Algorithms for speech and natural language processing: TD 1

Souhaib Attaiki
souhaib.attaiki@ens-paris-saclay.fr

1 Classification of single voice commands

The goal of this section is to train a classifier which, given a one second voice command, predicts the corresponding class, which corresponds to the spoken word. To do that, we will use the *Tensorflow* speech commands dataset.

1.1 Creation of the train/validation/test datasets

As the provided code contains a bug, which creates validation and test sets containing only 4 classes instead of 30, we rewrote the code for this part.

We loop through all the classes, and record all the audio files in a list. This list is then mixed randomly, and it is browsed to create the different dataset. This produces validation and test sets containing all classes, and which have the same number of elements on average.

In this following, we will test the effect of the different parameters of the feature functions, the regularization, the architecture of the classifiers, normalization and data augmentation on training the classifier of single voice commands. Our approach on choosing parameters will be based on recommended parameters in literature and using grid search.

Because we have limited computing power, and the different experiments took a long time to run, we cannot use a grid search on all the parameters of the pipeline (feature function + (regularization & architecture) + normalization + data augmentation), so we will make the assumption (for simplification) that the fine-tuned parameters of feature functions on logistic regression are the best parameters for the different classifiers, regularization and normalization, and they will be used later for fine tuning the other parameters (like the architecture of the MLP).

1.2 Fine tuning MFCC & mel-filterbanks parameters

For the fine tuning of the MFCC and mel-filterbanks, we will use a grid search coupled with the suggested parameters in the literature to limit the range of variation of the parameters.

As it mentioned in [1], **chapter 9.3**, a typical value for parameter α is between 0.9 and 1, a typical value for parameter **wlen** is 25 ms, a typical value for the frame shift is 10 ms, and finally, to respect the *shannon theorem*, **upperf** will be equal to framerate / 2.

After performing the grid search, these optimal values will be chosen:

- For mel-filterbanks:
  - nfilt = 40, as our results improved when we increased the number of filters
  - ncep = 0
  - lowerf = 20, lower frequency to have as much information as possible
  - upperf = 8000
  - alpha = 0.92, this value gives better results

  - frate = 100
  - wlen = 0.025
  - nfft = 512, standard value
  - compression = 'log', standard value
  - do_deltas = False
  - do_deltasdeltas = False

- For MFCC (same remarks as mel-filterbanks):
  - nfilt = 40
  - ncep = 13
  - lowerf = 20
  - upperf = 8000
  - alpha = 0.92
  - frate = 100
  - wlen = 0.025
  - nfft = 512
  - compression = 'log'
  - do_deltas = True
  - do_deltasdeltas = False (setting this to true gives a high dimension representation, which doesn't converge on my hardware)

1.3 Fine tuning regression

For the regression, we didn't use the *'sklearn.linear_model.LogisticRegression'*, as it take a long time to converge on my hardware (and sometimes it doesn't converge), and instead, we used *'sklearn.linear_model.SGDClassifier'* which is also a linear classifier.

To regularize it, we tried to change different parameters, as the *loss* type (hinge, log, etc), the penalty type (l2, l1), alpha, the constant that multiplies the regularization term, etc. After performing grid search, we find that the best parameters for MFCC give a validation score of **38.1** %, and the best parameters for mel-filterbanks give a validation score of **34.7** %.

1.4 Fine tuning MLP

For fine tuning the MLP, we tried different architectures (width/depth), different values for learning rates and the parameter alpha. Our best architecture is a 5 layer MLP, with a learning rate equal to 5×10^{-4}, and $\alpha = 10^{-3}$. This gave a validation score equal to **73.0** % for MFCC, and **64.3** % for mel-filterbanks.

It can be seen that MFCC features always give better results than mel-filterbanks. This result is not surprising for a language like English, as we have seen in class.

1.5 Feature normalization

In this section, we tried to perform a mean-variance feature normalization. This was done using the object *'StandardScaler'* provided by *sklearn*.

It was found that normalization does not improve results, either for logistic regression or for MLP. However, it has the effect of considerably reducing the training time, for example by a factor of 3 for MLP, which was expected.

1.6 Data augmentation

Data augmentation can be seen as another way of regularization. In order to create a noised sample, a noise wave is chosen randomly from the noises in '_background_noise_', and a random sample of 1 second is extracted from it and added to the original voice command.

This did not improve the results. This can be explained by the fact that the different parameters of feature functions and classifiers have been chosen using clean sounds. Another remark is that the data augmentation was not necessary in our case, because the latter is used when we have a limited dataset, whereas this is not the case. Our first limitation is a hardware limitation, otherwise, we could simply increase the number of examples per class (something we didn't do for the same reason).

1.7 Other classifiers

We tried to use other classifiers to improve our results. For this purpose, we tried to use the *'RandomForestClassifier'* and the *'XGBClassifier'*.

After a fine tuning of the *'RandomForestClassifier'* parameters, a validation score equal to **59.1** % is obtained. The particularity of this classifier is its fast convergence, unlike the *'XGBClassifier'*, which did not converge even after one hour of running.

1.8 Comments on the results

We tested our best classifier on the test set. This gave a score of **71.2** %.

In order to have an idea on classes that are the most difficult to recognize, we compute the intraclasse score. This is shown in the figure 1. We see that the classifier has difficulty with words that have a similar pronunciation, such as "no", "go", "wow". This is more apparent when looking at the classifier's predictions for all instances of the word "no" (figure 2).

One way to correct this may be the use of a more discriminating features to help the classifiers, or the use of a more sophisticated ones. This was not done because of the limitation of computing power. The confusion matrix in figure 3 gives a better view of the confusing words for each class.

2 Classification of segmented voice commands

Question 2.1 As the formula of **WER** contains only positive integers, there no way to have WER < 0. However, as the formula of WER can be rewritten as WER $= \frac{S+D+I}{S+D+C}$ with C the number of correct words, so if $I > C$, it is possible to have WER > 100 %.

Question 2.2 Because the likelihood can be rewritten as $P(X_i|W_i) \propto P_{discriminatorsingleword}(W_i|X_i)$ when $P(W_i) = constant$, the line in code that use this is:

posterior = model_predict_proba_function (features_input)

Question 2.3 In the provided example, we first commence by extracting a subset of *train_sequence_list*, and we use the *greedy algorithm* (which just chooses the word with the highest probability) to predict the corresponding sentences. We do the same for the test set, by predicting the corresponding sentences to the test audio signals using the same algorithm. Finally, we compute the WER for both sets using the *jiwer* library. It can be noticed that there is no notion of learning here, because the same algorithm is applied to both sets without learning anything.

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

$$P(W_i|W_{i-1}) = \frac{C(W_{i-1}W_i)}{C(W_{i-1})}$$

where $C(X)$ represents the number of occurrences of X.

Question 2.5 We first commence by creating a list containing all the training sentences by looping over 'train_sequence_list' and creating a list of lists of words. Also we created a function *'find_ngrams'* which is given a list of words and an integer 'n', returns all the n-grams from the given list. Finally, the function *'construct_ngram_matrix'* construct the transition matrix by taking as an input a list of sentences and an integer 'n'. It loops over all the sentences, and count the occurrences of n-grams and (n-1)-grams using the Object 'Counter', and returns the probability of the transition using the formula from the last question.

Question 2.6 The advantages of using a bigger N is to capture more context, and word ordering. However, this creates the problem of sparsity, where the majority of N-grams do not exist in the training corpus, so items not seen in the training data will be given a probability of 0.0. This problem can be mitigated using smoothing.

Question 2.7 Considering that the dimension of *data* is $N \times d$, where N is the number of rows, and d is the number of columns, the complexity of my implementation is $\mathcal{O}(N \times d \times beam\_size)$.

Question 2.8 The relationship between the probability to be in state k at step j , and the probabilities to be in state j' at step $k-1$ is $p(X_k = j) = \sum_{j'=1}^{d} a_{j',j} p(X_{k-1} = j')$ where $a_{j',j}$ is the probability transition from j' to j.

Considering that the dimension of *data* is $N \times d$, where N is the number of rows, and d is the number of columns, the complexity of Viterbi decoder is $\mathcal{O}(N \times d^2)$.

After evaluating the two algorithms, we found that Viterbi decoder gives better results (which is expected) than beam search (with a beam size of 5), with a $WER = 0.11$ for the first on the test set, and $WER = 0.20$ for the second. It has been found that for beam search, increasing the beam size improves the results as expected. For the two implementation, we used a transition matrix based on the bigram matrix. We notice that unlike the greedy algorithm, the two decoders sometimes choose a word even if its probability is not the highest individually, but they privilege the words that maximize the probability of the sentence. For example, in the sentence "happy bird on happy bird wow", decoders have chosen the word "happy" as the fourth word, while the word with the highest probability (in the sense of the greedy algorithm) is the word "five".

Question 2.9 The language model influences the quality of the decoder. If a transition (from one word to another) is rare in the training corpus, this causes the decoder to choose a more likely transition, but which may be wrong. An example of this is the sequence **"zero down zero"**. The decoder did not predict the first word correctly and chose the word **"up"**, and since the transition **"up down"** does not appear in the training corpus, the decoder chose the word **"stop"** because its transition probability is higher, despite the fact that the classifier predicted with a high probability the word **"down"**. The result of this is to get the sentence **"up stop zero"**.

Question 2.10 A backoff strategies to face rare seen words is to use smoothing. For my experiences , I used Laplace smoothing. Contrary to what is expected, this did not improve the result.

Question 2.11 Jointly optimizing an acoustic model and language model is an active area of research. One way of doing so is to model the acoustic part using context-dependent hidden Markov models (HMMs) consisting of N states, and each state contains Gaussian mixtures of K components. On the other hand, the language part is modeled using n-grams, these n-grams are integrated into a decoding graph using a sequence of WFST operation. The resulting decoding graph carries the n-gram probabilities as weights distributed on the transition arcs. The model parameters are jointly optimized using gradient probabilistic descent. For more information, see [2].

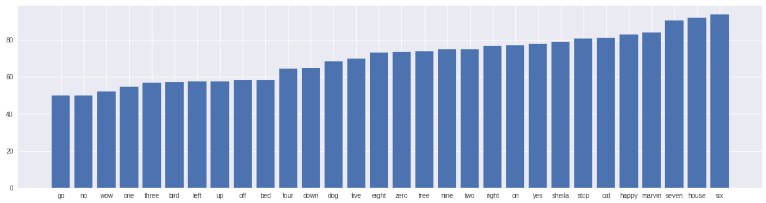

Figure 1: Intraclass score

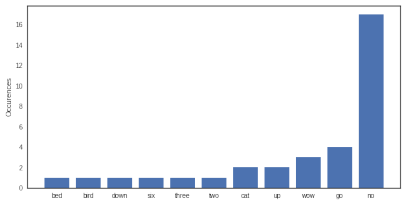

Figure 2: Predictions for the word ”No”

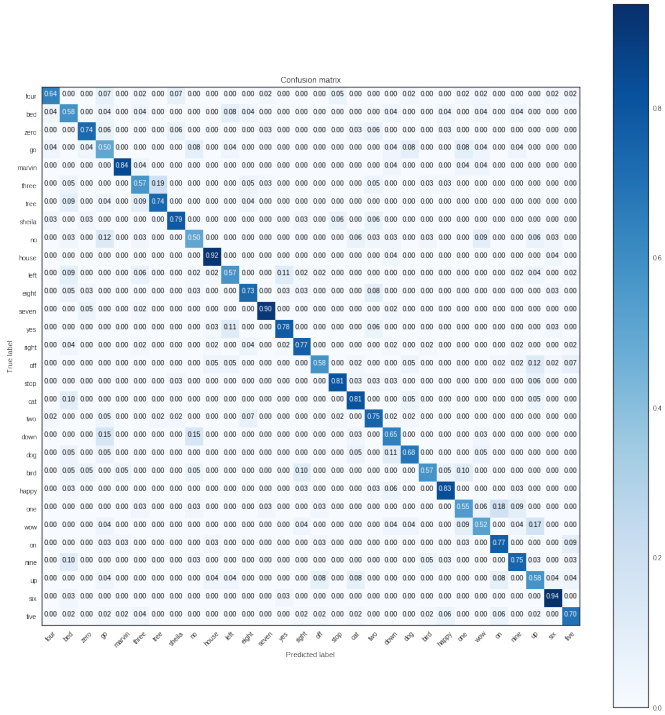

Figure 3: Confusion matrix

References

[1] James H. Martin Daniel Jurafsky. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. 2000.

[2] Abdelaziz A.Abdelhamid and Waleed H.Abdulla. Joint discriminative learning of acoustic and-language models on decoding graphs. 2018.