Speech and Natural Language Processing: Practical Work I

Mohamed Kerroumi
mohamed.kerroumi@student.ecp.fr

1. Part 1

- **Question 1.1**: The influence of the frequency range, hyper-parameter for the MFCC, on the validation performance. Using the logistic regression model.

lowerf - higherf (KHz)	0.5 - 5	1 - 7	0 - 8
Validation Accuracy	6.1 % %	5.2 %	**10.7 %**

Table 1. Models Comparison

In the model, we used $framerate = 16KHz$, so by the Nyquist theorem, $f_{max} - f_{min} = 8KHz$, that's why in the two first cases, we got bad accuracy, because after filtering, the features aren't enough for the model.

- **Question 1.2**: The influence of the number of filters for the mel-log filterbanks using the logistic regression model.

nfit	15	35	50
Validation Accuracy	9.8 %	7.2 %	**21.0 %**

Table 2. Models Comparison

We saw in the class that features are vectors of length $n = number_of_filters$, hence the more filters we have, the more accurate the model is.
The influence of cepstral coefficient for the MFCC using the logistic regression model.

nCep	8	16	30
Validation Accuracy	29.1 %	26.9 %	**45.2 %**

Table 3. Models Comparison

Same remark as before, the more cepstral coefficients we use, the more accurate the model is.

- **Question 1.3**: The influence of the delta and delta_delta for the MFCC using the logistic regression model.
The validation accuracy is high when we useΔ, this is obvious because we include the first derivative in the features, so more information about the waveform,

Model	No Δ No $\Delta\Delta$	Δ	Δ and $\Delta\Delta$
Validation Accuracy	34.9 %	**45.2** %	23.2 %

Table 4. Models Comparison

however the accuracy drops in the case of Δ and Δ^2, this can be justified by the overfitting of the models to the training set, because the number of features becomes bigger.

- **Question 1.4:** the influence of the normalization (per-channel and across channel normalization) using the MLP Classifier.

Normalization	Without	Per-channel	Across channel
Validation Accuracy	67.1 %	14.6 %	**68.2** %

Table 5. Models Comparison

The across channel normalization increases slightly the validation accuracy, so we will work for the rest of the assignment with the across channel scaled data set.
Then, We increase the number of samples per class from 300 to 2000. This data augmentation allows us to reach **77.2%** in the validation accuracy.

- **Question 1.5**: The influence on the performance of the model choices and their hyper-parameters.
We used a model with **mfcc** and the MLP Classifier, and the hyper-parameters of the model are the following: $nfit = 20$, $nCep = 8$, $n\_ex\_per\_class = 2000$, then we picked different values for α the L2 penalty (regularization term) parameter, we report the performance of the models in the following table:

α	0.0001	0.05	0.1	0.2
Validation Accuracy	77.2 %	**82.5** %	80.5 %	76.1%

Table 6. Models Comparison

From the table, we deduce that Regularization increases the validation accuracy of the model, and that $\alpha = 0.05$ is the suitable value for the model.

- **Question 1.6**: The performance for the test set of the best configuration.

For the best model I found on the validation, it uses MLP Classifier with a regularization parameter $\alpha = 0.05$ and $hidden\_layer\_sizes = (200, 100)$, for the data set, I used 2000 samples per class scaled across channels, 20 filters and 8 cepstral coefficients. After the training , we found $Test\_accuracy = 79.3\%$, this value is close to the validation accuracy (82.5%).
The image below presents the confusion matrix on the test set:

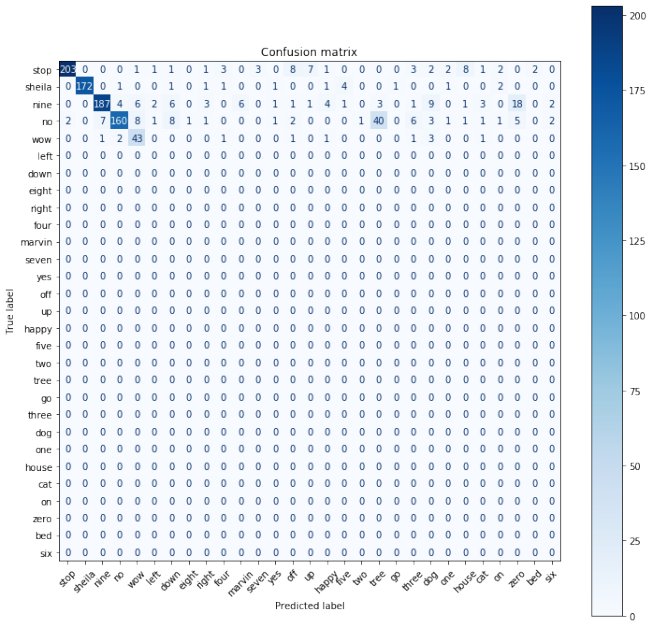

Figure 1. Confusion matrix of the best model.

From the figure above, we can see that the test set is unbalanced, and contains only the first 5 classes. Most of the mistakes are made on the word 'on' , it's confused with the word 'tree', which is weird because the pronunciation of the two words is very different.

2. Part 2

- the prior probability of each word W_i is a constant because we trained our discriminator with a balanced data set (same number of examples per class). The lines of code that do that are :
  elif train_labels.count(label) < nb_ex_per_class:
  fs, waveform = wav.read(full_name)
  train_wavs.append(waveform)
  train_labels.append(label)

- **Question 2.2**: It isn't possible that $WER < 0$ because $WER = 100.\frac{S+D+I}{N}\%$ and $S \geq 0$,$D \geq 0$ and $I \geq 0$ .However it's possible that $WER > 100$ for example, if we substitute all the correct words and we insert wrong words, $S = N$ and $D \geq 0$ and $I > 0$, so $WER > 100$.

- **Question 2.3:** Since each input is matched with one single output, there's no alignement problem, so $D = 0$ and $I = 0$. In the assignement we considered the True sentence: **go marvin one right stop**, the greedy search predicts, **go marvin one right stop**, so the decoder didn't make any substitution mistake , so $S = 0$ and $N = 5$, hence $WER : 0$.

- **Question 2.4**: The Bigram approximation formula of the language model is :

$$\mathbb{P}(w_i|w_{i-1}, w_{i-2}..., w_1) = \mathbb{P}(w_i|w_{i-1})$$

- **Question 2.5:** Implementation choices.
  I added to the labels set a new class , $< stop >$, I insert this new word in the beginning of each sentence, in order to have an approximation of the probability of beginning a sentence with a given word. I implemented only the BiGram language model such as $BiGram[i, j]$ approximates $\mathbb{P}(w_j|w_i)$, **BiGram** is defined by $BiGram[i, j] = \frac{\#(i,j)}{\#i}$, but for the bigram (i, j) that does not appear in the training samples, this definition raise a problem, So I used 1-Laplace smoothing, which adds 1 to every bigram count. and the estimation of the probability becomes: $BiGram[i, j] = \frac{1+\#(i,j)}{\#i+\#labels\_set}$

- **Question 2.6**: If we increase N, we will end up with more realistic model that capture longer-time dependencies, however the time and memory complexity will increase exponentially.

- **Question 2.7:** The time complexity of the Beam search algorithm I implemented is:

$$\mathcal{O}(B \times N \times T \times \log(B \times N))$$

where, B :the beam size.
N : The vocabulary size.
T : Lenght of the sentence.
And for the memory complexity, it's $\mathcal{O}(B \times N)$.

- **Question 2.8**: Let $q_{t,s}$ denotes the probability to be in state s at step t, Then :

$$q_{t,s} = \max_{s'} q_{t-1,s'} \times \mathbb{P}(s|s') \times \mathbb{P}(X_t|s)$$

Where $\mathbb{P}(s|s')$ is given by the BiGram transition matrix, and $\mathbb{P}(X_t|s)$ is given by the Discriminator.
The time complexity of the Viterbi algorithm is $\mathcal{O}(T \times N^2)$.
The memory complexity is $\mathcal{O}(T)$.

algorithm	Greedy	Beam search	Viterbi
Test WER	16.99 %	3.75 %	**2.55 %**
Evaluation time(s)	56.49	**55.08**	55.15

Table 7. decoding algorithms Comparison

- **Question 2.9**:
  In theory Viterbi algorithm is the best, because it explores all the possible sequences and outputs the one that has the best score. Indeed this is the case of our decoder, the Viterbi algorithm gives the smallest WER (see table above), however the Beam search gives also a very low WER without exploring all the sequences. Hence the Viterbi algorithm gives all the time the best result, but in the case of data set with large number of classes, Beam search is the appropriate algorithm because of its small time complexity.

- **Question 2.10**: If we consider the : *go sheila one down stop*, the Viterbi and beam search decoders both output the sentence: *go right one down stop*. So here the language model found that the positional word *right* is more probable to be after the word *go*, which distorts the output of the decoder.

- **Question 2.11**: to face rare seen words(or sequence of words), I used Laplace smoothing when implementing the BiGram transition matrix.