Signal and Language Processing - Lab 2

Καρποντίνης Δημήτρης, Λυμπερόπουλος Θοδωρής

27 Αυγούστου 2022

1.

We start our project with a short description of some basic terms that are used or calculated in our project.

Mel-Frequency Cepstral Coefficients. Speech processing is a basic prerequisite step for many applications of speech and language processing (SLP). In the process of speech processing, a signal is first filtered and sliced into small, overlapping frames, where we safely assume that the frequencies for every frame are (approximately) constant. A window function is then applied to each frame (usually the Hamming window) in order to emphasize the central values of the frame while reducing spectral leakage. Afterwards, a Short-Time Discrete Fourier Transform is applied, expressing the signal in the frequency domain, and then the power spectrum is extracted. In the following step, triangular filters are applied, that respect the non-linear human ear perception (more selective at lower frequencies), in order to compute the spectrogram of the signal. A spectrogram's contribution is to visualize the strength of a signal over time at different frequencies. At last, a Discrete Cosine Transform is applied in order to decorrelate the (highly correlated) filter bank coefficients, resulting to the MFCCs. The MFCCs can be viewed as tensors; an MFCC characterizes and correspond to a particular signal. We can use those tensors as input data, to train many machine learning algorithms, in order for many SLP tasks. Frequently, we append the first and second derivatives of the values to those tensors, in order to make them more discriminative. The usual length of the vector is 13, but with the addition of the derivatives, the length end up to be 39.

Language Models. A language model is a statistical tool to predict words. This is achieved by finding patterns in the human language and thus a very large and rich dataset with texts needs to be provided. Usually we feed a language model with task-specific context, since the language and its structure might differ in different cases (e.g. tweets in twitter). These models can be applied to predict the spoken word in an audio recording, the next word in a sentence, and which email is spam.

Acoustic Models. The task of representing the relationship between an audio signal and the phonemes or other linguistic units that make up speech is essential in automatic speech recognition (ASR). This is the task of an acoustic model, which is trained from a set of audio recordings and their corresponding transcripts. The richer the contexts the units are found, the better the statistical distributions for each of the model states.

It is worth noting that modern speech recognition systems use both acoustic and language models to represent the statistical properties of speech. The acoustic model is responsible for modeling the relationship between the audio signal and the phonetic units in the language. The language model discovers the statistical properties of the word sequences in the language. These two models are combined to improve their performance to the task of finding the corresponding text of a given audio segment.

2. Data Preprocessing

We should first notice that we created a new folder *usc* corresponding to the usc fonder in the kaldi directory, in order not to have to upload kaldi. The results of every process of the project, as well as the scripts and commands for creating them can be found there. Also, we only kept some of the models and data in the folder uploaded to helios, since the limit of the uploaded size did not

let us upload the whole project. The parts left in it contain the code and a part of the data and models, descriptive enough to understand the whole process. The whole project can be fould here: https://github.com/theoOmathimatikos/SLP_lab2

We followed the instructions of the 3 step. In 3.4, we created many bash and python files, each corresponding to a different functionality:

1) This particular script *uttids.sh* is used in order to list all the utterance ids for the training test and dev datasets.

2)The utterance to speaker script *utt2spk.sh* puts together all utterance ids with the corresponding speaker, in the form utterance-id speaker. The script reads each line of the training, testing or validation texts, stored in filesets, and produces a utt2spk text file named simply *utt2spk*. Each line read is split using the understrike symbol ('_') in two components utterance number and utterance speaker. Afterwards we output the utterance id (as whole) and the utterance speaker to a new line recursively.

3)The utterance to wav script *wav_scp.sh* is used in order to match all utterance ids with the suitable .wav file. This is achieved by simply concatenating the suffix '.wav' after each line read from the corresponding file (training, testing, validation). Finally we write to the each new line of the new file (*wav.scp*) the utterance id and the wav file's name

4)The utterance to text script *text.sh* lists all utterance ids along with the relevant text. This is done by reading the appropriate line (given by the utterance id number) of the transcriptions text file (*transcriptions.txt*) then writing to each new line of the new file (*text*) the utterance id and the relative text .

5) *convert_text_to_lexicon* is a python method in the *text_to_lexicon* script that is responsible for the substitution of the sentences with its corresponding sequences of phonemes. The function first reads the lines of the lexicon file and creates a dictionary that matches each word to its corresponding phoneme. Then, for each sentence of the different files, it splits it into words and finds its corresponding phonemes. After that, it adds the strings of the phonemes of a sentence and writes the results in a new file.

Then, we follow the instructions of the steps 4.1 and 4.2. Some of the bash commands that we used are saved in bash files of the corresponding folders (e.g. for 2.4.5, for sorting the files we run the bash file sort.sh in the usc folder).

Question 1

The perplexity of a sentence is given as the n-th root of the inverse of the probability that this sentence occurs.
Specifically :

$$PP = \sqrt[N]{\frac{1}{\mathbb{P}[W_1, W_2, \ldots, W_N]}}$$

With the use of the kaldi command compile-lm -e we can evaluate the perplexity of the bigram and unigram models on the test and development sets respectively.
The calculated values are presented below:

Model Type	Dataset	Perplexity
Unigram	Test	54.54
Unigram	Development	41.77
Bigram	Test	27.65
Bigram	Development	19.51

Table 1: Perplexity Table

As we can observe the perplexity of the model decreases as n increases. This is expected as a more detailed model, which is properly trained, is considered more accurate. Specifically we can deduct, that as the amount of information the n-gram gives us about the word sequence increases (i.e. as n increases) the model perplexity decreases. As n increases, the number of parameters of our model increases, which in turn leads to a more detailed model. If N is the amount of words we have, a unigram only requires N parameters to be trained, bigram N squared, and so on, leading to a more intricate model.

Question 2

In many applications of the ASR systems, researchers have noticed a degradation of the performance of those systems, due to mismatch between training and test conditions. Training data are usually clean, while test data (that are acquired when the models are applied to real-world scenarios) are noisy. CMVM is a feature normalization technique; these techniques modify the noisy test features in order to match the statistics of the clean training data. Thus, they work as a filter for the training set, before being fed to the models.

Cepstral Mean Normalization associates the first order moment of every training and test utterance by removing their respective time average, and transforming each utterance to zero mean. On the other hand, CMVN matches both mean and variance by transforming every training and test utterance to zero mean and unit variance. Let x_t denote a 13-dimensional cepstral vector at time t of an utterance, and $x_t(i)$ represent the i^{th} component of x_t. Let $X = [x_1, x_2, , x_t, , x_]T$ denote an utterance of length T. CMVN is performed by first computing the mean (μ) and variance (σ^2) independently for every dimension in maximum likelihood (ML) framework, as shown below.

$$\mu_{ML}(t) = \frac{1}{T}\sum_{t=1}^{T} x_t(i) \; 1 \leq i \leq 13$$

$$\sigma^2_{ML}(i) = \frac{1}{T-1}\sum_{t=1}^{T}(x_t(i) - \mu_{ML}(i))^2 \; 1 \leq i \leq 13$$

The mean and variance normalized frame $\hat{x}_t$ is computed for all 13 dimensions and for all frames as shown in the equation below, to obtain the normalized utterance $\hat{X}$.

$$\hat{x}_t(i) = \frac{x_t(i) - \mu_{ML}(i)}{\sigma_{ML}(i)} \;\; 1 \leq t \leq T, \; 1 \leq i \leq 13$$

Question 3

In order to find the total number of frames for each wav file, we first need to read the files themselves. Using the python library scipy.io.wavfile and specifically the library's method *read* we can acquire the signal and its sample rate. More specifically this method returns the values of the signal for each time step.

Therefore, by dividing the length of the signal (number of timesteps), with the amount of samples taken per millisecond (i.e. the frame rate) we calculate the total duration of the wav file.

Now, provided that the stride and frame_size are known, we can calculate the number of frames in each wav file. The formula giving the wanted quantity is:

$frame\_size = \lceil \frac{T-f}{s} \rceil$

Where f is the frame size and s the stride, both measured in milliseconds.

The frame size (usually chosen between 10 and 40 miliseconds) denotes the number of miliseconds that each frame contains, while the stride defines the step taken each time a new frame is created. Note that it is common for overlapping to occur between different frames (i.e. a millisecond of the original signal might be accounted for in different frames), if stride is smaller than the frame size. This overlapping is sought after, as we wish to acquire all possible phones. In order to achieve this we take a substantially small frame size and an even smaller stride. The resulting overlap ensures us that almost all information will be accounted for regarding the different phones.

The most common values for the stride and frame size are :

$stride = 10\ ms$ (0.01), $frame\_size = 25\ ms$ (0.025)
Using the above methodology the script *wav_frames.py* calculates the desired number of frames for the first five wav files of the training dataset as: [317, 370, 398, 328, 463]

The features produced from these windows would have dimensions : 39 (3 * (12 + 1) x number of frames. The 39 features created are the mfccs the energy their deltas and the acceleration. This are produced for each of frame created. Therefore we have dimensions of : [39 x 317, 39 x 370, 39 x 398, 39 x 328, 39 x 463] for the first five wav files on the training datset.

2. Models

We followed the steps of 4.4. We saved our models in a new directory, *usc/models*, where we can find three folders (*mono*, *mono_ali*, *tril*) as well as a bash script with a command for 4.4.1. In the folder *mono*, we can find two bash scripts, namely *create_hclg.sh* and *decode.sh* for the steps 4.4.2 and 4.4.3. respectively. We did not create any bash file for 4.4.5, since the steps are the same as of 4.4.1-4.4.4. We can observe the Phone Error Rate (PER) in the development and test set for the two different models.

Model Type	Test Set	Development Set
Mono + Unigram	52.37%	52.92%
Mono + Bigram	45.71%	47.14%
Tril + Unigram	39.89%	39.97%
Tril + Bigram	35.69%	36.87%

Table 2: Accuracy of Models.

As we can observe, the more sophisticated the model, the better the results and the smaller the Phone Error Rate. Both acoustic models yield better results with the use of bigrams instead of unigrams, and for every language model the triphones are better than the monophones. Encouraged by those results, we could think of creating even more complex language models that capture more patterns of the language. Another approach, in case we do not have enough data for more spicific models could be to use backoff techniques (bigram instead of trigram or smaller values for the N-gram). Additionally we could interpolate more specific models with less specific ones in order to increase model robustness. This technique is called smoothing interpolation.

Questions 4, 5

At the description of the mfccs, we saw that we can convert an input sound file O to a sequence of vectors of lenght 13 (or 39) that represent a transformation of the power spectrum of the signal. Successive vectors (that correspond to successive slices of the sound input) are time dependent; they share a part of the signal. Also, we expect that for a small window the signal would not have changed drastically, and thus the information (that in our case could be the signal of a phoneme, or the change from one phoneme to another) would be distributed into successive vectors. Thus, it is only logical to consider an HMM model, with hidden states that model this sequential time dependency.

Bayes Rule. By treating the acoustic input O as a sequence of vectors, we can write

$$O = o_1, o_2, ..., o_T.$$

The corresponding sentence that this sound produces can be written as

$$W = w_1, w_2, ..., w_n.$$

For a new sound file, we want to predict the sentence that it generates. For a generative model such as the GMM-HMM, we maximize the probability for the given observation (sound) to generate a particular sentence W, thus

$$\hat{W} = \underset{W}{argmax}\, P(W|O) = \underset{W}{argmax}\, P(O|W)P(W)$$

$P(W)$ is computed by the language model, which, in our case, is an N-gram, while $P(O|W)$ is computed by the acoustic model, here the GMM-HMM.

HMM. The hidden states of the HMM correspond to parts of phones. That means that a phone is modelled by multiple hidden states (usually 3). Transitions between those states are strict; the model is a left-to-right HMM, following the sequential structure of the language. On the other hand, each observation (that is produced from a hidden state) is information about a particular time of the signal. The decoding process maps the sequence of information to phones and words. We now have to answer how to compute the likelihood of a feature vector given an HMM state.

GMM. A particular cepstral feature that an HMM's hidden state tries to model, might not follow a normal distribution, but a more complex pattern instead. Still, mathematics provide us with useful tools; every distribution can be approximately modelled by a large number of Gaussians. Thus, a Gaussian Mixture Model achieves to handle those irregularities.

Training. After describing the architecture of the model, the training phase follows. The parameters of the model are calculated from the training dataset. We use the Baumm-Welch algorithm to calculate the probability of being in each state j at while observing o_t, as well as to calculate the contribution of each mixture (of Gaussians) to an observation o_t. The algorithm iteratively calculates this probability, as well as the parameters of the gaussians.

Thus, for a unigram language model, the probability of a sequence of words is the product of the probabilities of the words themselves (which is the number of occurences of the word divided by the total number of words). Thus, $P(W)$ can be easily calculated.

Prediction. For a given sound signal, the forward algorithm is used to predict the most probable sentence that corresponds to that signal. The algorithm calculates for every time t and state q, which is the most probable path until that state, and stores the intermediate results. Its computational complexity is $O(N^2T)$, where N is the number of states and T the time.

Question 6

The graph HCLG is a composition of the following fsts:

- H contains the HMM definitions. Its input symbols are transition-ids, which encode the probability density function id, while its output symbols are context-dependent phones (i.e. phones that belong to the same window).
- C represents the context-dependency: its input symbols are the context dependent phones that are passed from H, and its output symbols are context independent phones. C can be thought as a transducer.
- L is the lexicon, which accepts the phones produced by C and returns words
- Finally G is an acceptor, which means that it accepts and returns the same symbols. In our case words. G encodes the language model.

It is essential to ensure that the graph is stochastic, while its output is minimized and determinized. In order for the graphs output to be determinizable we require disambiguation symbols. This symbols are used in order to produce some form of hierarchy between our phoneme sequences (e.x. determine a phoneme sequence as a suffix or prefix of another).

Overall this graph takes as input encoded pdf ids and returns the most relevant word.

Our code can be also found here: https://github.com/theoOmathimatikos/SLP_lab2