CONTENTS

1. INTRODUCTION

1.1 Problem Description and Motivation

1.2 Previous work

In this section we will be reviewing representative works in pronunciation assessment at phone level, which is the topic of interest in this thesis.

One of the simplest ways of scoring a phone pronunciation found in the literature is to use segment duration scores ([1, 2]), which are obtained through the following procedure. First, a duration distribution is generated for each phone using the native training data. Then, phone durations in frames for nonnative utterances are obtained and its value is normalized to compensate for rate of speech. Finally, phone-segment-duration scores are computed as the log-probability of the normalized duration, using the duration distributions previously mentioned. Phone durations are obtained from Viterbi alignments. In the same works, other two methods that use *Hidden Markov Models* (HMMs) to obtain spectral matches and compute pronunciation scores are tested. Phonetic time alignments for nonnative utterances are generated from an HMM-based speech recognition system trained with native instances. In order to do that, the text pronounced by the student must be known. This is achieved by eliciting speech in a constrained way, such as reading a predefined text. From these phonetic segmentations two probabilistic measures based on HMMs are computed as scores: log-likelihood and log-posterior probabilities. The underlying asumption is that the logarithm of the likelihood or the posterior of the speech data, computed by the Viterbi algorithm using the HMMs trained with speech from native speakers is a good mesasure of the similarity between the students's speech and native-sounding speech.

For each phone segment the log-likelihood score $\hat{\text{l}}$ is computed as:

$$\hat{l} = \frac{1}{d} \sum_{t=t_0}^{t_0+d-1} log\ p(y_t|q_i) \tag{1.1}$$

where $p(y_t|q_i)$ is the likelihood at time t with observation vector y_t given the phone class q_i, d is the duration in frames of the phone segment and t_0 is the starting frame index of the phone segment. The likelihood is computed through a forced recognition phase by using a known ortographic transcription of the speech signal. Duration normalization is done to eliminate the dependency of the pronunciation score on the duration of the phone.

Alternatively, log-posterior scores can be computed for each time t:

$$P(q_i|y_t) = \frac{p(y_t|q_i)P(q_i)}{\sum_{j=1}^{M} p(y_t|q_j)P(q_j)} \tag{1.2}$$

Likelihoods in both numerator and denominator are computed in the same way as in Equation 1.1. The sum over j runs over a set of context-independent models for all phone classes. $P(q_i)$ represents the prior probability of the phone class q_i.

Finally, the posterior score $\hat{\rho}$ for the phone segment is defined as:

$$\hat{\rho} = \frac{1}{d} \sum_{t=t_0}^{t_0+d-1} log\ P(q_i|y_t) \tag{1.3}$$

To test each method, a database of nonnative read speech is transcribed and scored for pronunciation quality by expert human raters. Log-posterior probabilities achieves the highest correlation with human ratings, outperforming log-likelihood and normalized duration scores.

A very similar approach to log-posterior probabilities named *Goodness of Pronunciation* (GOP) was developed at the same time as log-posteriors and is investigated in some works [3, 4, 5]. The quality of pronunciation for any phone p is defined to be the duration-normalized log of the posterior probability that the speaker uttered phone p given the corresponding vector of observations $y = \{y_{t_0}, \dots y_{t_0+d-1}\}$

$$GOP(p) = \left| log \left(\frac{p(y|q)P(p))}{\sum_{q \in Q} p(y|q)P(q))} \right) \right| \Big/ d \tag{1.4}$$

The likelihoods $p(y|q)$ are obtained from the recognizer as a sum of the likelihoods over all observations in the phone. The difference between GOP scores and the log-posterior score technique previously mentioned is that in GOP, the likelihood for both numerator and every phone in the denominator is computed at segment level as a sum of log-likelihoods per frame over the duration of the phone p, while in Equation 1.2 log-posterior probabilities are computed at the frame level and averaged over the segment's length.

A different technique is explored in [6], which studies the relationship between phonetic mispronunciations that occur in a word and the actual word rating. The research is based on a set of phonological rules representing phonetic mispronunciations between L1 (native language of the students) and L2 (the target language). In this case, the corpus is obtained from Chinese Learners of English and all the speech data of the corpus is phonetically transcribed by trained linguists. In addition, crowdsourcing is used to collect perceptual gradations of word-level mispronunciations and the WorkerRank algorithm is used to conduct quality control to filter crowdsourced data in terms of reliability. The gradation score for a phonological rule is obtained as the average of the aggregated values of the words (rated via crowdsourcing) containing that rule. On the other hand, gradation for a word is calculated as either the maximum value among all phonological rules that are present in that word or a linear combination of them. Even though the work is focus on the use of phonological rules in word-level mispronunciation gradation, the same approach can be used to evaluate mispronunciation gradation at phone level. In order to derive the phonological rules from the mispronunciations, the previous strategy requires a phonetically labeled nonnative database. Taking that into account, another paper [7] works well as complement to [6], because it develops a speech recognition system for automatic mispronunciation detection capable of transcribing the learner's input speech.

So far, all the aforementioned methods but the last one are based on confidence measures obtained from Automatic Speech Recognition (ASR) systems. Scores measure how closely the utterance of the speaker matches the recognizer's acoustic model. Mismatches result in low confidence scores, which provide a profile of the speakers' production errors. Nevertheless, the accuracy of the assessment based on these confidence scores can be quite low [8]. In addition, ASR systems based on HMMs are both time-consuming to train and extremely vulnerable to acoustic interference and variation in speaking style, and the conventional methods for enhancing ASR performance often require enormous amounts of data collection and annotation, as well as extensive training on representative material. For those reasons, other types of classifiers are explored in many works, specially after the 1990s.

Support Vector Machines (SVMs) are a preferred choice due to their excellent generalization capability and suitability for 2-class classification problems. Moreover, in contrast to the confidence models described above, which do not take into account misspronounced data, SVMs are trained directly to discriminate the two classes of interest. Of course, this poses a new requirement: both correctly and incorrectly-pronounced data should be available for training.

In [9], SVMs are used to discriminate between good and mispronounced vowels in Dutch. Three different types of features for training the models are evaluated: log-posterior-based scores obtained from HMMs, MFCCs and a set of phonetic features (first three formants, pitch and intensity). Despite the existence of nonnative-speech databases for Dutch language, they were considered too small for the purpose of the research. For that reason, phonemic substitutions that induce vocalic errors are simulated by artificially introducing them in the native corpus. It is worth mentioning that training and testing the models with artificially generated data is not recommended because it can lead to models that don't generalize well to real nonnative data. Having said that, the results show that the best results are obtained by using MFCCs as features of the SVM models, followed by confidence-measure-based scores and finally phonetic features, though substantial improvements can be obtained by combining them.

In a different work [10], SVM is used as the classifier and the log-likelihood ratios between all the acoustic models and the model corresponding to the given phone are employed as features for the classifier. In other words, given the observation y, the feature vector for the classification problem is computed as:

$$[LLR(y|q,q_1),\ LLR(y|q,q_2)\dots LLR(y|q,q_Q)], \tag{1.5}$$

where q is the canonical label of the observation being pronounced, $q_1, q_2 \dots q_Q$ represent the set of acoustic models of all the phones and LLR is defined as:

$LLR(y|q,q_i) = log\ p(y|q_i) - log\ p(y|q)$, where likelihood is computed at segment level as in Equation 1.4 for GOP scores. The reason for choosing these features is that q can be mispronounced as any other phone and the likelihood ratio is a powerful method to detect this kind of mispronunciation. Classifiers trained with this kind of features, however, can only effectively deal with a phone mispronounced as another phone. This kind of acoustic models is less effective for partially changed mispronunciations, that are the most frequent mispronunciations in practice. In order to detect all kinds of mispronunciation, a technique called *Pronunciation Space Models* (PSMs) is introduced. The idea is to represent pronunciation variations of different proficiency levels. The traditional phone-based model q is expanded to $\{q_1, q_2 \dots q_K\}$ by means of an unsupervised algorithm that doesn't require speech data labeled with proficiency level. This set of models represents K types of pronunciation variations ranging from perfect to totally wrong pronunciations, where K is a tunable parameter. To build the PSM models for a given phone q, all the instances are sorted according to their posterior probabilities obtained from the original acoustic model for q, and then uniformly split into K group of the same size. Finally, an individual new acoustic model is trained with the observation from each group via maximum likelihood estimation. The feature vector dimension of 1.5 is therefore increased from Q to $Q * K$. Experiments are carried out on the Mandarin mispronunciation detection task for native Chinese speakers with various dialect accents, and data labeled as "correct" or "mispronounced" by experienced human annotators is used to train and test the models. Results show that SVM based on log-likelihood ratio features outperforms GOP scores (used as baseline), and some improvements are obtained when adding PSMs.

Finally, a last example of SVM-based resarch is found in [8, 11]. A landmark-based SVM is introduced and compared with a confidence scoring method over 10 phonemes where L2 English learners, whose native language is Korean, make frequent errors. Landmark theory relies on the fact that humans can perceive distinctive features using only spectral features extracted from the time frame including and adjacent to a landmark (sudden signal change). A particular SVM for each phoneme is trained. Vowel SVMs are trained using one or more frames from the vowel center. Frames from both onset and offset (prevocalic and postvocalic position) are selected and used in the training of consonant SVMs. The confidence score method shows a similar performance to SVM, though by combining the two scores, statistically significant improvements are achieved for almost all phonemes.

A different strategy for discrimination of Dutch velar fricative /x/ versus the velar plosive /k/ is studied in [12]. Latent Discriminant Analysis (LDA) over two different sets of features are explored and compared with previous approaches: aforementioned GOP scores and Weigelt algorithm. The latter is based on three simple measures that can be easily obtained: log root-mean-square (rms) energy, the derivative of log rms energy (*Rate of Rise* or ROR) and zero-crossing rate. Weigelt algorithm discriminate plosives from fricatives by using ROR values, based on the fact that the release of the burst of the plosives causes an abrupt rise in amplitude therefore yielding higher values compared with fricatives. On the other hand, LDA weights are assigned to each feature to find the linear combination of features which best separates the classes. Selecting the most relevant features turns out to give an advantage compared to other classifiers. The two LDA methods yielded the best performing scores followed by GOP and Weigelt.

Relying on labeled nonnative speech data usually leads to more flexible models with better performance than those trained with native speech when working on pronunciation assessment for specific combinations of L1 and L2. On the one hand, the set of common pronunciation errors tend to be particular for a given <L1, L2> pair. On the other hand, models trained with nonnative speech are better at capturing the subtle differences between the nonnative speech realizations that are considered acceptable versus the nonnative speech realizations that are considered mispronounced, in a similar way that an annotator would do. However, labeled nonnative speech databases are rarely available. Annotation of nonnative speech is an expensive and time-consuming task, and in some cases it is not feasible. Models trained with annotated nonnative speech usually have better performance than those trained with native data, though the cost of nonnative database collection and annotation is very high.

To conclude this section, the papers that set the starting point for the current thesis will be reviewed. They are the two latest works in one of the most important lines of investigation in the field of pronunciation assessment at phone level. Both use a Latin-American Spanish speech database that includes recordings from native and nonnative speakers, where nonnative speech is produced by speakers whose native language is American English.

The first of these works [13] is a continuation of [2], where log-likelihood and log-posterior scores were proposed as effective methods for assessing pronunciation. In [13], the same log-posterior method (Equation 1.2 and 1.3) is applied using models trained on native data and is used as baseline and compared with a proposed technique based on log-likelihood ratio (LLR). To compute LLRs, the phonetically labeled nonnative database is used to train two different *Gaussian mixture models* (GMMs) for each phone class: one model is trained with the "correct" native-like pronunciations of a phone, while the other model is trained with the "mispronounced" or nonnative pronunciations of the same phone. In the evaluation phase, for each phone segment q_i, a length-normalized log-likelihood ratio $LLR(q_i)$ score is computed for the phone segment by using the "mispronounced" and "correct" pronunciation models λ_M and λ_C respectively:

$$LLR(q_i) = \frac{1}{d} \sum_{t=t_0}^{t_0+d-1} [log\ p(y_t|q_i, \lambda_M) - log\ p(y_t|q_i, \lambda_C)] \tag{1.6}$$

The normalization by d allows the definition of unique thresholds for each phone class, independent of the length of the segments. A mispronunciation is detected when the LLR is above that threshold. The LLR method performed better than the posterior-based method for almost all phone classes, specially for those with the highest consistency across transcribers. These results are in line with the fact that models trained on nonnative speech data have better results than those trained on native speech, as it was previously mentioned.

Finally, the most recent work of the series [14] extends the research proposing two new methods that also explicitly model both mispronunciations and correct pronunciations by nonnative speakers. The LLR method developed in [13] is used as baseline and compared with the new modeling techniques.

The first proposed method also uses LLR based on GMMs trained on nonnative data (Equation 1.6). The difference resides in the way these GMMs are obtained. In the baseline approach, for each phone class two different GMMs are trained: one model is trained with the "correct" native-like pronuncations of a phone while the other is trained with the "mispronounced" or nonnative pronunciations of the same phone. In the new approach, the models for each class for a given phone are obtained by adaptation. The model to which they are adapted is trained using all the samples from a given phone, ignoring the class ("correct" or "mispronounced"). Bayesian adaptation [**ubm**] is used to adapt this class-independent GMM to all the "correctly" pronounced training examples for a phone and obtain the adapted "correct" model for that phone. An analogous process is followed to obtain the adapted "mispronounced" model. For these class-dependent GMMs both the means and the mixture weights are adapted to the class-specific data.

The second proposed method follows a discriminative approach. It uses the class-independent GMM of the previous experiment to create supervectors by adapting this GMM to each phone instance. The supervector for a certain phone instance is obtained by adapting the means and the mixture weights of the original GMM to the acoustic feature vector representing the phone. These supervectors are then used as features to train a linear SVM classifier. To evaluate test instances of a given phone, its supervector is calculated by adaptation of the class-independent GMM to the phone feature vector frames. Finally, the distance of that supervector to the SVM hyperplane is taken as the score for the phone.

The two methods presented in [14] show better results than the mispronunciation detection system based on LLR of independently trained GMMs developed in [13], which at the same time outperforms the standard method that uses phone log posteriors as scores [2]. This is one of the most important lines of research in pronunciation assessment at phone level, positioning the methods described in [14] among the best systems in the field. For that reason, both generative and discriminative models are used as baseline and as a starting point in the work here presented. A more detailed explanation of these systems will be provided in the next section.

2. METHOD

In this section we will review the way the pronunciation scoring system works and explain the concepts and techniques on which it is based. The diagram bellow shows the overall architecture of the system:

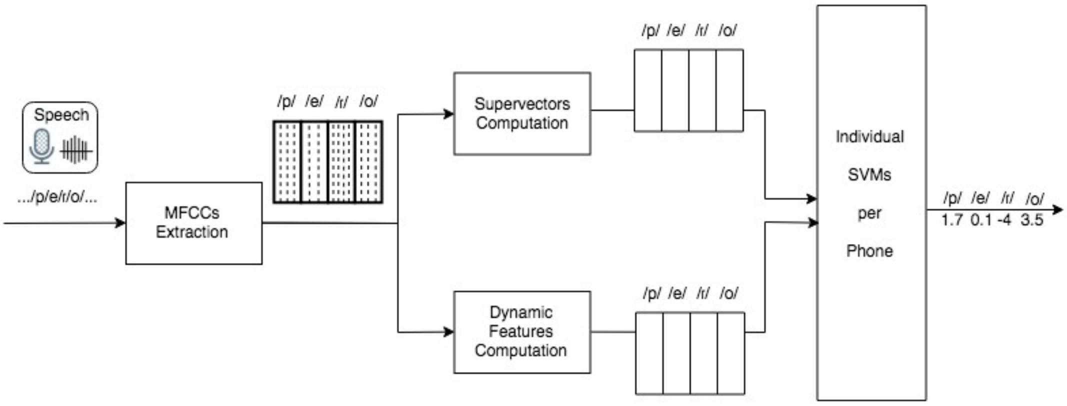


Fig. 2.1: General System Architecture

Given a test instance, the objective is to compute a score for each phone that is uttered in that instance. In order to do so, the flow goes as follows:

1. The first step is to calculate the audio features from the recordings. This is done in order to summarize the signal information in a measurable way through numeric values. The chosen features are the *Mel Frecuency Cepstral Coefficients* (MFCCs) because they are one of the most standard and widely-spread features in the speech processing field.

2. The MFCCs are then splitted in segments according to the phones to which they belong. Each segment is at the same time divided in frames, that represent short intervals where the audio signal is not supposed to be changing so much.

3. MFCCs of each instance of the phone present in the utterance are then used to compute the more complex features on which the SVMs operates. Supervectors are the base and proven to work features, while the Dynamic Features are the experimental features to be studied. There is an individual classifier for each of the phones.

4. Finally an individual score is obtained for each of the phones present in the utterance. Scores with positive sign corresponds to correctly pronounced instances of the phone while scores with negative sign corresponds to mispronunciations. The magnitude of the score is associated with how well or bad the phone is pronounced. The bigger the magnitude of the score, the better the pronunciation when the sign is positive and the worst the mispronunciation when the sign is negative.

2.1 Features

In this section we describe the chosen features for the current study. The features can be classified in two different categories: initial or base features and main features.

Initial or base features are computed directly from the speech signal, and they carry spectral density information. For the current work, the standard Mel Frequency Cepstral Coefficients were chosen.

The initial features are then used as base to compute more complex features: The main features. These are the features used to train the SVM and classify the instances. In the current work, two types of main features are studied: *Supervectors* which are obtained from an adaptation process of a Gaussian Mixture Model trained on the MFCCs and the *Dynamic Features*, which are computed as the coefficients of different interpolation methods of the MFCCs values through time.

In the following subsections a detailed description of each of the aforementioned features is provided.

2.1.1 Acoustic Features: MFCCs

The first step in order to build the system is to extract acoustic features from the speech data. The feature extraction process aims at approximating the linguistic content that is conveyed by the speech signal, by identifying its relevant aspects and discarding other unuseful properties like background noise, emotions, etc.

In this work we base our features in the standard and widely used Mel Frecuency Cepstral Coefficients (MFCCs). They were introduced by Davis and Mermelstein in 1980 and have been state-of-the-art ever since [15]. MFCC coefficients represent the spectral envelope of the speech signal on the Mel-frequency scale, which relates the perceived frequency of a pure tune to its actual measured frequency inspired by the human hearing.

The process to compute the MFCCs coefficients can be summarized into applying the following steps:

1. Preemphasis: Preemphasis aims at increasing the amplitude of high frequency bands while decreasing the amplitudes of lower bands by means of a high-pass filter. This stage is performed in order to balance the frequency spectrum since high frequencies usually have smaller magnitudes compared to lower ones.

2. Windowing: The MFCCs are computed in time intervals where the audio signal isn't supposed to be changing so much. Because of that reason the signal is divided in frames using a 25 ms duration window and 10 ms shift (standard values). A 25 ms duration window provides enough samples to get a reliable spectral estimate: 200 samples per frame at 8 kHz, and at the same time is short enough to minimize the signal changes. On the other hand, a 10 ms shift leads to a 15 ms overlap between consecutive frames, ensuring that the information between adjacent frames is also captured in the middle of another frame.

3. Discrete Fourier Transform: This is the first and main step in regard to spectral analysis. Frequency domain approaches are proven to be among the best options in speech classification tasks. The *spectral density*, which is the distribution of power into the frequency components composign the signal, is obtained for each frame by using the Discrete Fourier Transform technique.

4. Mel Filter-bank: At this stage, a perceptual scale of pitches inspired in human hearing called Mel scale is used. Because humans are much better at discerning small changes in pitch at low frequencies than they are at high frequencies, the Mel frequency scale is linear up to 1000Hz and logarithmic thereafter according to the following formula:

   $$M(f) = 1125 * ln(1 + \frac{f}{700}) \tag{2.1}$$

   The spectrum obtained in (3) is passed through a series of traingular filters uniformly spaced on the Mel scale to produce the so called filter-bank energies. The original values of these energies are replaced by their natural logarithm values, also motivated by human hearing: loudness is not perceived according to a linear but an exponential scale.

5. Cepstral Coefficients: The Discrete Cosine Transform (DCT) is applied to the logarithm of the filterbank energies to obtain the Cepstral Coefficients. In general, only the lower 12 Cepstral Coefficients are kept. The resulting features are called Mel Frequency Cepstral Coefficients. Additionally, a 13th MFCC computed as the sum of the energy in the frame is included, because it usually improves the performance in phone detection related tasks.

6. Deltas and Double Deltas: Finally, the speech also carries information in the dynamics, i.e, trajectories of the MFCC coefficients over time. Calculating the MFCC trajectories and appending them to the original MFCC vector usually increase the performance of the systems. Both Delta and Delta-Delta features are included, adding to a total of 39 features per frame.

2.1.2 Main Features: Supervectors

Supervectors, which have already been succesfully used in [16, 14] are one of the two features used in this work to train the SVM classifier. Supervectors for a given phone are obtained by extracting the means and weights of different Gaussian Mixture Models derived from the adaptation of a class-independently trained GMM to each of the instances.

So in order to go into more details of the supervectors features, a brief introduction to GMMs is provided along with an explanation of the adaptation process.

Gaussian Mixture Model

Gaussian Mixture Model is a parametric probability density function represented as a weighted sum of Gaussian component densities. GMMs are often used in biometric systems, specially in speaker recognition systems, due to their ability to represent a large class of sample distributions. One of the powerful attributes of the GMM is to form smooth approximations to arbitrarily shaped densities [17].

The densitiy function of a GMM is defined as:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \tag{2.2}$$

Each Gaussian density $\mathcal{N}(x|\mu_k, \Sigma_k)$ is called a component of the mixture and has its own mean μ_k and variance Σ_k. The parameters π_k are called mixing coefficients and they can be thought of as the prior probability of picking the k^{th} component [18].

In this work, GMMs are used to model the acoustic features mentioned in the previous section: MFCCs (13), plus deltas (13) and double deltas (13). So the features for our GMMs are 39-dimensional vectors corresponding to each frame of each instance of a given phone.

It is desirable, when possible, to transform the feature space into a normally distributed feature space because it leads to more robust results. For that reason, the acoustic features are standardized by removing the mean and scaling to unit variance before training the GMMs.

Universal Background Model Adaptation

As in the previous works of the current line of investigation [13, 14], the GMMs are derived by the adaptation of a *Universal Background Model* (UBM) [19], a technique that has been effective on significantly improving accuracy in the speaker recognition field.

The UBM is a large class-independent GMM intended to represent the class-independent distribution of the features. In a GMM-UBM system, the GMM is derived by adapting the parameters of the UBM to a vector of instances with a *Maximum a Posteriori* (MAP) estimation. This provides a tighter coupling between the generated models that produces better performance than using decoupled models such as class-dependent GMMs. In addition, it can be specially useful when just a small number of training instances is available.

The method described in [19] involves adapting weight, mean and variance of each mixture, though in the current work only weights and means are adapted because of the limited size of the dataset. For each gaussian k composing the GMM-UBM, the weight and mean parameters are updated according to the following equations:

$$\pi_k^{new} = [\alpha_k \pi_k^{'} + (1 - \alpha_k)\pi_k]\gamma \tag{2.3}$$

$$\mu_k^{new} = \alpha_k E_k(x) + (1 - \alpha_k)\mu_k \tag{2.4}$$

where π_k and μ_k are the original values of the weight and mean of the k$^{\text{th}}$ component of the GMM-UBM, $\pi_k^{'}$ and $E_k(x)$ are the estimated values for the weight and mean calculated on the vector of instances provided for the adaptation and the scale factor γ is computed over all the adapted mixture weights to ensure they sum to unity. α_k is an adaptation coefficient that controls the balance between old and new estimates, and depends on the value of a relevance factor input parameter r. A higher value of r produces a decrease in the value of α_k, thus adapting in a more conservative way by assigning more importance to the previous values. By contrary, a lower value of r produces an increment in the value of α_k, thus leading to a more aggresive adaptation strategy by assigning more importance to the new values.

Supervectors computation

In order to compute the supervectors for a given phone, which are then used to train the SVM classifier, the following steps are performed:

1. At first, a GMM-UBM is trained on the acoustic features (13 MFCCs plus deltas and double deltas) of all the available training instances for that phone, regardless of their class.
2. After that, an individual MAP adaptation process is carried out for each of the N training intances, thus generating N different GMMs.
3. Finally, for each of the training instances i the supervector is obtained by stacking the means and the weights of each of the gaussians composing the GMM obtained from the i$^{\text{th}}$ instance.

The supervector computation process for a given phone instance is summarized in the figure shown below:

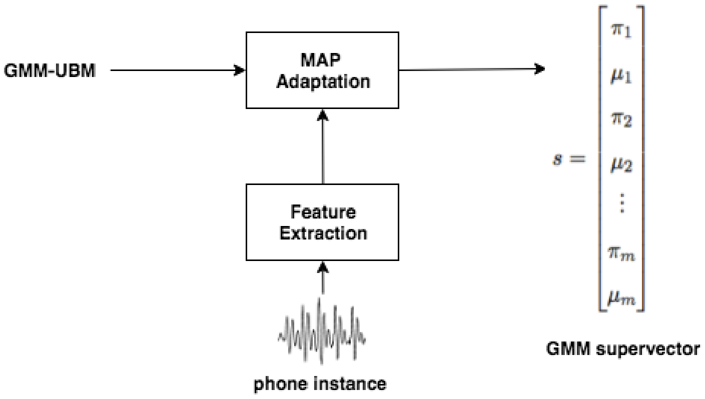


Fig. 2.2: Adapted figure from Campbell, Sturim, Reynolds, Solomonoff 2006 [16]. A MAP Adaptation of a GMM-UBM with m gaussians is performed using the acoustic features of a phone instance. The resulting supervector is made up of the stacked weights π_i and means μ_i $\forall\ 1 \leq i \leq m$, where each weight is a real value and each mean is a 39-dimensional vector.

As each derived GMM has the same number of gaussian components as the GMM-UBM, then the dimension of each supervector is: $39 * m + m$, where m is the number of gaussians of the GMM-UBM. The first term corresponds to the means while the second one to the weights.

Supervectors for testing are calculated following the same procedure as for training except for the first step: the same GMM-UBM obtained in training is used for the adaptation process of the testing instances.

2.2 Support Vector Machines

The chosen model for the discriminative analysis is the *Support Vector Machine* classifier, an approach for classification that was developed in the computer science community in the 1990s and that has grown in popularity since then. SVM has been successfully used in many speech processing fields, such as speaker recognition and pronunciation assessment.

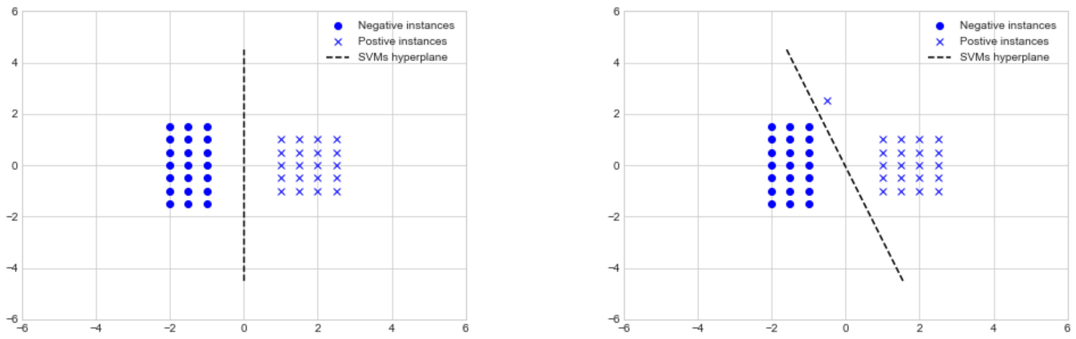
Given a p-dimensional feature space, an SVM constructs an hyperplane of dimension p-1 that separates the space in two halves, which allows to classify the instances according to their relative position to the hyperplane (above or bellow). The classification procedure is then just simply a case of determining which side a test observation falls on.

Among all the potentially infinite separating hyperplanes that can be generated from a given training set (it would be possible to slightly translate or rotate such plane without touching any training observation), SVM generates the hyperplane that has the largest distance to the nearest training data points of either of both classes. Such a classifier is known as Maximal Margin Classifier, and in most of the cases generalizes better to unobserved data than any of the other possible hyperplanes.

One of the key features of the Max Margin Classifier is that the location of the hyperplane only depends on the support vectors, which are the training observations that lie directly on the margins' boundaries.

2.2.1 Soft Margin

One of the problems with Max Margin Classifiers is that they can be very sensitive to the addition of new training observations: The hyperplane may be adjusted substantially to force that instance to lie in the right side of the margin or hyerplane, thus leading to overfitting while reducing the value of the margin (Figure 2.3). Moreover, many datasets are not perfectly separable via a linear hyperplane at all. Because of these reasons, the requirement that a separating hyperplane will perfectly separate every training observation on the correct side of the line according to its class label can be relaxed by introducing a new concept developed in [20]: the Soft Margin.



(a) An initial training set is optimally divided by the Max Margin Hyperplane of an SVM

(b) The additon of a single instance substantially modifies the position of the hyperplane.

Fig. 2.3: Margin adjustment when inserting a new training instance

A Support Vector Classifier or Soft Margin Classifier allows some observations to be on the incorrect side of the margin or even on the incorrect side of the hyperplane, thus providing a ”soft” separation. The soft margin method will choose a hyperplane that splits the example as cleanly as possible while still maximizing the distance to the nearest training instances.

Given a labeled dataset D of n points of the form: $D = \{\ (x_i, y_i)\ |\ x_i \in R^p,\ y_i \in \{-1, 1\}\ \}$, the problem can be formulated as:

$$\min_{\omega,\xi} \left\{ \frac{1}{2}\|w\|^2 + C\sum_{i=i}^{n} \xi_i \right\} \tag{2.5}$$

Subject to (for any $i = 1 \ldots n$):

$$y_i * (\omega^t \cdot x_i + b\ ) \geq 1 - \xi_i,\ \xi_i \geq 0 \tag{2.6}$$

Each x_i is a p-dimensional real number vector. The hyperplane $\omega^t \cdot x + b = 0$ splits the instances of both classes in two. The condition $y_i * (\omega^t \cdot x_i - b) \geq 1$ for any instance x_i is known as the margin condition, and ensures at least a margin on each side of:

$$\gamma = \frac{|\omega^t \cdot x_i + b|}{\|\omega\|} = \frac{1}{\|\omega\|} \tag{2.7}$$

Finally, slack variables ξ_i allow the individual observations to be on the wrong side of the margin or hyperplane, while C is an input parameter that determines the cost of the errors, thus establishing a trade off between a large margin and the error penalty.

Different values of C were tested at the beginning of the current work using at first an exponential scale: $10^{-i} \dots 10^{i}$ with $i = 4$, an after that reducing the search interval to $[-1 \dots 1]$. The best results, however, were obtained when using a default value suggested in the docs of SVM-Light, an implementation of SVMs in C language: $\frac{1}{avg(x*x)}$.

To conclude this section, it is worth mentioning that feature scaling is an important step to be performed before training the linear SVM classifier to avoid the domination of the features with highest values when calculating the hyperplane. For this reason, like in the GMM case, both supervectors and dynamic features are standardized by removing the mean and scaling to unit variance.

2.3 Legendre Polynomials

As it was said before, *MFCCs* are the acoustic features chosen for this work. Even though these raw features extracted from each frame composing the utterances of a phones would be enough to train a classifier that separates correctly pronounced and mispronounced phones, a different strategy can be taken. The features of all the frames of a particular utterance can be gathered together in order to summarize the general properties of the utterance. This is the purpose of using Legendre Polynomials when generating the features for the classifier. This technique has been used successfully in speaker recognition systems [22].

Legendre functions are the solutions to *Legendre's differential equation*:

$$(1 - x^2)y''(x) - 2xy'(x) + n(n+1)y(x) = 0, \; for - 1 \leq x \leq 1 \tag{2.8}$$

The solution for each particular $n = 0, 1, 2 \dots \in \mathbb{N}$ is a polynomial of degree n: $P_n(x)$. These solutions are well known for each $n \in \mathbb{N}$, and they can even be generated recursively. The first few Legendre Polynomials are:

$P_0(x) = 1$

$P_1(x) = x$

$P_2(x) = \frac{1}{2}(3x^2 - 1)$

$P_3(x) = \frac{1}{2}(5x^3 - 3x)$

$P_4(x) = \frac{1}{8}(35x^4 - 30x^2 + 3)$

Legendre Polynomials are orthogonal with respect to the *L2 norm* on the interval $-1 \leq x \leq 1$:

$$\int_{-1}^{1} P_n(x) * P_m(x)dx = 0,\ m \neq n \tag{2.9}$$

Moreover, in the interval $-1 \leq x \leq 1$ any function f can be represented as sum of Legendre Polynomials, leading to the Fourier-Legendre Series:

$$f(x) = \sum_{0}^{\infty} a_n P_n(x) \tag{2.10}$$

So an infinite terms of Legendre Polynomials over the interval $-1 \leq x \leq 1$ can be used to approximate any function. Each coefficient models a particular aspect of the curve: a_0 is interpreted as the *mean* of the segment, a_1 is the slope, a_2 gives information about the curvature of the segment and the next coefficients model the fine details. A particular cutoff c is chosen in order to limit the number of coefficients of the Legendre Polynomials to be picked in order to approximate the function in a reasonable manner while avoiding the complex details that may lead to overfitting.

2.3.1 Lasso Regression

Given the time instants $X = [x_1, x_2 \ldots x_t]$ and their respective values of a particular MFCC coefficient at those instants: $Y = [y_1, y_2 \ldots y_t]$, the problem of finding the coefficients of *Legendre Polynomial* of degree k (for a previously defined k) that best approximates the values can be formulated in terms of a *Least Squares* minimization problem:

$$\min_{C} \|AC - Y\|^2 \tag{2.11}$$

where the solution of the system C is a $(k+1)$ vector that respresents the coefficients of the *Legendre Polynomial* of degree k that best approximates the points. A is a t x (k+1) matrix, where each column i represents the result of computing *Legendre's* P_i polynomial (2.3) for each of the values $[x_1, x_2, \ldots x_t]$, also known as the *Legendre-Vandermonde* matrix:

$$A = \begin{pmatrix} P_0(x_1) & P_1(x_1) & \cdots & P_k(x_1) \\ P_0(x_2) & P_1(x_2) & \cdots & P_k(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ P_0(x_t) & P_1(x_t) & \cdots & P_k(x_t) \end{pmatrix} \tag{2.12}$$

When modeling a particular event is preferable to pick the simplest hypothesis that best explains the observations, because simpler hypothesis usually generalize better to new observations than more complex hypotheses. As it was said before, the higher the degree of the *Legendre Polynomial*, the more subtle details it models. In order to achieve the right balance between decreasing the least squares error and keeping the hypothesis simple two different measures are taken. On the one hand the degree of the polynomial k is limited up to a few coefficients. On the other hand, a regularization technique of the *Least Squares* family's problems is applied.

The chosen regularization technique is called *Lasso Regression*, and it modifies the original minimization problem by adding an additional term that penalizes the L1 norm of the solution:

$$\min_{C} \|AC - Y\|^2 + \lambda \|C\|_1 \tag{2.13}$$

Regularization terms leads to solutions where more responsibility is assigned to the coefficients that contributes the most in lowering the square errors of the error term while shrinking the coefficients with low contribution to the square error term.

There exists other alternatives to *Lasso* with regard to regularization. A widely-used one is *Tikhonov Regularization*, also known as *Ridge Regression*, where the squared of the *L2 norm:* $\|C\|_2^2$ is used as regularization term.

In particular, the choice of using the *L1 norm:* $\|C\|_1$ instead of the standard square of the *L2 norm:* $\|C\|_2^2$ leads to an interesting and useful property. In *Ridge Regression*, as the penalty is increased, all parameters are reduced while still remaining non-zero, while in *Lasso* increasing the penalty will cause more and more of the parameters to be driven to zero. Thus *Lasso* automatically selects more relevant features and discard the others whereas *Ridge Regression* never fully discard any features.

An intuitive explanation of why *Lasso* behaves this way can be sketch out in terms of the shape of the sharped-edge region determined by the constraints of the regularization term. The level curves imposed by the least squares term will most likely hit the corners of that sharped-edge region, driving the values of some of the coefficients to zero.

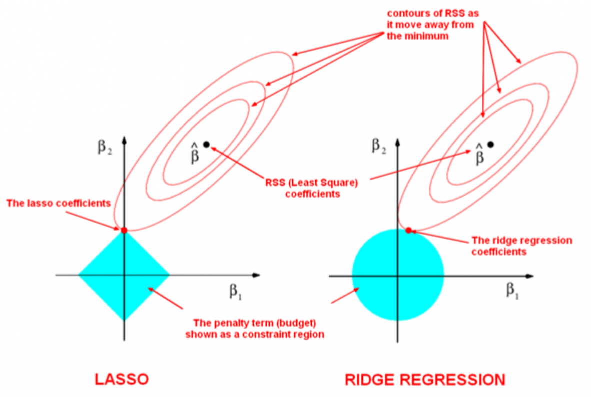

contours of RSS as it move away from the minimum
β₂
β̂
RSS (Least Square) coefficients
The lasso coefficients
β₁
The penalty term (budget) shown as a constraint region
β₂
β̂
The ridge regression coefficients
β₁
LASSO
RIDGE REGRESSION

Fig. 2.4: Intersection of the level curves of the *Residual Sum of Squares RSS* with the regions imposed by the regularization terms. Unlike *Ridge Regression*, in *Lasso Regression* the instersection is most likely to happen at a corner of the sharped-edge region determined by the *L1-norm* regularization constraint, where the value of some of the coefficients is equal to 0.

2.4 Discrete Cosine Transform

An alternative to Legendre Polynomials to capture the general aspects of the utterance in order to summarize the information carried out by the *MFCCs* of each frame is the *Discrete Cosine Transform*. Variations on the values of MFCCs through time may be explained using periodic functions instead of polynomial ones, making *DCT* a more suitable technique for the task. *DCT* is used in many processes related with science and engineering such as lossy compression of audio (e.g. *MP3*) and images (e.g. *JPEG*), which are among the most popular formats in their respective fields. This technique has also been used to approximate prosodic features in speaker verification tasks [23].

DCT belongs to the family of the Fourier analysis. As a member of the family, it provides a way to approximate a general function by sums of simpler trigonometric functions. These transformations map a function to a set of coefficient of basis functions, where the basis functions are sinusoidal and are therefore strongly localized in the frecuency spectrum. In particular, *DCT* expresses a finite sequence of data points in terms of a sum of *cosine* functions oscillating at different frequencies. Unlike the *Fourier Transform* that expresses the sequence in terms of both sine and cosine functions, thus needing a complex number to represent the coefficient of each frecuency, it only uses real numbers to output the coefficient of each frecuency.

There exists different variants of the *DCT*, being the type-II the most common and the one that it is used in the current work:

$$X_k = \sum_{n=0}^{N-1} x_n cos[\frac{\pi}{N}(n+\frac{1}{2})k], \; for \; k = [0, 1, \ldots, N-1] \tag{2.14}$$

where N is the number of the extracted samples of the signal to be decomposed. In this way, k coefficients are obtained corresponding to each of the frequencies from 0 to $N-1$. Each coefficient is obtained from the sum of the individual contributions of the samples to the k^{th} frequency.

The evolution of each of the *MFCCs* along the frames composing the phone utterance is then approximated by a sum of cosine functions obtained through *DCT*, and the coefficients of the cosine functions will be used as features to train the *SVM* classifier.

2.5 Performance Measures

In order to measure the performance of the different systems the *Equal Error Rate* metric is chosen. As its name suggests, the *EER* prioritizes in an equal manner the *false acceptance rate* and the *false rejection rate*. This metric was already used in the previous works of the current line of investigation related to pronunciation assessment at phone level [13, 14], so the same approach was taken in the present work.

The *EER* can be easily explained by using other important concept related to performance measures of machine learning classifiers: the *Receiving Operating Characteristic* curve. So in first place a briefly explanation of the *ROC* curve will be given, and after that the explanation of the *EER* will be introduced.

2.5.1 Receiving Operating Characteristic

The *Receiving Operating Characteristic* (ROC) curve of a binary classifier is a plot that resumes information of how well a model can correctly detect positive instances at expenses of raising false alarms after being tested against a data set.

When testing a given classifier on a dataset the results can be resumed in a particular table known as *confusion matrix*:

	Actual Class		
Predicted Class	*Actual Positive*	*Actual Negative*	*Total Count*
Predicted Positive	True Positive (TP)	False Positive (FP)	Total Positives (P)
Predicted Negative	False Negative (FN)	True Negative (TN)	Total Negatives (N)

True Positive Rate is calculated as the number of instances predicted as *positive* which actually are *positive* over the total number of *positives* while *False Positive Rate* is calculated as the number of instances predicted as *positive* which actually are negative:

$$TPR = \frac{TP}{P} \quad (2.15) \qquad FPR = \frac{FP}{N} \quad (2.16)$$

In a *ROC* curve the *TPR* is plotted in function of the *FPR* for different cut-off points. Each point of the curve represents a *FPR*,*FPR* pair corresponding to a particular decision threshold. A test with perfect discrimination (no overlap between the positives and negatives distributions) would have a *ROC* curve that passes through the top left corner. So the closer the *ROC* curve is to the upper left corner, the higher the overall accuracy of the test.

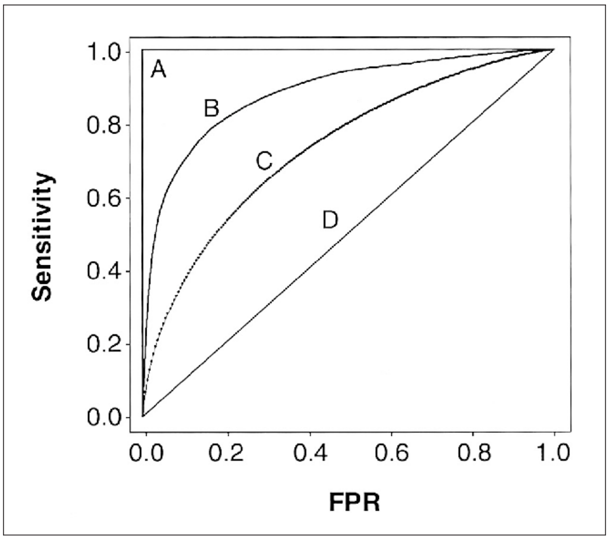


Fig. 2.5: ROC curve example. The closer the curve to the upper left corner, the higher the accuracy of the system

2.5.2 Equal Error Rate

TPR (Eq. 2.15) is also called *sensitivity* in the sense that measures the capacity of the system of detecting positive instances and classifying them correctly as positives. At the same time, *specificity* is a measure of the capacity of the system of avoid raising false alarms, when classifiyng instances that actually belongs to the negative class. Or, in other words, the capacity of the system of detecting negative instances and classifying them correctly as negatives. *FPR* (Eq. 2.16) is actually the complement of *specificity*:

$$specificity = \frac{TN}{N} = \frac{TN}{TN+FP} = 1 - \frac{FP}{TN+FP} = 1 - FPR \qquad (2.17)$$

Equal Error Rate (EER) is the value on which *sensitivity* is equal to *specificity.* At this point the system classifies correctly positive instances as well as it classifies negative instances, thus prioritizing both classes in the same way. It can be calculated easily as the point of intersection between the ROC curve and the *specificity* curve: $1 - FPR$

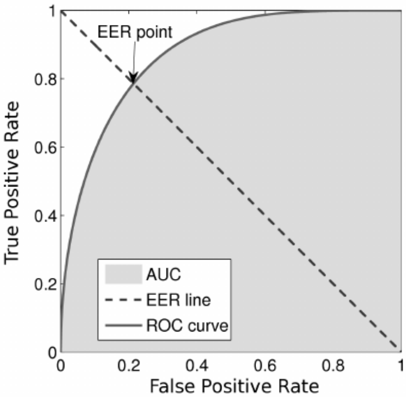


Fig. 2.6: EER is equal to the value of the intersection between the ROC curve and the Sensitivty curve as function of FPR

EER is a proper performance measure for the task of pronunciation assessment at phone level because both correctly pronounced and mispronounced classes are equally important. In other tasks, such as *Bank Fraud* or *Spam Detection*, it may be essential to strictly prioritize one class over the other. In the current task however it is reasonable to assign the same priority to both *sensitivity* and *specificity.* Anyway, it is worth mentioning that during language learning a student may get discouraged if she pronounces an utterance correctly and the system classifies her utterance as incorrect. A threshold can be chosen in order to ensure that a mispronunciation flag is raised only when the resulting score is above that threshold. The system could have made a mistake for instances labeled as incorrect with values below the threshold, so a conservative decision of not raising a mispronunciation flag can be taken to avoid discouraging the students.

2.6 Statistical Significance Techniques

One of the main objectives of this work is to find out whether or not there is a gain in performance when combining the features derived from *GMM Adaptation* with features that carry information of the evolution through time of the phonetic features. The chosen approach to achieve that is to carry out a sample based study and analyse if the finding generalizes to the parent population. A subset of the samples is reserved with this purpose in order to test the models and generate statistics of its performance. The statistic to be analysed in this case is the *EER*, as it was mentioned in the previous *Performance Measures* section.

But how can be ensured that the sample based study generalizes to the parent population? A summary statistic may fluctuate from sample set to sample set and it would be desirable to somehow quantify the confidence regarding to the generalization of the obtained statistics to the parent population.

In order to do so, two different techiques are used in this work: *Bootstrapping* and *McNemar's Test*. They both focus on different aspects and thus allowing a complementary analysis.

2.6.1 Bootstrapping

[24] *Bootstrapping* is a general intuitive method applicable to almost any kind of sample statistic and can be understood without much theoretical knowledge of sampling distributions. It was introduced in 1979 by B. Efron and it has been widely used in numerous areas since then.

Suppose it were possible to draw repeated samples of the same size from the population of interest a large number of times. Then a fairly good idea about the sampling distribution of a particular statistic from the collection of its values arising from these repeated samples could be obtained. This method doesn't make sense as it would be too expensive and defeat the purpose of a sample study, that is gather information in the cheapest and effortless way. The idea behind *Bootstrapping* is to use the data of a sample study at hand as a "surrogate population" for the purpose of approximating the sampling distribution of a statistic. This is achieved by resampling with replacement from the sample data at hand and create a large number of alternative or duplicated sample sets known as bootstrap samples. The sample summary is then computed on each of the bootstrap samples (usually between 1-10 thousand). A histogram of the set of these computed values is referred to as the bootstrap distribution of the statistic.

For the sake of understanding the maths behind this technique, suppose a population parameter θ is the target of the study. A random sample of size n yields the data $(X_1, X_2, \dots X_n)$, and the corresponding sample statistic computed from this dataset is $\hat{\theta}$. *Central Limit Theorem* states that for most sample statistics the sampling distribution of $\hat{\theta}$ for large n ($n \geq 30$ is generally accepted as large sample size) is bell shaped with center θ and standard deviation $a/\sqrt{(}n)$, where the positive number a depends on the population and the type of statistic $\hat{\theta}$. Often, there are serious technical complexities in approximating the required standard deviation from the data. *Bootstrapping* offers a bypass to this situation. Let $\hat{\theta}_B$ stand for a random quantity which represents the same statistic computed on a bootstrap sample drawn out of $(X_1, X_2, \ldots X_n)$. In the limit as $n \to \infty$, the sampling distribution of $\hat{\theta}_B$ is also bell shaped with $\hat{\theta}$ as the center and the same standard deviation $a/\sqrt(n)$. Then, bootstrap distribution of $\hat{\theta}_B - \hat{\theta}$ approximates fairy well the sampling distribution of $\hat{\theta} - \theta$.

It the current work, *Bootstrapping* technique is used along with Confidence Intervals in order to determine if the *SVM* trained with the combined features performs better than the one trained only with features derived from the *GMM* adaptation. Bootstrap samples are extracted from the *test* data in order to generate a distribution of *EER* for both types of classifiers. After that, a 95% confidence interval is computed for each distribution by finding the interval $[\hat{\theta}_{B1}, \hat{\theta}_{B2}]$ that enclose the 95% of the instances composing each distribution. If it's true that the *SVM* trained with the combined features performs better than the *SVM* trained with a single feature source, then its confidence interval should be shifted to the left with respect to the confidence interval of the latter system. The bigger the overlap between both intervals, the lesser the evidence of the results coming from different distributions and thus resulting in less significant results.

2.6.2 McNemar's Test

McNemar's Test is used as an alternative to determine whether the *SVM* based on the combined features and the *SVM* based only on features derived from the *GMM* adaptation are significantly different.

[25] Given a set of n test samples labeled by two systems, A_1 and A_2 the results can be resumed in:

	Correctly Predicted by A_2	Incorrectly Predicted by A_2
Correctly Predicted by A_1	$N_{0,0}$	$N_{0,1}$
Incorrectly Predicted by A_1	$N_{1,0}$	$N_{1,1}$

- $N_{0,0}$ is the number of samples which A_1 classifies correctly and A_2 classifies correctly
- $N_{0,1}$ is the number of samples which A_1 classifies correctly and A_2 classifies incorrectly
- $N_{1,0}$ is the number of samples which A_1 classifies incorrectly and A_2 classifies correctly
- $N_{1,1}$ is the number of samples which A_1 classifies incorrectly and A_2 classifies incorrectly

Let's denote p_1 the error rate of the system A_1 and p_2 the error rate of the system A_2. Let's also denote $q_{0,0}$ as the probability of (A_1 correct $\wedge$ A_2 correct), $q_{0,1}$ the probability of (A_1 correct $\wedge$ A_2 incorrect), etc. Hence:

$$p_1 = q_{1,0} + q_{1,1} \qquad (2.18) \qquad\qquad p_2 = q_{0,1} + q_{1,1} \qquad (2.19)$$

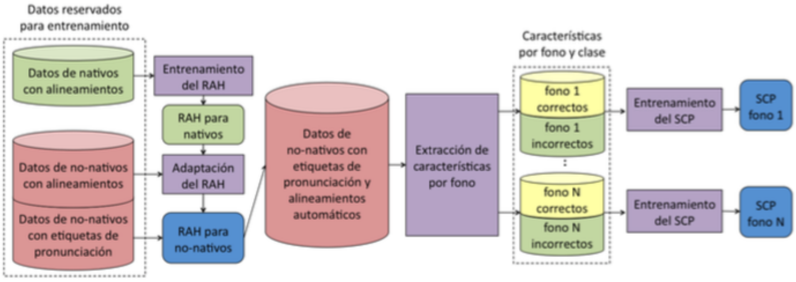
The *null hypothesis* $p1 = p2$ is equivalent to $q_{1,0} = q_{0,1}$. Defining $q = q_{1,0}/(q_{1,0} + q_{0,1})$ the *null hypothesis* becomes $q = \frac{1}{2}$. The probability q is the probability that given that only one of the systems made an error, it was system A_1 that did it. Under the *null hypothesis*, $N_{1,0}$ is distributed as a Binomial Distribution $B(n, 1/2)$, where n is the observed number of samples for which only one system made an error: $N_{1,0} + N_{0,1}$. At this point, a two-tail test can be applied to obtain a *p-value.*

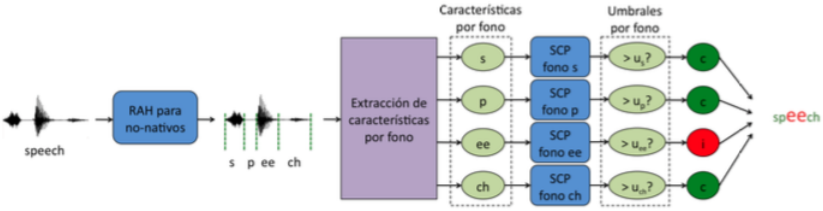
In conclusion *McNemar's Test* is based on cross information by comparing the results of evaluating both systems on exactly the same sample. *Bootstrapping* differs on this point because each sample is evaluated independently for each system. So a complementary analysis using both different techniques can be carried out in order to determine the statistic significance of the results.

3. EXPERIMENTAL DESIGN

3.1 Architecture Overview

(Los siguientes diagramas fueron extrados del toolkit description. La idea es armar unos nuevos propios con descripciones en ingls)


Datos reservados para entrenamiento
Datos de nativos con alineamientos
Datos de no-nativos con alineamientos
Datos de no-nativos con etiquetas de pronunciación
Entrenamiento del RAH
RAH para nativos
Adaptación del RAH
RAH para no-nativos
Datos de no-nativos con etiquetas de pronunciación y alineamientos automáticos
Extracción de características por fono
Características por fono y clase
fono 1 correctos
fono 1 incorrectos
fono N correctos
fono N incorrectos
Entrenamiento del SCP
Entrenamiento del SCP
SCP fono 1
SCP fono N



3.2 Database Description

3.3 Development and Test Sets Definition

3.4 GMMs Experiments

3.4.1 Independently-trained models

3.4.2 Adapted models

3.5 SVM Experiments

3.5.1 Supervectors

3.5.2 Legendre

- Modeling time dependencies

- Difference with Supervectors obtained from GMM adaptation

3.5.3 DCT

3.5.4 Features Combination

3.5.5 Score Combination

4. RESULTS

Experiments in the development set were carried out using 4-Fold Cross Validation. Models were trained using data from three folds and tested on the remaining one, rotating the procedure four times over the four partitions. There were no common speakers across any of the partitions. When reporting results for a given system, the errors obtained for each of the four folds are pooled to obtain averaged performance results on the development set.

Additionally, a *heldout* set with the same number of instances of each of the folds in the *development* set was kept separately to test the final models. There were also no common speakers between the *heldout* set and any of the folds of the development set. When reporting results for a given system using the heldout data, the errors are obtained in a straightforward way by testing the model trained on the development set against the heldout data.

In all the experiments, an individual system is trained for each different phoneme.

4.1 Baseline Experiments

The current work uses the same database used in [14] with the exception that data is split differently: In [14], a four-way jackknifing procedure containing all the instances is used to train and test the models, while in the current work a heldout set was kept separately of the development set. The number of Gaussian Mixtures composing each GMM is proportional to the number of training instances for that phone. In [14], a proportion of 1/25 computed beforehand over the total number of instances was used. In the current work, only 60% of the total instances for a given phoneme is used to train each model, so the proportion of Mixture Components per number of instances was set to 1/15 to preserve the number of Gaussians of each model.

The initial experiments were carried out in order to replicate the results obtained in [14] (task that should be possible because of using the same database). These systems could then be used as baselines and as starting points for the next systems. In this previous work three different systems are compared:

1. Independenty Trained GMMs

2. Adapted GMMs

3. SVMs trained on supervectors

Results presented in [14] had shown that the adaptation of the class-independent GMM produced an overall weighted EER relative reduction of 3.5% relative to the results obtained with independently trained GMMs.

Two initialization approaches to be performed before training the GMMs were explored in the current work: *KMeans* and *Agglomerative Clustering.* The objective was to see if additional gain could be obtain by performing an additional initialization before training the GMMs. Both approaches were tested in the development set producing similar results. *Agglomerative Clustering*, however, requires $O(n^3)$ steps to be computed (where n is equal to the number of instances), making it very impractical as an initialization step. For both reasons, *KMeans* was chosen as the representative initialization method.

Results of the Adapted GMM initialized with KMeans experiment performed in the development set generated a 3% relative reduction of the overall weighted EER compared with results of Independently Trained GMMs initialized with KMeans. These results, as it was expected, are strongly correlated with the results obtained in [14] and thus determining a succesful replication of the initial experiment. Similar results were obtained when comparing the Adapted GMMs with the Independently Trained GMMs but using *Agglomerative Clustering* initialization.

4.1.1 GMM Adapt K-Means vs Supervectors

The next step was to compare the Adapted GMMs systems with the SVMs trained on supervectors. Results presented in [14] shown that the SVMs systems produced an overall weighted EER relative reduction of 1.3% with respect to the Adapted GMMs.

Below is shown the comparison between the results obtained with the Adapted GMMs initialized with KMeans and the results obtained with the SVMs system in the development set. The UBM-GMMs used to generate the supervector features for the SVM are trained using no initialization (such as KMeans) because it leads to better results.

The results are sorted in descending order according to their Kappa Coefficient. It seems to be some correlation between the phonemes with high kappa and those who achieved the best results. This seems reasonable, because it may exist clearer differences between correctly and mispronounced utterances of the phonemes that produced better agreement over the transcribers. These differences may lead the classifiers to perform better. Among the eight phonemes with higher Kappa values (K $>$ 0.4, which implies moderate agreement): /β/, /δ/, /γ/, /b/,

/w/, /m/, /i/, /s/, only /s/ exceeds an EER of 0.3. Results show that the SVMs system produced an overall weighted EER relative reduction of 2.5% with respect to the Adapted GMMs with K-Means initialization, which is in line with the results obtained in [14].

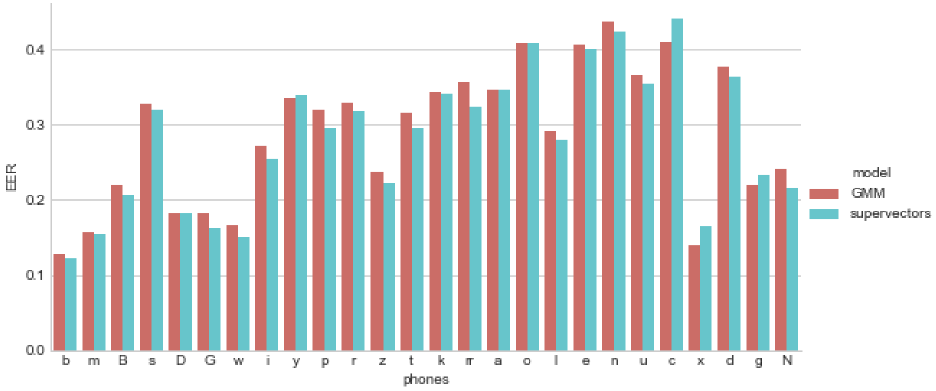


Fig. 4.1: Comparison of EER between LLR from Adapted GMMs with K-Means initialization and SVM trained on Supervectors for all phones in the development set, sorted descendently by Kappa values.

The same experiment was carried out in the heldout set leading to the results shown below. Both models were trained using all the instances in the development set. Both systems had a very similar performance for each phoneme compared with the results obtained in the development set, from which one can deduce that both classifiers generalize well to unobserved data. This time, however, it was the Adapted GMM with K-Means initialization which produced an overall weighted EER relative reduction of 0.8% with respect to the SVMs trained on supervectors. Even though the expected result would have been that the SVM performed slightly better than the GMM as in [14] and as it occurred in the development set, the degradation is less than 1% so it is still a reasonable result. In all the experiments carried out to compare the SVMs trained on supervectors and the Adapted GMMs, both classifiers shown a very similar performance on each phoneme.

After the successful replication of the initial experiments, a validated SVM model based on supervectors was obtained. This classifier is used as baseline and starting point for the next experiments, whose objective is to analyse if an aditional gain in the performance could be achieved by combining the supervectors features with features that carry temporal information.

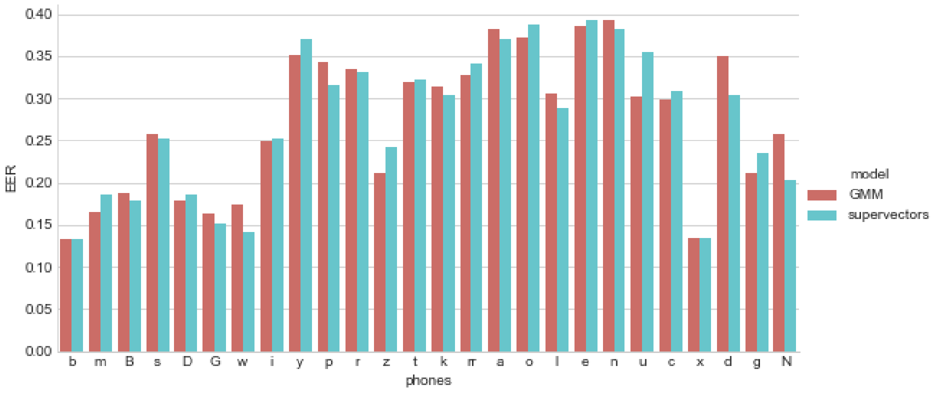


Fig. 4.2: Comparison of EER between LLR from Adapted GMMs with K-Means initialization and SVM trained on Supervectors for all phones in the heldout set, sorted descendently by Kappa values.

4.2 Tunning Experiments

Tunning experiments are carried out in order to compare both alternative dynamic features that capture the temporal dependencies on each instance. These features are used as training instances of an SVM classifier.

Features are first analysed in isolation to determine the best possible configuration for each technique. After that, the features generated with these configurations are combined with the supervectors and compared to each other.

Tunning is again performed in the deveopment set using 4-Fold Cross Validation. Results are calculated only for the phonemes with high Kappa values taking the average of all the values. Only these phonemes are considered in the process because they are the most reliable ones.

The optimal parameters to be obtained at this point: mainly the number of coefficients to be used of both Legendre and DCT, are shared by all the phonemes. This helps to keep the models simple by reducing the number of phone-dependent configuration parameters.

4.2.1 Legendre Best System

Tunning the best Legendre configuration involves the analysis of three different parameters:

- Time axis normalization between -1 and 1
- Inclusion of duration
- Lasso-Regression normalization

The objective of the Legendre tunning experiments is to find the optimum degree of the Legendre Polynomials along with the decision of normalizing or not the time axis and applying or not Lasso Regularization over the coefficients. Degree was varied between 0 and 6, because polynomials with higher degrees run the risk of overfit to training instances by modeling very specific not generalizable details.

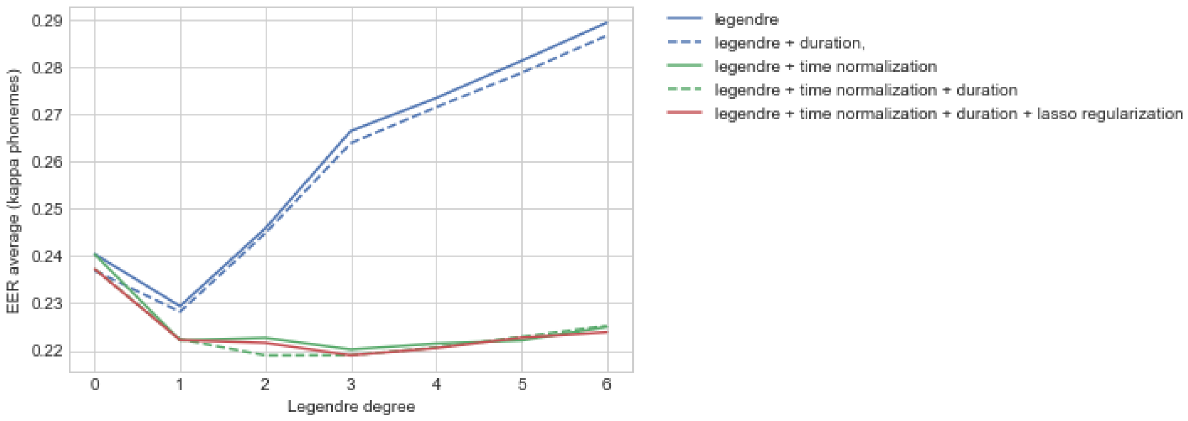


Fig. 4.3: EER average over phonemes with high Kappa for different degrees of Legendre Polynomials in the training set. Additional configurations are also studied along with the degree and represented by different curves: time normalization, appending the duration and applying Lasso Regression

Time normalization before approximating the MFCCs turned out to be an essential property for the Legendre Polynomials, evidenced by a considerable reduction of the EER average (gap between the blue and green curves). At the same time, appending the duration to the features produced a slight improvement in the performance. Lasso Regularization, however, doesn't contribute to reduce the average EER. Results show that the optimal configuration is to model the dynamics of the temporal dependencies using Legendre Polynomials of degree 2, normalizing the time axis and appending the duration of the phone utterance to the features (green dotted line).

4.2.2 DCT Best System and Comparison Between Features

Analogous to Legendre, the objective of the DCT tunning experiments is to find the optimum number of coefficients of DCT to approximate the dynamics of the temporal dependencies. In the DCT case, the only additional parameter to be determined is whether or not appending the duration to the DCT coefficients.

As in the Legendre optimal configuration, DCT tunning experiments also showed that appending the duration to the coefficients produced a slight reduction in the averaged EER (around

1% relative).

A comparative plot between the variation of the EER as function of the number of coefficients is shown below. The additional parameters were fixed according to their optimal configuration for both techinques: Normalizing the time axis and appending the duration for Legendre (without Lasso Regularization), and appending the duration for DCT. The degree of the Legendre coefficients is transformed to the number of coefficients in the *x-axis* to allow the comparison: 1 coefficient for Legendre Polynomials of degree 0, 2 coefficients for polynomials of degree 1, etc.

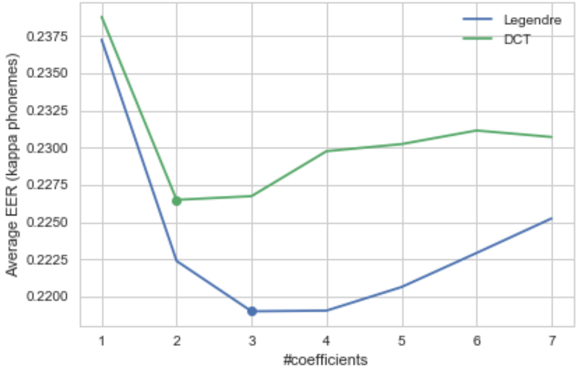


Fig. 4.4: EER average over phonemes with high Kappa as function of the number of coefficients for both Legendre Polynomials and DCT in the training set.

A dot is placed on each curve showing the optimal number of coefficients for both techniques. As it was known from 4.3, the optimal number of coefficients for the Legendre Polynomials is 3 (Polynomials of degree 2). Polynomials with 4 coefficients performs almost as well as polynomials with 3 coefficients, but it is always preferable to keep the model as simple as possible. On the other hand, the optimal number of coefficients for DCT is 2.

The results show that when the analysis is done in isolation (in absence of the supervectors features), Legendre Polynomials lead to a better performance, generating a relative reduction of the average EER of 3.3% compared with DCT.

4.2.3 Comparing the Combination of Dynamic Features with Supervectors

Even though the SVM based solely on Legendre performed better than the SVM based solely on DCT, a new experiment is carried out in order to determine which feature integrates better with the supervectors features to produce the best results.

Two types of combination are studied in the current thesis: a Features Combination and a

Score Combination. In the former, features from both sources are mixed in some proportion and used to train a single SVM, and the results are obtained by running this SVM on the test data. In the latter, an independent SVM is trained for each feature source and run against the test data and then the results are combined in some proportion to generate the final results.

The experiment is scoped again to phonemes with high Kappa values. In this case, however, the proportions to be used when combining the features and the scores are kept phoneme-dependent. The objective is thus to compute for each phoneme two optimal proportions: one for the features combinations and the other one for the score combination.

To make the experiment, proportions of the dynamic features with respect to supervectors features were varied between 0 and 1 with steps of 0.1. In other words, the following proportions were tested: [0.0, 0.1, 0.2 . . . 1.0]. Both ends corresponds to use only a feature source and not using the other source at all. A proportion of 0.0 of the dynamic features with respect to supervectors leads to not using the dynamic features at all while a proportion of 1.0 leads to not using the supervectors features at all. In the case of the Features Combination, an extra exploration is done in the interval [0.0 - 0.1], because achieving an optimal feature combination may have required a more refined granularity in the proportions. No significant gains, however, were observed when zooming in in the interval [0.0 - 0.1].

For the Features Combination case, the optimal proportion for the majority of the phonemes for both Legendre and DCT approaches is 0.1. There are also some phonemes such as /γ/ and /b/ where the best proprtions is 0.0. That is, that there is no proportion of the dynamic features that produces an improvement when combined with the supervector features.

On the other hand, for the Score Combination case, the optimal proportions for the phonemes are more scattered along the whole interval [0.0 - 1.0] for both Legendre and DCT. Again, there are also some phonemes where the optimal proportions is 0.0, but they are considerable fewer than in the Features Combination case.

	Legendre + Supervectors	DCT + Supervectors
EER Features Combination (Avg)	0.188	0.187
EER Score Combination (Avg)	0.189	0.187

For both combination types, the DCT features yield better results than the Legendre features, though the results for both features are very close to each other (around 1% of relative difference).

Once taken the decision of using DCT derived features to model the dynamic temporal information, the phoneme-dependent proportions for each of the remaining phonemes with lower Kappa values are computed for both Features Combination and Score Combination.

4.3 Main Experiment

The final experiment is carried out to determine if there is a real gain when combining supervectors with DCT features by testing the final models against both the development and the heldout data. Relative gain is calculated with respect to the baseline system: SVM trained on supervectors features only

A McNemar test is performed as previous step to scope the analysis to a subset of phonemes that yields statistically significant results in the development data (*p-value* < 0.05). McNemar is used to estimate confidence of the two classifiers being actually different using the paired nominal results of both classifiers (2.6.2). The McNemar test is run using the SVM trained only on supervectors as baseline of the comparison.

4.3.1 Development results

Once again, the experiment in the development set is carried out using 4-Fold Cross Validation and taking the average of the results obtained in each fold.

Results are included only for those phonemes that were pointed out as statistically significant for the McNemar test. Those are:

- /s/, /y/, /e/, /o/, /l/, /n/, /m/ in Features Combination
- /s/, /y/, /e/, /o/, /l/, /z/, /n/, /m/, /rr/, /t/ in Score Combination

It is worth noting that phonemes with statistically significant gains in the Features Combination experiment are a subset of the statistically significant phonemes in the Score Combination experiment. For that reason, even though the barplots includes the results for the ten statistically significant phonemes of the Score Combination, additional marks (*) and (**) are included for the Features Combination bars to point out whether or not those results are significant in the Features Combination case.

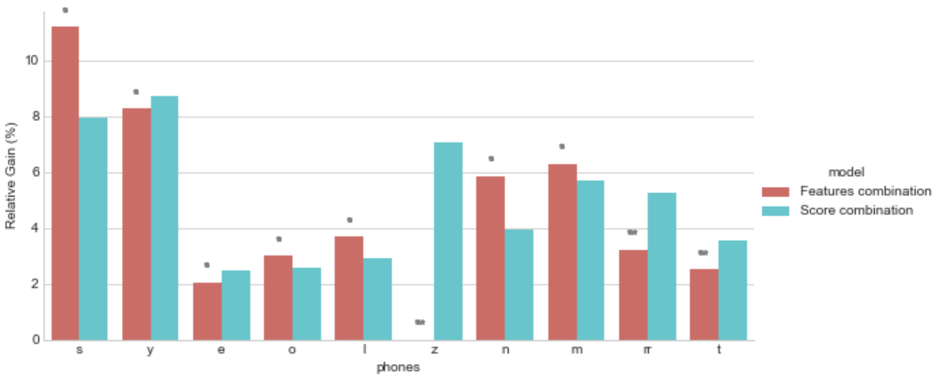


Fig. 4.5: Comparison of performance between Features Combination system and the score combination of individual systems in the training set for the phones whose results were significant in the training set, sorted descendently by their p-value obtained in the McNemar test. For all the score combinations phones the results were significant in the training set, whereas for the features combination phones only the phones marked with (*) were significant, and those marked with (**) weren't significant.

4.3.2 Heldout results

To run the experiment against the heldout data, the models were trained using all the instances of the development set. A new McNemar test is performed for the results obtained in the heldout data for the same phoneme with statistically significant results in the development data, but this time any of them gave a significant p-value < 0.05. This can be explained by the fact that heldout data has around 1/4 of the data of the development data, affecting the efectiveness of the test.

The results are then still scoped to the phonemes with statistically significant results in the development experiment, and the same notation (*) and (**) is kept as a reference.

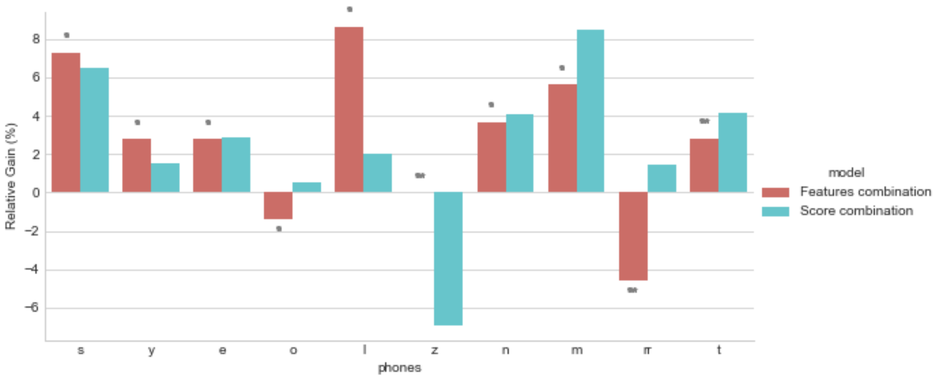


Fig. 4.6: Comparison of performance between Features Combination system and the score combination of individual systems in the test set for the phones whose results were significant in the training set, sorted descendently by their p-value obtained in the McNemar test. For all the score combinations phones the results were significant in the training set, whereas for the features combination phones only the phones marked with (*) were significant, and those marked with (**) weren't significant.

Among all the phonemes that were statistically significant in the Score Combination experiment carried out in the development set, only 'z' degraded its performance compared with the baseline system, though in a substantial way (around 6% relative). Among the remaining phonemes, some reduced its gain compared with the development set (such as /y/ from 8% to less than 2% and /rr/ from almost 6% to less than 2%). Others such as /m/ improved its performance from 6% to more than 8%.

On the other hand, among all the phonemes that were statistically significant in the Features Combination experiment carried out in the development set, only /o/ degraded its performance and in a slightly way. The plots also show a degradation of the /rr/ performance, but /rr/ is not statistically significant for the Features Combination experiment. As in the case of the Score Combination, there are cases where both improvements and degradations of the performance is observed.

A positive aspect of the analysis is that both /s/ and /m/, that are phonemes with high Kappa values, yields almost the best results in both development and heldout experiments, This fact matches the expected results and helps to increase the confidence over the obtained results.

One important aspect to be analysed in the next section is whether or not the Score Combination of the results leads to a more robust classifier than the Features Combination technique.

5. CONCLUSIONS

5.1 Improvements by Adding Dynamic Features

5.2 Comparing Robustness of Fusion Systems

6. APPENDIX A

6.1 Results Tables for All Phones

6.1.1 Features combination

phones	baseline	training	delta	pvalue	#pos	#neg	baseline	test	delta	pvalue	#pos	#neg	kappa
b	0.122	0.122	%0.0	1	528	395	0.133	0.133	%0.0	1	140	95	0.9
m	0.154	0.144	%6.5	0.047	3234	686	0.186	0.176	%5.4	0.268	801	188	0.76
B	0.206	0.2	%2.9	0.374	428	1169	0.179	0.189	-%5.6	0.523	126	302	0.7
s	0.319	0.283	%11.3	8.32E-19	7555	480	0.252	0.234	%7.1	0.454	1963	107	0.57
D	0.182	0.182	%0.0	1	920	2009	0.185	0.185	%0.0	1	268	486	0.55
G	0.162	0.162	%0.0	1	222	643	0.152	0.152	%0.0	1	61	145	0.51
w	0.151	0.151	%0.0	1	743	500	0.141	0.141	%0.0	1	179	128	0.43
i	0.254	0.252	%0.8	0.481	4929	1238	0.252	0.256	-%1.6	0.714	1224	301	0.41
y	0.338	0.31	%8.3	2.02E-05	2453	574	0.371	0.361	%2.7	0.844	596	156	0.39
p	0.295	0.295	%0.0	0.715	1657	1055	0.315	0.316	-%0.3	1	441	254	0.36
r	0.317	0.315	%0.6	0.561	3650	2617	0.331	0.315	%4.8	0.046	910	641	0.36
z	0.222	0.222	%0.0	1	189	997	0.242	0.242	%0.0	1	49	247	0.35
t	0.295	0.288	%2.4	0.183	2938	1542	0.323	0.314	%2.8	0.207	733	360	0.34
k	0.341	0.335	%1.8	0.339	1708	1472	0.304	0.305	-%0.3	0.913	434	388	0.32
rr	0.324	0.314	%3.1	0.299	491	1739	0.341	0.357	-%4.7	0.359	122	453	0.29
a	0.346	0.345	%0.3	0.712	10144	2069	0.371	0.354	%4.6	0.041	2509	548	0.26
o	0.408	0.396	%2.9	0.002	8040	2077	0.387	0.392	-%1.3	0.598	2030	548	0.23
l	0.279	0.269	%3.6	0.019	3505	1373	0.289	0.264	%8.7	0.009	851	356	0.22
e	0.4	0.392	%2.0	0.022	10597	3484	0.392	0.381	%2.8	0.095	2658	899	0.18
n	0.424	0.399	%5.9	1.27E-04	7152	476	0.382	0.368	%3.7	0.772	1792	125	0.15
u	0.354	0.344	%2.8	0.296	1948	482	0.355	0.336	%5.4	0.308	471	110	0.14
x	0.164	0.164	%0.0	1	590	153	0.135	0.162	-%20.0	1	161	37	-
d	0.364	0.364	%0.0	1	773	89	0.304	0.304	%0.0	1	191	18	-
g	0.234	0.232	%0.9	0.222	887	114	0.236	0.276	-%16.9	2.16E-07	237	29	-
N	0.217	0.198	%8.8	0.088	911	443	0.203	0.217	-%6.9	0.296	246	116	-

Tab. 6.1: Results of SVM system trained on Features Combination of supervectors and DCT, for both training and test sets, compared to the SVM trained on supervectors only (baseline system). For each dataset, the relative improvements along with McNemar p-value, and positive and negative number of instances are included. Phones are sort by decreasing Kappa values.

6.1.2 Score combination

phones	baseline	training	delta	pvalue	#pos	#neg	baseline	test	delta	pvalue	#pos	#neg	kappa
b	0.122	0.122	%0.0	1	528	395	0.133	0.133	%0.0	1	140	95	0.9
m	0.154	0.145	%5.8	0.01	3234	686	0.186	0.17	%8.6	0.233	801	188	0.76
B	0.206	0.199	%3.4	0.096	428	1169	0.179	0.185	-%3.4	0.289	126	302	0.7
s	0.319	0.294	%7.8	3.67E-07	7555	480	0.252	0.236	%6.3	0.004	1963	107	0.57
D	0.182	0.179	%1.6	0.327	920	2009	0.185	0.193	-%4.3	0.146	268	486	0.55
G	0.162	0.162	%0.0	1	222	643	0.152	0.159	-%4.6	1	61	145	0.51
w	0.151	0.151	%0.0	1	743	500	0.141	0.141	%0.0	1	179	128	0.43
i	0.254	0.248	%2.4	0.074	4929	1238	0.252	0.248	%1.6	0.039	1224	301	0.41
y	0.338	0.309	%8.6	7.40E-06	2453	574	0.371	0.365	%1.6	0.842	596	156	0.39
p	0.295	0.295	%0.0	1	1657	1055	0.315	0.315	%0.0	1	441	254	0.36
r	0.317	0.316	%0.3	0.285	3650	2617	0.331	0.328	%0.9	0.832	910	641	0.36
z	0.222	0.206	%7.2	0.002	189	997	0.242	0.259	-%7.0	0.774	49	247	0.35
t	0.295	0.285	%3.4	0.024	2938	1542	0.323	0.31	%4.0	0.056	733	360	0.34
k	0.341	0.333	%2.3	0.102	1708	1472	0.304	0.283	%6.9	8.78E-04	434	388	0.32
rr	0.324	0.307	%5.2	0.022	491	1739	0.341	0.336	%1.5	0.913	122	453	0.29
a	0.346	0.344	%0.6	0.883	10144	2069	0.371	0.362	%2.4	0.139	2509	548	0.26
o	0.408	0.398	%2.5	2.13E-04	8040	2077	0.387	0.385	%0.5	1	2030	548	0.23
l	0.279	0.271	%2.9	4.31E-04	3505	1373	0.289	0.283	%2.1	0.152	851	356	0.22
e	0.4	0.39	%2.5	1.78E-04	10597	3484	0.392	0.381	%2.8	0.022	2658	899	0.18
n	0.424	0.407	%4.0	0.006	7152	476	0.382	0.366	%4.2	0.103	1792	125	0.15
u	0.354	0.343	%3.1	0.134	1948	482	0.355	0.327	%7.9	0.02	471	110	0.14
c	0.44	0.404	%8.2	0.01	405	105	0.308	0.353	-%14.6	0.442	104	24	-
x	0.164	0.159	%3.0	1	590	153	0.135	0.112	%17.0	0.227	161	37	-
d	0.364	0.361	%0.8	0.28	773	89	0.304	0.319	-%4.9	0.031	191	18	-
g	0.234	0.228	%2.6	0.774	887	114	0.236	0.238	-%0.8	0.625	237	29	-
N	0.217	0.203	%6.5	0.303	911	443	0.203	0.215	-%5.9	0.015	246	116	-

Tab. 6.2: Results of the Score Combination between SVM system trained on supervectors and SVM system trained on DCT coefficients, for both training and test sets, compared to the SVM trained on supervectors only (baseline system). For each dataset, the relative improvements along with McNemar p-value, and positive and negative number of instances are included. Phones are sort by decreasing Kappa values.

6.2 Fusion Plots for All Phones

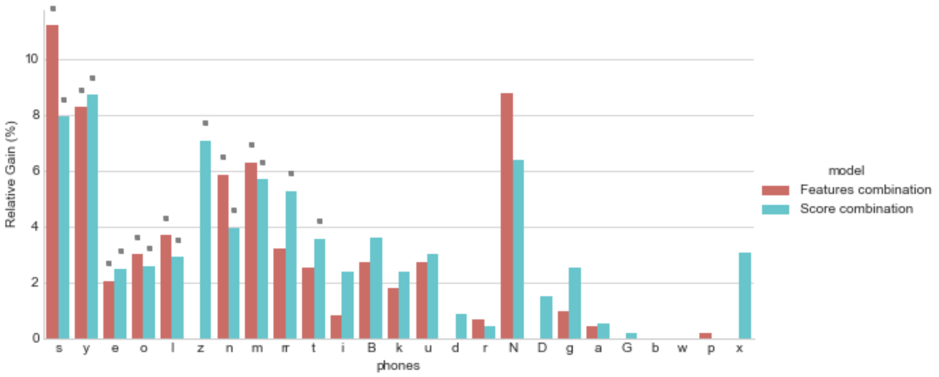


Fig. 6.1: Comparison of performance between Features Combination system and the Score Combination of individual systems in the training set for all phones, sorted descendently by their McNemar p-value obtained in the training set. Only the phones marked with (*) were significant in the training set.

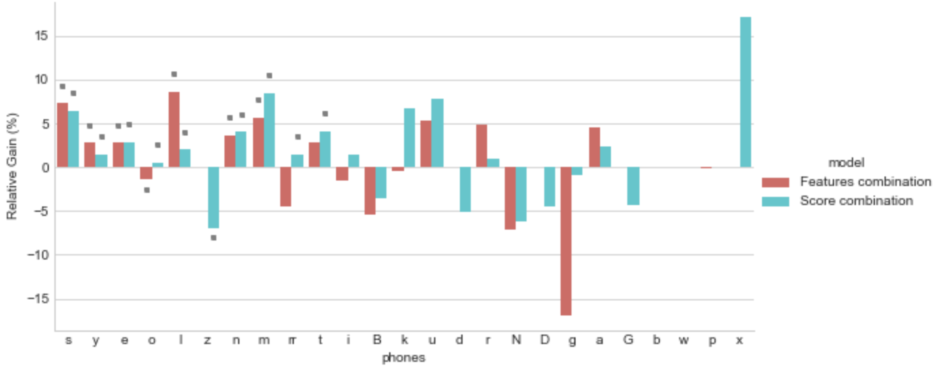


Fig. 6.2: Comparison of performance between Features Combination system and the Score Combination of individual systems in the test set for all phones, sorted descendently by their McNemar p-value obtained in the training set. Only the phones marked with (*) were significant in the development set.

BIBLIOGRAPHY

[1] H. Franco, L. Neumayer, Y. Kim, and O. Ronen. “Automatic pronunciation scoring for language instruction”. In: *Proc. ICASSP 97* (1997).

[2] Y. Kim, H. Franco, and L. Neumayer. “Automatic pronunciation scoring of specific phone segments for language instruction”. In: *Proc. EUROSPEECH 97* (1997).

[3] S. Witt and S. Young. “Language learning based on non-native speech recognition”. In: *Proc. EUROSPEECH 97* (1997).

[4] S. Witt and S. Young. “Phone-level pronunciation scoring and assessment for interactive language learning”. In: *Speech Comunication* (2000).

[5] S. Kanters, C. Cucchiarini, and H. Strik. “The Goodness of Pronunciation Algorithm: a Detailed Performance Study”. In: (2009).

[6] H. Wang, X. Qian, and H. Meng. “Predicting gradation of L2 english mispronunciations using crowdsourced ratings and phonological rules”. In: *Proc. of Speech and Language Technology in Education (SLaTE)* (2013).

[7] A. Harrison, W Lo, X. Qian, and H. Meng. “Implementation of an extended recognition network for mispronunciation detection and diagnosis in computer-assisted pronunciation training”. In: *Proc. of the 2nd ISCA Workshop on Speech and Language Technology in Education* (2009).

[8] S. Yoon, M. Hasegawa-Johnson, and R. Sproat. “Automated pronunciation scoring using confidence scoring and Landmark-based SVM”. In: (2009).

[9] J. Van Doremalen, C. Cucchiarini, and H. Strik. “Automatic Detection of Vowel Pronunciation Errors using multiple information sources”. In: (2009).

[10] S. Wei, H. Guoping, Y. Hu, and R. Wang. “A new method for mispronunciation detection using Support Vector Machine based on Pronunciation Space Models”. In: (2009).

[11] S. Yoon, M. Hasegawa-Johnson, and R. Sproat. “Landmark-based Automated Pronunciation Error Detection”. In: (2010).

[12] H. Strik, K. Truong, F. De Wet, and C. Cucchiarini. “Comparing classifiers for pronunciation error detection”. In: (2009).

[13] H. Franco, L. Neumayer, M. Ramos, and H. Bratt. “Automatic detection of phone-level mispronunciations for language learning”. In: *Proc. Eurospeech 99* (1999).

[14] H. Franco, L. Ferrer, and H. Bratt. “Adaptive and discriminative modelling for improved mispronunciation detection”. In: *ICASSP 14* (2014).

[15] S. Davis and P. Mermelstein. “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* (1980).

[16] W. Campbell, D. Sturim, D. Reynolds, and A. Solomonoff. “SVM based speaker verification using a GMM supervector kernel and NAP variability compensation”. In: *ICASP* (2006).

[17] D.A Reynolds. “Gaussian Mixture Models”. In: *MIT Lincoln Laboratory* ().

[18] C.M Bishop. “Pattern Recognition and Machine Learning”. In: *Springer* (2006).

[19] D. Reynolds, T. Quatieri, and R. Dunn. “Speaker Verification Using Adapted Gaussian Mixture Models”. In: *Digital Signal Processing 10* (2000).

[20] C. Corinna and V. Vapnik. “Support-Vector Networks”. In: *Machine Learning, volume 20* (1995).

[21] James, Witten, Hasti, and Tibshirani. “An Introduction to Statistical Learning”. In: *Springer* (2013).

[22] N. Dehak, P. Dumouchel, and P. Kenny. “Modeling Prosodic Features with Joint Factor Analysis for Speaker Verification”. In: *IEEE Transactions on Audio, Speech and Language Processing* (2007).

[23] M. Kockmann, L. Ferrer, L. Burget, and J. Cernocky. “iVector Fusion of Prosodic and Cepstral Features for Speaker Verification”. In: *Interspeech* (2011).

[24] K. Singh and M. Xie. “Bootstrap: A Statistical Method”. In: (2008).

[25] Luciana Ferrer. “Statistical Modeling of Heterogeneous Features for Speech Processing Tasks”. In: *Ph.D. Dissertation, Standford University* (2008).