ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Επεξεργασία Φωνής και Φυσικής Γλώσσας

Προπαρασκευή + Κυρίως μέρος 2ου Εργαστηρίου:

Αναγνώριση φωνής με το KALDi TOOLKiT

Αναφορά

Μάρκος Γιαννόπουλος 03118103

Νικόλαος Καραφύλλης 03119890

Μάιος 2023

Θεωρητικό Υπόβαθρο

MFCCs

Τα MFCCs (Mel frequency cepstral coefficients) είναι χαρακτηριστικά που χρησιμοποιούνται ευρέως σε συστήματα αναγνώρισης φωνής.

Τα βήματα που ακολουθούνται για τον υπολογισμό τους είναι τα εξής:

- Pre-emphasis: Εφαρμόζουμε ένα φίλτρο που δίνει έμφαση στις υψηλότερες συχνότητες γιατί συνήθως έχουν μικρότερη ένταση
- Framing: χωρίζουμε το σήμα σε frames των 20-40 ms με περίπου 50% overlap για να υπολογίσουμε τα (σχεδόν σταθερά στα frames) χαρακτηριστικά σε διαφορετικές χρονικές στιγμές
- Windowing: Φιλτράρουμε όλα τα frames με ένα παράθυρο Hamming για να τα ομαλοποιήσουμε
- FFT: Κάνουμε FFT σε κάθε frame για να πάμε στο πεδίο της συχνότητας. Ο συνδυασμός της χρονικής και συχνότικης πληροφορίας είναι ο STFT του σήματος.
- Mel filterbank: Εφαρμόζουμε τα τριγωνικά φίλτρα της κλίμακας mel που προσομοιάζει την ανθρώπινη ακοή και αντίληψη του ήχου που αντιλαμβάνεται καλύτερα τις χαμηλές συχνότητες
- DCT: Εφαρμόζουμε μετασχηματισμό DCT για να μειώσουμε την συσχέτιση των συντελεστών και παίρνουμε τους cepstral συντελεστές. Κρατάμε συνήθως τους 13 πρώτους σαν χαρακτηριστικά.

Γλωσσικά μοντέλα

Τα γλωσσικά μοντέλα (language models) είναι στατιστικά μοντέλα που χρησιμοποιούνται στην επεξεργασία φυσικής γλώσσας και εκτιμούν τις πιθανότητες εμφάνισης μιας ή περισσότερων λέξεων σε μια γλώσσα.

Τα γλωσσικά μοντέλα n-gram εκτιμούν την πιθανότητα εμφάνισης μιας λέξης με βάση τις n-1 προηγούμενες λέξεις, π.χ. unigram (μόνο a priori), bigram (1 λέξη context).

Φωνητικά μοντέλα

Τα φωνητικά μοντέλα (acoustic models) εκτιμούν τις κατανομές πιθανοτήτων των ακουστικών χαρακτηριστικών (πχ MFCCs) για δεδομένες λέξεις ή φωνήματα. Ένα παράδειγμα φωνητικού μοντέλου, το οποίο χρησιμοποιήσαμε σε αυτό το εργαστήριο και αναλύεται παρακάτω είναι το HMM-GMM.

Τα φωνητικά μοντέλα είναι πολύ σημαντικά για την αναγνώριση φωνής καθώς συνδέουν τα ακουστικά σήματα φωνής με την γλωσσική πληροφορία που περιέχουν.

ΠΡΟΠΑΡΑΣΚΕΥΗ

Αφού εγκαταστήσουμε το εργαλείο Kaldi, κατεβάσουμε τα δεδομένα και δημιουργήσουμε τους φακέλους που ζητούνται, δημιουργούμε τα αρχεία:

- utt2spk μέσω του bash script make_utt2spk.sh:

```bash
#!/bin/bash

# Define the list of input and output file pairs
input_files=("./data/train/uttids" "./data/test/uttids" "./data/dev/uttids")
output_files=("./data/train/utt2spk" "./data/test/utt2spk" "./data/dev/utt2spk")

# Loop through each input/output file pair
for i in "${!input_files[@]}"; do

  # Initialize line counter
  line_num=1

  # Loop through each line in the input file
  while read line; do

    # Take the first two characters of the line
    speaker_id="${line:0:2}"

    # Combine line number with "utterance_id"
    utterance_id="utterance_id_${line_num}"

    # Write the new line to the output file
    echo "$utterance_id $speaker_id" >> "${output_files[$i]}"

    # Increment the line number counter
    line_num=$((line_num+1))

  done < "${input_files[$i]}"

done
```

οπότε δημιουργούνται αρχεία της μορφής :

```
f1_003 f1
f1_004 f1
f1_005 f1
f1_007 f1
```

- wav.scp μέσω του script make_wav.scp.sh:

```bash
#!/bin/bash

# Define the list of input and output file pairs
input_files=("./data/train/uttids" "./data/test/uttids" "./data/dev/uttids")
output_files=("./data/train/wav.scp" "./data/test/wav.scp" "./data/dev/wav.scp")

# Loop through each input/output file pair
for i in "${!input_files[@]}"; do

  # Initialize line counter
  line_num=1

  # Loop through each line in the input file
  while read line; do

    # Split the line into an array of strings
    IFS=' ' read -r -a strings <<< "$line"

    # Combine line number with "utterance_id"
    utterance_id="utterance_id_${line_num}"
    path="./wav/${strings[0]}.wav"

    # Write the new line to the output file
    echo "$utterance_id $path" >> "${output_files[$i]}"

    # Increment the line number counter
    line_num=$((line_num+1))

  done < "${input_files[$i]}"

done
```

οπότε δημιουργούνται αρχεία της μορφής :

```
f1_003 ./wav/f1_003.wav
f1_004 ./wav/f1_004.wav
f1_005 ./wav/f1_005.wav
f1_007 ./wav/f1_007.wav
```

- text μέσω του script make_text.sh:

```bash
#!/bin/bash

# Define the list of input and output file pairs
input_files=("./data/train/uttids" "./data/test/uttids" "./data/dev/uttids")
output_files=("./data/train/text" "./data/test/text" "./data/dev/text")

transcriptions_file=("./transcriptions.txt")

# Loop through each input/output file pair
for i in "${!input_files[@]}"; do

  # Initialize line counter
  line_num=1

  # Loop through each line in the input file
  while read line; do

    # Split the line into an array of strings
    IFS='_' read -r -a strings <<< "$line"

    # Combine line number with "utterance_id"
    utterance_id="utterance_id_${line_num}"

    search_string="${strings[1]}"

    # Search for lines where the first string is the search_string
    result=$(grep "^${search_string}" "${transcriptions_file}")

    # If a matching line is found, extract the rest of the string
    if [ -n "${result}" ]; then
        text=$(echo "${result}" | cut -d$'\t' -f2-)
    else
        echo "No matching line found"
    fi

    # Write the new line to the output file
    echo "$utterance_id $text" >> "${output_files[$i]}"

    # Increment the line number counter
    line_num=$((line_num+1))

  done < "${input_files[$i]}"

done
```

οπότε δημιουργούνται αρχεία της μορφής :

```
f1_003 She is thinner than I am.
f1_004 Bright sunshine shimmers on the ocean.
f1_005 Nothing is as offensive as innocence.
```

και για το τελευταίο βήμα μετατρέπουμε τις λέξεις των προτάσεων των αρχείων text σε φωνήματα μέσω του script text2phonemes.sh :

```
#!/bin/bash

# Define the list of input and output file pairs
input_files=("./data/train/text" "./data/test/text" "./data/dev/text")
output_files=("./data/train/phonemes" "./data/test/phonemes" "./data/dev/phonemes")

lexicon_file=("./lexicon.txt")

# Loop through each input/output file pair
for i in "${!input_files[@]}"; do

    # Loop through each line in the input file
    while read -r line; do

        # Split line into utterance ID and sentence
        utterance_id=${line%% *}
        sentence=${line#* }

        # Make sentence uppercase
        sentence=$(echo "$sentence" | tr '[:lower:]' '[:upper:]')
        # Remove special characters except single quotes
        sentence=$(echo "$sentence" | tr -cd "[:alnum:][:space:]'\-'"| sed 's/-/ /g')
        # Split sentence into words
        words=($sentence)

        # Replace each word with its corresponding phonemes from the lexicon
        phonemes=()
        for word in "${words[@]}"; do
            #echo ${word}
            # Search for lines where the first string is the search_string
            result=$(grep "^${word}[[:space:]]" "${lexicon_file}")

            # If a matching line is found, extract the rest of the string
            if [ -n "${result}" ]; then
                phonemes+=$(echo "${result}" | cut -d$'\t' -f2-)" "
            else
                echo "No matching line found for word: " $word
            fi
        done

        # Write new sentence to output file
        echo "$utterance_id "sil" "${phonemes[@]}" "sil"" >> "${output_files[$i]}"

    done < "${input_files[$i]}"

done
```

οπότε δημιουργούνται τα αρχεία phonemes που έχουν την μορφή:

```
f1_003 sil  sh iy ih z th ih n er dh ae n ay ae m  sil
f1_004 sil  b r ay t s ah n sh ay n sh ih m er z aa n dh ah dh iy ow sh ah n  sil
f1_005 sil  n ah th ih ng ih z ae z ah f eh n s ih v ae z ih n ah s ah n s  sil
```

ΤΕΛΟΣ ΠΡΟΠΑΡΑΣΚΕΥΗΣ

4 Βήματα κυρίως μέρους

4.1 Προετοιμασία διαδικασίας αναγνώρισης φωνής για τη USC-TIMIT

Η υλοποίηση της 4.1 σε κώδικα βρίσκεται στο script 4.1.sh, από όπου εμφανίζουμε τα παρακάτω code snippets

1.

```
#4.1.1
cd ../ #go to egs dir
cp wsj/s5/path.sh usc
cp wsj/s5/cmd.sh usc
```

και τροποποιούμε τα αρχεία όπως στην εκφώνηση

2.

```
#4.1.2
cd usc
ln -s ../wsj/s5/steps ./steps
ln -s ../wsj/s5/utils ./utils
```

3.

```
#4.1.3
#maybe do this in data dir
mkdir local
ln -s ../steps/score_kaldi.sh ./local/score_kaldi.sh
```

4.

```
#4.1.4
mkdir conf
wget https://raw.githubusercontent.com/slp-ntua/slp-labs/master/lab2/mfcc.conf -P conf/
```

5.

```
#4.1.5
mkdir data/lang
mkdir data/local/dict
mkdir data/local/lm_tmp
mkdir data/local/nist_lm
```

4.2 Προετοιμασία γλωσσικού μοντέλου

Η υλοποίηση της 4.2 σε κώδικα βρίσκεται στο script 4.2.sh, από όπου εμφανίζουμε τα παρακάτω code snippets.

1.

```
#4.2.1
touch data/local/dict/silence_phones.txt &&  echo "sil" > data/local/dict/silence_phones.txt
touch data/local/dict/optional_silence.txt &&  echo "sil" > data/local/dict/optional_silence.txt

./create_nonsilent_vocab.sh

./create_phonem_lexicon.sh ./data/local/dict/silence_phones.txt ./data/local/dict/nonsilence_phones.txt

./create_lms.sh

touch ./data/local/dict/extra_questions.txt
```

Κάποιες εξηγήσεις για τον κώδικα:

Για να δημιουργήσουμε το αρχείο nonsilence_phones.txt χρησιμοποιούμε το script create_nonsilent_vocab.sh το οποίο κρατάει τα μοναδικά φωνήματα από το αρχείο lexicon.txt(το αρχικό, με τις λέξεις) και τα σορτάρει, οπότε δημιουργείται το αρχείο με 40 διαφορετικά φωνήματα.

Για να δημιουργήσουμε το αρχείο lexicon.txt χρησιμοποιούμε το script create_phonem_lexicon.sh το οποίο αφού ενώσει τα 2 αρχεία (silent και nonsilent) τυπώνει από 2 φορές το ανάλογο φώνημα κάθε γραμμής και τα βάζει στο lexicon.txt, οπότε δημιουργείται το αρχείο που ξεκινά έτσι:

```
1   sil sil
2   aa aa
3   ae ae
4   ah ah
```

Για να δημιουργήσουμε τα αρχεία lm_train.txt χρησιμοποιούμε το script create_lms.sh
Για να δημιουργήσουμε το κενό αρχείο extra questions.txt χρησιμοποιούμε την εντολή touch.

2. Χρησιμοποιούμε το script lm_build.sh ώστε να εκτελέσουμε την build-lm.sh με τιμή παραμέτρου n 1 και 2 (unigram και bigram) για τα 3 αρχεία txt, οπότε δημιουργούμε συνολικά 6 αρχεία τύπου .ilm.gz

Παρακάτω ο κώδικας του lm_build.sh

```
#!/bin/bash

source ./path.sh

# Define the list of input and output file pairs
input_files=("./data/local/dict/lm_train.txt" "./data/local/dict/lm_test.txt" "./data/local/dict/lm_dev.txt")
output_dict=("./data/local/lm_tmp/train" "./data/local/lm_tmp/test" "./data/local/lm_tmp/dev")

# Loop through each input
for i in "${!input_files[@]}"; do
    #unigrams
    build-lm.sh -i "${input_files[i]}" -n 1 -o "${output_dict[i]}_unigram.ilm.gz"
    #bigrams
    build-lm.sh -i "${input_files[i]}" -n 2 -o "${output_dict[i]}_bigram.ilm.gz"
done
```

3. Εκτελούμε τις παρακάτω εντολές(και για τα test και dev, λόγω του ότι θα χρησιμοποιηθούν παρακάτω)

```
#4.2.3
compile-lm ./data/local/lm_tmp/train_unigram.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c > ./data/local/nist_lm/lm_phone_ug.arpa.gz
compile-lm ./data/local/lm_tmp/train_bigram.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c > ./data/local/nist_lm/lm_phone_bg.arpa.gz

compile-lm ./data/local/lm_tmp/test_unigram.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c > ./data/local/nist_lm/test_ug.arpa.gz
compile-lm ./data/local/lm_tmp/test_bigram.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c > ./data/local/nist_lm/test_bg.arpa.gz

compile-lm ./data/local/lm_tmp/dev_unigram.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c > ./data/local/nist_lm/dev_ug.arpa.gz
compile-lm ./data/local/lm_tmp/dev_bigram.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c > ./data/local/nist_lm/dev_bg.arpa.gz
```

4.

```
#4.2.4
prepare_lang.sh ./data/local/dict "<oov>" ./data/local/lang ./data/lang
```

5. Σορτάρουμε τα αρχεία με το script sort.sh

```
#!/bin/bash

# sort text wav.scp and utt2spk in all 3 folders
sort -o ./data/train/text ./data/train/text
sort -o ./data/train/wav.scp ./data/train/wav.scp
sort -o ./data/train/utt2spk ./data/train/utt2spk

sort -o ./data/test/text ./data/test/text
sort -o ./data/test/wav.scp ./data/test/wav.scp
sort -o ./data/test/utt2spk ./data/test/utt2spk

sort -o ./data/dev/text ./data/dev/text
sort -o ./data/dev/wav.scp ./data/dev/wav.scp
sort -o ./data/dev/utt2spk ./data/dev/utt2spk
```

6.

```
#4.2.6
./utils/utt2spk_to_spk2utt.pl ./data/train/utt2spk > ./data/train/spk2utt
./utils/utt2spk_to_spk2utt.pl ./data/test/utt2spk > ./data/test/spk2utt
./utils/utt2spk_to_spk2utt.pl ./data/dev/utt2spk > ./data/dev/spk2utt
```

Οπότε δημιουργούνται τα αρχεία spk2utt που μοιάζουν κάπως έτσι:

```
f1 f1_001 f1_006 f1_011 f1_016 f1_021 f1_026 f1_031 f1_036 f1_041 f1_
f5 f5_001 f5_006 f5_011 f5_016 f5_021 f5_026 f5_031 f5_036 f5_041 f5_
m1 m1_001 m1_006 m1_011 m1_016 m1_021 m1_026 m1_031 m1_036 m1_041 m1_
m3 m3_001 m3_006 m3_011 m3_016 m3_021 m3_026 m3_031 m3_036 m3_041 m3_
```

7.
Κατεβάζουμε το timit_format_data.sh και το εκτελούμε:

```
#4.2.7
wget https://raw.githubusercontent.com/slp-ntua/slp-labs/master/lab2/timit_format_data.sh

bash timit_format_data.sh
```

Ερώτημα 1
Εκτελούμε το script comp_perplexity.sh

```
#question 1
bash comp_perplexity.sh
```

Οπότε και έχουμε τα παρακάτω αποτελέσματα:

Set	n-gram type	Perplexity (%)
Validation	unigram	37.88
Validation	bigram	14.57
Test	unigram	38.18
Test	bigram	15.79

Από τις παραπάνω τιμές παρατηρούμε ότι τα μοντέλα bigram έχουν μικρότερο perplexity από τα μοντέλα unigram (και άρα έχουν μικρότερη αβεβαιότητα και μεγαλύτερη ακρίβεια ως προς τις προβλέψεις τους). Αυτό γίνεται λόγω του ότι τα bigram μοντέλα έχουν λάβει υπόψη τους και την προηγούμενη λέξη στις προβλέψεις τους(οπότε περιέχουν context , εξαρτήσεις λέξεων), σε αντίθεση με τα unigram που έχουν εκπαιδευτεί στο να γνωρίζουν μόνο πιθανότητες εμφάνισης κάθε λέξης ξεχωριστά.

4.3 Εξαγωγή ακουστικών χαρακτηριστικών

Η υλοποίηση της 4.3 σε κώδικα βρίσκεται στο script 4.3.sh, από όπου εμφανίζουμε τα παρακάτω code snippets.

Εξάγουμε τα MFCCs και για τα 3 σετ:

```
#!/bin/bash
source ./path.sh

# make directories for mfcc statistics
mkdir mfcc_logs

for dir in train test dev; do
    steps/make_mfcc.sh data/$dir mfcc_logs/$dir mfcc_${dir}
    steps/compute_cmvn_stats.sh data/$dir mfcc_logs/$dir mfcc
done

rm -R mfcc_logs
```

Ερώτημα 2

Με τη χρήση Cepstral Mean and Variance Normalization, μπορούμε να κάνουμε τα mfccs ανεξάρτητα από τον ομιλητή(τόνο, ένταση ομιλίας κλπ), ποιότητα ηχογράφησης, ώστε να έχουμε αποδοτικότερα μοντέλα. Γενικότερα, αυτή η μέθοδος προκαλεί μια ομοιομορφία στην κλίμακα των δεδομένων, που τα κάνει πιο συγκρίσιμα και κατάλληλα για εκπαίδευση μοντέλων μηχανικής μάθησης.

Ερώτημα 3

Εκτελούμε τα παρακάτω για να βρούμε τον αριθμό των ακουστικών frames.

```
# question 3

# frames per sentence
feat-to-len scp:data/train/feats.scp ark,t:data/train/feats.lengths
head -5 data/train/feats.lengths

# characteristics dimension
feat-to-dim ark:mfcc_train/raw_mfcc_train.1.ark -
```

Με τα παρακάτω αποτελέσματα για τα frames:

```
f1_003 317
f1_004 371
f1_005 399
f1_007 328
f1_008 464
```

Και με διάσταση 13 για τα χαρακτηριστικά.

4.4 Εκπαίδευση ακουστικών μοντέλων και αποκωδικοποίηση προτάσεων

Η υλοποίηση της 4.2 σε κώδικα βρίσκεται στο script 4.2.sh, από όπου εμφανίζουμε τα παρακάτω code snippets.

1. Εκπαιδεύουμε ένα monophone GMM-HMM ακουστικό μοντέλο πάνω στα train δεδομένα.

```
#!/bin/bash
source ./path.sh

# 4.4.1
# train monophone GMM-HMM model
steps/train_mono.sh data/train data/lang exp/mono
```

2. Δημιουργούμε τον γράφο HCLG σύμφωνα με την γραμματική G για τα unigram και τα bigram μοντέλα.

```
# 4.4.2
# create HCLG graph for unigram and bigram
utils/mkgraph.sh data/lang_phones_ug exp/mono exp/mono_graph_ug
utils/mkgraph.sh data/lang_phones_bg exp/mono exp/mono_graph_bg
```

3. Αποκωδικοποιούμε τις προτάσεις των test και dev δεδομένων, με τον αλγόριθμο Viterbi.

```
# 4.4.3
# decode validation and test data with Viterbi
for dir in test dev; do
    steps/decode.sh exp/mono_graph_ug data/$dir exp/mono/decode_${dir}_ug
    steps/decode.sh exp/mono_graph_bg data/$dir exp/mono/decode_${dir}_bg
done
```

4. Τα αποτελέσματα της καλύτερης αποκωδικοποίησης βρίσκονται στα αρχεία: exp/mono/decode_test_ug/scoring_kaldi/best_wer κ.ο.κ.
   Η μετρική που χρησιμοποιείται είναι το Phone Error Rate (PER):

$$PER = 100\frac{insertions + substitutions + deletions}{\#phonemes}$$

Το αποτέλεσμα σε όλες τι περιπτώσεις (train ή dev, unigram ή bigram) είναι 100% με 0 insertions. (Κάποιο πρόβλημα υπάρχει).
Οι δύο υπερπαράμετροι της διαδικασίας scoring είναι οι min-lwmt και max-lwmt και αντιπροσωπεύουν το ελάχιστο και το μέγιστο βάρος του γλωσσικού μοντέλου σε σχέση με το ακουστικό.

5. Κάνουμε alignment των φωνημάτων χρησιμοποιώντας το monophone μοντέλο.

Στη συνέχεια με αυτά τα alignments εκπαιδεύουμε ένα triphone μοντέλο.
Δημιουργούμε πάλι τον γράφο HCLG για τα unigram και bigram μοντέλα.
Αποκωδικοποιούμε πάλι τις προτάσεις, τα αποτελέσματα της καλύτερης αποκωδικοποίησης βρίσκονται στα αρχεία:
exp/tri1/decode_test_ug/scoring_kaldi/best_wer κ.ο.κ.
Το αποτέλεσμα είναι το ίδιο με πριν.

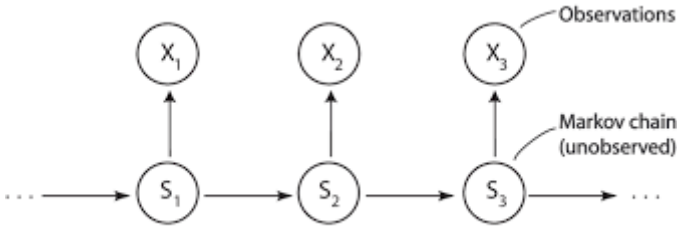
```
# 4.4.5
# align phones using monophone model
steps/align_si.sh data/train data/lang exp/mono exp/mono_ali

# train triphone model
steps/train_deltas.sh 2000 10000 data/train data/lang exp/mono_ali exp/tri1

# create HCLG graph for unigram and bigram
utils/mkgraph.sh data/lang_phones_ug exp/tri1 exp/tri1_graph_ug
utils/mkgraph.sh data/lang_phones_bg exp/tri1 exp/tri1_graph_bg

# decode validation and test data with Viterbi
for dir in test dev; do
    steps/decode.sh exp/tri1_graph_ug data/$dir exp/tri1/decode_${dir}_ug
    steps/decode.sh exp/tri1_graph_bg data/$dir exp/tri1/decode_${dir}_bg
done
```

Αναμένουμε ότι τα triphones θα έχουν μικρότερο PER λόγω του ότι λαμβάνουν υπόψη τους το φαινόμενο του co-articulation.

Ερώτημα 4

Το μοντέλο GMM-HMM είναι ένα ακουστικό μοντέλο που χρησιμοποιείται συχνά σε συστήματα αναγνώρισης φωνής. Αποτελείται από δύο βασικά μέρη:

- Το GMM (Γκαουσιανό Μοντέλο Μίξης) είναι μια μοντελοποίηση της κατανομής των χαρακτηριστικών των φωνημάτων (π.χ. mfccs). Υποθέτει ότι η κατανομή είναι ένα άθροισμα γκαουσιανών κατανομών με διαφορετικά βάρη. Κάθε γκαουσιανή αντιστοιχεί σε μια διαφορετική κατάσταση του φωνήματος.
- Το HMM (Κρυφό Μαρκοβιανό Μοντέλο) μοντελοποιεί την χρονική συσχέτιση των φωνημάτων. Αποτελείται από τις κρυφές καταστάσεις που αναπαριστούν φωνήματα (ή μέρη τους) και είναι μια μαρκοβιανή αλυσίδα, και από τις παρατηρήσιμες καταστάσεις που αναπαριστούν τα χαρακτηριστικά που αντιστοιχούν στα φωνήματα (mfccs), τα οποία βλέπουμε. Οι παράμετροι του είναι οι πιθανότητες μετάβασης από την μια κρυφή κατάσταση στην επόμενη, οι αρχικές πιθανότητες (priors) και οι πιθανότητες εμφάνισης των παρατηρήσεων δεδομένου του φωνήματος (ακολουθούν κατανομές GMM).



Το μοντέλο εκπαιδεύεται τον αλγόριθμο Baum-Welch ή forward-backward.
Ο αλγόριθμος είναι επαναληπτικός και είναι μια παραλλαγή του αλγορίθμου EM (Expectation Minimization). Μετά από μια αρχικοποίηση των παραμέτρων, επαναληπτικά υπολογίζονται οι πιθανότητες forward και backward και χρησιμοποιούνται για να ενημερωθούν οι παράμετροι του μοντέλου, εώς ότου συγκλίνουν.

Ερώτημα 5
Η a posteriori πιθανότητα στο πρόβλημα αναγνώρισης φωνής, δηλαδή η πιθανότητα ενός φωνήματος δεδομένης μιας παρατήρησης (ακολουθίας φωνητικών χαρακτηριστικών), υπολογίζεται σύμφωνα με τον τύπο του Bayes:

$$P(P|O) = P(O|P) \cdot P(P)/P(O)$$

Όπου:

- P(P|O) η a posteriori πιθανότητα
- P(O|P) η πιθανοφάνεια της παρατήρησης δεδομένου του φωνήματος, δηλαδή το ακουστικό μοντέλο που μοντελοποιήσαμε με GMM-HMM
- P(P) η a priori πιθανότητα του φωνήματος, δηλαδή το γλωσσικό μοντέλο
- P(O) η πιθανότητα εμφάνισης της παρατήρησης, κανονικοποιεί την πιθανότητα αλλά δεν επηρεάζει το φώνημα που την μεγιστοποιεί

Η πιο πιθανή ακολουθία φωνημάτων είναι το argmax των a posteriori πιθανοτήτων και υπολογίζεται με τον αλγόριθμο Viterbi.

Ερώτημα 6
Ο γράφος HCLG του Kaldi είναι η σύνθεση 4 διαφορετικών fst που αντιστοιχούν σε διαφορετικά στοιχεία του συστήματος αναγνώρισης φωνής:

- H.fst αντιστοιχεί στο HMM μοντέλο
- C.fst αντιστοιχεί στο context dependency
- L.fst αντιστοιχεί στο lexicon που παίρνει ως είσοδο μια λέξη και επιστρέφει τα φωνήματα που την αποτελούν
- G.fst αντιστοιχεί στην γραμματική, το γλωσσικό μοντέλο