Machine Learning and Machine Intelligence

MLMI14: Spoken Language Processing and Generation

Practical: Keyword Spotting

1 Introduction

One interesting application of speech technology is the detection of a particular word, or phrase, in a stream of audio. Applications in this area can be split into two broad area depending on the nature of the query (keyword/phrase) being searched for:

- *written terms*: the query term is, for example, typed in;
- *spoken terms*: the query term is spoken.

This practical will focus on the first form, written text.

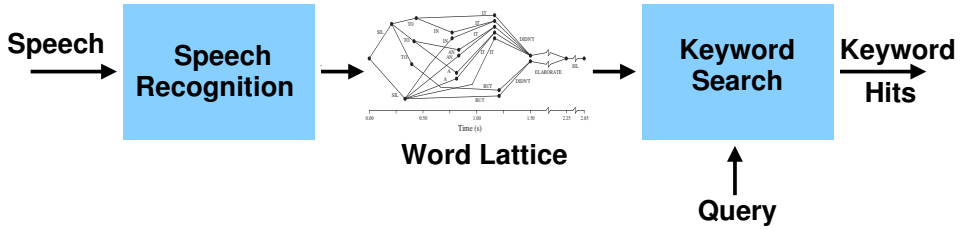

Figure 1: Lattice Based Keyword Spotting System

Most state-of-the-art speech keyword spotting (KWS) systems make use of large vocabulary speech recognition systems to generate lattice which are then used in the KWS process. This process is illustrated in figure 1. There are a number of challenges that need to be addressed, three of which are:

1. *efficiency*: how to make the search process run quickly so that large quantities of audio data can be searched for the query term;
2. *out-of-vocabulary queries*: in many applications the query terms are not known in advance. Thus it is necessary to be able to search for queries are not in the vocabulary of the speech recognition system.
3. *threshold selection*: once a set of possible locations for the query have been found (hits), it is necessary to select a threshold for the hit score, above which the hit is marked as valid. This threshold balances *false rejects* and *false accepts*.

In addition, in common with speech recognition systems, *system combination* can be applied to KWS systems. Here multiple sets of hits from different speech recognition

systems are combined together. All four of these issues will be touched on in this practical.

Having decided on a set of hits for a set of queries in the audio the performance of the system must be assessed. There are a number of approaches for this including *area under the curve* and *term weighted value* (TWV). For this practical the performance of the systems will be assessed by TWV. Rather than having to explicitly set a threshold for KWS, the best possible TWV value (the maximum TWV (MTWV)) score will be used. For details of the scoring requirements see Appendix A.

If you have a question, it may be useful to look at Appendix E, as this answers specific questions (these are taken from previous years' postings to the MLSALT/MLMI forum).

2 Theory

The aim of this practical is to write a simple KWS system, based on the 1-best output from a speech recognition system, rather than the lattice-based approaches described in lectures. Instead of the full lattice being available posterior scores derived from confusion networks are given for the 1-best output.

In addition to the basic KWS scheme based on the 1-best output with posterior scores, score-normalisation and system combination are also investigated. The theory for these two is briefly described in this section.

2.1 Score Normalisation

Rather than using the raw scores derived from the KWS process it is possible to perform score normalisation at a query term level prior to performing scoring. One of the simplest forms of score-normalisation is Sum-to-One (STO) normalisation. Here

$$\hat{s}_{ki} = \frac{s_{ki}^{\gamma}}{\sum_j s_{kj}^{\gamma}}$$

where s_{ki} is the score for the i^{th} hit for keyword k, the summation is over all hits for keyword k, and γ is a tunable parameter. Other forms of normalisation can be used (and trained). If time allows you should investigate *Keyword Specific Thresholds* (KST) [1]. For KST the length of speech data to run KWS over is 10 hours.

2.2 System Combination

If multiple sets of possible hits for a query are available, it is possible to combine these together in the same fashion as system combination for ASR (and many other tasks). As only a single audio stream is being used, so time stamp information is included in the query hit, it is necessary to merge queries from multiple systems. The simplest approach to doing this is to merge scores if the time-stamp information overlaps.

[1] *An In-Depth Comparison of Keyword Specific Thresholding and Sum-to-One Score Normalization* Yun Wang and Florian Metze, Carnegie Mellon University, InterSpeech 2014

3 Practical

The data for this practical is taken from the Babel project. The selected language is Swahili (the 2015 surprise evaluation language). As this is operating on "real" data, sometimes approaches that should work do not, and results are not always consistent. When this is the case (and you are confident it is not a bug) discuss the results and what you would have expected in the write-up.

3.1 Infrastructure

The following files are supplied for this practical. They may be found in the directory `/usr/local/teach/MLSALT5/Practical` On the system.

(a) `lib/ctms` - 1-best decoding output with confidence scores of the tasks

- `reference.ctm`: the reference output (including time-stamps)
- `decode.ctm`: decoding output from a word-based system
- `decode-morph.ctm`: decoding output from a morph-based system

(b) `lib/dicts` - a directory containing the mappings from word to morphs

- `morph.dct`: the mapping for all the words in the decoding vocabulary
- `morph.kwslist.dct`: the mapping for all the query terms

(c) `lib/kws` - the set of KWS queries and associated output from KWS on lattices from the CUED evaluation system output). Including:

- `queries.xml`: set of KWS queries - this gives the word and the id (required for scoring)
- `morph.xml`: generated from the morphologically decomposed system
- `word.xml`: generated from the word-based system
- `word-sys2.xml`: generated from a semi-supervised word-based system
- `grapheme.map`: grapheme confusions for the evaluation system

(d) `lib/terms` - contains a file giving the format for extracting a subset of queries for scoring (in this case IV/OOV)

- `ivoov.map`: mapping from query terms to whether they are in-vocabulary or out-of-vocabulary

(e) `scoring` - the correct output for scoring the reference ctm file.

(f) `scripts` - contains the scripts for scoring the output from the systems

- `score.sh`: generate the score (MTWV) for the set of queries
- `termselect.sh`: using a term-map generate the MTWV score for a subset

3.2 Experiments: 1-Best Keyword Spotting

The aim of this part of the practical is to write a basic keyword spotting system that enables query words and phrases in a 1-best list to be found. Though only 1-best lists are considered, the code should follow the index/search framework for KWS described in lectures.

1. Write software to convert the reference word CTM file (see Appendix B for format) supplied in

   `lib/ctms/reference.ctm`

   into an index, $\mathcal{I}(w)$, for each word that can be used for KWS. Ensure that the index is able to handle both words and phrases. As only a 1-best list is supplied the WFST form of index is not required. For this task only a single audio stream is available. It is therefore necessary to also include the time instance at which the keyword query starts and the duration.

   Write software to search this index to generate hits (perform KWS search), noting the requirement that the time between words in phrase should be less than or equal to 0.5 seconds, using the queries from the query XML file in

   `lib/kws/queries.xml`

   For details of the XML file format of this query file see Appendix C. This search process should generate a set of *hits*, hypotheses of where the query term occurs in the files. As the KWS is being performed on the reference the score for each of these hits should be 1.00.

   It is important that you get the format correct for the KWS hits - see Appendix D

   An example of (part of) the output required is given in Appendix D. There is also an example in the supplied KWS output from the evaluation systems, for example,

   `lib/kws/word.xml`

   Once the set of hits has been generated, they can be scored. To generate the score run

   ```
   scripts/score.sh output/reference.xml scoring
   ```

   where it is assumed that the output has been placed in the `output/reference.xml` and the scoring is to be generated in the directory `scoring`. This should allow you to debug the process. To extract the performance of the system you can run:

   ```
   ./scripts/termselect.sh  lib/terms/ivoov.map output/reference.xml scoring all
   ```

It is also possible to generate IV and OOV performance respectively using

```
./scripts/termselect.sh  lib/terms/ivoov.map output/reference.xml scoring iv
./scripts/termselect.sh  lib/terms/ivoov.map output/reference.xml scoring oov
```

The scoring requirements, and output for file `word.xml`, are described in Appendix A. For additional information two files are available describing detailed output for scoring, and the associated DET curve (not meaningful for the reference)

```
scoring/reference-res.txt
scoring/reference-det.png
```

To help debug the process the file in

```
scoring/reference/Full-Occur-MITLLFA3-AppenWordSeg.bsum.txt
```

contains the breakdown for every query. All should be correct if your system is working correctly.

2. Having established that your KWS system is correct, run the KWS on the word decoding hypothesis in:

   ```
   lib/ctms/decode.ctm
   ```

   Note you will need to consider how to combine the confidence scores when performing KWS with phrase queries. Comment on the performance of your system.

3. As an alternative to the word-based 1-best decoding the output from a morphological decomposition can be used. Write a script that applies the morphological decomposition in

   ```
   lib/dicts/morph.kwslist.dct
   lib/dicts/morph.dct
   ```

   to the query terms and the 1-best list respectively (consider the time-stamps and scores). Once this has been completed the KWS can be repeated. Again comment on the system performance.

   You should now perform KWS using the morphological decomposed queries on the morph system decoding output (`decode-morph.ctm`). Are the results different to that from decomposing the word-system output? Is this what you expect?

   As part your discussion you should include comments about how decoding with the morph language model impacts the OOV rates.

4. The KWS so far has used the scores for each hit generated from the KWS system. As discussed in lectures it is possible to apply score normalisation approaches. Implement score normalisation and investigate how it impacts the performance of the system. Comment on the results.

5. As an alternative to using morphological decomposition it is possible to use the word-based decoding output (`decode.ctm`) with a *grapheme-confusion matrix* (a graphemic system was used for speech recognition rather than a phonetic system) to handle OOV words. The file

   ```
   lib/kws/grapheme.map
   ```

   is supplied to allow you to generate this matrix. The format of file is: reference grapheme; what it was recognised as; and the number of times this occurred.

   Using the supplied information design, implement, and evaluate an approach for handling OOV queries based on the word-based output, `decode.ctm`. For the write-up, you should clearly describe the approach that you have adopted, the computational cost and whether it is suitable for use with lattice-based KWS.

3.3 Experiments: System Combination

The results from the word and morph KWS results have been treated separately. The aim of this part of the practical is to combine the two sets of hits together to improve performance.

1. Using the KWS output (normalised or otherwise) write a tool that combines the two sets of hits together. Again comment on the performance and the impact of score normalisation. Be careful to consider what form of score merging should be used.

2. In addition to the CTM files, the posting-lists generated for the 2015 surprise language evaluation from three of the systems are given in

   ```
   lib/kws/morph.xml
   lib/kws/word.xml
   lib/kws/word-sys2.xml
   ```

   No normalisation has been applied to the scores, they are those obtained using the IBM WFST-based KWS software supplied to CUED.

   Using the previously developed normalisation and combination systems investigate the performance that can be obtained using these sets of hits (based on lattices). For example you may want to consider:

   - what happens to the IV performance as different sets of hits are combined;
   - what happens to the OOV performance as different sets of hits are combined;
   - is STO normalisation important for combination - when should it be applied;
   - what happens to performance as the length of the query increases (how should this be measured?). To extract the performance for different length queries use the `termselect.sh` script and modify the ivoov.map file to extract lengths instead of iv/oov.

Plots of the DET curves are available in the scoring directory (look for the png files).

4 Write-Up

Your report for this practical should include details of the code that you have written as well as the experiments that you have run. Ensure that you do as much analysis as possible when discussing your results. You should also discuss, and motivate why you selected, the approaches you have adopted to make the index search efficient, and whether the approaches you have adopted are scalaeble to very large amounts of audio data.

In total your report should be around 2,000 to 3,000 words long. Note you should ensure that the code that you have written for 1-best output KWS and system combination can be made available if necessary (though the path should not be included in the write-up).

A KWS Scoring

The KWS process for this practical uses a signal unsegmented audio stream. Thus, it is necessary to include time-stamp information for every possible query hit. The accuracy required for the query is 0.5 seconds.

An interesting aspect of the system is when phrases occurs. For a sequence of words to be a valid sequence the start and end times of all the words must be less than or equal to 0.5 seconds. Thus even if a phrase occurs in the word sequence, if the gap between words is greater than 0.5 seconds it is not a valid hit. This is important to generate the correct reference scores.

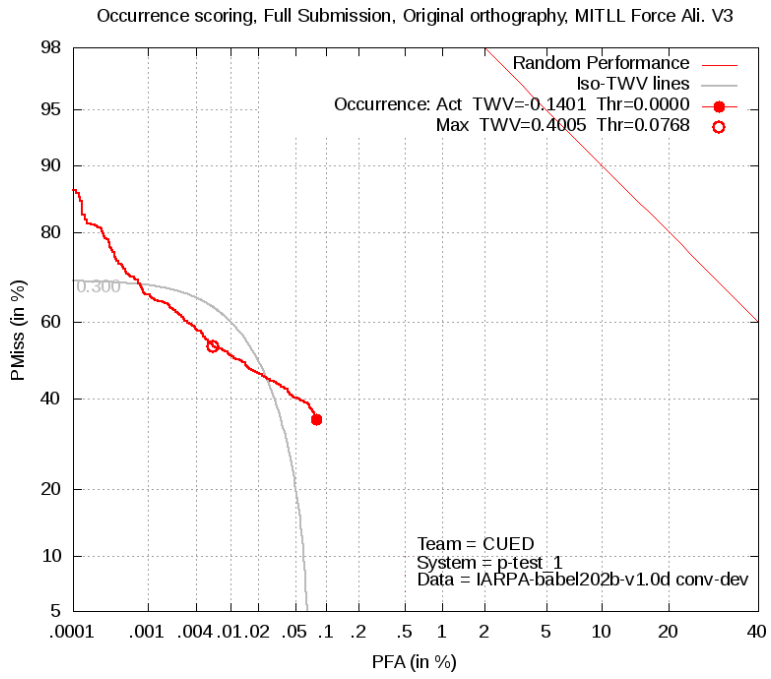

Figure 2: DET Curve for `word.xml`: file `scoring/word-det.eps`

An example of the output is in the directory `scoring`. The DET curve is shown in Figure 2. Running

```
scripts/termselect.sh  lib/terms/ivoov.map lib/kws/word.xml scoring all
```

yields:

```
all TWV=0.398656339047063 theshold=0.077 number=488
```

Note on occasion there can be small rounding errors, between MTWV values obtained using termselect and the DET and result files.

For the detailed results see `scoring/word-res.txt`.

B CTM Input Format

The input to the system for the 1-best KWS process is a CTM file. The format of this file is

```
BABEL_OP2_202_10524_20131009_200043_inLine 1 3.81 0.25       halo 0.974210
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
BABEL_OP2_202_10524_20131009_200043_inLine 1 9.02 0.58     mkubwa 0.487282
BABEL_OP2_202_10524_20131009_200043_inLine 1 13.18 0.58 nazungumza 0.610324
BABEL_OP2_202_10524_20131009_200043_inLine 1 13.76 1.04         na 0.587551
BABEL_OP2_202_10524_20131009_200043_inLine 1 14.80 0.31     rafiki 0.975930
BABEL_OP2_202_10524_20131009_200043_inLine 1 15.11 0.67      yangu 0.975930
BABEL_OP2_202_10524_20131009_200043_inLine 1 15.78 0.86     memory 0.209542
BABEL_OP2_202_10524_20131009_200043_inLine 1 16.64 0.44     kemboi 0.156483
BABEL_OP2_202_10524_20131009_200043_inLine 1 20.15 0.41     salama 0.936255
BABEL_OP2_202_10524_20131009_200043_inLine 1 20.56 0.48       sana 0.969491
BABEL_OP2_202_10524_20131009_200043_inLine 1 21.04 0.50 unaendelea 0.528374
BABEL_OP2_202_10524_20131009_200043_inLine 1 21.54 0.26        aje 0.667691
BABEL_OP2_202_10524_20131009_200043_inLine 1 24.67 0.26      yaani 0.590254
BABEL_OP2_202_10524_20131009_200043_inLine 1 24.93 0.20       niko 0.697072
BABEL_OP2_202_10524_20131009_200043_inLine 1 25.13 0.52      mzima 0.840153
BABEL_OP2_202_10524_20131009_200043_inLine 1 27.03 0.44       kazi 0.923664
BABEL_OP2_202_10524_20131009_200043_inLine 1 28.83 0.32         mi 0.928447
BABEL_OP2_202_10524_20131009_200043_inLine 1 29.15 0.71       niko 0.968491
```

The entries for this file are (and the links to the KWS format):

1. the file-name, relates to `kw file`
2. the channel, relates to `channel`
3. the start-time (seconds), relates to `tbeg`
4. the duration (seconds), relates to `dur`
5. the token, needs to be related to the keyword or phrase
6. the word posterior, relates to the `score`

C KWS Query Format

The `query.xml` file gives the text and KW-id that needs to be searched for in the index. The top few lines of this

```
<kwlist ecf_filename="IARPA-babel202b-v1.0d_conv-dev.ecf.xml" language="swahili" encoding="UTF-8" compareNo
rmalize="lowercase" version="202 IBM and BBN keywords">
  <kw kwid="KW202-00001">
    <kwtext>what she has gone</kwtext>
  </kw>
  <kw kwid="KW202-00002">
    <kwtext>ng'ombe</kwtext>
  </kw>
```

This shows two queries to search for - note you search for the *kwtext*, and generate hits with that text with the *kwid*, see Appendix D

1. kwid: `KW202-00001` kwtext: `what she has gone`
2. kwid: `KW202-00002` kwtext: `ng'ombe`

Thus there are four words in the first query and one word in the second query.

When morphological decomposition is applies, this only changes the *kwtext* - the *kwid* will be the same, allowing the same scoring file to be used whatever form of decomposition has been applied to the queries.

D KWS Output (hits) Format

In order to allow the scoring process to work it is necessary to format the output of the KW spotting process in the correct way. For this task the header must be correctly set-up. The file must start (and end with).

```
<kwslist kwlist_filename="IARPA-babel202b-v1.0d_conv-dev.kwlist.xml" language="swahili" system_id="">
....
</kwslist>
```

Each keyword has a separate set of entries that must start with (here for keyword KW202-0001)

```
<detected_kwlist kwid="KW202-00001" oov_count="0" search_time="0.0">
....
</detected_kwlist>
```

Each time a keyword is detected then a separate line is added of the form

```
<kw file="BABEL_OP2_202_29663_20131208_035816_outLine" channel="1" tbeg="217.41" dur="0.83" score="1.000000" decision="YES"/>
```

This gives the file, start-time, duration and score.

A complete example (for the reference so every score is 1.0)

```
<kwslist kwlist_filename="IARPA-babel202b-v1.0d_conv-dev.kwlist.xml" language="swahili" system_id="">
<detected_kwlist kwid="KW202-00001" oov_count="0" search_time="0.0">
<kw file="BABEL_OP2_202_29663_20131208_035816_outLine" channel="1" tbeg="217.41" dur="0.83" score="1.000000" decision="YES"/>
</detected_kwlist>
<detected_kwlist kwid="KW202-00002" oov_count="0" search_time="0.0">
<kw file="BABEL_OP2_202_39893_20140115_023429_inLine" channel="1" tbeg="347.73" dur="0.28" score="1.000000" decision="YES"/>
<kw file="BABEL_OP2_202_17115_20140218_210921_inLine" channel="1" tbeg="212.62" dur="0.31" score="1.000000" decision="YES"/>
<kw file="BABEL_OP2_202_17115_20140218_210921_inLine" channel="1" tbeg="246.72" dur="0.40" score="1.000000" decision="YES"/>
<kw file="BABEL_OP2_202_25242_20131203_015232_inLine" channel="1" tbeg="141.78" dur="0.34" score="1.000000" decision="YES"/>
<kw file="BABEL_OP2_202_25242_20131203_015232_inLine" channel="1" tbeg="144.38" dur="0.35" score="1.000000" decision="YES"/>
<kw file="BABEL_OP2_202_25242_20131203_015232_outLine" channel="1" tbeg="147.51" dur="0.35" score="1.000000" decision="YES"/>
<kw file="BABEL_OP2_202_25242_20131203_015232_outLine" channel="1" tbeg="342.25" dur="0.63" score="1.000000" decision="YES"/>
</detected_kwlist>
<detected_kwlist kwid="KW202-00003" oov_count="0" search_time="0.0">
<kw file="BABEL_OP2_202_92740_20130923_235638_outLine" channel="1" tbeg="104.61" dur="0.75" score="1.000000" decision="YES"/>
</detected_kwlist>
<detected_kwlist kwid="KW202-00004" Oov_count="0" search_time="0.0">
</detected_kwlist>
<detected_kwlist kwid="KW202-00005" oov_count="0" search_time="0.0">
<kw file="BABEL_OP2_202_77990_20131007_063102_inLine" channel="1" tbeg="492.72" dur="0.47" score="1.000000" decision="YES"/>
</detected_kwlist>
<detected_kwlist kwid="KW202-00006" oov_count="0" search_time="0.0">
<kw file="BABEL_OP2_202_10524_20131009_200043_outLine" channel="1" tbeg="592.56" dur="1.42" score="1.000000" decision="YES"/>
</detected_kwlist>
<detected_kwlist kwid="KW202-00007" oov_count="0" search_time="0.0">
<kw file="BABEL_OP2_202_25242_20131203_015232_outLine" channel="1" tbeg="103.31" dur="1.16" score="1.000000" decision="YES"/>
</detected_kwlist>
</kwslist>
```

Note it is not necessary to set the values of `oov_count` or `search_time`, these can be set to 0 and 0.0 respectively (as above). When no occurrences of the token are found an empty entry, for example `KW202-00004` can be generated.

E FAQ

1. **KWS system format/code?** This is up to you. You must write the KWS system of as indexer followed by search. The input file will be a CTM, the output format an XML file (see handout). Any (correct) implementation of indexer/search is acceptable for the practical. However the indexer is meant to make the search efficient - you do need to consider search efficiency when designing the indexer.

2. **What is the score (word posterior) in the CTM file?** This is an estimate of the probability of the word in the CTM (with those time boundaries) being the correct word.

3. **I don't get morphological decomposition**. Consider the word sequence

   ```
   "they abandoned me"
   ```

   If "abandoned" is OOV then this detected in the ASR output. Imagine that "abandon" and "ed" are in the morphologically decomposed word-list. Now perform morphological decomposition on the query (abandoned) and the word sequence

   ```
   abandoned -> abandon ed
   they abandoned me -> they abandon ed me
   ```

   You now search the decomposed word sequence for the decomposed query - it is IV. You should perform morphological decomposition on the queries and word system output (decode.ctm) as requested in the handout (you need to consider the score/timestamps) and run KWS, as well running KWS using the morphologically decomposed queries on the output of the morph decoding (decode-morph.ctm).

4. **Can you clarify the 0.5 second rule for phrases?** The easiest way to do this is by a couple of examples. For the phrase "naendelea kungoja" - this would be a valid hit

   ```
   BABEL_OP2_202_15420_20140210_010333_inLine 1 169.82 0.470 naendelea 1.0000
   BABEL_OP2_202_15420_20140210_010333_inLine 1 170.29 0.360 kungoja 1.0000
   ```

   The second word directly follows the first. However if the CTM was

   ```
   BABEL_OP2_202_15420_20140210_010333_inLine 1 169.82 0.470 naendelea 1.0000
   BABEL_OP2_202_15420_20140210_010333_inLine 1 171.29 0.360 kungoja 1.0000
   ```

   The gap between the end of the first word (170.29 second) and start of the second word is 1 second. This would not be a valid hit, the gap between the words is greater than 0.5 seconds.

5. **The morphological decomposition for a couple of words is different between the query and decoding information?** This is a minor inconsistency that results from capitalisation. This does not impact performance and can be ignored.

6. **What machines has the scoring been checked on?** The scoring has been checked on:

   - `mlsalt-cpu1`
   - `mlsalt-cpu2`

   If you are having difficulty with the scoring you should check that you are using one of these machines.

7. **Do you have a development set for training an optimal system combination system?** As with many practical applications there is limited manual annotated data for assessing system performance. If you plan to build trainable combination systems, or KWS systems, you can use N-fold cross-validation approaches for example.

8. **How do the system in setions 3.2 and 3.3 relate to each other?** The two systems operate from the exactly the same form of ASR system. Thus the performance that you see from scoring the provided outputs in section 3.3 can be related (and combined with if desired) the systems from section 3.2.