UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning, Speech and Language Technology
2015-2016

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

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** ☐	**A+** ☐	**A** ☐	**A-** ☐	**B+** ☐	**B** ☐	**C+** ☐	**C** ☐	**Unsatisfactor** ☐
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 five 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, Confusion Network Combination and their implementation 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 from the Challenge evaluation dataset were also included. Python and bash 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

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

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

Its error rate is shown in Table 1:

Table 1: Scoring Result for 1-best Hypothesis

SPKR	# Snt	# Wrd	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 given LMs were applied to the **dev03** development dataset, and the outputs were rescored. Results are shown in table 2.

Table 2: Scoring Result for dev03 in LM1-5

LM	SPKR	# Snt	# Wrd	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 are supposed to show better performance.

The results shown in table 2 match the expectation above. It can be seen that the WER is `17.7%` on **LM1** and `18.6%` on **LM2**, which is the best 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

```
for ((j=1;j<=5;j++))
do
  base/bin/LPlex -C lib/cfgs/hlm.cfg -u -t lms/lm${j} \
  lib/texts/dev03.dat
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 2.2**.

2.4 LM interpolation

In order to interpolate language models, the following steps are taken to estimate interpolation weights and merge different LMs:

1. Obtain streams of probabilities of the five LMs
2. Estimate interpolation weights using *Interpolation.py* script
3. Merge LMs accoding to interpolation weights obtained in step 2

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

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

and

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

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 6 it is clear that the WER and perplexity of dev03 has been significantly improved by applying the interpolation. For **eval03** dataset, the WER is 15.0% and the perplexity is 154.8256, which are much better than the performance of the individual 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**[1]

   - 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 2.4 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.

[1] *interlolation_challenge.sh* is only used for Challenge development dataset

If the evaluation dataset are in the different topic from 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

```
while read line
do
  ./scripts/mergelats.sh $line  plp-bg rescore plp-bg
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

```
# plp-adapt-by-grph-plp as an example
# rescore the merged lattice with grph-plp system
while read line
do
```

```
  ./scripts/hmmrescore.sh $line plp merge grph-plp grph-plp
done < ${testlist}/dev03.lst
# perform cross-adaptation
while read line
do
  ./scripts/hmmadapt.sh $line grph-plp decode plp-adapt-grph-plp plp
done < ${testlist}/dev03.lst
# rescore the adapted lattice
while read line
do
  ./scripts/hmmrescore.sh -ADAPT plp-adapt-grph-plp adapt ${line} plp \
   merge plp-adapt-grph-plp plp
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 1 and step 3 using *score.sh* command

The cross-adaptation was applied to each system, and the scoring results are shown in table 9[2].

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

[2]The 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`[3] 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 handout). 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.

[3]The actual name for each adaptation is **Supervisor-adapt-Adaptation**, which is written as **Supervisor-adapt-by-Adaptation** in this report to avoid mis-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

$$
\begin{aligned}
&\{\\
&\{entry_1 : \{word : [w_1, w_2, ..., w_n],\ start : [t_1, ..., t_2],\ end : [e_1, ..., e_n],\ score : [s_1, ..., s_n]\},\\
&\{entry_2 : \{word : [w_1, w_2, ..., w_n],\ start : [t_1, ..., t_2],\ end : [e_1, ..., e_n],\ score : [s_1, ..., s_n]\},\\
&...\\
&\{entry_n : \{word : [w_1, w_2, ..., w_n],\ start : [t_1, ..., t_2],\ end : [e_1, ..., e_n],\ score : [s_1, ..., s_n]\}\\
&\}
\end{aligned}
$$

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

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

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

   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.

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

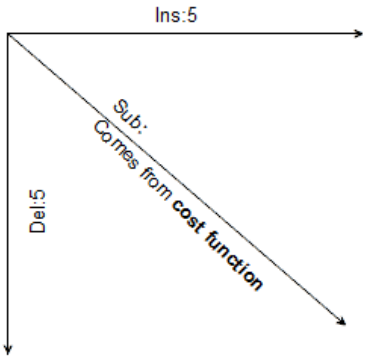

Figure 1: Penalty for deletion, insertion and substitution

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

$$
\begin{aligned}
&\{\\
&\quad entry_1 :\\
&\quad \{word : [!NULL, he\_\langle \text{ALTSTART} \rangle\_his\_\langle \text{ALT} \rangle\_!NULL\_\langle \text{ALTEND} \rangle, ...],\\
&\quad start : 0000000, end : 0000100, score : [1, 0.23 \_ 0.62 \_ 0.18, ...]\},\\
&\quad entry_2 : \{.....\}\\
&\quad .....\\
&\quad entry_n : \{.....\}\\
&\}
\end{aligned}
$$

***main* function**
Input: two lattices paths, decodetype[4], 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 files[5].
3. Load the mlf files from last step, select the best path from them and rewrite into the MLF files.

[4] '*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.

[5] In 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 11[6].

Table 11: scoring results of dev03 from combination systems

$System_1$	$System_2$	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%.

[6]All 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 be 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 was 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 produces 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

$System_1$	$System_2$	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 YTBEdev 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 Generatetion 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 YTBEdev 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 YTBEdev from combination systems

$System_1$	$System_2$	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, 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

```
# -*- coding: UTF-8 -*-
'''
ConvertData.py file
Function: Convert 1-best hypothesis into suitable data file, so that it
    can be used
to generate the interpolation weights

Description:
Usage: ConvertData.py dataset 1best_path savepath
-dataset: the data list. eg: challenge_dev, challenge_eval, dev03,
    eval03
-1best_path: the directory of the 1best hypothesis (after./Major)
-savepath: the directory to store the converted data file.

Author: Junjie Pan
Latest Modified: 2016/04/25
'''
#read the MLF file and save it into dictionary
def load_dict_from_file(filepath):
  try:
    _dict = {}
    with open(filepath, 'r') as dict_file:
      sent=0
      for line in dict_file:
        wordline = line.strip().split(' ')
        if wordline[0] == '.':
          sent+=1
        elif len(wordline)==1:
          continue
        else:
          if sent not in _dict.keys():
            _dict[sent]=[]
          _dict[sent].append(wordline[2])

  except IOError as ioerr:
      print "File %s does not exist" % (filepath)

```

```
  return _dict

# save the dictionary into reuqire data format
def save_dict_to_file(_dict, filepath):
  try:
    with open(filepath, 'a') as dict_file:
      for key in _dict.keys():
        dict_file.write('''<s> ''')
        for word in _dict[key]:
          dict_file.write(word+' ')
        dict_file.write('''</s>\n''')
  except IOError as ioerr:
    print "File %s unable to create" % (filepath)

#read dataset list
def readlist(filepath):
  try:
    with open (filepath,'r') as fs:
      store=[]
      for line in fs:
        filename=line.strip().split()
        if len(filename) == 1:
          store.append(filename[0][2:])
    return store
  except IOError as ioerr:
    print "error happens when writing %s" %filepath

def main(lists,path,output):
  files=readlist(r'/home/jp697/Major/exp/temp_file/%s'%lists)
  for f in files:
    _dict = load_dict_from_file(r'/home/jp697/Major/%s/%s/1best/LM12.0
    _IN-10.0/rescore.mlf'%(path,f))
    save_dict_to_file(_dict, output)

if __name__ == '__main__':
  import argparse
  parser=argparse.ArgumentParser(decription='Convert 1best Hyp into
    suitable format')
  parser.add_argumrnt('dataset_list',type=str)
  parser.add_argumrnt('1best_path',type=str)
  parser.add_argumrnt('output',type=str)
  args=parser.parse_args()
  main(args.dataset_list, args.1best_path, args.output)
```

A.1.2 Interpolation.py

Interpolation.py

```
#!usr/env/bin python
#-*-coding:UTF-8-*-
import numpy as np
'''
Interpolation.py

This script is to interpolate LM weights, with uniform intial weights
    setting

Usage: Interpolation.py stream_path weight_path
-stream_path: the directory of stream files
-weight_path: the directory to save the generated weights into file

Author: Junjie Pan
Latest Modified: 2016/04/26
'''

# read stream files into dictiory
def read_file(filepath):
  try:
    ls=[]
    with open(filepath,'r') as ls_file:
      for line in ls_file:
        p=float(line)
        ls.append(p)
    return ls
  except IOError as ioerr:
    print "File %s cannot be open"%filepath

#Interpolation LM weights
def inter(ls):
  #set the initial weights uniform
  w=np.array([0.2,0.2,0.2,0.2,0.2])
  PP=w[0]*ls[0]+w[1]*ls[1]+w[2]*ls[2]+w[3]*ls[3]+w[4]*ls[4]
  temp=0 #store the current perplexity
  while(abs(sum(PP)-temp)>1e-12):
    temp=sum(PP)
    # update rules
    for i in range(5):
      p=(w[i]*ls[i])/PP
      w[i]=np.mean(p)
    PP=w[0]*ls[0]+w[1]*ls[1]+w[2]*ls[2]+w[3]*ls[3]+w[4]*ls[4]
```

```
  return w

#main function to perform interpolation
def main(streamdir,weightsdir):
  ls=[]
  for j in range(5):
    ls.append([])
    ls[j]=read_file("%s/stream%d" % (streamdir,j+1))
  ls=np.array(ls)
  weight=inter(ls)

  weightfile=open(weightsdir,'w')
  for w in weight:
    weightfile.write(str(w)+'\n')
  weightfile.close()

if __name__=='__main__':
  import argparser
  parser=arg.ArgumentParser(description='Interpolation LM Weights')
  parser.add_argument('StreamPath',type=str)
  parser.add_argument('WeightPath',type=str)
  args=parser.parse_args()
  main(args.StreamPath, args.WeightPath)
```

A.1.3 RoverCombi.py

RoverCombi.py

```
#!usr/env/bin python
# -*- coding: UTF-8 -*-
'''
ROVER combination script

This script uses DP to perform alignment to the two input MLF files.
The final version of MLF will be stored in ./Major/challenge/
    mlf_combination

Usage: RoverCombi.py mlf_A mlf_B decodetype testlist --pass1 --pass2
-mlf_A: reference mlf file
-mlf_B: the other mlf file
-decodetype: decode or decode_cn
-testlist: the list of dataset
-pass1 2: the pass directory of the mlf1,2 files

Author: Junjie Pan
```

```
Latest Modified: 2016/04/26
'''
import re
from numpy import *
#global setting
delpenalty=5 #deletion and insertion penalty

#load the MLF file and convert it into python dictionary
def load_dict_from_file(filepath):
  try:
    _dict = {}
    with open(filepath, 'r') as dict_file:
        for line in dict_file:
          currentline = line.strip().split(' ')
          #jump the head the end and space line
          if currentline[0] == "#!MLF!#" or len(currentline)==0 or
    currentline[0] == '.' :
            continue
          # read the head, and inital the sub-dictionary
          elif len(currentline)==1:
            head='\"*'+currentline[0][-55:]
            _dict[head]={'word':[],'score':[],'start':[],'end':[]}
          # read the entries information
          else:
            _dict[head]['start'].append(currentline[0])
            _dict[head]['end'].append(currentline[1])
            _dict[head]['word'].append(currentline[2])
            _dict[head]['score'].append(currentline[3])
  except IOError as ioerr:
      print "File %s does not exist" % (filepath)
  return _dict

#cost evaluation
def costfunction(entry1,entry2,ind1,ind2):
    first=entry1['word'][ind1]
    second=entry2['word'][ind2]
    if len(first) > len(second):
      first,second = second,first

    first_length = len(first) + 1
    second_length = len(second) + 1
    distance_matrix = [range(second_length) for x in range(first_length
    )]
    #print distance_matrix
    for i in range(1,first_length):
      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  """
76  ROVER combination usi7g Dynamic programming
77
78  Data structure:
79  ref={'filename1':{'word':[],'start':[],'end':[],'score':[]},'filename2
        ':{....}}
80  record=[('f1 or f2 or X',index)]
81  penalty=[[s11,s12,...],[s21,s22,...],...]
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 inital 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 trackback"
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

```
#!usr/env/bin python
#-*-UTF-8-*-
'''
Confusion Network Combination

The CNC script is similar to ROVER script. The difference is that CNC use the lattices to perform the combination rather than
MLF files. In this case, the DP process used in CNC is exactly the same as ROVER. The script will first convert the lattice
into MLF file format, and doing the combination and best path selecting.

Usage: CNC.py system_A system_B dataset_list --pass1 --pass2
-system_A: the lattice A
-system_B: the lattice B
-dataset_list: the list of dataset using in this case
-pass1 2: the pass directory of the mlf1,2 files


Author: Junjie Pan
Latest Modified: 2016/04/26
'''
import os,sys
import gzip
import re
import numpy as np
#gloabl setting
delpenalty=3 #deletion and insertion penalty
altpenalty=5 #substitution penalty

# selecting the entries with highes scores in each turn
def bestpath(inputdir,outputdir):
  _dict=load_dict_from_file(inputdir)
  for head in _dict.keys():
    for i in range(len(_dict[head]['word'])):
      w=filter(lambda x: len(x) > 0,re.split(r"_<ALTSTART>_|_<ALT>_|_<ALTEND>",_dict[head]['word'][i]))
      s=list(float(s) for s in str(_dict[head]['score'][i]).split('_'))
      for j in range(len(s)):
        if s[j]==max(s):
          ind=j
      _dict[head]['word'][i]=w[ind]
```

```
39          _dict[head]['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 inital 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 inital 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
```

```
    record[0][0] = ('Begin', '')
    for i in range(1,len(f1['word'])+1):
      penalty[i][0] = i*delpenalty
      record[i][0] = ('f1', i-1)
    for j in range(1,len(f2['word'])+1):
      penalty[0][j] = j*delpenalty
      record[0][j] = ('f2', j-1)

    #2. Matrix Fill (scoring)
    for i in range(1,len(f1['word'])+1):
      for j in range(1,len(f2['word'])+1):
        # mlf_A match mlf_B
        if f1['word'][i-1] == f2['word'][j-1]:
          f1p=(('f1m', i-1, j-1), penalty[i-1][j])
        # mlf_A match !NULL in mlf_B
        else:
          f1p=(('f1', i-1), penalty[i-1][j]+\
            delpenalty)
        # !NULL in mlf_A match mlf_B
        f2p=(('f2', j-1), penalty[i][j-1]+\
          delpenalty)
        # mlf_A subsitute mlf_B
        f12p=(('X', i-1, j-1), penalty[i-1][j-1]+\
          costfunction(f1,f2,i-1,j-1))
        # choose moving step by selecting the minimum penalty
        moving=min(f1p,f2p,f12p,key=lambda x: x[1])
        #store the penalty and record information to corresponding
    matrix
        penalty[i][j]=moving[1]
        record[i][j]=moving[0]
    #3. Trackback (alignment)
    path = []
    i = len(record)-1
    j = len(record[0])-1
    while record[i][j][0] != 'Begin':
      if record[i][j][0] == 'f1':
        path.append((record[i][j][1],'!NULL'))
        i-=1
      elif record[i][j][0] == 'f2':
        path.append(('!NULL',record[i][j][1]))
        j -= 1
      elif record[i][j][0] == 'X':
        path.append((record[i][j][1],record[i][j][2]))
        i -= 1
        j -= 1
      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 trackback"
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
```

```
def readlist(filepath):
  try:
    with open (filepath,'r') as fs:
      store=[]
      for line in fs:
        filename=line.strip().split()
        if len(filename) == 1:
          store.append(filename[0])
    return store
  except IOError as ioerr:
    print "error happens when writing %s" %filepath

'''
main function: to implement the CNC
c1: system 1 eg: plp-bg
c2: system 2 eg: grph-plp-bg
dataset: the file store the dataset list eg:challenge_dev
challenge: optional (challenge or .) To determine whether it is the
    dev03/eval03 task or challenge datset task
        Default value - challenge dataset

The output is in ./Major/challenge/cnc_mlf directory, where challenge
    is determined by the last parameter
'''
def main(c1,c2, decodetype,dataset,pass1, pass2):
  files=readlist(dataset)
  comb='%s+%s_%s'%(c1,c2, decodetype)
  for f in files:
    #read lattices A
    _dict1 = read_lattice(r'/home/jp697/Major/%s/%s/%s/%s/lattices/'\
      %(pass1,c1,f,decodetype))
    #read lattices B
    _dict2 = read_lattice(r'/home/jp697/Major/%s/%s/%s/%s/lattices/'\
      %(pass2,c2,f,decodetype))
    # save lattice A into mlf file
    save_dict_to_file(_dict1,r'/home/jp697/Major/%s/%s/%s/%s/
    rescore_lattices.mlf'\
      %(pass1,c1,f,decodetype))
    # save lattice B into mlf file
    save_dict_to_file(_dict2,r'/home/jp697/Major/%s/%s/%s/%s/
    rescore_lattices.mlf'\
      %(pass2,c2,f,decodetype))
    bestpath(r'/home/jp697/Major/%s/%s/%s/%s/rescore_lattices.mlf'%(
    pass1,c1,f, decodetype),\
      r'/home/jp697/Major/%s/%s/%s/%s/rescore_lattices_rover.mlf'%(
    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

```
# '''
# Interpolate the LMs
# Process:
#   a. Convert 1-best output to data file
#   b. Generate stream files from the data file in step a
#   c. Interpolate the Language Model
# Output:
#   a. ./challenge/store_dat/YTBEdev.dat - data file generated from 1 best hypothesis
#   b. ./challenge/streams - streams generated from LM1~5
#   c. ./exp/temp_file/weight_challenge_dev  - Estimation of interpolation weights
#      ./lm_int_challenge - Interpolation LM
# '''
store='/home/jp697/Major/exp/temp_file'
challengepath="/home/jp697/Major/challenge/streams/"
storedat="/home/jp697/Major/challenge/store_dat/"

#step a
python /home/jp697/Major/exp/shell/ConvertData.py \
/hone/jp697/Major/exp/temp_file/temp1 \
challenge/lm_int_plp /home/jp697/Major/challenge/store_dat/YTBEdev.dat

#step b
echo "Begin to generate the stream files"
for ((j=1;j<=5;j++))
do
  base/bin/LPlex -C lib/cfgs/hlm.cfg -s stream${j} -u -t \
  lms/lm${j} ${storedat}/dev03.dat
  cp stream${j} ${challengepath}
  rm stream${j}
done

echo "begin to interpolate the weights"
# step c
calculate the weights
python "/home/jp697/Major/exp/shell/interpolation.py" /home/jp697/Major/challenge/streams /home/jp697/Major/exp/temp_file/weight_challenge_dev
#Read the weights and interpolation
```

```
echo "begin to merge the LMs"
j=0
weight=[]
while read line
do
  weight[j]=$line
  let j=$j+1
  echo $j
done < ${store}/weight_challenge_dev
echo "Start the weight_challenge_dev LM merge"
echo ${weight[0]}
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_challenge
```

A.2.2 show-specific.sh

show-specific.sh

```
mainpath='/home/jp697/Major'
devpath='/home/jp697/Major/lattices'
store='/home/jp697/Major/exp/temp_file'
task1path="${mainpath}/exp/task1"

echo "start to create the 1best hypothesis"
echo \
'''\
step 1: use the dev03 generated LM to rescore each show in the eval
    sets\
'''
while read line
do
  ${mainpath}/scripts/lmrescore.sh $line lattices decode lm_int plp-
    tglm_int TRUE
done < "$store/eval03"

#checking the completion of step 1
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"
passdir="rescore" #"decode" #"decode_cn" #"adapt"
```

```
outdir='rescore.mlf' #rescore.mlf #LOG.align
while read line
do
  while(true)
    do
      test -e "./${tgdir}/$line/${passdir}/${outdir}" && break
    done
  echo "$line finished!"
done < $store/eval03


echo \
'''\
step 2: generate the 1best hypothesis from the eval lattices\
'''

while read line
do
  ./scripts/1bestlats.sh $line ./plp-tglm_int rescore plp-tglm_int
  echo "complete one task"
done < "$store/eval03"

#checking the completion of step 2
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"
passdir="1best/LM12.0_IN-10.0" #"decode" #"decode_cn" #"adapt"
outdir='rescore.mlf' #rescore.mlf #LOG.align
while read line
do
  while(true)
    do
      test -e "./${tgdir}/$line/${passdir}/${outdir}" && break
    done
  echo "$line finished!"
done < $store/eval03


echo \
'''\
step 3: convert the 1best hypothesis into suitable format\
'''
python ./exp/shell/ConvertData.py -$store/eval03 plp-tglm_int ./exp/temp_file/eval03.dat

echo \
```

```
'''
step 4: generate streams from the converted data file
'''
for ((i=1;i<=5;i++))
do
  echo | base/bin/LPlex -C lib/cfgs/hlm.cfg -s stream_eval${i} -u -t
    lms/lm${i} ./exp/temp_file/eval03.dat \
  >> exp/5.4.6.LOG
  cp stream_eval${i} ${task1path}
  rm stream_eval${i}
done

echo \
'''
step 5: compute the weights from the streams generated from last step
'''
python "/home/jp697/Major/exp/shell/Interpolation.py" ${task1path} ${
    store}/weight_eval


echo \
'''
step 6: generate the interpolation LM according to the weights
'''
weight=[]
while read line
do
  weight[j]=$line
  let j=$j+1
  echo $j
done < ${store}/weight_eval

echo "Start the evalLM merge"
echo ${weight[0]}
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


echo \
'''
step 7: use the new_interpolated LM to rescore the eval data_set
'''
while read line
do
```

```
  ${mainpath}/scripts/lmrescore.sh -OUTPASS specific $line lattices
    decode lm_int_specific plp-tglm_int TRUE
done < "$store/eval03"

#checking the completion of step 7
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"
passdir="specific" #"decode" #"decode_cn" #"adapt"
outdir='rescore.mlf' #rescore.mlf #LOG.align
while read line
do
  while(true)
    do
      test -e "./${tgdir}/$line/${passdir}/${outdir}" && break
    done
  echo "$line finished!"
done < $store/eval03

echo \
'''
WER and perplexity for eval03 dataset from LM1-5, lm_int, and
    lm_int_specific
'''
echo 'scoring all the files'
for ((i=1;i<=5;i++))
do
  echo "LM${i}" >> ./exp/5.4.6_LM1-5_eval
  echo | ./scripts/score.sh plp-tglm-eval${i} eval03 rescore >> ./exp
    /5.4.6_LM1-5_eval
  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
done
echo "int_LM" >> ./exp/5.4.6_LM1-5_eval
echo | ./scripts/score.sh plp-tglm_int/eval03 eval03 rescore >> ./exp
    /5.4.6_LM1-5_eval
echo | base/bin/LPlex -C lib/cfgs/hlm.cfg -u -t lm_int lib/texts/eval03
    .dat >> ./exp/5.4.6_LM1-5_eval

echo "int_specified_lm" >> ./exp/5.4.6_LM1-5_eval
echo | ./scripts/score.sh plp-tglm_int/eval03 eval03 specific >> ./exp
    /5.4.6_LM1-5_eval
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
```