Improving $k$ Nearest Neighbors and Naïve Bayes Classifiers Through Space Transformations and Model Selection

JOSÉ ORTIZ-BEJAR$^{1,2}$, (Member, IEEE), ERIC S. TÉLLEZ$^{2,4}$, (Member, IEEE), MARIO GRAFF$^{2,4}$, (Senior Member, IEEE), DANIELA MOCTEZUMA$^{3,4}$, AND SABINO MIRANDA-JIMÉNEZ$^{2,4}$

$^1$School of Electrical Engineering, Universidad Michoacana de San Nicolás de Hidalgo (UMSNH), Morelia 58000, México
$^2$INFOTEC Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación, México 14050, México
$^3$CentroGEO Centro de Investigación en Ciencias de Información Geoespacial, México 14240, México
$^4$CONACyT Consejo Nacional de Ciencia y Tecnología, Dirección de Cátedras, México 03940, México

Corresponding author: Eric S. Téllez (eric.tellez@infotec.mx)

This work was supported by the Proyecto FORDECyT “Consortio en Inteligencia Artificial,” under Grant 296737.

ABSTRACT Improving classifiers’ performance is the goal of techniques like prototype selection, normalization, and feature mapping; these techniques aim to reduce the complexity and improve the accuracy of models. In this manuscript, we present a boosting artifact for the well-known single-label $k$-Nearest Neighbors ($k$-NN) classifier and the Naïve Bayes (NB) classifier. The improvement comes as a pipeline that includes several data transformations orchestrated by a model selection scheme. The construction of these classifiers relies on the composition of simpler parts found in several open-source libraries and can be effortlessly put together to replicate our proposal. We also explore ensembling and the effect of preprocessing and normalizing the data. We compare our approach experimentally with 17 popular classifiers using raw and rank-based scores on 34 different benchmarks; statistical tests support our results. For instance, our results regarding average performance ranks under balanced error rate show that the models created with our proposal achieve first, third, and fourth-best ranks, compared with 10th position of raw $k$-NN and 14th of raw NB.

INDEX TERMS $k$-nearest neighbors, feature mapping, model selection, automatic classification, supervised learning, machine learning.

I. INTRODUCTION

Classification algorithms are fundamental in pattern recognition and computational intelligence. From text classification [2], [16], [34] to computer vision tasks [29], [69], the complexity and diversity of classification problems have been continuously increasing due to more diverse information sources [6], [31], [41]. These tendencies require algorithms to adapt to the task’s diversity without a significant impact on interpretability and model quality.

An automatic classifier [30] is a model that can predict the class of an object based on a learning process performed on a given training set, i.e., a set of instances $X = \{x_1, \ldots, x_n\}$ and its associated labels (linked to the valid classes) $y = \{y_1, \ldots, y_n\} \subseteq \mathcal{L}$. Note that the training set is part of a universe of valid objects $X \subseteq U$ and $U$ contains all possible inputs. Automatic classifiers are a fundamental part of more complex tasks in many fields of science and industrial processes. The correct selection of a classifier is driven by its performance and computational costs, as described by [73]. A classification problem is binary when the training set is composed only of positive and negative examples of the task being solved; a multiclass classification problem when examples can be marked choosing a label from more than two labels. Regarding the number of labels associated with each example, there exist single-label classification problems, and multi-label classification problems [52].

On the first hand, the $k$-NN method is a straightforward way to learn from examples. It is a non-parametric algorithm that uses all observations to predict outcomes based on a similarity function, see [30]. $k$-NN is flexible enough to work with both similarity and dissimilarity functions. When $k = 1$ the...
method works as follows: Given a vector $u$, its nearest neighbor $x_i \in X$ is located, then the label $y_i$ is associated with $u$. That is, for a dot product similarity, the most similar object is computed as $\arg \max_{1 \leq i \leq |X|} d(u, x_i)$ or $\arg \min_{1 \leq i \leq |X|} d(u, x_i)$ in the case of a dissimilarity (distance) function, i.e., $d : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^+$ (a two-argument function maps to a positive real). When $k > 1$, the prediction can be computed as the most popular label among $k$ nearest neighbors. The similarity between $u$ and some neighbor $x$ can also be used to weight each label.

On the other hand, Bayes classifiers (see [30]) rely on the Bayes theorem to create a prediction model by using the conditional probability of a class label $(l \in \mathcal{L})$ given for a data sample $(x \in X)$. Assuming that $x$ is a $m$-dimensional vector with features $\{a_1, a_2, \ldots, a_m\}$, the probability $P(l|x)$ is written as follows:

$$ P(l|x) = \frac{P(a_1, a_2, \ldots, a_m | \theta) \cdot P(l)}{P(a_1, a_2, \ldots, a_m)} = \prod_{a=1}^{m} \frac{P(l)}{P(a_1, a_2, \ldots, a_m)} = \int_{a} P(l) \quad \text{(1)} $$

where $P(l|x)$ is the probability of $x$ being in class $l$.

The procedure to label $x$ is as follows. Firstly, the probability $P(l|x)$ is calculated for each $l \in \mathcal{L}$, and then, $x$ will be associated with that label with the most significant probability value. Bayes classifiers achieve excellent performance in practice whenever the dataset is representative of the problem under analysis. Nonetheless, as the number of features increases, the computation of conditional probabilities become intractable rapidly.

A simplified approach of the Bayes classifier is to avoid the conditional probability calculation. This assumption is removed by considering each input feature as independent from all other ones. This change gets rid of the conditional probability dependence. Equation 2 describes the probability model with independent events.

$$ P(l|x) = \frac{P(a_1 | l) \cdot P(a_2 | l) \cdots P(a_m | l) \cdot P(l)}{P(a_1) \cdot P(a_2) \cdots P(a_m)} = \prod_{a} \frac{P(l)}{P(a)} \quad \text{(2)} $$

Note that the product of probabilities replaces the joint probability by assuming independence, yielding to the so-called Naïve Bayes (NB).

As mentioned, our approach is based on space transformations and model selection; and for our purposes, space transformations are performed through sampling methods and kernel functions. We consider prototype selection and prototype generation as sampling methods in this contribution, contrary to the related literature [50]. That is, their objective is to generate or select valid points to project objects in the dataset. On the other hand, kernel-methods are typically used to provide linear classifiers with the power of tackling non-linear problems due to the space transformation capabilities of kernel functions; see [14] for an in-depth introduction to kernel methods.

The $k$-NN method exemplifies a straightforward kernel-based method since the similarity function is a kernel function. More sophisticated kernels can be applied straightforwardly by function composition. Conveniently, the $k$-NN approach does not reside on the sense of global separation, like Support Vector Machines (SVM) [32], but in a local-sense of class separation. This strategy helps in solving different kinds of problems without the need for sophisticated kernel functions. The rationale behind Naïve Bayes in our pipeline is that the process produces attributes with some degree of independence in the mapped space. The process will be detailed in §III-C.

Both prototype selection and kernel methods are active research fields; the next section presents a concise literature review about these fields.

### A. OUR CONTRIBUTION

This manuscript focuses on boosting the well-known $k$ nearest neighbors classifier ($k$-NN) and the Naïve Bayes (NB) classifier. Both classifiers are relatively simple methods with some degree of interpretability. The improvement comes from using several kinds of feature transformations based on sampling methods and kernel functions carefully selected through a model selection process. This process produces competitive classifiers that achieve state-of-the-art performances. We focus on single-label classification; our approach is characterized and compared with state-of-the-art alternatives in a large number of diverse on a large variety of binary and multi-class classification tasks.

### B. ROADMAP

This manuscript is organized as follows. The current section introduces our contribution; Section II reviews the related work. Our approach is described in detail in Section III. The experimental comparison and the results are presented in Section IV. Finally, some conclusions and future research directions are given in Section V.

### II. RELATED WORK

Since our contribution is based on the feature transformation based on sampling methods and kernel functions, this section describes the related work based on similar contributions, particularly those based on kernel methods and prototype selection.

Kernel methods have been proved to be useful to improve several machine learning approaches, for instance, fisher discriminant [39], [44], Support Vectors Machines (SVM) [23], [61], [64], manifold learning [7], [9], [13], and $k$-means [65]. They have been used successfully in the non-parametric density estimation based classifier [59] and even as Neural Networks activation functions (for example see [58]). However, under the hood, kernel methods tend to increase data’s dimensionality; fortunately, this situation is efficiently handled whenever the kernel function follows Mercer’s condition. The computations can be made in the original space through a dot product. This technique is the so-called kernel trick [46], [47].

Furthermore, kernel methods have been used successfully in many classification tasks. For instance, Esmaeilzehi and Moghaddam [17] present a classifier with sparse-kernel
representation achieving high performances on face recognition and traditional machine learning datasets. Maggu and Majumdar [42] introduce Kernel transform learning that produces encodings based on Kernel functions. Authors compare their approach to different computer vision tasks. Huang et al. [28] show that Radial Basis Functions (RBF) kernel-based SVM ensembles outperform other classifiers for breast cancer prediction at large scale datasets. Xu et al. [70] explore Kernel k-NN for road traffic states prediction. The deep learning domain has also touched by kernel methods in [12]. Wang et al. [67] present a unifying framework for deep learning and multiple kernel learning, a work of theoretical interest for those interested in using both schemes. Afzal et al. [1] also explore the possibility of using arc-cosine kernel layers and fast methods to produce parameter matrices. Liu et al. [40] presented a linearized kernel sparse representative classifier applied to subcortical brain segmentation achieving relevant image segmentation results.

In its core, all kernel methods need a kernel matrix \( K \); this can have a high computational cost. A common approach to overcome this cost is to create the approximation matrix \( \tilde{K} \), and here Nyström methods are peculiarly efficient. The Nyström method was initially proposed to get approximated solutions for integral equations [49]; fortunately, the Nyström approximation solution was used to speedup kernel-based classifiers firstly by Williams and Seeger [68]. This approximation method is an efficient solution, which is based on sampling a small number of features (i.e., columns in the matrix) to form the matrix \( K \), then sample columns are used to generate an approximation \( \tilde{K} \). Nyström approximation stands that it is possible to apply any kernel method over \( \tilde{K} \) with little impact on the quality of the result [68].

Kumar et al. [36] provide a comprehensive survey to Nyström methods applied to supervised learning. The state-of-the-art sampling strategy for Nyström method is described in [74], [75]; it is based on the usage of the K-Means algorithm for computing column’s centroids; centroids are used to map original data to \( \tilde{K} \). Further analysis and improvements of the k-Means method are reported at [25], [66], where authors determined the k-Means is optimal regarding \( \| K - \tilde{K} \| \) as error function.

In our contribution, we boost the performance of the Naïve Bayes (NB) classifier because, despite its strong assumptions of independence among attributes, the NB classifier is a popular algorithm among practitioners. It is particularly effective in text classification tasks [5], [19], [35], [55] and popular among researchers of some specific domains. For instance, recently, Niazi et al. [48] use NB to monitor and maintain photovoltaic modules; Shen et al. [60] use it to handle dependencies in medical ontologies. In [63], Valdiviezo-Diaz et al. use NB for collaborative filtering in Recommender Systems. Despite its simplicity, NB has competitive performances in several domains while maintaining some degree of explainability.

The prototype selection algorithms aim to reduce the number of items in the training set for both tackling large datasets and minimize the impact of noisy data. Perhaps the most representative method of this kind is the Nearest Centroid (NC) classifier; NC computes one centroid per class, i.e., through the geometric dimension mean [33]. We found multiple methods for prototype generation in the literature. In [38], an optimization method, inspired by the gravitational model, is used to determine a weighted mass factor for each prototype; this method is especially useful for imbalanced data sets. In [37], the initial centroids are optimized by minimizing the hypothesis margin under the structural risk minimization principle. Finally, the kernel method is used to deal with linear inseparability in the original feature space.

Multiple prototypes per class can also be used; some of these methods are Self-generating prototypes (SGP) [18], Reduced Space Partition (RSP) [57], and Pairwise Opposite Class-nearest Neighbor (POC-NN) [53]. These methods split the dataset into clusters of homogeneous elements, i.e., all of them belong to the same class. The main difference among these methods is the way how elements are selected. For instance, SGP uses hyperplanes and singular value decomposition, while RSP selects the furthest elements at each non-homogeneous cluster. In POC-NN, the cluster division is led by POC-NN prototypes, used as locations for setting separating hyperplanes. Triguero et al. [62] survey them and present a taxonomy of them.

III. A MODEL SELECTION APPROACH FOR KERNEL-BASED TRANSFORMATIONS

For our purposes, a classifier is a function \( h : \mathbb{R}^d \rightarrow \mathcal{L} \); that is, \( h \) maps a real-valued \( d \)-dimensional vector to a member of \( \mathcal{L} \), which is a set of categorical values named labels. In particular, \( h \) is an item among \( \mathcal{H} \), the infinite set containing all possible functions with the \( h \)’s signature. Given a classification task \( (X, y, \text{err}) \), the idea is to find a function \( h \in \mathcal{H} \) that reaches an acceptable error ratio under a cross-validation scheme. The selection of such \( h \) is known as training step. Testing set \( X', y' \) is used to validate \( h \)’s performance; the testing set is not available during the training step.

In this context, \( X \) and \( X' \) are subsets of \( \mathbb{R}^d \), of size \( n \) and \( m \), respectively; on the other hand, \( y \) and \( y' \) are subsets of \( \mathcal{L}^n \) and \( \mathcal{L}^m \), respectively. Finally, the function \( \text{err} : \mathcal{L}^n \times \mathcal{L}^m \rightarrow \mathbb{R}^+ \) computes the fitness between its arguments. Therefore, the training step is the process of finding \( h \) such that the function \( \text{err}(y, h(X)) \) is minimized.

The error can be measured in several ways; one of the most popular measures is the error rate, which is defined as the ratio of misclassified items in a set of examples. Regarding binary classification problems,

\[
\text{ER} = \frac{FP + FN}{TP + FP + TN + FN},
\]

here \( FP \), \( FN \), \( TP \), and \( TN \) means for false positive, false negative, true positive, and true negative results. Note that the denominator sums up the entire set of evaluation examples. In multiclass problems with \( \mathcal{L} \) as set of labels, the definition of
error rate can be extended as follows:

\[
ER = \sum_{c \in \mathcal{L}} \frac{FP_c + FN_c}{\#\text{examples}}.
\]

Please note that the error rate is the complement of the well known accuracy, that is \(\text{accuracy} = 1 - ER\). While this measure is well-known and accepted in many domains, it can be tricked on highly imbalanced benchmarks. For instance, consider a dataset with a label occurring 90% of the time; it would be enough to mark objects with the most popular label to achieve an error rate of 0.10.

A more fair measurement of the error in unbalanced benchmarks is the Balanced Error Rate (BER) which is defined as the average proportion error, per-class, as follows:

\[
\text{BER} = \frac{1}{|\mathcal{L}|} \sum_{c \in \mathcal{L}} \frac{FP_c + FN_c}{\#\text{examples}}.
\]

Due to their balancing effect and the diverse nature of our benchmarks, see §IV, we use the BER measure in most comparisons and experiments. Please note that small error values are better than higher ones, and therefore we are interested in \(h\) functions such that \(\text{err}(y, h(X))\) is minimal.

Our method searches for a competitive set of parameters \((k, d, S, f, t, C, s, d', \kappa)\), see Table 1, that can prove competitive performance for the given classification task \((X, y, \text{err})\). These parameters, jointly, define a configuration; the set of all possible configurations is named a configuration space. A configuration is a meta-specification of a classifier (function \(h \in \mathcal{H}\)); the whole set of possible values of all parameters, like those specified in Table 1, is a meta-specification of the configuration space to search for. Since we use stochastic algorithms to explore the configuration space, it is desirable to produce high-quality performance predictors for each configuration; we choose to embed a cross-validation scheme into the \(\text{err}\) function.

The rest of this section details these parameters and describes how to integrate them to learn and predict; here, we also cover the model selection procedure.

**A. SELECTION OF THE SET OF REFERENCES**

We consider four different sampling methods; we distinguish the use of centers and centroids as samples. A center point is an item that is part of the dataset, while a centroid is the geometric mean of a group found by the clustering algorithm. The computation of centroids is straightforward using regions of a Voronoi partition induced by centers. Hereafter, we will use the term references to indicate the use of both centers and centroids.

\(a:\) RANDOM SELECTION

It is a stochastic algorithm based on taking a random sample, evaluate each configuration, and select the best-scored sample among the sample. More detailed, we select \(R \subset X\) randomly; as commented, it is possible to create a set of centroids computing the nearest neighbor in \(R\) of each item in \(X\). The geometric mean of those items having \(c \in R\) has its nearest neighbor produces the centroid associated with \(c\)'s region. In some sense, random selection copies the input distribution; however, there is no control about handling very dense or very sparse regions.

\(b:\) K-MEANS

The set of references is computed employing the K-Means clustering algorithm; we use the \(k\text{means}++\) to select initial centroids (seeds) and reduce intra-cluster variance [3]. We only consider centroid references and the Euclidean distance as the dissimilarity measure for this method.

\(c:\) DENSITY-BASED SELECTION

This iterative algorithm starts with an empty \(R\) and selects a random item in \(c \in X\); the set of \(\kappa\) nearest neighbors of \(c\) in \(X\) is removed; the procedure repeats while \(|X| > 0\). Each \(c\) is added into \(R\) to create the set of references. Also, the set of nearest neighbors are used to compute the related centroids. The number of references is \(k = \lfloor X/\kappa \rfloor\). This approach is related to Density-net construction; the procedure yields to remove most probable regions first and continues removing \(\kappa\) items per iteration.

**Algorithm 1 The Farthest First Traversal Algorithm**

**Require:** A database \(X\) and a distance function \(d\)

**Require:** The number of centers \(k\)

**Ensure:** The set of furthest samples \(R\)

\[
R \leftarrow \{c\} \quad \Rightarrow \text{A randomly chosen } c \text{ from } X
\]

while \(|C| < k\) do

Let \(d_{\text{min}}(x) = \min\{d(x, c) \mid c \in R\}\)

\(w \leftarrow \arg\max\{d_{\text{min}}(x) \mid x \in X \setminus R\}\)

\(R \leftarrow R \cup \{w\}\)

end while

\(d:\) FARthest FIRST TRAVERSAL (FFT)

This algorithm approximates the \(k\)-centers problem, and it was simultaneously proposed by [22] and [26]; the approximation of FFT is at most two times the optimal solution. Alg. 1 defines the Furthest First Traversal method. The algorithm selects a set of centers \(R\) such all items in \(R\) are furthest among them. Two characteristics are preserved for the radius \(r\), the last evaluation of \(d_{\text{min}}\):

---

**TABLE 1. Configuration space of our classification pipeline.**

| Symbol | Name | Possible values |
|--------|------|-----------------|
| \(k\)  | Number of references | \(\{4, 8, 32, 64\}\) |
| \(d\)  | Distance function | (Angle, Euclidean) |
| \(S\)  | Sampling method | (Density, FFT, K-Means, Random) |
| \(f\)  | Kernel function | (Linear, Gaussian, Sigmoid, Cauchy) |
| \(t\)  | Reference’s type | (Centers, Centroids) |
| \(C\)  | Internal classifier | (k-NN, Naive Bayes) |
| \(s\)  | k-NN weighting scheme | (Distance, Uniform) |
| \(d'\) | k-NN distance function | (Angle, Euclidean) |
| \(N\)  | Number of neighbors | \(\{1,5,11,21\}\) |
J. Ortiz-Bejar et al.: Improving $k$ Nearest Neighbors and Naïve Bayes Classifiers Through Space Transformations and Model Selection

FIGURE 1. A two-dimensional toy example of using different sampling strategies to select prototypes; references are drawn as a labeled-red square. Each group is indicated with a unique marker and a unique color; these regions are computed as the nearest neighbors to centers.

1) AN ILLUSTRATIVE EXAMPLE FOR DIFFERENT SAMPLING STRATEGIES

Figures 1 show a 2-dimensional toy-dataset to develop insight about each sampling strategy; the point collection was generated to have three Gaussian-distributed imbalanced and overlapping groups. We compute nine prototypes based on the Euclidean distance for each of the previous sampling strategies, i.e., $c_0, \ldots, c_8$. Note that we are using three times more prototypes than clusters, note how each sampling strategy captures different dataset’s properties, and therefore, they can be used to fit for different tasks.

Figures 1(a) illustrates regions generated with K-Means; prototypes induced by this strategy are concentrated around the higher mass region and evenly distributed around them. Dense regions can be oversampled, while low-density regions can remain untouched. Figure 1(c) illustrates the partition induced by FFT; please notice how prototypes are evenly distant from each other and around the dataset, independently of their mass-density, maximizing the volume of each region; some irrelevant zones can be considered due to outliers. Density-based selection, Fig. 1(d), is guided by dense zones; as explained in §III-A, it ensures the covering of the entire dataset, but low-density zones will have few prototypes, independently of the volume of each region. Lastly, Fig. 1(e) shows how random sampling is prone to select prototypes from high-density clouds; note that favoring dense zones, but in contrast to Density-based selection, dense regions can be oversampled more easily.

The rationale behind our model selection scheme is to recognize the diversity of the particular task.
functions used in this paper, that is, those which are used for our experiments:

\[
\text{linear}(x, c) = d(x, c) \quad (3)
\]

\[
\text{gaussian}(x, c) = \exp(-d(x, c)/\sigma_c) \quad (4)
\]

\[
\text{sigmoid}(x, c) = \frac{1}{1 + \exp(\sigma_c - d(x, c))} \quad (5)
\]

\[
\text{cauchy}(x, c) = \frac{1}{1 + d(x, c)/\sigma_c} \quad (6)
\]

A reader interested in the properties of these kernel functions is referenced to the related literature [21], [27], [46], [76]. Note that these functions are compositions over a distance function \(d\) using for two dataset elements \(x\) and \(c\). Here, \(\sigma_c\) is the maximum intra-cluster distance, that is, the maximum distance from center \(c\) to any object having \(c\) as its nearest neighbor among the set of references; therefore, each region has its own \(\sigma_c\) value. Notice that \(\sigma_c\) can be set as the last \(r\) value for FFT since these references will be distributed evenly.

**C. THE INTERNAL CLASSIFIER**

Our approach uses \(k\)-NN and Naïve Bayes as internal classifiers. These classifiers were selected because they are simple with a fast construction; both are quite popular, can be found in most machine learning libraries, or can be implemented quickly. Additionally, our KMS also works better using them, as described in §IV-A.

Recall that \(k\)-NN is a non-linear classifier and is straightforward to use over the new space. It is worth mentioning that \(k\)-NN uses a distance function to work independently of the kernel used in previous stages. It is possible to use several strategies to decide the label, like using one or more neighbors or even weight neighbors by their rank or actual distance. In the case of NB, we apply Gaussian NB on the output of the kernelized mapping.

**D. LEARNING AND PREDICTION PROCEDURES**

The diagram flow in Figure 2 illustrates the training algorithm of our approach. The diagram shows generic boxes where the parts are embedded; that is, the sampling method, the kernel function, and the internal classifier. Given a training dataset, we select a subset \(R \in X\) using a sampling strategy, a kernel function, and an internal classifier; all these hyper-parameters must be among the valid ones listed in Table 1. Similarly, the hyper-parameters of the internal classifier are also specified. The algorithm’s core idea is to map the original space \(X\) into a kernelized space, induced by the set of references and the kernel function, where the mapped dataset \(\hat{X}\) become a relatively simple instance for the internal classifier.\(^1\)

If memory is scarce, it is also possible to compute \(\hat{X}\) elements online; however, this decision will trade speed by memory. Moreover, some of the steps can be specialized to avoid some computations; for instance, we can avoid the computation of \(\{\sigma_i\}\) for FFT and K-Means, since it is a side-product of these algorithms.

The process of predicting a new object’s label is described in the diagram flow of Figure 3. The prediction process uses the model configuration and the set \(\{\sigma_i\}\); the parameters learned by the classifier are also needed. The classifier is then used to predict the label of the sample using its mapped representation.

Until now, nothing has been said about how to select the hyperparameters for a task; nonetheless, the learning and prediction algorithms listed here work using specific parameters (a configuration) to learn for a task. The process of selecting a competitive model for a task is described in the next paragraphs.
E. HYPERPARAMETER OPTIMIZATION

Once our pipeline is defined, it is necessary to select the precise configuration for each classification task. We perform a model selection over a large configuration space, defined in Table 1; please recall that we describe our set of classifiers through its configuration in §III. We call this process, and our classifier, as Kernel-based Model Selection (KMS).

A KMS model is specified by a configuration \((k, d, S, f, t, C, s, d', \kappa)\) and the \(X, y\) dataset. Each parameter describes the methods used to compute the set of references, what kind of references must be used (i.e., centers or centroids), the kernel function, and its \(\sigma_f\) parameters, the internal classifier, and its hyperparameters. Table 1 produces more than five thousand valid configurations, i.e., we remove those not being valid values like those using NB as the internal classifier and having \(k\)-NN hyperparameters.

The model’s performance error \(err\) leads the selection procedure, and in particular, we use the balanced error rate (BER) measure. In particular, we compute \(err\) using a \(k\)-fold scheme, that is, creates \(k\) different train and test partitions of the data. Each item is part of the training set in \(k - 1\) partitions and part of the test in one partition. Figure 4 shows an \(3\)-fold example partitioning. The training dataset is split into three partitions; a model is trained and validated in each partition. The final error prediction is computed as the average error found in all folds. This validation strategy reduces the chances of overfitting and stabilizes the prediction error. For instance, our experimental section use \(err\) computed with a \(3\)-fold scheme and randomized inputs; the goal is to reduce the chances of a model’s overfitting while ensuring a competitive selection.

While the configuration space evaluation can be performed in several ways, we decide to use the low-cost Random Search (RS) meta-heuristic over parameter boundaries defined in Table 1. The random search consists of uniformly sampling the configuration space, evaluating each configuration’s performance in the sample, and selecting the best performing setup regarding \(err\) function. For instance, our experimental results of the next sections were computed with a sample of 128 configurations; this size was determined experimentally. We also decide to perform Grid Search (GS) over the configuration space with the idea of determining an upper bound of Random search over the defined configuration space. The interested reader in these meta-heuristics and its implications is referred to [8].

The computational advantage of using RS instead of GS is remarkable; for instance, each configuration pipeline is evaluated in 4.11 seconds,\(^3\) the time becomes 8.76 minutes for 128 instances for RS and close to six hours for GS.

Figure 5 illustrates how the selection is performed. On the right side, the flow is described. Firstly, the configuration space is defined (see Table 1). After that, we use a Random search to reduce the number of model training; see Figure 2 for more details about KMS model construction. Once the model is constructed, the \(err\) function is used to measure performance, e.g., the BER function. The error is averaged over \(k\)-fold partitioning; see Figure 4, and the evaluation is made as described in Figure 3. On the left side, an illustration of the flow is depicted.

1) IMPROVING PERFORMANCE VIA ENSEMBLING

To stabilize and improve the performance, we ensemble a group of KMS instances into a KMS ensemble (KMSE); the ensemble is also used to avoid overfitting since several configurations are harder to overfit. The procedure requires selecting a group of models having a proven high performance; therefore, we select top-performing classifiers under the random search application. While this procedure keeps the ensembling method simple, we also maintain a construction cost low since the construction time remains almost identical to any of our hyper-heuristic optimization schemes, Figure 5 illustrates this process.

In particular, our ensemble implementation uses a voting scheme to determine the label of new samples. A procedure to determine the size of the ensemble is studied in §IV-B.

IV. EXPERIMENTAL RESULTS

This section is dedicated to characterize and experimentally prove the performance of our classifiers. Our testing methodology is composed of two parts. The first one characterizes our method using nine datasets for this purpose. The nine databases used to tune our methods, more precisely, select the size of our ensemble and some of the numerical limits presented in Table 1. These benchmarks were obtained from Gunnar Raetsch’s collection;\(^2\) Table 2 shows their characteristics like its dimension, the number of total examples, and their split distribution for training and test. These benchmarks are binary problems (2-classes) and have a relatively low number of training samples. Despite these limitations, these datasets have been used widely to measure classification methods in the literature. Please note that each of these benchmarks has 100 training and test splits, except for the image benchmark, which has only 20 splits. Following the literature standard, we work with average measures over these splits.

\(^2\)Using our open-source implementation https://github.com/kyriox/KernelMethods.jl under CentOS Linux on Intel Xeon E5-2640 v3 @ 2.60GHz.

\(^3\)http://theoval.cmp.uea.ac.uk/matlab/default.html#benchmarks
The second part is dedicated to validation; the core idea is to use an independent collection of datasets to validate the first stage’s decisions. Table 2 also describes these benchmarks; we can observe that the number of classes varies from 2 to 26; the number of dimensions spans from 4 to 170. Moreover, the number of samples is also more varied and more extensive than those found in first-stage benchmarks. Most of these datasets were collected from UCI’s Machine Learning Repository\(^4\) with the exception of semeval and tass. These datasets were generated from the datasets provided for two Twitter Sentiment Analysis challenges, namely, TASS’16 (Spanish Sentiment Analysis, General Corpus \cite{43}) and SemEval’2017 (Task 4: English Sentiment Analysis \cite{45}). Text’s feature vectors were computed using the fastText tool.\(^5\) Table 2 shows the number of elements dedicated for training and test sets, where most benchmarks follow a 70-30 and 80-20 splits, or pretty similar proportions. The exception comes with the tass dataset that uses a 10-90 split. It is worth mentioning that these distributions are standard for these benchmarks, and we kept them to compare with other approaches using the same datasets.

The label’s distribution of each benchmark, for both training and test parts are also shown in terms of the entropy of that distribution, normalized by the entropy of the uniform distribution. Thus, for some benchmark the formulation is as follows:

\[
\text{entropy} = \frac{1}{\log |\mathcal{L}|} \sum_{c \in \mathcal{L}} p_c \log \frac{1}{p_c},
\]

where \(p_c\) is the probability of the label \(c\) in the set (training or test), and \(\mathcal{L}\) the set of possible labels for that benchmark. Note that while some information is lost when normalized, we gain a fixed scale between 0 and 1 for all benchmarks. Small values of the normalized entropy imply that the dataset is unbalanced, and values close to 1 imply that the labels are balanced in population. Characterization benchmarks are almost balanced, while validation benchmarks have a more diverse distribution.

In general, we produce statistics based on BER and ranks. We also give Wilcoxon signed-rank tests to test statistical significance among pairs of classifiers \cite{15}.

\(^4\)https://archive.ics.uci.edu/ml/datasets.php
\(^5\)https://github.com/facebookresearch/fastText
A. PARAMETER ANALYSIS

We reviewed the parameters at the top-1 configurations obtained in the characterization databases. To compute the top-1 classifier, we performed two kinds of selection: a model selection on the training set and a model selection directly on the test set. While the latter scheme is unrealistic, it is an indicator of the upper bound performance. Before analyzing the configuration space parameters, it is relevant to point out that as a first approach, linear classifiers LDA (Linear Discriminant Analysis), Ridge, and Linear SVM (LinearSVC) were included as part of the configuration space. The results obtained gives evidence that $k$-NN and Gaussian NB are more suitable for KMS. The proportion of top-1 places of each of the evaluated classifiers is shown in Figure 6; as can be seen, $k$-NN classifier outperforms others in both train and test evaluation. NB is the second-best classifier over train and test sets. It is fair to consider more or fewer classifiers in the internal selection; however, $k$-NN and NB are both fast and precise in our mapped space. It is also desirable to keep this number low due to its performance impact in the model selection.

Table 3 shows a brief analysis of the structural composition of the classifiers over our characterization benchmarks; more precisely, it shows the empirical probability that each classifier was selected. The table includes the names of the classifiers and the corresponding parameters such as the number of classes, dimensions, samples, training and test samples, and entropy.
parameter is selected to be used by the best classifier using Random search over the configuration space. The table is divided into several groups that indicate the parameter being analyzed. There are three columns, and the first indicates the parameter name and two more columns to list the selection’s empirical probabilities. Please notice how for several of these parameters, the probability of being selected (at the training stage) is close to being uniform or meaningful in any case. We take these distributions in training and test datasets as evidence that these parameters contribute to solving tasks with low errors successfully. That is, we consider them to define our configuration space. Please note that keeping these parameters as possible values produces the problem of which precise configuration must be used for each task, and that is why we rely on model selection techniques, see III-C.

Note that we only look at characterization benchmarks for this analysis. Please also recall that validation benchmarks are more diverse; this was a conscious decision to promote generalization and avoid cherry-picking.

B. DETERMINING THE SIZE OF THE ENSEMBLE

As described in §III-E1, we use ensembles to improve, stabilize the expected performance, and avoid overfitting. Our KMSE includes the selection of top-\(\ell\) performing instances, measured in the training set. In particular, we use the already evaluated classifiers in the Random search procedure; therefore, the cost of creating a KMSE is almost identical to that achieved with the Random search method. The overall prediction is made with a majority voting scheme; in the case of ties, the prediction is randomly selected among the most voted predictions.

Instead of selecting the ensemble’s size based on an additional cross-validation stage, we determine it using a consensus scheme. More detailed, the consensus is measured as the agreement in the label’s predictions between a KMSE with top-\(\ell\) instances and those labels predicted with a KMSE with top-\((\ell + i)\) instances. Figure 7 illustrates the consensus of KMSE on our nine characterization benchmarks; in particular, the figure starts on \(\ell = 3\) and fixes \(i = 2\). Each curve is the proportion of discordant predictions KMSE with different sampling methods, namely, K-means, FFT, Random, and Density-based selection. On the other hand, KMSE with Random search allows the selection of any sampling methods. All discordant ratios were normalized by the maximum ratio per benchmark to obtain values between 0 and 1.

Figure 7 shows how small values of \(\ell\) produce the most significant differences among predictions. It is worth mentioning that the prediction’s computational cost is tightly linked to \(\ell\). We shall select a value that ensures low variance in the predicted labels and a low-cost prediction procedure. Based on the figure, \(\ell\) should be between 9 and 25; these values yield a consensus with relatively low computing cost. We decided to use \(\ell = 15\) since it has a stable consensus in almost all characterization benchmarks; the performance is compared in the following paragraphs.

C. THE PERFORMANCE OF ENSEMBLE-BASED KMSE CLASSIFIERS

Table 4 shows the average BER for our characterization benchmarks. The selected models are the best ones in the training set using the specified method, and the error is measured in the testing set. The table shows two kinds of values: average BER values and average rank positions; methods are ordered by each method’s average rank in all benchmarks. Please recall that lower BER values are better; we also desire lower average ranks since the best possible rank is 1.

The table lists the performance of both single-instance methods and ensemble methods. Ensemble methods listed in the table have fifteen instances under a voting scheme; these
instances were selected as the best-performing ones in the training set, among the evaluated configurations. In particular, both RS and GS perform the optimization considering all sampling methods, while methods listed as a single sampling method perform an RS fixing the sampling method.

Table 4 shows that for the characterization datasets, methods based on ensembling outperform, consistently, its equivalent single instance classifiers; the precise method to determine the size of the ensemble is detailed in §IV-B. The best ensemble method is found with GS, followed by RS. In third place, we found FFT, which achieves two best positions for heart and thyroid. However, both versions of FFT has the lowest performance on ringnorm. So, here resides GS and RS’s power since both explore the entire set of parameters and can turn around when bad cases arise for a particular method. On the rest of the benchmarks, Random selection achieves two best places for banana and twonorm; Density selection performs the better for ringnorm and GS for german. Note that not ensemble versions Density and FFT obtained one best place each one, in diabetes and image datasets, respectively. Please recall that the seven best places arise on ensemble methods. It is worth to mention that ensembles GS and RS obtained just one best place each one, and both occupy the best positions on the global ranking due to their competitive and low variant performance.

Please note that the selection of the ensemble’s size $\ell$ was made with our characterization dataset. On the other hand, Table 5 lists the performance of the same methods (excepting for GS based methods, due to its high computational cost) on the validation benchmarks; ensembles fix $\ell = 15$. Unlike our previous experiment, ensemble methods do not surpass single instance methods. This performance indicates that $\ell$ must be adapted for the precise benchmark; the homogeneity of the characterization datasets was causing performance’s domination. For the validation datasets, all single instance classifiers get a better average rank than their ensemble
version. It is worth to mention that FFT based classifiers are the best for both single instance and ensembles, followed by the version which includes all sampling strategies at a time. Note that the configuration with the best performance in more problems are single instances KMS FFT and KMS RS, but while FFT got the best result for five problems, RS got six best places. KMSE FFT and KMS Density achieve the second and fourth places, respectively; both got the best performance for three problems. This result may suggest that KMSE produces a higher performance variance than single classifier versions; this situation is less evident when using all sampling strategies or only FFT sampling for the optimization process.

It is worth noticing that both ensembles and single-instance classifiers perform pretty similar regarding validation benchmarks if they have the same sampling method. Contrary to characterization benchmarks, ensemble-based classifiers do not dominate. This may be an indicator that ensemble parameters must be adjusted better for validation benchmarks; see §IV-B. Nonetheless, our methodology requires us to fix these parameters based on the characterization benchmarks, so we kept this setup in the following experiments.

Regarding sampling methods, we can conclude that our four strategies are useful, and mostly because we use a model selection scheme that tries to select among them for a given task. In the case we must prioritize their use, the order must be FFT, Density, Random, and K-means, based on our experimental results. To avoid flooding tables and figures with variants of KMS, in the following experiments, we consider KMS-RS, KMSE-RS, and KMS-FFT.

D. COMPARISON AMONG DIFFERENT ALTERNATIVES
As shown in Zhang et al. [73], the comparison of classifiers simplifies selecting the correct technique for tasks. In this experiment, we compare our approach (KMS RS, KMSE RS, and KMS FFT) with 16 implemented in scikit-learn [51] and an additional classifier based on standard Nyström and a Linear SVM; we named the later as LinearSVC Nyström. The number of Nystöm features is fixed to 64, which corresponds with the upper limit for the references compute by our KMS approach.

Table 6 shows the results for the characterization datasets; they are ordered by average rank; the best performance among each classification dataset is in boldface to facilitate the reading. Our KMSE RS gets the best performance on two benchmarks thyroid and waveform with an average rank of 3.78; please recall that the best possible average rank is 1. On the other hand, KMS FFT gets the second-best average rank and the lower BER value for the diabetis benchmark.

### TABLE 5. BER performance comparison of different sampling strategies for the KMS FFT, KMS RS, and KMSE RS on our validation benchmarks; ensembles use \( t = 15 \) instances. The last row summarizes the method's performance with the average rank over benchmarks; note that methods (columns) are sorted by average rank. The best values per row are marked in bold.

| Benchmark          | KMS FFT | KMSE FFT | KMS RS | KMSE RS | KMS Density | Random | KMSE Density | K-means | Random | KMSE K-means |
|--------------------|---------|----------|--------|---------|-------------|--------|--------------|---------|--------|-------------|
| agaricus-lepiota    | 38.52   | 39.09    | 39.43  | 38.48   | 40.19       | 39.42  | 39.97        | 40.13   | 41.95  | 50.91       |
| aps                 | 7.16    | 7.33     | 6.42   | 7.33    | 7.28        | 8.10   | 8.06         | 13.28   | 8.42   | 23.67       |
| bank                | 39.92   | 34.93    | 31.33  | 35.19   | 33.29       | 39.68  | 38.29        | 38.04   | 39.12  | 48.05       |
| banknote            | 0.50    | 0.50     | 0.50   | 0.50    | 0.00        | 0.42   | 0.21         | 0.42    | 1.00   | 0.71        |
| biodeg              | 23.26   | 24.03    | 27.29  | 24.90   | 26.13       | 23.64  | 23.86        | 28.13   | 26.52  | 31.75       |
| census-income       | 32.19   | 32.09    | 27.33  | 32.27   | 28.87       | 32.47  | 31.86        | 39.01   | 33.33  | 42.74       |
| cmc                 | 46.44   | 47.09    | 50.80  | 46.98   | 49.35       | 47.33  | 49.66        | 51.62   | 45.90  | 51.93       |
| drug-consumption     | 69.85   | 68.94    | 69.78  | 70.42   | 73.50       | 71.19  | 72.48        | 70.43   | 71.10  | 70.26       |
| fertility            | 48.00   | 50.00    | 50.00  | 54.00   | 52.00       | 56.00  | 56.00        | 46.00   | 54.00  | 50.00       |
| indian-livel-patient | 35.64   | 36.05    | 34.81  | 35.64   | 42.65       | 37.39  | 44.08        | 35.74   | 45.11  | 35.74       |
| iris                | 2.08    | 2.08     | 3.94   | 2.08    | 2.08        | 4.17   | 2.08         | 1.85    | 2.08   | 2.08        |
| kropt               | 60.58   | 65.70    | 60.39  | 64.76   | 68.80       | 62.93  | 64.33        | 66.49   | 67.61  | 72.76       |
| letter-recognition  | 9.93    | 10.01    | 9.58   | 9.65    | 11.71       | 11.73  | 12.81        | 17.88   | 12.93  | 40.33       |
| ml-prove            | 2.72    | 4.87     | 5.12   | 5.07    | 2.44        | 3.07   | 3.43         | 11.80   | 3.97   | 11.72       |
| musk1               | 14.50   | 13.89    | 20.43  | 16.96   | 16.35       | 15.13  | 17.38        | 18.20   | 14.31  | 24.49       |
| musk2               | 8.39    | 9.75     | 5.43   | 7.28    | 5.28        | 8.65   | 8.36         | 11.98   | 8.24   | 20.72       |
| optdigits           | 3.52    | 4.03     | 4.09   | 4.37    | 4.31        | 4.47   | 4.87         | 3.92    | 4.64   | 6.26        |
| page-blocks         | 16.58   | 17.65    | 18.88  | 19.74   | 23.62       | 22.86  | 15.68        | 43.99   | 26.54  | 67.44       |
| parkinsons          | 25.36   | 35.54    | 30.45  | 33.58   | 28.49       | 25.36  | 33.58        | 28.49   | 33.58  | 35.54       |
| pendigits           | 2.90    | 3.19     | 3.97   | 3.76    | 4.86        | 4.05   | 4.51         | 3.60    | 4.23   | 14.23       |
| segmentation        | 18.81   | 19.38    | 19.38  | 21.67   | 19.67       | 22.52  | 24.14        | 19.14   | 21.71  | 31.19       |
| semeval             | 37.45   | 37.15    | 37.70  | 36.74   | 37.00       | 39.51  | 36.84        | 38.90   | 37.30  | 37.66       |
| tae                 | 50.51   | 50.51    | 46.46  | 53.54   | 52.53       | 40.40  | 52.02        | 44.70   | 44.95  | 57.58       |
| tass                | 46.51   | 46.46    | 48.18  | 47.71   | 47.13       | 46.54  | 48.35        | 50.10   | 47.66  | 49.13       |
| yeast               | 67.67   | 66.53    | 69.91  | 66.97   | 71.32       | 69.55  | 69.39        | 72.10   | 66.42  | 75.86       |
| Avg. Rank           | 3.00    | 4.08     | 4.68   | 4.96    | 5.08        | 5.76   | 5.80         | 6.24    | 6.48   | 8.92        |
TABLE 6. BER performance comparison of the KMS FFT, KMS RS, and KMSE with 16 classifiers and a Linear SVM using Nyström features. Columns are ordered by average rank. Best values are marked in bold.

| Method               | banana | diabetes | german | heart | image | ringnorm | thyroid | twonorm | waveform | Average Rank |
|----------------------|--------|----------|--------|-------|-------|----------|---------|---------|----------|--------------|
| KMSE RS             | 11.81  | 28.55    | 28.14  | 17.15 | 5.09  | 1.82     | 5.80    | 2.35    | 9.58     | 3.78         |
| KMS FFT             | 12.24  | 28.13    | 28.40  | 18.11 | 4.27  | 2.74     | 6.15    | 2.74    | 10.67    | 4.44         |
| KMS RS              | 12.35  | 28.51    | 28.74  | 18.18 | 4.52  | 2.23     | 6.50    | 2.75    | 10.78    | 5.67         |
| LinearSVC Nyström  | 11.03  | 28.79    | 33.07  | 18.20 | 8.59  | 1.63     | 6.58    | 2.62    | 10.88    | 6.67         |
| RBF SVC             | 11.59  | 28.82    | 33.31  | 17.75 | 8.86  | 1.84     | 8.08    | 2.73    | 10.74    | 7.22         |
| GaussianNB          | 41.65  | 28.61    | 30.58  | 16.34 | 36.67 | 1.44     | 11.51   | 2.40    | 12.21    | 7.56         |
| GradientBoost       | 13.84  | 28.52    | 31.08  | 21.06 | 2.09  | 6.65     | 8.25    | 5.74    | 13.62    | 8.33         |
| MLP                  | 18.75  | 28.61    | 32.03  | 18.33 | 3.35  | 10.96    | 9.31    | 2.82    | 11.38    | 8.89         |
| NearestCentroid     | 46.33  | 28.19    | 27.68  | 16.66 | 37.09 | 24.09    | 22.49   | 2.32    | 12.96    | 9.00         |
| AdaBoost            | 28.16  | 29.32    | 31.58  | 22.51 | 3.20  | 7.01     | 8.73    | 5.35    | 14.64    | 10.89        |
| LinearSVC           | 49.97  | 28.47    | 32.44  | 16.80 | 18.65 | 25.18    | 16.91   | 3.58    | 14.20    | 10.89        |
| LogisticRegression  | 49.99  | 28.29    | 32.07  | 16.97 | 18.74 | 25.29    | 20.18   | 2.92    | 13.73    | 11.22        |
| ExtraTrees           | 13.49  | 32.99    | 36.95  | 21.06 | 2.38  | 8.42     | 6.56    | 7.76    | 16.48    | 11.44        |
| RandomForest         | 13.59  | 31.93    | 36.72  | 21.42 | 2.10  | 9.58     | 7.86    | 8.89    | 16.87    | 11.78        |
| kNN                  | 11.92  | 32.12    | 36.18  | 18.64 | 5.26  | 43.72    | 12.06   | 3.76    | 13.80    | 12.22        |
| BernoulliNB          | 45.46  | 31.88    | 32.78  | 16.62 | 39.03 | 28.13    | 32.47   | 5.83    | 14.33    | 14.00        |
| DecisionTrees        | 15.21  | 33.85    | 36.51  | 27.06 | 3.38  | 19.20    | 9.28    | 20.87   | 20.78    | 14.44        |
| SGD                  | 50.28  | 34.03    | 38.50  | 22.19 | 25.12 | 32.65    | 17.69   | 3.89    | 17.61    | 17.00        |
| PassiveAggressive    | 49.04  | 34.68    | 38.65  | 23.31 | 26.24 | 31.48    | 19.48   | 3.85    | 17.10    | 17.11        |
| Perceptron           | 50.15  | 34.56    | 37.47  | 21.59 | 26.69 | 33.49    | 18.18   | 3.92    | 19.03    | 17.44        |

Please note that our KMS RS is the third-best even though it does not achieve any best position; this situation suggests that it can be competitive from a global perspective. LinearSVC with Nyström features achieves one best position for the banana dataset and the fourth place in the average rank. In contrast, the Gaussian Naïve Bayes achieve the sixth position in the global rank having two best positions (heart and ringnorm). This result indicates high variance in some benchmarks; for instance, its performance on both banana and image benchmarks is low. On the contrary, Bernoulli Naïve Bayes reaches the fifth least performing method. The kernel-based method Support Vector Machine (SVM) performance with Radius Basis Kernel (RBF) is pretty good, achieving the fifth-best position in the global ranking with an average rank of 7.22. The linear SVM is six positions below with an average rank of 10.89; this behavior results from having several non-linear problems among our benchmarks.

Other methods with best-performing classifiers are Gradient Boosting and Nearest Centroid for image and twonorm, respectively. Several of the remaining methods achieve pretty good results in some benchmarks but perform pretty bad in others. It is worth mentioning that the \( k \) nearest neighbors method is the sixth least performance method with an average rank of 12.22; this classification method is the KMS’s internal classifier without either the machinery for space transformation hyperparameter optimization.

Figure 8 compares the BER performance of our contributions with the other 17 classifiers. Methods are presented in ascending average rank, so smaller is better; the box-plots show the distribution of ranks per benchmark, per method. Box’s colors are fixed in both figures to simplify tracking methods on both sides; please note that each side has a different scale of ranks.

The left side of the figure shows the performance on our characterization datasets where we tune our configuration space, i.e., select the size of ensembles and the limits of its numeric hyperparameters. We can observe that KMSE achieves the best performance, KMS FFT achieves the second place, while KMS RS gets the third place. It is worth to notice that variance is relatively small for these methods. The LinearSVC Nyström achieves the third better average rank, followed by the Support Vector Machines with RBF kernel and Gaussian Naïve Bayes. While Gradient Boosting got the fifth-best average rank, note that the latter has a small variance. Also, note that \( k \)-NN performs relatively bad since it achieves the 15th position on this rank; please recall that KMS and KMSE use \( k \)-NN and Gaussian Naïve Bayes internally; therefore, our procedure improves over them. A Wilcoxon signed-rank test over BER shows that KMSE RS is statistically similar to KMS FFT and NearestCentroid in both cases with a p-value of 0.16; then, it is similar to GaussianNB with a p-value of 0.07. On the other hand, KMS RS is statistically different from all other methods: for KMS RS and LinearSVC Nyström with a p-value of 0.05, a p-value of 0.03 for MLP, and p-values \( \leq 0.01 \) for the remaining classifiers. While KMS FFT is similar to LinearSVC Nyström with a p-value of 0.43, to RBF SVC with a p-value of 0.3, to NearestCentroid and GaussianNB (both with a p-value of 0.13); and KMS FFT is statistically different from the remaining methods with p-values \( \leq 0.04 \).

On the right side of Figure 8, the performance of KMS FFT achieves the best ranks for validation benchmarks; here, Gradient Boosting achieves the second better place. KMS RS and KMSE RS achieve third and fourth places, respectively; followed by Extra trees, MLP, Decision Trees, and Random Forests; except for MLP, most of them are different...
kinds of tree-based classifiers. $k$-NN achieves the ninth and GaussianNB the sixth positions; it is worth mentioning that our KMS and KMSE methods significantly improve their raw input. In contrast, Support vector machines with RBF kernel pass from the fifth position on the characterization datasets to the fifteenth position in validation benchmarks. Note that LinearSVC with Nyström features exhibited the most extreme drop in the range; the following section provides further analysis of this performance.

A Wilcoxon signed-rank test over BER performances shows that KMS FFT is similar to four of the tree-based models ExtraTrees (p-value of 1.0), Gradient Boosting (p-value of 0.85), Decision Trees (p-value of 0.5), Random Forest (p-value of 0.24). Furthermore, KMS FFT is similar to MLP with a p-value of 0.13, and KMS RS with a p-value of 0.17. On the other side, KMS FFT is statistically better than $k$-NN with a p-value of 0.03; this is relevant since it is the internal classifiers of KMS. Furthermore, KMS FFT performance is statistically different from all the remaining classifiers (KMSE RS included) with a p-value of 0.0.

### E. KMS AND Nyström FEATURES

As was already state, the Nyström approximation is better as the sample size grows; we decided to contrast the performance of linear SVM if the number of Nyström features increases. Figure 9 shows the BER performance for the LinearSVC Nyström (solid blue curve) over the fifteen validation set with more than 1024 training samples; the performance of KMS FFT is set as a baseline (dashed orange line). Please note that the $x$ axis is in log scale.

From Figure 9 it can be observed three different behaviours: the first one is shown in the top rows (aps, bank, census-income, pendigits, semeval and tass) Nyström features got a poor performance, clearly inferior to KMS FFT. The second case is the one exhibited in agaricus-lepiota and cmc datasets where the BER decreases as the number of features grow and get pretty close or equal to KMS FFT. However, at some point, BER starts to increases again ($2^6$ for agaricus-lepiota and $2^9$ for cmc). Finally, for the remaining datasets, BER constantly decreases; for yeast and krkopt, BER decreases until the point Nyström outperforms KMS FFT. This result corroborates that Nyström features may increase classifier performance for some problems, but this is not always the case. Furthermore, even though our KMS classifier has slightly lower performance in some problems, for most of them, KMS excels, and for the same number of references (64), KMS is always better than standard Nyström.

### F. THE EFFECT OF INPUT’s NORMALIZATION

Since our methods are based on kernel functions, their performance is driven by the selected distance function’s effectiveness. Therefore, each variable’s distribution and scale may determine the performance since distance functions; for instance, a variable with a significantly bigger scale may dominate the final distance values hiding useful information from the rest of the variables.

In this experiment, we compare the performance impact of different techniques to scale the input data. The experiment reports results for our contribution, and several popular classifiers, i.e., those explained in the previous experiment. More precisely, we test the following scalers:

- **Standardization** is perhaps one of the most common transformations applied for any machine learning user; the process removes the mean and scales data to have unit variance.
- **MinMax** that scales each feature between a range defined by a minimum and maximum value.
- **MaxAbs** scales each feature by dividing by its maximum absolute value.
- **Quantile** transforms features by using quantile information.
• **Yeo-Johnson** transformer, which is part of the power transformers family whose aim to improve data’ normality and symmetry [71].

Table 7 shows the average rank for each one of the evaluated classifiers in validation benchmarks, using **BER** as the ranking score. The table contains the average rank achieved for each method over raw input and the five scaling methods mentioned above; the last column shows the average rank obtained from the performance achieved on different scaling methods (including the raw input). This experiment compares using average values and presents an average rank with them, i.e., the grand average rank. These results help us determine the robustness of a method to input data and how it can the final classification can be improved with data’s preprocessing. It is not intended to compare the performance on single benchmarks.

As illustrated in Table 7, our KMS FFT achieves the best result, followed by MLP, KMS RS, and KMSE RS. KMS FFT achieves the four best positions for Raw, MinMax, Quantile, and Yeo-Johnson while keeping an excellent performance for all scalers. MLP takes advantage of scaling, going from the eighth rank in raw to a grand average rank of two; it also achieves two best positions for the Standardization and MaxAbs scalers. Gradient boosting changes its global rank, and it goes from a third rank to a fifth because of MLP’s, KMS RS, and KMSE RS improvement. In this experiment, both \( k \)-NN and GaussianNB have a relatively bad performance; it is worth to recall that our KMS is based on them; this significant performance’s difference evidence that our contribution, as a whole, improves its parts. Note that LinearSVC Nyström took advantage of scalers, and it went from the last place to the fourth-least place.

From Table 7 can be seen that KMS based classifiers barely change their rank position for all the evaluated transformations. Base on rank results, it is possible to conclude that KMS has a good generalization strength under data scaling. Despite the robustness of KMS under different data scale transformations, we identify some situations that can affect our approach’s performance. As KMS depends on kernel function similarity, its performance is positively related to the selected distance function; for instance, a variable with a significantly bigger scale may dominate distance values concealing useful information from other variables. In this same line, using a distance function that is not capable of capturing similarities between objects may lead to poor results. However, the issues above are concerned with data representation/analysis and usually are solved for each application domain. To this point, the empirical evidence suggests KMS can lead to competitive results where the data representation is informative enough to induce groups under a kernel function.

A Wilcoxon signed-rank test shows that KMS FFT is only similar to MLP with a \( p \)-value of 0.06 and different from all the remaining methods since the \( p \)-value is \( \leq 0.03 \). Our other KMS approaches (KMSE RS and KMS RS) have similar performance to MLP, too, since \( p \)-values are of 0.16 in both cases; KMSE RS and KMSE RS have a \( p \)-value of 0.44. On the other side, Gradient Boosting is similar to KMSE RS and KMS RS with a \( p \)-value of 0.06. It is worth noticing
that MLP and Gradient Boosting are barely similar since they have a p-value of 0.06. Beyond the mentioned p-values for KMS FFT, KMS RS, KMSE RS, MLP, and Gradient Boosting, other combinations (among them and other methods) yield top-values smaller than 0.05. Therefore, even when KMS-based classifiers excel in their performance, both are statistically similar to other popular classifiers when using different scalers, at least on our validation benchmarks.

PERFORMANCE ON UNBALANCED ERROR RATE FUNCTION (ER)

Our KMS is designed to improve the performance of \(k\)-NN and NB classifiers using space transformations based on sampling methods and kernel functions. Our KMS is also capable of optimizing several error functions using model selection. In particular, we focus on the BER function due to our interest in unbalanced datasets, which are frequent

### TABLE 7
Comparison of the average rank while measuring BER for KMS FFT, KMS RS, KMSE RS, and popular classifiers when using different scaling strategies. Rows are sorted by the last column, the grand average rank (i.e., the mean of average ranks). The raw column also presents ranks inside the parenthesis.

| Classifier   | Raw | Standardization | MinMax | MaxAbs | Quantile | Yeo-Johnson | Average |
|--------------|-----|-----------------|--------|--------|----------|-------------|---------|
| KMS FFT      | 6.36\(_1\) | 6.44 | 5.71 | 6.16 | 5.56 | 5.44 | 5.96 |
| MLP          | 8.28\(_1\) | 6.20 | 6.04 | 6.08 | 5.96 | 5.80 | 6.39 |
| KMS RS       | 7.40\(_1\) | 7.52 | 7.12 | 7.20 | 6.44 | 6.56 | 7.02 |
| KMSE RS      | 7.28\(_1\) | 6.92 | 7.96 | 7.76 | 6.88 | 6.88 | 7.28 |
| GradientBoost| 6.96\(_1\) | 8.12 | 8.24 | 8.40 | 8.36 | 8.40 | 8.08 |
| ExtraTrees   | 8.24\(_1\) | 9.00 | 9.52 | 9.56 | 10.16 | 9.36 | 9.31 |
| DecisionTrees| 8.32\(_1\) | 9.92 | 9.96 | 10.24 | 10.20 | 10.00 | 9.77 |
| RandomForest | 8.88\(_1\) | 9.88 | 10.40 | 9.80 | 10.48 | 10.52 | 9.99 |
| RBF SVC      | 13.00\(_1\) | 9.32 | 9.40 | 9.44 | 9.84 | 9.52 | 10.09 |
| \(k\)-NN     | 9.12\(_1\) | 10.36 | 10.04 | 10.40 | 9.76 | 10.52 | 10.20 |
| NearestCentroid | 12.20\(_1\) | 10.40 | 10.44 | 10.48 | 10.64 | 10.72 | 10.81 |
| LogisticRegression | 10.36\(_1\) | 11.12 | 11.16 | 11.20 | 11.56 | 11.20 | 11.10 |
| LinearSVC    | 12.12\(_1\) | 10.76 | 10.68 | 11.12 | 11.08 | 11.28 | 11.17 |
| GaussianNB   | 9.36\(_1\) | 12.04 | 11.96 | 12.24 | 12.08 | 12.12 | 11.63 |
| AdaBoost     | 10.52\(_1\) | 12.24 | 12.44 | 12.44 | 12.52 | 12.60 | 12.13 |
| LinearSVC Nyström | 16.08\(_1\) | 11.64 | 11.44 | 11.56 | 11.24 | 11.48 | 12.24 |
| Perceptron   | 14.76\(_1\) | 13.92 | 13.68 | 14.16 | 13.92 | 13.88 | 14.05 |
| BernoulliNB  | 13.76\(_1\) | 14.08 | 14.04 | 14.32 | 14.12 | 14.24 | 14.09 |
| PassiveAggressive | 13.32\(_1\) | 15.52 | 14.40 | 13.36 | 14.48 | 13.48 | 14.09 |
| SGD          | 13.68\(_1\) | 14.60 | 13.92 | 14.08 | 13.72 | 16.00 | 14.33 |

### TABLE 8
Comparison of the average rank while measuring the unbalanced error rate ER (i.e., \(1 - \text{accuracy}\)) for KMS FFT, KMS RS, KMSE RS, and popular classifiers when using different scaling strategies. Rows are sorted by the last column, the grand average rank (i.e., the mean of average ranks). The raw column also presents ranks inside the parenthesis.

| Classifier   | Raw | Standardization | MinMax | MaxAbs | Quantile | Yeo-Johnson | Average |
|--------------|-----|-----------------|--------|--------|----------|-------------|---------|
| MLP          | 6.40\(_1\) | 4.04 | 4.08 | 3.88 | 4.00 | 3.60 | 4.33 |
| GradientBoost | 4.24\(_1\) | 5.12 | 5.16 | 5.24 | 5.08 | 5.16 | 5.00 |
| ExtraTrees   | 5.68\(_1\) | 5.88 | 6.20 | 6.08 | 6.96 | 5.72 | 6.09 |
| RBF SVC      | 9.64\(_1\) | 5.32 | 5.20 | 5.72 | 5.52 | 5.52 | 6.15 |
| KMS RS       | 5.80\(_1\) | 6.44 | 6.20 | 6.40 | 6.40 | 6.20 | 6.24 |
| KMS FFT      | 5.16\(_1\) | 6.88 | 6.88 | 6.68 | 6.48 | 7.20 | 6.53 |
| RandomForest | 6.75\(_1\) | 7.04 | 7.40 | 6.36 | 7.60 | 6.88 | 7.00 |
| KMS RS       | 6.52\(_1\) | 6.84 | 7.92 | 7.40 | 6.32 | 7.52 | 7.09 |
| LogisticRegression | 5.96\(_1\) | 7.40 | 7.36 | 7.48 | 7.56 | 7.36 | 7.19 |
| LinearSVC Nyström | 11.48\(_1\) | 7.84 | 7.16 | 7.52 | 6.84 | 7.24 | 8.01 |
| LinearSVC    | 10.92\(_1\) | 7.56 | 7.32 | 7.92 | 7.40 | 7.80 | 8.15 |
| \(k\)-NN     | 7.76\(_1\) | 8.84 | 8.88 | 9.24 | 8.88 | 8.72 | 8.72 |
| AdaBoost     | 8.92\(_1\) | 10.56 | 10.64 | 10.64 | 10.60 | 10.40 | 10.29 |
| DecisionTrees | 9.56\(_1\) | 10.76 | 10.68 | 10.84 | 10.96 | 11.00 | 10.63 |
| SGD          | 13.04\(_1\) | 13.72 | 12.24 | 13.04 | 12.88 | 14.28 | 13.20 |
| PassiveAggressive | 13.00\(_1\) | 13.52 | 14.04 | 12.24 | 14.28 | 12.80 | 13.31 |
| Perceptron   | 14.16\(_1\) | 14.12 | 14.24 | 14.40 | 14.04 | 14.16 | 14.19 |
| BernoulliNB  | 14.76\(_1\) | 15.88 | 15.88 | 16.24 | 15.96 | 15.96 | 15.78 |
| GaussianNB   | 14.44\(_1\) | 16.08 | 16.16 | 16.32 | 16.08 | 16.16 | 15.87 |
| NearestCentroid | 15.84\(_1\) | 16.16 | 16.36 | 16.36 | 16.16 | 16.32 | 16.20 |
in practice. However, the literature has large studies using the unbalanced error rate, most of them for label-balanced datasets. Here we provide the performance regarding ER to simplify the comparison of our approach and other literature methods using this kind of error function.

Table 8 shows KMS’s performance regarding ER (i.e., $1 - \text{accuracy}$). In this setup, MLP and GradientBoost outperform alternatives with an average rank lesser than 5. Our KMSE-RS and KMS-FFT achieve a 5th and 6th position in the rank with an average rank of 6.24 and 6.55; KMS-RS achieves the 8th position with an average rank of 7.00. While these average rank performances are competitive among the compared alternatives, regarding ER as error function, our approach should be used whenever k-NN or NB classifiers are part of the requirements. Our KMS should be preferred for balanced error functions.

V. CONCLUSION

This work introduces a new family of classifiers based on boosting classifiers using space transformation and model selections. The idea is to find a projection for the input data where fast and straightforward classifiers improve its performance, particularly k-NN and NB classifiers. We call our approach Kernel-based Model Selection (KMS).

Our KMS is a pipeline that adapts to a wide range of classification problems, from linear to non-linear tasks. This pipeline is composed of a distance function, a sampling method, a kernel function, and a simple classifier working on the mapped space. We experimentally show that some sampling strategies adapt better to different problems. We also assembled several KMS instances to compose our KMSE; this ensemble-based classifier may improve over single model classifiers and also performs competitively with other industrial-strength alternatives on our benchmarks. Particularly, ensembling is recommended on small training datasets. Regarding computational cost, the prediction cost increases proportionally to the ensemble’s size, but the construction cost is almost similar to the single-instance KMS.

We validated our claims experimentally by using a characterization set of benchmarks and a validation set too. The idea of using two kinds of benchmarks comes from the necessity of finding evidence of generalization through removing the bias induced by hyperparameter tuning and design, so we only touch characterization benchmarks for the early stages when we set hyperparameter limits and values. The results show that our methods are competitive in both characterization and validation benchmarks under average BER as compared with a wide range of industrial-strength classification methods like Gradient Boosting, Neural nets, SVM, AdaBoost, Random Forest, among other methods available in the scikit-learn package. We also found that our methods have the best mean rank among all compared methods under our benchmarks. In general, our methods are simple to implement and have an excellent performance, based on the experimental evidence. For instance, when compared with a list of 17 alternative approaches, varied and proven performance, our approach is consistently among the best performing classifiers. Compared to the raw k-NN and raw NB, the improvement is significant, going from rank 10th and 14th to consistently preserve top positions in ranking comparisons.

Regarding interpretability, it is worth mentioning that it is beyond this work and requires further research; however, we can note that KMS has the same behavior as plain k-NN. More detailed, KMS is an instance-based learning algorithm; the interpretation comes from directly looking at those examples being used for classifying. We can examine these objects in the original space, and KMS can also look at the references used to map those objects to explain decisions. The interpretation related to Naïve Bayes is also linked with references and its attributes.

DISTANCE FUNCTIONS, KNOWN ISSUES, AND FUTURE WORK

Our approach’s fundamental gear is the (dis)similarity notion between any pair of valid objects. Here, we found a myriad of possible alternatives [10], [72]. Similarity functions abstract data and contains domain knowledge in many cases. Some generic functions work well on several data models, like vectors, sets, or sequences. Similarity functions like intersection cardinality, Jaccard, or Dice indexes can be used in set representations; for strings and sequences, we can use Hamming, LCS, or Levenshtein distances. Nonetheless, these functions suppose a uniform weighting and do not take into account any weighting scheme. There exist variants related to these functions that support weighting; however, they will require a depth understanding of the particular domain being solved [24], [54].

In the case of vectors, which is perhaps one of the most popular data models, numerical data is supported straightforwardly; for example, the angle between vectors and Minkowski $L_p$ norms like Manhattan, Euclidean, and Chebychev distance [10], [72]. However, most of these functions suppose that all attributes have the same numerical scale, and the proper preprocessing must be applied before being evaluated. These normalization procedures are routine, but it deserves interest in some cases, such as missing data, attributes encoding special-cases using dedicated values, or whenever a single attribute encodes more attributes using different ranges.

Another weakness arises when data mixes both categorical and numerical data; the data must be transformed or encoded to fit some existing similarity function properly. Please note that classifiers based on decision trees are expected to work well in many of these cases [11]. It is an open research problem to create similarity functions able to measure mixed data taking advantage of the task automatically.

Another essential part of our scheme is sampling methods since each captures different characteristics. Currently, we use them to generate references without information about the task or domain. More research is needed to use task and domain information to select the set of references.
Note that our KMS methods improve \( k \)-NN and NB consistently, both used as internal classifiers in our scheme. However, even when we remain competitive from a global perspective, using the unbalanced error rate ER as the error function achieves lower rank performances than those achieved with BER. The optimal configuration space to adapt more error functions requires additional research.

It is necessary to mention that KMS's performance is linked to its parameters. While our experimental results prove the benefits of our scheme based training and validation benchmarks, it is a reasonable assumption that the configuration spaces can be adjusted to fit individual tasks, and the proper guides to do it is part of further research. The same case applies for KMSE and its hyperparameters, i.e., the ensemble’s size and the summarizing algorithm (simple voting schema for our current implementation) requires further research.

Finally, when \( k \)-NN is used as an internal classifier, and data is described with a large dimensionality, we can observe high predicting costs, especially for very large datasets. It is possible to improve the prediction performance using a metric index \([4],[56]\). Another possible solution is to replace the dataset with fewer elements, i.e., prototypes \([20],[62]\).

The integration of prototype selection and our KMS also requires additional research.

**ACKNOWLEDGMENT**

The authors would like to thank anonymous reviewers for their valuable comments.

**REFERENCES**

[1] A. Alzaf, N. K. Nair, and S. Asharaf, “Deep kernel learning in extreme learning machines,” *Pattern Anal. Appl.*, early access, Jun. 27, 2020, doi: 10.1007/s10044-020-00891-8.

[2] B. Altunel and M. C. Ganiz, “Semantic text classification: A survey of past and recent advances,” *Inf. Process. Manage.*, vol. 54, no. 6, pp. 1129–1153, Nov. 2018.

[3] D. Arthur and S. Vassilvitskii, “K-means++: The advantages of careful seeding,” in *Proc. 18th Ann. ACM-SIAM Symp. Discrete Algorithms*, 2007, pp. 1027–1035.

[4] M. Aumüller, E. Bernhardsson, and A. Faithfull, “ANN-benchmarks: A benchmark tool for approximate nearest neighbor algorithms,” *Inf. Syst.*, vol. 87, Jan. 2020, Art. no. 101374.

[5] J. Awwal, A. A. Bakar, and M. R. Yaakob, “Hybrid N-gram model using Naive Bayes for classification of political sentiments on Twitter,” *Neural Comput. Appl.*, vol. 31, no. 12, pp. 9207–9220, Dec. 2019.

[6] T. Baltrusaitis, C. Ahuja, and L.-P. Morency, “Multimodal machine learning: A survey and taxonomy,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 2, pp. 423–443, Feb. 2019.

[7] M. Belkin and P. Niyogi, “Laplacian eigenmaps for dimensionality reduction and data representation,” *Neural Comput.*, vol. 15, no. 6, pp. 1373–1396, Jun. 2003.

[8] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, 2012.

[9] T. Berry and T. Sauer, “Local kernels and the geometric structure of data,” *Appl. Comput. Harmon. Anal.*, vol. 40, no. 3, pp. 439–469, May 2016.

[10] T. Chan and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2016, pp. 785–794.

[11] Y. Cho and L. K. Saul, “Kernel methods for deep learning,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 342–350.

[12] R. R. Coifman and S. Lafon, “Diffusion maps,” *Appl. Comput. Harmon. Anal.*, vol. 21, no. 1, pp. 5–30, Jul. 2006.

[13] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[14] J. Demšar, “Statistical comparisons of classifiers over multiple data sets,” *J. Mach. Learn. Res.*, vol. 7, no. 1, pp. 1–30, 2006.

[15] A. Dhar, H. Mukherjee, N. S. Dash, and K. Roy, “Text categorization: Past and present,” *Artif. Intell. Rev.*, early access, Sep. 30, 2020, doi: 10.1007/s10462-020-09919-1.

[16] A. Esmaeilzehi and H. A. Moghaddam, “Nonparametric kernel sparse representation-based classifier,” *Pattern Recognit.*, vol. 89, pp. 46–52, Apr. 2019.

[17] H. A. Fayyad, S. R. Hashem, and A. F. Atiya, “Self-generating prototypes and NB combination for pattern classification,” *Pattern Recognit.*, vol. 40, no. 5, pp. 1498–1509, May 2007.

[18] C. A. Flores, R. L. Figueroa, J. E. Pezoa, and Q. Zeng-Treitler, “CREGEX: A biomedical text classifier based on automatically generated regular expressions,” *IEEE Access*, vol. 8, pp. 29270–29280, 2020.

[19] S. García, J. Derrac, J. R. Cano, and F. Herrera, “Prototype selection for nearest neighbor classification: Taxonomy and empirical study,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 3, pp. 417–435, Mar. 2012.

[20] M. G. Genton, “Classes of kernels for machine learning: A statistics perspective,” *J. Mach. Learn. Res.*, vol. 2, no. 12, pp. 299–312, 2001.

[21] T. F. Gonzalez, “Clustering to minimize the maximum intercluster distance,” *Theor. Comput. Sci.*, vol. 38, pp. 293–306, 1985.

[22] J. Guo, P. Yi, R. Wang, Q. Ye, and C. Zhao, “Feature selection for least squares projection twin support vector machine,” *Neurocomputing*, vol. 144, pp. 174–183, Nov. 2014.

[23] D. Hassan, U. Aickelin, and C. Wagner, “Comparison of distance metrics for hierarchical data in medical databases,” in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2014, pp. 3636–3643.

[24] L. He and H. Zhang, “Kernel K-means sampling for Nyström approximation,” *IEEE Trans. Image Process.*, vol. 27, no. 5, pp. 2108–2120, May 2018.

[25] D. S. Hochbaum and D. B. Shmoys, “A best possible heuristic for the K-center problem,” *Math. Oper. Res.*, vol. 10, no. 2, pp. 180–184, May 1985.

[26] T. Hofmann, B. Schölkopf, and A. J. Smola, “Kernel methods in machine learning,” *Ann. Statist.*, vol. 36, no. 3, pp. 1171–1220, Jun. 2008.

[27] M.-W. Huang, C.-W. Chen, W.-C. Lin, S.-W. Ke, and C.-F. Tsai, “SVM and SVM ensembles in breast cancer prediction,” *PLOS ONE*, vol. 12, no. 1, Jan. 2017, Art. no. e0161501.

[28] A. Ioannidou, E. Chatzilari, S. Nikolopoulos, and I. Kompatsiaris, “Deep learning advances in computer vision with 3D data: A survey,” *ACM Comput. Surv.*, vol. 50, no. 1, pp. 1–38, Jun. 2017.

[29] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, vol. 112, New York, NY, USA: Springer, 2013.

[30] S. Ji, W. Xu, M. Yang, and K. Yu, “3D convolutional neural networks for human action recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, Jan. 2013.

[31] T. Joachims, “Making large-scale SVM learning practical,” in *Advances in Kernel Methods—Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA, USA: MIT Press, 1999, ch. 11.

[32] Z. Q. John Lu, “The elements of statistical learning: Data mining, inference, and prediction,” *J. Roy. Statist. Soc., Ser. A (Statist. Soc.*), vol. 173, no. 3, pp. 693–694, Jan. 2010.

[33] A. I. Kadhim, “Survey on supervised machine learning techniques for text classification,” *Artif. Intell. Rev.*, vol. 52, no. 1, pp. 273–292, Jun. 2019.

[34] H. Kang, S. J. Yoo, and D. Han, “Senti-lexicon and improved Naive Bayes algorithms for sentiment analysis of restaurant reviews,” *Expert Syst. Appl.*, vol. 39, no. 5, pp. 6000–6010, Apr. 2012.

[35] X. Li, J. Ouyang, and X. Zhou, “A kernel-based centroid classifier using hypothesis margin,” *J. Expirator. Theor Artif. Intell.*, vol. 28, no. 6, pp. 955–969, Nov. 2016.

[36] C. Liu, W. Wang, G. Tu, Y. Xiang, S. Wang, and F. Lv, “A new centroid-based classification model for text categorization,” *Knowl.-Based Syst.*, vol. 136, pp. 15–26, Nov. 2017.

[37] Q. Liu, H. Lu, and S. Ma, “Improving kernel Fisher discriminant analysis for face recognition,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 1, pp. 42–49, Jan. 2004.
[40] Y. Liu, Y. Wei, and C. Wang, “Subcortical brain segmentation based on atlas registration and linearized kernel sparse representative classifier,” IEEE Access, vol. 7, pp. 31547–31557, 2019.

[41] A. C. Lorenza, L. P. F. Garcia, J. Lehmann, M. C. P. Souto, and T. K. Ho, “How complex is your classification problem?: A survey on measuring classification complexity,” ACM Comput. Surv., vol. 52, no. 5, pp. 1–34, Oct. 2019.

[42] J. Maggu and A. Majumdar, “Kernel transform learning,” Pattern Recognit. Lett., vol. 98, pp. 117–122, Oct. 2017.

[43] E. Martínez-Camara, C. M. Diaz-Galiano, M. A. García-Cumbrales, M. García-Vega, and J. Villena-Roman, “Overview of TASS 2017,” in Proc. Workshop Semantic Anal. SEPLN, 2017, pp. 13–21.

[44] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K. R. Mullers, “Fisher discriminant analysis with kernels,” in Proc. Neural Netw. Signal Process., IEEE Signal Process. Soc. Workshop, Dec. 1999, pp. 41–48.

[45] S. Miranda-Jiménez, M. Graff, E. S. Tellez, and D. Moctezuma, “Kafnets: Kernel-based non-parametric activation functions for neural networks,” Neural Netw., vol. 110, pp. 19–32, Feb. 2019.

[46] I. Triguero, J. Derrac, S. Garcia, and F. Herrera, “A taxonomy and experimental study on prototype generation for nearest neighbor classification,” IEEE Trans. Syst., Man, Cybern. C, Appl. Rev., vol. 42, no. 1, pp. 86–100, Jan. 2012.

[47] P. Valdiviezo-Díaz, F. Ortega, E. Cobos, and R. Lara-Cabrera, “A collaborative filtering approach based on Naïve Bayes classifier,” IEEE Access, vol. 7, pp. 108581–108592, 2019.

[48] V. Vapnik, The Nature of Statistical Learning Theory. New York, NY, USA: Springer, 2013.

[49] S. Wang, E. Zhu, J. Hu, M. Li, K. Zhao, N. Hu, and X. Liu, “Efficient multiple kernel k-Means clustering with late fusion,” IEEE Access, vol. 7, pp. 61109–61120, 2019.

[50] S. Wang, A. Gitteres, and M. W. Mahoney, “Scalable kernel k-Means clustering with Nystrom approximation: Relative-error bounds,” J. Mach. Learn. Res., vol. 20, no. 1, pp. 431–479, 2019.

[51] J. Ortiz-Bejar et al.: Improving k Nearest Neighbors and Naïve Bayes Classifiers Through Space Transformations and Model Selection
MARIO GRAFF (Senior Member, IEEE) received the Ph.D. degree from the School of Computer Science and Electronic Engineering, University of Essex, working under the supervision of Professor Riccardo Poli. He is currently a Researcher with the National Council of Research and Technology (CONACYT) of Mexico commission to INFOTEC. His research interests are evolutionary computation, genetic programming, machine learning, natural language processing, and time series forecasting.

DANIELA MOCTEZUMA received the Ph.D. degree in computer science from the University Rey Juan Carlos, Madrid, Spain. She is currently a Researcher with the National Council of Research and Technology (CONACYT) of Mexico assigned to CentroGEO. Her primary research interests include computer vision, machine learning, text categorization, vision and language, intelligent video surveillance systems, and remote sensing.

SABINO MIRANDA-JIMÉNEZ received the Ph.D. degree in computer science from the Center for Computing Research, National Polytechnic Institute, Mexico, in 2013. He is currently a Researcher with the Mexican National Council of Science and Technology at INFOTEC (CONACYT-INFOTEC) and a member of the National System of Researchers of Mexico. His research interests include natural language processing, text classification, and knowledge representation.