UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and, Machine Intelligence 2018-2019

Module Coursework Feedback

Module Title: Speech and Language Processing
Module Code: MLMI5
Candidate Number: F606F
Coursework Number: 1
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: Click here to enter a date. Marker's Name(s): Click here to enter text.
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)	**Outstanding**	**A+**	**A**	**A-**	**B+**	**B**	**C+**	**C**	**Unsatisfactory**
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, or part 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.

MLMI5: Speech and Language Processing
Practical Report: Keyword Spotting

1 Experiments: 1-Best Keyword Spotting

1.1 Basic Keyword Spotting System Design

1.1.1 Index Software

The index software is designed to convert the CTM file into an index file. It firstly reads in the CTM file, groups words (which belong to the same document and obey the 0.5 second rule) and returns as a sentence. Then it collects the time-stamp and document information of the sentence, and return these with the sentence to the file as an index.
For instance, the index software reads the following lines from `Practical\lib\ctm\reference.ctm`,

```
BABEL_OP2_202_10524_20131009_200043_inLine 1 6.30 0.340 halo 1.0000
BABEL_OP2_202_10524_20131009_200043_inLine 1 6.64 0.370 habari 1.0000
BABEL_OP2_202_10524_20131009_200043_inLine 1 7.01 0.690 ya 1.0000
```

and rewrite into the format of,

```
BABEL_OP2_202_10524_20131009_200043_inLine halo|habari|ya 6.30,6.64,7.01 0.340,0
.370,0.690
```

This forms an index of the sentence which is able to handle both words and phrases.
After the index file is generated, phrase can be searched efficiently in the sentence since the repetitive time difference calculation for 0.5 seconds rule can be ignored. The document and time-stamp information for a particular phrase can also be extracted efficiently with indexing.

1.1.2 Hit detection Software

The hit detection software is designed to search matches for query terms in the index file, according to the query file. These matches are named as hits in which each hit is defined as the exact match between query term and keywords regardless of letter case.
For example, it is searching for the following query from `Practical\lib\kws\queries.xml` in `reference.ctm`,

```
<kw kwid="KW202-00010">
  <kwtext>utoke</kwtext>
</kw>
```

It find the following matches,

```
BABEL_OP2_202_24239_20140206_191516_outLine kutokea 19.27 0.940
BABEL_OP2_202_69964_20131012_170534_inLine ndiyo|tuweze|kuona|kama|utapitia|huku
|twende|tutoke|kutoka|huku ...
BABEL_OP2_202_16249_20131202_232723_outLine unataka|pesa|u-|utoke|wapi|na|pesa|n
a-|na|hufanyi|kazi ...
```

Only the third match generates the hit since the string "utoke" matches exactly, but other matches ("kutokea" and "tutoke") only contain the query as the substring.
Also, the hit detection is regardless of letter case, so the "Utoke" is still counted as a hit of "utoke".

Once a hit is detected, the system automatically stores its timp-stamp and document information. If the query is a word, the time-stamp information (the start-time and the duration) can be extracted directly. If the query is a phrase, its start-time is the start-time of the first word, and its duration is calculated as follows.
Assume the phrase P is a list of word $\{w_i\}_{i=1}^{N}$, the duration is,

$$dur(P) = tbeg(w_N) + dur(w_N) - tbeg(w_1)$$

, where *dur* the duration and *tbeg* the start time.

Once all the hits for a particular query are detected and their information are stored, the software writes those hits information in the xml file.
For example, one hit is detected for the query term "utoke" with kwid="KW202-00010", the hit information is listed as follows,

```
(doc, query, tbeg, dur)
=(BABEL_OP2_202_16249_20131202_232723_outLine, utoke, 301.49, 0.31)
```

Then, the software writes the hit for the query as,

```
<detected_kwlist kwid="KW202-00010" oov_count="0" search_time="0.0">
<kw file="BABEL_OP2_202_16249_20131202_232723_outLine" channel="1" tbeg="301.49"
 dur="0.31" score="1.0" decision="YES"/>
</detected_kwlist>
```

For simplicity, in this basic KWS system, `score` and `channel` are set to 1, `oov_count` and `search_time` are set to 0, and `decision` is set to "YES".

1.1.3 Performance Evaluation

Based on the hit file generated, the KWS system performance is evaluated by the commands `score.sh` and `term_select.sh` which are used to extract the term-weighted value (TWV) of the system.
The TWV is computed by first computing the miss and false alarm probabilities for each query separately, then using these and an prior probability to compute query-specific values, and finally averaging these query-specific values over all queries q to produce an overall system value (Mamou et al. [2007]):

$$TWV(\theta) = 1 - average_q\{P_{miss}(q,\theta) + \beta \times P_{FA}(q,\theta)\}$$

, where $\beta = 999.9$, P_{miss} is the miss probability and P_{FA} is the false alarm probability for a given query q with threshold θ.

$$P_{miss}(q,\theta) = 1 - \frac{N_{correct}(q,\theta)}{N_{true}(q)}$$

$$P_{FA}(q,\theta) = \frac{N_{spurious}(q,\theta)}{N_{NT}(q)}$$

, where:

$N_{correct}(q,\theta)$ is the number of correct detections (retrieved by the system) of the query q with a score greater than or equal to θ.

$N_{spurious}(q,\theta)$ is the number of spurious detections of the query q with a score greater than or equal to θ.

$N_{true}(q)$ is the number of true occurences of the query q in the corpus.

$N_{NT}(q)$ is the number of opportunities for incorrect detection of the query q in the corpus; it is the "Non-Target" query trials. It has been defined by the following formula: $N_{NT}(q) = T_{speech} - N_{true}(q)$. T_{speech} is the total amount of speech in the collection (in seconds).
This quantity gives a description of how the system performs as the system with better performance always has a higher TWV.

The scoring system also generates the `res.txt` file which reveal more details of the performance about the system, such as ATWV, MTWV, number of missing words, number of false alarm, etc. ATWV is the "actual term-weighted value"; it is the detection value attained by the system as a result of the system output and the binary decision output for each putative occurrence. It ranges from $-\infty$ to $+1$. MTWV is the "maximum term-weighted value" over the range of all possible values of θ. It ranges from 0 to $+1$. The detection error tradeoff (DET) curve of miss probability (P_{miss}) against false alarm probability (P_{FA}) is also generated and provided after scoring as the image of `det.png`.

1.1.4 Experiments

To examine the performance of the system, an experiment is conducted. An index file is generated from `reference.ctm` via the index software. According to the queries listed in `queries.xml`, the hits xml file is generated from searching in the index file via the hits detection software.
The output is scored, and the performance is extracted. It yields,

```
all TWV=1 theshold=1.000 number=488
```

This shows the basic KWS system is running properly.

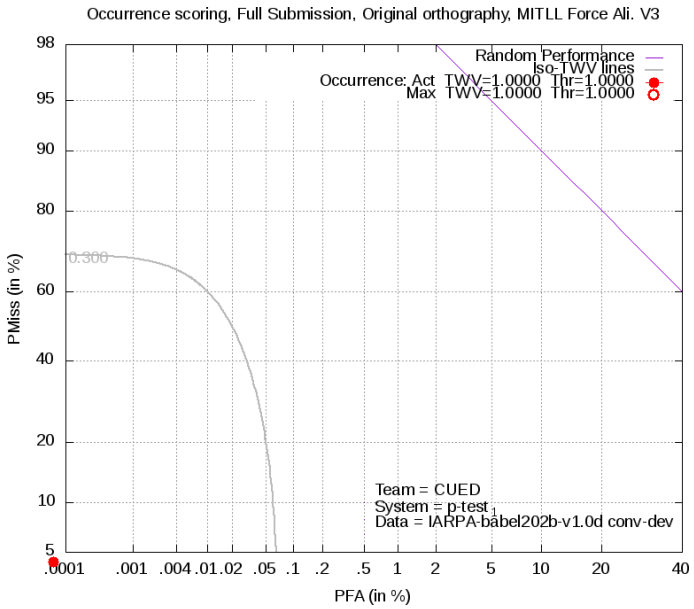

Figure 1: DET curve for the reference.ctm

1.2 Confidence Scores Combination

1.2.1 System Modification

The basic system from Section 1.1 is modified to handle the confidence scores.
For the index software, it can now group the score information based on the rules mentioned in Section 1.1.1. For instance, it reads the following from `Practical\lib\ctm\decode.ctm`,

```
BABEL_OP2_202_10524_20131009_200043_inLine 1 6.32 0.34 halo 0.975370
BABEL_OP2_202_10524_20131009_200043_inLine 1 6.66 0.37 habari 0.975930
BABEL_OP2_202_10524_20131009_200043_inLine 1 7.03 0.63 ya 0.960372
```

and rewrites in the format of,

```
BABEL_OP2_202_10524_20131009_200043_inLine halo|habari|ya 6.32,6.66,7.03 0.34,0.37,0.63 0.975370,0.975930,0.960372
```

For the hits detection software, once a hit is detected, the score information is also stored. If the query is a word, its score is the word's score. If the query is a phrase, the score is calculated as follows,
Assume the phrase P is a list of word $\{w_i\}_{i=1}^{N}$, the score of the phrase can be computed via the product of all scores of the words,

$$s(P) = \prod_{i=1}^{N} s(w_i)$$

or via the summation of all scores of the words,

$$s(P) = \sum_{i=1}^{N} s(w_i)$$

, where s is the score.

1.2.2 Sum-to-one normalization

A standard normalization method – sum-to-one (STO) normalization is implemented in the hit detection software. Given a query w with scores $s_{w,1}, \ldots, s_{w,n}$, where n is the n-th hits of the query, the normalized score is given by,

$$\frac{s_{q,i}}{\sum_{j=1}^{N_q} s_{q,j}}$$

1.2.3 Experiments

Some experiments are conducted to examine the performance of different score combination methods with and without normalization.
These scores combination methods without normalization are firstly implemented by the system. And the performance results are listed in Table 1.

TWV	**all**	**iv**	**oov**	threshold
sum	0.3197	0.4021	0.0000	0.1473
product	0.3189	0.4011	0.0000	0.0425

Table 1: TWVs and thresholds for different score combinations

From Table 1, the summation method gives a greater TWV than the product method. But the difference is not significant around 0.0008 only. The out-of-vocabulary TWV is always 0 because the hit detection is always 0 for `oov` terms.
If the score is combined through the product method, the score tends to underflow, which leads to the low decision threshold.
These scores combination methods with normalization are also implemented by the system. And the performance results are given in Table 2.

TWV	**all**	**iv**	**oov**	**threshold**
sum	0.3197	0.4021	0.0000	0.0128
product	0.3196	0.4019	0.0000	0.0074

Table 2: TWVs and thresholds for different score combination methods with STO normalization

From Table 2, the TWV difference between sum and product methods becomes smaller around 0.0001 after normalization. The TWV increases in general but the summation method is still better than the product method.
The summation method still has a higher threshold than the product one, and all the decision thresholds decrease after normalization.
Thus, the summation method is chosen to combine the confidence scores to avoid score underflow. Since

other normalization methods have not been tested, which score normalization method should be used is not decided yet.

1.3 Morphological Decomposition

1.3.1 Software Design

This software is designed to perform morphological decomposition to any word or phrase, according to the rules listed in the `morph.kwslist.dct` and `morph.dct`. It can be used to decompose any term from the query file `queries.xml` and the index file into the morphological form.
The software is set to divide time-stamp information for a word evenly to all the morphological components. Therefore, given that a word w is a list of morphemes $\{m_i\}_{i=1}^{N}$, where N is the total number of morphemes belong to the word and i is the index of the morpheme in the word. The average duration for each morpheme is given by,

$$dur(m_i) = \frac{dur(w)}{N}$$

, where *dur* is the duration.
The start-time for each morpheme is given by,

$$tbeg(m_i) = tbeg(w) + dur(m_i) \times (i-1)$$

, where *tbeg* the start-time.
The system also assumes that the morphological components of an word share the same score with the original word.
The score s of each morpheme is given by,

$$s(m_i) = s(w)$$

The score can also be splitted by $s(m_i) = \frac{s(w)}{N}$ or $s(w)^{\frac{1}{N}}$, but the division causes the score underflow and the root may cause the score overflow, so they are not applied. Therefore, the following line from the original index file,

```
BABEL_OP2_202_10524_20131009_200043_inLine halo 3.81 0.25 0.974210
```

can be rewritten as,

```
BABEL_OP2_202_10524_20131009_200043_inLine ha|lo 3.81,3.935 0.125,0.125
0.974210,0.974210
```

With the new query and index files, it allows the hit detection software to generate hits for the morphologically decomposed system.

1.3.2 Experiments

The first experiment has been conducted to examine how the morphological decomposition influences the performance. The procedures are outlined as follows, the morphological decomposition software is firstly run on the word-system index file from Section 1.2 and the query file `queries.xml`. Based on the new index and queries files, the new hit file is generated via the hit detection software. Score is normalized with STO and performance evaluation is performed, which yields the following,

TWV	**all**	**iv**	**oov**	**#FA**	**#Miss**
WS	0.3197	0.4021	0.0000	320	558
MDWS	0.3232	0.4018	0.0180	602	543

Table 3: Performances of the word system (WS) and the morphologically decomposed word system (MDWS)

In comparison with the WS, the MDWS improves the all and oov TWVs, but decreases the iv TWV. It can be explained as follows, assume that there is a number of distinct words $W = \{w_i\}_{i=1}^{N_w}$ sharing the

same morphological sequence $M = \{m_i\}_{j=1}^{N_m}$, if $w_k \in W$ is the correct hit of $w_l \in W$, it can reduce the miss probability and increase the oov TWV. Otherwise, it increases the false alarm probability and decreases the iv TWV. These can be shown by the increase of $\#FA$ (number of false alarms) and the decrease of $\#Miss$ (number of missing words) in the MDWS when compared with the WS.

To compare the performance difference between the morphological system (MS) and the MDWS, the performance of the morph system decoding output `decode-morph.ctm` is extracted and listed in Table 4.

TWV	**all**	**iv**	**oov**	**#FA**	**#Miss**
MS	0.3271	0.3939	0.0678	541	553

Table 4: Performance of the MS

When comparing the MDWS with the MS, the MS has better all and oov TWVs, but worse iv TWV. The TWVs differences are expected since the MDWS is set to assume that the score of each morpheme is equal to the score of the word it belongs to, but the MS does not have that assumption.
So the MDWS has a different threshold which affects which words are classified as miss or false alarm, and affects the performance.

1.4 Score Normalization

1.4.1 Sum-to-one normalization

Some researchers further extend the STO method from Section 1.2.2 by adding the tuning parameter γ. Given a query w with scores $s_{w,1}, \ldots, s_{w,n}$, where n is the n-th hits of the query, the normalized score is given by,

$$\frac{s_{q,i}^{\gamma}}{\sum_{j=1}^{N_q} s_{q,j}^{\gamma}}$$

1.4.2 Query-length normalization

The QL normalization is defined as follows (Jonathan Mamou [2013]),

$$s_{q,i}^{\frac{1}{\Delta_{avg(q)}}}$$

, where $\Delta_{avg}(q)$ is the average duration of all returned hits for the query q.
It is inspired by the idea that hits with a longer duration are more likely to be correct, so it is expected to reduce the false alarm rate.

1.4.3 Keyword Specific Thresholds

The estimated Keyword Specific Threshold (KST) is given by (Wang and Metze [2014]),

$$thr(w) = \frac{\beta \cdot \alpha \cdot S(w)}{T + (\beta - 1) \cdot \alpha \cdot S(w)}$$

, where α is the boosting factor $= 1.5$, $S(w)$ the posterior sum of the keyword w, the constants $\beta = 999.9$ and $T = 36000$.
When the confidence of the detected keyword is below KST (Damianos Karakos [2013]), it is rejected and is not regarded as a hit. Since the word with lower confidence score is more likely to be a false alarm, KST can be used to reduce the false alarm rate.

1.4.4 Experiments

Some experiments are conducted to investigate how the tuning parameter affects the STO-normalized system (`decode.ctm`) performance. The results are listed as follows,

γ	**all**	**iv**	**#FA**	**#Miss**	**threshold**
without STO	0.3197	0.4021	320	558	0.1473
0	0.3191	0.4013	320	558	0.0400
0.5	0.3195	0.4018	320	558	0.0233
1	0.3197	0.4021	320	558	0.0128
2	0.3203	0.4029	320	558	0.0033
3	0.3208	0.4035	320	558	0.0007
4	0.3204	0.4030	320	558	0.0001

Table 5: Performances for STO-normalized systems with different tuning parameters

The threshold decreases with the increase of tuning parameter and the introduction of STO normalization. The performance seems to increase with the tuning parameters until $\gamma = 3$, but the performance improvement is not significant.
Some experiments on the score normalization methods have been conducted. The followings are the results,

TWV	**all**	**iv**	**#FA**	**#Miss**	**threshold**
QL	0.3181	0.4001	320	558	0.0029
KST	0.3200	0.4025	303	561	0.1473
KSTSTO ($\gamma = 1$)	0.4023	0.0000	303	561	0.0273
KSTQL	0.3186	0.4007	303	561	0.0029

Table 6: Performances for systems with different score normalization

The STO normalization improves the all TWV. But the QL normalization decreases the all TWV, and unexpectedly has no effect on numbers of false alarms and miss words. The KST greatly improves the performance, in comparison to other methods, by greatly reducing the number of false alarms but slightly increasing the number of miss. However, if the system is further normalized, the TWV value decreases conversely. But the system with KST followed by STO is still better than the one with only STO, and the system with KST followed by QL is better than that with only QL.

1.5 Grapheme-Confusion Matrix

1.5.1 Software Design

The software is designed to convert the out-of-vocabulary into the keyword from the ctm file with the highest probability computed according to `grapheme.map`. The construction of the software is based on the assumption that only one edit option can be made on any query, which is due to fact that it is rare for an ASR system to output multiple incorrect words continuously which are the oovs, and a human to make multiple continuous mistakes when speaking.
For an oov query, assume it is a word w which is a list of graphemes $\{g_i\}_{i=1}^{N}$, the word-replacement option with the least edit distance (≤ 3) is chosen from the ctm file. If multiple options occur with the same edit distance, the one with the highest probability of replacing the original grapheme set to the replaced grapheme set is chosen.
The probability of replacing one grapheme g_i with another g_j is given by,

$$P(g_j|g_i) = \frac{\#(g_i \longrightarrow g_j)}{\#(g_i \longrightarrow all)}$$

, where $\#(g_i \longrightarrow g_j)$ is the number of transition from g_i to g_j and $\#(g_i \longrightarrow all)$ the total number of transition made by g_i.
However, it is found that some transitions from one grapheme to another are not listed in the map. To back off the probability, some transition probabilities are given by,

$$P(g_j|g_i) \approx \frac{1}{\#(g_i \longrightarrow all)}$$

The negative log likelihood of every correction c_i from string a to b (i.e. $q_{ai} \longrightarrow q_{bi}$) is added to give the edit cost,

$$\mathcal{L}(a \longrightarrow b) = -\sum_{i=1}^{N_c} log(P(c_i)) \leq 20$$

, where the edit cost has a lower bound of 20 which reduces the false alarm rate.
If the query is a phrase $P = \{w_i\}_{i=1}^{N}$, an additional step is necessary to decide which word in the phrase should be changed, the edit priority of each word is ranked by its instability which is defined as follows, Consider the probability of changing a grapheme is given by,

$$P(g_i) = \frac{\#(g_i \longrightarrow all)}{\sum_{j=1}^{N_g} \#(g_j \longrightarrow all)}$$

The word w instability is given by,

$$\rho(w) = -\sum_{i=1}^{N_g} log(P(g_i))$$

The word with the highest instability in the phrase is corrected. Some examples are listed in Table 7.

Original	**Corrected**
what she has gone	what she has one
nipelekwe	nipeleke
wenga fans	wengi fans
head office	head ufike

Table 7: List of corrected words

1.5.2 Experiments

The grapheme-confusion matrix is implemented into the system to handle oov queries based on the word-based output, `decode.ctm`. Some experiments have been conducted to examine how different edit distance options can affect the system performance.

edit distance	**all**	**iv**	**oov**	**#FA**	**#Miss**
1	0.3248	0.4021	0.0249	566	554
2	0.3347	0.4021	0.0732	1025	548
3	0.3341	0.4021	0.0701	1032	548

Table 8: Performance comparison for different edit distances

The implementation of the GCM improves all TWVs. When the edit distance is adjusted to 2, the TWV of the system peaks. The number of FA is greatly increased with the edit distance. The number of miss continues to decrease with the edit distance until the edit distance is set to 3. The approach greatly increases the computational time (cost)from less than 1 minute to 5-10 minutes (depends on the edit distance).
Some experiments are also conducted to test the threshold affects the performance.

threshold	**all**	**iv**	**oov**	**#FA**	**#Miss**
1	0.3347	0.4021	0.0732	1025	548
2	0.3347	0.4021	0.0732	1025	548
5	0.3347	0.4021	0.0732	1025	548
10	0.3347	0.4021	0.0732	1025	548
20	0.3341	0.4021	0.0701	1032	548

Table 9: Performance comparison for different threshold settings

When the threshold is within the range of 1 to 10, the performance remains unchanged. But when it is set to 20, the system becomes worse due to the increase of number of false alarms.
It is suitable for use with lattice-based KWS because it can be easily implemented by WFST. Searching for possible keywords can be implemented by composition between acceptor and transducer and shortest distance operation easily. The threshold to filter out low probability keywords can be implemented by weight pushing.

2 Experiments: System Combination

2.1 Fusion Methods

If multiple sets of possible hits for a query from different systems are available, they can be combined as a meta hit if their time-stamp and document information overlap (Jonathan Mamou [2013]).
The following fusion methods can be used to merge the scores from different systems,

2.1.1 CombSUM

Denote the hit that contributes to the meta-hit H from the i-th system among n systems as h_i and the score of the hit h_i as $s(h_i)$, the score of meta-hit H is given by,

$$s(H) = \sum_{i=1}^{n} s(h_i)$$

2.1.2 CombMNZ

The score of the meta-hit computed by CombMNZ method is given by,

$$s(H) = m_H \times \sum_{i=1}^{n} s(h_i)$$

, where m_H is the number of indices having a non-zero score for the meta-hit H.

2.1.3 WCombSUM

An extension of CombSUM is the linear combination method where each component system i is assigned a weight w_i which is computed by,

$$w_i^{MTWV} = \frac{MTWV_i}{\sum_{j=1}^{n} MTWV_j}$$

, where $MTWV_i$ is the maximum term-weighted value of the i-th system. Therefore, the score of the meta-hit can be computed by,

$$s(H) = \sum_{i=1}^{n} w_i^{MTWV} \cdot s(h_i)$$

2.1.4 WCombMNZ

Similarly, the score computed by CombMNZ can be extended as follows,

$$s(H) = m_H \times \sum_{i=1}^{n} w_i^{MTWV} \cdot s(h_i)$$

2.1.5 Experiments

Experimental studies has been conducted on these fusion methods. The unnormalized outputs of combined systems which are formed by the normalized outputs of word and morphological systems with various fusion methods yield the following performances,

all TWV	**CombSUM**	**CombMNZ**	**WCombSUM**	**WCombMNZ**
MS+WS	0.3302	0.3314	0.3286	0.3302

Table 10: TWVs of the unnormalized combined systems with different fusion methods

The STO-normalized outputs of combined system which are formed by the normalized outputs of word and morphological systems with different fusion methods give the following TWVs,

all TWV	**CombSUM**	**CombMNZ**	**WCombSUM**	**WCombMNZ**
MS+WS	0.3297	0.3316	0.3286	0.3297

Table 11: TWVs of the normalized combined systems with different fusion methods

Among the fusion methods, CombMNZ seems to give a better performance but the performance difference is not significant. The fusion methods with the weights of the systems do not seem to further improve the performance. The normalization also has no significant effect on the performance.
However, the average TWV of the combined system is around 0.330 which is better than those from the WS (0.320) and MS (0.327). This shows the combined system has improved the performance.
Since the performance is independent of the fusion method, wCombSUM method is randomly chosen to combine the latter systems.

2.2 IBM WFST-based KWS software

2.2.1 Experiments

Since the xml files provided (`morph.xml`, `word.xml` and `word-sys2.xml`) are not normalized, several experiments are begun with STO normalization, and those normalized outputs are used to examine how the combinations of the IBM system outputs with wCombSUM can influence the performance.

TWV	**all**	**iv**	**oov**
morph-system	0.5200	0.5594	0.3674
word-system 1	0.4604	0.5790	0.0000
word-system 2	0.4651	0.5850	0.0000
MS+WS1	0.5195	0.5588	0.3668
MS+WS1+WS2	0.5229	0.5639	0.3641

Table 12: TWVs for different systems formed by normalized outputs

When the outputs from those 3 systems are combined, the iv performance of the combined system is better than that of the morph-system, but worse than those from the word systems. This may indicate those word systems help to improve the iv performance.

The oov performance of the combined system is worse than that of the morph-system, but better than those from the word-systems. This may indicate the morph-system helps to improve the oov performance.

Some experiments are also conducted on unnormalized supplied files. The output of the combined system is firstly formed by the unnormalized outputs of the IBM scripts, then it is normalized by STO method. So the normalized combined output can be used to examine when to apply normalization affects the performance.

TWV	all	iv	oov
morph-system	0.3597	0.4296	0.0886
word-system 1	0.3987	0.5014	0.0000
word-system 2	0.4033	0.5073	0.0000
MS+WS1+WS2	0.5110	0.5484	0.3660

Table 13: TWVs for different systems formed by unnormalized outputs

The normalized system formed by unnormalized IBM outputs has worse performance than that formed by normalized outputs.

2.2.2 Performance for different query lengths

To extract information for differnt query lengths using `termselect.sh`, the `ivoov.map` file is modified. Apart from labelling the queries with iv or oov in the 1^{st} column, they are labelled with the length of the query (from 1 to 5). The 2^{nd} column remains the same for representing the query index. The 3^{rd} column now counts the number of queries with the same length. This new map is named `qlen.map`.
To extract the performance for different query lengths, the following command is run,

```
./scripts/termselect.sh qlen.map output/reference.xml scoring {1,2,3,4,5}
```

Based on `qlen.map`, an experiment is conducted to examine the performance for different query lengths.

TWV	1	2	3	4	5
morph-system	0.5257	0.5747	0.2930	-0.0028	0.0000
word-system 1	0.4438	0.5299	0.3416	0.1000	0.0000
word-system 2	0.4498	0.5339	0.3374	0.1000	0.0000
MS+WS1+WS2	0.5290	0.5775	0.2930	-0.0028	0.0000

Table 14: TWVs for differnt query lengths

The TWV performance increases from the query length of 1 to 2, and decreases from 3 to 5. The combined system has not detected any word with query length of 5, but has detected some False positives for word with query length of 4, leading to the negative value.

References

Jonathan Mamou, Bhuvana Ramabhadran, and Olivier Siohan. Vocabulary independent spoken term detection. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 615–622, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-597-7. doi: 10.1145/1277741.1277847. URL http://doi.acm.org/10.1145/1277741.1277847.

Xiaodong Cui Mark J.F. Gales Philip C. Woodland Jonathan Mamou, Jia Cui. System combination and score normalization for spoken term detection. 2013.

Yun Wang and Florian Metze. An in-depth comparison of keyword specific thresholding and sum-to-one score normalization. In *INTERSPEECH*, 2014.

Stavros Tsakalidis Le Zhang Shivesh Ranjan Tim Ng Damianos Karakos, Richard Schwartz. Score normalization and system combination for improved keyword spotting. 2013.