Massachusetts Institute of Technology
Department of Electrical Engineering & Computer Science

6.345 Automatic Speech Recognition
Spring, 2003

Issued: 4/11/03
Due: 4/23/03

Assignment 9
An Introduction to the SUMMIT Speech Recognition System

Introduction

In this lab, we will train and test a complete speech recognition system from beginning to end using the sentences in the **Pegasus** domain (flight status inquiries). We will proceed step by step through the entire process, examining the information/data that is needed at each stage, building/training the necessary models, and evaluating the performance and behavior of the models.

In the process, you will also become familiar with the components of the SUMMIT speech recognition system so that you will be able to use it as the recognition engine in your term project.

As in previous assignments, the following tasks (marked by **T**'s) should be completed during your lab session. Answers to the questions (marked by **Q**'s) should be handed in on the due date.

Software

In this lab, we will use a suite of programs that together comprise the SUMMIT recognition system. We will be using version 3.7 of the system which is specified by setting the environment variable `SLS_HOME` to `/usr/sls/sls/v3.7`. Note that this is done for you automatically in the `.cshrc` file of the class accounts.

A descriptive listing of the programs used in this lab is given in the **SUMMIT Tools** appendix at the end of this handout. Please look over the description of the programs before you start the lab. Also be aware that some of the more computationally intensive components are designed to run in a distributed manner making use of the SLS group's *host dispatcher*. Look over the section on distributed processing in the appendix.

Getting Started

To get started with this lab, type the following command at the UNIX prompt:

```
% start_lab9.cmd
```

This will create a new subdirectory under your home directory called `lab9` and populate it with the set of files needed for this lab. These files contain data, models, and parameter specifications used during the lab. Descriptions of the files can be found in the **SUMMIT Files** appendix at the end of this handout. In the first part of the lab, we will look at some of these files in detail.

Part I: Data Files

In this part of the lab, we will examine the different data files needed to build a speech recognition system. In particular, we will look at the following:

- the partitioning of a corpus into disjoint training, development, and testing sets,
- audio waveforms,
- word and phonetic level transcriptions, and
- pronunciation dictionaries/lexicons.

Corpus

The SUMMIT system makes use of a **corpus** file which contains specifications of all the data for several domains. The corpus file has already been created and can be found in the file `galaxy.corpus`. The program `corpus_tool` can be used to examine the properties and sets defined in the corpus file.

T1: Display the usage of the `corpus_tool` program by supplying the `-help` command line option. The corpus is partitioned into several sets and for each set, there are several subsets. A subset corresponds to a single speaker (or phone call) and contains several utterances. Use the `-list_sets` option of the program to examine the partitioning of the corpus into different data sets:

```
% corpus_tool -corpus galaxy.corpus -list_sets
```

This will display a list of all the sets represented in `galaxy.corpus`. Note that the sets for **Pegasus** domain are named `p_*`. Use the `-set` option to display all the subsets available within a specific set:

```
% corpus_tool -corpus galaxy.corpus -set p_1
```

Similarly, you can specify a subset name at the `-set` argument. Successively examine smaller "data sets" until you reach the list of individual utterances or "datum." Use the `-datum` option to view the properties of an actual utterance. For example:

```
% corpus_tool -corpus galaxy.corpus \
              -datum p_pegasus-phone-258-0301-19990107-006-000
```

This will show the properties for the individual datum and the location of the waveform file.

Q1: **(a)** How many different data sets is the corpus partitioned into?

(b) How many speakers (or phone calls) are there in each **Pegasus** set?

(c) What are the properties defined in this corpus?

Waveforms

Notice that one of the properties in the corpus file is the specification of the location of the audio waveform file (`waveform_file`) for each utterance (datum) in the corpus. These waveform files contain the digitized speech. They are stored in NIST waveform format (`.wav`) with a 1024 byte ASCII header. We can look at the header information using the standard UNIX paging commands: `more` or `less`. We can listen to the audio file using the program `waveform_play`.

T2: Look at the header of one of the waveform files. Use `waveform_play` to listen to the waveform. (Remember to use `-help` if you need to see the usage.) Compare what you hear to the word-level transcription given in the `orthography` property.

Note that these sentences were orthographically transcribed by a human. One of the most time consuming tasks in the development of a speech recognition system is the collection and preparation of the data needed for training and testing.

Lexicons

In addition to speech waveforms and word-level transcriptions, some more information is needed in order to build a speech recognition system. This includes the vocabulary or list of words that the system can recognize and a phonetic transcription of the words. Building the vocabulary can be quite a tedious task considering the wide range of things that users can typically ask. The list of vocabulary words for this lab is given in the file `pegasus.vocab`.

Once a vocabulary is defined, the baseform (or canonical) pronunciation for each vocabulary word must be defined. The easiest way to obtain baseform pronunciations is to look them up in a dictionary. We utilize a tool called `baseform_tool.pl` to create our initial baseform pronunciation file.

T3: Generate a baseform file automatically by executing the following command:

```
% baseform_tool.pl -vocab pegasus.vocab \
                   -dict sls_pronlex.syl \
                   -convert_stops \
                   -rules baseform.rules \
                   -out pegasus.baseforms \
                   -baseform galaxy.baseforms
```

This `baseform_tool.pl` command looks up every word in `pegasus.vocab` and looks up its pre-defined baseform pronunciation in `galaxy.baseforms`. If the baseform cannot be found, it will look up the pronunciation in a pronunciation dictionary `sls_pronlex.syl` and try to generate the baseform. The command line option `-convert_stops` forces the script to convert all stop labels into a specific context-dependent allophonic variant which is based on the phonemes surrounding the stop, the stop's position in the syllable, and the stress pattern of the surrounding syllables. The `-rules` argument specifies a set of rewrite rules which removes stress information and applies other SLS phonemic unit conventions. The set of phonemic units used in SLS baseforms files are provided in the **Phonemic Unit Set** appendix.

Warning messages will be given if some words in the vocabulary cannot be found in the dictionary. You will need to manually enter these pronunciations if this happens.

T4: Examine the list of vocabulary words in `pegasus.vocab` and their phonetic transcriptions in `pegasus.baseforms`.

Q2: **(a)** What is the size of the recognition vocabulary?

(b) In the automatically generated `pegasus.baseforms` file, examine the pronunciation entries. Can you suggest some improvements to the pronunciations such that they can reflect better what might actually be spoken by users?

The basic phonetic transcriptions given in `pegasus.baseforms` are "expanded" (to multiple pronunciations) using a set of phonological rules. These rules allow for context-dependent variation in the phonetic realization of words. They usually have the largest impact at word boundaries, as word internal variation can often be represented in the baseform pronunciations. The phonological rules are contained in the file `pegasus.pron.rules`.

T5: Look at the pronunciation rules in the file `pegasus.pron.rules`. Try to understand the meaning of the rules. You don't have to know all the details of the rules for this lab.

Transcriptions

In order to train acoustic models for the subword phonetic units, which we will do in Part III of the lab, we need to find the corresponding segments in the acoustic waveform that map to the particular phones. In other words, we need a time-aligned phonetic transcription for each sentence in the training set. One way to do this is to perform what is known as *forced alignment*. For each sentence, we run it through the speech recognizer using an existing acoustic model and a *highly constrained* language model.

The acoustic model can be from another corpus, if we are just starting on a new domain and want to bootstrap off an existing domain. Alternatively, it can be the model from a prior training run of the current domain, if we are trying to iteratively improve the acoustic model. The language model is constrained to be only the allowable sequence of phones for the given word sequence in that particular training sentence.

The program `fst_trans_compute` (use `-shelp` for help messages) is used to perform the forced alignments. It makes use of the parameter specifications in the `paths.espec` file (See the **SUMMIT Files** appendix for a description of the content and format of this file). Forced word and phonetic level alignment transcription files (`.wrd` and `.phn`) for all the training sentences have already been created for you so you do **not** have to redo them. They are located in

```
/t/summit/forced_paths/6.345/<group>/<tag>.wrd
/t/summit/forced_paths/6.345/<group>/<tag>.phn
```

where `<group>` takes on values specified in the corresponding properties of the corpus file and `<tag>` is the label of that utterance (datum in the corpus).

We will graphically examine the alignment of the words and phones to the waveform for some utterances using the transcription view program `tv`. This program makes use of the parameter specifications in the `tv.espec` file. The existing espec file will randomly select 50 sentences from the `p_3` set to be examined.

T6: Look at several of the time-aligned word and phonetic transcription files:

```
/t/summit/forced_paths/pegasus/<group>/<tag>.wrd
/t/summit/forced_paths/pegasus/<group>/<tag>.phn
```

We will examine in **Sapphire** the alignment of the words and phones to the waveform. The program is invoked using the `tcl` script, `tv.tcl`:

```
% tv -ctl tv.espec
```

Make use of the information in the spectrogram as well as the capability to play selected segments of the waveform (Ctrl-v) that correspond to the words and phones to evaluate the quality of the word and phonetic alignments. There will, undoubtedly be some errors in the alignment.

Q3: Qualitatively, how good are the word and phonetic level alignments? What kinds of alignment errors did you see from your sample of sentences?

Part II: Language Models

To build a language model the system needs a vocabulary file and a set of training data. The system also has the option of using a set of rules which place words in the vocabulary file into word classes.

T7: Examine the file `train_sents.txt`. These training sentences are transcribed from spontaneous queries from actual calls to the **Pegasus** system. You should be able to find numerous occurrences of spontaneous speech effects such as partial words, filled pauses (um, uh), and ungrammatical speech present in the training data. Note that all acoustic events have been transcribed and not just the spoken words.

T8: Examine the file `pegasus.class.rules`. This file contains the word classes that have been defined for this domain. Can you think of any other classes that might be useful to model? Note that commonly used nouns, verbs, and function words are <u>not</u> placed into classes. Why is it helpful to create word classes for semantic objects such as cities and airline carriers?

You have explored some language models in **Assignment 6**. Here we will use a PERL script to create all the bigram and trigram language models needed by the recognizer.

T9: Train the **class** language models by executing the following command:

```
% ngram_create.pl -prefix pegasus -sents train_sents.txt \
                  -rules pegasus.class.rules
```

Later in this lab, you will create **word** language models by skipping the word class rules specified in the file `pegasus.class.rules`. You will have the chance to compare **word** n-gram and **class** n-gram models in recognition.

Part III: Acoustic Models

In this part of the lab, we will train and examine acoustic models. Specifically, we will:

- train a principal components rotation and scaling matrix,
- train mixture diagonal-covariance Gaussian models for the acoustic features of the different diphones, and
- look at some of the properties of the trained acoustic model.

We will do this for context-dependent diphone boundary models. The recognizer also has the capability to specify segment acoustic models. Similar training procedures are used to train the segment models. The file that "drives" the training and recognition process is called the recognition or "rec" file. For this lab, this file is `pegasus.rec`. This file lists the processing modules and their parameter specifications for each step in the training and testing process. (See the **SUMMIT Tools** appendix for pointers to descriptions of the processing modules and the **SUMMIT Files** appendix for a more detailed description of this file).

T10: Look at the contents of the `pegasus.rec` file and try to understand its content.

Q4: **(a)** How many samples are there in one frame (window) of the short-time Fourier transform (STFT)?

(b) How many MFCC coefficients are being computed?

(c) What are the components of the 112-dimensional feature vector that is being computed for each diphone boundary?

(d) What is the maximum allowable number of mixture components for each `mixture_diagonal_gaussian` model?

Diphone Clustering

Since we are using limited **Pegasus** domain training data in this lab, some diphones will not have enough training tokens from the training set. To alleviate such a sparse data problem, an automatic clustering procedure is used to allow similar diphones to share the same Gaussian mixture model. Since it takes a long time to perform this clustering procedure, a pre-clustered diphone label file `pegasus.blabels` is given to you. In this file, all the diphones in the same line share the same acoustic model. The tools used for clustering are explained in the **Diphone Clustering** appendix.

T11: Examine the `pegasus.blabels` file. Pay attention to the diphones that share the same acoustic model. Can you identify their similarities?

Principal Components Analysis and Scaling

A rotation and scaling of the feature vectors based on principal components analysis is done to decorrelate the components of the feature vectors and to adjust their variance to one. These operations make the features more conducive for modeling by mixtures of diagonal-covariance Gaussians. The transformation matrix is computed once from all the training data and is used to transform both the training and test data.

T12: Compute the principal components scaling (pcs) matrix on the training data using the program `train_pcs` as follows:

```
% train_pcs -newpcs pegasus.bpcs \
            -bound \
            -blabels pegasus.blabels
```

This program uses the information in `pegasus.rec` as well as parameter specifications in the `trans.espec` file. When we are training diphone models, we specify the `-bound` command line option. The program `pcs_tool` can be used to examine the transformation matrix.

NOTE: This program creates temporary files which should be cleaned up after you have successfully finished this lab. (See the **SUMMIT Files** appendix for a description of the temporary files and subdirectories.)

Acoustic Models

We can now train mixture diagonal-covariance Gaussian models for each diphone using the program `train_models`. This program uses the information in `pegasus.rec` as well as parameter specifications in the `trans.espec` file.

T13: Train acoustic models of the different diphones on the training data using the program `train_models` as follows:

```
% train_models -newmodels pegasus.bmodels \
               -pcs pegasus.bpcs \
               -bound \
               -blabels pegasus.blabels
```

Note that this program, like `train_pcs`, also creates temporary files. The temporary directory used is displayed at the beginning of the output messages. You should remember this temporary directory which will be used in **T15**.

Again, since we are training diphone models here, we specify the `-bound` flag. Other types of models (segment, for example) can also be trained using the same program.

After training the models, we can look at some of the statistics to see if the models are behaving properly.

T14: Use the program `models_tool` to examine the acoustic diphone model `pegasus.bmodels`.

Q5: **(a)** How many different models are there?

(b) Which diphone models do you think are trained well?

(c) Which ones are trained poorly?

We can also graphically examine how good the models are by superimposing scatter plots of the training data and two-dimensional "contour plots" of the Gaussian mixture models. This can be done using some MATLAB tools.

T15: We will convert the models and measurements into matlab `.mat` format via the following tools:

```
% models_to_matlab -in pegasus.bmodels \
                   -label "t(ah|m)" \
                   -out models.mat
```

In this example, we convert the model parameters for the diphone "t(ah|m)" (transition from [ah] to [m]) into matlab readable format. `models.mat` contains the parameters of the mixture diagonal Gaussian model of our desired diphone.

Next, load in the feature measurements for the same diphone from the temporary directory `temp/t1`. Note that the temporary directory may be different depending on how many different training runs you have performed. It should the directory used by the latest run of the `train_models` program. This is done using the tool `measurements_to_matlab`. It also takes in an argument in `-name` for the names of the data files in the specified temporary directory.

```
% measurements_to_matlab -directory temp/t1 \
                         -name bound_scores.vec \
                         -labels_file pegasus.blabels \
                         -label "t(ah|m)" \
                         -out measurements.mat
```

To graphically display the trained models, first start up MATLAB in a new window. We will load in the measurements as well as the model parameters and then create a scatter plot of the training data for two specified feature dimensions (dimension 1 and 2 in this example) and superimpose the contour plot of the corresponding mixture Gaussian model:

```
% matlab
>> load measurements
>> load models
>> plot_clusters(measurements,1,2); hold;
>> plot_mix_contour(model_means,model_variances,model_weights,1,2)
```

Part IV: Recognition and Performance

In this part of the lab, we will use the language and acoustic models we trained previously to perform recognition on a new set of sentences (the development set) and measure the recognition performance. Specifically, we will:

- perform 1-best recognition using the word and class bigram language models,
- perform N-best recognition using a word trigram language model, and

- measure recognition performance and examine recognition errors.

The program `fst_build_recognizer.cmd` is used to construct a full lexical finite state transducer (FST) representing the lexical search space. The FST construction process also involves the use of a set of phonological rules which expands each baseform pronunciation into all possible phonetic realizations for that baseform. The phonological rules are contained in the file `pegasus.pron.rules`. It also incorporates the language models we built.

The program `fst_test_rec_accuracy` is used to perform recognition. It uses the information in `pegasus.rec` as well as parameter specifications in the `pegasus.espec` file. Note the set definition in the `pegasus.espec` file:

```
set: <p_7>*<[artifact == no]>*<[contains_p_oov == no]>
```

It specifies sentences in the development set which do not have any artifacts or out of vocabulary (OOV) words.

1-Best Recognition

T16: Perform 1-best (Viterbi search) recognition on the development set using the **class** bigram language model we built in **T9**. First, construct the FST for the recognizer using the following command:

```
% fst_build_recognizer.cmd pegasus 0 0
```

In this command, the first argument is the domain name, the second argument is a word transition weight (WTW) which is applied to every word, and the third argument is a boolean value indicating whether a full trigram model should be used. We typically set WTW to zero in the network since the weight can also be controlled by an argument to the search operation in the `pegasus.rec` file. This weight is used to control the insertion/deletion properties of the recognizer.

Note that a collection of several FST files is actually created by the system. Each FST serves a different purpose in the full search. The primary FST file is called `pegasus.ffst`. This file contains the full FST network used for the initial forward search.

The `fst_test_rec_accuracy` command is used to test the recognition performance:

```
% fst_test_rec_accuracy -overwrite
```

Note that the `-overwrite` option is given to the command `fst_test_rec_accuracy` here to overwrite outputs from previous recognition runs.

The recognition is performed distributedly on a collection of SLS servers. You can specify `-local` option to force the recognition to be done on your local machine. Before you run the recognition in the distributed mode, you should use the `-local` option and make sure the recognizer is running properly.

Note that several files are generated by this program: a sentence hypothesis file (`.hyp`), a sentence reference file (`.ref`), and a performance summary file (`.sum`). The base name of

the files has the format: `<machine>_<process_id>`. The standard performance measure for a speech recognition system is word error rate which is composed of the sum of the following types of errors: substitutions, deletions, and insertions.

T17: Run recognition using the provided acoustic model file `galaxy.bmodels` which is trained on a much larger corpus:

```
% fst_test_rec_accuracy -bpcs galaxy.bpcs \
                        -bmodels galaxy.bmodels \
                        -blabels galaxy.blabels \
                        -overwrite
```

Q6: **(a)** Make a table of the values of the different errors (substitutions, deletions, insertions, and total error) for the two recognition experiments.

(b) Which one has better performance?

(c) Examine acoustic model `pegasus.bmodels` you just trained and the provided model `galaxy.bmodels` using `models_tool`. Give some reasons why one performs better than the other.

T18: Run recognition using a **word** bigram language model. You should rebuild the language models by skipping the class rule file:

```
% ngram_create.pl -prefix pegasus -sents train_sents.txt
```

Then rebuild the FSTs for the recognizer:

```
% fst_build_recognizer.cmd pegasus 0 0
```

You are now ready to evaluate the recognizer by issuing the following command:

```
% fst_test_rec_accuracy -overwrite
```

Q7: **(a)** Examine and compare the performance summary (`.sum`) files for the **word** and **class** bigram runs using the same acoustic model, `pegasus.bmodels`, and make a table of the values of the different errors (substitutions, deletions, insertions, and total error).

(b) Which one has better performance?

(c) Does this result agree with your conclusions based on the perplexity measurements in Assignment 6?

(d) From your examination of the performance summary files, what are the most frequently inserted words?

(e) What are the most frequently deleted words?

N-Best Recognition

T19: Perform N-best (A* search) recognition on the development set using both the **word** bigram and the **word** trigram language model.

Since we have just built the **word** n-gram language models by skipping the class rules, we can construct the recognizer FSTs with both the **word** bigram and the **word** trigram language models using the following command:

```
% fst_build_recognizer.cmd pegasus 0 1
```

Then, test the recognizer using the following command:

```
% fst_test_rec_accuracy -nbest 10 -overwrite
```

Note that in addition to the hypothesis (`.hyp`), reference (`.ref`), and performance summary (`.sum`) files, N-best sentence hypothesis files will also be created for each sentence. If you look at the `-nbest_out` specification in the `pegasus.espec` file, you will see that these files are located in the subdirectory `temp/nbest`.

T20: Repeat the N-best recognition using the **class** bigram and the **class** trigram language models. You will need to rebuild the language models using the class rules, rebuild the recognizer FSTs specifying the trigram option, and then test the recognizer:

```
% ngram_create.pl -prefix pegasus -sents train_sents.txt \
                  -rules pegasus.class.rules

% fst_build_recognizer.cmd pegasus 0 1

% fst_test_rec_accuracy -nbest 10 -overwrite
```

Examine the performance summary (`.sum`) files for these N-best recognition experiments. Also look at the N-best lists (in `temp/nbest`) and note how similar the top ten scoring sentence hypotheses are.

Q8: **(a)** What are the values of the different errors (substitutions, deletions, insertions, and total error) for these N-best recognition experiments?

(b) How does the performance of the class trigram compare with the word trigram?

(c) How does the performance compare to the previous recognition runs which didn't use the trigram language model?

Appendix: SUMMIT Tools

This section gives pointers to some available online documentation for the SUMMIT tools, provides a descriptive index of the tools used in the lab, and mentions some of the essentials of what you need to know in order to complete the lab.

Online Documentation

An index of many of the SUMMIT tools along with a high level description is available online and accessible via a web browser at the URL:

`file:/usr/sls/sls/v3.7/html/index.html`.

A descriptive index of some SAPPHIRE modules used by many of the SUMMIT tools, in particular the specifications in the `pegasus.rec` file, can be found at the URL:

`file:/usr/sls/sls/v3.7/html/sapphire.html`.

Note that these files are only accessible to machines within the SLS group.

Command Line Usage Help

Note that for most of the programs, supplying the command line argument `-help` (for regular programs) and/or `-shelp` (for Sapphire Tcl scripts) will list the program's usage along with valid command line arguments. This is useful if you forget the exact usage syntax or want to see what other options are available.

Distributed Processing

Some of the more computationally intensive components, such as `train_pcs`, `train_models`, and `fst_test_rec_accuracy` are designed to run in a distributed manner making use of the SLS group's *host dispatcher*. Note that these programs can be made to run *locally* on your workstation by specifying the command line argument `-local` when you run the program. When running locally, more diagnostic information is printed out to the screen. This "local mode" is useful for debugging experiments and making sure that they run properly before you distribute the jobs everywhere. The only program related to the host dispatcher that you need to know about is:

phd Shows the current status of hosts: `phd -status`.

A more detailed description of the host dispatcher can be found at these URLs:

```
file:/usr/sls/sls/v3.7/html/phd.html
file:/usr/sls/sls/v3.7/html/dispatch.html.
```

Program Index and Description

cfg_create Creates context free grammar (CFG) given a set of classes/rules (`.rules`) and a word lexicon (`.wlex`). The CFG is used in training the n-gram language models.

corpus_tool Displays and allows the modification of the properties and sets defined in an existing corpus file.

fst_test_rec_accuracy A Sapphire Tcl script to perform recognition and to evaluate the performance of the system. It uses the information in the domain rec file (e.g., `pegasus.rec`) as well as parameter specifications in the domain espec file (e.g., `pegasus.espec`). Many parameter values can also be specified via the command line. Use the `-shelp` option to see the usage. Several files are generated by this program: a sentence hypothesis file (`.hyp`), a sentence reference file (`.ref`), and a performance summary file (`.sum`). The basename of the files have the format: `<machine>_<process_id>`.

fst_trans_compute A Sapphire Tcl script to compute a set of forced paths by supplying the `-forced` command line option. It uses the information in the domain rec file (e.g., `pegasus.rec`) as well as parameter specifications in the `paths.espec` file. The output can be used to train PCS and acoustic models. This program defaults to running in distributed mode and will generate intermediate files in the temporary directory.

models_tool Displays various information about a trained model file. Use the command line options `-detailed` and `-very_detailed` to print increasing amounts of information.

ngram_create.pl Creates forward and reverse versions of both bigram and trigram models. Each of these models are used at various points in the recognizer search.

ngram_create_random_sentences Randomly generate sentences using the statistics in the specified n-gram language model. This is useful for qualitatively evaluating an n-gram model.

ngram_perplexity Compute the perplexity of a specified set of utterances using a given n-gram language model. Useful for getting a quantitative measure of the performance of an n-gram model.

ngram_tool Displays information about a given n-gram model file. Also allows checking of the parameter values and the setting of smoothing parameters.

pcs_tool Displays information about a given principal components scaling matrix (`.pcs`) file.

train_models A Sapphire Tcl script to train up a new set of acoustic models based on forced or hand generated/verified transcriptions. It uses the information in the domain .rec file (e.g., `pegasus.rec`) as well as parameter specifications in the `trans.espec` file. This program defaults to running in distributed mode and will generate intermediate files in the temporary directory.

train_pcs A Sapphire Tcl script to train up a new set of principal components based on forced or hand generated transcriptions. It uses the information in the domain rec file (e.g., `pegasus.rec`) as well as parameter specifications in the `trans.espec` file. This program defaults to running in distributed mode and will generate intermediate files in the temporary directory.

tv A Sapphire Tcl script for graphically looking at utterances and aligned transcriptions (the `.wrd` and `.phn` files). This program makes use of the parameter specifications in the given espec file. For example, `tv.espec` will randomly select 50 sentences from the `p_3` set to be examined. The tool displays the spectrogram, the waveform, the time aligned phonetic and word level transcription of an utterance. You can play individual segments and words by pointing the mouse over the desired segment and hit Ctrl-v.

waveform_play Plays a NIST (`.wav`) format waveform.

wlex_create Creates a word lexicon (`.wlex`) file from a basic vocabulary (`.vocab`) file. The word lexicon file is used in building n-gram language models.

Appendix: SUMMIT Files

This section contains a descriptive index of the files used in the lab. These files contain data, models, and parameter specifications needed to build a speech recognizer.

Temporary Directory and Files

The `fst_trans_compute`, `train_pcs`, and `train_models` programs create temporary subdirectories to hold intermediate data files created during the training process. These subdirectories are located in `temp/t<n>` where `<n>` is a unique integer determined by the system to not conflict with existing temporary directories. These directories tend to get large so please remove them after you have successfully run your experiments.

ESPEC Files

The ESPEC type, which stands for experiment specification, is used to specify any parameters and files relevant to the usage of a SUMMIT tool. An espec file is a simple text file, with a `.espec` extension, that records information used to run a given experiment. Detailed information about espec files can be found online at the following URLS:

```
file:/usr/sls/sls/v3.1/html/espec_control_usage.html
file:/usr/sls/sls/v3.1/html/espec.html
```

paths.espec Experimental parameter specifications for `fst_trans_compute`. Specifies the data set to run forced alignment on, the allowed hosts, the name of the utterance, the location of the waveform, the reference word sequence, and the name of the output time aligned word and phonetic transcriptions files.

pegasus.espec Experimental parameter specifications for `fst_test_rec_accuracy`. Specifies the data set to perform recognition on, the allowed host machines on which to run the distributed jobs, the name of the utterance, the location of the waveform, and the reference word sequence (for measuring performance).

trans.espec Experimental parameter specifications for `train_pcs` and `train_models`. Specifies the data set to train on, the allowed hosts, the name of the utterance, the location of the waveform, the time aligned phonetic transcription, and the reference word sequence.

tv.espec Experimental parameter specifications for `tv`. Specifies the set of data to examine and the location of the needed files: waveform, and time aligned word and phonetic transcriptions.

File Index and Description

domain.info Definition and specification of variable and parameter values that will be used for this domain. This file is used by most of the programs to get default parameter settings.

galaxy.corpus The corpus file for several experimental domains, including **Mercury**, **Jupiter** and **Pegasus**, etc. The sets `p_*` are for Pegasus. Contains properties and set definitions for the data set. Use the program `corpus_tool` to examine the contents.

pegasus.baseforms Basic phonetic pronunciation dictionary for all vocabulary words. Each line has the format:
word : *phone1 phone2 ...*

Alternative pronunciations are specified by (*phone1, phone2*) and & is used to indicate a word boundary.

pegasus.blabels List of all clustered diphone (boundary) labels. All the labels in the same line share the same acoustic model. Acoustic models will be built for all the diphone clusters.

pegasus.class.rules Set of word to class mappings or rules. This is used to create a CFG file which is then used to build class n-gram language models.

pegasus.pron.rules Set of phonological rules used to expand (to multiple pronunciations) the basic phonetic transcriptions in the baseforms file. These rules allow for context-dependent variation in the phonetic realization of words. They usually have the largest impact at word boundaries, as word internal variation can often be represented in the baseform pronunciations.

pegasus.rec This is the file that "drives" the training and recognition process. It lists the processing modules and their parameter specifications for each step in the training and testing process. It particular, it specifies the signal processing this is to be performed on the waveform, the type and size of the feature vectors to be computed, the principal components analysis to be done, the type of acoustic models to be trained, and the type of search to use during recognition.

pegasus.vocab List of all the vocabulary words in the domain.

train_sents.txt List of utterances in the **Pegasus** training set and some extra sentences. They are used to train the language models.

Appendix: Diphone Clustering

To perform diphone clustering, you need to extract the diphone labels from the forward search FST file `pegasus.ffst`:

```
% get_blabels_from_fst.pl -fst pegasus.ffst
```

It will write the diphone labels to the file `out.blabels`. Then the diphone clusters are obtained using the tool `create_blabels_clusters.pl`:

```
% create_blabels_clusters.pl
   -blabels out.blabels \
   -slabels decision_tree/pegasus.slabels \
   -initial_questions decision_tree/initial_diphone_clusters.txt \
   -cluster_questions decision_tree/diphone_questions.txt \
   -traindir temp \
   -bpcs galaxy.bpcs \
   -out new.blabels
```

Note that you should remove all the files in the `temp` directory first. It will take approximately 50 minutes to perform the clustering. The output file is `new.blabels`, where the diphone labels of the same cluster are put in the same line. This file is the same as the `pegasus.blabels` file provided to you.

Appendix: Phonemic Unit Set

VOWELS

`aa` ⟶ t<u>o</u>t	`er` ⟶ b<u>er</u>t
`ae` ⟶ b<u>a</u>t	`ey` ⟶ b<u>ai</u>t
`ah` ⟶ b<u>u</u>t	`ih` ⟶ b<u>i</u>t
`ao` ⟶ b<u>ough</u>t	`iy` ⟶ b<u>ee</u>t
`ax` ⟶ <u>a</u>bout	`ow` ⟶ b<u>oa</u>t
`aw` ⟶ b<u>ou</u>t	`oy` ⟶ b<u>oy</u>
`ay` ⟶ b<u>uy</u>	`uh` ⟶ b<u>oo</u>k
`eh` ⟶ b<u>e</u>t	`uw` ⟶ b<u>oo</u>t

SEMIVOWELS

`l` ⟶ <u>l</u>et	`y` ⟶ <u>y</u>es
`r` ⟶ <u>r</u>at	`w` ⟶ <u>w</u>et

NASALS

`m` ⟶ <u>m</u>et	`ng` ⟶ son<u>g</u>
`n` ⟶ <u>n</u>et	

STOPS

`p` ⟶ p̲et	`pd` ⟶ ta̲p̲
`t` ⟶ t̲ip	`td` ⟶ ta̲t̲
`k` ⟶ k̲it	`kd` ⟶ ta̲ck̲
`b` ⟶ b̲et	`bd` ⟶ ta̲b̲
`d` ⟶ d̲ebt	`dd` ⟶ ta̲d̲
`g` ⟶ g̲et	`gd` ⟶ ta̲g̲
`p-` ⟶ s̲peak	`tf` ⟶ bu̲t̲ter
`t-` ⟶ s̲teak	`df` ⟶ shu̲dd̲er
`k-` ⟶ s̲keet	

FRICATIVES

`f` ⟶ f̲at	`v` ⟶ v̲at
`th` ⟶ t̲hird	`dh` ⟶ t̲hat
`s` ⟶ s̲at	`z` ⟶ z̲oo
`sh` ⟶ s̲hoot	`zh` ⟶ as̲ia
`hh` ⟶ h̲at	

AFFRICATES

`ch` ⟶ c̲hew	`jh` ⟶ j̲udge

SPECIAL SEQUENCES

`ao r` ⟶ bo̲ar̲
`eh r` ⟶ be̲ar̲
`ih r` ⟶ be̲er̲
`ix ng` ⟶ writi̲ng̲
`tq en` ⟶ writ̲ten̲
`nt` ⟶ i̲nterstate