Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Επεξεργασία Φωνής και Φυσικής Γλώσσας, 2024-2025

Αναφορά 1ου Εργαστηρίου: Αναγνώριση φωνής με το Kaldi Toolkit

Όνομα:	Γεώργιος
Επώνυμο:	Οικονόμου
ΑΜ:	03121103

Εισαγωγή

Ο κώδικας, συνοδευόμενος από σύντομα σχόλια, παρουσιάζεται στο Παράρτημα της αναφοράς. Εναλλακτικά, είναι διαθέσιμος στο ακόλουθο **αποθετήριο Git**.

Θεωρητικό Υπόβαθρο

Mel-frequency Cepstral Coefficients (MFCCs)

Τα MFCCs αποτελούν ακουστικά χαρακτηριστικά που εξάγονται από φωνητικά δεδομένα σε συστήματα επεξεργασίας και αναγνώρισης φωνής. Πρόκειται για συντελεστές Cepstrum, οι οποίοι προκύπτουν μέσω της αναπαράστασης του φάσματος του ήχου, ώστε να προσομοιάζουν τον τρόπο που οι άνθρωποι αντιλαμβάνονται τις συχνότητες, κατά την κλίμακα Mel. Για να επιτευχθεί αυτό:

Προέμφαση (Pre-Emphasis)

Το φίλτρο προέμφασης (Pre-Emphasis Filter) εφαρμόζεται για να ενισχύσει τις υψηλές συχνότητες. Μαθηματικά, περιγράφεται από την εξίσωση: $y(t) = x(t) - ax(t-1)$.

Πλαισίωση (Framing)

Το σήμα χωρίζεται σε μικρά χρονικά πλαίσια, καθώς οι συχνότητες του μπορούν να θεωρηθούν σχεδόν σταθερές σε πολύ σύντομα διαστήματα. Αυτό επιτρέπει την

αποτελεσματική εφαρμογή του μετασχηματισμού Fourier, διατηρώντας τις μεταβολές του φάσματος.

Παραθυροποίηση (Windowing)

Σε κάθε χρονικό πλαίσιο εφαρμόζεται ένα παράθυρο, σαν του Hamming, προκειμένου να ομαλοποιηθούν τα άκρα.

Συχνοτική Ανάλυση (Spectral Analysis)

Σε κάθε χρονικό πλαίσιο εφαρμόζεται ο μετασχηματισμός FFT για τον υπολογισμό της κατανομής της ενέργειας σε κάθε συχνότητα, δηλαδή, το φάσμα ισχύος (Power Spectrum).

Filter Banks

Εφαρμόζονται τριγωνικά φίλτρα, σε κλίμακα Mel, με γραμμικά μειούμενο πλάτος στο φάσμα των συχνοτήτων. Η κλίμακα Mel έχει σκοπό να μιμηθεί την μη γραμμική αντίληψη του ήχου από το ανθρώπινο αυτί.

Οι συντελεστές των Filter Bank που υπολογίστηκαν στο προηγούμενο βήμα είναι εξαιρετικά συσχετισμένοι, κάτι που μπορεί να δημιουργήσει προβλήματα σε ορισμένους αλγορίθμους μηχανικής μάθησης. Για να διορθωθεί αυτό:

Discrete Cosine Transform

Εφαρμόζεται ο μετασχηματισμός DCT και ως αποτέλεσμα έχουμε συμπιεσμένες εκδοχές των Filter Bank, οι οποίες αποτελούν τα ζητούμενα MFCCs.

Γλωσσικά Μοντέλα (Language Models)

Τα γλωσσικά μοντέλα είναι μοντέλα μηχανικής μάθησης που εκπαιδεύονται για να υπολογίζουν την a posteriori πιθανότητα της επόμενης λέξης, βασισμένα στις προηγούμενες λέξεις, με σκοπό να προβλέψουν την επόμενη λέξη που ακολουθεί σε μια πρόταση.

N-gram

Η χρήση του Kaldi δημιουργεί γλωσσικά μοντέλα τύπου n-gram, τα οποία βασίζονται στην πιθανότητα εμφάνισης μιας λέξης ή μιας ακολουθίας λέξεων σε σχέση με τις προηγούμενες n-1 λέξεις. Μαθηματικά, περιγράφονται από τις αλυσίδες Markov.

Φωνητικά Μοντέλα (Acoustic Models)

Τα φωνητικά μοντέλα χρησιμοποιούν στατιστικές μεθόδους για μοντελοποιήσουν τη σχέση των ακουστικών σημάτων και των γλωσσικών χαρακτηριστικών τους, όπως τα φωνήματα. Μαθηματικά, τις περισσότερες φορές περιγράφονται από τα Κρυφά Μαρκοβιανά Μοντέλα.

Βήματα Προπαρασκευής

Για τις ανάγκες του εργαστηρίου, εγκαταστάθηκε το εργαλείο Kaldi στο περιβάλλον WLS2 (Windows Subsystem for Linux). Χρειάστηκε, επίσης, να εγκατασταθεί το πακέτο της Python 2 [1] και συγκεκριμένα πακέτα που υποδείχθηκαν από το script check_dependencies.sh.

Τέλος, εγκαταστάθηκαν τα απαραίτητα δεδομένα των ηχογραφήσεων από τους τέσσερις ομιλητές με ονόματα: m1, m3 (άντρες) και f1, f5 (γυναίκες).

Στον φάκελο egs/project1 δημιουργήθηκαν οι υποφάκελοι data/train, data/dev και data/test. Στην συνέχεια, δημιουργήθηκαν τα απαραίτητα αρχεία uttids, utt2spk, wav.scp και text σε κάθε υποφάκελο. Για να επιτευχθεί αυτό:

uttids

Με το bash command cp [2], τα περιεχόμενα των αρχείων testing.txt, training.txt και validation.txt από τον φάκελο filesets αντιγράφηκαν στα αντίστοιχα αρχεία uttids στους φακέλους data/test, data/train και data/dev, αντίστοιχα.

utt2spk

Εκτελέστηκε το bash script kaldi_utt2spk_prep.sh [3]. Ο κώδικας διαβάζει κάθε utterance_id από το αρχείο uttids και εξάγει το speaker_id από τα δύο πρώτα του γράμματα (speaker_id="${utterance_id:0:2}"). Στη συνέχεια, προσθέτει την αντιστοίχιση utterance_id speaker_id στο αρχείο utt2spk.

wav.scp

Εκτελέστηκε το bash script kaldi_wav_prep.sh [4]. Ο κώδικας διαβάζει κάθε utterance_id από το αρχείο uttids, δημιουργεί τη διαδρομή του αντίστοιχου αρχείου ήχου (wav_path="$WAV_DIR/$utterance_id.wav") και προσθέτει την αντιστοίχιση utterance_id wav_path στο αρχείο wav.scp.

text

Εκτελέστηκε το bash script kaldi_text_prep.sh [5]. Ο κώδικας διαβάζει το αρχείο transcriptions.txt και αποθηκεύει τις μεταγραφές σε έναν πίνακα (declare -A transcriptions), χρησιμοποιώντας το αριθμητικό του utterance_id ως κλειδί. Για κάθε utterance_id από το αρχείο uttids, αναζητά τη σχετική εγγραφή και την προσθέτει στο αρχείο text.

Τέλος, εκτελέστηκε το Python script kaldi_lexicon_prep.py [6], το οποίο αντικατέστησε τις λέξεις στις προτάσεις με τις αντίστοιχες ακολουθίες φωνημάτων του λεξικού lexicon.txt. Ο κώδικας χρησιμοποιεί τη to_lexicon() για να φορτώσει τις αντιστοιχίες λέξεων-φωνημάτων και τη prepare() για να μετατρέπει το κείμενο σε πεζά και να αφαιρεί μη αλφαβητικούς χαρακτήρες. Στη συνέχεια, η to_phonemes() αντικαθιστά κάθε λέξη, ενώ η to_files() εφαρμόζει αυτή τη μετατροπή σε όλα τα αρχεία text των συνόλων train, dev και test, αποθηκεύοντας το αποτέλεσμα στα αντίστοιχα final_text.

Με το bash command mv [7], τα αρχεία final_text μετονομάζονται σε text, ενώ τα αρχικά αρχεία text μετονομάζονται σε previous_text.

Βήματα Κυρίως Μέρους

Για τα βήματα του εργαστηρίου ο κώδικας, συνοδευόμενος από σύντομα σχόλια, παρουσιάζεται σε bash scripts προκειμένου να υπάρχει ευκολία στον έλεγχο.

Προετοιμασία διαδικασίας αναγνώρισης φωνής για τη USC-TIMIT

1. Με το bash command cp, τα αρχεία path.sh και cmd.sh αντιγράφηκαν από τον φάκελο /home/zeno/kaldi/egs/wsj/s5 στον φάκελο του εργαστηρίου. Στα αρχεία πραγματοποιήθηκαν οι απαραίτητες τροποποιήσεις: στο path.sh ορίστηκε σωστά ο φάκελος εγκατάστασης του Kaldi [7], ενώ στο cmd.sh ενημερώθηκαν οι τιμές των μεταβλητών train_cmd, decode_cmd και cuda_cmd ώστε να χρησιμοποιούν run.pl [8].
2. Με το bash command ln -s [9], δημιουργήθηκαν τα soft links steps και utils στον φάκελο του εργαστηρίου.
3. Με το bash command ln -s, δημιουργήθηκε το soft link score_kaldi.sh στον φάκελο /home/zeno/kaldi/egs/project1/data/local.
4. Το αρχείο mfcc.conf δημιουργήθηκε με το περιεχόμενο που προέρχεται από το αποθετήριο Git του εργαστηρίου.
5. Τέλος, δημιουργήθηκαν οι απαραίτητοι υποφάκελοι data/lang, data/local/dict, data/local/lm_tmp και data/local/nist_lm.

Από εδώ και στο εξής, είναι σημαντικό να γίνεται source ./path.sh στην αρχή κάθε bash script, προκειμένου να είναι διαθέσιμες όλες οι εντολές του Kaldi.

Προετοιμασία γλωσσικού μοντέλου

Τα προκαταρκτικά βήματα για τη δημιουργία των απαραίτητων αρχείων για το γλωσσικό μοντέλο εκτελέστηκαν μέσω του bash script kaldi_4_2_1.sh [10]. Ο κώδικας εξηγείται λεπτομερώς μέσω σχολίων.

Build

Εκτελέστηκε το bash script kaldi_4_2_2.sh [11] προκειμένου να δημιουργηθεί η ενδιάμεση μορφή του γλωσσικού μοντέλου. Χρησιμοποιήθηκε η εντολή build-lm.sh του πακέτου IRSTLM για μοντέλα δύο τύπων, για n=1 (unigram) και n=2 (bigram).

Compile

Εκτελέστηκε το bash script kaldi_4_2_3.sh [12] προκειμένου να δημιουργηθεί η τελική ARPA μορφή του γλωσσικού μοντέλου. Χρησιμοποιήθηκε η εντολή compile-lm του πακέτου IRSTLM.

Language Finite State Transducer

Εκτελέστηκε το bash script kaldi_4_2_4.sh [13] προκειμένου να δημιουργηθεί το FST του λεξικού της γλώσσας. Χρησιμοποιήθηκε η εντολή prepare_lang.sh με ορίσματα τα <dict-src-dir> <oov-dict-entry> <tmp-dir> <lang-dir>. Κατά την εκτέλεση του script, στον φάκελο /home/zeno/kaldi/egs/project1/data/lang δημιουργούνται τα δύο αρχεία FST: L.fst και L_disambig.fst, καθώς και τα αρχεία oov.txt, phones.txt, και words.txt.

oov.txt

Περιλαμβάνει τις λέξεις εκτός λεξικού (out of vocabulary), δηλαδή τις λέξεις που δεν αναγνωρίζονται από το λεξικό.

phones.txt

Περιλαμβάνει τους φθόγγους που χρησιμοποιούνται στο λεξικό για την επεξεργασία και αναγνώριση φωνής.

words.txt

Περιλαμβάνει τις λέξεις που χρησιμοποιούνται στο λεξικό με τις αναπαραστάσεις τους σε φθόγγους.

Επίσης, τα αρχεία wav.scp, text και utt2spk ταξινομήθηκαν στους φακέλους data/test, data/train και data/dev, αντίστοιχα, εκτελώντας το bash script kaldi_4_2_5.sh [14].

utt2spk

Εκτελέστηκε το script utils/utt2spk_to_spk2utt.pl [15]. Δημιουργήθηκε το ζητούμενο αρχείο spk2utt στους φακέλους data/test, data/train και data/dev, αντίστοιχα. Το αρχείο spk2utt είναι το αντίστροφο του utt2spk. Δείχνει ποιες προτάσεις ανήκουν σε κάθε ομιλητή, αντί ποιος ομιλητής αντιστοιχεί σε ποια πρόταση.

Grammar Finite State Transducer

Το αρχείο timit_format_data.sh [16] δημιουργήθηκε με περιεχόμενο από το αποθετήριο Git του εργαστηρίου. Έγιναν ελάχιστες τροποποιήσεις, αποκλειστικά για τη σωστή προσαρμογή των paths. Κατά την εκτέλεση του script, για καθένα από τα έξι μοντέλα (που προκύπτουν από τα τρία στάδια: dev, train, test, για τα δύο unigram και bigram), δημιουργείται το FST της γραμματικής: G.fst.

Συνολικά, τα αυτόματα L.fst και G.fst συνδυάζονται για την επιτυχή αναγνώριση φωνής. Το L.fst αντιστοιχεί λέξεις σε ακολουθίες φωνημάτων, ενώ το G.fst κωδικοποιεί τις πιθανότητες ακολουθιών λέξεων από τα γλωσσικά μοντέλα n-grams.

Ερώτημα 1

Η αξιολόγηση μεγάλων και πολύπλοκων μοντέλων επεξεργασίας φωνής είναι πράξη συχνά πολύ δαπανηρή. Για το λόγο αυτό, χρησιμοποιούνται μετρικές, όπως το perplexity, που μπορούν γρήγορα να αξιολογούν τις πιθανές βελτιώσεις των μοντέλων γιατί είναι ανεξάρτητες από οποιαδήποτε συγκεκριμένη εφαρμογή.

Το perplexity δείχνει πόσο "μπερδεμένο" είναι το μοντέλο όταν προβλέπει την επόμενη λέξη (token). Όσο χαμηλότερη είναι αυτή η τιμή, τόσο πιο σίγουρο είναι το μοντέλο, καθώς θεωρεί λιγότερες λέξεις (token) πιθανές να ακολουθήσουν. Μαθηματικά, περιγράφεται ως την αντίστροφη πιθανότητα του test set, κανονικοποιημένη ως προς τον αριθμό των λέξεων (token).

Εκτελέστηκε το bash script kaldi_performance.sh [16] και υπολογίζεται το perplexity στο validation και στο test set.

N-gram	Set	Perplexity
Unigram	Validation	54.61
Bigram	Validation	39.54
Unigram	Test	56.04
Bigram	Test	38.15

Προκύπτει πως όσο αυξάνεται το παράθυρο (context) των λέξεων (token) που “βλέπει” το μοντέλο n-gram, όσο περισσότερη πληροφορία έχει, τόσο υψηλότερη πιθανότητα αποδίδει σωστά στην αλληλουχία των λέξεων. Ένα bigram μοντέλο μπορεί καλύτερα να προβλέψει τι θα ακολουθήσει και δίνει υψηλότερη πιθανότητα. Και όσο υψηλότερη πιθανότητα, τόσο χαμηλότερη η μετρική του perplexity.

Εξαγωγή ακουστικών χαρακτηριστικών

Εκτελέστηκε το bash script kaldi_4_3.sh [17] για την εξαγωγή των MFCCs και την εκτέλεση του Cepstral Mean and Variance Normalization (CMVN) στα τρία σύνολα δεδομένων. Το script make_mfcc.sh υπολογίζει τα MFCC χρησιμοποιώντας τις παραμέτρους του αρχείου mfcc.conf. Έπειτα, το script compute_cmvn_stats.sh εφαρμόζει το CMVN για να αφαιρέσει τη μέση τιμή και να κανονικοποιήσει τη διακύμανση των MFCCs.

Ερώτημα 2

Το σήμα της ομιλίας συχνά περιέχει θόρυβο και παρεμβολές από το περιβάλλον ή άλλες πηγές. Επομένως, είναι απαραίτητο να εφαρμόσουμε τεχνικές αποθορυβοποίησης κατά την επεξεργασία της φωνής, ώστε να βελτιώσουμε την ακρίβεια της αναγνώρισης ομιλίας.

Fourier Transform

Μαθηματικά, δίνεται το σήμα εισόδου $x[n]$ και το φίλτρο απόκρισης $h[n]$. Τότε, η συνέλιξη τους είναι $y[n] = x[n] * h[n]$. Στο φάσμα των συχνοτήτων, από τις ιδιότητες του μετασχηματισμού Fourier, είναι $Y[f] = X[f]\,H[f]$.

Cepstrum

Υπολογίζεται το cepstrum μέσω του λογαρίθμου του φάσματος. Δηλαδή, είναι $Y[\tau] = log(X[f]\,H[f]) = log(X[f]) + log(H[f]) = X[\tau] + H[\tau]$.

Στο φάσμα των συχνοτήτων, οποιοσδήποτε θόρυβος προστίθεται αθροιστικά στο σήμα εισόδου. Υποθέτοντας ότι η παλμική απόκριση $h[n]$ παραμένει σταθερή για κάθε τυχαίο πλαίσιο σαν στατικά μεγέθη, μπορούμε να αναλύσουμε το i-οστό πλαίσιο ως: $Y_i[\tau] = X_i[\tau] + H[\tau]$.

Cepstral Mean

Υπολογίζεται ο μέσος όρος των πλαισιών. Δηλαδή, είναι $\frac{1}{N}\sum_i Y[\tau]_i = \frac{1}{N}\sum_i \mathrm{X}[\tau]_i + H[\tau]$.

Τότε, ορίζεται η διαφορά του i-οστού πλαισίου από του μέσου όρου, η οποία αποδεικνύεται πως ισούται με την διαφορά του $X_i[\tau]$ από τον μέσο όρο των παραθύρων:

$$Y_i[\tau] - \frac{1}{N}\sum_i Y[\tau]_i = X_i[\tau] + H[\tau] - (\frac{1}{N}\sum_i \mathrm{X}[\tau]_i + H[\tau]) = X_i[\tau] - \frac{1}{N}\sum_i \mathrm{X}[\tau]_i.$$

Συμπερασματικά, το Cepstral Mean and Variance Normalization (CMVN) είναι χρήσιμο, καθώς εξασφαλίζει ότι το σήμα στο νέο πεδίο είναι απαλλαγμένο από την επίδραση του θορύβου, βελτιώνοντας έτσι την ακρίβεια της επεξεργασίας της φωνής.

Ερώτημα 3

Με τα bash command [18] και, συγκεκριμένα, τις εντολές feat-to-dim και feat-to-len προκύπτει πως η διάσταση των χαρακτηριστικών MFCC είναι δεκατρία (13) και τα ακουστικά frames που εξήχθησαν για κάθε μία από τις πέντε πρώτες προτάσεις του training set, που αντιστοιχούν στον ομιλητή f1, είναι 317, 371, 399, 328, και 464 αντίστοιχα. Το πλήθος των ακουστικών frames αποθηκεύτηκε στο αρχείο raw_mfcc_train.1.len στον φάκελο /home/zeno/kaldi/egs/project1/data/train/mfcc.

```
zeno@George:~/kaldi/egs/project1/data$ . ~/kaldi/egs/wsj/s5/path.sh
zeno@George:~/kaldi/egs/project1/data$ feat-to-dim ark:/home/zeno/kaldi/egs/project1/
feat-to-dim ark:/home/zeno/kaldi/egs/project1/data/train/mfcc/raw_mfcc_train.1.ark -
13
```

```
zeno@George:~/kaldi/egs/project1/data/train/mfcc$ cat raw_mfcc_train.1.len
f1_003 317
f1_004 371
f1_005 399
f1_007 328
f1_008 464
```

Εκπαίδευση ακουστικών μοντέλων και αποκωδικοποίηση προτάσεων

Monophone GMM-HMM

Το monophone GMM-HMM είναι μια βασική προσέγγιση στην αναγνώριση φωνής, όπου κάθε φώνημα μοντελοποιείται ως ένα ανεξάρτητο Κρυφό Μαρκοβιανό Μοντέλο (HMM) με Gaussian Mixture Models (GMMs) για την κατανομή των ακουστικών χαρακτηριστικών. Τα monophone αποτελούν το αρχικό στάδιο εκπαίδευσης σε σύγχρονα συστήματα πριν από την εισαγωγή πιο πολύπλοκων μοντέλων όπως τα triphone, τα οποία λαμβάνουν υπόψη κάποιο context.

Εκτελέστηκε το bash script kaldi_4_4_1.sh [19] προκειμένου να δημιουργηθεί το monophone. Χρησιμοποιήθηκε το script /steps/train_mono.sh με ορίσματα τα <data-dir> <lang-dir> <exp-dir>. Έπειτα, με το bash script /gmmbin/gmm-info έγινε κατανοητό πως το ακουστικό μοντέλο προσδιόρισε 165 φωνήματα (phones), έμαθε 125 πιθανούς διακριτούς

πυρήνες Gaussian (PDFs), οι οποίοι αναπαριστούν τις στατιστικές κατανομές των χαρακτηριστικών. Περιλαμβάνει 505 καταστάσεις (states).

HCLG Graph

Εκτελέστηκε το bash script kaldi_4_4_2.sh [20] προκειμένου να δημιουργηθούν οι γράφοι HCLG για τα δύο μοντέλα unigram και bigram στο train set. Έπειτα, χρησιμοποιήθηκε η εντολή fstinfo και προκύπτει:

- Ο HCLG γράφος του unigram έχει 152 καταστάσεις και 392 μεταβάσεις, με 2 τελικές καταστάσεις και 64 ε εισόδους και 308 εξόδους. Είναι κυκλικός και προσβάσιμος, αλλά δεν είναι ταξινομημένος.
- Ο HCLG γράφος του bigram έχει 874 καταστάσεις και 4324 μεταβάσεις, με 3 τελικές καταστάσεις και 546 ε εισόδους και 1952 εξόδους. Είναι κυκλικός, αλλά δεν είναι ταξινομημένος.

Παρατηρείται πως ο γράφους του bigram έχει πολλές περισσότερες καταστάσεις και μεταβάσεις καθώς, εξαιτίας του παραθύρου (context) των λέξεων (token) που “βλέπει”, καταγράφει περισσότερους δυνατούς συνδυασμούς λέξεων, σε αντίθεση με το unigram που βασίζεται μόνο στην εμφάνιση των λέξεων χωριστά.

Viterbi

Εκτελέστηκε το bash script kaldi_4_4_3.sh [21]. Χρησιμοποιήθηκε το script /steps/decode.sh με ορίσματα τα <graph-dir> <data-dir> <decode-dir>.

Είναι σημαντικό να αναφερθεί πως στα αποτελέσματα της ανάλυσης του μοντέλου bigram, υπάρχουν προειδοποιήσεις για την πιθανή μη βέλτιστη παρουσία της σιωπής (sil) στην αρχή και το τέλος των προτάσεων. Θα χρειαστεί να λυθούν για να υπάρξουν βελτιώσεις στο μοντέλο.

Phone Error Rate

Η μετρική Phone Error Rate (PER) έχει υπολογιστεί και βρίσκεται στον υποφάκελο decode_test/scoring_kaldi και decode_ved/scoring_kaldi στο αρχείο best_wer κάθε μοντέλου.

N-gram	Set	PER
Unigram	Validation	53.35%
Bigram	Validation	46.71%
Unigram	Test	52.56%
Bigram	Test	**45.78%**

```
zeno@George:~/kaldi/egs/project1/data/kaldi_monophone/graph/u/decode_test/scoring_kaldi$ cat best_wer
%WER 52.56 [ 6482 / 12332, 127 ins, 3718 del, 2637 sub ] /home/zeno/kaldi/egs/project1/data/kaldi_mono
zeno@George:~/kaldi/egs/project1/data/kaldi_monophone/graph/u/decode_dev/scoring_kaldi$ cat best_wer
%WER 53.35 [ 2530 / 4742, 91 ins, 1355 del, 1084 sub ] /home/zeno/kaldi/egs/project1/data/kaldi_monop
zeno@George:~/kaldi/egs/project1/data/kaldi_monophone/graph/b/decode_test/scoring_kaldi$ cat best_wer
%WER 45.78 [ 5646 / 12332, 215 ins, 2552 del, 2879 sub ] /home/zeno/kaldi/egs/project1/data/kaldi_mono
```

```
zeno@George:~/kaldi/egs/project1/data/kaldi_monophone/graph/b/decode_dev/scoring_kaldi$ cat best_wer
%WER 46.71 [ 2215 / 4742, 127 ins, 922 del, 1166 sub ] /home/zeno/kaldi/egs/project1/data/kaldi_mono
```

Επίσης, οι 2 υπερπαράμετροι της διαδικασίας scoring σε αυτό το βήμα είναι οι min_lmwt και max_lmwt, οι οποίες αντιπροσωπεύουν το ελάχιστο και μέγιστο LM-weight για το lattice rescoring αντίστοιχα. Το ελάχιστο LM-weight είναι 7.

Triphone GMM-HMM

Εκτελέστηκε το bash script kaldi_4_4_5.sh [22] προκειμένου να εκπαιδεύσουμε το triphone μοντέλο, ακολουθώντας, έπειτα των εντολών steps/align_si.sh, steps/train_deltas.sh, την ίδια διαδικασία με πριν για το monophone. Αξίζει να σημειωθεί πως προκαταρκτικά το αρχείο final.mdl αντιγράφεται από τον φάκελο kaldi_triphone στους υποφακέλους graph/trigram/u και graph/trigram/b με το bash script cp και το αρχείο G.fst αντιγράφεται από τον φάκελο /data/lang_phones_ug/ στον /data/lang/. Η επιλογή του φακέλου lang_phones_ug δεν επηρεάζει το αποτέλεσμα, καθώς το αρχείο G.fst της γραμματικής είναι ίδιο για όλα τα σύνολα δεδομένων και για όλα τα μοντέλα n-gram.

Η μετρική PER έχει υπολογιστεί και βρίσκεται στον υποφάκελο decode_test/scoring_kaldi και decode_ved/scoring_kaldi στο αρχείο best_wer κάθε μοντέλου.

N-gram	Set	PER
Unigram	Validation	40.76%
Bigram	Validation	36.50%
Unigram	Test	40.78%
Bigram	Test	**35.61%**

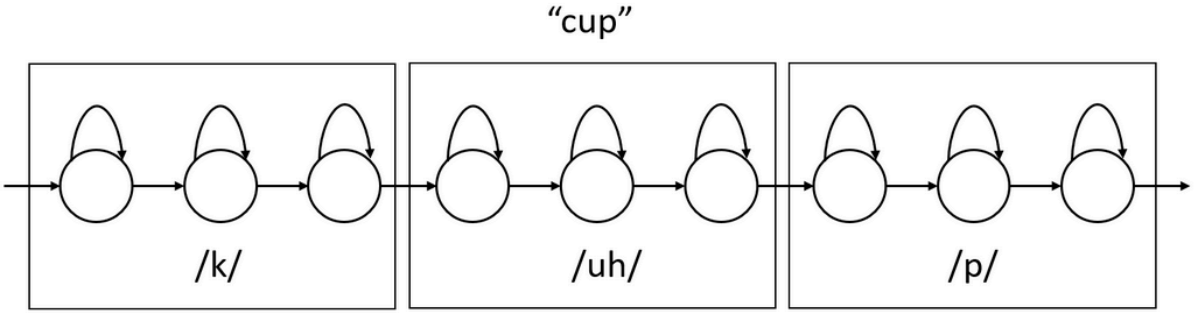
```
zeno@George:~/kaldi/egs/project1/data/kaldi_triphone/graph/trigram/u/decode_dev/scoring_kaldi$ cat best_wer
%WER 40.76 [ 1933 / 4742, 272 ins, 545 del, 1116 sub ] /home/zeno/kaldi/egs/project1/data/kaldi_triphone/gr
zeno@George:~/kaldi/egs/project1/data/kaldi_triphone/graph/trigram/u/decode_test/scoring_kaldi$ cat best_wer
%WER 40.78 [ 5029 / 12332, 504 ins, 1952 del, 2573 sub ] /home/zeno/kaldi/egs/project1/data/kaldi_triphone/g
zeno@George:~/kaldi/egs/project1/data/kaldi_triphone/graph/trigram/b/decode_dev/scoring_kaldi$ cat best_wer
%WER 36.50 [ 1731 / 4742, 269 ins, 419 del, 1043 sub ] /home/zeno/kaldi/egs/project1/data/kaldi_triphone/gr
zeno@George:~/kaldi/egs/project1/data/kaldi_triphone/graph/trigram/b/decode_test/scoring_kaldi$ cat best_wer
%WER 35.61 [ 4392 / 12332, 508 ins, 1382 del, 2502 sub ] /home/zeno/kaldi/egs/project1/data/kaldi_triphone/g
```

Η νέα αποκωδικοποίηση στο triphone έχει οδηγήσει σε βελτιωμένα αποτελέσματα, με τις μετρικές PER να εμφανίζουν μείωση περίπου 5% σε κάθε μοντέλο και σε κάθε σύνολο δεδομένων.

Ερώτημα 4

Όπως έχει αναλυθεί προηγουμένως, ένα ακουστικό μοντέλο GMM-HMM μοντελοποεί κάθε φώνημα ως ένα ανεξάρτητο Κρυφό Μαρκοβιανό Μοντέλο (HMM) με Gaussian Mixture Models (GMMs) για την κατανομή των ακουστικών χαρακτηριστικών.

Συγκεκριμένα, τα Μαρκοβιανά Μοντέλα αποτελούν αυτόματα πεπερασμένων καταστάσεων. Έχουν ως βάρη στις μεταβάσεις των καταστάσεων τους τις πιθανότητες μετάβασης μεταξύ των φωνημάτων σε λέξεις.



Η εικόνα απεικονίζει τη φωνητική αποσύνθεση της αγγλικής λέξης "cup" σε τρία μεμονωμένα φωνήματα: /k/, /uh/, και /p/. Τα Μαρκοβιανά Μοντέλα επεξεργάζονται κάθε φώνημα ξεχωριστά για ακριβή αποκωδικοποίηση και οι μεταβάσεις μεταξύ των καταστάσεων αντιπροσωπεύουν τις πιθανότητες μετάβασης από το ένα φώνημα στο επόμενο. Τα μίγματα γκαουσιανών κατανομών χρησιμοποιούνται στα αυτόματα για την ομοδοποίηση φωνημάτων σε υποκατηγορίες, βασισμένα στην πιθανότητα εμφάνισής τους σε διάφορες λέξεις.

Η εκπαίδευση ενός μονοφωνικού μοντέλου γίνεται μέσω καταλλήλων αλγορίθμων, οι οποίοι βελτιστοποιούν τις παραμέτρους των κατανομών, χρησιμοποιώντας τις ετικέτες των φωνημάτων στο train set για να προσαρμόσει τις παραμέτρους του μοντέλου. Επαναλαμβάνεται η διαδικασία μέχρι η ακρίβεια να βελτιωθεί.

Ερώτημα 5

Τα γλωσσικά μοντέλα είναι μοντέλα μηχανικής μάθησης που εκπαιδεύονται για να υπολογίζουν την a posteriori πιθανότητα της επόμενης λέξης. Σύμφωνα με τον τύπο του Bayes, υπολογίζεται η a posteriori πιθανότητα της λέξης W δεδομένης του διανύσματος των παρατηρήσεων X.

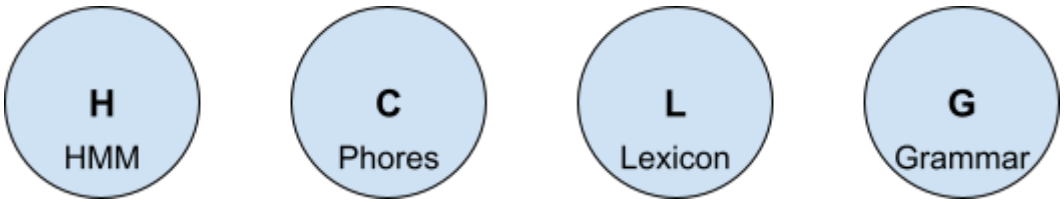
$$P(W|X) = \frac{P(W)\,P(X|W)}{P(X)}$$

Η πιθανότητα $P(W)$ μοντελοποεί το γλωσσικό μοντέλο και η δεσμευμένη πιθανότητα $P(X|W)$ το ακουστικό μοντέλο. Η πιο πιθανή λέξη W για την παρατηρούμενη ακολουθία ακουστικών χαρακτηριστικών X υπολογίζεται με το maximum likelihood ως εξής:

$$W = argmax P(W|X)$$

Ερώτημα 6

Η δομή του γράφου HCLG του Kaldi είναι μία ακολουθία τεσσάρων γράφων.



Οι γράφοι είναι:

- H είναι η απεικόνιση μεταβάσεων ενός Κρυφού Μαρκοβιανού Μοντέλου στον επιθυμητό χώρο των context-dependent label.
- C είναι η απεικόνιση μεταβάσεων των context-dependent label στα φωνήματα.
- L είναι το λεξικό, όπου στην περίπτωση του Kaldi και της αναγνώρισης των φωνημάτων, και μια αντιστοιχία των φωνημάτων με τον εαυτό τους.
- G είναι ο αποδοχέας, όπου κωδικοποιεί το γλωσσικό μοντέλο.

Επίσης, οι τελευταίοι δύο γράφοι κάνουν reweight.

Βελτιώσεις

Το μοντέλο επεξεργασίας και αναγνώρισης φωνής μπορεί να βελτιωθεί με τους εξής τρόπους:

- Να χρησιμοποιηθεί n-gram υψηλότερης τάξης, δηλαδή, να αναπτυχθεί ένα μοντέλο με μεγαλύτερο παράθυρο (context) και, άρα, μικρότερες τιμές perplexity.
- Να εμπλουτιστούν τα δεδομένα εκπαίδευσης με περισσότερες ηχογραφήσεις από διαφορετικούς ομιλητές, διαλέκτους και θορυβώδη περιβάλλοντα, βελτιώνοντας έτσι τη γενίκευση του μοντέλου.
- Να εφαρμοστούν τεχνικές κανονικοποίησης και επαύξησης δεδομένων, όπως data augmentation.

Παράρτημα

1.

```
sudo add-apt-repository ppa:deadsnakes/ppa -y
sudo apt update
sudo apt install python2 -y
python2 --version
```

2.

```
cp filesets/training.txt /home/zeno/kaldi/egs/project1/data/train/uttids
cp filesets/validation.txt /home/zeno/kaldi/egs/project1/data/dev/uttids
cp filesets/testing.txt /home/zeno/kaldi/egs/project1/data/test/uttids
```

3.

```
#!/bin/bash

FOLDER_DIR="."

echo "In Progress.."

for subset in train dev test; do
```

```
    echo "Processing $subset..."

    uttids_file="$FOLDER_DIR/$subset/uttids"
    utt2spk_file="$FOLDER_DIR/$subset/utt2spk"

    if [[ ! -f "$uttids_file" ]]; then
        echo "Error: $uttids_file not found!"
        continue
    fi

    # Adding
    while IFS= read -r utterance_id; do
        # Extracting speaker_id from the utterance_id
        speaker_id="${utterance_id:0:2}"

        echo "$utterance_id $speaker_id" >> "$utt2spk_file"
    done < "$uttids_file"

    echo "$subset processing completed."
done

echo "All subsets processed."
```

4.

```
#!/bin/bash

FOLDER_DIR="."
WAV_DIR="/home/zeno/kaldi/egs/project1/dataset/usc/wav"

echo "In Progress.."

for subset in train dev test; do
    echo "Processing $subset..."

    uttids_file="$FOLDER_DIR/$subset/uttids"
    wav_scp_file="$FOLDER_DIR/$subset/wav.scp"

    if [[ ! -f "$uttids_file" ]]; then
        echo "Error: $uttids_file not found!"
        continue
    fi

    # Adding
    while IFS= read -r utterance_id; do

        wav_path="$WAV_DIR/$utterance_id.wav"

        echo "$utterance_id $wav_path" >> "$wav_scp_file"
    done < "$uttids_file"

    echo "$subset processing completed."
done

echo "All subsets processed."
```

5.

```
#!/bin/bash

emove leading spaces
        utt_id=$(echo "$line" | awk '{print $1}')
        text=$(echo "$line" | sed -E 's/^[0-9]+[[:space:]]+//')

        # Store the transcription text in the array, with the numeric ID as the key
        transcriptions["$utt_id"]="$text"
    done < "$transcriptions_file"
    > "$text_file"

    while IFS= read -r utterance_id; do
        # Extract the numeric part of the utterance_id (e.g., m5_330 -> 330)
        numeric_id="${utterance_id##*_}"
        text="${transcriptions[$numeric_id]}"

        if [[ -n "$text" ]]; then
            echo "$utterance_id $text" >> "$text_file"
        else
            echo "Warning: Missing transcription for $utterance_id"
        fi
    done < "$uttids_file"

    echo "$subset processing completed."
done

echo "All subsets processed."
```

6.

```
import string

def prepare(text):
    text = text.lower()
    text = ''.join(char for char in text if char in string.ascii_lowercase + ' ');
    words = text.split() # Split
    return words

def to_lexicon(lexicon_file):
    lexicon = {}

    with open(lexicon_file, 'r') as file:
        for line in file:
            parts = line.strip().split()
            word = parts[0].lower()
            phonemes = ' '.join(parts[1:])
            lexicon[word] = phonemes

    return lexicon

def to_phonemes(sentence, lexicon):
    words = prepare(sentence)
    phoneme_sentence = ['sil']  # Add sil
```

```
    for word in words:
        if word in lexicon:
            phoneme_sentence.append(lexicon[word])
        else:
            phoneme_sentence.append('sil')  # Add sil

    phoneme_sentence.append('sil')  # Add sil
    return ' '.join(phoneme_sentence)

def to_files(lexicon_file, input_file, output_file):
    lexicon = to_lexicon(lexicon_file)

    for subset in ['train', 'dev', 'test']:
        text_file = f'{input_file}/{subset}/text'
        output_file = f'{output_file}/{subset}/final_text'

        with open(text_file, 'r') as file:
            lines = file.readlines()

        with open(output_file, 'w') as output:
            for line in lines:
                utt_id, sentence = line.strip().split(' ', 1)
                phoneme_sentence = to_phonemes(sentence, lexicon)
                output.write(f'{utt_id} {phoneme_sentence}\n')

lexicon_file = '/home/zeno/kaldi/egs/project1/dataset/usc/lexicon.txt'
input_file = '/home/zeno/kaldi/egs/project1/data'
output_file = '/home/zeno/kaldi/egs/project1/data'

to_files(lexicon_file, input_file, output_file)
```

7.

```
export KALDI_ROOT=/home/zeno/kaldi
…
```

8.

```
export train_cmd=run.pl
export decode_cmd=run.pl # --mem 2G"
# the use of cuda_cmd is deprecated, used only in 'nnet1',
export cuda_cmd=run.pl #--gpu 1"

if [ "$(hostname -d)" == "fit.vutbr.cz" ]; then
  queue_conf=$HOME/queue_conf/default.conf # see example
/homes/kazi/iveselyk/queue_conf/default.conf,
  export train_cmd="run.pl --config $queue_conf --mem 2G --matylda 0.2"
  export decode_cmd="run.pl --config $queue_conf --mem 3G --matylda 0.1"
  export cuda_cmd="run.pl --config $queue_conf --gpu 1 --mem 10G --tmp 40G"
fi
```

9.

```
ln -s /home/zeno/kaldi/egs/wsj/s5/steps steps
ln -s /home/zeno/kaldi/egs/wsj/s5/utils utils
```

10.

```
#!/bin/bash

# Source Kaldi's environment variables (sets up paths and tools)
. ./path.sh

# Define directory paths for dictionary files and datasets
target_dir="/home/zeno/kaldi/egs/project1/data/local/dict"
lexicon_dir="/home/zeno/kaldi/egs/project1/dataset/usc/lexicon.txt"
nonsilence_phones_dir="$target_dir/nonsilence_phones.txt"
new_lexicon_dir="$target_dir/lexicon.txt"
train_file="/home/zeno/kaldi/egs/project1/data/train/text"
dev_file="/home/zeno/kaldi/egs/project1/data/dev/text"
test_file="/home/zeno/kaldi/egs/project1/data/test/text"

# Create the 'dict' directory if it doesn't exist
mkdir -p $target_dir

# Create 'silence_phones.txt' and 'optional_silence.txt' with "sil" (silence token)
echo "sil" > $target_dir/silence_phones.txt
echo "sil" > $target_dir/optional_silence.txt

# Generate 'nonsilence_phones.txt':
# 1. Extract all non-silence phones from the lexicon (skip the first column, which is the
word).
# 2. Remove the "sil" entry (if present).
# 3. Sort and deduplicate the list.
awk '{for (i=2; i<=NF; i++) print $i}' "$lexicon_dir" | \
    grep -v '^sil$' | \
    sort -u > "$nonsilence_phones_dir"

# Create 'lexicon.txt':
# 1. Add a default entry for silence ("sil sil").
# 2. For each non-silence phone, add an entry mapping the phone to itself (e.g., "p p").
echo "sil sil" > $new_lexicon_dir
while read phone; do
    echo "$phone $phone" >> $new_lexicon_dir
done < $nonsilence_phones_dir

# Define output paths for language model (LM) training files
train_output="$target_dir/lm_train.text"
dev_output="$target_dir/lm_dev.text"
test_output="$target_dir/lm_test.text"

# Function to add <s> and </s> tokens to utterances in a dataset file
add_special_tokens() {
    input_file=$1    # Input file (e.g., train/text)
    output_file=$2   # Output file (e.g., lm_train.text)

    echo "Creating $output_file..."

    # Process each line:
    # 1. Extract the utterance ID (first field).
    # 2. Extract the phones (remaining fields).
    # 3. Wrap the phones with <s> and </s> tokens.
```

```
    while read -r line; do
        id=$(echo "$line" | awk '{print $1}')
        phones=$(echo "$line" | awk '{for(i=2;i<=NF;i++) printf "%s ", $i; print ""}')
        phones=$(echo "$phones" | sed 's/[[:space:]]*$//')  # Trim trailing whitespace
        echo "$id <s> $phones </s>" >> $output_file
    done < $input_file
}

# Process train, dev, and test files to add special tokens
add_special_tokens $train_file $train_output
add_special_tokens $dev_file $dev_output
add_special_tokens $test_file $test_output
```

11.

```
. ./path.sh

export IRSTLM=$KALDI_ROOT/tools/irstlm/
export PATH=${PATH}:$IRSTLM/bin

cd "/home/zeno/kaldi/egs/project1/data/local/dict"

build-lm.sh -i "lm_train.text" -n 1 -o lm_train1.ilm.gz
build-lm.sh -i "lm_train.text" -n 2 -o lm_train2.ilm.gz
build-lm.sh -i "lm_test.text" -n 1 -o lm_test1.ilm.gz
build-lm.sh -i "lm_test.text" -n 2 -o lm_test2.ilm.gz
build-lm.sh -i "lm_dev.text" -n 1 -o lm_dev1.ilm.gz
build-lm.sh -i "lm_dev.text" -n 2 -o lm_dev2.ilm.gz

mv lm_train1.ilm.gz ../lm_tmp
mv lm_train2.ilm.gz ../lm_tmp
mv lm_test1.ilm.gz ../lm_tmp
mv lm_test2.ilm.gz ../lm_tmp
mv lm_dev1.ilm.gz ../lm_tmp
mv lm_dev2.ilm.gz ../lm_tmp

cd "../../../"
```

12.

```
. ./path.sh

export IRSTLM=$KALDI_ROOT/tools/irstlm/
export PATH=${PATH}:$IRSTLM/bin

cd "/home/zeno/kaldi/egs/project1/data/local/lm_tmp"

compile-lm lm_train1.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c >
lm_phone_ug.arpa.gz
compile-lm lm_train2.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c >
lm_phone_bg.arpa.gz

compile-lm lm_test1.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c >
lm_phone_ug_test.arpa.gz
compile-lm lm_test2.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c >
lm_phone_bg_test.arpa.gz
```

```
compile-lm lm_dev1.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c >
lm_phone_ug_dev.arpa.gz
compile-lm lm_dev2.ilm.gz -t=yes /dev/stdout | grep -v unk | gzip -c >
lm_phone_bg_dev.arpa.gz

mv lm_phone_ug.arpa.gz ../nist_lm
mv lm_phone_bg.arpa.gz ../nist_lm
mv lm_phone_ug_test.arpa.gz ../nist_lm
mv lm_phone_bg_test.arpa.gz ../nist_lm
mv lm_phone_ug_dev.arpa.gz ../nist_lm
mv lm_phone_bg_dev.arpa.gz ../nist_lm

cd "../../../"
```

13.

```
. ./path.sh
. ./cmd.sh

export IRSTLM=$KALDI_ROOT/tools/irstlm/
export PATH=${PATH}:$IRSTLM/bin
./utils/prepare_lang.sh  /home/zeno/kaldi/egs/project1/data/local/dict "<oov>"
/home/zeno/kaldi/egs/project1/data/local/lm_tmp /home/zeno/kaldi/egs/project1/data/lang
```

14.

```
. ./path.sh
base_dir="/home/zeno/kaldi/egs/project1/data"

sort $base_dir/train/wav.scp -o $base_dir/train/wav.scp
sort $base_dir/train/text -o $base_dir/train/text
sort $base_dir/train/utt2spk -o $base_dir/train/utt2spk

sort $base_dir/dev/wav.scp -o $base_dir/dev/wav.scp
sort $base_dir/dev/text -o $base_dir/dev/text
sort $base_dir/dev/utt2spk -o $base_dir/dev/utt2spk

sort $base_dir/test/wav.scp -o $base_dir/test/wav.scp
sort $base_dir/test/text -o $base_dir/test/text
sort $base_dir/test/utt2spk -o $base_dir/test/utt2spk
```

15.

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

16.

```
#!/bin/bash

# Copyright 2013  (Author: Daniel Povey)
# Apache 2.0

# This script takes data prepared in a corpus-dependent way
```

```
# in data/local/, and converts it into the "canonical" form,
# in various subdirectories of data/, e.g. data/lang, data/train, etc.

. ./path.sh || exit 1;

echo "Preparing train, dev and test data"
lmdir=/home/zeno/kaldi/egs/project1/data/local/nist_lm
tmpdir=/home/zeno/kaldi/egs/project1/data/local/lm_tmp
lexicon=/home/zeno/kaldi/egs/project1/data/local/dict/lexicon.txt
mkdir -p $tmpdir

# Next, for each type of language model, create the corresponding FST
# and the corresponding lang_test_* directory.

echo "Preparing language models for test"

for lm_suffix in ug ug_dev ug_test bg bg_dev bg_test; do
  outlang=/home/zeno/kaldi/egs/project1/data//lang_phones_${lm_suffix}
  mkdir -p $outlang
  cp -r /home/zeno/kaldi/egs/project1/data//lang/* $outlang

  lm_file="$lmdir/lm_phone_${lm_suffix}.arpa.gz"

  if [[ -f "$lm_file" ]]; then
    gunzip -c "$lm_file" | \
      arpa2fst --disambig-symbol=#0 \
               --read-symbol-table=$outlang/words.txt - $outlang/G.fst
    fstisstochastic $outlang/G.fst
    # The output is like:
    # 9.14233e-05 -0.259833
    # we do expect the first of these 2 numbers to be close to zero (the second is
    # nonzero because the backoff weights make the states sum to >1).
    # Because of the <s> fiasco for these particular LMs, the first number is not
    # as close to zero as it could be.

    # Everything below is only for diagnostic.
    # Checking that G has no cycles with empty words on them (e.g. <s>, </s>);
    # this might cause determinization failure of CLG.
    # #0 is treated as an empty word.
    mkdir -p $tmpdir/g
    awk '{if(NF==1){ printf("0 0 %s %s\n", $1,$1); }} END{print "0 0 #0 #0"; print "0";}' \
      < "$lexicon"  >$tmpdir/g/select_empty.fst.txt
    fstcompile --isymbols=$outlang/words.txt --osymbols=$outlang/words.txt
$tmpdir/g/select_empty.fst.txt | \
     fstarcsort --sort_type=olabel | fstcompose - $outlang/G.fst > $tmpdir/g/empty_words.fst
    fstinfo $tmpdir/g/empty_words.fst | grep cyclic | grep -w 'y' &&
      echo "Language model has cycles with empty words" && exit 1
    rm -r $tmpdir/g

    utils/validate_lang.pl $outlang || exit 1
  else
    echo "Warning: Language model file $lm_file not found. Skipping..."
  fi
done
```

```
echo "Succeeded in formatting data."
rm -r $tmpdir
```

17.

```
. ./path.sh
. ./cmd.sh

cd /home/zeno/kaldi/egs/wsj/s5
data_dir="/home/zeno/kaldi/egs/project1/data"

for x in train test dev; do
    /home/zeno/kaldi/egs/wsj/s5/steps/make_mfcc.sh --mfcc-config
/home/zeno/kaldi/egs/project1/data/conf/mfcc.conf --nj $(nproc) --cmd run.pl $data_dir/$x
$data_dir/$x/logs $data_dir/$x/mfcc || exit 1;
    /home/zeno/kaldi/egs/wsj/s5/steps/compute_cmvn_stats.sh $data_dir/$x
$data_dir/$x/logs $data_dir/$x/mfcc || exit 1;
done
```

18.

```
. ~/kaldi/egs/wsj/s5/path.sh
feat-to-dim ark:/home/zeno/kaldi/egs/project1/data/train/mfcc/raw_mfcc_train.1.ark -
feat-to-len ark:/home/zeno/kaldi/egs/project1/data/train/mfcc/raw_mfcc_train.1.ark
ark,t:/home/zeno/kaldi/egs/project1/data/train/mfcc/raw_mfcc_train.1.len
```

19.

```
. ./path.sh
. ./cmd.sh

cd /home/zeno/kaldi/egs/wsj/s5
data_dir="/home/zeno/kaldi/egs/project1/data"

/home/zeno/kaldi/egs/wsj/s5/steps/train_mono.sh --nj $(nproc) --cmd run.pl \
    $data_dir/train $data_dir/lang $data_dir/kaldi_monophone || exit 1;

/home/zeno/kaldi/egs/wsj/s5/steps/align_si.sh --nj $(nproc) --cmd run.pl \
    $data_dir/train $data_dir/lang $data_dir/kaldi_monophone $data_dir/kaldi_monophone
```

20.

```
. ./path.sh
. ./cmd.sh

cd /home/zeno/kaldi/egs/wsj/s5
data_dir="/home/zeno/kaldi/egs/project1/data"

# Unigram
/home/zeno/kaldi/egs/wsj/s5/utils/mkgraph.sh --mono $data_dir/lang_phones_ug \
  $data_dir/kaldi_monophone $data_dir/kaldi_monophone/graph/u

# Bigram
/home/zeno/kaldi/egs/wsj/s5/utils/mkgraph.sh --mono $data_dir/lang_phones_bg \
  $data_dir/kaldi_monophone $data_dir/kaldi_monophone/graph/b
```

21.

```
. ./path.sh
. ./cmd.sh

cd /home/zeno/kaldi/egs/wsj/s5
data_dir="/home/zeno/kaldi/egs/project1/data"

# Usage: steps/decode.sh [options] <graph-dir> <data-dir> <decode-dir>
# Unigram Dev
/home/zeno/kaldi/egs/wsj/s5/steps/decode.sh $data_dir/kaldi_monophone/graph/u
$data_dir/dev $data_dir/kaldi_monophone/graph/u/decode_dev || exit 1;
# Unigram Test
/home/zeno/kaldi/egs/wsj/s5/steps/decode.sh $data_dir/kaldi_monophone/graph/u
$data_dir/test $data_dir/kaldi_monophone/graph/u/decode_test|| exit 1;

# Bigram Dev
/home/zeno/kaldi/egs/wsj/s5/steps/decode.sh $data_dir/kaldi_monophone/graph/b
$data_dir/dev $data_dir/kaldi_monophone/graph/b/decode_dev || exit 1;
# Bigram Test
/home/zeno/kaldi/egs/wsj/s5/steps/decode.sh $data_dir/kaldi_monophone/graph/b
$data_dir/test $data_dir/kaldi_monophone/graph/b/decode_test || exit 1;
```

22.

```
. ./path.sh
. ./cmd.sh

cd /home/zeno/kaldi/egs/wsj/s5
data_dir="/home/zeno/kaldi/egs/project1/data"

# Alignment Usage: steps/align_si.sh <data-dir> <lang-dir> <src-dir> <align-dir>
/home/zeno/kaldi/egs/wsj/s5/steps/align_si.sh --cmd "$train_cmd" \
  $data_dir/train $data_dir/lang $data_dir/kaldi_monophone $data_dir/kaldi_monophone ||
exit 1;

# Train Deltas Usage: steps/train_deltas.sh <num-leaves> <tot-gauss> <data-dir>
<lang-dir> <alignment-dir> <exp-dir>
/home/zeno/kaldi/egs/wsj/s5/steps/train_deltas.sh --cmd "$train_cmd" 2500 15000 \
  $data_dir/train $data_dir/lang $data_dir/kaldi_monophone/mono_ali
$data_dir/kaldi_triphone || exit 1;

# Unigram Graph Creation for Triphone Model
/home/zeno/kaldi/egs/wsj/s5/utils/mkgraph.sh $data_dir/lang_phones_ug \
  $data_dir/kaldi_triphone $data_dir/kaldi_triphone/graph/trigram/u || exit 1;

# Bigram Graph Creation for Triphone Model
/home/zeno/kaldi/egs/wsj/s5/utils/mkgraph.sh $data_dir/lang_phones_bg \
  $data_dir/kaldi_triphone $data_dir/kaldi_triphone/graph/trigram/b || exit 1;

# Decoding - Unigram (Dev and Test)
/home/zeno/kaldi/egs/wsj/s5/steps/decode.sh $data_dir/kaldi_triphone/graph/trigram/u \
  $data_dir/dev $data_dir/kaldi_triphone/graph/trigram/u/decode_dev || exit 1;

 /home/zeno/kaldi/egs/wsj/s5/steps/decode.sh $data_dir/kaldi_triphone/graph/trigram/u \
  $data_dir/test $data_dir/kaldi_triphone/graph/trigram/u/decode_test || exit 1;
```

```
# Decoding - Bigram (Dev and Test)
/home/zeno/kaldi/egs/wsj/s5/steps/decode.sh $data_dir/kaldi_triphone/graph/trigram/b \
  $data_dir/dev $data_dir/kaldi_triphone/graph/trigram/b/decode_dev || exit 1;

 /home/zeno/kaldi/egs/wsj/s5/steps/decode.sh $data_dir/kaldi_triphone/graph/trigram/b \
  $data_dir/test $data_dir/kaldi_triphone/graph/trigram/b/decode_test || exit 1;
```