EXPLORANDO ESTRATÉGIAS BAYESIANAS EFICIENTES E EFICAZES PARA CLASSIFICAÇÃO DE TEXTO

FELIPE A. RESENDE VIEGAS

EXPLORANDO ESTRATÉGIAS BAYESIANAS EFICIENTES E EFICAZES PARA CLASSIFICAÇÃO DE TEXTO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Marcos André Gonçalves
Coorientador: Leonardo Chaves Dutra da Rocha

Belo Horizonte
Maio de 2015

FELIPE A. RESENDE VIEGAS

EXPLOITING EFFICIENT AND EFFECTIVE BAYESIAN STRATEGIES FOR TEXT CLASSIFICATION

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Marcos André Gonçalves
Co Advisor: Leonardo Chaves Dutra da Rocha

Belo Horizonte
May 2015

A. Resende Viegas, Felipe
V672e Explorando Estratégias Bayesianas Eficientes e Eficazes para Classificação de Texto / Felipe A. Resende Viegas. — Belo Horizonte, 2015
xx, 64 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais. Departamento Ciência da Computação.

Orientador: Marcos André Gonçalves.
Coorientador: Leonardo Chaves Dutra da Rocha.

1. Computação - Teses. 2. Indexação Automática - Teses . 3. Teoria Bayesiana de Decisão Estatística - Teses. I Orientador. II Coorientador. Título.

CDU 519.6*73(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Exploiting efficient and effective bayesian strategies for text classification

FELIPE AUGUSTO RESENDE VIEGAS

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. MARCOS ANDRÉ GONÇALVES - Orientador
Departamento de Ciência da Computação - UFMG

PROF. LEONARDO CHAVES DUTRA DA ROCHA - Coorientador
Departamento de Ciência da Computação - UFSJ

PROF. FERNANDO HENRIQUE JESUS MOURÃO
Departamento de Computação - UFSJ

PROF. RENATO MARTINS ASSUNÇÃO
Departamento de Ciência da Computação - UFMG

PROF. WELLINGTON SANTOS MARTINS
Instituto de Informática - UFG

Belo Horizonte, 22 de maio de 2015.

"Our virtues and our failings are inseparable, like force and matter. When they separate, man is no more."

(Nikola Tesla)

Resumo

Classificação automática de documentos (CAD) é a base de muitas aplicações importantes, tais como filtragem de spam, mineração de opinião, organizadores de conteúdo e identificação de autoria. Devido à sua simplicidade, eficiência, ausência de parâmetros e eficácia em diversos cenários, abordagens Naive Bayes (NB) são amplamente utilizadas como paradigmas de classificação. Contudo, estas abordagens não apresentam eficácia competitiva quando comparada a outros métodos de aprendizagem estatística modernos, como SVMs, em tarefas de CAD. Este comportamento está relacionado com a falta de robustez do NB em abordar algumas características das coleções reais de documentos, como desbalanceamento de classes e esparsidade dos dados. Nesta dissertação, investigamos se a combinação de alguns modelos de aprendizagem NB com diferentes propostas de ponderação de atributos pode melhorar a eficácia do NB em tarefas CAD, considerando várias coleções de dados do mundo real. Verificamos que uma combinação adequada destas estratégias pode produzir resultados equivalentes ou mesmo superiores quando comparado com SVM. Além disso, investigamos o relaxamento da suposição de independência dos atributos do Naive Bayes (também conhecido como abordagens Semi-Naive Bayes) em grandes coleções textuais. Dados os elevados custos computacionais dessas investigações, aproveitamos as arquiteturas das GPUs para apresentarmos uma versão massivamente paralela da abordagem NB. Além disso, com esta solução paralela, propomos quatro novas abordagens Semi-NB *lazy*. Em nossos experimentos, nossas novas soluções *lazy*, não só são mais eficientes do que as abordagens Semi-NB já existentes, assim como superam em termos de eficácia nossas estratégias NB incrementadas que já tiveram um desempenho melhor do que o SVM.

Palavras-chave: Classificação Automática de Documentos, Naive Bayes, Semi-Naive Bayes, Ponderação de Atributos, Paralelização.

Abstract

Automatic Document Classification (ADC) is the basis of many important applications such as spam filtering, opinion mining, content organizers and authorship identification. Naive Bayes (NB) approaches are widely used as a classification paradigm, due to their simplicity, efficiency, absence of parameters and effectiveness in several scenarios. However, NB solutions do not present competitive effectiveness in ADC tasks when compared to other modern statistical learning methods, such as SVMs. This is related to some characteristics of real document collections, such as class imbalance and feature sparseness. In this master thesis, we investigate whether the combination of some alternative NB learning models with different feature weighting techniques may improve the NB effectiveness in ADC tasks, considering several standard real-world datasets. We verify that a proper combination of these strategies may produce comparable or even superior results when compared to SVM. Moreover, we also present an investigation on the relaxation of the NB attribute independence assumption (aka, Semi-Naive approaches) in large text collections. Given the high computational costs of these investigations, we take advantage of current manycore GPU architectures and present a massively parallelized version of the NB approach. Moreover, supported by the parallel implementations, we propose four novel Lazy Semi-NB approaches. In our experiments, the new lazy solutions not only are more efficient than existing Semi-NB approaches, but also surpassed our improved NB solutions in terms of effectiveness that had already outperformed SVMs.

Palavras-chave: Text Classification, Naive Bayes Classifier, Semi-Naive Bayes heuristics, Feature Weighting techniques, Parallelization.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Context and Motivation

The amount of data created and shared nowadays in all types of platforms reached unprecedented levels, making the organization and extraction of useful knowledge from this huge amount of data one of the biggest challenges in Computer Science. Machine learning techniques, such as Automatic Document Classification (ADC), have demonstrated to be a viable path towards this goal. Particularly, ADC techniques aim at building effective models capable of associating documents with well-defined semantic categories in an automated way. ADC techniques are the core component of many important applications such as spam filtering (Hovold [2005]), organization of topic directories (Fahmi [2004]), identification of writing styles or authorship (Zheng et al. [2006]), among many others.

ADC methods usually exploit a supervised learning paradigm (Sebastiani [2002]), i.e., a classification model is first "learned" using previously labeled documents (training set), and then used to classify unseen documents (the test set). There is a plethora of supervised ADC algorithms available in the literature, such as Nearest-Neighbor classifiers (Yang [1999]), Support Vector Machines (Fan et al. [2008]), boosting (Schapire and Singer [2000]) and Bayesian models (Manning et al. [2008]). In this work, we focus on the latter approach, due to its simplicity, efficiency, and effectiveness in several scenarios. In particular, we focus on Naive Bayes approaches, the most widely used Bayesian paradigm for text classification.

Although being a widely used classification paradigm in ADC, other statistical learning methods, such as SVMs, have presented superior effectiveness than Naive Bayes approaches. The lack of robustness of NB is related to some characteristics present in real document collections, such as class imbalance and feature sparseness,

that may compromise some of the Naive Bayes premises (Rennie et al. [2003]; Kim et al. [2006]).

1.2 Master Thesis Hypotheses

In this section, we present the fundamental hypotheses used as guide for the construction of the work:

- There is a combination of modeling and feature weighting that makes NB competitive with state-of-the-art classifiers in real scenarios.
- Alleviating or removing the premise of independence among attributes may improve the effectiveness of NB in real scenarios.
- Adopting new parallel techniques (e.g., GPU) may allow more efficient NB implementation.
- This parallel strategy may make feasible the execution of more complex Bayesian networks in real scenarios, since they are extremely expensive in large scenarios.

1.3 Work Description

Naive Bayes is often used as a baseline in text classification because it is fast and easy to implement. However, some characteristics present in text collections, such as class imbalance and feature sparseness may compromise the Naive Bayes classification effectiveness. First, class imbalance problem (Matthijssen [2000]) happens when the number of documents of one or few classes spams most of the documents in a dataset. This introduces a bias in the trained classifier towards assigning most unseen documents to the largest classes, incurring in a poor classification effectiveness in the minority classes, the most important ones in many applications (e.g. email spam, vandalism). Second, sparseness problem is related to the low frequency of certain features (i.e., terms/words) in some documents (Matthijssen [2000]). In Naive Bayes, the conditional probability of a term a_j given a class c_i is estimated using all training documents from c_i in which a_j occurs. Such conditional probabilities may be negatively affected if a_j occurs only in a few documents, especially in smaller classes (i.e., those with few documents).

There are some proposals in the literature to overcome such problems, either by proposing some changes in the construction of the Naive Bayes learning model (Zhang

and Oles [2000]; Rennie et al. [2003]; Adewole et al. [2014]) or by means of feature weighting strategies more adequate to ADC, in a preprocessing phase prior to the model construction (Kim et al. [2006]; Matthijssen [2000]). Although both research lines may produce significant improvements in the Naive Bayes effectiveness, when compared to its original version, the resulting methods are still not capable of surpassing some state-of-the-art ADC method such as SVMs. Thus, as our **first contribution** we present a broad and original (never reported) study on the combination of different Naive Bayes-based learning models with different feature weighting strategies, evaluating these combinations in several real-world datasets. Our experimental results show that a proper combination of learning paradigms and weighting strategies may produce results comparable or even superior to SVMs in several datasets, at a lower cost.

A third research line that has been investigated in order to improve the Naive Bayes effectiveness are the so-called *Semi-Naive Bayes* methods, which relax the Naive Bayes attribute independence assumption (Manning et al. [2008]), by reduction of data (Chen and Wang [2012]) or, mainly, by means of extensions of the structure of the learning model to represent feature dependencies (Friedman et al. [1997]; Keogh and Pazzani [1999]). These strategies have produced gains when compared to NB in small datasets, such as those related to bioinformatics. However due to their high computational cost, inherent to the complexity of representing the term dependencies, they cannot scale to large classification tasks. Thus, investigating whether the relaxation of the Naive Bayes attribute independence assumption is effective in large ADC tasks is still an open problem. In this context, we present our **second** and **third** contributions. The **second contribution** corresponds to an exclusive parallel version of the Naive Bayes approach using graphic processing units (GPUs). This parallel version allowed us to implement some Semi-Naive Bayes approaches, capable of running in large text collections. Thus, our **third contribution** is an original study about the impacts of the relaxation of the Naive Bayes assumption in large ADC tasks. In this investigation, we also introduce four original parallel lazy Semi-Naive Bayes strategy proposals which exploit the information of the document to be classified to reduce the complexity of the Semi-Naive Bayes learned model. Our experimental results point out that further improvements can be obtained with these models, depending on some dataset characteristics.

In summary, the main research questions we address in this work are: (Q1) Can a proper combination of a Naive Bayes learning paradigm with feature weighting strategies, specially designed to deal with the ADC idiosyncrasies, be competitive of surpass state-of-the-art classifiers such as SVMs? (Q2) Can we design an efficient Naive Bayes implementation that allows to testing interesting, but very costly, proposals that

relax the Naive Bayes independence assumption in large ADC tasks? and (Q3) If yes for (Q2), are these Semi-Naive Bayes proposals capable of improving even further the best combinations found in the answer of (Q1)?

1.4 Contributions

The main contributions of the work are:

1. a thorough study on the aforementioned combinations of Naive Bayes Strategies and feature weighting approaches in five widely ADC datasets. More specifically,

    - We review some proposals in the literature that try to overcome the ADC idiosyncrasies, either by proposing some changes in the construction of the Naive Bayes learning model (Rennie et al. [2003]; Zhang and Oles [2000]; Adewole et al. [2014]) or by means of feature weighting strategies more adequate to the ADC task, in a preprocessing phase prior to the construction of the model (Kim et al. [2006]; Matthijssen [2000]).
    - We proposed a methodology that enables a deeper study of the impact of these Naive Bayes strategies (Rennie et al. [2003]; Zhang and Oles [2000]; Adewole et al. [2014]) when combined with the feature weighting approaches (Kim et al. [2006]; Matthijssen [2000]). We applied these combinations into five real textual collections and compared with two ADC algorithms.

2. the proposal and implementation of a parallel version of the Naive Bayes algorithm using graphic processing units (GPUs). This parallel version allowed us to build a more complex Bayesian network, capable of running in large collections.

3. the proposal of new GPU-based lazy Semi-Naive Bayes approaches. Again, more specifically,

    - The introduced GPU-based parallel strategy of the Naive Bayes algorithm on the previous contribution allowed us to alleviate the independence assumption between the attributes, allowing us to build a more complex Bayesian network.
    - We investigate the real impact of the Naive Bayes attribute independence assumption in real text collections and the proposed Semi-Naive Bayes approaches.

1.5 Roadmap

The remainder of this work is organized as follows.

Chapter 2 In this chapter, we briefly describe the supervised ADC task, some evaluation strategies and the GPU parallelism. We also present some of the notation conventions adopted in this work.

Chapter 3 In this chapter, we describe related works. We start by describing some problems found in Text Classification. Then we discuss the strategies proposed in the literature that seek to alleviate these problems. We distinguish three broad areas for doing so: applying feature weighting, modifying the structure of the Naive Bayes and extending the Naive Bayes model by alleviating the independence between attributes. Finally, also discussed the use of parallelism through the graphics processing units in machine learning techniques.

Chapter 4 In this chapter, we describe the Naive Bayes learning paradigms and the several weighting schemes we exploit. We provide an extensive combination of proposed feature weighting strategies with different Naive Bayes models. We also describes our experiments over theses approaches applied in five real textual datasets.

Chapter 5 In this chapter, we describe our GPU-based parallel implementation of the Naive Bayes algorithm.

Chapter 6 In this chapter, we describe and evaluate our four Semi-Naive Bayes approaches based on the several Semi-Naive Bayes we study. We start by introducing the Semi-Naive Bayes which we based and the proposing strategies. We also describe our GPU-based parallel implementation of these Semi-Naive Bayes.

Chapter 7 Finally, in this chapter we conclude the dissertation, summarizing our main findings and proposing some directions for further investigation.

Chapter 2

Basic Concepts and Settings

In this chapter, we briefly describe what is automatic document classification and assessment strategies adopted throughout the work. We also present the parallelization environment (GPU - CUDA), which was adopted to implement the strategies presented in this master thesis.

2.1 Automated Document Classification

ADC methods usually exploit a supervised learning paradigm (Sebastiani [2002]), i.e., a classification model is first "learned" using previously labeled documents (training set), and then used to classify unseen documents (the test set). Let $d_i = (\vec{x}, c_i)$ be a document, where $\vec{x}$ denotes its vectorial (bag of words) representation and c_i is a categorical attribute from a finite set $c_i \in \mathbb{C}$ indicating its class ($\mathbb{C}$ is a finite set composed by all the possible classes). In a probabilistic perspective, the main goal of ADC algorithm is to learn a discrete approximation of the class a posteriori probability distribution $P(c_i|d_i)$ that underlies the relationships between documents and their associated classes. This probability distribution is learned according to a training set composed by already classified documents. There are two approaches for doing so, either based on a direct estimation of $P(c_i|d_i)$, or based on an indirect estimation of $P(c_i|d_i)$.

The first approach is called discriminative classifier and it tries to model dependency on the observed data without making any assumption regarding the probability density function for each class. It makes fewer assumptions on the distributions. On the other hand, a generative classifier tries to learn the model by estimating the conditional probability and the prior probability to estimate the class posteriori probability distribution $P(c_i|d_i)$. In this case, one should assume a model for the class densi-

ties $P(d_i|c_i)$ and its parameters are estimated from the training set. For example, a normal distribution may be chosen, and its mean and variance parameters are estimated according to the already classified data. Then the class a posteriori probability distribution $P(c_i|d_i)$ is estimated according to the Bayes' rule:

$$P(c_i|d_i) = \frac{P(c_i) \cdot P(d_i|c_i)}{\sum_{c' \in C} P(c') \cdot P(d_i|c')}$$

where $P(c)$ is the prior probability and $P(d_i|c_i)$ is the conditional probability.

We assume $c = f(\vec{x})$ for some unknown function f, and the goal of learning is to estimate the function f given a labeled training set, and then to make predictions using $\hat{c} = \hat{f}(\vec{x})$. The quality of such approximation is based on how well $\hat{f}$ predicts the class of unseen documents. Clearly, a function $\hat{f}$ that accurately predicts all training documents $\mathbb{D}$ may not be accurate to predict the classes of unseen documents. In this case, we say $\hat{f}$ is overfitted w.r.t. $\mathbb{D}$. Hence, there exist a trade-off between complexity (the more complex $\hat{f}$ is, more specific patterns observed in the training set are learned) and generalization power of $\hat{f}$ (the more specific patterns observed in unseen documents may not be observed in $\mathbb{D}$).

It has been already proved that, asymptotically, discriminative classifiers are superior to generative ones (Vapnik [1998]), with several reported experiments corroborating this finding (Drummond [2006]). Indeed, if there are not enough training examples, the parametric model is deemed to overfit, decreasing its generalization power (Hastie et al. [2001]). However, some authors claim, based on experimental evaluation, that with realistic training set sizes, the generative classifiers may also perform as well as or better than discriminative ones. This comes true if the assumed parametric model used by the generative classifier is correct. In this case, the class priors become a useful information which is ignored by the discriminative classifiers. In this work we show that our generative classifier was able to outperform the discriminative classifiers (SVM and KNN).

2.1.1 ADC Classifier

We selected two representative and widely used ADC algorithms as baselines in our study. These algorithms are:

KNN: a lazy classifier that assigns to a test document $\vec{d_{test}} = (w_{1,j}, w_{2,j}, ..., w_{V,j})$, the majority class among those of its k nearest neighbor training documents in the vector space (where k is some user specified constant). Choosing an appropriate k value is significant. If the k value is too small it is susceptible to overfitting.

As well as having a k value that is too small it is important to choose a value that isn't too large as it can also lead to misclassification.

KNN determines the decision boundary locally, considering each training document independently. Here, we use cosine similarity to determine the nearest neighbors of a test document. The cosine similarity is measured by Equation 2.1.

$$cos(d_{neighbor_j}, d_{test}) = \frac{\vec{d_{neighbor_j}} \bullet \vec{d_{test}}}{|\vec{d_{neighbor_j}}| \times |\vec{d_{test}}|} \tag{2.1}$$

The similarity of cosine of neighbors that belong to the same class c_i are grouped. The class that express the highest similarity is the chosen one. Thus, the KNN's decision function is expressed by the Equation 2.2.

$$c_{map} = \underset{c_i \in \mathbb{C}}{argmax} \left[\sum_{j=1}^{K} cos(d_{neighbor_j}, d_{test}) \times \lambda_i \right] \tag{2.2}$$

where $\lambda_i = 1$ if $d_{neighbor_j}$ belong to c_i. Otherwise, $\lambda_i = 0$.

Support Vector Machine (SVM): the SVM classifier aims at finding an optimal separating hyperplane between positive and negative training documents, maximizing the distance (margin) to the closest points from both class. Given N training documents represented as pairs (x_i, y_i), where x_i is the weighted feature vector of the i^{th} training document and $y_i \in -1, +1$ the set membership of the document, SVM tries to maximize the margin between them on the training data, which leads to better classification effectiveness on test data. We may state the problem as

$$\underset{\beta,\beta_0}{min} \ \frac{1}{2}||\beta||^2 \text{ subject to } y_i(x_i^T\beta + \beta_0) \geq 1, \tag{2.3}$$

where β is a vector normal to the hyperplane (the so-called weight vector), β_0 is its intercept, and $0 \leq i \leq N$.

After introducing Lagrange multipliers α_i ($0 \leq i \leq N$) for each inequality constraints in Equation 2.3, along with slack variables ξ_i to account for non-separable data (a bounded tolerable training error rate), we form the following Lagrangian (primal):

$$L_P = \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(x_i^T\beta + \beta_0) - (1-\xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i, \tag{2.4}$$

which we minimize with respect to β, β_0 and ξ_i, where μ_i are Lagrange multipliers employed to enforce $\xi_i > 0$. Setting the corresponding derivatives to zero, this yields:

$$\beta = \sum_{i=1}^{N}\alpha_i y_i x_i \tag{2.5}$$

$$0 = \sum_{i=1}^{N}\alpha_i y_i \tag{2.6}$$

$$\alpha_i = C - \mu_i, \tag{2.7}$$

where $\alpha_i \geq 0$, $\mu_i \geq 0$ and $\xi_i \geq 0$, $\forall i$. By substitution into 2.4, we get the so-called Lagrangian Wolfe (dual) function:

$$L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j x_i^T x_j.$$

Furthermore, the solution must satisfy the Karush-Kuhn-Tucker (KKT) conditions, which include, along with Equations 2.5, 2.6 and 2.7, the following ones:

$$\begin{aligned} \alpha_i[y_i(x_i^T\beta + \beta_0) - (1-\xi_i)] &= 0 \\ \mu_i\xi_i &= 0 \\ y_i(x_i^T\beta + \beta_0) - (1-\xi_i) &\geq 0, \end{aligned} \tag{2.8}$$

where $0 \leq i \leq N$.

Finally, the solution for β is $\hat{\beta} = \sum_{i=1}^{N}\hat{\alpha}_i y_i x_i$, with non-zero $\hat{\alpha}_i$ for support points which lie in the support vectors. The solution for β_0 may be devised by Equation 2.8, normally averaging the solutions regarding the support points to achieve numerical stability.

Thus, we can express the SVM's decision function as:

$$\hat{F} = sign(x^T\beta + \beta_0),$$

where the sign of the score is used to predict the example's class. Since SVM is a binary classifier, it should be adapted to handle multiclass classification problems. The two most common strategies for doing so are the one-against-one and the one-against-all (Manning et al. [2008]).

2.2 Evaluation Techniques

An important aspect to be considered is how to evaluate the effectiveness of a classifier (that is, its accuracy in classifying unseen data or, in other words, its generalization power), assessed by first learning a classification model based on the training set and then applying it to classify a set of unseen documents (the test set). Some measures of classification effectiveness are then used to assess the quality of the classification model learned. Several measures for this purpose were proposed in the literature and some of them are widely used by the Machine Learning community. Among the most used measures are precision, recall and F1 measure. In order to describe each of these measures in a binary classification, let's consider the contingency table represented in Table 2.1 (also known as confusion matrix), where TP, TN, FP and FN denote, respectively, the number of true positives, true negatives, false positives and false negatives, defined as:

True Positive (TP): positive test document correctly classified into the positive class.

True Negative (TN): negative test document correctly classified into the negative class.

False Positive (FP): negative test document incorrectly classified into the positive class.

False Negative (FN): positive test document incorrectly classified into the negative class.

The precision p of a performed classification denotes the fraction of all documents assigned to the positive class c_i by the classifier that really belong to c_i. In terms of the contingency table, this translates into

$$p = \frac{TP}{TP + FP}$$

Positive Class = c_i		**Ground Truth**	
		c_i	$\overline{c}_i$
Prediction	c_i	TP	FP
	$\overline{c}_i$	FN	TN

Table 2.1: Contingency Table for Classification Effectiveness Evaluation.

The recall r of a performed classification denotes the fraction of all documents that belong to the positive class c_i that were correctly assigned to c_i by the classifier. Again, in terms of the contingency table, this can be expressed as

$$r = \frac{TP}{TP + FN}$$

Finally, the F_1 measure is defined as the harmonic mean of the precision and the recall, given by

$$F_1 = \frac{2pr}{p + r}$$

There are two conventional methods to evaluate classification algorithms when applied to problems with more than two classes, namely by micro-averaging and macro-averaging the F_1 measure. The micro-averaged F_1 (MicroF_1) is calculated from a global contingency table (similarly to Table 2.1), with the precision and recall being calculated as a sum of each entry of the table:

$$p_{micro} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} TP_i + FP_i}$$

$$r_{micro} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} TP_i + FN_i}$$

In contrast, the macro-averaged F_1 (MacroF_1) is calculated by first calculating the precision and recall values for each class and computing their average value:

$$p_{macro} = \frac{1}{|C|} \sum_{i=1}^{|C|} \frac{TP_i}{TP_i + FP_i}$$

$$r_{macro} = \frac{1}{|C|} \sum_{i=1}^{|C|} \frac{TP_i}{TP_i + FN_i}$$

Notice that the main difference between both strategies is that the MicroF_1 is a document pivoted measure that gives equal weights to the documents while the

MacroF_1 measure is a class pivoted measure that gives equal weights to the classes.

Since the ADC task may be seen as a stochastic process, it is fundamental to adopt some evaluation strategies that guarantee the statistical validity of the obtained classification results, which is achieved by replicating the experiments using different training sets to learn a classification model. For this purpose, the cross validation strategy has become a standard in the machine learning community. There are, at least, two usual strategies for cross validation, the K-fold cross validation and the repeated random sub-sampling (Kohavi [1995]).

The K-fold cross validation consists of randomly splitting the data into K independent folds. At each iteration, one fold is retained as the test set, and the remaining K - 1 folds are used as training set. The repeated random sup-sampling consists of randomly selecting a fraction of documents from the dataset, without replacement, to compose the test set, and the remaining documents retained as the training set. This is performed for each replication. Since in the K-fold cross validation the size of the folds are dependent of the number of iterations, it becomes more suitable to medium/large sized datasets, while the repeated random sub-sampling is usually adopted to small sized datasets when the number of replications is large.

For more details on ADC and evaluation strategies, we refer the reader to Baeza-Yates and Ribeiro-Neto [2011]; Hastie et al. [2001]; Manning et al. [2008].

2.3 GPU and CUDA

General purpose parallel programming on GPUs is a relatively recent phenomenon. GPUs were originally hardware blocks optimized for a small set of graphics operations. As demand arose for more flexibility, GPUs became increasingly more programmable. Early approaches for computing on GPUs cast computations into a graphics framework, allocating buffers (arrays) and writing shaders (kernel functions). Several research projects looked at designing languages to simplify this task. In late 2006, NVIDIA introduced its CUDA architecture and tools to make data parallel computing on a GPU more straightforward. Not surprisingly, the data parallel features of CUDA map pretty well to the data parallelism available on NVIDIA GPUs.

A GPU is connected to a host (CPU) through a high speed IO bus slot, typically PCI-Express in current high performance systems. The GPU has its own device memory, up to several gigabytes in current configurations. Data is usually transferred between the GPU and host memories using programmed DMA, which operates concurrently with both the host and GPU compute units, though there is some support

for direct access to host memory from the GPU under certain restrictions. As a GPU is designed for stream or throughput computing, it does not depend on a deep cache memory hierarchy for memory performance. The device memory supports very high data bandwidth using a wide data path.

In 2007, NVIDIA saw an opportunity to bring GPUs into the mainstream by adding an easy-to-use programming interface, which it dubbed CUDA, or Compute Unified Device Architecture. This opened up the possibility to program GPUs without having to learn complex shader languages, or to think only in terms of graphics primitives (Cook [2013]).

CUDA is an extension to the C language that allows GPU code to be written in regular C. The code is either targeted for the host processor (the CPU) or targeted at the device processor (the GPU). The host processor spawns multithread tasks (or kernels as they are known in CUDA) onto the GPU device. The GPU has its own internal scheduler that will then allocate the kernels to whatever GPU hardware is present. We'll cover scheduling in detail later. Provided there is enough parallelism in the task, as the number of SMs in the GPU grows, so should the speed of the program. CUDA, unlike its predecessors, has now actually started to gain momentum and for the first time it looks like there will be a programming language that will emerge as the one of choice for GPU programming. Given that the number of CUDA-enabled GPUs now number in the millions, there is a huge market out there waiting for CUDA-enabled applications.

In CUDA, a kernel function specifies the code to be executed by all threads during a parallel phase. Since all these threads execute the same code, CUDA programming is an instance of the well-known SPMD - Single Program, Multiple Data (Atallah and Blanton [2010]) parallel programming style, a popular programming style for massively parallel computing systems. Figure 2.1[1] illustrates the concept of data parallelism with a vector addition example. In this example, each element of the sum vector C is generated by adding an element of input vector A to an element of input vector B. For example, C[0] is generated by adding A[0] to B[0], and C[3] is generated by adding A[3] to B[3]. All additions can be perform in parallel.

When a host code calls or launches a kernel, it is executed a large number of threads on a device. All threads that are generated by a kernel launch are collectively called a grid. Each grid is organized into an array of thread blocks. All blocks of a grid are of the same size and can contain up to 1,024 threads. The number of threads in each thread block is specified by the host code when a kernel is launched. Unique coordinates

[1]Image taken from the book Programming Massively Parallel Processors

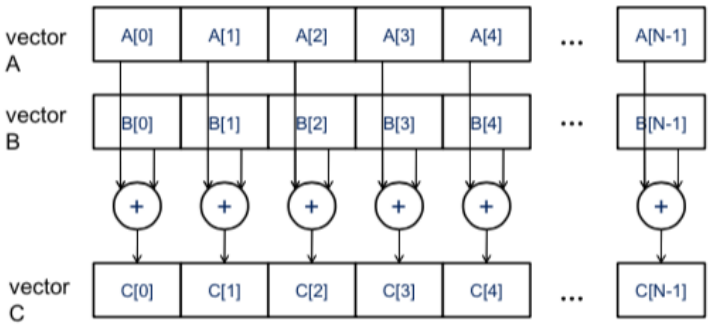

Figure 2.1: Data Parallelism in vector addition.

(*blockIdx* and *threadIdx*) variables allow threads of a grid to identify themselves and their domains of data. This model of programming compels the programmer to organize threads and their data into hierarchical and multidimensional organizations.

The GPU architecture has two levels of parallelism, wherein the first level there are P streaming multiprocessors (SMs) and within each multiprocessor there are p streaming processors (SPs). Thus a parallel program can be first divided into blocks of computation that can run independently on the P SMs (fat cores), without communicating with each other. These blocks, in turn, have to be further divided into smaller tasks (threads) that execute on the SPs (thin cores), but with each thread being able to communicate with other threads in the same block. Each of these threads has access to a larger global memory as well as to a small but fast shared memory and registers.

The GPU supports thousands of light-weight concurrent threads and, unlike the CPU threads, the overhead of creation and switching is negligible. To hide the high latency of the global memory, it is important to have more threads than the number of SPs and to have threads accessing consecutive memory addresses that can be easily coalesced. Another important data movement channel is the PCI-Express connection, whereby CPU and GPU can exchange data between each one's address space but in a much slower speed. The GPU programming model requires that part of the application runs on the CPU while the computationally-intensive part is accelerated by the GPU. Figure 2.2 shows the logical organization of a GPU architecture.

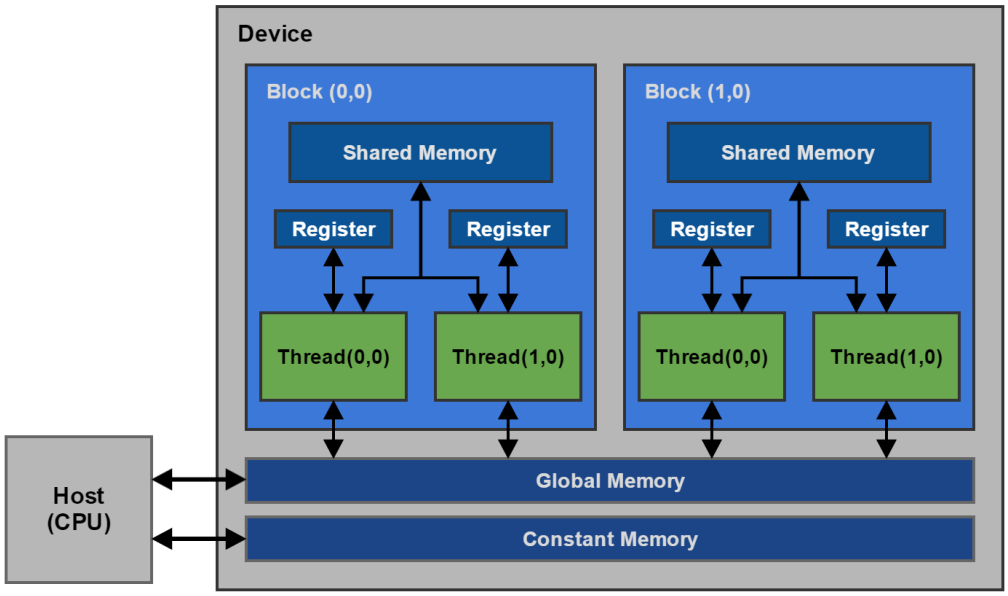

Figure 2.2: Architecture of a simplified GPU.

Chapter 3

Related Work

In this chapter we present the strategies proposed in the literature to deal with the problems discussed in the previous section. First, we present the strategies that perform some adjustments in the Naive Bayes model, making it more robust for text classification. Then, we present the feature weighting approaches that are used to weight the attributes values of each document in the training set. Then, we present some proposals in the literature that extend the Bayesian model, alleviating the assumption of independence between attributes assumed by the Naive Bayes algorithm. These strategies are called Semi-Naive Bayes. Finally, we discuss the use of GPU-based parallel implementations in classification algorithms. These implementations are able to achieve high levels of parallelism and lower power consumption. A strategy that could be exploited in the Semi-Naive Bayes strategies, since they are extremely costly and unfeasible in terms of time execution when applied to real scenarios, such as real textual collections.

3.1 Naive Bayes Limitations

A fundamental assumption assumed by most automatic classifiers is that the data used to learn the classification model are random samples independently and identically distributed (i.i.d.) of a statistical distribution that governs the data. However, this may not be the case. Indeed, in many real scenarios the training data do not follow the same distribution of the test data, compromising the effectiveness of the classification algorithms.

The Naive Bayes algorithm is one of the most widely used techniques, due to its simplicity and efficiency in several scenarios, especially when applied to scenarios in which the attributes are independent, making their “ naive ” assumption more re-

liable. Normally in text classification, its effectiveness is not as good as some other statistical learning methods. This fact is related to some characteristics presented in real document collections, such as unbalanced data, sparsity, among others, that may compromise some of Naive Bayes premises.

In Rennie et al. [2003] and Kim et al. [2006], the authors described some problems faced by the Naive Bayes classifiers, when applied to real textual scenarios. The principal factors are:

- Class imbalance
- Document length
- Feature sparseness

The unbalanced class problem (Witten and Frank [1999]) refers to scenarios where the number of documents of one or few classes far exceeds the number of documents in the other classes. The amount of documents in classes is an important factor, since it is related to the amount of information used to learn the classification model ($P(d_i|c_i)$). So, if the class has sufficient samples, the classifier can learn properly to classify new instances of that class. As for the minor classes, the lack of information inhibits the prediction of new samples, so that the classifier tends to classify them as samples of the majority class.

The document length is another important factor, which may affect in the effectiveness of the classification, especially when this factor is combined with the class imbalance. Short documents are brief and the frequency of words is generally low, thus the relevant words occur once or twice in the document. In longer documents, words are more frequent, so relevant words usually occur more often in the document. Thus, when a collection has long and short documents, the relevant words of the short documents are negligible when compared with the relevant words of longer documents, since these are more frequent. Imagine the case where minority classes have a majority of short documents. In this case, the class has a much lower amount of information due to infrequency of documents and attributes. Thus, the estimate $P(d_i|c_i)$ for a test document for the minority class will be hidden by the estimate of the majority classes, converging the classification of minority class documents to the majority classes.

3.2 Naive Bayes Strategies and Feature Weighting

The Naive Bayes is one of the most popular machine learning methods. Its simplicity, combined with its efficiency, makes it a very attractive method in various classifica-

tion scenarios. However, this simplicity in the construction of the classification models may significantly compromise the effectiveness of NB in some scenarios, such as textual document classification. In Rennie et al. [2003], several disadvantages of the more traditional Naive Bayes Multinomial model (Manning et al. [2008]) are presented in real-world text collections. The authors argue that some characteristics of document collections such as imbalance among classes and data sparsity, significantly compromise the learning model proposed by the Naive Bayes method. Based on this study, the authors proposed a simple heuristic based on data transformations and simple adjustments in the construction of the learning model, which was called Transformed Weight-normalized Complement Naive Bayes (TWCNB). The main idea of this heuristic is to perform a number of feature frequency transformations and adjustments in computing probabilities to improve the Naive Bayes modeling text classification. Following this idea, Zhang and Oles [2000] presented another proposal for construction of models that combine Multinomial Naive Bayes with the technique one-versus-all (Zhang and Oles [2000]), widely used in multiclass classification scenarios. Basically this proposal uses the one-versus-all technique to balance information among classes, smoothing the imbalance.

A more recent study shows that smoothing Naive Bayes may improve the classification performance (Adewole et al. [2014]), since smoothing aims to adjust the probability of an unseen event, which arises due the data sparseness. According to Zhai and Lafferty [2001] smoothing may improve the reliability of Bayesian model, by assigning non-zero probabilities to terms that do not occur in the document. Following this research strategy, the authors of Adewole et al. [2014] applied the concept of linear interpolation smoothing (Jelinek-Mercer smoothing) to Naive Bayes Spam Classification. This combination performed well at improving spam classification, also reducing false positives.

Another research direction proposed to make Naive Bayes algorithm more effective in ADC is the use of different feature weighting strategies. In Kim et al. [2006] the authors proposed a per-document length normalization approach by introducing multivariate Poisson model for Naive Bayes text classification. In addition, the authors proposed a method of weighting that may improve the efficiency of the Naive Bayes classification in rare classes. Although both research lines showed interesting results in a isolated way, by being able to improve the effectiveness of Naive Bayes when compared to their traditional Multinomial version, we are not aware of studies that combine these two lines.

3.3 Semi-Naive Bayes

As mentioned earlier, Naive Bayes is based on the premise that feature occurrences in documents of different classes are independent (Manning et al. [2008]). The main reason for the adoption of this premise is the time complexity, since the construction of a complete and optimal Bayesian network is an NP-hard problem (Cooper [1988]). Thus, a strategy that has being considered (called semi-Naive Bayes) is based on heuristics that relax the assumption of independence without ensuring the optimal model. A number of semi-Naive Bayes methods have been proposed in recent years (Friedman et al. [1997]; Keogh and Pazzani [1999]; Zhang et al. [2005]). They fall into two categories: **data-based** and **structure-based** methods. Those on the first category aim at choosing the training data such that the dependencies within the chosen data are weaker than those in the whole dataset. The local learning methods, such as Lazy Bayesian Rules (Zheng and Webb [2008]), accommodate violations of the independence assumption by choosing a desired set of the training samples on which Naive Bayes is applied. Another group of methods in the data-based category apply Naive Bayes on a subset of attributes. This is achieved by feature selection (Zheng et al. [2004]) which removes irrelevant and redundant attributes.

The structure-based group of Semi-Naive Bayes techniques extend the structure of the Naive Bayes (structure extension), approximating it as much as possible to a Bayesian network, either by some latent variables to connect correlated features, as Hidden Naive Bayes (Zhang et al. [2005]) or by explicitly representing feature dependencies (Keogh and Pazzani [1999]). Interdependencies among features are also allowed, being usually modeled by the z-dependence concept, introduced in Sahami [1996], in which each feature is considered dependent on the class and on other z features, making possible the construction of Bayesian networks. The Naive Bayes establishes a 0-dependence while most existing methods of this group works with a 1-dependence, such as Tree Augmented Naive Bayes, also known as TAN (Friedman et al. [1997]). In the case of this latter work, learning is performed using a conditional mutual information of each pair of features, keeping the pairs that increment quality classification the most. The Super Parent TAN (SP-TAN) is a heuristic that extends the TAN. The heuristic is greedy and optimizes the search space by correlated features (Keogh and Pazzani [1999]). According to experimental results conducted by the authors, the strategy is more effective than traditional Naive Bayes. However, due to its high computational cost, in both time and memory consumption, all these strategies have been evaluated only in small datasets, such as small bioinformatics databases. To the best of our knowledge, the effectiveness of suppressing this premise in large ADC tasks has not

been evaluated yet.

3.4 GPU parallelization

A practical challenge common to all machine learning approaches is how to apply them in real scenarios, where the volume of data is typically huge and the characteristics of the collections vary significantly. Some of the most commonly used techniques to deal with such issues are dimensionality reduction by feature selection (Zheng et al. [2004]), whose goal is to keep only the features that better "define" the documents, and data indexing (Christen [2012]; Cha and Yoon [2002]), whose goal is to represent the data in a more direct and efficient way. Another solution that has been adopted to address these issues is to exploit parallel computation, either by means of distributed memory strategies (Ruocco and Frieder [1997]; Kruengkrai and Jaruskulchai [2002]), or by means of massive parallelism through graphic processors (Kumarihamy and Arundhati [2009]; Grahn et al. [2011]; Lin and Chien [2010]; Garcia et al. [2008]; Andrade et al. [2013]). This last group of strategies has recently demonstrated very interesting results, since they are capable of producing parallelism levels much higher than those achieved by CPUs, associated with a smaller energy consumption (Timm et al. [2010]).

In Kumarihamy and Arundhati [2009], the authors presented an interesting study about how several machine learning techniques may be implemented using GPUs, more specifically the CUDA architecture(Fatica and Luebke [2007]). Regarding ADC, the authors of Grahn et al. [2011] presented a parallel implementation in CUDA for the meta-learning approach Random Forest, with a significant reduction in execution time. Parallel versions of SVM (Lin and Chien [2010]), KNN (Garcia et al. [2008]) and DBSCAN (Andrade et al. [2013]) algorithms are also available. More specifically, in Lin and Chien [2010], the authors introduced several techniques to accelerate SVM in GPUs, including a sparse matrix format to enhance performance, achieving speedups between $55x$ and $134x$. In Garcia et al. [2008], the authors applied a data segmentation method to distance calculations, adapted to the model of threads and hierarchy of memories of the GPU, also achieving interesting results. Finally, in Andrade et al. [2013] the authors adopted a simple data indexing by graphs that allows various parallelization opportunities to be explored in GPU. This GPU version achieves speedups of up to $100x$. However, to the best of our knowledge, there is no GPU-based implementation of the Naive Bayes algorithm reported in the literature. Due to widely spread use of this technique, its massive parallelization is by itself a relevant contribution.

3.5 Chapter Summary

In this chapter we introduced the problems present in classification (especially with the Naive Bayes algorithm) when applied in textual datasets. We also reported some studies that seek to solve these problems through feature weighting techniques or improving the structure of the Naive Bayes model. Furthermore, we discussed studies that extends the Bayesian network, relaxing the assumption of independence between attributes of the Naive Bayes algorithm, this strategies are called semi-Naive Bayes. Since this Semi-Naive Bayes strategies are extremely costly and unfeasible in terms of time execution when applied to real scenarios, such as real textual collections, we discussed the use of GPU for parallel implementation for machine learning algorithms.

Chapter 4

Combining Learning Models and Feature Weighting Strategies

4.1 Naive Bayes Learning Models

As previously mentioned, some premises of the traditional NB model are significantly compromised by some characteristics of real text collections, such as the class imbalance and the term sparseness. In this Section, besides presenting the traditional Naive Bayes, we introduce some of the most important extensions aimed at making it more resilient to the problems presented in Chapter 3. All the models presented in this Section are used in our experimentation.

4.1.1 "Traditional" Naive Bayes

The Naive Bayes (Manning et al. [2008]) calculates the "score" of a class c_i as the probability of a document d_t being assigned to class c_i. Based on this score, a class ranking is created and NB assigns to d_t the class in the top of the ranking. More formally, let $P(c_i|d_t)$ be the probability of a test document $d_t(a_1, a_2, \ldots, a_j)$ to belong to the c_i class, where $(a_1, a_2, \ldots, a_j)$ is a feature vector (binary or weighted) representing the term set of document d_t. This probability is calculated using the Bayes Theorem. However, the calculation of $P(c_i|d_t)$ has a high computational cost, given that the number of possible vectors d_t is very high due to the potential number of term dependencies. To attenuate this problem, the NB algorithm assumes that the occurrence of all terms is independent from each other (the NB independence assumption), a simplification that makes the classification problem computationally viable. Thus, $P(c_i|d_t)$ is defined as:

$$P(c_i|d_t) = \frac{P(c_i)\prod_{\forall j \in d_t} P(a_j|c_i)}{P(d_t)} \tag{4.1}$$

By observing Equation 4.1, we see that the fundamental question in the $P(c_i|d_t)$ definition is how to estimate $P(a_j|c_i)$. Thus, according to Manning et al. [2008], we define:

$$P(a_j|c_i) = \frac{Tc_i(a_j) + \alpha}{\sum_{v \in V} Tc_i(v) + \alpha \cdot |\mathbb{V}|}$$

where $Tc_i(a_j)$ is the number of occurrences of term a_j in class c_i and $\sum_{v \in \mathbb{V}} Tc_i(v)$ is the summation of the occurrences of all terms in documents belonging to c_i. To avoid inconsistencies (e.g., division by zero), a weighted Laplace smoothing is usually applied, consisting of an addition of an α parameter to the numerator and as a multiplication factor to the vocabulary size in the denominator.

In ADC, the goal is to find the "best" class for d_t, usually defined as the one with the highest conditional probability. To avoid computational precision losses (e.g., due to floating point underflow), it is usually more adequate to add the log of the probabilities instead of performing the previously defined multiplication. Thus, this maximization, in most NB implementations is defined as:

$$c_{map} = \underset{c_i \in \mathbb{C}}{argmax} \left[logP(c_i) + \sum_{1 \leq j \leq |\mathbb{V}|} logP(a_j|c_i) \right] \tag{4.2}$$

4.1.2 Interpolated Naive Bayes

The authors Rennie et al. [2003]; Kim et al. [2006] argument that the traditional NB has a natural bias towards the largest classes, due do data imbalance issues, as discussed in Chapter 3. A solution, adopted by Adewole et al. [2014], to overcome this problem is to add the Jelinek-Mercer smoothing method into the classification model. This smoothing method corresponds to a linear interpolation aimed at balancing two estimates using a *lambda* parameter. In our context, the estimates of the most informative terms may be overshadowed by inconsistent estimates of noisy terms, mainly in the minority classes. The interpolation can balance the term estimates given a class

$(P(a_j|c_i))$ along with the estimates of a_j in the collection $(P(a_j))$. Thus, the authors substitute the original formula 4.2 by the formula 4.3.

$$c_{map} = \underset{c_i \in \mathbb{C}}{argmax} \left[logP(c_i) + \sum_{1 \leq j \leq |\mathbb{V}|} \Big(\lambda \cdot logP(a_j) + (1 - \lambda) \cdot logP(a_j|c_i) \Big) \right] \tag{4.3}$$

4.1.3 One versus All Naive Bayes

In Zhang and Oles [2000], the authors proposed another solution to the class imbalance problem. This strategy consists of combining the *One versus All* (OVA) strategy, commonly used in binary classifiers such as SVMs for solving multiclass classification problems, to the traditional NB classifier. The OVA strategy reduces a multiclass problem to a binary one, by considering one class as the positive and the union all others as the negative one. Thus, the OVA-NB uses more information to build the classification model for each class, favoring minority classes.

To accomplish the OVA strategy into NB, the complement of the probability for a given class $P(a_j|\overline{c_i})$, is used. More formally, to calculate the complement of the probability of a term a_j in a given class c_i, we use the information contained in all classes but c_i, as follows:

$$P(a_j|\overline{c}_i) = \frac{T\overline{c}_i(a_j) + \alpha}{\sum_{v \in V} T\overline{c}_i(v) + \alpha \cdot \mathbb{V}}$$

where, $T\overline{c}_i(a_j)$ is the number of occurrences of term a_j in all classes but c_i and $\sum_{v \in \mathbb{V}} T\overline{c}_i(v)$ is the summation of the number of occurrences of all terms in the documents of all classes, but c_i. To avoid inconsistencies in the probability estimates, a Laplace weighting strategy is also adopted. Due to imbalance issues, which may be exacerbated by the union of the "negative" classes, the apriori estimate of the class is not used in this model. The OVA-NB classification model is summarized as:

$$c_{map} = \underset{c_i \in \mathbb{C}}{argmax} \left[\sum_{1 \leq j \leq |\mathbb{V}|} logP(a_j|c_i) - logP(a_j|\overline{c}_i) \right]$$

4.1.4 Complement Naive Bayes

Finally, in Rennie et al. [2003], the authors proposed the Complement Naive Bayes (CNB) learning model, exploited in the TWCNB proposal. Actually, CNB is an extension of OVA-NB in which, instead of calculating the $P(d_t|c_i)$ using information from c_i, the CNB uses only the complement of the probability $P(d_t|c_i)$. This complement balances the information used in the learning model, not favoring the majority classes.

$$c_{map} = \underset{c_i \in \mathbb{C}}{argmax} \left[\sum_{1 \leq j \leq |\mathbb{V}|} -logP(a_j|\bar{c}_i) \right]$$

4.2 Feature Weighting Strategies

In this section, we present some of the main feature weighting strategies used by Bayesian models in order to overcome problems related to the imbalance and sparsity of real textual collections.

4.2.1 Term Frequency

One of the main assumptions made in ADC is that, by means occurrence frequencies of terms, it is possible to determine the topic of a text. A term that occurs more frequently in a given document is more important than an infrequent term. The term frequency (tf) measures the number of occurrences of a term in each single text document (Matthijssen [2000]). Although it is the simplest weighting strategy, tf is commonly used in ADC.

The theory behind this weighting strategy is that terms with high frequency are good to represent the contents of long documents. For short documents, this assumption is more problematic as most terms occur with low frequency. This behavior emphasizes that the occurrence of a rare term in a short document is more significant than the same event in a long document. In collections with documents varying widely in length, the use of the logarithm of tf ($log(tf)$), instead of tf, is usually a better option, since the logarithm smoothies the occurrence frequency of terms. We also analyze this smoothing approach in our evaluation.

4.2.2 Product of the term and inverse document frequency (tf-idf)

After elimination of stopwords, a document still may contain many terms that are poor indication of a class. Common terms tend to occur in numerous documents in a collection and are often randomly distributed over all of them. The importance of a term is related to the number of documents in which it occurred. The more documents a term occurs, the less important it may be. For instance, the term "computer" is not a discriminative term in a document collection computing, no matter what its overall frequency.

Therefore, a commonly adopted premise is that the more rarely a term occurs in a collection, the more discriminative a term is. Thus, the weight of the term should be inversely related to the number of documents in which the term occurs. An inverse document frequency (idf) factor is commonly used to represent this effect. In this case, the logarithm smooth the effect of the inverse document frequency factor.

$$log\left(\frac{N}{n_i}\right)$$

On the other hand, the frequency of the term within the document (tf) should be considered as measure of importance of the term in the collection. This suggests that the relevance of a term in a document can be measured by the product of frequency of occurrence (tf or $log(tf)$) of the term within the document by the inverse document frequency factor of the term. This feature weighting method is known as $tf - idf$:

$$tf_i \cdot log\left(\frac{N}{n_i}\right)$$

4.2.3 Relative Frequency

Relative Frequency is a length normalization method that normalizes the term frequency by dividing it by a normalization factor given by the maximum frequency of a term occurring in any text document of the collection, as shown in Equation 4.4.

$$RF = \frac{tf_i}{NF} \tag{4.4}$$

In real text collections, the amount of infrequent terms is very high. In Kim et al. [2006] the authors considered and evaluated this characteristic as a normalization factor (NF), based on the unique terms in a collection. The authors designed a linearly interpolated normalization factor (NF_u) using the average document length in the

collection, i.e., $NF_u = \alpha \cdot avdu + (1-\alpha) \cdot du_j$, where $avdu$ indicates the average number of unique terms in a document over the collection and du_j means the number of tokens and unique terms in a document d_j. This linear interpolation smoothies the length of each document using the characteristics of document collection.

$$RF_u = \frac{tf_i}{NF_u} \tag{4.5}$$

The Relative Frequency method based on unique terms (Equation 4.5) significantly has increased the effectiveness of Naive Bayes algorithm (Kim et al. [2006]), proving to be more effective than BM25 (Robertson et al. [1999]) and PLN (Singhal et al. [1999]) methods.

4.3 Experimental Evaluation

4.3.1 Datasets

In order to evaluate the selected combinations, we consider five real-world textual datasets, namely, 20 Newsgroups; Four Universities; Reuters; ACM Digital Library and RCV1 datasets. We performed a traditional preprocessing task for all datasets: we removed stopwords, using the standard SMART list, and applied a simple feature selection by removing terms with low "document frequency (DF)"[1]. Next, we give a brief description of each dataset.

4 Universities (4UNI), a.k.a, WebKB this dataset contains Web pages collected from Computer Science departments of four universities by the Carnegie Mellon University (CMU) text learning group. There is a total of 8,277 web pages, classified into 7 categories (such as student, faculty, course and project web pages).

Reuters (REUT90) this is a classical text dataset, composed by news articles collected and annotated by Carnegie Group, Inc. and Reuters, Ltd. We consider here a set of 13,327 articles, classified into 90 categories.

ACM-DL (ACM) a subset of the ACM Digital Library with 24,897 documents containing articles related to Computer Science. We considered only the first level of the taxonomy adopted by ACM, where each document is assigned to one of 11 classes.

20 Newsgroups (20NG) this dataset containing 18,805 newsgroup documents, partitioned almost evenly across 20 different newsgroups categories. 20NG has become a popular dataset for experiments in text applications of machine learning techniques, such as text classification and text clustering.

[1] We removed all terms with DF ≤ 2

RCV1 Uni The Reuters Corpus Volume 1 (RCV1) is a dataset with 804,427 English language news stories. We considered the complete topics taxonomy comprised of 103 classes. However, the original RCV1 is a multilabel dataset with the multilabel cases needing special treatment, such as score threshold, etc. (see Lewis et al. [2004] for details). As our current focus is on unilabel tasks, to allow a fair comparison among the other datasets (which are also unilabel) and all baselines (which also focus on unilabel tasks), we decided to transform all multilabel cases into unilabel ones. In order to do this fairly, we randomly selecting, for each documents with more than one label, a single label to be attached to that document. This procedure was applied in about 20% of the documents of RCV1 which happened to be multilabel.

Collections	Distribution of the Classes					Number of	Number of
	Classes	Minor Class	Median	Mean	Major Class	Attributes	Documents
4UNI	7	137	930	1.182	3.759	40.195	8.277
REUT90	90	2	29	148,1	3.964	19.590	13.327
20NG	20	628	979	940,2	999	61.050	18.805
ACM	11	63	2.041	2.263,36	6.562	56.449	24.897
RCV1 Uni	103	91.399	165.850	201.107	381.328	155.796	804.427

Table 4.1: General information of the data collections.

4.3.2 Evaluation, Algorithms and Procedures

The Naive Bayes models were compared using micro averaged F_1 (MicF_1) and macro averaged F_1 (MacF_1), standard information retrieval measures (Lewis et al. [1996]; Yang [1999]; Yang and Pedersen [1997]). While the MicF_1 measures the classification effectiveness over all decisions (i.e., the pooled contingency tables of all classes), the MacF_1 measures the classification effectiveness for each individual class and averages them. All experiments were executed using a $5-$fold cross-validation (Breiman and Spector [1992]) procedure. The parameters were set via cross-validation on the training set[2], and the effectiveness of the algorithms, running with distinct types of features, were measured in the test partition. To compare the average results on our cross-validation experiments, we assess the statistical significance of our results by means of a paired t-test with 95% confidence. This test assures that the best results, marked in bold, are statistically superior to others..

[2]In other words, a portion of the training set was separated, as a type of validation set, to be used for parameterization.

In order to evaluate the performance of different Naive Bayes model combined with different feature weighting approaches, we compare then with SVM and KNN classifiers. Regarding SVM classifier, we adopt the LIBLINEAR (Fan et al. [2008]) implementation, using a linear kernel on which the regularization parameter was chosen among eleven values from 2^{-5} to 2^{15} by using $5-$fold cross-validation on each training dataset. In the KNN classifier, the size of neighborhood was set via cross-validation in the training set. We also tested the four described feature weighting schemes with KNN, SVM. The best overall results for SVM and KNN were obtained using the $log(TF) - IDF$ weighting scheme, therefore these versions will be used as baselines.

All experiments were run on a Quad-Core Intel Xeon E5620, running at 2.4GHz, with 16Gb RAM. The GPU experiments were run on a GeForce GTX TITAN Black, with 6Gb RAM. The machine was reserved to execute each of the experiments individually, to guarantee accuracy of the efficiency results for each implementation.

We would like to point out that some of the results may differ from the ones reported in other works for the same datasets (e.g., Kim et al. [2006]; Rennie et al. [2003]; Godbole and Sarawagi [2004]). Such discrepancies may be due to several factors such as differences in dataset preparation[3]; the use of different splits of the datasets (e.g., some datasets have "default splits" such as REUT90 and RCV1[4]), the application of some score threshold, such as SCUT, PCUT, etc., which, besides being an important step for multilabel problems, also affects classification performance by minimizing class imbalance effects, among other factors. We would like to stress that we ran all alternatives under the same conditions in all datasets, using standardized and well accepted cross-validation procedures that optimize parameters for each alternatives, and applying the proper statistical tools for the analysis of the results. All our datasets are available for others to replicate our results and test different configurations.

4.3.3 Results

Table 4.2 shows the results of all selected combinations. For each dataset, we tested the four described feature weighting strategies along with the four discussed NB variants. The colors indicate the results for the methods as originally proposed: blue corresponds to INB which used the TF weighting scheme, green corresponds to the original proposal of the CBN using log(TF)-IDF and finally, red corresponding to OVA-NB which originally exploited the TF scheme.

[3]For instance, some works do exploit complex feature weighting schemes or feature selection mechanisms that do favour some algorithms in detriment to others.

[4]Indeed, we do believe that running experiments only in the default splits is not the best experimental procedure as it does not allow a proper statistical treatment of the results.

Overall, we can observe in the Table that the combination of the INB learning model with the $RF - u$ weighting strategy produced the best results in all datasets. We can also observe that the use of the $RF - u$ strategy produced some expressive gains over the other weighting strategies in several cases (e.g., Mac$F1$ for $OVA - NB$ in RCV1 and Mac$F1$ for NB in Reuters). It can also be observed that the use of this same weighting strategy resulted in some gains in the best learning model INB), such as the statistically significant gains in Mac$F1$ and Mic$F1$ in $4UNI$ over the other strategies.

Table 4.3 presents a comparison among SVM, KNN and the best NB approach for each collection. We can see that most of the results of Best-NB (4 out of 5) uses the weighting strategy $RF-u$, three with the INB and one of the OVA-NB learning models. Only for the ACM dataset the best results were achieved with $log(TF) - IDF + INB$. But even in these cases, the combination $RF - u + INB$ was very close to the best results. In fact, there is a statistical tie in ACM. Notice that the best combination for a particular dataset can be determined in the validation set. This is very feasible, especially due to the NB efficiency.

We can see in Table 4.3 that the Best-NB combination outperforms or ties with KNN in eight out of ten cases (six wins and two ties). When compared to SVM, the Best-NB ties or outperforms this classifier also in eight out of ten cases (seven wins and one tie), loosing in other two (regarding Mic$F1$). In fact, if we look at the Mac$F1$ results, Best-NB outperformed or tied SVM and KNN in all five datasets. This is consistent with the fact that most of the learning models and weighting scheme adaptations were designed to deal with the class imbalance issue, which tends to favor Mac$F1$.

Thus, we conclude that, if good performance across all classes, mainly the minority ones, is the main goal for a particular application, our results demonstrate that there is probably a combination of a NB learning model and a weighting strategy (most probably $RF - u + INB$) that may deliver a top-notch performance.

4.4 Chapter Summary

In this chapter, we exploit new combinations of Naive Bayes Strategies selected in the literature that try to overcome the ADC idiosyncrasies with feature weighting strategies

Collections	Feature Weighting	NB		INB		CNB		OVA - NB	
		Mac$F1$	Mic$F1$	Mac$F1$	Mic$F1$	Mac$F1$	Mic$F1$	Mac$F1$	Mic$F1$
4UNI	TF	52.57 ± 2.73	61.64 ± 1.85	54.41 ± 2.46	61.11 ± 1.82	51.1 ± 2.47	62.81 ± 1.49	54.85 ± 1.4	60.93 ± 1.83
	TF-IDF	53.71 ± 1.6	60.3 ± 1.61	51.59 ± 0.71	61.1 ± 0.75	38.39 ± 1.55	42.36 ± 1.52	53.19 ± 1.89	60.57 ± 1.71
	log(TF)-IDF	51.21 ± 1.72	57.41 ± 0.95	51.13 ± 1.25	61.13 ± 0.97	38.3 ± 1.73	43.87 ± 0.86	50.89 ± 1.6	56.68 ± 0.95
	$RF - u$	55.84 ± 1.51	$\mathbf{63.14 \pm 1.37}$	$\mathbf{57.57 \pm 1.54}$	$\mathbf{64.07 \pm 1.11}$	51.11 ± 1.4	$\mathbf{64.04 \pm 1.7}$	$\mathbf{57.25 \pm 1.21}$	$\mathbf{64.11 \pm 1.65}$
REUT90	TF	16.17 ± 1.06	65.39 ± 1.09	34.64 ± 1.61	66.74 ± 0.66	27.06 ± 1.68	$\mathbf{65.23 \pm 1.08}$	28.58 ± 1.7	$\mathbf{66.15 \pm 0.63}$
	TF-IDF	23.62 ± 0.64	$\mathbf{65.58 \pm 0.81}$	$\mathbf{35.29 \pm 1.69}$	$\mathbf{66.12 \pm 0.51}$	27.3 ± 1.81	63.55 ± 1.11	30.57 ± 2.57	64.56 ± 0.57
	log(TF)-IDF	19.23 ± 1.44	$\mathbf{65.61 \pm 0.88}$	34.3 ± 1.68	66.36 ± 0.57	26.1 ± 1.66	64.01 ± 1.3	29.61 ± 1.25	64.91 ± 0.67
	$RF - u$	32.64 ± 2.28	$\mathbf{66.03 \pm 0.82}$	$\mathbf{36.02 \pm 2}$	$\mathbf{66.17 \pm 0.76}$	27.25 ± 2	$\mathbf{67.28 \pm 1.13}$	32.66 ± 2.28	$\mathbf{66.47 \pm 0.77}$
20NG	TF	85.31 ± 0.81	$86.0.4 \pm 0.82$	89.25 ± 0.59	$\mathbf{89.67 \pm 0.56}$	87.74 ± 0.85	88.39 ± 0.8	89.2 ± 0.77	89.54 ± 0.75
	TF-IDF	87.04 ± 0.62	87.47 ± 0.61	$\mathbf{89.05 \pm 0.47}$	$\mathbf{89.45 \pm 0.5}$	87.32 ± 0.79	87.9 ± 0.75	$\mathbf{88.95 \pm 0.62}$	$\mathbf{89.33 \pm 0.63}$
	log(TF)-IDF	87.14 ± 0.71	87.56 ± 0.66	$\mathbf{89.23 \pm 0.72}$	$\mathbf{89.65 \pm 0.67}$	87.76 ± 1.02	88.27 ± 0.94	88.78 ± 0.61	89.16 ± 0.63
	$RF - u$	86.6 ± 0.65	87.05 ± 0.629	$\mathbf{89.64 \pm 0.6}$	$\mathbf{90 \pm 0.54}$	88.15 ± 0.78	88.76 ± 0.76	$\mathbf{89.74 \pm 0.75}$	$\mathbf{90.07 \pm 0.74}$
ACM	TF	56.43 ± 0.4	73.56 ± 0.25	$\mathbf{62.02 \pm 1.05}$	$\mathbf{73.64 \pm 0.49}$	$\mathbf{60 \pm 1.85}$	73.28 ± 0.73	61.04 ± 1.5	73.26 ± 0.51
	TF-IDF	59.57 ± 0.83	73.87 ± 0.4	$\mathbf{61.01 \pm 1.66}$	74.31 ± 0.58	57.3 ± 0.87	70.16 ± 0.74	$\mathbf{61.01 \pm 1.42}$	73.35 ± 0.45
	log(TF)-IDF	59.3 ± 0.36	74.28 ± 0.41	$\mathbf{62.8 \pm 1.63}$	73.52 ± 0.56	58.84 ± 1.25	71.72 ± 0.74	60.32 ± 0.91	74.12 ± 0.48
	$RF - u$	57.49 ± 0.53	72.89 ± 0.25	$\mathbf{61.57 \pm 1.6}$	$\mathbf{74.61 \pm 0.5}$	$\mathbf{60.02 \pm 1.96}$	73.88 ± 0.59	$\mathbf{60.76 \pm 1.06}$	$\mathbf{75.14 \pm 0.4}$
RCV1 Uni	TF	45.48 ± 0.39	$\mathbf{71.53 \pm 0.12}$	50.40 ± 0.46	69.35 ± 0.07	33.4 ± 0.27	64.69 ± 0.2	44.5 ± 0.59	70.87 ± 0.14
	TF-IDF	49.59 ± 0.49	69.53 ± 0.09	50.38 ± 0.64	68.15 ± 0.09	20.93 ± 0.13	52.38 ± 0.17	31.62 ± 0.49	60.57 ± 0.16
	log(TF)-IDF	47.28 ± 0.46	69.79 ± 0.08	49.93 ± 0.64	68.03 ± 0.07	20.28 ± 0.13	52.07 ± 0.21	30.20 ± 0.38	60.56 ± 0.13
	$RF - u$	51.65 ± 0.89	69.28 ± 0.07	$\mathbf{51.68 \pm 0.85}$	69.28 ± 0.08	19.78 ± 0.26	47.51 ± 0.17	$\mathbf{51.18 \pm 0.72}$	70.09 ± 0.12

Table 4.2: Feature Weighting approaches applied on Naive Bayes Models.

Collections	SVM		KNN		Best-NB		Best-NB (Source)
	Mac$F1$	Mic$F1$	Mac$F1$	Mic$F1$	Mac$F1$	Mic$F1$	
4UNI	$\mathbf{56.02 \pm 2.02}$	70.91 ± 1.9	$\mathbf{58.24 \pm 2.92}$	$\mathbf{73.78 \pm 0.7}$	$\mathbf{57.57 \pm 1.54}$	64.07 ± 1.11	$RF - u + INB$
REUT90	30.38 ± 2.04	65.09 ± 0.21	29.95 ± 1.14	$\mathbf{68.04 \pm 0.62}$	$\mathbf{36.02 \pm 2}$	$\mathbf{66.17 \pm 0.76}$	$RF - u + INB$
20NG	84.92 ± 0.54	85.1 ± 0.6	88.41 ± 0.27	88.69 ± 0.67	$\mathbf{89.74 \pm 0.75}$	$\mathbf{90.07 \pm 0.74}$	$RF - u + OVA - NB$
ACM	58.00 ± 1.33	69.67 ± 0.45	59.75 ± 1.72	71.6 ± 0.46	$\mathbf{62.8 \pm 1.63}$	$\mathbf{73.52 \pm 0.56}$	$log(TF) - IDF + INB$
RCV1 Uni	47.05 ± 0.35	$\mathbf{75.85 \pm 0.08}$	46.32 ± 0.72	68.23 ± 0.16	$\mathbf{51.68 \pm 0.85}$	69.28 ± 0.08	$RF - u + INB$

Table 4.3: Best results of Naive Bayes compared with SVM classifier.

specially developed to ADC task. By means of our evaluation we are able to answer our first research question, a proper combination of Naive Bayes with feature weighting Strategies can outperform in most cases the strongest textual classifier known in the datasets we experiment with: Support Vector Machines. In fact, Best-Naive Bayes outperformed or tied SVM in all five datasets considering the MacF1 results. This is consistent with the fact that that most of the learning models and weighting strategies adaptations were designed to deal with the class imbalance issue, which tends to favor MacF1. Moreover, our results demonstrate that a combination of a INB learning model with the feature weighting RF_u can deliver top-notch performance.

Chapter 5

Parallelization of Naive Bayes Classifier Using GPU

Over the last years, the computer industry has shifted from pushing clock frequency to parallelism. Rather than increasing the speed of its individual processor cores, traditional CPUs have increased the number of cores (multicore processors) as the primary basis for increasing system performance. This trend has also been followed by many core GPU chips which have now thousands of processing cores. The general perception is that processors are not getting faster, but instead are getting wider, and the only way to improve performance is through the exploitation of parallelism.

CPUs are optimized for low-latency access and excel in running single-threaded processes whereas GPUs are optimized for high throughput and can deal with massive multithreaded parallelism. GPUs can handle massive amounts of data, which is critical for Information Retrieval (IR), however its different architecture and memory hierarchy requires the design of novel algorithms and new implementation approaches. This situation has hampered the exploitation of this opportunity by the IR community. However, recent works on Database Scalability, Document Clustering, Learning to Rank, Big Data Analytics and Interactive Visualization (Chang et al. [2012]; Teitler et al. [2014]; Shchekalev [2014]; Netzer [2014]; Graham and Mostak [2014]) have produced some encouraging results.

The probabilistic Naive Bayes (NB) algorithm presents many opportunities for a highly parallel GPU implementation. Since the terms of a document can be processed independently, both the model generation and the classification task phases can take advantage of the high computational power of GPUs. The combination between NB models and feature weighting strategies opens up new opportunities for exploiting parallelism in a massive multithreaded way. In this chapter we present our proposal

of a new version of Naive Bayes parallelized using Graphics Processors Units (GPU), which corresponds to another contribution of this master thesis. Moreover, as we will discuss later, this parallelized version was adapted to evaluate the actual impact of NB attribute independence assumption in real text collections.

5.1 Indexing Data

Given that document collections are usually sparse, with terms occurring only in a few documents, we chose to represent the documents by a compact data structure, based on inverted index. For this, we used four vectors: $docIndexVector$ representing the documents; $docAttributeVector$ that stores the attributes of each document; $docValueVector$ that stores the weights of the attributes for each document; and finally, $docClass$ that stores the classes of all documents. At $docIndexVector$ the index represents the document, and in each position of this vector we store the position where the list of terms of the document begins in vector $docAttributeVector$ and the frequency in the vector $docValueVector$. At $docClass$ each index also represents a document, and each position stores the respective document class.

Figure 5.1 shows an example of the data structure used. As can be seen, the document 0 of the database, starts its list of terms at the index 0 of the vector $docAttributeVector$ and has two terms (terms 0 and 1), each of which has frequency equal to 1 as can be seen in vector $docValueVector$. In turn, the document 4 begins its list of terms in the position 8 of the vectors $docAttributeVector$ and $docValueVector$ and has only the term 1, with frequency also equal to 3. The document classes are in the vector $docClass$, where the documents 0 and 4 have classes 0 and 1 respectively.

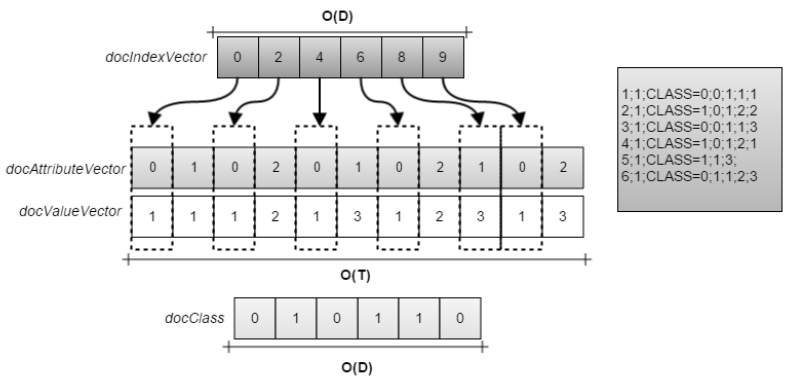

Figure 5.1: Data structure to represent the documents.

5.2 GPU-NB

Our parallelization strategy for the Naive Bayes is based on a kernel implemented using the C language in CUDA. Kernel is the name given to the functions that run on GPU. Thus, the kernel exploits the maximum parallelism possible, minimizing the dependence of data among threads. It also focuses on maximizing the amount of computation performed by each thread. Since the Naive Bayes algorithm assumes independence among the attributes and that the classification can be done independently for each test document, the problem can take full advantage of the GPU parallelism. We describe below the parallelization strategy used to implement the kernel on GPU.

The proposed kernel exploits two levels of parallelism, the first associates a block of threads to a test document and, at the second level, each of these threads is associated with a term of the same document. Thus, each block is responsible for classifying an specific test document while each thread of the block is responsible for calculating the probability of an attribute for each class and thus cooperating to produce the NB model. If the number of attributes is greater than the number of threads of a block, the document is split into k parts, where k is the rounded up ratio between the number of attributes of the document and the number of threads in the block. Then, each k part is calculated one at a time, in a round robin fashion, and the results accumulated. As for the calculation of the logarithmic summation of the classification model, we implemented a binary tree parallel reduction using shared memory. Therefore, we allocated an array in shared memory whose size is equal to the number of threads in the block. Then, each thread stores the conditional probability $P(a_j|c_i)$ in its respective position in the shared memory vector. The vector is divided into two sub vectors and then these sub-vectors are added. The process is repeated until the first position of the vector contains the total sum of the logarithms.

All information required for the probability calculations, such as frequency of attributes on classes, occurrence of classes and occurrence of attributes were pre computed in CPU, while reading the training documents, and later transferred to the GPU memory. This was much simpler, and more efficient, than storing the training documents in a data structure, transferring to the GPU, and then computing this information in the GPU. The test documents are stored in the data structure and transferred to the GPU.

5.3 Efficiency of our proposed algorithm

Figure 5.2 illustrates the percentage of time spent in each of the main stages of the Naive Bayes algorithm. The Reading Training step allocates and initializes variables, besides performing the reading of the training documents. During the reading of the documents, the frequencies are computed to calculate conditional probabilities used to build the Naive Bayes model. The next step, to calculate the table of conditional probabilities of the attributes for each class. In the Reading Test step, the information of test documents are stored in the data structure shown in Figure 5.1. Finally, in the Classifying step, the Naive Bayes model (sum of terms probabilities) is applied for each test examples and then the quality of the prediction is measured by the evaluation techniques.

As can be seen in Figure 5.2 the data communication time is more demanding than the computation time. Although our GPU-based parallel implementation of Naive Bayes has been developed bearing in mind the GPU idiosyncrasies, paying special attention to its memory hierarchy, multiprocessor occupancy and coalesced memory access, the time spent transferring the datasets to the GPU global memory heavily hurt performance, resulting in speedups not greater than a few units. These data are transferred in the Reading Test step. Thus, we can observe that the GPU-based parallel implementation spend more time than the CPU version.

However, the relaxation of the Naive Bayes feature independence assumption, that is the Semi-Naive approach, was able to take full advantage of the proposed parallel implementation since these approaches require intensive computation once the dataset has been transferred to the GPU. In addition, the use of our compact data structure allowed us to drastically reduce memory consumption, allowing us to work with large collections of documents, that would otherwise take too much time or require the use of an expensive computational platform. We focus on these Semi-Naive Bayes proposals next, putting particular emphasis in our original contributions aimed at improving their efficiency and scalability.

5.4 Chapter Summary

In this chapter, we proposed GPU-NB, a parallel implementation of the Naive Bayes algorithm. In our proposal, we exploit a very compact data structure to index the documents aiming at minimizing memory consumption, as well as maximizing the opportunities for massive parallelization of the algorithm in GPUs. In our GPU-based implementation we exploit the maximum parallelism possible, minimizing the depen-

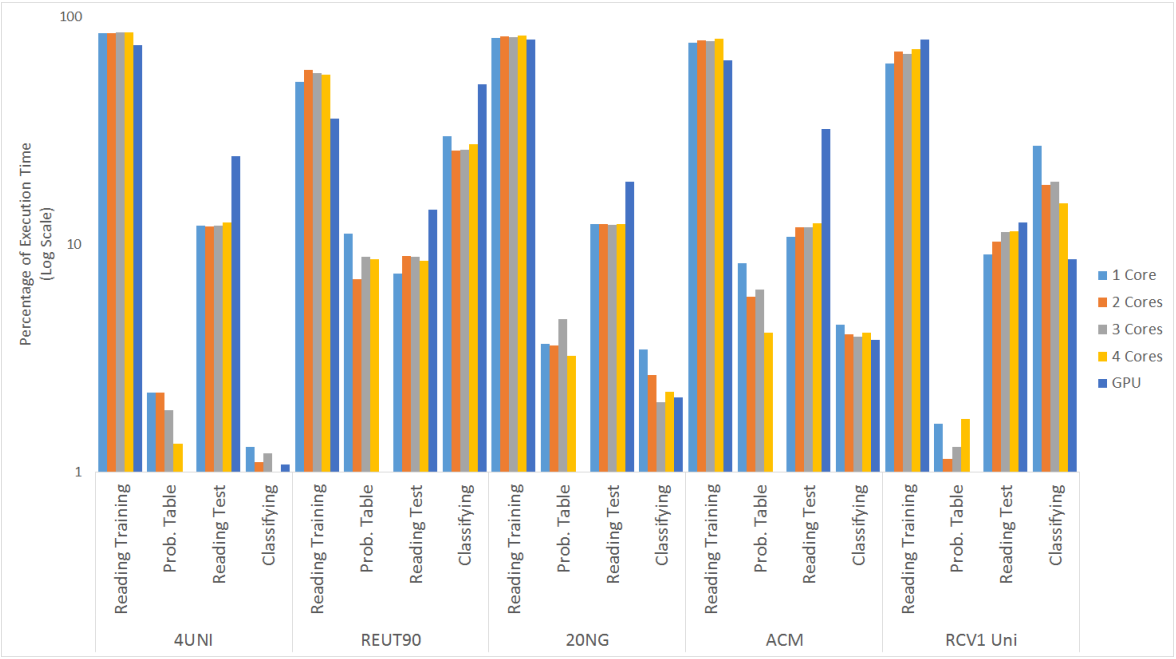

Figure 5.2: Efficiency of the steps of Naive Bayes algorithm implementation on CPU and GPU.

dence of data among threads. It also focuses on maximizing the amount of computation performed by each thread. However, we observed in our results that the communication time of the Naive Bayes algorithm data is more representative than the computation time. Although our GPU-based parallel implementation of Naive Bayes has been developed bearing in mind the GPU idiosyncrasies, the time spent transferring the datasets to the GPU global memory heavily hurt performance, resulting in speedups not greater than a few units. However, the relaxation of the Naive Bayes feature independence assumption, that is the Semi-Naive approach, was able to take full advantage of the proposed parallel implementation since these approaches require intensive computation once the dataset has been transferred to the GPU, as we shall see later.

Chapter 6

Semi-Naive Bayes Methods

Over the years various techniques have been proposed to extend the classification model generated by Naive Bayes. In Friedman et al. [1997], the authors compared the Naive Bayes algorithm with Bayesian networks, which are more robust models to represent probability dependencies, since they have no restrictions. Surprisingly in some scenarios, Bayesian networks degraded the quality of classification. Thus, the authors proposed a restricted Bayesian model that combines the ability of Bayesian networks to represent dependencies among attributes and the simplicity of Naive Bayes. This representation, called Tree Augmented Naive Bayes (TAN), is defined by the main constraint: attributes may depend on only one other attribute in addition to the class (Friedman et al. [1997]). This restriction makes feasible the probabilistic calculations, since it is computationally very costly to compute multiple dependencies for any given attribute[1].

6.1 Tree Augmented Naive Bayes

A Tree Augmented Naive Bayes model (TAN) is a technique that extends the structure of the Naive Bayes by alleviating the independence assumption among attributes, approximating it as much as possible to a Bayesian network.

Figure 6.1 shows the TAN model for the Pima dataset, to classify if a patient tests positive or negative for diabetes. The variables in this dataset are all derived from patients who were women older than 21 years of age. In this dataset, the variables Glucose and Insulin are related in determining the class variable. For example, very low glucose level, seen independently is surprising. But given the fact that the insulin level

[1]We will use the terms *attribute* and *feature* interchangeably, from now on.

is also high, then it becomes unsurprising (Friedman et al. [1997]). This correlation can be captured in the TAN model. But, in case of Naive Bayes model, they are considered as two separate abnormal events. Thus, Naive Bayes would over penalize the class label (Friedman et al. [1997]).

We observe in Figure 6.1 that all the attributes, except the attribute Pregnant, have two parents, the class and another attribute. The interactions among attributes are shown using solid edges and the interaction between the class and an attribute is shown using dotted edges. The attribute Pregnant is the root node of the tree and it has only one parent, the class. This is an example of a tree structure, constructed among attributes of the system, in a TAN model.

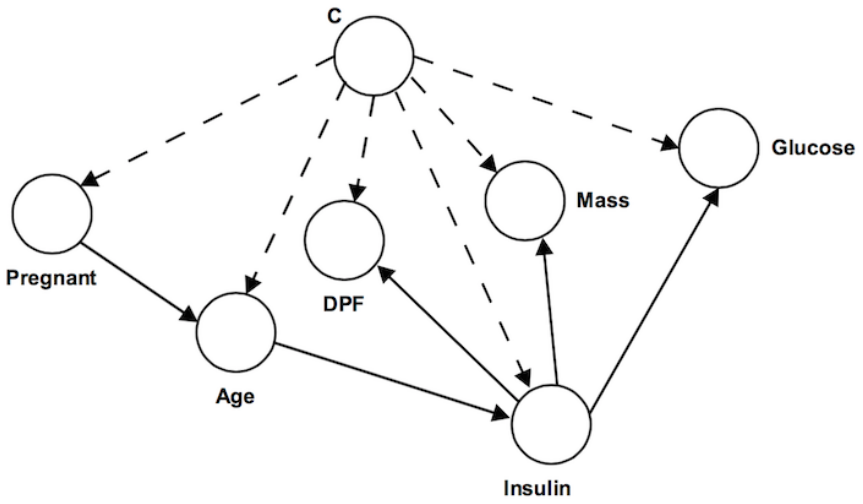

Figure 6.1: TAN model learned for the dataset "Pima".

The TAN heuristic computes the conditional mutual information of all pairs of attributes and constructs a weighted undirected graph, where the nodes of the graph are the attributes and the edges represent the mutual information of the pair of vertexes. Then, a maximum cost spanning tree is built. The spanning tree is transformed into a directed tree, indicating the dependency among nodes. Finally, the class is added as the parent of all attributes. The resulting classification model can be represented by maximizing the Equation 6.1, where $\pi(a_j)$ is the parent attribute of the attribute a_j.

$$c_{map} = \underset{c_i}{argmax} \left(P(c_i) \prod_{j=1}^{n} P(a_j | c_i, \pi(a_j)) \right) \tag{6.1}$$

Keogh and Pazzani [1999] proposed a new strategy to build a Tree Augmented Naive Bayes using the greedy method Hill Climbing Search (HSC). The technique uses a unique representation to improve the quality of classification, rather than trying to directly approximate the probability distribution of the attributes. Thus, the TAN search heuristic using HSC initially builds the Naive Bayes model and initializes a set O of orphans, inserting in O all the attributes of the vocabulary. Then, the technique evaluates each attribute relationship a_i to a_j ($i \neq j$, $a_j \in O$). If the relation among these attributes increases the classification quality, this relationship (dependency) is maintained in classification model and the attribute a_j is removed from the set of orphans. The heuristic returns searching for new dependencies between attributes until there are no more attributes in O or no more dependencies between attributes to be evaluated. Figure 6.2 illustrates the construction of TAN model. The green node represents the orphan attributes while the red nodes represents the attributes that have another parent beyond the class.

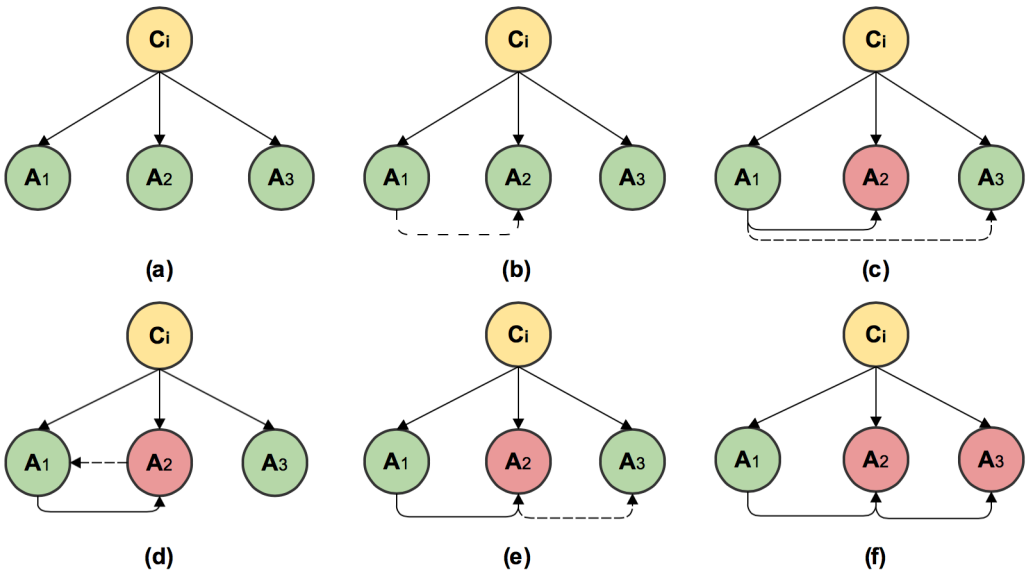

Figure 6.2: Example of the construction of the TAN.

6.2 Super Parent TAN

The Super Parent TAN (SP-TAN), a variant of TAN, uses a different approach to build dependencies among attributes (Keogh and Pazzani [1999]). It uses the same representation as TAN, but it uses a greedy search method (Hill Climbing Search) to

add dependencies to the Naive Bayes model. The construction of the SP-TAN network is given by the concepts of Super Parent and Favorite Child.

- Super Parent (SP): Given a Bayesian network, if we create a dependency to an attribute x for all the orphans attributes, it is called Super Parent. Orphans attributes are the attributes that have only the class as a parent.

- Favorite Child (FC): If we extend a dependency of a Super Parent for each orphan and evaluate the quality of each dependence individually, the attribute that improves accuracy the most is the Favorite Child.

Thus, the SPTAN search heuristic initially builds the Naive Bayes model and initializes a set O of orphans, inserting in O all the attributes of the vocabulary. Then, the technique evaluates each attribute as Super Parent (A_{SP}) and in the end, the attribute that most increases the classification quality is selected as A_{SP} . The next step is to find the Favorite Child of the Super Parent attribute. Each attribute is evaluated individually given the A_{SP} ($P(A_i|C, A_{SP})$), and the Favorite Child is subsequently selected as the attribute A_i that most increases the classification quality given A_{SP}. The process is repeated until there are no more orphans or increases in the classification quality. The Figure 6.3 illustrates the construction of SP-TAN model. The graph *(a)* represents the Naive Bayes model, the graphs in *(b)* and *(d)* the selection of a Super Parent attribute and the graphs in *(c)* and *(e)* the choice of a Favorite Child.

The choice of Super Parents and Favorite Children attributes are conducted using a validation set, since the heuristic uses a prediction metric for evaluating the classification quality of the networks built by each candidate as Super Parent. Thus, all dependencies are selected by using this validation set and subsequently evaluated on the test set. However, some of these dependencies among attributes may be specific to the validation set. In ADC scenarios, dependencies between words may vary according to the document set. So when the classification model is applied on the test set, the model fails to capture certain dependencies occurring in that particular data. Furthermore, the model may add dependencies that do not exist in the test set, causing inconsistencies in the classification of the documents. In other words, the model does not *generalize* to unseen documents.

SP-TAN produced some gains compared with the traditional Naive Bayes (Keogh and Pazzani [1999]) in small datasets. However, when applied to real-world scenarios, such as large ADC datasets, this technique tends to loose performance due to overfitting issues caused by the strong dependency on the validation set. Moreover, the two

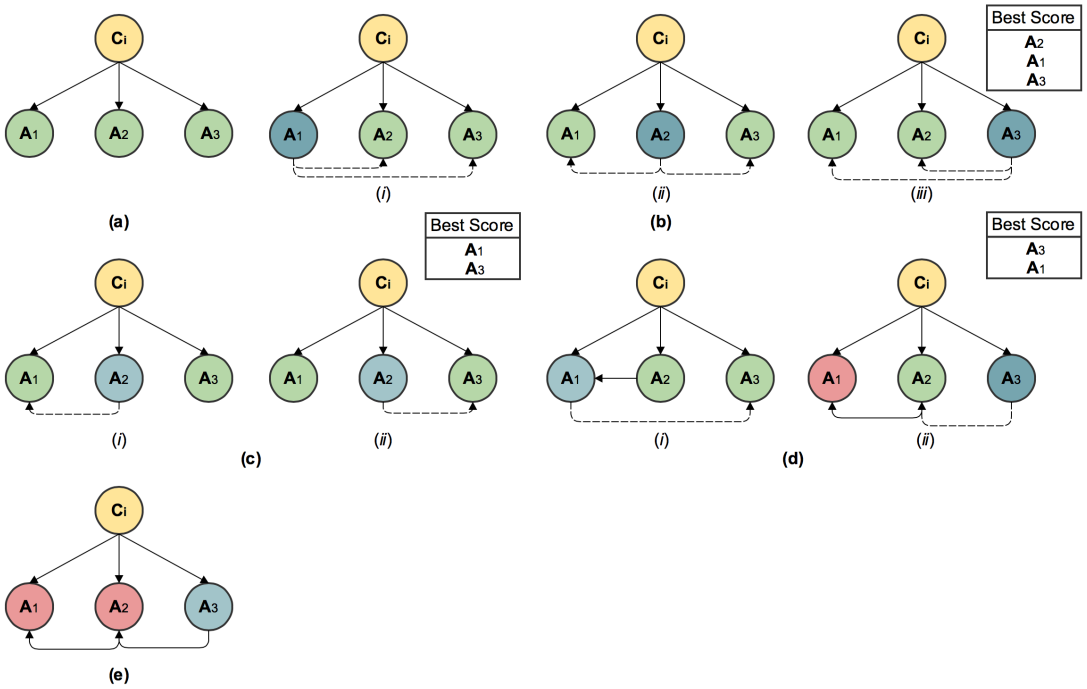

Figure 6.3: Example of the construction of the SP-TAN.

strategies (TAN and SP-TAN) are extremely costly in terms of computation time, preventing the execution of these heuristics in large datasets. Despite presenting different approaches to find attributes dependencies, both strategies have the same time complexity, $O(|\mathbb{D}_{train}| \times |\mathbb{C}| \times |\mathbb{V}^3|)$, with $|\mathbb{D}_{train}|$ being the number of training documents, $|\mathbb{C}|$ the number of classes, and $|\mathbb{V}|$ the number of attributes. Although the SP-TAN and TAN heuristics present the same complexity in the worst case, the SP-TAN heuristic has an average case less complex, since its complexity varies according to the number of selected dependencies. Thus, the heuristic only reaches the same complexity of the TAN if $(|\mathbb{V}| - 1)$ dependencies are found, which is actually difficult to happen, since the SP-TAN tends to select fewer dependencies than TAN, keeping the effectiveness (Keogh and Pazzani [1999]).

In this master thesis, we propose a heuristic, called Lazy Super Parent Tree Augmented Naive Bayes (LSPTAN) that seeks to solve the problems discussed above, enabling the application of a semi-Naive Bayes techniques in large ADC tasks. Thus, we can evaluate whether the premise of independence among attributes, assumed by Naive Bayes, impacts effectiveness in large ADC tasks, an open research problem.

6.3 Lazy Super Parent Tree Augmented Naive Bayes (LSPTAN)

The Lazy Super Parent TAN (LSPTAN) heuristic is a postergated version of the SP-TAN that constructs a Tree Augmented Naive Bayes for each test example. Attributes dependencies are generated based on information from the example that is being classified. To build a lazy version of SP-TAN we adapted the method of evaluation and the selection of candidates for Super Parent and Favorite Children.

The SP-TAN algorithm exploits accuracy to select a candidate to Super Parent (A_{SP}). In our strategy, we select the candidate A_{SP} whose classification model generates the highest probability $P(d_t|c_i, A_{SP})$ for the document d_t. Thus, each candidate A_{SP} builds its own model and classifies the document d_t, returning the predicted class c_i and its respective score $P(d_t|c_i, A_{SP})$. Based on the best scores for each candidate A_{SP}, we build a prediction score rank. Thus, the heuristic selects as best A_{SP} the attribute that holds the first position of this rank using the information of the (test) document being classified. This helps to reduce the number of candidates to Super Parents and Favorite Children, scaling the overall classification process.

Building a Tree Augmented Naive Bayes for each test example is computationally expensive. Therefore, LSPTAN builds a simpler network than the SP-TAN. We select only the best Super Parent to a test document. But there is no limitation to the choice of the Favorite Children. Thus, all the children attributes, which increment the probability that the document belongs to a class, are included in the classification model.

The LSPTAN heuristic initially builds the model based on NB and initializes a set of orphans O, inserting into O all the attributes of the vocabulary. Then, for each test document, the technique evaluates each attribute as a Super Parent (A_{SP}) and, at the end, it selects as A_{SP} the attribute that has the highest probability $P(d_t|c_i, A_{SP})$. The next step is to find the Favorite Child of the Super Parent attribute. Each attribute is individually evaluated given the A_{SP} ($P(A_i|C, A_{SP})$) and subsequently selected as one of the Favorite Children considering if the possible dependence of A_{SP} for A_i ($SP \neq i$) increases the probability of the document belonging to class c_i.

We additionally propose three novel strategies that exploit different ideas for building a LSPTAN model. These strategies are described below:

LSPTAN$_{SP}$ The LSPTAN$_{SP}$ is a simpler strategy that does not select the Favorite Children, called LSPTAN$_{SP}$. The idea of this version is to analyze the impact of only selecting the best candidate to Super Parent of all attributes without their

children. This strategy has the potential to be much more efficient and scalable, although less effective in theory, since it captures less information in the model.

LSPTAN$_{class}$ The LSPTAN$_{Class}$ is very similar to the strategy presented previously. So it uses the same method to evaluate and select candidates of the LSPTAN, however, this strategy selects a Super Parent for each class. There is no limitation for the number of Favorite Children selected. The idea of this strategy is to capture the dependencies between attributes in each class. Thus, we give a chance to other Super Parent candidates to increase their relevance when related only to their Favorite Children.

The LSPTAN$_{class}$ heuristic initially builds the model based on Naive Bayes and initializes a set of orphans O, inserting into O all the attributes of the vocabulary. Then, for each test document, the technique evaluates each attribute as a Super Parent (A_{SP}) and, at the end, for each class c_i it selects as A_{SP} the attribute that has the highest probability $P(d_t|c_i, A_{SP})$ for the class c_i. At the end of this step, C Super Parents attributes will be selected. The next step is to find the Favorite Children of each Super Parent attribute. Each attribute is individually evaluated given the A_{SP} ($P(A_j|c_i, A_{SP})$) and subsequently selected as Favorite Child considering if the possible dependence of A_{SP} for A_i ($SP \neq i$) increases the probability of the document belonging to class c_i.

LSPTAN$_{Restricted}$ The LSPTAN$_{Restricted}$ is a more efficient version of LSPTAN, in which the Super Parent candidates are evaluate along with their Favorite Children. Thus, when we select the best Super Parent, we select the Super Parent and its Favorite Children. In this strategy, we call Favorite Child the attribute that has the higher conditional probability $P(a_j|c_i, a_{sp})$ than the conditional probability $P(a_j|c_i)$. Therefore, the Super Parent candidates are associated with their Favorite Children rather than being associated with all orphans attributes (set O). So, only the attributes dependencies that increase the probability of a document belonging to a class c_i are included in the classification model.

The Super Parent candidates also have a restriction: they must exist in the test document in order to be chosen as the best Super Parent. Thus, we limit the search space of candidates, avoiding that a very informative candidate (probably belonging of a majority class) is chosen.

The LSPTAN$_{Restricted}$ heuristic initially builds the model based on Naive Bayes and initializes a set of orphans O, inserting into O all the attributes of the vocabulary. Then, for each test document, the technique evaluates each attribute which

belongs to the test document as a Super Parent (A_{SP}) and, in the end, it selects as A_{SP} the attribute that has the highest probability $P(d_t|c_i, A_{SP})$ for the class c_i. The next step is to find the Favorite Children of each Super Parent attribute. Each attribute is individually evaluated given the A_{SP} ($P(A_j|c_i, A_{SP})$) and subsequently selected as Favorite Child considering if the possible dependence of A_{SP} for A_i ($SP \neq i$) respect the condition of $P(A_j|c_i, A_{SP}) > P(a_j|c_i)$.

6.4 LSPTAN GPU Implementation

Our parallelization proposal for the LSPTAN strategy is based on eight kernels implemented using the C language in CUDA. Just like the GPU-NB, these kernels exploit the maximum possible parallelism, minimizing data dependencies among threads. It also focuses on maximizing the amount of computation performed by each thread. Since the computation of each Super Parent and Favorite Child candidates can be done independently (as well as the actual classification) for each test document, the problem takes full advantage of the GPU parallelism. The following describes the parallelization strategy used to implement the eight kernels on the GPU:

1. **Naive Bayes model:** Given the frequencies of the terms occurred in the classes and the test documents data, the classification of the test documents can be performed in parallel. Thus, given a test document, each thread is responsible for calculating the conditional probability ($P(a_j|c_i)$) of an attribute a_j for each class, cooperating to produce the NB model.

2. **Cumulative frequency of the attributes give the SP candidates within the classes:** This kernel is responsible for computing the summation of occurrences of all terms in documents that co-occur with the respective Super Parent candidate in each class ($\sum_{v \in V} Tc_i, a_{sp}(v)$). Since there is no dependency among the Super Parent candidates, this kernel can take full advantage of the GPU parallelism. While reading the training documents data, each thread is responsible for adding the occurrence of a term that occur together with the Super Parent candidate in each class.

3. **Finding the documents having SP candidates:** In text collections, usually the documents do not have all the vocabulary words. Thus, this kernel is responsible for selecting the training documents data that have the Super Parent candidates. While reading the training documents data, each thread is responsible for checking the respective term as a Super Parent candidate. If one of

the threads finds the Super Parent candidate, the training document is tagged. These tagged documents will be used in the next kernel.

4. **Computing the frequency of attributes given the SP candidate:** Once a candidate attribute is selected as a Super Parent, this kernel computes the frequency of the attributes given the Super Parent for each class ($Tc_i, a_{sp}(a_j)$).

5. **Calculation of the term probabilities in the classes given a SP:** It calculates the conditional probability of attributes given the Super Parent for each class. Thus, each thread is responsible for computing the conditional probability of an attribute given the Super Parent candidate for each class.

6. **Construction of the classification model given the SP:** Given the probability matrix built in the previous kernel for the Super Parent candidate, this kernel works in the same way that the Naive Bayes kernel does. Thus, given a test document each thread is responsible for calculating the conditional probability ($P(a_j|c_i, a_{sp})= \frac{Tc_i,a_{sp}(a_j)+\alpha}{\sum_{v\in V} Tc_i,a_{sp}(v)+\alpha\cdot V}$) of an attribute a_j for each class given the candidate of Super Parent a_{sp} and thus cooperating to produce the LSPTAN model.

7. **Construction of the classification model given the SP and the FC:** After selecting the best Super Parent, the classification of documents for each Favorite Child can be performed in parallel. Each thread represents a Favorite Child candidate. Thus, each thread is responsible for classifying all test documents for a Favorite Child for each class given the best Super Parent.

8. **Updating of the classification model:** After calculating the probabilities of the Favorite Children given the best Super Parent for a test document data, this kernel has the function of updating the model built by NB, adding the probability of the Favorite Child given the best Super Parent that increased the model probability. Each thread is responsible for an attribute and only the thread corresponding to a Favorite Child adds its probability to the NB model.

Figure 6.4 shows the sequence of execution of the LSPTAN strategies through the kernels. Thus, the LSPTAN$_{SP}$ and LSPTAN$_{Restricted}$ strategies are faster than the other LSPTAN strategies because they only execute until the kernel 6.

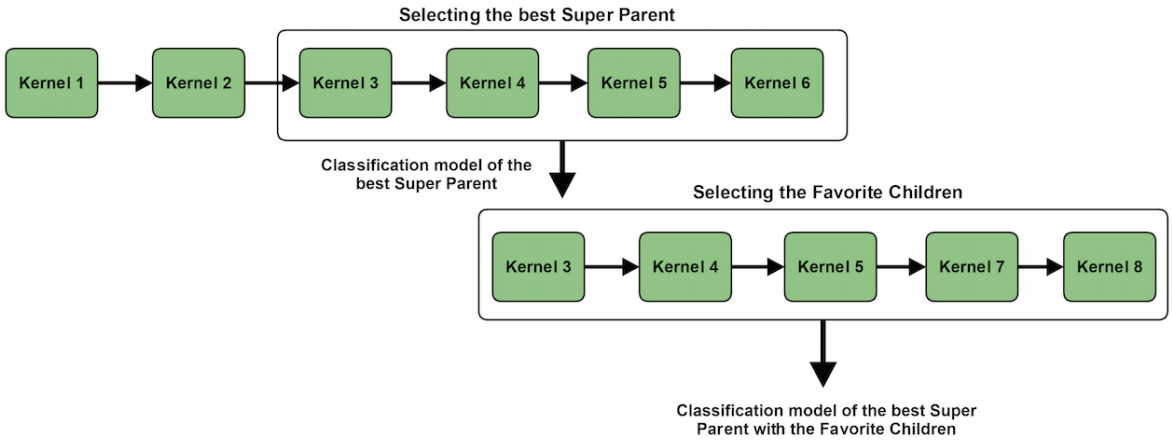

Figure 6.4: Execution of kernels

6.4.1 Kernel Optimization

In this section, we present the optimization applied in the kernels, in order to exploit the maximum parallelism possible, minimizing the dependence of data among threads. It also focuses on maximizing the amount of computation performed by each thread.

Kernels 1 and 5 exploit two levels of parallelism. In the first, it associates a block of threads with each test document and, at the second level, each of these threads is associated with a term of the same document. Thus, each block is responsible for classifying a specific test document while each thread of the block is responsible for calculating the probability of an attribute for each class, cooperating to produce the NB model. If the number of attributes is greater than the number of threads of a block, the document is split into k parts, where k is the rounded up ratio between the number of attributes of the document and the number of threads in the block. Then, each k part is calculated one at a time, in a round robin fashion, and the results are accumulated. As for the calculation of the logarithmic summation of the classification model, we implemented a binary tree parallel reduction using shared memory. Therefore, we allocated an array in shared memory whose size is equal to the number of threads in the block. Then, each thread stores the conditional probability in its respective position in the shared memory vector. The vector is divided into two sub vectors and then these sub-vectors are added. The process is repeated until the first position of the vector contains the total sum of the logarithms.

Kernels 2, 3 and 4 also exploit two levels of parallelism: the first associates a block of threads to training documents and, at the second level, each thread is associated a term of the same document. Thus, in the second level, each kernel performs its respective roles, as previously described. In kernel 2, as we are calculating the overall frequency of the attributes given the super parent candidate in each class, we use the

atomicAdd CUDA function to prevent access conflicts.

6.4.2 Analysis of Complexity for the Solution

In this section we analyze the amount of time used to learn and evaluate Super Parent candidates, as well as the amount of time used to select the Favorite Children of the best selected Super Parent. The first step of learning a model given the Super Parent candidate consists in computing the attribute frequencies to build the probability table for each class. These frequencies are obtained by kernels 2 and 4. During the execution of these kernels, the training documents data $\mathbb{D}_{train}$ are read in parallel, each thread block is responsible for a document. However, as described in section 2.3, the GPU has only a few (P) multiprocessors (SMs) and so there will be more thread blocks than SMs. Although the GPU does the block scheduling automatically, the parallelism is limited to P multiprocessors. In addition, inside each SM there are only p processors (SPs) which again limits the second level of parallelism, inside each SM, to a total of p processors. Therefore, the time spent in kernels 2 and 4 is given by $O(\sum_{d=1}^{\frac{|\mathbb{D}_{train}|}{P}} \frac{|\mathbb{V}_d|}{p})$, where $|\mathbb{V}_d|$ is the document vocabulary whose terms are processed in as many steps of p threads are needed.

After computing all the frequencies, the next step is to build the probability table given the Super Parent candidate. This probability table is two-dimensional: vocabulary by class ($|\mathbb{V}| \times |\mathbb{C}|$). The time spent to build this table is $O(\frac{|\mathbb{V}|}{P \times p} \times |\mathbb{C}|)$, because each thread is responsible for an attribute. Since we are working with real collections, it is not feasible in terms of memory space to build a three-dimensional table ($|SP| \times |\mathbb{V}| \times |\mathbb{C}|$), that stores the probability of all the Super Parent candidates. We have a trade-off between space and time. So we have to update this probability table and the respective attribute frequencies (kernels 2 and 4) whenever the strategy needs to evaluate a new candidate.

The next step consists in building the classification model (Bayesian network) for each test document given the Super Parent candidate for each class. The time spent to build the classification model for the test documents is very similar to the time spent to compute the attribute frequencies given the Super Parent candidate in kernels 2 and 4, because we use the same two-level parallelism strategy. However, in this case test documents data $\mathbb{D}_{test}$ are read (instead of training documents data) for each class. The time[2] spent in this step is $O(\sum_{d=1}^{\frac{|\mathbb{D}_{test}|}{P}} \frac{|\mathbb{V}_d|}{p} \times |\mathbb{C}|)$.

[2]The kernels 1 and 5 have the same complexity, because they exploit the same parallelism strategy, the only difference lies on the probabilities used to build the learning model.

Once all the Super Parents candidates are evaluated and the best Super Parent is selected, the Favorite Children must be selected. For this, we must compute the attribute frequencies and build the probability table for the best Super Parent. Then, the classification models are built considering only the dependencies between an attribute (Favorite Child candidate) and the the Super Parent. Thus the time spent to evaluate all Favorite Children candidates is $O(\frac{|\mathbb{V}|}{P \times p} \times \sum_{d=1}^{|\mathbb{D}_{test}|} |\mathbb{V}_d| \times |\mathbb{C}|)$, with each thread being responsible for each Favorite Child candidate. When a Favorite Child is selected, the classification model must be updated. So the time spent to update the model is $O(\frac{|\mathbb{V}_d|}{P \times p})$.

6.5 Efficiency of GPU LSPTAN

Table 6.1 shows the average time (seconds) to evaluate a Super Parent candidates using our CPU and GPU implementations. Note that the evaluation of the Super Parent candidates includes the building of the predictive model and the actual classification of all test documents. More precisely, we measure the execution time of kernels 1 to 6, where the kernels 3 to 6 are executed for each Super Parent candidate. Since the whole vocabulary (large number of attributes) is evaluated for finding Super Parent candidates, we measure the time to evaluate each candidate in a test fold. Notice that the average time to evaluate a Super Parent candidate is practically the time to compute the attribute frequencies, build the probability table given the candidate and classify all test examples. As can be seen in Table 6.1, our GPU implementation shows significant Speedup[3], ranging from 8.34 to 63.36 when compared to the sequential implementation. It is not our goal to show that our GPU version is more efficient than a CPU/multicore version, because other aspects should be considered. We chose an optimization using CUDA/GPU because it was a relatively simple solution and produced a substantial reduction of time, enabling the experiments. We believe that the performance of a parallel implementation in CPU would approach the efficiency we obtained. We rely on this hypothesis based on a recent study which shows that CPU and GPU are much closer in performance. The large gap in performance between CPU and GPU can be reduced using specific optimization for each architecture (Lee et al. [2010]). However, we must consider that there is a programming effort in choosing the parallel strategy. We believe that the cost of programming a CPU/multicore parallel strategy is considerably greater than a GPU-based parallel version, since we explore a SPMD (Single Program, Multiple Data) parallel paradigm, which is easily exploited

[3]Speedup is a metric for relative performance improvement when executing a task.

by GPUs.

Collections	CPU	GPU	Speedup
4UNI	0.081 ± 0.0019	0.0038 ± 0.00018	21.45
REUT90	0.96 ± 0.041	0.046 ± 0.0005	20.85
20NG	0.52 ± 0.025	0.0081 ± 0.00004	63.36
ACM	0.079 ± 0.0034	0.0094 ± 0.012	8.34
RCV1 Uni	93.82 ± 3.27	2.36 ± 0.046	39.8

Table 6.1: Efficiency of the GPU-based implementation in select the best Super Parent - time in seconds.

6.6 Effectiveness of LSPTAN

In this section, we present the experiments performed to evaluate the effectiveness of our Semi-Naive Bayes strategies. As mentioned before, the original SP-TAN proposal does not scale to the size of the datasets we experiment with, therefore we can not present results for this proposal.

Table 6.2 shows the results of our four LSPTAN strategies applied on the five real-world textual datasets. We compare our semi-Naive Bayes strategies with Naive Bayes, as we want to know if theses strategies are capable to extend Naive Bayes structure, increasing its classification effectiveness. For comparison purposes, we select the best results of Naive Bayes (Best-NB) obtained in Chapter 4. The $LSPTAN$ column refers to the original strategy that selects the Super Parent attribute and their Favorite Children.

We initialize the LSPTAN network using the INB model with the $RF - u$ feature weighting strategy, since this model generated the best results. INB is also the simplest model to be extended when compared to CNB and OVA NB, because it has fewer likelihoods to calculate, besides being the closest to the Multinomial version.

We can see in Table 6.2 that the best overall strategy was $LSPTAN_{SP}$, tying or improving the Best-NB results in all ten results. Indeed, some MicF1 and specially MacF1 results for 4UNI and RCV1[4] are among the best results ever reported for these datasets, with gains of up to 6.7% in MacF1 in 4UNI and 12% in MicF1 for this same dataset. The results of $LSPTAN_{SP}$ over $LSPTAN$ also show that the costly process

[4]For RCV1 only, due to its size and high dimensionality, we reduced the training set in about 40% and adopted a constraint applied in Webb et al. [2005], in which only features occurring in more than 30 documents are candidates for Super Parent. Even with these limitations, we were able to obtain improvements.

of selecting the Favorite Children is unnecessary in most cases, helping to scale the technique. We also observe that the $LSPTAN_{Class}$ reach the best results observed for the REUT90 dataset, tying with the Best-NB. The best results for ACM and 20NG were obtained with for $LSPTAN_{Restricted}$, also with gains over Best-NB.

Collections	Best-NB		LSPTAN		$\text{LSPTAN}_{\text{SP}}$		$\text{LSPTAN}_{\text{Class}}$		$\text{LSPTAN}_{\text{Restricted}}$	
	Mac$F1$	Mic$F1$	Mac$F1$	Mic$F1$	Mac$F1$	Mic$F1$	Mac$F1$	Mic$F1$	Mac$F1$	Mic$F1$
4UNI	57.58 ± 1.54	64.07 ± 1.11	$\mathbf{59.91 \pm 1.45}$	67.38 ± 0.96	$\mathbf{61.48 \pm 1.68}$	$\mathbf{71.99 \pm 1.01}$	57.87 ± 1.68	64.08 ± 1.19	59.92 ± 0.89	68.14 ± 1.72
REUT90	$\mathbf{36.02 \pm 2}$	$\mathbf{66.17 \pm 0.76}$	31.37 ± 1.39	64.2 ± 0.77	30.86 ± 0.74	63.97 ± 0.77	$\mathbf{36.21 \pm 2.11}$	$\mathbf{66.57 \pm 0.57}$	31.41 ± 1.62	63.4 ± 0.68
20NG	$\mathbf{89.64 \pm 0.6}$	$\mathbf{90 \pm 0.54}$	$\mathbf{89.66 \pm 0.59}$	$\mathbf{89.91 \pm 0.58}$	$\mathbf{89.01 \pm 0.38}$	$\mathbf{89.25 \pm 0.39}$	88.95 ± 0.58	89.35 ± 0.5	$\mathbf{90.24 \pm 0.56}$	$\mathbf{90.42 \pm 0.56}$
ACM	$\mathbf{61.57 \pm 1.6}$	74.61 ± 0.5	60.28 ± 0.92	$\mathbf{75.06 \pm 0.57}$	58.71 ± 0.92	$\mathbf{74.4 \pm 0.57}$	59.91 ± 0.6	$\mathbf{74.43 \pm 0.68}$	$\mathbf{63.05 \pm 1.47}$	$\mathbf{75.65 \pm 0.33}$
RCV1 Uni	51.68 ± 0.85	69.28 ± 0.08	–	–	$\mathbf{55.33 \pm 0.72}$	$\mathbf{73.41 \pm 0.14}$	–	–	52.83 ± 0.61	70.16 ± 0.12

Table 6.2: Results of the LSPTAN strategies considering the Super Parent and Favorite Children.

We observe by Table 6.3 that in nine out of ten cases we find a LSPTAN strategy equal or better than SVM (one ties and eight wins). When compared to KNN, the LSPTAN ties or outperforms this classifier in eight out of ten cases (one tie and seven wins). Overall, we can conclude that taking into account "some" dependencies (e.g., based on Super Parents and Favorite Children) may help to improve effectiveness in (large) datasets, depending on their characteristics. However, we were not able to choose the best LSPTAN strategy, since three of the four strategies have proven effective in different datasets. As future work, we intend to investigate how the dataset characteristics affect the potential gains of using our lazy Semi-NB strategies.

Collections	SVM		KNN		Best-LSPTAN		Best-LSPTAN (Source)
	Mac$F1$	**Mic**$F1$	**Mac**$F1$	**Mic**$F1$	**Mac**$F1$	**Mic**$F1$	
4UNI	56.02 ± 2.02	70.91 ± 1.9	$\mathbf{58.24 \pm 2.92}$	$\mathbf{73.78 \pm 0.7}$	$\mathbf{61.48 \pm 1.68}$	71.99 ± 1.01	$LSPTAN_{SP}$
REUT90	30.38 ± 2.04	65.09 ± 0.21	29.95 ± 1.14	$\mathbf{68.04 \pm 0.62}$	$\mathbf{36.21 \pm 2.11}$	66.57 ± 0.57	$LSPTAN_{Class}$
20NG	84.92 ± 0.54	85.1 ± 0.6	88.41 ± 0.27	88.69 ± 0.67	$\mathbf{90.24 \pm 0.56}$	$\mathbf{90.42 \pm 0.56}$	$LSPTAN_{Restricted}$
ACM	58.00 ± 1.33	69.67 ± 0.45	59.75 ± 1.72	71.6 ± 0.46	$\mathbf{63.05 \pm 1.47}$	$\mathbf{75.65 \pm 0.33}$	$LSPTAN_{Restricted}$
RCV1 Uni	47.05 ± 0.35	$\mathbf{75.85 \pm 0.08}$	46.32 ± 0.72	68.23 ± 0.16	$\mathbf{55.33 \pm 0.72}$	73.41 ± 0.14	$LSPTAN_{SP}$

Table 6.3: Best results compared with SVM and KNN.

6.7 Chapter Summary

In this chapter we conducted a study on the relaxation of the independence assumption of the Naive Bayes algorithm. We presented the main semi-Naive Bayes (TAN and SP-TAN) strategies proposed in the literature that alleviate the structure of the

Naive Bayes algorithm, relaxing the independence assumption. These strategies produced some gains with the Multinomial Naive Bayes in small datasets. However, when applied to real-world scenarios, such as large ADC datasets, this technique tends to loose performance due to overfitting. Moreover, these strategies are extremely costly in terms of computation time, preventing the execution of these heuristics in large datasets. Thus, we extended the SP-TAN heuristic proposing four lazy semi-Naive Bayes strategies (LSPTAN) that seek to solve the problems discussed above, enabling the application of a semi-Naive Bayes techniques in large ADC tasks. Furthermore, we used our GPU-based parallel Implementation strategy to accelerate the processing of semi-Naive Bayes models. In our efficiency result we were able to speedup the execution in up to 63.36, when compared to our sequential implementation, answering our second research question. We initialize the LSPTAN network using the INB model with the RF_u feature weighting strategy, since this model generated the best results in the previous experiment. Effectiveness results showed that considering dependencies between attributes may help to improve effectiveness in large datasets, but in our evaluation we were not able to choose the best LSPTAN strategy, since three of the four strategies have proven effective in different collections of texts. Then we leave this question open: Which dataset characteristics (or combination of characteristics) interfere in each LSPTAN strategy?

Chapter 7

Conclusions and Future Work

In this chapter we summarize the research contributions of this master thesis and point out some directions for further investigation.

7.1 New Combinations for Feature Weighting and Naive Bayes Models

In this master thesis we exploited new combinations of Naive Bayes Strategies selected in the literature, that try to overcome the ADC idiosyncrasies, with feature weighting strategies specially developed to ADC task, in a preprocessing phase. We demonstrated that a proper combination may positively impact the effectiveness of the Naive Bayes classifier in five real textual datasets. Based on our evaluation we were able to answer our first research question: a proper combination of Naive Bayes with feature weighting strategies may outperform in most cases the strongest textual classifier known in the datasets we experiment with: Support Vector Machines. In fact, Best-Naive Bayes outperformed or tied SVM in all five datasets in the MacF1 results. This is consistent with the fact that that most of the learning models and weighting strategies adaptations were designed to deal with the class imbalance issue, which tends to favor MacF1. Moreover, our results demonstrate that a combination of a INB learning model with the feature weighting RF_u may deliver a top-notch performance.

7.2 GPU-based Parallel Strategy of Naive Bayes

We also proposed GPU-NB, a parallel implementation of the Naive Bayes algorithm. In our proposal, we exploit a very compact data structure to index the documents

aiming at minimizing memory consumption, as well as maximizing the opportunities for massive parallelization of the algorithm in GPUs. In our GPU-based implementation we exploit the possible maximum parallelism, minimizing the dependence of data between threads. It also focuses on maximizing the amount of computation performed by each thread. However, we observed in our results that the communication time of the Naive Bayes algorithm data is more representative than the computation time. Although our GPU-based parallel implementation of Naive Bayes has been developed bearing in mind the GPU idiosyncrasies, the time spent transferring the datasets to the GPU global memory heavily hurt performance, resulting in speedups not greater than a few units. However, the relaxation of the Naive Bayes feature independence assumption, that is the Semi-Naive approach, was able to take full advantage of the proposed parallel implementation since these approaches require intensive computation once the dataset has been transferred to the GPU. In addition, the parallel strategy opens a new perspective and encourages the GPU-based parallelism of other Semi-NB strategies of the Structure-based group. Our GPU-based parallel strategy is generic and can be applied into other heuristics of this group.

7.3 Lazy GPU-based Semi-Naive Bayes proposal

Finally, we conducted a study on the relaxation of the independence assumption of the Naive Bayes algorithm. Thus, we present the main semi-Naive Bayes (TAN and SP-TAN) strategies proposed in the literature that alleviate the structure of the Naive Bayes algorithm, relaxing the independence assumption. These strategies produced some gains with the Multinomial Naive Bayes in small datasets. However, when applied to real-world scenarios, such as large ADC datasets, this technique tends to loose performance due to overfitting. Moreover, the strategies are extremely costly in terms of computation time, preventing the execution of these heuristics in large datasets. Thus, we extended the SP-TAN heuristic proposing four lazy semi-Naive Bayes strategies (LSPTAN) that seeks to solve the problems discussed above, enabling the application of a semi-Naive Bayes techniques in large ADC tasks. Furthermore, we used our our GPU-based parallel Implementation strategy to accelerate the processing of semi-Naive Bayes models. In our results of efficiency we were able to speedup the execution in up to 63.36 in relation to our sequential implementation which answers our second research question. We initialize the LSPTAN network using the INB model with the RF_u feature weighting strategy, since this model generated the best results. In our evaluation of the LSPTAN effectiveness we showed that dependencies between attributes can help

to improve effectiveness in large datasets. In fact, gains of more than 18.5% (MacF1 in REUT90) can be obtained by some of our strategies. However we were not able to choose the best LSPTAN strategy, since three of the four strategies have proven effective in different collections of texts. Then we leave this question open: how do the dataset's characteristics (or combination of characteristics) interfere in each LSPTAN strategy?

7.4 Future Work

As future work, we intend to answer the question raised in chapter 6. For this, we will perform a characterization of datasets evaluated with the goal to find which factors reduce and increment the quality of classification of semi-Naive Bayes proposed strategies. We believe this characterization allow us to propose new techniques, improvements in the model structure, or even isolate these factors, since we will have a better knowledge of the problem. Therefore, we can also study in details the specificity of each of these characteristics, helping to understand the impacts or justifications for the gains and losses of each of the strategies. We also intend to perform a characterization of the datasets evaluated considering the efficiency of our GPU-based parallel implementation. This characterization will help us to perform a detailed study of the efficiency of our strategies, allowing us to exploit improvements over the data structures, and to propose new strategies for parallel implementations, such as, multi-GPU solutions, CPU-GPU, or MAP-REDUCE.

In this master thesis, we select as Super Parent the attribute that has the highest probability $P(d_t|c_i, A_{SP})$ for a given document. The same strategy is used for selecting the Favorite Children. We adopted this strategy to allow that dependencies among attributes could be generated based on information from the example that is being classified, avoiding the necessity of using a validation set to select dependencies. Thus, the search for new strategies to select candidates as Super Parent, such as techniques that are able to capture the quality of dependencies between attributes is relevant for the continuity of this work. One way to capture dependencies between attributes would be through association rules, or through co-occurrence attributes, called c-features (Figueiredo et al. [2011]). These c-features are composed of terms that co-occur in documents without any restrictions order or the distance between terms within a document. A second interesting way to evaluate dependencies between attributes would be through the construction of latent variables as Super Parent attribute, i.e., to create an "attribute" which in fact is a combination of observed influences (hypothesis) of the

attributes present in the dataset. These will be two line of investigation for future work.

We also limit the construction of the Bayesian network by restricting the number of selected Super Parents. This limitation has been performed to maintain the efficiency of strategies. We do not know if adding more dependencies can positively interfere in the classification model. Therefore, it is important to seek new strategies that enable us to add new dependencies without compromising the efficiency of the LSPTAN strategy.

We will also investigate new extensions of semi-Naive Bayes proposals based on recent results. For instance, recently, some attempts have been made to relax the independent assumption by feature weighting in a Bayesian model. In these methods (Gärtner and Flach [2001]; Lee et al. [2011]) the posteriori probability is estimated by $P(a_j|c_i) = \sum_t^V P(a_j|c_i)^{w(j)}$, where $w(j)$ denotes the weight assigned to the jth attribute. This is an interesting line of investigation to be exploited by our LSPTAN strategies.

We found an effective feature weighting strategy (RF_u) for all evaluated real-world collections, but we can not guarantee that this strategy will be constantly good in all collections, because these techniques are generally designed to be applied to any type of scenarios (i.e., they are generic). Therefore, it is extremely important to investigate new feature weighting approaches that can explore the unique characteristic of each collection, maintaining the efficacy. A work that can be investigated and used for such purpose, is the method called Component Combined Approach (CCA), which uses Genetic Programming to combine several term-weight components (i.e., term frequency, normalization) extracted from well-known ranking functions. This strategy builds more effective functions than the generic strategies (de Almeida et al. [2007]). We think this is another very promising avenue for future work.

Bibliography

Adewole, A. P., Fakorede, O. J., and Akwuegbo, S. O. N. (2014). Evaluation of linear interpolation smoothing on naive bayes spam classifier.

Andrade, G., Ramos, G., Madeira, D., Oliveira, R. S., Ferreira, R., and da Rocha, L. C. (2013). G-dbscan: A gpu accelerated algorithm for density-based clustering. In *ICCS*, pages 369–378.

Atallah, M. J. and Blanton, M., editors (2010). *Algorithms and Theory of Computation Handbook: General Concepts and Techniques*. Chapman & Hall/CRC, 2 edition.

Baeza-Yates, R. A. and Ribeiro-Neto, B. A. (2011). *Modern Information Retrieval - the concepts and technology behind search, Second edition*. Pearson Education Ltd., Harlow, England.

Breiman, L. and Spector, P. (1992). Submodel Selection and Evaluation in Regression. The X-Random Case. *International Statistical Review*, 60(3).

Cha, G.-H. and Yoon, Y.-I. (2002). Clustered indexing technique for multidimensional index structures. In *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, DEXA '02, pages 935--944, London, UK, UK. Springer-Verlag.

Chang, Y.-S., Sheu, R.-K., Yuan, S.-M., and Hsu, J.-J. (2012). Scaling database performance on gpus. *Information Systems Frontiers*, 14(4):909--924.

Chen, L. and Wang, S. (2012). Automated feature weighting in naive bayes for high-dimensional data classification. In *CIKM'12*, pages 1243--1252.

Christen, P. (2012). A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24:1537–1555.

Cook, S. (2013). *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

Cooper, G. F. (1988). *Probabilistic inference using belief networks is NP-hard*. Knowledge Systems Laboratory, Computer Science Department, Univ.

de Almeida, H. M., Gonçalves, M. A., Cristo, M., and Calado, P. (2007). A combined component approach for finding collection-adapted ranking functions based on genetic programming. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 399--406, New York, NY, USA. ACM.

Drummond, C. (2006). Discriminative vs. generative classifiers for cost sensitive learning. In Lamontagne, L. and Marchand, M., editors, *Advances in Artificial Intelligence*, volume 4013 of *Lecture Notes in Computer Science*, pages 479–490. Springer Berlin Heidelberg.

Fahmi, I. (2004). Examining learning algorithms for text classification in digital libraries. In *Master's thesis*, University of Groninge, Netherland.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Fatica, M. and Luebke, D. (2007). High performance computing with CUDA. Supercomputing 2007 tutorial. In Supercomputing 2007 tutorial notes.

Figueiredo, F., Rocha, L., Couto, T., Salles, T., Gonçalves, M. A., and Jr., W. M. (2011). Word co-occurrence features for text classification. *Information Systems*, 36(5):843–858.

Friedman, N., Geiger, D., and Goldszmidt, M. (1997). Bayesian network classifiers. *Mach. Learn.*, 29(2-3):131--163.

Garcia, V., Debreuve, E., and Barlaud, M. (2008). Fast k nearest neighbor search using gpu. In *CVPRW '08. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops.*, pages 1–6.

Godbole, S. and Sarawagi, S. (2004). Discriminative methods for multi-labeled classification. In *In Pacific-Asia KDD*, pages 22--30. Springer.

Graham, T. and Mostak, T. (2014). Map-d: A gpu database for real-time big data analytics and interactive visualization. In *NVIDIA GTC-GPU Technology Conference-2014*.

Grahn, H., Lavesson, N., Lapajne, M. H., and Slat, D. (2011). Cudarf: A cuda-based implementation of random forests. In *Proceedings of the 2011 9th IEEE/ACS International Conference on Computer Systems and Applications*, AICCSA '11, pages 95--101, Washington, DC, USA. IEEE Computer Society.

Gärtner, T. and Flach, P. A. (2001). Wbc_svm: Weighted bayesian classification based on support vector machines. In *IN PROCEEDINGS OF THE EIGHTEENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING*, pages 207--209. Morgan Kaufmann.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

Hovold, J. (2005). Naive bayes spam filtering using word-position-based attributes. In *CEAS 2005 - Second Conference on Email and Anti-Spam*.

Keogh, E. J. and Pazzani, M. J. (1999). Learning augmented bayesian classifiers: A comparison of distribution-based and classification-based approaches.

Kim, S.-B., Han, K.-S., Rim, H.-C., and Myaeng, S. H. (2006). Some effective techniques for naive bayes text classification. *IEEE TKDE*, 18(11):1457 –1466.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137--1143, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Kruengkrai, C. and Jaruskulchai, C. (2002). A parallel learning algorithm for text classification. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 201--206, New York, NY, USA. ACM.

Kumarihamy, D. and Arundhati, L. (2009). Implementing data mining algorithms using NVIDIA CUDA.

Lee, C.-H., Gutierrez, F., and Dou, D. (2011). Calculating feature weights in naive bayes with kullback-leibler measure. In *Proceedings of the 2011 IEEE 11th Interna-*

tional Conference on Data Mining, ICDM '11, pages 1146--1151, Washington, DC, USA. IEEE Computer Society.

Lee, V. W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., and Dubey, P. (2010). Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu. *SIGARCH Comput. Archit. News*, 38(3):451--460.

Lewis, D., Schapire, R. E., Callan, J. P., and Papka, R. (1996). Training algorithms for linear text classifiers. pages 298--306.

Lewis, D., Yang, Y., Rose, T., and Li, F. (2004). RCV1: a new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361--397.

Lin, T.-K. and Chien, S.-Y. (2010). Support vector machines on gpu with sparse matrix format. In *Machine Learning and Applications (ICMLA)*, pages 313–318.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*, chapter 13, pages 234--265. Cambridge University Press.

Matthijssen, L. (2000). Marie-francine moens, automatic indexing and abstracting of document texts, the kluwer international series on information retrieval vol. 6. *Artificial Intelligence and Law*, 8(4):343–347.

Netzer, O. (2014). Getting big data done on a gpu-based database. In *NVIDIA GTC-GPU Technology Conference-2014*.

Rennie, J. D. M., Shih, L., Teevan, J., and Karger, D. R. (2003). Tackling the poor assumptions of naive bayes text classifiers. In *ICML'03*, pages 616--623.

Robertson, S., Walker, S., Beaulieu, M., and Willett, P. (1999). Okapi at trec-7: Automatic ad hoc, filtering, vlc and interactive track. *In*, 21:253--264.

Ruocco, A. S. and Frieder, O. (1997). Clustering and classification of large document bases in a parallel environment.

Sahami, M. (1996). Learning limited dependence bayesian classifiers. In *In KDD-96*, pages 335--338. AAAI Press.

Schapire, R. E. and Singer, Y. (2000). Boostexter: A boosting-based system for text categorization. In *Machine Learning*, pages 135--168.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47.

Shchekalev, A. (2014). Using gpus to accelerate learning to rank. In *NVIDIA GTC-GPU Technology Conference-2014*.

Singhal, A., Choi, J., Hindle, D., Lewis, D. D., and Pereira, F. (1999). At&t at trec-7. In *TREC-7)*, pages 239--252.

Teitler, B. E., Sankaranarayanan, J., Samet, H., and Adelfio, M. D. (2014). Online document clustering using gpus. In *New Trends in Databases and Information Systems*, pages 245--254. Springer.

Timm, C., Gelenberg, A., Marwedel, P., and Weichert, F. (2010). *Reducing the Energy Consumption of Embedded Systems by Integrating General Purpose GPUs*. Technical reports in computer science. TU, Department of Computer Science.

Vapnik, V. N. (1998). *Statistical learning theory*. Wiley, 1 edition.

Webb, G. I., Boughton, J. R., and Wang, Z. (2005). Not so naive bayes: Aggregating one-dependence estimators. In *Machine Learning*, pages 5--24.

Witten, I. H. and Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 1st edition.

Yang, Y. (1999). An evaluation of statistical approaches to text categorization. *Inf. Retr.*, 1(1-2):69--90.

Yang, Y. and Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 412--420, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Zhai, C. and Lafferty, J. (2001). The dual role of smoothing in the language modeling approach. In *Proceedings of the Workshop on Language Models for Information Retrieval (LMIR) 2001*, pages 31--36.

Zhang, H., Jiang, L., and Su, J. (2005). Hidden naive bayes. In Veloso, M. M. and Kambhampati, S., editors, *AAAI*, pages 919–924. AAAI Press / The MIT Press.

Zhang, T. and Oles, F. J. (2000). Text categorization based on regularized linear classification methods. *Information Retrieval*, 4:5--31.

Zheng, R., Li, J., Chen, H., and Huang, Z. (2006). A framework for authorship identification of online messages: Writing-style features and classification techniques. *JASIST*, 57(3):378--393.

Zheng, Z. and Webb, G. I. (2008). Lazy learning of bayesian rules. In *Machine Learning*, pages 53--84. Kluwer Academic Publishers.

Zheng, Z., Wu, X., and Srihari, R. (2004). Feature selection for text categorization on imbalanced data. 6:80--89.