A Branch-and-Bound Approach to Efficient Classification and Retrieval of Documents

Kotaro Ii[1], Hiroto Saigo[1] and Yasuo Tabei[2]
[1]*School of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan*
[2]*Center for Advanced Intelligence Project, RIKEN, Tokyo, Japan*

Keywords: Text Classification, Text Retrieval, Suffix Tree, Branch and Bound, SEQL, fastText.

Abstract: Text classification and retrieval have been crucial tasks in natural language processing. In this paper, we present novel techniques for these tasks by leveraging the invariance of feature order to the evaluation results. Building on the assumption that text retrieval or classification models have already been constructed from the training documents, we propose efficient approaches that can restrict the search space spanned by the test documents. Our approach encompasses two key contributions. The first contribution introduces an efficient method for traversing a search tree, while the second contribution involves the development of novel pruning conditions. Through computational experiments using real-world datasets, we consistently demonstrate that the proposed approach outperforms the baseline method in various scenarios, showcasing its superior speed and efficiency.

1 INTRODUCTION

Text classification has been an important task in natural language processing, where it is used for text retrieval, spam filtering, sentiment analysis and opinion mining (Pang and Lee, 2008; Medhat et al., 2014). Great recent advances have been made using neural networks (Vaswani et al., 2017; Johnson et al., 2017). Among text classification methods, linear classifiers has long been popular due to their simplicity and interpretability (Joachims, 1998; Fan et al., 2008). Their properties are studied well, and known to often perform quite well if the correct features are provided (Wang and Manning, 2012). Moreover, they can easily scale to very large corpora (Agarwal et al., 2014). For instance, *fastText* has scaled linear classifiers so that a billion of words can be trained (Joulin et al., 2017). They also found that the use of N-grams has a potential to improve the classification accuracy by implicitly considering the local word orders. Although the use of N-grams as features confronts a combinatorial explosion problem, there exists efficient implementations such as *SEQL* (Ifrim et al., 2008; Ifrim and Wiuf, 2010).

In this paper, we introduce a simple yet efficient approach for evaluating text classification models by making use of the invariance of the feature order to the evaluation results. The proposed method has various applications, but the most useful applications would be; i) fast evaluation of document classification model, and ii) fast retrieval of the relevant documents. In computational experiments, we demonstrate the efficiency of our approach in evaluating new test documents in both classification and retrieval tasks. We employ two existing methods for building text classification models; 1) *SEQL* is a logistic regression-based classifier with L1/L2 norm regularization. A notable ability of *SEQL* is its efficient extraction of N-gram features by a branch-and-bound strategy. 2) *fastText* is an another linear text classifier, but is able to employ pre-trained word vectors in a similar way as in *word2vec* (Mikolov et al., 2013). We utilized three commonly used datasets (Review, Spam, EC) for computational evaluation and confirmed the enhanced efficiency achieved by our approach in all cases.

The contributions of our approach are the followings:

1. Proposal of an efficient way of traversing a search tree.
2. Development of new bounding conditions for information retrieval and classification.
3. Demonstration of the effectiveness in computational experiments using three real-world datasets.

Ii, K., Saigo, H. and Tabei, Y.
A Branch-and-Bound Approach to Efficient Classification and Retrieval of Documents.
DOI: 10.5220/0012310600003654
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 13th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2024)*, pages 205-214
ISBN: 978-989-758-684-2; ISSN: 2184-4313
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

This paper is organized as follows. Section 2 gives a brief background about linear models in text classification. In Section 3, we describe details about our proposed methods. In Section 4, we describe the datasets used in the experiments, and experimental conditions. Section 5 gives experimental results. Related works and the differences from our approach are described in Section 6. Section 7 summarizes the paper. Section 8 overviews the possible impact of this work, and Section 9 shows the limitation.

2 TEXT CLASSIFICATION WITH BAG OF WORDS

The bag of words (BoW) is a simple and an efficient approach for representing sentences, where a text is represented as the collection of its words. Typically, linear models such as linear / logistic regression or SVM are trained on the BoW representation. Consider a dataset be $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_i$ represents the i-th text represented in the bag of features, and $y_i \in \{0, 1\}$ is the corresponding class label. After training linear models, we can obtain a set of features $\boldsymbol{w}$ and the corresponding weights $\boldsymbol{\beta}$. The resulting model for predicting the response of a new text $\boldsymbol{x}_i$ can be represented as

$$f(\boldsymbol{x}_i) = \sum_{j=1}^{m} \beta_j I(w_j \in \boldsymbol{x}_i), \quad (1)$$

where $I(w_j \in \boldsymbol{x}_i)$ is an indicator function that returns 1 if w_j appears in x_i, and 0 otherwise. Here, m is the number of learned features. Equation (1) is readily used for regression or ranking problems, but can be used for a classification tasks, since plugging it into the sigmoid function $\sigma(\boldsymbol{x}) = \frac{e^{\boldsymbol{x}}}{1+e^{\boldsymbol{x}}}$ results in logistic regression. This representation can accommodate word embedding models of the form $\sigma(\boldsymbol{WEx})$), where $\boldsymbol{E}$ denotes a pre-trained word embedding matrix, $\boldsymbol{W}$ denotes the weight matrix specific to each task, and $\boldsymbol{x}$ denotes a one-hot vector. We can observe that once pre-trained word embedding $\boldsymbol{E}$ is obtained, then the model is linear in $\boldsymbol{W}$.

Below we consider a situation of evaluating the test dataset $\mathcal{D}$ by a learned linear function $f(.)$.

3 METHODS

In this section, we present our novel approach for efficiently evaluating linear models. Firstly, we introduce the proposed data structure based on a trie. Next, we describe our novel algorithms designed for the efficient evaluation of prediction models. Our focus is on document classification and retrieval, where we propose distinct algorithms with different pruning conditions.

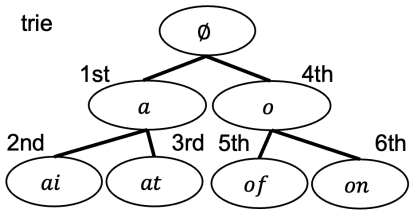

Figure 1: Illustration of a trie storing {a, ai, at, o, on, of}. The order of traversing the tree is shown next to each node.

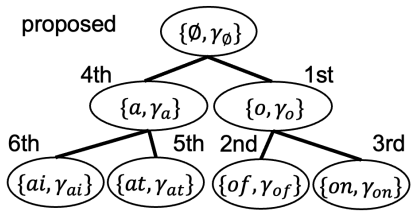

Figure 2: Illustration of our proposed trie. Apart from the original trie, each node stores not only a string, but also a real number γ. Based on the value of γ, the order of tree traversal changes. The order of traversing the tree is shown next to each node.

3.1 A Modified Trie Data Structure

A suffix tree, also known as a trie, is a tree-shaped data structure commonly used for string matching. In a trie, each node represents a string, and all the children of a node share a common prefix associated with the parent node. The root of the trie corresponds to the empty string. The basic trie data structure is shown in Figure 1. In our approach, each node in a trie not only stores a string but also a weight obtained from a classification algorithm (Figure 2). These weights play a crucial role in efficient classification. During the construction of a trie, we aggregate the weights of all the descendant nodes, and record it as γ. Then, while traversing the trie, we prioritize nodes with larger absolute γ. In this manner, we can efficiently traverse the part of a trie with the large weights, and terminate the traversal before visiting nodes with small weights. It therefore allows us to focus on the most relevant parts of the trie and avoid unnecessary traversals.

3.2 Document Classification

A naïve document classification algorithm is shown in Algorithm 1. For each data point, the algorithm examines whether each feature is present or not. If a feature

is included, the corresponding weight is summed up, as shown in Equation (1). At the end of evaluation, classification results are determined by the signs of the function values.

Data: $\boldsymbol{D}$, $\boldsymbol{w}$, $\boldsymbol{\beta}$
Result: $\boldsymbol{y}$
for $i = 1, \ldots, n$ **do**
  $s = 0$;
  for $j = 1, \ldots, m$ **do**
    if $w_j \in x_i$ **then**
      // string matching
    end
    $s \mathrel{+}= \beta_j$;
  end
  if $s \geq 0$ **then**
    $y_i = 1$;
  end
  else
    $y_i = 0$;
  end
end

Algorithm 1: Naïve document classifier.

An efficient document classifier is shown in Algorithm 2, in which we employ a modified trie data structure. The overall preprocessing step consists of summing up the weights and sorting the weights, together requires $O(m \log m)$.

Next, we elaborate on our novel tree pruning condition that can skip unnecessary search space. For a notational simplicity, we call a function value as a score s below, and the function value evaluated up to the j-th feature in the i-th document as s_{ij}, such that

$$s_{ij} = \sum_{k=1}^{j} \beta_k I(w_k \in \boldsymbol{x}_i)$$

.

Theorem 1. *Suppose that up to the j-th features are evaluated in the i-th document, and let the current estimate of score be s_{ij}. If the lowerbound of s_{ij} is strictly positive, then one can determine the prediction for the i-th document as positive, and safely skip the evaluation of the rest of the features, where the lowerbound is given as:*

$$s_{ij} + \sum_{\{k|j<k, \beta_k<0\}} \beta_k. \tag{2}$$

Proof. We split the score s_{ij} into the already evaluated part and the not yet evaluated part.

$$s_{ij} + \sum_{\{k|j<k\}} \beta_k \geq s_{ij} + \sum_{\{k|j<k, \beta_k<0\}} \beta_k.$$

Data: $\boldsymbol{D}$, $\boldsymbol{w}$, $\boldsymbol{\beta}$
Result: $\boldsymbol{y}$
$\beta^- = \sum_{\{k|\beta_k<0\}} \beta_k$;
$\beta^+ = \sum_{\{k|\beta_k>0\}} \beta_k$;
for $l = 1, \ldots, n$ **do**
  $\gamma_l^- = \sum_{\{k|\beta_k<0, k \in subtree(l)\}} \beta_k$;
  $\gamma_l^+ = \sum_{\{k|\beta_k>0, k \in subtree(l)\}} \beta_k$;
end
for $i = 1, \ldots, n$ **do**
  for $j = 1, \ldots, m$ **do**
    $need_{ij} =$ True;
  end
end
Sort($\boldsymbol{\beta}$, $key = abs(\gamma^-) + \gamma^+$);
for $i = 1, \ldots, n$ **do**
  $s = 0$;
  for $j = 1, \ldots, m$ **do**
    if $need_{ij} = False$ **then**
      continue;
    end
    if $w_j \in \boldsymbol{x}_i$ **then**
      $s \mathrel{+}= \beta_j$;
      if $\beta_j > 0$ **then**
        $\beta^+ \mathrel{-}= \beta_j$;
      end
      else
        $\beta^- \mathrel{+}= \beta_j$;
      end
    end
    else
      $\beta^+ \mathrel{-}= \gamma_j^+$;
      $\beta^- \mathrel{+}= \gamma_j^-$;
      for $k \in subtree(j)$ **do**
        $need_{kj} =$ False;
      end
    end
    if $s \geq 0$ *and Equation (2) holds* **then**
      $y_i = 1$;
      break;
    end
    if $s < 0$ *and Equation (3) holds* **then**
      $y_i = 0$;
      break;
    end
  end
end

Algorithm 2: Efficient document classifier.

Then the lowerbound can be obtained by discarding the positive weights. Once the lowerbound turns out to be strictly positive, then it is guaranteed that the prediction is positive without evaluating the rest of the

features. □

Similarly, if the upperbound below is strictly negative, then the prediction of the i-th document is guaranteed to be negative.

$$s_{ij} - \sum_{\{k|j<k,\beta_k>0\}} \beta_k. \tag{3}$$

3.3 Document Retrieval

A naïve algorithm for document retrieval is shown in Algorithm 3, where the goal is to retrieve the document with the smallest score. For each data point, the algorithm verifies whether a specific string (feature) is included or not. If a string (feature) is included, then the corresponding weight is summed up.

Data: $\boldsymbol{D}$, $\boldsymbol{w}$, $\boldsymbol{\beta}$
Result: i^*
$\mathbf{s} = \mathbf{0}$
for $j = 1, \ldots, m$ **do**
  for $i = 1, \ldots, n$ **do**
    if $w_j \in \boldsymbol{x}_i$ **then**
      $s_i \mathrel{+}= \beta_j$;
    end
  end
end
$i^* = \arg\min_i \mathbf{s}$;
return i^*;

Algorithm 3: Naïve document retriever.

The proposed efficient document retrieval algorithm is shown in Algorithm 4. Similarly to the document classification case, we employ a modified trie as a basic data structure (Figure 2), and restrict the search space by employing pruning conditions.

Theorem 2. *Suppose that up to the j-th features are evaluated, and the current minimum score s^*_{lj} is identified in the l-th document . If the following condition is satisfied, then one can safely discard the document i for further evaluation without losing the optimality;*

$$s^*_{lj} + \sum_{\{k|j<k,\beta_k>0\}} \beta_k < s_{ij} + \sum_{\{k|j<k,\beta_k<0\}} \beta_k. \tag{4}$$

Proof. Similarly to the document classification case, we split s_{ij} into the already evaluated part and the not yet evaluated part, and estimate the lowerbound as:

$$s_{ij} + \sum_{\{k|j<k\}} \beta_k \geq s_{ij} + \sum_{\{k|j<k,\beta_k<0\}} \beta_k. \tag{5}$$

On the other hand, the upperbound of the current minimum score s^*_{lj} with respect to the i-th document can

Data: $\boldsymbol{D}$, $\boldsymbol{w}$, $\boldsymbol{\beta}$
Result: i^*
$\beta^- = \sum_{\{k|\beta_k<0\}} \beta_k$;
$\beta^+ = \sum_{\{k|\beta_k>0\}} \beta_k$;
$\gamma^-_l = \sum_{\{k|\beta_k<0, k\in subtree(l)\}} \beta_k$;
$\gamma^+_l = \sum_{\{k|\beta_k>0, k\in subtree(l)\}} \beta_k$;
for $i = 1, \ldots, n$ **do**
  for $j = 1, \ldots, m$ **do**
    $need_{ij} =$ True;
  end
end
Sort($\boldsymbol{\beta}$, $key = abs(\gamma^-) + \gamma^+$);
$\mathbf{s} = \mathbf{0}$;
$\mathbf{I} = \{1, 2, \ldots, n\}$;
for $j = 1, \ldots, m$ **do**
  $s^* = \infty$;
  for $i \in \mathbf{I}$ **do**
    if $need_{ij} = False$ **then**
      continue;
    end
    else if $w_j \in \boldsymbol{x}_i$ **then**
      $s_i \mathrel{+}= \beta_j$;
      if $\beta_j > 0$ **then**
        $\beta^+ \mathrel{-}= \beta_j$;
      end
      else
        $\beta^- \mathrel{+}= \beta_j$;
      end
    end
    else
      $\beta^+ \mathrel{-}= \gamma^+_j$;
      $\beta^- \mathrel{+}= \gamma^-_j$;
      for $k \in subtree(j)$ **do**
        $need_{kj} =$ False;
      end
    end
    if $s^* > s_i$ **then**
      $s^* = s_i$;
      $i^* = i$;
    end
  end
  for $i \in \mathbf{I}$ **do**
    if *Equation (4) holds* **then**
      remove i from $\mathbf{I}$;
    end
  end
  if $len(\mathbf{I}) == 1$ **then**
    $i^* = i$;
    Break;
  end
end
return i^*;

Algorithm 4: Efficient document retriever.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

be estimated in a similar fashion as:

$$s^*_{lj} + \sum_{\{k|j<k\}} \beta_k \leq s^*_{lj} + \sum_{\{k|j<k,\beta_k>0\}} \beta_k. \quad (6)$$

If the lowerbound of s_{ij} turns out to be larger than the upperbound of s^*_{lj}, the it is guaranteed that the i-th document never obtains smaller score than that of the l-th document, so one can skip the further evaluation of the i-th document without losing optimality. □

The time complexity for the proposed approach remains unchanged from that of text classification and remains $O(mlogm)$. Note that it is straightforward to extend our approach to retrieve the top-k documents instead of retrieving the best document.

4 EXPERIMENTS

In this section, we demonstrate the effectiveness of the proposed approach in three real-world datasets. Our proposed approach can be applied to any linear model, and enjoy the improved efficiency, however, in the experiments below, we employ *SEQL* (Ifrim et al., 2008; Ifrim and Wiuf, 2010) [1] and *fastText* (Joulin et al., 2017) [2] due to their public availability and rich features in text modeling.

4.1 Text Modeling by SEQL

SEQL is a linear model, and its objective function is given as

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \xi(y_i, x_i, \boldsymbol{\beta}) + C\alpha\|\boldsymbol{\beta}\|_1 + C(1-\alpha)\|\boldsymbol{\beta}\|_2, \quad (7)$$

where C and α are regularization parameters, and $\xi(.)$ denotes a loss function chosen either from hinge loss or logistic loss. In our experiments, we set C to 1 and α to 0. In addition, we choose logistic loss.

4.2 Text Modeling by fastText

fastText is a popular library for text classification. In *fastText*, the possibility that the i-th data is classified to a class c is given as $p(y_i = c) = softmax_c(\boldsymbol{WEx}_i)$, where $\boldsymbol{W}$ is weight matrix and $\boldsymbol{E}$ is a pre-trained word embedding matrix. The model is trained to minimize the negative log-likelihood over the classes;

$$\min_{\boldsymbol{W}} -\frac{1}{N} \sum_{i=1}^{n} y_n log(f(\boldsymbol{WEx}_i)) \quad (8)$$

[1] Available from https://github.com/heerme/seql-sequence-learner

[2] Available from https://fasttext.cc/

4.3 Dataset

The first dataset used in the experiment is the IMDB movie review dataset. The label indicates whether the text is positive or negative. There are 25,000 movie review texts with positive and negative contents, respectively. The second dataset used in the experiment is spam ham dataset available on Kaggle. The label indicates whether the text is spam or ham. The data size is 5171, and there are 1499 spam texts and 3672 ham texts. The third dataset used in the experiment is EC dataset available on kaggle. The label indicates whether the text is about household or books. The data size is 31133, and there are 19313 texts about household and 11820 texts about books.

4.4 Settings

For matching strings in texts, we employed a popular Boyer-Moore (BM) algorithm (Boyer and Moore, 1977). However, one can alternatively employ Knuth-Morris-Pratt (KMP) (Knuth et al., 1977) as well. Experiments were conducted to evaluate the pruning efficiency and time savings achieved by the proposed method. So we investigate the extent to which the method could efficiently prune branches within the search space and quantify the resulting reduction in processing time. In order to investigate the effect of the size of the dataset, we fixed the number of features to 1000, and selected {20, 40, 60, 80, 100} samples randomly from each dataset. We ran the above procedures 100 times in each condition, and the averages are reported.

5 RESULTS

Table 1 summarizes the efficiency gained for document classification problem. We can confirm that our pruning strategy was effective in all the datasets. It appears that *fastText* slightly benefits more from pruning than *SEQL*. Table 2 summarizes the efficiency gained for document retrieval problem. We can again confirm the effectiveness of our approach in all the datasets. By comparing Tables 1 and 2, we observed that that the document retrieval task benefits slightly more than the classification task.

In the following subsections, we discuss the results obtained by each method in each dataset into details.

Table 1: Pruning efficiency in document classification problem.

	SEQL				fastText			
	With Pruning		Without Pruning		With Pruning		Without Pruning	
	Traversed[%]	time[s]	Traversed[%]	time[s]	Traversed[%]	time[s]	Traversed[%]	time[s]
review	85.1	5935.8	100	6798.9	67.2	6770.2	100	9632.0
spam	73.8	374.1	100	485.9	68.4	765.5	100	984.9
EC	86.9	2282.9	100	2538.3	66.3	3831.7	100	5655.7

Table 2: Pruning efficiency in document retrieval problem.

	SEQL				fastText			
	With Pruning		Without Pruning		With Pruning		Without Pruning	
	Traversed[%]	time[s]	Traversed[%]	time[s]	Traversed[%]	time[s]	Traversed[%]	time[s]
review	71.6	5150.1	100	6717.8	59.6	6279.6	100	9911.8
spam	60.0	334.9	100	485.9	64.6	706.9	100	925.1
EC	82.8	2246.9	100	2500.8	57.0	3458.5	100	5636.6

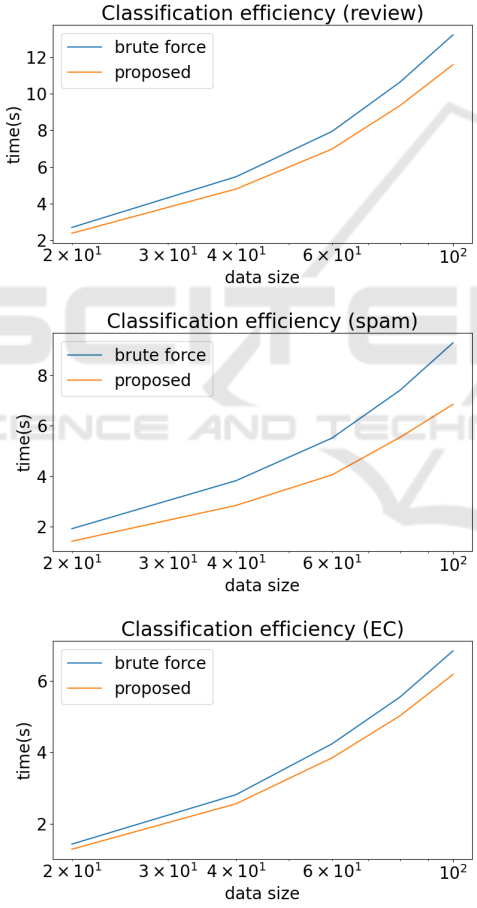

Figure 3: Rise of the execution time in classifying documents modeled by *SEQL* with respect to the increase in the dataset size in the review dataset (top), the spam dataset (middle) and the EC dataset (bottom).

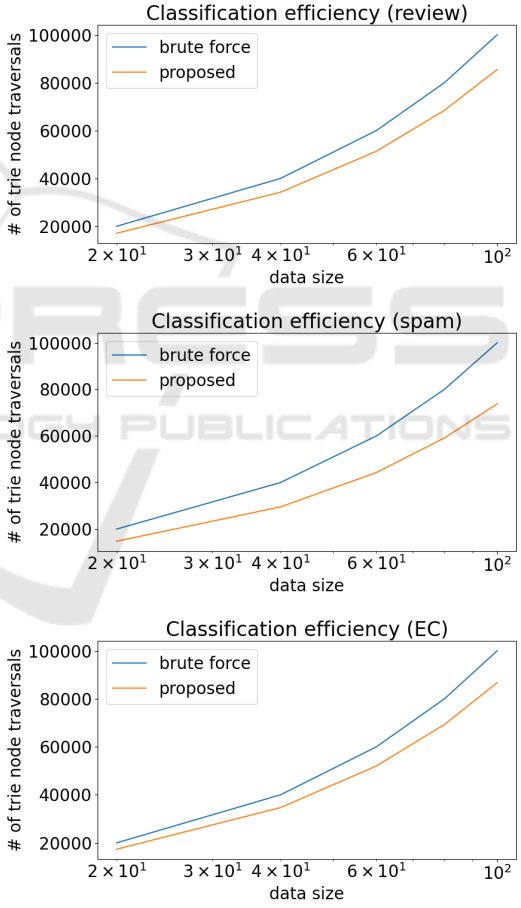

Figure 4: Rise of the number of the prunings in classifying documents modeled by *SEQL* with respect to the increase in the dataset size in the review dataset (top), the spam dataset (middle) and the EC dataset (bottom).

5.1 Document Classification

Figure 3 and Figure 4 shows the rise of the execution time and the rise of the number of prunings, respectively, in classifying documents modeled by *SEQL* with respect to the increase in the dataset size in three real datasets. The corresponding results for *fastText*

are shown in Figure 7 and Figure 8. We can observe in all the case that our pruning approach was more efficient than the brute-force method over varying dataset sizes.

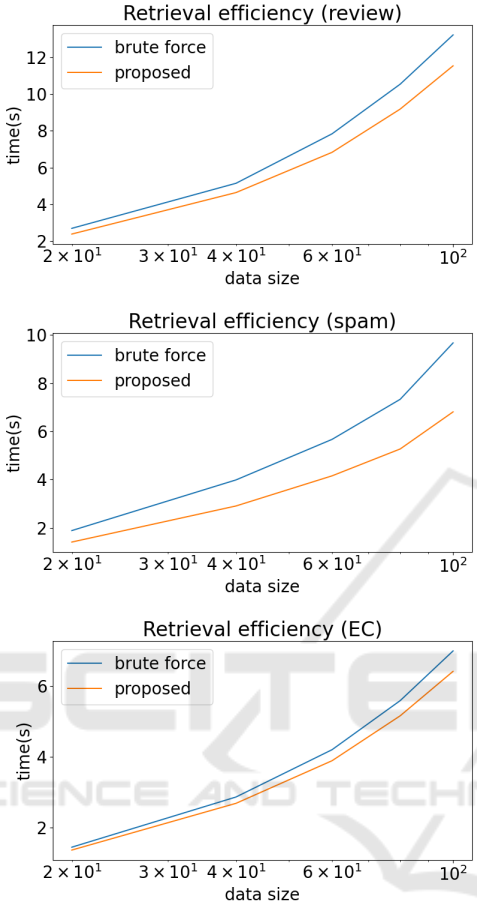

Figure 5: Rise of the execution time in retrieving the best document modeled by *SEQL* with respect to the increase in the dataset size in the review dataset (top), the spam dataset (middle) and the EC dataset (bottom).

5.2 Document Retrieval

Figure 5 and Figure 6 shows the rise of the execution time and the rise of the number of prunings, respectively, in retrieving documents modeled by *SEQL* with respect to the increase in the dataset size in three real datasets. The corresponding results for *fastText* are shown in Figure 9 and Figure 10. We can observe in all the case that our pruning approach was more efficient than the brute-force method over varying dataset sizes. This observation suggests that the proposed method enjoys greater effectiveness when applied to larger datasets.

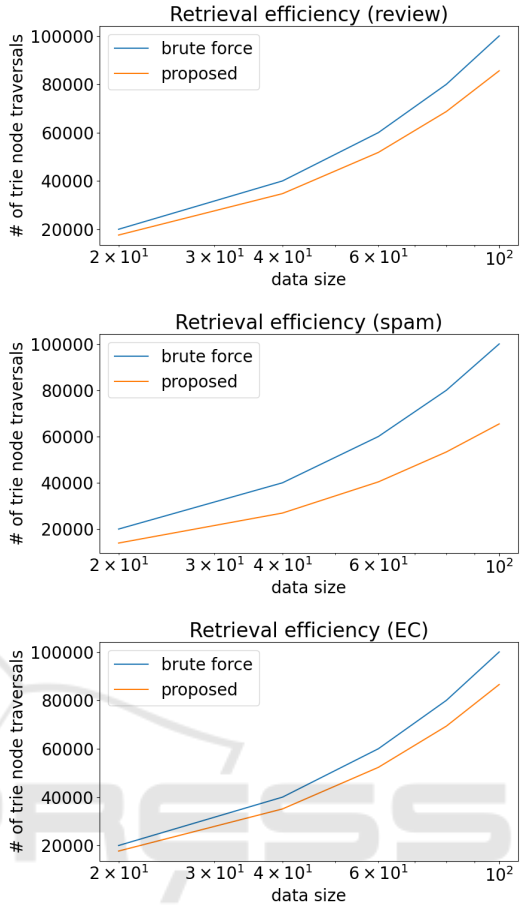

Figure 6: Rise of the number of the prunings in retrieving the best document modeled by *SEQL* with respect to the increase in the dataset size in the review dataset (top), the spam dataset (middle) and the EC dataset (bottom).

6 RELATED WORKS

In this subsection, we first give related works and elaborate on the difference from this work. Retrieval of top-k documents has been studied in the information retrieval (Ding and Suel, 2011; Dimopoulos et al., 2013). In this context, our approach is close to a Term-At-A-Time (TAAT) traversal with early termination. However, there exists substantial differences. First of all, in information retrieval, the goal is to retrieve relevant documents as fast as possible, so the documents are pre-processed such that the appearance of key terms in each document is recorded in inverted lists. Although it plays a key role in fast information retrieval, not-yet processed documents are kept out of the scope. On the other hand, our approach has a built-in string matching engine, which enables us to handle raw streaming data such as those in SNS. As a result, one can pre-train a classifier using hate speech dataset,

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

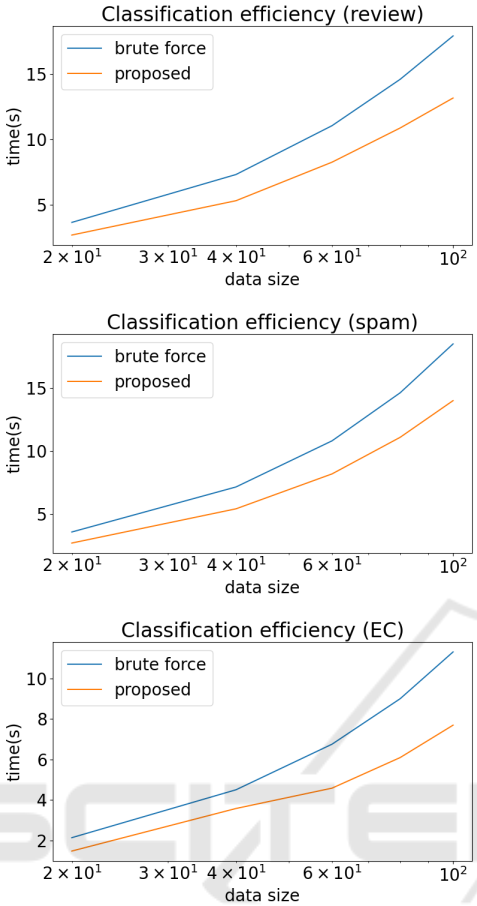

Figure 7: Rise of the execution time in classifying documents modeled by *fastText* with respect to the increase in the dataset size in the review dataset (top), the spam dataset (middle) and the EC dataset (bottom).

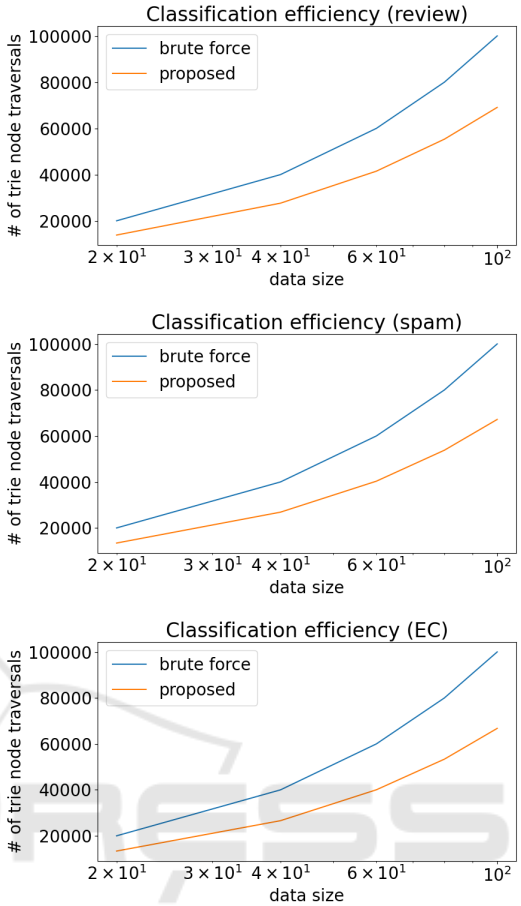

Figure 8: Rise of the number of the prunings in classifying documents modeled by *fastText* with respect to the increase in the dataset size in the review dataset (top), the spam dataset (middle) and the EC dataset (bottom).

then use it to identify suspicious posts quickly. In addition to the aforementioned contributions, we also proposed an efficient method for classifying all the documents, which sets our approach apart from existing methods in text retrieval.

7 CONCLUSION AND DISCUSSION

In this paper, we have proposed efficient approaches for document classification and document retrieval.Our approach encompasses two key contributions. The first one introduces an efficient method for traversing a search tree, while the second one involves the development of novel pruning conditions. Computational experiments using the real world dataset confirmed the effectiveness of the pruning consistently over three real-world datasets. One of our future work is to investigate the effectiveness of our approach when applied to pre-trained word embedding model, especially in very large datasets. Lastly, our approach is applicable to any linear model and its extensions. Examples include logistic regression (Harrell, 2001; Hosmer and Lemeshow, 2000; Chen et al., 2017), linear SVM (Demiriz et al., 2002), naive Bayes (Kim et al., 2000), decision tree, and boosted linear model (Mamitsuka and Naoki, 1998; Schapire and Singer, 2000).

8 IMPACT OF THIS WORK

The evaluation of a linear model is indeed a fundamental task in machine learning, and our approach can be beneficial for various learning tasks. Examples of these tasks include regression, classification, and ranking. Our method is particularly useful for lin-

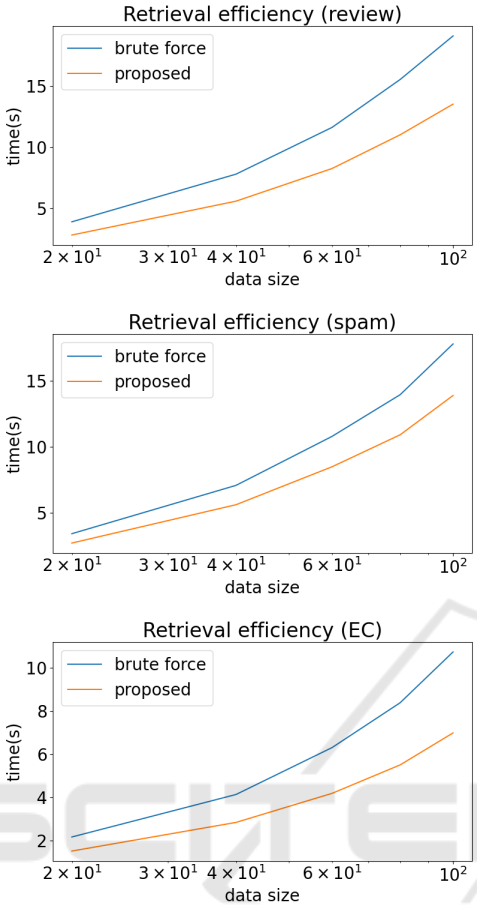

Figure 9: Rise of the execution time in retrieving the best document modeled by *fastText* with respect to the increase in the dataset size in the review dataset (top), the spam dataset (middle) and the EC dataset (bottom).

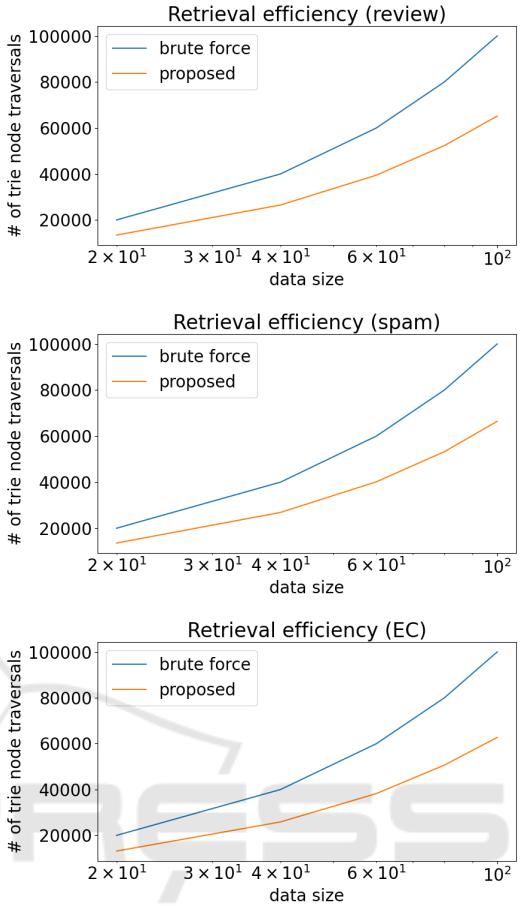

Figure 10: Rise of the number of the prunings in retrieving the best document modeled by *fastText* with respect to the increase in the dataset size in the review dataset (top), the spam dataset (middle) and the EC dataset (bottom).

ear models where the features have structures such as texts, trees, and graphs, as evaluating such models requires repeated matching of structures. The benefits of our approach would be maximized when evaluating graphs, as it involves solving the subgraph isomorphism problem (Kudo et al., 2005).

9 LIMITATION

In this paper, the focus of the document classification problem was restricted to a binary case. While it is relatively straightforward to extend our approach to handle the multi-class case, the efficiency benefits in that scenario may be less pronounced. The same phenomena would likely be observed for the top-k document retrieval problem, especially when dealing with a large value of k.

REFERENCES

Agarwal, A., Chapelle, O., Dudikib, M., and Langford, J. (2014). A reliable effective terascale linear learning system. *Journal of Machine Learning Research*, 15:1111—-1133.

Boyer, R. and Moore, J. S. (1977). A fast string searching algorithm. *Comm. ACM. New York: Association for Computing Machinery*, 20(10):762—-772.

Chen, W., Xie, X., Wang, J., Pradhan, B., Hong, H., Bui, D., Duan, Z., and Ma, J. (2017). A comparative study of logistic model tree, random forest, and classification and regression tree models for spatial prediction of landslide susceptibility. *Catena*, 151:147—160.

Demiriz, A., Bennet, K., and Shawe-Taylor, J. (2002). Linear programming boosting via column generation. *Machine Learning*, 46(1-3):225–254.

Dimopoulos, C., Nepomnyachiy, S., and Suel, T. (2013). Faster top-k document retrieval using block-max in-

dexes. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM)*.

Ding, S. and Suel, T. (2011). Faster top-k document retrieval using block-max indexes. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

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

Harrell, F. (2001). *Ordinal logistic regression. (In Regression Modeling Strategies)*. Springer:Berlin/Heidelberg,Germany.

Hosmer, D. and Lemeshow, S. (2000). *R.X. Applied Logistic Regression*. John Wiley and Sons: Hoboken, NJ, USA.

Ifrim, G., Bakir, G., and Weikum, G. (2008). Fast logistic regression for text categorization with variable-length n-grams. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*.

Ifrim, G. and Wiuf, C. (2010). Bounded coordinate-descent for biological sequence classification in high dimensional predictor space. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *In Proceedings of the 10th European Conference on Machine Learning*, pages 137—-142.

Johnson, M., Schuster, M., Le, Q., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., Hughes, M., and Dean, J. (2017). Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, pages 339–351.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2017). Bag of tricks for efficient text classification. In *In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*.

Kim, Y., Hahn, S., and Zhang, B. (2000). Text filtering by boosting naive bayes classifiers. In *In Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Knuth, D., Morris, J., and Pratt, V. (1977). Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323—-350.

Kudo, T., Maeda, E., and Matsumoto, Y. (2005). An application of boosting to graph classification. In *Advances in Neural Information Processing Systems 17*, pages 729–736. MIT Press.

Mamitsuka, H. and Naoki, A. (1998). Query learning strategies using boosting and bagging. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML98)*.

Medhat, W., Hassan, A., and Korashy, H. (2014). Sentiment analysis algorithms and applications: A survey. *Ain Sham Engineering Journal*, 5:1093—-1113.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *In Proceedings of the 1st International Conference on Learning Representations*.

Pang, B. and Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1—135.

Schapire, R. and Singer, Y. (2000). Boostexter: A boosting-based system for text categorization. *Mach. Learn.*, 39:135—-168.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *In Proceedings of the 31th Conference on Neural Information Processing Systems*.

Wang, S. and Manning, C. (2012). Baselines and bigrams: Simple, good sentiment and topic classification. In *In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 90—94.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS