Keyword Spotting

MLMI14 - Spoken Language Processing

March 22, 2022

Candidate no.: J902C

Word count: 2996[1]

UNIVERSITY OF CAMBRIDGE
Department of Engineering

[1] Excluding appendix, tables, footnotes, images and titles

Trumpington Street
Cambridge CB2 1PZ
Tlf.: +44 01223 332600
http://www.eng.cam.ac.uk/

Table of contents

1. Introduction

This project aims to investigate the recognition of words or phrases, known as keyword spotting (KWS), with written queries on languages with a small amount of data, in this case Swahili. As figure 1.1 illustrates, a source of audio is interpreted by a Automatic Speech Recognition (ASR) system, producing a token (word, morph...) lattice, that can be analised by a keyword spotting system, the object of study of this practical.

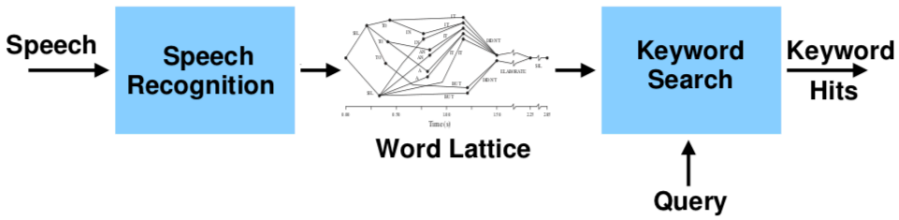


Figure 1.1: Lattice-based Keyword Spotting System

There are several challenges that a good keyword spotting system must address, such as efficiency, managing with out-of-vocabulary (OOV) words because the query terms might be not in the corpus, selecting a threshold to determine a valid hit or even combining different KWS systems to improve their individual performance.

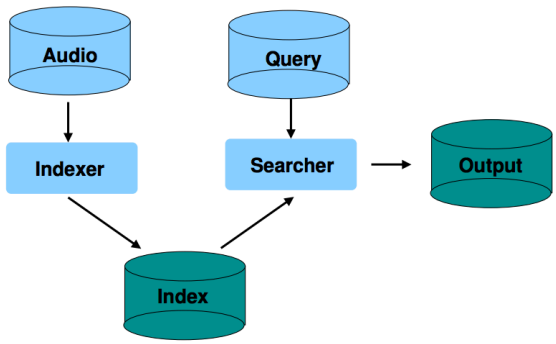
The evaluation metric of systems investigated in this practical is going to be the MTWV (Maximum Term-Weighted Value), which is the maximum of the term weighted value over the range of all possible values of the detection threshold:

$$TWV = 1 - (P_{miss} + \beta P_{FA}), \tag{1.1}$$

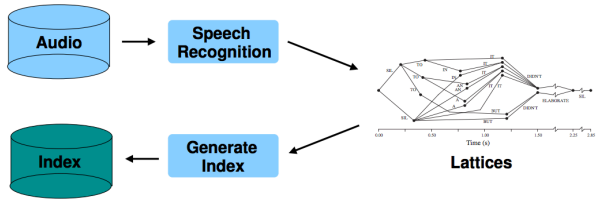
where β is some constant.

2. 1-Best Keyword Spotting

The output of an 1-best ASR system with posterior scores is used as input data for the KWS system. The KWS has to handle the lattice representation to recognise the query terms in the corpus. The general picture of that process is shown in 2.1.



(a) General process of a recognition system



(b) Index generation for KWS system

Figure 2.1: How a KWS system uses an indexer and a searches to match query terms to corpus data

2.1. Building an Index

To efficiently look for queries in the ASR output data, an index is implemented. For this practical the index is designed as a set of two dictionaries (or hashmap): the first dictionary is indexed by token (the token is usually a word, but can be a morpheme or a grapheme later), that points to a list of tuples $(doc\_id, position)$, where $position$ is the position of the indexed word in the document doc_id. The second dictionary is indexed by doc_id and every entry points to a list of vectors, where each vector represents the token information: doc_id, token, channel, beginning time, duration and score.

The computational cost of building the index is linear in the number of tokens in the corpus, since the script to build the index reads every token of every document just once.

2.2. Searching in the Index

To search for a query (word or phrase), the first word is indexed in the first dictionary, getting every document (described by its doc_id) and the position where it appears. Then, the second dictionary is employed for every document containing the first word. If the query is just a word, the searcher returns every appearance (hit) of that word in the corpus; if the query is a phrase, we have to check that the entire phrase is contained in every document after the first word, making sure that there is no more than half a

second between words. If the time condition is satisfied and the entire query phrase is in the searched document, then all the information is returned to describe the hit. To summarise a phrase hit its used the first word, its beginning time, the sum of the duration of the words of the query and the *scores are combined using its product*, since this combining method captures the decaying probability as a function of the query length (a query is more unlikely as its length increases).

This search procedure has a *computational cost* of $O(ml)$, where m is the number of occurrences of the first word of the query in the corpus, and l is the query length, because for each appearance of the first word, the query phrase has to be checked.

2.3. Performing KWS on word-based and phrase-based queries

The correct implementation of the index and the search function can be tested using the reference output file `reference.ctm`, and since the KWS is being performed on the reference the score for each of these hits should be 1.00, i.e., all queries should be correct if the indexer and searcher are working correctly. Table 2.1 shows the results for this experiment, as well as the results of executing the KWS system on the provided 1-best ASR output file `decode.ctm`.

System	All	IV	OOV	Th	P_FA	P_miss
reference.ctm	1.0	1.0	1.0	1.0	0.0	0.0
decode.ctm	0.319	0.401	0.0	0.043	0.00002	0.663

Table 2.1: Results for KWS using the reference output reference.ctm and the ASR output decode.ctm

As expected, the systems performs perfectly (MTWV = 1.0) on the reference file. However, it is not performing so well in a real ASR system output file, `decode.ctm`. The final MTWV for this system is 0.319, and the errors are accumulated mainly because OOV words are not handled, resulting in a OOV TWV of 0.0 and a probability of miss of 66.3%.

In the next sections, the problem of OOV words is going to be assessed.

3. Morphological decomposition

One technique to assess OOV words is to apply morphological decomposition to the queries and to the ASR system output if this is not morpheme-based. A script to decompose query files and KWS input files (.ctm) into morphemes has been implemented. Query terms are decomposed in morphemes and the new query can be thought as a phrase whose tokens/words are morphemes, so the indexer and searcher implementation does not need to be changed. Similarly, for the ASR output files (.ctm) they are parsed in order to have the lattice information for each morpheme of each token. If a word is decomposed into morphemes, each morpheme is considered to last for $1/n$ seconds of the total duration of the word, where n is the number of morphemes of the word. Additionally, as a simplification each morpheme has the same score as the entire word.

The files `morph.dct` and `morph.kwslist.dct` specify a mapping between words and their morphemes of the ASR output files and queries respectively. The are used to transform the word-based ASR output `decode.ctm` and the query file `query.xml`, splitting words into morphemes as explained.

The performance of systems using morphological decomposition is shown in table 3.1, and they are compared with the previous result using only `decode.ctm`.

System	All	IV	OOV	Th	P_FA	P_miss
decode.ctm	0.319	0.401	0.0	0.043	0.00002	0.663
decode.ctm + morph.dct	0.314	0.390	0.018	0.004	0.00003	0.652
decode-morph.ctm	0.318	0.383	0.067	0.061	0.00003	0.651

Table 3.1: Results for KWS after applying morphological decomposition

The overall performance of the KWS is slightly reduced but the results of OOV words are better. When the IV words are decomposed, the number of hits increases and so the number of false alarms, reducing the TWV of IV words. However, the performance of OOV words has increased since several OOV words are decomposed into morphemes that now are IV.

The type of the ASR system is also quite important when executing morphological decomposition. For example, if 'abandon' is queried, the system can find a hit with the segmented ASR output 'abandon' 'ed'. This would be a true positive if the corpus reference is erroneous (the word-based ASR system decoded 'abandoned'), and it would be a false positive if the reference for that word is 'abandoned'.

The system that takes as input the morpheme-based ASR output `decode-morph.ctm` is performing better than `decode.ctm + morph.dct` and `decode.ctm`, since it has reduced the probability of miss, and the results on OOV words are enhanced.

4. Score Normalisation

So far the raw scores were used for each hit generated from the ASR system. Score normalisation (SN) is a technique employed to avoid using these raw scores at query term level prior to performing scoring [1]. In this practical, the Sum-To-One (STO) normalisation is used, that has the expression:

$$\hat{S}_{ki} = \frac{S_{ki}^{\gamma}}{\sum_j S_{kj}^{\gamma}},$$

where S_{ki} is the raw score for the i-th hit for query k, the summation os over all hits for query k, and γ is a tunable parameter that for this project $\gamma = 1$.

System	All	IV	OOV	Th	P_FA	P_miss
decode.ctm + SN	0.320	0.402	0.0	0.020	0.00002	0.663
decode.ctm + morph.dct + SN	0.320	0.397	0.018	0.036	0.00002	0.663
decode-morph.ctm + SN	0.326	0.393	0.068	0.057	0.00001	0.661

Table 4.1: Results for KWS after applying score normalisation

Table 4.1 collects the results for the investigated systems so far using now STO normalisation. It is easy to see that the performances are improved, above all IV words. The performance on OOV is quite similar, but score normalisation does a good job at improving the results at IV level. The probability of missed detection is higher because the raw score of rare keywords tended (before normalising) to be lower than the threshold. Additionally, the number of false alarms is decreased since the normalised score for frequent keywords is lower, resulting in a MTWV upgrade.

The most improved system is the one using the morpheme-based ASR system output `decode-morph.dct`, with a MTWV of 0.326, the best result so far.

There are other techniques of score normalisation, like Keyword Specific Thresholding (KST) [2], that presents a formula for computing a decision (Yes/No) for each keyword detection. This function declares that a detection for keyword w is present if its posterior score (the probability of a keyword being correct) is above the threshold:

$$thr(w) = \frac{\beta \cdot N_{true}(w)}{T + (\beta - 1) \cdot N_{true}(w)}. \tag{4.1}$$

The motivation behind this formula is that scores of keywords with generally low scores (low $thr(w)$) get a boost, if $thr(w)$ is below the global threshold thr. Conversely, keywords with generally higher scores get attenuated, so that, in the end, all keywords have the same global threshold. The non-linearity of the formula is useful in making the scores more distinguishable in the regime of interest.

Although KST and STO approach the need for keyword specific thresholds from different angles, their outcomes turn out similar (section 2.3 of [2]). Because of that, and because for KST the length of speech data to run KWS over is 10 hours, KST is not further investigated in this practical. Other projects have studied KST, STO and Estimated KST [2] normalisation methods, showing that there is almost no difference between STO and KST, but Estimated KST slightly outperforms the other two.

5. Grapheme systems

In section 3 one technique to handle OOV query terms was discussed. In this section, other method is investigated: mapping OOV terms into IV terms (*proxy keywords*) using a *grapheme confusion matrix*. The probability of one IV word given a recognised OOV word can be estimated using the grapheme confusion matrix, where the data counts how many times a reference grapheme was recognised as other one.

$$P(IV_{term}|OOV_{term}) = P(predicted_{term}|reference_{term}).$$

For example, consider the query word 'abandon', that is OOV, and the reference IV word 'abandoned'. Then the probability of 'abandon' being recognised as 'abandoned' can be computed by:

$$P(abandoned|abandon) = P(a|a)P(b|b)P(a|a)P(n|n)\dots P(n|n)P(e|'')P(d|'').$$

We are going to call the probability $P(IV_{term}|OOV_{term})$ the *confusion probability* of word IV_{term} given OOV_{term}. In the new grapheme-based model, when an OOV word is found, instead of rejecting the possible hit, the OOV word is associated with its closest IV word, replaced in the query and the score of the IV word is multiplied by the confusion probability.

Given an OOV word, the closest IV words is computed using the Levenshtein distance [3] that it'is a string metric for measuring the difference between two sequences. The Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other. However, very similar words can have the same editing distance, but just one them might be correct one in the query. To handle this problem, the M closest IV words are calculated for the OOV term, and then the confusion probability of all of them is compouted. The final proxy IV word for the OOV term is the word with the highest confusion probability.

To sum up, the grapheme-based model works like this for every word/morpheme/-token in a query phrase:

1. Compare query token with IV tokens. If it is IV, continue as usual.

2. If the query token is OOV, calculate the M closest IV tokens using Levenshtein distance and compute the confusion probability for all of them.
3. Get the IV token with the highest confusion probability and replace the OOV with it (proxy word).
4. Continue the normal procedure, remembering to multiply the token score by the confusion probability when computing the final hit score.

Score normalisation can be applied with out loss of generality after the query search, just keeping in mind that all hit scores must add to one if STO is being used.

The systems previously used are executed again, but this time using a grapheme confusion matrix to map OOV terms to IV terms. Table 5.1 collects the results.

System	All	IV	OOV	Th	P_FA	P_miss
decode + GPM	0.319	0.401	0.0	0.043	0.00002	0.663
decode + morph + GPM	0.312	0.388	0.018	0.011	0.00003	0.652
decode-morph + GPM	0.318	0.383	0.067	0.061	0.00003	0.651
decode + GPM + SN	**0.346**	**0.416**	**0.077**	**0.076**	**0.00003**	**0.626**
decode + morph + GPM + SN	0.318	0.395	0.017	0.036	0.00002	0.663
decode-morph + GPM + SN	0.324	0.391	0.067	0.057	0.00002	0.661

Table 5.1: Results for KWS using graphemes to handle OOV tokens

Comparing with table 3.1, we can see that using a grapheme confusion matrix barely changed the results. This is because the IV queries are computed in the same way, since the grapheme confusion matrix just performs when there is a term or multiple terms OOV, and the hits of OOV queries have tiny scores since the token scores are being multiplied by the confusion probability, decreasing the overall score of the hit and making it difficult to make a significant change from what was done before. However, when performing score normalisation the scores are normalised and they have more weight in the overall scoring.

The ASR output `decode.ctm`, employing a grapheme confusion matrix and performing score normalisation, achieves the best result so far, with a overall MTWV of 0.346 and improving significantly the OOV performance, with a OOV TWV of 0.077. The probability of miss has also decreased considerably compared to other systems.

Systems using two techniques to avoid OOV terms end up being even worse, because they are overparsing the ASR output when performing KWS, since the terms are first splitted into morphemes, and then they might be splitted into graphemes if they are OVV terms. Additionally, grapheme-based systems seem to outperform morpheme-based systems (at least in this practical) mainly because grapheme-based systems handle OOV terms in a probabilistic way, whereas morphological systems use a non-probabilistic procedure and potentially mistaken by performing a hard decision.

We can also compare the DET (Detection Error Tradeoff) curves for vanilla `decode.ctm` (after applying STO) and `decode.ctm` after applying both STO and a grapheme confusion matrix. This is shown in fig. 5.1 and fig. 5.2

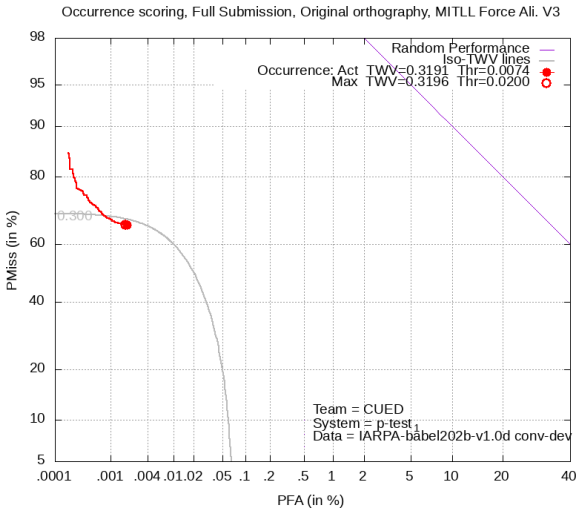


Figure 5.1: DET curve for `decode-STO`. The curve has a low x-axis value and a relative large y-axis value, what means that the system is having more false negatives than false positives, so it is underconfident. Additionally, it turns out that ATWV is in the exact same spot as MTWV, what means that the system is not hypothesising many hits, so it is including almost all of them.

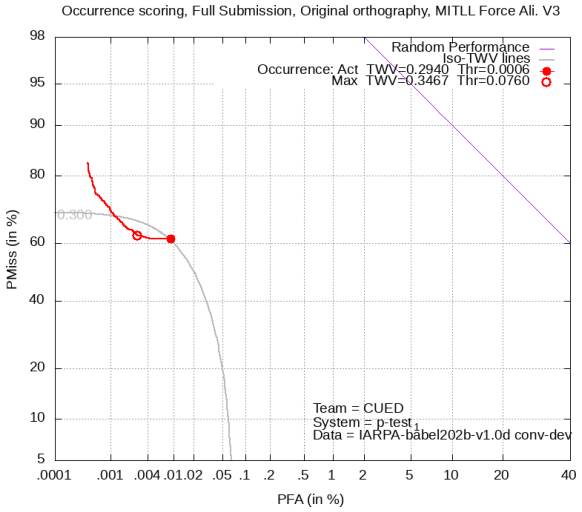


Figure 5.2: DET curve for `decode-graph-STO`. In comparison with fig. 5.1, it has a larger x-axis value, what means that the system is hypothesising more hits, so having more false alarms, i.e., false positives. Also, ATWV and MTWV are no longer identical, what means that the system is hypothesising a larger amount of hits. It seems that the grapheme confusion matrix is working properly and identifying more OVV terms.

6. System combination

Sometimes ASR outputs are combined to increase the robustness of the recognition. Similarly, combining multiple KWS systems together can lead to dramatic improvements. The goal of system combination is to take as input the detections of several systems, and then come up with a new list of hits which has performance that exceeds that of any individual system.

In this section, KWS system outputs are combined by merging their query hits if they refer to the same document and if the time overlaps. There are several way of combining the hit scores, but in this project two techniques are investigated based on their good performance on previous experiments [1]:

- **combSUM**: consists of computing the summation of the hit scores from the different indices: $\sum_{i=1}^{n} s(h_i)$, where $s(h_i)$ is the score of hit h_i among n systems.
- **WcombSUM**: computes the summation of the hit scores but the systems are weighted using their overall MTWV, so the best performing systems contributes more to the final merged score. The final expression of this fusion technique is: $\sum_{i=1}^{n} w_i^{MTWV} s(h_i)$, where $w_i^{MTWV} = \frac{MTWV_i}{\sum_{j=1}^{n} MTWV_j}$, among n total systems.

In addition to the previous used CTM files, we can investigate with the posting-lists generated for the 2015 surprise language evaluation systems: `morph.xml`, `word.xml` and `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. The obtained TWVs of those systems are shown in table 6.1, where we also include the results after applying score normalisation. Clearly, STO normalisation improved all the results, and the KWS system based on the input `morph` showed the best performance. The system `word-sys2` (applying STO) performed the best in IV queries, but `morph_SN` completely outperformed all the other systems in OOV queries. This proves again that morphological decomposition (section 3) is quite important to improve a system.

System	All	IV	OOV	Th	P_FA	P_miss
morph	0.360	0.430	0.089	0.071	0.00008	0.559
word	0.399	0.501	0.0	0.077	0.00006	0.536
word-sys2	0.403	0.507	0.0	0.047	0.00009	0.508
morph + SN	**0.520**	**0.559**	**0.367**	**0.039**	**0.00006**	**0.424**
word + SN	0.460	0.579	0.0	0.036	0.00006	0.479
word-sys2 + SN	0.465	0.585	0.0	0.030	0.00006	0.470

Table 6.1: Results for IBM WFST-based KWS

The provided IBM WFST-based systems are combined with our own models. Table 6.2 gathers all the experiments run for system combination, where plenty of models have been integrated.

The very first thing to highlight is that score normalisation makes a significant difference. The fact that SN was already useful to increase scores of rare queries and decrease the score of frequent queries, now it is also useful because unnormalised systems could be not able to combine since their scores were not comparable. Additionally, it is easy to check that the best performing models are those using combinations of the IBM WFST-based systems: `word_SN + morph_SN` is the best model with a MTWV of 0.544 (with a TWV of 0.588 IV and a 0.372 OOV), followed by `word_SN + word-sys2_SN + morph_SN` with a MTWV of 0.526. This emphasises the fact that word lattices ASR output is preferable compared to 1-best output.

System combination also favours our models, and pretty decent results are got when combined with IBM systems. In fact, combining `decode_SN` (using graphemes) with `word_SN` and `morph_SN` is the best performing system regarding OOV keywords, with a TWV of 0.388, the greatest in this project.

Finally, note that the combination technique does not make almost any difference. In general, combSUM tends to perform slightly better, but WcombSUM moderately outperforms combSUM when used with our own systems, like `decode_graph_SN`.

System	Method	All	IV	OOV	Th	P_FA	P_miss
word + word-sys2	cSUM	0.388	0.488	0.0	0.121	9e-5	0.524
morph + word + word-sys2	cSUM	0.386	0.463	0.087	0.197	1e-4	0.517
word_SN + word-sys2_SN	cSUM	0.454	0.571	0.0	0.072	7e-5	0.479
word_SN + morph_SN	**cSUM**	**0.544**	**0.588**	**0.372**	**0.027**	**8e-5**	**0.377**
morph_SN + word_SN + word-sys2_SN	**cSUM**	**0.526**	**0.568**	**0.364**	**0.072**	**1e-4**	**0.376**
decode_SN + morph_SN + word_SN + word-sys2_SN	cSUM	0.511	0.548	0.369	0.147	7e-5	0.423
decode_SN + decode-morph_SN + word_SN + word-sys2_SN	cSUM	0.455	0.555	0.068	0.090	8e-5	0.466
decode_graph_SN + morph_SN	cSUM	0.499	0.530	0.378	0.056	8e-5	0.417
decode_graph_SN + word_SN + morph_SN	**cSUM**	**0.517**	**0.551**	**0.388**	**0.052**	**12e-5**	**0.363**
decode_SN + decode-morph_SN + decode_graph_SN + morph_SN	cSUM	0.493	0.520	0.386	0.058	9e-5	0.415
decode-morph_SN + decode-morph_graph_SN + decode_graph_SN + morph_SN	cSUM	0.493	0.522	0.379	0.136	6e-5	0.452

decode-morph_SN + decode-morph_graph_SN + decode_graph_SN + morph_SN + word_SN + word-sys2_SN	cSUM	0.511	0.546	0.376	0.173	8e-5	0.413
decode_graph_SN + morph_SN + word_SN + word-sys2_SN	cSUM	0.509	0.543	0.375	0.147	8e-5	0.414
word_SN + morph_SN	**WcSUM**	**0.541**	**0.584**	**0.372**	**0.027**	**8e-5**	**0.381**
morph_SN + word_SN + word-sys2_SN	**WcSUM**	**0.526**	**0.568**	**0.364**	**0.024**	**1e-4**	**0.377**
decode_graph_SN + morph_SN	WcSUM	0.502	0.532	0.389	0.033	8e-5	0.421
decode_graph_SN + word_SN + morph_SN	**WcSUM**	**0.519**	**0.552**	**0.388**	**0.020**	**11e-5**	**0.368**
decode_graph_SN + morph_SN + word_SN + word-sys2_SN	WcSUM	0.511	0.545	0.376	0.035	8e-5	0.409
decode_SN + decode-morph_SN + de-code_graph_SN + morph_SN	WcSUM	0.498	0.527	0.388	0.020	8e-5	0.416
decode-morph_SN + decode-morph_graph_SN + decode_graph_SN + morph_SN	WcSUM	0.497	0.525	0.386	0.019	9e-5	0.415
decode-morph_SN + decode-morph_graph_SN + decode_graph_SN + morph_SN + word_SN + word-sys2_SN	WcSUM	0.511	0.544	0.386	0.029	8e-5	0.407

Table 6.2: Results for KWS using system combination. cSUM indicates that combSUM was run to combine scores, and WcSUM means that WcombSUM was used.

6.1. Query length analysis

We investigate what happens to performance as the length of the query increases. To extract the performance for different length queries the script termselect.sh script is used next to a modified version of `ivoov.map` file, called `length.map` to extract lengths instead of IV/OOV words.

We compare the performance of 4 systems that performed decently in the previous sections in table 6.3. To measure the performance, MTWV is employed.

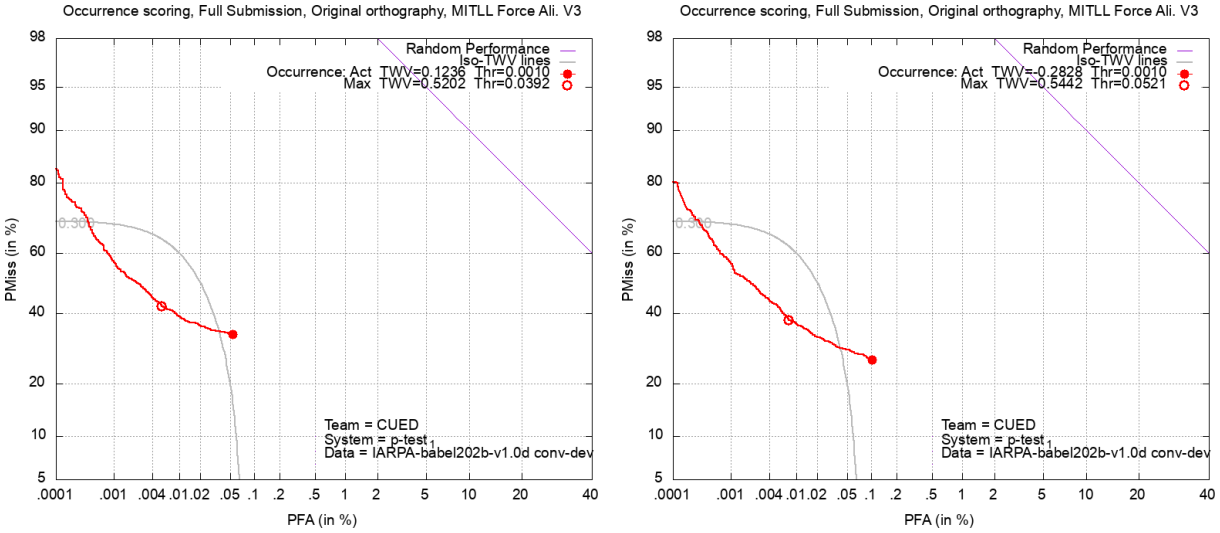
Length	% Queries	morph	morph_SN	morph_SN +word_SN	decode_graph_SN +morph_SN +word_SN
1	57.8%	0.372	0.526	**0.543**	0.502
2	34.8%	0.384	**0.575**	**0.598**	**0.591**
3	4.6%	0.197	0.293	0.389	0.380
4	2%	0.0	0.003	0.097	0.097
5	0.8%	0.0	0.0	0.0	0.0

Table 6.3: MTWVs of several systems investigating the importance of the query length

The best performing type of queries are those of length 2, where combined system using STO almost reach 0.6 of MTWV. On the other hand, queries of length 5 resulted catastrophic since there are none hits in any of the systems evaluated.

The usage of system combination and score normalisation STO are crucial to improve the performance. This can be clearly seen in queries of length 3 and 4, where the improvement in MTWV is the largest when using normalised combined systems.

As done before, we can also check the DET curves of some systems. In fig. 6.1 we compare the best individual system, `morph` applying score normalisation, and the best investigated system, the combination of `morph` with `word` and applying STO.



(a) DET curve for `morph` + SN

(b) DET curve after combining `morph` and `word`

Figure 6.1: Comparison of the best individual system vs. the best overall system after combination. In general, the curves are larger and has bigger x-axis values than fig. 5.1 and fig. 5.2, so the system are hypothesising more hits. In addition, the curves are closer to the $(0, 0)$ point, that would be the perfect performance, so the MTWV is greater. The combined system is having slightly more false positives but while reducing false negatives, ending up having a better PMiss vs. PFA tradeoff and larger MTWV.

7. Results and Conclusion

In this practical keyword spotting was studied using written queries on a low-resource language (Swahili). To handle OOV terms, morphological decomposition and grapheme systems were investigated. Additionally, score normalisation is computed in order to avoid using raw scores at query term level and to make different systems comparable. Last, KWS systems are combined to boost their individual performance, including IBM WFST-based KWS systems provided to CUED. The most remarkable results are summarised in table 7.1.

System	All	IV	OOV	Th	P_FA	P_miss
decode + GPM + SN	0.346	0.416	0.077	0.076	0.00003	0.626
morph + SN	0.520	0.559	0.367	0.039	0.00006	0.424
word_SN +morph_SN (cSUM)	0.544	0.588	0.372	0.027	8e-5	0.377
morph_SN +word_SN + word-sys2_SN (cSUM)	0.526	0.568	0.364	0.072	1e-4	0.376

Table 7.1: Summary of the most remarkable results of the project

Overall, score normalisation improved the MTWV for every system, and its usage was crucial when combining systems. The best implemented system is the one using `decode.ctm` with a grapheme confusion matrix to boost OOV performance and score normalisation (overall MTWV of 0.346). The best individual system was `morph`, that is a IBM KWS system with really good IV and OOV performance, with an overall MTWV of 0.52. Finally, system combination further improved the results, above all when combining IBM KWS systems, since the best performing system is the one uniting `word` and `morph`, that nor only obtains the highest overall MTWV (0.544), but also the highest IV MTWV (0.588) and OVV MTWV (0.372).

Bibliography

[1] J. Mamou, J. Cui, X. Cui, Mark J.F. Gales, K. Knill, et.al. System Combination and Score Normalisation for Spoken Term Detection. *IEEE International Conference on Acoustics, Speech and Signal Processing*. pages 8272-8276. 2013.

[2] Y. Wang, and F. Metze. An in-depth comparison of keyword specific thresholding and sum-to-one score normalizatio. *Proceedings of the Annual Conference of the International Speech Communication Association*. pages 2474-2478. 2014.

[3] Levenshtein_distance. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Levenshtein_distance