MVA 2020 - Algorithms for speech and natural language processing (TP1)

Ariane ALIX

February 17th, 2020

---

1 Classification of segmented voice command

Question 1.1

We tested the given Neural Network model on the validation set with different freuency ranges in the MFCC. We must note that a maximum frequency greater then 8kHz returns an error, which is logical since we would not respect the Nyquist-Shannon conditions given the 16kHz sampling rate. The results are given in the following figure :

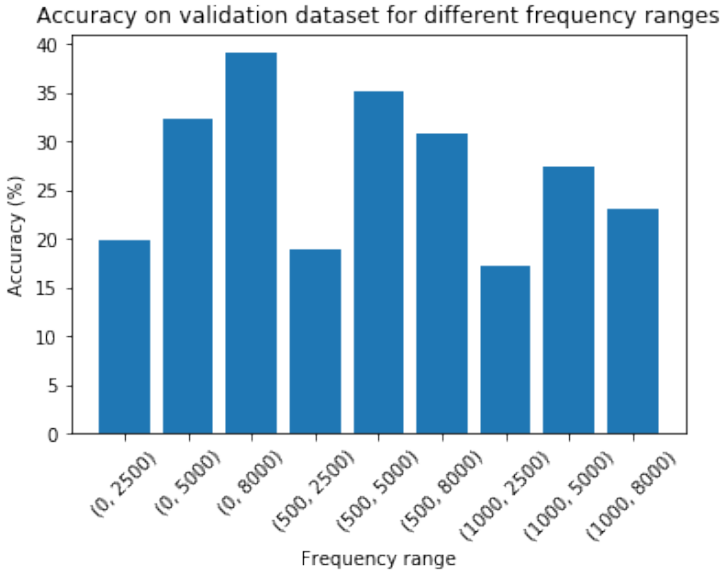


Figure 1: Accuracy of command recognition with a Neural Network and MFCC

We note that the performances are bad when we do not look above 2500Hz, which makes sense since English is a non-tonal language for which consonant are really important, and their frequency is typically between 2kHz and 4kHz. The best results are obtained by using the full range available (0-8000hZ), which seems reasonable.

Question 1.2

We tested the Log Regression model for different number of filters for the Mel-log filterbanks and with a frequency range of 0-8000Hz (which gave the best results on the previous question. Since the results are quite variable from one try to the other, we ran the Log Reg 5 times for each Mel-Log filterbanks and compute the average accuracy obtained on the validation set:

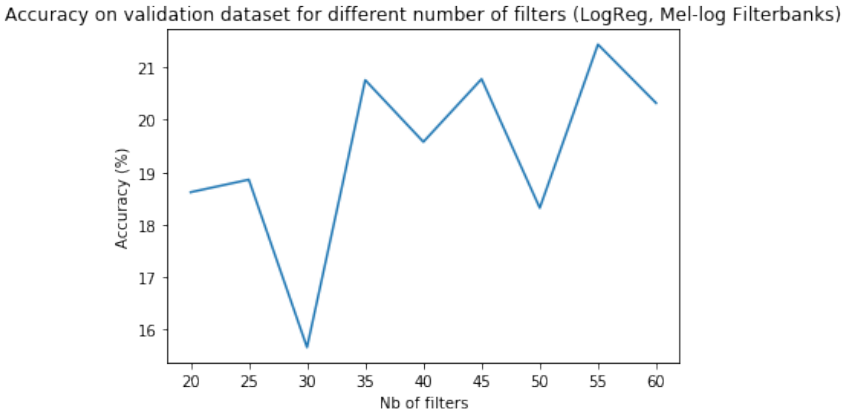


Figure 2: Accuracy of command recognition with a Log Regression and Mel-log Filterbanks

We then tested the Neural Network model for different number of cepstral coefficients for the MFCC and with a frequency range of 0-8000Hz.

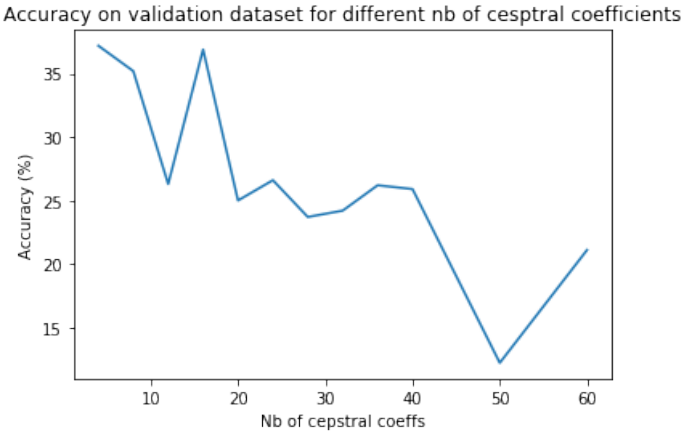
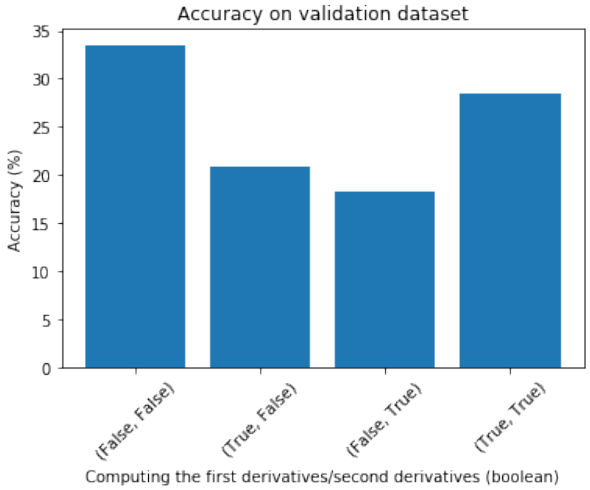


Figure 3: Accuracy of command recognition with a Neural Network and MFCC

We notice that increasing the number of cepstral coefficients does not increase the performances, on the contrary. This is due to the fact that we typically keep coefficeints 2 to 13 (the first one corresponding to the loudness), and that the other coefficients represent fine details like the pitch that do not contribute to Speech Recognition.

Question 1.3

Keeping 12 cepstral coefficients and the 0-8000Hz frequency range, we tested the Neural Network with MFCC with or without the computations of its first and second derivatives:



The highest accuracy is reached without any derivative. However, in general the first and second derivatives bring additionnal information for the speech recognition : they represent the context and dynamic information between frames. We will therefore keep the computation of the derivatives for the rest of the project.

Question 1.4

We tested new techniques to try and improve the previous performances. We first tried to change the batch size (the original one was 200) and to normalize the data either per channel or across channels. The best accuracy reached was of 59.1%, obtained for a batch size of 900 and a normalization per channel. The accuracies obtained for all the combinations tested are summarized in the heatmap of Figure 4.

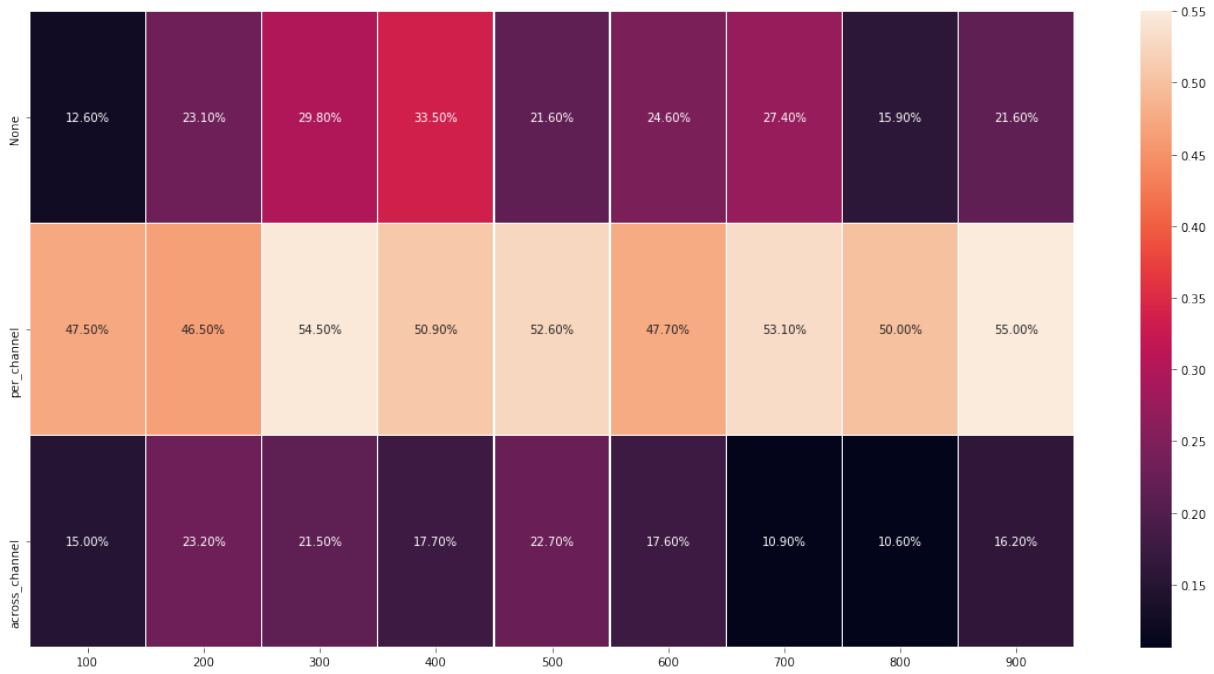


Figure 4: Accuracy of command recognition with a Neural Network and MFCC, 12 cepstral coefficients; x-axis: batch size; y-axis: normalization

Keeping the batch size of 900 and the normalization per channel, we then tried to do data augmentation by adding noise. To that end, we used the noises available in the dataset (dishes, bike, white noise etc.). For each sound in the training set, we added a new sound made by additioning the speech command and one of the available noises chosen randomly and with a random offset. We augmented the dataset 20 times with different noise additions, and We kept the best augmented dataset for the rest of the tests (which gave 69.2% on validation).

Question 1.5

Here, while keeping the same pre-processing and MFCC features as before, we tried to test different hyper-parameters and models to try and get better results on the validation.

Using a GridSearch, we looked for the best hyper-parameters for the 1-layer Neural Network. The best parameters were *{'activation': 'relu', 'alpha': 0.0001, 'batch_size': 900, 'hidden_layer_sizes': (300,), 'learning_rate': 'adaptive', 'learning_rate_init': 0.01, 'max_iter': 500, 'solver': 'lbfgs' }* and gave a 100% score on training and 70.4% on validation.

We then tried to build a Convolutional Neural Network (CNN) to try and improve the performances. With a network made of 4 convolutionnal layers and some batching (see code), we could reach a 84.4% on the validation set in 72 seconds, making it the best model by far.

Question 1.6

The best model is the CNN described in the previous question, with the MFCC features followed by a noise addition for data augmentation and a normalization per feature. We therefore tried it on the test set, and obtained an accuracy of 85%.

2 Classification of segmented voice commands

Question 2.1

The following lines of code assumed that we had the same number of examples for each command, hence it approximated the prior probability to be equal for all words.

```
elif train_labels.count(label) < nb_ex_per_class:
        fs, waveform = wav.read(full_name)
        train_wavs.append(waveform)
```

Question 2.2

According to the definition of WER, we have that WER $= \frac{S+D+I}{N} \geq 0$ since S, D, I and N are positive. Moreover, the predictions can return any number of words, so WER > 100 is possible (for example if we return N+1 words with none of them matching the references, we get $S + D + I = N + 1$ hence WER > 100).

Question 2.3

For this part we use the best performing model of the previous part: the Convolutional Neural Network already trained on the independant commands.

The true sentence is *go marvin one right stop*, and the one predicted with the greedy model is *go marvin one up stop*. The true sentence is obtained by substituting the newt-to-last word. Hence we have S = 1, D = 0, I = 0 and N = 5, so WER = 20%.

Question 2.4

Using the chain rule we have the formula :

$$P(W_1, ..., W_T) = P(W_1) \sum_{i=1}^{T-1} P(W_{i+1}|W_1, ..., W_i)$$

With the Bigram approximation where we only consider the previous word, the formula becomes :

$$P(W_1, ..., W_T) = P(W_1) \sum_{i=1}^{T-1} P(W_{i+1}|W_i)$$

Question 2.5

We chose to us a Bigram model which takes into account the relationship between words. Indeed, it is simpler than the N-gram models with larger Ns, and in our case it is probably sufficient since the sentences are short.

Question 2.6

N-grams with larger Ns work better on longer sentences which showcase more complex dependance between their words. Indeed in complex and natural speech, they help guessing each pronounced word by taking into account its context in the sentence, on a less local fashion than a Bigram. The main issue is that the computational complexity increases greatly with larger Ns since we need to represent all the possible transitions in an N-gram.

Since we will use a Bigram model, we computed the Bigram transition matrix of the words in the given sentences. The Figure 5 on the next page is a visualization of those transition probabilities.

Question 2.7

Let S be the length of the considered sentence, V the size of the vocabulaty (30 in our case) and k the beam size (5 by default). Then the time complexity is $O(kSVlog(kV))$ and the spatial complexity is $O(kV)$.

Question 2.8

Let $p(j, k)$ be the probability to be at state j at state k. We have the relationship:

$$p(j, k) = \max_i \{p(i, k-1)p_t(j|i)p_p(j|k)\}$$

Where $p_t(j|i)$ is the transition probability between i and j, and $p_p(j|k)$ the probability of word j given state k (hence the acoustic interpretation).

Thus the complexity of the Viterbi algorithm is $O(SV^2)$.

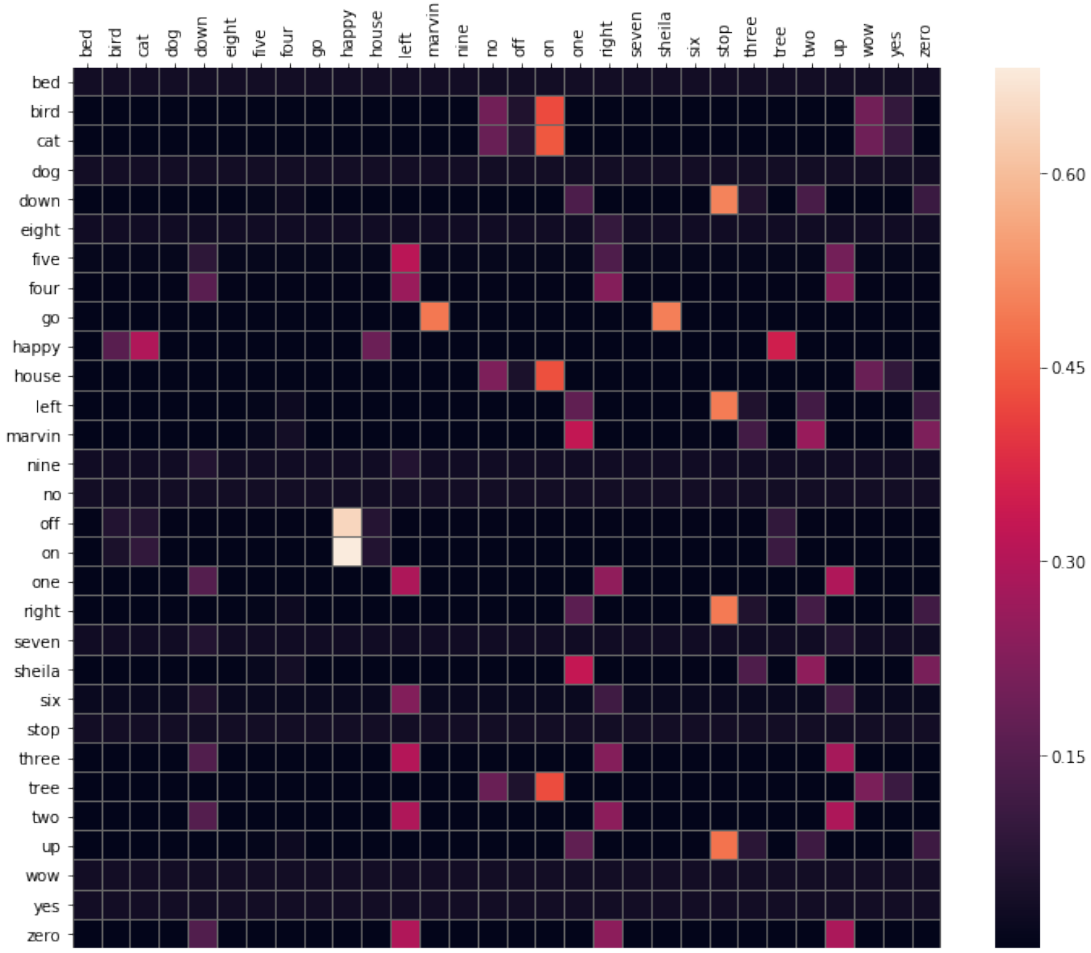


Figure 5: Bigram transition matrix

Question 2.9

In our case we obtained the results:

	Algorithms		
Set	Greedy	Beam Search	Viterbi
Train subset	29.32%	15.13%	19.69%
Test set	27.94%	14.22%	20.97%

Table 1: WER of command recognition with different algorithms

As expected, using a language model improves the performances. The Viterbi algorithm is less performant than the Beam search in our case, but it is 1.5 times faster. Here we used a Bigram model, but for more complex cases with more vocabulary the peformances might might with larger N-grams.

Question 2.10

We obtain some systematic errors due to the facts that some combinations of 2 words (bigrams) are not present or not well represented in the training set, and therefore cannot be predicted for the test set when we use a language model (very low probability in the transition matrix).

Below are some examples of uncommon bigrams, and the prediction errors for the Beam Search caused by this issue:

```
Uncommon bigram in training: ['left', 'five'] (Proba = 0.67%)
- Reference: ['go', 'marvin', 'zero', 'left', 'five', 'left', 'zero', 'left', 'stop']
- Prediction: ['go', 'marvin', 'zero', 'up', 'stop', 'left', 'stop', 'left', 'stop']

Uncommon bigram in training: ['up', 'five'] (Proba = 0.69%)
- Reference: ['go', 'marvin', 'zero', 'right', 'two', 'up', 'five', 'right', 'three', 'down', 'stop']
- Prediction: ['go', 'marvin', 'zero', 'left', 'two', 'up', 'one', 'left', 'three', 'left', 'stop']

Uncommon bigram in training: ['left', 'four'] (Proba = 2.27%)
- Reference: ['go', 'sheila', 'three', 'left', 'four', 'down', 'stop']
- Prediction: ['go', 'sheila', 'three', 'left', 'stop', 'down', 'stop']

Uncommon bigram in training: ['right', 'five'] (Proba = 0.82%)
- Reference: ['go', 'sheila', 'one', 'right', 'five', 'up', 'two', 'up', 'stop']
- Prediction: ['go', 'sheila', 'one', 'right', 'one', 'up', 'two', 'up', 'stop']

Uncommon bigram in training: ['left', 'four'] (Proba = 2.27%)
- Reference: ['go', 'marvin', 'two', 'down', 'zero', 'left', 'four', 'right', 'stop']
- Prediction: ['go', 'marvin', 'two', 'down', 'zero', 'left', 'stop', 'right', 'stop']
```

Figure 6: Systematic errors due to uncommon bigrams

Question 2.11

We tried a type of smoothing on the transition probabilities based on a sigmoid on the differences with 1/30 (the balanced assumption). Thanks to this method, the WER of the Beam search decreased to 13.85% on the training set and 13.73% on the test set.

Question 2.12

It should be possible to optimize jointly an acoustic and a language model with a Recurrent Neural Network. The idea would be to use the audio signals as the inputs of an acoustic model, then outputs the transition probabilities that would be used as inputs for the language model.