Classificação Fonética Usando SVM e Seleção de Parâmetros

Rafael Marinho, Talisman Teixeira Jr. e Aldebaro Klautau

***Resumo*— Este trabalho apresenta novos resultados para classificação fonética usando o corpus TIMIT. Os classificadores são baseados em support vector machines (SVM). Um conjunto relativamente grande de parâmetros é obtido a partir de sete front ends: MFCC, PLP, RASTA, "synchrony", envelope, formantes e "filter-bank". Um subconjunto desses parâmetros é escolhido automaticamente através de um método baseado no algoritmo AdaBoost. Os resultados das simulações são competitivos com os previamente publicados na literatura, e permitem inferir acerca de quais os melhores parâmetros para a composição de um front end heterogêneo.**

***Palavras-Chave*— Classificação fonética, SVM, TIMIT, AdaBoost.**

***Abstract*— This work presents new results for phone classification using the TIMIT corpus. The support vector machines (SVM) is adopted as the learning algorithm for training classifiers. A relatively large set of features is used, which are obtained from seven front ends: MFCC, PLP, RASTA, Seneff's synchrony, Seneff's envelope, formants and filter-bank. A subset of these features is chosen through automatic feature selection based on the AdaBoost algorithm. The simulation results are competitive with others previously published, and provide insight about useful features for composing an heterogeneous front end.**

***Keywords*— Phone classification, SVM, TIMIT, AdaBoost.**

I. INTRODUÇÃO

Na grande maioria dos sistemas automáticos de reconhecimento de voz (RAV), a extração de parâmetros (*front end*) é baseada em coeficientes cepestrais usando a escala *mel* (MFCC) [1] ou coeficientes de predição linear perceptual (PLP) [2].

Este trabalho investiga alternativas a duas características dos front ends convencionais. A primeira é que os mesmos são *baseados em conhecimento*, pois foram desenvolvidos tendo como base o nosso entendimento do processo da fala. Por exemplo, experimentos psico-acústicos com percepção de pitch e mascaramento de tons levaram às escalas de freqüência mel [3] e *Bark* (e.g., [4]), que são usadas para o cálculo dos coeficientes MFCC e PLP, respectivamente. Alternativamente, esse trabalho utiliza uma técnica automática (*data-driven*) para o projeto de front ends baseada na seleção de parâmetros [5].

A segunda característica dos front ends convencionais é que os mesmos adotam um único conjunto *homogêneo* de parâmetros para representar qualquer som, independente de suas características acústicas (e.g., vogal, fricativa, etc). Não é difícil encontrar argumentos que indicam não ser essa a situação ideal. Por exemplo, sons nasais e oclusivos possuem requisitos conflitantes em termos de análise espectral, com os nasais necessitando uma maior resolução espectral e os oclusivos uma maior resolução temporal [6]. Em contraste, o front end pode usar um conjunto *heterogêneo* de parâmetros [6], [7].

Um dos problemas da adoção de parâmetros heterogêneos em RAV é que os mesmos não são facilmente incorporados aos tradicionais modelos *geracionais* (ou *generative* [8]), tais como cadeias escondidas de Markov (HMM). Por exemplo, o algoritmo Baum-Welch precisou ser modificado em [9] para suportar parâmetros heterogêneos. A situação muda se a modelagem acústica baseia-se em classificadores discriminativos tais como *support vector machines* (SVM) [10]. Nesses casos, existe um novo grau de liberdade para o projeto de front ends, uma vez que algumas arquiteturas adotadas em RAV para esses classificadores [11] naturalmente suportam parâmetros heterogêneos. Por exemplo, uma forma eficiente de se obter um classificador baseado em SVMs para problemas com mais de duas classes, consiste em se projetar uma SVM para cada par de classes [12]. Assumindo que cada classe representa um fonema, é intuitivo que os melhores parâmetros para servir de entrada à SVM que tenta distinguir uma vogal de uma oclusiva, são diferentes dos de outra SVM que deva distinguir um par de vogais.

Esse artigo encontra-se organizado da seguinte forma. Na seção II discutimos brevemente aspectos de front ends heterogêneos baseados em SVM. A seção III aborda a seleção de parâmetros usando AdaBoost. Os resultados experimentais são discutidos nas seções IV e V, sendo seguidos pelas conclusões.

II. SVMS COM PARÂMETROS HETEROGÊNEOS

Um típico front end converte trechos (ou *janelas*) $\mathbf{s}$ de voz digitalizada (e.g., 30 milisegundos) em vetores de parâmetros $\mathbf{x} = (x_1, \ldots, x_L)$, onde $x_i \in \mathcal{X}_i$ e $\mathbf{x} \in \mathcal{X}^L = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_L$. A conversão ocorre à taxa r, e $1/r$ é a duração do *quadro* (equivalente ao deslocamento da janela). Tipicamente, $r = 100$ Hz e $L = 39$ parâmetros.

Dizemos que o conjunto de parâmetros $\mathcal{X}^L$ é *homogêneo* com respeito a um dado sistema RAV baseado em SVM, se os L parâmetros são usadas por todas as SVMs. O conjunto $\mathcal{X}^L$ é chamado heterogêneo se essa condição não é respeitada, havendo no mínimo dois subconjuntos distintos de $\mathcal{X}^L$ que sejam usados por SVMs específicas. Similarmente, quando a modelagem acústica é baseada em HMMs, $\mathcal{X}^L$ é heterogêneo com respeito ao sistema se subconjuntos distintos são usados como espaço amostral para as distribuições de saída das HMMs (e.g., quando os parâmetros dependem da classe fonética a qual pertence a HMM).

Ressalta-se a distinção entre *multi-stream* [13], [14] e parâmetros heterogêneos. No primeiro caso, o front end usa

Laboratório de Processamento de Sinais, Universidade Federal do Pará, CP 8619, Belém, PA, 66075-900, Brasil, *http://www.laps.ufpa.br*, E-mail: a.klautau@ieee.org.

diferentes algoritmos, tais como MFCC e PLP, para compor o conjunto de parâmetros, mas todos esses parâmetros são sempre simultaneamente usados na modelagem acústica.

Um front end heterogêneo consiste então em um mapeamento $F : \mathbf{s} \rightarrow \mathbf{x}$ de trechos de voz em parâmetros, e uma lista de conjuntos com os índices selecionados $\{\mathcal{I}_b\}$, os quais determinam os subconjuntos de $\mathcal{X}^L$ que devem ser usados. Em RAV baseado em SVM, a b-th SVM é treinada usando-se $L_b = |\mathcal{I}_b|$ parâmetros, onde $|\cdot|$ representa a cardinalidade de um conjunto e $1 \leq L_b \leq L$. No presente trabalho usou-se o mesmo valor de L_b para todas as SVMs.

Uma questão pertinente é por que não utilizar sempre L parâmetros? Por exemplo, as SVMs poderiam identificar os parâmetros supérfluos, ruidosos, etc., e desenfatizar (ou até mesmo eliminar) suas influências. Na prática, entretanto, além de diminuir o custo computacional, a redução do número de parâmetros pode até mesmo levar a uma melhor capacidade de generalização quando há uma quantidade limitada de dados para treinamento [15]. Dessa forma, um objetivo válido é reduzir L_b retendo informação suficiente para atingir baixa taxa de erro com a correspondente b-ésima SVM.

Neste trabalho, estuda-se o projeto de front end heterogêneos utilizando uma abordagem data-driven. A mesma consiste em extrair um grande conjunto de parâmetros (possivelmente redundantes), e selecionar um subconjunto dos mesmos em uma segunda etapa. Essa abordagem foi utilizada com sucesso para o processamento de imagens em [16]. A próxima seção descreve o método de seleção utilizado.

III. Seleção de parâmetros com AdaBoost

Os métodos para reduzir a dimensionalidade de $\mathcal{X}^L$ podem ser organizados em dois grupos: *seleção* e *extração* de parâmetros [17]. Métodos de seleção tentam identificar e reter apenas os parâmetros que mais contribuem para a execução de uma dada tarefa. Esses métodos não modificam os parâmetros, mas apenas escolhem um subconjunto dentre o total de 2^L possíveis subconjuntos. Métodos de extração de parâmetros transformam o espaço original $\mathcal{X}^L$ em um espaço de menor dimensão, modificando os parâmetros originais. Um exemplo seria o uso de *principal component analysis* (PCA) [17] seguido do descarte de componentes de menor importância.

Uma segunda possível caracterização dos métodos de seleção de parâmetros é como filtros (*filters*) ou *wrappers* [5]. Os últimos são métodos que selecionam os parâmetros utilizando o próprio algoritmo de treinamento $\mathcal{L}$ que será usado no projeto do classificador (SVM no presente caso). Na maioria dos casos, é inviável computacionalmente treinar uma SVM para cada um dos 2^L possíveis conjuntos de $\mathcal{X}^L$ ($2^L - 1$ se for imposta a seleção de ao menos um parâmetro). Assim, na prática, wrappers utilizam uma heurística possivelmente subótima [5]. Os métodos do tipo filtro avaliam um dado parâmetro x_i através do uso de heurísticas, tais como a correlação de x_i com a classe a qual x_i pertence. Nesse trabalho, utilizamos um método filtro baseado em *boosting* para realizar a seleção de parâmetros.

Boosting é um método de meta-aprendizado que busca atingir um bom classificador combinando classificadores (ou *hipóteses*) h_i relativamente simples geradas por qualquer algoritmo de aprendizado $\mathcal{L}$, aqui chamado algoritmo *fraco*. O mais famoso dos algoritmos de boosting é o AdaBoost [18], que compõe o classificador final $H(\mathbf{x})$ como uma ponderação de T hipóteses fracas h_t:

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right). \quad (1)$$

Em síntese, AdaBoost exige como entrada um conjunto de treinamento com exemplos $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ onde $\mathbf{x}_i$ é o vetor de parâmetros e $y_i \in \{-1, 1\}$ a classe (assume-se aqui problemas binários). AdaBoost chama o algoritmo fraco $\mathcal{L}$ iterativamente $t = 1, \ldots, T$ e, em cada iteração, armazena o classificador retornado por $\mathcal{L}$, modifica o peso de cada exemplo para a próxima iteração, e volta a chamar $\mathcal{L}$.

Para manusear os pesos, AdaBoost mantém uma distribuição D_t para os exemplos da iteração t, onde $D_t(i)$ é o peso do exemplo i e $\sum_{i=1}^{m} D_t(i) = 1, \forall t$. Para a primeira iteração pode-se, por exemplo, assumir uma distribuição uniforme $D_1(i) = 1/m$. A cada iteração, os pesos dos exemplos classificados erroneamente são aumentados de forma que o algoritmo fraco seja forçado a se concentrar nos exemplos mais difíceis. Nota-se que o problema repassado a $\mathcal{L}$ só muda a cada iteração em termos dos pesos $D_t(i)$. Existem algoritmos que podem lidar diretamente com os pesos de cada exemplo, e para os que não podem, costuma-se usar uma reamostragem dos exemplos baseada em $D_t(i)$.

Em síntese, os pesos são modificados da seguinte forma. Considere que o classificador (hipótese fraca) h_t levou a um erro ε_t ao ser testado com o conjunto de m exemplos de treino. AdaBoost então calcula α_t na Equação (1) usando

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right).$$

Nota-se que, pelo fato do problema ser binário, caso $\varepsilon_t > 0.5$, pode-se adotar $-h_t$ ao invés de h_t. Em outras palavras, AdaBoost assume que é possível encontrar um classificador binário para o qual $\varepsilon_t < 0.5$. Assim, tem-se $\alpha_t \geq 0$, quando $0 \leq \varepsilon_t \leq 0.5$.

AdaBoost modifica a distribuicao D_t para a próxima iteração de acordo com:

$$D_{t+1}(i) = \frac{D_t(i) \exp\left(-\alpha y_i h_t(\mathbf{x}_i)\right)}{Z_t}.$$

Como $\alpha > 0$, caso a predição $h_t(\mathbf{x}_i)$ seja correta, o peso $D_t(i)$ é diminuído, já que $h_t(\mathbf{x}_i)$ e y_i teriam o mesmo sinal e o argumento da exponencial seria negativo. Se a predição for errada, i.e., $y_i h_t(\mathbf{x}_i) < 0$, $D_t(i)$ é aumentado. O termo Z_t é apenas um fator de normalização, uma vez que se deve garantir que $\sum_{i=1}^{m} D_{t+1}(i) = 1$.

AdaBoost, em sua forma geral, não realiza seleção de parâmetros. Mas para isso, basta que se escolha um algoritmo fraco que gere a cada iteração de boosting, hipóteses que utilizem apenas um subconjunto de parâmetros. Nesse trabalho, adotamos *decision stumps*, que podem ser vistas como uma árvore de decisão de um único nó [16]. Assumindo que $\mathbf{x} \in \mathbb{R}^L$, treinar uma stump consiste em escolher uma coordenada $z \in \{1, \ldots, L\}$, e um limiar l_z associado. Dado

um vetor a ser classificado, a stump verifica se o valor do z-ésimo elemento desse vetor é menor do que l_z e atribui o vetor a uma das duas classes.

Para este trabalho, modificamos AdaBoost (mais especificamente a versão AdaBoost.M1 [19]) no pacote Weka [20] para que o mesmo funcione da seguinte maneira. O número L_b de parâmetros a serem selecionados é especificado e, para cada iteração $t = 1, \ldots, L_b$ de boosting com stumps, um parâmetro x_z, $z \in \mathcal{Z}_t$, é escolhido e seu índice armazenado $k_t = z$. Na primeira iteração, $\mathcal{Z}_1 = \{1, \ldots, L\}$ e, para iterações subsequentes, o parâmetro que foi previamente selecionado não mais concorre, ou seja, $\mathcal{Z}_{t+1} = \mathcal{Z}_t - \{k_t\}$.

Para achar a melhor stump, Weka exige um custo computacional da orde de $\mathcal{O}(m \log m)$, onde m é o número de exemplos de treino. Para diminuir o tempo de execução, o algoritmo foi modificado de acordo com a estratégia descrita em [7], a qual reduz a complexidade para $\mathcal{O}(m)$.

IV. Configuração dos Experimentos

Por simplicidade, testamos a metodologia de projeto de front ends heterogêneos usando classificação fonética baseada no corpus TIMIT. Esta seção descreve como os experimentos foram realizados.

A. Corpus TIMIT

O corpus TIMIT é o mais popular dentre os distribuídos pelo Linguistic Data Consortium.[1] Ele foi gravado pela Texas Instruments (TI), transcrito no Massachusetts Institute of Technology (MIT), e preparado para distribuição pelo National Institute of Standards and Technology (NIST) dos EUA. O corpus inclui os arquivos com as formas de onda e suas transcrições ortográficas e fonéticas. A transcrição fonética, indicando onde inicia e termina cada alofone (ou *fone*), é o que torna o TIMIT tão especial. Ela demandou de 100 a 1000 horas de trabalho para transcrever cada hora de voz, com o projeto custando mais de 1 milhão de dólares [21]. Assim, TIMIT é o corpus perfeito para a realização dos estudos com classificação fonética deste trabalho.

Cada locutor contribui com 10 sentenças, que são divididas nos tipos *sx*, *si* e *sa*. As transcrições fonéticas do TIMIT adotam 61 símblos. Nesse trabalho, como geralmente feito, esses símbolos foram redistribuídos nas $K = 39$ classes fonéticas definidas por Kai-Fu Lee [22]. Apesar de alguns representarem uma classe, os 39 símbolos serão chamados de fones por simplicidade. Ainda seguindo a praxe, as sentenças do tipo *sa* foram excluídas do treino e teste, e os resultados são reportados para um subconjunto das sentenças de teste chamado *core test set*, que inclui apenas 192 sentenças (24 locutores contribuem com 8 sentenças cada, pois as duas do tipo *sa* são descartadas).

Para efeito de ilustração, a Tabela I apresenta alguns resultados da literatura, incluindo os que acreditamos serem os de menor taxa de erro. GMM refere-se ao classificador que usa uma mistura de Gaussianas, o qual pode ser visto como uma HMM *contínua* de um único estado. Ressalta-se que os trabalhos mencionados não usaram exatamente a mesma metodologia, e os números devem ser interpretados a partir de uma leitura atenta dos artigos. Por exemplo, alguns autores descartam completamente a oclusiva glotal "q", enquanto outros não. Outro ponto de divergência é que alguns trabalhos reportam o erro para todo o conjunto de teste, enquanto outros adotam apenas para um subconjunto, tal como o *core test set*.

[1] *http://www.ldc.upenn.edu/Catalog/top_ten.html*.

TABELA I

Taxa de erro (%) para classificação fonética usando TIMIT. Os dois primeiros sistemas usam HMMs que dependem do contexto (CD), ou seja, dos fones adjacentes.

Autor(es) [referência]	Erro	Observação
Chengalvarayan and Deng [23]	18.5	CD HMM
Chengalvarayan and Deng [24]	16.5	CD HMM
Chengalvarayan and Deng [23]	31.8	HMM
Salomon et al [25]	28.6	SVM
Chengalvarayanand and Deng [24]	27.6	HMM
Zahorian et al [26]	23.0	redes neurais
Hazen and Halberstadt [27]	20.2	GMM
Halberstadt [6]	18.3	GMM

B. SVMs

Em sua formulação original, as SVMs são restritas à solução de problemas binários. Como a tarefa é distinguir a classe fonética correta dentre $K = 39$ opções, lançou-se mão do esquema *error-correcting output code* (ECOC) com a matriz *todo-os-pares* (ou *all-pairs*) e decodificação de Hamming [12]. Em síntese, a matriz todos-os-pares exige o treinamento de uma SVM para cada par de fones. Nesse caso, o número de SVMs que treinamos é $\binom{K}{2} = 741$. Assim, nota-se que cada SVM lida com um problema binário. Na fase de teste, o vetor $\mathbf{x}$ é submetido a todas as $\binom{K}{2}$ SVMs e o fone com o maior número de "vitórias" é declarado vencedor. Nota-se que cada fone participa de $K-1$ SVMs, sendo esse o máximo número de vitórias que um fone pode alcançar.

Outra restrição das SVMs é que, como todo classificador (redes neurais, GMMs, etc.), as mesmas exigem vetores de entrada com um número fixo de elementos. Os fones, entretanto, possuem duração variável e seria incompatível simplesmente extrair um número de parâmetros proporcional à duração. Uma das maneiras de contornar esse problema é mapear qualquer fone em um vetor de comprimento L, tal como feito em [28]. Em síntese, a idéia é utilizar um front end *frame-based* convencional no primeiro estágio que, para cada fone, gere T vetores de K parâmetros cada, à uma taxa fixa r (utilizamos $r = 100$ Hz). Após isso, divide-se linearmente os T quadros do fone em três segmentos de acordo com a razão 3-4-3. Por exemplo, se um fone possui $T = 20$ quadros, o primeiro segmento será composto pelos 6 primeiros quadros, o segundo pelos 8 próximos quadros e o terceiro pelos 6 quadros restantes. A literatura acerca de *dynamic-time warping* (DTW) (e.g., [29]) indica que essa não é a estratégia ideal para voz, mas é atrativa devido à sua simplicidade. Uma vez determinados os três segmentos, retira-se a média dos parâmetros associados a cada um deles e obtém-se um vetor com $L = 3K$ elementos. A adoção de três segmentos inspira-se em RAV baseados em HMMs, onde os fones são geralmente

modelados como compostos de três partes aproximadamente estacionárias. Dessa forma, cada fone é representado por um vetor de dimensão L e a correspondente classe y.

Para melhor convergência das SVMs, os arquivos de treino foram normalizados de forma que todos os parâmetros ficassem restritos à faixa $[0, 1]$. Os mesmos fatores de normalização obtidos com os dados de treino foram utilizados para normalizar os dados de teste.

SVMs permitem flexibilidade na escolha do *kernel*, que pode ser polinomial, Gaussiano, etc. [30]. Para diminuir o custo computacional, adotamos nesse trabalho exclusivamente o kernel linear, que permite que a SVM seja convertida em um *perceptron*, evitando-se armazenar os vetores de suporte.

C. Front ends

Para esse trabalho, utilizamos parâmetros obtidos por sete diferentes front ends. A simples concatenação de todos esses parâmetros origina um conjunto altamente redundante. A partir daí, lançamos mão de um método automático para selecionar parâmetros, o qual busca identificar aqueles que possuem informação complementar aos já selecionados.

O primeiro front end é chamado *formantes*, e consiste da freqüência fundamental F0 (também chamada de *pitch*, apesar de não serem a mesma coisa), probabildade de vozeamento e as primeiras quatro freqüências formantes (F1-F4). Estes parâmetros foram obtidos usando-se algoritmos desenvolvidos por David Talkin [31], [32] e outros. Os mesmos eram parte do popular pacote Waves+, o qual não é mais comercializado. Contudo, foram recentemente incorporados ao pacote *open-source* Snack, o qual pode ser obtido em *www.speech.kth.se/snack*. Usamos a versão 2.2 do Snack.

O segundo conjunto de parâmetros foi obtido com um típico front end PLP [2] com 39 parâmetros por quadro, correspondendo a 13 coeficientes *estáticos* (12 coeficientes e energia do quadro), e estimativas de suas primeira e segunda derivadas.

Os próximos dois conjuntos de parâmetros, com 40 parâmetros cada, correspondem aos estágios *synchrony* e *envelope* do modelo de Seneff para o sistema auditivo [33]. Nesse caso, todo o processamento é feito no domínio do tempo. Técnicas como FFT não são utilizadas devido às não-linearidades assumidas pelo modelo. Assim, o tempo de processamento do front end Seneff chega a ser maior do que 100 vezes o tempo real (ou seja, para um segundo de voz, gasta-se mais de 100 segundos de processamento).

De forma similar ao front end *plp*, os próximos dois front ends adotam 39 parâmetros a cada quadro, baseados em MFCC [1] e RASTA [34], respectivamente.

O último front end, com 50 parâmetros por quadro e chamado *filter-bank*, consiste na potência de saída de 24 filtros espaçados de acordo com a escala mel [3], a energia normalizada do quadro e a estimativa da primeira derivada desse vetor com 25 elementos.

Concatenando-se os parâmetros dos sete front ends, obtém-se um vetor com $6+39+40+40+39+39+50 = 253$ elementos para representar cada quadro. Divindo-se os quadros de cada fone em três segmentos na razão 3-4-3, calculando-se a média em cada segmento e concatenando-se os valores, obtém-se $3 \times 253 = 759$ parâmetros. Nota-se que para alguns front ends, a média é calculada no domínio cepestral, o que corresponde a uma média geométrica no domínio da freqüência. Como um último parâmetro, foi acrescida a duração de cada fone em número de quadros, resultando em $L = 760$ parâmetros para representar cada fone.

TABELA II
CLASSIFICAÇÃO FONÉTICA USANDO SVMS E SELEÇÃO DE PARÂMETROS.

Front end	parâmetros por fone	parâmetros por SVM	parâmetros distintos	erro (%)
plp + duração	118	118	118	28.2
mfcc + duração	118	118	118	29.1
todos	760	760	760	21.0
todos	760	100	758	22.7
todos	760	40	755	25.3
todos	760	25	498	26.6
todos	760	5	464	34.6

V. RESULTADOS

Nesta seção apresentamos os resultados obtidos com SVMs para classificação fonética, associadas ao uso de AdaBoost para a seleção de parâmetros.

A Tabela II mostra a taxa de erro para diferentes situações. A penúltima coluna indica o número de parâmetros distintos quando são aglutinados todos os parâmetros utilizados pelas SVMs. Por exemplo, caso fossem adotados 25 parâmetros por SVM (o que já levaria a um erro menor do que os obtidos pelos populares parâmetros MFCC e PLP nas duas primeiras linhas), seriam utilizados apenas 498 dos 760 parâmetros originalmente disponiveis. Por outro lado, usando-se 100 parâmetros por SVM atinge-se um uso da quase totalidade dos 760 parâmetros.

Nota-se que os resultados na Tabela II são competitivos com os apresentados da literatura, ilustrados pela Tabela I. Por exemplo, alguns de nossos resultados usando SVM com kernel linear são superiores ao publicado em [25], onde SVMs com kernels não-lineares conduziram a um erro de 28.6%. Em especial, os resultados indicam que SVMs são particularmente eficientes quando o espaço de entrada possui dimensão elevada, mesmo que os parâmetros sejam altamente redundantes. O valor de 21% de erro é relativamente próximo dos 18.3% reportados em [6], que são os melhores resultados que conhecemos ao se tratar de modelos que não usam o contexto. Mais interessante do que as taxas de erro em si, é a mineração dos dados em busca de informação que possa aperfeiçoar o projeto de front ends.

A Tabela III mostra os 10 parâmetros mais "importantes" de acordo com AdaBoost para a distinção entre os pares de consoantes plosivas (p,b), (d,t) e (g,k). Como cada parâmetro, com exceção de *duração*, é utilizado em 3 segmentos, os sufixos "s1", "s2" e "s3" são usados para indicar do primeiro ao terceiro segmento, respectivamente. As letras "d" e "a" indicam a primeira e segunda derivada, respectivamente, para os front ends *mfcc*, *plp*, *rasta* e *filter-bank* (este último não usa a segunda derivada). Como podia ser esperado, a Tabela III indica que para os três pares de consoantes, o parâmetro considerado mais importante foi a probabilidade

TABELA III

OS PRIMEIROS 10 PARÂMETROS SELECIONADOS POR ADABOOST PARA CLASSIFICAR PARES DE CONSOANTES PLOSIVAS CUJA DISTINÇÃO É A EXISTÊNCIA OU NÃO DE VOZEAMENTO.

rank	**p** vs. **b**	**d** vs. **t**	**k** vs. **g**
1	probVozeamento-s1	probVozeamento-s2	probVozeamento-s1
2	duração	duração	duração
3	seneff-env-13-s2	mfcc1-s1	F1-s3
4	seneff-syn-8-s1	plp2-s1	seneff-env-34-s1
5	seneff-syn-7-s1	F0-s2	seneff-env-35-s1
6	fbank-4-s2	seneff-env-14-s1	plp-d-3-s2
7	seneff-env-40-s1	seneff-env-19-s1	rasta-d-9-s2
8	rasta-d-8-s2	rasta-d-2-s2	probVozeamento-s3
9	F1-s3	rasta-a-2-s3	F0-s1
10	mfcc-d-4-s1	seneff-env-4-s1	plp-d-8-s3

TABELA IV

OS PRIMEIROS 10 PARÂMETROS SELECIONADOS POR ADABOOST PARA CLASSIFICAR ALGUNS PARES DE FONES.

rank	**s** vs. **f**	**aa** vs. **ow**	**n** vs. **ng**
1	fbank-22-p2	F1-p2	seneff-env-24-p1
2	mfcc2-p2	fbank-d-12-p2	seneff-env-23-p1
3	seneff-syn-38-p2	plp3-p3	seneff-env-25-p1
4	fbank-3-p3	mfcc3-p3	seneff-syn-32-p1
5	seneff-syn-39-p2	F3-p3	seneff-syn-31-p1
6	plp5-p2	F1-p3	fbank-d-3-p1
7	plp2-p2	F0-p3	seneff-env-24-p3
8	mfcc2-p1	rasta3-p3	fbank-d-6-p1
9	fbank-10-p3	F1-p1	seneff-env-23-p3
10	fbank-d-energy-p1	seneff-syn-34-p2	plp-d-1-p1

de vozeamento (seguida pela *duração*). Vale ressaltar que AdaBoost busca iterativamente encontrar parâmetros que são complementares aos já escolhidos, o que leva a melhores resultados do que a escolha independente dos parâmetros, como mostrado em [7]. Por exemplo, para o par (p,b), após a escolha de *probVozeamento-s1*, os demais parâmetros não possuem uma relação direta com vozeamento. Por outro lado, para o par (k,g), AdaBoost volta a utilizar parâmetros como *probVozeamento-s3* e *F0-s1*.

A Tabela IV é similar à anterior, mas adota outros pares de fones. Pode-se observar que AdaBoost privilegia os parâmetros do front end de Seneff (ambos *synchrony* e *envelope*) para a distinção das nasais (n,ng). Obviamente, isso não atesta que outros parâmetros não possam atingir bom desempenho na classificação de (n,ng), mas aponta para a necessidade de uma maior investigação. Nota-se a escolha de *fbank-22-p2* na distinção entre (s,f), onde esse parâmetro é a potência na saída do 22^{o} (ante-penúltimo) filtro. Esse tipo de informação, obtida em freqüência relativamente alta, não é privilegiada em front ends convencionais. Outra observação é a importância dada às formantes no caso do par (aa,ow), enquanto que nos outros dois as formantes não aparecem.

Nas Figuras 1 e 2 são mostrados os histogramas dos parâmetros selecionados ao se usar $L_b = 5$ e 100 por SVM, respectivamente. Para melhorar a visualização, foram agrupadas as ocorrências dos parâmetros nos três segmentos. Por exemplo, todas as ocorrências de *plp5-p1*, *plp5-p2* e *plp5-p3* foram somadas e estão representadas pelo quinto elemento do front end *plp*. O parâmetro *duração* não aparece nos histogramas, mas foi selecionado 221 e 479 vezes, quando

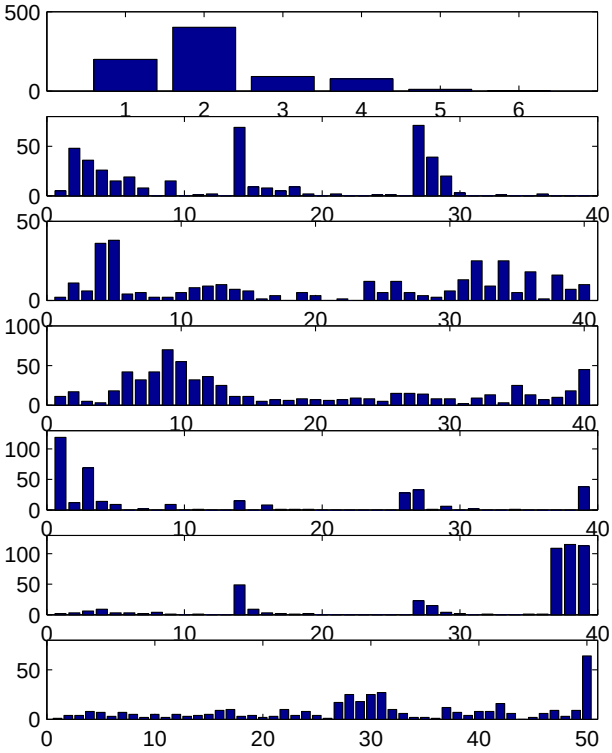

Fig. 1. Histogramas para 5 parâmetros selecionados por SVM, para os front ends *formantes* (acima), *plp*, *synchrony*, *envelope*, *mfcc*, *rasta*, e *filter-bank* (abaixo).

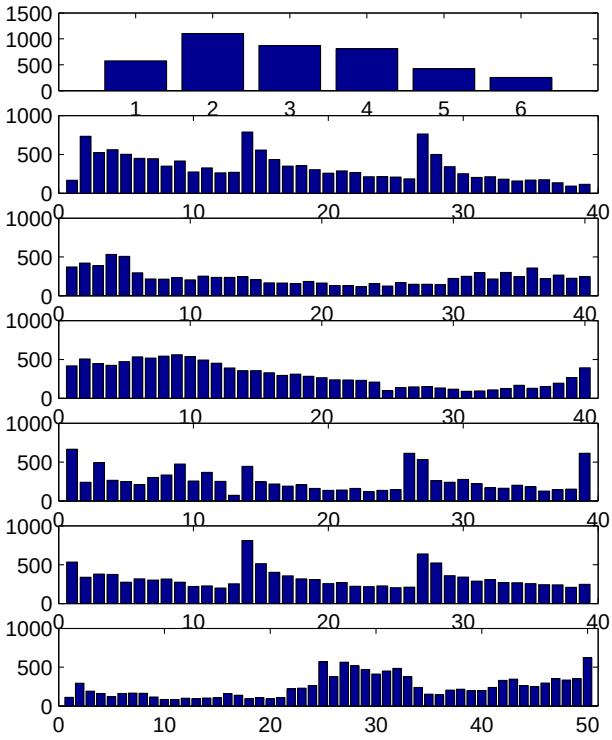

Fig. 2. Histogramas para 100 parâmetros selecionados por SVM, para os front ends *formantes* (acima), *plp*, *synchrony*, *envelope*, *mfcc*, *rasta*, e *filter-bank* (abaixo).

TABELA V
PARÂMETROS MAIS "IMPORTANTES" E NÚMERO DE VEZES QUE FOI ESCOLHIDO PARA TODOS OS PARES DE VOGAIS VERSUS CONSOANTES COM $L_b = 10$ PARÂMETROS POR SVM.

rank	parâmetro	ocorrências
1	probVoicing-p2	178
2	duration	168
3	F1-p2	117
4	F0-p2	113
5	F2-p2	104
6	rasta-a-energy-p1	104
7	rasta-a-12-p1	102
8	F4-p2	99
9	plp1-p2	97
10	rasta-a-11-p1	93

$L_b = 5$ e 100, respectivamente. Considerando as ocorrências nos três segmentos, *probVozeamento* foi escolhida 472 e 1102 vezes, para $L_b = 5$ e 100, respectivamente. Obviamente, com o aumento de L_b os histogramas (se normalizados) convergem para uma distribuição uniforme. Comparando-se as Figuras 1 e 2 observa-se que os parâmetros de cada front end mantém aproximadamente sua importância relativa aos outros do mesmo front end ao variarmos L_b de 5 para 100. A exceção notória são os três últimos parâmetros do *rasta*, que despontam quando $L_b = 5$ mas não mantém o status na Figura 2. Essa e outras das informações "mineradas" nos experimentos, merecem mais pesquisa em busca de explicações adequadas.

Outro tipo de inferência que estamos fazendo consiste em identificar os parâmetros que facilitam a distinção entre classes fonéticas distintas. Por exemplo, a Tabela V mostra os parâmetros mais "importantes" e o número de vezes que cada um foi escolhido ao se considerar $L_b = 10$ e todos os pares envolvendo vogais (e semi-vogais) versus consoantes. Pode-se atentar para a importância das formantes e do uso da segunda derivada (indicador "a" no nome do parâmetro).

VI. CONCLUSÕES

Esse trabalho estende os resultados apresentados em [7], apresentando novas inferências acerca do projeto de um front end heterogêneo. Os resultados indicam que SVMs são particularmente eficientes quando o espaço de entrada possui dimensão elevada, mesmo que os parâmetros sejam altamente redundantes.

A continuação desse trabalho engloba novos métodos para seleção de parâmetros, em especial *wrappers*, e a implementação de um front end heterogêneo para reconhecimento de dígitos. A maior dificuldade para esse último passo é a incorporação de um mecanismo para explorar a informação de contexto. Os sistemas baseados em HMMs se beneficiam imensamente do uso de contexto (a popularidade de trifones é uma prova disso). Contudo, SVMs, redes neurais e similares, possuem maior dificuldade em aproveitar adequadamente essa informação.

REFERÊNCIAS

[1] S. Davis and P. Merlmestein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Trans. on ASSP*, pages 357–366, Aug. 1980.

[2] H. Hermansky. Perceptual linear predictive (PLP) analysis of speech. *Journal of the Acoustical Society of America*, 87(4):1738–52, Apr. 1990.

[3] S. Stevens and J. Volkman. The relation of pitch to frequency. *Journal of Psychology*, 53:329, 1940.

[4] B. Novorita. Incorporation of temporal masking effects into Bark spectral distortion measure. In *ICASSP*, 1999.

[5] M. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, University of Waikato, 1999.

[6] A. Halberstadt. *Heterogeneous acoustic measurements and multiple classifiers for speech recognition*. PhD thesis, MIT, 1998.

[7] A. Klautau. Mining speech: Automatic selection of heterogeneous features using boosting. In *ICASSP*, 2003.

[8] A. Ng and M. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, 2002.

[9] P. Baggenstoss. A modified Baum-Welch algorithm for hidden Markov models with multiple observation spaces. *IEEE Trans. on Speech and Audio Proc.*, 9(4):411–16, 2001.

[10] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[11] A. Klautau, N. Jevtić, and A. Orlitsky. Speech recognition based on discriminative classifiers. In *SBT, Rio de Janeiro, Brazil*, 2003.

[12] A. Klautau, N. Jevtić, and A. Orlitsky. On nearest-neighbor ECOC with application to all-pairs multiclass SVM. *Journal of Machine Learning Research*, 2003.

[13] H. Bourlard, S. Dupont, and C. Ris. Multi-stream speech recognition. *CC-AI, The Journal for the Integrated Study of Artificial Intelligence, Cognitive Science and Applied Epistemology*, 15(3):215–34, 1998.

[14] H. Christensen, B. Lindberg, and O. Andersen. Employing heterogeneous information in a multi-stream framework. In *ICASSP*, volume 3, pages 1571–4, 2000.

[15] A. Ng. On feature selection: Learning with exponentially many irrelevant features as training examples. In *International Conf. on Machine Learning*, pages 404–412, 1998.

[16] K. Tieu and P. Viola. Boosting image retrieval. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 228–235, 2000.

[17] A. Webb. *Statistical Pattern Recognition*. Oxford University Press Inc., 1999.

[18] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conf. on Computational Learning Theory*, pages 23–37, 1995.

[19] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conf. on Machine Learning*, pages 148–156, 1996.

[20] http://www.cs.waikato.ac.nz/ml/weka.

[21] J. Picone. Talk at SRSTW'02, http://www.isip.msstate.edu, 2002.

[22] K.-F. Lee and H.-W. Hon. Speaker-independent phone recognition using hidden Markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(11):1641–8, Nov. 1989.

[23] R. Chengalvarayan and Li Deng. Use of generalized dynamic feature parameters for speech recognition. *IEEE Transactions on Speech and Audio Processing*, 5:232–242, 1997.

[24] R. Chengalvarayan and Li Deng. Speech trajectory discrimination using the minimum classification error learning. *IEEE Transactions on Speech and Audio Processing*, 6:505–515, 1998.

[25] J. Salomon, K. Simon, and M. Osborne. Framewise phone classification using support vector machines. In *ICSLP*, pages 2645–2648, 2002.

[26] Stephen A. Zahorian, Peter L. Silsbee, and Xihong Wang. Phone classification with segmental features and a binary-pair partitioned neural network classifier. In *Proc. ICASSP97*, pages 1011–1014, 1997.

[27] T. J. Hazen and A. K. Halberstadt. Using aggregation to improve the performance of mixture Gaussian acoustic models. In *ICASSP*, pages 653–656, 1998.

[28] A. Ganapathiraju. *Support Vector Machines for Speech Recognition*. PhD thesis, Mississippi State University, 2002.

[29] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. on ASSP*, 26(1):43–49, 1978.

[30] B. Scholkopf and A. Smola. *Learning with kernels*. MIT Press, 2002.

[31] D. Talkin. Speech formant trajectory estimation using dynamic programming with modulated transition costs. Technical report, AT & T Bell Laboratories 11222-870720-07TM, 1987.

[32] D. Talkin. A robust algorithm for pitch tracking (RAPT). In W. Kleijn and K. Paliwal, editors, *Speech Coding and Synthesis*. Elsevier, 1995.

[33] S. Seneff. Pitch and spectral estimation of speech based on auditory synchrony model. *ICASSP 84*, 3:p.36.2/1–4, Mar. 1984.

[34] H. Hermansky and N. Morgan. Rasta processing of speech. *IEEE Transactions on Speech and Audio Processing*, 2(4):578–89, Oct. 1994.