Speech and Natural Language Processing

Homework 1

Student :
Abdellah Kaissari

Professors :
Emmanuel Dupoux
Benoît Sagot

Part 1: Classification of segmented voice commands

In this Part, we will we will build a classifier to recognize voice commands recorded by devices such as Amazon Alexa or Google Echo assistants from the audio signal. We will study different steps for this task from feature extraction to building a prediction model.

The dataset is stored as waveforms, each one being a 1 second file containing one voice command. From these waveforms, we build a train dataset having 30 balanced classes with 1000 sample for each class. To find the best model, we have tried different models and different hyperparameters in the features extractions. We have used Melfilterbanks and MFCC as for feature extraction and two classifiers Logistic Regression and the MLPClassifier (a Neural Network imported from sklearn) with different hyper parameters.

On all the questions, we chose the neural network for the study. It takes less time and works better. And the metric used to measure the performance is the accuracy.

Question 1.1 : The influence of the frequency range for the MFCC

To search the best frequency range, we will try different values for lower bound and upper bound. In fact, MFCC is based on human hearing perceptions and humans can hear frequencies between 20Hz and 20 KHz and are most sensitive to frequencies between 2,000 and 5,000 Hz, in addition that the upper bound is limited to 8Khz because the signal is sampled on 16Khz. Then, we tried lower frequency (lowerf) in [0,100,200,300,400,500] and upper frequency (upperf) in [5000,5500,6000,6500,7000,7500,8000].

$\frac{upperf}{lowerf}$	5000	5500	6000	6500	7000	7500	8000
0	50.3	52.8	45.2	52.0	48.5	52.3	49.0
100	44.4	48.2	52.2	52.7	**54.1**	53.3	49.5
200	44.8	47.0	44.7	44.3	41.8	53.1	51.7
300	19.6	48.7	47.1	48.7	46.0	43.5	40.3
400	43.8	45.9	44.4	47.3	48.9	41.3	41.9
500	44.4	48.0	48.2	49.7	36.2	42.8	37.8

In next experiences, we will fix $lowerf = 100Hz$ and $upperf = 7000Hz$

Question 1.2: Influence of the number of filters

- **Mel-filterbanks:** We used only in this case the regression logistic to find the best number of filters. (the neural network does not work well with this features extraction)

nfilters	10	20	30	40	50
accuracy	14.8	**21**	12.7	14.2	20.7

- **MFCC:** we have tried first number of filters in [10,20,30,40,50] and after that we have tried for all integer between 10 and 30 and we obtained the following results:42.8 number of filters= 20 accuracy en val set: 54.1 number of filters= 30 accuracy en val set: 53.6 number of filters= 40 accuracy en val set: 48.3 number of filters= 50 accuracy en val set: 50.1

nfilters	10	20	30	40	50
accuracy	42.8	**54.1**	53.6	48.3	50.1

nfilters	11	12	13	14	15	16	17	18	19
accuracy	45.4	50.4	50.9	45.0	49.0	53.9	49.7	50.7	46.3

nfilters	21	22	23	24	25	26	27	28	29
accuracy	39.7	**58.5**	53.4	51.4	55.6	50.9	50.2	46.6	57.8

The MFCC works better than Mel-filterbanks as we have seen in the class. In the next experiences, we will use only MFCC with $nfilters = 22$.

Question 1.3: Influence of do_deltas and do_deltasdeltas

In this selection, we have tried all four possibilities:

- do_delta=True, do_deltasdeltas=True, accuracy= 44.2
- do_delta=True, do_deltasdeltas=False, accuracy= **58.5**
- do_delta=False, do_deltasdeltas=True, accuracy= 46.6
- do_delta=False, do_deltasdeltas=False, accuracy=47.3

We have found that for do_delta=True, do_deltasdeltas=False we have the best accuracy. These features are used to add information about the dynamics i.e the trajectories of the MFCC coefficients. It is different from what we have seen in the class. In the class, we have seen that in general the best model is with do_delta=True, do_deltasdeltas=True. We can explain that by two reasons. First, we use a neural network to these experience and since deltas and deltasdeltas is a linear transformation of the input the model can learn it on his own. And second, the fact that we have found the best lowerf, uppef and number of filters using (do_delta=True, do_deltasdeltas=False) then these configuration work well in this case, but if we had tried from the first with do_delta=True, do_deltasdeltas=True we would have different results. (We didn't do these experiences because it takes much time for each parameter)

Question 1.4: The influence of the other choices

Normalisation:

This choice is important in general and it improves the accuracy score. In this homework it improves the accuracy from 58.8 to **60.69**.

Training size:

Number of example per class	1000	2000	3000	4000	5000
accuracy	60.69	**69.6**	66.9	66.9	66.9

alpha:

alpha	0.5	0.6	0.7	0.8	0.9	0.97
accuracy	67.3	69.6	65.3	63.7	66.9	**73.0**

We have her the same alpha used in the literature. It improves the model very well.

wlen:

wlen	0.025	0.030	0.035
accuracy	68.1	66.4	**73.0**

ncep:

ncep	8	9	10	11	12	13	14	15
accuracy	**73.0**	67.80	68.89	68.0	71.7	69.1	63.9	63.2

It is different from what is expected (ncep=12-13 as seen in class). In fact, higher order coefficients represent increasing levels of spectral details but in this case we only need 8 coefficients. It is also can explained as for do_deltas and do_deltasdeltas. In fact, to have the best parameters we have to use the grid-search i.e try all combinations possible where each parameter has a list of choices predefined but it takes a huge time. In this study, we found each best parameter assuming a value for ncep in the first which is biased when we want to find the best ncep for example.

Adding noise:

Adding the noise we obtained an accuracy score of **75.9**. It doesn't improve the model, but with different architecture (we will use after) it will increase the accuracy .

Question 1.5: Test of different models

Regression logistic:

We have tried a classical regression logistic without fine-tuning the hyper-parameters like coefficient of normalization... We obtained an accuracy score of **34.0**. It' normal that it does not work as well as the MLPClassifier since all the study was done using the neural network. In addition, the logistic regression, in this case, takes too much time than the MLPClassifier and we don't have a lot of options like for the MLPClassifier.

MLPClassifier

We have tried different architecture and in each time we took the best accuracy on the validation set with respect to the epoch. For This training, we have used the data without noise. We used the sgd optimizer with a learning rate adaptive.

Architecture	(50,)	(100,)	(150,)	(200,)	(250,)	(300,)
Accuracy Score	70.5	76.7	76.3	77.2	76.7	**77.5**

Architecture	(300,50)	(300,100)	(300,150)	(300,150,50)
Accuracy Score	79.8	79.5	**80.0**	**81.3**

Finally, we trained the best model on the training set with noise to reach an accuracy of **84.1** on the validation set. We obtained this model after **35** epochs. **It is our best model.**

Question 1.6: Performance of the best model in the test dataset

We recall her the model obtained at the end. For the features extraction, we used **MFCC** with **number of example per class=2000** , **frequencies range = [100, 7000]**, **nfilters=22**, **dela=True**, **deltasdeltas=False**, **alpha=0.97**, **wlen=0.035**, **ncep=0.035** and we added a **noise** to the dataset.
The model used is the **MLPClassiefier** with 3 hidden layers **(300,150,50)** using the optimizer **SGD**, a learning rate **adaptive** with learning_rate_init=0.01 and **35** epochs.
we tested this model in a new test set to have an unbiased score and we reached in the test set an accuracy score of **82.0**.

Part 2 : Classification of segmented voice commands

In the second part of the practical work, we will use a new dataset composed of commands from the same dataset as the first part. This dataset is still composed of audio signal, but of variable lengths.

We denote by $X_{i=1...M}$ input sequence of spoken words, and the goal is to find the most likely sequence of words $W_{i=1...T}$.

$$\underset{W}{\operatorname{argmax}} P(W|X) \approx P(X|W)P(W)$$

Question 2.1:

The line in the code that approximated the prior probability of each word W_i to be equal is the following :

```
elif train_labels.count(label) < nb_ex_per_class:
```

where nb_ex_per_class is equal to 2000 for our case. Then, each word will be represented by 2000 samples.

Question 2.2:

The Word Error Rate is calculated as following $WER = 100.\frac{S+D+I}{N}$ with S is the total number of substitutions, D the total number of deletions, I the total number of insertions and N the number of words in the reference sentence.
The WER can not be negative, the minimum value which is the best is 0, when all the words are well recognized S=0, no word is omitted D=0, and no word is inserted I=0.
But, we can have a value greater than 100%, when for example, we have a lot of additional words inserted to the hypothesis sequence and no word is deleted.
In this study, there is no possibility of deleting or introducing words, we can only substitute words, then the WER can not then be larger than 100% in this case.

Question 2.3:

True sentence: *go marvin one right stop*
Predicted sentence: *go marvin one on stop*

The $WER = 0.2$ in this example. All the words are well predicted except *right* which is predicted *on*, there is one substitution $S = 1$, no word deleted $D = 0$ and no word inserted $I = 0$. So, $WER = \frac{1}{5} = 0.2$

Question 2.4:

We have the following formula for the language model for n words:

$$P(W_1, ...W_n) = P(W_1)P(W_2|W_1)P(W_3|W_2, W - 1)...P(W_n|W_{n-1}, ..., W_1)$$

The bigram model assume that each word in the sequence depends only on the previous word. We have then the formula:

$$P(W_1, ...W_n) = P(W_1)P(W_2|W_1)P(W_3|W_2)...P(W_n|W_{n-1})$$

Question 2.5:

To use the bi-gram algorithm for the language model, we first compute the bi-gram transition matrix where each coefficient $a_{i,j}$ of the matrix is the probability of the word W_j given the word W_i. This matrix has a shape of 30 x 30 because the size of the vocabulary used is 30. To implement that, we initialized the matrix with ones to avoid zero probabilities for rare and unseen words, then iterate all the sentences of the training corpus and add 1 for each coefficient $a_{i,j}$ when the word W_j come after the word W_i in a sentence. We normalized at the end to have probabilities.

Question 2.6:

The advantage of increasing N in the N-gram gives a better modelisation of the language which is logic, since we have in some sentence relations between distant words in the same sentence. And the drawbacks in increasing N implies the size of the transition matrix grows exponentially with the number of grams used and the size of the corpus for the approximation and the computing time become very huge.

Question 2.7:

We denote n the length of the sequences, h the number of size and B The beam size.
The Beam-Search algorithm implemented has a time complexity of $O(nhBlog(hB))$. In fact, we iterate the data sequence which cost $O(n)$ and sort a matrix of size Bxh which cost a complexity of $O(hBlog(hB)$ using the merge sort, then the complexity is $O(nhBlog(hB))$.
The space complexity is $O(Bh)$

Question 2.8:

In the virtebi algorithm, the formula used to find the most likely sequence is defined by (the formula demanded):

$$\begin{aligned} if \quad k=1, \quad & P(w_1=j) = P(x_1|w_1=j)P(w_1=j) \\ \forall k \geq 2 \quad & P(w_k=j) = \max_{w_1,\dots,w_{k-}} P(w_k=j, w_{k-1},\dots,w_1|x_1,\dots,x_k) \\ & = P(x_k|w_k=j)\max_{j}^{'} P(w_k|w_{k-1}=j^{'})P(w_{k-1}=j^{'}) \end{aligned}$$

with $P(x_k|w_k=j)(w_k=j|x_k)$ under the acoustic model approximation used.
The time complexity of this algorithm is $O(nh^2)$ and the space complexity is $O(nh)$ where n is the length of the sequence and h the number of the words.

Algorithm	Greedy	Beam-search	Viterbi algorithm
Subset Train WER	0.1086	0.0689	0.0689
Test WER	0.1029	0.0702	0.0691

Question 2.9:

We tested the three algorithms in a same subset of 300 examples from the training set and on all the test set and we found : In theory, The Viterbi algorithm works better than the Beam-search which works better than the Greedy algorithm. In fact the Beam-search expands all possible next steps and keeps the k (beam-size) most likely which is better than the Greedy which keeps only the first most likely (Larger beam-size gives better performance but take more time for decoding). And the Viterbi algorithm finds the maximum path, then the best score. The limitation of Viterbi is when the number of states is too large.
In this homework, we have the Viterbi works better than Beam-search with a very small difference of the WER and both outperform the greedy search. It is as expected since the vocabulary is small and then the number of states is small.

Question 2.10:

To find errors due to language model, we saw the worst ten WER score and analyse the difference between the predicted sequence and the reference. We have found these examples:
True sentences : *go marvin one left stop.* *cat off cat yes*
Predicted sentence : *tree marvin on left stop.* *stop off stop yes*
with $WER = 0.4$ $WER = 0.5$ respectively.
This errors is very probably due to the language model because there in no similarity between go and tree or stop and cat in pronunciation.
We found also some mispredictions due to the acoustic model like this example:
True sentences: *tree off tree wow*
Predicted sentence: *three off three wow*, with $WER = 0.5$.

Question 2.11:

To avoid assigning zero probability of rare seen words or sequences, especially for higher N (where many sequences can be rare) , we used a smoothing like in tfidf where we add 1 to all counts before normalizing into probabilities. In our case, we have initiated the transition matrix by ones instead of zeros.

Question 2.12:

To optimize jointly an acoustic model and language model, we can use and end-to-end recurrent neural network such that (LSTM, Bi-LSTM) which take into account the language model parameters of the features to make the prediction and optimizing these parameters.