Speech Recognition assignment: MVA 2020

1 Classification of segmented voice commands: Part 1

In this section we will use our own architecture of the neural network from *keras* instead of the MLP from scikit learn, in order to use the Google Collab's GPU and accelerate the tuning of parameters. (Please refer to notebook for further details about the settings).
-**The dataset:**The Training set is a balanced set with 30 categories and 30,000 samples. For the validation and test sets we only have few classes present in a very unbalanced way (Appendix A).

1.1 Influence of the frequency range

We set the following ranges:
- `Lower_bound = [0,20,50,100]`
- `Higher_bound =[5000,7000,8000]`
We chose these bound in order to be on the same range as the frequency range of audible sounds. The best bounds are **100** and **8000** with a validation accuracy of **75.8%** (Appendix B).

1.2 influence of the number of Cepstrals and number of filters

1.2.1 MFCC

The best classifier uses only 9 Cepstrals (Appendix C)compared to 13 cepstrals mentionned in the lecture because the detailed changes in further cepstrals does not contribute to improving the results since we have very short sequences in the vocabulary.

1.2.2 Melfbanks

The table in Appendix D shows that melfilter banks are less performing than MFCC and the reason behind this drop of performance may be caused by the choice of CNN classifier and the correlations in the dataset.

1.3 Influence of Deltas

The best results (Appendix E) are obtained by **activating** $\Delta\Delta$ which is referred to as the acceleration term and it gives richer information about the frames context which explains the improvement of the performance (up to 71.6% of test accuracy).

1.4 Influence of Normalization

We only applied feature normalization on the 3 datatsets and it surprisingly decreased the performance of our classifier.

Configuration	Val. Acc.
With feature normalization	70.6%
Without feature normalization	**77.2**%

1.5 Influence of adding noise with data augmentation

In this part, we chose random noise addition, shifting and time-stretching as data augmentation techniques[1] by using the `librosa library`. Due to GPU and memory constraints, we couldn't perform the Data augmentation on all the training samples and we only trained the model on 32 000 training samples. The results in Appendix F shows that data augmentation improves the performance. In the next steps we will use the non-augmented dataset with the best configuration so far, due to material constraints.

1.6 Influence of model choice: CNN, SGD and LightGBM

The table in Appendix G compares the results using different algorithms using the MFCC filter. The best performing algorithm is the **CNN** without normalization with an accuracy of **80.6%** which outperforms the **Logistic regression** and **LightGBM** classifiers due to the multiplicity of Covolutional layers and the dropout coefficient of 0.3 that we used.

2 Classification of segmented voice commands: Part 2

2.1 Prediction of Sequences

Question 2.1 We considered that $P(W_i|X_i)P(X_i|W_i)P(W_i)$ and we also approxi-

[1] https://medium.com/@alibugra/audio-data-augmentation-f26d716eee66

mated
$P(X_i|W_i) \propto P_{\text{discriminator single word}}(W_i|X_i)$, which means that we considered that $p(W_i)$ are equal.

Question 2.2

We have $\boxed{WER = 100\frac{D+I+S}{N}}$ with S is the total number of substitutions, D is the total number of deletions, I is the total number of insertions,N is the number of words in the reference sentence.
- WER is a positive score as it is a sum of positive members, then **WER>0**.
- WER can be unbounded and therefore **WER may exceed 100**. In fact, the number of substitutions and insertions and deletions can randomly exceed the number of word in the reference sentence.

Question 2.3

In the example cited on the reference notebook we had these results:
- **True sentence:** `go marvin one right stop`
- **Predicted sentence** with greedy search: `go marvin one on stop`

Here $N = 5$, $S = 1$, $I = 0$ and $D = 0$ and therefore $\boxed{WER = 100\frac{1}{5} = 20\%}$ The same result as mentioned in the notebook.

2.2 Inject Language Models in the path scoring

Question 2.4: The bigram approximation formula is the following:

$$\boxed{p(w_i|w_{i-1}, w_{i-2}, ..w_1) \simeq p(w_i|w_{i-1})}$$

Question 2.5:

The bigram function returnd the transition matrix $M = M_{i,j}$ s.t $M_{i,j} = p(w_j|w_i)$ and approximated this probability with $\boxed{p(w_j|w_i) \simeq \frac{\#(w_i, w_j) + 1}{\#w_i + |corpus|}}$ as the common approximation ($\frac{\#(w_i,w_j)}{\#w_i}$) is not stable for unknown words (*Response to question 2.11*). We also visualized the adjacency matrix to check the accuracy of bigram algorithm and the most common sequences in the vocabulary are `('on','happy')` and `('happy','tree')` (Appendix H).

Question 2.6:
- **Advantages**: More accuracy in the results.
- **Disadvantages** of an increase in N: Huge space complexity due to memory storage of very spare matrices.

2.3 Beam Search

Question 2.7: For the algorithm that I implemented and if we denote k = beam size, C = size of corpus and L= commands length
-**Time complexity:** $O(kLC\log(C))$
-**Space complexity:** $O(kC)$

2.4 Viterbi decoder

Question 2.8:

Let's denote $p_k(j)$ the transition probability of having the word w_j at step k and $P(X_k|j')$ is given by the discriminator. Therefore, we have $\boxed{p_k(j) = max_{j'}(p_{k-1}(j')p(X_k|j'))}$
- **Time complexity:** $O(LC^2)$
- **Memory complexity:** $O(LC)$

2.5 Decoding results

Question 2.9: Using the **reference sentence**=`go marvin one right stop`, we got the following results with an equal performance for viterbi and greedy. However Beam search seems not to perform well.
- **Greedy** with WER=0.2 and predicted sentence =`go marvin one up stop`
- **Beam** with WER=0.4 and predicted sentence =`seven marvin one up stop`
- **Viterbi** with WER=0.2 and predicted sentence =`go marvin one up stop`

The same performance is noticed after training (Appendix I) and testing on a larger set. Viterbi is the most performing algorithm, while Beam is the fastest which corresponds to the theory.

Question 2.10: One systematic error while decoding is due to the problem of rare words that can add bias to Viterbi and beam-search results. Also these two algorithms does not take into account the words that are not included in the vocabulary.

Question 2.11: One strategy that i already used while calculating the bigrams is to add a smoothing while calculating the probability of occurrence of the words in any sequence. Results on the effect of the smoothing are in Appendix J

A Dataset distributions

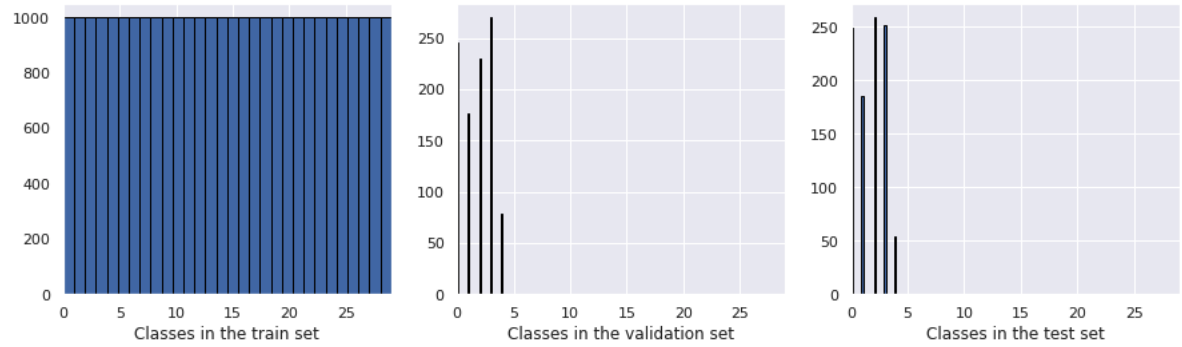

Figure 1: Unbalanced test and validation sets

B Influence of the frequency range: results

Ranges	0-5000	0-7000	0-8000	20-5000	20-7000	20-8000	50-5000	50-7000	100-7000	100-8000
Val. Acc.	66.1%	68.5%	71.0%	69.0%	66.7%	73.8%	71.9%	72.1%	75.7%	**75.8**%
Val. Err.	1.21	1.26	1.12	1.09	1.23	1.02	1.05	1.16	0.91	**0.88**

C Results with different Cepstrals coefficients

ncep	8	9	10	12	13
Val. Acc	71.4%	**77.2**%	76.1%	73.4%	73.7%
Val. Acc	1.08	**0.89**	0.97	1.10	1.09

D Results with different filters on Melfilter banks

n filter	10	20	50	60	80
Val. Acc	**65.9**%	65.1%	61.0%	53.8%	59.9%
Val. Acc	**1.30**	1.50	2.46	2.65	2.51

E Results on different delta configurations

Deltas Configuration	Val Acc.
Δ = F & $\Delta\Delta$ =T	**71.6**%
Δ = T & $\Delta\Delta$ =F	64.7%
Δ = T & $\Delta\Delta$ =T	70.9%
Δ = F & $\Delta\Delta$ =F	68.4%

F Results with Data Augmentation

Configuration	Val Acc.
With Data Augmentation	**79.7**%
Without Data Augmentation	77.2%

G Results on different classifiers

Model	Val. Acc.
CNN	**80.6%**
CNN + Normalization	70.6
Logistic regression	28.9%
Logistic regression + normalization	9.5%
LightGBM	56.2%
LightGBM + Normalization	45.7%

H Bigram results : heatmap

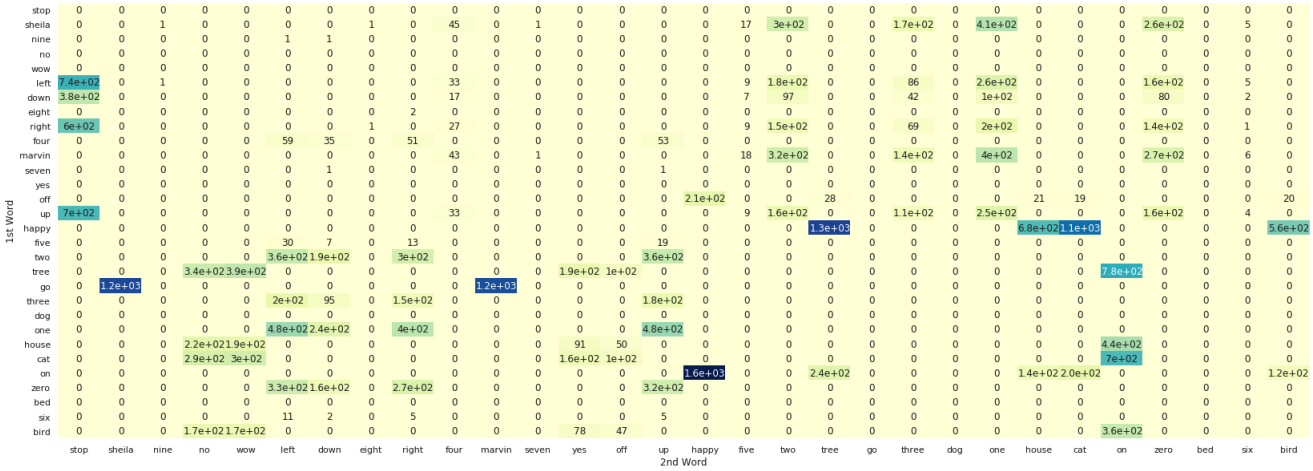

Figure 2: Bigram matrix representation

I Results with decoding approaches

Algorithm	test. WER	Eval. time
Greedy	0.25	54.3
Beam	0.79	**50.0**
Viterbi	**0.08**	56.4

J Smoothing on bigrams : solution for rare words

Configurations (+Viterbi)	test. WER
No smoothing and no normalization	0.074
smoothing and normalization	**0.08**