1 Classification of single voice commands

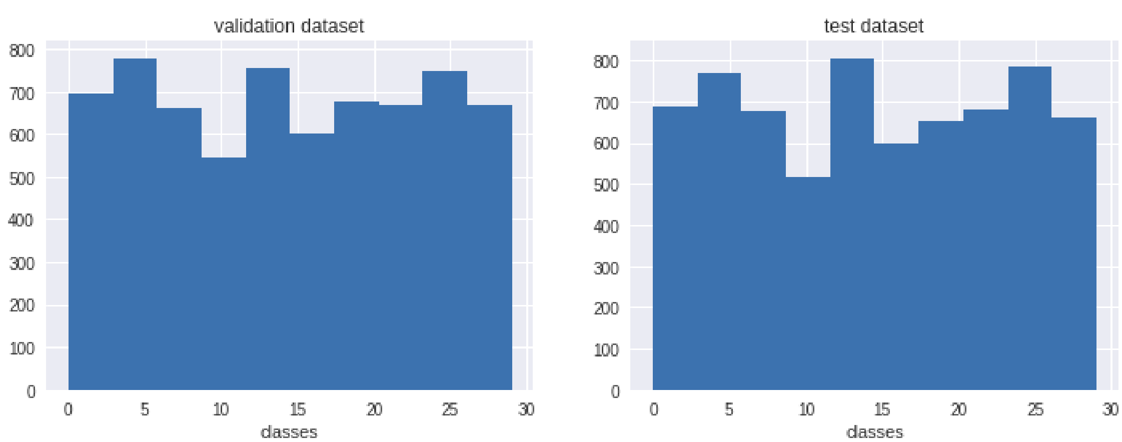
This part of the work embodies two important parts: *features extraction* and *the predictive model setup*. We conduct two groups of experiments. In the first bunch, (i) we keep unchanged the parameters of MFCC and MeltFilterBank (ii) and deploy two kinds of models, i.e. a logit and a simple MLP from shallower to deeper architectures. Actually, we didn't spend too much time on the logistic regression since neural networks have higher capacity[1]. Every network was trained on 100 epochs, while keeping the best peforming parameters on the validation dataset with a stochastic gradient descent algorithm, a learning rate equal to 0.01 and categorical crossentropy as loss.

The results of first experiments are reported in the table 1 below. In this table MLP$(c_1, ..., c_h)$ stands for a Multi-Layer-Perceptron with h hidden layers (except output layer) with respectively $c_1, ..., c_h$ neurons. We see that it is expedient to use deeper neural architectures since it increases the performances both on training and validation datasets. Moreover the neural networks are far better than the results of Logit Model. However, this conclusion should be taken cautiously since the validation data was unbalanced unlike the training dataset.

	meth.	feats.	train	val	meth.	feats	train	val
Logit	MFCC	1616	11.06	5.4	Mel.	2020	11.63	**27.49**
MLP(100)	MFCC	1616	42.69	**47.09**	Mel.	2020	9.30	13.14
MLP(200)	MFCC	1616	56.69	**55.6**	MFCC	2020	21.14	38.6
MLP(200, 100)	MFCC	1616	59.91	**59.3**	MFCC	2020	44.54	48.40
MLP(200, 100, 100)	MFCC	1616	57.80	**61.3**	MFCC	2020	59.87	54.3

Table 1: Outcome of first experiments

Figure 1: train/dev results w.r.t. the number of epochs



[1]More precisely a logistic regression is a simple network without hidden layers using a softmax/sigmoid activation function.

Like we said, despite the fact that the training dataset was balanced, the validation and test sets weren't, only five labels out of thirty were present[2]. Yet for a neural network 9000 samples for training is too small, therefore in the secound round, we decided to (i) increase the training set size, (ii) increase and balance (to the best we can) the test and validation datasets. The distribution of the labels over these datasets is portrayed on figure 1 .We also modified the parameters of the filters used to extract the features. We content ourselves to take the values suggested by the authors of the library `python_speech_features`[3] as rules of thumb. Finally for MFCC and Melfiterbanks, we test the variants with and without deltas. We consider mainly a MLP$(200, 100, 100)$ denoted by MLPa and a MLP$(500, 300, 200, 100)$ regularized with a dropout of rate 0.5 at the third layer denoted by MLPb. Thus a a glance at table 2 suggests to use the Melfilterbansks withoud deltas with MLPb and MLPa. The performances of MLPa and MLPb on the test size are respectively around 69.96 and 74.48 [4]. Ultimately MLPb is especially strugling to recognize the sounds `go`, `no` and `dog` given their low F-scores equal to 0.505, 0.524 and 0.574. Beside them, sounds `five`, `down`, `on`, `one` recorded Fscore equal to 0.631, 0.639, 0.669, 0.688, while the other sounds do all have Fscores greater than 0.7.

Table 2: Outcomes of second round experiments

meth.	delta	feats.	model	train	val
Mel	No	2626	Logit	21.64	20.15
Mel	No	2626	MLPa	85.27	**71.27**
Mel	No	2626	MLPb	85.27	**74.27**
Mel	Yes	7878	Logit	24.65	21.40
Mel	Yes	7878	MLPa	86.29	70.46
MFCC	Yes	3939	Logit	21.85	19.55
MFCC	Yes	3939	MLPa	81.68	70.60
MFCC	No	1611	Logit	16.51	15.34
MFCC	No	1611	MLPa	81.62	70.56

2 Classification of segmented voice commands

Q2.1 By definition WER is always positive and cannot be strictly negative. On the other hand, WER can be strictly greater than 100. For instance if the prediction $w_1, w_2, ..., w_n$ is a permutation of the original sequence $W_1, W_2, ..., W_n$ without any fixed point, we have then WER $= 100$. Now if we replace for instance w_1 by any word w' who was not in the original sequence we get $S = n$, $D = 0$ and $I = 1$ so that WER $= 100 + \frac{100}{n} > 100$.
Q2.2 The line in the code above approximated the prior probability of each word W_i to be equal is: `posterior = model_predict_proba_function(features_input)`. It works

[2]For the valid dataset we got the following configuration: (**0**: 256), (**3**:246), (**2**:170), (**4**:166), (**1**:162). For exemple we got 170 observations with label **0**. And the test set had the following configuration: (**2**: 259), (**3**: 249), (**0**: 180), (**1**: 158), (**4**: 154).

[3]available at http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

[4]We run MLPa and MLPb two times and we report the average performance.The results for MLPa and MLPb were respectively (69.46, 70.46) and (74.63, 74.34).

because the classes have the same probability, otherwise we shoud have had a multiplication by the vector storing the words frequency.

Q2.3 The idea is to find the most likely sequence of words $W_1^*, W_2^*, ..., W_M^*$, given the extracted features $X_1, ..., X_M$. The trick is to operate individually on each feature, that is given a feature X_i, we want to find the word W_j which is its most likely causal source among the vocabulary $(W_1, W_2, ..., W_T)$ so that at each step we solve the problem:

$$W_i^* = \arg\max_{1 \leq j \leq T} \mathbb{P}(X_i \mid W_j) = \arg\max_{1 \leq j \leq T} \mathbb{P}(W_j \mid X_i)$$

The latter hold since $\mathbb{P}(W_j)$ is constant and Bayes Rule implied $\mathbb{P}(X_i \mid W_j) \propto \mathbb{P}(W_j \mid X_i)$.Thus we compute the WER on the true sequence $W_1, W_2, ..., W_M$ and the predicted sequence $W_1^*, W_2^*, ..., W_M^*$.

Some experiments are reported in table 3. For the test dataset the evaluation is done on the whole dataset, but for the training the evaluations are carried out on sampled sub-datasets of size 300. Overall MLPb is better than MLPa both on training and test datasets. In addition it seems to be more stable on the training dataset that its MLPa counterpart.

Table 3: WER performances (train vs test)

	train							test
Model	trial 1	trial 2	trial 3	trial 4	trial 5	mean	std	test
MLPa	0.312	0.314	0.351	0.354	0.326	0.331	0.017	0.303
MLPb	0.323	0.334	0.316	0.307	0.324	**0.320**	**0.008**	**0.279**

Q2.4 the Bigram approximation formula of the language model is:

$$\mathbb{P}(w_n \mid w_{n-1}, ..., w_2, w_1) = \mathbb{P}(w_n \mid w_{n-1})$$

Q2.5 Imagine that we are given a set of sentences $\mathbf{S} := (s_1, .., s_N)$ and a vocabulary $V = (w_1, w_2, ..., w_{N_W})$. We limit our implementations to bigrams, so that our purpose is to estimate the quantitities $\mathbb{P}(w_i)$ and $\mathbb{P}(w_i \mid w_j)$. So throughout the sentences we denote respectively by N_w, $N_{ww'}$ the counts for the word w and the bigram ww', and N_T the total number of words in the corpus $\mathbf{S}$. We also denote by $N_{w.}$ the counts of ford w in $\mathbf{S}$ except for appearances as a last word, that is $N_{w.} = \sum_{w'} N_{ww'}$. A simple estimator would be the use of global and conditional averages as below:

$$\hat{\mathbb{P}}(w) = \frac{N_w}{N_T} \text{ and, } \hat{\mathbb{P}}(w' \mid w) = \frac{N_{ww'}}{N_{w.}}$$

However there is two issues with the estimators mentioned above. If a word or a bigram has never been seen, it will have a probability equal to zero. In addition a probability wrongfully set to zero could cause systematic bias in search algorithms such as Viterbi and also numerical issues. Therefore we use a laplacian correction, where by default we respectively add α and β to the counts of words and n-grams. Accordingly our final estimators are given by:

$$\hat{\mathbb{P}}_\alpha(w) = \frac{N_w + \alpha}{N_T + \alpha N_W} \text{ and, } \hat{\mathbb{P}}_\beta(w' \mid w) = \frac{N_{ww'} + \beta}{N_{w.} + \beta N_W}$$

Q2.6 Greater values for N are more realistic and encapsulate a better long-term sequence

dependency. On the other hand, storage costs increase with N as well as never-seen n-grams.
Q2.7 See our beam-research code in the attcached jupyter notebook, the complexity of the algorithm is $O(T\Gamma N_W)$, where T is the maximal length of sequences, and Γ the beam size.
Q2.8 See our viterti algorithm code in the attcached jupyter notebook, The relationship between the probabilities is derived from Hamiton-Jacobi-Bellmann equation:

$$V_{k|j} = \max_{j'} \mathbb{P}(X_k \mid j)\mathbb{P}(j' \mid j)V_{k-1|j'}$$

The complexity of the algorithm is bound by $O(TN_W^2)$, where T is the maximal length of sequences.
Q2.9 A systematic error is that: a bigram who never appeared in the training corpus for the transition matrix would never be recognized. That is, if we have never seen the bigram $j'j$ then $\mathbb{P}(X_k \mid j)\mathbb{P}(j' \mid j)V_{k-1|j'} = 0$, so that both viterbi algorithm and beam search will never output $j'j$ as an output subsequence. For instance the bigram `(two,down)` roughly *never*[5] appeared in training set, and when we tried to reconstruct the 300-th sentence of the training set which is `"go sheila` **`two down`** `three right one left zero right stop"`, both viterbi algorithm and beam search output `"`**`no`**` sheila `**`go`**` `**`down`**` three right one left zero right stop"`. Another systematic error is that words who rarely appeared in the training corpus, will tend to be missed in the sentence reconstruction when they are at the beginning like the word `go` in the same example, we see that our algorithms output `no` instead. We checked and we see that $\mathbb{P}_{0.01}(\texttt{go}) \approx 4.6 \times 10^{-7}$.
Q2.10 Instead of arbitrarily setting the extra counts for words, we can design a more specific count augmenting strategy by *negative sampling*. However we did not implement it. Instead we just try a less rigorous back up strategy by trying to optimize the augmenting parameters α and β, While with negative sampling we will rather end up with a α_w and β_w specific to every word. So we test the effectiveness of our back-up strategy by fixing $\alpha = 25$ and varying β in the range $(1, 2, 3, 4, 5)$ for the testing dataset and using Viterbi algorithm. By doing so, the WER decreases both for MLPa and MLPb as it is portrayed by table 4. So optimizing in the simple case where $(\alpha, \beta) \in [25] \times [1, 2, 3, 4, 5]$, we get $\hat{\alpha} = 25$ and $\hat{\beta} = 5$. So widenning the range of (α, β) should lead to better results[6].

Table 4: The effect of β on WER for $\alpha = 25$ (Viterbi algorithm, test dataset)

	β				
Model	1	2	3	4	5
MLPa	0.3044	0.3011	0.2995	0.2984	**0.2982**
MLPb	0.2984	0.2832	0.2812	0.2798	**0.2785**

[5] Indeed $\mathbb{P}_{0.01}(\texttt{down} \mid \texttt{two}) \approx 8.10^{-6}$

[6] We also found similar results for α, when we fix $\beta = 1$.