Sistema Computacional de Codificação
Automática de Atividades Econômicas

Projeto Classificação Automática em CNAE-Subclasses

Relato de Cumprimento de Metas No. 4

Meta Física 1.1/2008
Meta Física 1.2/2008
Meta Física 1.3/2008
Meta Física 1.4/2008

Fundação Espírito-Santense de Tecnologia
Av. Fernando Ferrari, 845 | Goiabeiras | 29.060-410 - Vitória - ES
Tel: +55 27 3345-7555 | Fax: +55 27 3345-7668 | www.fest.org.br

Sumário

SUMÁRIO **2**

ÍNDICE DE FIGURAS **4**

ÍNDICE DE TABELAS **5**

1 INTRODUÇÃO **6**

1.1 MOTIVAÇÃO E JUSTIFICATIVA 6
1.2 OBJETIVOS 7
1.3 ORGANIZAÇÃO DESTE DOCUMENTO 7

2 CLASSIFICAÇÃO AUTOMÁTICA EM CNAE-SUBCLASSES **8**

3 METAS FÍSICAS ALCANÇADAS **10**

3.1 META FÍSICA 1.1/2008: DESENVOLVIMENTO DE MECANISMO DE CODIFICAÇÃO BASEADO EM REDES NEURAIS ARTIFICIAIS – FUNDAMENTAÇÃO DO CÓDIGO 11
3.1.1 Redes Neurais Probabilísticas *11*
3.2 META FÍSICA 1.2/2008: DESENVOLVIMENTO DE MECANISMO DE CODIFICAÇÃO BASEADO EM REDES BAYESIANAS – FUNDAMENTAÇÃO DO CÓDIGO 15
3.2.1 Camada de Documentos de Atividade *15*
3.2.2 Propagação de Crenças *15*
3.2.3 Extensões Usando Heurísticas *16*
3.2.4 Ferramenta Computacional *17*
3.2.5 Resultados *19*
3.2.6 Conclusão *20*
3.3 META FÍSICA 1.3/2008 – DESENVOLVIMENTO DE MECANISMO DE CODIFICAÇÃO BASEADO EM LATENT SEMANTIC INDEXING – FUNDAMENTAÇÃO DO CÓDIGO 21
3.3.1 Vector Space CUDA (VSC) *21*
3.3.2 Implementação Paralela do VS_CORE em C+CUDA *27*
3.4 META FÍSICA 1.4/2008: DESENVOLVIMENTO DE MECANISMO DE COMPOSIÇÃO DOS RESULTADOS DA CODIFICAÇÃO ATRAVÉS DE REDES NEURAIS ARTIFICIAIS, REDES BAYESIANAS E *LATENT SEMANTINC INDEXING* EM UMA ÚNICA CODIFICAÇÃO, MAIS ROBUSTA – FUNDAMENTAÇÃO DO CÓDIGO 32
3.4.1 Combinação Estática *34*
3.4.2 Combinação Dinâmica *34*
3.4.3 Combinações Empregadas *35*
3.4.4 Combinação Estática no ENSEMBLE *35*
3.4.5 Combinação Dinâmica no ENSEMBLE *35*
3.4.6 Resultados *36*

4 OUTRAS REALIZAÇÕES TÉCNICO-CIENTÍFICAS **37**

4.1 ORGANIZAÇÃO E PARTICIPAÇÃO EM EVENTOS CIENTÍFICOS 37
4.1.1 COLA 2009 ..... *37*
4.1.2 ESANN 2009: Sessão Especial em Weightless Neural Systems *37*
4.2 PUBLICAÇÕES 39
4.3 ORIENTAÇÕES 40
4.3.1 Orientações em Andamento *40*
4.3.2 Orientações Concluídas *41*

5 PARTICIPAÇÃO DA EQUIPE CIENTÍFICA EM ENCONTROS RELEVANTES **42**

5.1 V ENAT 42
5.1.1 Reunião da Subcomissão CNAE *42*
5.1.2 Reunião para apresentação do SCAE para o Secretário Adjunto da RFB *42*

Fundação Espírito-Santense de Tecnologia
Av. Fernando Ferrari, 845 | Goiabeiras | 29.060-410 - Vitória - ES
Tel: +55 27 3345-7555 | Fax: +55 27 3345-7668 | www.fest.org.br

5.1.3 Apresentação do Projeto SCAE em Sessão do V ENAT *42*

6 LIÇÃO APRENDIDA NO PERÍODO **43**

6.1 QUANTO À IMPORTÂNCIA DA MANUTENÇÃO DAS NORMAS E REGULAMENTOS 43

7 BIBLIOGRAFIA **44**

8 APÊNDICE: PROTÓTIPO DO SCAE-FISCAL – ATUALIZAÇÃO **46**

8.1.1 Preparação para a Instalação do SCAE *47*
8.1.2 Instalando o SCAE *50*
8.1.3 Configuração *52*
8.1.4 Uso *63*

Fundação Espírito-Santense de Tecnologia
Av. Fernando Ferrari, 845 | Goiabeiras | 29.060-410 - Vitória - ES
Tel: +55 27 3345-7555 | Fax: +55 27 3345-7668 | www.fest.org.br

Índice de Figuras

Figura 3-1 : Arquitetura da RNP. 11
Figura 3-2: Gráfico de comparação da evolução das GPUs com relação às CPUs (NVIDIA, 2008b) 23
Figura 3-3: Gráfico da evolução da banda de memória (NVIDIA, 2008b) 23
Figura 3-4: Arquitetura de um Stream Multi-Processor (NVIDIA, 2006) 24
Figura 3-5: Arquitetura da GeForce GTX 280 (NVIDIA, 2008a) 25
Figura 3-6: Arquitetura da linguagem C-CUDA (HALFHILL, 2008) 25
Figura 3-7: Hierarquia de *threads* em C-CUDA (NVIDIA, 2007) 26
Figura 3-8: Exemplo de como as *threads* fazem o produto matriz x vetor 28
Figura 3-9: Código do produto matriz x vetor 29
Figura 3-10: Exemplo da soma em árvore (*sum tree like reduction*) 30
Figura 3-11: Código da soma em árvore (*sum tree like reduction*) 30
Figura 3-12: Inclusão de núcleos no SCAE 32
Figura 3-13: Interligação do ENSEMBLE com outros núcleos do SCAE 33
Figura 3-14: Classificação pelo ENSEMBLE 33
Figura 3-15: Combinação estática no ENSEMBLE 35
Figura 8-1: Arquitetura do SCAE 46
Figura 8-2: Processo de criação de tabelas do SCAE 53
Figura 8-3: Ações realizadas no treino do CORE 62
Figura 8-4: Ações realizadas no teste do CORE 65
Figura 8-5: Interface Web de classificação de atividades. 67
Figura 8-6: Classificação da atividade Cultivo de arroz com o CORE WNN_COR pelo browser 68
Figura 8-7 - Classificação de atividade econômica com correção ortográfica pelo *browser*. . 69

Fundação Espírito-Santense de Tecnologia
Av. Fernando Ferrari, 845 | Goiabeiras | 29.060-410 - Vitória - ES
Tel: +55 27 3345-7555 | Fax: +55 27 3345-7668 | www.fest.org.br

Índice de Tabelas

Tabela 3-1: Experimentos realizados para avaliar a RNP 14
Tabela 3-2: Resultados experimentais obtidos com a RNP 14
Tabela 3-3: Resultados dos experimentos 19
Tabela 3-4: Resultados detalhados 20
Tabela 3-5: Experimentos com o classificador VSC_CORE 31
Tabela 3-6: Desempenho do classificador VSC_CORE 31
Tabela 3-7: *Speed-up* do algoritmo VSC_CORE 31
Tabela 3-8: Comparativo entre a combinação estática e a dinâmica 36
Tabela 8-1: Lista de Tabelas de Dicionários do SCAE 55
Tabela 8-2: Exemplo de nomes para o *lexicon* 57
Tabela 8-3: Exemplo de descrições para o *lexicon* 57
Tabela 8-4: Limites das Tabelas existentes atualmente no SCAE 58
Tabela 8-5: Operações de Filtro do SCAE 59
Tabela 8-6: Scripts de treino dos CORES 61
Tabela 8-7: Relação entre o nome do CORE e os diretórios de treino. 63
Tabela 8-8: Scripts de teste dos CORES 64

Fundação Espírito-Santense de Tecnologia
Av. Fernando Ferrari, 845 | Goiabeiras | 29.060-410 - Vitória - ES
Tel: +55 27 3345-7555 | Fax: +55 27 3345-7668 | www.fest.org.br

1 Introdução

Este documento descreve as atividades levadas a cabo visando o cumprimento das Metas Físicas 1.1/2008, 1.2/2008, 1.3/2008 e 1.4/2008, previstas em convênio firmado em dezembro de 2006 entre a Receita Federal do Brasil (Receita) e a Fundação Espírito-Santense de Tecnologia (FEST). Este convênio (Convênio Receita/FEST) tem como objeto a cooperação tecnológica entre os partícipes visando ao desenvolvimento de estudos e pesquisas que subsidiem a análise da aplicabilidade de técnicas de Inteligência Computacional na classificação automática de documentos, sendo os documentos de interesse descrições de atividades econômicas dos agentes econômicos e a classificação de interesse a identificação de códigos da tabela CNAE-Subclasses (Classificação Nacional de Atividades Econômicas – Subclasses Fiscais) correspondentes às atividades econômicas descritas nos documentos. Por meio deste convênio foi viabilizada a realização do Projeto de Pesquisa "Classificação Automática em CNAE-Subclasses".

Acompanha este Relato um DVD com os códigos e bases de dados desenvolvidos, além de outros documentos relevantes.

As redundâncias observáveis entre este documento e o "Relato de Cumprimento de Metas No. 1" (SCAEa, 2007), o "Relato de Cumprimento de Metas No. 2" (SCAEb, 2007) e o "Relato de Cumprimento de Metas No. 3" (SCAE, 2008), referentes ao mesmo Projeto, visam apenas permitir uma leitura autocontida do presente documento. É importante, contudo, observar que nos documentos anteriores podem ser encontrados complementos e detalhamentos de modelos aqui apresentados e desenvolvidos.

1.1 Motivação e justificativa

No XVI Encontro Nacional de Auditores e Fiscais de Tributos Municipais, ocorrido em 8 e 9 de Julho de 2004 em Manaus, constatou-se que os municípios perdem cada vez mais receitas e albergam cada vez mais atribuições. A correta interpretação e aplicação das leis tributárias municipais é fonte inequívoca de mais e melhores receitas para o município. Uma das formas de se melhorar os mecanismos de arrecadação é a adoção de um sistema consistente de codificação tributária.

O inciso XXII do Art. 37, incluído na Constituição por emenda constitucional em dezembro de 2003, versa que "as administrações tributárias da União, dos Estados, do Distrito Federal e dos Municípios (...) atuarão de forma integrada, inclusive com o compartilhamento de cadastros e de informações fiscais, na forma da lei ou convênio". No 1º Encontro Nacional de Administradores Tributários (ENAT), realizado em Salvador – BA, em julho de 2004, e com a participação das secretarias de fazenda dos estados, capitais e da Receita, foi iniciada a discussão sobre as iniciativas necessárias para o cumprimento desse dispositivo através de ações conjuntas. Neste cenário de atuação integrada, a adoção da CNAE-Subclasses assumiu um papel mandatório para a implementação dos cadastros compartilhados.

No modelo atual, a atribuição de códigos a agentes econômicos segundo a CNAE-Subclasses (classificação em CNAE-Subclasses) é realizada manualmente por diversos agentes codificadores (contabilistas, funcionários de órgãos públicos, etc...) e em diversas fases da

existência do agente econômico, o que resulta numa baixa qualidade de codificação. Neste contexto, o desenvolvimento de uma ferramenta automática para tal finalidade se apresenta como uma estratégia bastante promissora para atacar este problema.

Assim, a principal motivação para a realização do Projeto de Pesquisa "Classificação Automática em CNAE-Subclasses" é a oportunidade que ele oferece de se investigar a solução de um problema prático específico, mas cujas técnicas de solução têm aplicação em diversos outros cenários relevantes. Outra motivação importante é a possibilidade que ele oferece de estender o estado da arte em ciência da informação (classificação de texto), ciência da computação (classificação automática de texto, computação de alto desempenho) e ciência da cognição (representação de conhecimento). Este trabalho se justifica pela importância econômica e governamental da correta, eficiente e eficaz classificação de atividades econômicas.

1.2 Objetivos

O objetivo principal do Projeto de Pesquisa "Classificação Automática em CNAE-Subclasses" é desenvolver ou adaptar algoritmos e heurísticas que viabilizem a implementação de um protótipo de um Sistema Computacional de Codificação Automática de Atividades Econômicas (SCAE) e comparar o desempenho deste sistema com o de codificadores humanos. Também são objetivos deste Projeto implementar um protótipo de um SCAE com interface Web, expandir o estado da arte nas áreas de pesquisa associadas ao projeto e formar pessoal especializado.

1.3 Organização deste documento

Após esta introdução, na Seção 2 apresentamos brevemente o problema científico de interesse, isto é, a classificação automática em CNAE-Subclasses. Na Seção 3, relatamos o cumprimento das Metas Físicas 1.1/2008, 1.2/2008, 1.3/2008 e 1.4/2008. Na Seção 4 apresentamos outras realizações técnico-científicas associadas ao Projeto e, na Seção 5, breve relato da participação da equipe científica em eventos relevantes. Por fim, na Seção 6, discutimos importantes lições aprendidas desde a apresentação do Relato de Cumprimento de Metas No. 3.

Fundação Espírito-Santense de Tecnologia
Av. Fernando Ferrari, 845 | Goiabeiras | 29.060-410 - Vitória - ES
Tel: +55 27 3345-7555 | Fax: +55 27 3345-7668 | www.fest.org.br

2 Classificação Automática em CNAE-Subclasses

A Classificação Nacional de Atividades Econômicas (CNAE), uma tabela hierárquica de atividades econômicas e seus códigos associados, é a classificação oficialmente adotada pelo Sistema Estatístico Nacional e pelos órgãos gestores federais de registros administrativos associados. Com base na Resolução do Presidente do IBGE No. 054, de 19/12/1994, publicada no Diário Oficial da União No. 244, em 26/12/1994, vem sendo implementada desde 1995 pelo Sistema Estatístico Nacional e órgãos da administração federal. A CNAE foi desenvolvida tendo por referência a *International Standard Industrial Classification of All Economic Activities* - ISIC, 3ª revisão, das Nações Unidas. O responsável pela gestão e manutenção da CNAE é o IBGE, a partir das deliberações da Comissão Nacional de Classificação – CONCLA. A partir da elaboração da CNAE, que hoje contempla 672 classes de atividades (CNAE 2.0), foi derivada outra classificação, a CNAE-Subclasses.

A CNAE-Subclasses é um detalhamento das classes da CNAE para uso nos cadastros da administração pública, em especial da administração tributária, nas três esferas do governo. Sua estrutura é igual à estrutura da CNAE, que possui código de 5 dígitos para cada classe, adicionada de um nível hierárquico, codificado com 2 dígitos. Ou seja, as subclasses da CNAE-Subclasses possuem código de 7 dígitos, sendo os dois dígitos adicionais resultantes do detalhamento de cada classe da CNAE. Este detalhamento foi feito especificamente para atender necessidades da organização dos cadastros de pessoas jurídicas no âmbito das três esferas da Administração Pública. Na CNAE 2.0 existem 1301 subclasses.

A CNAE-Subclasses é usada como instrumento de padronização nacional dos códigos de atividade econômica utilizados pelos diversos órgãos públicos da administração direta, facilitando a gerência e o controle de ações da competência de cada esfera. Nos cadastros da administração tributária o código CNAE-Subclasse é atribuído a todos os agentes econômicos que estão engajados na produção de bens e serviços, podendo compreender estabelecimentos de empresas privadas ou públicas, estabelecimentos agrícolas, organismos públicos e privados, instituições sem fins lucrativos, e agentes autônomos (pessoa física). Na Secretaria da Receita Federal, um ou mais códigos CNAE-Subclasse devem ser informados na Ficha Cadastral de Pessoa Jurídica (FCPJ) quando do cadastro de uma nova pessoa jurídica ou quando da alteração dos seus atos constitutivos – a FCPJ alimenta o Cadastro Nacional de Pessoa Jurídica (CNPJ) da Receita Federal do Brasil.

Atualmente, em muitos órgãos usuários, a determinação de quais códigos devem ser atribuídos a cada agente econômico, a codificação em CNAE-Subclasses, é feita manualmente por codificadores humanos treinados para tal, apoiados por ferramentas computacionais de busca em versões eletrônicas da tabela CNAE-Subclasses. O codificador humano treinado deve associar/combinar a informação na tabela CNAE-Subclasses com seu conhecimento, fruto de seus vários anos de educação e experiência profissional, para, com o conjunto, atribuir códigos CNAE-Subclasse para o agente econômico cujas atividades estão sendo codificadas.

Na verdade, para fazer a codificação, o codificador humano precisa compreender quais são as atividades do agente econômico e qual é a correspondência entre elas e os descritores de uma ou mais Subclasses da CNAE-Subclasses. Para operar com eficácia equivalente, um Sistema

Fundação Espírito-Santense de Tecnologia
Av. Fernando Ferrari, 845 | Goiabeiras | 29.060-410 - Vitória - ES
Tel: +55 27 3345-7555 | Fax: +55 27 3345-7668 | www.fest.org.br

Computacional para a Codificação Automática de Atividades Econômicas (SCAE) precisa gerar representações da tabela CNAE-Subclasses e das atividades do agente econômico, internas ao sistema. Estas representações têm que ser tais que permitam identificar a correta correspondência semântica entre a descrição das atividades do agente econômico e um ou mais subclasses/descritores da tabela CNAE-Subclasses. Mecanismos para o enriquecimento da representação interna da tabela também devem ser incorporados ao SCAE de modo a prover um equivalente computacional dos vários anos de educação e experiência profissional do codificador humano.

A codificação automática em CNAE-Subclasses de interesse compreenderá, então, a identificação do(s) descritor(es) de atividade(s) e respectivo(s) código(s) CNAE-Subclasses de um agente econômico a partir:

- das descrições completas de suas atividades econômicas;
- da Tabela CNAE-Subclasses e seus instrumentos de apoio à codificação;
- de uma base de dados representativa de codificações corretas; e
- de um dicionário eletrônico da língua portuguesa e outras bases de dados que permitam criar uma representação interna ao computador das atividades do agente econômico e da tabela CNAE-Subclasses ricas o suficiente para permitir ao computador identificar as correspondências entre as atividades e as entradas da tabela.

Fundação Espírito-Santense de Tecnologia
Av. Fernando Ferrari, 845 | Goiabeiras | 29.060-410 - Vitória - ES
Tel: +55 27 3345-7555 | Fax: +55 27 3345-7668 | www.fest.org.br

3 Metas Físicas Alcançadas

No período de agosto de 2008 a maio de 2009 foram alcançadas as Metas Físicas:

Meta Física 1.1/2008 – Desenvolvimento de mecanismo de codificação baseado em redes neurais artificiais

Meta Física 1.2/2008 – Desenvolvimento de mecanismo de codificação baseado em redes Bayesianas

Meta Física 1.3/2008 – Desenvolvimento de mecanismo de codificação baseado em *Latent Semantic Indexing*

Meta Física 1.4/2008 – Desenvolvimento de mecanismo de composição dos resultados da codificação através de neurais artificiais, redes Bayesianas e *Latent Semantic Indexing* em uma única codificação, mais robusta

Estas metas foram previstas no Plano de Trabalho vigente do Convênio Receita/FEST para o período de novembro de 2008 a maio de 2009 e compreendem avanços previstos sobre tópicos de pesquisa e desenvolvimento já abordados em outras etapas do Plano de Trabalho.

A seguir entregamos os resultados físicos associados a cada meta (códigos, bases de dados, relatórios e outros documentos), conforme previsto no Plano de Trabalho vigente do Convênio Receita/FEST. Este Relato é acompanhado de um DVD com os códigos e bases de dados desenvolvidos, além de outros documentos relevantes.

3.1 Meta Física 1.1/2008: Desenvolvimento de Mecanismo de Codificação Baseado em Redes Neurais Artificiais – Fundamentação do Código

Nesta seção apresentamos uma Rede Neural Probabilística para Categorização de Texto desenvolvida dentro do escopo deste Projeto.

3.1.1 Redes Neurais Probabilísticas

A Rede Neural Probabilística (RNP) é uma rede neural artificial que realiza um mapeamento não linear entre sua entrada e sua saída de forma a se aproximar da decisão ótima de Bayes. Além disso, diferente das redes neurais sem peso (RNSP), esta rede neural pode armazenar valores contínuos nas sinapses de seus neurônios.

A RNP foi originalmente projetada para problemas de único rótulo (SPECHT, 1990). Assim, sua arquitetura padrão foi modificada ligeiramente de forma a ser capaz de tratar problemas multi-rotulados (OLIVEIRA; et al, 2008). Nesta versão multi-rotuladora da rede, a RNP é composta por três camadas: a camada de entrada, a camada de padrões e a camada de soma, como ilustrada na Figura 3-1. O número de neurônios da camada de soma é igual ao número de categorias em que a rede pode classificar. Assim como na estrutura original, o treinamento desta RNP é realizado em um único passo; desta forma, seu treinamento é muito mais rápido se comparado às outras redes neurais do tipo *feed-foward* (DUDA; et al, 2001).

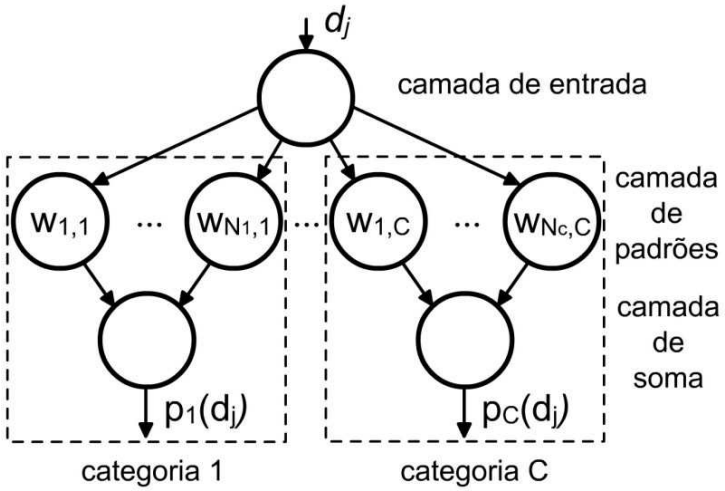

Figura 3-1 : Arquitetura da RNP.

O treinamento dela consiste em associar cada amostra de treinamento *w* da categoria *i* a um neurônio da camada de padrões da categoria *i*. Assim, o vetor de pesos, ou sinapses, deste neurônio é o vetor de características da amostra. Como consequência, o número de neurônios da camada de padrões da categoria *i* é igual ao número de amostras de treino do mesmo.

Na fase de classificação é apresentada uma amostra d_j (um documento convertido para uma representação vetorial) à camada de entrada, conforme mostra a Figura 3-1. Na camada de entrada não é realizada nenhuma operação, ela simplesmente envia a amostra d_j para os neurônios da camada de padrões. Nesta camada é realizado, em cada neurônio, o cálculo de uma Gaussiana como ilustrado na Equação 3-1.

$$F_{k,i}(d_j) = \frac{1}{2\pi\sigma^2}\exp\left(\frac{d_j^{\ T} w_{k,i} - 1}{\sigma^2}\right) \qquad k = 1,\ldots,N_i \qquad i = 1,\ldots,C$$

Equação 3-1

Na Equação 3-1 temos que d_j é a amostra de entrada que desejamos classificar, $w_{k,i}$ é a k-éssima amostra de treinamento da categoria i, N_i é o número de neurônios da camada de padrões da categoria i e C é o número total de categorias. Os vetores d_j e $w_{k,i}$ devem estar normalizados de forma que $d_j^T d_j = 1$ e $w_{k,i}^T w_{k,i} = 1$. O sigma (σ) é o desvio padrão da Gaussiana e determina o campo receptivo da curva da Gaussiana do neurônio. Valores pequenos de σ causam uma atuação do neurônio muito ruidosa e sensível; enquanto que, para valores altos de sigma, a atuação do neurônio é muito suave e causa perda de detalhes.

A próxima camada da RNP é a camada de soma. Nesta camada todas as saídas da camada anterior são somadas em cada grupo i, conforme a Equação 3-2, produzindo os valores de $p_i(d_j)$.

$$p_i(d_j) = \sum_{k=1}^{N_i} F_{k,i}(d_j) \qquad i = 1,\ldots,C$$

Equação 3-2

Finalmente, para a seleção das categorias que serão associadas a cada amostra d_j através desta rede neural, consideramos categorias cuja saída da camada de soma seja maior ou igual a um limiar τ_i escolhido. Se $p_i(d_j) >= \tau_i$, então i é associada a d_j.

Diferentemente de outros tipos de redes, a RNP proposta necessita de poucos parâmetros para ser configurada, quais sejam, o σ (veja a Equação 3-1) e os limiares τ_i. Outra vantagem das redes neurais probabilísticas é que é fácil adicionar novas categorias, ou novas amostras de treinamento, em uma estrutura já em funcionamento (DUDA; et al, 2001). Por outro lado, uma de suas desvantagens é o grande número de neurônios na camada de padrões que, como conseqüência, pode produzir um consumo alto de memória e taxa lenta de classificação. Além disso, a presença de amostras repetidas pode prejudicar o desempenho do classificador.

3.1.1.1 Categorização de Documentos com a RNP proposta

Os passos necessários para se classificar documentos textuais com a RNP proposta são descritos a seguir.

Inicialmente, o conjunto de documentos, tanto de treino quanto os que irão ser classificados, deverão ser representados, cada um, na forma de um vetor multidimensional $V = w = \{v_1, v_2, \ldots, v_{|V|}\}$, onde cada elemento v_i corresponde a um peso associado a um termo específico do vocabulário de interesse. Em seguida, tais vetores, de treino e os que serão classificados, são normalizados, de forma a obter $V^T V = 1$.

Em seguida, uma RNP (veja Figura 3-1) é treinada com os documentos de treino de acordo com o que foi previamente mencionado. Se, por exemplo, um documento de treino *w* estiver associado às classes *1* e *3*, então é inserido um neurônio tanto na camada de padrões da categoria *1*, quanto na da categoria *3*, cujas sinapses serão iguais aos próprios elementos de *w*. Cada neurônio da camada de soma representará uma categoria. Assim, se houver 10 categorias, haverá 10 neurônios na camada de soma. Além disso, um valor de sigma (σ) deve ser escolhido.

A seguir é realizado o passo de classificação, onde um vetor d_j, forma vetorial do documento que se deseja classificar, é apresentado à camada de entrada do classificador. Esta camada passa d_j para a próxima camada – a camada de padrões – onde são realizados os cálculos da Equação 3-1. Tais resultados passam para a camada de soma, modelada pela Equação 3-2. A saída da rede é um vetor, onde cada elemento é a saída de um neurônio da camada de soma, que corresponde ao grau de pertinência de uma determinada categoria ao documento d_j. Este vetor é normalizado, de forma que os valores nele contidos estejam na faixa entre 0 e 1. Finalmente, um valor limite τ_t para cada categoria *i* é empregado a fim de definir o conjunto de categorias a serem atribuídas ao documento d_j, de tal forma que, se $p_i(d_j) >= \tau_t$, então *i* é atribuída ao documento.

A RNP apresentada foi implementada e adicionada ao SCAE, e corresponde ao core PNN_CORE, onde o acrônimo PNN vem do inglês *Probabilistic Neural Network*.

3.1.1.2 Experimentos e Resultados

Para avaliação da Rede Neural Probabilística (RNP) foi utilizada uma base de dados composta de 3281 documentos, que representam descrições de atividades econômicas de empresas da cidade de Vitória-ES, e 1183 definições das subclasses da tabela CNAE.

Antes de ser utilizada, foi realizado um pré-processamento desta base de dados. A partir deste pré-processamento foram retiradas as preposições existentes nas descrições de atividades econômicas e cada documento foi representado como um vetor no espaço $\Re^n$, conforme feito em (OLIVEIRA, et al, 2007), onde *n* é o número total de termos distintos encontrados na base de dados. Após isso, foi computado o peso de cada termo segundo a técnica TFIDF (SEBASTIANI, 2002).

Foram realizados oito experimentos sendo que a divisão da base de dados em cada um deles foi realizada conforme mostrado na Tabela 3-1 (a última coluna da Tabela 3-1 exibe o desempenho segundo a métrica 1-*One Error* em formato percentual (SCAE, 2008)).

Fundação Espírito-Santense de Tecnologia
Av. Fernando Ferrari, 845 | Goiabeiras | 29.060-410 - Vitória - ES
Tel: +55 27 3345-7555 | Fax: +55 27 3345-7668 | www.fest.org.br

Tabela 3-1: Experimentos realizados para avaliar a RNP

Exp.	Dados de Treino			Dados de Teste				Rev. do SCAE	Desem-penho
	Tabela	Coluna	Limites	Tabela	Coluna	Limites	Nível		
7.1.1.1	110_SUBCL	DESCR_SUB	0 a 1182	110_SUBCL	DESCR_SUB	0 a 1182	SUBCL	758	99,15%
7.1.1.2	110_SUBCL	DESCR_SUB	0 a 1182	DADOS_VIX_110	OBJ_SOCIAL	0 a 3280	SUBCL	758	63,00%
7.1.1.3	110_SUBCL	DESCR_SUB	0 a 1182	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	SUBCL	758	67,03%
7.1.1.4	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	SUBCL	758	83,61%
7.1.1.5	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	CLASSE	758	86,17%
7.1.1.6	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	GRUPO	758	88,91%
7.1.1.7	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	DIVISÃO	758	92,38%
7.1.1.8	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	SEÇÃO	758	94,58%

Para todos os experimentos foram utilizados o valor de sigma igual a 0,25, que é o valor padrão caso não seja fornecido nem um parâmetro de entrada. Ao total foram usadas 13 métricas, sendo que para as métricas *One Error*, *Ranking Loss*, *Coverage*, *Hamming Loss* e *R-Hamming Loss* quanto menor for o valor delas, melhor será o desempenho (SCAE, 2008). Para as métricas restantes, quanto maior for, melhor o desempenho do classificador como um todo. Na Tabela 3-2 são mostrados os resultados obtidos com cada uma destas métricas.

Tabela 3-2: Resultados experimentais obtidos com a RNP

Experimento	7.1.1.1	7.1.1.2	7.1.1.3	7.1.1.4	7.1.1.5	7.1.1.6	7.1.1.7	7.1.1.8
One Error	0,008453	0,370009	0,329677	0,163924	0,138330	0,110908	0,076173	0,054235
Ranking Loss	0,000014	0,114596	0,097657	0,017142	0,012479	0,012374	0,013385	0,017602
Coverage	0,008453	277,777802	249,589890	65,765388	24,106033	8,975625	2,608775	1,135893
Average Precision	0,995773	0,526034	0,564376	0,716809	0,772474	0,823537	0,894472	0,939557
R-Precision	0,991547	0,485539	0,524440	0,659437	0,707566	0,755282	0,830146	0,889671
Hamming Loss	0,000014	0,004074	0,003911	0,003025	0,004285	0,007494	0,014233	0,026598
R-Hamming Loss	0,016906	1,028922	0,951120	0,681127	0,584867	0,489436	0,339707	0,220658
Microaveraged Precision	0,991547	0,434082	0,459567	0,582005	0,639780	0,688000	0,780013	0,863402
Microaveraged Recall	0,991547	0,434082	0,459567	0,582005	0,639780	0,688000	0,780013	0,863402
Macroaveraged Precision	0,991547	0,485539	0,524440	0,659437	0,707566	0,755282	0,830146	0,889671
Macroaveraged Recall	0,991547	0,485539	0,524440	0,659437	0,707566	0,755282	0,830146	0,889671
Microaveraged F1	0,991547	0,434082	0,459567	0,582005	0,639780	0,688000	0,780013	0,863402
Macroaveraged F1	0,991547	0,485539	0,524440	0,659437	0,707566	0,755282	0,830146	0,889671

3.2 Meta Física 1.2/2008: Desenvolvimento de Mecanismo de Codificação Baseado em Redes Bayesianas – Fundamentação do Código

Como descrito no relatório anterior (SCAEb, 2007), foi desenvolvido uma ferramenta de categorização CNAE baseada em teoria de redes *Bayesianas*. O desempenho desta ferramenta mostrou-se relativamente limitado principalmente devido à dificuldade de se incluir documentos multi-rotulados (*multi-label*) na fase de treinamento. Nesta seção, apresentamos uma solução encontrada, ou seja, uma nova arquitetura de nossa rede *Bayesiana*. A novidade principal consiste na introdução de uma nova camada intermediária de nós na rede. Para conseguir isso, foi necessário desenvolver uma técnica para se propagar crenças através desta camada.

Na próxima subseção, apresentamos a nova camada de nós. Posteriormente, descrevemos o mecanismo de propagação de crenças utilizando esta camada. Nas subseções finais, descrevemos os elementos básicos da ferramenta computacional que implementa a nova rede, os resultados obtidos com ajuda da nova arquitetura e nossas conclusões.

3.2.1 Camada de Documentos de Atividade

A dificuldade principal na realização do treinamento *multi-label* foi uma divergência na propagação de crenças devido às categorizações complementares nos dados de treinamento. Por exemplo, se uma empresa informa além de sua atividade principal, digamos "fabricação de sorvetes", uma atuação na área financeira, qualquer outra atividade referindo aos sorvetes aumentará a crença relativa à atividade financeira, isto é, o termo "sorvete", sozinho, já faz uma referência à atividade financeira. Portanto, precisaremos de alguns termos (nós na rede) compostos, tais como "sorvete e fundo de investimento" para as referências adequadas.

Uma solução natural é aproveitar os próprios documentos de atividades utilizados na fase de treinamento com suas categorizações conhecidas. Desta forma, para um documento ser categorizado, procuraremos primeiro os documentos mais próximos e escolheremos subclasses mais prováveis entre aquelas ligadas a estes documentos. Neste contexto, nossa rede *Bayesiana* será utilizada no primeiro passo para recuperação de documentos e só no segundo passo para a própria categorização.

3.2.2 Propagação de Crenças

A inclusão de uma camada intermediária de documentos na estrutura da rede *Bayesiana* implica que devemos definir como propagar as crenças associadas nos documentos encontrados. Inicialmente, aplicamos os mesmos cálculos de propagação entre os nós de termos e os nós de atividades que anteriormente foram utilizados entre os nós de termos e os nós de subclasses (SCAEb, 2007). Isto é, definimos primeiro os pesos,

$$w_{ij} = \alpha^{-1} \frac{tf_{ij} \times idf_i^2}{\sqrt{\sum_{T_k \in R_{\pi(D_j)}} tf_{kj} \times idf_k^2}},$$

que são utilizados na soma ponderada das crenças associadas aos nós pais do documento D_i:

$$p(d_j|Q) = \sum_{T_i \in R_{\pi(D_j)}} w_{ij} p(t_i|Q).$$

Aqui, d_j representa um valor possível da variável randômica D_j.

Na definição dos pesos, tf_{ij} é a freqüência do termo T_i no documento D_j e o valor idf_i é definido pela equação

$$idf_i = log(\frac{N}{n_i}).$$

Aqui, N é o número total de documentos no treinamento e n_i o número de documentos que incluem o termo T_i. Comparando esta definição com aquela apresentada no relatório anterior, observamos que a interpretação agora está relacionada aos documentos, não às subclasses como considerado anteriormente.

Em princípio, agora calculamos a crença final de uma subclasse adicionando as crenças dos seus "pais" na rede, isto é, as crenças dos documentos utilizados no treinamento, que possuem aquela subclasse entre suas subclasses conhecidas.

3.2.3 Extensões Usando Heurísticas

No artigo de CAMPOS et al, 2003, discute-se alternativas para calcular os valores de crenças na camada de documentos. Em nosso trabalho, propomos extensões que melhoram os resultados de forma significativa. Por enquanto, as escolhas feitas baseiam-se nos testes preliminares, sem comparações sistemáticas.

3.2.3.1 Propagação de Crença dos Termos para um Documento

Inicialmente, calculamos a soma ponderada *s* apresentada na subseção anterior. Calculamos também a soma *tdfIdfSum* dos valores *tf * idf * idf* dos termos "pais" do documento e posteriormente os valores:

*p = nObserved * tfIdfSum / nTerms;*

*q = nObserved / Math.sqrt (nQueryTerms * nTerms);*

e finalmente

*crença = p * q * s.*

Aqui, *nTerms* é o número de termos no documento, dos quais *nObserved* termos são comuns aos termos do documento a ser classificado, que tem *nQueryTerms* termos. Nota-se que o valor *q* é de fato o valor do *cosseno* entre os vetores de ocorrências do documento e o documento a ser classificado.

De forma alternativa, cada elemento na soma *tfIdfSum* é multiplicado pelo incremento da crença de cada termo "pai".

3.2.3.2 Propagação de Crença do Documento para uma Subclasse

Voltamos a examinar os termos "pai" do documento e calculamos o valor médio dos valores *idf * idf*, denotado por *meanIdf* e o valor:

$$p = 0.001 * meanIdf * Math.log1p(meanIdf).$$

A soma dos valores de crença acumulados na subclasse será multiplicada por *p*. O coeficiente 0.001 é inserido somente para manter o resultado legível mais facilmente.

Podemos admitir que os cálculos de crença apresentados sejam de natureza heurística, sem fundamento teórico. Será objeto de uma pesquisa futura entender melhor por que os resultados sem eles são tão fracos.

3.2.4 Ferramenta Computacional

O sistema BN_CORE desenvolvido compõe-se de três partes definidas pelas suas funções: (1) interface com o sistema SCAE, (2) módulo de treinamento e (3) módulo de categorização. O sistema é escrito em linguagem de programação Java. Considerando a característica orientada a objeto da linguagem, os componentes em arquivos são denominados classes e as operações definidas para as classes, métodos.

3.2.4.1 Interface SCAE – BN_CORE

A interface é implementada pela classe Java *RPCHandler*, que define os métodos para iniciar e finalizar treinamentos e testes, salvar e recarregar a rede construída no treinamento, como também um método para responder a chamada de classificação de um único documento originada na interface Web do SCAE.

O treinamento é inicializado pelo método *beginTraining*(), que faz somente algumas inicializações. O método *trainLine*() recebe informações sobre um documento de atividade na forma de dois vetores, as freqüências *tf* dos termos e os códigos CNAE associados ao documento. O método gera as instâncias das classes *ActivityDoc*, *Term* e *DocClass* que correspondem ao documento dado, aos seus termos e às subclasses CNAE associadas, respectivamente.

O treinamento começa somente após terem sido recebidos todos os documentos de atividade de entrada e é realizado pelo método *endTraining*(), pois precisamos do conhecimento de todos os elementos participantes para construir a rede. Após algumas operações iniciais, o método gera uma instância da classe *NetworkBuilder* e chama o seu método principal *doJob*(). Este método retorna uma rede *Bayesiana*. O método *save*() captura o resultado e constrói uma serialização da rede, isto é, uma representação compacta para ser salva em um arquivo.

A categorização dos documentos inicia com a chamada do método *reload*(), que gera uma instância da classe *BayesNetwork*, lê a rede serializada e recarrega os seus objetos. O método *beginTest*() somente inicia o contador de documentos recebidos pelas chamadas sucessivas do método *testLine*(), que passa a tarefa de categorização para a instância da classe *BayesNetwork* recarregada. O método *endTest*() é implementado apenas para responder à chamada correspondente do SCAE.

Outro método para categorização, *classify*(), é também implementado pela classe *RPCHandler*.

3.2.4.2 Módulo de Treinamento

A classe Java responsável pela construção de uma rede *Bayesiana* é a classe *NetworkBuilder* com seu método público *doJob*(). Nele, define-se dois processadores auxiliares, *PrimProcessor* e *PolyTreeProcessor*. O primeiro é responsável pela construção de uma árvore geradora máxima, onde os nós são os termos e os pesos associados aos arcos são os valores de dependência entre os termos.

A tarefa do *PolyTreeProcessor* é analisar se as direções de alguns arcos da árvore construída deveriam ser trocadas. O resultado é uma poliárvore, isto é, um grafo direcionado onde cada par de nós tem apenas um caminho entre eles.

Com uma poliárvore pronta, o *NetworkBuilder* cria tabelas de probabilidades condicionais para os termos e calcula os pesos para definir a propagação de crença dos nós de termo para os nós representando os documentos de atividade.

No final, o método *doJob*() retorna uma instância da rede *Bayesiana*.

3.2.4.3 Módulo de Categorização

A classe *BayesNetwork* oferece dois métodos para categorização de um documento de atividade: *testLine*() e *classifyDocument*().

O método *testLine*() extrai os termos do documento do vetor de valores *tf* e as subclasses conhecidas do vetor de códigos CNAE. Uma instância da classe *ActivityDoc* é gerada antes da chamada do método *classifyDoc*(), um método interno para categorização. O método *classifyDoc*() realiza uma seqüência de chamadas dos métodos apresentados a seguir:

- *initNetwork*() / *reInitNetwork*(): inicialização e reinicialização dos valores de variáveis utilizados na propagação de crenças na rede Bayesiana;
- *updateNetwork*(): atualização dos valores de variáveis de propagação pelo conhecimento dos termos presentes no documento a ser categorizado; e início da propagação pelas chamadas dos métodos *sendLambdaMessage*() e *sendPiMessage*() definidas pela classe *Node* (uma superclasse da classe *Term*) e aplicadas agora aos termos observados no documento;
- *propagateToActivityDocLevel*(): após a finalização da propagação de crença na poliárvore de termos, os valores de crença para todas as instâncias da classe *ActivityDoc* são calculados, isto é, cada termo propaga sua crença aos documentos associados a ele durante o treinamento;
- *collectActivityDocResult*(): as instâncias de classe *ActivityDoc* são inseridas numa lista de prioridades, ordenando-as em ordem não-crescente relativa à crença acumulada. A ordenação foi estabelecida com o propósito de não utilizar todos os documentos no cálculo de crenças das instâncias da classe *DocClass*. Se todos fossem utilizados, como é o caso da versão atual, a ordenação não seria necessária;

- *propagateToDocumentClasses*(): as instâncias da classe *ActivityDoc* propagam suas crenças para as instâncias da classe *DocClass* a elas associadas;
- *collectDocClassResults*(): também uma opção de ordenar as instâncias da classe *DocClass* na ordem da maior crença para a menor. Quando os valores de crença são retornados em um vetor na ordem original das instâncias, a ordenação não será necessária.

O método *classifyDocument*() chamado por *RPCHandler* é semelhante ao método interno *classifyDoc*. A única diferença é na construção de índices a serem retornados.

3.2.5 Resultados

Na Tabela 3-3, os resultados dos experimentos utilizados para avaliar o desempenho de categorização de nossa nova rede são apresentados no formato utilizado no relatório anterior (SCAEb, 2007). Além das colunas relativas aos dados de treino e teste, a última coluna mostra o desempenho do sistema através do valor 1-*OneError* em percentagem.

Tabela 3-3: Resultados dos experimentos

Exp.	Dados de Treino			Dados de Teste				Rev. do SCAE	Des-emp.
	Tabela	Coluna	Limites	Tabela	Coluna	Limites	Nível		
4.1.1.1	110_SUBCL	DESCR_SUB	0 a 1182	110_SUBCL	DESCR_SUB	0 a 1182	SUBCL	744	99,16%
4.1.1.2	110_SUBCL	DESCR_SUB	0 a 1182	DADOS_VIX_110	OBJ_SOCIAL	0 a 3280	SUBCL	744	66,29%
4.1.1.3	110_SUBCL	DESCR_SUB	0 a 1182	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	SUBCL	744	70,32%
4.1.1.4	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	SUBCL	744	85,74%
4.1.1.5	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	CLASSE	744	87,51%
4.1.1.6	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	GRUPO	744	90,80%
4.1.1.7	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	DIVISÃO	744	93,60%
4.1.1.8	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	SEÇÃO	744	95,55%

Na Tabela 3-4, são compilados os resultados detalhados relativos aos mesmos experimentos (outras métricas de desempenho apresentadas em (SCAEb, 2007).

Tabela 3-4: Resultados detalhados

Experimento	4.1.1.1	4.1.1.2	4.1.1.3	4.1.1.4	4.1.1.5	4.1.1.6	4.1.1.7	4.1.1.8
One Error	0.008453	0.337092	0.296770	0.142596	0.124924	0.092017	0.063985	0.044485
Ranking Loss	0.000014	0.134291	0.115438	0.015276	0.011193	0.010679	0.012179	0.014784
Average Precision	0.995773	0.591237	0.620359	0.748653	0.797928	0.850725	0.911907	0.952041
R-precision	0.991547	0.511754	0.546769	0.685941	0.731164	0.781129	0.853547	0.910349
Hamming Loss	0.000014	0.003765	0.003577	0.002732	0.003797	0.006530	0.011929	0.021615
Microaveraged Precision	0.991547	0.476909	0.505701	0.622561	0.680915	0.728271	0.815916	0.889421
Microaveraged Recall	0.987320	0.255898	0.260725	0.258709	0.272837	0.399862	0.579067	0.683497
Macroaveraged Precision	0.991547	0.475236	0.505125	0.622295	0.680553	0.727771	0.815134	0.888439
Macroaveraged Recall	0.991547	0.268924	0.268016	0.212051	0.229129	0.323632	0.488514	0.617110
Microaveraged F1	0.991547	0.476071	0.505413	0.622428	0.680734	0.728021	0.815525	0.888930
Macroaveraged F1	0.988729	0.228557	0.235562	0.217790	0.236219	0.342947	0.510607	0.638403

3.2.6 Conclusão

Dando seqüência no desenvolvimento do projeto, o presente relatório descreve uma nova arquitetura da rede *Bayesiana,* que insere uma camada adicional entre os nós de termos e nós de subclasses. Neste estágio nosso objetivo foi tornar possível o uso da estratégia de treinamento *multi-label*. Adicionalmente, alguns procedimentos foram desenvolvidos com o propósito de destacar as diferenças entre os valores de crença produzidos pelo mecanismo de propagação da rede *Bayesiana*.

3.3 Meta Física 1.3/2008 – Desenvolvimento de Mecanismo de Codificação Baseado em Latent Semantic Indexing – Fundamentação do Código

Nesta seção, apresentamos o algoritmo Vector Space CUDA (VSC), uma versão em paralelo do algoritmo VS. Também descreveremos brevemente a biblioteca *Compute Unified Device Architecture* (C-CUDA) e a arquitetura *Graphics Processor Unit* (GPU).

3.3.1 *Vector Space* CUDA (VSC)

Com a evolução e o grande poder computacional das *Graphics Processor Units* (GPUs), elas passaram a ser usadas para vários propósitos. Em 2006, a NVIDIA lançou a arquitetura CUDA (*Compute Unified Device Architecture*). A partir disso, o uso das GPUs para processamento não gráfico ganhou força na comunidade científica, sendo publicados muitos trabalhos sobre sua aplicação em diversas áreas e, inclusive, em áreas correlatas ao projeto SCAE (GARCIA et al., 2008). Em alguns casos, o desempenho chega a ser 100 vezes maior que o de *Central Processor Units* (CPU).

Além do grande desempenho, GPUs ocupam pouco espaço, são fáceis de instalar e configurar, e são mais baratas se comparadas a um *cluster*. Uma GPU GeForce GTX 280, que atualmente é a topo de linha, possui desempenho máximo teórico de 933 GFlops/s, enquanto que o *cluster* antigo do Laboratório de Computação de Alto Desempenho (LCAD), com 64 computadores Athlon 1800, chegava a apenas 195,8 Gflop/s. A grande limitação das GPU em comparação com um *cluster* é a quantidade de memória. A GTX 280 possui 1GB, enquanto que o cluster antigo do LCAD possuía 16 GB (LCAD, 2009; NVIDIA, 2008b).

A NVIDIA (principal fabricante de GPUs) criou CUDA para facilitar a programação e o uso das placas de vídeo na execução de programas de propósito geral. Muito embora GPUs sejam diferentes de CPUs no que diz respeito à arquitetura e à forma de programar, elas já vêm sendo usadas em aplicações não gráficas há algum tempo (FUNG; MANN, 2004).

O algoritmo do categorizador VS do SCAE (SCAEb, 2007) foi implementado em paralelo usando C+CUDA. O novo algoritmo é 2,62 vezes mais rápido do que o anterior (considerando todo o sistema SCAE).

3.3.1.1 *Graphics Processor Unit* (GPU)

Nesta seção, apresentamos um pouco da história de como e quando as principais GPUs surgiram. Também, descrevemos brevemente a arquitetura de uma GPU e como ela funciona depois da unificação do *pipeline* gráfico. Finalmente, apresentamos uma descrição da extensão de C para a programação de GPUs CUDA *enabled* (ambiente de programação em C+CUDA) e as principais diferenças em relação à linguagem C padrão, além de particularidades associadas à programação em C+CUDA.

3.3.1.2 História das GPUs

Em 1987, a IBM lançou no mercado o primeiro *chip* gráfico chamado de *Video Graphics Array* (VGA). Ele era conhecido como *dumb* (pouco inteligente) *frame buffer*, pois as atualizações de todos os *pixels* eram feitas pela CPU (RANDIMA; MARK, 2003).

Antes das primeiras gerações de GPUs, as empresas Silicon Graphics (SGI) e Evans & Sutherland desenvolviam *hardwares* gráficos especializados. Essas companhias desenvolveram vários conceitos, tais como transformação de *vertex* e mapeamento de textura. Entretanto, os *hardwares* gráficos não tiveram muita aceitação, devido a seu custo elevado (RANDIMA; MARK, 2003).

Por volta de 1998, vieram as primeiras GPUs incluindo a NVIDIA TNT2, ATI Rage e as 3dfx Voodoo3. Estas GPUs eram capazes de pré-transformar os triângulos e aplicar uma ou duas texturas. Elas também eram implementadas com as características do DirectX 6. No entanto, estas GPUs não eram capazes de transformar *vertex* de objetos 3D; assim, estas transformações precisavam ser feitas pela CPU (RANDIMA; MARK, 2003).

Entre 1999 e 2000, a NVIDIA lançou a GPUs GeForce 256, GeForce 2, a ATI a Radeon 8500 e a S3 a Savage3D. Estas GPUs eram capazes de realizar a transformação de *vertex* 3D e manipulação de luz tirando estas tarefas da CPU. A transformação rápida dos *vertexes* foi uma das principais capacidades que diferenciou as GPUs desta geração das anteriores. Por outro lado, muito embora essas GPUs pudessem ser configuradas, elas ainda não eram verdadeiramente programáveis (RANDIMA; MARK, 2003).

Em 2001, a NVIDIA apresentou a Geforce 3 e a ATI a Radeon 9500. Esta geração passou a proporcionar a programação de operações sobre *vertex* em vez de simplesmente oferecer mais configurabilidade. Em 2003, a NVIDIA lança sua quinta geração de GPUs, a Geforce FX, e a ATI a Radeon 9700. Esta geração passou a ser programável no nível de *pixel* e *vertex*. O lançamento da linguagem Cg nessa mesma época facilitou essa programação, uma vez que ela é bastante parecida com a linguagem de programação C. A partir daí, alguns pesquisadores passam a usá-las para processamentos não gráficos. Contudo, programar para propósito geral utilizando Cg se mostrou muito complexo (APIs) OpenGL e DirectX (RANDIMA; MARK, 2003).

Em 2006, a NVIDIA lançou no mercado as novas GPUs, série 8, com *shader* (vários passos do *pipeline* da GPU, como, por exemplo, *vertex shader* e *pixel shader*) unificado, que receberam a denominação de *stream processors* (NVIDIA, 2006). Nesta mesma época, a NVIDIA também lança a extensão da linguagem C para a programação de GPUs CUDA *enabled*. Com tanto poder computacional, e agora com uma linguagem de programação mais simplificada, as GPUs passam a ser usadas não só para processamento gráfico, mas também para computação de propósito geral (HALFHILL, 2008)

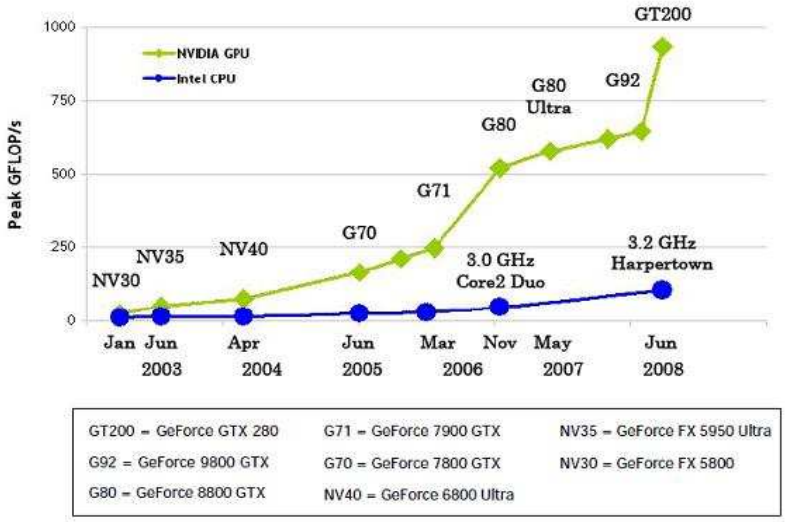

Figura 3-2: Gráfico de comparação da evolução das GPUs com relação às CPUs (NVIDIA, 2008b)

Em poucos anos, o poder computacional bruto das GPUs superou em muito o das CPUs, como mostra a Figura 3-2. Com vários *cores* e uma grande largura de banda de memória, elas se tornaram poderosos recursos tanto para processamento gráfico quanto para o de propósito geral (NVIDIA, 2007). A Figura 3-3 mostra como a taxa de transferência dos dados pelo barramento de memória das GPUs cresceu; este crescimento foi necessário, pois as GPUs têm *caches* pequenas.

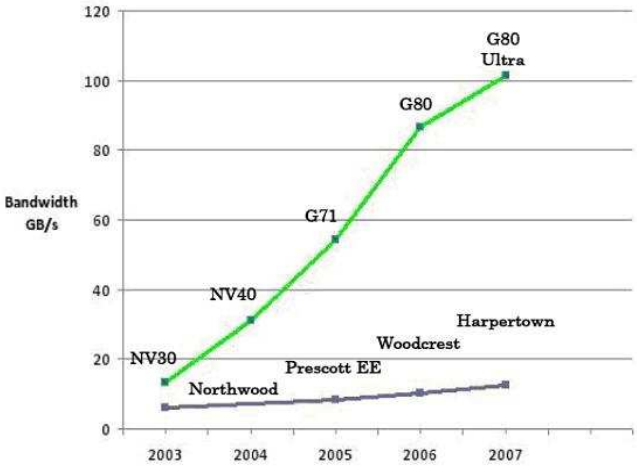

Figura 3-3: Gráfico da evolução da banda de memória (NVIDIA, 2008b)

3.3.1.3 Arquitetura de GPUs CUDA

As Geforce séries 8 e 9, e as da série GTX 200 usam vários *Stream Processors* (SPs) para executar operações inteiras e de ponto flutuante. Os SPs são muito eficientes, pois executam operações sobre um *stream* de entrada, enquanto produzem um *stream* de saída que pode ser usado por outros SPs. Os SPs são agrupados em *Stream Multiprocessors* (SMs) que, por sua vez, são agrupados em *Texture/Processor Clusters* (TPCs) para prover um grande poder de

processamento paralelo. A Figura 3-4 mostra um TPC composto por 2 SMs com 8 SPs cada (NVIDIA, 2006).

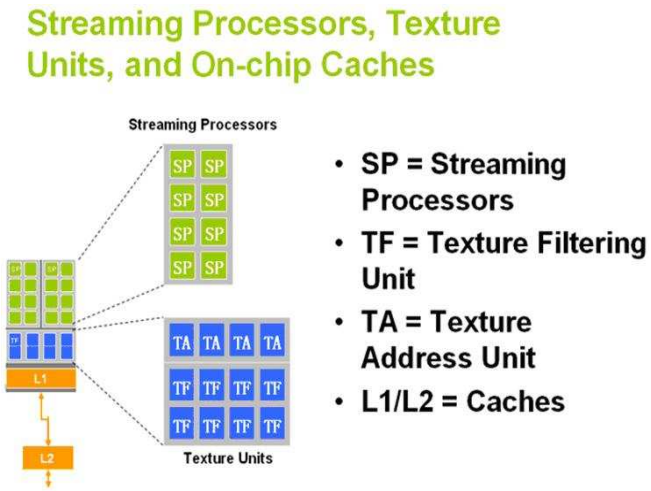

Figura 3-4: Arquitetura de um Stream Multi-Processor (NVIDIA, 2006)

Os TPCs possuem hardware capaz de decodificar e enviar instruções para seus SPs em grande velocidade. Operações similares são executadas em diferentes elementos de um *data stream*. A memória, tipicamente usada para armazenar a saída de um SP, pode ser lida como entrada por outros SPs. Tudo isso é controlado por instruções *Single Instruction/Multiple Thread* (SIMT) que controlam agrupamentos de SPs de maneira eficiente (NVIDIA, 2006).

Milhares de *threads* podem ser despachadas para execução dentro de uma GPU CUDA. O hardware CUDA mantém os SPs ocupados usando seu escalonador e despachando vários tipos de *threads* para execução em paralelo. Todos os SPs são conduzidos por um *clock* de alta velocidade que é separado do *clock* do *core* que comanda o resto do *chip* (NVIDIA, 2006).

A placa gráfica GTX 280, que foi usada neste trabalho, possui GPU com 30 SMs, cada um com 8 SPs, totalizando 240 SPs, e opera com um clock de 1.4 GHz. A memória principal é uma GDDR3 de 1GB, clock de 2322 MHz, barramento de 512 bits e a taxa de transmissão de 142 GB/s. Esta GPU tem três áreas de armazenamento, além da memória principal (*global memory*), quais sejam: a *constant memory* (memória de constantes) de 65536 bytes; a *shared memory* (memória compartilhada), que é compartilhada pelas *threads* de um bloco de *threads*, onde cada bloco pode alocar até 16384 *bytes*; e os *registers* que compreendem 16384 registradores de 32 bits (NVIDIA, 2008a; NVIDIA, 2008b). Na Figura 3-6 apresentamos a arquitetura da GPU da placa gráfica empregada.

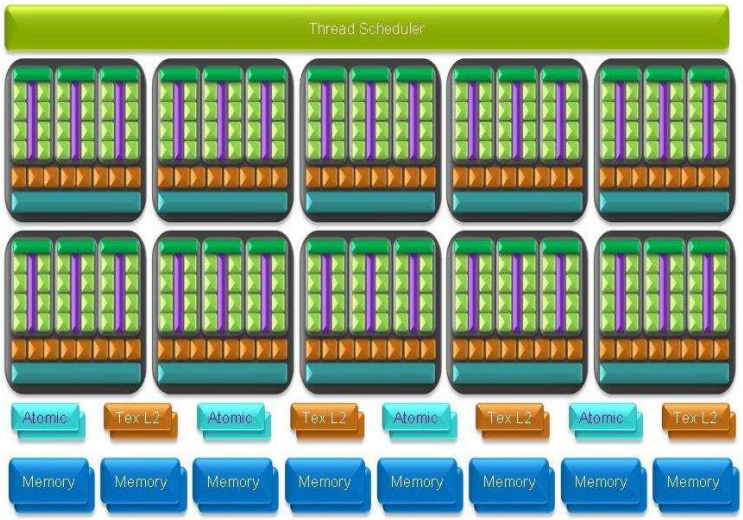

Figura 3-5: Arquitetura da GeForce GTX 280 (NVIDIA, 2008a)

Esta placa permite criar 512 *threads* por bloco, que podem ser distribuídas em três dimensões x, y e z (a dimensão z pode somente endereçar até 64) – quando são declaradas 16 *threads* em x e 16 em y, na verdade serão criados 256 *threads*. Pode ser criado um grande número de blocos também em 3 dimensões (por exemplo, 65535 em x e 65535 em y). Ela pode processar dados em precisão simples e dupla, mas o desempenho máximo em precisão dupla é de 80 Gflop/s, enquanto que o em precisão simples é igual à 933 Gflop/s (NVIDIA, 2008b).

3.3.1.4 Programação em C+CUDA

A programação em C+CUDA é viabilizada por uma pequena extensão da linguagem C e por uma nova biblioteca C. A Figura 3-6 apresenta os níveis de abstração de um programa C+CUDA.

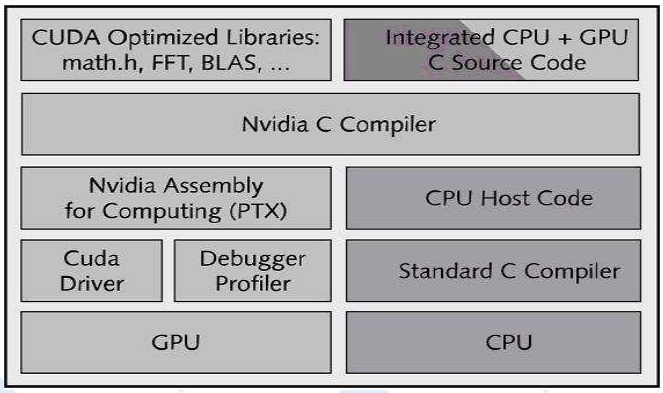

Figura 3-6: Arquitetura da linguagem C-CUDA (HALFHILL, 2008)

Como mostra a Figura 3-6, no nível mais alto, um programa fonte em C+CUDA é especificado em C mais comandos específicos para a GPU, e pode incluir chamadas a funções das bibliotecas otimizadas CUDA da Nvidia. Um fonte em C+CUDA deve ser compilado pelo *Nvidia C Compiler*, que gera código *assembly* para o *assembler* da Nvidia e código em C para ser compilado por um compilador C padrão (gcc). O código *assembly* traduzido para

código de máquina de GPU é levado à GPU pelo driver da placa de vídeo CUDA *enabled* sob demanda do código de máquina de CPU produzido pelo compilador C.

A GPU (*device*) é vista pela CPU (*host*) como um co-processador capaz de executar um número muito grande de *threads* em paralelo (NVIDIA, 2007). Tanto o *host* como o *device* mantêm memória (*Dynamic Random Access Memory* - DRAM) própria, chamadas de *host memory* e *device memory*. Os dados podem ser copiados de forma otimizada de uma DRAM para a outra através de chamadas à biblioteca CUDA (NVIDIA, 2007).

3.3.1.5 Principais Diferenças entre C+CUDA e C Padrão

Em um programa C+CUDA, funções usuais da linguagem C, que são chamadas e executadas pela CPU, são chamadas de funções *host*. Além das funções *host*, podem existir dois novos tipos de função em programas C+CUDA: funções *kernel* e funções *device*. Funções *kernel* são executadas pelo *device* e invocadas pelo *host*. Para especificar que uma função é deste tipo é necessário utilizar o qualificador "__global__", que deve ser inserido antes do tipo de retorno da função, que deve ser sempre *void*. Funções *device* são chamadas e executadas somente pelo *device*. Seu qualificador é "__device__", que também deve ser colocado antes do tipo de retorno da função. Neste tipo de função é permitido o retorno de qualquer tipo (NVIDIA, 2008a).

Um programa em C+CUDA especifica uma hierarquia de *threads*. Esta hierarquia é formada por um *grid*, que pode ter até duas dimensões de blocos de *threads*, sendo que os blocos podem ter até três dimensões de *threads*, conforme mostrado na Figura 3-7. A função "__syncthreads()" funciona como uma barreira e permite fazer o sincronismo de *threads* de um mesmo bloco (NVIDIA, 2008a).

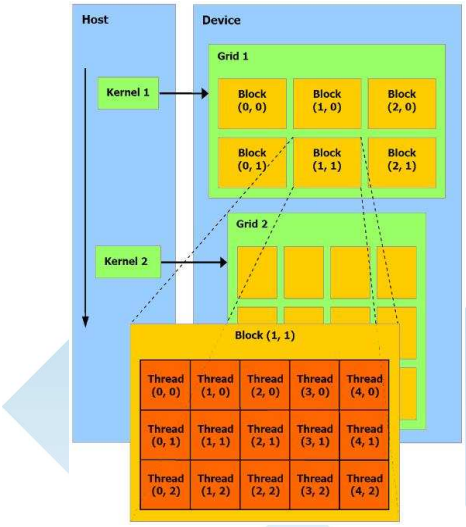

Figura 3-7: Hierarquia de *threads* em C-CUDA (NVIDIA, 2007)

C+CUDA oferece ao programador autonomia para usar três tipos de memória presentes na GPU por meio de qualificadores do tipo das variáveis empregadas. O qualificador __device__ especifica que uma variável deve ser alocada na *global memory*, o qualificador __constant__ especifica que uma variável deve ser alocada na *constant memory*, e o qualificador __shared__ especifica que uma variável deve ser alocada na *shared memory*. Uma variável

automática definida em uma função *device* geralmente reside em um registrador (estas variáveis podem ser movidas para outras regiões de memória se não houverem registradores suficientes).

Os registradores são de acesso exclusivo de cada *thread*. A *shared memory* é compartilhada por todas as *threads* de um bloco e, apesar de pequena, é muito importante, pois possui baixa latência de acesso. A *constant memory* também possui baixa latência e pode ser acessada por todas as *threads* de um grid, mas apenas para leitura – apenas o *host* pode escrever nesta memória. A *global memory* é o local onde o *host* publica os dados que serão processados. Ela é compartilhada por todas as *threads* de um *grid*, mas possui alta latência (400 a 600 ciclos de *clock* para uma leitura). Existe outra memória específica para aplicações gráficas, a *texture memory* (memória de textura), que é uma memória somente de leitura (NVIDIA, 2008a).

3.3.2 Implementação Paralela do VS_CORE em C+CUDA

No treinamento, o VS_CORE original recebe e armazena uma matriz, *D*, de *documentos x termos*, e uma matriz, *C*, de *documentos x códigos CNAE-Subclasse* (SCAEa, 2007). Um documento d_i descrevendo uma atividade econômica é, na verdade, uma linha da matriz *D* (seu índice *i*) e seus códigos CNAE-Subclasse associados são aqueles da mesma linha *i* da matriz *C*. Os termos reconhecidos pelo VS_CORE (conjunto de palavras relevantes existente na base de treinamento convertidas para suas formas canônicas) que aparecem em cada documento d_i são as colunas da linha *i* da matriz *D*. Estes termos podem equivaler à freqüência de uma determinada palavra no documento d_i (*Term Frequency – TF)*, ou a um peso computado de forma apropriada (por exemplo, *TFIDF*; ver SCAEa, 2007).

Na fase de teste do VS_CORE original, é necessário computar a distância de um documento de teste d_j para cada documento d_i em D. Esta distância é estimada por meio do cosseno do ângulo entre o vetor que representa d_j e cada documento d_i, que pode ser computado como (SCAEa, 2007):

$$\cos(d_i, d_j) = \frac{d_i . d_j}{|d_i||d_j|}$$

A principal operação demandada pelo cálculo do cos(d_i, d_j) para cada documento d_i de *D* é o produto *matriz x vetor* $D \cdot d_j$. Assim, esta operação foi paralelizada por meio de um algoritmo implementado C+CUDA. Para compreender como esta paralelização foi feita, suponha que a matriz *documentos x termos* possui 2780 linhas e 1454 colunas.

Para a matriz 2780x1454, foi criado um *grid* unidimensional com 128 blocos unidimensionais, cada um com 256 *threads*. Este *grid* pode ser visto como uma matriz de *threads* de dimensões 128x256. A cada passo do algoritmo, as 256 *threads* de cada um dos blocos recebem um elemento de uma linha de *D* e o elemento correspondente de d_j. A Figura 3-8 mostra um exemplo de como as *threads* são mapeadas aos elementos de *D* para um *grid* de dimensões 4x4 e uma matriz de *D* de dimensões 7x8. Na figura, o mapeamento das *threads* referente ao primeiro passo do algoritmo aparece em destaque.

(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)
(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)

Figura 3-8: Exemplo de como as *threads* fazem o produto matriz x vetor

A cada passo do algoritmo, o *grid* é deslocado para a direita de um número de colunas igual ao número de *threads* por bloco. Em cada um deste passos, cada *thread* computa o produto de um elemento de um d_i por um elemento correspondente de d_j, e acumula este produto em uma variável denominada *sum*. Quando o *grid* é deslocado de modo a cobrir todos os elementos de uma linha, as variáveis *sum* são publicadas em um vetor de cada bloco de *threads* denominado *accumR* previamente alocado em memória compartilhada. O $\cos(d_i, d_j)$ de um documento d_j percorrido pelo *grid* é igual a soma dos elementos deste vetor. Os elementos do vetor *accumR* são somados por meio do algoritmo *Sum tree like reduction*, o qual é descrito na Seção 3.3.2.1. Após computar todos os $\cos(d_i, d_j)$ dos documentos d_j vizitados pelo *grid* no seu deslocamento para a direita, o mesmo é reposicionado à esquerda e deslocado para baixo de um número da linhas de *D* igual ao número de blocos de *threads*.

A Figura 3-9 mostra o código em C+CUDA de nosso algoritmo que deve ser executado por cada *thread*, e que realiza o produto *matriz x vetor* $D \cdot d_j$. No código, a variável *m* é a matriz de treinamento do algoritmo de classificação. Ela foi representada na forma de vetor. A variável *first* guarda o índice para o início da linha da matriz *m*, a variável *last* guarda o índice do final das linhas de *m*. O uso destas variáveis no código viabiliza o uso de um *grid* com dimensões não múltiplas das da matriz *D*.

As variáveis *nline* e *ncol* contêm o número de linhas e colunas de *m*, respectivamente; o vetor *doc* é o documento d_j que desejamos classificar; *matrix_norm* guarda a norma $|d_i|$ de cada linha de *m*; a variável *norm_doc* guarda a norma $|d_j|$ do vetor *doc*; e o vetor *out* armazena $\cos(d_i, d_j)$ para cada linha de *m*.

As iterações do *loop* (*for*) mais externo deslocam o *grid* para baixo, enquanto que as do *loop* (*for*) mais interno deslocam o *grid* para a esquerda ao mesmo tempo em que computam os produtos de cada elemento de d_i por cada elemento correspondente de d_j e acumulam estes produtos na variável *sum* de cada *thread*.

```
__global__ void
matrix_vector_prod (float *m, int nline, int ncol, float *out, float *doc, float *matrix_norm)
{
        float sum, norm;
        int first, last, bx, k, i;
        __shared__ float accumR[NUMTHREAD];

        for(bx = blockIdx.x; bx < nline; bx += gridDim.x)
        {
                sum = 0.0;
                first = bx * ncol;
                last = first + ncol;
                k = threadIdx.x;

                for (i = threadIdx.x + first; i < last; i += blockDim.x, k += blockDim.x)
                        sum += doc[k]*m[i];

                accumR[threadIdx.x] = sum;
                __syncthreads();
                sum_tree_like_reduction(accumR, NUMTHREAD, &sum);

                if(threadIdx.x == 0)
                {
                        norm = (matrix_norm[bx] * norm_doc);

                        if(norm > 0.0)
                                out[bx] = sum / norm;
                        else
                                out[bx] = 0;
                }
        }
}
```

Figura 3-9: Código do produto matriz x vetor

Depois de cada *thread* percorrer as posições e fazer o produto, elas colocam o resultado no vetor *accumR*. Cada *thread* fica responsável por uma posição do vetor: a *thread* 0 pela posição 0, a 1 pela 1, e assim sucessivamente. Como mencionado anteriormente, para somar os elementos de *accumR* foi usado o algoritmo *sum tree like reduction*, descrito a seguir.

3.3.2.1 Algoritmo *Sum Tree Like Reduction*

O algoritmo *sum tree like reduction* tem como propósito somar todos os elementos de um vetor e pode ser facilmente paralelizado em C+CUDA.

Suponha que o vetor cujos elementos queremos somar tenha dimensão 8. Para somar seus elementos, os mesmos são divididos em dois grupos (ver Figura 3-10). São somados os elementos 0 e 4 e o resultado é colocado em 0; 1 e 5 e o resultado é colocado em 1; 2 e 6 e o resultado é colocado em 2; e por fim, são somados os elementos 3 e 7 e o resultado é colocado em 3. Em seguida, o vetor é reduzido para tamanho 4 e novamente é dividido em dois. Assim são somados os elementos 0 e 2 e o resultado é colocado em 0; e 1 e 3 e o resultado é colocado em 1. Finalmente, os elementos onde foram colocados os resultados das últimas somas são divididos novamente em dois grupos e são somados os elementos 0 e 1, e o resultado é colocado em 0. Assim, o resultado da soma de todos os elementos do vetor fica no elemento 0 ao final da execução do algoritmo. A Figura 3-10 ilustra o exemplo descrito.

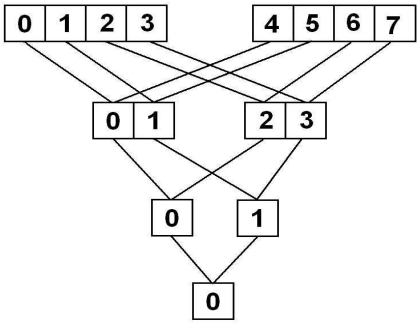

Figura 3-10: Exemplo da soma em árvore (*sum tree like reduction*)

A Figura 3-11 mostra o código paralelo em C+CUDA do algoritmo *sum tree like reduction*. Os parâmetros de entrada do algoritmo são: o vetor *v* que desejamos somar, o tamanho deste vetor e variável *sum* onde o resultado será retornado. É importante lembrar que o tamanho do vetor deve ser potência de dois. No final da computação da soma, a *thread* 0 será a única que terá o valor da soma. As iterações do *loop* (*for*) mais externo fazem a divisão do vetor que desejamos somar, e as do mais interno são responsáveis por fazer a soma em paralelo.

```
__device__ void sum_tree_like_reduction
(
        float *v        ,
        int size        ,
        float *sum
)
{
        for(int k  = size / 2;k > 0; k >>=1)
        {
                __syncthreads();
                for(int j =threadIdx.x; j < k; j += blockDim.x)
                {
                        v[j] += v[k + j];
                }

        }
        __syncthreads();
        if(threadIdx.x==0) *sum=v[0];
}
```

Figura 3-11: Código da soma em árvore (*sum tree like reduction*)

3.3.2.2 Resultados

Para avaliação da Vector Space CUDA (VSC_CORE), foi utilizada uma base de dados composta de 3280 descrições de atividades econômicas de empresas da cidade de Vitória-ES, e 1183 definições das subclasses da tabela CNAE. Os resultados obtidos são mostrados na Tabela 3-5 e na Tabela 3-6.

Tabela 3-5: Experimentos com o classificador VSC_CORE

Exp.	Dados de Treino			Dados de Teste				Rev. do SCAE	Desem-penho
	Tabela	Coluna	Limites	Tabela	Coluna	Limites	Nível		
7.1.1.1	110_SUBCL	DESCR_SUB	0 a 1182	110_SUBCL	DESCR_SUB	0 a 1182	SUBCL	758	99,07%
7.1.1.2	110_SUBCL	DESCR_SUB	0 a 1182	DADOS_VIX_110	OBJ_SOCIAL	0 a 3280	SUBCL	758	63,82%
7.1.1.3	110_SUBCL	DESCR_SUB	0 a 1182	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	SUBCL	758	67,28%
7.1.1.4	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	SUBCL	758	69,23%
7.1.1.5	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	CLASSE	758	73,74%
7.1.1.6	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	GRUPO	758	77,64%
7.1.1.7	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	DIVISÃO	758	83,42%
7.1.1.8	110_SUBCL DADOS_VIX_110	DESCR_SUB OBJ_SOCIAL	0 a 1182 0 a 1639	DADOS_VIX_110	OBJ_SOCIAL	1640 a 3280	SEÇÃO	758	86,96%

Tabela 3-6: Desempenho do classificador VSC_CORE

Experimento	7.1.1.1	7.1.1.2	7.1.1.3	7.1.1.4	7.1.1.5	7.1.1.6	7.1.1.7	7.1.1.8
One Error	0,009	0,362	0,327	0,308	0,263	0,224	0,166	0,130
Ranking Loss	0,000	0,108	0,091	0,016	0,014	0,016	0,022	0,034
Coverage	0,009	264,466	238,526	57,703	22,913	9,244	2,987	1,302
Average Precision	0,995	0,555	0,583	0,639	0,696	0,757	0,840	0,895
R-Precision	0,991	0,491	0,527	0,568	0,620	0,673	0,763	0,825
Hamming Loss	0,000	0,004	0,004	0,004	0,005	0,009	0,018	0,038
R-Hamming Loss	0,019	1,016	0,944	0,864	0,761	0,654	0,474	0,350
Microaveraged Precision	0,991	0,437	0,462	0,510	0,569	0,616	0,721	0,807
Microaveraged Recall	0,991	0,437	0,462	0,510	0,569	0,616	0,721	0,807
Macroaveraged Precision	0,991	0,491	0,527	0,568	0,620	0,673	0,763	0,825
Macroaveraged Recall	0,991	0,491	0,527	0,568	0,620	0,673	0,763	0,825
Microaveraged F1	0,991	0,437	0,462	0,510	0,569	0,616	0,721	0,807
Macroaveraged F1	0,991	0,491	0,527	0,568	0,620	0,673	0,763	0,825

A Tabela 3-5 e a Tabela 3-6 basicamente mostram que a implementação do VSC_CORE está correta, pois os resultados são idênticos aos do VS_CORE.

Para medir o desempenho em termos de tempo do VSC_CORE se comprado ao VS_CORE utilizamos a métrica *speed-up*. O *speed-up* mede o quanto o algoritmo paralelo é mais rápido que o seqüencial, é dado por:

$$speed - up = \frac{tempo_{Seq}}{tempo_{Pal}}$$

onde $tempo_{Seq}$ é o tempo do algoritmo executando em uma máquina seqüencial e $tempo_{Pal}$ é tempo do algoritmo executando em uma máquina paralela.

Tabela 3-7: *Speed-up* do algoritmo VSC_CORE

***Speed-up* do algoritmo VSC**	
VSC_CORE	2,62

3.4 Meta Física 1.4/2008: Desenvolvimento de Mecanismo de Composição dos Resultados da Codificação Através de Redes Neurais Artificiais, Redes Bayesianas e *Latent Semantinc Indexing* em uma Única Codificação, mais Robusta – Fundamentação do Código

Em seções anteriores foram expostas diferentes metodologias para a classificação de documentos que descrevem atividades econômicas:

- Redes Neurais Sem Peso (RNSPs);
- Modelo Vetorial;
- Redes Bayesianas.

A seguir, serão apresentados novos mecanismos para compor essas metodologias, utilizando um subconjunto qualquer delas para formar um novo classificador. Além dos COREs, ou núcleos, já implementados no SCAE, outros mecanismos de classificação poderão fazer parte do CORE ENSEMBLE – denominação do CORE que compõe os resultados de classificação de outros COREs – à medida que forem incorporados ao sistema.

A Figura 3-12 ilustra a inclusão de um novo núcleo (CORE) que implementa, por exemplo, uma *Support Vector Machine* (SVM).

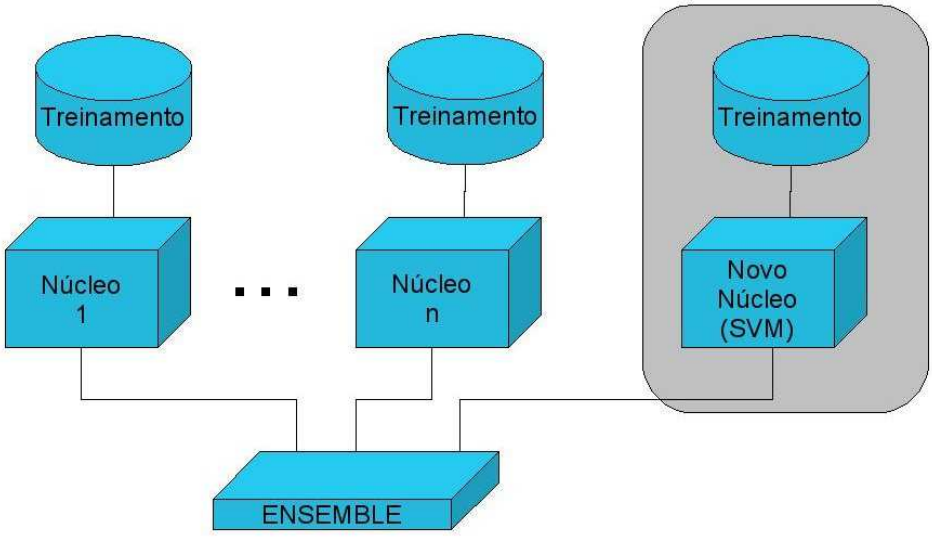

Figura 3-12: Inclusão de núcleos no SCAE

O treinamento dos núcleos subordinados, apesar de poder ser invocado a partir do ENSEMBLE, permanece independente dele. É possível até mesmo que as bases de treinamento dos núcleos subordinados sejam atualizadas uma a uma, sem que isto prejudique o funcionamento do ENSEMBLE. A Figura 3-13 mostra a interligação do ENSEMBLE com outros núcleos.

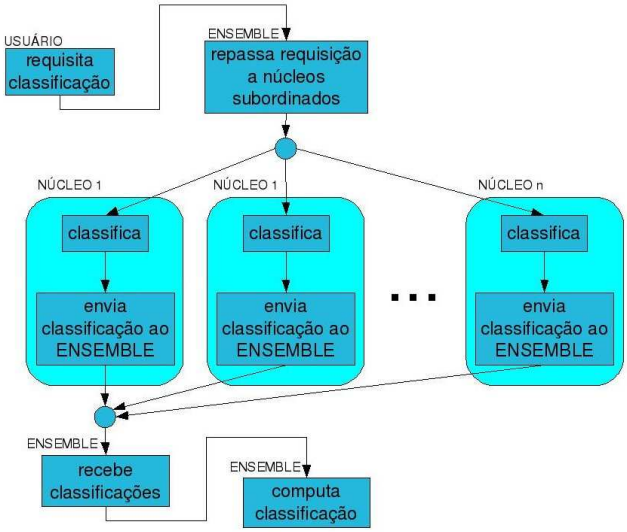

Figura 3-13: Interligação do ENSEMBLE com outros núcleos do SCAE

Para realizar uma classificação, o ENSEMBLE recebe um dado texto descrevendo atividades econômicas a ser classificado através da interface do SCAE e repassa-o para os núcleos subordinados. Cada um deles informa ao ENSEMBLE uma classificação que não é necessariamente composta por apenas uma classe. De posse destas informações, ou crenças dos classificadores a respeito da pertinência da classificação da atividade econômica descrita de forma textual (um objeto social, por exemplo) dentro de uma ou mais subclasses CNAE, o ENSEMBLE computa uma única classificação, que reflete as contribuições dos núcleos subordinados, e retorna-a para o usuário. Este processo está ilustrado na Figura 3-14.

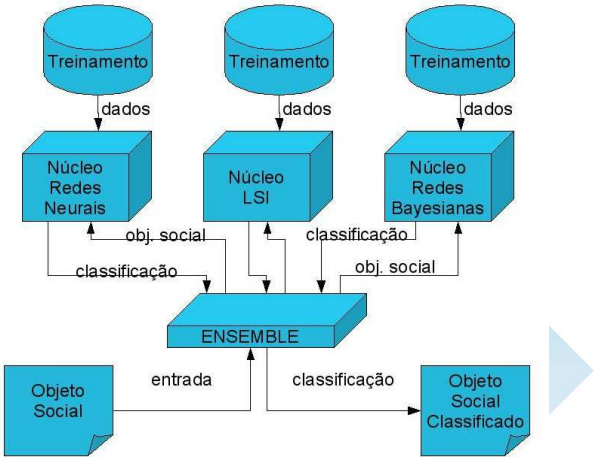

Figura 3-14: Classificação pelo ENSEMBLE

Há diversas alternativas de combinação das crenças dos núcleos subordinados em uma única classificação. Estas podem, grosseiramente, ser agrupadas naquelas que combinam de forma estática ou dinâmica.

3.4.1 Combinação Estática

Uma possibilidade para combinar crenças de classificadores estaticamente é o algoritmo conhecido como *Boosting*. *Boosting* é baseado na questão proposta por Kearns (KEARNS, 1988): "*Can a set of weak learners create a single strong learner?*" Para resolver esta questão, muitas variantes do algoritmo *Boosting* foram propostas. As primeiras, cujas autorias atribuem-se a Schapire (SCHAPIRE, 1990) e Freund (FREUND, 1990), não são adaptativas, no sentido de que não alteram os pesos de cada classificador. Por isso, esses algoritmos não exploram totalmente o potencial dos classificadores. Originalmente, *Boosting* foi implementado com classificadores fracos, que são aqueles que apresentam, individualmente, taxas de acerto pouco superiores a 50%.

O algoritmo *AdaBoost*, introduzido em 1996 (FREUND; SCHAPIRE, 1996), é uma modificação do *Boosting* original. Em contraste com *Boosting*, no *AdaBoost* os pesos de cada um dos *n* classificadores que compõem o classificador final são obtidos de forma adaptativa. Durante o treinamento, cada um dos classificadores fracos recebe os padrões de entrada. A função de distribuição de pesos (que inicialmente é uniforme) é alterada usando-se uma fração do menor erro obtido dentre os *n* classificadores. Assim, ao final do treinamento, uma função de distribuição de pesos que pondera cada classificador é encontrada, e a soma ponderada de suas respostas é a resultante do classificador final.

3.4.2 Combinação Dinâmica

Em sistemas que combinam classificadores dinamicamente, os pesos associados a cada classificador subordinado são determinados por uma análise do comportamento dos classificadores durante o processo de treinamento. No caso do SCAE, o cálculo dinâmico dos pesos dos classificadores poderá ser obtido por meio de um segundo nível de aprendizado de máquina. Este seria realizado a partir de uma realimentação introduzida no sistema, que poderá vir de duas formas:

- como questionamento à classificação de um determinado objeto social (realimentação negativa);
- pela repetição de solicitações para a classificação de textos equivalentes de objetos sociais (com mesmo mapeamento para o vetor de pesos que descreve um documento internamente ao SCAE) sem questionamento.

A realimentação negativa relacionada à eventual necessidade de uma reclassificação manual causaria a diminuição do peso dos classificadores votantes na categoria questionada. Já a realimentação positiva ocorreria em duas situações: (i) nos casos de reclassificação manual, através do aumento significativo de peso do(s) classificador(es) votante(s) na nova categoria e (ii) no caso de novas consultas a textos equivalentes, aumentando marginalmente o peso do(s) classificador(es) votante(s) com uma realimentação positiva. Observe que, neste último caso, uma interpretação de aceitação da classificação é assumida.

3.4.3 Combinações Empregadas

Para fins desta meta, projetamos uma versão do ENSEMBLE que, a critério do usuário, adota combinação estática ou dinâmica. A seguir, apresentamos as características destas combinações.

3.4.4 Combinação Estática no ENSEMBLE

A combinação estática no ENSEMBLE é obtida atribuindo a mesma importância às crenças dos núcleos subordinados, ou seja, os pesos de cada um deles são iguais. Para cada subclasse (ou classe, grupo etc) é calculado o somatório de suas posições nos rankings informados por cada um dos núcleos. De posse destes somatórios, passa-se para a etapa de composição do ranking final (do ENSEMBLE), no qual os códigos que possuírem menor valor de somatório ocorrerão nas primeiras posições do ranking. A Figura 3-15 ilustra um caso hipotético de combinação estática sobre os rankings de dois núcleos (ou COREs) subordinados, onde cada ranking contém cinco códigos CNAE (A, B, C, D, E).

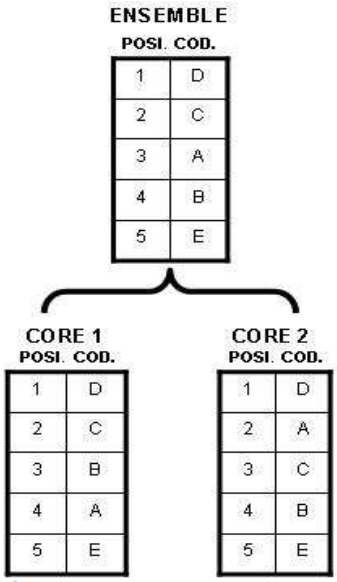

Figura 3-15: Combinação estática no ENSEMBLE

Note que, pela Figura 3-15, o código D ocorre na primeira posição do ranking composto (do ENSEMBLE) uma vez que possui um somatório (S_A) de posições menor que os demais códigos ($S_A = 6$, $S_B = 7$, $S_C = 5$, $S_D = 2$, $S_E = 10$). É importante mencionar que, em caso de empate entre dois ou mais códigos, uma permutação randômica é efetuada entre os códigos.

3.4.5 Combinação Dinâmica no ENSEMBLE

No ENSEMBLE, a combinação dinâmica é alcançada por meio do uso de uma matriz de pesos que associa cada classificador subordinado a cada código existente na base de treino. Esta matriz, inicialmente, apresenta pesos iguais (na atual implementação, atribui-se 1,0),

indicando que os núcleos subordinados possuem, a princípio, 100% de certeza sobre a posição dos códigos no ranking retornado ao ENSEMBLE.

Ao longo das classificações (procedimento de teste) esta matriz é atualizada por um processo de realimentação que se dá pelas três formas possíveis:

- para todo código pertinente não predito por um core subordinado há um incremento nos pesos destes códigos associados a este núcleo (atualmente, este incremento é de 0,001 com saturação em 2,0);
- para todo código predito incorretamente por um núcleo subordinado há um decremento nos pesos destes códigos associados a este núcleo (atualmente, este decremento é de 0,001 com saturação em 0,0);
- para todo código predito corretamente por um núcleo subordinado os pesos destes códigos associados a este núcleo permanecem inalterados.

A computação do ranking final (do ENSENBLE) adotando a combinação dinâmica é muito semelhante à combinação estática, a diferença reside no fato que as posições dos códigos nos rankings dos núcleos subordinados são modificadas por um multiplicador (peso existente entre o código e o núcleo subordinado). Esta multiplicação pode representar tanto um reforço (peso maior que 1,0) quanto um enfraquecimento (peso menor que 1,0). Nota-se, então, que a realimentação produzida ao final da classificação do documento corrente impactará nas classificações subseqüentes.

3.4.6 Resultados

O ENSEMBLE_CORE, conforme descrito acima, foi implementado no SCAE e um comparativo entre a combinação estática e a dinâmica pode ser apreciado pela Tabela abaixo que reporta os resultados para o experimento que consistiu em treinar o categorizador com a CNAE e primeira metade dos dados de Vitória, e testá-lo com a segunda metade.

Tabela 3-8: Comparativo entre a combinação estática e a dinâmica

Combinação	One Error	Ranking Loss	Coverage	Average Precision^d	Hamming Loss	Exact Match
Estática	0,2096	0,0159	74,2693	0,7089	0,0030	0,3425
Dinâmica	0,4613	0,0200	78,9634	0,5057	0,0040	0,1414

A observação da Tabela nos leva a concluir que a atual implementação de combiangção dinâmica necessita de aperfeiçoamentos (com a exceção da métrica **average precision**, quanto menor o valor, melhor o desempenho do categorizador). Estes resultados podem ser explicados pela grande influência que as realimentações anteriores possuem sobre a constituição do ranking corrente. Uma solução alternativa, a fim de melhorar o desempenho do ENSEMBLE com combinação dinâmica, seria ajustar a função de realimentação como, por exemplo, diminuir a quantidade incrementada (decrementada) dos pesos.

4 Outras Realizações Técnico-Científicas

O envolvimento dos pesquisadores principais em outras atividades técnicas e científicas de interesse do Projeto SCAE é aqui relatado.

4.1 Organização e Participação em Eventos Científicos

Algumas oportunidades de participação em eventos científicos, nacionais e internacionais, foram aproveitadas no decorrer do Projeto SCAE. É aqui relatada a organização da Sessão Especial em *Weightless Neural Systems*, integrante do *17th European Symposium on Artificial Neural Networks* — ESANN 2009, e do *6th Latin American Web Congress* (LA-Web'08). Também são relacionadas as participações dos pesquisadores em Comitês de Programa, neste e em outros eventos.

4.1.1 COLA 2009

A Professora Priscila M. V. Lima é membro do Comitê de Programa do COLA — *Computational Logic with Applications* (http://epia2009.web.ua.pt/cola/), divisão temática do EPIA 2009 — *14th Portuguese Conference on Artificial Intelligence* (http://epia2009.web.ua.pt/). A professora Priscila M. V. Lima também atuou como revisora dos trabalhos submetidos.

4.1.2 ESANN 2009: Sessão Especial em *Weightless Neural Systems*

Os professores Felipe M. G. França e Priscila M. V. Lima foram convidados pelo pesquisador italiano Massimo De Gregorio, do *Istituto di Cibernetica* — CNR, Nápolis, para organizar uma sessão especial em sistemas neurais sem-peso (*Weightless Neural Systems*), tipo de classificador neural de grande interesse para o Projeto SCAE, para o *17th European Symposium on Artificial Neural Networks* — ESANN 2009 (http://www.dice.ucl.ac.be/esann/index.php?pg=specsess), ocorrido em Bruges, Bélgica, de 22 a 24 de Abril de 2009.

4.1.2.1 LA-Web'08, TIL'08 e WebMedia 2008

A pesquisadora Claudine Badue foi Coordenadora do Comitê Local de Organização do *6th Latin American Web Congress* — LA-Web'08 (http://www.cwr.cl/la-web2008/), e também foi membro do Comitê de Organização do *XIV Brazilian Symposium on Multimedia and the Web* — WebMedia 2008 (http://www.inf.ufes.br/webmedia2008/webmedia2008_home.html), ocorridos, em co-locação, em Vila Velha, ES, de 28 a 30 de Outubro de 2008. Os professores Alberto Ferreira De Souza e Elias de Oliveira também foram membros dos referidos comitês. Os professores Alberto Ferreira De Souza e Elias de Oliveira, juntamente com a pesquisadora Claudine Badue, também foram membros do Comitê Local de Organização do *Workshop Information and Human Language Technology* — TIL'08 (http://www.inf.ufes.br/webmedia2008/webmedia2008_wtil2008_ing.html), evento também co-locado ao *XIV Brazilian Symposium on Multimedia and the Web* — WebMedia 2008.

4.1.2.2 SBAC-PAD 2008 e WSCAD 2008

Os professores Alberto Ferreira De Souza, Felipe M. G. França e Priscila M. V. Lima foram membros do *Program Committee* do *20th International Symposium on Computer Architecture and High Performance Computing* — SBAC-PAD 2008 (http://www.cpcx.ufms.br/union/index.php?cp=133). Os professores Alberto Ferreira De Souza e Felipe M. G. França, foram membros do Comitê de Programa do 9º *Workshop em Sistemas Computacionais de Alto Desempenho* — WSCAD-SSC 2008 (http://www.cpcx.ufms.br/union/index.php?cp=136). Ambos os eventos, ocorreram em co-locação em Campo Grande, MS, de 29 de Outubro a 1 de Novembro de 2008.

4.1.2.3 SBAC-PAD 2009 e WSCAD 2009

O professor Alberto Ferreira De Souza é membro do *Program Committee* do *21st International Symposium on Computer Architecture and High Performance Computing* — SBAC-PAD 2009 (http://regulus.pcs.usp.br/sbac2009/). Os professores Alberto Ferreira De Souza, Felipe M. G. França e Priscila M. V. Lima, são membros do Comitê de Programa do 10º *Workshop em Sistemas Computacionais de Alto Desempenho* — WSCAD-SSC 2009 (http://www.dcc.ufrj.br/~wscad2009/index.php). Ambos os eventos, ocorrerão em co-locação em São Paulo, SP, de 28 a 31 de Outubro de 2009.

4.1.2.4 SEMISH 2009

O Professor Alberto Ferreira De Souza é membro do Comitê de Programa do XXXVI *Seminário Integrado de Hardware e Software* — SEMISH 2009 (http://csbc2009.inf.ufrgs.br/index.php?option=com_content&task=view&id=28&Itemid=73), evento previsto para 20 a 24 de Julho de 2009, em Bento Gonçalves, RS.

4.1.2.5 CISIM 2009

O Professor Alberto Ferreira De Souza é membro do *Program Committee* do *8th International Conference on Computer Information Systems and Industrial Management Applications* (http://www.mirlabs.org/cisim09/index.html), previsto para 9 a 11 de Dezembro de 2009, em Coimbatore, Índia.

4.1.2.6 CSTST 2008

O Professor Alberto Ferreira De Souza é membro do *Program Committee* do *5th International Conference on Soft Computing as Transdisciplinary Science and Technology* (http://sigappfr.acm.org/cstst08/), ocorrido entre 9 a 11 de Outubro de 2008, em Paris, França.

4.1.2.7 SoCPaR 2009

O Professor Alberto Ferreira De Souza é membro do *Program Committee* do *International Conference on Soft Computing and Pattern Recognition* (http://www.mirlabs.org/socpar/), previsto para 4 a 7 de Dezembro de 2009, em Malacca, Malásia.

Fundação Espírito-Santense de Tecnologia
Av. Fernando Ferrari, 845 | Goiabeiras | 29.060-410 - Vitória - ES
Tel: +55 27 3345-7555 | Fax: +55 27 3345-7668 | www.fest.org.br

4.2 Publicações

1. DE SOUZA, Alberto F. ; PEDRONI, Felipe ; OLIVEIRA, Elias ; CIARELLI, Patrick M. ; HENRIQUE, Wallace Favoreto ; VERONESE, Lucas ; BADUE, Claudine . Automated multi-label text categorization with VG-RAM weightless neural networks. Neurocomputing (Amsterdam), Vol. 72, No. 10-12 (June 2009), p. 2209-2217.
2. FREITAS, Fabio D. ; DE SOUZA, Alberto F. ; DE ALMEIDA, Ailson R. . Prediction-based portfolio optimization model using neural networks. Neurocomputing (Amsterdam), Vol. 72, No. 10-12 (June 2009), p. 2155-2170.
3. DE SOUZA, A. F. ; BUYYA, Rajkumar . Introduction to the Special Issue on the 18th International Symposium on Computer Architecture and High Performance Computing. International Journal of Parallel Programming, v. 36, p. 163-165, 2008.
4. ROUNCE, Peter ; DE SOUZA, A. F. . Dynamic Instruction Scheduling in a Trace-based Multi-threaded Architecture. International Journal of Parallel Programming, v. 36, p. 184-205, 2008.
5. ALMEIDA, Fernado Líbio Leite ; DE SOUZA, A. F. ; FERNANDES, Edil Severiano Tavares . DTSD: Uma Arquitetura com Mecanismo Híbrido de Execução. In: IX Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD-SSC), 2008, Campo Grande - MS. Anais do IX Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD-SSC). Porto Alegre - RS : Sociedade Brasileira de Computação, 2008. p. 45-52.
6. DE SOUZA, Alberto F. ; SOUZA, Sotério Ferreira de ; AMORIM, Claudio Luiz de ; LIMA, P. M. V. ; ROUNCE, Peter . Hardware Supported Synchronization Primitives for Clusters. In: International Conference on Parallel and Distributed Processing Techniques and Applications, 2008, Las Vegas. Proceedings of the 2008 International Conference on Parallel and Distributed Processing Techniques and Applications, 2008. p. 520-526.
7. OLIVEIRA, E. ; CIARELLI, Patrick M. ; BADUE, Claudine ; DE SOUZA, A. F. . A Comparison between a KNN Based Approach and a PNN Algorithm for a Multi-label Classification Problem. In: Eighth International Conference on Intelligent Systems Design and Applications - ISDA 2008, 2008, Kaoshiung - Taiwan (Formosa). Proceedings of the Eighth International Conference on Intelligent Systems Design and Applications - ISDA 2008. Los Alamitos - USA : IEEE Computer Society, 2008. p. 628-633.
8. CIARELLI, P. M. ; OLIVEIRA, E. . A Comparison Between a kNN based Aproach and a PNN Algorithm for a Multi-Label Classification Problem. In: International Conference on Intelligent Systems Design and Applications (ISDA), 2008, Taiwan. Proceedings of VIII ISDA, 2008.
9. OLIVEIRA, Márcia G. ; ZANDONADE, E. ; OLIVEIRA, E. . Uma metodologia para avaliação formativa em um ambiente de ensino e aprendizagem de classificação em Biblioteconomia. In: Encontro Nacional de Pesquisa em Ciência da Informação (ENANCIB), 2008, São Paulo. Anais do IX ENANCIB, 2008.
10. CIARELLI, P. M. ; KROHLING, R. A. ; OLIVEIRA, E. . Particle Swarm Optimization Applied to Parameters Learning of Probabilistic Neural Networks for

Classification of Economic Activities. Particle Swarm Optimization. Viena, Austria: I-Tech Education and Publishing, 2008.

11. OLIVEIRA, E. ; CIARELLI, P. M. ; PEDRONI, F. ; HENRIQUE, W. F. ; VERONESE, L. . Avaliação do Desempenho do Algoritmo ML-kNN em Classificação de Textos Livres de Atividades Econômicas. In: Workshop em Tecnologia da Informação e da Linguagem Humana (TIL), 2008, Vila Velha. Anais do 6o Workshop em Tecnologia da Informação e da Linguagem Humana, 2008.
12. LIMA, P. M. V. ; MORVELI-ESPINOZA, M. M. M. ; PEREIRA, Gláucia C ; FERREIRA, T. O. ; FRANÇA, Felipe Maia Galvão . Logical Reasoning via Satisfiability Mapped into Energy Functions. International Journal of Pattern Recognition and Artificial Intelligence, v. 22, p. 1031-1043, 2008.
13. GRIECO, B. ; LIMA, P. M. V. ; GREGORIO, M. ; FRANÇA, FELIPE M. G. . Extracting fuzzy rules from "mental" images generated by modified WiSARD perceptrons. In: 11th European Symposium on Artificial Neural Networks, 2009, Bruges - Bélgica. Proceedings of the 11th European Symposium on Artificial Neural Networks.
14. ALEKSANDER, I. ; GREGORIO, M. ; FRANÇA, F. M. G. ; LIMA, P. M. V. ; MORTON, H. . A brief introduction to Weightless Neural Systems. In: 11th European Symposium on Artificial Neural Networks, 2009, Bruges - Bélgica. Proceedings of the 11th European Symposium on Artificial Neural Networks.
15. PRADO, C. B. ; FRANÇA, F. M. G. ; DIACOVO, R. ; LIMA, PRISCILA M. V. . The Influence of Order on a Large Bag of Words. In: Eight International Conference on Intelligent Systems Design and Applications, 2008, Kaohsiung. Proceedings of ISDA'08. Los Alamitos, CA, USA : IEEE Computer Society Press, 2008. v. 1. p. 432-436.

4.3 Orientações

São relatadas aqui, de forma não exaustiva, as atividades de formação, em andamento e concluídas, sob a forma de orientação exercida pelos pesquisadores do Projeto SCAE.

4.3.1 Orientações em Andamento

Orientador: Alberto Ferreira De Souza
Orientados: Bruno Zanetti Melotti (mestrado), Jairo Lucas de Moraes (mestrado), Nuno Rasseli (mestrado);

Orientador: Claudine Badue
Orientados: Caribe Zampirolli (mestrado), Vicente Bissoli (mestrado);

Orientador: Elias Oliveira
Orientados: Marcia Goncalves Oliveira (doutorado), Patrick M. Ciarelli (doutorado);

Fundação Espírito-Santense de Tecnologia
Av. Fernando Ferrari, 845 | Goiabeiras | 29.060-410 - Vitória - ES
Tel: +55 27 3345-7555 | Fax: +55 27 3345-7668 | www.fest.org.br

4.3.2 Orientações Concluídas

Orientador: Elias Oliveira
Orientado: Márcia Gonçalves Oliveira (mestrado);

Orientador: Claudine Santos Badue Gonçalves
Orientado: Vicente Bissoli (graduação);

Orientador: Claudine Santos Badue Gonçalves
Orientado: Rickson Guidolini (graduação);

5 Participação da Equipe Científica em Encontros Relevantes

Ao longo do período foram realizados encontros e reuniões que contribuíram para o bom andamento do projeto.

5.1 V ENAT

Durante o 5º Encontro Nacional de Administradores Tributários, ocorrido em Brasília, DF, de 12 a 14 de Novembro de 2008, o Professor Alberto Ferreira De Souza representou o Projeto SCAE nas seguintes ocasiões:

5.1.1 Reunião da Subcomissão CNAE

O Professor Alberto Ferreira De Souza realizou apresentação do estágio atual das pesquisas e do plano de construção da base de codificações corretas e aprovação da Coleta CNAE para a fase piloto na Subcomissão CNAE.

5.1.2 Reunião para apresentação do SCAE para o Secretário Adjunto da RFB

Na ocasião, foi possível realizar uma apresentação do Projeto SCAE para o Secretário Adjunto da Receita Federal.

5.1.3 Apresentação do Projeto SCAE em Sessão do V ENAT

O Projeto SCAE foi apresentado no referido encontro, onde foram debatidas questões científicas relacionadas ao aprimoramento dos modelos de categorização de texto no contexto do projeto.

6 Lição Aprendida no Período

"Você nunca sabe que resultados virão da sua ação. Mas se você não fizer nada, não existirão resultados." — Mahatma Gandhi

Nas práticas e teorias sobre o bom funcionamento das organizações sociais, mesmo diante do estabelecimento de um planejamento adequado e antecipado, é sempre saudável manter uma avaliação contínua sobre os vários aspectos estratégicos para um sistema que preza o aprimoramento. Neste sentido, a intenção desta seção é identificar experiências e conhecimentos importantes adquiridos durante o período pertinente a este relato.

6.1 Quanto à Importância da Manutenção das Normas e Regulamentos

O trabalho cooperativo é baseado em premissas que podem vir a ser violadas em função de mudanças de normas ou regulamentos superiores que o regem ocasionadas fatores ou atores externos ao objeto de cooperação. Nestes casos, é de fundamental importância que a cooperação seja reforçada, ainda que as mudanças nas normas ou regulamentos imponham prejuízos àqueles que cooperam.

7 Bibliografia

CAMPOS, L. M. de, FERNÁNDEZ-LUNA, J. M., HUETE, J. F. The BNR Model: Foundations and Performance of a Bayesian Network-Based Retrieval Model. International Journal of Approximative Reasoning, (34), p. 265-285, 2003.

DUDA, R. O.; HART, P. E.; STORK, D. G. Pattern Classification. Wiley-Interscience, New York, Second Edition, 2001.

FREUND, Y. Boosting a Weak Learning Algorithm by Majority. In: Proceedings of the Third Annual Workshop on Computational Learning Theory, p. 202-216, 1990.

FREUND, Y.; SCHAPIRE, R. E. A Decision-Theoretic Generalization of on-line Learning and an Application to Boosting. In: Proceedings of the Ninth Annual Conference on Computation Learning Theory, p. 325-332, 1996.

FUNG, J.; MANN, S. Using Multiple Graphics Cards as a General Purpose Parallel Computer Applications to Computer Vision. In: Proceedings of the 17th International Conference on Pattern Recognition (ICPR'04), p. 805-808, 2004.

GARCIA, V.; DEBREUVE, E.; BARLAUD, M.. Fast k Nearest Neighbor Search Using GPU. In: Proceedings of the Computer Vision and Pattern Recognition Workshops 2008, p. 1-6, 2008.

HALFHILL, T. R. Parallel Processing with CUDA: Nvidia's High-Performance Computing Platform Uses Massive Multithreading. Microprocessor, January 2008.

KEARNS, M. Thoughts on Hypothesis Boosting. Unpublished manuscript, 1988. Project for Ron Rivest's machine learning course at MIT. Disponível em: <http://www.cis.upenn.edu/~mkearns >. Acesso em: 15 de agosto de 2008.

LCAD. Laboratório de Computação de Alto Desempenho. http://www.lcad.inf.ufes.br/index.php?option=com_content&task=view&id=14&Itemid=65, último acesso em 10/01/2009.

NVIDIA. Technical Brief: NVIDIA GeForce 8800 GPU Architecture Overview. 2006.

NVIDIA. NVIDIA CUDA: Compute Unified Device Architecture - Programming Guide 1.0. 2007.

NVIDIA. NVIDIA CUDA: Compute Unified Device Architecture - Programming Guide 2.0. 2008a.

NVIDIA. Technical Brief: NVIDIA GeForce GTX 200 GPU Architectural Overview. 2008b.

OLIVEIRA, E.; CIARELLI, P. M.; SOUZA, A. F. DE; BADUE, C. Using a Probabilistic Neural Network for a Large Multi-Label Problem. Proceedings of the 10th Brazilian Symposium on Neural Networks (SBRN'08), 2008

OLIVEIRA, E.; CIARELLI, P. M.; Santos, M. H.; Costa, B. O. Um Modelo Algébrico para Representação, Indexação e Classificação Automática de Documentos Digitais. Revista Brasileira de Biblioteconomia e Documentação, 3:75-98, 2007.

RANDIMA, F.;MARK, K. The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics, Addison Wesley, 2003.

SCAE, Equipe do Projeto. Relato de Cumprimento de Metas No. 1. 2007.

SCAE, Equipe do Projeto. Relato de Cumprimento de Metas No. 2, 2007.

SCAE, Equipe do Projeto. Relato de Cumprimento de Metas No. 3, 2008.

SCHAPIRE, R. E. The Strength of Weak Learnability. Machine Learning, 5(2): 197-227, June, 1990.

SEBASTIANI, F. Machine Learning in Automated Text Categorization. ACM Computing Surveys, 34(1): p. 1-47, 2002.

SPEACHT, D. Probabilistic Neural Networks. Elsevier Science Ltd, 3(1):109–118, 1990.

8 Apêndice: Protótipo do SCAE-Fiscal – Atualização

O Sistema de Codificação Automática de Atividades Econômicas (SCAE) possui a arquitetura apresentada na Figura 8-1. O Sistema pode ser utilizado de duas formas, via linha de comando na máquina onde o SCAE está instalado, ou via navegador (*browser*), que se comunica com o módulo Servidor de Aplicação (SA) do SCAE. Este, por sua vez, se comunica com os outros dois módulos do SCAE: CORE e Banco de Dados, DB_CORE. Maiores detalhes a respeito das formas de utilização do SCAE na seção 8.1.4.2.

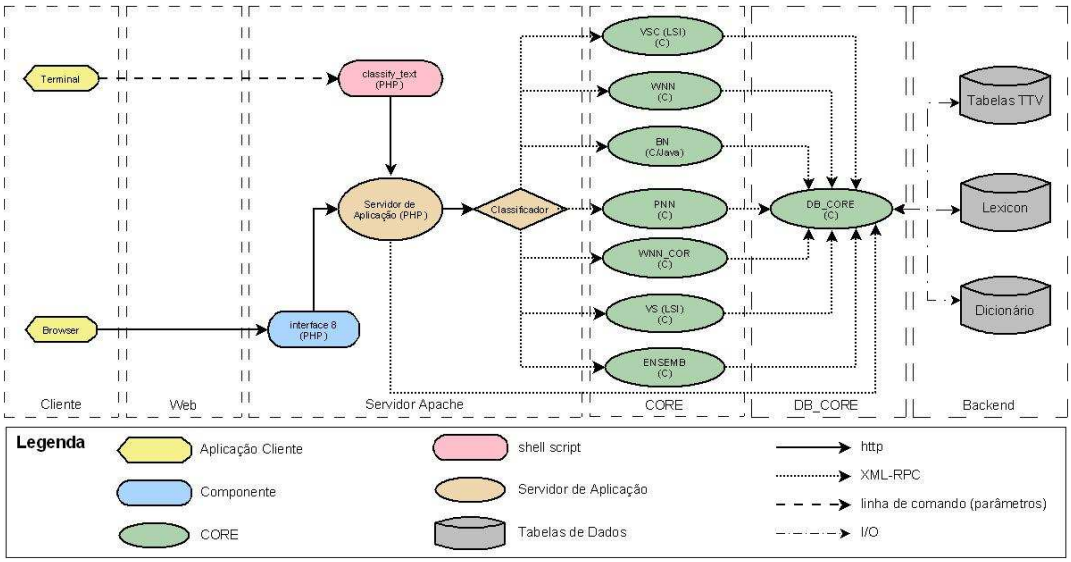

Figura 8-1: Arquitetura do SCAE

Em uma solicitação de classificação de atividade econômica, o usuário envia ao SA uma descrição de atividade econômica. O SA, por sua vez, envia essa descrição ao CORE, que a classifica e que, por sua vez, devolve os códigos CNAE juntamente com as medidas de confiança quanto às associações destes códigos com a descrição de atividade econômica recebida. De posse dos códigos CNAE, o SA requisita ao DB_CORE os textos que descrevem os referidos códigos.

Para realizar as classificações, os COREs também se comunicam com o módulo DB_CORE, que é responsável por armazenar todo o conhecimento do SCAE (dicionário eletrônico; representação interna ao sistema da tabela CNAE e das descrições de atividades econômicas usadas em treino; etc.).

O SA foi desenvolvido em PHP, o DB_CORE em C e Java, e o COREs em C, Java e C+CUDA (*Compute Unified Dedvice Architeture - CUDA*).

8.1.1 Preparação para a Instalação do SCAE

8.1.1.1 Requisitos Mínimos Necessários

Para instalar o SCAE em uma única máquina faz-se necessário que esta possua as seguintes características mínimas:

- Espaço em disco: 30 GB ou superior.
- Memória: 3 GB ou superior
- Sistema Operacional: Fedora Core 8.
- Placa de vídeo Nvidia *CUDA enable* (opcional).

8.1.1.2 Preparando o Ambiente para Instalação do SCAE

A preparação do ambiente para instalação do SCAE consiste, basicamente, na instalação e configuração das bibliotecas e pacotes de software de que ele necessita. Abaixo são listadas as bibliotecas e pacotes e os passos necessários para sua instalação:

- w3c-libwww
- w3c-libwww-devel
- xmlrpc-c
- xmlrpc-c-devel
- xforms
- xforms-devel
- freeglut
- freeglut-devel
- libnet-devel
- byacc
- httpd
- php
- wine

Para instalação dos mesmos, efetue *login* como usuário *root* e entre com o comando a seguir para instalar as bibliotecas e pacotes com a ferramenta *yum*. Note que, caso alguma das bibliotecas ou pacotes já estejam instalados, o *yum* cuidará para que esta ou este não sejam instalados novamente e, caso haja dependência com outros componentes de software, o *yum* cuidará de instalar tais componentes. {Responda "y" quando for perguntado ***"Is this ok [y/N]:"***}

```
yum install -y w3c-libwww w3c-libwww-devel xmlrpc-c xmlrpc-c-devel
xforms xforms-devel freeglut freeglut-devel libnet-devel byacc httpd
php wine
```

8.1.1.3 Aplicativos instaláveis a partir do DVD

JDK

O SCAE necessita de uma máquina virtual Java instalada na máquina onde estarão rodando o DB_CORE e o BN_CORE. A versão necessária é a 1.6.0_04. Para facilitar a instalação, foi disponibilizado no DVD o arquivo binário correspondente a esta versão.

Para copiar o arquivo *bin* disponibilizado no DVD para o */opt*:

```
#> cp /media/SCAE/libs/jdk/jdk-6u4-linux-i586.bin /opt
```

Transforme este arquivo em um executável:

```
#> chmod +x /opt/jdk-6u4-linux-i586.bin
```

Mude para o diretório /opt e instale o pacote com o comando a seguir:

```
#> cd /opt

#> ./jdk-6u4-linux-i586.bin
```

A licença do pacote será exibida (aperte a tecla de espaço para passar cada tela) e, em seguida, você será perguntado se concorda com ela. Responda '*yes*' para dar início a instalação. Em seguida, copie o script de configuração com os seguintes comandos, que também o tornam executável:

```
#> cp /media/SCAE/libs/jdk/java.sh /etc/profile.d/

#> chmod +x /etc/profile.d/java.sh
```

Agora execute o seguinte comando para atualizar as variáveis de ambiente do arquivo java.sh:

```
#> source /etc/profile.d/java.sh
```

Então, execute o seguinte comando para verificar se o *path* está correto:

```
#> which java
```

Você deverá obter algo como:

```
/opt/jdk-latest/bin/java
```

Agora execute os seguintes comandos:

```
#> /usr/sbin/alternatives --install /usr/bin/java java
/opt/jdk1.6.0_04/bin/java 2

#> /usr/sbin/alternatives --config java
```

Após ter inserido o último comando, você será perguntado sobre o tipo de Java que deseja para o sistema. Pressione 3 (a opção que indica o JDK 1.6.0_04) e pressione *enter*.

Para confirmar a versão e a instalação do seu *Java Runtime Environment*, execute:

```
#> java -version
```

A saída esperada é:

```
java version "1.6.0_04"
Java™ SE Runtime Environment (build 1.6.0_04-b12)
Java HotSpot(TM) Client VM (build 10.0-b19, mixed mode, sharing)
```

Por último, adicione as seguintes linhas ao seu arquivo *.bashrc* no diretório home do *root*:

```
 #JAVA
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/jre/lib/i386
export LD_LIBRARY_PATH
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/jre/lib/i386/client
export LD_LIBRARY_PATH
```

MAE

Como *root*, copie a MAE para o */opt* com o comando a seguir:

```
#> cp -r /media/SCAE/libs/MAE /opt/
```

Compile a MAE com o comando a seguir:

```
#> cd /opt/MAE

#> make clean

#> make -f Makefile.no_interface

#> cd
```

Insira as seguintes linhas ao *.bashrc* do *root*:

```
# MAE
export MAEHOME=/opt/MAE
PATH=$PATH:$MAEHOME/bin
```

CUDA

O SCAE necessita do *Toolkit* do *CUDA* para rodar o VSC_CORE, que tem como requisito uma placa de vídeo da marca Nvidia que seja *CUDA enable*. Para verificar se a placa de vídeo da máquina na qual o SCAE está sendo instalado é Nvidia *CUDA enable*, verifique o *link* http://www.nvidia.com/object/cuda_learn_products.html.

Caso não encontre o modelo da placa de vídeo, não há empecilhos para rodar o SCAE. Todos os COREs funcionarão perfeitamente, com exceção apenas do VSC_CORE. Entretanto, a instalação do *Toolkit* é obrigatória.

Como *root*, copie a *Toolkit* do *CUDA* para o */opt* com o comando a seguir:

```
#> cp
/media/SCAE/libs/nvidia/NVIDIA_CUDA_Toolkit_2.0beta2_Fedora8_x86.run
/opt/
```

Transforme este arquivo em um executável:

```
#> chmod +x /opt/NVIDIA_CUDA_Toolkit_2.0beta2_Fedora8_x86.run
```

Mude para o diretório */opt* e instale o *Toolkit* com o comando a seguir:

```
#> cd /opt

#> ./NVIDIA_CUDA_Toolkit_2.0beta2_Fedora8_x86.run
```

Tecle <*enter*> duas vezes para instalar no diretório padrão.

Insira as seguintes linhas ao *.bashrc* do *root*:

```
#CUDA
PATH=$PATH:/usr/local/cuda/bin
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib
export PATH
export LD_LIBRARY_PATH
```

Ao finalizar a instalação das bibliotecas e pacotes, não esqueça de atualizar as variáveis de ambiente em todos os terminais abertos com o comando:

```
#> source .bashrc
```

8.1.2 Instalando o SCAE

8.1.2.1 Instalando o Corretor Ortográfico

O SCAE utiliza como corretor ortográfico uma versão modificada do corretor ortográfico *Aspell*. Com isto, existe apenas a necessidade de se configurar o diretório onde se encontram as bibliotecas desta versão.

Para configurar o corretor ortográfico do SCAE, inclua no arquivo *.bashrc* do *root* o caminho das bibliotecas do dicionário:

```
#SCAE SPELLER
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/relato4/CORES/DB_CORE/SPELLER/s
caeaspell-0.60.5/.libs
export LD_LIBRARY_PATH
```

Não se esqueça de atualizar as variáveis de ambiente em todos os terminais abertos com o comando:

```
#> source .bashrc
```

8.1.2.2 Instalando os Classificadores

Ainda como *root*, copie o SCAE para o diretório *root* com o comando:

```
#> cp -r /media/SCAE/relato4 /root
```

Em seguida, basta executar os comandos:

```
#> cd /root/relato4/CORES

#> ./configure
```

```
#> make

#> make install
```

O comando *make* compilará todos os CORES (ilustrados na Figura 8-1) existentes no SCAE.

Com a execução do comando *make install*, o SCAE adotará uma configuração *default* de treino e estará pronto a ser utilizado para fins de classificação. Contudo, caso se deseje adotar outras configurações de treino, vide a Seção 8.1.3, onde são apresentados mecanismos de construção de tabelas (Seção 8.1.3.1) e de execução de treino (Seção 8.1.3.3).

Após finalizar o *make install*, verifique se os classificadores estão executando. Para isso, execute o comando:

```
#> /etc/init.d/classifier_cores status
```

A saída esperada é apresentada a seguir, onde xxxxx representa o número do processo de cada classificador:

```
db_core (pid xxxxx) está rodando…
vs_core (pid xxxxx) está rodando…
wnn_core (pid xxxxx xxxxx) está rodando…
wnn_cor_core (pid xxxxx xxxxx) está rodando…
bnn_core (pid xxxxx) está rodando…
pnn_core (pid xxxxx) está rodando…
ensemb_core (pid xxxxx) está rodando…
```

Após executar o comando, caso a saída não seja igual à anterior, execute os seguintes comandos:

```
#> /etc/init.d/classifier_cores stop

#> /etc/init.d/classifier_cores start
```

Para maiores detalhes a respeito dos comandos acima consulte a Seção 8.1.4.

8.1.2.3 Instalando o classificador VSC_CORE

No procedimento anterior, o classificador VSC_CORE não é instalado. Caso a placa de vídeo da máquina onde o SCAE está sendo instalado seja Nvidia *CUDA enable*, siga os procedimentos a seguir para instalar o VSC_CORE.

Ainda como *root*, mude para o diretório relato4/CORES/VSC_CORE executando os comandos:

```
#> cd relato4/CORES/VSC_CORE

#> nohup ./vsc_core &
```

Altere para o diretório relato4/CORES/USER_INTERFACE e treine o classificador com os comandos:

```
#> cd relato4/CORES/USER_INTERFACE

#> ./vsc_default_train.bat
```

Ao finalizar o treino do VSC_CORE, renicialize o CORE carregando o treino anterior, com os comandos:

```
#> pkill vcs_core

#> nohup ../VSC_CORE/vsc_core &

#> ./user_interface read_ports ports.cfg reload VSC
TREINAMENTO_DEFAULT
```

8.1.2.4 Instalando a Interface Web

Configure o servidor *Apache* para inicializar durante o *boot* da máquina:

```
#> chkconfig --level 5 httpd on
```

Caso o *Apache* não esteja ativo, inicialize-o com o comando:

```
#> /etc/init.d/httpd restart
```

Copie todo o conteúdo do diretório relato4/scaeweb para */var/www/html/*:

```
#> cp -r /root/relato4/scaeweb /var/www/html/
```

Altere o usuário e o grupo do diretório *scaeweb* para o usuário do *Apache*. Por padrão, o usuário e o grupo do servidor *Apache* 2.2.3, que já vem com o Fedora Core 8, é *apache* e *apache*, respectivamente:

```
#> chown apache:apache /var/www/html/scaeweb -R
```

Você pode então acessar o SCAE com um browser pela URL: http://127.0.0.1/scaeweb, onde a imagem da Figura 8-5 deve ser mostrada. Caso ela não apareça, há problemas de configuração no *Apache* ou PHP.

8.1.3 Configuração

8.1.3.1 Criação de Tabelas de Vetores de Treino e Teste (Tabelas TTV)

Após a instalação dos classificadores, podemos utilizar o DB_CORE para criar um conjunto básico de tabelas TTV que permitem demonstrar as funcionalidades do SCAE (assuma sempre que o diretório corrente é o último especificado por um comando cd). Para isto é necessário executar o script abaixo:

```
#> cd /root/relato4/CORES/DB_CORE

#> ./default_build.bat
```

A saída deste comando deve ser:

```
Locale set to pt_BR.UTF-8.
Loading known tables from .csv
Number of known tables = 29
Creating KNOWN_LEXICONS_saved.csv file.
Creating KNOWN_TTVS_saved.csv file.
Creating KNOWN_TRAININGS_saved.csv file.
Creating KNOWN_TESTS_saved.csv file.
Loading dictionary .csv
Number of dictionary words = 23160
Number of distincts words = 17365
Loading cnae subclasses .csv
Number of CNAE-Subclasses = 1183
Loading dados 'CSV_FILES/dados_vitoria_110.csv'
Number of economic activities descriptions = 14204
Number of distinct economic activities descriptions = 3281
Number of replicated cnae codes in the same activity = 0
Lexicon size = 5018. Number of words discarded due to word frequency
(PFS) = 0.
TTV 'TTV_C1S_DESC_TF' size = 1183
TTV 'TTV_DVS1_OBJS_TF' size = 3281
TTV 'TTV_C1S_DESC_TFIDF' size = 1183
TTV 'TTV_DVS1_OBJS_TFIDF' size = 3281
Saving tables…Done!
```

O *script default_build.bat* permite a criação das tabelas de treino e teste do SCAE. Ele realiza as seguintes ações, conforme demonstra a Figura 8-2:

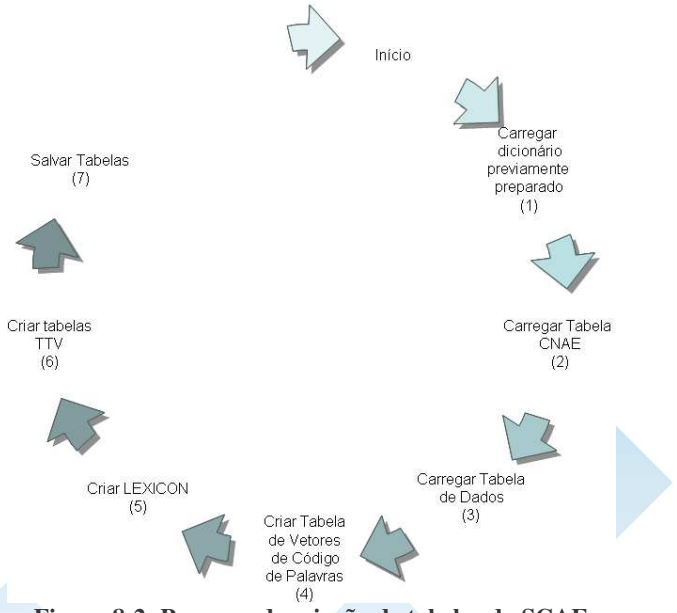

Figura 8-2: Processo de criação de tabelas do SCAE

As ações podem ser executadas com os seguintes parâmetros:

(1) `load_dictionary <NOME_DICIONARIO>`

(2) `load_csv_cnae_subclasse <NOME_TABELA_CNAE>`

(3) `load_csv_dados <NOME_TABELA_DADOS>`

(4) `create_word_vectors_table <<NOME_TABELA>-<NOME_CAMPO>>`

(5)
```
create_lexicon <NOME_LEXICON>
<"<DESCRICAO_LEXICON>"> <NUMERO_TABELAS>
<NOME_TABELA>-<<NOME_CAMPO>:<INI>:<FIM>>
[<NOME_TABELA2>-<<NOME_CAMPO2>:<INI2>:<FIM2>>...
<NOME_TABELAn>-<<NOME_CAMPOn>:<ININ>:<FIMn>>]
<"[CLASSE_GRAMATICAL1] [CLASSE_GRAMATICAL2] ...
[CLASSE_GRAMATICALn]"> <PFS> <EXCEPTIONS>
```

(6)
```
create_ttv <NOME_TTV> <NOME_LEXICON>
<NOME_TABELA> <NUMERO_CAMPOS> <NOME_CAMPO>
[<NOME_CAMPO2> ... <NOME_CAMPOn>]
<TIPO_CONSTRUCAO> <PESO_TERMOS>
```

(7) `save_tables_in_binary_format`

Os termos separados pelos sinais de maior e menor (<>) representam parâmetros obrigatórios e os termos separados por colchetes ([]) são opcionais. Tais parâmetros são descritos a seguir:

1. NOME_DICIONARIO – representa um dicionário preparado para o SCAE. Por meio de pesquisa exaustiva foram preparados vários dicionários. Estes dicionários foram gerados a partir de várias tabelas de dados, onde algumas destas tabelas foram fornecidas para o Projeto enquanto que outras foram obtidas por meio de pré-processamentos:
    a. NILC – Dicionário da Língua Portuguesa (Brasil), fornecido pelo Núcleo Interinstitucional de Lingüística Computacional (NILC)
    b. CNAE_110_SUBCLASSE – Tabela CNAE 1.1
    c. CNAE_110_SUBCLASSE_CORRIGIDO – Correção ortográfica manual da Tabela CNAE 1.1.
    d. DADOS_VITORIA_SUB – Tabela das descrições das atividades econômicas de empresas localizadas na região de Vitória (ES) com seus respectivos códigos CNAE.
    e. DADOS_VITORIA_SUB_CORRIGIDO – Correção ortográfica manual da tabela DADOS_VITORIA_SUB.
    f. DADOS_BH_SUB_110 – Tabela das descrições das atividades econômicas de empresas localizadas na região de Belo Horizonte (MG) com seus respectivos códigos CNAE.

Para a geração das bases foram utilizados os seguintes processos, algumas vezes em separado e outras em seqüência, conforme mostra a Tabela 8-1:

1. Correção Ortográfica Manual
2. Criação de Tabelas de Dicionário Canônico
3. Criação de Tabelas de Dicionário Radicalizado

Tabela 8-1: Lista de Tabelas de Dicionários do SCAE

Valor	Origem						Processo		
	A	B	C	D	E	F	1	2	3
DICIONARIO_SUBCLASSE		X						X	
DICIONARIO_COMPLETO	X							X	
DICIONARIO_COMPLETO_CORRIGIDO	X		X		X		X	X	
DICIONARIO_110_SUB+BH		X				X		X	
DICIONARIO_COMPLETO+BH	X					X		X	
DICIONARIO_COMPLETO+BH_SEM_N.C								X	
DICIONARIO_SEM_STOP_STEMM_SEM_ACCENT		X		X					X
DICIONARIO_SEM_STOP_STEMM		X		X					X

2. NOME_TABELA_CNAE – representa os tipos de Tabela CNAE existentes no DB_CORE. Atualmente existem os seguintes tipos:
   a. CNAE_110_SUBCLASSE – Tabela CNAE 1.1
   b. CNAE_110_SUBCLASSE_CORRIGIDO – Correção ortográfica manual (com intervenção humana) da Tabela CNAE 1.1.
   c. CNAE_110_SUBCLASSE_CORRIGIDO_AUTO – Correção ortográfica automática (sem intervenção humana, realizada com o corretor ortográfico do SCAE) da Tabela CNAE 1.1.
3. NOME_TABELA_DADOS – representa os tipos de Tabela de Dados existentes no DB_CORE. Suas tabelas foram geradas de forma análoga às Tabelas CNAE sendo que, tendo por base a tabela das descrições de atividades econômicas de empresas localizadas na região de Vitória (ES) com seus respectivos códigos CNAE. Atualmente existem os seguintes tipos:
   a. DADOS_VITORIA_SUB_110 – Tabela das descrições das atividades econômicas de empresas localizadas na região de Vitória (ES) com seus respectivos códigos CNAE-Subclasse da Tabela CNAE 1.1.
   b. DADOS_VITORIA_SUB_110_CORRIGIDO – Correção ortográfica manual (com intervenção humana) da tabela de dados DADOS_VITORIA_SUB_110.
   c. DADOS_VITORIA_SUB_110_CORRIGIDO_AUTO – Correção ortográfica automática (sem intervenção humana, realizada com o

corretor ortográfico do SCAE) da tabela de dados DADOS_VITORIA_SUB_110.

4. NOME_TABELA – este parâmetro indica a tabela utilizada para gerar o vetor de palavras, o *lexicon* (conjunto de palavras que formam um vocabulário de interesse) ou as tabelas de treino e teste. Atualmente, o sistema permite gerar o vetor de palavras para as seguintes tabelas:
    a. Tabela CNAE – logo, o sistema reconhece os valores identificados no parâmetro 1 (NOME_TABELA_CNAE, acima). Caso seja desejado criar o vetor de palavras a partir desta tabela, o usuário deverá informar para o parâmetro NOME_CAMPO (que representa o campo de onde serão recuperados os descritores para se formar o vetor de palavras) o valor DESCRICAO_SUB.
    b. Tabela de Dados – logo, o sistema reconhece os valores identificados no parâmetro 3 (NOME_TABELA_DADOS, acima). Caso seja desejado criar o vetor de palavras a partir desta tabela, o usuário deverá informar para o parâmetro NOME_CAMPO o valor OBJETO_SOCIAL.
5. NOME_LEXICON – este parâmetro indica o nome do *lexicon* a ser criado. Ele pode ser um texto livre e não deve conter espaços ou caracteres especiais. Contudo, sugere-se que o mesmo possua o seguinte padrão de nomenclatura:

    LEXICON[_LESS<[_ART][_CONJ][_CONTR][_INTERJ][_PREP][_PRON]>], onde os valores acima representam as classes gramaticais a serem removidas, conforme será explicado a seguir. Como por exemplo, podemos citar: LEXICON, LEXICON_LESS_ART, LEXICON_LESS_CONJ, LEXICON_LESS_INTERJ, LEXICON_LESS_ART_PRON, etc.
6. CLASSE_GRAMATICAL – este parâmetro indica o nome da classe gramatical a ser excluída na construção do *lexicon*. Podem ser informados os seguintes valores para a classe gramatical:
    a. abr. – indica que removerá todos as abreviações.
    b. adj. – indica que removerá todos os adjetivos.
    c. adv. – indica que removerá todos os advérbio.
    d. art. – indica que removerá todos os artigos.
    e. conj. – indica que removerá todas as conjunções.
    f. contr. – indica que removerá todas as contrações
    g. interj. – indica que removerá todas as interjeições.
    h. nom. – indica que removerá todos os nomes próprios.
    i. num. – indica que removerá todos os numerais.
    j. prep. – indica que removerá todas as preposições.
    k. pron. – indica que removerá over todos os pronomes.

l. sig. – indica que removerá todas as siglas.

m. sub. – indica que removerá todos os substantivos.

n. v. – indica que removerá todos os verbos.

A título de exemplo, a Tabela 8-2 indica um conjunto de nomes sugeridos de acordo com um subgrupo de classes gramaticais a serem removidas (apenas o subgrupo em que obtivemos os melhores resultados é mostrado):

Tabela 8-2: Exemplo de nomes para o *lexicon*

Nome sugerido	art.	Conj.	contr.	Interj.	prep.	pron.
LEXICON						
LEXICON_LESS_ART	X					
LEXICON_LESS_CONJ		X				
LEXICON_LESS_ INTERJ				X		
LEXICON_LESS_ART_PRON	X					X

7. DESCRICAO_LEXICON – este parâmetro indica a descrição do *lexicon* a ser criado. Ele pode ser um texto livre. Sugere-se que o mesmo identifique as classes gramaticais que estão sendo removidas, conforme demonstrado na Tabela 8-3:

Tabela 8-3: Exemplo de descrições para o *lexicon*

Nome sugerido	art.	Conj.	contr.	Interj.	prep.	pron.
Lexicon sem remoção de classes gramaticais						
Lexicon sem artigo	X					
Lexicon sem conjunção		X				
Lexicon sem interjeição				X		
Lexicon sem artigo e sem pronome	X					X

8. NUMERO_TABELAS – este parâmetro indica o número de tabelas a serem utilizadas para a construção do *lexicon*.
9. INI – este parâmetro indica o número da linha inicial da tabela utilizada para a construção do *lexicon*.

10. FIM – este parâmetro indica o número da linha final da tabela utilizada para a construção do *lexicon*. Não pode ser informado um número maior do que o número de linhas da Tabela de Dados ou CNAE. Os limites existentes atualmente estão apresentados a seguir:

Tabela 8-4: Limites das Tabelas existentes atualmente no SCAE

Tabela	INI	FIM
DADOS_VITORIA_SUB_110	0	3280
DADOS_VITORIA_SUB_110_CORRIGIDO	0	3280
DADOS_VITORIA_SUB_110_CORRIGIDO_AUTO	0	3280
CNAE_110_SUBCLASSE	0	1182
CNAE_110_SUBCLASSE_CORRIGIDO	0	1182
CNAE_110_SUBCLASSE_CORRIGIDO_AUTO	0	1182

11. PFS – este parâmetro indica a freqüência acima da qual a palavra não será incluída no lexicon.
12. EXCEPTIONS – este parâmetro indica se as preposições no arquivo CSV_FILES/exceptions.csv são para serem mantidas ou não na geração do *lexicon*. Com o valor EXCEPTIONS_OFF, essas palavras não fazem parte do *lexicon*. Com o valor EXCEPTIONS_ON, essas palavras passam a constituí-lo.
13. NOME_TTV – este parâmetro indica o nome da tabela de treino e teste. Ele pode ser um texto livre, porém não deve conter espaços ou caracteres especiais.
14. NUMERO_CAMPOS – este parâmetro indica o número de campos da tabela que será utilizada para a criação da TTV. Por default, utiliza-se apenas o número 1.
15. TIPO_CONSTRUCAO – este parâmetro indica o método de construção da TTV. Atualmente, o DB_CORE aceita somente o valor DEFAULT.
16. PESO_TERMOS – este parâmetro denota a função para o cálculo dos pesos dos termos, que podem ser computados como a freqüência dos termos (*term frequency* (TF)) ou como a freqüência dos termos multiplicada pela freqüência inversa dos mesmos nos documentos (*term frequency inverse document frequency* (TFIDF) (SCAEb, 2007)). Atualmente este parâmetro permite os valores TF e TFDIF.

Criação de Tabelas de Dicionários Radicalizados

Após a instalação dos classificadores, é possível também criar dicionários radicalizados utilizando o procedimento de *Stemming*. Para isto, é necessário executar o script `build_filtered_dictionary_from_tables.bat`, que usa o executável **db_core** para criar um dicionário filtrado. O script é apresentado abaixo.

```
./db_core \
load_csv_cnae_subclasse <NOME_TABELA_CNAE>\
load_csv_dados <NOME_TABELA_DADOS>\
create_filtered_dictionary_from_tables
<NOME_DICIONARIO_A_GERAR> <ID_FILTRO> <"[[OPERACAO1] [+
<OPERACAO2> [+ <OPERACAO3>]]]"> <NUMERO_TABELAS>
<NOME_TABELA>-<NOME_CAMPO> [<NOME_TABELA2>-<NOME_CAMPO2> ..
<NOME_TABELAn>-<NOME_CAMPOn>]
```

onde:

1. NOME_DICIONARIO_A_GERAR – representa o nome do dicionário a ser construído para o SCAE;
2. ID_FILTRO – um número inteiro identificando o filtro. Atualmente o sistema admite apenas um único valor (1);
3. OPERACAO – este parâmetro indica as operações que podem ser realizadas pelo filtro. Ele permite informar os seguintes valores:

Tabela 8-5: Operações de Filtro do SCAE

Operação	Descrição
STEMM	Permite reduzir as palavras existentes no dicionário na forma radicalizada
ACCENT	Permite remover os acentos das palavras
STOP	Permite remover as palavras que são consideradas *stop words* do dicionário.

As operações podem ser combinadas em até oito possibilidades, incluindo a opção de não se informar a operação. Neste caso deve-se informar branco (“”).

8.1.3.2 Correção Ortográfica das Bases

Após a instalação dos classificadores, é possível realizar a correção ortográfica das Bases utilizando o corretor ortográfico do SCAE.

Primeiramente, um dicionário baseado no formato do *Aspell* e uma lista de palavras com respectivas freqüências precisam ser gerados. Por padrão, o SCAE já possui o dicionário e a lista gerados, não havendo necessidade, por parte do usuário, de gerá-los.

O dicionário, arquivo *dict_scae.rws*, está localizado no diretório *DB_CORE/SPELLER/dictionary* e a lista de palavras, words_frequency.csv, está localizada no diretório *DB_CORE/CSV_FILES*.

A correção ortográfica de uma Base é realizada pelo *script build_corrected_tables.bat* localizado no diretório DB_CORE. O *script* é apresentado abaixo:

```
(1) db_core
(2) load_dictionary <NOME_DICIONARIO> \
(3) check_table        <NOME_LISTA_PALAVRA_FREQUENCIA>
    <NOME_TABELA1> <NUMERO_CAMPOS1> <NOME_CAMPO1>
    [check_table       <NOME_LISTA_PALAVRA_FREQUENCIA>
    <NOME_TABELA2> <NUMERO_CAMPOS2> <NOME_CAMPO2> \
                        ....
    check_table        <NOME_LISTA_PALAVRA_FREQUENCIA>
    <NOME_TABELAn> <NUMERO_CAMPOSn> <NOME_CAMPOn> \]
```

Os termos separados pelos sinais de maior e menor (<>) representam parâmetros obrigatórios. Tais parâmetros são explicados a seguir:

1. NOME_LISTA_PALAVRA_FREQUENCIA – representa a lista de palavras com as respectivas freqüências. Atualmente, o SCAE permite informar o valor WORDS_FREQUENCY.
2. NOME_TABELA – representa o nome da tabela a ser corrigida. Os possíveis valores para este parâmetro são os apresentados em NOME_TABELA_CNAE e NOME_TABELA_DADOS;
3. NUMERO_CAMPOS – este parâmetro indica o número de campos da tabela que será utilizada para a correção. Por default, utiliza-se apenas o número 1;
4. NOME_CAMPO – este parâmetro indica o nome do campo da tabela a ser corrigido. Os possíveis valores para este parâmetro foram explicados anteriormente.

Para executar o script digite na linha de comando:

```
#> cd /root/relato4/CORES/DB_CORE
#> ./build_corrected_tables.bat
```

Após a correção ortográfica, a tabela corrigida é salva no diretório *DB_CORE/CSV_FILES* com o mesmo nome da tabela original, mas com o prefixo _corrigido_auto.

8.1.3.3 Treino

Após criarmos as tabelas de treino e teste no DB_CORE, podemos realizar o treino dos COREs classificadores. O pré-requisito para realizar o treino de um CORE é que esse e o DB_CORE estejam ativos, ou seja, “escutando” em uma determinada porta.

O treino é realizado por meio de *scripts* localizados no diretório *relato4/CORES/USER_INTERFACE/*. A Tabela 8-6 mostra a relação entre os scripts e os COREs.

Tabela 8-6: Scripts de treino dos CORES

Nome do script	Treino do CORE
wnn_default_train.bat	WNN
wnn_cor_default_train.bat	WNN_COR
pnn_default_train.bat	PNN
vs_default_train.bat	VS
vsc_default_train.bat	VSC
bn_default_train.bat	BN
ensemb_default_train.bat	ENSEMB

Para realizar o treino de um determinado CORE, execute o seguinte comando:

```
#> ./<nome do script>
```

Caso tenha escolhido realizar o treino do WNN_COR, o script apresenta a seguinte mensagem na tela do terminal:

```
Locale set to pt_BR.UTF-8.

SERVERS'S STATUS:
*** DB is ON ***
*** WNN is OFF ***
*** VS is OFF ***
*** WNN_COR is ON ***
*** BN is OFF ***
*** ENSEMB is OFF ***
Training WNN_COR CORE …
Training successfully finished.
Saving TREINAMENTO_DEFAULT in WNN_COR CORE was completed.
```

O script de treino realiza as seguintes ações:

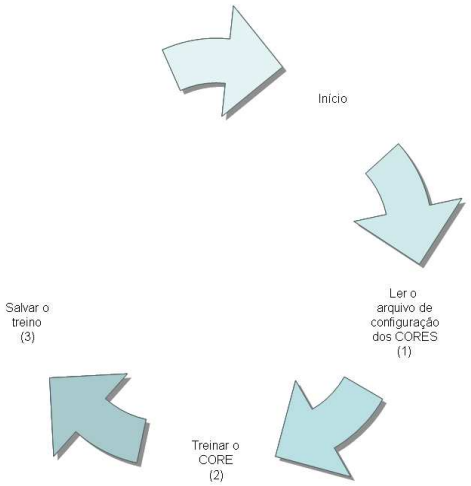

Figura 8-3: Ações realizadas no treino do CORE

As ações podem ser executadas com os seguintes parâmetros:

(1) `read_ports ports.cfg`

(2) `train <NOME_CORE> <NIVEL_CNAE> <NOME_TREINO> <"<DESCRICAO_TREINO>"> <NUMERO_TTVS> <NOME_TTV1 INI1 FIM1> [<NOME_TTV2 INI2 FIM2> ... <NOME_TTVn ININ FIMn>]`

(3) `save <NOME_CORE> <"<NOME_TREINO>">`

Os termos separados pelos sinais de maior e menor (<>) representam parâmetros obrigatórios. Tais parâmetros são explicados a seguir:

1. NOME_CORE – representa o nome do CORE que será treinado. Atualmente, os seguintes nomes são aceitos: WNN, WNN_COR, PNN, VS, VSC, BN e ENSEMB;
2. NIVEL_CNAE – indica em qual nível CNAE será realizado o teste. Os níveis CNAE podem ser: SECAO, DIVISAO; GRUPO, CLASSE e SUBCLASSE;
3. NOME_TREINO – representa o nome do treino. Pode ser um texto livre, porém não deve conter espaços ou caracteres especiais;
4. DESCRICAO_TREINO – é uma descrição do treino a ser realizado. Ele pode ser um texto livre. Sugere-se que o mesmo identifique as tabelas de treino e teste que estão sendo utilizadas. A descrição do treino deve ser colocado entre aspas duplas(");
5. NUMERO_TTVS – indica o número de tabelas de treino e teste que serão utilizadas para o treino;

6. NOME_TTV – indica o nome da tabela de treino e teste criada na execução do `default_build.bat`;
7. INI – número da linha inicial da tabela de treino e teste para o treino do CORE;
8. FIM – número da linha final da tabela de treino e teste para o treino do CORE. Não pode ser informado um número maior do que o número de linhas da Tabela de Dados ou CNAE. Os limites existentes foram apresentados na Tabela 8-4.

Os COREs, com exceção do ENSEMB, salvam o treinamento em um diretório localizado no diretório de cada CORE. A Tabela 8-7 mostra a relação entre o nome do CORE, o diretório do mesmo e o diretório de treino.

Tabela 8-7: Relação entre o nome do CORE e os diretórios de treino.

Nome do CORE	Diretório do CORE	Diretório de treino
WNN	WNN_CORE	WNN_MEMORIES
WNN_COR	WNN_COR_CORE	WNN_MEMORIES
PNN	PNN_CORE	PNN_NETWORKS
VS	VS_CORE	VECTORS_SPACES
VSC	VSC_CORE	VECTORS_SPACES
BN	BN_CORE	NETWORKS
ENSEMB	ENSEMB_CORE	-

8.1.4 Uso

Atualmente, o SCAE disponibiliza duas funcionalidades de uso: o teste e a classificação de atividades econômicas. Os procedimentos para utilizá-las são apresentados nas seções 8.1.4.1 e 8.1.4.2.

8.1.4.1 Testes

Após treinarmos um determinado CORE, podemos realizar o teste do desempenho do mesmo segundo diversas métricas. O pré-requisito para realizar o teste de um CORE é que esse e o DB_CORE estejam ativos, ou seja, “escutando” numa determinada porta, e o que CORE tenha sido treinado.

O teste é realizado por meio de *scripts* localizados no diretório *relato4/CORES/USER_INTERFACE/*. A Tabela 8-8 mostra a relação entre os *scripts* e o nome dos COREs.

Tabela 8-8: Scripts de teste dos CORES

Nome do script	Teste do CORE
wnn_default_test.bat	WNN
wnn_cor_default_test.bat	WNN_COR
pnn_default_train.bat	PNN
vs_default_test.bat	VS
vsc_default_train.bat	VSC
bn_default_test.bat	BN
ensemb_default_test.bat	ENSEMB

Para realizar o teste de um determinado CORE, execute o seguinte comando:

```
#> ./<nome do script>
```

Caso tenha escolhido realizar o teste do WNN_COR, o script apresenta a seguinte mensagem na tela do terminal:

```
Locale set to pt_BR.UTF-8.

SERVERS'S STATUS:
*** DB is ON ***
*** WNN is OFF ***
*** VS is OFF ***
*** WNN_COR is ON ***
*** BN is OFF ***
*** ENSEMB is OFF ***
Reloading training TREINAMENTO_DEFAULT in WNN_COR CORE ...
Training TREINAMENTO_DEFAULT in WNN_COR CORE was reloaded
successfully.
Testing WNN_COR CORE ...

#### METRICS ####
Ordinal ranking - One Error = 0,015844
Ordinal ranking - Ranking Loss = 0,000771
Ordinal ranking - Coverage = 3,674589
Ordinal ranking - Average Precision^d = 0,985469
Ordinal ranking - R-Precision^d = 0,974158
Ordinal ranking - Hamming Loss = 0,000088
Ordinal ranking - R-Hamming Loss = 0,051684
Ordinal ranking - Exact Match = 0,948812
Ordinal ranking - Microaveraged Precision = 0,987899
Ordinal ranking - Microaveraged Recall = 0,987899
Ordinal ranking - Macroaveraged Precision^d = 0,974158
Ordinal ranking - Macroaveraged Recall^d = 0,974158
Ordinal ranking - Macroaveraged Precision^c = 0,579466
Ordinal ranking - Macroaveraged Recall^c = 0,576256
Ordinal ranking - Microaveraged F_1 = 0,987899
Ordinal ranking - Macroaveraged F_1^d = 0,974158
Ordinal ranking - Macroaveraged F_1^c = 0,577375
Modified competition ranking - One Error = 0,016453
```

```
Modified competition ranking - Ranking Loss = 0,000771
Modified competition ranking - Coverage = 4,234613
Modified competition ranking - Average Precision^d = 0,984766
Modified competition ranking - R-Precision^d = 0,973833
Modified competition ranking - Hamming Loss = 0,000086
Modified competition ranking - R-Hamming Loss = 0,051084
Modified competition ranking - Exact Match = 0,954296
Modified competition ranking - Microaveraged Precision = 0,987212
Modified competition ranking - Microaveraged Recall = 0,989180
Modified competition ranking - Macroaveraged Precision^d = 0,973833
Modified competition ranking - Macroaveraged Recall^d = 0,976078
Modified competition ranking - Macroaveraged Precision^c = 0,579420
Modified competition ranking - Macroaveraged Recall^c = 0,576846
Modified competition ranking - Microaveraged F_1 = 0,988195
Modified competition ranking - Macroaveraged F_1^d = 0,974755
Modified competition ranking - Macroaveraged F_1^c = 0,577674
```

As seguintes ações são realizadas pelo script de teste:

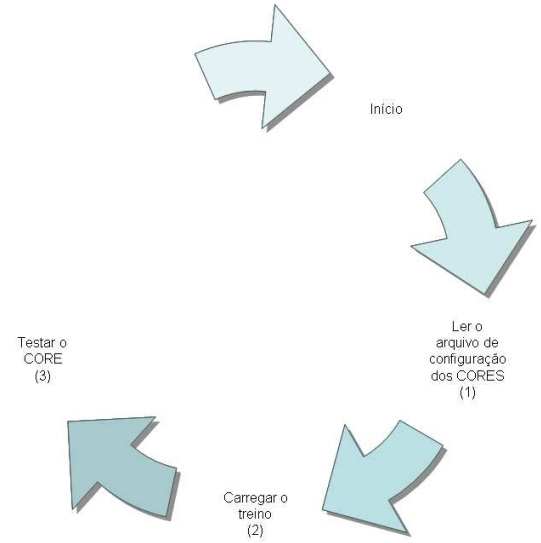

Figura 8-4: Ações realizadas no teste do CORE

Estas ações podem ser executadas com os seguintes parâmetros:

(1) `read_ports ports.cfg`

(2) `reload <NOME_CORE> <NOME_TREINO>`

(3) `test <NOME_CORE> <NOME_TREINO> <"<DESCRICAO_TESTE>"> <NUMERO_TTVS> <NOME_TTV1 INI1 FIM1> [<NOME_TTV2 INI2 FIM2> ... <NOME_TTVn INIn FIMn>]`

Os termos separados pelos sinais de maior e menor (<>) representam parâmetros obrigatórios. Tais parâmetros são explicados a seguir:

1. NOME_CORE – representa o nome do CORE que será treinado. Atualmente, os seguintes nomes são aceitos: WNN, WNN_COR, VS, VSC, BN e ENSEMB;
2. NOME_TREINO – representa o nome do treino utilizado para treinar o CORE;
3. DESCRICAO_TESTE – é uma descrição do teste a ser realizado. Ele pode ser um texto livre. Sugere-se que o mesmo identifique as tabelas de treino e teste que estão sendo utilizadas. A descrição do teste deve ser colocada entre aspas duplas (“”);
4. NUMERO_TTVS – indica o número de tabelas de treino e teste que serão utilizadas para o teste;
5. NOME_TTV – indica o nome da tabela de treino e teste criada na execução do `default_build.bat`;
6. INI – número da linha inicial da tabela de treino e teste para o teste do CORE;
7. FIM – número da linha final da tabela de treino e teste para o teste do CORE. Não pode ser informado um número maior do que o número de linhas da Tabela de Dados ou CNAE. Os limites existentes foram apresentados na Tabela 8-4.

8.1.4.2 Classificação de Atividades Econômicas

Além do treino e teste, o sistema SCAE permite que descrições de atividades econômicas na forma de texto livre sejam classificadas (categorizadas), ou seja, dada uma descrição de atividade econômica, o sistema retorna possíveis códigos CNAE. A classificação de atividades econômicas pode ser feita de duas maneiras pelo sistema: uma é por meio de um *browser*, e a outra é por meio do script *classify_text.bat* localizado no diretório relato4/CORES/USER_INTEFACE.

Classificação via Browser

Na primeira opção, com o *browser* aberto, o usuário deve digitar a URL http://127.0.0.1/scaeweb. Como conseqüência, aparecerá uma página *Web* semelhante à Figura 8-5. Para realizar a classificação, o usuário precisa: digitar a descrição da atividade econômica no campo Descrição das Atividades e selecionar qual CORE que será utilizado para realizar a classificação. Atualmente, as opções de CORE disponíveis no SCAE são WNN, WNN_COR, PNN, VS(LSI), VSC, BN e ESEMB. Lembrando que, o VSC deve ser selecionado somente se foi instalado conforme a Seção 8.1.2.3.

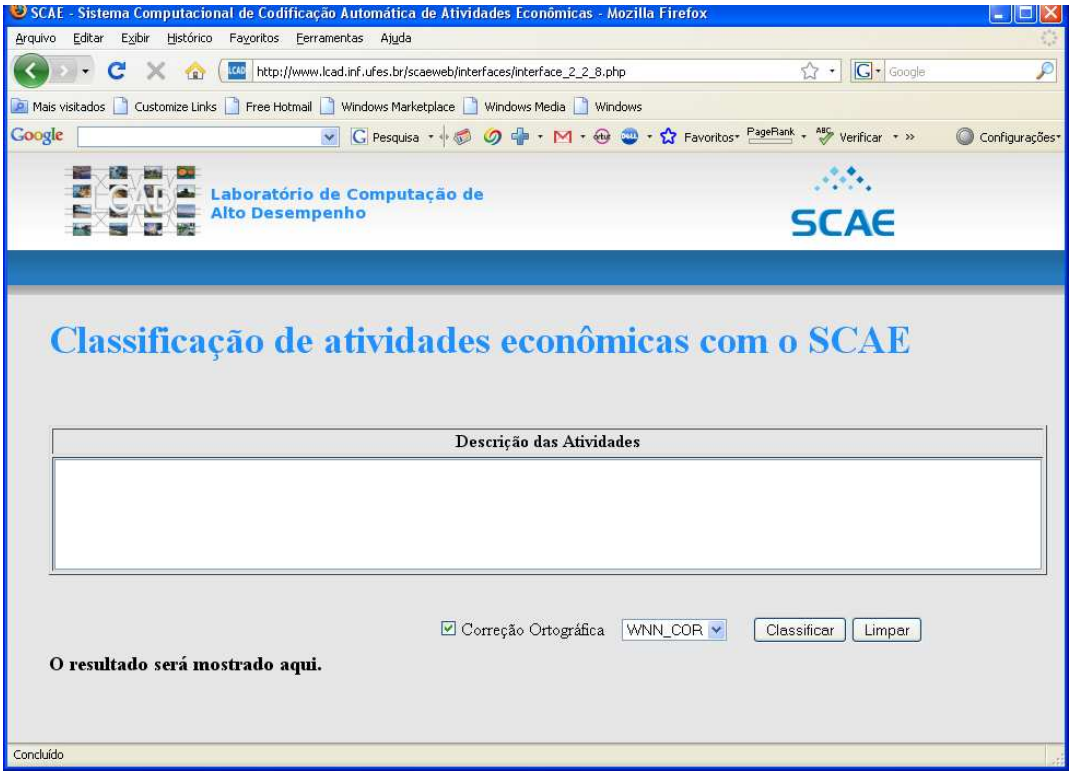

Figura 8-5: Interface Web de classificação de atividades.

Caso deseje que o texto digitado seja corrigido automaticamente pelo corretor ortográfico do SCAE, marque a opção Correção Ortográfica. Por padrão, essa opção está selecionada. Após preencher todas as informações necessárias, o usuário deve clicar no botão Classificar para classificar a atividade descrita. Ao clicar no botão Classificar, uma mensagem "Classificando" é mostrada informando o status da classificação da atividade. Para apagar as informações digitadas na página, o usuário pode clicar no botão Limpar ou pressionar a tecla F5.

O resultado da classificação é mostrado no item "O resultado será mostrado aqui.". Esse item também é utilizado para informar os possíveis erros decorrentes de comunicação com os CORE s, mensagens de erro XML, campos preenchidos incorretamente, etc. A Figura 8-6 mostra a classificação da atividade "Cultivo de arroz" utilizando o WNN_COR.

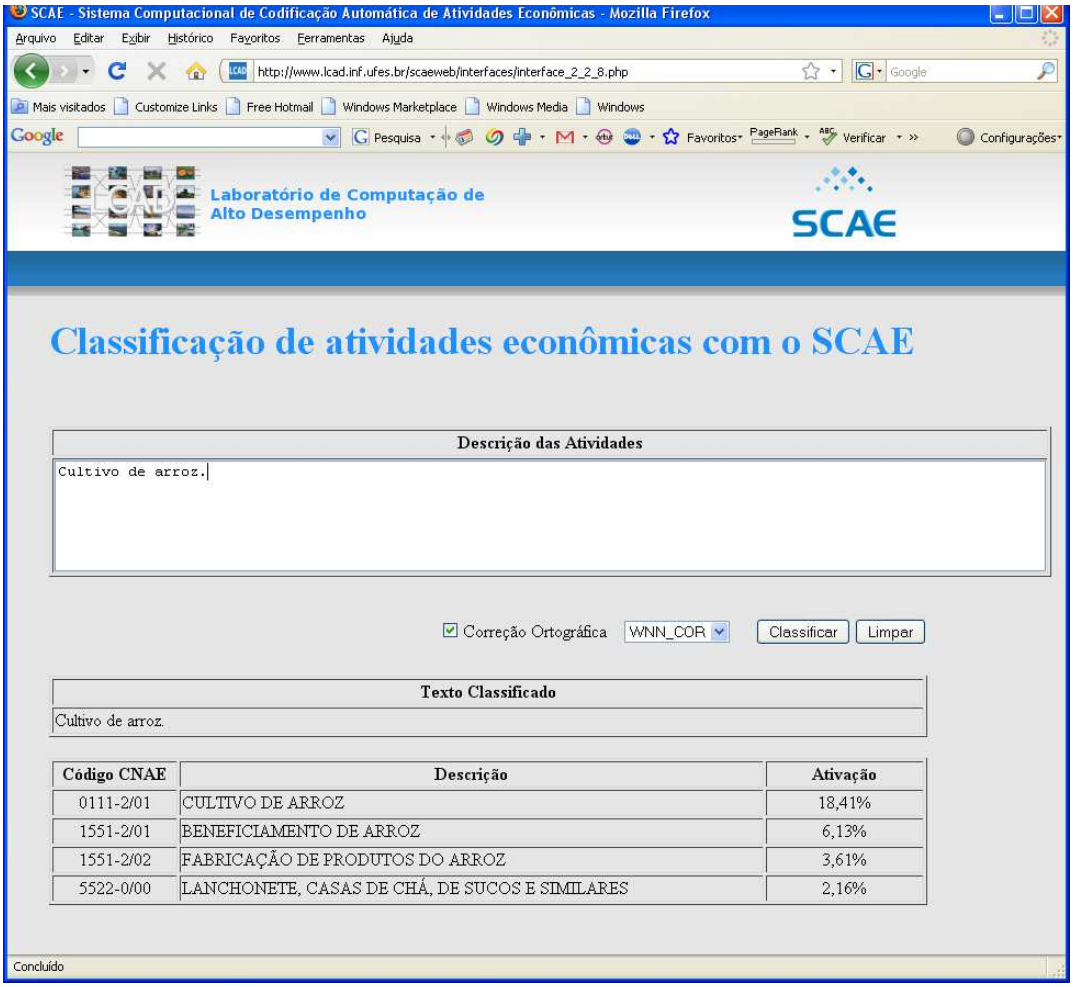

Figura 8-6: Classificação da atividade Cultivo de arroz com o CORE WNN_COR pelo browser

O resultado da classificação é mostrado no formato de tabela, conforme Figura 8-6, onde a primeira coluna representa o código CNAE-Subclasse, a segunda a descrição do código (denominação do código segundo a tabela CNAE) e a terceira a Ativação, no caso do WNN e WNN_COR, ou Cosseno do Ângulo, no caso do VS(LSI) e VSC, ou Crença, no caso do PNN, BN e ENSEMB, atribuída pelo CORE.

Caso o usuário tenha digitado a palavra "arroz" incorretamente como "Cultivo de aroz" e selecionado a opção de correção ortográfica, o texto é corrigido automaticamente e mostrado em Texto Classificado com as palavras corrigidas em vermelho, conforme Figura 8-7.

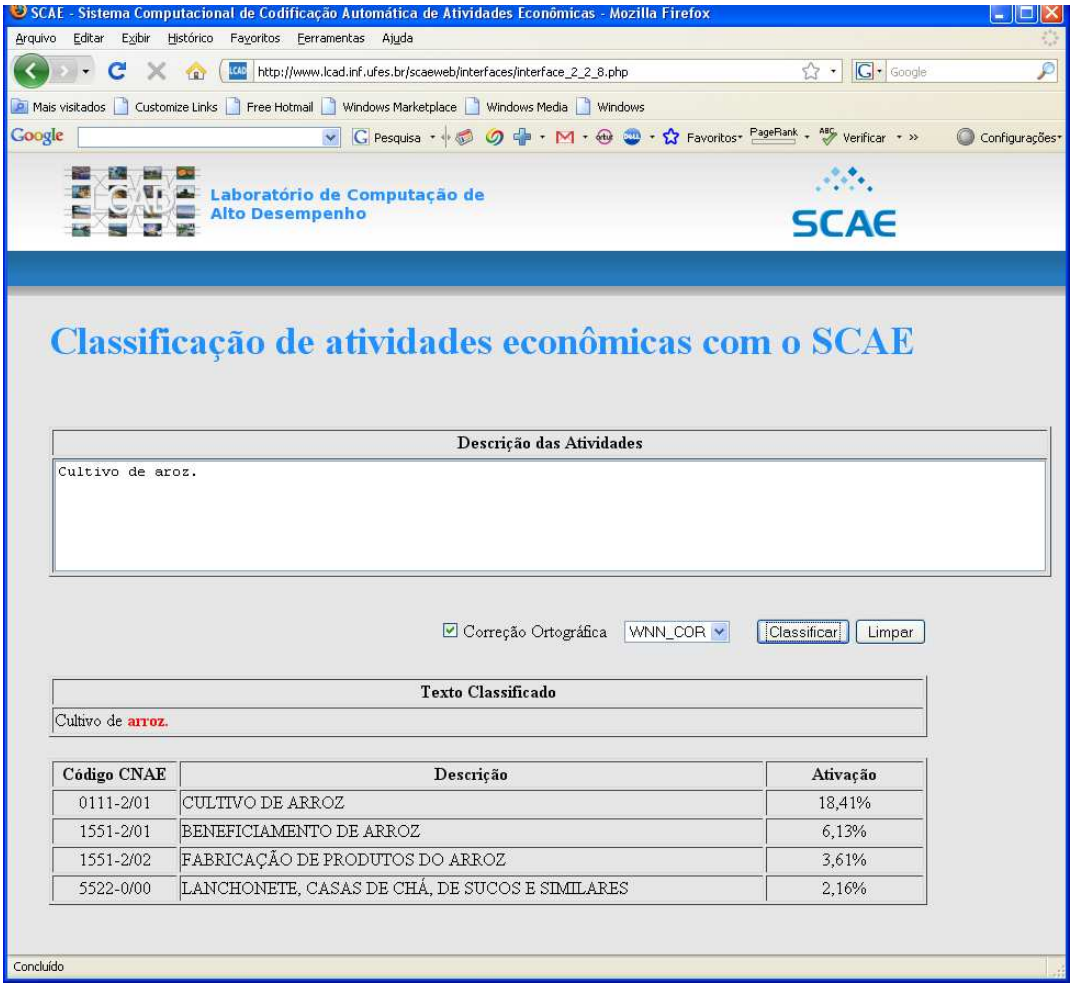

Figura 8-7 - Classificação de atividade econômica com correção ortográfica pelo *browser*.

Classificação via Script

Para realizar a classificação de atividades econômicas utilizando o *script*, o usuário deve editar o arquivo *classify_text.bat* localizado no diretório *relato4/CORES/USER_INTEFACE* e executar o seguinte comando:

```
#> ./classify_text.bat
```

A ação de classificação é executada pelo seguinte parâmetro:

(1) `classify_text <"<NOME_CORE>"> <"<DESCRICAO ATIVIDADE>"> <"<CORRECAO_ORTOGRAFICA>">`

Os termos separados pelos sinais de maior e menor (<>) representam parâmetros obrigatórios. Tais parâmetros são explicados a seguir:

1. NOME_CORE – representa o nome do CORE que será utilizado. Atualmente, os seguintes nomes são aceitos: WNN, WNN_COR, VS, VSC, BN e ENSEMB. O nome do CORE deve ser colocado entre aspas duplas (“”);
2. DESCRICAO_ATIVIDADE – representa a descrição da atividade a ser classificada. É um texto livre que deve colocado entre aspas duplas (“”).
3. CORRECAO_ORTOGRAFICA - indica se é para corrigir o texto digitado em DESCRICAO_ATIVIDADE. As opções são V indicando para corrigir ou F para não corrigir.

Por padrão, esse *script* é configurado para classificar a descrição de atividade “Cultivo de arroz” utilizando o WNN_COR:

```
#> ./user_interface \
classify_text "WNN_COR" "Cultivo de arroz." "F"
```

Importante observar as aspas duplas entre os parâmetros. A mensagem retornada pelo script é a seguinte:

```
Locale set to pt_BR.UTF-8.
CORE_NAME  : "WNN_COR"
TEXT TYPED : "Cultivo de arroz"
CORRECTION FLAG : "F"
0111-2/01       CULTIVO DE ARROZ        18,41%
1551-2/01       BENEFICIAMENTO DE ARROZ 6,13%
1551-2/02       FABRICAÇÃO DE PRODUTOS DO ARROZ 3,61%
5522-0/00       LANCHONETE, CASAS DE CHÁ, DE SUCOS E SIMILARES 2,16%
```

A segunda linha indica o nome do CORE selecionado, a terceira linha a descrição da atividade econômica para classificação, a quarta linha indica se foi feita opção pela correção ortográfica e as linhas restantes representam a classificação da atividade pelo CORE, sendo que a primeira coluna representa o código CNAE, a segunda o descritor e a terceira a Ativação (ou Crença, ou Cosseno do ângulo) atribuída pelo CORE.

8.1.4.3 Gerenciamento dos COREs

Além de permitir ao usuário a realização de procedimentos de treino e teste, o SCAE disponibiliza um script para gerenciamento dos COREs. Este script pode ser encontrado em /etc/init.d.

Através deste script é possível:

- Inicializar todos os COREs:

```
#> /etc/init.d/classifier_cores start
```

- Verificar status de todos os COREs:

```
#> /etc/init.d/classifier_cores status
```

- Parar todos os COREs:

```
#> /etc/init.d/classifier_cores stop
```