Sandvik CODE assignment

VoxForge repository analysis

1 - Introduction

The aim of the following report is to present the approach of my analysis. After answering the questions from the instruction file, I will highlight some issues which were not addressed.

I chose Python 2.7 with anaconda environment to address this interesting challenge. In order to run and test my code easily, I have pushed all of my work in the *Bitbucket* repository provided by mail.

Although I was not familiar with audio analysis, I highly enjoyed discovering this rich domain of data science and some useful python tools for audio analysis, such as the python library "thinkdsp". I hope that you will appreciate this report as much I did while performing the analysis.

2 – Approach and steps of analysis

As mentioned below, my global approach was always to test my code over jupyter notebook. It enabled me to build my script basis from a small sample of data and to be able to reproduce the logic on the entire dataset and industrialize the analysis.

a) *Exploring the VoxForge repository website*

I mainly used the common python web scraping library: Beautiful Soup. After exploring the html content of the webpage, I discovered that each downloadable link was composed by the same main part and ended by the folder name.
I got every folder name thanks to beautiful soup and stored it in a csv file entitled scraping_result.csv which included the folder name ("sub_link"), the recording date, and the archive size (the downloading is under ".tgz" format).

b) *Industrializing the web scraping*

After loading scraping_result.csv as a data frame, I implemented a loop on the field "sub_link" to get each archive contained in the web page.
This script was entitled "web_scraping_download_files.py" and took approximately 10 hours to be completed. It is not efficient, and we will see in the fourth part of this report how to reduce the running time.
After downloading all archives, I implemented two shell scripts to extract content from the archives in the same directory (*/data/downloads/*).

```
In [12]: #Create a dataframe to analyze
         d = {'sub_link':sub_link,'date':information, 'size':size}

         scraping_df = pd.DataFrame(d)
         scraping_df['date'] = pd.to_datetime(scraping_df['date'])
         scraping_df.head()
```

Out[12]:

	date	size	sub_link
0	2010-07-20 09:47:00	1.3M	1028-20100710-hne.tgz
1	2017-03-24 04:09:00	1.1M	1337ad-20170321-ajg.tgz
2	2017-03-24 04:09:00	1.1M	1337ad-20170321-tkg.tgz
3	2017-03-24 04:09:00	1.0M	1337ad-20170321-ynk.tgz
4	2012-05-11 04:52:00	1.5M	1snoke-20120412-hge.tgz

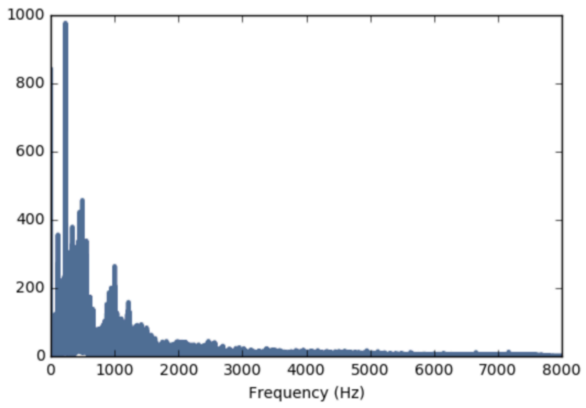
c) *Audio features exploring*

Through the use of the ThinDSP libraiy, I easily got the frequency of some audio records. Based on this insight, I calculated each features in the list of the instruction file.
Please find below some display of this part of the analysis.

```
In [3]:  wave = thinkdsp.read_wave('/Users/kevenlemoing/Sites/sandvik_code_assignement/data/downloads/abc-20091120-mfr/wav/a0066

In [4]:  spectrum = wave.make_spectrum()

In [28]: spectrum.plot()
         thinkplot.config(xlabel='Frequency (Hz)')
```

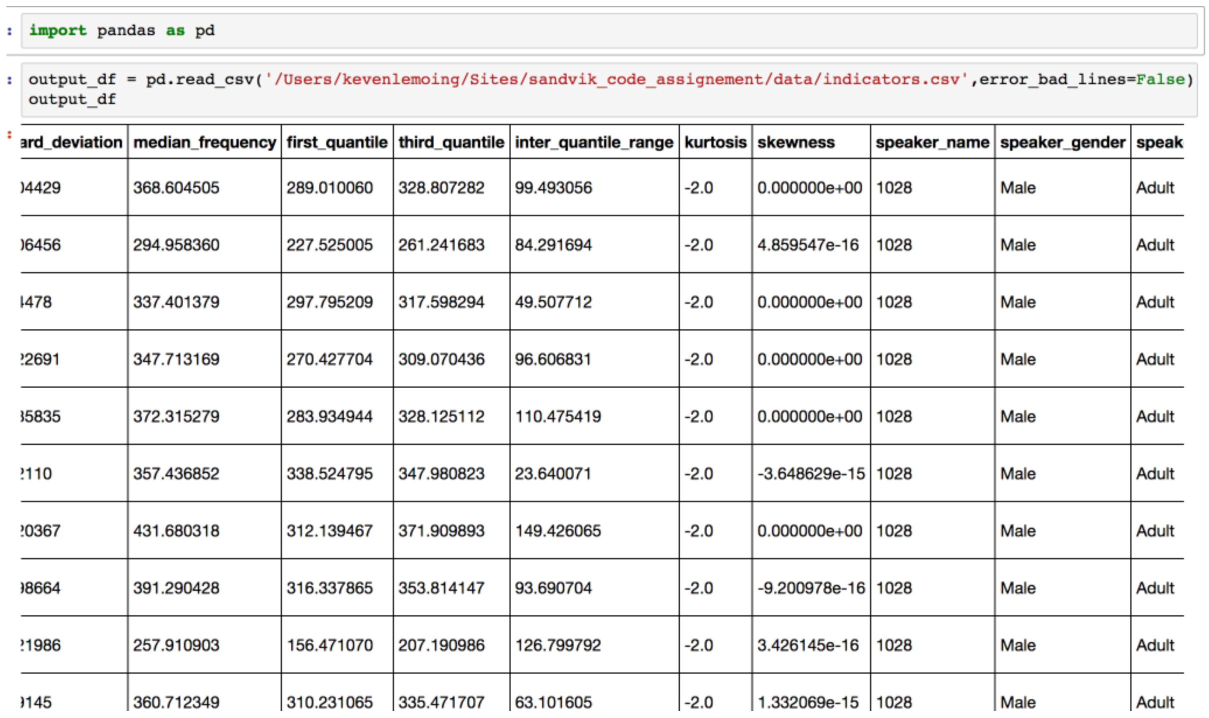


d) *No-audio data extraction*

Subsequently, I identified “README” and “prompts-original” as the main no-audio data sources to be extracted.
The library “*csv*” enabled me to have a “line by line” approach and to get the required data.
Please look at “*extracting_non_audio_data.ipynb*”.

e) Industrializing the indicators extraction (audio features and text data)

As I previously did to download each audio folders, I industrialized the indicators extraction through a unique python script. It ran every folder and wrote the indicators for each audio files line by line. This step is more detailed in my answer of the first assignment question.



```
import pandas as pd
```

```
output_df = pd.read_csv('/Users/kevenlemoing/Sites/sandvik_code_assignement/data/indicators.csv',error_bad_lines=False)
output_df
```

ard_deviation	median_frequency	first_quantile	third_quantile	inter_quantile_range	kurtosis	skewness	speaker_name	speaker_gender	speak
4429	368.604505	289.010060	328.807282	99.493056	-2.0	0.000000e+00	1028	Male	Adult
6456	294.958360	227.525005	261.241683	84.291694	-2.0	4.859547e-16	1028	Male	Adult
478	337.401379	297.795209	317.598294	49.507712	-2.0	0.000000e+00	1028	Male	Adult
2691	347.713169	270.427704	309.070436	96.606831	-2.0	0.000000e+00	1028	Male	Adult
5835	372.315279	283.934944	328.125112	110.475419	-2.0	0.000000e+00	1028	Male	Adult
110	357.436852	338.524795	347.980823	23.640071	-2.0	-3.648629e-15	1028	Male	Adult
0367	431.680318	312.139467	371.909893	149.426065	-2.0	0.000000e+00	1028	Male	Adult
8664	391.290428	316.337865	353.814147	93.690704	-2.0	-9.200978e-16	1028	Male	Adult
1986	257.910903	156.471070	207.190986	126.799792	-2.0	3.426145e-16	1028	Male	Adult
145	360.712349	310.231065	335.471707	63.101605	-2.0	1.332069e-15	1028	Male	Adult

f) Classification model implementation

Predictive models are only implemented on jupyter notebooks (cf. /notebooks/models).
I have tested features impact through three common classifiers such as Random forest, SVM and GaussianNB.
We can also see how it's important to launch supervised learning functions on clustered data: results are more successful if the model is trained over records in the same age range and in the same language.

3 – Assignment's questions

<u>How did you go about extracting features from the raw data?</u>
→ Being unfamiliar with audio data analysis, I focused my features extraction on the list provided in the instruction file. I found the python package "thinkx" for using the library "thinkdsp" and "thinkplot". These tools were very useful to calculate and visualize features based on frequency.
Regarding the huge amount of data, I noticed that I should separate the audio analysis models and the features extraction because it will not be possible to keep all of my data sets in live memory (my personal laptop is about 8Go RAM).
That is why, after scraping the VoxForge repository, I firstly decided to build a dataset with all indicators needed for the models' implementation (cf indicators_calculation.py).

The goal was to get all relevant information for each record: audio features and text information (contained in the folder /etc).
In order to optimize the use of resources, I built the "indicators" dataset line by line. In every folder downloaded, the script gets the following list of properties for each record.

Audio features:

- Mean frequency
- Standard deviation
- Median Frequency
- First quantile of the frequency
- Third quantile of the frequency
- Inter quantile range of the frequency
- Kurtosis of the frequency
- Skewness of the frequency

No-audio indicators:

- Folder
- Name of the record
- Speaker's name
- Speaker's gender
- Speaker's age range
- Speaker's language
- Speaker's dialect
- Sampling rate
- Sampling rate format

Then, predictive models can be trained without loading all audio files but just by reading a data frame of indicators needed.

Although this approach requires few in-memory resources, it takes a very long time to be done and we will see how to improve it in the fourth part of this report.

Which features do you believe contain relevant information?
→ Standard deviation and inter-quantile range.

How did you decide which features matter most?
→ I proceeded in two steps to draw this important conclusion.
First of all, I tried to identify which features were the most related to the gender just by looking at the data frame built through the script "indicators_calculation.py".
Then, the second step was to verify my assumptions by using a predictive model.

Although some features could matter most, it is important to notice that the feature selection is not enough to get the higher percentage of success with classification models. The data organization is also important, and results can be improved significantly by processing clusters of related data.
For instance, I got the best results when the model was trained on records registered by speakers with the same age range, same language and same dialect.

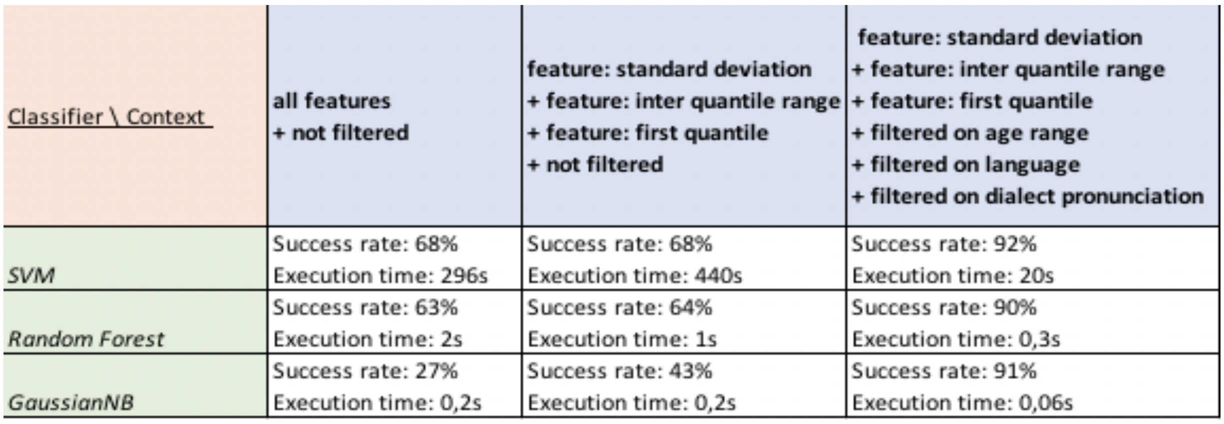
<u>Do any features contain similar information content?</u>
→ Yes, I obtained exactly the same values for mean and median frequency for a specific audio file and same kurtosis results for each record.

<u>Are there any insights about the features that you didn't expect? If so, what are they?</u>
→ As discussed above, I was surprised by the equality between mean and median frequency and always getting the same result for kurtosis calculations.
Neither did I expect the error with the mode frequency calculation (cf. unit_exploring_and__features_extraction.ipynb). After getting more information about this error, I realized that it makes sense because only discrete and continuous distributions have a mode at all times.

<u>Are there any other (potential) issues with the features you've chosen? If so, what are they?</u>
→ I did not have any other specific issues with the features. Being unfamiliar with audio data, I spent some time refreshing my knowledge of features mentioned in the instructions and found a useful python library which enabled me to easily get the frequency.

<u>Which goodness of fit metrics have you chosen, and what do they tell you about the model(s) performance?</u>
→ As explained above, after choosing standard deviation, inter quantile range and first quantile, I noticed that the model performance is highly increased if audio records processed have some common properties, especially the speaker's age, language and pronunciation.

<u>Which model performs best?</u>
→ As you can see in the directory "*/notebooks/models/*", the classifier which provide the best success rate on our dataset is SVM with 92% of success rate. It's also the slower with more than 20 seconds of execution on a reduced data frame (best results are obtained after filtering records on speakers' age range, language and dialect).



<u>Classifier \ Context</u>	**all features** **+ not filtered**	**feature: standard deviation** **+ feature: inter quantile range** **+ feature: first quantile** **+ not filtered**	**feature: standard deviation** **+ feature: inter quantile range** **+ feature: first quantile** **+ filtered on age range** **+ filtered on language** **+ filtered on dialect pronunciation**
SVM	Success rate: 68% Execution time: 296s	Success rate: 68% Execution time: 440s	Success rate: 92% Execution time: 20s
Random Forest	Success rate: 63% Execution time: 2s	Success rate: 64% Execution time: 1s	Success rate: 90% Execution time: 0,3s
GaussianNB	Success rate: 27% Execution time: 0,2s	Success rate: 43% Execution time: 0,2s	Success rate: 91% Execution time: 0,06s

How would you decide between using a more sophisticated model versus a less complicated one?
→ My personal approach was to firstly test a less complicated model. I decided to conduct the analysis this way to get results, and thereafter increase my knowledge of audio data as soon as possible.

What kind of benefits do you think your model(s) could have as part of an enterprise application or service?
→ A model which predicts the gender of the speaker's voice could be a part of a larger program or service. I think it should be useful to help customer application services which have already identified gender specific behavior and interests. We can merge the gender recognition with gender habits to automatize shopping recommendations by gender.

For instance, we assume that the application services have already built a program which detects that males are more interested in products A and B and females in C and D. It is used in the context of a mall, or some other market place where there is a lot of product diversification.
Once somebody calls the customer services to get information or any assistance, the merge between our model and the existing program could automatically attach recommendations to the client profile (registered by the support staff during their conversation).

4 – Improvements points

a) *Coding style*

- ❖ The best approach is more like coding small independent and callable functions. My scripts need to be more divided in functions that I should unit test.
- ❖ Manage exceptions → many exceptions are still managed by hand, especially for getting audio file names in the "*indicators_calculation.py*" script.

b) *Calculations time*

Although my coding style is perfectible, calculation time is an issue which needs to be specifically addressed. All of my computations were launched only under one thread while I could use four. There are two ways to implement the multi-threading approach:

- ❖ Using the famous python library called "thread"
- ❖ Use Spark and adapt the code to its own framework called "Pyspark"

c) Additional audio features

After getting more information about audio analysis, I found some other “classical” features which I did not consider as indicators. Please find the list below:

- Time domain features:
  - RMSE
  - WaveForm

- Frequency domain feature:
  - Amplitude of individuals frequency

- Perceptual feature:
  - MFCC

- Windowing feature:
  - Hamming distances of windows