



MODULE COURSEWORK FEEDBACK

Student Name: Junjie Pan

Module Title: Major Practical

CRSiD: jp697

Module Code: MLSALT11

College: Hughes Hall

Coursework Number: Speech Recognition

I confirm that this piece of work is my own unaided effort and adheres to the Department of Engineering's guidelines on plagiarism

Junjie Pan

Date Marked:

Marker's Name(s):

Marker's Comments:

This piece of work has been completed to the following standard *(Please circle as appropriate)*:

	Distinction			Pass			Fail (C+ - marginal fail)		
Overall assessment (circle grade)	Outstandi <input type="checkbox"/>	A+ <input type="checkbox"/>	A <input type="checkbox"/>	A- <input type="checkbox"/>	B+ <input type="checkbox"/>	B <input type="checkbox"/>	C+ <input type="checkbox"/>	C <input type="checkbox"/>	Unsatisfactor <input type="checkbox"/>
Guideline mark (%)	90-100	80-89	75-79	70-74	65-69	60-64	55-59	50-54	0-49
Penalties	10% of mark for each day late (Sunday excluded)								

The assignment grades are given **for information only**; results are provisional and are subject to confirmation at the Final Examiners Meeting and by the Department of Engineering Degree Committee.

1 Introduction

In the practical, language modelling, acoustic modelling(speaker-adaptation) and system combination methods were implemented to build evaluation systems for dev03/eval03 datasets and Challenge datasets. This report consisted of 5 sections. **Section 2** included the experiments and results in language modelling on dev03/eval03 datasets. **Section 3** contained the scoring results obtained by cross-adaptation on dev03/eval03 datasets. **Section 4** described the ROVER combination and Confusion Network Combination and the implementation of them to dev03/eval03 datasets. **Section 5** concluded the whole process to construct the evaluation system for dev03/eval03 datasets and Challenge datasets. Discussions about the final scores got from the Challenge evaluation dataset were also included. The scripts used in this practical were attached in **Appendix** and their locations on the MLSALT computer were specified as well.

2 Language Model Improvements

2.1 1-best Hypothesis Generation

The command:

```
1 while read line
2 do
3   ./scripts/1bestlats.sh dev03_$line lattices decode plp-bg
4 done < ${store}/dev03
```

was used to generate the 1-best hypotheses from the dev03 development dataset.

The error rate of it is shown in Table 1:

Table 1: Scoring Result for 1-best Hypothesis

SPKR	# Snt	# Wrđ	Corr	Sub	Del	Ins	Err	S.Err	NCE
Sum/Avg	572	25569	82.6	12.8	4.6	2.4	19.8	83.0	-0.596

2.2 Evaluate performance in Word Error Rate (WER)

The five LMs given were applied to the **dev03** development dataset to examine its WER. The results are shown in table 2:

Table 2: Scoring Result for dev03 in LM1-5

LM	SPKR	# Snt	# Wrđ	Corr	Sub	Del	Ins	Err	S.Err	NCE
1	Sum/Avg	572	25569	85	11.5	3.5	2.7	17.7	81.7	-0.729
2	Sum/Avg	572	25569	83.8	12.6	3.6	2.8	19.1	83.0	-0.598
3	Sum/Avg	572	25569	80.8	14.4	4.8	2.4	21.6	83.7	-0.502
4	Sum/Avg	572	25569	80.1	14.9	4.9	2.7	22.5	84.8	-0.460
5	Sum/Avg	572	25569	83.7	12.2	4.1	2.3	18.6	81.1	-0.647

As given in the handout, the information for each LM is shown in table 3.

Table 3: Language Model Training texts and sizes

LM	Source	Type	Size(MW)
lm1	PSMs BN transcripts 92-99 TDT2&TDT3 captions	newswire	275
lm2	Transcripts from CNNs website 99-00	newswire	66
lm3	TDT4 captions	acoustic	2
lm4	NISTs BN training data from 97/98 Marketplace show transcripts	acoustic	2
lm5	Newswire LAT and WP 95-98, NYT97-00 Associated Press 97-00	newswire	674

From table 3 it can be seen that **LM1**, **LM2** and **LM5** were trained from newspaper text, while **LM3** and **LM4** were trained from spoken data. In common sense, spoken language is much more causal and flexible in grammar compared to newspaper language, so that **LM3** and **LM4** are expected to have higher WER on **dev03** dataset. Meanwhile, as the **LM1** and **LM5** have larger training sets than others, they should have the best performance.

The results shown in table 2 match the expectation above. It can be seen that the WER is 17.7% from **LM1** and 18.6% from **LM2**, which is the top 2 results. The WER from **LM2** is 19.1%, which is better than **LM3** and **LM4** (with WER=21.6% and 22.5% respectively).

2.3 Perplexity of each language models

```

1 for ((j=1;j<=5;j++))
2 do
3   base/bin/LPlex -C lib/cfgs/hlm.cfg -u -t lms/lm${j} \
4   lib/texts/dev03.dat
5 done

```

By using the *LPlex* command, the perplexity of each language model to **dev03** development set can be obtained.

Table 4: Perplexity for LM1-5

LM models	LM1	LM2	LM3	LM4	LM5
WER	17.7	19.1	21.6	22.5	18.6
Perplexity	198.6167	242.7938	283.7183	337.4166	220.4432

From table 4 it can be noticed that the perplexities of LMs correlate well with the WER - LM with lower WER has lower perplexity, confirming our expectation in **Section 1.2**.

2.4 LM interpolation

In order to interpolate the language model, the following steps are taken to interpolate the LM from **dev03** dataset:

1. Obtain streams of probabilities for each of the LMs to be interpolated
2. Estimate interpolation weights
3. Merge LMs using the interpolation weights

In step 2, *Interpolation.py* python script is used. There are two functions used in this script, *read_file* and *inter*. The *read_file* function is to read the stream files. The *inter* function is to estimate the optimal weights. There is no closed form solution of weights λ_n , so EM algorithm is adopted to find the local optimal values. The related formulae are listed below

$$P(a|\mathbf{w}, i, \tau) = \frac{\lambda_a^{(\tau)} P_a(\omega_{i-2}, \omega_{i-1})}{\lambda_a^{(\tau)} P_a(\omega_i|\omega_{i-2}, \omega_{i-1}) + \lambda_n^{(\tau)} P_n(\omega_i|\omega_{i-2}, \omega_{i-1})} \quad (1)$$

and

$$\lambda_a^{(\tau+1)} = \frac{1}{K+1} \sum_{i=1}^{K+1} P(a|\mathbf{w}, i, \tau) \quad (2)$$

As EM is guaranteed not to decrease the log-likelihood (means not to increase the perplexity in this case), it can finally converge. However, EM can only find the local optimum, which means that the estimation of the interpolation weights is sensitive to the initialisation of weights. In my codes, the initial weights are set equally as 0.2.

The interpolated LM is then applied to the **dev03** and **eval03** datasets. The WER and perplexity for each dataset is shwon in table 5 and Table 6.

Table 5: WER and Perplexity for dev03 and eval03 by interpolated LM

Dataset	WER	Perplexity
dev03	16.8	151.5168
eval03	15.0	154.8256

Table 6: dev03 and eval03 scoring details by interpolated LM

Data Sets		#Snt	#Wrd	Corr	Sub	Del	Ins	Err	S.Err	NCE
dev03	Sum/Avg	572	25571	85.6	10.6	3.8	2.3	16.8	79.7	-0.769
eval03	Sum/Avg	508	24938	87.0	9.7	3.3	2.0	15.0	84.3	-0.887

Comparing Table 4 with Table 5 and Table 6 it is clear that the WER and perplexity of dev03 has been significantly improved by using the interpolated LM. For the **eval03** dataset, the WER is 15.0% and the perplexity is 154.8256, which are much better than the performance from the single LM 1-5 in table 7.

Location of Scripts:

1. ConvertData.py

- Appendix/Python Scripts/A.1.1 ConvertData.py
- /home/jp697/Major/exp/shell/ConvertData.py

2. Interpolation.py

- Appendix/Python Script/A.1.2 Interpolation.py
- /home/jp697/Major/exp/shell/Interpolation.py

3. interpolation_challenge.sh

Note: here is the one used in challenge development dataset.

- Appendix/Bash Scripts/A.2.1 Interpolation_challenge.sh
- /home/jp697/Major/exp/shell/interpolation_challenge.sh

2.5 show-specific LM interpolation

Rather than fixing the language model interpolation weights, performing unsupervised language model adaptation and using show-specific language model to deal with the evaluation dataset may improve the recognition performance.

The show-specific LM interpolation is implemented as follows:

When the interpolated LM **lm_int** is obtained, it can be applied to the evaluation dataset to generate lattices for each show in **eval03**, and 1best hypothesis is generated from the lattices by using *1bestlats.sh*. The *ConvertData.py* is used to convert the 1best hypothesis into suitable date format, and step 1 to step 3 in section 1.4 can be repeated to generate the show-specific interpolation language model named **lm_int_specific** in my case.

The WER and perplexity of eval03 from LM1-5, interpolation LM and show-specific interpolation LM are shown in table 7.

Table 7: WER and perplexity of eval03

Eval03	LM1	LM2	LM3	LM4	LM5	lm_int	lm_int_specific
WER	15.9	17.4	19.9	20.6	17.1	15.0	15.0
Perplexity	189.0499	235.4384	289.2490	323.5735	214.1698	154.8256	155.1063

From the table above it can be seen that the interpolation can greatly improve the performance on the eval03 dataset, as the WER and perplexity has been reduced to 15.0 and approximately 155 respectively. However, the difference between the interpolation LM with and without show-specific is not obvious. The reason should be that the evaluation dataset *eval03* has the similar nature as the development dataset dev03, so the show-specific method does not make much sense.

If the evaluation dataset are not in the same topic as the development dataset, the show-specific LM should be able to improve the performance of the results significantly.

Location of show-specific.sh

- Appendix/Bash Scripts/A.2.2 *show-specific.sh*
- /home/jp697/Major/exp/shell/5.5.6.sh

3 Acoustic Model Adaptation

3.1 Minimise Lattices

```

1 while read line
2 do
3   ./scripts/mergelats.sh $line plp-bg rescore plp-bg
4 done < $store/dev03

```

In this part, *mergelats.sh* command was applied to the lattices to perform aggressive beam pruning and arcs-per-second pruning. The scoring results of the lattices after minimising is shown in table 8.

Table 8: Scoring results of dev03 from original and pruned lattices

dev03		#Snt	#Wrd	Corr	Sub	Del	Ins	Err	S.Err	NCE
Original Lattices	Sum/Avg	527	25454	7.2	6.9	85.8	1.7	94.4	87.4	0.046
Pruned Lattices	Sum/Avg	527	25454	82.4	13.0	4.6	2.4	19.9	83.4	-0.586

As can be seen that the WER has been greatly reduced from 94.4 (original lattices) to 19.9 (minimised lattices).

3.2 Cross-Adaptation

```

1 # plp-adapt-by-grph-plp as an example
2 # rescore the merged lattice with grph-plp system
3 while read line
4 do

```

```

5  ./scripts/hmmrescore.sh $line plp merge grph-plp grph-plp
6  done < ${testlist}/dev03.lst
7  # perform cross-adaptation
8  while read line
9  do
10 ./scripts/hmmadapt.sh $line grph-plp decode plp-adapt-grph-plp plp
11 done < ${testlist}/dev03.lst
12 # rescore the adapted lattice
13 while read line
14 do
15 ./scripts/hmmrescore.sh -ADAPT plp-adapt-grph-plp adapt ${line} plp \
    merge plp-adapt-grph-plp plp
16 done < ${testlist}/dev03.lst

```

In this part, the experiment 5.5.2 and 5.5.3 in handout were performed. The process is:

1. Choose a system from `plp`, `grph-plp`, `tandem`, `grph-tandem`, `hybrid`, and rescore the lattice with the chosen system using *hmmrescore.sh* command
2. Perform different adaptation on the 1-best hypothesis from step 1 using *hmmadapt.sh* command
3. Rescore the transform from step 2 using *hmmrescore.sh*
4. Scoring the output from step 2 and step 3 using *score.sh* command

The cross-adaptation was applied to each system, and the scoring results are shown in table 9¹.

Table 9: scoring results of dev03 from each system

Supervisor System	Adaptation					Original	CN
	plp	grph-plp	tandem	grph-tandem	hybrid		
plp	14.9	14.5	15.1	14.9	13.8	16.8	16.3
grph-plp	-	-	-	-	-	17.4	17.3
tandem	15.2	14.9	16.6	15.5	14.4	17.3	17.4
grph-tandem	14.9	15.3	15.7	16.1	14.3	16.8	16.8
hybrid	-	-	-	-	-	12.9	12.8

¹The adaptations unrelated to language modelling were deleted by mistake, so the scoring results here came from **Section 5.1**, the construction of evaluation systems on dev03/eval03, where all acoustic models were related to language modelling.

When I attempted to apply cross-adaptation to grph-plp system, the results showed WER=99.1%. There might be some errors in the *hmmadapt.sh* codes, so the adaptation to grph-plp was not achieved. In that case, cross-adaptation to grph-plp will be ignored in the following experiments.

It can be found that except for hybrid and grph-plp system, the WER of the other 3 systems all have been reduced by implementing adaptation. Among these adaptation results, the `plp-adapt-by-hybrid`² system showed the best performance with WER=13.8%.

However, among all systems the HMM-DNN system (`hybrid`) had the best performance with WER=12.8% .

4 System Combination

ROVER combination and Confusion Network Combination were investigated in this part.

4.1 ROVER Combination

There are three stages in ROVER combination.

- The first stage is to use Viterbi-based algorithm to make alignment between two outputs. In my script, dynamic programming alignment with a cost function was used to achieve it.
- The second stage is to generate a word transition network based on the alignment. !NULL symbol is inserted to deal with the deletion and insertion cases. The score for the !NULL is set as 0.2 in my codes (as suggested in the hand-out). If the matching case happens, the score is simply the average of the two outputs.
- Then the best path can be selected from the transition network by choosing the word with higher score, and generate combination results.

²The actual name for each adaptation is **Supervisor-adapt-Adaptation**, which is written as **Supervisor-adapt-by-Adaptation** in this report to avoid miss understanding.

4.1.1 Description of ROVER combination script

Location

- Appendix/Python Script/A.1.3 RoverCombi.py
- /home/jp697/Major/exp/shell/RoverCombi.py

load_dict_from_file function:

Input: MLF file directory

Output: Suitable form of MLF file that can be used in ROVER combination

Description: This function construct a data structure as

```
{
  {entry1 : {word : [w1, w2, ..., wn], start : [t1, ..., t2], end : [e1, ..., en], score : [s1, ..., sn]},
  {entry2 : {word : [w1, w2, ..., wn], start : [t1, ..., t2], end : [e1, ..., en], score : [s1, ..., sn]},
  ...
  {entryn : {word : [w1, w2, ..., wn], start : [t1, ..., t2], end : [e1, ..., en], score : [s1, ..., sn]}}
}
```

costfunction:

Input: entries of the two mlf files, with their index.

Output: substitution penalties

Description:

This function is used to calculate the distance between two words. The cost function is:

$$penalty = pt + pw$$

where

$$pt = |(startTime_1 - startTime_2) + (endTime_1 - endTime_2)|$$

and pw is obtained by another DP alignment between the two target words with deletion, insertion and substitution penalty equalling to 1.

merge_mlf function:**Input:** two mlf files in the data structure described above.**Output:** record matrix, which contains the moving steps from DP alignment**Description:**

This function implement the DP process. There are three steps in DP:

1. Initialization Two matrices *penalty* and *record* are used in DP. They are initialised as

$$record = \begin{bmatrix} Begin & (f_2, 0) & (f_2, 0) & \dots & \dots & (f_2, 0) \\ (f_1, 0) & & & & & \\ (f_1, 0) & & & & & \\ \dots & & & & & \\ (f_1, 0) & & & & & \end{bmatrix}$$

$$penalty = \begin{bmatrix} 0 & 5 & 10 & \dots & \dots & 5N \\ (5) & & & \dots & & \\ (10) & & & \dots & & \\ \dots & & & \dots & & \\ (5N) & & & \dots & & \end{bmatrix}$$

The penalty setting is shown in figure 1

2. Matrix Fill (scoring)

After the initialisation of *penalty* and *record*, scoring is performed between each word in a particular entry of the two input mlf files.

The way used in my code is to find the minimum alignment penalty by starting in the upper left hand corner in the matrix and finding the minimum penalty $M_{i,j}$ for each position in the penalty matrix. In order to find $M_{i,j}$ for any i,j it is minimal to know the penalty for the matrix positions to the left, above and diagonal to i, j. In terms of matrix positions, it is necessary to know $M_{i-1,j}$, $M_{i,j-1}$ and $M_{i-1,j-1}$. The corresponding moving step is recorded in the record matrix.

For each position, $M_{i,j}$ is defined to be the maximum score at position i, j.

$$M_{i,j} = \text{MINIMUM} = \begin{cases} M_{i-1,j-1} + PENALTY_{sub} & (\text{match/mismatch in the diagonal}) \\ M_{i,j-1} + PENALTY_{del} & (\text{gap in } mlf_1) \\ M_{i-1,j} + PENALTY_{ins} & (\text{gap in } mlf_2) \end{cases}$$

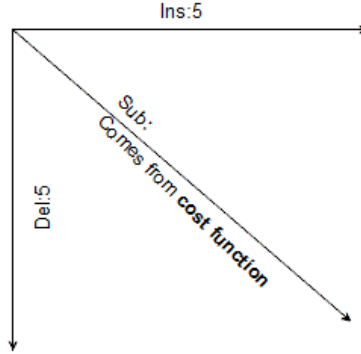


Figure 1: Penalty for deletion, insertion and substitution

where $PENALTY_{del} = PENALTY_{ins} = 5$, and $PENALTY_{sub}$ is determined by *costfunction*.

3. Traceback (alignment)

After the scoring step, the minimum alignment penalty for the two mlf file can be obtained. The traceback step determines the actual alignment that results in the minimum penalty. It is possible to get different paths through this method. In order to solve the ambiguous situation, mlf_1 is manually set as the reference, and is preferred when the same penalties occur in traceback process. It should be aware that the reference setting may cause the MLF combination sensitive to the order of the two input MLF files.

The traceback step begins in the M,J position in the matrix, that is, in other words, the position that leads to the minimal penalty. In this case, there is always the end of the word sequence.

The optimal path is exactly the combined MLF file from the two input MLF files, and can be written into MLF format using *recovermlf* function.

***recovermlf* function**

Input: MLF_1 , MLF_2 , Record Matrix, and !NULL score

Output: the combine mlf in python dictionary format

Description:

The Record Matrix is from *merge_mlf* function and the default value of !NULL score is 0.2.

This function recovers the combined mlf from the two original mlf according to the DP results, returns a dictionary format of the combined MLF.

***bestpath* function**

Input: MLF file in dictionary format

Output: the final combined MLF file

Description: This function chooses the entries with highest scores and generate the final MLF file by calling `save_dict_to_file` function.

***save_dict_to_file* function**

Input: The data structure obtained from the *bestpath* function, and the output file directory.

Output: The final combined MLF file that can be used for scoring.

Description: This function generates the final version of MLF file and saves it into the file.

***read_list* function**

Input: The dataset list

Output: The dictionary containing the list

***main* function**

Input: mlf 1, mlf 2, decodetype, dataset list, pass 1, pass 2

Output: Rover combination results.

Description:

This is the main function to perform ROVER combination. The general process is:

1. Initialisation: define the name of the result directory, and load the dataset list
2. Convert the MLF file into the data structure described in *load_dict_from_file* function.
3. Perform ROVER to the two input MLF files and write the combined result into MLF file (with alternative paths)

4. Selecting the best path from the generated MLF, and rewrite the result into it.

4.1.2 Confusion Network Generation and Confidence Score Mapping

When the MLF file is obtained, confusion networks can be generated using *cn-rescore.sh* command. The confusion network associates each word with a confidence score. However, the confidence scores obtained are typically too high to perform a reasonable comparison. In this case, *-CONFTREE* option can be activated in *score.sh* script to realise the score mapping in scoring process. As we want the MLF files for combination, *smoothtree-mlf.pl* script can be used to map confidence scores and piped the output into a file for combination.

By checking the *rescore.mlf* file it can be found that a large number of words were associated with the same score 1.0, and after mapping they were reduced to different values. The impact of score mapping was not obvious in single system, but it improved the performance of the combination system as shown in table 10. It reduced the WER of plp and plp-adapt-by-grph-plp ROVER combination from 15.3% to 14.6%.

Table 10: Comparison between WER before and after confidence scores mapping

Acoustic Model	WER(no mapping)	WER(mapping)
plp	16.4	16.3
grph-plp	17.3	17.3
ROVER(plp, plp-adapt-by-grph-plp)	15.3	14.6

4.2 Confusion Network Combination

4.2.1 Description of CNC

Location

- Appendix/Python Script/A.1.4 CNC.py
- /home/jp697/Major/exp/shell/CNC.py

CNC is similar to ROVER combination. The case now is to align and combine the confusion network lattices rather than the MLF files. In that case, the same DP scheme can be used in CNC to align confusion networks. In that case, similar distance measures are applied as the ROVER combination.

4.2.2 Description of CNC script

In CNC python script all the other functions are similar to those used in ROVER combination, except for the *read_lattice* and *main* function.

read_lattice function

Input: directory of the target show

Output: python dictionary containing the lattice information

Description:

The data structure used to store the lattice information is

```
{
    entry1 :
        {word : [!NULL, he_<ALTSTART>_his_<ALT>_!NULL_<ALTEND>, .....],
          start : 00000000, end : 0000100, score : [1, 0.23 - 0.62 - 0.18, ...]},
    entry2 : {.....}
    .....
    entryn : {.....}
}
```

main function

Input: two lattices paths, decodetype³, dataset list, pass_1, pass_2

Output: CNC results

Description:

The general process of CNC is:

1. Initialisation. Define the combination name and load the dataset list.
2. Load two input lattices, convert them into dictionary format and save them as mlf files⁴.
3. Load the mlf files from last step, select the best path from them and rewrite into the MLF files.

³'decodetype' parameter is not needed in CNC, but it is retained to keep the same input parameter format as ROVER. It makes further work easier.

⁴In the lattices, there are alternative paths existing with different scores, start and end time stamp. In this case, all the possible paths will be record in the data structure with labels <ALTSTART><ALT> and <ALTEND> in the interval.

4. Reload the new MLF file and perform the CNC process.
5. Recover the MLF from the CNC result and save it as MLF file.
6. Select the best path from the MLF file from last step and rewrite it as the final CNC MLF file.

4.3 Implementation of Combination to dev03 dataset

In order to find the best version of system combination, different systems were chosen to be combined together. The scoring results is shown in Table 11⁵.

Table 11: scoring results of dev03 from combination systems

<i>System₁</i>	<i>System₂</i>	ROVER	CNC
hybrid	grph-tandem-adapt-by-hybrid	13.5	14.2
hybrid	plp-adapt-by-grph-tandem	13.6	14.7
hybrid	plp-adapt-by-plp	13.6	14.6
hybrid	tandem-adapt-by-hybrid	13.7	14.1
hybrid	plp-adapt-by-grph-plp	13.8	14.4
hybrid	plp-adapt-by-hybrid	13.8	14.8
tandem-adapt-by-hybrid	plp-adapt-by-hybrid	13.9	15.5
hybrid	grph-tandem-adapt-by-plp	13.9	14.5
hybrid	grph-tandem-adapt-by-grph-plp	13.9	14.8
hybrid	plp-adapt-by-tandem	14.0	14.6

From Table 11 it can be found that the best combination scheme for dev03 dataset is the ROVER combination with hybrid and grph-tandem-adapt-by-hybrid systems, which has the WER of 13.5%.

In this case, the same combination scheme was applied to the eval03 dataset to check its performance. The final WER is 14.9%.

⁵All the systems here were not related to language modelling, so the WER should be higher than that in Section 5.1

5 Evaluation System Development

Two evaluation systems were constructed for dev03/eval03 and Challenge dataset respectively.

5.1 Evaluation System using dev03 dataset

Scripts Location

/home/jp697/Major/exp/shell/training_dev03.sh

Description

The development process of the evaluation system is:

1. Language Modelling

In this part, 1-best hypothesis was generated from the **dev03** dataset. The WER of the 1-best hypothesis was 19.8%. This hypothesis was then converted to the data file with suitable format using *ConvertData.py* script. After that, streams can be generated using *LPlex* command, and used to estimate the interpolation weights by *Interpolation.py* script. The description of this script has been shown in **Section 1.4** and **Appendix/Python Script/A.1.2 Interpolation.py**. Interpolation can then be implemented using *LMerge* based on the weights obtained.

This interpolation LM was evaluated using the **dev03** dataset by *lmrescore.sh* script. MLF files and lattices were obtained from this process.

2. Acoustic Modelling

The lattices obtained from the interpolation LM were firstly minimised using *mergelats.sh*. The minimised lattices were then rescored under the supervision from different systems. Here all the five systems were taken into account. The rescored results located in *./plp/rescore*, *./grph-plp/rescore*, *./tandem/rescore*, *./grph-tandem/rescore* and *./hybrid/rescore*.

3. Speaker-adaptation Implementation

The adaptation process used the 1-best hypothesis from the bigram lattices to produce cascaded **CMLLR** and **MLLR** transforms. There were two stages for this transform: force align and transform estimation. In the first stage, the hypothesis using the acoustic model and the hypothesis were aligned and

a phone-sequence related to the hypothesis was generated using *HVite* HTK tool. In the second stage, *HERest* command was used to produce a global **CMLLR** and two **MLLR** transforms. The *hmmadapt.sh* command was applied to perform this adaptation. The scoring results of the systems with and without adaptation are shown in Table 9. It is clear that the performance of each system (except for hybrid, which does not have adaptation) has been improved after the adaptation implementation.

4. Confusion Networks and Confidence Scores Mapping

The lattices generated before can also be used to produce confusion networks with confidence scores using *cnrescore.sh*. The results located in `./{systemName}/decode_cn` directory. The confidence scores here were typically too high, so that mapping tree was applied to reduce them to obtain better performance. The scoring results were shown in Table 12. As can be seen that the WER of most systems have been slightly improved after generating the confusion networks.

Table 12: scoring results of confusion networks from dev03

Supervisor System	Adaptation					Original	CN
	plp	grph-plp	tandem	grph-tandem	hybrid		
plp	14.9	14.5	15.1	14.9	13.8	16.8	16.3
grph-plp	-	-	-	-	-	17.4	17.3
tandem	15.2	14.9	16.6	15.5	14.4	17.3	17.4
grph-tandem	14.9	15.3	15.7	16.1	14.3	16.8	16.8
hybrid	-	-	-	-	-	12.9	12.8

5. ROVER Combination Investigation

Till now, five systems (*plp*, *grph-plp*, *tandem*, *grph-tandem*, *hybrid*) with and without adaptation have been built on dev03, including the original MLF files as well as confusion networks. In order to investigate the performance of different MLF combinations, all the rescore.mlf files were combined iteratively. The scoring results are shown in Table 13. As the number of combinations are too large, only top 10 combinations are listed here.

From Table 13 it can be found that the combination of **hybrid** system and **grph-tandem-adapt-by-hybrid** system shows the best performance with WER=12.7%.

6. Confusion Network Combination

Table 13: scoring results of dev03 from ROVER combination

<i>System₁</i>	<i>System₂</i>	ROVER	CNC
hybrid	grph-tandem-adapt-by-hybrid	12.7	13.1
hybrid	plp-adapt-by-grph-tandem	12.9	14.2
hybrid	plp-adapt-by-plp	13.1	14.0
hybrid	tandem-adapt-by-hybrid	13.1	13.9
hybrid	plp-adapt-by-grph-plp	13.4	14.1
hybrid	plp-adapt-by-hybrid	13.6	14.2
tandem-adapt-by-hybrid	plp-adapt-by-hybrid	13.7	14.9
hybrid	grph-tandem-adapt-by-plp	13.8	13.9
hybrid	grph-tandem-adapt-by-grph-plp	13.9	14.2
hybrid	plp-adapt-by-tandem	13.9	14.1

The CNC methodology was also applied to different systems to check its performance, and the scoring results for each case are shown in Table 13.

From Table 13 it can be found that the CNC of **hybrid** system and **grph-tandem-adapt-by-hybrid** system performs the best with WER = 13.1%. It is worse than the performance of the best ROVER combination system, which does not match my expectation. As CNC tries to combined systems from the lattices while ROVER combines the mlf outputs directly, CNC can consider all the possible paths from both lattices and is expected to have better performance on the combination results.

7. Final Version of Evaluation System

From previous steps it can be concluded that the best evaluation system for dev03 should be:

- (a) Generate 1-best hypothesis from the lattices provided, and use it to interpolate a LM from the five LMs provided.
- (b) Use the interpolation LM to rescore the dataset, and generate LM-related lattices.
- (c) Generate hybrid and grph-tandem acoustic model from the LM-related lattices obtained from previous step, and applied hybrid adaptation to the grph-tandem model.
- (d) Generate confusion networks from the two acoustic models in last step, and apply mapping tree to the MLF file to reduce the confidence scores.

- (e) Implement ROVER combination to `./hybrid/decode_cn/rescore_mappingtrees.mlf` and `./grph-tandem-adapt-hybrid/decode_cn/rescore_mappingtrees.mlf`.

8. Evaluate the **eval03** dataset

The evaluation described in step 7 is chosen for evaluating eval03 dataset, and the final WER is 14.4%.

5.2 Evaluation System using YTBEddev dataset

Scripts Location

`/home/jp697/Major/exp/shell/training_challenge.sh`

Description

The development process of the evaluation system to the challenge dataset is:

1. Language Modelling

ConvertData.py was used to convert the 1best hypothesis into suitable format and then *Interpolation.py* was applied to the streams from LM1-5 to generate interpolation weights. A final interpolation LM was generated and named `lm_int_challenge`.

2. Acoustic Modelling

The language model related lattice was produced by the *lmrescore.sh* and minimised by *mergelats.sh*. The scoring result from the *lmrescore.sh* was WER=43.1%. Based on the merged lattices, `plp`, `grph-plp`, `tandem`, `grph-tandem`, and `hybrid` systems can be implemented. The outputs located in `/home/jp697/Major/challenge/`.

3. CN Generation and Cross-adaptation

In this step, *cnrescore.sh* command was firstly used to generate confusion networks for each systems. Then all the adaptations were applied to the `plp`, `tandem` and `grph-tandem` systems. The results are shown in table 14.

As can be found that the confusion network has slightly improved the performance of each system. Furthermore, **speaker-adaptation** significantly reduced the WER of them. Similar to adaptation systems based on dev03 dataset, the best adaptation system here was also the `plp-adapt-by-hybrid`

Table 14: scoring results of YTBEddev from adaptation systems

Supervisor System	Adaptation					Original	CN
	plp	grph-plp	tandem	grph-tandem	hybrid		
plp	39.1	38.4	37.5	37.5	37.3	43.1	41.9
grph-plp	-	-	-	-	-	43.0	42.3
tandem	39.6	39.4	41.0	39.7	38.8	41.9	41.8
grph-tandem	39.5	39.5	37.5	40.3	38.4	41.2	41.2
hybrid	-	-	-	-	-	38.4	37.8

system with WER=37.3%, which was even better than the HMM-DNN system hybrid.

4. Combination

Both ROVER and CNC combination were applied to each system to check its performance. The scoring results are shown in Table 15.

Comparing the combination results with the Table 14 and Table 15, the WER after combination has been greatly reduced. The best combination was the ROVER combination between hybrid system and plp-adapt-by-hybrid system with WER=36.9%.

Based on the combined systems, I planned to combine more systems together to check its performance. However, when I applied ROVER combination to hybrid, plp-adapt-by-hybrid, and plp-adapt-by-grph-plp, it showed an even worse result with WER=37.1%.

In this case, this configuration I chosen to evaluate the **Challenge Evaluation Dataset** is the ROVER combination between hybrid and plp-adapt-by-hybrid systems.

5. Results from the Challenge Evaluation Dataset

The final score I got from my two submission is 36.0% and 36.8%, which will be discussed in the next section.

Table 15: scoring results of YTBEddev from combination systems

<i>System₁</i>	<i>System₂</i>	WER(ROVER)	WER(CNC)
hybrid	plp-adapt-by-hybrid	36.9	37.2
hybrid	plp-adapt-by-grph-plp	37.0	37.1
hybrid	grph-tandem-adapt-by-hybrid	37.0	37.3
hybrid	plp-adapt-by-grph-tadem	37.1	37.5
hybrid	plp-adapt-by-plp	37.2	37.5
hybrid	tandem-adapt-by-hybrid	37.3	37.4
plp-adapt-by-hybrid	tandem-adapt-by-hybrid	37.3	37.8
hybrid	grph-tandem-adapt-by-plp	37.5	37.9
hybrid	grph-tandem-adapt-by-grph-plp	37.6	38.1
hybrid	tandem-adapt-by-tandem	37.8	37.9

5.3 Discussion

Results from the Challenge evaluation dataset and the limitation of this practical infrastructure were carefully discussed in this section.

5.3.1 Challenge Evaluation Results

The MLF files I submitted got the WER of 36.0% and 36.8% respectively. It is much worse than what I expected.

In my first submission, no combination was applied. The system I built was just constructed by applying hybrid adaptation to plp system, because the ROVER combination script at that time seemed to have some bugs and cause rubbish results. For the development dataset, the final WER was 37.3%, so the result from the evaluation dataset feedback seemed reasonable.

However, in my second submission, I constructed the evaluation system as described in **Section 4.2**. The evaluation system was a ROVER combination between **hybrid** and **plp-adapt-by-hybrid** systems. For the development dataset, the WER was 36.9%, which showed an obvious improvement compared to the developing results from my first evaluation system. However, the feedback from the evaluation dataset was even worse than the first one, though it was still better than the development result. I thought maybe I submitted a wrong MLF file by mistake, which may be generated from the evaluation system without language modelling.

5.3.2 The Limitation of the Practical Infrastructure

- **Language Modelling**

This practical provides five tri-gram word level LMs. Implementation with higher gram LM may cause better performance. Besides, among the 5 LMs provided, the sizes of the two trained by acoustic data are too small compared to the other three, which would cause insignificant influence to the final results.

- **Acoustic Modelling**

Only one HMM-DNN acoustic model (**hybrid**) built using PLP features and the phonetic lexicon is provided, and the others are all HMM-GMM acoustic models. From the construction process I found that the **hybrid** system always showed much better performance than the HMM-GMM acoustic models. In this case, if more HMM-DNN systems (e.g. with tandem features) are provided with adaptation available, the final evaluation system is likely to perform better. In addition, there is an error occurring when I tried to implement speaker-adaptation to **grph-plp** system.

- **System Combination**

Log-likelihood combination is not investigated in this practical, which may cause different results.

- **Resource**

The challenge datasets all come from the Youtube video about election. If resources with different topics are provided, the construction of the evaluation system would be more flexible and difficult.

References

- [1] Prof. Mark Gales, *MLSALT11 Handout: Speech Practical - Large Covabulary Speech Recognition*, Department of Engineering, University of Cambridge, 2015/2016.
- [2] *Dynamic Programming Tutorial*.
Link: <http://www.avatar.se/molbioinfo2001/dynprog/dynamic.html>

A Appendix

A.1 Python Scripts

A.1.1 ConvertData.py

ConvertData.py

```
1  # -*- coding: UTF-8 -*-
2  '''
3  ConvertData.py file
4  Function: Convert 1-best hypothesis into suitable data file , so that it
5           can be used
6           to generate the interpolation weights
7
8  Description:
9  Usage: ConvertData.py dataset 1best_path savepath
10 -dataset: the data list. eg: challenge_dev , challenge_eval , dev03 ,
11          eval03
12 -1best_path: the directory of the 1best hypothesis (after ./Major)
13 -savepath: the directory to store the converted data file .
14
15 Author: Junjie Pan
16 Latest Modified: 2016/04/25
17 '''
18 #read the MLF file and save it into dictionary
19 def load_dict_from_file(filepath):
20     try:
21         _dict = {}
22         with open(filepath , 'r') as dict_file:
23             sent=0
24             for line in dict_file:
25                 wordline = line.strip().split(' ')
26                 if wordline[0] == '.':
27                     sent+=1
28                 elif len(wordline)==1:
29                     continue
30                 else:
31                     if sent not in _dict.keys():
32                         _dict[sent]=[]
33                     _dict[sent].append(wordline[2])
34
35     except IOError as ioerr:
36         print "File %s does not exist" % (filepath)
```



```

36     return _dict
37
38 # save the dictionary into reuquire data format
39 def save_dict_to_file(_dict, filepath):
40     try:
41         with open(filepath, 'a') as dict_file:
42             for key in _dict.keys():
43                 dict_file.write(' '<s> ')
44                 for word in _dict[key]:
45                     dict_file.write(word+' ')
46                 dict_file.write(' '</s>\n')
47     except IOError as ioerr:
48         print "File %s unable to create" % (filepath)
49
50 #read dataset list
51 def readlist(filepath):
52     try:
53         with open(filepath, 'r') as fs:
54             store=[]
55             for line in fs:
56                 filename=line.strip().split()
57                 if len(filename) == 1:
58                     store.append(filename[0][2:])
59     return store
60 except IOError as ioerr:
61     print "error happens when writing %s" %filepath
62
63 def main(lists, path, output):
64     files=readlist(r'/home/jp697/Major/exp/temp_file/%s'%lists)
65     for f in files:
66         _dict = load_dict_from_file(r'/home/jp697/Major/%s/%s/1best/LM12.0
        _IN-10.0/rescore.mlf'%(path, f))
67         save_dict_to_file(_dict, output)
68
69 if __name__ == '__main__':
70     import argparse
71     parser=argparse.ArgumentParser(decription='Convert 1best Hyp into
        suitable format')
72     parser.add_argument('dataset_list', type=str)
73     parser.add_argument('1best_path', type=str)
74     parser.add_argument('output', type=str)
75     args=parser.parse_args()
76     main(args.dataset_list, args.1best_path, args.output)

```

A.1.2 Interpolation.py

Interpolation.py

```
1  #!/usr/env/bin python
2  # -*- coding:UTF-8 -*-
3  import numpy as np
4  '''
5  Interpolation.py
6
7  This script is to interpolate LM weights, with uniform initial weights
   setting
8
9  Usage: Interpolation.py stream_path weight_path
10 -stream_path: the directory of stream files
11 -weight_path: the directory to save the generated weights into file
12
13 Author: Junjie Pan
14 Latest Modified: 2016/04/26
15 '''
16
17 # read stream files into dictiory
18 def read_file(filepath):
19     try:
20         ls=[]
21         with open(filepath, 'r') as ls_file:
22             for line in ls_file:
23                 p=float(line)
24                 ls.append(p)
25         return ls
26     except IOError as ioerr:
27         print "File %s cannot be open"%filepath
28
29 #Interpolation LM weights
30 def inter(ls):
31     #set the initial weights uniform
32     w=np.array([0.2,0.2,0.2,0.2,0.2])
33     PP=w[0]*ls[0]+w[1]*ls[1]+w[2]*ls[2]+w[3]*ls[3]+w[4]*ls[4]
34     temp=0 #store the current perplexity
35     while(abs(sum(PP)-temp)>1e-12):
36         temp=sum(PP)
37         # update rules
38         for i in range(5):
39             p=(w[i]*ls[i])/PP
40             w[i]=np.mean(p)
41         PP=w[0]*ls[0]+w[1]*ls[1]+w[2]*ls[2]+w[3]*ls[3]+w[4]*ls[4]
```

```

42     return w
43
44 #main function to perform interpolation
45 def main(streamdir , weightsdir):
46     ls=[]
47     for j in range(5):
48         ls.append([])
49         ls[j]=read_file ("%s/stream%d" % (streamdir ,j+1))
50     ls=np.array(ls)
51     weight=inter(ls)
52
53     weightfile=open(weightsdir , 'w')
54     for w in weight:
55         weightfile.write(str(w)+'\n')
56     weightfile.close()
57
58 if __name__=='__main__':
59     import argparse
60     parser=arg.ArgumentParser(description='Interpolation LM Weights')
61     parser.add_argument('StreamPath',type=str)
62     parser.add_argument('WeightPath',type=str)
63     args=parser.parse_args()
64     main(args.StreamPath, args.WeightPath)

```

A.1.3 RoverCombi.py

RoverCombi.py

```

1  #!/usr/env/bin python
2  # -*- coding: UTF-8 -*-
3  '''
4  ROVER combination script
5
6  This script uses DP to perform alignment to the two input MLF files .
7  The final version of MLF will be stored in ./Major/challenge/
   mlf_combination
8
9  Usage: RoverCombi.py mlf_A mlf_B decodetype testlist --pass1 --pass2
10 -mlf_A: reference mlf file
11 -mlf_B: the other mlf file
12 -decodetype: decode or decode_cn
13 -testlist: the list of dataset
14 -pass1 2: the pass directory of the mlf1,2 files
15
16 Author: Junjie Pan

```

```

17 Latest Modified: 2016/04/26
18 ',,'
19 import re
20 from numpy import *
21 #global setting
22 delpenalty=5 #deletion and insertion penalty
23
24 #load the MLF file and convert it into python dictionary
25 def load_dict_from_file(filepath):
26     try:
27         _dict = {}
28         with open(filepath, 'r') as dict_file:
29             for line in dict_file:
30                 currentline = line.strip().split(' ')
31                 #jump the head the end and space line
32                 if currentline[0] == "#!MLF!#" or len(currentline)==0 or
currentline[0] == '.':
33                     continue
34                 # read the head, and initail the sub-dictionary
35                 elif len(currentline)==1:
36                     head='\'*'+currentline[0][-55:]
37                     _dict[head]={ 'word':[], 'score':[], 'start':[], 'end':[] }
38                 # read the entries information
39                 else:
40                     _dict[head][ 'start' ].append(currentline[0])
41                     _dict[head][ 'end' ].append(currentline[1])
42                     _dict[head][ 'word' ].append(currentline[2])
43                     _dict[head][ 'score' ].append(currentline[3])
44     except IOError as ioerr:
45         print "File %s does not exist" % (filepath)
46     return _dict
47
48 #cost evaluation
49 def costfunction(entry1,entry2,ind1,ind2):
50     first=entry1[ 'word' ][ind1]
51     second=entry2[ 'word' ][ind2]
52     if len(first) > len(second):
53         first,second = second,first
54
55     first_length = len(first) + 1
56     second_length = len(second) + 1
57     distance_matrix = [range(second_length) for x in range(first_length
)]
58     #print distance_matrix
59     for i in range(1,first_length):
60         for j in range(1,second_length):

```

```

61         deletion = distance_matrix[i-1][j] + 1
62         insertion = distance_matrix[i][j-1] + 1
63         substitution = distance_matrix[i-1][j-1]
64         if first[i-1] != second[j-1]:
65             substitution += 1
66         distance_matrix[i][j] = min(insertion, deletion, substitution)
67     pw=distance_matrix[first_length-1][second_length-1]
68     ptb=float(entry1['start'][ind1])-float(entry2['start'][ind2])
69     pte=float(entry1['end'][ind1])-float(entry2['end'][ind2])
70     pt=abs(ptb+pte)/1e7
71     penalty=pt+pw
72     return penalty
73
74 """
75 ROVER combination usi7g Dynamic programming
76
77 Data structure:
78 ref={'filename1':{'word':[], 'start':[], 'end':[], 'score':[]}, 'filename2':{...}}
79 record=[('f1 or f2 or X',index)]
80 penalty=[[s11,s12,...],[s21,s22,...],...]
81 """
82
83 def merge_mlf(dict1, dict2):
84     mlfcom={}
85     for head in dict1.keys():
86         f1=dict1[head]
87         if head not in dict2.keys():
88             f2={'word':[], 'start':[], 'end':[], 'score':[]}
89         else:
90             f2=dict2[head]
91
92     #1. DP intialisation
93     penalty=[] # store penalty information in each step chosen
94     record=[] # store moving step information in each step chosen
95     # assign initial value to the first column and row in penalty and record matrix
96     for i in range(len(f1['word'])+1):
97         penalty.append([])
98         record.append([])
99         for j in range(len(f2['word'])+1):
100             penalty[i].append(0)
101             record[i].append('')
102     #initialisation
103     record[0][0] = ('Begin', '')
104     for i in range(1,len(f1['word'])+1):

```

```

105     penalty[i][0] = i*delpenalty
106     record[i][0] = ('f1', i-1)
107 for j in range(1, len(f2['word'])+1):
108     penalty[0][j] = j*delpenalty
109     record[0][j] = ('f2', j-1)
110
111 #2. Matrix Fill (scoring)
112 for i in range(1, len(f1['word'])+1):
113     for j in range(1, len(f2['word'])+1):
114         # mlf_A match mlf_B
115         if f1['word'][i-1] == f2['word'][j-1]:
116             f1p=(( 'f1m', i-1, j-1), penalty[i-1][j])
117         # mlf_A match !NULL in mlf_B
118         else:
119             f1p=(( 'f1', i-1), penalty[i-1][j]+\
120                 delpenalty)
121         # !NULL in mlf_A match mlf_B
122         f2p=(( 'f2', j-1), penalty[i][j-1]+\
123             delpenalty)
124         # mlf_A subsitute mlf_B
125         f12p=(( 'X', i-1, j-1), penalty[i-1][j-1]+\
126             costfunction(f1, f2, i-1, j-1))
127         # choose moving step by selecting the minimum penalty
128         moving=min(f1p, f2p, f12p, key=lambda x: x[1])
129         #store the penalty and record information to corresponding
matrix
130         penalty[i][j]=moving[1]
131         record[i][j]=moving[0]
132 #3. Trackback (alignment)
133 path = []
134 i = len(record)-1
135 j = len(record[0])-1
136 while record[i][j][0] != 'Begin':
137     if record[i][j][0] == 'f1':
138         path.append((record[i][j][1], '!NULL'))
139         i-=1
140     elif record[i][j][0] == 'f2':
141         path.append(('!NULL', record[i][j][1]))
142         j -= 1
143     elif record[i][j][0] == 'X':
144         path.append((record[i][j][1], record[i][j][2]))
145         i -= 1
146         j -= 1
147     elif record[i][j][0] == 'f1m':
148         path.append((record[i][j][1], record[i][j][2], 'match'))
149         i -= 1

```

```

150         j -= 1
151     else:
152         print "An unknown error occurs in traceback"
153     mlfcom[head]=list(reversed(path))
154     # print "DP process completed."
155     return mlfcom
156
157 # rewrite DP results into required format MLF file , !NULL score is set
    as 0.2
158 def recovermlf(dict1 , dict2 , mlfcom , NULL_Score=0.2):
159     comdict={}
160     for head in mlfcom.keys():
161         comdict[head]={ 'word':[] , 'start':[] , 'end':[] , 'score':[] }
162         for step in mlfcom[head]:
163             if 'match' in step:
164                 w=dict1[head][ 'word' ][ step[0]]
165                 s=str(( float(dict1[head][ 'score' ][ step[0]])+float(dict2[head][ '
score' ][ step[1]]))/2.0)
166                 start=dict1[head][ 'start' ][ step[0]]
167                 end=dict1[head][ 'end' ][ step[0]]
168             elif step[0]=='!NULL':
169                 w=dict2[head][ 'word' ][ step[1]]+ '_<ALTSTART>_<ALTEND>'
170                 s=dict2[head][ 'score' ][ step[1]]+ '_'+str(NULL_Score)
171                 start=dict2[head][ 'start' ][ step[1]]
172                 end=dict2[head][ 'end' ][ step[1]]
173             elif step[1]=='!NULL':
174                 w=dict1[head][ 'word' ][ step[0]]+ '_<ALTSTART>_<ALTEND>'
175                 s=dict1[head][ 'score' ][ step[0]]+ '_'+str(NULL_Score)
176                 start=dict1[head][ 'start' ][ step[0]]
177                 end=dict1[head][ 'end' ][ step[0]]
178             else:
179                 w=dict1[head][ 'word' ][ step[0]]+ '_<ALTSTART>_'+dict2[head][ 'word
' ][ step[1]]+ '_<ALTEND>'
180                 s=dict1[head][ 'score' ][ step[0]]+ '_'+dict2[head][ 'score' ][ step
[1]]
181                 start=dict1[head][ 'start' ][ step[0]]
182                 end=dict1[head][ 'end' ][ step[0]]
183                 comdict[head][ 'word' ].append(w)
184                 comdict[head][ 'score' ].append(s)
185                 comdict[head][ 'start' ].append(start)
186                 comdict[head][ 'end' ].append(end)
187     return comdict
188
189 # save final results into MLF file
190 def save_dict_to_file(_dict , filepath):
191     try:

```

```

192     with open(filepath, 'w') as sf:
193         sf.write(''#!MLF#\n'')
194         for head in _dict.keys():
195             sf.write(''%s\n'% head)
196             for i in range(len(_dict[head]['word'])):
197                 if _dict[head]['start'][0]==0 and _dict[head]['end'][0]==1:
198                     break
199                 else:
200                     sf.write('%s %s %s %s\n'%(_dict[head]['start'][i], _dict[
head]['end'][i],\
201                         _dict[head]['word'][i], _dict[head]['score'][i]))
202             sf.write('\n')
203     except IOError as ioerr:
204         print "File %s unable to generate, \nPlease checking the path and
the storage" % (filepath)
205
206     # selecting the word with highest score in each case, and save it to
mlf file
207     def bestpath(inputdir, outputdir):
208         _dict=load_dict_from_file(inputdir)
209         for head in _dict.keys():
210             for i in range(len(_dict[head]['word'])):
211                 w=filter(lambda x: len(x) > 0, re.split(r"<ALTSTART>_|<ALT>_|<
ALTEND>", _dict[head]['word'][i]))
212
213                 s=list(float(s) for s in str(_dict[head]['score'][i]).split('_'))
214                 for j in range(len(s)):
215                     if s[j]==max(s):
216                         ind=j
217                         _dict[head]['word'][i]=w[ind]
218                         _dict[head]['score'][i]=s[ind]
219         save_dict_to_file(_dict, outputdir)
220
221     # load the dataset list
222     def readlist(filepath):
223         try:
224             with open(filepath, 'r') as fs:
225                 store=[]
226                 for line in fs:
227                     filename=line.strip().split()
228                     if len(filename) == 1:
229                         store.append(filename[0])
230             return store
231         except IOError as ioerr:
232             print "error happens when writing %s" %filepath
233

```



```

234 def main(c1,c2,decodetype, testlist, pass1, pass2):
235     comb=c1+"+"+c2+"_"+decodetype
236     # print 'ok'
237     # print testlist
238     files=readlist(testlist)
239     for f in files:
240         # for CN mlf files
241         if decodetype=="decode_cn":
242             _dict1 = load_dict_from_file(r'/home/jp697/Major/%s/%s/%s/%s/
rescore_mappingtrees.mlf'%(pass1, c1,f,decodetype))
243             _dict2 = load_dict_from_file(r'/home/jp697/Major/%s/%s/%s/%s/
rescore_mappingtrees.mlf'%(pass2, c2,f,decodetype))
244             # for original mlf files
245             elif decodetype=='decode':
246                 _dict1 = load_dict_from_file(r'/home/jp697/Major/%s/%s/%s/%s/
rescore.mlf'%(pass1,c1,f,decodetype))
247                 _dict2 = load_dict_from_file(r'/home/jp697/Major/%s/%s/%s/%s/
rescore.mlf'%(pass2,c2,f,decodetype))
248             # for 2nd combination
249             else:
250                 _dict1 = load_dict_from_file(r'/home/jp697/Major/mlf_combine/%s/
combine/%s/decode_cn/rescore.mlf'%(c1,f))
251                 _dict2 = load_dict_from_file(r'/home/jp697/Major/mlf_combine/%s/
combine/%s/decode_cn/rescore.mlf'%(c2,f))
252                 mlfcom=merge_mlf(_dict1,_dict2)
253                 dictcom = recovermlf(_dict1,_dict2, mlfcom)
254                 save_dict_to_file(dictcom,r'/home/jp697/Major/mlf_combine/%s/%s.mlf
'%(comb,f))
255                 bestpath(r'/home/jp697/Major/mlf_combine/%s/%s.mlf'%(comb,f),\
256                     r'/home/jp697/Major/mlf_combine/%s/combine/%s/%s/rescore.mlf'%(
comb,f,decodetype))
257
258 if __name__ == '__main__':
259     import argparse
260     parser=argparse.ArgumentParser(description='Combine two mlf files')
261     parser.add_argument('sys1',type=str)
262     parser.add_argument('sys2',type=str)
263     parser.add_argument('decode_type',type=str)
264     parser.add_argument('testlist',type=str)
265     parser.add_argument('--pass1',type=str)
266     parser.add_argument('--pass2',type=str)
267     args=parser.parse_args()
268     main(args.sys1,args.sys2,args.decode_type, args.testlist, args.pass1,
args.pass2)

```

A.1.4 CNC.py

CNC.py

```
1  #!/usr/env/bin python
2  # -*-UTF-8-*-
3  '''
4  Confusion Network Combination
5
6  The CNC script is similar to ROVER script. The difference is that CNC
7  use the lattices to perform the combination rather than
8  MLF files. In this case, the DP process used in CNC is exactly the same
9  as ROVER. The script will first convert the lattice
10 into MLF file format, and doing the combination and best path selecting
11 .
12
13 Usage: CNC.py system_A system_B dataset_list --pass1 --pass2
14 -system_A: the lattice A
15 -system_B: the lattice B
16 -dataset_list: the list of dataset using in this case
17 -pass1 2: the pass directory of the mlf,2 files
18
19 Author: Junjie Pan
20 Latest Modified: 2016/04/26
21 '''
22 import os, sys
23 import gzip
24 import re
25 import numpy as np
26 #gloabl setting
27 delpenalty=3 #deletion and insertion penalty
28 altpenalty=5 #substitution penalty
29
30 # selecting the entries with highest scores in each turn
31 def bestpath(inputdir, outputdir):
32     _dict=load_dict_from_file(inputdir)
33     for head in _dict.keys():
34         for i in range(len(_dict[head]['word'])):
35             w=filter(lambda x: len(x) > 0, re.split(r"<ALTSTART>_|<ALT>_|<
36             ALTEND>", _dict[head]['word'][i]))
37             s=list(float(s) for s in str(_dict[head]['score'][i]).split('_'))
38             for j in range(len(s)):
39                 if s[j]==max(s):
40                     ind=j
41             _dict[head]['word'][i]=w[ind]
```

```

39         _dict[_dict['score'][i]]=round(np.exp(float(s[ind])),6)
40
41     save_dict_to_file(_dict,outputdir)
42
43     # read the lattices list from the target directory
44     def read_lattice(filepath):
45         lattice = {}
46         for head in os.listdir(filepath):
47             # initialise the variables
48             path = filepath + head
49             head='\"/' +head[:-6]+'rec\"'
50             lattice[head]={ 'word':[], 'start':[], 'end':[], 'score':[] }
51             words=[]
52             starts=[]
53             ends=[]
54             scores=[]
55             # for each lattice
56             with gzip.open(path, 'rb') as fs:
57                 flag=0 # 0 for start a new turn, 1 for process the current turn
58                 jump=0 # 0 for start reading lattice, 1 for end reading lattice
59                 for line in fs:
60                     info=line.strip().split()
61                     # skip the starting line
62                     if len(info)==1 and info[0][0]!='N':
63                         flag=1
64                         jump=0
65                         word=''
66                         count=int(info[0][2:])
67                         continue
68                     # start reading lattice
69                     elif flag==1 and jump==0:
70                         # end of the lattice
71                         if info[0][2:]=='<s>':
72                             count-=1
73                             jump=1
74                             continue
75                         # the begining of lattice
76                         elif info[0][2:]=='</s>':
77                             count-=1
78                             flag=0
79                             continue
80                     else:
81                         #start word of alternative path
82                         if word=='' and count>1:
83                             word=info[0][2:]+'_<ALTSTART>_'
84                             score=info[3][2:]+'_'

```

```

85         start=float(info[1][2:])
86         end=float(info[2][2:])
87         count-=1
88     #unique path
89     elif word==' ' and count==1:
90         word=info[0][2:]
91         score=info[3][2:]
92         start=float(info[1][2:])
93         end=float(info[2][2:])
94         flag=0
95     #middle of alternative path
96     elif count>1:
97         word+=info[0][2:] + ' _<ALT> _ '
98         score+=info[3][2:] + ' _ '
99         start=max(float(info[1][2:]),start)
100        end=max(float(info[2][2:]),end)
101        count-=1
102    #end of alternative path
103    elif count==1:
104        word+=info[0][2:] + ' _<ALTEND> '
105        score+=info[3][2:]
106        start=max(float(info[1][2:]),start)
107        end=max(float(info[2][2:]),end)
108        flag=0
109    else:
110        continue
111    # end one turn, store the current results
112    if flag==0:
113        words.append(word)
114        scores.append(score)
115        starts.append(int(start*(1e7)))
116        ends.append(int(end*(1e7)))
117        lattice[head]['word']=list(reversed(words))
118        lattice[head]['start']=list(reversed(starts))
119        lattice[head]['end']=list(reversed(ends))
120        lattice[head]['score']=list(reversed(scores))
121    return lattice
122
123    # read MLF files (for bestpath selecting)
124    def load_dict_from_file(filepath):
125        try:
126            _dict = {}
127            with open(filepath, 'r') as dict_file:
128                for line in dict_file:
129                    currentline = line.strip().split(' ')
130                    #jump the head the end and space line

```

```

131         if currentline[0] == "#!MLF!" or len(currentline)==0 or
currentline[0] == '.' :
132             continue
133         # read the head, and initial the sub-dictionary
134         elif len(currentline)==1:
135             head='\"'+currentline[0][-55:]
136             _dict[head]={ 'word':[], 'score':[], 'start':[], 'end':[] }
137         # read the entries information
138         else:
139             _dict[head][ 'start' ].append(currentline[0])
140             _dict[head][ 'end' ].append(currentline[1])
141             _dict[head][ 'word' ].append(currentline[2])
142             _dict[head][ 'score' ].append(currentline[3])
143     except IOError as ioerr:
144         print "File %s does not exist" % (filepath)
145     return _dict
146
147 # save dictionary into suitable MLF file
148 def save_dict_to_file(_dict, filepath):
149     try:
150         with open(filepath, 'w') as sf:
151             sf.write('\"#!MLF!#\n\"')
152             for head in _dict.keys():
153                 sf.write('\"%s\n\"'% head)
154                 for i in range(len(_dict[head][ 'word' ])):
155                     if _dict[head][ 'start' ][0]==0 and _dict[head][ 'end' ][0]==1:
156                         break
157                     else:
158                         sf.write("%s %s %s %s\n"%(_dict[head][ 'start' ][i], _dict[
head][ 'end' ][i],\
159                             _dict[head][ 'word' ][i], _dict[head][ 'score' ][i]))
160                 sf.write('.\n')
161     except IOError as ioerr:
162         print "unable %s" % (filepath)
163
164 #cost evaluation
165 def costfunction(entry1, entry2, ind1, ind2):
166     first=entry1[ 'word' ][ind1]
167     second=entry2[ 'word' ][ind2]
168     if len(first) > len(second):
169         first, second = second, first
170
171     first_length = len(first) + 1
172     second_length = len(second) + 1
173     distance_matrix = [range(second_length) for x in range(first_length
)]

```

```

174     #print distance_matrix
175     for i in range(1,first_length):
176         for j in range(1,second_length):
177             deletion = distance_matrix[i-1][j] + 1
178             insertion = distance_matrix[i][j-1] + 1
179             substitution = distance_matrix[i-1][j-1]
180             if first[i-1] != second[j-1]:
181                 substitution += 1
182             distance_matrix[i][j] = min(insertion,deletion,substitution)
183     pw=distance_matrix[first_length-1][second_length-1]
184     ptb=float(entry1['start'][ind1])-float(entry2['start'][ind2])
185     pte=float(entry1['end'][ind1])-float(entry2['end'][ind2])
186     pt=abs(ptb+pte)/1e7
187     penalty=pt+pw
188     return penalty
189
190 """
191 CNC using Dynamic programming, same as the ROVER DP
192
193 Data structure:
194 ref={'filename1':{'word':[],'start':[],'end':[],'score':[]},'filename2':{...}}
195 record=[('f1 or f2 or X',index)]
196 penalty=[[s11,s12,...],[s21,s22,...],...]
197 """
198 def merge_mlf(dict1, dict2):
199     mlfcom={}
200     for head in dict1.keys():
201         f1=dict1[head]
202         if head not in dict2.keys():
203             f2={'word':[],'start':[],'end':[],'score':[]}
204         else:
205             f2=dict2[head]
206
207     #1. DP intialisation
208     penalty=[] # store penalty information in each step chosen
209     record=[] # store moving step information in each step chosen
210     # assign initial value to the first column and row in penalty and record matrix
211     for i in range(len(f1['word'])+1):
212         penalty.append([])
213         record.append([])
214         for j in range(len(f2['word'])+1):
215             penalty[i].append(0)
216             record[i].append('')
217     #initialisation

```

```

218 record[0][0] = ( 'Begin' , '' )
219 for i in range(1,len(f1[ 'word' ])+1):
220     penalty[i][0] = i*delpenalty
221     record[i][0] = ( 'f1' , i-1)
222 for j in range(1,len(f2[ 'word' ])+1):
223     penalty[0][j] = j*delpenalty
224     record[0][j] = ( 'f2' , j-1)
225
226 #2. Matrix Fill (scoring)
227 for i in range(1,len(f1[ 'word' ])+1):
228     for j in range(1,len(f2[ 'word' ])+1):
229         # mlf_A match mlf_B
230         if f1[ 'word' ][i-1] == f2[ 'word' ][j-1]:
231             f1p=(( 'f1m' , i-1, j-1), penalty[i-1][j])
232         # mlf_A match !NULL in mlf_B
233         else:
234             f1p=(( 'f1' , i-1), penalty[i-1][j]+\
235                 delpenalty)
236         # !NULL in mlf_A match mlf_B
237         f2p=(( 'f2' , j-1), penalty[i][j-1]+\
238             delpenalty)
239         # mlf_A subsitute mlf_B
240         f12p=(( 'X' , i-1, j-1), penalty[i-1][j-1]+\
241             costfunction(f1,f2,i-1,j-1))
242         # choose moving step by selecting the minimum penalty
243         moving=min(f1p,f2p,f12p,key=lambda x: x[1])
244         #store the penalty and record information to corresponding
245         matrix
246         penalty[i][j]=moving[1]
247         record[i][j]=moving[0]
248 #3. Trackback (alignment)
249 path = []
250 i = len(record)-1
251 j = len(record[0])-1
252 while record[i][j][0] != 'Begin':
253     if record[i][j][0] == 'f1':
254         path.append((record[i][j][1], '!NULL'))
255         i-=1
256     elif record[i][j][0] == 'f2':
257         path.append(( '!NULL', record[i][j][1]))
258         j -= 1
259     elif record[i][j][0] == 'X':
260         path.append((record[i][j][1], record[i][j][2]))
261         i -= 1
262         j -= 1
263     elif record[i][j][0] == 'f1m':

```

```

263         path.append((record[i][j][1], record[i][j][2], 'match'))
264         i -= 1
265         j -= 1
266     else:
267         print "An unknown error occurs in traceback"
268     mlfcom[head]=list(reversed(path))
269     # print "DP process completed."
270     return mlfcom
271
272 # rewrite DP results into required format MLF file, !NULL score is set
    as 0.2
273 def recovermlf(dict1, dict2, mlfcom, NULL_Score=0.2):
274     comdict={}
275     for head in mlfcom.keys():
276         comdict[head]={ 'word':[], 'start':[], 'end':[], 'score':[] }
277         for step in mlfcom[head]:
278             if 'match' in step:
279                 w=dict1[head][ 'word' ][ step[0]]
280                 s=str(( float(dict1[head][ 'score' ][ step[0]])+float(dict2[head][ '
score' ][ step[1]]))/2.0)
281                 start=dict1[head][ 'start' ][ step[0]]
282                 end=dict1[head][ 'end' ][ step[0]]
283             elif step[0]=='!NULL':
284                 w=dict2[head][ 'word' ][ step[1]]+ '_<ALTSTART>_<ALTEND>'
285                 s=dict2[head][ 'score' ][ step[1]]+ '_'+str(NULL_Score)
286                 start=dict2[head][ 'start' ][ step[1]]
287                 end=dict2[head][ 'end' ][ step[1]]
288             elif step[1]=='!NULL':
289                 w=dict1[head][ 'word' ][ step[0]]+ '_<ALTSTART>_<ALTEND>'
290                 s=dict1[head][ 'score' ][ step[0]]+ '_'+str(NULL_Score)
291                 start=dict1[head][ 'start' ][ step[0]]
292                 end=dict1[head][ 'end' ][ step[0]]
293             else:
294                 w=dict1[head][ 'word' ][ step[0]]+ '_<ALTSTART>_'+dict2[head][ 'word
' ][ step[1]]+ '_<ALTEND>'
295                 s=dict1[head][ 'score' ][ step[0]]+ '_'+dict2[head][ 'score' ][ step
[1]]
296                 start=dict1[head][ 'start' ][ step[0]]
297                 end=dict1[head][ 'end' ][ step[0]]
298                 comdict[head][ 'word' ].append(w)
299                 comdict[head][ 'score' ].append(s)
300                 comdict[head][ 'start' ].append(start)
301                 comdict[head][ 'end' ].append(end)
302     return comdict
303
304 # read dataset list

```



```

305 def readlist(filepath):
306     try:
307         with open(filepath, 'r') as fs:
308             store=[]
309             for line in fs:
310                 filename=line.strip().split()
311                 if len(filename) == 1:
312                     store.append(filename[0])
313             return store
314     except IOError as ioerr:
315         print "error happens when writing %s" %filepath
316
317 '''
318 main function: to implement the CNC
319 c1: system 1 eg: plp-bg
320 c2: system 2 eg: grph-plp-bg
321 dataset: the file store the dataset list eg:challenge_dev
322 challenge: optional (challenge or .) To determine whether it is the
323             dev03/eval03 task or challenge dataset task
324             Default value - challenge dataset
325
326 The output is in ./Major/challenge/cnc_mlf directory, where challenge
327 is determined by the last parameter
328 '''
329 def main(c1,c2, decodetype,dataset,pass1, pass2):
330     files=readlist(dataset)
331     comb='%s+%s_%s'%(c1,c2, decodetype)
332     for f in files:
333         #read lattices A
334         _dict1 = read_lattice(r'/home/jp697/Major/%s/%s/%s/%s/lattices/'\
335                               %(pass1,c1,f,decodetype))
336         #read lattices B
337         _dict2 = read_lattice(r'/home/jp697/Major/%s/%s/%s/%s/lattices/'\
338                               %(pass2,c2,f,decodetype))
339         # save lattice A into mlf file
340         save_dict_to_file(_dict1,r'/home/jp697/Major/%s/%s/%s/%s/\
341                             rescore_lattices.mlf'\
342                             %(pass1,c1,f,decodetype))
343         # save lattice B into mlf file
344         save_dict_to_file(_dict2,r'/home/jp697/Major/%s/%s/%s/%s/\
345                             rescore_lattices.mlf'\
346                             %(pass2,c2,f,decodetype))
347         bestpath(r'/home/jp697/Major/%s/%s/%s/%s/rescore_lattices.mlf'%(
348             pass1,c1,f, decodetype),\
349                 r'/home/jp697/Major/%s/%s/%s/%s/rescore_lattices_rover.mlf'%(
350             pass1,c1,f, decodetype))

```

```

345     bestpath(r'/home/jp697/Major/%s/%s/%s/%s/rescore_lattices.mlf'%(
pass2,c2,f,decodetype),\
346     r'/home/jp697/Major/%s/%s/%s/%s/rescore_lattices_rover.mlf'%(
pass2,c2,f,decodetype))
347     # doing the confidence score mapping
348     os.popen('echo | base/conftools/smoothtree-mlf.pl lib/trees/plp-
bg_decode_cn.tree\
349     ./%s/%s/%s/%s/rescore_lattices_rover.mlf \
350 > ./%s/%s/%s/%s/rescore_lattices_rover_mappingtrees.mlf'%(pass1,c1,f,
decodetype,pass1,c1,f,decodetype))
351     os.popen('echo | base/conftools/smoothtree-mlf.pl lib/trees/plp-
bg_decode_cn.tree\
352     ./%s/%s/%s/%s/rescore_lattices_rover.mlf \
353 > ./%s/%s/%s/%s/rescore_lattices_rover_mappingtrees.mlf'%(pass2,c2,f,
decodetype,pass2,c2,f,decodetype))
354     #load MLF files
355     _dict1 = load_dict_from_file(r'/home/jp697/Major/%s/%s/%s/%s/
rescore_lattices_rover_mappingtrees.mlf'%(pass1,c1,f,decodetype))
356     _dict2 = load_dict_from_file(r'/home/jp697/Major/%s/%s/%s/%s/
rescore_lattices_rover_mappingtrees.mlf'%(pass2,c2,f,decodetype))
357     #Doing the CNC
358     mlfcom=merge_mlf(_dict1,_dict2)
359     dictcom = recovermlf(_dict1,_dict2,mlfcom)
360     save_dict_to_file(dictcom,r'/home/jp697/Major/mlf_cnc/%s/%s.mlf'%(
comb,f))
361     #Selecting the best path and save into MLF file
362     bestpath(r'/home/jp697/Major/mlf_cnc/%s/%s.mlf'%(comb,f),\
363     r'/home/jp697/Major/mlf_cnc/%s/combine/%s/decode_cn/rescore.mlf'
%(comb,f))
364
365 if __name__ == '__main__':
366     import argparse
367     parser=argparse.ArgumentParser(description='CNC')
368     parser.add_argument('sys1',type=str)
369     parser.add_argument('sys2',type=str)
370     parser.add_argument('decodetype',type=str)
371     parser.add_argument('dataset',type=str)
372     parser.add_argument('--pass1',type=str)
373     parser.add_argument('--pass2',type=str)
374     args = parser.parse_args()
375     main(args.sys1, args.sys2, args.decodetype, args.dataset, args.pass1,
args.pass2)

```

A.2 Bash Scripts

A.2.1 interpolation_challenge.sh

Interpolation_challenge.sh

```
1 # '''
2 # Interpolate the LMs
3 # Process:
4 #   a. Convert 1-best output to data file
5 #   b. Generate stream files from the data file in step a
6 #   c. Interpolate the Language Model
7 # Output:
8 #   a. ./challenge/store_dat/YTBEddev.dat - data file generated from 1
9 #       best hypothesis
10 #   b. ./challenge/streams - streams generated from LM1~5
11 #   c. ./exp/temp_file/weight_challenge_dev - Estimation of
12 #       interpolation weights
13 #   ./lm_int_challenge - Interpolation LM
14 # '''
15 store='/home/jp697/Major/exp/temp_file '
16 challengepath="/home/jp697/Major/challenge/streams/"
17 storedat="/home/jp697/Major/challenge/store_dat/"
18
19 #step a
20 python /home/jp697/Major/exp/shell/ConvertData.py \
21 /home/jp697/Major/exp/temp_file/temp1 \
22 challenge/lm_int_plp /home/jp697/Major/challenge/store_dat/YTBEddev.dat
23
24 #step b
25 echo "Begin to generate the stream files"
26 for ((j=1;j<=5;j++))
27 do
28     base/bin/LPlex -C lib/cfgs/hlm.cfg -s stream${j} -u -t \
29     lms/lm${j} ${storedat}/dev03.dat
30     cp stream${j} ${challengepath}
31     rm stream${j}
32 done
33
34 echo "begin to interpolate the weights"
35 # step c
36 calculate the weights
37 python "/home/jp697/Major/exp/shell/interpolation.py" /home/jp697/Major
38 /challenge/streams /home/jp697/Major/exp/temp_file/
39 weight_challenge_dev
40 #Read the weights and interpolation
```

```

37 echo "begin to merge the LMs"
38 j=0
39 weight=[]
40 while read line
41 do
42     weight[j]=$line
43     let j=j+1
44     echo $j
45 done < ${store}/weight_challenge_dev
46 echo "Start the weight_challenge_dev LM merge"
47 echo ${weight[0]}
48 base/bin/LMerge -C lib/cfgs/hlm.cfg \
49 -i ${weight[0]} lms/lm1 \
50 -i ${weight[1]} lms/lm2 \
51 -i ${weight[2]} lms/lm3 \
52 -i ${weight[3]} lms/lm4 \
53 lib/wlists/train.lst \
54 lms/lm5 lm_int_challenge

```

A.2.2 show-specific.sh

show-specific.sh

```

1 mainpath='/home/jp697/Major '
2 devpath='/home/jp697/Major/lattices '
3 store='/home/jp697/Major/exp/temp_file '
4 task1path="${mainpath}/exp/task1"
5
6 echo "start to create the 1best hypothesis"
7 echo \
8 '''\
9 step 1: use the dev03 generated LM to rescore each show in the eval
10 sets\
11 '''
12 while read line
13 do
14     ${mainpath}/scripts/lmrescore.sh $line lattices decode lm_int plp-
15     tglm_int TRUE
16 done < "$store/eval03"
17
18 #checking the completion of step 1
19 tgdir="plp-tglm_int" #"adapt-lm-plp" #"adapt-lm-grph" #"adapt-lm-tandem
20 " #"adapt-lm-grph-tandem" #"lm_int_plp" #"lm_int-grph" #"
21 lm_int-tandem" #"lm_int-grph-tandem"
22 passdir="rescore" #"decode" #"decode-cn" #"adapt"

```

```

19 outdir='rescore.mlf' #rescore.mlf #LOG.align
20 while read line
21 do
22     while (true)
23     do
24         test -e ".$${tgdir}/${line}/${passdir}/${outdir}" && break
25     done
26     echo "$line finished!"
27 done < $store/eval03
28
29
30 echo \
31 ',,\
32 step 2: generate the 1best hypothesis from the eval lattices\
33 ',,\
34
35 while read line
36 do
37     ./scripts/1bestlats.sh $line ./plp-tglm-int rescore plp-tglm-int
38     echo "complete one task"
39 done < "$store/eval03"
40
41 #checking the completion of step 2
42 tgdir="plp-tglm-int" #"adapt-lm-plp" #"adapt-lm-grph" #"adapt-lm-tandem
43     #"adapt-lm-grph-tandem" #"lm-int-plp" #"lm-int-grph" #"
44     lm-int-tandem" #"lm-int-grph-tandem"
45 passdir="1best/LM12.0_IN-10.0" #"decode" #"decode-cn" #"adapt"
46 outdir='rescore.mlf' #rescore.mlf #LOG.align
47 while read line
48 do
49     while (true)
50     do
51         test -e ".$${tgdir}/${line}/${passdir}/${outdir}" && break
52     done
53     echo "$line finished!"
54 done < $store/eval03
55
56
57 echo \
58 ',,\
59 step 3: convert the 1best hypothesis into suitable format\
60 ',,\
61 python ./exp/shell/ConvertData.py - $store/eval03 plp-tglm-int ./exp/
62     temp_file/eval03.dat
63
64 echo \

```

```

62  ', '
63  step 4: generate streams from the converted data file
64  ', '
65  for (( i=1;i<=5;i++))
66  do
67      echo | base/bin/LPlex -C lib/cfgs/hlm.cfg -s stream_eval${i} -u -t
        lms/lm${i} ./exp/temp_file/eval03.dat \
68  >> exp/5.4.6.LOG
69      cp stream_eval${i} ${task1path}
70      rm stream_eval${i}
71  done
72
73  echo \
74  ', '
75  step 5: compute the weights from the streams generated from last step
76  ', '
77  python "/home/jp697/Major/exp/shell/Interpolation.py" ${task1path} ${
        store}/weight_eval
78
79
80  echo \
81  ', '
82  step 6: generate the interpolation LM according to the weights
83  ', '
84  weight=[]
85  while read line
86  do
87      weight[j]=$line
88      let j=j+1
89      echo $j
90  done < ${store}/weight_eval
91
92  echo "Start the evalLM merge"
93  echo ${weight[0]}
94  base/bin/LMerge -C lib/cfgs/hlm.cfg -i ${weight[0]} lms/lm1 -i ${weight
        [1]} lms/lm2 -i ${weight[2]} lms/lm3 -i ${weight[3]} lms/lm4 lib/
        wlists/train.lst lms/lm5 lm_int_specific
95
96
97  echo \
98  ', '
99  step 7: use the new_interpolated LM to rescore the eval data_set
100  ', '
101  while read line
102  do

```

```

103 | ${mainpath}/scripts/lmrescore.sh -OUTPASS specific $line lattices
      decode lm_int-specific plp-tglm_int TRUE
104 | done < "$store/eval03"
105 |
106 | #checking the completion of step 7
107 | tgdir="plp-tglm_int" #"adapt-lm-plp" #"adapt-lm-grph" #"adapt-lm-tandem
      " #"adapt-lm-grph-tandem" #"lm_int-plp" #"lm_int-grph" #"
      lm_int-tandem" #"lm_int-grph-tandem"
108 | passdir="specific" #"decode" #"decode_cn" #"adapt"
109 | outdir='rescore.mlf' #rescore.mlf #LOG.align
110 | while read line
111 | do
112 |     while (true)
113 |     do
114 |         test -e "${tgdir}/${line}/${passdir}/${outdir}" && break
115 |     done
116 |     echo "$line finished!"
117 | done < $store/eval03
118 |
119 | echo \
120 | , , ,
121 | WER and perplexity for eval03 dataset from LM1-5, lm_int , and
      lm_int-specific
122 | , , ,
123 | echo 'scoring all the files '
124 | for ((i=1;i<=5;i++))
125 | do
126 |     echo "LM${i}" >> ./exp/5.4.6_LM1-5_eval
127 |     echo | ./scripts/score.sh plp-tglm-eval${i} eval03 rescore >> ./exp
      /5.4.6_LM1-5_eval
128 |     echo | base/bin/LPlex -C lib/cfgs/hlm.cfg -u -t lms/lm${i} lib/texts/
      eval03.dat >> ./exp/5.4.6_LM1-5_eval
129 | done
130 | echo "int_LM" >> ./exp/5.4.6_LM1-5_eval
131 | echo | ./scripts/score.sh plp-tglm_int/eval03 eval03 rescore >> ./exp
      /5.4.6_LM1-5_eval
132 | echo | base/bin/LPlex -C lib/cfgs/hlm.cfg -u -t lm_int lib/texts/eval03
      .dat >> ./exp/5.4.6_LM1-5_eval
133 |
134 | echo "int_specified_lm" >> ./exp/5.4.6_LM1-5_eval
135 | echo | ./scripts/score.sh plp-tglm_int/eval03 eval03 specific >> ./exp
      /5.4.6_LM1-5_eval
136 | echo | base/bin/LPlex -C lib/cfgs/hlm.cfg -u -t lm_int-specific lib/
      texts/eval03.dat >> ./exp/5.4.6_LM1-5_eval

```