A Comparison of Subspace Feature-Domain Methods for Language Recognition∗

W. M. Campbell, D. E. Sturim, P. Torres-Carrasquillo, D. A. Reynolds

MIT Lincoln Laboratory
wccampbell@ll.mit.edu, sturim@ll.mit.edu, ptorres@ll.mit.edu, dar@ll.mit.edu

Abstract

Compensation of cepstral features for mismatch due to dissimilar train and test conditions has been critical for good performance in many speech applications. Mismatch is typically due to variability from changes in speaker, channel, gender, and environment. Common methods for compensation include RASTA, mean and variance normalization, VTLN, and feature warping. Recently, a new class of subspace methods for model compensation have become popular in language and speaker recognition—nuisance attribute projection (NAP) and factor analysis. A feature space version of latent factor analysis has been proposed. In this work, a feature space version of NAP is presented. This new approach, fNAP, is contrasted with feature domain latent factor analysis (fLFA). Both of these methods are applied to a NIST language recognition task. Results show the viability of the new fNAP method. Also, results indicate which of the different methods perform best.

Index Terms: language recognition, feature compensation, support vector machines

1. Introduction

Typically, automatic language recognition has been performed with two techniques. A first method is based upon phone tokenization followed by language modeling, the PPRLM approach [1]. The PPRLM approach has traditionally been the most accurate. A second approach, that has received significant recent attention, is based upon shifted-delta cepstral coefficients (SDCCs) [2]. The SDCC methods have the advantage that they require no specialized language knowledge (i.e., phone labeling) and are computationally simple.

Gaussian mixture models, GMMs, can be used to model the distribution of the SDCCs for different languages. This modeling approach enables the use of two different subspace methods for model compensation, latent factor analysis (LFA) [3] and nuisance attribute projection (NAP) [4, 5]. These methods have been shown to be effective in both language and speaker recognition tasks.

Both LFA and NAP model a nuisance subspace and try to eliminate this subspace to reduce mismatch between training and testing. LFA arose in the context of modeling GMM supervectors, which are the stacked means from a GMM model. The subspace in this case is described with hidden factors with a Gaussian distribution which act as coordinates for transformation to a large dimensional space. NAP, in contrast, was designed as a general method for compensation of distances in SVM-based recognition. NAP attempts to project out directions in an SVM expansion space which are nuisances to an optimal distance metric for SVM design. NAP can be applied to GMM models via an approximate KL divergence kernel [5].

Recent work by Vair, et. al. [6] showed that LFA can be transferred to the feature domain. Our goal in this work is to extend this approach to NAP. We explore this approach in the context of GMMs trained with SVMs [7, 8] and standard maximum likelihood. We perform an extensive set of experiments to understand the performance of subspace methods and the corresponding tuning needed to achieve good performance.

The outline of the paper is as follows. In Section 2, we review language recognition based on SVMs and GMM mean and covariance supervectors. Section 3 discusses scoring for the SVM model. Sections 4 and 5 review NAP and LFA, respectively. Section 6 shows a general architecture for subspace methods and describes fNAP. Finally, Section 7 details experiments on the proposed technique on a NIST language recognition task.

2. SVM Language Recognition Training

An SVM [9] is a two-class classifier constructed from sums of a kernel function $K(\cdot, \cdot)$,

$$f(x) = \sum_{i=1}^{N} \alpha_i K(x, x_i) + d,$$

where $\sum_{i=1}^{N} \alpha_i = 0$ and $\alpha_i \neq 0$. The vectors $x_i$ are support vectors and obtained from the training set by an optimization process [10].

For closed-set language recognition, the goal is to determine the language of an utterance from a set of known languages. Since the SVM is a two-class classifier, we handle language recognition as a verification problem. That is, we use a one vs. rest strategy. For language recognition, we train a target model for the language. The set of known non-targets are used as the remaining class.

A straightforward method of performing language recognition with SVMs is to use kernels that compare sequences of feature vectors [11]. One technique for comparing sequences is to adapt a language independent GMM per utterance and then calculate a distance between the distributions [5].

Assuming this strategy, suppose we have a Gaussian mixture model UBM,

$$g(x) = \sum_{i=1}^{N} \lambda_i \mathcal{N}(x; m_i, \Sigma_i)$$

where $\lambda_i$ are the mixture weights, $\mathcal{N}(\cdot)$ is a Gaussian distribu-
tion, and \( \mathbf{m} \) and \( \Sigma \) are the mean and covariance of the Gaussian distributions, respectively.

Assume we have two utterances, \( \text{utt}_a \) and \( \text{utt}_b \). We train GMMs \( g_a \) and \( g_b \) as in (2), on the two utterances, respectively, using MAP adaptation. A natural distance between the two utterances is the KL divergence,

\[
D(g_a \| g_b) = \int_{\mathcal{X}} g_a(x) \log \left( \frac{g_a(x)}{g_b(x)} \right) dx
\]

Unfortunately, the KL divergence does not satisfy the Mercer condition, so using it in an SVM is difficult and computationally expensive. Instead of using the divergence directly, an approximation is typically used [5]. The idea is to bound the divergence using the log-sum inequality [5],

\[
D(g_a \| g_b) \leq \sum_{i=1}^{N} \lambda_i D\left( \mathcal{N}(\cdot; \mathbf{m}_{a,i}, \Sigma_{a,i}) \right| \mathcal{N}(\cdot; \mathbf{m}_{b,i}, \Sigma_{b,i}) \right)
\]

where we have represented the \( i \)th mixture component means of the adapted supervectors by \( \mathbf{m}_{a,i} \) and \( \mathbf{m}_{b,i} \), and the adapted covariances are similarly denoted. After several further approximations from (4), see [7], we arrive at a mean plus covariance kernel,

\[
K(g_a, g_b) = \sum_{i=1}^{N_a} \lambda_i \mathbf{m}_{a,i} \Sigma_{a,i}^{-1} \mathbf{m}_{b,i} + \sum_{i=1}^{N_b} \lambda_i \text{tr}\left( \Sigma_{a,i} \Sigma_{b,i}^{-1} \Sigma_{b,i} \right)
\]

For the kernel (5), our convention is to stack means followed by covariances.

### 3. SVM Language Recognition Scoring

Rather than score directly using the standard SVM formulation (1), recent work has shown that a GMM scoring of a pushed model yields better results [7, 8]. To obtain the pushed GMM model, our strategy is to normalize the support vectors to produce means and covariances. We divide the support vectors, \( \{ \mathbf{x}_i \} \), into two sets based on the sign of \( \alpha_i \), and construct two supervectors for a plus and minus GMM model. The plus model, \( g_p \), has mean and covariance parameters given by

\[
\mathbf{D}^{-1} \mathbf{x}_p = \mathbf{D}^{-1} \frac{1}{\sum_{i: \alpha_i > 0} \alpha_i} \sum_{i: \alpha_i > 0} \alpha_i \mathbf{x}_i.
\]

The matrix \( \mathbf{D} \) is diagonal matrix given by

\[
\mathbf{D} = \text{diag}(\text{diag}(\mathbf{D}_m), \text{diag}(\mathbf{D}_s))
\]

\[
\mathbf{D}_m = \text{diag}\left( \text{diag}(\sqrt{\lambda_1} \Sigma_{1}^{-1/2}), \cdots, \text{diag}(\sqrt{\lambda_N} \Sigma_{N}^{-1/2}) \right)
\]

\[
\mathbf{D}_s = \text{diag}\left( \text{diag}(\sqrt{\lambda_1} \Sigma_{1}^{-1}), \cdots, \text{diag}(\sqrt{\lambda_N} \Sigma_{N}^{-1}) \right)
\]

A similar process yields a minus model, \( g_m(\mathbf{x}) \). It is easy to verify that the covariances for this process are positive.

Scoring with these GMM models is straightforward. For an input set of vectors, \( \mathbf{y}_i \), we produce a log likelihood ratio,

\[
score = \sum_i \log \left( g_p(\mathbf{y}_i) \right) - \sum_i \log \left( g_m(\mathbf{y}_i) \right).
\]

### 4. Nuisance Attribute Projection

Nuisance Attribute Projection (NAP) is a method for improving performance in SVM recognition with knowledge of nuisance attributes such as channel, session, language, etc. [4, 5]. The starting point for understanding NAP is to note that an SVM kernel induces a distance, \( d(\mathbf{x}, \mathbf{y}) \),

\[
d^2(\mathbf{x}, \mathbf{y}) = K(\mathbf{x}, \mathbf{x}) + K(\mathbf{y}, \mathbf{y}) - 2K(\mathbf{x}, \mathbf{y}) = \| \phi(\mathbf{x}) - \phi(\mathbf{y}) \|^2
\]

where the SVM feature space is the output of \( \phi(\cdot) \) and \( K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y}) \). The relation in (9) shows that the distance for optimizing the SVM for maximum margin is directly impacted by having a kernel insensitive to channel effects.

In SVM expansion space, directions correspond to classification operations. I.e., if we have a hyperplane characterized by a normal vector, \( \mathbf{w} \), then we can classify according to the score \( \mathbf{w}^T \phi(\mathbf{x}) + d \) (this is just (1) rewritten). If we could find directions that classify nuisances, then it would be natural to remove them to obtain an improved feature space. An orthogonal projection, \( \mathbf{P} \), is a solution \( (\mathbf{P}^2 = \mathbf{P}, \mathbf{P}^T = \mathbf{P}) \) for removing this nuisance. Removing a few directions leads to \( \mathbf{P} \) of the form,

\[
\mathbf{P} = I - \sum_{i=1}^{N_p} \mathbf{v}_i \mathbf{v}_i^T
\]

where \( N_p \) is a “small” number.

If we have a data set, \( \{ \mathbf{x}_i \} \), labeled by language, then the NAP criterion to reduce session variability is to minimize the average variation; i.e.,

\[
\arg\min_{\mathbf{P}} \sum_{i,j} W_{i,j} \| \mathbf{P} \phi(\mathbf{x}_i) - \mathbf{P} \phi(\mathbf{x}_j) \|^2
\]

where \( W_{i,j} = 1 \) if \( \mathbf{x}_i \) and \( \mathbf{x}_j \) are the same language, and \( W_{i,j} = 0 \) otherwise. Solving the NAP problem in (11) can be accomplished using a kernel approach. We solve for the eigenvalues and eigenvectors of the following,

\[
Z(\mathbf{W}) \mathbf{K} \mathbf{w}_i = N_w \lambda_i \mathbf{w}_i
\]

where \( \mathbf{W} \) is the matrix with elements \( W_{i,j} \), \( Z(\mathbf{W}) = \text{diag}(\mathbf{W}) - \mathbf{W} \), \( \text{diag} \) is the operation of turning a vector into a matrix, and \( I \) is the vector of all ones. The matrix \( \mathbf{K} \) has elements \( K_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j) \). The eigenvalues are positive and real and are ordered from largest to smallest with \( \lambda_1 \) being the largest. The vectors, \( \mathbf{v}_i \), in (10) are then given by

\[
\mathbf{v}_i = \frac{[\phi(\mathbf{x}_1) \phi(\mathbf{x}_2) \cdots \phi(\mathbf{x}_N)] \mathbf{w}_i}{\| \mathbf{w}_i \|}
\]

### 5. Latent Factor analysis

In the latent factor analysis method of Kenny [3], the GMM supervector \( \mathbf{m}(l, i) \) is dependent on the language \( l \) and the utterance \( i \). The supervector is a sum of a language component, \( \mathbf{m}(l) \), and an utterance dependent vector,

\[
\mathbf{m}(l, i) = \mathbf{m}_{\text{lm}} + \mathbf{m}(l) + \mathbf{U} \mathbf{z}(l, i).
\]

The latent factor \( \mathbf{z}(l, i) \) is assumed to be zero mean, unit variance and Gaussian.

One key problem in factor analysis is to estimate the latent factor \( \mathbf{z}(l, i) \) given Baum-Welch statistics from an utterance [12]. A standard assumption is to set \( \mathbf{m}(l) = 0 \) and use
sufficient statistics based on the UBM. The resulting equation for determining the factors is
\[
(I + U\Sigma^{-1}NU)z = U\Sigma^{-1}F
\]  
(15)
In the equation, I is the identity matrix, N is a diagonal matrix with the occupancy counts for an utterance, \(\Sigma\) is the diagonal matrix formed by from the stacked UBM covariance vectors, and F is the UBM centered first order statistics.

Another key problem in factor analysis is the training of the hyperparameter, U. For a large number of utterances, from (14) we note that
\[
m = \frac{1}{n}\sum_{i=1}^{n}m(l, i) - m_{\text{ubm}} \approx m(l).
\]  
(16)
Thus, we can derive the subspace generated by U by finding the principal components of the autocorrelation matrix
\[
R = \sum_{i=1}^{n}(m - m(l, i))(m(l, i))^t.
\]  
(17)
We use MAP adaptation to determine the \(m(l, i)\) and assume (16) when determining U. The principal components are calculated via kernel PCA.

6. Subspace Feature Compensation
We would now like to apply the model domain methods shown in Sections 4 and 6 in the feature domain. The key equation in making the connection between these approaches is to first look at how feature vectors, \(\{y_j\}\), are mapped into SVM expansion space. If N is the occupancy counts in diagonal matrix form, then given a large number of vectors, \(\phi(\{y_j\})\) is
\[
D_mN^{-1}\sum_j [p(1|y_j)y_j^t \cdots p(N|y_j)y_j^t]^t
\]  
(18)
where \(D_m\) is given in (7), and we are using only the mean term in the SVM kernel (5). In (18), \(p(i|y_j)\) is the posterior probability of being in mixture \(i\) of the UBM given the observation \(y_j\).

Suppose we have a nuisance vector determined from NAP, \(b = VV^t\phi(\{y_j\})\), where V is the matrix with columns \(v_i\), see (13). Then we can map to a mean vector as \(m_b = D_m^{-1}b\). Applying the NAP compensation to (18) then yields an expansion vector with the term \(p(i|y_j)y_j\) replaced by \(p(i|y_j)(y_j - m_b,i)\) in (18) (since \(m_b\) is a constant and \(n_i = \sum_j p(i|y_j)\)). Here \(m_b,i\) is the ith subvector of \(m_b\).

Now to go back into the feature domain, we apply a common lossy process [6, 13]. We have a scaled expansion vector of the form,
\[
[p(1|y_j)(y_j^t - m_b,i) \cdots p(N|y_j)(y_j^t - m_b,i)]^t.
\]  
(19)
Summing across all subvectors in (19) gives the subspace compensation technique,
\[
y = y - \sum_{i=1}^{N}p(i|y_j)m_b,i
\]  
(20)
We refer to this as the compensation in (20) with the NAP mean bias vector \(m_b\) as fNAP.

7. Experiments
Experiments were performed on the 14 language closed-set task for the NIST 2007 language recognition evaluation (LRE) data. Target languages include Arabic, Bengali, Chinese, English, Farisi, German, Hindustani, Japanese, Korean, Russian, Spanish, Tamil, Thai, and Vietnamese.

Training data was primarily from Callfriend and Callhome; although, for languages such as Arabic, data was also used from Fisher and Mixer. Additional data was supplied by NIST/LDC for Arabic, Bengali, Thai, and Chinese dialects.

Our development set, LRE07 DEV, included approximately 6000 utterances per duration for durations of 3, 10, and 30 seconds. LRE07 DEV was used for training backend fusion.

The criterion for evaluation was pooled EER. In the EER calculation, we equalized the priors of languages in the test set, so that each language had an equal contribution to the EER. This strategy corresponds to the NIST scoring criterion which balances priors for the Cavg score.

For feature extraction, SDCC features were used with a 7-1-3-7 parameterization [2]. This corresponds to 7 delta cepstral coefficients stacked from 7 different time locations. We also included cepstral coefficients for a total of 56 features per frame at 100 frames per second. Additional processing included RASTA, 0/1 feature normalization, and VTML.

Every effort was made to ensure similarity in training setup between fNAP and fLFA. Common training sets and common feature sets were used. The significant application difference between the two techniques was that method-specific training was used to obtain the subspace. For fNAP, training was accomplished using the mean only portion of the SVM kernel in (5). The subspace was then obtained using the solution method in (12) and (13). For fLFA, hyperparameters were extracted using an eigenvector methods as described in Section 5.

Language and gender independent GMM UBMs were trained using all of the training data with 5 iterations of EM adapting all parameters—means, mixture weights, and diagonal covariances. SVMs were trained with the mean plus covariance kernel in (5). GMM models for the SVM parameters were generated and scored using the modeling pushing method described in Section 3. Also, as a baseline, we trained GMM language specific models with the maximum-likelihood training (ML) by using a starting point of the GMM UBM and performing 10 iterations of EM per language model modifying all parameters—means, weights, and covariances.

For all systems, using the raw scores for classification was suboptimal, so final scores were processed using a backend fusion system [14]. The backend transforms scores using linear discriminant analysis and models the resulting vector using a tied covariance Gaussian per language. Per language scores are calibrated separately using an artificial neural network. All fusion was trained on the development set which was disjoint from the NIST LRE 2007 corpus.

Our first set of comparisons was to sweep two parameters—the dimension of the subspace and the number of mixture components for the subspace feature extraction. We note that in all of our experiments the GMM language models had 2048 mixture components. Also note that the GMM mixture order of the feature extraction does not have to match the language model. The results are shown in Table 1.

Table 1 shows that the optimal dimension is around 64. Higher dimensions remove too much energy and degrade performance. The general trend observed is that for dimensions less than 64, the error rate decreases with increasing dimen-
Table 1: A comparison of EERs for different GMM subspace mixture orders (num mix), subspace dimensions (dim), and durations (30s and 10s) for fNAP and fLFA. GMM language models were 2048 mixture components and were trained with the SVM mean and covariance kernel.

<table>
<thead>
<tr>
<th>num mix</th>
<th>dim</th>
<th>fNAP 30 EER (%)</th>
<th>fLFA 30 EER (%)</th>
<th>fNAP 10 EER (%)</th>
<th>fLFA 10 EER (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>32</td>
<td>1.47</td>
<td>1.51</td>
<td>5.57</td>
<td>5.53</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>1.43</td>
<td>1.45</td>
<td>5.30</td>
<td>5.19</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>1.43</td>
<td>1.53</td>
<td>4.99</td>
<td>5.68</td>
</tr>
<tr>
<td>2048</td>
<td>32</td>
<td>1.64</td>
<td>1.51</td>
<td>5.45</td>
<td>5.34</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>1.65</td>
<td>1.47</td>
<td>5.67</td>
<td>5.42</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>1.45</td>
<td>1.60</td>
<td>5.37</td>
<td>5.79</td>
</tr>
</tbody>
</table>

Table 2: A comparison of the EER for different training methods for fLFA and fNAP with a subspace dimension of 64, durations 10s and 30s, and a GMM subspace model with 512 mixture components.

<table>
<thead>
<tr>
<th>Training Method</th>
<th>Subsp. method</th>
<th>EER 30s (%)</th>
<th>EER 10s (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML</td>
<td>fLFA</td>
<td>2.78</td>
<td>6.83</td>
</tr>
<tr>
<td>ML</td>
<td>fNAP</td>
<td>2.83</td>
<td>7.04</td>
</tr>
<tr>
<td>SVM</td>
<td>fLFA</td>
<td>1.45</td>
<td>5.19</td>
</tr>
<tr>
<td>SVM</td>
<td>fNAP</td>
<td>1.39</td>
<td>5.30</td>
</tr>
</tbody>
</table>

Another observation from the table is that lower mixture orders tend to produce moderately better results. This trend is useful since feature extraction with a lower mixture order reduces computation significantly. Finally, we note that the general trend with both methods is that fNAP and fLFA produce similar EERs for a variety of parameter settings.

We next focused on the performance of fLFA and fNAP under different training methods—SVM or ML. Results show that both fLFA and fNAP improve results for these training methods. Also, at 10s, fLFA has a slight performance edge.

Our final comparison is shown in Table 3. We experimented with turning subspace methods off at various points in training and testing. For instance, if a training mask of 011 was used, then feature extraction for training the UBM would use only SDCC features with no subspace feature method; the remaining language model and scoring used the subspace methods.

From Table 3 several interesting observations can be made. First, the improvement from no compensation, 000, to full compensation, 111, yields the biggest improvement in EER of about 33% reduction at 30s. The 10s case shows less improvement, presumably since it is difficult to get a good estimate of the nuisance space with little test data. A second point is that the table allows us to gauge the importance of the subspace methods in the various stages. If we toggle the UBM bit and leave other bits constant, one can see that the subspace methods have the least impact in the UBM training. In the middle is language model training, and subspace methods have the most impact at scoring. An interesting result is that applying subspace methods only in scoring yields large gains.

8. Conclusions

An in depth analysis of feature domain subspace method for language recognition was presented. A new subspace method, fNAP, was derived. Results showed the new method works well and achieves similar performance to latent factor methods.

Table 3: Comparison of the EER for different combinations of feature extraction for fLFA and fNAP with 512 mixture components and a subspace dimension of 64. The training mask indicates where the subspace feature extraction was used—UBM (0/1), language model training (0/1), and scoring (0/1).

<table>
<thead>
<tr>
<th>Training Mask</th>
<th>fNAP 30 EER (%)</th>
<th>fLFA 30 EER (%)</th>
<th>fNAP 10 EER (%)</th>
<th>fLFA 10 EER (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>2.10</td>
<td>1.45</td>
<td>5.67</td>
<td>5.37</td>
</tr>
<tr>
<td>001</td>
<td>1.52</td>
<td>1.55</td>
<td>5.56</td>
<td>5.19</td>
</tr>
<tr>
<td>010</td>
<td>1.90</td>
<td>1.98</td>
<td>5.95</td>
<td>5.85</td>
</tr>
<tr>
<td>011</td>
<td>1.58</td>
<td>1.44</td>
<td>5.40</td>
<td>5.35</td>
</tr>
<tr>
<td>100</td>
<td>2.23</td>
<td>2.06</td>
<td>6.06</td>
<td>6.26</td>
</tr>
<tr>
<td>101</td>
<td>1.57</td>
<td>1.43</td>
<td>5.68</td>
<td>5.37</td>
</tr>
<tr>
<td>110</td>
<td>2.07</td>
<td>1.89</td>
<td>5.76</td>
<td>6.05</td>
</tr>
<tr>
<td>111</td>
<td>1.39</td>
<td>1.45</td>
<td>5.30</td>
<td>5.19</td>
</tr>
</tbody>
</table>

9. References