Research Article Open Access

Raji Ghawi* and Jürgen Pfeffer

Efficient Hyperparameter Tuning with Grid Search for Text Categorization using kNN Approach with BM25 Similarity

https://doi.org/10.1515/comp-2019-0011
Received February 20, 2019; accepted July 18, 2019

Abstract: In machine learning, hyperparameter tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. Several approaches have been widely adopted for hyperparameter tuning, which is typically a time consuming process. We propose an efficient technique to speed up the process of hyperparameter tuning with Grid Search. We applied this technique on text categorization using kNN algorithm with BM25 similarity, where three hyperparameters need to be tuned. Our experiments show that our proposed technique is at least an order of magnitude faster than conventional tuning.

Keywords: hyperparameter tuning, text categorization, grid search, kNN, BM25

1 Introduction

In machine learning, a hyperparameter is a parameter whose value is set before the learning process begins. Hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. It aims at finding a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data. The objective function takes a tuple of hyperparameters and returns the associated loss [1]. Cross-validation is often used to estimate this generalization performance [2]. For decades, the de-facto standard for hyperparameter optimization in machine learning has been grid search, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. Other approaches proposed over the years include random search [2], gradient-based optimization [3, 4], and Bayesian optimization [5–7]. Despite these efforts, grid search prevails as the state of the art to hyperparameter optimization due to its simplicity to implement and parallelization, and its reliability in low dimensional spaces.

Grid search exhaustively enumerates all combinations of hyperparameters and evaluates each combination. Depending on the available computational resources, the nature of the learning algorithm and size of the problem, each evaluation may take considerable time. Thus, the overall optimization process is time consuming. This leads to an increasing need for efficient methods to optimize hyperparameters. Most of proposed optimization methods tend to reduce the number of evaluations. In this work, we address the efficiency of grid search from another perspective, where we propose to reduce the time of each evaluation instead of reducing the number of evaluations. Our use case is text categorization using k-NN algorithm (k nearest neighbor) with BM25 similarity measure [8, 9]. Text Categorization is a supervised learning task, defined as assigning category labels to new documents based on the likelihood suggested by a training set of labeled documents [10]. k-NN is a well known statistical approach that has been applied to text categorization [11–13] and has been shown to be one of the top performing methods. BM25 similarity measure represents the state-of-the-art TF-IDF-like retrieval functions used in document retrieval. The problem of text categorization using k-NN and BM25 has three hyperparameters: k and b, the free parameters of BM25 similarity measure, and t, the number of top-ranking neighbors to predict the category of an input document.

The contributions of this paper are as follows:

1. We propose an efficient method, called Fast Tuning, for hyperparameter tuning with grid search for text categorization using k-NN approach using BM25 similarity.
2. We present detailed algorithmic procedures to describe our proposed tuning method, as well as the conventional tuning method.

***Corresponding Author: Raji Ghawi:** Technical University of Munich, Germany, E-mail: raji.ghawi@tum.de
Jürgen Pfeffer: Technical University of Munich, Germany, E-mail: juergen.pfeffer@tum.de

3. We describe how our proposed tuning method is applied with cross-validation.
4. We conduct several experiments to evaluate the performance of our proposed tuning method comparing to conventional tuning, in terms of speed-up and time reduction factors.
5. Our experiments show that the proposed tuning method outperforms the conventional tuning method, as it is by at least one order of magnitude faster.

The paper is orgarnized as follows. Section 2 presents foundations and related works. In section 3, we present the conventional tuning method, where in Section 4, we present our proposed tuning method. Section 5 describes how our Fast Tuning method can be applied with cross-validation. Conducted experiments are presented in Section 6. Section 7 is dedicated for discussion and future work, whereas Section 8 concludes the paper.

2 Foundations and related works

2.1 Hyperparameter optimization

In order to formally describe the hyperparameter optimization problem, we adopt the formulation given in [14]. Given a machine learning algorithm A having hyperparameters $\lambda_1, \cdots, \lambda_n$ with respective domains $\Lambda_1, \cdots, \Lambda_n$, we define its hyperparameter space as $\Lambda = \Lambda_1 \times \cdots \times \Lambda_n$. For each hyperparameter setting $\lambda \in \Lambda$, we use A_λ to denote the learning algorithm A using this setting. We use $l(\lambda) = L(A_\lambda, D_{train}, D_{valid})$ to denote the validation loss that A_λ achieves on data D_{valid} when trained on D_{train}. The hyperparameter optimization problem is then to find $\lambda \in \Lambda$ minimizing $l(\lambda)$. Each objective function evaluation requires evaluating the performance of a model trained with hyperparameters λ. Depending on the available computational resources, the nature of the learning algortihm A and the problem size (D_{train}, D_{valid}) each evaluation may take considerable time.

The traditional way of performing hyperparameter optimization has been *grid search*, which is simply an exhaustive searching through a specified subset of the hyperparameter space Λ of a learning algorithm A. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set.

Other approaches proposed over the years include random search [2], gradient search [3, 4, 15–17], and Bayesian optimization [5–7]. *Random search* replaces the exhaustive enumeration of all parameter combinations by selecting them randomly. This can be simply applied to the discrete setting described above, but also generalizes to continuous and mixed spaces. It can outperform Grid search, especially when only a small number of hyperparameters affects the final performance of the machine learning algorithm [2].

Bayesian optimization [18, 19] is a global optimization method for noisy black-box functions. Applied to hyperparameter optimization, Bayesian optimization builds a probabilistic model M of the function f mapping from hyperparameter values to the objective evaluated on a validation set D_{valid}. By iteratively evaluating a promising hyperparameter configuration λ based on the current model, and then updating it, Bayesian optimization, aims to gather observations about this function and the location of the optimum. The three most popular implementations of Bayesian optimization are Spearmint [7], which uses a Gaussian process (GP) model for M; SMAC [20], which uses random forests modified to yield an uncertainty estimate; and the Tree Parzen Estimator (TPE) [6], which constructs a density estimate over good and bad instantiations of each hyperparameter to build M. In practice, Bayesian optimization has been shown [6, 7, 20, 21] to obtain better results in fewer evaluations compared to grid search and random search, due to the ability to reason about the quality of experiments before they are run.

Gradient-based optimization [3, 4, 15–17] can be applied for specific learning algorithms, where it is possible to compute the gradient with respect to hyperparameters and then optimize the hyperparameters using gradient descent. The first usage of these techniques was focused on neural networks.[16] Since then, these methods have been extended to other models such as support vector machines [3] or logistic regression [4].

There are several reasons why *grid search* prevail as the state of the art [2] despite decades of research into global optimization and the several proposed hyperparameter optimization approaches:
- it is simple to implement and parallelization is trivial;
- it typically finds a better $\widehat{\lambda}$ than purely manual sequential optimization (in the same amount of time);
- it is reliable in low dimensional spaces.

2.2 Text categorization

Text categorization (or classification) is a supervised learning task, defined as assigning category labels to new documents based on the likelihood suggested by a training set of labeled documents [10]. Text classification finds applications in a wide variety of domains in text mining.

Some examples of domains in which text classification is commonly used include: News filtering and organization, document organization and retrieval, opinion mining, and email classification and spam filtering [22]. There are several variants of text categorization problem. *Binary categorization* consider only two categories, and is commonly used in tasks such as: information retrieval (relevant vs. non-relevant documents), spam filtering (spam vs. non-spam), and opinion mining (positive vs. negative). On the other hand, *multi-class categorization* consider more than two categories. It is commonly used in topic categorization (e.g., sports, science, travel, business, etc), and email routing. Other variants include *hierarchical categorization*, where categories form a hierarchy.

An increasing number of learning approaches have been applied on text categorization, including regression models [10], nearest neighbor classification (k-NN) [23–26], Bayesian probabilistic approaches [27–29], decision trees [30, 31], neural networks [32–35] and Support Vector Machines [36–38]. Among others, k-NN is one of the top-performing approaches.

2.3 Nearest neighbor classification

The k-Nearest Neighbor (k-NN) is a well known similarity-based learning approach that has been shown to be very effective for a variety of problem domains including text categorization [39]. The k-NN approach has been intensively studied in pattern recognition [40], and has been applied to text categorization since the early days of its research [11–13]. It is known to be one of the most effective methods [10, 26]. The k-NN algorithm is quite simple, it uses the training examples as a basis for computing similarity. For a test document d to be classified, the algorithm finds k examples in the training set that are most similar to d (k nearest neighbors), and this forms a neighborhood of d. Majority voting among the documents in the neighborhood is used to decide the classification for d, that is, the algorithm assigns to d the category that is most common in the neighbor documents.

The algorithm can be improved by considering the distance of a neighbor (a closer neighbor has more influence). That is, the similarity score of each neighbor document to the test document is used as the weight of the categories of the neighbor document. If several of the k nearest neighbors share a category, then the per-neighbor weights of that category are added together, and the resulting weighted sum is used as likelihood score of that category with respect to the test document. By sorting the scores of candidates categories, a ranked list is obtained for the test document, and the top-ranking category is assigned as the correct category [10, 22, 41]. However, to apply k-NN we need to choose an appropriate value for k, and the success of classification is very dependent on this value. kNN is a lazy learning method as no model needs to be built (learned) and nearly all computation takes place at the classification stage. This prohibits it from being applied to areas where dynamic classification is needed for a large repository [26]. Moreover, to apply kNN we need a similarity function to measure similarity of two text documents.

2.4 VSM, TF.IDF, and BM25

The "Vector Space" model (VSM) is a text representation model commonly used in Information Retrieval [42]. In this vector model, each document is a vector and its element corresponds to words in the whole document collection. Assume a vocabulary V of atomic terms that appear in each document, a text object (sentence, paragraph, or document) is represented as a vector of real numbers $\mathbf{v} \in \mathbb{R}^{|V|}$, where each component corresponds to a term $w \in V$.

Basically, the values in the vector can be binary, 1 for presence of the word and 0 for absence of the word. Alternatively, it is common to take the within document term frequency (TF), the number of occurrences of the given term in the given document, into consideration as a basic factor for weighting documents: $TF_{w,d} = c(w, d)$. For the similarity between documents, several measures are available including 1-norm, 2-norm, cosine measure, and dot product [42]. Using dot product, the similarity between a query $\mathbf{q}$ and a document $\mathbf{d}$ is given by:

$$sim(q, d) = \mathbf{v_q}.\mathbf{v_d} = \sum_{w \in q \cap d} c(w, q)c(w, d)$$

However, the general idea behind the vector-space representation is that the magnitude of a component should be related to the "importance" of w in the document represented by $\mathbf{v}$, with higher weights assigned to terms that are frequent in the document and infrequent in the collection as a whole. Therefore, TF is typically combined with the *Inverse Document Frequency*, IDF, which is given by:

$$idf(w) = log\left(\frac{M - df(w) + 0.5}{df(w) + 0.5}\right) \quad (1)$$

where M is the number of document in the collection, and $df(w)$ is the document frequency of term w (count of documents containing w). The TF.IDF scheme is the most popularly used scheme in Information Retrieval [42, 43]. However, another effective way to assign weights to terms when representing a document as a weighted term vector is the popular term weighting method BM25 developed by Robertson et al. [8, 9]. BM25 (and its variants, e.g. BM25F) represent state-of-the-art TF-IDF-like retrieval functions used in document retrieval. Mainly, BM25 extends TF.IDF model in two aspects: 1) term frequency transformation, and 2) document length normalization.

Term frequency transformation

In order to control the influence of high weights, while retaining the influence of small counts, term frequency tf is computed using the transformation function:

$$tf = \frac{(k+1).c(w,d)}{k + c(w,d)}$$

where parameter $k > 0$ is used to control the upper bound which improves the robustness as it avoids overly rewarding the matching of any particular term.

Document length normalization

The term frequency is dependent on the document length, i.e. the number of tokens in a document, and needs to be normalized [44, 45] for two reasons: 1) the same term usually occurs repeatedly in long documents; 2) long document has usually a large size of vocabulary. As a consequence, a weighting model without employing normalization could produce biased weights with respect to the document length, favouring long documents. A classical method of document length normalization is the pivoted normalization approach proposed by Singhal et. al. [44], given by:

$$1 - b + b\,\frac{|d|}{avgdl}$$

where $|d|$ is the length of document d in words, $avgdl$ is the average document length over the text collection, and $b \in [0, 1]$ is a free controlling parameter.

In summery, BM25 combines the previous ingredients together, in the following similarity function:

$$sim(q,d) = \sum_{w} c(w,q) \frac{(k+1)\,c(w,d)}{c(w,d) + k(1 - b + b\frac{|d|}{avgdl})} .idf(w) \tag{2}$$

where $c(w, d)$ and $c(w, q)$ are the term frequencies of term w in query q and document d respectively, $|d|$ the length of document d, $avgdl$ is the average document length, $idf(w)$ is the inverse document frequency of term w (see equation (1)), and k and b are the BM25 parameters: k controlling term frequency saturation rate and b controlling document length normalization. Note that $k \geq 0$ and $0 \leq b \leq 1$.

Although most of the work in information retrieval has focused on how to assess the similarity of a keyword query and a text document, rather than the similarity between two documents, many weighting schemes, including BM25, can also be applied as similarity functions between two documents. For instance, in a text categorization task using kNN approach, BM25 can be applied as a similarity measure where the test document being classified is considered as a query, and compared to training documents collection. A number of papers [46–49] have used BM25 as a similarity measure for the kNN classification approach in a number of different application domains. Moreover, a number of papers [45, 50, 51] have addressed parameter tuning for BM25. In the next section, we address in details the hyperparameter tuning problem for kNN approach using BM25 similarity.

3 Conventional tuning

For simplicity, we consider multi-class single-label classification (each document belongs to exactly one category). We assume that our approach can be generalized to more complex classification tasks (multi-label, hierarchical) with appropriate modifications. Having a training set of documents, we want to train a k-NN classifier for text categorization using BM25 similarity measure. However, we have three free parameters to which we need to set values, namely: k and b parameters of BM25 similarity measure; and t the parameter of nearest neighbor algorithm[1]

The choice of those parameters affect the performance of the classifier, and thus there values should be selected carefully. The main way for selecting best values is to try a wide range of values for each parameter and test the performance. This process is called parameter tuning or optimization. To tune the parameters we need: 1) a validation set of documents: to test the performance, and 2) a set/range of possible values for each parameter: to test the different combinations of parameters.

In the following sections we will use the following notations. The training and validation sets of documents of

1 Traditionally, the parameter of nearest neighbor algorithm is denoted k, hence it is so-called k-NN; but in this paper we instead denote it t to avoid confusion with k parameter of BM25.

a classification task are denoted D_{train} and D_{valid}, respectively. Moreover, the set of ground truth categories of the validation set is denoted G_{valid}. We use f to denote the chosen performance measure used for validation, e.g., F1 measure. We use K, B, and T to denote the sets from which the hyperparameters k, b, and t are drawn, respectively. Finally, we assume the following procedures are available:

– `top`(S, t): this procedure sorts S in ascending order and returns the top t values.
– `vote`(S): counts the occurrence frequency of the elements in S, and returns the most recurrent element.
– `BM25`(d, d', k, b): this procedure calculates BM25 similarity between two documents d and d' using parameters k and b.
– `related`(d, D): returns a subset of training documents D that are related to an input document d (two documents are related if they share at least one word).
– `eval`(P, G): evaluates a set of predictions P against a ground truth set G. This can be any typical performance metric e.g., Recall, Precision, F-measure, or Accuracy; however, we used F-measure in our experiments (Section 6).

In the following, we explain k-NN classification approach using two simple procedures: `applyKNN` and `classify`.

3.1 `applyKNN`

Given a training and a validation sets of documents, D_{train} and D_{valid}, respectively, text categorization aims at assigning for each document $d \in D_{valid}$ a category c. To do so, k-NN approach seeks to find, for $d \in D_{valid}$, the top-t similar documents in D_{train}. We consider BM25 similarity that rely on two parameters k and b. This approach is illustrated using the following procedure `applyKNN` which takes as input: the training and validation sets of documents: D_{train}, D_{valid}, and three parameters: k, b and t. This procedure returns as output a set of predictions p whose elements are (document, category) pairs: (d, c) where c is the predicted category of document $d \in D_{valid}$. The predictions set p is initially empty. For each document d in the validation set D_{valid}, the procedure `classify` is used to find a predicted category c. The pair (d, c) is then appended to p.

3.2 `classify`

The `classify` procedure is the core component of k-NN approach. It takes as input: a document d, the training set D_{train}, and the three parameters k, b and t. It returns as output: a category c predicted for document d.

First, the procedure finds a set S of pairs (d', σ) where d' is a training document and σ is the BM25 similarity between d and d'. The set S is initially empty. The procedure `related` is used to find, among training documents D_{train} those documents R_d that are related to the input document d (share at least a word). For each document d' in R_d, the BM25 similarity σ is calculated between d and d' using the parameters k and b, and the pair (d', σ) is appended to S.

Then, the procedure `top` is used to sort S in descending order based on similarity values (σ), and to return top t elements. The result S' is also a set of pairs (d', σ) and contain neighbor documents of d. The procedure `vote` is then used to find the category c that is most common in the documents in S'. Finally, the category c is returned as the predicted category of the input document d.

Algorithm 1 `applyKNN` in Conventional Tuning

```
1: procedure APPLYKNN($D_{train}$, $D_{valid}$, $k$, $b$, $t$)
2:     $P \leftarrow \emptyset$
3:     for $d \in D_{valid}$ do
4:         $c \leftarrow$ classify($d$, $D_{train}$, $k$, $b$, $t$)
5:         $P \leftarrow P \cup \{(d, c)\}$
6:     return $P$
```

Algorithm 2 `classify` in Conventional Tuning

```
1: procedure CLASSIFY($d$, $D_{train}$, $k$, $b$, $t$)
2:     $S \leftarrow \emptyset$
3:     $R_d \leftarrow$ related($d$, $D_{train}$)
4:     for $d' \in R_d$ do
5:         $\sigma \leftarrow$ BM25($d$, $d'$, $k$, $b$)
6:         $S \leftarrow S \cup \{(d', \sigma)\}$
7:     $S' \leftarrow$ top($S$, $t$)
8:     $c \leftarrow$ vote($S'$)
9:     return $c$
```

It is important to note that `classify` procedure can be executed in parallel (e.g., using multi-threads), because the document d it classifies is independent of other documents in the validation set D_{valid}. This parallelism is crucial for our Fast Tuning method as we will see later.

3.3 `Tuning` procedure

The conventional way to tune parameters is exhaustive search, where all possible combinations of parameter values are tried. For each combination the k-NN classification is applied over the validation set, thus the predicted categories of input documents are evaluated against the actual categories (ground truth). The evaluation can be done using a variety of performance measures (e.g., precision, recall, F_1, etc). In our case, as we have three parameters to optimize, the tuning procedure will mainly consist of three nested loops (one loop per parameter), and inside of these loops the k-NN is applied.

Algorithm 3 illustrates the conventional tuning process. The input is the training and validation datasets: D_{train}, D_{valid}, a validation ground truth set G_{valid} and the sets of parameter values: K, B, and T. The output is the optimal parameter values: $\widehat{k} \in K$, $\widehat{b} \in B$, and $\widehat{t} \in T$, and the performance score at those optimal values: $\widehat{f}$.

First, optimal parameters are initialized to the first element in their respective ranges, and the variable of best performance score $\widehat{f}$ is set to some lowest values (e.g., zero). Then, three nested loops over the ranges of parameter values are used to enumerate all possible combinations of parameter values. For each such combination (k, b, t), the procedure `applyKNN` is used to find the set of predictions P, which comprises (document, category) pairs for each document in the validation set D_{valid}. Then, `eval` is used to evaluate the predictions set P against ground truth set G_{valid} giving a performance score f. If this score f is greater than the so-far-optimal score $\widehat{f}$, then the current parameter combination (k, b, t) is the best combination so far, and thus those parameter values are designated as the so-far-optimal parameters: $(\widehat{k}, \widehat{b}, \widehat{t})$. Finally, by the end of the procedure, the optimal performance score $\widehat{f}$, and the optimal parameters $\widehat{k}$, $\widehat{b}$, and $\widehat{t}$ are returned as output.

Algorithm 3 Conventional Tuning

```
1: procedure TUNING($D_{train}$, $D_{valid}$, $G_{valid}$, $K$, $B$, $T$)
2:     $\widehat{k} \leftarrow K[0]$, $\widehat{b} \leftarrow B[0]$, $\widehat{t} \leftarrow T[0]$
3:     $\widehat{f} \leftarrow 0$
4:     for $k \in K$ do
5:         for $b \in B$ do
6:             for $t \in T$ do
7:                 $P \leftarrow$ applyKNN($D_{train}$, $D_{valid}$, $k$, $b$, $t$)
8:                 $f \leftarrow$ eval($P$, $G_{valid}$)
9:                 if $f > \widehat{f}$ then
10:                    $\widehat{f} \leftarrow f$, $\widehat{k} \leftarrow k$, $\widehat{b} \leftarrow b$, $\widehat{t} \leftarrow t$
11:    return $\widehat{f}$, $\widehat{k}$, $\widehat{b}$, $\widehat{t}$
```

As discussed earlier, depending on the available computational resources and the problem size, each evaluation may take considerable time. Thus, the overall optimization process is time consuming. This leads to an increasing need for efficient methods to optimize hyperparameters. In our case, the main issue is with calling `appyKNN` procedure (which in turn calls `classify` procedure) inside three nested loops over the three sets of parameter values K, B, and T. Thus, we aim at improving the optimization process by rewriting the tuning procedure such that we push the three nested loops inside the `applyKNN` procedure.

4 Fast tuning

We propose an efficient method, called *Fast Tuning*, for optimizing hyperparameters of k-NN approach with BM25 similarity. This method is based on the following idea: We re-arrange the control-flow components of the conventional tuning algorithm, by pushing the nested loops (over sets of parameter values) to the inside of the `classify` procedure. Thus, we need to call `applyKNN` procedure only **once**, instead of $|K| \times |B| \times |T|$ times. Therefore, the procedure `classify` need to be called $|D_{valid}|$ times only (instead of $|K| \times |B| \times |T| \times |D_{valid}|$ times in conventional tuning). Since `classify` can be executed in parallel, the time needed for the overall tuning process will be dramatically reduced.

Figure 1 illustrates how the control flow of tuning procedure is arranged in conventional tuning and Fast Tuning. The main difference is: in conventional tuning nested loops over sets of parameter values are outside the `applyKNN` procedure, whereas in Fast Tuning these loops are inside the `applyKNN` procedure (more precisely, inside the `classify` procedure).

Recall that `classify` procedure takes as input a document $d \in D_{valid}$ and returns a predicted category for that document as output. However, it assumes a fixed combination of parameters (k, b, t) which is given as input too.

In Fast Tuning method, `classify` procedure will handle all possible combinations of parameters in one shot. This implies that the input will not be a single combination (k, b, t), but three sets of parameter values K, B and T. This also implies that the output will not be a single predicted category, but a grid that associates each combination $(k, b, t) \in K \times B \times T$ with a predicted category for the input document based on that parameter combination. Therefore, appropriate data structures are crucial: Internally, for each $\langle k, b, t \rangle$ combination, we have a predicted category for an input document. Thus, we need a 3D grid to hold those categories, and convey them to `applyKNN`. This can be implemented as a 3-key value hash table, or nested hash maps. In the following subsections we present the new versions of the procedures `applyKNN`, `classify` and `Tuning` in our Fast Tuning method.

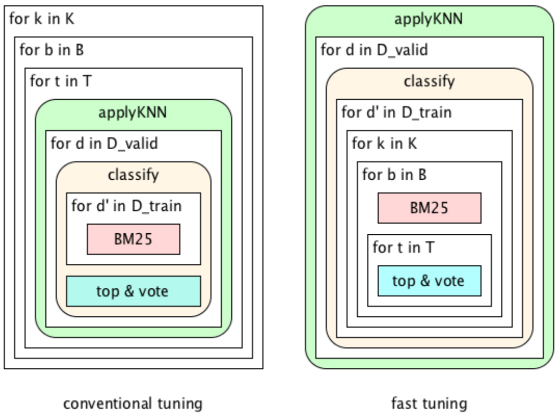


Figure 1: Control flow of Fast Tuning method comparing to conventional tuning

4.1 applyKNN

This procedure takes as input the training and validation datasets: D_{train} and D_{valid} respectively, and the sets of parameter values K, B, and T. It returns as output a 3-keys table Ω whose 3-keys are parameter combinations $(k, b, t) \in K \times B \times T$. This table associates each such combination with a set P of (document, category) pairs. The table Ω is initially empty. For each document d in the validation set D_{valid}, the procedure `classify` is used to classify the document d for each combination (k, b, t) using the training dataset D_{train}, thus `classify` returns a 3-key table Z that associates with each combination (k, b, t) a predicted category c for the document d. Then, for each combination (k, b, t) the (document, category) pair (d, c) is inserted properly into Ω table.

Although Ω and Z tables look similar as they both have 3 keys: k, b, and t, there are some differences between them as illustrated in Figure 2. First, the procedure uses a single Ω table, but as many Z tables as the documents in D_{valid}. Second, in Z table, each (k, b, t) cell has as value a single predicted category c, whereas in Ω table, each (k, b, t) cell has as value a set of (document, category) pairs.

Algorithm 4 `applyKNN` in Fast Tuning

```
 1: procedure APPLYKNN($D_{train}$, $D_{valid}$, $K$, $B$, $T$)
 2:     $\Omega[.][.][.] \leftarrow \emptyset$                         ▷ 3D table
 3:     for $d \in D_{valid}$ do
 4:         $Z \leftarrow$ classify($d$, $D_{train}$, $K$, $B$, $T$)
 5:         for $k \in K$ do
 6:             for $b \in B$ do
 7:                 for $t \in T$ do
 8:                     $c \leftarrow Z[k][b][t]$
 9:                     $\Omega[k][b][t] \leftarrow \Omega[k][b][t] \cup \{(d, c)\}$
10:     return $\Omega$
```

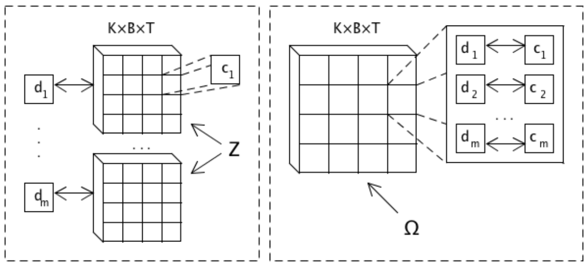


Figure 2: Structure of Z and Ω tables

4.2 classify

This procedure takes as input a document d, the training set D_{train}, and three sets of parameters K, B and T. It returns as output a 3-keys table Z that associates each combination $(k, b, t) \in K \times B \times T$ with a category c predicted for document d. The procedure consists of two parts, the first aims at finding an auxiliary 2-key table Q, while the second uses Q to compute the output 3-key table Z.

The auxiliary table Q associates each combination (k, b) with a set S of (d', σ) pairs, where d' is a training document and σ is the BM25 similarity between d and d'. The table Q is initially empty. The procedure `related` is used to find, among training documents D_{train} those documents R_d that are related to the input document d. For each document d' in R_d, and for each combination $(k, b) \in K \times B$, the BM25 similarity σ is calculated between d and d' using the parameters k and b, and the pair (d', σ) is properly inserted into Q (i.e., appended to $Q[k][b]$).

In the second part of the procedure, the output table Z is initially empty. For each combination (k, b), the corresponding set S of (d', σ) pairs is extracted from Q and sorted in descending order based on similarity values (σ). Then, for each value of the third parameter $t \in T$, the procedure `top` is used to find the top t elements of S. The result S' is also a set of pairs (d', σ) and contain neighbor documents of d. The procedure `vote` is then used to find the category c that is most common in the documents in S'. This category c is designated as the predicted category of the input document d for the current combination (k, b, t), hence it is inserted properly into the table Z, which is finally returned as output.

Algorithm 5 `classify` in Fast Tuning

```
1:  procedure CLASSIFY($d, D_{train}, K, B, T$)
2:      $Q[.][.] \leftarrow \emptyset$                          ▷ 2D table
3:      $R_d \leftarrow$ related($d, D_{train}$)
4:      for $d' \in R_d$ do
5:          for $k \in K$ do
6:              for $b \in B$ do
7:                  $\sigma \leftarrow$ BM25($d, d', k, b$)
8:                  $Q[k][b] \leftarrow Q[k][b] \cup \{(d', \sigma)\}$
9:      $Z[.][.][.] = \emptyset$                          ▷ 3D table
10:     for $k \in K$ do
11:         for $b \in B$ do
12:             $S \leftarrow Q[k][b]$
13:             for $t \in T$ do
14:                 $S' \leftarrow$ top($S, t$)
15:                 $c \leftarrow$ vote($S'$)
16:                 $Z[k][b][t] \leftarrow c$
17:     return $Z$
```

4.3 `Tuning` procedure

The main *Tuning* procedure in Fast Tuning method is depicted in Algorithm 6. It is basically very similar to *Tuning* procedure in conventional tuning. The input is the training and validation datasets: D_{train}, D_{valid}, a validation ground truth set G_{valid} and the sets of parameter values: K, B, and T, whereas the output is the optimal parameter values: $\widehat{k} \in K$, $\widehat{b} \in B$ and $\widehat{t} \in T$, and the performance score at those optimal values: $\widehat{f}$. The major difference is that `applyKNN` procedure is called only once, before the three nested loops over the sets of parameters K, B, and T. Here, `applyKNN` procedure returns a 3-key table Ω that associates each $(k, b, t) \in K \times B \times T$ with a set P of predictions (d, c) where each validation document d has a predicted category c. Inside the three nested loops, the set of predictions P for each parameter combination (k, b, t) is extracted from Ω, and evaluated against the ground truth G_{valid} using `eval` procedure, giving a performance score f. If this score f is greater than the so-far-optimal score $\widehat{f}$, then the current parameter combination (k, b, t) is the best combination so far, and thus those parameter values are designated as the so-far-optimal parameters: $(\widehat{k}, \widehat{b}, \widehat{t})$. Finally, by the end of the procedure, the optimal performance score $\widehat{f}$, and the optimal parameters $\widehat{k}$, $\widehat{b}$, and $\widehat{t}$ are returned as output.

Algorithm 6 Fast Tuning

```
1:  procedure TUNING($D_{train}, D_{valid}, G_{valid}, K, B, T$)
2:      $\widehat{k} \leftarrow K[0], \widehat{b} \leftarrow B[0], \widehat{t} \leftarrow T[0]$
3:      $\widehat{f} \leftarrow 0$
4:      $\Omega \leftarrow$ applyKNN($D_{train}, D_{valid}, K, B, T$)
5:      for $k \in K$ do
6:          for $b \in B$ do
7:              for $t \in T$ do
8:                  $P \leftarrow \Omega[k][b][t]$
9:                  $f \leftarrow$ eval($P, G_{valid}$)
10:                 if $f > \widehat{f}$ then
11:                     $\widehat{f} \leftarrow f, \widehat{k} \leftarrow k, \widehat{b} \leftarrow b, \widehat{t} \leftarrow t$
12:     return $\widehat{f}, \widehat{k}, \widehat{b}, \widehat{t}$
```

5 Fast tuning with cross-validation

So far, we have seen Fast Tuning technique for a single training-validation pair. Typically, hyperparameter optimization is performed with *cross-validation*. Cross validation is model validation technique for assessing how the model will generalize to an independent data set.

A common type of cross-validation is n-fold cross-validation, where the original sample is randomly partitioned into n equal sized subsamples. Of the n subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $n - 1$ subsamples are used as training data. The cross-validation process is then repeated n times, with each of the n subsamples used exactly once as the validation data. The n results can then be averaged to produce a single estimation.

In our case of text categorization, n-fold cross validation turns out to be a set of n classification tasks Δ. Each task $\delta \in \Delta$ consists of a training and a validation sets of documents D^{δ}_{train}, D^{δ}_{valid}, and a validation ground truth G^{δ}_{valid}. If each task is used independently to tune the hyperparameters, we will have different values for k, b and t parameters per task. But we would like to have for each parameter a single, optimal value for all classification tasks together. Therefore, we need to rewrite the tuning procedure such that it optimizes the hyperparameters k, b and *t jointly* for all the classification tasks. Now we revisit the procedure `tuning`, and adjust it in order to adapt to cross-validation. The procedures `applyKNN` and `classify` stay the same, as described in Algorithms 4 and 5, respectively.

With cross-validation, we have a set of classification tasks Δ, instead of a single pair of training-validation data sets. Thus, the input of `Tuning` procedure consists of Δ and the three sets of parameter values K, B, and T. The output consists, as usual, of the optimal parameter values: $\widehat{k} \in K$, $\widehat{b} \in B$ and $\widehat{t} \in T$, and the performance score at those optimal values: $\widehat{f}$.

Since now we have multiple classification tasks, the procedure `applyKNN` is called for each task δ with the training and validation datasets of this task D^{δ}_{train}, D^{δ}_{valid} as arguments (besides K, B and T). The resulting 3-key table Ω_δ is associated with an identifier of the task δ in an auxiliary table X. Recall that the table Ω_δ associates each parameter combination $(k, b, t) \in K \times B \times T$ with a set P of predictions (d, c) where each validation document d has a predicted category c.

Now, inside the three nested loops over the sets of parameter values, for each task $\delta \in \Delta$, the 3-key table Ω_δ is extracted from the auxiliary table X, then the set of predictions P for each parameter combination (k, b, t) is extracted from Ω_δ, and evaluated against the ground truth of that task G^{δ}_{valid} using `eval` procedure, giving a performance score f_δ. This performance score is averaged over all tasks: $\overline{f} = \frac{1}{n}\sum_{\delta\in\Delta} f_\delta$. If this average performance score $\overline{f}$ is greater than the so-far-optimal score $\widehat{f}$, then the current parameter combination (k, b, t) is the best combination so far, and thus those parameter values are designated as the so-far-optimal parameters: $(\widehat{k}, \widehat{b}, \widehat{t})$. Finally, by the end of the procedure, the optimal performance score $\widehat{f}$, and the optimal parameters $\widehat{k}$, $\widehat{b}$, and $\widehat{t}$ are returned as output.

Algorithm 7 Fast Tuning with cross-validation

```
1:  procedure TUNING($\Delta$, $K$, $B$, $T$)
2:      $X[.] \leftarrow \emptyset$
3:      for $\delta \in \Delta$ do
4:          $\Omega_\delta \leftarrow$ applyKNN($D^{\delta}_{train}$, $D^{\delta}_{valid}$, $K$, $B$, $T$)
5:          $X[\delta] \leftarrow \Omega_\delta$
6:      $\widehat{k} \leftarrow K[0]$, $\widehat{b} \leftarrow B[0]$, $\widehat{t} \leftarrow T[0]$
7:      $\widehat{f} \leftarrow 0$
8:      for $k \in K$ do
9:          for $b \in B$ do
10:             for $t \in T$ do
11:                 $\overline{f} \leftarrow 0$
12:                 for $\delta \in \Delta$ do
13:                     $\Omega_\delta \leftarrow X[\delta]$
14:                     $P_\delta \leftarrow \Omega_\delta[k][b][t]$
15:                     $f_\delta \leftarrow$ eval($P_\delta$, $G^{\delta}_{valid}$)
16:                     $\overline{f} \leftarrow \overline{f} + f_\delta$
17:                 $\overline{f} \leftarrow \overline{f}/n$
18:                 if $\overline{f} > \widehat{f}$ then
19:                     $\widehat{f} \leftarrow \overline{f}$, $\widehat{k} \leftarrow k$, $\widehat{b} \leftarrow b$, $\widehat{t} \leftarrow t$
20:     return $\widehat{f}$, $\widehat{k}$, $\widehat{b}$, $\widehat{t}$
```

6 Experiments

6.1 Datasets

In our experiments we use two data sets. The first one is the *News Popularity in Multiple Social Media Platforms Data Set*[2] [52]. It is a large data set of news items and their respective social feedback on multiple platforms: Facebook, Google+ and LinkedIn. The collected data relates to a period of 8 months, between November 2015 and July 2016, accounting for about 90,000 news items on four different topics: Economy, Microsoft, Obama and Palestine. For each news item, we take the Title and the Headline as a document, and the Topic as category. The average document length is about 20 words per document. The second dataset is the popular *BBC* dataset[3] [53]. It consists of 2225 documents from the BBC news website corresponding to stories in five topical areas from 2004-2005: business, entertainment, politics, sport, and tech. The average document length is about 200 words. For both datasets, the classification task is to classify a text document into one of the topical areas.

6.2 Setup

All experiments are conducted on a PC operated by Linux Ubuntu 16.04 (64-bit), and equipped with an Intel Core i7-6700 CPU @ 3.40GHz with 8 cores, and 32 GB of RAM. Algorithms are implemented in Java 8. Parallelism is achieved via multi-threading, where an Executor Service[4] is used over a fixed thread pool of 8 threads. Both conventional tuning and Fast Tuning methods are parallelised.

2 https://archive.ics.uci.edu/ml/datasets/News+Popularity+in+Multiple+Social+Media+Platforms

3 http://mlg.ucd.ie/datasets/bbc.html

4 https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html

The document preprocessing pipeline we used comprises tokenization, and normalization (no stemming). Normalization consists of: 1) removal of non-ASCII tokens, 2) conversion to lowercase, and 3) removal of stopwords[5], punctuation and numbers.

6.3 Experiments design

The main aim of conducted experiments is to evaluate the efficiency of our Fast Tuning method compared to the conventional tuning. We have conducted five types of experiments using both News Popularity and BBC datasets. The first three experiments use a single classification task (a single training-validation pair of datasets), whereas the fourth experiment uses 5-fold cross-validation (five classification tasks). In the first experiment, we fix the three sets of parameter values (K, B, and T), and vary the dataset size. In the second experiment, we fix the dataset size, and vary one set of parameters while fixing the remaining two (thus experiments has three parts). In the third experiment, we also fix the dataset size, and vary the three sets of parameters simultaneously. In the fourth experiment, we use 5-fold cross-validation, we fix the three sets of parameter values (K, B, and T), and vary the dataset size. The fifth experiment is dedicated to compare Fast Tuning to a simple two steps tuning. We fix the three sets of parameter values (K, B, and T), and vary the dataset size. Table 1 summarizes this design of experiments and shows the sizes of parameter sets and datasets in each experiment.

Table 1: Experiments design

Exp.	$\|K\|$	$\|B\|$	$\|T\|$	dataset size		tasks
				BBC	NwP	
1	10	10	10	200..2000		1
2-1	2..20	10	10	1000	5000	1
2-2	10	2..20	10			
2-3	10	10	2..20			
3	2..20			1000		1
4	10	10	10	100..1000	500..5000	5
5	9	11	10		500..5000	1

In every experiment we measure the time (in seconds) needed for parameter tuning using conventional tuning and our Fast Tuning method. To compare these times we used the following two metrics:

5 we used the default English stopwords list from https://www.ranks.nl/stopwords

Speed Up: it measures how much Fast Tuning is *faster* than conventional tuning:

$$SU = \frac{\theta_C}{\theta_F}$$

Time Reduction: it measures the percentage of time reduced when Fast Tuning is used comparing to conventional tuning.

$$TR = (\frac{\theta_C - \theta_F}{\theta_C}) \times 100\%$$

where θ_C and θ_F are tuning time using conventional and fast methods, respectively.

Since the objective of the experiments is to measure and compare the *tuning time* for conventional tuning and Fast Tuning methods, we do not report in our experiments the optimal values of hyperparameters k, b and t, nor the values of classification performance metric, as those values are not relevant to the objective of the experiments, and have no impact on the results. However, it is worth mentioning that, in every experiment, both the conventional tuning and Fast Tuning methods find exactly the same optimal parameter values, and the same value of the classification performance metric.

6.4 Results

6.4.1 Experiment 1

In this experiment, we use fixed sets of parameters: $K = B = \{0, 0.1, \cdots, 0.9\}$, and $T = \{3, 5, \cdots, 21\}$ (hence $|K| = |B| = |T| = 10$), and several samples of the datasets of different sizes ranging from 100 to 2000 documents. For each sample, we use 80% of documents as a training set, and the remaining 20% as a validation set. We measured time needed for parameter tuning (find optimal parameters) for every sample using conventional tuning and our Fast Tuning methods, and calculated the speed up (SU) and time reduction (TR) factors. Table 2 shows a snippet of obtained results, while the complete results are listed in Appendix A.1. Figure 3 depicts how the tuning time (in seconds) changes over the dataset size.

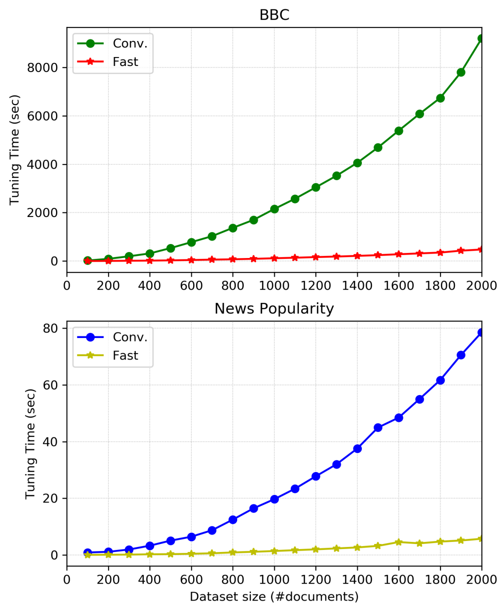


Figure 3: Experiment 1; variable dataset size, fixed K, B and T

Table 2: Experiment 1

Dataset Size	Tuning Time (s) Conv.	Fast	SU	TR %
	BBC			
500	535.226	29.605	18.08	94.5
1000	2150.680	116.208	18.51	94.6
1500	4691.521	244.595	19.18	94.8
2000	9196.227	477.491	19.26	94.8
	News Popularity			
500	5.082	0.311	16.34	93.9
1000	19.689	1.414	13.92	92.8
1500	44.972	3.228	13.93	92.8
2000	78.586	5.741	13.69	92.7

We observe that the tuning time increases very rapidly in conventional method, while this increase is much slower in Fast Tuning. For instance, for 2000 documents of BBC dataset, conventional tuning took more than two and half hours, while Fast Tuning took less than 8 minutes. The Speed-Up factor is in the range 16 to 20 for BBC dataset (i.e., Fast Tuning is 16 $\sim$ 20× faster), and in 10 to 17 for News-Popularity set. In general, Fast Tuning is at least an order of magnitude faster than conventional tuning. The efficiency of Fast Tuning is also clear from the time reduction factor which is at least 90%, i.e., using Fast Tuning, we can save more than 90% of the time needed using conventional tuning.

6.4.2 Experiment 2

In this experiment, we use a fixed-size sample of data sets (BBC: 1000 documents, News-Popularity: 5000), each sample is split into [80%, 20%] training-validation sets. We use variable-size parameter sets. This experiment comprises three parts, in each part we vary one set of parameters while fixing the remaining two. First, in part 2-1, we vary K and fix B and T; then in part 2-2, we vary B and fix K and T, finally in part 2-3, we vary T and fix K and B. In all these three parts, the size of fixed sets is always 10, while the size of the variable set ranges from 2 to 20. As usual, we measure the tuning time for both conventional and Fast Tuning methods. The detailed results are listed in Appendix A.2. Figure 4 shows how tuning time changes with respect to the size of the variable parameter set.

It is clear that both conventional and Fast Tuning have linear behavior with respect to the size of parameter sets. However, Fast Tuning is much faster than conventional tuning. To validate this finding, we fit the results of experiment 2 to linear models[6], and find the coefficients (slope of the straight line) for each case, as shown in Table 3. We assumed the intercept to be 0, since when the size of the parameter is 0 the tuning time is also considered 0. We find that in all cases, the coefficients of Fast Tuning models are much smaller than their corresponding coefficients of conventional tuning; and this again confirms the efficiency of fast tuning method.

Table 3: Coefficients of linear models of experiments 2

	BBC		News Popularity	
	Conv.	Fast	Conv.	Fast
Expr. 2-1	218.91	**11.66**	48.68	**3.45**
Expr. 2-2	238.26	**12.81**	45.20	**3.33**
Expr. 2-3	238.72	**9.15**	47.27	**2.42**

Moreover, we observed that in experiment 2-3 where we vary only the set T, the tuning time of Fast Tuning method is almost constant. To validate this observation, we first plot the results on a logarithmic scale, as shown in Figure 5. This demonstrates that changes of fast tuning is constant indeed, comparing to the increased change of conventional tuning.

6 using Python module: `sklearn.linear_model.LinearRegression`

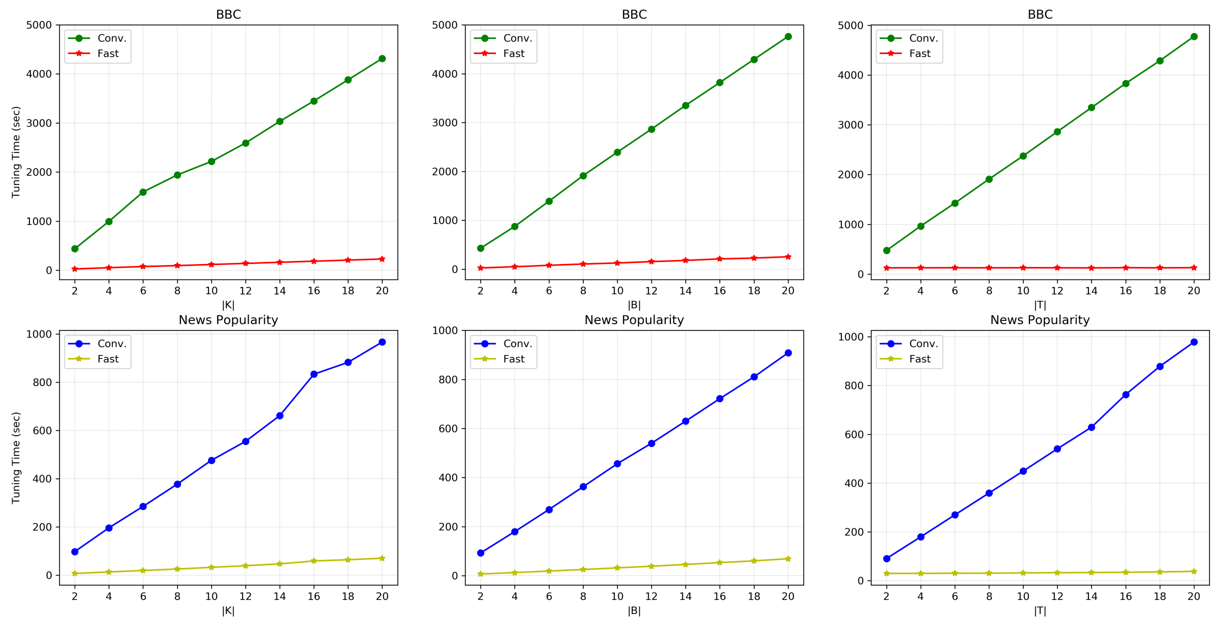


Figure 4: Results of experiment 2. Fixed dataset size (BBC: 1000 documents, News-Popularity: 5000). Left column (expr.2-1): variable K, fixed B, T. Middle column (expr.2-2): variable B, fixed K, T. Right column (expr.2-3): variable T, fixed K, B.

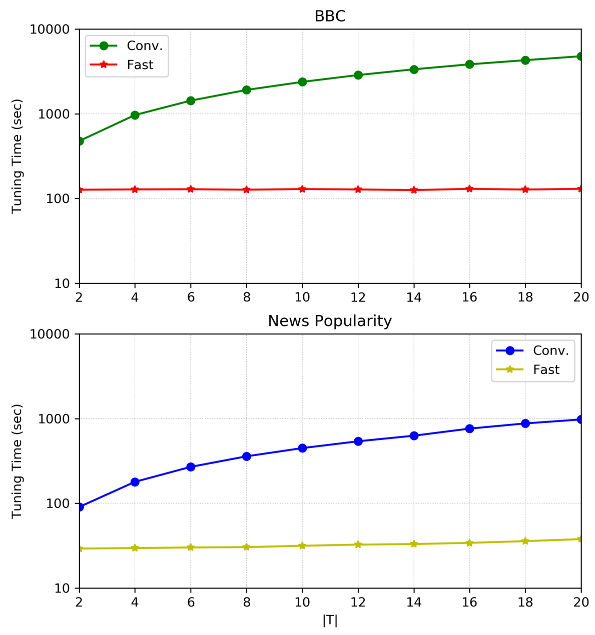


Figure 5: Results of experiment 2-3; logarithmic scale; fixed dataset size, variable K; fixed B and T

We also calculated the standard deviation of each case as shown in Table 4. We find that the standard deviation values for the case of Fast Tuning in experiment 2-3 are very close to 1 comparing to the rest of cases. This also confirms the stability of Fast Tuning with respect to changes of the size of set T.

Table 4: Standard deviation of the results of experiments 2

	BBC		News Popularity	
	Conv.	Fast	Conv.	Fast
Expr. 2-1	1186.95	64.09	284.65	20.83
Expr. 2-2	1390.09	73.22	259.43	19.80
Expr. 2-3	1371.24	**1.32**	283.59	**2.68**

6.4.3 Experiment 3

In this experiment we also used fixed-size samples of the datasets (1000 documents), and we vary the three sets of parameters K, B, and T simultaneously. That is, we run the experiment 10 times, at each run, the parameter sets K, B and T have the same size, namely $2, 4, \cdots, 20$. We measure tuning time for both conventional and Fast Tuning methods, and compute speed up (SU) and time reduction (TR) factors. The complete results are listed in Appendix A.3, and plotted in Figure 6.

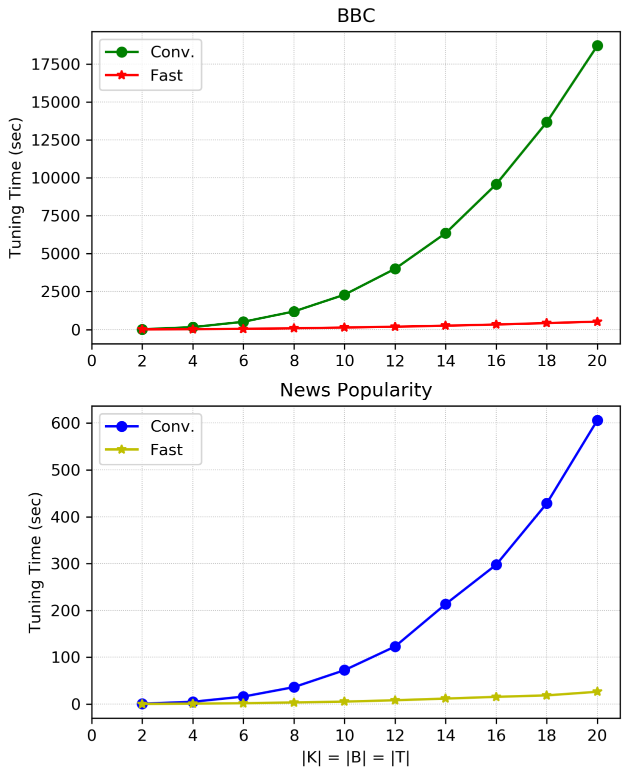
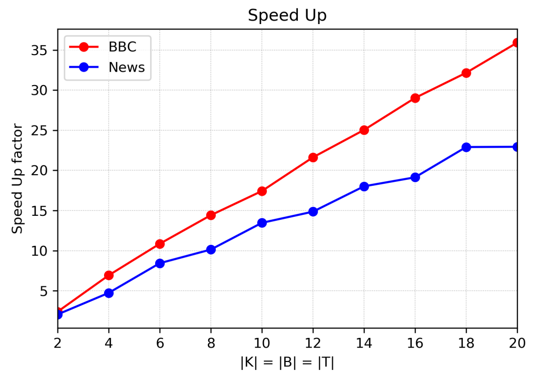
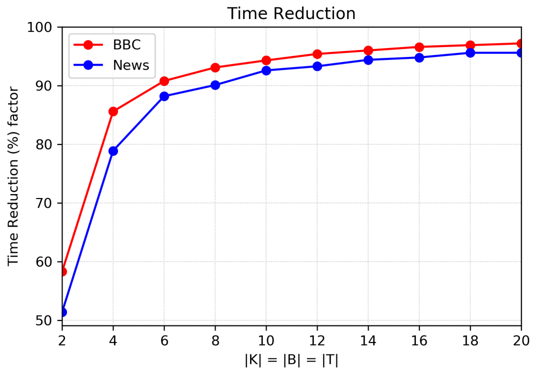


Figure 6: Results of experiment 3; fixed dataset size; simultaneously varying parameter sets K, B and T

We observe how the tuning time increases with the size of the parameter sets. This increase is very rapid in conventional method, whereas it is slow in our fast method. For example, for BBC dataset, when the grid size is 20^3, the conventional method took 18,700 seconds (> 5 hours), while Fast Tuning method took only 520 sec (~ 8.5 min), giving a Speed-Up factor of about 36, and a Time-Reduction of about 97%. Moreover, we observe that the Speed-Up and Time Reduction factors are not steady, they are actually increasing too. Figure 7 depicts how these factors change.



(a) Speed Up



(b) Time Reduction

Figure 7: Speed-Up and Time-Reduction factors; with variable sizes of K, B and T

The Speed-Up factor increases linearly with the size of the parameter grid; while the Time Reduction factor increases in a logarithmic manner. This means that the Fast Tuning method is not only faster than the conventional method, but also becomes even more faster when the grid size increases. We can also observe that the values of both factors are higher for BBC dataset than for News Popularity. Recall that BBC dataset has longer documents than in News Popularity dataset, thus, we conclude that fast tuning method is also relatively more faster with longer documents.

6.4.4 Experiment 4

In this experiment, we use 5-fold cross-validation. We use a fixed sets of parameters, each of size 10, and varying-size samples of data sets. From BBC, we draw 10 samples of size 100 up to 1000, and from News Popularity we draw 10 samples of size 500 up to 5000 documents. Each sample is split into (80%, 20%) training-validation sets. We run conventional tuning and our Fast Tuning method with cross validation (Section 5). The complete results are listed in Appendix A.4, and depicted in Figure 8.

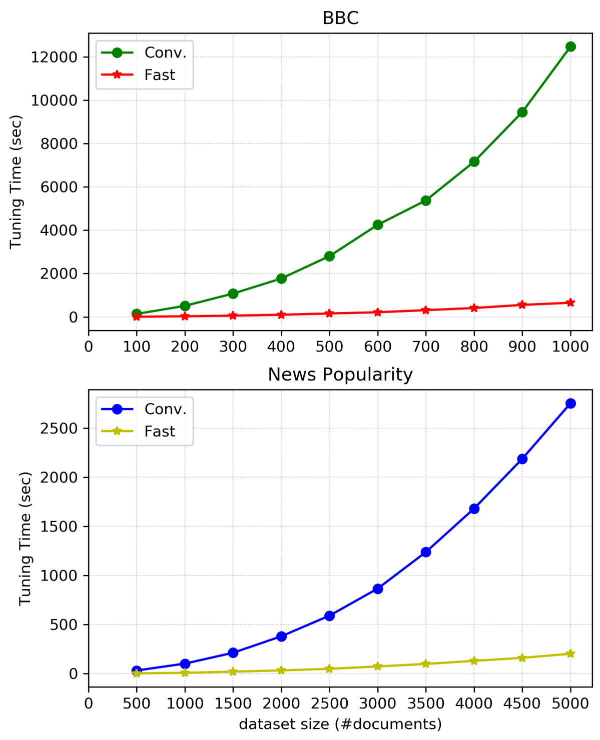


Figure 8: Results of experiment 4; fixed parameter sets; variable-size dataset samples

Once more, the results show how our Fast Tuning method outperforms the conventional method. For BBC dataset, the mean Speed-Up factor is about 18, and the mean Time-Reduction factor is about 94.5%. For News Popularity dataset, the mean Speed-Up factor is about 13.4, and the mean Time-Reduction factor is about 92%.

6.4.5 Experiment 5

In this experiment, we compare Fast Tuning to a simple optimization of two-steps:

1. grid search of BM25 parameters k and b
2. grid search of kNN parameter t given the best configuration of BM25.

This simple optimization reduces the tuning time comparing to the conventional approach. However, it does not guarantee to find the optimal parameter values that conventional and Fast Tuning find, because it does not scan the entire 3D grid $K \times B \times T$. We use a fixed sets of parameters: $K = \{1.2, 1.3, \cdots, 2.0\}$, $B = \{0.0, 0.1, \cdots, 1.0\}$, $T = \{5, 7, \cdots, 23\}$, and varying-size samples of data sets. From News Popularity we draw 10 samples of size 500 up to 5000 documents. Each sample is split into (80%, 20%) training-validation sets.

Table 5 shows the optimal parameter values and F1 scores for the two-steps optimization and Fast Tuning. We observe that the optimal parameter values found by the two-steps optimization are different from those found by the Fast Tuning method. Consequently, the F1-measure obtained by two-steps optimization is lower than what we obtain using Fast Tuning, for almost all trials, as shown in Figure 9a. More interestingly, Fast Tuning outperforms the two-steps optimization even in terms of the tuning time, as shown in Table 6 and Figure 9b.

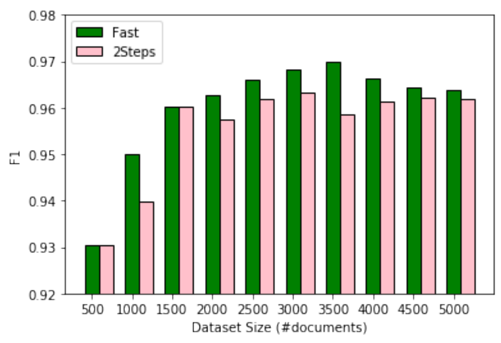
Table 5: Results of experiment 5: optimal parameter values and F1 scores for two-steps optimization and Fast Tuning.

size	Two Steps				Fast Tuning			
	$\widehat{k}$	$\widehat{b}$	$\widehat{t}$	$\widehat{F_1}$	$\widehat{k}$	$\widehat{b}$	$\widehat{t}$	$\widehat{F_1}$
500	1.2	0.0	15	0.9304	1.2	0.0	15	0.9304
1000	1.3	0.7	15	0.9397	1.4	1.0	19	**0.9501**
1500	1.2	1.0	21	0.9601	1.2	0.3	23	0.9601
2000	1.4	1.0	23	0.9575	1.9	1.0	21	**0.9626**
2500	1.8	0.9	19	0.9619	2.0	1.0	23	**0.9659**
3000	1.2	0.1	15	0.9633	1.9	1.0	23	**0.9683**
3500	1.5	0.3	17	0.9586	1.9	1.0	23	**0.9700**
4000	2.0	0.8	23	0.9612	1.2	1.0	23	**0.9663**
4500	1.7	0.8	23	0.9622	1.7	0.6	23	**0.9644**
5000	1.8	0.7	23	0.9619	1.7	1.0	23	**0.9639**

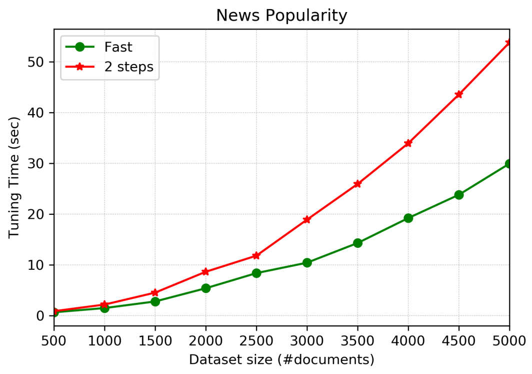
Although the two-steps optimization is very fast comparing to the conventional tuning, however, it still can not beat our Fast Tuning method which is faster than the two-steps method with a speed-up ratio of 1.64 on average, and 38% time reduction factor (Table 6).

Table 6: Experiment 5. Tuning Time

Size	Tuning Time (s)		SU	TR %
	2Step.	Fast		
500	0.885	0.668	1.32	24.52
1000	2.187	1.500	1.46	31.41
1500	4.542	2.801	1.62	38.33
2000	8.645	5.400	1.60	37.54
2500	11.809	8.378	1.41	29.05
3000	18.902	10.431	1.81	44.82
3500	25.921	14.306	1.81	44.81
4000	33.930	19.210	1.77	43.38
4500	43.525	23.801	1.83	45.32
5000	53.760	29.936	1.80	44.32
avg.			1.643	38.35



(a) F1 measure



(b) Tuning Time

Figure 9: Fast Tuning vs. two-steps optimization

7 Discussion and future work

The results of conducted experiments show that Fast Tuning is a promising technique to speed up the process of hyperparameter optimization. The efficiency of this technique is due to a smart combination of three simple ingredients:

1. separation of independent calculations,
2. utilization of an appropriate data structure, and
3. parallelisation.

Since the computation of BM25 similarity is independent from hyperparameter t of kNN, it is possible to precalculate it for all combinations of k and b values, and to store it in an indexed structure, which is accessed and used later to perform kNN classification for all t values. From an algorithmic point of view, this arrangement reduces the complexity of the tuning algorithm, in terms of parallel jobs, from $|K|\times|B|\times|T|\times|D_{valid}|$ jobs in conventional tuning to only $|D_{valid}|$ jobs in Fast Tuning.

Here, each job is mainly a call to `classify` procedure, which comprises, in conventional tuning, a call to `BM25` procedure $|D_{train}|$ times, and one call to `top` and `vote` procedures, whereas in Fast Tuning, it comprises a call to `BM25` procedure $|K|\times|B|\times|D_{train}|$ times and to `top` and `vote` procedures $|K|\times|B|\times|T|$ times.

More formally, let $\alpha = |K|$, $\beta = |B|$, $\gamma = |T|$, $\delta = |D_{valid}|$. Let η denote the average number of related (training) documents to a validation document. Let c_1 and c_2 denote the costs of calling the procedures `related` and `BM25`, respectively, and let c_3 denotes the cost of calling `top` and `vote` procedures (together). The number of parallel jobs in conventional tuning is $\alpha.\beta.\gamma.\delta$, which is reduced to δ in Fast tuning. Moreover, the cost of each job in conventional tuning is $c_1 + \eta.c_2 + c_3$, which increases to be $c_1 + \alpha.\beta(\eta.c_2 + \gamma.c_3)$ in Fast tuning. However, the overall cost, which is number of jobs multiplied by the job cost, of conventional tuning is still larger than of the Fast tuning, as shown in Table 7. The difference (conventional cost minus Fast cost) corresponds to the gain in tuning time that Fast Tuning method provide, and is given by:

$$\delta[(\alpha.\beta.\gamma - 1).c_1 + \alpha.\beta.\eta(\gamma - 1).c_2]$$

which is proportional to the number of validation documents δ and to the size of the grid α, β, and γ.

It is also worthy to note that in the case of two-steps tuning mentioned in experiment 5, the number of parallel jobs is $(\alpha.\beta + \gamma).\delta$, which is much less than $\alpha.\beta.\gamma.\delta$ in conventional tuning.

Table 7: Complexity of conventional tuning and Fast tuning

	Conv.	Fast
# Jobs	$\alpha.\beta.\gamma.\delta$	δ
Job cost	$c_1 + \eta.c_2 + c_3$	$c_1 + \alpha.\beta(\eta.c_2 + \gamma.c_3)$
Total cost	$\alpha.\beta.\gamma.\delta.c_1 +$ $\alpha.\beta.\gamma.\delta.\eta.c_2 +$ $\alpha.\beta.\gamma.\delta.c_3$	$\delta.c_1 +$ $\alpha.\beta.\delta.\eta.c_2 +$ $\alpha.\beta.\gamma.\delta.c_3$
diff.	$\delta[(\alpha.\beta.\gamma - 1).c_1 + \alpha.\beta.\eta(\gamma - 1).c_2]$	

The Fast Tuning method can be easily extended to other hyperparameter optimization approaches, such as Random Search [2]. For instance, with random search, combinations of hyberparameters are generated at random, and each combination is evaluated independently, where the best performing combination is considered as the optimal one. In our case of text categorization using kNN and BM25, let Y be the set of randomly-generated parameters combinations (k, b, t), then in conventional tuning with random search the procedure `applykNN` would be called for each combination $(k, b, t) \in Y$. To implement Fast Tuning technique for Random Search, we need to push the loop over Y inside `classify` procedure as shown in Figure 10.

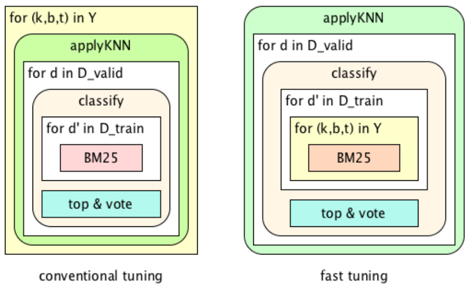


Figure 10: Control flow of Fast Tuning method comparing to conventional tuning, when applied for Random Search approach

We assume that Fast Tuning technique will outperform conventional tuning also for Random Search approach. However, a thorough investigation of Fast Tuning technique for Random Search is still necessary, and we consider it as a future work, that will also consider a comparison between Grid Search and Random Search approaches.

Despite the important advantages that Fast Tuning method provides over conventional tuning in terms of *time gain*, it still has some drawbacks. The efficiency of Fast Tuning method comes with a cost in terms of *memory consumption*. For instance, the 3-key value hash tables Z and Ω used in Fast Tuning procedure consume a large amount of main memory. Overall, this makes Fast Tuning require more memory comparing to the conventional tuning. However, this trade-off between time gain and memory consumption deserves a dedicated comparative study which we will conduct in future work.

8 Conclusion

In this paper, we have presented an efficient method, called Fast Tuning, for hyperparameter optimization with grid search for text categorization using k-NN approach and BM25 similarity. The main idea behind this method is to re-arrange the control flow of the conventional tuning algorithm such that the time-expensive nested loops over the parameter sets are pushed inside the classification procedure. As consequence, the classification procedure need to be executed once, instead of as many times as the size of the parameter grid ($|K| \times |B| \times |T|$ in kNN approach). Moreover, since the classification procedure is easy to implement in parallel, the overall tuning time is dramatically reduced comparing to the time of conventional tuning. We also addressed how we can apply our tuning method along with cross-validation.

In order to test the efficiency of our tuning method, we run an extensive set of experiments using two popular datasets and various settings: varying dataset size, varying sets of parameters, with cross-validation, etc. In all experiments our Fast Tuning method outperforms the conventional method. Our primary finding is that Fast Tuning is at least an order of magnitude faster than conventional tuning. This Speed-Up factor exceeds 36 in some experiments. Our method reduces the tuning time by at least 90% in most cases. Another finding is that our method becomes even more faster with larger grids and longer documents.

Although we only study k-NN classification approach using BM25 similarity, we assume that the principle of Fast Tuning presented here can be adapted for any approach of text categorization or information retrieval, and for any similarity measure. Therefore, future works should address the generalization of the method to other approaches and other similarity measures. Moreover, the method will be easily extended to other hyperparameter optimization approaches, such as random search.

References

[1] Claesen M., Moor B.D., Hyperparameter Search in Machine Learning, CoRR, abs/1502.02127, 2015

[2] Bergstra J., Bengio Y., Random Search for Hyper-parameter Optimization, Journal of Machine Learning Research, 13(1), 2012, 281–305

[3] Chapelle O., Vapnik V., Bousquet O., Mukherjee S., Choosing Multiple Parameters for Support Vector Machines, Machine Learning, 46(1-3), 2002, 131–159, 10.1023/A:1012450327387

[4] Do C.B., Foo C.S., Ng A.Y., Efficient Multiple Hyperparameter Learning for Log-linear Models, In Proceedings of the 20th International Conference on Neural Information Processing Systems, NIPS'07, Curran Associates Inc., USA, 2007, 377–384

[5] Wang Z., Hutter F., Zoghi M., Matheson D., De Freitas N., Bayesian Optimization in a Billion Dimensions via Random Embeddings, Journal of Artificial Intelligence Research, 55(1), 2016, 361–387

[6] Bergstra J., Bardenet R., Bengio Y., Kégl B., Algorithms for Hyper-parameter Optimization, In Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS'11, Curran Associates Inc., USA, 2011, 2546–2554

[7] Snoek J., Larochelle H., Adams R.P., Practical Bayesian Optimization of Machine Learning Algorithms, In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2, NIPS'12, USA, 2012, 2951–2959

[8] Robertson S., Walker S., Some Simple Effective Approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval, In Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '94, Springer-Verlag New York, Inc., New York, NY, USA, 1994, 232–241

[9] Robertson S., Zaragoza H., The Probabilistic Relevance Framework: BM25 and Beyond, Foundations and Trends in Information Retrieval, 3(4), 2009, 333–389, 10.1561/1500000019

[10] Yang Y., Liu X., A Re-examination of Text Categorization Methods, in Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, ACM, 1999, 42–49

[11] Masand B., Linoff G., Waltz D., Classifying News Stories Using Memory Based Reasoning, In Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '92, ACM, New York, NY, USA, 1992, 59–65, 10.1145/133160.133177

[12] Yang Y., Expert Network: Effective and Efficient Learning from Human Decisions in Text Categorization and Retrieval, in B.W. Croft, C.J. van Rijsbergen, eds., SIGIR '94, Springer London, London, 1994, 13–22

[13] Iwayama M., Tokunaga T., Cluster-based Text Categorization: A Comparison of Category Search Strategies, In Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '95, ACM, New York, NY, USA, 1995, 273–280, 10.1145/215206.215371

[14] Domhan T., Springenberg J.T., Hutter F., Speeding Up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves, In Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15, AAAI Press, 2015, 3460–3468

[15] Bengio Y., Gradient-Based Optimization of Hyperparameters, Neural Computation, 12, 2000, 1889–1900

[16] Larsen J., Hansen L.K., Svarer C., Ohlsson M., Design and regularization of neural networks: the optimal use of a validation set, in Neural Networks for Signal Processing [1996] VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop, IEEE, 1996, 62–71

[17] Maclaurin D., Duvenaud D., Adams R., Gradient-based hyperparameter optimization through reversible learning, in International Conference on Machine Learning, 2015, 2113–2122

[18] Jones D.R., Schonlau M., Welch W.J., Efficient Global Optimization of Expensive Black-Box Functions, Journal of Global Optimization, 13(4), 1998, 455–492, 10.1023/A:1008306431147

[19] Brochu E., Cora V.M., de Freitas N., A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning, CoRR, abs/1012.2599, 2010

[20] Hutter F., Hoos H.H., Leyton-Brown K., Sequential Model-Based Optimization for General Algorithm Configuration, in C.A.C. Coello, ed., Learning and Intelligent Optimization, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, 507–523

[21] Thornton C., Hutter F., Hoos H.H., Leyton-Brown K., Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms, In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2013, 847–855

[22] Aggarwal C.C., Zhai C., A Survey of Text Classification Algorithms, Springer US, Boston, MA, 2012, 163–222, 10.1007/978-1-4614-3223-4_6

[23] Yang Y., Chute C.G., An Example-based Mapping Method for Text Categorization and Retrieval, ACM Trans. Inf. Syst., 12(3), 1994, 252–277, 10.1145/183422.183424

[24] Cohen W.W., Hirsh H., Joins that Generalize: Text Classification Using WHIRL, In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), New York City, New York, USA, August 27-31, 1998, 1998, 169–173

[25] Han E.H., Karypis G., Kumar V., Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification, In Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD '01, Springer-Verlag, Berlin, Heidelberg, 2001, 53–65

[26] Guo G., Wang H., Bell D., Bi Y., Greer K., Using kNN Model for Automatic Text Categorization, Soft Computing, 10(5), 2006, 423–430, 10.1007/s00500-005-0503-y

[27] Chakrabarti S., Dom B., Agrawal R., Raghavan P., Using Taxonomy, Discriminants, and Signatures for Navigating in Text Databases, In Proceedings of the 23rd International Conference on Very Large Data Bases, VLDB '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, 446–455

[28] Koller D., Sahami M., Hierarchically Classifying Documents Using Very Few Words, In Proceedings of the Fourteenth International Conference on Machine Learning, ICML '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, 170–178

[29] Lewis D.D., Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval, In Proceedings of the 10th European Conference on Machine Learning, ECML'98, Springer-Verlag, Berlin, Heidelberg, 1998, 4–15, 10.1007/BFb0026666

[30] Lewis D.D., Catlett J., Heterogeneous uncertainty sampling for supervised learning, in Machine Learning Proceedings, Elsevier, 1994, 148–156

[31] Cohen W.W., Singer Y., Context-sensitive Learning Methods for Text Categorization, ACM Trans. Inf. Syst., 17(2), 1999, 141–173, 10.1145/306686.306688

[32] Ng H.T., Goh W.B., Low K.L., Feature Selection, Perceptron Learning, and a Usability Case Study for Text Categorization, In Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '97, ACM, New York, NY, USA, 1997, 67–73, 10.1145/258525.258537

[33] Wiener E., O. Pedersen J., S. Weigend A., A Neural Network Approach to Topic Spotting, SDAIR, 1995, 317–332

[34] Ruiz M.E., Srinivasan P., Hierarchical Neural Networks for Text Categorization, In Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '99, ACM, New York, NY, USA, 1999, 281–282, 10.1145/312624.312700

[35] Kowsari K., Brown D.E., Heidarysafa M., Meimandi K.J., Gerber M.S., Barnes L.E., HDLTex: Hierarchical Deep Learning for Text Classification, CoRR, abs/1709.08267, 2017

[36] Joachims T., Text Categorization with Support Vector Machines: Learning with Many Relevant Features, In Proceedings of the 10th European Conference on Machine Learning, ECML'98, Springer-Verlag, Berlin, Heidelberg, 1998, 137–142, 10.1007/BFb0026683

[37] Joachims T., A Statistical Learning Learning Model of Text Classification for Support Vector Machines, In Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '01, ACM, New York, NY, USA, 2001, 128–136, 10.1145/383952.383974

[38] Zhang W., Yoshida T., Tang X., Text Classification Based on Multi-word with Support Vector Machine, Know.-Based Syst., 21(8), 2008, 879–886, 10.1016/j.knosys.2008.03.044

[39] Yang Y., An Evaluation of Statistical Approaches to Text Categorization, Information Retrieval, 1(1), 1999, 10.1023/A:1009982220290

[40] Dasarathy B.V., Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques, IEEE Computer Society Press, Los Alamitos, CA, 1991

[41] Yang Y., A Study of Thresholding Strategies for Text Categorization, In Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '01, New York, NY, USA, 2001, 137–145, 10.1145/383952.383975

[42] Salton G., Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989

[43] Fang H., Tao T., Zhai C., A Formal Study of Information Retrieval Heuristics, In Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '04, ACM, New York, NY, USA, 2004, 49–56, 10.1145/1008992.1009004

[44] Singhal A., Buckley C., Mitra M., Pivoted Document Length Normalization, In Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '96, ACM, New York, NY, USA, 1996, 21–29, 10.1145/243199.243206

[45] He B., Ounis I., Term Frequency Normalisation Tuning for BM25 and DFR Models, in D.E. Losada, J.M. Fernández-Luna, eds., Advances in Information Retrieval, Springer Berlin Heidelberg, 2005, 200–214

[46] Murata M., Kanamaru T., Shirado T., Isahara H., Using the K Nearest Neighbor Method and BM25 in the Patent Document Categorization Subtask at NTCIR-5, In Proceedings of the Fifth NTCIR Workshop Meeting on Evaluation of Information Access Technologies: Information Retrieval, Question Answering and Cross-Lingual Information Access, NII, Tokyo, Japan, 2005, 324–331

[47] Geng X., Liu T.Y., Qin T., Arnold A., Li H., Shum H.Y., Query Dependent Ranking Using K-nearest Neighbor, In Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '08, ACM, New York, NY, USA, 2008, 115–122, 10.1145/1390334.1390356

[48] Hu H., Zhu R., Wang Y., Feng W., Tan X., Huang J.X., A Best Match KNN-based Approach for Large-scale Product Categorization, in ACM SIGIR Workshop on eCommerce (SIGIR 2018 eCom Data Challenge), ACM, New York, NY, USA, 2018

[49] Wang X.l., Zhao H., Lu B.l., Enhanced K-Nearest Neighbour Algorithm for Large-scale Hierarchical Multi-label Classification, In Proceedings of the Joint ECML/PKDD PASCAL Workshop on Large Scale Hierarchical Classification, volume 5, Athens, Greece, 2011

[50] He B., Ounis I., A Study of Parameter Tuning for Term Frequency Normalization, In Proceedings of the Twelfth International Conference on Information and Knowledge Management, CIKM '03, ACM, New York, NY, USA, 2003, 10–16, 10.1145/956863.956867

[51] Taylor M., Zaragoza H., Craswell N., Robertson S., Burges C., Optimisation Methods for Ranking Functions with Multiple Parameters, In Proceedings of the 15th ACM International Conference on Information and Knowledge Management, CIKM '06, ACM, New York, NY, USA, 2006, 585–593, 10.1145/1183614.1183698

[52] Moniz N., Torgo L., Multi-Source Social Feedback of Online News Feeds, CoRR, 2018

[53] Greene D., Cunningham P., Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering, In Proc. 23rd International Conference on Machine learning (ICML'06), ACM Press, 2006, 377–384

A Appendices

A.1 Results of Experiment 1

Dataset Size	Tuning Time (s)		SU	TR %
	Conv.	Fast		
	BBC			
200	91.919	5.613	16.38	93.9
400	315.586	18.491	17.07	94.1
600	780.428	43.429	17.97	94.4
800	1370.827	73.632	18.62	94.6
1000	2150.680	116.208	18.51	94.6
1200	3044.546	164.330	18.53	94.6
1400	4056.449	214.020	18.95	94.7
1600	5388.106	281.662	19.13	94.8
1800	6739.049	351.180	19.19	94.8
2000	9196.227	477.491	19.26	94.8
AVG.			18.236	94.5
	News Popularity			
200	1.139	0.102	11.17	91.0
400	3.294	0.265	12.43	92.0
600	6.458	0.440	14.68	93.2
800	12.480	0.908	13.74	92.7
1000	19.689	1.414	13.92	92.8
1200	27.771	1.997	13.91	92.8
1400	37.559	2.677	14.03	92.9
1600	48.487	4.572	10.61	90.6
1800	61.729	4.738	13.03	92.3
2000	78.586	5.741	13.69	92.7
AVG.			13.685	92.63

Table 8: Experiment 1; fixed parameter sets; variable datasets.

A.2 Results of Experiment 2

Results of Experiment 2 - 1

|K|	Tuning Time (s)		SU	TR %
	Conv.	Fast		
2	438.559	27.265	16.09	93.8
4	996.262	52.927	18.82	94.7
6	1596.518	76.357	20.91	95.2
8	1942.167	95.429	20.35	95.1
10	2217.448	118.125	18.77	94.7
12	2594.170	140.630	18.45	94.6
14	3033.776	163.186	18.59	94.6
16	3450.672	185.262	18.63	94.6
18	3880.480	207.484	18.70	94.7
20	4315.408	230.489	18.72	94.7
AVG.			18.803	94.67

Table 9: Results of experiment 2-1; BBC dataset ($|D| = 1000$); fixed dataset size, fixed B and T, variable K

|K|	Tuning Time (s)		SU	TR %
	Conv.	Fast		
2	97.506	6.750	14.45	93.1
4	196.220	13.081	15.00	93.3
6	285.090	19.163	14.88	93.3
8	377.503	25.247	14.95	93.3
10	476.304	32.304	14.74	93.2
12	555.022	39.085	14.20	93.0
14	661.839	46.485	14.24	93.0
16	833.548	58.817	14.17	92.9
18	882.841	63.738	13.85	92.8
20	967.177	70.047	13.81	92.8
AVG.			14.429	93.07

Table 10: Results of experiment 2-2; News Popularity dataset ($|D| = 5000$); fixed dataset size, fixed B and T, variable K

Results of Experiment 2 - 2

|B|	Tuning Time (s)		SU	TR %
	Conv.	Fast		
	BBC ($\|D\|$ = 1000)			
2	429.376	25.850	16.61	94.0
4	876.606	49.647	17.66	94.3
6	1391.142	79.105	17.59	94.3
8	1913.551	105.117	18.20	94.5
10	2393.510	126.706	18.89	94.7
12	2866.498	154.668	18.53	94.6
14	3352.408	179.058	18.72	94.7
16	3818.868	210.430	18.15	94.5
18	4293.188	226.813	18.93	94.7
20	4762.854	254.429	18.72	94.7
AVG.			18.2	94.5
	News Popularity ($\|D\|$ = 5000)			
2	93.004	6.954	13.37	92.5
4	179.900	12.646	14.23	93.0
6	269.904	18.908	14.27	93.0
8	362.696	25.295	14.34	93.0
10	456.562	31.947	14.29	93.0
12	539.946	38.693	13.95	92.8
14	630.192	45.896	13.73	92.7
16	721.767	53.464	13.50	92.6
18	810.643	60.414	13.42	92.5
20	908.746	69.241	13.12	92.4
AVG.			13.822	92.75

Table 11: Results of experiment 2-2; fixed dataset size, fixed K and T, variable B

Results of Experiment 2 - 3

|T|	Tuning Time (s)		SU	TR %
	Conv.	Fast		
	BBC			
2	475.908	126.720	3.76	73.4
4	965.548	127.827	7.55	86.8
6	1428.281	128.409	11.12	91.0
8	1909.354	126.799	15.06	93.4
10	2375.907	129.023	18.41	94.6
12	2862.697	127.654	22.43	95.5
14	3346.357	125.495	26.67	96.2
16	3831.825	129.781	29.53	96.6
18	4287.822	127.303	33.68	97.0
20	4775.804	129.795	36.79	97.3
AVG.			20.5	92.18
	News Popularity			
2	90.610	29.254	3.10	67.7
4	179.550	29.658	6.05	83.5
6	269.697	30.166	8.94	88.8
8	359.262	30.420	11.81	91.5
10	449.353	31.573	14.23	93.0
12	540.550	32.583	16.59	94.0
14	629.257	33.084	19.02	94.7
16	763.427	34.164	22.35	95.5
18	878.833	35.827	24.53	95.9
20	978.757	37.836	25.87	96.1
AVG.			15.249	90.07

Table 12: Results of experiment 2-3; News Popularity dataset ($|D|$ = 5000); fixed dataset size, fixed K and B, variable T

A.3 Results of Experiment 3

K, B, T Size	Tuning Time (s)		SU	TR %
	Conv.	Fast		
2	18.287	7.633	2.40	58.3
4	159.185	22.969	6.93	85.6
6	511.590	47.124	10.86	90.8
8	1191.031	82.581	14.42	93.1
10	2296.797	131.775	17.43	94.3
12	4003.506	185.129	21.63	95.4
14	6349.800	253.599	25.04	96.0
16	9574.152	329.810	29.03	96.6
18	13660.224	424.822	32.16	96.9
20	18699.881	520.614	35.92	97.2
AVG.			19.582	90.42

Table 13: Results of experiment 3; BBC dataset; fixed dataset size; simultaneously varying parameter sets K, B, and T

K, B, T Size	Tuning Time (s) Conv.	Fast	SU	TR %
2	0.918	0.446	2.06	51.4
4	5.044	1.065	4.74	78.9
6	16.111	1.909	8.44	88.2
8	36.337	3.583	10.14	90.1
10	72.520	5.382	13.47	92.6
12	123.088	8.282	14.86	93.3
14	213.557	11.853	18.02	94.4
16	297.604	15.551	19.14	94.8
18	428.518	18.705	22.91	95.6
20	606.055	26.421	22.94	95.6
AVG.			13.672	87.49

Table 14: Results of experiment 3; News Popularity dataset; fixed dataset size; simultaneously varying parameter sets K, B, and T

A.4 Results of Experiment 4

Dataset Size	Tuning Time (s) Conv.	Fast	SU	TR %
	BBC			
100	138.075	6.951	19.86	95.0
200	511.316	27.972	18.28	94.5
300	1076.775	60.163	17.90	94.4
400	1773.102	99.562	17.81	94.4
500	2806.150	158.154	17.74	94.4
600	4249.985	213.543	19.90	95.0
700	5376.069	311.219	17.27	94.2
800	7162.647	410.314	17.46	94.3
900	9454.979	553.155	17.09	94.1
1000	12472.158	654.435	19.06	94.8
AVG.			18.237	94.51
	News Popularity			
500	31.107	1.585	19.63	94.9
1000	100.966	7.081	14.26	93.0
1500	211.485	19.307	10.95	90.9
2000	379.301	31.952	11.87	91.6
2500	589.498	47.611	12.38	91.9
3000	865.104	72.220	11.98	91.7
3500	1237.458	97.362	12.71	92.1
4000	1680.611	129.273	13.00	92.3
4500	2186.986	159.612	13.70	92.7
5000	2752.455	201.666	13.65	92.7
AVG.			13.413	92.38

Table 15: Results of experiment 4; 5-fold cross-validation; fixed sets of parameters; variable-size dataset samples