Understanding the relevance of parallelising machine learning algorithms using CUDA for sentiment analysis

Dakun Mang Chai*
Department of Computational Engineering Sciences
Cranfield University
Bedford, Bedfordshire, United Kingdom
dakun.chai@cranfield.ac.uk

Irene Moulitsas
Department of Computational Engineering Sciences
Cranfield University
Bedford, Bedfordshire, United Kingdom
i.moulitsas@cranfield.ac.uk

Desmond Bala Bisandu
Department of Computational Engineering Sciences
Cranfield University
Bedford, Bedfordshire, United Kingdom
desmond.bisandu@cranfield.ac.uk

Abstract

Sentiment classification is essential in natural language processing, leveraging machine learning algorithms to understand the sentiment expressed in textual data. Over the years, advancements in machine learning, particularly with Naive Bayes (NB) and Support Vector Machines (SVM), have tremendously improved sentiment classification. These models benefit from word embedding techniques such as Word2Vec and GloVe, which provide dense vector representations of words, capturing their semantic and syntactic relationships. This paper explores the parallelisation of NB and SVM models using CUDA on GPUs to enhance computational efficiency and performance. Despite the computational power offered by GPUs, the literature on parallelising machine learning methods, especially for sentiment classification, remains limited. Our work aims to fill this gap by comparing the performance of NB and SVM on CPU and GPU platforms, focusing on execution time and model accuracy. Our experiments demonstrate that NB outperforms SVM in execution time and overall efficiency, mainly when using GPU acceleration. The NB model consistently achieves higher accuracy, precision, and F1 scores with Word2Vec and GloVe embeddings. The results show the importance of leveraging GPU acceleration using varying numbers of threads per block for large-scale sentiment analysis and laying the foundation for parallelising sentiment classification tasks.

CCS Concepts

• **Computing methodologies** → Information extraction; Feature selection; Cross-validation; Parallel algorithms.

Keywords

Sentiment Analysis, CUDA, Machine Learning, Word Embedding

ACM Reference Format:
Dakun Mang Chai, Irene Moulitsas, and Desmond Bala Bisandu. 2024. Understanding the relevance of parallelising machine learning algorithms using CUDA for sentiment analysis. In *2024 The 8th International Conference on Advances in Artificial Intelligence (ICAAI 2024), October 17–19, 2024, London, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3704137.3704142

*corresponding author.

ICAAI 2024, London, United Kingdom

ACM ISBN 979-8-4007-1801-4/24/10
https://doi.org/10.1145/3704137.3704142

1 Introduction

In the era of big data, sentiment analysis has emerged as a tool for understanding public opinion and extracting insights from large volumes of textual data. Sentiment analysis (SA), a subset of natural language processing (NLP), involves using machine learning algorithms to classify the tone in text data [1], [2], [3]. This capability is crucial for businesses, governments, and researchers who aim to gauge public sentiment, improve customer satisfaction, and make informed decisions. The exponential growth of unstructured data generated online drives the need for sentiment analysis. With millions of new posts and reviews appearing daily, manually analysing this data is impractical. Therefore, automated sentiment analysis using NLP techniques is essential for efficiently processing and interpreting these vast datasets. NLP allows machines to understand and process human language, enabling more sophisticated and accurate sentiment classification [4]. Machine learning (ML) algorithms have been used to classify the sentiment of documents, and some of these algorithms have shown high accuracy in classification [5], [6], [7]. These models train by converting the textual data into vector representations through traditional word embedding techniques, including word2vec [8]. Global Vector (GloVe) [9] and Fast Text [10].

However, the complexity and volume of data involved in sentiment analysis present significant computational challenges. The Central Processing Units (CPUs) often struggle to handle the intensive computations required for training and deploying advanced ML models. Graphics Processing Units (GPUs) have shown high computational power in parallel computing [11]. Initially designed for rendering graphics, GPUs have proven to be highly effective for accelerating machine learning tasks due to their parallel processing capabilities [12]. Unlike CPUs, which have a few powerful cores optimised for sequential processing, GPUs consist of many smaller cores designed for handling multiple tasks simultaneously. This architecture makes GPUs particularly well-suited for parallelising ML algorithms [11].

Parallelisation with GPUs, using frameworks like NVIDIA's Compute Unified Device Architecture (CUDA), allows for significant improvements in the speed and efficiency of computations [11]. CUDA is a parallel computing platform that uses the concept of

threads on a GPU that builds on the C and C++ programming languages. CUDA enables developers to harness the full power of GPUs to perform computations in parallel, drastically reducing the time required for data pre-processing, feature extraction, and model training. This acceleration is crucial for handling the real-time demands of modern applications [3].

In this paper, we present the parallelisation of parallelisable ML algorithms by utilising the GPUs' ability to handle large volumes of data to perform sentiment analysis classification. Training complex models on large datasets can be computationally intensive. GPUs accelerate this process by distributing the computations across many cores, significantly reducing the time required. Existing works on sentiment analysis still do not exploit the capabilities of GPUs to parallelise existing ML algorithms for sentiment classification, which offers a significant advantage in speed, scalability and performance.

The rest of this paper discusses the relevant literature and our contribution in Section 2; Section 3 discusses the methodology we used and the research architecture. We present the experimental results and discussion in Section 4. Finally, we present the conclusion and future direction in Section 5.

2 Related Work

The authors [8], [13], [14], [15] use word embedding Word2Vec, combining the CBOW and Skip-Gram models. Word2vec finds the distribution of the representation of words by training these terms against each other. After training, it chooses vectors from a pre-existing corpus. Word2vec embedding uses two approaches: skip-gram and a continuous bag of words. The authors in [9] present another word embedding method, the GloVe, that creates vector representations based on word co-occurrences. GloVe employs a more mathematical approach than Word2Vec, utilising statistics and co-occurrence matrices. The GloVe method is popular because it trains the model quickly and has a parallel implementation capacity. The work [16] presents another word embedding technique called the Valence Aware Dictionary and sEntiment Reasoner (VADER). VADER is a lexicon and rule-based sentiment analysis tool that optimises texts from social media platforms. It is used to understand the sentiment in tweets, Facebook posts, comments, and reviews. The authors [17] present a document-level embedding known as the Term Frequency Inverse-Document Frequency (TF-IDF), a statistical measure used in information retrieval and text mining to evaluate the importance of a word in a document relative to a collection of documents. Word embedding techniques are highly relevant in sentiment analysis because they capture semantic and syntactic relationships between words.

Sentiments are categorised by machine learning (ML) algorithms using both supervised and unsupervised learning methods. Unsupervised methods for sentiment analysis rely on knowledge bases that contain detailed, pre-selected information tailored explicitly for sentiment analysis. In contrast, supervised learning methods are the most commonly used due to their proven accuracy. These algorithms are trained on a labelled training set before being applied to actual data for sentiment classification. In [6], the authors classify sentiments of movie reviews using Naïve Bayes (NB), Maximum Entropy (ME) and Support Vector Machine (SVM). They use different n-gram feature sets for sentiment analysis classification. Their results show good accuracy, particularly for NB and SVM methods. The work [18] Presents sentiment analysis on tweets using a distant supervision method. They use NB, SVM and ME to classify emoticon data, achieving good results. The authors [19] present a classification of the sentiment of restaurant reviews in Cantonese. In their work, they evaluate the impact of the size of features and their representation using NB and SVM. NB outperformed SVM with a high accuracy.

SVM is popularly used in the literature for its high accuracy when handling classification and regression tasks [ref]. However, SVM is computationally expensive due to its training phase [20]. Several studies have been conducted in sequential programming on reducing training time for the linear SVM [14], [21], [22], [23]. The works in [15], [24], [25] have used parallel techniques for SVM. In [3], a novel parallel SVM training implementation was proposed, accelerating the cross-validation procedure by running multiple training tasks simultaneously on a GPU. This approach significantly reduces the overall cost of the cross-validation process.MASCOT, a method that uses GPUs and SSDs to enable fast and scalable SVM cross-validation, achieving orders of magnitude speedup over existing algorithms and enabling cross-validation on large datasets that existing algorithms cannot handle, is proposed in [26]. The authors [27] compare the performance of SVM training using CPU and GPU optimisation, finding that GPU optimisation achieves better performance with a 3.11x speedup. In [28], the authors propose a GPU-accelerated SVM approach for traffic classification, which achieves accuracy similar to the existing CPU-based LibSVM.

The relevant literature is still deficient in utilising the computational power that the GPU provides for parallelising sentiment analysis task models. Moreover, this is achievable using CUDA programming and the libraries that enable parallel writing of serial codes. In this paper, we compare the performance of two ML algorithms, NB and SVM, on the Amazon polarity review dataset. We parallelise their serial implementation using CUDA and perform sentiment classification.

2.1 Machine Learning Methods

In this section, we describe the ML algorithms used in this paper. These algorithms are chosen because they have demonstrated effectiveness in classification tasks and have shown promising results when parallelised [3], [12].

2.1.1 Support Vector Machine. Support Vector Machine (SVM) is a supervised learning model for classification and regression analysis. The primary objective of SVM is to find the optimal hyperplane that maximises the margin between data points of different classes. This hyperplane serves as a decision boundary that best separates the classes. In n-dimensional space, the hyperplane is an (n-1)-dimensional flat affine subspace; in 2D space, it is a line, and in 3D space, it is a plane. SVM aims to maximise the distance between the hyperplane and the nearest data points from each class, known as support vectors. These points are critical because they define the position and orientation of the hyperplane. If these points are removed, the position of the hyperplane will change [29]. SVM's advantage is its effectiveness in handling high-dimensional space,

which is the case in this work. In this paper, we use the linear SVM because of our linearly separable data, so SVM finds a straight line separating the classes [30].

For a binary classification problem, given training data (x_i, y_i) where x_i , is the feature vector, and y_i , is the class label such that $y_i \in \{-1, 1\}$. Equation 1) presents the SVM optimisation problem.

$$\min_{w,b,\xi} \frac{1}{2} w^2 + C \sum_{i=1}^{n} \xi_i \tag{1}$$

Subject to $y_i(w \cdot x_i + b) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, \ldots, n$

Where w is the weight vector, b is the bias, ξ_i , are the slack variables that allow for misclassification, C is the regularisation parameter.

In this paper, we parallelise the standard linear SVM method using the concept of threads. The objective is to minimise equation 1). We divide the training dataset into k chunks and each chunk is processed concurrently. We denote the chunk-specific variables and update them as follows: Let w^j and b^j , be the weight vector and bias vector for the j^{th} chunk. Each thread T_j processes a subset of data ($\{x_i^j, y_i^j\}$). We then adapt the SVM optimisation problem for each thread chunk j as prescribed in equation 2).

$$\frac{1}{2}\left\|w^j\right\|^2 + C \sum\nolimits_{i=1}^{n_j} \xi_i^j \tag{2}$$

Subject to $y_i^j(w^j \cdot x_i^j + b^j) \geq 1 - \xi_i^j, \xi_i^j \geq 0$

Each thread initialises local parameters w^j and b^j, a local update is performed for each data point (x_i^j, y_i^j) in the j^{th} chunk as shown in equations 3) and (4).

$$w^j = w^j + C \cdot \sum_{i-1}^{n_j} y_i^j x_i^j \qquad if\ y_i^j \left(w^j \cdot x_i^j + b^j\right) < 1 \tag{3}$$

$$b^j = b^j + C \cdot \sum_{i-1}^{n_j} y_i^j \qquad if\ y_i^j \left(w^j \cdot x_i^j + b^j\right) < 1 \tag{4}$$

After all the threads complete their local updates, we aggregate the results to update the global parameters w and b as presented in equations 5) and (6).

$$w = \sum_{j=1}^{k} w^j \tag{5}$$

$$b = \sum_{j=1}^{k} b^j \tag{6}$$

We present the parallelised SVM algorithm in Algorithm 1

2.1.2 Naïve Bayes. The Naive Bayes algorithm is a probabilistic classifier that utilises Bayes' theorem, making strong (naive) independence assumptions between features. It is widely employed for text classification tasks, including spam detection, sentiment analysis, and document categorisation. Given a text document D with words $w_1, w_2, \ldots w_n$, we want to classify the document into a sentiment class C (positive or negative). Using the Naive Bayes classifier, we compute the probability of each sentiment class given the document and choose the class with the highest probability.

The prior probability $P(C)$ of each sentiment class is estimated by the fraction of documents belonging to that class in the training set, as shown in equation 7).

$$P\left(C\right) = \frac{\text{Number of documents in class C}}{Total\ number\ of\ docunments} \tag{7}$$

Equation 8) presents the likelihood $P(w_i|C)$ each word given a sentiment class is estimated by the fraction of occurrences of the word in documents of that class.

$$P\left(w_i|C\right) = \frac{Count\ of\ w_i\ in\ document\ of\ class\ C + \alpha}{Total\ number\ of\ words\ in\ class\ C + \alpha \cdot V} \tag{8}$$

Here, α is the smoothing parameter (usually 1), and V is the vocabulary size (number of unique words). To classify a new document D, Calculate the likelihood probability for each sentiment class and select the class with the highest probability, as shown in equation 9).

$$\hat{C} = arg\max_{C} P\left(C\right) \cdot \prod_{i=1}^{n} P\left(w_i|C\right) \tag{9}$$

We also parallelise the NB method as shown in Algorithm 2. The prior probabilities are computed once, and there is no need to parallelise, as shown in equation 7). We then parallelise the likelihood $P(w_i|C)$ across multiple threads, where each thread computes the likelihood for a subset of the vocabulary. For each thread t, handling a subset of words $W_t \in W$ (where W is the complete vocabulary). The parallelised likelihood is presented in equation 10) for $w_i \in W_t$.

$$P_t\left(w_i|C\right) = \frac{Count\ of\ w_i\ in\ the\ document\ of\ class\ C + \alpha}{Total\ number\ of\ words\ in\ class\ C + \alpha \cdot V} \tag{10}$$

Each thread computes $\{P_t(w_i|C)|\ w_i \in W_t\}$. These results are then combined to form the complete set of likelihoods shown in equation 11).

$$P\left(w_i|C\right) = U_t P_t\left(w_i|C\right) \tag{11}$$

For each document D to be classified, we parallelise the posterior probability computation. Each thread can compute the product for a subset of words in the document. We split document D into subsets, $D_t \in D$. The partial product for thread t is computed as $\prod_{w_i \in D_t} P(w_i|C)$. Finally, we combine the partial products from all threads to compute the final posterior probability using equation 12).

$$P\left(C|D\right)\ \alpha P\left(C\right) \cdot \prod_{t} \left(\prod_{w_i \in D_t} P\left(w_i|C\right) \right) \tag{12}$$

3 Methodology

In this section, we present the parallel implementations of two ML algorithms (NB and SVM) that have shown good performance in the literature for classification tasks. Figure 1 shows the framework of this work. We perform data pre-processing on our dataset, which involves preparing the raw text data for analysis. This includes cleaning the data, removing stop words, tokenisation, and other pre-processing tasks to prepare it for the ML models. The pre-processed text data is transformed into a numerical format using word embeddings. We then feed the vectors into the SVM and NB models (serial and parallel). Selecting features that improve these models' performance by applying suitable feature selection techniques is essential. The next step is to train and validate the sentiment classification models before final predictions.

Algorithm 1 Parallelised SVM using threads

Input:
Training dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$
Regularisation parameter C
Number of threads k
Output:
Weight vector w
Bias term b
Divide dataset D into k chunks $D_1, D_2, \ldots, D_k$
For each thread T_j $(where\ j = 1, 2, \ldots, k)$ do:
$w^j = [0, 0, \ldots, 0]\ and\ b^j = 0$
For each data point (x_i, y_i) in chunks D_j
If $y_i^j (w^j \cdot x_i^j + b^j) < 1$
Update $w^j\ as\ w^j = w^j + C \cdot y_i^j \cdot x_i^j$
Update $b^j as\ b^j = b^j + C \cdot y_i^j$
End if
End For
End for
For each thread T_j do:
Update the global weight vector $w : w = w + w^j$
Update global bias term $b : b = b + b^j$
End for

Algorithm 2 Parallelised NB using threads

Input:
Training dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$
Number of threads k
Output:
Class prior probabilities $P(y)$
Conditional probabilities $P(x_i|y)$ for each feature x_j
Initialisation:
Divide dataset D into k chunks $D_1, D_2, \ldots, D_k$
Initialize counts for class priors and conditional probabilities for each chunk
For each thread T_j $(where\ j = 1, 2, \ldots, k)$ do:
Initialise local counts for class priors and conditional probabilities
For each data point (x_i, y_i) in chunks D_j
Update the local count of y_i
For each $x_{ij} in\ x_i$:
Update the local counts of $x_{ij}\ given\ y_i$
End if
End For
End for
Aggregate the local counts into global counts
Compute the global class prior probabilities $P(y)$ and conditional probabilities $P(x_i|y)$

3.1 Pre-processing

Data pre-processing is a critical step in the data analysis and machine learning pipeline, involving transforming raw data into a clean and usable format. This step is essential for ensuring the quality and accuracy of the subsequent analysis or model development. Ensuring accurate, consistent, and complete data is critical for obtaining reliable analysis and results. Proper pre-processed

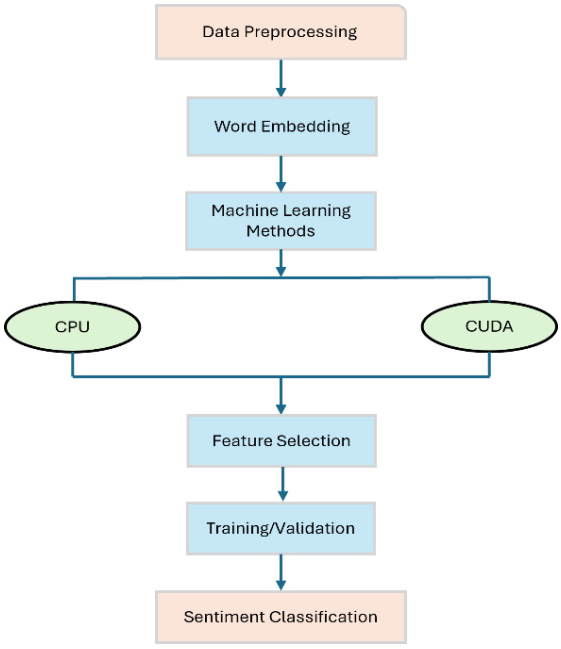

Figure 1: Serial and parallel machine learning framework for sentiment classification

data can reveal insights hidden by noise, leading to more informed, accurate results. We discuss the relevant pre-processing steps we carried out in this paper: Stopword removal is an essential pre-processing step in NLP and text mining. It involves identifying and removing common words that are typically considered to have little or no informational value sentiment classification [31], [32]. These common words, known as stopwords, include terms like "the," "is," "in," and "to." Converting text to lowercase ensures that words are represented uniformly. For instance, "Apple," "apple," and "APPLE" are all transformed to "apple" This uniformity helps treat these variations as the same word. Lowercasing simplifies the text and reduces variability without losing significant information. Tokenisation is a fundamental pre-processing step in NLP and text mining. It involves splitting text into smaller units called tokens, words, phrases, or characters. Tokenisation is crucial for preparing text data for further analysis or processing by machine learning algorithms [33]. For instance, the text "The quick brown fox jumps over the lazy dog" would be tokenised as ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]. We perform lemmatisation, a text normalisation technique that reduces words to their base or root form, known as a "lemma". Unlike stemming, which truncates words to remove prefixes or suffixes, lemmatisation considers the context and converts words to their meaningful base form. Lemmatisation reduces the number of unique tokens in the text, leading to a smaller, more manageable feature space [13].

3.2 Word Embedding

Word embedding is a technique that represents words as dense, continuous, and low-dimensional vectors in a high-dimensional space. These vectors capture semantic and syntactic relationships between

words, allowing machines to understand the context and meaning of words better. In this work, we employed the Word2Vec and GloVe embedding techniques. Word2Vec utilises neural networks to generate word embeddings, producing dense vector representations of words in a continuous vector space. It employs two models: Continuous Bag of Words (CBOW) and skip-gram. CBOW predicts a target word based on its surrounding context words, while skip-gram predicts the context words based on a target word. Both methods use a simple neural network with one hidden layer.

3.2.1 CBOW. Predict the target word w given its context words $c_1, c_2, \ldots c_c$. For CBOW, the hidden layer representation h is the average of the embeddings of the context words as shown in equation 13):

$$h = \frac{1}{c} \sum_{i=1}^{c} W^T x_{c_i} \tag{13}$$

where x_{c_i} is the one-hot encoded vector of the context word c_i, and $W^T x_{c_i}$ gives the embedding of c_i. The output logits u for the target word w are computed as in equation 14):

$$u = W^{'T} h \tag{14}$$

Equation 15) calculates the softmax function applied to u to get the probability distribution over the vocabulary:

$$y_i = \frac{exp\,(u_j)}{\sum_{k=1}^{v} \exp(u_k)} \tag{15}$$

The cross-entropy loss function for CBOW is computed using equation 16):

$$L = -\log\,(y_w) \tag{16}$$

where y_w , is the predicted probability of the actual target word w.

3.2.2 Skip-gram. Predicts the context words $c_1, c_2, \ldots c_c$ given a target word w. The hidden layer representation h is the embedding of the target word w: $h = W^T x_w$ where x_w , is the one-hot encoded vector of the target word w. The output logits u_{c_i} for each context word c_i , are computed as in equation 17):

$$u_{c_i} = W^{'T} h \tag{17}$$

The softmax function is applied to u_{c_i} , to get the probability distribution over the vocabulary using the equation 18):

$$y_j^{(i)} = \frac{exp\,(u_{c_i,j})}{\sum_{k=1}^{v} exp\,(u_{c_i,k})} \tag{18}$$

where $y_j^{(i)}$, is the probability of the j^{th} word in the vocabulary is the i^{th}context word. The cross-entropy loss for Skip-Gram is computed in equation 19), which is the sum of the losses for each context word:

$$L = -\sum_{i=1}^{c} \log\,(y_{c_i}) \tag{19}$$

where y_{c_i} is the predicted probability of the actual i^{th} context word c_i.

Notations:

V: Vocabulary size (number of unique words in the corpus).
N: Dimension of the word embedding vector.
w: A specific word in the vocabulary.
c: A context word surrounding the target word.
W: Weight matrix from the input to the hidden layer (dimension $V \times N$).
$W^{'}$: Weight matrix from the hidden layer to the output layer (dimension $N \times V$).
h: Hidden layer representation (embedding of the input word).
u: Output layer (logits before applying softmax).
y: Output probability distribution after applying softmax.
C: Context window size.

4 Results and discussion

In this section, we discuss the effectiveness of two word embedding models, Word2Vec and GloVe, on ML models and their computational efficiencies on the CPU and GPU. The remaining part of this section presents the experimental setup, results from the experiment, and interpretations of the results.

4.1 Hardware and Software Configuration

We experimented using the Python Jupyter Notebook to pre-process and vectorise data. We then used CUDA programming built on C and C++ for SVM and NB's serial and parallel implementations. In this experiment, we utilised the Nvidia A100-PCIE-40GB GPU. The CUDA driver and runtime versions are 11.2/11.8, with 42 multiprocessors and 64 cores per multiprocessor. On this GPU, there are 1,024 maximum threads per block. These features, combined with sizeable global memory and extensive support for advanced CUDA capabilities, make it well-suited for complex machine-learning tasks.

4.1.1 Dataset Description. This paper uses the Amazon review polarity dataset constructed in [34] for sentiment classification, which is originally from the work in [35], contains 3.2 million samples. In the dataset, class 1 is negative, and class 2 is positive. Three features correspond to the class index, review title, and review text. In this paper, we used the class and review test columns for our experiment.

4.1.2 Results. We conducted our experiment on our dataset using the two ML models, as presented in section 3. This experiment classifies sentiments (positive or negative), and we perform a comparative analysis of the performance of the ML algorithms; this is to understand the effectiveness of different word embedding techniques and the parallel computing capabilities of the GPU for sentiment classification tasks.

4.1.3 Evaluation Metrics. We evaluate our work using the most commonly used metrics for sentiment analysis: accuracy, precision, recall, F1 score, and the ROC curve. True Positives (TP) and True Negatives (TN) correctly predict positive and negative instances. Meanwhile, the False Positives (FP) and False Negatives (FN) are incorrectly predicted positively and negatively, respectively. The formulas for these metrics are provided in equations 20), (21), (22), and (24).

- Accuracy is the ratio of correctly predicted instances to the total number of instances, indicating how often the classifier makes correct predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{20}$$

Table 1: Performance metrics for SVM and NB

Word Embedding	SVM - CPU and GPU (%)	NB - CPU and GPU (%)
Word2Vec	Accuracy: 74.78	Accuracy: **76.10**
	Precision: 74.13	Precision: **76.92**
	Recall: 76.12	Recall: 74.57
	F1 Score: 75.11	F1 Score: 75.73
GloVe	Accuracy: 73.01	Accuracy: **76.10**
	Precision: 74.33	Precision: **76.44**
	Recall: 70.27	Recall: 75.41
	F1 Score: 72.25	F1 Score: 76.00

- Precision is the ratio of correctly predicted positive instances to the total predicted positive instances. It measures the accuracy of positive predictions.

$$Precision = \frac{TP}{TP + FP} \tag{21}$$

- Recall, also called the True Positive Rate, is the ratio of correctly predicted positive instances to all actual positive instances. It measures the ability of the model to capture all positive instances.

$$Recall = \frac{TP}{TP + TN} \tag{22}$$

- F1 Score is the harmonic mean of precision and recall. It provides a single metric that balances both precision and recall, which is especially useful when the classes are imbalanced.

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{23}$$

4.1.4 Discussion. In this paper, we used cross-validation [36], [37], a statistical method used to assess the performance and generalisability of a machine learning model that involves dividing the dataset into multiple subsets. The model is trained on some of these subsets and validated on the remaining ones. This procedure is repeated multiple times, allowing each data point to be included in both the training and validation sets. The results from each iteration are then averaged to provide a more accurate evaluation of the model's performance. In [3], n-fold cross-validation is generally applied to SVM training. We used the 5-fold cross-validation technique to train our models (SVM and NB) with a range of hyperparameters for the training of the SVM. The learning rates (lr) are 0.1, 0.01, $1e-3$, and $5e-3$ together with the regularisation parameters (C). In this work, we only report the results with the best hyperparameters. We split our data into 80% for training and 20% for testing.

Our results are classified into two categories: the implementation of serial and parallel SVM using word2vec and GloVe embeddings for sentiment classification and the implementation of serial and parallel NB using word2vec and GloVe embeddings for sentiment classification.

Table 1 shows that NB performed better than SVM across both word embedding techniques in terms of accuracy. Specifically, NB achieves 76.10% accuracy with Word2Vec and GloVe, while SVM achieves 74.78% and 73.01%, respectively. Precision is consistently higher for NB compared to SVM for both embeddings. With Word2Vec, NB achieves 76.92% against SVM's 74.13%. With GloVe, NB achieves 76.44% against SVM's 74.33%. We can also observe that SVM demonstrates higher recall than NB with Word2Vec embeddings (76.12% vs 74.57%), indicating that SVM is more effective at identifying positive instances. However, with GloVe embeddings, NB shows a higher recall (75.41% vs SVM's 70.27%), suggesting better performance of NB in capturing actual positive instances for this embedding.

Table 2 presents the CPU and kernel execution times, measured in seconds, for SVM and NB models using two different word embedding techniques: Word2Vec and GloVe. In our result, for SVM using word2vec, the GPU (7.65 seconds) is about 2.83x faster than the CPU (21.63 seconds), whereas the GPU (7.73 seconds) is about 2.95x faster than the CPU (22.85 seconds) using GloVe. For NB using word2vec, the GPU (0.38 seconds) is about 1.47x faster than the CPU (0.56 seconds), whilst the GPU (0.37 seconds) is about 1.49x faster than the CPU (0.55 seconds) using GloVe. We present these results in Figures 10 and 11. The use of GPUs substantially reduces execution times for both SVM and NB models, highlighting the advantage of leveraging GPU acceleration in sentiment classification tasks. This mainly benefits large datasets and complex models with high computational demands.

The experimental results in Table 3 and 4 evaluate the performance of SVM and NB algorithms with GloVe and word2vec embeddings, focusing on the impact of different numbers of threads per block on the GPU training time. The results in Table 1 show consistency in the performance metrics across the different numbers of threads, which suggests that the choice of threads per block primarily affects computational efficiency rather than the model's predictive quality. The results in Tables 3 and 4 show that for GloVe and word2vec embeddings on the SVM algorithm, the optimal GPU kernel time was achieved with 128 and 256 threads per block, indicating that both configurations are efficient. Higher threads, such as 1024 threads per block, introduce overhead, increasing kernel execution time. For the NB model, both word embedding techniques, 256 threads per block, provide the best balance between GPU training time and overall computational efficiency.

Figure 2 shows the confusion matrix for the SVM classifier with Word2Vec embeddings; the model demonstrates a strong ability to correctly identify positive sentiments, as seen by the high number

Table 2: CPU and GPU kernel execution times for SVM and NB

	Word2VecTime (seconds)	GloVeTime (seconds)
Host/Device	SVM NB	SVM NB
CPU	21.63 0.56	22.85 0.55
GPU	**7.65 0.38**	**7.73 0.37**

Table 3: CPU and GPU kernel execution times for SVM and NB on 128, 256, 512 and 1024 threads per block using GloVe embedding

	CPU ExecutionTime (seconds)	GPU Kernel ExecutionTime (seconds)
Number of threads Per block	SVM NB	SVM NB
128	21.79 0.42	7.73 0.53
256	22.85 0.55	**7.73 0.39**
512	21.81 0.44	7.73 0.53
1024	21.80 0.44	8.07 0.61

Table 4: CPU and GPU kernel execution times for SVM and NB on 128, 256, 512 and 1024 threads per block using word2vec embedding

	CPU ExecutionTime (seconds)	GPU Kernel ExecutionTime (seconds)
Number of threads Per block	SVM NB	SVM NB
128	21.67 0.47	7.66 0.55
256	21.67 0.56	**7.66 0.38**
512	21.81 0.46	7.67 0.55
1024	21.66 0.47	8.01 0.58

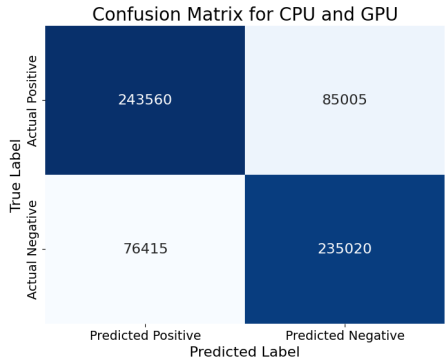

Figure 2: Confusion matrix for SVM on CPU and GPU using word2vec embedding

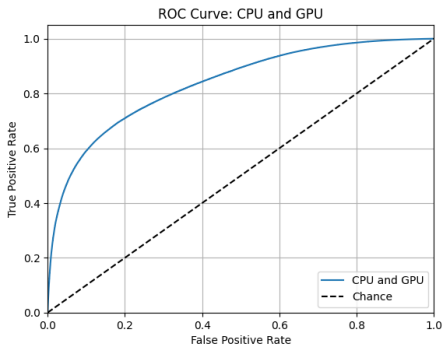

Figure 3: ROC curve for SVM on CPU and GPU using word2vec embedding

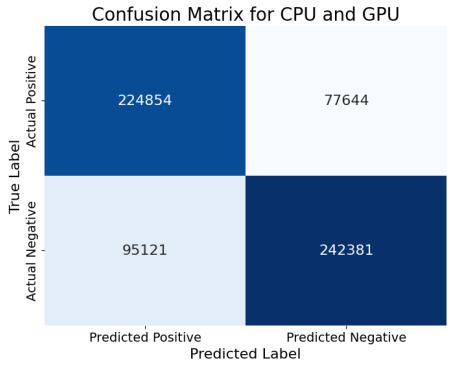

Figure 4: Confusion matrix for SVM on CPU and GPU using GloVe embedding

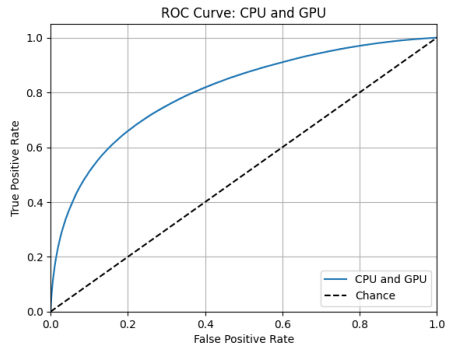

Figure 5: ROC curve for SVM on CPU and GPU using GloVe embedding

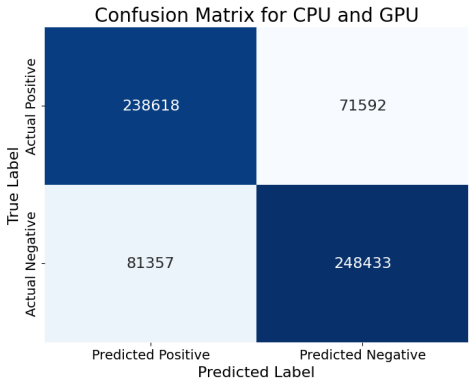

Figure 6: Confusion matrix for NB on CPU and GPU using Word2Vec embedding

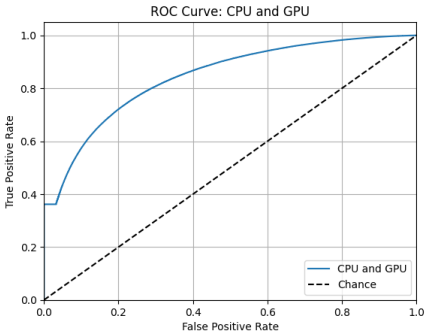

Figure 7: ROC curve for NB on CPU and GPU using Word2Vec embedding

of true positives (243,560). The result indicates that the Word2Vec embedding effectively captures the nuances of positive language. It is a reliable tool for applications that accurately identify positive feedback, such as customer satisfaction surveys and product reviews. However, the 76,415 false positives in the confusion matrix, where the model incorrectly labels negative sentiments as positive, suggests that the Word2Vec model sometimes misinterprets negative feedback. This misclassification can lead to the assumption that there are more positive sentiments than there are, resulting in a biased analysis and overestimated conclusions on customer satisfaction. Additionally, the model has 85,005 false negatives, indicating it occasionally fails to recognise positive sentiments, instead categorising them as negative. This underestimation of positive sentiment could result in missing critical positive feedback, which is essential for understanding customer satisfaction and identifying strengths in products or services.

Figure 4 shows the SVM classifier with GloVe embeddings. The model correctly identifies negative sentiments, as seen by the 242,381 true negatives. The result also suggests that GloVe embeddings are particularly effective at distinguishing negative sentiment, where accurately filtering out negative content is crucial. The model also shows many false positives (95,121), meaning it misclassifies negative sentiments as positive. This can lead to incorrect assessments of sentiments. The GloVe embedding model also presents 77,644 false negatives, where it fails to identify positive sentiments, mistakenly categorising them as negative. This misclassification could lead to an underrepresentation of positive feedback, affecting the overall understanding of customer sentiment and potentially missing opportunities for leveraging positive feedback to improve products or services.

The ROC curves in Figures 3 and 5 show strong performance as they rise steeply towards the top left corner. This indicates that the classifier has a high true positive rate while maintaining a relatively low false positive rate for a significant range of thresholds.

In the context of sentiment classification, the choice between Word2Vec and GloVe embeddings should be guided by the specific goals of the analysis. If the primary objective is to maximise the accurate identification of positive feedback, the Word2Vec embedding might be more suitable. On the other hand, if the focus is on accurately identifying and filtering out negative feedback, the GloVe embedding could be more appropriate. Understanding these differences allows for more informed decisions in tailoring sentiment analysis models to better meet the needs of various applications.

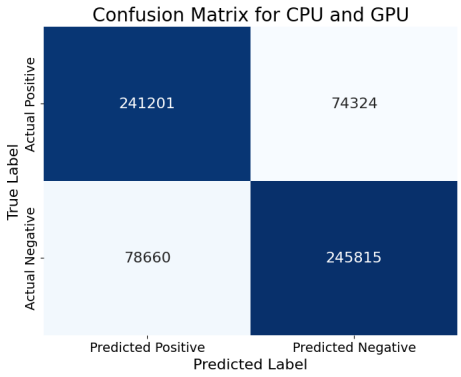

Figure 8: Confusion matrix for NB on CPU and GPU using GloVe embedding

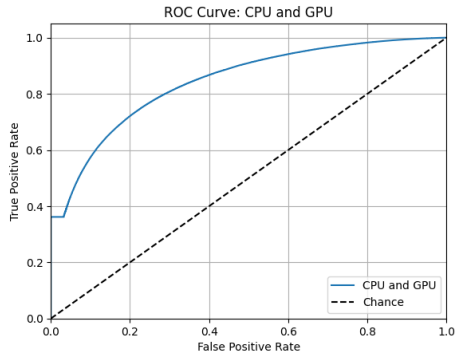

Figure 9: ROC curve for NB on CPU and GPU using GloVe embedding

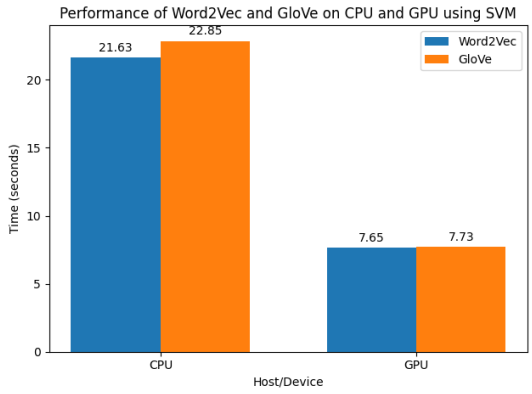

Figure 10: Execution time of the Word2Vec and GloVe models on the CPU and GPU using SVM

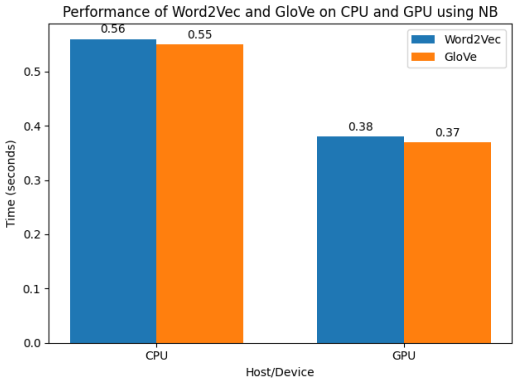

Figure 11: Execution time of the Word2Vec and GloVe models on the CPU and GPU using NB

Figure 6 presents the confusion matrix for the NB classifier with Word2Vec embeddings and shows that the model correctly identifies a substantial number of positive sentiments, with 238,618 true positives. This indicates that the Word2Vec embedding captures the positive features well, making it useful for applications where detecting positive feedback is crucial, such as customer reviews or satisfaction surveys. The model incorrectly labels 81,357 negative sentiments as positive. This can create the impression of more positive sentiments than there are, leading to overly optimistic views on customer satisfaction or product reception. Additionally, the model misses 71,592 positive sentiments, labelling them as negative. The model correctly identifies 248,433 negative sentiments, which is a significant number, indicating its effectiveness in recognising negative language. However, the false positives and negatives highlight areas where the model could improve to provide a more balanced and accurate sentiment analysis.

In Figure 8, we have the NB classifier with GloVe embeddings, and the model shows a strong ability to identify positive sentiments, with 241,201 true positives correctly. This implies that GloVe embeddings effectively capture positive samples, making them suitable for sentiment analysis applications focused on understanding positive customer feedback. The model also has 78,660 false positives, showing that it misinterprets negative sentiments as positive. Additionally, there are 74,324 false negatives, where the model fails to recognise positive sentiments, mistakenly categorising them as negative. This misclassification could result in underrepresenting positive feedback, affecting the overall understanding of customer sentiment. The model correctly identifies 245,815 negative sentiments, indicating its ability to identify negative samples accurately. However, the false positives and false negatives suggest that there is still room for improvement in achieving a more precise sentiment classification.

In Figures 7 and 9, the ROC curve for the Naive Bayes classifier lies above the diagonal line, showing that the model is significantly better than random guessing, thus demonstrating its effectiveness in classifying sentiments.

5 Conclusion and future work

Sentiment classification, a crucial task in natural language processing, has seen significant advancements over the years with the development and application of various ML algorithms. Initially, simpler models like NB and SVM are widely used due to their robustness and effectiveness in handling text data. These models laid

the groundwork for more sophisticated approaches by demonstrating the potential of machine learning in understanding and interpreting human sentiments from textual data. SVMs' capability to maximise the margin between classes has contributed significantly to their effectiveness in sentiment analysis, especially in scenarios with well-separated class boundaries. NB models have consistently shown strong performance in sentiment classification due to their simplicity and effectiveness in handling high-dimensional data. They operate on the principle of conditional probability, making them particularly efficient in predicting the likelihood of class membership.

Despite the inherent advantages of SVMs, their computational complexity, particularly with large datasets, has been a challenge. Some works have attempted to parallelise SVM for various tasks, such as large-scale classification and regression problems. These efforts have leveraged modern computational architectures, including multi-core CPUs and GPUs, to distribute the computational load and accelerate training and inference processes. Such parallelisation has shown promising results in reducing execution times and improving the scalability of SVM models. However, the parallelisation of ML methods, including SVM, for sentiment classification tasks has been relatively limited in the literature. This gap is primarily due to the unique challenges posed by text data, such as the need for complex pre-processing and the high dimensionality of word embeddings. While progress has been made, comprehensive studies and implementations that fully exploit parallel computing architectures for sentiment analysis remain sparse. This work contributes to filling this gap by evaluating the performance of NB and SVM models on both CPU and GPU platforms, demonstrating the significant benefits of parallelisation.

In this work, we evaluated the performance of SVM and NB models for sentiment classification using Word2Vec and GloVe embeddings. Our results indicate that the NB model consistently outperforms the SVM model in terms of execution time and efficiency, especially when utilising GPU acceleration. The NB model also shows robust performance metrics, including higher accuracy, precision, and F1 scores. In both the SVM and NB algorithms, consistent predictive performance across different thread configurations suggests that computational efficiency is the primary factor affected by the number of threads per block. We recommend 128 or 256 threads per block for SVM with GloVe and word2vec embeddings for optimal GPU performance and computational efficiency on our dataset. In comparison, 256 threads per block are optimal for NB with both word embeddings. Our results show that a higher number of threads introduces overhead that negatively impacts the GPU training time without improving the predictive performance.

The significant reduction in execution times with GPU usage shows the importance of leveraging parallel computing for large-scale sentiment analysis. While parallelising ML methods for sentiment classification has been limited in the literature, this paper demonstrates the potential for substantial improvements in computational efficiency and scalability.

We will consider deep learning algorithms as a future direction because of their potential for advancing sentiment classification.

Acknowledgments

We acknowledge the Petroleum Technology Development Fund (PTDF) Nigeria funding, which sponsors the first author's PhD research, with ID PTDF/ED/OSS/PHD/DMC/1972/22.

References

[1] J. Khan, A. Alam, and Y. Lee, "Intelligent Hybrid Feature Selection for Textual Sentiment Classification," IEEE Access, vol. 9, pp. 140590–140608, 2021, doi: 10.1109/ACCESS.2021.3118982.

[2] F. K. Khattak, S. Jeblee, C. Pou-Prom, M. Abdalla, C. Meaney, and F. Rudzicz, "A survey of word embeddings for clinical text," J Biomed Inform, vol. 100, p. 100057, 2019, doi: 10.1016/j.yjbinx.2019.100057.

[3] Q. Li, R. Salman, E. Test, R. Strack, and V. Kecman, "Parallel multitask cross validation for Support Vector Machine using GPU," J Parallel Distrib Comput, vol. 73, no. 3, pp. 293–302, Jun. 2013, doi: 10.1016/j.jpdc.2012.02.011.

[4] J. Khan and B. S. Jeong, "Summarising customer review based on product feature and opinion," in 2016 International Conference on Machine Learning and Cybernetics (ICMLC), IEEE, Jun. 2016, pp. 158–165. doi: 10.1109/ICMLC.2016.7860894.

[5] J. Khan, A. Alam, J. Hussain, and Y.-K. Lee, "EnSWF: effective features extraction and selection in conjunction with ensemble learning methods for document sentiment classification," Applied Intelligence, vol. 49, no. 8, pp. 3123–3145, Jun. 2019, doi: 10.1007/s10489-019-01425-4.

[6] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? Sentiment Classification using Machine Learning Techniques," CoRR, vol. cs.CL/0205, 2002, [Online]. Available: https://arxiv.org/abs/cs/0205070

[7] M. R. Saleh, M. T. Martín-Valdivia, A. Montejo-Ráez, and L. A. Ureña-López, "Experiments with SVM to classify opinions in different domains," Expert Syst Appl, vol. 38, no. 12, pp. 14799–14804, Jun. 2011, doi: 10.1016/j.eswa.2011.05.070.

[8] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in Advances in Neural Information Processing Systems, C. J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2013. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

[9] J. Christopher D Pennington, R. Socher, and Manning, "Glove: Global vectors for word representation," in Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, pp. 1532–1543.

[10] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," Trans Assoc Comput Linguist, vol. 5, pp. 135–146, Jun. 2017, doi: 10.1162/tacl_a_00051.

[11] A. Garg, D. Gupta, P. P. Sahadev, and S. Saxena, "Comprehensive Analysis of the Uses of GPU and CUDA in Soft-Computing Techniques," in 2019 6th International Conference on Signal Processing and Integrated Networks (SPIN), IEEE, Mar. 2019, pp. 584–589. doi: 10.1109/SPIN.2019.8711671.

[12] A. Ioannis Athanasopoulos, A. Dimou, V. Mezaris, and Kompatsiaris, "GPU acceleration for support vector machines," in Procs. 12th Inter. Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2011), 2011.

[13] D. Effrosynidis, S. Symeonidis, and A. Arampatzis, "A Comparison of Preprocessing Techniques for Twitter Sentiment Analysis," 2017, pp. 394–406. doi: 10.1007/978-3-319-67008-9_31.

[14] R.-E. Thorsten Fan, P.-H. Chen, C.-J. Lin, and Joachims, "Working set selection using second order information for training support vector machines," Journal of machine learning research, vol. 16, no. 12, 2005.

[15] G.-B. Huang, K. Z. Mao, C.-K. Siew, and D.-S. Huang, "Fast Modular Network Implementation for Support Vector Machines," IEEE Trans Neural Netw, vol. 16, no. 6, pp. 1651–1663, Jun. 2005, doi: 10.1109/TNN.2005.857952.

[16] C. Hutto and E. Gilbert, "VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text," Proceedings of the International AAAI Conference on Web and Social Media, vol. 8, no. 1, pp. 216–225, Jun. 2014, doi: 10.1609/icwsm.v8i1.14550.

[17] A. Patil, "Word Significance Analysis in Documents for Information Retrieval by LSA and TF-IDF using Kubeflow," 2022, pp. 335–348. doi: 10.1007/978-981-16-2126-0_29.

[18] A. Lei Go, R. Bhayani, and Huang, "Twitter sentiment classification using distant supervision," CS224N project report, Stanford, vol. 1, no. 12, p. 2009, 2009.

[19] Z. Zhang, Q. Ye, Z. Zhang, and Y. Li, "Sentiment classification of Internet restaurant reviews written in Cantonese," Expert Syst Appl, vol. 38, no. 6, pp. 7674–7682, Jun. 2011, doi: 10.1016/j.eswa.2010.12.147.

[20] J. C. Platt, "Fast Training of Support Vector Machines Using Sequential Minimal Optimisation," in Advances in Kernel Methods, The MIT Press, 1998. doi: 10.7551/mitpress/1130.003.0016.

[21] C.-C. Chang and C.-J. Lin, "LIBSVM," ACM Trans Intell Syst Technol, vol. 2, no. 3, pp. 1–27, Jun. 2011, doi: 10.1145/1961189.1961199.

[22] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to Platt's SMO Algorithm for SVM Classifier Design," Neural Comput, vol. 13, no. 3, pp. 637–649, Jun. 2001, doi: 10.1162/089976601300014493.

[23] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," in Neural Networks for Signal Processing VII. Proceedings

of the 1997 IEEE Signal Processing Society Workshop, IEEE, pp. 276–285. doi: 10.1109/NNSP.1997.622408.

[24] R. Collobert, S. Bengio, and Y. Bengio, "A Parallel Mixture of SVMs for Very Large Scale Problems," in Advances in Neural Information Processing Systems, T. Dietterich, S. Becker, and Z. Ghahramani, Eds., MIT Press, 2001. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2001/file/36ac8e558ac7690b6f44e2cb5ef93322-Paper.pdf

[25] J. Dong, A. Krzyżak, and C. Suen, "A Fast Parallel Optimization for Training Support Vector Machine," in Machine Learning and Data Mining in Pattern Recognition, Springer Berlin Heidelberg, pp. 96–105. doi: 10.1007/3-540-45065-3_9.

[26] Z. Wen, R. Zhang, K. Ramamohanarao, J. Qi, and K. Taylor, "MASCOT: Fast and Highly Scalable SVM Cross-Validation Using GPUs and SSDs," in 2014 IEEE International Conference on Data Mining, IEEE, Jun. 2014, pp. 580–589. doi: 10.1109/ICDM.2014.35.

[27] N. S. M. Salleh and M. F. Baharim, "Performance Comparison of Parallel Execution Using GPU and CPU in SVM Training Session," in 2015 4th International Conference on Advanced Computer Science Applications and Technologies (ACSAT), IEEE, Jun. 2015, pp. 214–217. doi: 10.1109/ACSAT.2015.31.

[28] G. Sun, "GPU-Accelerated Support Vector Machines for Traffic Classification," International Journal of Performability Engineering, 2018, doi: 10.23940/ijpe.18.05.p28.10881098.

[29] V. N. Vapnik, The Nature of Statistical Learning Theory. Springer New York, 2000. doi: 10.1007/978-1-4757-3264-1.

[30] A. K. Uysal and S. Gunal, "A novel probabilistic feature selection method for text classification," Knowl Based Syst, vol. 36, pp. 226–235, Dec. 2012, doi: 10.1016/J.KNOSYS.2012.06.005.

[31] N. Babanejad, A. Agrawal, A. An, and M. Papagelis, "A Comprehensive Analysis of Pre-processing for Word Representation Learning in Affective Tasks," in Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2020, pp. 5799–5810. doi: 10.18653/v1/2020.acl-main.514.

[32] E. Haddi, X. Liu, and Y. Shi, "The Role of Text Pre-processing in Sentiment Analysis," Procedia Comput Sci, vol. 17, pp. 26–32, 2013, doi: 10.1016/j.procs.2013.05.005.

[33] J. Camacho-Collados and M. T. Pilehvar, "On the Role of Text Pre-processing in Neural Network Architectures: An Evaluation Study on Text Categorisation and Sentiment Analysis," CoRR, vol. abs/1707.0, 2017, [Online]. Available: http://arxiv.org/abs/1707.01780

[34] X. Zhang and Y. LeCun, "Text Understanding from Scratch," Feb. 2015, [Online]. Available: http://arxiv.org/abs/1502.01710

[35] J. McAuley and J. Leskovec, "Hidden factors and hidden topics: Understanding rating dimensions with review text," in RecSys 2013 - Proceedings of the 7th ACM Conference on Recommender Systems, 2013, pp. 165–172. doi: 10.1145/2507157.2507163.

[36] Y. Bengio and Y. Grandvalet, "No Unbiased Estimator of the Variance of K-Fold Cross-Validation."

[37] M. Stone, "Cross-Validatory Choice and Assessment of Statistical Predictions," J R Stat Soc Series B Stat Methodol, vol. 36, no. 2, pp. 111–133, Jan. 1974, doi: 10.1111/j.2517-6161.1974.tb00994.x.

Understanding the relevance of parallelising machine learning algorithms using CUDA for sentiment analysis

Chai, Dakun Mang

2024-10-17

Chai DM, Moulitsas I, Bisandu DB. (2024) Understanding the relevance of parallelising machine learning algorithms using CUDA for sentiment analysis. In: Proceedings of the 2024 8th International Conference on Advances in Artificial Intelligence ICAAI 2024, 17 - 19 Oct 2024, London, United Kingdom, pp. 28-38

https://doi.org/10.1145/3704137.3704142