Introduction to Meta-embeddings

by Niko Brummer 5 December 2017, Strand

See the full document at: https://github.com/bsxfan/meta-embeddings

Embeddings in machine learning and also in speech are now quite popular. Interspeech 2017 had 18 papers with "embedding" in the title. We need to work with complex patterns (images, speech, ...) that are difficult to compare and difficult to model probabilistically. For example, how do you compute (or even define) the `distance' between two variable-length sequences of MFCCs?

Embeddings make that better by mapping the input to a much simpler vector space, for example R^n. Now distances and probability distributions become much easier.

In the rest of machine learning (e.g. Facenet), they seem to be pretty much stuck at the idea of using distances (Euclidean, or cosine) to compare embeddings. We should not be too critical, because they get good results.

But in speaker recognition, we try to be more sophisticated and we use probabilistic methods such as PLDA in i-vector space, and more recently also PLDA in embedding space. Again those recipes work very well and we should appreciate that fact.

The idea of meta-embeddings is to do things just a little better than the extractor + PLDA recipe. The problem is that the typical extractor ignores the uncertainty associated with the ivector/embedding extraction. Although there have been some partially successful methods to propagate this uncertainty into PLDA, the idea of meta-embeddings is to tackle that problem from another angle, with fresh theory and tools, to see if we can do it better. Rather than gluing uncertainty on as an afterthought, let us design the uncertainty in there from the beginning.

We can define, roughly:

    meta-embedding = embedding + uncertainty

The theoretical insight that got the whole idea started at JHU was that the meta-embeddings, although more complex than a traditional embeddings, can still be viewed as *points* in a vector space. The really beautiful fact is now that the inner product between two meta-embeddings (each of them representing a recording) *is* the speaker verification likelihood ratio! And this LR properly takes account of the uncertainty that is built into the meta-embedding. If you build a well-calibrated meta-embedding extractor, you don't need PLDA postprocessing and you should also not need score calibration. (Haha, we'll see how that pans out in practice. Did I hear anyone slyly mumbling, "what about score norm"?)

The vector space where meta-embeddings live, is no longer plain Euclidean space (R^n), it is an infinite-dimensional Hilbert space. This somewhat complex theory is useful in guiding us towards algorithms to do the practical things we need to be able to do with meta-embeddings. But then we can come back to earth, to finite-dimensional computer-storable representations of the meta-embeddings and we can code scoring and training algorithms.

I think if I have to pick a particular strength of meta-embeddings it is that although they are (arguably) as flexible as deep generative models, they can be discriminatively trained, which is (much) easier than the tricky stochastic VB methods that are needed to train deep generative models. In a VB-autoencoder-type recipe for training a deep generative model, we need to build both an encoder and a decoder, even though at runtime we only ever need to use the encoder. For discriminative training, we only need the encoder (meta-embedding extractor). In practice, that means less than half the work, for both man and machine.

Also, VB can easily produce bad calibration. In contrast, the calibration-sensitive discriminative training criteria that I am proposing (there are several), should help to get calibration just right---see calibration experiments in chapter 8.

That said, meta-embeddings do not exclude generative methods. If you have (or can train) a generative model, then it naturally forms a very nice meta-embedding extractor. I explore one such model in chapter 7 and it comes in really useful later to prove that my discriminatively trained meta-embedding extractor is working properly.

What about weaknesses? Meta-embeddings are more complex than i-vector PLDA and we have to be very careful or things are going to get really slow! Scoring can be slow (e.g. a Cholesky factorization for every trial) and training can involve computing very many scores. The complexity of scoring and training are multiplicative. If you have to compute very many slow scores for training, you're screwed!

Fortunately there are solutions, some of which are explored throughout chapters 5,6,7 and 8. More work is needed, but I have already made enough progress that I believe things will work out.

Our first experiments on real data (planned with Lukas and Anya) will be to experiment with i-vectors as input. Eventually, we want to use MFCCs (or similar) as input, in particular so that we can extract the important uncertainty from the input data, which is no longer available in the i-vector. But to get things started, let's start with an easier goal.

So if we work with i-vectors is there anything useful for the meta-embedding to do that PLDA can't do? Yes, if we switch off length norm. The i-vectors with larger norms are noisier (or data-set shifted) and the meta-embeddings extracted from them should have larger uncertainty. This is explained in chapter 7, where a particular

form of heavy-tailed PLDA model is derived. It is just complex enough to produce meta-embeddings with variable uncertainty (Gaussian PLDA does not), yet still simple enough to allow fast scoring (no Cholesky!). I am really pleased with this model as a tool to do tyre-kicking of the whole idea.

As noted, training can compound complexity. The theoretically nicest, by-the-book training criterion has complexity that scales with the Bell number, that is literally astronomically. If you have a training database of just 75 recordings, the by-the-book discriminative training criterion is going to require about 10^80 CPU/GPU cores---that is one core for every atom in the observable universe!

Fortunately we have alternatives. One alternative is binary cross-entropy applied to pairs of recordings. This will probably work, although I have not tried it.

Another alternative (probably new to speaker recognition) is pseudolikelihood. This is what I have used in my first successful training experiment. I like it, because it is much faster than binary cross-entropy :-) As a caveat, others may not like it, because it is unfriendly towards automatic differentiation as implemented in the popular toolkits and it is also apparently hard to optimize with minibatch stochastic gradient. (In this lab, with MATLAB, handcoded derivatives and an L-BFGS full batch optimizer, we like pseudolikelihoood.)