicks*. The solution was using Selenium Driver, that can directly communicate with a browser a perform a human-like browsing.

Example of cookie notification message

- Some songs have several styles, so we have some **duplicates** we had to deal with.

All this process can be found in these two notebooks:

- Scraping — Extracting filter criteria.ipynb.
- Scraping — Extracting songs.ipynb

. . .

Feature Extraction

The result of the previous process was a dataset containing the following columns:

- **url**: Raw URL of the song. String.
- **name**: clean name of the song. String. Already cleaned in previous phase.
- **decade:** raw decade of the song. String with format: *1990s*.
- **genre:** raw genre data. String of genres separated by *%%*. Can contain repeated genres (eg. *Folk%%Country%%Country*).
- **chords**: list of chords. String containing the chords. Can be converted to a python list using *eval*. Eg. [‘C’, ‘F’, ‘C’, ‘D7’]
- **uuid**: unique identifier automatically generated by scraper.

Raw dataset sample

From this raw data, several features were extracted, not only to explore if they could be useful to train a model but also to have the ability so analyse songs ins a better way. The dataset i

Let's explain some of these features

- **artist**: extracted from url. From this point, we could inspect the difference between the music composed by, for example, David Bowie and Justin Bieber.
- **decade**: converted to numeric.
- **genres**: not duplicated but still separated by %%.

Genres by number of appearances

- **cardinality**: number of chords of the songs. It counts repeated chords. Eg. for *[D,D,E]*, cardinality = 3.

Cardinality distribution. Mean around 85.

- **unique cardinality**: number of *different* chords of the songs. It does not counts repeated chords. Eg. for *[D,D,E]*, cardinality = 2. This feature is much more interesting than cardinality because if indicates the *horizontal (in time) richness* of a song.

Unique cardinality

The mode is around 4: the popular music is not very rich in general in ***unique cardinality*** terms, but remember that there are other methods to generate interesting music, such as melody, timbre or rhythm.

Jazz is, with difference, the richest music in unique cardinality terms.

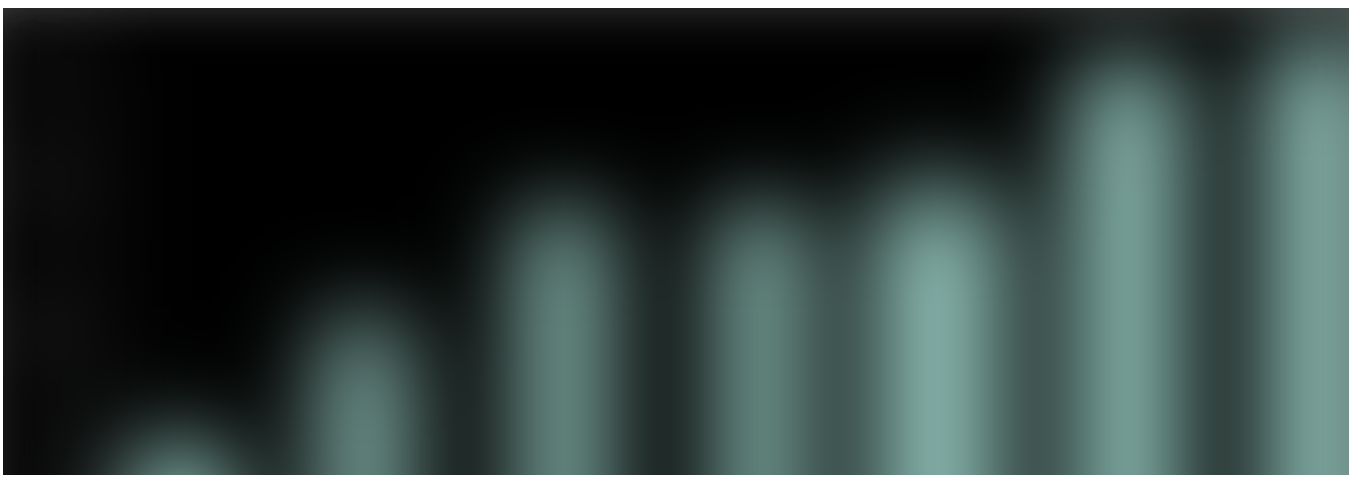
- **major cardinality, minor cardinality** and **neutral cardinality.** Being simplistic:

1. A chord can be considered *happy* (*major* is the musical term) if contains the *3th major* interval (2 tones).
2. A chord can be considered *sad* if contains the *3th minor* interval (1 tone and a semitone).
3. The rest of chords (relatively rare chords) can be considered *neutral*.

These three features contains the number of happy, sad or neutral chords for each song.

- **sadness**. Generated from previous features. Is the ratio between minor chords and total chords.

Generally, music tends to be happy



But it is becoming sad in a worrying progression...

- **harmonic mean:** mean position of song in 5th circle (see first section of this article). A chord has a position in this circle. This position is encoded by using X and Y. The harmonic mean of a song is the average of the Xs and Ys among its chords.

Harmonic mean. Tonalities with # are more usual. The shorter the radius is, the longs the harmonic deviation of the song

- **subdominant deviation:** difference between harmonic mean and farthest chord in counterclockwise direction.

Subdominant deviation distribution

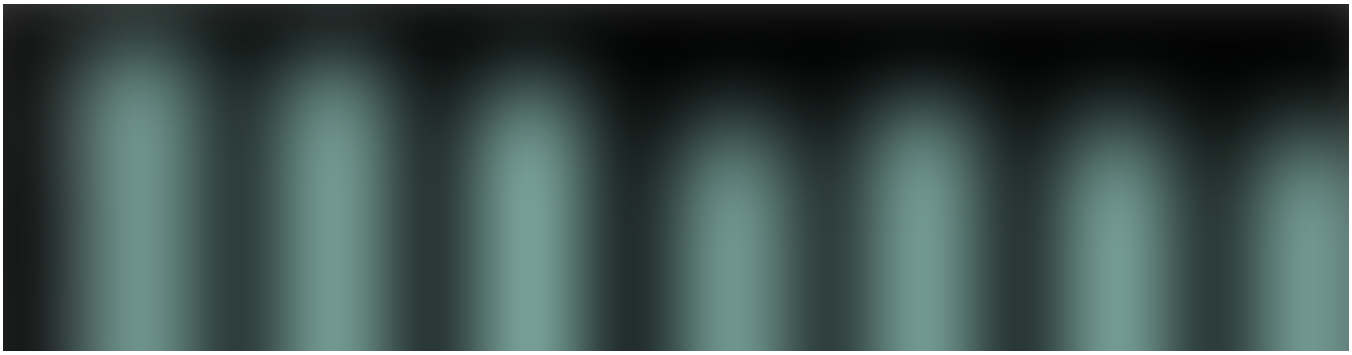
- **dominant deviation:** difference between harmonic mean and farthest chord in counterclockwise direction.

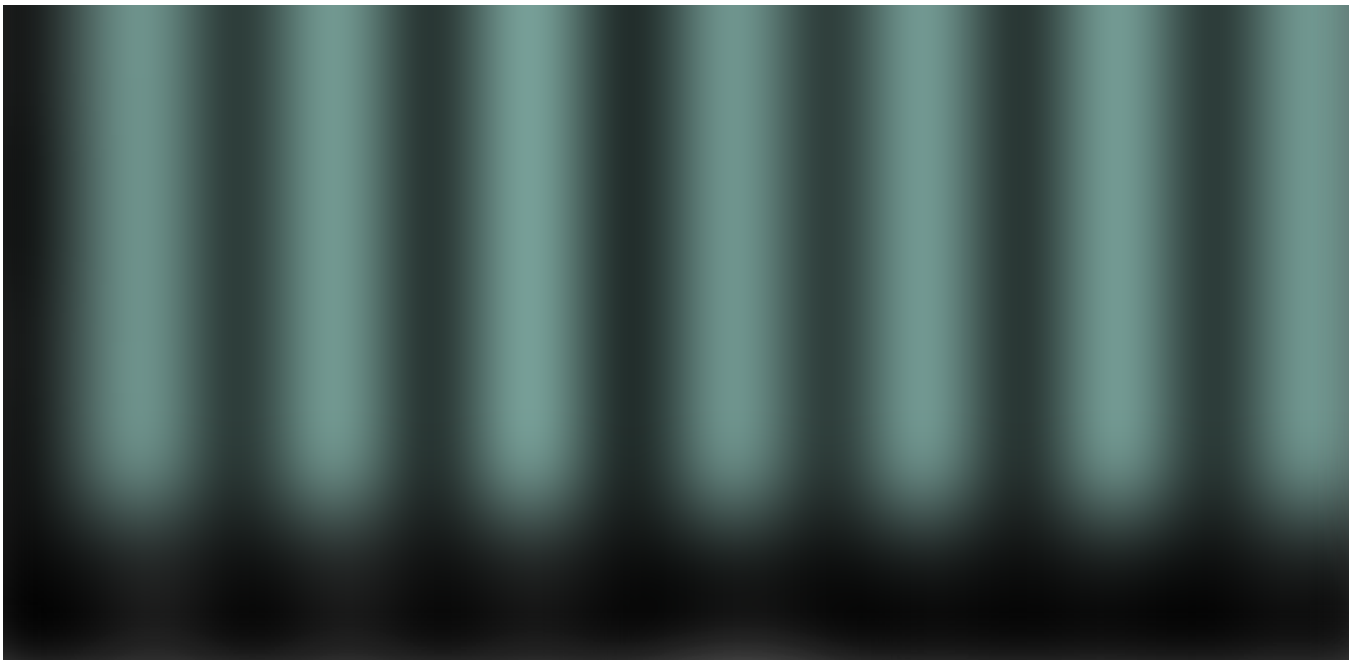
Dominant deviation. Increase in 3–4 step could indicate the major chord borrowed by minor tonality

- **harmonic width**: sum of dominant deviation and subdominant deviation.

Jazz is, again, the richest genre in terms of harmonic width

- **complexity:** a chord is complex if the frequency relationship between its notes is complex (see first section of this post). This feature indicates the mean complexity of each song.





We are getting simpler and simpler...

- **normalised chords**: a chord sequence can be transposed to an equivalent one that just have a lower or higher pitch. The model, or at least that was my hypothesis, will learn better is we transpose (~normalise) these sequences.

The generated dataset has value by itself, because it can used as a starting point for any other DataScience project. Some examples:

- Clustering songs, artists or genres by these characteristics.
- Composing music of a given genre or decade. For artist, we would need more data.
- Adding the new features to Spotify API song characteristics.

All the feature extraction process can be found in this notebook.

. . .

Model

An LSTM was trained in order to, given an input chord sequence, predict the next one.

This problem could be treated as a simple word prediction. However, I found some differences:

Music = Maths

Circle of fifths

A chord can me represented mathematically without using embedding approaches (eg. word2vec). This thesis has researched the possibility to represent the chord as:

1. A 12-size vector, one feature per note, containing 0 (note present) or 1 (present).
2. A vector containing some of the previous described features (harmonic position, quality, etc.).

When using this approaches, the problem I found is that the appearance of a note depends on the appearance of the other. Eg. it is unlikely to have a C and a C# in a chord but it is very common to find a C and a E together in a chord. Therefore, these features are covariant/contravariant and no appropriate to feed a machine learning model.

There are synonyms

Cmaj is the same as *C* or *CM*. Other approaches do not consider this situation, but in our system we have normalised the data in order to convert all the variants into a canonical version.

C, Cmaj and CM are the same

We can transpose data

As we saw in *feature extraction* section, the songs can be transposed to have a normalised sequence with the same harmonic mean.

The model receives the normalised sequence and then the predictions are *denormalised* (i.e. transposed back to their right tonality).

. . .

Hyper-parameters

When training a neural network, there are lots of hyper-parameters to introduce. Let's talk about some of them:

I have used a fixed-topology neural network:

Topology of the network

Finally, I had to use an embedding layer although I was trying to avoid it…

The optimiser has been an *Adam* and the loss function *categorical_crossentropy* since it is a classification problem.

The variable hyper-parameters I have chosen are:

- The **input sequence length** (called W on the notebook). Tested 10 and 20. Finally 20 was chosen.
- The **learning rate**. Tested 0.1 (stuck), 0.01 (accuracy decay), 0.001 (better but overfitting) and 0.0005 (best).

With learning rate of 0.01 we have an accuracy decay from epoch 46

With learning rate of 0.001 we lose validation accuracy from epoch 22 (overfitting)

Training using learning rate of 0.0005. The graph indicates that maybe we could increment the number of epochs

- The **batch size**. Tested 1024 (stuck), 128 (best) and 32 (too slow).
- **Epochs** . Initially 20 to have a faster feedback and finally 50 when hyper-parameters chosen.
- **Normalise** of not normalise data. I found better performance normalising the data, but not so high as expected.

The LSTM training can be found in this notebook.

Conversion

In order to use the model in production (a *React* web application with no need of backend), the trained models have been converted to *Tensorflow.js* format.

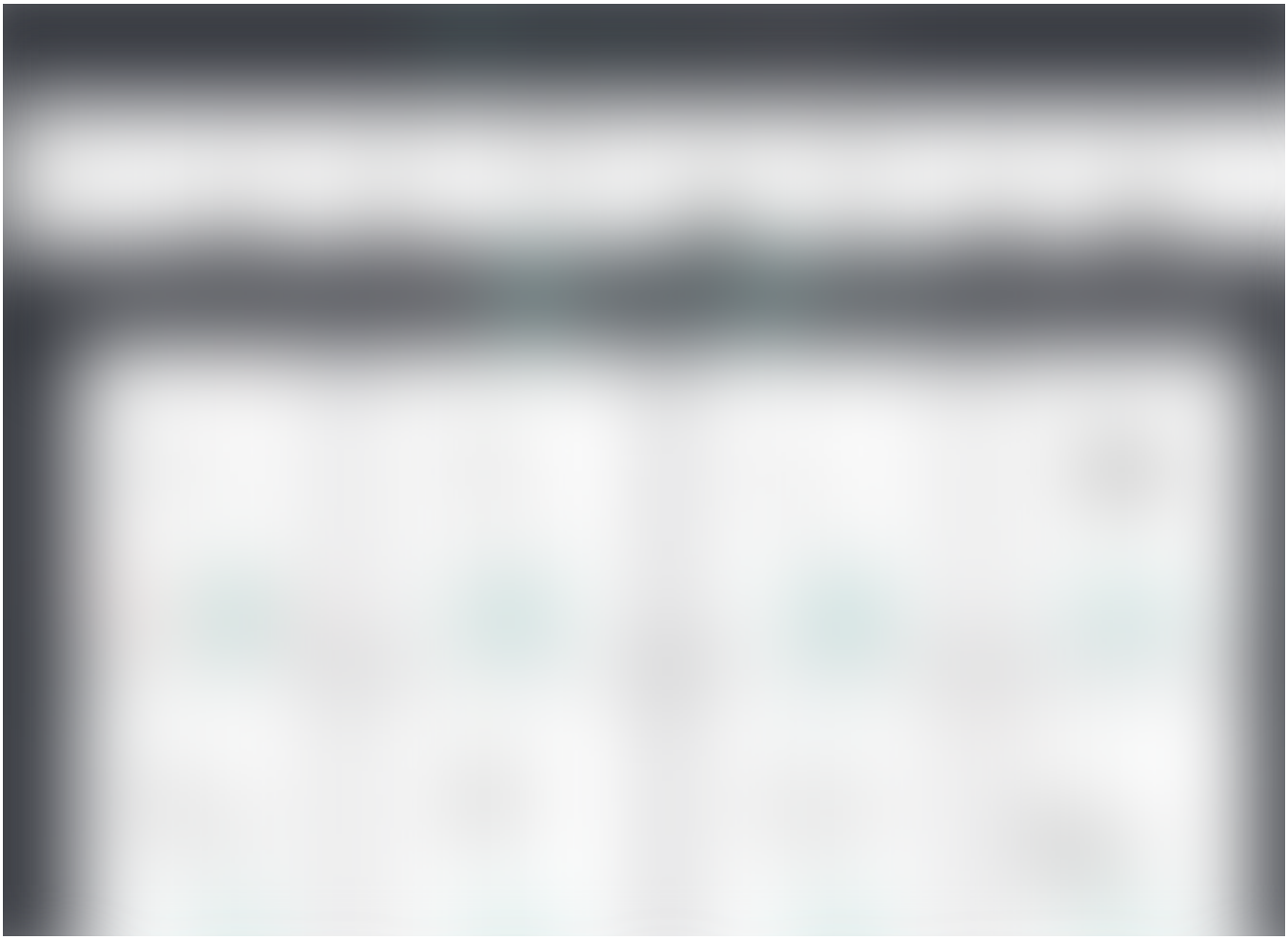
The instructions to convert the model can be found in the readme file.

Additionally, some python dictionaries used during feature extraction have been exported to json to use them in the frontend. The conversion is done in this notebook.

. . .

Frontend

A React application has been created to interactively test the model. The look and feel of the application is the following:



Look and feel of the application

1. Sheet where our composition is shown. In this case we can see the chords of Pachebel's Canon (they appear by default).

2. The most probable chord suggested by the model.

3. The second most probable chord suggested by the model.

4. Probability calculated by the model for C chord.

5. Click to listen C chord.

6. Click to add C chord to the sheet (1).

7. Click to choose a custom chord (it can be played or added by using the analogous buttons).

8. Transpose the sheet one semitone down.

9. Transpose the sheet one semitone up.

10. Remove last chord from sheet.

11. Clear the whole sheet.

12. Plays the song.

13. Loads default composition (Pachebel's canon).

14. Switches from two models (one trained with normalised data, default, and other trained with non-normalised data).

- To listen the songs, Tone.js has been used.
- To show the sheets, VexFlow has been used.

. . .

Some considerations about cost of error, testing and continuous integration

The errors in a Data Science, like in any other software scope, have a cost that can be economic or even human.

The later an error if found the higher is the cost. Although errors are unavoidable, we can try to find them as soon as possible.

In my personal case, an error could make me lose time looking for the problem, that is also an important thing :).

Therefore, a set of tests have been created in both python and javascript projects to early find possible errors when creating features, cleaning data, arranging data for training, etc.

CI/CD should be a must in any Data Science Project. As little example brand new feature *GitHub Actions* have been configured in python repository to automatically run the tests.

. . .

Conclusions

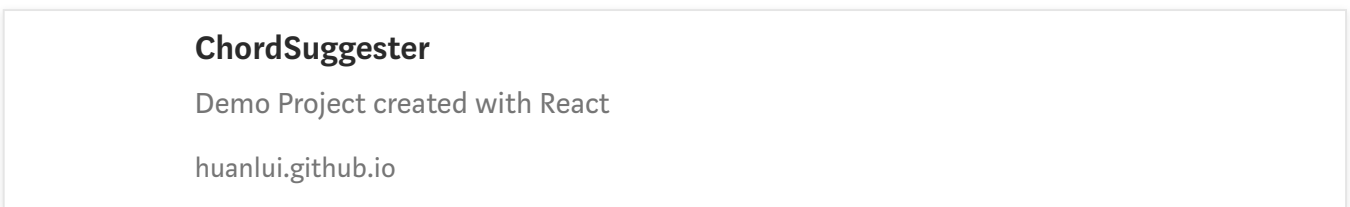
It's been a project where:

- I have had the chance to have a journey through all the stages of a Data Science project.
- I have applied Data Science techniques to a field I love: **music**.
- I have managed to make some innovative things compared to similar existing projects.
- I have presented the data in a beautiful way, having a product that could even have real users.
- I have applied Software Engineering principles to improve quality and reduce the error probability, something very important in this field.

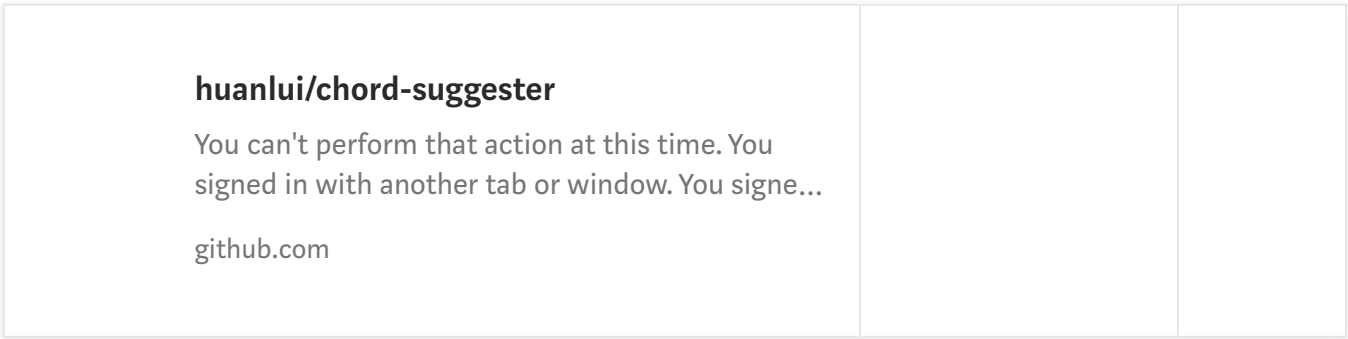
. . .

Ok, but...show me the code!

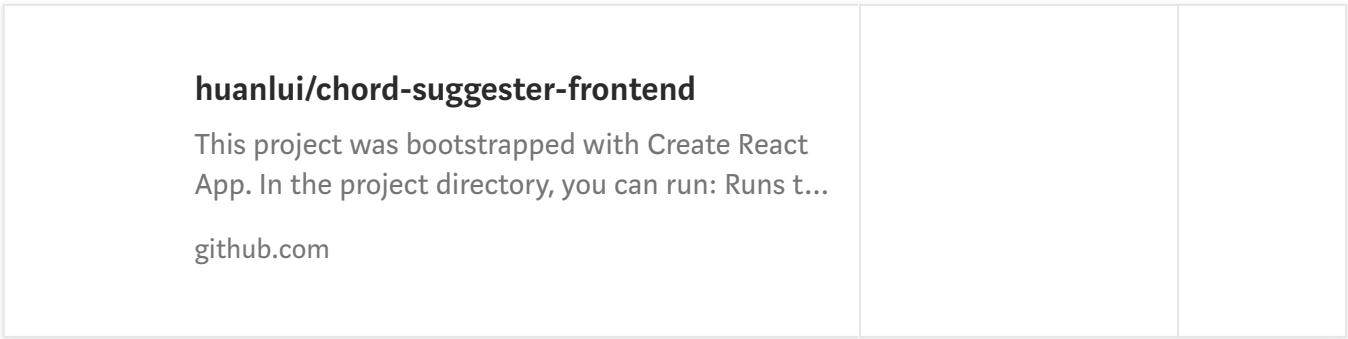
- **Demo**: https://huanlui.github.io/



- **Source code**: Scraping, cleaning and training in Python (*BeautifulSoup*, *Pandas*, *Keras, music21* and a custom version of *pychord*):



- **Source code**: Front-end (*React* application and *Tensorflow.js*)



Data Science Deep Learning Music Lstm Scraping

About Help Legal