Efficient k-nearest neighbor search based on clustering and adaptive k values

Antonio Javier Gallego, Juan Ramón Rico-Juan , Jose J. Valero-Mas

PII:	S0031-3203(21)00536-7
DOI:	https://doi.org/10.1016/j.patcog.2021.108356
Reference:	PR 108356

To appear in:	*Pattern Recognition*

Received date:	6 March 2021
Revised date:	21 August 2021
Accepted date:	27 September 2021

Highlights

- Efficient kNN search based on feature learning, clustering, and adaptive k values.
- Several proposals to automatically optimize the search parameters.
- Comprehensive experimentation with 10 datasets of different typology and size.
- Results demonstrate that the proposal outperforms state-of-the-art methods.

Efficient k-nearest neighbor search based on clustering and adaptive k values

Antonio Javier Gallego[a,*], Juan Ramón Rico-Juan[a], Jose J. Valero-Mas[a]

[a] *Departamento de Lenguajes y Sistemas Informáticos, Universidad de Alicante, Carretera San Vicente del Raspeig s/n, Alicante, 03690, Spain*

Abstract

The k-Nearest Neighbor (kNN) algorithm is widely used in the supervised learning field and, particularly, in search and classification tasks, owing to its simplicity, competitive performance, and good statistical properties. However, its inherent inefficiency prevents its use in most modern applications due to the vast amount of data that the current technological evolution generates, being thus the optimization of kNN-based search strategies of particular interest. This paper introduces the caKD+ algorithm, which tackles this limitation by combining the use of feature learning techniques, clustering methods, adaptive search parameters per cluster, and the use of pre-calculated K-Dimensional Tree structures, and results in a highly efficient search method. This proposal has been evaluated using 10 datasets and the results show that caKD+ significantly outperforms 16 state-of-the-art efficient search methods while still depicting such an accurate performance as the one by the exhaustive kNN search.

Keywords: k-Nearest Neighbor, Efficient search, Clustering, Feature learning

1. Introduction

The k-Nearest Neighbor (kNN) rule is one of the most well-known search and classification algorithms in the supervised Pattern Recognition field [1]. Most of its popularity is due to its conceptual simplicity and straightforward implementation as it only requires the definition of a dissimilarity function for comparing and indexing the elements in the training set. Moreover, kNN presents excellent statistical properties as its theoretical error is bounded by twice the Bayes error when considering a sufficient number of samples [2]. Hence, this algorithm has been largely applied to a wide range of disparate fields, being some recent examples the areas of health [3], industry [4] or electronics [5].

As a representative example of instance-based supervised method, kNN relies on directly using the training samples for performing the search task rather than deriving a model out of them [6]. Such

*Corresponding author: Tel.: +349-65-903772; Fax: +349-65-909326
Email addresses: jgallego@dlsi.ua.es (Antonio Javier Gallego), juanramonrico@ua.es (Juan Ramón Rico-Juan), jjvalero@dlsi.ua.es (Jose J. Valero-Mas)

principle implies that the algorithm does not create a global target function to deal with the entire instance space but instead it works on local approximations to that function which only consider the elements in the neighborhood of the query. Note that this principle is generally considered as an advantage compared to other learning paradigms, at least when modelling large and complex functions [7].

However, the main counterpart of the instance-based paradigm is its inherent inefficiency and lack of scalability since all the instances of the training corpus must be consulted when a query is produced [8]. While this limitation may be obviated in situations with a reduced amount of data, kNN becomes intractable as the size of the corpora to deal with becomes larger and larger. This fact may somehow constrain the use of such algorithms in modern applications due to the rapid creation and growth of information in our modern society [9].

These drawbacks have been extensively studied in the literature, resulting in a wide range of proposals which aim at reducing the total number of searches and, hence, speeding up the process. In a broad sense, these approaches work on the premise of either eliminating spurious instances without ideally affecting the scheme performance [10], taking advantage of the metrical properties of the dissimilarity measure considered for avoiding unnecessary distance computations [11] or creating search structures for fast prototype consulting [12]. Nevertheless, the applicability of some of these schemes can be severely limited by the dimensionality of the feature space considered [13]: for example, when considering K-Dimensional Tree (KD Tree) structures [14], if the dimensionality of the data is very large, the performance can degrade up to an inefficient scenario as the one of the exhaustive search of the original kNN algorithm [15].

Recent advances in neural networks in general, and the so-called Deep Neural Networks (DNN) and Convolutional Neural Networks (CNN) in particular, have meant a great advance in the ability to learn appropriate features for Pattern Recognition tasks [16]. Instead of using automated heuristic processes or human experts in feature extraction, DNNs are trained to learn a suitable representation for a given task from the raw input data in a process known as *feature learning* [17]. Such techniques have dramatically improved the state of the art in various fields and applications, especially when dealing with unstructured data, as occurs with audio, image, and video recognition tasks [18].

In this sense, it is possible to use the DNN as a feature extractor to obtain a suitable mid-level vector representation (also called Neural Codes or NC [19]) that is later used as input to a search algorithm such as kNN [20]. This is done by feeding the network with the raw data and extracting the NC from one of the last layers of the network, most commonly the penultimate one [21, 22]. In fact some studies such as the works by Ren et al. [23] and Gallego et al. [24] show that the combination of NC and kNN as a classifier improves the results obtained by a sole DNN architecture even when applied in the same domain. Moreover, this combined approach of DNN and kNN has several additional advantages: a first one is that the use of kNN instead of the typical fully-connected layer in DNN extends the use of these techniques from pure classification scenarios to search cases; additionally, it improves the representation of the data, resulting in much better performance rates than if kNN was applied directly to the original data; a third advantage is that it allows adjusting the size of the layer from which the NCs are extracted, making it possible to reduce the feature size of the elements and thus solving the aforementioned dimensionality issue.

This paper takes as starting point the ckNN+ proposal by Gallego et al. [25] for efficient kNN search and classification and proposes a method called caKD+ which further improves its efficiency and efficacy by combining the use of DNN with a clustering-based search approach. In general lines, caKD+ first uses a clustering algorithm to reduce the search space by assuming that the k-neighbors of each prototype are in the same cluster. In order to guarantee such premise, a cluster augmentation technique which basically consists of adding the k-closest prototypes to each sample within the same cluster they belong to is applied. Moreover, rather than using a global k value for the entire corpus or a local k value for each prototype as some works propose [26, 10, 27], it applies a preprocess to optimize the value of k at the cluster level, thus avoiding over-adaptation problems and also improving the performance rate. Finally, this method also considers the use of KD Tree data structures to further speed up the search for centroids and prototypes.

As introduced, the proposed caKD+ method comprises a series of complementary processes for addressing the efficiency issue in kNN. On the one hand, the use of DNN allows us to learn a good internal representation, which improves the clustering process and, therefore, the precision of the search. On the other, the disadvantages of the approximate search given by the clustering process are compensated by the good representation of the data as well as by the cluster augmentation process, an adaptive k value, and the use of KD Tree structures.

The proposed method was evaluated using 10 different datasets, with a distinct number of samples (comprising small, medium, and large-scale scenarios) and classes, and including different types of data, such as images and feature vectors of diverse sizes. In addition, 6 DNN architectures with distinct numbers of layers, types of layers, and parameters were also evaluated. The results show that the proposed method improves both the efficiency and the efficacy of ckNN+ and of 16 representative state-of-the-art methods.

Another contribution of this work is a method for the automatic selection of the search parameters. Since there are two criteria that can be optimized (efficiency and efficacy), which are often opposed because attempting to optimize the result of one worsens the result of the other, it is necessary to establish a mechanism for their selection. This proposal achieves optimal results without having to evaluate all possible combinations of parameters, which considerably speeds up training time.

In summary, this work presents the following contributions:

- An efficient kNN search method based on clustering, adaptive k values, and pre-calculated data structures to perform similarity search and classification. It improves both the efficiency and efficacy of different state-of-the-art methods considered for the same task.

- New proposals to automate the selection of the parameters of the method, including the number of clusters and the k search values per cluster.
- Comprehensive experimentation with 10 diverse-size datasets of different typology and 6 different DNN and CNN architectures, with an in-depth analysis of the reported results supported by statistical significance tests and a comparison with 16 different efficient kNN search state-of-the-art approaches.

The remainder of the paper is structured as follows: Section 2 provides a brief review of the state of the art of efficient search methods for kNN; after that, Section 3 presents the proposed approach; then, Section 4 shows a description of the datasets, DNN architectures, and metrics used in the evaluation; Section 5 reports the evaluation results; after that, Section 6 provides a general discussion about the insights gathered from the experimentation; finally, our conclusions and future work are addressed in Section 7.

2. Background on efficient kNN search

As introduced, the kNN rule does not create a model out of the training data but instead it consults the complete training set every time a new query is produced. This characteristic results in both a high memory consumption and low efficiency figures, especially when the size of the corpus is relatively high. In this regard, several strategies have been posed to palliate this drawback, which are generally divided into three categories [28]:

- Fast Similarity Search (FSS) [29]: This is a family of methods that uses an auxiliary structure on the training data to perform quick searches without reducing the overall performance rate.
- Data Reduction (DR) [10]: These approaches encompass the subset of data preprocessing techniques whose objective is to reduce the size of the initial training set without significantly affecting the overall performance.
- Approximated Similarity Search (ASS) [30]: This approach is based on the premise of searching the training set for prototypes that are sufficiently similar for a given query, rather than searching for the closest exact instance. These methods typically also result in a slight decrease in the overall performance of the scheme.

The FSS family of methods is further divided into indexing algorithms [14] and the subfamily of Approximating and Eliminating Search Algorithms (AESA) [31]. The indexing algorithms divide the search space into partitions that are referenced from an auxiliary structure (usually hierarchical) in order to achieve an efficient search. When searching for a new element, certain partitions are selected or discarded according to various criteria, and an exhaustive search is then made within the selected partitions. This implies that only a subset of the training set has to be searched to classify a new instance. Some examples of methods that employ tree-like hierarchical structures are K-Dimensional Tree (KD-Tree) structures [14], BallTree [32], metric-trees [33] and, more recently, the EPBST and RPBST methods, which are based on binary trees [34]. The main problem with these methods is that they are extremely sensitive to dimensionality [15], in addition to the fact that they require the input data to be represented as feature vectors. In contrast, AESA-type algorithms require only a metric space as input, that is, one in which a dissimilarity between pairs of prototypes can be defined [35, 11]. These strategies generally use the triangle inequality property and pre-calculated distances to reject prototypes that cannot be the closest neighbor to a given prototype. Their main disadvantages are that they focus solely on searching for the closest nearest neighbor and that they become inefficient with large datasets.

Regarding the DR paradigm, this family is also subdivided into two general approaches: (i) Prototype Generation, which studies the creation of new artificial prototypes from the original ones to subsequently replace them; and (ii) Prototype Selection, which aims at selecting a representative subset of the original prototypes, being the rest of them discarded [36]. Being the so-called Condensed Nearest Neighbor (CoNN) [37] one of the first proposals in DR, a large variety of methods may now be found in literature under the generation [38] and selection paradigms [39]. More recently, it is possible to find proposals such as the Instance Reduction Algorithm using Hyperrectangle Clustering [40], Reduction through Homogeneous Clusters (RHC) [41], or Edited Natural Neighbor [42]. In general, the main drawback of these methods is that they cause a significant loss of performance precision [10] and several strategies have, therefore, been proposed to address those deficiencies, such as considering the boosting schemes [43], the fusion of feature and prototype selection using genetic algorithms [44, 38], or considering the results of the selection or generation algorithms as guidelines in order to focus the search exclusively on the most promising classes [8].

In terms of the ASS family of methods, one of the most popular techniques is the use of *hashing* approaches to code the nearby prototypes in the training set. Some examples of this technique are Product Quantization (PQ) [45], Spectral Hashing (SH) [46] or Local Sensitive Hashing (LSH) forest [47]. Other type of approaches within this family are cluster-based search methods [48, 25] or those that consider approximate KD Tree structures for searches, such as the Fast Library for Approximate Nearest Neighbors [49].

Furthermore, there are proposals that focus only on improving the kNN performance precision. Traditional algorithms use a global k for the processing of the entire training set. However, the representation of the data is not homogeneous in the whole space spanned by the training set. For this reason, some proposals perform a learning preprocess that selects an optimal k for each sample rather than using a generic one [26, 27]. However, these approaches may also have the problem of an over-adaptation to the training feature space that has to be corrected by, for example, weighting with the global k or using the average of the optimal k values of the closest samples in an area [10]. These proposals do not, however, reduce the high computational cost, which is the main disadvantage of kNN, but merely help to slightly improve the performance rate.

The proposed caKD+ method could be cataloged within the ASS family, since it performs an approximate search based on clusters. It also uses techniques focused on improving classification precision, such as adaptive k and cluster augmentation. In this case, the problem of over-adaptation to the training set is avoided since k values are calculated per cluster and not per prototype. Furthermore, this method is also combined with FSS techniques (specifically KD Tree structures) in order to further accelerate the search for centroids and prototypes within clusters. Overall, the proposed method improves the results of the state of the art in terms of both efficiency and efficacy.

3. Method

This section presents the proposed caKD+ method for efficient kNN search. This method is divided into two phases (see Figure 1): (i) a training stage in which the training data is preprocessed and prepared to accelerate the search; and (ii) an inference stage in which the data from the training stage is used to search for new prototypes. The method initially uses an encoder (a particular type of DNN scheme) to perform the feature learning process and thus improving the results of the subsequent clustering process. The centroids of these clusters are pre-calculated and a series of additional processes for both improving the classification results and speeding up the search process are applied to them. The following sections explain the different steps of the algorithm in detail. Note that, while the necessary mathematical definitions are provided in the different sections, Appendix A also lists this nomenclature for the sake of clarity.

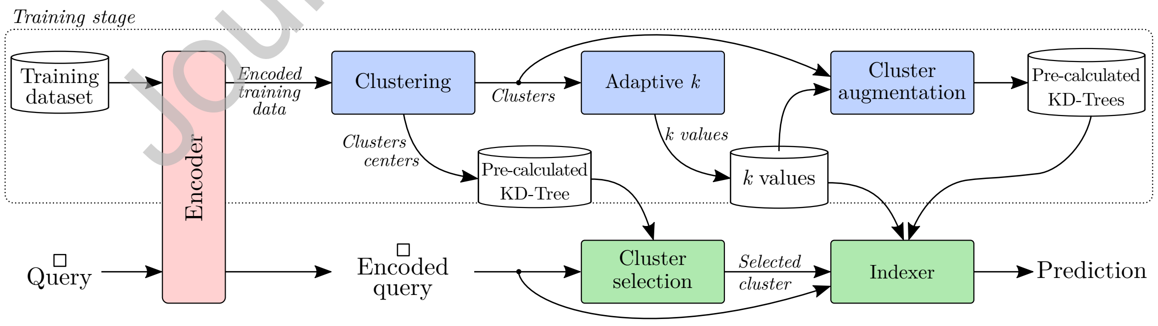

Figure 1: Outline of the proposed caKD+ method.

3.1. Data encoder

In this step of the algorithm, a DNN is used to transform the feature space of the input data into a smaller dimensional space that also improves the grouping of the intrinsic features of the data. This new representation of the elements in the corpus is also known as Neural Codes (NC) [19].

The justification for this proposal can be explained in the following terms. When trained for supervised classification, the layers of a DNN eventually extract a set of features that are suitable for the task at hand, in which the last layer simply learns a linear mapping to provide each possible category of the classification domain with a probability. Hence, the DNN layers prior to the last one behave like feature extractors that map the input data onto a new space in which the categories are expected to be linearly separable. This is particularly interesting in the case of clustering-based search and classification since the distribution of samples in the space is a key aspect in the creation of clusters. Furthermore, since the size of the layers is adjustable, it is also possible to reduce the dimensionality of the extracted features.

The first step in the proposed algorithm is, therefore, to train a DNN in a supervised fashion by providing pairs containing the input samples and their labels. Let $\mathcal{T} = \{(x_m, y_m))\}_{m=1}^{|\mathcal{T}|}$ be the set of $|\mathcal{T}|$ training data where each sample x_m drawn from space $\mathcal{X} \in \mathbb{R}^D$ has an associated label y_m from the set of possible labels $\mathcal{Y} = \{Y_1, \ldots, Y_L\}$. The DNN used for encoding (denoted by Enc) implements the function $Enc : \mathcal{X} \rightarrow \mathcal{Y}$ that maps an instance $x \in \mathcal{X}$ onto a label of $\mathcal{Y}$. The process of training Enc consists of adjusting the set of network weights using the training set $\mathcal{T}$ to minimize the classification error of the network according to a given loss function $\mathcal{L}$ [16].

Once Enc has been trained, it is used to obtain the set of encoded training data. This is done by forwarding the samples of set $\mathcal{T}$ through this network to extract the feature vectors from a user-defined feature layer, denoted as Enc^F. That is, $Enc^F : \mathcal{X} \rightarrow \mathbb{R}^N$ maps the D-dimensional instance $x \in \mathcal{T}$ onto a new N-dimensional feature space $\mathcal{X}^F \in \mathbb{R}^N$ representation, which constitutes the aforementioned NC encoding. Note that in the proposed method we consider the last hidden layer of the architecture for Enc^F. Besides, the mapped elements from $\mathcal{T}$ are stored in the set $\mathcal{T}^F$ for its further use in the subsequent steps of the algorithm.

3.2. Clustering process

Using clusters to subdivide the search space is an efficient, albeit approximate, strategy to speed up the search process. This technique consists of grouping the data collection $\mathcal{T}^F$ in c different partitions or sets, i.e. $\mathcal{C} = \{C_1, \ldots, C_c\}$, in an attempt to minimize certain specific criteria, which depends on the particular clustering strategy, being each cluster represented by a single element, usually its centroid. The search for the k-nearest neighbors of a sample, therefore, consists of locating the closest cluster using just their respective centroids for then performing the exhaustive search within the prototypes of that cluster.[1]

We propose to perform the clustering process by employing the c-means algorithm [50, 51], since it is one of the most widely used algorithms for this task.[2] Considering the set of data $\mathcal{T}^F$, this method retrieves a set of cluster centroids $\mathcal{C}^c = \{C_1^c, C_2^c, ..., C_c^c\}$ by performing the following optimization process:

$$\arg\min_{\mathcal{C}} \sum_{i=1}^{c} \sum_{x \in C_i} \|x - C_i^c\|_2^2 \tag{1}$$

where C_i^c represents the i-th cluster centroid from set $\mathcal{C}^c$ and C_i the set of elements from $\mathcal{T}^F$ assigned to that cluster, i.e., $C_i = \left\{x_j \in \mathcal{T}^F : \arg\min_i d\left(x_j, C_i\right)\right\}$ with $1 \leq j \leq |\mathcal{T}^F|$ and $1 \leq i \leq |\mathcal{C}|$.

It must be pointed out that the considered method retrieves a set of disjoint clusters which satisfy $\bigcup_{i=1}^{c} C_i = \mathcal{T}^F$ as well as $\bigcap_{i=1}^{c} C_i = \emptyset$. As it will be later studied, since this particularity usually implies a drop in terms of efficacy for search-based tasks, we will propose a strategy for palliating this effect by allowing some overlap among the clusters.

Regarding the initialization of the cluster values for the optimization process, we considered the *c-means++* algorithm proposed by Arthur & Vassilvitskii [52] as it is reported to achieve better results compared to a random-based initialization. The following subsection discusses how to select the number of clusters c in order to optimize the result of the algorithm.

Note that, as an approximate-search method, the considered clustering-based search process usually entails a decrease in terms of efficacy. This effect is usually due to the fact that the obtained clusters do not contain all the relevant instances for properly classifying each element within the cluster. Thus, in order to palliate this issue, a cluster augmentation process which shall be later explained is introduced in the pipeline of the caKD+ method.

Once the clusters have been calculated, and with the aim of further accelerating the search process during the inference stage, our caKD+ proposal introduces the use of KD Tree structures [53] for constructing efficient search arrangements. These structures are created at two levels: (i) a first single tree in which only the centroids obtained from the clustering process are considered; and (ii) a set of trees for efficiently searching within the prototypes of different clusters. These data structures are pre-calculated during the training stage for its use in the inference stage.

It is important to highlight that, besides the c-means method, other more sophisticated clustering policies may report additional benefits to the caKD+ proposal. For instance, the use of efficient hierarchical clustering techniques [54, 55, 56] could imply the removal of some data search structures used in this work at the expense of a stratified disposition of the data.

[1]Since in some particular cases the number of instances resulting in a cluster may be lower than the k search parameter, our implementation limits the value of this parameter to the size of the cluster.

[2]This method is also known in the literature as k-means but could, in this text, lead to confusion with the parameter k for the calculation of the k-nearest neighbor.

Finally, it should be highlighted that the application of the clustering method on the mapped $\mathcal{T}^F$ set yields to better performance rates than if applied to the original space representation $\mathcal{T}$. More precisely, as it will be shown in the experimental section of the work, the partitions obtained in this alternative representation allow obtaining very efficient search structures which can be easily exploited by the rest of the stages in the method.

3.2.1. Selection of the number of clusters

The number of clusters (parameter c) is critical for the performance of the algorithm, both in terms of efficacy — measured as the search accuracy — and efficiency of the scheme. The ckNN+ method [25] does not provide any indication on how to select this parameter automatically, but it only evaluates the effect of different cluster sizes on the two criteria mentioned. Thus, a proposal with which to automate the selection of this parameter is of particular interest.

However, note that it is not possible to optimize both criteria — efficacy and efficiency — at the same time as they generally constitute opposite objectives: reducing the number of pairwise distances to compute between the query and the training data may imply some performance loss whereas, if higher precision is pursued, it will be necessary to calculate a greater number of distances. In this sense, we now present our two proposals for optimizing each of the aforementioned success criteria for a cluster-based efficient kNN search scheme:

Optimize search efficiency. Regarding the efficiency of the algorithm, it is important to remark that the total number of computed distances does not always decrease as the number of clusters increases. When performing a search, first the closest centroid is retrieved (from among the c possible centroids), after which the algorithm searches within the prototypes of the corresponding cluster, which have an average of m/c samples, being m the size of the training set. Thus, for a given query, the average number of calculated distances in a cluster-based efficient kNN scheme is given by the following expression:

$$d(c) = c + \frac{m}{c}\ . \tag{2}$$

Since m is a fixed parameter, the number of computed distances depends only on the number of clusters c, signifying that $c = \sqrt{m}$ clusters would have to be created to minimize the expected number of operations, case in which an average of $d(c = \sqrt{m}) = 2\sqrt{m}$ distances are calculated. Finally, note that as the c parameter moves away from the $c = \sqrt{m}$ optimal point, the efficiency value of the cluster-based scheme degenerates. The limit cases are those in which the c is set to either the unit (one single cluster) or m (as many clusters as data instances), computing in both cases a number of pairwise distances of $d(c = 1) = d(c = m) = m+1$, which equals to the exhaustive kNN search.

Optimize search efficacy. With regard to the efficacy optimization of the caKD+ algorithm, the Elbow method [57] is proposed to discover the appropriate amount of clusters. This is a heuristic method that performs clustering using the c-means algorithm for each c value and which analyzes the percentage of variance explained by the clusters against the number of clusters. The optimal value of c is selected when the marginal gain decreases by adding a new partition (it creates an angle on the graph, hence its name).

While the main counterpart of this method is its reported inefficiency as it exhaustively assesses each clustering value c until the optimum is found, in the current proposal this does not suppose any inconvenience as this step is done during the training stage, thus not affecting the inference phase. Other alternative optimization methods such as the one by Chowdhury et al. [58] perform this search in a more efficient manner, but we discarded them since they either lead to sub-optimal solutions or require an additional set of parameters.

3.3. Adaptive k values

When considering any kNN-based retrieval task, it is necessary to determine the appropriate value of k as it directly influences the precision of the result. One possible solution is to determine this parameter by performing cross validation on the training set. However, it is quite likely that the same value of k would not be optimal for the whole space spanned by the training set. That is, different regions of the feature space may require different values of k.

As indicated in the introduction section, there are various approaches in literature with which to calculate local k values for each prototype [26, 27]. However, this level of detail or over-adaptation usually leads to problems for which there are only partial solutions as, for instance, smoothing (that is, weighting local k values by employing a global k).

Since in our case the space is subdivided into clusters, we propose the use of values of k adapted for each cluster. In our experimentation, this approach obtained better results than the use of global and local k values, perhaps because it is an intermediate solution that prevents the over-adaptation problem. Note that such solution differs from the one for the ckNN+ method [25] since that work only assessed the effect of applying different values of k.

The leave-p-out cross-validation (LpOCV) estimation method is proposed to calculate the value of k adapted for each cluster. In general, it is recommended to consider a low value of p to obtain a more accurate result. However, if the number of elements in the corpus is very large or in cases in which the training time is critical, a higher value of p can be used. In our experimentation, since training time is not a critical parameter in this case, a value of $p = 1$ was used.

For each cluster, the value of k that obtains the best average result is selected using the LpOCV method. It should be remarked that, when the different k values achieve the same success rate, the lowest k parameter is selected to reduce the computational load during the inference stage.

Finally, note that all these processes are pre-calculated during the training stage and stored in a set $\mathcal{K}^c = \{k_1, k_2, ..., k_c\}$ of adaptive k values for each of the c clusters in the training set for their use during the inference phase.

3.4. Clustering augmentation

The use of a cluster-based search approach generally entails a noticeable increase in terms of search efficiency paired with a decrease in the performance figures as it performs an approximate search. While mechanisms such as the commented feature transformation or the use of adaptive k values may somehow palliate this effect, in the cases in which the search prototype is located among two or more clusters, the search process may lead to search and classification errors. Thus, with the idea of improving the performance of the scheme at the expense of marginally decreasing the efficiency, we propose to slightly increase the individual size of the disjoint clusters $\mathcal{C} = \{C_1, \ldots, C_c\}$, so that some overlapping occurs among their borders, hence retrieving set $\mathcal{C}^a = \{C_1^a, \ldots, C_c^a\}$ of augmented clusters.

The gist behind this process is that, for each prototype $x \in \mathcal{T}^F$ of a certain cluster $C_i \in \mathcal{C}$, the procedure searches for its closest neighbors in the entire set $\mathcal{T}^F$ considering the corresponding $k_i \in \mathcal{K}^c$ adaptive k value of the cluster. Mathematically, considering a $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_0^+$ dissimilarity metric, the i-th cluster C_i is augmented using the following expression:

$$C_i^a = C_i \cup \arg\min_{p \in \mathcal{T}^F} {}_{k_i} \{d(x, p)\} \quad \forall x \in C_i \tag{3}$$

where the procedure retrieves the k_i closest elements in set $\mathcal{T}^F$ for each element $x \in C_i$ which, together with the members of C_i, constitute the augmented set C_i^a. This process is repeated for all groups in $\mathcal{C}$, hence obtaining the collection $\mathcal{C}^a$ of augmented clusters.

Finally, note that, in opposition to initial set $\mathcal{C}$, the clusters in $\mathcal{C}^a$ may not be disjoint, i.e., $\bigcap_{i=1}^{c} C_i^a \neq \emptyset$. We will later study the implications that this process entails on the overall success of the proposed method.

3.5. Training stage

Algorithm 1 shows the formalization of the entire training process described in the previous sections. The algorithm receives the training set $\mathcal{T}$ and the encoder network topology as input, and returns the trained encoder as output, along with the set $\mathcal{K}^c$ of adapted k values, and sets KDt^c and KDt^a of KD Tree structures for performing the efficient centroids and prototypes search, respectively.

First, the DNN topology used as an encoder, i.e. *Enc*, is trained (line 1). The prototypes of the set $\mathcal{T}$ are then forwarded through the encoder (line 2) in order to extract their NC representation by using the feature layer Enc^F, and they are stored in the set $\mathcal{T}^F$. The number of clusters c is then selected (line 3), by using either one of the methods proposed in Section 3.2.1 or setting this value manually. The clustering process is subsequently performed on $\mathcal{T}^F$ using the number of clusters c (line 4), the result of which is the sets of centroids $\mathcal{C}^c$ and clusters $\mathcal{C}$. These data are then employed to construct the KD Tree structure KDt^c that will be used to perform the centroid search (line 5), after which it iterates through all the clusters (line 6) applying the adaptive k (line 7) and the cluster augmentation (lines 8-11) processes. Finally, the KD Tree structure KDt_i^a that will be used to search for prototypes in that cluster is constructed for each augmented cluster C_i^a (line 12).

Algorithm 1: caKD+ training stage

```
Input  : $\mathcal{T} \leftarrow \{(x_m, y_m)\}_{m=1}^{|\mathcal{T}|}$
         $Enc \leftarrow$ Deep Neural Network topology
Output: $Enc, \mathcal{K}^c, KDt^c, KDt^a$
1  $Enc \leftarrow$ Fit $Enc$ with $\mathcal{T}$
2  $\mathcal{T}^F \leftarrow Enc^F(\mathcal{T})$                                   ▷ Section 3.1
3  $c \leftarrow$ cluster_size_criteria($\mathcal{T}^F$)                    ▷ Section 3.2.1
4  $\mathcal{C}^c, \mathcal{C} \leftarrow Clustering(\mathcal{T}^F, c)$                       ▷ Section 3.2
5  $KDt^c \leftarrow BuildKDTree(\mathcal{C}^c)$
6  foreach $i \in [1, \dots, c]$ do
7      $k_i \leftarrow AdaptiveK(C_i)$                                  ▷ Section 3.3
8      $C_i^a \leftarrow C_i$                                            ▷ Section 3.4
9      foreach $x \in D_i$ do
10         $C_i^a \leftarrow C_i^a \cup \arg\min_{p \in \mathcal{T}} {}_{k_i} \{d(x, p)\}$
11     end foreach
12     $KDt_i^a \leftarrow BuildKDTree(C_i^a)$
13 end foreach
```

Note that this algorithm only requires the training set $\mathcal{T}$, signifying that it can be performed as a preprocess before the inference stage. It does not, therefore, affect the efficiency of the search, but rather the opposite, thus allowing a much faster search.

In the remainder of the paper, we shall refer to this strategy as caKD+, where the letter "c" refers to the clustering process and "a" to the adaptive k, while the symbol "+" refers to the cluster augmentation process. During the experimentation, the different steps of the method will be evaluated separately and, following the commented notation, we shall indicate caKD the case in which the cluster augmentation is deactivated (avoiding lines 8 to 11 of the algorithm) and cKD when the adaptive k process is not considered either (line 7). It is important to mention that, since the comparison of a hybrid approach DNN-kNN against the individual kNN or DNN-based classifiers has already been addressed in some previous works [24, 59, 60], we shall skip this question to focus on the mentioned experiments to properly assess the efficient search scheme proposed.

3.6. Inference stage

The inference process for a given query comprises a series of steps which uses the data and structures prepared during training (see Figure 1). This process is formally described in Algorithm 2. The query sample q is initially forwarded through the learned encoder Enc to extract its feature vector q^F (line 1); after that, the pre-calculated KD Tree KDt^c is used to find the index i of the closest cluster, i.e., C_i^a (line 2); finally, the KD Tree KDt_i^a is used to search within C_i^a using the corresponding $k_i \in \mathcal{K}^c$ of adapted k values, retrieving the set of elements $\mathcal{N}_q \subseteq C_i^a$ (line 3).

Algorithm 2: caKD+ inference stage

```
Input  : $q, Enc, \mathcal{K}^c, KDt^c, KDt^a$
Output: $\mathcal{N}_q$
1 $q^F \leftarrow Enc^F(q)$
2 $i \leftarrow SearchKDTree(q^F, KDt^c, 1)$
3 $\mathcal{N}_q \leftarrow SearchKDTree(q^F, KDt_i^a, k_i)$
```

4. Experimental setup

This section describes the different corpora, network architectures, and evaluation metrics used for assessing the proposed strategy.

4.1. Assessment corpora and preprocessing

We evaluated the proposed method using a set of 10 different corpora of feature vectors and images with different number of samples, classes, and features. Note that, to validate the applicability of our proposal in several set-size scenarios, the choice of these corpora was meant to exemplify the different levels — small, medium, and large — established by Garcia et al. [39].[3] In the particular case of image datasets, pixel values were used as input features of the system. These datasets are now described as well as summarized in Table 1:

- Landsat [61]: Satellite images analyzed in four spectral bands in image neighborhoods of 3x3 pixels.
- Gisette [61]: Dataset of isolated handwritten digits that focuses on exclusively separating digits 4 and 9. 5000 features are generated for each digit by combining randomly selected pixels, higher-order features, and also adding a number of distractor features.

[3]Garcia et al. [39] proposed a taxonomy for data corpora based on their size, being *small* up to 2,000 prototypes, *medium* up to 20,000 elements, and *large* as any corpus above that.

- United States Postal Service (USPS) [62]: Images of single handwritten digits.
- Pendigits [61]: Compilation of handwritten isolated digits. The features of this dataset represent a sequence of 8 points (x, y) regularly spaced along the stroke made to draw the digit.
- Handwritten Online Musical Symbol (HOMUS) [63]: Image corpus of isolated handwritten music symbols.
- Letter [61]: Collection of handwritten examples of capital letters from the English language. The dataset features represent primitive numeric attributes extracted from each image, such as pixel counts, correlations, statistical moments, or edge counts.
- NIST Special Database (NIST) [64]: Image corpus of upper case handwritten letters.
- German Traffic Sign Recognition Benchmark (GTSRB) [65]: Collection of traffic sign images obtained from the real world in different sizes, positions, and lighting conditions.
- Math symbols dataset (Math) [66]: Corpus of images with isolated handwritten mathematical symbols.
- MNIST [67]: Dataset containing images of isolated handwritten digits.

Table 1: Summary of the different corpora used for the evaluation of the proposed method. For each dataset, the number of samples, classes, and features are shown, as well as their data type and size category according to the taxonomy by Garcia et al. [39]. In the case of the image datasets, $(c \times w \times h)$ denotes the image size in pixels, where c is the number of channels, w the width, and h the height.

Corpus	Description				
	Type	Samples	Size category	Classes	Features
Landsat [61]	Image	6,435	Small/medium	6	36 (4×3×3)
Gisette [61]	Feature vector	7,000	Small/medium	2	5000
USPS [62]	Image	9,298	Small/medium	10	256 (1×16×16)
Pendigits [61]	Feature vector	10,992	Medium	10	16
HOMUS [63]	Image	15,200	Medium	32	1600 (1×40×40)
Letter [61]	Feature vector	20,000	Medium	26	16
NIST [64]	Image	44,951	Large	26	1024 (1×32×32)
GTSRB [65]	Image	51,839	Large	43	4800 (3×40×40)
Math [66]	Image	56,000	Large	56	2025 (1×45×45)
MNIST [67]	Image	70,000	Large	10	784 (1×28×28)

Regarding the preprocessing of the input data, the pixel values of the MNIST, NIST, HOMUS, GTSRB, and Math images were divided by 255 for normalization, while the Gisette, Letter, Pendigits, Landsat, and USPS data were used without preprocessing since they are already normalized at their origin.

4.2. DNN architectures for feature learning

A total of 6 DNN architectures with different number of layers, types of layers, and parameters were used for the encoding stage. While these architectures were determined empirically for each dataset, they are based on proposed topologies for similar tasks from the literature. However, the goal is not to outperform the state of the art, but to obtain a network with similar classification results. As previously stated, these networks are used to transform the feature space of the input data into a smaller dimensional space that also improves the grouping of the data.

Table 2 shows the details of the specific network configurations used to process each of the datasets. Note that, although these architectures differ as regards the type of layers and the number of parameters, they all share two final fully-connected layers: $FC(N)$ from which the NCs of dimension N are extracted (this allows us to adjust the dimension representation), and a final layer $FC(L)$ with which to classify the L possible classes of each particular corpus.

Finally, these architectures were trained by means of standard backpropagation using Stochastic Gradient Descent [70] and considering the adaptive learning rate method proposed by Zeiler [71]. In the backpropagation algorithm, *categorical crossentropy* was used as the loss function between the DNN output and the expected result. The training stage lasted a maximum of 300 epochs with a mini-batch size of 32 samples.

4.3. Evaluation protocol and metrics

We used an n-fold cross validation (with $n = 5$) in all the experiments, since it yields a better Monte-Carlo estimate than when performing the tests solely with a single random partition [72]. The datasets were consequently divided into n mutually exclusive folds. For each fold, we used one partition for test (with 20 % of the samples) and the rest for training (80 %). A validation subset with 10 % of the training samples was additionally used for the adjustment of the hyperparameters. The training and testing processes were repeated $n = 5$ times, using the different partitions of the dataset, and finally, the average result was calculated.

The proposed method was evaluated by measuring both efficacy —in terms of the classification performance— and efficiency. Since the number of samples per class for some of the datasets used is not uniformly balanced, the metric used to evaluate the performance was the weighted average of the F_1 score obtained for each class. Taking one class of the corpus as positive and the rest as negative, F_1 is defined as:

Table 2: DNN network configurations considered for the mapping of the initial data features of the datasets onto their respective NC representations. Notation: $Conv(f, w \times h)$ stands for a layer with f convolution operators of size $w \times h$ pixels, $FC(n)$ represents a fully-connected layer of n neurons, $MaxPool(w \times h)$ stands for the max-pooling operator of dimensions $w \times h$ pixels, $Drop(d)$ implements a dropout stage with a value of d %, and $BatchNorm()$ applies a Batch Normalization layer [68]. ReLU [69] was used as the activation function for all layers except for the output layer, for which the Softmax activation function was used. NCs are extracted from the next-to-last layer with size N. The parameter L of the last layer stands for the number of classes in the dataset.

Dataset	CNN configuration							

MNIST
USPS
	Conv(32,3×3)	Conv(32,3×3)
		MaxPool(2×2)
		Drop(0.25)
			FC(N)
			Drop(0.5)
				FC(L)				

HOMUS
NIST
	Conv(256,3×3)
	MaxPool(2×2)
	Drop(0.2)
		Conv(128,3×3)
		MaxPool(2×2)
		Drop(0.2)
			Conv(128,3×3)
			Drop(0.2)
				Conv(64,3×3)
				Drop(0.2)
					FC(512)
					Drop(0.1)
						FC(256)
						Drop(0.1)
							FC(N)
							Drop(0.1)
								FC(L)

Letter
Pendigits
	FC(512)
	BatchNorm()
		FC(512)
		BatchNorm()
			FC(256)
			BatchNorm()
				FC(N)
				BatchNorm()
					FC(L)			

Gisette	FC(256)
	Drop(0.1)
		FC(256)
		Drop(0.1)
			FC(N)
			Drop(0.1)
				FC(L)				

Landsat	Conv(256,3×3)
	UpSamp(2×2)
	Drop(0.4)
		Conv(128,5×5)
		UpSamp(2×2)
		Drop(0.4)
			Conv(64,7×7)
			MaxPool(2×2)
			Drop(0.4)
				FC(N)
				Drop(0.4)
					FC(L)			

GTSRB
Math
	Conv(96,5×5)
	MaxPool(2×2)
		Conv(128,3×3)
		MaxPool(2×2)
			Conv(256,5×5)
			MaxPool(2×2)
				FC(512)
				Drop(0.2)
					FC(N)
					Drop(0.2)
						FC(L)		

$$F_1 = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

where TP, FP, and FN denote the number of true positives, false positives, and false negatives, respectively.

In order to assess the efficiency of the proposal, we discarded the runtime since this metric depends on factors that are not related to the method itself but on the programming language, the libraries used for the implementation, or the hardware settings, among others. Thus, as efficiency metric we considered the algorithmic cost $\mathcal{O}(D)$, where D stands for the number of pairwise computed distances. In terms of dissimilarity metric, we have resorted to the Euclidean distance since, as described above, the NC vector obtained from the feature learning step are numerical feature representations. While other dissimilarity functions could also be considered, like Manhattan or Mahalanobis [73], the original implementation of some of efficient search methods is based on the Euclidean metric. Finally, in order to facilitate comparison, in our experiments the cost is calculated as a percentage normalized to the highest cost, which is obtained when considering the exhaustive kNN search.

5. Experiments

In this section, the proposed method is evaluated using the corpora and network topologies described in the previous section. In order to comprehensively assess the proposal, we first study the performance according to the dimension of the NC to select an appropriate feature size value; then, we assess the influence of the cluster augmentation and adaptive k values stages on the overall performance of the system; after that we compare our proposal with the ckNN+ method; finally, a comparison against 16 state-of-the-art efficient kNN search methods is presented.

All the experiments were carried out using the Python programming language with the TensorFlow (v. 1.14) and Scikit-learn (v. 0.20) libraries. The machine used consists of an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz with 24 GB RAM, a Nvidia GeForce GTX 1080 GPU with the cuDNN library. Also note that, for the sake of reproducibility, all processes involved (clustering, network training, search structures...) consider a common fixed seed.

5.1. Evaluation of the NC dimensionality

One of the first parameters to select is the size of the intermediate representation of the data, that is, the dimensionality of the NC, which is determined by the parameter N of the DNN architecture used (see Table 2). We evaluated the influence of this parameter by performing an experiment in which we measured both classification rate as F_1 and efficiency/cost as the parameter N increases.

Figure 2 shows the results of this experiment, in which each point represents the average obtained after carrying out the cross-validation process with the 10 datasets considered. The horizontal axis represents the NC size from 1 to 4096 (on a logarithmic scale), while the vertical axis shows both F_1 and cost. Note that cost is obtained as a percentage referred to its maximum value given by $N = 4096$.

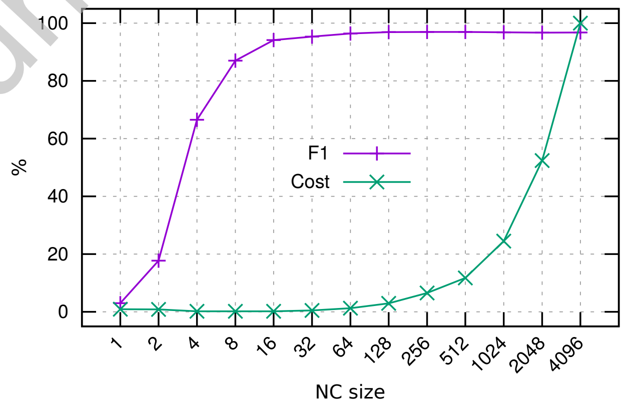

Figure 2: Performance in terms of both accuracy (F_1) and efficiency (cost) with respect to the size of the NC.

Assessing the results obtained, it may be checked that classification performance in terms of the F_1 metric stabilizes from size $N = 128$ while cost continues to increase exponentially. For this reason, an NC dimension of $N = 128$ will be used in the remaining experiments.

5.2. *Effect of cluster augmentation and adaptive k*

This section analyzes the influence of the cluster augmentation (marked with the "+" symbol in the name of the proposal, caKD+) and the adaptive k (denoted by the letter "a" in caKD+) processes in the overall performance. For performing such analysis we shall examine the different configurations obtained when activating and deactivating these processes separately.

Table 3 shows the results of this experiment (in terms of F_1 and distance computations) for cluster sizes c in the range $[10, 1000]$. Distance calculations are shown as percentages with respect to the exhaustive kNN search. Moreover, in order to facilitate the identification of whether each process is applied, they are marked using the following symbols: ▲ and △ indicate whether or not the cluster augmentation process is applied, respectively, while ● and ○ denote whether the adaptive k process is applied or not, respectively.

Table 3: Comparison of the improvement obtained by applying the cluster augmentation and the adaptive k processes. For this, the result obtained by applying these processes is compared with that obtained if they are not applied. To facilitate the identification of whether each process is applied, they are marked using the following symbols: ▲ and △ respectively indicate whether or not the cluster augmentation process is applied, while ● and ○ respectively indicate whether or not the adaptive k process is applied. The best results obtained on average and by cluster size are marked in bold type (when the result is the same, that which also optimizes the other metric, F_1 or number of computed distances, is marked).

	F_1 (%)				Distances (%)			
c	cKD △○	cKD+ ▲○	caKD △●	caKD+ ▲●	cKD △○	cKD+ ▲○	caKD △●	caKD+ ▲●
10	95.83	96.27	96.12	96.31	0.45	0.49	0.22	0.24
15	95.83	96.29	96.14	**96.31**	0.46	0.47	0.22	0.23
20	95.70	96.14	95.97	96.18	0.43	0.48	0.20	0.22
25	95.65	96.12	95.95	96.16	0.40	0.46	0.18	0.20
30	95.63	96.08	95.90	96.13	0.39	0.45	0.18	0.19
100	95.67	95.99	95.85	96.00	0.30	0.38	0.14	0.16
500	95.76	95.99	95.88	95.99	0.20	0.28	0.13	**0.14**
1000	95.94	96.12	96.06	96.15	0.18	0.26	0.14	0.14
Average	95.75	96.12	95.98	**96.15**	0.35	0.41	0.18	**0.19**

As it can be observed in this table, the cluster augmentation process slightly improves the classification rate in terms of the F_1 metric. As previously indicated, note that the result obtained by the exhaustive kNN search (i.e., calculating all possible pairwise distances) is 96.94 %, being thus the results achieved quite close to the maximum achievable rate and, therefore, the improvement margin is remarkably narrow. This process specifically improves the F_1 score by 0.37 % when the adaptive k process is not applied (that is, the combination cKD △○ → cKD+ ▲○), and 0.17 % when it is applied (for combination caKD △● → caKD+ ▲●).

The adaptive k process also improves the average F_1 score by 0.23 % when the cluster augmentation process is not applied (combination cKD △○ → caKD △●) and 0.03 % when it is applied (combination cKD+ ▲○ → caKD+ ▲●). In the latter case, note that the cluster augmentation has already been applied, which already improves the result by 0.17 %.

When analyzing both processes together, it can be seen that, on average, they improve the result by 0.4 %, and up to 0.5 % in the case of $c = 15$. The best result obtained by caKD+ is only 0.63 % worse than the best possible result obtained with the exhaustive kNN search. However, the proposed method only requires a 0.23 % of the maximum number of possible pairwise distances.

With regard to the number of distances, as expected, the cluster augmentation process produces a slight increase in the number of distances: 0.06 % when the adaptive k process is not applied and 0.01 % when it is applied. Furthermore, the adaptive k process reduces the number of distances, specifically 0.18 % less when clustering augmentation is not applied (combination cKD △○ → caKD △●) and 0.22 % less when it is applied (combination cKD+ ▲○ → caKD+ ▲●). This reduction is due to the fact that, on average, this method requires a lower value of k, as will be discussed in the following section.

In order to rigorously assess the improvements attained when employing these processes, we resorted to the Wilcoxon test signed-rank test [74]. More precisely, the idea was to assess whether the improvement observed in terms of both the F_1 score and the number of distances computed after applying these methods was statistically significant in comparison to the results obtained when not considering them. Table 4 shows the results of this comparison, considering all possible combinations of applying and not applying each process. The symbol ✓ indicates that the method specified in the row is significantly better than that indicated in the column for all the datasets, folds, and cluster sizes evaluated, considering $p < 0.05$ as statistical significance threshold.

Focusing on the classification rate results (first four columns of the table), it can be observed that the application of the cluster augmentation and adaptive k processes, both separately and combined, report significant improvements in the results with respect to not being applied. It can also be seen that the use of cluster augmentation (marked as ▲) usually entails a superior improvement than that of the adaptive k process (highlighted with ●), and that the combination of both processes yield to better results than when they are applied separately.

Regarding the efficiency analysis (last four columns of the table), note that the cases in which the adaptive k stage is considered (last two rows of the table) consistently outperform those in which the process is elided. In contrast, the test also shows that the cluster augmentation process

Table 4: Results of the Wilcoxon signed-rank test comparing the improvement obtained for both classification rate and number of distances computed when applying the cluster augmentation and the adaptive k processes. The symbol ✓ indicates that the method specified in the row is significantly better (when considering a statistical significance threshold of $p < 0.05$) than that indicated in the column for all the corpora, folds, and cluster sizes evaluated.

		Accuracy (F_1)				Efficiency (Distances)			
		cKD	cKD+	caKD	caKD+	cKD	cKD+	caKD	caKD+
		△○	▲○	△●	▲●	△○	▲○	△●	▲●
cKD	△○	–				–	✓		
cKD+	▲○	✓	–	✓			–		
caKD	△●	✓		–		✓	✓	–	✓
caKD+	▲●	✓	✓	✓	–	✓	✓		–

(denoted as ▲) does not report any statistical improvement in terms of the reduction in the number of distances computed with respect to not considering it. It must be remarked that such conclusions constitute a rather expected result given that the cluster augmentation process entails an increase in the reference set size. In this sense, while one might argue the usefulness of this augmentation process in the overall pipeline, note that its relevance is justified due to its reported significant improvement in the classification rate of the scheme.

Finally, the reader may have noticed that classification performance and efficiency generally constitute two conflicting goals in which the improvement in one of them implies the deterioration of the other. Since one may prioritize one of these goals depending on the actual context of use, it is thus not possible to depict a single optimal configuration. Related literature addresses this issue considering a Multi-Objective Problem framework, which basically retrieves a set of possible solutions to the task without any particular preference among them [8, 28, 75]. Section 5.5.1 of this work provides the analysis of the caKD+ proposal considering that particular framework for further providing additional insights about the method.

5.3. Adaptive k process evaluation

One interesting parameter to analyze is the value of k calculated using the adaptive k method. When calculating k without using the adaptive process, an average of $k = 4.9$ is obtained for all the datasets considered. This value is used by cKD and cKD+ for all cluster sizes since these methods use a general k, regardless of the cluster sizes.

When applying the adaptive k process, an average of $k = 1.5$ is obtained for all the cluster sizes and datasets considered. Specifically, the following values of $k = \{1.9, 1.7, 1.6, 1.6, 1.5, 1.3, 1.1, 1.1\}$ are obtained for cluster sizes $c = \{10, 15, 20, 25, 30, 100, 1000\}$, respectively. Hence, the adaptive k process obtains a lower value of k, on average.

Also note that this k value decreases as the number of clusters c increases. This is because many clusters are well grouped and consequently employ $k = 1$, and although there are some clusters with a higher value of k, the overall average drops considerably. This reduction in the value of k also leads to a reduction in the number of computed distances, as seen in the previous section.

5.4. Cluster size analysis

As the results presented in the previous sections show, the cluster size c affects both the F_1 and the number of pairwise distances to compute. Gallego et al. [25] do not indicate how to select this size for ckNN+, but rather evaluate different cluster sizes by applying a *grid-search* technique [76] to select the value of c that optimizes the criterion of interest (efficiency or efficacy). However, the proposed caKD+ method does specify how to automatically select this value based on the criteria to be optimized (see Section 3.2.1).

Table 5 shows a comparison of the result obtained with caKD+ when using the proposed automatic selection method versus the best result previously obtained when evaluating cluster sizes from 10 to 1000 (see Table 3).

Table 5: Comparison of the results obtained with caKD+ when applying the method proposed to automatically calculate the number of clusters based on the metric to be optimized (F_1 or distances) against the ones obtained from a grid-search process. The results from the latter process may be checked in Table 3. The best results for F_1 and distances are marked in bold type (when the result is the same, that which also optimizes the other metric, F_1 or distance, is marked).

	Cluster size c	F_1 (%)	Distances (%)
Optimize F_1	Grid search (c=15)	96.31	0.23
	$c = Elbow_c$	**96.31**	0.20
Optimize distances	Grid search (c=500)	95.99	0.14
	$c = \sqrt{m}$	95.96	**0.13**

As it can be seen, the proposed optimization approaches yield to better results for both metrics: on the one hand, in the case of the efficiency optimization, the proposal improves a 0.1 % with respect to the grid-search process; on the other hand, in the case of the F_1 optimization, while the results obtained by both the Elbow and the *grid-search* process match, note that our proposal calculates a smaller number of distances. Finally, while it may be argued that the difference in the results may be negligible, it is important to remark that the optimization strategy proposed avoids the exploration of the entire solution space that a grid-search process requires, thus considerably reducing the burden of the training stage.

5.5. Comparison with ckNN+

Having thoroughly analyzed the proposed caKD+ on its own, we shall now compare it with the ckNN+ method by Gallego et al. [25] as it constitutes the starting point of the presented method. Table 6 shows the result of this comparison in terms of F_1 and distances for values of c in the range $[10, 1000]$. In addition, in order to compare these results with those that would be obtained by using an exhaustive and non-approximate search technique (such as kNN), the value of $c = 1$ —i.e., classifying with a single cluster— is also included. The value of k was selected for both methods using a validation partition with 10 % of the training samples and analyzing values in the range $k \in [1, 15]$. Also note that the average (last row of the table) was calculated discarding the value of $c = 1$ to avoid skewing the data (mainly the number of distances).

Table 6: Comparison of caKD+ and ckNN+ methods in terms of F_1 and total number of distances for different cluster sizes. Each result represents the average obtained when employing the cross validation process with all the datasets considered. The best results obtained on average and by cluster size are marked in bold type (when the result is the same, that which also optimizes the other metric, F_1 or distance, is marked).

	F_1 (%)		Distances (%)	
c	ckNN+	caKD+	ckNN+	caKD+
1	96.94	96.94	100.00	11.85
10	96.29	96.31	14.92	0.24
15	96.29	**96.31**	10.80	0.23
20	96.15	96.18	8.75	0.22
25	96.12	96.16	7.10	0.20
30	96.08	96.13	6.02	0.19
100	95.99	96.00	3.14	0.16
500	95.99	95.99	4.98	**0.14**
1000	96.12	96.15	8.99	0.14
Average	96.13	**96.15**	8.09	**0.19**

An initial remark to begin with is that, when considering the case of $c = 1$, the ckNN+ method is equivalent to applying kNN, reason why it calculates 100 % of the distances. However, in the case of caKD+, $c = 1$ is equivalent to considering a KD Tree structure of the entire training set, which reduces the number of distance computations to a value of 11.85 % with respect to the exhaustive search.

When considering cluster values higher than the unit, it can be checked that the proposed caKD+ method consistently improves ckNN+ in both F_1 and distance computations. While the classification rate in terms of F_1 improves by an average of only 0.02 % and 0.06 % in the case of $c = 30$, it must be considered that this result is very close to the best possible theoretical result given by the exhaustive kNN search ($F_1 = 96.94$ %), being thus the improvement margin quite narrow. Nonetheless, a much more remarkable improvement is obtained in the number of computed distances, since caKD+ calculates only 0.19 % of distances (7.90 % less than ckNN+), yet still improving the classification rate.

In order to statistically assess the improvement depicted by both the classification rate and reduction in the number of distanced computed, we considered the non-parametric Wilcoxon signed-rank test as in previous experiments in this manuscript. More precisely, the idea was to assess whether the results obtained by the caKD+ method consistently improved those by the ckNN+ approach for all corpora, folds, and cluster sizes. Note that we consider a significance threshold of $p < 0.05$ for this analysis.

When assessing the classification rate figures with the proposed strategy, a p-value of 0.0377 was obtained. On the other hand, when considering the reduction in the number of distances, a $p < 10^{-6}$ figure was retrieved. Hence, attending to the aforementioned significance threshold, this test reflects that the caKD+ proposal significantly outperforms the ckNN+ method in terms of both efficacy and efficiency.

5.5.1. Multi-objective optimization problem

The metrics considered for the assessment of this problem (F_1 and the number of distances) are, quite often, opposite, since an improvement on one of them usually entails a deterioration in the other one. From this point of view, this task can be seen as a Multi-objective Optimization Problem (MOP) in which two functions are meant to be optimized simultaneously.

The most common means of tackling such problems is resorting to the concept of non-dominance: one solution is said to dominate another if, and only if, it is better or equal in each objective function and, at least, strictly better in one of them. The best solutions (there may be more than one) are, therefore, those that are non-dominated. In the MOP framework, the strategies within this set define the so-called Pareto frontier and can be considered the best without any particular order among them [77].

Assuming this MOP scenario, Figure 3 shows the results obtained using the caKD+ and ckNN+ methods in which each point is a 2-dimensional value defined by its F_1 and calculated distances obtained from averaging all folds and corpora for the corresponding algorithm configuration. Note that in this case we are not considering any of the alternative caKD+ methods which result when certain steps of the algorithm are deactivated.

As it can be checked, the best results for both criteria are obtained by the proposed caKD+ method as they consistently achieve either a higher classification performance or a lower number of computed distances than the ckNN+ approach. Regarding the actual MOP analysis, note that the Pareto frontier is defined exclusively by different configurations of the caKD+ proposal. Obviating the case of $c = 1$ which is equivalent to a KD Tree applied to the entire training set, the

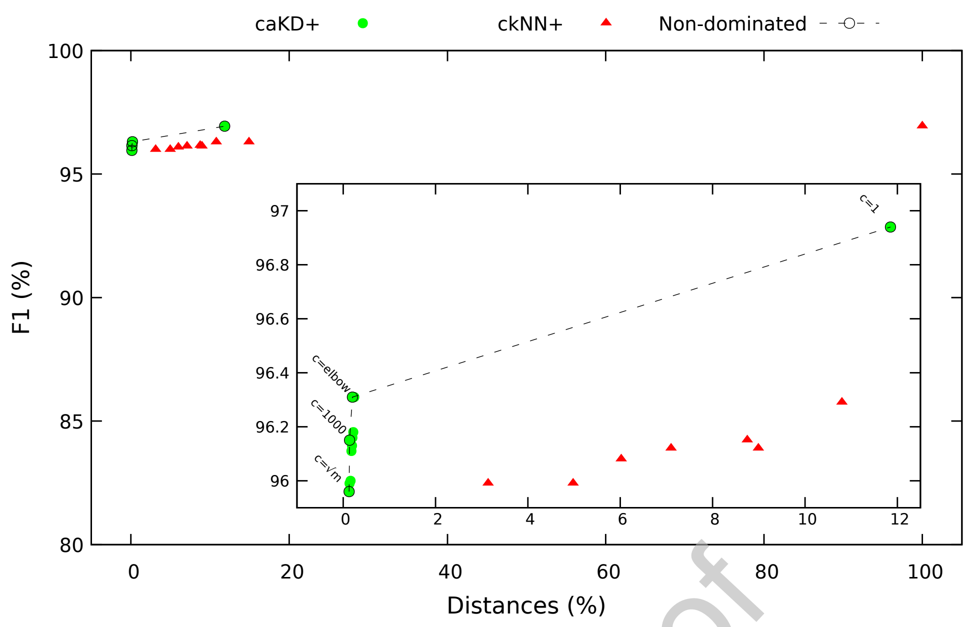

Figure 3: Analysis of F_1 and efficiency as a Multi-objective Optimization Problem (MOP). Non-dominated elements are highlighted.

non-dominated configurations that define the frontier are: $c = \sqrt{m}$ with the minimum number of calculated distances, $c = Elbow$ with the best F_1, and $c = 1000$ with an intermediate result for both metrics. It is worth remarking that two of the non-dominated solutions are obtained when considering the proposed methods to automate the calculation of the number of clusters in terms of either the classification performance or the optimization of the number of distances.

5.6. *Comparison with state-of-the-art efficient kNN search algorithms*

This section presents a benchmark comparison of the proposed caKD+ method against 16 different state-of-the-art efficient kNN search approaches. Since many works deal with this task, we have selected a representative collection of each of the FSS, DR, and ASS families (detailed in Section 2) for the comparison. Table 7 summarizes the different strategies compared, including the parameters used in each case.

Figure 4 shows the results of this comparison when considering the same evaluation methodology in terms of cross-validation, metrics, and datasets as in the previous sections. The result obtained by the conventional exhaustive kNN search is also included to provide a reference for the improvement obtained after applying these methods. The results are shown in different colors and shapes depending on the type of approach used. Furthermore, for each efficient-search family, the Pareto frontier is marked separately, along with its non-dominated results.

As it can be seen, the proposed caKD+ method achieves the best F_1 results for the different parameter configurations evaluated (i.e., different number of clusters c), for which it maintains quite

Table 7: Summary of the different efficient kNN search strategies compared. The identifier, algorithmic family, and set of parameters evaluated for each method are indicated.

Algorithm	**Identifier**	**Family**	**Parameters evaluated**
k-Nearest Neighbor	kNN	-	$\mathrm{k} \in \{1, 3, 5, 7, 9\}$
KDTree [14]	KDTree	FSS	$\mathrm{leaf} \in \{10, 20, 30, 40\}$
BallTree [32]	BallTree	FSS	$\mathrm{leaf} \in \{10, 20, 30, 40\}$
Local Sensitive Hashing [47]	LSH	ASS	$\mathrm{trees} \in \{10, 20, 30, 40\}$
Spectral Hashing [46]	SH	ASS	$\mathrm{nbits} \in \{40, 80, 100, 120\}$
Product Quantization [45]	PQ	ASS	$\mathrm{nsubq} \in \{1, 2, 4, 8\}$
Clustering-based k-Nearest Neighbor [25]	ckNN+	ASS	$\mathrm{c} \in \{10, 15, 20, 25, 30, 100, 500, 1000\}$
Reduction through Homogeneous Clusters [41]	RHC	DR	–
Condensing Nearest Neighbor [8, 37]	CoNN	DR	$\mathrm{k} \in \{1, 3, 5, 7, 9\}$, $\mathrm{c} \in \{1, 2, 3\}$
Editing Condensing Nearest Neighbor [8, 78]	ECNN	DR	$\mathrm{k} \in \{1, 3, 5, 7, 9\}$, $\mathrm{c} \in \{1, 2, 3\}$
Fast Condensing Nearest Neighbor [8, 79]	FCNN	DR	$\mathrm{k} \in \{1, 3, 5, 7, 9\}$, $\mathrm{c} \in \{1, 2, 3\}$
Farther Neighbor [8, 80]	FN	DR	$\mathrm{k} \in \{1, 3, 5, 7, 9\}$, $\mathrm{c} \in \{1, 2, 3\}$, $\mathrm{r} = 0.1$
Nearest to Enemy [8, 80]	EN	DR	$\mathrm{k} \in \{1, 3, 5, 7, 9\}$, $\mathrm{c} \in \{1, 2, 3\}$, $\mathrm{r} = 0.1$
Instance Rank based on Borders [8, 81]	IRB	DR	$\mathrm{k} \in \{1, 3, 5, 7, 9\}$, $\mathrm{c} \in \{1, 2, 3\}$, $\mathrm{r} = 0.1$
Decremental Reduction Optimization Procedure 3 [8, 82]	DROP3	DR	$\mathrm{k} \in \{1, 3, 5, 7, 9\}$, $\mathrm{c} \in \{1, 2, 3\}$
Iterative Case Filtering [8, 83]	ICF	DR	$\mathrm{k} \in \{1, 3, 5, 7, 9\}$, $\mathrm{c} \in \{1, 2, 3\}$

stable results. As expected from the analysis in previous sections, the next best-performing method concerning F_1 is the ckNN+ one. After that, in general terms, the FSS family of algorithms prove to be the more competitive algorithms with a quite steady performance. Finally, methods belonging to the ASS and DR families show a great variability in terms of classification rate depending on the type of approach and parameterization considered.

Regarding to the number of computed distances, note that the proposed caKD+ method and some of the methods in the DR family (specifically IRB [8]) obtain similar reduction results. However, the DR family methods are not competitive in terms of F_1 score, since the reduced number of calculated distances is achieved at the cost of obtaining a very poor classification precision (62 % worse than the proposed method). After caKD+ and DR, the next method that computes fewer number of distances is ckNN+, followed by the ASS family of methods and finally FSS. In general, it is observed that caKD+ obtains the best results for both metrics and remains much more stable to changes in its parameters.

5.6.1. Comparison considering Neural Codes representation

In order to further study the capabilities of the proposed caKD+ method, we now present an additional study which assesses the influence of the NC representation on the overall success of the task. More precisely, this experiment extends the previous comparison against the different state-of-

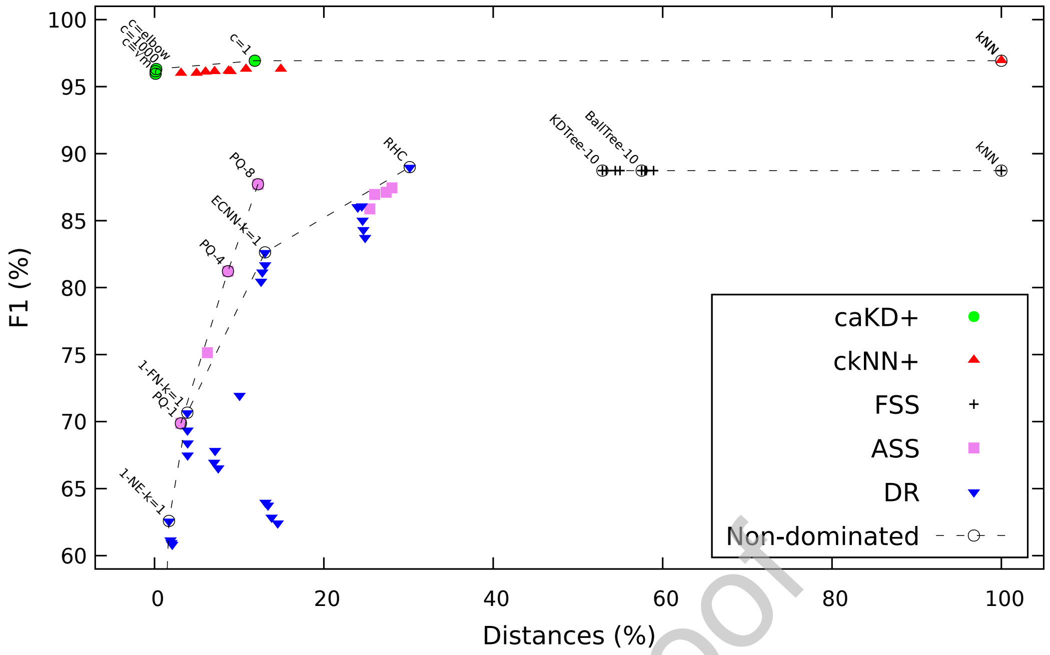

Figure 4: Comparison of the proposed caKD+ method with the considered efficient-search strategies for kNN in terms of F_1 and number of distances computed. Non-dominated elements are highlighted in each case.

the-art efficient search methods by considering the NC representation as input of those algorithms.

It must be noted that the feature learning process that retrieves the NC representation is one of the different processes in the pipeline of the proposed caKD+ method while in the rest of the efficient-search algorithms considered this process is additionally included to perform the aforementioned assessment. The analysis of the results obtained shall provide additional insights about the benefits of considering such compact representation in other schemes.

Having introduced the aim of the experiment, Figure 5 graphically shows the results obtained with the different efficient-search algorithms when considering the NC representation as input to the methods. These results may be directly comparable with the ones in Figure 4 as the only difference is the general use of the NC representation for the data.

As it can be observed, the performance of all schemes is generally improved, both in terms of efficiency and efficacy. More precisely, the minimum classification score is now located close to $F_1 = 80\%$ for a DR strategy while this score was around $F_1 = 60\%$ when not considering NC representations. In a similar sense, the maximum number of distances computed is obtained by a FSS scheme with a value around a 40% of the maximum while in the raw data representation this value was close to a 60% of the maximum number of distances to compute. These results support the idea that a NC representation obtained with the proper feature learning process does produce a more compact data representation which eventually benefits the subsequent stages of the efficient-search scheme.

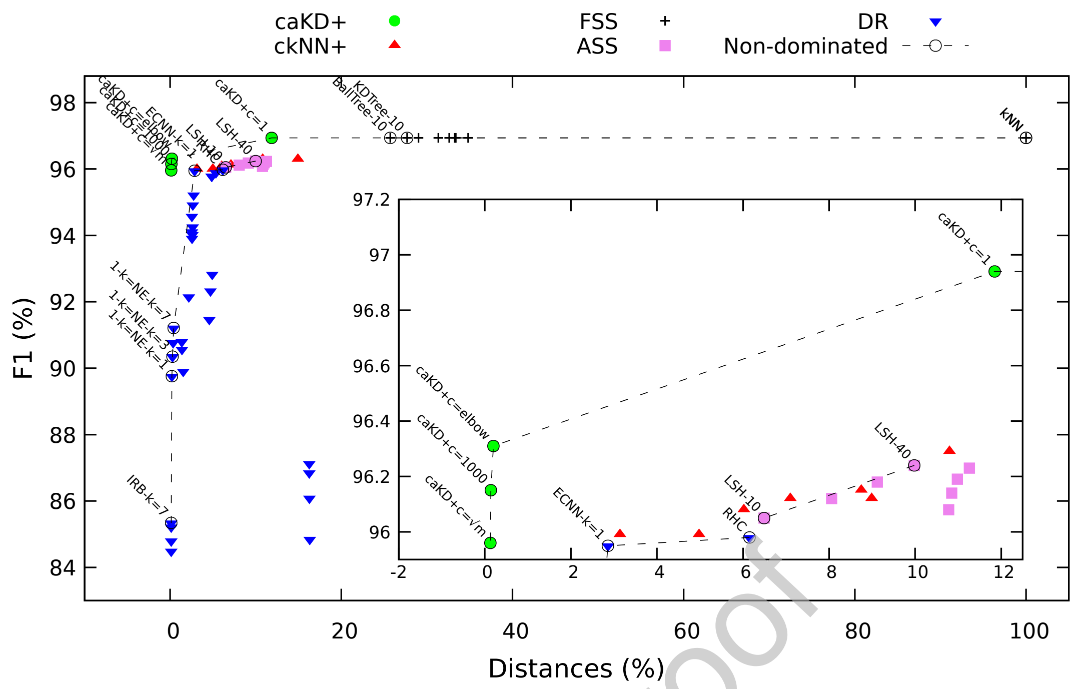

Figure 5: Comparison of the proposed caKD+ method with the different efficient-search strategies for kNN in terms of F_1 and number of distances computed when considering the Neural Codes representation as input. Non-dominated elements are highlighted in each case.

655 However, while all the considered schemes do report a benefit in their performance with this NC-based representation, still the results obtained by the different configurations of the proposed caKD+ method stand out as the ones achieving the best compromise between efficiency and efficacy since none of the alternative methods dominate them in the graph.

6. Discussion

660 Having presented the experiments for both performing an in-depth analysis of the caKD+ proposal and its comparative assessment with various state-of-the-art methods for efficient search, this section provides an additional discussion about the degree of contribution of the different stages of caKD+ to the overall success of the task.

As presented, the proposed caKD+ method aims at improving the efficiency issue inherent to the 665 kNN search algorithm. For that, our proposal comprises a set of stages which can be summarized in the use of a feature learning process, a clustering stage, the computation of a set of KD-Tree structures for fast prototype indexing, and the optimization of the k value for the eventual search.

The use of the feature learning allows the scheme to both reduce the dimensionality of the data and retrieve a suitable representation for the task at issue. This stage is crucial for the success of the 670 scheme, at least for most high-dimensional corpora, since the subsequent stages of the proposal, and particularly the KD-Tree indexing structures, severely degenerate its performance as the number of dimensions grows. Additionally, the representation obtained is usually more compact than the raw one from the original data, which improves the clustering stage since, in this new space, instances belonging to the same class are expected to be located close to each other and far from the dissimilar ones. This is due to, as stated in Section 3.1, the DNN schemes responsible for this space mapping are meant to produce an alternative representation in which the different categories involved in the task are expected to be linearly separable. This fact has been assessed in Section 5.6.1 in which different efficient-search algorithm, along with caKD+, have been fed with the learned representation from the encoder used in the work and all of them improved both their efficiency and efficacy figures with respect to directly considering the raw data. As a last point to comment in this regard, Gallego et al. [25] quantitatively studied the quality of data clusters in both a DNN-based learned representation with respect to that of the original space using a set of metrics from the clustering-related literature, namely *Silhouette Coefficient*, *Calinski–Harabaz Index*, *Homogeneity Score*, and *Completeness Score*. This study concluded that the feature learning was significantly beneficial for the clustering process, being thus of remarkable applicability its use in efficient-based kNN search proposals.

Regarding the clustering process, this stage groups the existing instances into c different sets according to their proximity in the dimensional space obtained in the feature learning stage. When queried, this grouping allows a faster while approximate search than that of the exhaustive case. Nevertheless, as stated in Section 3.2.1, not all c values report a benefit in the overall performance, but some particular values optimize the process in terms of distance computations. This effect has been experimentally assessed in Section 5.4 in which the postulated theoretical optimal points match the ones obtained with an exhaustive grid-search optimization. Note that in the case of the classification performance optimization, while still computationally complex, the considered Elbow method reduces the computation time with respect to the commented exhaustive search. This assertion is based on the fact that, while the former strategy stops the process once the optimal point is reached [57], the latter case requires examining the entire solution space.

Additionally, in order to palliate the commented efficacy decrease due to the use of the cluster-based search strategy, caKD+ introduces a cluster augmentation process which allows some overlapping among them with the aim of compensating such loss. The effect of such stage has been quantitatively analyzed in Section 5.2, concluding that such overlap does always report an improvement in the efficacy of the scheme at the expense of marginally increasing the number of computed distances.

The use of KD-Tree structures for both indexing the different clusters and the instances inside clusters also implies a remarkable improvement in the efficiency of the scheme against the exhaustive search. As observed in Section 5.5, even if considering a cluster size $c = 1$ (i.e., not applying any clustering), the number of computed distances is reduced to a 11.85% of the total. While it might be argued that such efficiency figures may be sufficient and thus it would not be necessary to introduce the aforementioned clustering process, the experimentation presented proves that combining such clustering with a stack of KD Tree structures instead of a single general one further improves the efficiency figures to values several orders of magnitude lower than the exhaustive case.

Finally, the use of adaptive k values inside each cluster reduces the number of comparisons to be performed for a given new query and improves the general efficiency of the scheme, as quantitatively proved in Section 5.3. Besides, this process also matches the local approximation principle of the kNN-based search which, as stated in the Introduction, is one of the benefits of such instance-based algorithm.

6.1. Comparison of runtime cost

We now include an additional benchmark which compares the search cost, measured in execution time, of five of the considered situations in the Experimentation section: the exhaustive kNN search, the FSS-based methods BallTree [14] and KD-Tree [32], the ckNN+ method [25], and the proposed caKD+ strategy. These calculations have been done for all corpora considered in the present work as well as for an supplementary one, the ImageNet collection [84], which we considered due to its large number of elements and hence assess the performance of those methods in a much extensive database than the ones considered. This latter corpus is a generic-purpose dataset for object classification used in the Large Scale Visual Recognition Challenge (ILSVRC) which comprises a total of $1,331,167$ color images of 224×224 pixels of $1,000$ different classes. Note that in this particular image collection the NC representation was extracted resorting to the MobileNet v2 architecture [85] initialized with the pre-trained weights from the ILSVRC dataset and performing an additional fine-tuning process for adapting its output to our NC layer size.

Regarding the actual configuration of the search methods, we considered the parameters which optimized the performance of these schemes in the previous experimentation. More precisely, for the ckNN+ and caKD+ cases we fixed the number of clusters c to the value that minimizes the number of distances computed by the cluster-based scheme, i.e. $c = \sqrt{m}$. In the case of the BallTree and KD-Tree methods a leaf value of 10 was used as it represents the particular configuration which achieves the non-dominated solutions for both strategies.

Table 8 shows, for each of the corpora considered, the average time per query obtained as the mean of the time consumed by all the queries to the system, considering the same train/test partitions and 5-fold cross validation divisions described in Section 4. The machines and programming languages are the same as the ones used in the rest of the work, which are described at the beginning of Section 5.

As it can be observed, the efficiency of the exhaustive kNN case degrades as the size of the corpus increases. This is a rather expected behavior since such algorithm requires the examination of the

Table 8: Comparative of the average search time, in milliseconds, obtained by the exhaustive kNN, and compared to that of the KDTree, BallTree, ckNN+, and caKD+ methods for the different corpora considered.

		ms. / query				
Corpus	Training set size	kNN	BallTree	KD-Tree	ckNN+	caKD+
Landsat	5,151	43.23	23.73	20.89	5.06	2.80
Gisette	5,600	45.29	27.14	24.14	8.03	1.78
USPS	7,444	60.21	35.03	32.25	4.87	3.05
Pendigits	8,797	71.89	40.97	39.04	3.96	2.10
HOMUS	12,160	98.02	57.12	49.70	5.00	3.19
Letter	16,012	130.70	79.82	68.74	5.52	3.17
NIST	35,972	311.02	172.11	159.11	4.17	2.56
GTSRB	41,471	344.96	197.86	181.16	5.49	1.00
Math	44,800	376.46	214.27	201.15	5.78	0.85
MNIST	56,004	492.26	285.80	243.41	5.52	0.80
ImageNet	1,281,167	11,368.47	6,213.12	5,131.92	119.27	14.27

entire corpus to perform the search task, being thus its performance completely related to the size of the collection.

Regarding the BallTree and KD-Tree schemes, the efficiency trend they depict with respect to the size of the training data is similar to that of the exhaustive kNN method. As it can be observed, the average search time progressively increases as the corpus set size grows. However, note that both FSS-based methods depict much more efficient figures than those of the kNN strategy, performing the same search task in, approximately, half of the computation time.

In the cases of both ckNN+ and caKD+ the degeneration in terms of query execution time is not that noticeable as in the previous cases. This is a reasonable behavior since both strategies derive a series of structures and comprise different processes to optimize the search process. In the sole case of ImageNet the search time severely increases up to an order of magnitude higher than the rest of the considered corpora, being still the search times retrieved remarkably lower than the exhaustive search case.

Finally, note that the proposed caKD+ method consistently achieves the lowest search time of the considered cases. This fact, together with the conclusions gathered from the experimentation and analysis in the rest of the work, proves the validity and usefulness of caKD+ as a new and competitive strategy for performing efficient search tasks in the context of the kNN rule.

6.2. Computational cost related to set size

In relation to the previous study about execution time, we provide an additional experiment to further analyze the caKD+ proposal. More precisely, we aim at assessing the relation of both the computational cost and search performance of the caKD+ method with the train set size of the corpus. For that, we consider the ImageNet collection and evaluate the two commented figures of merit as the amount of training data is progressively increased up to the point of considering the complete canonical train partition, i.e., the 1,281,167 prototypes. Note that we selected this particular corpus as it contains the largest amount of data among the corpora used in the work.

Figure 6 provides the results of this study, being the computational cost deemed as the invested time per input query and the search performance as the classification rate in terms of the F_1 metric.

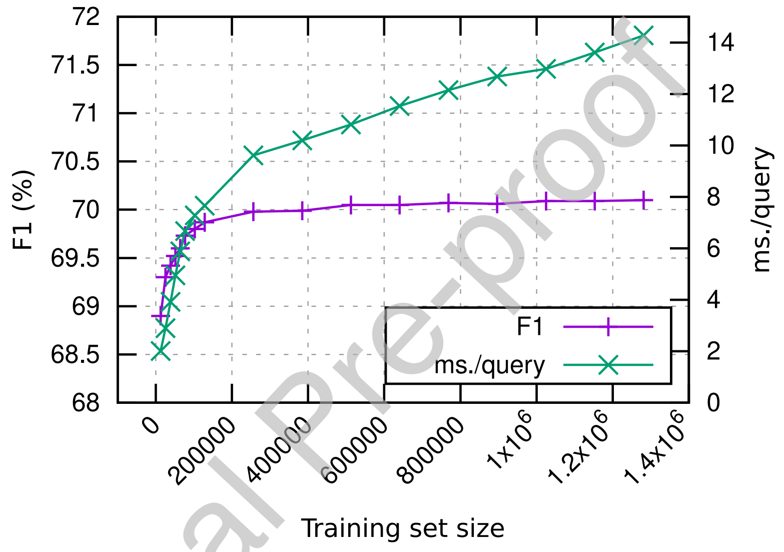

Figure 6: Analysis of the computational cost and search performance (measured as search time per input query and classification rate in terms of the F_1 metric, respectively) related to the size of the training partition of the ImageNet collection. Note that the different sizes evaluated have been simulated by reducing the canonical train partition of the corpus.

Attending to the results obtained, it can be observed that the classification accuracy achieved by the scheme does not remarkably differ as the size of the training partition increases. More precisely, this rate improves from an initial figure of, roughly, $F_1 = 69\%$ to a value of $F_1 = 70.1\%$, which supposes an absolute improvement of 1.1%. This is no strange since, as commented in Section 3.1, the feature learning stage maximizes the separation among the different classes of the corpus, independently of the amount of data. Thus, the use of larger amounts of data results in more populated clusters, without a remarkable performance improvement.

Regarding the computation cost, the experimental results prove that training set size and search time generally depict a lineal relation, thus validating the proposed pipeline of the caKD+ method.

This analysis also reports that, when considering small set sizes, this cost does not follow such trend, most likely due to the cost of the different data structures considered. In any case, while this training set size and search time may not be linearly related in every single case, the different analyses and experiments in this work have proved the caKD+ method as a significantly efficient, yet accurate, alternative for kNN search.

7. Conclusions and future work

The k-Nearest Neighbor (kNN) rule stands as one of the main supervised learning algorithms for both its simplicity and reported good results in terms of search and classification tasks. Nevertheless, this algorithm exhibits an inherent issue in terms of scalability since it does not derive a model out of the training data but it relies on exhaustively consulting it every time a new query is produced. In this regard, due to its relevance in the Pattern Recognition field, a large number of approaches have been historically proposed to palliate the commented efficient issue of the kNN search algorithm.

This work presents a new method for efficient kNN search based on clustering and feature space transformation based on the previously proposed ckNN+ algorithm [25]. This new proposal introduces a set of complementary processes that reduce the search time and improve classification results, such as using adaptive k values per cluster or pre-calculated KD Tree structures. A further contribution of this paper is a proposal that can be employed to automate the calculation of the cluster size based on the metric to be optimized — the efficacy or the efficiency. This proposal obtains optimal results for the selected metric without having to carry out a grid-search process, thus considerably speeding up the training time.

The proposed method, called caKD+, has been validated using 10 datasets with different number of samples, classes and features, including images, and feature vectors. For the feature space transformation, 6 different topologies with a varied number of layers, types of layers, and parameters has been assessed to optimize this step.

The results show that caKD+ considerably reduces the number of distances calculated compared to the previously proposed ckNN+ approach while also improving the classification result. These figures were validated by means of statistical tests in order to both demonstrate the significance of the improvement and validate the contribution of applying the different stages of the caKD+ method. Furthermore, a lineal relation between the size of the corpora and the computational cost was experimentally assessed, proving its validity for scenarios differing in the size of the corpus. Additionally, the proposed scheme was compared to 16 state-of-the-art strategies for efficient kNN search, attaining the best overall results in the compromise between efficiency and efficacy while also proving to be a much more stable method as regards changes in its configuration parameters. Only a technique based on data reduction (IRB [8]) calculates a smaller number of distances, but at the cost of considerably reducing the efficacy of the scheme (62 % worse), while caKD+ calculates a similar number of distances but obtains the best classification results of all the methods compared and is only 0.63 % worse than the best theoretically possible result, i.e., the one obtained with the exhaustive kNN rule.

As future work, we aim at studying more advanced network architectures that improve the clustering process, such as employing specialized loss functions. Another avenue of this proposal that could be explored in greater depth is that of evaluating the system with time-evolving corpora as, for example, varying the number of samples or classes. As a last promising line of work, we consider the study of more sophisticated clustering policies, such as a hierarchical methods, as they could not only improve the search performance of the scheme but also avoid some of the currently necessary tree-based indexing structures.

Acknowledgment

This work was supported by the Spanish Ministry HISPAMUS project TIN2017-86576-R, partially funded by the EU, as well as by the “Programa I+D+i de la Generalitat Valenciana” through grant APOSTD/2020/256.

Appendix A. Nomenclature and mathematical definitions

For the sake of conciseness, this appendix gathers and lists the mathematical symbols and definitions used in the manuscript:

- $\mathcal{X}$: Initial D-dimensional feature space of the data.
- x_m: Single element from space $\mathcal{X}$, i.e., $x_m \in \mathcal{X}$.
- $\mathcal{Y}$: Set of possible target labels., i.e., $\mathcal{Y} = \{Y_1, \ldots, Y_L\}$.
- y_m: Single label from space $\mathcal{Y}$, i.e., $y_m \in \mathcal{Y}$.
- $\mathcal{T}$: Corpus of labeled data comprising a set of duples $\{(x_m \in \mathcal{X}, y_m \in \mathcal{Y})\}_{m=1}^{|\mathcal{T}|}$.
- Enc: Deep Neural Network trained for performing the mapping $Enc : \mathcal{X} \rightarrow \mathcal{Y}$.
- Enc^F: Subnetwork extracted from Enc used for mapping data in the initial space $\mathcal{X}$ to $\mathcal{X}^F$, i.e., $Enc^F : \mathcal{X} \rightarrow \mathcal{X}^F$.
- $\mathcal{L}$: Loss function considered for training the Enc neural scheme.
- $\mathcal{X}^F$: Target N-dimensional space which D-dimensional space $\mathcal{X}$ is projected to.
- x_m^F: Single element from space $\mathcal{X}^F$, i.e., $x_m^F \in \mathcal{X}^F$.
- $\mathcal{T}^F$: Equivalent to labeled corpus $\mathcal{T}$ in the projected space by Enc^F. Mathematically, this is defined as $\mathcal{T}^F = \{(x_m^F \in \mathcal{X}^F, y_m \in \mathcal{Y})\}_{m=1}^{|\mathcal{T}^F|} = \{(Enc^F(x_m \in \mathcal{X}), y_m \in \mathcal{Y})\}_{m=1}^{|\mathcal{T}|}$.

- c: Number of clusters considered for the method.
- $\mathcal{C}$: Initial set of c data groups $\{C_1, \ldots, C_c\}$ obtained by applying the clustering method to $\mathcal{T}^F$.
- $\mathcal{C}^a$: Set of overlapping data clusters $\{C_1^a, \ldots, C_c^a\}$ obtained by applying a cluster augmentation process to initial set $\mathcal{C}$.
- $\mathcal{C}^c$: Set of c centroids $\{C_1^c, \ldots, C_c^c\}$ of the different clusters in $\mathcal{C}$.
- $\mathcal{K}^c$: Collection $\{k_1, \ldots, k_c\}$ of adaptive k values for each of the clusters in set $\mathcal{C}^a$.
- KDt^c: K-Dimensional Tree structure obtained with the $\mathcal{C}^c$ set of centroids for the efficient cluster selection.
- KDt^a: Set of K-Dimensional Trees for each of the augmented clusters in $\mathcal{C}^a$ for the efficient search within the cluster.
- $\mathcal{N}_q$: Subset of elements $\mathcal{N}_q \subseteq \mathcal{T}$ retrieved by the search method in response to query q.

References

References

[1] R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification, 2nd Edition, John Wiley & Sons, New York, NY, 2001.

[2] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE transactions on information theory 13 (1) (1967) 21–27.

[3] B. Saçlı, C. Aydınalp, G. Cansız, S. Joof, T. Yilmaz, M. Çayören, B. Önal, I. Akduman, Microwave dielectric property based classification of renal calculi: Application of a knn algorithm, Computers in biology and medicine 112 (2019) 103366.

[4] X.-l. Chen, P.-h. Wang, Y.-s. Hao, M. Zhao, Evidential knn-based condition monitoring and early warning method with applications in power plant, Neurocomputing 315 (2018) 18–32.

[5] S. Bairagi, S. W. Ali, Poly (vinylidine fluoride)(pvdf)/potassium sodium niobate (knn) nanorods based flexible nanocomposite film: Influence of knn concentration in the performance of nanogenerator, Organic Electronics (2019) 105547.

[6] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[7] T. M. Mitchell, Machine Learning, McGraw-Hill, Inc., 1997.

[8] J. Calvo-Zaragoza, J. J. Valero-Mas, J. R. Rico-Juan, Improving kNN multi-label classification in prototype selection scenarios using class proposals, Pattern Recognition 48 (5) (2015) 1608–1622.

[9] X. Wu, X. Zhu, G.-Q. Wu, W. Ding, Data mining with big data, IEEE Trans. on Knowl. and Data Eng. 26 (1) (2014) 97–107.

[10] S. García, J. Luengo, F. Herrera, Data Preprocessing in Data Mining, Springer, 2015.

[11] J. Micó, L.; Oncina, A constant average time algorithm to allow insertions in the laesa fast nearest neighbour search index, in: Proc. of the 20th International Conference on Pattern Recognition, ICPR 2010, Istanbul, Turkey, 2010, pp. 23–26.

[12] P. N. Yianilos, Data structures and algorithms for nearest neighbor search in general metric spaces, in: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '93, Society for Industrial and Applied Mathematics, USA, 1993, p. 311–321.

[13] I. Wald, V. Havran, On building fast kd-trees for ray tracing, and on doing that in o (n log n), in: 2006 IEEE Symposium on Interactive Ray Tracing, IEEE, 2006, pp. 61–69.

[14] J. H. Friedman, J. L. Bentley, R. A. Finkel, An Algorithm for Finding Best Matches in Logarithmic Expected Time, ACM Transactions on Mathematical Software 3 (3) (1977) 209–226.

[15] C. D. Toth, J. O'Rourke, J. E. Goodman, Handbook of discrete and computational geometry, Chapman and Hall/CRC, 2017.

[16] Y. Bengio, I. Goodfellow, A. Courville, Deep learning, Vol. 1, MIT press, 2017.

[17] Y. Bengio, A. Courville, P. Vincent, Representation learning: A review and new perspectives, IEEE transactions on pattern analysis and machine intelligence 35 (8) (2013) 1798–1828.

[18] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[19] A. Babenko, A. Slesarev, A. Chigorin, V. Lempitsky, Neural codes for image retrieval, in: D. Fleet, T. Pajdla, B. Schiele, T. Tuytelaars (Eds.), Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I, Springer International Publishing, Cham, 2014, pp. 584–599.

[20] A. Gallego, J. Calvo-Zaragoza, J. R. Rico-Juan, Insights into efficient k-nearest neighbor classification with convolutional neural codes, IEEE Access 8 (2020) 99312–99326. doi:10.1109/ACCESS.2020.2997387.

[21] F. Huang, Y. LeCun, Large-scale learning with SVM and convolutional nets for generic object categorization, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2006, Vol. 1, 2006, pp. 284–291.

[22] A. S. Razavian, H. Azizpour, J. Sullivan, S. Carlsson, CNN Features Off-the-Shelf: An Astounding Baseline for Recognition, in: Proceedings of the 2014 IEEE Conference on Computer Vision

and Pattern Recognition Workshops, CVPRW '14, IEEE Computer Society, Washington, DC, USA, 2014, pp. 512–519.

[23] W. Ren, Y. Yu, J. Zhang, K. Huang, Learning convolutional nonlinear features for k nearest neighbor image classification, in: Pattern Recognition (ICPR), 2014 22nd International Conference on, IEEE, 2014, pp. 4358–4363.

[24] A.-J. Gallego, A. Pertusa, J. Calvo-Zaragoza, Improving convolutional neural networks' accuracy in noisy environments using k-nearest neighbors, Applied Sciences 8 (11). `doi:10.3390/app8112086`.
URL `https://www.mdpi.com/2076-3417/8/11/2086`

[25] A. J. Gallego, J. Calvo-Zaragoza, J. J. Valero-Mas, J. R. Rico-Juan, Clustering-based k-nearest neighbor classification for large-scale data with neural codes representation, Pattern Recognition 74 (1) (2018) 531–543.

[26] S. Zhang, M. Zong, K. Sun, Y. Liu, D. Cheng, Efficient knn algorithm based on graph sparse reconstruction, in: International Conference on Advanced Data Mining and Applications, Springer, 2014, pp. 356–369.

[27] S. Zhang, X. Li, M. Zong, X. Zhu, R. Wang, Efficient knn classification with different numbers of nearest neighbors, IEEE transactions on neural networks and learning systems 29 (5) (2017) 1774–1785.

[28] J. R. Rico-Juan, J. J. Valero-Mas, J. Calvo-Zaragoza, Extensions to rank-based prototype selection in k-nearest neighbour classification, Applied Soft Computing 85 (2019) 105803.

[29] P. Jain, B. Kulis, I. S. Dhillon, K. Grauman, Online metric learning and fast similarity search, in: Proceedings of the 21st International Conference on Neural Information Processing Systems, NIPS'08, Curran Associates Inc., USA, 2008, pp. 761–768.

[30] J. Wang, H. T. Shen, J. Song, J. Ji, Hashing for similarity search: A survey, arXiv preprint arXiv:1408.2927.

[31] E. Vidal, An algorithm for finding nearest neighbours in (approximately) constant average time, Pattern Recognition Letters 4 (3) (1986) 145–157.

[32] T. Liu, A. W. Moore, A. Gray, Efficient exact k-nn and nonparametric classification in high dimensions, in: Proceedings of the 16th International Conference on Neural Information Processing Systems, MIT Press, 2003, pp. 265–272.

[33] P. Ciaccia, M. Patella, P. Zezula, M-tree: An efficient access method for similarity search in metric spaces, in: VLDB '97: Proceedings of the 23rd International Conference on Very Large Data BasesAugust, 1997, pp. 426–435.

[34] A. B. Hassanat, Two-point-based binary search trees for accelerating big data classification using knn, PloS one 13 (11).

[35] L. Rico-Juan, J. R.; Micó, Comparison of aesa and laesa search algorithms using string and tree edit distances, Pattern Recognition Letters 24(9) (2003) 1427–1436.

[36] L. Nanni, A. Lumini, Prototype reduction techniques: A comparison among different approaches, Expert Systems with Applications 38 (9) (2011) 11820–11828.

[37] P. Hart, The condensed nearest neighbor rule (corresp.), IEEE Transactions on Information Theory 14 (3) (1968) 515–516.

[38] J. Derrac, C. Cornelis, S. García, F. Herrera, Enhancing evolutionary instance selection algorithms by means of fuzzy rough set based feature selection, Information Sciences 186 (1) (2012) 73–92.

[39] S. Garcia, J. Derrac, J. Cano, F. Herrera, Prototype selection for nearest neighbor classification: Taxonomy and empirical study, IEEE Transactions on Pattern Analysis and Machine Intelligence 34 (3) (2012) 417–435.

[40] J. Hamidzadeh, R. Monsefi, H. S. Yazdi, Irahc: instance reduction algorithm using hyperrectangle clustering, Pattern Recognition 48 (5) (2015) 1878–1889.

[41] S. Ougiaroglou, G. Evangelidis, RHC: a non-parametric cluster-based data reduction for efficient k-NN classification, Pattern Analysis and Applications 19 (1) (2016) 93–109.

[42] L. Yang, Q. Zhu, J. Huang, D. Cheng, Adaptive edited natural neighbor algorithm, Neurocomputing 230 (2017) 427–433.

[43] N. García-Pedrajas, A. De Haro-García, Boosting instance selection algorithms, Knowledge-Based Systems 67 (2014) 342–360.

[44] C.-F. Tsai, W. Eberle, C.-Y. Chu, Genetic algorithms in feature and instance selection, Knowledge-Based Systems 39 (2013) 240–247.

[45] H. Jegou, M. Douze, C. Schmid, Product quantization for nearest neighbor search, IEEE transactions on pattern analysis and machine intelligence 33 (1) (2011) 117–128.

[46] Y. Weiss, A. Torralba, R. Fergus, Spectral Hashing, in: Advances in Neural Information Processing Systems, 2008, pp. 1753–1760.

[47] M. Bawa, T. Condie, P. Ganesan, Lsh forest: self-tuning indexes for similarity search, in: Proceedings of the 14th international conference on World Wide Web, ACM, 2005, pp. 651–660.

[48] Z. Deng, X. Zhu, D. Cheng, M. Zong, S. Zhang, Efficient knn classification algorithm for big data, Neurocomputing 195 (2016) 143–148.

[49] M. Muja, D. G. Lowe, Scalable nearest neighbor algorithms for high dimensional data, IEEE Transactions on Pattern Analysis and Machine Intelligence 36 (11) (2014) 2227–2240.

[50] S. Theodoridis, K. Koutroumbas, Pattern Recognition, Third Edition, Academic Press, Inc., Orlando, FL, USA, 2006.

[51] L. Rokach, A survey of Clustering Algorithms, Springer US, Boston, MA, 2010, Ch. 14, pp. 269–298. doi:10.1007/978-0-387-09823-4_14.
URL https://doi.org/10.1007/978-0-387-09823-4_14

[52] D. Arthur, S. Vassilvitskii, K-means++: The advantages of careful seeding, in: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007, pp. 1027–1035.

[53] J. L. Bentley, Multidimensional binary search trees in database applications, IEEE Transactions on Software Engineering SE-5 (4) (1979) 333–340.

[54] Y.-m. Cheung, Y. Zhang, Fast and accurate hierarchical clustering based on growing multilayer topology training, IEEE Transactions on Neural Networks and Learning Systems 30 (3) (2019) 876–890. doi:10.1109/TNNLS.2018.2853407.

[55] J. Huang, M. Ng, H. Rong, Z. Li, Automated variable weighting in k-means type clustering, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (5) (2005) 657–668. doi:10.1109/TPAMI.2005.95.

[56] E. Y. Chan, W. K. Ching, M. K. Ng, J. Z. Huang, An optimization algorithm for clustering using weighted dissimilarity measures, Pattern recognition 37 (5) (2004) 943–952.

[57] R. L. Thorndike, Who belongs in the family?, Psychometrika 18 (4) (1953) 267–276. doi:10.1007/BF02289263.

[58] K. Chowdhury, D. Chaudhuri, A. K. Pal, Seed point selection algorithm in clustering of image data, in: Progress in Intelligent Computing Techniques: Theory, Practice, and Applications, Springer, 2018, pp. 119–126.

[59] J. Calvo-Zaragoza, A.-J. Gallego, A. Pertusa, Recognition of handwritten music symbols with convolutional neural codes, in: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), Vol. 1, IEEE, 2017, pp. 691–696.

[60] N. Papernot, P. McDaniel, Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning, arXiv preprint arXiv:1803.04765.

[61] M. Lichman, UCI Machine Learning Repository (2013).
URL http://archive.ics.uci.edu/ml

[62] J. Hull, A database for handwritten text recognition research, IEEE Transactions on Pattern Analysis and Machine Intelligence 16 (5) (1994) 550–554.

[63] J. Calvo-Zaragoza, J. Oncina, Recognition of Pen-Based Music Notation: the HOMUS dataset, in: Proceedings of the 22nd International Conference on Pattern Recognition (ICPR), Stockholm, Sweden, 2014, pp. 3038–3043.

[64] R.-A. Wilkinson, J. Geist, S. Janet, P.-J. Grother, et al., The first census optical character recognition system conference, Tech. rep., US Department of Commerce (1992). doi:10.18434/T4H01C.

[65] J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition, Neural Networks 32 (2012) 323 – 332. doi:https://doi.org/10.1016/j.neunet.2012.02.016.

[66] X. Nano, Handwritten math symbols dataset, https://www.kaggle.com/xainano/handwrittenmathsymbols (1 2017).

[67] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, in: Proc. of the IEEE, Vol. 86, 1998, pp. 2278–2324.

[68] S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, JMLR W&CP 37.

[69] X. Glorot, A. Bordes, Y. Bengio, Deep Sparse Rectifier Neural Networks, Journal of Machine Learning Research (JMLR) W&CP 15 (2011) 315–323.

[70] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.

[71] M. D. Zeiler, Adadelta: an adaptive learning rate method, arXiv preprint arXiv:1212.5701.

[72] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings IJCAI, Vol. 2 of IJCAI'95, Morgan Kaufmann Publishers Inc., San

Francisco, CA, USA, 1995, pp. 1137–1143.
URL http://dl.acm.org/citation.cfm?id=1643031.1643047

[73] J. Walters-Williams, Y. Li, Comparative study of distance functions for nearest neighbors, in: Advanced techniques in computing sciences and software engineering, Springer, 2010, pp. 79–84.

[74] J. Demsar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.

[75] J. J. Valero-Mas, J. Calvo-Zaragoza, J. R. Rico-Juan, J. M. Iñesta, An experimental study on rank methods for prototype selection, Soft Computing 21 (19) (2017) 5703–5715.

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

[77] K. Miettinen, Nonlinear multiobjective optimization, Kluwer Academic Publishers, Boston, 1999.

[78] B. V. Dasarathy, J. S. Sánchez, S. Townsend, Nearest Neighbour Editing and Condensing Tools-Synergy Exploitation, Pattern Anal. Appl. (2000) 19–30.

[79] F. Angiulli, Fast Nearest Neighbor Condensation for Large Data Sets Classification, Knowledge and Data Engineering, IEEE Transactions on 19 (11) (2007) 1450–1464.

[80] J. R. Rico-Juan, J. M. Iñesta, New rank methods for reducing the size of the training set using the nearest neighbor rule, Pattern Recognition Letters 33 (5) (2012) 654–660.

[81] P. Hernandez-Leal, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, J. A. Olvera-Lopez, Instancerank based on borders for instance selection, Pattern Recognition 46 (1) (2013) 365–375.

[82] D. R. Wilson, T. R. Martinez, Instance pruning techniques, in: Proceedings of the Fourteenth International Conference on Machine Learning, ICML '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, pp. 403–411.

[83] H. Brighton, C. Mellish, On the Consistency of Information Filters for Lazy Learning Algorithms, in: J. Żytkow, J. Rauch (Eds.), Principles of Data Mining and Knowledge Discovery, Vol. 1704 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1999, pp. 283–288. doi:10.1007/978-3-540-48247-5_31.

[84] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE conference on computer vision and pattern recognition, Ieee, 2009, pp. 248–255.

[85] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 4510–4520.

Antonio-Javier Gallego is an assistant professor in the Department of Software and Computing Systems at the University of Alicante, Spain. He received B.Sc. & M.Sc. degrees in Computer Science from the University of Alicante in 2004, and a PhD in Computer Science and Artificial Intelligence from the same university in 2012. He has been a researcher on 10 research projects funded by both the Spanish Government and private companies. He has authored or co-authored 45 works published in international journals, conferences, books and book chapters. His research interests include Deep Learning, Pattern Recognition and Computer Vision.

Juan Ramón Rico-Juan received the Ph.D. degree from the University of Alicante, Spain, in 2001. He is currently a Lecturer with University of Alicante and a researcher with pattern recognition and artificial intelligence group. His research interests include areas such as pattern recognition and machine learning, as well as working on structured data learning, distance editing, prototype selection and generation, and deep neural networks. He has participated in 9 national projects which have resulted in the publication in 23 high impact journals (JCR) and 17 in international conferences.

Jose J. Valero-Mas obtained the M.Sc. in Telecommunications Engineering from the University Miguel Hernández of Elche in 2012, the M.Sc. in Sound and Music Computing from the Universitat Pompeu Fabra in 2013, and the Ph.D. in Computer Science from the University of Alicante in 2017. He is currently a postdoctoral researcher with a grant from the Valencian Government at the Department of Software and Computing Systems of the University of Alicante, Spain. His research interests include Pattern Recognition, Machine Learning, Music Information Retrieval, and Signal Processing for which he has co-authored 20 works within international journals, conferences, books and book chapters.

Declaration of interests

[X] The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: