# Identifying contributors in the BBC World Service Archive

*Yves Raimond and Thomas Nixon*

BBC R&D

firstname.lastname@bbc.co.uk

## Abstract

In this paper we describe the speaker identification feature of the BBC World Service Archive prototype, an experiment run by BBC R&D to investigate alternative ways of publishing large radio archives. This feature relies on diarization of individual programmes, supervector-based speaker models, crowdsourcing for speaker identities, and a fast distributed index based on Locality Sensitive Hashing techniques to propagate these identities. We also describe how crowdsourced data can be used to continuously evaluate and refine our mapping from speaker models to speaker identities. We believe this experiment is one of the largest of its kind.

**Index Terms**: speaker identification, locality-sensitive hashing, crowdsourcing

10.21437/Interspeech.2014-17

## 1. Introduction

The BBC (British Broadcasting Corporation) has broadcast radio programmes since 1922 and has accumulated a very large archive of programmes over the years. A significant part of this archive has been manually catalogued by professional archivists to facilitate reuse, in other words to enable programme makers to easily find snippets of content to include in their own, newly commissioned, programmes. The coverage of such metadata is not uniform across the BBC's archive. For example, it excludes the BBC World Service, which has been broadcasting since 1932.

Between 2005 and 2008 the BBC World Service digitised the contents of its recorded radio programme library, amounting to around 70,000 programmes. The digitisation project was a great success but the metadata for it was of limited quality and quantity. It would take a significant amount of time and resource to manually annotate this archive. We therefore considered bootstrapping this annotation process using a suite of automated interlinking tools working from text and audio. We used the resulting data to publish this archive on the web (`http://worldservice.prototyping.bbc.co.uk`) and bootstrap search and navigation within it. We also built mechanisms for users to validate, correct and augment this automatically extracted data. This feedback can in turn be used to continuously evaluate and refine our algorithms.

Our first automated interlinking tool was tagging programmes with Linked Data web identifiers, using a combination of speech-to-text and a new algorithm for automated semantic tagging [12]. A benefit of using Linked Data web identifiers as tags is that they are unambiguous, and that we can retrieve more information about those tags when needed. For example, programmes tagged with places can be plotted on a map, or aggregation pages can be enriched with information about the corresponding topic. By having these anchor points in the Linked Data web, we can accommodate a wide range of unforeseen use-cases.

As well as knowing what topics have been discussed during a particular programme and use these topics to search the archive, it would be very valuable to be able to browse the archive by contributor: famous people, BBC presenters and correspondents, etc. Over the last year we have worked on adding this feature to the BBC World Service Archive prototype, releasing it across the entire archive in December 2013. This feature consists of a unique combination of speaker supervectors, Locality-Sensitive Hashing and crowdsourcing.

## 2. Related work

Speaker detection, identification and verification has been an active research field for a long time, often as part of speaker diarization (recognising distinct speakers and associated segments in an audio stream) and speech recognition (where acoustic models can be adapted to the distinct speakers recognised in the audio to achieve better accuracy). Detection and identification algorithms generally try to identify characteristics of each speaker's voice, capture these characteristics in a model, and match these models. For example [1] uses a linear prediction model and a simple distance between the resulting models to identify speakers.

One of the most popular types of speaker models is based on Gaussian Mixture Models (GMM). A set of audio features is derived from the audio signal, capturing some of its characteristics. Mel-Frequency Cepstral Coefficients (MFCCs) are often used for that purpose. The parameters of a GMM are then estimated over these features, to try and best match the distribution of the feature vectors. These models can be trained using maximum a-posteriori (MAP) adaptation of the means of a Universal Background Model (UBM) [13], a technique currently used in most speaker identification systems.

Then, these models can be compared and clustered in order to recognise speakers across different audio streams. A popular method is to use the Cross-Likelihood Ratio (CLR) [14], where for each pair of speakers the likelihoods of each speaker model to produce the sequence of features associated with the other speaker are compared. A few variants of this approach are described and evaluated in [17]. However these methods are relatively hard to scale to very large datasets. Direct comparison of models is often more efficient. For example the approach described in [3], on which our algorithm is based, uses a simple distance measure depending only on the speaker model parameters and shows that the algorithm is 3 times faster than a standard CLR-based approach without any loss in performance. Recent approaches to this problem are based on similar ideas — comparing speaker models, or descriptions of them, directly. These model-based approaches can be difficult to scale to very large datasets, as they need each pair of models to be compared. The main focus of the work presented below is to reduce the number of comparisons that need to be performed. Speaker 'su-

pervectors' (combining all means of a mean-adapted UBM in a single vector) are widely used as such models [5], or lower-dimensional representations, e.g. 'i-vectors' [6].

Another problem is the creation of a database of speaker identities. Most related work assumes the existence of such a database. Some methods have also been proposed to use multimodal data as a source for such identities, such as [4], using people's names captured from news video broadcasts. In our system this database is continuously evolving through crowdsourcing.

# 3. System architecture

We now describe the speaker identification system we built for the BBC World Service Archive prototype. We start by applying speaker diarization to each individual programme, leading to mean-adapted GMMs for each speaker in each programme, from which we generate speaker supervectors. We store these supervectors in a fast index and use this index to propagate crowdsourced speaker identities.

## 3.1. Speaker diarization and model training

The first step in our system is to segment and cluster the different speakers contributing to individual programmes. We use the LIUM toolkit for speaker diarization to perform this task [10], and in particular the algorithm developed for the ESTER2 evaluation campaign and described in [7]. At the end of the diarization process we end up with one cluster per distinct voice per programme, which we will denote as "speaker" in the following. For each speaker we also get an associated GMM with 512 Gaussian components, trained by MAP adaptation of the means of a UBM. We also throw out speaker models that are below a given likelihood threshold. This will exclude for example models trained on very small amounts of audio, which might be very inaccurate. At this stage we could use CLR clustering across programmes, as mentioned in Section 2. However in order to match speaker models across a large archive it is often not practical to require the features data, which the CLR method requires. Ultimately we'd like a method that is solely based on the adapted speaker models.

We now want to get to a measure, based on the parameters of these models only, of how similar two speakers are, which could be used as a basis for speaker identification. We use the measure presented in [3] as it only depends on the speaker models and is efficient to compute.

The symmetric Kullback-Leibler divergence $KL2$ between two mean-adapted speaker models $P_X$ and $P_Y$ for two speakers $X$ and $Y$ is upper bounded by the following distance [8]:

$$D_E(X,Y) = \sum_{k,d} w_k \frac{(\mu^X_{k,d} - \mu^Y_{k,d})^2}{\sigma^2_{k,d}}$$

Where $\mu^X_{k,d}$ and $\mu^Y_{k,d}$ are the $d$ components of the mean vectors of Gaussian $k$ in $P_X$ and $P_Y$ respectively, $w_k$ is the weight of Gaussian $k$ in both models and $\sigma^2_{k,d}$ is their common covariance matrix in Gaussian $k$.

We then use the detection score defined in [3], which is based on this distance measure. If $P_X$ and $P_Y$ are the models for speaker $X$ and speaker $Y$ respectively, the detection score will compare the distance between $P_Y$ and the UBM, denoted $\Omega$, with the distance between $P_Y$ and $P_X$.

$$D_E(P_Y, P_\Omega)^2 - D_E(P_Y, P_X)^2$$

However the distance between the adapted model and $\Omega$ will be highly dependent upon the amount of training data available, so the scores may be very heterogeneous. In order to deal with that we apply the 'M-Norm' normalisation method described in [2]. This normalisation consists of putting all the mean-adapted speaker models at a unit distance from $\Omega$. Once the normalisation applied, the detection score becomes equivalent to $1 - D_E(P_Y, P_X)^2$.

Assuming the availability of a database of speaker identities (a set $PI$ of tuples of the form $(P_X, I_X)$, where $P_X$ is the model for speaker $X$ and $I_X$ is a unique identifier for speaker $X$, and where a single identity can be associated with multiple models—for example derived from multiple programmes), we can propagate these identities to unknown speakers using this detection score. If $Y$ is an unknown speaker, we define $M_Y$ as the set of all speaker models for which the detection score is above a given threshold $T$.

$$M_Y = \{P_X \mid 1 - D_E(P_Y, P_X)^2 > T\}$$

We can then pick the most common identity associated with members of $M_Y$ through $PI$ to predict the identity $\hat{I_Y}$ of $Y$.

$$\hat{I_Y} = \arg\max_{I_M} |\{P_M \mid P_M \in M_Y, (P_M, I_M) \in PI\}|$$

However this identification process still requires pairwise comparisons between speaker models. On commodity hardware[1] and averaged over 1000 comparisons one comparison can take up to 1.2ms. The entire BBC World Service archive holds around 70,000 programmes. If ten distinct speakers per programme are detected on average, we would have to perform 490 billion comparisons to derive all possible matches. This would take around 18 years on one CPU. This number would be even higher when taking input/output into account, as it is unlikely all these models would fit in memory[2]. In order to match a new speaker against the full archive it would take around 10 minutes, and that number would go up linearly as more and more speakers get added to the database. We therefore consider creating an index that would enable us to quickly identify speakers across a large archive.

## 3.2. Locality-Sensitive Hashing

In particular, we consider using an index based on Locality-Sensitive Hashing (LSH). An introduction to this technique in a media indexing context is given in [15]. LSH is based on the simple idea that two points that are close to each other will remain close after a projection operation. In order to create a new LSH index we start by generating $n$ vectors $x$ where components are drawn at random from a Gaussian distribution, e.g. $\mathcal{N}(0,1)$. In order to index a new vector $v$ we define a binary hashing function as follows[3]:

$$h^x(v) = sign(x.v)$$

We repeat this operation $n$ times to transform our vector $v$ in $n$ bits, which will constitute our hash. This hashing function meets our requirement — if two points are close to each other in the original space, then they will have a higher probability of sharing the same hash. If two points are far from each other,

[1] Tested on an i7-3537U CPU

[2] At 96KB per model, all speaker models would occupy around 64GB of memory

[3] We use binary LSH which, as shown in [16], is very efficient.

they will have a low probability of sharing the same hash. Increasing $n$ will improve the precision of our hashing function — only points that are very close to each other will be likely to share the same hash.

Querying the index with a vector $q$ consists of computing its hash, finding all the vectors in the index that share that same hash, and returning those. In order to find the true nearest neighbours in that set of results we then need to order them by their distance to the vector $q$.

We also use $L$ independent projections to minimise the impact of an unlucky quantisation. For each vector $v$ we reach $L$ distinct hashes. When querying the index we pool the results from all these projections. By varying $L$ we can reach an arbitrary probability of getting the true nearest neighbours when querying the index.

The two main parameters needed to define a new binary LSH index are then:

- $n$, the number of bits constituting a single hash;
- $L$, the number of independent projections.

These two parameters capture a trade-off between performance and accuracy — between the amount of results returned for each query (which should be small in order to minimise the amount of time it takes to find the true nearest neighbours in them) and the recall of the results. The higher $n$ is, the more precise each individual hash will be, so the less results we will get for each query. Having very precise hashes also means the recall will be low — nearest neighbours will be less and less likely to be included in the results. The higher $L$ is, the better the recall of the index will be, but the lower its precision will be, i.e. the more results we will get for each query. Each type of problem will need very different $n$ and $L$ parameters. In most applications they are set empirically, but a few methods exist to find the optimal values for those parameters, e.g. [16] or the FLANN toolkit[4] [11].

### 3.3. Locality-Sensitive Hashing for speaker supervectors

We consider using a binary LSH index for the speaker models constructed and normalised above. We construct speaker supervectors for each speaker GMM by concatenating the normalised means of each Gaussian component. For each speaker, we get a vector $S$ of dimensionality $k * d$ (number of Gaussians multiplied by the size of the mean vector for each Gaussian), so 12288 components ($512 * 24$). We consider using LSH 'as-is' to index our speaker supervectors, to which we subtract the UBM supervector $U$. So our hashing function is defined for a supervector $S$ as:

$$h^x(S) = sign(x.(S - U))$$

This ensures our supervectors are centred around zero, which helps to minimise the number of bits per hash needed by the LSH index to achieve sufficient accuracy. If our supervectors are not centred, we would need a higher number of bits to create a high number of hash buckets in the region around the UBM supervector.

When queried with a seed speaker model, this index will return a reasonably small list of candidate speakers that may correspond to the same person, which can be used to greatly reduce the number of comparisons needed. An additional speed-up is introduced by distributing the supervectors between several of these indexes, each executing queries in parallel. We adapt the speaker identification algorithm described above to use this index:

1. Detect all distinct speakers in a new programme, and generate one supervector for each;
2. Insert the resulting speaker supervectors in an LSH index;
3. For each resulting speaker $X$ detected within the programme, query the LSH index with its supervector;
4. Dismiss any in-programme results, and compute the $D_E$ distance between the query and each result;
5. If that distance is below a given threshold, add the corresponding speaker to the set of matches for the speaker $X$ ($M_X$);
6. Assign to $X$ the most common identity associated with matching speakers ($M_X$).

### 3.4. Performance of LSH speaker index

To evaluate the performance of LSH speaker indexing, we sample 10,000 speaker supervectors from those generated while processing the archive. These supervectors are inserted into two similar single-threaded indexes, one using brute force comparison of supervectors, and one using LSH indexing as described.

Figure 1 shows the time taken to query each index as the number of supervectors stored increases. In this configuration, we find that the index using LSH only compares 0.92% of the indexed supervectors for each query, and thus gives an asymptotic speed-up of around 100 times.

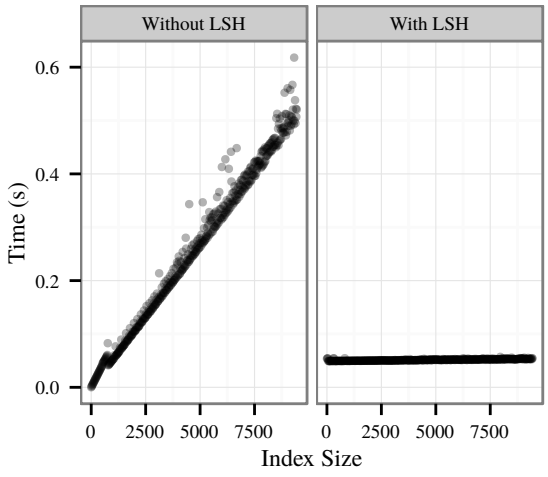


Figure 1: Index query time, using a sample of 10,000 speaker models from the archive. Anomalous points are caused by garbage collection in our implementation.

### 3.5. Crowdsourcing and propagating speaker identities

We started this prototype with an empty database of speaker identities. In order to bootstrap the creation of such a database, each unidentified speaker $X$ is displayed on the corresponding programme page with a "Speaker N" label, which links to a page listing all items in $M_X$. We then allow users to input or confirm identities for speakers as they listen to programmes.

[4] See `http://mloss.org/software/view/143/`.

We therefore need to implement step 6 in our algorithm to handle an ever-evolving database of speaker identities. We define a speaker $X$ to be *directly identified* as $I$ if the latest identification by a user for $X$ was $I$ — this constitutes our database of speaker identities $PI$. A speaker $X$ is *indirectly identified* as $I$ if most user identifications for speakers matching $X$ ($M_X$) are for $I$. When determining the identity of a speaker in the UI, the direct identity is preferred to the indirect identity if available. If only the indirect identity is available, we ask users to confirm or correct it. When confirmed, this indirect identity will be promoted to a direct identity.

Indirect identities can be slow to compute, especially for programmes with many speakers, each having many matches. To solve this problem, we implement a cache which stores possible identities for each speaker. This allows fast lookups of identities, while being relatively cheap to update with any new identifications from users.

For each speaker with a possible identity, we store:

- The current best identity (direct or indirect).
- A boolean flag stating whether this speaker id has been directly or indirectly identified.
- A list of pairs $(I, m)$ where $I$ is a speaker identity, and $m$ is the number of matches of this speaker which are directly identified as $I$.

To find the identity of a speaker, we simply look up the speaker in this cache. If there is no matching entry it has not been directly or indirectly identified.

When a speaker $X$ is identified or confirmed as $I$ by a user, we:

- Update the entry in the cache for $X$, setting the identity to $I$, and set the direct flag.
- Update the entry for each $M_X$, incrementing the count for $I$, and setting the identity to the identity with the highest count.

This process allows us to efficiently show and update indirect speaker identifications, but also has an additional benefit. By indexing the identity attribute of items in the cache, we can easily find a list of speakers with particular indirect identities. This allows a user to navigate between two programmes containing the same person, even if our speaker matching process did not produce a match for the relevant speakers in those programmes. This is especially useful in the context of LSH speaker indexing, where missing matches are more common.

## 4. Evaluation

We want to evaluate the accuracy of the contributor-based navigation within the prototype, i.e. how likely a user is to get to a programme actually including speaker $Y$ when on the page corresponding to a matching set $M_Y$. For that we consider evaluating the precision and recall of these sets against the data captured by the crowdsourcing components of the prototype. We took a snapshot of all user edits as of the 12th of July, 2013. This snapshot holds 787 user-added and user-validated speaker names for 238 distinct programmes. We report our results for various values of $T$ in Table 1, as well as the difference in performance introduced by an LSH index. We use 260 projections ($L$) and 16 bits per projection ($n$), which were cross-validated on a development dataset. We consider favouring precision in our application, and choose a value of $T$ accordingly.

| Algorithm | Precision | Recall | F-Measure |
|---|---|---|---|
| $T$ = 0.74 | | | |
| Without LSH | 0.9645 | 0.4027 | 0.5681 |
| With LSH | 0.9619 | 0.3401 | 0.5025 |
| $T$ = 0.78 | | | |
| Without LSH | 0.9563 | 0.4714 | 0.6314 |
| With LSH | 0.9553 | 0.3885 | 0.5524 |
| $T$ = 0.88 | | | |
| Without LSH | 0.8461 | 0.6 | 0.7021 |
| With LSH | 0.8814 | 0.4606 | 0.605 |

Table 1: Precision and recall of speaker aggregation pages for varying distance threshold values, without LSH index ([3] + KNN) and with LSH index.

Overall the LSH-based algorithm makes the F-Measure drop, which is due to the noise the index step is adding. However, we note that this drop is mostly in terms of recall. The LSH-based algorithm tends to preserve precision, and in some cases even improves it. This is due to pairs of speakers that are very close to the distance threshold. They will be more likely to not be matched with the LSH-based algorithm, as they will be more likely to be excluded from the results provided by the index.

We also track these measures in the prototype itself, and see how they evolve as we aggregate more and more user feedback[5]. The various anomalies in the graph correspond to various events. We first introduced the feature on a subset of 1,000 programmes in mid-2012 without the LSH index, and then deployed it to the rest of the archive in December 2013, using the new LSH index. As expected, when released across the whole archive we noticed a drop—from around 92% precision and 44.9% recall to 86.6% precision and 44.5% recall. Other anomalies correspond to bursts of new user feedback data and changes in the algorithm and its parameters.

## 5. Conclusion and future work

We described the speaker identification feature of the BBC World Service Archive prototype, combining a fast Locality-Sensitive Hashing-based index for speaker supervectors and crowdsourced speaker identities. This set-up enabled us to provide this feature across the 70,000 programmes featured in this archive, which is to the extent of our knowledge one of the largest experiments of its kind.

Our set-up is horizontally scalable, by distributing the diarization and model building processes (through our COMMA[6] platform) as well as the LSH indexes and the identity caches. We are therefore considering extending it to larger archives from the BBC and partner institutions. We are also investigation the use of i-vectors [6], as implemented in the LIUM toolkit [9], to reduce the dimensionality of our speaker vectors. This would significantly decrease the complexity of the hashing process and therefore reduce the fixed time offset introduced by the LSH indexes. This would also reduce the channel variability of our models and may therefore lead to an improvement in our evaluation results.

[5] See `http://worldservice.prototyping.bbc.co.uk/statistics/speakers`.

[6] See `http://www.bbc.co.uk/rd/projects/comma`.

## 6. References


[1] B.S. Atal. Effectiveness of linear prediction characteristics of the speech wave for automatic speaker identification and verification. *Journal of the Acoustical Society of America*, 55(6):1304–1312, 1974.

[2] Mathieu Ben and Frédéric Bimbot. D-MAP: A distance-normalized map estimation of speaker models for automatic speaker verification. In *Proceedings of ICASSP*, Hong Kong, China, 2003.

[3] Mathieu Ben, Guillaume Gravier, and Frédéric Bimbot. A model space framework for efficient speaker detection. In *Proceedings of INTERSPEECH*, 2005.

[4] Hervé Bredin and Johann Poignant. Integer linear programming for speaker diarization and cross-modal identification in TV broadcast. In *Proceedings of Interspeech*, Lyon, France, 2013.

[5] W. M. Campbell, D. E. Sturim, and D. A. Reynolds. Support vector machines using GMM supervectors for speaker verification. *IEEE Signal Processing Letters*, 13(5):308–311, May 2006.

[6] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, , and P. Ouellet. Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech and Language Processing*, 19(4):788–798, May 2011.

[7] P. Deléglise, Y. Estève, S. Meignier, and T. Merlin. Improvements to the LIUM French ASR system based on CMU Sphinx: what helps to significantly reduce the word error rate? In *Proceedings of Interspeech*, Brighton, United Kingdom, 2009.

[8] M. N. Do. Fast approximation of kullback-leibler distance for dependence trees and hidden markov models. In *IEEE Signal Processing Letters*, 2003.

[9] G. Dupuy, M. Rouvier, S. Meignier, and Y. Estève. I-vectors and ilp clustering adapted to cross-show speaker diarization. In *Interspeech, Portland, Oregon (USA)*, 2012.

[10] Sylvain Meignier and Teva Merlin. LIUM SPKDIARIZATION: An open source toolkit for diarization. In *Proceedings of the CMU SPUD workshop*, Dallas, TX, USA, 2010.

[11] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proceedings of the International Conference on Computer Vision Theory and Applications*, 2009.

[12] Yves Raimond and Chris Lowis. Automated interlinking of speech radio archives. In *Proceedings of the Linked Data on the Web workshop, World Wide Web conference*, Lyon, France, 2012.

[13] D. Reynolds, T. Quatieri, and R. Dunn. Speaker verification using adapted gaussian mixture models. *Digital Signal Processing*, 10:19–41, 2000.

[14] D.A. Reynolds, E. Singer, B.A. Carlson, G.C. O'Leary, J.J. McLaughlin, and M.A. Zissman. Blind clustering of speech utterances based on speaker and language characteristics. In *Proceedings of ICSLP*, Sydney, Australia, 1998.

[15] Malcolm Slaney and Michael Casey. Locality-sensitive hashing for finding nearest neighbors. *IEEE Signal Processing Magazine*, pages 128–131, March 2008.

[16] Malcolm Slaney, Yury Lifshits, and Junfeng He. Optimal parameters for locality-sensitive hashing. *Proceedings of the IEEE*, pages 1–20, 2012.

[17] Viet-Anh Tran, Viet Bac Le, Claude Barras, and Lori Lamel. Comparing multi-stage approaches for cross-show speaker diarization. In *Proceedings of Interspeech*, Florence, Italy, 2011.