Hubness-aware Classification, Instance Selection and Feature Construction: Survey and Extensions to Time-Series

Nenad Tomašev, Krisztian Buza, Kristóf Marussy, and Piroska B. Kis

Abstract Time-series classification is the common denominator in many real-world pattern recognition tasks. In the last decade, the simple nearest neighbor classifier, in combination with dynamic time warping (DTW) as distance measure, has been shown to achieve surprisingly good overall results on time-series classification problems. On the other hand, the presence of hubs, i.e., instances that are similar to exceptionally large number of other instances, has been shown to be one of the crucial properties of time-series data sets. To achieve high performance, the presence of hubs should be taken into account for machine learning tasks related to time-series. In this chapter, we survey hubness-aware classification methods and instance selection, and we propose to use selected instances for feature construction. We provide detailed description of the algorithms using uniform terminology and notations. Many of the surveyed approaches were originally introduced for vector classification, and their application to time-series data is novel, therefore, we provide experimental results on large number of publicly available real-world time-series data sets.

Key words: time series classification, hubs, instance selection, feature construction

Nenad Tomašev
Institute Jožef Stefan, Artificial Intelligence Laboratory, Jamova 39, 1000 Ljubljana, Slovenia
e-mail: nenad.tomasev@gmail.com

Krisztian Buza
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw (MIMUW), Banacha 2, 02-097 Warszawa, Poland, e-mail: chrisbuza@yahoo.com

Kristóf Marussy
Department of Computer Science and Information Theory, Budapest University of Technology and Economics, Magyar tudósok krt. 2., 1117 Budapest, Hungary, e-mail: marussy@cs.bme.hu

Piroska B. Kis
Department of Mathematics and Computer Science, College of Dunaújváros, Táncsics M. u. 1/a, 2400 Dunaújváros, Hungary, e-mail: pbkism@yahoo.com

1 Introduction

Time-series classification is one of the core components of various real-world recognition systems, such as computer systems for speech and handwriting recognition, signature verification, sign-language recognition, detection of abnormalities in electrocardiograph signals, tools based on electroencephalograph (EEG) signals ("brain waves"), i.e., spelling devices and EEG-controlled web browsers for paralyzed patients, and systems for EEG-based person identification, see e.g. [34, 35, 37, 45]. Due to the increasing interest in time-series classification, various approaches have been introduced including neural networks [26, 38], Bayesian networks [48], hidden Markov models [29, 33, 39], genetic algorithms, support vector machines [14], methods based on random forests and generalized radial basis functions [5] as well as frequent pattern mining [17], histograms of symbolic polynomials [18] and semi-supervised approaches [36]. However, one of the most surprising results states that the simple k-nearest neighbor (kNN) classifier using dynamic time warping (DTW) as distance measure is competitive (if not superior) to many other state-of-the-art models for several classification tasks, see e.g. [9] and the references therein. Besides experimental evidence, there are theoretical results about the optimality of nearest neighbor classifiers, see e.g. [12]. Some of the recent theoretical works focused on a time series classification, in particular on why nearest neighbor classifiers work well in case of time series data [10].

On the other hand, Radovanović et al. observed the presence of hubs in time-series data, i.e., the phenomenon that a few instances tend to be the nearest neighbor of surprisingly lot of other instances [43]. Furthermore, they introduced the notion of bad hubs. A hub is said to be bad if its class label differs from the class labels of many of those instances that have this hub as their nearest neighbor. In the context of k-nearest neighbor classification, bad hubs were shown to be responsible for a large portion of the misclassifications. Therefore, hubness-aware classifiers and instance selection methods were developed in order to make classification faster and more accurate [8, 43, 50, 52, 53, 55].

As the presence of hubs is a general phenomenon characterizing many datasets, we argue that it is of relevance to feature selection approaches as well. Therefore, in this chapter, we will survey the aforementioned results and describe the most important hubness-aware classifiers in detail using unified terminology and notations. As a first step towards hubness-aware feature selection, we will examine the usage of distances from the selected instances as features in a state-of-the-art classifier.

The methods proposed in [50, 52, 53] and [55] were originally designed for vector classification and they are novel to the domain of time-series classification. Therefore, we will provide experimental evidence supporting the claim that these methods can be effectively applied to the problem of time-series classification. The usage of distances from selected instances as features can be seen as transforming the time-series into a vector space. While the technique of projecting the data into a new space is widely used in classification, see e.g. support vector machines [7, 11] and principal component analysis [25], to our best knowledge, the particular proce-

dure we perform is novel in time-series classification, therefore, we will experimentally evaluate it and compare to state-of-the-art time-series classifiers.

The remainder of this chapter is organized as follows: in Sect. 2 we formally define the time-series classification problem, summarize the basic notation used throughout this chapter and shortly describe nearest neighbor classification. Section 3 is devoted to dynamic time warping, and Sect. 4 presents the hubness phenomenon. In Sect. 5 we describe state-of-the-art hubness-aware classifiers, followed by hubness-aware instance selection and feature construction approaches in Sect. 6. Finally, we conclude in Sect. 7.

2 Problem Formulation and Basic Notations

The problem of classification can be stated as follows. We are given a set of instances and some groups. The groups are called classes, and they are denoted as $C_1, \ldots, C_m$. Each instance x belongs to one of the classes.[1] Whenever x belongs to class C_i, we say that the class label of x is C_i. We denote the set of all the classes by $\mathscr{C}$, i.e., $\mathscr{C} = \{C_1, \ldots, C_m\}$. Let $\mathscr{D}$ be a dataset of instances x_i and their class labels y_i, i.e., $\mathscr{D} = \{(x_1, y_1) \ldots (x_n, y_n)\}$. We are given a dataset $\mathscr{D}^{train}$, called *training data*. The task of classification is to induce a function $f(x)$, called *classifier*, which is able to assign class labels to instances not contained in $\mathscr{D}^{train}$.

In real-world applications, for some instances we know (from measurements and/or historical data) to which classes they belong, while the class labels of other instances are unknown. Based on the data with known classes, we induce a classifier, and use it to determine the class labels of the rest of the instances.

In experimental settings we usually aim at measuring the performance of a classifier. Therefore, after inducing the classifier using $\mathscr{D}^{train}$, we use a second dataset $\mathscr{D}^{test}$, called *test data*: for the instances of $\mathscr{D}^{test}$, we compare the output of the classifier, i.e., the predicted class labels, with the true class labels, and calculate the accuracy of classification. Therefore, the task of classification can be defined formally as follows: given two datasets $\mathscr{D}^{train}$ and $\mathscr{D}^{test}$, the task of classification is to induce a classifier $f(x)$ that maximizes prediction accuracy for $\mathscr{D}^{test}$. For the induction of $f(x)$, however, solely $\mathscr{D}^{train}$ can be used, but not $\mathscr{D}^{test}$.

Next, we describe the k-nearest neighbor classifier (kNN). Suppose, we are given an instance $x^* \in \mathscr{D}^{test}$ that should be classified. The kNN classifier searches for those k instances of the training dataset that are most similar to x^*. These k most similar instances are called the k nearest neighbors of x^*. The kNN classifier considers the k nearest neighbors, and takes the majority vote of their labels and assigns this label to x^*: e.g. if $k = 3$ and two of the nearest neighbors of x^* belong to class C_1, while one of the nearest neighbors of x belongs to class C_2, then this 3-NN classifier recognizes x^* as an instance belonging to the class C_1.

[1] In this chapter, we only consider the case when each instance belongs to exactly one class. Note, however, that the presence of hubs may be relevant in the context of multilabel and fuzzy classification as well.

Table 1 Abbreviations used throughout the chapter and the sections where those concepts are defined/explained.

Abbreviation	Full name	Definition
AKNN	adaptive kNN	Sect. 5.5
$BN_k(x)$	bad k-occurrence of x	Sect. 4
DTW	Dynamic Time Warping	Sect. 3
$GN_k(x)$	good k-occurrence of x	Sect. 4
h-FNN	hubness-based fuzzy nearest neighbor	Sect. 5.2
HIKNN	hubness information k-nearest neighbor	Sect. 5.4
hw-kNN	hubness-aware weighting for kNN	Sect. 5.1
INSIGHT	instance selection based on graph-coverage and hubness for time-series	Sect. 6.1
kNN	k-nearest neighbor classifier	Sect. 2
NHBNN	naive hubness Bayesian k-nearest Neighbor	Sect. 5.3
$N_k(x)$	k-occurrence of x	Sect. 4
$N_{k,C}(x)$	class-conditional k-occurrence of x	Sect. 4
$\mathscr{S}_{N_k(x)}$	skewness of $N_k(x)$	Sect. 4
RImb	relative imbalance factor	Sect. 5.5

We use $\mathscr{N}_k(x)$ to denote the set of k nearest neighbors of x. $\mathscr{N}_k(x)$ is also called as the k-neighborhood of x.

3 Dynamic Time Warping

While the kNN classifier is intuitive in vector spaces, in principle, it can be applied to any kind of data, i.e., not only in case if the instances correspond to points of a vector space. The only requirement is that an appropriate distance measure is present that can be used to determine the most similar train instances. In case of time-series classification, the instances are time-series and one of the most widely used distance measures is DTW. We proceed by describing DTW. We assume that a time-series x of length l is a sequence of real numbers: $x = (x[0], x[1], \ldots, x[l-1])$.

In the most simple case, while calculating the distance of two time series x_1 and x_2, one would compare the k-th element of x_1 to the k-th element of x_2 and aggregate the results of such comparisons. In reality, however, when observing the same phenomenon several times, we cannot expect it to happen (or any characteristic pattern to appear) always at exactly the same time position, and the event's duration can also vary slightly. Therefore, DTW captures the similarity of two time series' shapes in a way that it allows for elongations: the k-th position of time series x_1 is compared to the k'-th position of x_2, and k' may or may not be equal to k.

DTW is an edit distance [30]. This means that we can conceptually consider the calculation of the DTW distance of two time series x_1 and x_2 of length l_1 and l_2 respectively as the process of transforming x_1 into x_2. Suppose we have already transformed a prefix (possibly having length zero or l_1 in the extreme cases) of x_1

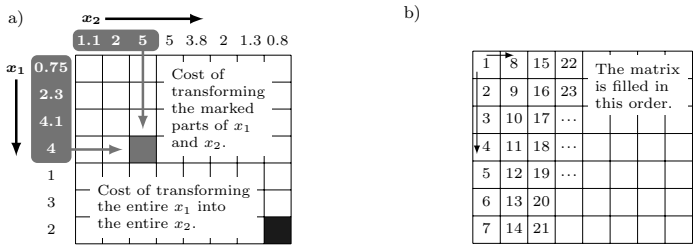

Fig. 1 The DTW-matrix. While calculating the distance (transformation cost) between two time series x_1 and x_2, DTW fills-in the cells of a matrix. a) The values of time series $x_1 = (0.75, 2.3, 4.1, 4, 1, 3, 2)$ are enumerated on the left of the matrix from top to bottom. Time series x_2 is shown on the top of the matrix. A number in a cell corresponds to the distance (transformation cost) between two prefixes of x_1 and x_2. b) The order of filling the positions of the matrix.

into a prefix (possibly having length zero or l_2 in the extreme cases) of x_2. Consider the next elements, the elements that directly follow the already-transformed prefixes, of x_1 and x_2. The following editing steps are possible, both of which being associated with a cost:

1. *replacement* of the next element of x_1 for the next element of x_2, in this case, the next element of x_1 is matched to the next element of x_2, and
2. *elongation* of an element: the next element of x_1 is matched to the last element of the already-matched prefix of x_2 or vice versa.

As result of the replacement step, both prefixes of the already-matched elements grow by one element (by the next elements of x_1 and x_2 respectively). In contrast, in an elongation step, one of these prefixes grows by one element, while the other prefix remains the same as before the elongation step.

The cost of transforming the entire time series x_1 into x_2 is the sum of the costs of all the necessary editing steps. In general, there are many possibilities to transform x_1 into x_2, DTW calculates the one with minimal cost. This minimal cost serves as the distance between both time series. The details of the calculation of DTW are described next.

DTW utilizes the dynamic programming approach [45]. Denoting the length of x_1 by l_1, and the length of x_2 by l_2, the calculation of the minimal transformation cost is done by filling the entries of an $l_1 \times l_2$ matrix. Each number in the matrix corresponds to the distance between a subsequence of x_1 and a subsequence of x_2. In particular, the number in the i-th row and j-th column[2], $d_0^{DTW}(i, j)$ corresponds to the distance between the subsequences $x'_1 = (x_1[0], \ldots, x_1[i])$ and $x'_2 = (x_2[0], \ldots, x_2[j])$. This is shown in Fig. 1.

[2] Please note that the numbering of the columns and rows begin with zero, i.e., the very-first column/row of the matrix is called in this sense as the 0-th column/row.

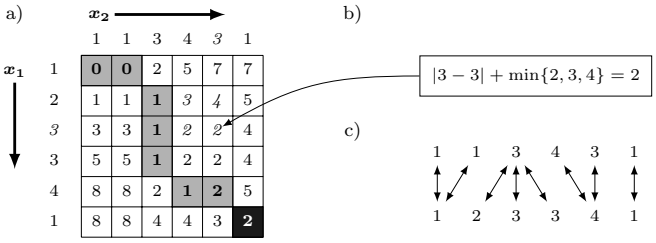

Fig. 2 Example for the calculation of the DTW-matrix. a) The DTW-matrix calculated with $c_{tr}^{DTW}(v_A, v_B) = |v_A - v_B|$, $c_{el}^{DTW} = 0$. The time series x_1 and x_2 are shown on the left and top of the matrix respectively. b) The calculation of the value of a cell. c) The (implicitly) constructed mapping between the values of the both time series. The cells are leading to the minimum in Formula (1), i.e., the ones that allow for this mapping, are marked in the DTW-matrix.

When we try to match the i-th position of x_1 and the j-th position of x_2, there are three possible cases: (i) elongation in x_1, (ii) elongation in x_2, and (iii) no elongation.

If there is no elongation, the prefix of x_1 up to the $(i-1)$-th position is matched (transformed) to the prefix of x_2 up to the $(j-1)$-th position, and the i-th position of x_1 is matched (transformed) to the j-th position of x_2.

Elongation in x_1 at the i-th position means that the i-th position of x_1 has already been matched to at least one position of x_2, i.e., the prefix of x_1 up to the i-th position is matched (transformed) to the prefix of x_2 up to the $(j-1)$-th position, and the i-th position of x_1 is matched again, this time to the j-th position of x_2. This way the i-th position of x_1 is elongated, in the sense that it is allowed to match several positions of x_2. The elongation in x_2 can be described in an analogous way.

Out of these three possible cases, DTW selects the one that transforms the prefix $x_1' = (x_1[0], \ldots, x_1[i])$ into the prefix $x_2' = (x_2[0], \ldots, x_2[j])$ with minimal overall costs. Denoting the distance between the subsequences x_1' and x_2', i.e. the value of the cell in the i-th row and j-th column, as $d_0^{DTW}(i,j)$, based on the above discussion, we can write:

$$d_0^{DTW}(i,j) = c_{tr}^{DTW}(x_1[i], x_2[j]) + \min \left\{ \begin{array}{l} d_0^{DTW}(i, j-1) + c_{el}^{DTW} \\ d_0^{DTW}(i-1, j) + c_{el}^{DTW} \\ d_0^{DTW}(i-1, j-1) \end{array} \right\}. \qquad (1)$$

In this formula, the first, second, and third terms of the minimum correspond to the above cases of elongation in x_1, elongation in x_2 and no elongation, respectively. The cost of matching (transforming) the i-th position of x_1 to the j-th position of x_2 is $c_{tr}^{DTW}(x_1[i], x_2[j])$. If $x_1[i]$ and $x_2[j]$ are identical, the cost of this replacement is zero. This cost is present in all the three above cases. In the cases, when elongation happens, there is an additional elongation cost denoted as c_{el}^{DTW}.

According to the principles of dynamic programming, Formula (1) can be calculated for all i, j in a column-wise fashion. First, set $d_0^{DTW}(0,0) = c_{tr}^{DTW}(x_1[0], x_2[0])$.

Then we begin calculating the very first column of the matrix ($j = 0$), followed by the next column corresponding to $j = 1$, etc. The cells of each column are calculated in order of their row-indexes: within one column, the cell in the row corresponding $i = 0$ is calculated first, followed by the cells corresponding to $i = 1$, $i = 2$, etc. (See Fig. 1.) In some cases (in the very-first column and in the very-first cell of each row), in the min function of Formula (1), some of the terms are undefined (when $i-1$ or $j-1$ equals -1). In these cases, the minimum of the other (defined) terms are taken.

The DTW distance of x_1 and x_2, i.e. the cost of transforming the entire time series $x_1 = (x_1[0], x_1[1], \ldots, x_1[l_1 - 1])$ into $x_2 = (x_2[0], x_2[1], \ldots, x_2[l_2 - 1])$ is

$$d_{DTW}(x_1, x_2) = d_0^{DTW}(l_1 - 1, l_2 - 1). \tag{2}$$

An example for the calculation of DTW is shown in Fig. 2.

Note that the described method implicitly constructs a mapping between the positions of the time series x_1 and x_2: by back-tracking which of the possible cases leads to the minimum in the Formula (1) in each step, i.e., which of the above discussed three possible cases leads to the minimal transformation costs in each step, we can reconstruct the mapping of positions between x_1 and x_2.

For the final result of the distance calculation, the values close to the diagonal of the matrix are usually the most important ones (see Fig. 2 for an illustration). Therefore, a simple, but effective way of speeding-up dynamic time warping is to restrict the calculations to the cells around the diagonal of the matrix [45]. This means that one limits the elongations allowed when matching the both time series (see Fig. 3).

Restricting the warping window size to a pre-defined constant w^{DTW} (see Fig. 3) implies that it is enough to calculate only those cells of the matrix that at most w^{DTW} positions far from the main diagonal along the vertical direction:

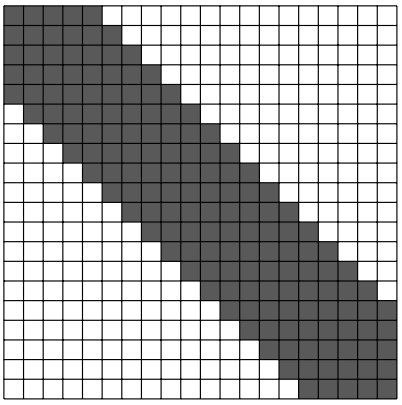

Fig. 3 Limiting the size of the warping window: only the cells around the main diagonal of the matrix (marked cells) are calculated.

$$d_0^{DTW}(i,j) \text{ is calculated} \Leftrightarrow |i-j| \leq w^{DTW}. \quad (3)$$

The warping window size w^{DTW} is often expressed in percentage relative to the length of the time series. In this case, $w^{DTW} = 100\%$ means calculating the entire matrix, while $w^{DTW} = 0\%$ refers to the extreme case of not calculating any entries at all. Setting w^{DTW} to a relatively small value such as 5%, does not negatively affect the accuracy of the classification, see e.g. [9] and the references therein.

In the settings used throughout this chapter, the cost of elongation, c_{el}^{DTW}, is set to zero:

$$c_{el}^{DTW} = 0. \quad (4)$$

The cost of transformation (matching), denoted as c_{tr}^{DTW}, depends on what value is replaced by what: if the numerical value v_A is replaced by v_B, the cost of this step is:

$$c_{tr}^{DTW}(v_A, v_B) = |v_A - v_B|. \quad (5)$$

We set the warping window size to $w^{DTW} = 5\%$. For more details and further recent results on DTW, we refer to [9].

4 Hubs in Time-Series Data

The presence of hubs, i.e., that some few instances tend to occur surprisingly frequently as nearest neighbors while other instances (almost) never occur as nearest neighbors, has been observed for various natural and artificial networks, such as protein-protein-interaction networks or the internet [3, 22]. The presence of hubs has been confirmed in various contexts, including text mining, music retrieval and recommendation, image data and time series [49, 46, 43]. In this chapter, we focus on time series classification, therefore, we describe hubness from the point of view of time-series classification.

For classification, the property of hubness was explored in [40, 41, 42, 43]. The property of hubness states that for data with high (intrinsic) dimensionality, like most of the time series data[3], some instances tend to become nearest neighbors much more frequently than others. Intuitively speaking, very frequent neighbors, or hubs, dominate the neighbor sets and therefore, in the context of similarity-based learning, they represent the centers of influence within the data. In contrast to hubs, there are rarely occurring neighbor instances contributing little to the analytic process. We will refer to them as *orphans* or *anti-hubs*.

In order to express hubness in a more precise way, for a time series dataset $\mathscr{D}$ one can define the *k-occurrence* of a time series x from $\mathscr{D}$, denoted by $N_k(x)$, as the number of time series in $\mathscr{D}$ having x among their k nearest neighbors:

$$N_k(x) = |\{x_i | x \in \mathscr{N}_k(x_i)\}|. \quad (6)$$

[3] In case of time series, consecutive values are strongly interdependent, thus instead of the length of time series, we have to consider the *intrinsic* dimensionality [43].

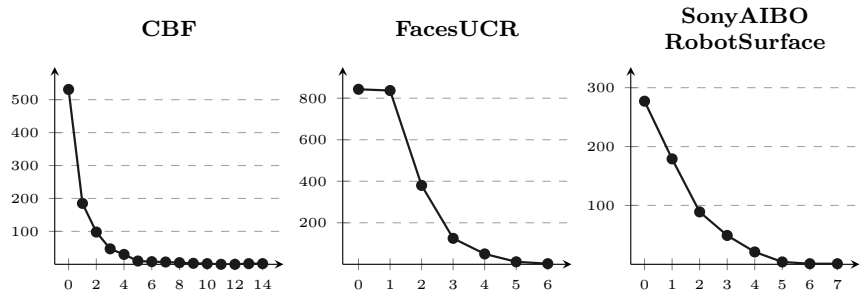

Fig. 4 Distribution of $GN_1(x)$ for some time series datasets. The horizontal axis corresponds to the values of $GN_1(x)$, while on the vertical axis one can see how many instances have that value.

With the term *hubness* we refer to the phenomenon that the distribution of $N_k(x)$ becomes significantly skewed to the right. We can measure this skewness, denoted by $\mathscr{S}_{N_k(x)}$, with the standardized third moment of $N_k(x)$:

$$\mathscr{S}_{N_k(x)} = \frac{E[(N_k(x) - \mu_{N_k(x)})^3]}{\sigma^3_{N_k(x)}} \tag{7}$$

where $\mu_{N_k(x)}$ and $\sigma_{N_k(x)}$ are the mean and standard deviation of the distribution of $N_k(x)$. When $\mathscr{S}_{N_k(x)}$ is higher than zero, the corresponding distribution is skewed to the right and starts presenting a long tail. It should be noted, though, that the occurrence distribution skewness is only one indicator statistic and that the distributions with same or similar skewness can still take different shapes.

In the presence of class labels, we distinguish between *good hubness* and *bad hubness*: we say that the time series x' is a *good k-nearest neighbor* of the time series x, if (i) x' is one of the k-nearest neighbors of x, and (ii) both have the same class labels. Similarly: we say that the time series x' is a *bad k-nearest neighbor* of the time series x, if (i) x' is one of the k-nearest neighbors of x, and (ii) they have different class labels. This allows us to define *good (bad) k-occurrence* of a time series x, $GN_k(x)$ (and $BN_k(x)$ respectively), which is the number of other time series that have x as one of their good (bad respectively) k-nearest neighbors. For time series, both distributions $GN_k(x)$ and $BN_k(x)$ are usually skewed, as it is exemplified in Fig. 4, which depicts the distribution of $GN_1(x)$ for some time series data sets (from the UCR time series dataset collection [28]). As shown, the distributions have long tails in which the good hubs occur.

We say that a time series x is a good (or bad) hub, if $GN_k(x)$ (or $BN_k(x)$ respectively) is exceptionally large for x. For the nearest neighbor classification of time series, the skewness of good occurrence is of major importance, because some few time series are responsible for large portion of the overall error: bad hubs tend to misclassify a surprisingly large number of other time series [43]. Therefore, one has to take into account the presence of good and bad hubs in time series datasets. While

the kNN classifier is frequently used for time series classification, the k-nearest neighbor approach is also well suited for learning under class imbalance [21, 16, 20], therefore hubness-aware classifiers, the ones we present in the next section, are also relevant for the classification of imbalanced data.

The total occurrence count of an instance x can be decomposed into good and bad occurrence counts: $N_k(x) = GN_k(x) + BN_k(x)$. More generally, we can decompose the total occurrence count into the class-conditional counts: $N_k(x) = \sum_{C \in \mathscr{C}} N_{k,C}(x)$ where $N_{k,C}(x)$ denotes how many times x occurs as one of the k nearest neighbors of instances belonging to class C, i.e.,

$$N_{k,C}(x) = |\{x_i | x \in \mathscr{N}_k(x_i) \ \wedge \ y_i = C\}| \tag{8}$$

where y_i denotes the class label of x_i.

As we mentioned, hubs appear in data with high (intrinsic) dimensionality, therefore, hubness is one of the main aspects of the curse of dimensionality [4]. However, dimensionality reduction can not entirely eliminate the issue of bad hubs, unless it induces significant information loss by reducing to a very low dimensional space - which often ends up hurting system performance even more [40].

5 Hubness-aware Classification of Time-Series

Since the issue of hubness in intrinsically high-dimensional data, such as time-series, can not be entirely avoided, the algorithms that work with high-dimensional data need to be able to properly handle hubs. Therefore, in this section, we present algorithms that work under the assumption of hubness. These mechanisms might be either explicit or implicit.

Several hubness-aware classification methods have recently been proposed. An instance-weighting scheme was first proposed in [43], which reduces the bad influence of hubs during voting. An extension of the fuzzy k-nearest neighbor framework was shown to be somewhat better on average [53], introducing the concept of *class-conditional hubness* of neighbor points and building an occurrence model which is used in classification. This approach was further improved by considering the self-information of individual neighbor occurrences [50]. If the neighbor occurrences are treated as random events, the Bayesian approaches also become possible [55, 52].

Generally speaking, in order to predict how hubs will affect classification of non-labeled instances (e.g. instances arising from observations in the future), we can model the influence of hubs by considering the training data. The training data can be utilized to learn a neighbor occurrence model that can be used to estimate the probability of individual neighbor occurrences for each class. This is summarized in Fig. 5. There are many ways to exploit the information contained in the occurrence models. Next, we will review the most prominent approaches.

While describing these approaches, we will consider the case of classifying an instance x^*, and we will denote its nearest neighbors as x_i, $i \in \{1, \ldots, k\}$. We assume

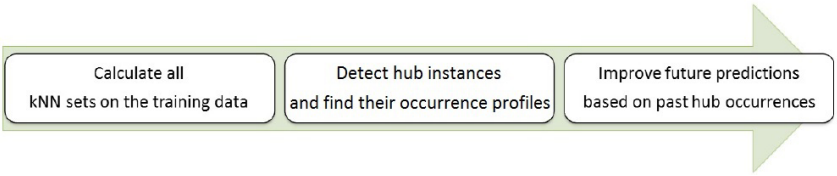

Fig. 5 The hubness-aware analytic framework: learning from past neighbor occurrences.

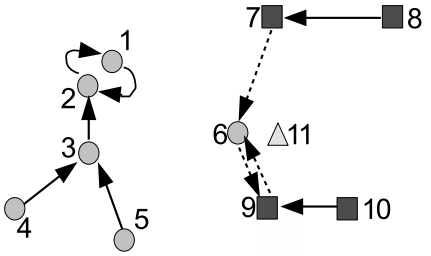

Fig. 6 Running example used to illustrate hubness-aware classifiers. Instances belong to two classes, denoted by circles and rectangles. The triangle is an instance to be classified.

that the test data is not available when building the model, and therefore $N_k(x)$, $N_{k,C}(x)$, $GN_k(x)$, $BN_k(x)$ are calculated on the training data.

5.1 hw-kNN: Hubness-aware Weighting

The weighting algorithm proposed by Radovanović et al. [41] is one of the simplest ways to reduce the influence of bad hubs. They assign lower voting weights to bad hubs in the nearest neighbor classifier. In hw-kNN, the vote of each neighbor x_i is weighted by $e^{-h_b(x_i)}$, where

$$h_b(x_i) = \frac{BN_k(x_i) - \mu_{BN_k(x)}}{\sigma_{BN_k(x)}} \tag{9}$$

is the standardized bad hubness score of the neighbor instance $x_i \in \mathcal{N}_k(x^*)$, $\mu_{BN_k(x)}$ and $\sigma_{BN_k(x)}$ are the mean and standard deviation of the distribution of $BN_k(x)$.

Example 1. We illustrate the calculation of $N_k(x)$, $GN_k(x)$, $BN_k(x)$ and the hw-kNN approach on the example shown in Fig. 6. As described previously, hubness primarily characterizes high-dimensional data. However, in order to keep it simple, this illustrative example is taken from the domain of low dimensional vector classification. In particular, the instances are two-dimensional, therefore, they can be mapped to points of the plane as shown in Fig. 6. Circles (instances 1-6) and rectangles (instances 7-10) denote the training data: circles belong to class 1, while rectangles belong to class 2. The triangle (instance 11) is an instance that has to be classified.

Table 2 $GN_1(x)$, $BN_1(x)$, $N_1(x)$, $N_{1,C_1}(x)$ and $N_{1,C_2}(x)$ for the instances shown in Fig. 6.

Instance	$GN_1(x)$	$BN_1(x)$	$N_1(x)$	$N_{1,C_1}(x)$	$N_{1,C_2}(x)$
1	1	0	1	1	0
2	2	0	2	2	0
3	2	0	2	2	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	2	2	0	2
7	1	0	1	0	1
8	0	0	0	0	0
9	1	1	2	1	1
10	0	0	0	0	0
mean	$\mu_{GN_1(x)} = 0.7$	$\mu_{BN_1(x)} = 0.3$	$\mu_{N_1(x)} = 1$		
std.	$\sigma_{GN_1(x)} = 0.823$	$\sigma_{BN_1(x)} = 0.675$	$\sigma_{N_1(x)} = 0.943$		

For simplicity, we use $k = 1$ and we calculate $N_1(x)$, $GN_1(x)$ and $BN_1(x)$ for the instances of the training data. For each training instance shown in Fig. 6, an arrow denotes its nearest neighbor in the training data. Whenever an instance x' is a good neighbor of x, there is a continuous arrow from x to x'. In cases if x' is a bad neighbor of x, there is a dashed arrow from x to x'.

We can see, e.g., that instance 3 appears twice as good nearest neighbor of other train instances, while it never appears as bad nearest neighbor, therefore, $GN_1(x_3) = 2$, $BN_1(x_3) = 0$ and $N_1(x_3) = GN_1(x_3) + BN_1(x_3) = 2$. For instance 6, the situation is the opposite: $GN_1(x_6) = 0$, $BN_1(x_6) = 2$ and $N_1(x_6) = GN_1(x_6) + BN_1(x_6) = 2$, while instance 9 appears both as good and bad nearest neighbor: $GN_1(x_9) = 1$, $BN_1(x_9) = 1$ and $N_1(x_9) = GN_1(x_9) + BN_1(x_9) = 2$. The second, third and fourth columns of Table 2 show $GN_1(x)$, $BN_1(x)$ and $N_1(x)$ for each instance and the calculated means and standard deviations of the distributions of $GN_1(x)$, $BN_1(x)$ and $N_1(x)$.

While calculating $N_k(x)$, $GN_k(x)$ and $BN_k(x)$, we used $k = 1$. Note, however, that we do not necessarily have to use the same k for the kNN classification of the unlabeled/test instances. In fact, in case of kNN classification with $k = 1$, only one instance is taken into account for determining the class label, and therefore the weighting procedure described above does not make any difference to the simple 1 nearest neighbor classification. In order to illustrate the use of the weighting procedure, we classify instance 11 with $k' = 2$ nearest neighbor classifier, while $N_k(x)$, $GN_k(x)$, $BN_k(x)$ were calculated using $k = 1$. The two nearest neighbors of instance 11 are instances 6 and 9. The weights associated with these instances are:

$$w_6 = e^{-h_b(x_6)} = e^{-\frac{BN_1(x_6) - \mu_{BN_1(x)}}{\sigma_{BN_1(x)}}} = e^{-\frac{2-0.3}{0.675}} = 0.0806 \tag{10}$$

and

$$w_9 = e^{-h_b(x_9)} = e^{-\frac{BN_1(x_9) - \mu_{BN_1(x)}}{\sigma_{BN_1(x)}}} = e^{-\frac{1-0.3}{0.675}} = 0.3545. \tag{11}$$

As $w_9 > w_6$, instance 11 will be classified as rectangle according to instance 9.

From the example we can see that in hw-kNN all neighbors vote by their own label. As this may be disadvantageous in some cases [49], in the algorithms considered below, the neighbors do not always vote by thier own labels, which is a major difference to hw-kNN.

5.2 h-FNN: Hubness-based Fuzzy Nearest Neighbor

Consider the relative class hubness $u_C(x_i)$ of each nearest neighbor x_i:

$$u_C(x_i) = \frac{N_{k,C}(x_i)}{N_k(x_i)}. \tag{12}$$

The above $u_C(x_i)$ can be interpreted as the fuzziness of the event that x_i occurred as one of the neighbors, C denotes one of the classes: $C \in \mathscr{C}$. Integrating fuzziness as a measure of uncertainty is usual in k-nearest neighbor methods and h-FNN [53] uses the relative class hubness when assigning class-conditional vote weights. The approach is based on the fuzzy k-nearest neighbor voting framework [27]. Therefore, the probability of each class C for the instance x^* to be classified is estimated as:

$$u_C(x^*) = \frac{\sum_{x_i \in \mathscr{N}_k(x^*)} u_C(x_i)}{\sum_{x_i \in \mathscr{N}_k(x^*)} \sum_{C' \in \mathscr{C}} u_{C'}(x_i)}. \tag{13}$$

Example 2. We illustrate h-FNN on the example shown in Fig. 6. $N_{k,C}(x)$ is shown in the fifth and sixth column of Table 2 for both classes of circles (C_1) and rectangle (C_2). Similarly to the previous section, we calculate $N_{k,C}(x_i)$ using $k = 1$, but we classify instance 11 using $k' = 2$ nearest neighbors, i.e., x_6 and x_9. The relative class hubness values for both classes for the instances x_6 and x_9 are:

$$u_{C_1}(x_6) = 0/2 = 0\ ,\ \ u_{C_2}(x_6) = 2/2 = 1\ ,$$

$$u_{C_1}(x_9) = 1/2 = 0.5\ ,\ \ u_{C_2}(x_9) = 1/2 = 0.5.$$

According to (13), the class probabilities for instance 11 are:

$$u_{C_1}(x_{11}) = \frac{0+0.5}{0+1+0.5+0.5} = 0.25$$

and

$$u_{C_2}(x_{11}) = \frac{1+0.5}{0+1+0.5+0.5} = 0.75.$$

As $u_{C_2}(x_{11}) > u_{C_1}(x_{11})$, x_{11} will be classified as rectangle (C_2).

Special care has to be devoted to anti-hubs, such as instances 4 and 5 in Fig. 6. Their occurrence fuzziness is estimated as the average fuzziness of points from the same class. Optional distance-based vote weighting is possible.

5.3 NHBNN: Naive Hubness Bayesian k-Nearest Neighbor

Each k-occurrence can be treated as a random event. What NHBNN [55] does is that it essentially performs a Naive-Bayesian inference based on these k events

$$P(y^* = C|\mathscr{N}_k(x^*)) \quad \propto \quad P(C) \prod_{x_i \in \mathscr{N}_k(x^*)} P(x_i \in \mathscr{N}_k|C). \tag{14}$$

where $P(C)$ denotes the probability that an instance belongs to class C and $P(x_i \in \mathscr{N}_k|C)$ denotes the probability that x_i appears as one of the k nearest neighbors of any instance belonging to class C. From the data, $P(C)$ can be estimated as

$$P(C) \approx \frac{|\mathscr{D}_C^{train}|}{|\mathscr{D}^{train}|} \tag{15}$$

where $|\mathscr{D}_C^{train}|$ denotes the number of train instances belonging to class C and $|\mathscr{D}^{train}|$ is the total number of train instances. $P(x_i \in \mathscr{N}_k|C)$ can be estimated as the fraction

$$P(x_i \in \mathscr{N}_k|C) \approx \frac{N_{k,C}(x_i)}{|\mathscr{D}_C^{train}|}. \tag{16}$$

Example 3. Next, we illustrate NHBNN on the example shown in Fig. 6. Out of all the 10 training instances, 6 belong to the class of circles (C_1) and 4 belong to the class of rectangles (C_2). Therefore:

$$|\mathscr{D}_{C_1}^{train}| = 6\ ,\ \ |\mathscr{D}_{C_2}^{train}| = 4\ ,\ \ P(C_1) = 0.6\ ,\ \ P(C_2) = 0.4.$$

Similarly to the previous sections, we calculate $N_{k,C}(x_i)$ using $k = 1$, but we classify instance 11 using $k' = 2$ nearest neighbors, i.e., x_6 and x_9. Thus, we calculate (16) for x_6 and x_9 for both classes C_1 and C_2:

$$P(x_6 \in \mathscr{N}_1|C_1) \approx \frac{N_{1,C_1}(x_6)}{|\mathscr{D}_{C_1}^{train}|} = \frac{0}{6} = 0\ ,\ \ P(x_6 \in \mathscr{N}_1|C_2) \approx \frac{N_{1,C_2}(x_6)}{|\mathscr{D}_{C_2}^{train}|} = \frac{2}{4} = 0.5,$$

$$P(x_9 \in \mathscr{N}_1|C_1) \approx \frac{N_{1,C_1}(x_9)}{|\mathscr{D}_{C_1}^{train}|} = \frac{1}{6} = 0.167\ ,\ \ P(x_9 \in \mathscr{N}_1|C_2) \approx \frac{N_{1,C_2}(x_9)}{|\mathscr{D}_{C_2}^{train}|} = \frac{1}{4} = 0.25\ .$$

According to (14):

$$P(y_{11} = C_1|\mathscr{N}_2(x_{11})) \propto 0.6 \times 0 \times 0.167 = 0$$

$$P(y_{11} = C_2|\mathscr{N}_2(x_{11})) \propto 0.4 \times 0.5 \times 0.25 = 0.05$$

As $P(y_{11} = C_2|\mathscr{N}_2(x_{11})) > P(y_{11} = C_1|\mathscr{N}_2(x_{11}))$, instance 11 will be classified as rectangle.

The previous example also illustrates that estimating $P(x_i \in \mathscr{N}_k|C)$ according to (16) may simply lead zero probabilities. In order to avoid it, instead of (16), we can estimate $P(x_i \in \mathscr{N}_k|C)$ as

$$P(x_i \in \mathscr{N}_k|C) \approx (1-\varepsilon)\frac{N_{k,C}(x_i)}{|\mathscr{D}_C^{train}|} + \varepsilon \tag{17}$$

where $\varepsilon \ll 1$.

Even though k-occurrences are highly correlated, NHBNN still offers some improvement over the basic kNN. It is known that the Naive Bayes rule can sometimes deliver good results even in cases with high independence assumption violation [44].

Anti-hubs, i.e., instances that occur never or with an exceptionally low frequency as nearest neighbors, are treated as a special case. For an anti-hub x_i, $P(x_i \in \mathscr{N}_k|C)$ can be estimated as the average of class-dependent occurrence probabilities of non-anti-hub instances belonging to the same class as x_i:

$$P(x_i \in \mathscr{N}_k|C) \approx \frac{1}{|\mathscr{D}_{class(x_i)}^{train}|} \sum_{x_j \in \mathscr{D}_{class(x_i)}^{train}} P(x_j \in \mathscr{N}_k|C). \tag{18}$$

For more advanced techniques for the treatment of anti-hubs we refer to [55].

5.4 HIKNN: Hubness Information k-Nearest Neighbor

In h-FNN, as in most kNN classifiers, all neighbors are treated as equally important. The difference is sometimes made by introducing the dependency on the distance to x^*, the instance to be classified. However, it is also possible to deduce some sort of global neighbor relevance, based on the occurrence model - and this is what HIKNN was based on [50]. It embodies an information-theoretic interpretation of the neighbor occurrence events. In that context, rare occurrences have higher self-information, see (19). These more informative instances are favored by the algorithm. The reasons for this lie hidden in the geometry of high-dimensional feature spaces. Namely, hubs have been shown to lie closer to the cluster centers [54], as most high-dimensional data lies approximately on hyper-spheres. Therefore, hubs are points that are somewhat less 'local'. Therefore, favoring the rarely occurring points helps in consolidating the neighbor set locality. The algorithm itself is a bit more complex, as it not only reduces the vote weights based on the occurrence frequencies, but also modifies the fuzzy vote itself - so that the rarely occurring points vote mostly by their labels and the hub points vote mostly by their occurrence profiles. Next, we will present the approach in more detail.

The self-information I_{x_i} associated with the event that x_i occurs as one of the nearest neighbors of an instance to be classified can be calculated as

$$I_{x_i} = \log \frac{1}{P(x_i \in \mathscr{N}_k)} \quad , \quad P(x_i \in \mathscr{N}_k) \approx \frac{N_k(x_i)}{|\mathscr{D}^{train}|}. \tag{19}$$

Occurrence self-information is used to define the relative and absolute relevance factors in the following way:

$$\alpha(x_i) = \frac{I_{x_i} - \min_{x_j \in \mathscr{N}_k(x_i)} I_{x_j}}{\log |\mathscr{D}^{train}| - \min_{x_j \in \mathscr{N}_k(x_i)} I_{x_j}}, \quad \beta(x_i) = \frac{I_{x_i}}{\log |\mathscr{D}^{train}|}. \tag{20}$$

The final fuzzy vote of a neighbor x_i combines the information contained in its label with the information contained in its occurrence profile. The relative relevance factor is used for weighting the two information sources. This is shown in (21).

$$P_k(y^* = C|x_i) \approx \begin{cases} \alpha(x_i) + (1-\alpha(x_i)) \cdot u_C(x_i), & y_i = C \\ (1-\alpha(x_i)) \cdot u_C(x_i), & y_i \neq C \end{cases} \tag{21}$$

where y_i denotes the class label of x_i, for the definition of $u_C(x_i)$ see (12).

The final class assignments are given by the weighted sum of these fuzzy votes. The final vote of class C for the classification of instance x^* is shown in (22). The distance weighting factor $d_w(x_i)$ yields mostly minor improvements and can be left out in practice, see [53] for more details.

$$u_C(x^*) \quad \propto \sum_{x_i \in \mathscr{N}_k(x^*)} \beta(x_i) \cdot d_w(x_i) \cdot P_k(y^* = C|x_i). \tag{22}$$

Example 4. Next, we illustrate HIKNN by showing how HIKNN classifies instance 11 of the example shown in Fig. 6. Again, we use $k' = 2$ nearest neighbors to classify instance 11, but we use $N_1(x_i)$ values calculated with $k = 1$. The both nearest neighbors of instance 11 are x_6 and x_9. The self-information associated with the occurrence of these instances as nearest neighbors:

$$P(x_6 \in \mathscr{N}_1) = \frac{2}{10} = 0.2 \ , \quad I_{x_6} = \log_2 \frac{1}{0.2} = \log_2 5,$$

$$P(x_9 \in \mathscr{N}_1) = \frac{2}{10} = 0.2 \ , \quad I_{x_9} = \log_2 \frac{1}{0.2} = \log_2 5.$$

The relevance factors are:

$$\alpha(x_6) = \alpha(x_9) = 0 \ , \quad \beta(x_6) = \beta(x_9) = \frac{\log_2 5}{\log_2 10}.$$

The fuzzy votes according to (21):

$$P_k(y^* = C_1|x_6) = u_{C_1}(x_6) = 0 \ , \quad P_k(y^* = C_2|x_6) = u_{C_2}(x_6) = 1,$$

$$P_k(y^* = C_1|x_9) = u_{C_1}(x_9) = 0.5 \ , \quad P_k(y^* = C_2|x_9) = u_{C_2}(x_9) = 0.5.$$

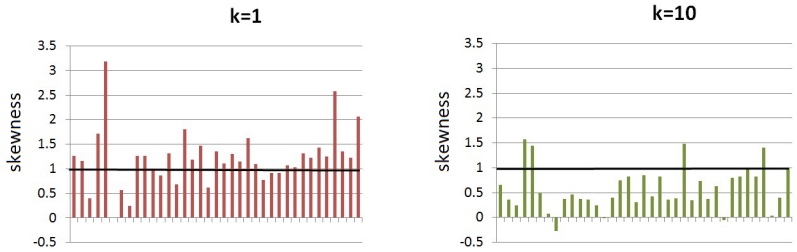

Fig. 7 The skewness of the neighbor occurrence frequency distribution for neighborhood sizes $k = 1$ and $k = 10$. In both figures, each column corresponds to a dataset of the UCR repository. The figures show the change in the skewness, when k is increased from 1 to 10.

The sum of fuzzy votes (without taking the distance weighting factor into account):

$$u_{C_1}(x_{11}) = \frac{\log_2 5}{\log_2 10} \cdot 0 + \frac{\log_2 5}{\log_2 10} \cdot 0.5\ , \quad u_{C_2}(x_{11}) = \frac{\log_2 5}{\log_2 10} \cdot 1 + \frac{\log_2 5}{\log_2 10} \cdot 0.5.$$

As $u_{C_2}(x_{11}) > u_{C_1}(x_{11})$, instance 11 will be classified as rectangle (C_2).

5.5 Experimental Evaluation of Hubness-aware Classifiers

Time series datasets exhibit a certain degree of hubness, as shown in Table 3. This is in agreement with previous observations [43].

Most datasets from the UCR repository [28] are balanced, with close-to-uniform class distributions. This can be seen by analyzing the relative imbalance factor (RImb) of the label distribution which we define as the normalized standard deviation of the class probabilities from the absolutely homogenous mean value of $1/m$, where m denotes the number of classes, i.e., $m = |\mathscr{C}|$:

$$\mathrm{RImb} = \sqrt{\frac{\sum_{C \in \mathscr{C}} (P(C) - 1/m)^2}{(m-1)/m}}. \tag{23}$$

In general, an occurrence frequency distribution skewness above 1 indicates a significant impact of hubness. Many UCR datasets have $\mathscr{S}_{N_1(x)} > 1$, which means that the first nearest neighbor occurrence distribution is significantly skewed to the right. However, an increase in neighborhood size reduces the overall skewness of the datasets, as shown in Fig. 7. Note that only a few datasets have $\mathscr{S}_{N_{10}(x)} > 1$, though some non-negligible skewness remains in most of the data. Yet, even though the overall skewness is reduced with increasing neighborhood sizes, the degree of major hubs in the data increases. This leads to the emergence of strong centers of influence.

Table 3 The summary of relevant properties of the UCR time series datasets. Each dataset is described both by a set of basic properties (size, number of classes) and some hubness-related quantities for two different neighborhood sizes, namely: the skewness of the k-occurrence distribution ($\mathcal{S}_{N_k(x)}$), the percentage of bad k-occurrences ($BN_k^{\%}$), the degree of the largest hub-point ($\max N_k(x)$). Also, the relative imbalance of the label distribution is given, as well as the size of the majority class (expressed as a percentage of the entire dataset).

Data set	size	$\lvert\mathscr{C}\rvert$	$\mathcal{S}_{N_1(x)}$	$BN_1^{\%}$	$\max N_1(x)$	$\mathcal{S}_{N_{10}(x)}$	$BN_{10}^{\%}$	$\max N_{10}(x)$	RImb	$P(c_M)$
50words	905	50	1.26	19.6%	5	0.66	36.2%	33	0.16	4.2%
Adiac	781	37	1.16	33.5%	6	0.36	51.8%	28	0.02	3.7%
Beef	60	5	0.40	50%	3	0.25	62%	18	0.0	20.0%
Car	120	4	1.72	24.2%	6	1.57	39.2%	42	0.0	25%
CBF	930	3	3.18	0.0%	22	1.44	0.1%	57	0.0	33.3%
ChlorineConcentration	4307	3	0.0	0%	2	0.50	31.6%	21	0.30	53.6%
CinC-ECG-torso	1420	4	0.57	0.0%	5	0.08	0.1%	25	0.0	25%
Coffee	56	2	0.25	3.6%	3	-0.27	31.6%	19	0.04	51.8%
Cricket-X	780	12	1.26	16.7%	6	0.38	33.1%	28	0.0	8.3%
Cricket-Y	780	12	1.26	18.2%	6	0.47	34.9%	30	0.0	8.3%
Cricket-Z	780	12	1.00	15.9%	5	0.37	33.2%	27	0.0	8.3%
DiatomSizeReduction	332	4	0.86	0.1%	5	0.36	0.1%	23	0.19	30.7%
ECG200	200	2	1.31	12.0%	6	0.24	19.7%	25	0.33	66.5%
ECGFiveDays	884	2	0.68	0.01%	4	-0.01	0.04%	25	0.0	50%
FaceFour	112	4	1.80	4.5%	7	0.40	13.5%	26	0.09	30.4%
FacesUCR	2250	14	1.19	1.4%	6	0.75	5.3%	36	0.12	14.5%
FISH	350	7	1.47	18.6%	6	0.83	33.5%	36	0.0	14.3%
Gun-Point	200	2	0.62	2.0%	4	0.31	5.2%	23	0.0	50.0%
Haptics	463	5	1.36	53.6%	6	0.85	60.9%	35	0.04	21.6%
InlineSkate	650	7	1.11	43.1%	6	0.42	59.3%	28	0.09	18.0%
ItalyPowerDemand	1096	2	1.30	4.5%	6	0.83	5.1%	46	0.01	50.1%
Lighting2	121	2	1.15	9.1%	5	0.36	28.8%	23	0.206	60.3%
Lighting7	143	7	1.63	21.0%	6	0.39	39.1%	26	0.15	26.6%
MALLAT	2400	8	1.09	1.3%	6	1.48	2.7%	53	0.0	12.5%
MedicalImages	1141	10	0.78	18.2%	4	0.35	31.6%	26	0.48	52.1%
MoteStrain	1272	2	0.91	5.0%	6	0.73	9.3%	33	0.08	53.9%
OliveOil	60	4	0.92	11.7%	4	0.38	28.0%	23	0.26	41.7%
OSULeaf	442	6	1.07	25.3%	5	0.63	44.8%	29	0.11	21.9%
Plane	210	7	1.03	0.0%	5	-0.05	0.1%	21	0.0	14.3%
SonyAIBORobotSurface	621	2	1.32	1.9%	7	0.80	4.4%	33	0.12	56.2%
SonyAIBORobotSurfaceII	980	2	1.22	1.5%	6	0.82	6.5%	35	0.23	61.6%
SwedishLeaf	1125	15	1.43	14.7%	8	0.97	23.5%	41	0.0	6.7%
Symbols	1020	6	1.25	1.8%	6	0.83	3.0%	38	0.02	17.7%
Synthetic-control	600	6	2.58	0.7%	12	1.40	2.0%	54	0.0	16.7&
Trace	200	4	1.36	0.0%	6	0.04	2.5%	22	0.0	25%
TwoLeadECG	1162	2	1.22	0.1%	6	0.40	0.3%	33	0.0	50%
Two-Patterns	5000	4	2.07	0.0%	14	1.01	0.1%	46	0.02	26.1%
Average	917.6	7.6	1.21	11.7%	6.24	0.58	21.2%	31.54	0.08	31.0%

We evaluated the performance of different kNN classification methods on time series data for a fixed neighborhood size of $k = 10$. A slightly larger k value was chosen, since most hubness-aware methods are known to perform better in such cases, as better and more reliable neighbor occurrence models can be inferred from more occurrence information. We also analyzed the algorithm performance over a range of different neighborhood sizes, as shown in Fig. 8. The hubness-aware classification methods presented in the previous sections (hw-kNN, NHBNN, h-FNN and HIKNN) were compared to the baseline kNN [15] and the adaptive kNN (AKNN) [56], where the neighborhood size is recalculated for each query point

based on initial observations, in order to consult only the relevant neighbor points. AKNN does not take the hubness of the data into account.

The tests were run according to the 10-times 10-fold cross-validation protocol and the statistical significance was determined by employing the corrected resampled t-test. The detailed results are given in Table 4.

The adaptive neighborhood approach (AKNN) does not seem to be appropriate for handling time-series data, as it performs worse than the baseline kNN. While hw-kNN, NHBNN and h-FNN are better than the baseline kNN in some cases, they do not offer significant advantage overall which is probably a consequence of a relatively low neighbor occurrence skewness for $k = 10$ (see Fig. 7). The hubness is, on average, present in time-series data to a lower extent than in text or images [49] where these methods were previously shown to perform rather well.

On the other hand, HIKNN, the information-theoretic approach to handling voting in high-dimensional data, clearly outperforms all other tested methods on these time series datasets. It performs significantly better than the baseline in 19 out of 37 cases and does not perform significantly worse on any examined dataset. Its average accuracy for $k = 10$ is 84.9, compared to 82.5 achieved by kNN. HIKNN outperformed both baselines (even though not significantly) even in case of the ChlorineConcentration dataset, which has very low hubness in terms of skewness, and therefore other hubness-aware classifiers worked worse than kNN on this data. These observations reaffirm the conclusions outlined in previous studies [50], arguing that HIKNN might be the best hubness-aware classifier on medium-to-low hubness data, if there is no significant class imbalance. Note, however, that hubness-aware classifiers are also well suited for learning under class imbalance [16, 20, 21].

In order to show that the observed improvements are not merely an artifact of the choice of neighborhood size, classification tests were performed for a range of different neighborhood sizes. Figure 8 shows the comparisons between kNN and HIKNN for $k \in [1,20]$, on Car and Fish time series datasets. There is little difference between kNN and HIKNN for $k = 1$ and the classifiers performance is similar in this case. However, as k increases, so does the performance of HIKNN, while the performance of kNN either decreases or increases at a slower rate. Therefore, the differences for $k = 10$ are more pronounced and the differences for $k = 20$ are even greater. Most importantly, the highest achieved accuracy by HIKNN, over all tested neighborhoods, is clearly higher than the highest achieved accuracy by kNN.

These results indicate that HIKNN is an appropriate classification method for handling time series data, when used in conjunction with the dynamic time warping distance.

Of course, choosing the optimal neighborhood size in k-nearest neighbor methods is a non-trivial problem. The parameter could be set by performing cross-validation on the training data, though this is quite time-consuming. If the data is small, using large k values might not make a lot of sense, as it would breach the locality assumption by introducing neighbors into the kNN sets that are not relevant for the instance to be classified. According to our experiments, HIKNN achieved very good performance for $k \in [5,15]$, therefore, setting $k = 5$ or $k = 10$ by default would usually lead to reasonable results in practice.

Table 4 Accuracy ± standard deviation (in %) for hubness-aware classifiers. All experiments were performed for $k = 10$. The symbols •/○ denote statistically significant worse/better performance ($p < 0.05$) compared to kNN. The best result in each line is in bold.

Data set	kNN	AKNN	hw-kNN	NHBNN	hFNN	HIKNN
50words	72.4 ± 3.2	68.8 ± 3.3 •	71.3 ± 3.3	74.1 ± 2.7	76.4 ± 3.0 ○	**80.1** ± **2.9** ○
Adiac	61.0 ± 4.1	58.6 ± 3.3	60.7 ± 4.0	63.8 ± 3.9	63.1 ± 3.8	**65.9** ± **4.0** ○
Beef	**51.5** ± **15.6**	31.8 ± 15.0 •	43.9 ± 14.3	50.4 ± 15.2	47.6 ± 15.6	47.8 ± 15.8
Car	71.7 ± 9.8	73.9 ± 9.4	76.0 ± 9.4	76.0 ± 8.6	76.8 ± 9.2	**79.1** ± **8.7** ○
CBF	**100.0** ± **0.0**	**100.0** ± **0.0**	**100.0** ± **0.0**	**100.0** ± **0.0**	**100.0** ± **0.0**	**100.0** ± **0.0**
Chlorine[a]	87.6 ± 1.3	68.7 ± 1.6 •	74.5 ± 1.6 •	68.7 ± 1.6 •	80.2 ± 1.4 •	**91.8** ± **1.0** ○
CinC-ECG-torso	99.8 ± 0.2	99.6 ± 0.3	99.7 ± 0.2	99.6 ± 0.3	**99.9** ± **0.1**	**99.9** ± **0.1**
Coffee	74.4 ± 15.9	68.7 ± 15.4	68.6 ± 15.1	71.2 ± 15.3	70.0 ± 15.2	**76.9** ± **13.4**
Cricket-X	78.0 ± 2.9	71.0 ± 2.9 •	77.9 ± 2.6	78.2 ± 3.0	79.3 ± 2.8	**81.9** ± **2.6** ○
Cricket-Y	77.7 ± 3.1	67.1 ± 3.1 •	76.7 ± 3.3	77.5 ± 3.2	79.9 ± 3.1	**81.6** ± **2.9** ○
Cricket-Z	78.0 ± 3.1	70.9 ± 3.3 •	78.1 ± 3.1	78.6 ± 3.0	79.8 ± 3.0	**82.2** ± **2.8** ○
Diatom[b]	98.5 ± 1.6	**100.0** ± **0.0**	98.7 ± 1.4	99.1 ± 1.1	99.6 ± 0.8	99.6 ± 0.7
ECG200	85.9 ± 5.8	83.0 ± 4.9	85.3 ± 5.1	83.9 ± 4.6	85.2 ± 5.0	**87.1** ± **5.0**
ECGFiveDays	98.5 ± 0.7	97.7 ± 1.0 •	98.3 ± 1.0	98.3 ± 0.9	98.8 ± 0.7	**99.0** ± **0.6**
FaceFour	91.1 ± 6.3	92.2 ± 5.7	91.8 ± 6.7	94.0 ± 5.1	93.6 ± 5.4	**94.1** ± **5.3**
FacesUCR	96.4 ± 0.7	95.9 ± 0.8	96.7 ± 0.8	97.1 ± 0.7	97.5 ± 0.7	**98.0** ± **0.6**
FISH	77.3 ± 5.1	72.5 ± 5.7 •	77.1 ± 5.3	77.1 ± 5.5	77.9 ± 5.1	**81.7** ± **4.5** ○
Gun-Point	**99.1** ± **1.4**	95.4 ± 3.4 •	98.2 ± 2.0	97.9 ± 2.3	98.8 ± 1.6	99.0 ± 1.4
Haptics	50.5 ± 4.8	52.6 ± 5.5	51.3 ± 4.8	50.1 ± 4.8	51.9 ± 4.7	**54.3** ± **4.6** ○
InlineSkate	55.0 ± 3.8	44.8 ± 3.9 •	53.1 ± 3.7	53.6 ± 3.8	54.3 ± 4.0	**60.1** ± **4.4** ○
Italy[c]	96.5 ± 1.1	**96.9** ± **1.1**	96.7 ± 1.1	**96.9** ± **1.0**	**96.9** ± **1.0**	96.8 ± 1.0
Lighting2	79.5 ± 8.2	76.5 ± 8.6	78.1 ± 8.2	74.4 ± 8.9	80.0 ± 7.6	**83.9** ± **7.2** ○
Lighting7	69.2 ± 9.1	65.9 ± 9.1	70.0 ± 9.5	74.3 ± 9.0	71.5 ± 8.7	**76.3** ± **7.8** ○
MALLAT	98.5 ± 0.5	98.4 ± 0.5	98.5 ± 0.5	98.4 ± 0.5	98.5 ± 0.4	**98.8** ± **0.4**
MedicalImages	79.2 ± 2.7	73.6 ± 3.1 •	78.9 ± 2.9	69.2 ± 2.5 •	79.2 ± 2.8	**81.3** ± **2.6** ○
MoteStrain	94.0 ± 1.4	94.3 ± 1.3	94.7 ± 1.4	94.3 ± 1.4	94.7 ± 1.3	**95.4** ± **1.2** ○
OliveOil	76.2 ± 13.7	73.4 ± 12.5	78.9 ± 13.0	**87.9** ± **9.9** ○	78.6 ± 12.6	87.1 ± 10.7 ○
OSULeaf	67.0 ± 5.0	60.1 ± 5.4 •	66.0 ± 5.1	64.1 ± 5.1	66.1 ± 4.5	**72.5** ± **4.9** ○
Plane	**99.7** ± **0.7**	98.9 ± 1.6	99.4 ± 1.0	99.5 ± 0.9	**99.7** ± **0.7**	**99.7** ± **0.7**
Sony[d]	96.7 ± 1.5	**98.9** ± **1.0** ○	97.8 ± 1.2 ○	97.4 ± 1.3 ○	97.7 ± 1.3 ○	97.5 ± 1.3 ○
SonyII[e]	95.8 ± 1.4	94.8 ± 1.6	96.0 ± 1.4	95.8 ± 1.4	96.3 ± 1.3	**96.5** ± **1.2**
SwedishLeaf	82.8 ± 2.1	82.6 ± 2.6	84.8 ± 2.3 ○	85.8 ± 2.2 ○	85.4 ± 2.3 ○	**86.0** ± **2.2** ○
Symbols	97.4 ± 1.0	97.1 ± 1.1	97.6 ± 0.9	97.6 ± 0.9	98.0 ± 0.8 ○	**98.2** ± **0.8** ○
Synthetic-control	99.0 ± 0.7	97.2 ± 1.4 •	**99.5** ± **0.6**	99.2 ± 0.7	99.2 ± 0.7	99.0 ± 0.7
Trace	98.9 ± 1.6	98.4 ± 2.0	98.4 ± 2.1	98.7 ± 1.8	99.7 ± 0.7	**99.9** ± **0.4**
TwoLeadECG	99.8 ± 0.2	99.7 ± 0.2	99.8 ± 0.2	99.8 ± 0.2	**99.9** ± **0.1**	**99.9** ± **0.1**
Two-Patterns	99.9 ± 0.0	99.9 ± 0.0	99.9 ± 0.0	99.9 ± 0.0	**100.0** ± **0.0**	**100.0** ± **0.0**
Average	82.5	79.4	81.9	82.2	82.9	**84.9**

[a]ChlorineConcentration, [b]DiatomSizeReduction, [c]ItalyPowerDemand, [d]SonyAIBORobotSurface, [e]SonyAIBORobotSurfaceII

6 Instance Selection and Feature Construction for Time-Series Classification

In the previous section, we described four approaches that take hubness into account in order to make time-series classification more accurate. In various applications, however, besides classification accuracy, the classification time is also important. Therefore, in this section, we present hubness-aware approaches for speeding-up time-series classification. First, we describe instance selection for kNN classification of time-series. Subsequently, we focus on feature construction.

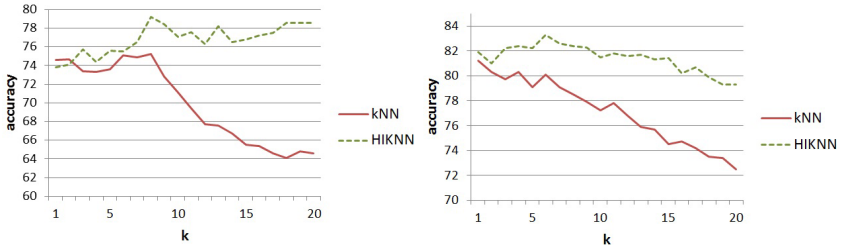

Fig. 8 The accuracy (in %) of the basic kNN and the hubness aware HIKNN classifier over a range of neighborhood sizes $k \in [1,20]$, on Car and Fish time series datasets.

6.1 Instance Selection for Speeding-Up Time-Series Classification

Attempts to speed up DTW-based nearest neighbor classification fall into four major categories: i) speeding-up the calculation of the distance of two time series (by e.g. limiting the warping window size), ii) indexing, iii) reducing the length of the time series used, and iv) instance selection. The first class of techniques was already mentioned in Sect. 3. For an overview of techniques for indexing and reduction of the length of time-series and more advanced approaches for limiting the warping window size, we refer to [9] and the references therein. In this section, we focus on how to speed up time-series classification via instance selection. We note that instance selection is orthogonal to the other speed-up techniques, i.e., instance selection can be combined with those techniques in order to achieve highest efficiency.

Instance selection (also known as *numerosity reduction* or *prototype selection*) aims at discarding most of the training time series while keeping only the most informative ones, which are then used to classify unlabeled instances. In case of conventional nearest neighbor classification, the instance to be classified, denoted as x^*, will be compared to all the instances of the training data set. In contrast, when applying instance selection, x^* will only be compared to the selected instances of the training data. For time-series classification, despite the techniques aiming at speeding-up DTW-calculations, the calculation of the DTW distance is still relatively expensive computationally, therefore, when selecting a relatively small number of instances, such as 10% of the training data, instance selection can substantially speed-up the classification of time-series.

While instance selection is well explored for general nearest neighbor classification, see e.g. [1, 6, 19, 24, 32], there are only few works for the case of time series. Xi et al. [57] presented the FastAWARD approach and showed that it outperforms state-of-the-art, general-purpose instance selection techniques applied for time-series.

FastAWARD follows an iterative procedure for discarding time series: in each iteration, the rank of all the time series is calculated and the one with lowest rank is discarded. Thus, each iteration corresponds to a particular number of kept time series. Furthermore, Xi et al. argue that the optimal warping window size depends

Algorithm 1 INSIGHT

Require: Time series dataset $\mathscr{D}$, Score Function $g(x)$ /* e.g. one of $GN_1(x)$, $RS(x)$ or $XI(x)$ */, Number of selected instances n_{sel}
Ensure: Set of selected instances (time series) $\mathscr{D}'$

1: Calculate score function $g(x)$ for all $x \in \mathscr{D}$
2: Sort all the time series in $\mathscr{D}$ according to their scores $g(x)$
3: Select the top-ranked n_{sel} time series and return the set containing them

on the number of kept time series. Therefore, FastAWARD calculates the optimal warping window size dependent on the number of kept time series.

In this section, we present a hubness-aware instance selection technique which was originally introduced in [8]. This approach is simpler and therefore computationally much cheaper than FastAWARD while it selects better instances, i.e., instances that allow more accurate classification of time-series than the ones selected by FastAWARD.

In [8] coverage graphs were proposed to model instance selection, and the instance selection problem was formulated as finding the appropriate subset of vertices of the coverage graph. Furthermore, it was shown that maximizing the coverage is NP-complete in general. On the other hand, for the case of time-series classification, a simple approach performed surprisingly well. This approach is called *Instance Selection based on Graph-coverage and Hubness for Time-series* or INSIGHT.

INSIGHT performs instance selection by assigning a score to each instance and selecting instances with the highest scores (see Algorithm 1), therefore the "intelligence" of INSIGHT is hidden in the applied score function. Next, we explain the suitability of several score functions in the light of the hubness property.

- **Good 1-occurrence Score –** INSIGHT can use scores that take into account how many times an instance appears as good neighbor of other instances. Thus, a simple score function is the *good 1-occurrence score* $GN_1(x)$.
- **Relative Score –** While x is being a good hub, at the same time it may appear as bad neighbor of several other instances. Thus, INSIGHT can also consider scores that take bad occurrences into account. This leads to scores that relate the good occurrence of an instance x to either its total occurrence or to its bad occurrence. For simplicity, we focus on the following *relative score*, however, other variations could be used too: *relative score* $RS(x)$ of a time series x is the fraction of good 1-occurrences and total occurrences plus one (to avoid division by zero):

$$RS(x) = \frac{GN_1(x)}{N_1(x)+1}. \tag{24}$$

- **Xi's score –** Notably, $GN_k(x)$ and $BN_k(x)$ allows us to interpret the ranking criterion used by Xi et al. in FastAWARD [57] as another form of score for relative hubness:

$$XI(x) = GN_1(x) - 2BN_1(x). \tag{25}$$

As reported in [8], INSIGHT outperforms FastAWARD both in terms of classification accuracy and execution time. The second and third columns of Table 5 present the average accuracy and corresponding standard deviation for each data set, for the case when the number of selected instances is equal to 10% of the size of the training set. The experiments were performed according to the 10-fold cross-validation protocol. For INSIGHT, the good 1-occurrence score is used, but we note that similar results were achieved for the other scores too.

In clear majority of the cases, INSIGHT substantially outperformed FastAWARD. In the few remaining cases, their difference are remarkably small (which are not significant in the light of the corresponding standard deviations). According to the analysis reported in [8], one of the major reasons for the suboptimal performance of FastAWARD is that the skewness degrades during the FastAWARD's iterative instance selection procedure, and therefore FastAWARD is not able to select the best instances in the end. This is crucial because FastAWARD discards the worst instance in each iteration and therefore the final iterations have substantial impact on which instances remain, i.e., which instances will be selected by FastAWARD.

6.2 Feature Construction

As shown in Sect. 6.1, the instance selection approach focusing on good hubs leads to overall good results. Previously, once the instances were selected, we simply used them as training data for the kNN classifier. In a more advanced classification schema, instead of simply performing nearest neighbor classification, we can use distances from selected instances as features. This is described in detail below.

First, we split the training data $\mathscr{D}^{train}$ into two disjoint subsets $\mathscr{D}_1^{train}$ and $\mathscr{D}_2^{train}$, i.e., $\mathscr{D}_1^{train} \cap \mathscr{D}_2^{train} = \emptyset$, $\mathscr{D}_1^{train} \cup \mathscr{D}_2^{train} = \mathscr{D}^{train}$. We select some instances from $\mathscr{D}_1^{train}$, denote these selected instances as $x_{sel,1}, x_{sel,2}, \ldots, x_{sel,l}$. For each instance $x \in \mathscr{D}_2^{train}$, we calculate its DTW-distance from the selected instances and use these distances as features of x. This way, we map each instance $x \in \mathscr{D}_2^{train}$ into a vector space:

$$x_{mapped} = \big(d_{DTW}(x, x_{sel,1}), d_{DTW}(x, x_{sel,2}), \ldots, d_{DTW}(x, x_{sel,l})\big). \tag{26}$$

The representation of the data in a vector space allows the usage of any conventional classifier. For our experiments, we trained logistic regression from the Weka software package[4]. We used the mapped instances of $\mathscr{D}_2^{train}$ as training data for logistic regression.

When classifying an instance $x^* \in \mathscr{D}^{test}$, we map x^* into the same vector space as the instances of $\mathscr{D}_2^{train}$, i.e., we calculate the DTW-distances between x^* and the selected instances $x_{sel,1}, x_{sel,2}, \ldots, x_{sel,l}$ and we use these distances as features of x^*. Once the features of x^* are calculated, we use the trained classifier (logistic regression in our case) to classify x^*.

[4] http://www.cs.waikato.ac.nz/ml/weka/

Table 5 Accuracy ± standard deviation (in %) for FastAWARD, INSIGHT and feature construction methods. The symbols •/○ denote statistically significant worse/better performance ($p < 0.05$) compared to FastAWARD. The best result in each line is in bold.

Data set	FastAWARD	INSIGHT	HubFeatures	RndFeatures
50words	52.6 ± 4.1	64.2 ± 4.6 ○	**65.5 ± 3.5** ○	65.2 ± 4.9 ○
Adiac	34.8 ± 5.8	46.9 ± 4.9 ○	48.5 ± 4.8 ○	**51.0 ± 5.2** ○
Beef	35.0 ± 17.4	33.3 ± 10.5	**38.3 ± 19.3**	36.7 ± 15.3
Car	45.0 ± 11.9	**60.8 ± 14.5** ○	47.5 ± 22.9	55.8 ± 20.8
CBF	97.2 ± 3.4	**99.8 ± 0.6** ○	**99.8 ± 0.5** ○	**99.8 ± 0.5** ○
ChlorineConcentration	53.7 ± 2.3	**73.4 ± 3.0** ○	54.8 ± 1.4	54.6 ± 1.7
CinC-ECG-torso	40.6 ± 8.9	96.6 ± 1.4 ○	**98.7 ± 1.2** ○	**98.7 ± 1.2** ○
Coffee	56.0 ± 30.9	60.3 ± 21.3	**66.0 ± 21.4**	59.0 ± 16.1
DiatomSizeReduction	97.2 ± 2.6	96.6 ± 5.8	**100.0 ± 0.0** ○	98.8 ± 2.2
ECG200	75.5 ± 11.3	83.5 ± 9.0	**86.0 ± 7.0** ○	84.5 ± 7.6
ECGFiveDays	93.7 ± 2.7	94.5 ± 2.0	96.5 ± 2.2 ○	**96.7 ± 1.7** ○
FaceFour	71.4 ± 14.1	89.4 ± 12.8 ○	**91.1 ± 8.4** ○	90.2 ± 8.9 ○
FacesUCR	89.2 ± 1.9	**93.4 ± 2.1** ○	91.4 ± 2.2 ○	91.5 ± 1.8 ○
FISH	59.1 ± 8.2	**66.6 ± 8.5**	59.7 ± 6.5	62.9 ± 9.3
Gun-Point	80.0 ± 12.4	**93.5 ± 5.9** ○	85.5 ± 6.4	84.5 ± 3.7
Haptics	30.3 ± 6.8	**43.5 ± 6.0** ○	33.9 ± 9.4	35.4 ± 7.9
InlineSkate	19.7 ± 5.6	**43.4 ± 7.7** ○	36.3 ± 7.5 ○	36.5 ± 8.7 ○
ItalyPowerDemand	**96.0 ± 2.0**	95.7 ± 2.8	95.8 ± 2.4	95.7 ± 2.1
Lighting2	69.4 ± 13.4	67.0 ± 9.6	67.7 ± 6.4	**75.1 ± 8.9**
Lighting7	44.7 ± 12.6	51.0 ± 8.2	**61.4 ± 10.5** ○	60.8 ± 9.3 ○
MALLAT	55.1 ± 9.8	**96.9 ± 1.3** ○	96.4 ± 1.5 ○	96.8 ± 1.1 ○
MedicalImages	64.2 ± 3.3	69.3 ± 4.9 ○	**73.4 ± 3.9** ○	73.0 ± 3.3 ○
MoteStrain	86.7 ± 4.2	90.8 ± 2.7 ○	93.3 ± 2.4 ○	**93.6 ± 2.2** ○
OliveOil	63.3 ± 10.0	71.7 ± 13.0	**80.0 ± 17.2** ○	75.0 ± 23.9 ○
OSULeaf	41.9 ± 5.3	53.8 ± 5.7 ○	**57.0 ± 6.7** ○	54.5 ± 8.1 ○
Plane	87.6 ± 15.5	**98.1 ± 3.2**	95.7 ± 5.2	94.8 ± 6.1
SonyAIBORobotSurface	92.4 ± 3.2	97.6 ± 1.7 ○	98.4 ± 1.3 ○	**98.7 ± 1.3** ○
SonyAIBORobotSurfaceII	91.9 ± 1.5	91.2 ± 3.3	94.6 ± 2.3 ○	**95.1 ± 2.8** ○
SwedishLeaf	68.3 ± 4.6	75.6 ± 4.8 ○	77.6 ± 5.1 ○	**77.9 ± 5.6** ○
Symbols	95.7 ± 1.8	**96.6 ± 1.6**	95.1 ± 2.1	95.6 ± 2.2
Synthetic-control	92.3 ± 6.8	**97.8 ± 2.6** ○	95.3 ± 2.6	94.0 ± 3.4
Trace	78.0 ± 11.7	**89.5 ± 7.2** ○	73.0 ± 8.6	74.0 ± 8.1
TwoLeadECG	97.8 ± 1.3	**98.9 ± 1.2**	93.6 ± 2.8 •	93.3 ± 2.9 •
Two-Patterns	40.7 ± 2.7	**98.7 ± 0.7** ○	98.4 ± 0.5 ○	98.4 ± 0.7 ○
Wafer	92.1 ± 1.2	99.1 ± 0.2 ○	**99.5 ± 0.2** ○	**99.5 ± 0.2** ○
WordsSynonyms	54.4 ± 5.8	63.7 ± 6.6 ○	65.3 ± 3.9 ○	**66.6 ± 5.3** ○
Yoga	55.0 ± 1.7	**87.7 ± 2.1** ○	86.4 ± 2.0 ○	86.7 ± 1.6 ○
Average	67.5	**79.2**	78.3	78.4

We used the 10-fold cross-validation protocol to evaluate this feature construction-based approach. Similarly to the case of INSIGHT and FastAWARD, the number of selected instances corresponds to 10% of the entire training data, however, as described previously, for the feature construction-based approach, we selected the instances from $\mathscr{D}_1^{train}$ (not from $\mathscr{D}^{train}$).

We tested several variants of the approach, for two out of them, the resulting accuracies are shown in the last two columns of Table 5.

The results shown in the fourth column of Table 5 (denoted as HubFeatures) refer to the case when we performed hub-based instance selection on $\mathscr{D}_1^{train}$ using good 1-occurrence score. The results shown in the last column of Table 5 (denoted as RndFeatures) refer to the case when the instances were randomly selected from $\mathscr{D}_1^{train}$.

Both HubFeatures and RndFeatures outperform FastAWARD in clear majority of the cases: while they are significantly better than FastAWARD for 23 and 21 data sets respectively, they are significantly worse only for one data set. INSIGHT, HubFeatures and RndFeatures can be considered as alternative approaches, as their overall performances are close to each other. Therefore, in a real-world application, one can use cross-validation to select the approach which best suits the particular application.

7 Conclusions and Outlook

We devoted this chapter to the recently observed phenomenon of hubness which characterizes numerous real-world data sets, especially high-dimensional ones. We surveyed hubness-aware classifiers and instance selection. Finally, we proposed a hubness-based feature construction approach. The approaches we reviewed were originally published in various research papers using slightly different notations and terminology. In this chapter, we presented all the approaches within an integrated framework using uniform notations and terminology. Hubness-aware classifiers were originally developed for vector classification. Here, we pointed out that these classifiers can be used for time-series classification given that an appropriate time-series distance measure is present. To the best of our knowledge, most of the surveyed approaches have not yet been used for time-series data. We performed extensive experimental evaluation of the state-of-the-art hubness-aware classifiers on a large number of time-series data sets. The results of our experiments provide direct indications for the application of hubness-aware classifiers for real-world time-series classification tasks. In particular, the HIKNN approach seems to have the best overall performance for time-series data.

Furthermore, we pointed out that instance selection can substantially speed-up time-series classification and the recently introduced hubness-aware instance selection approach, INSIGHT, outperforms the previous state-of-the-art instance selection approach, FastAWARD, which did not take the presence of hubs explicitly into account. Finally, we showed that the selected instances can be used to construct features for instances of time-series data sets. While mapping time-series into a vector space by this feature construction approach is intuitive and leads to acceptable overall classification accuracy, the particular instance selection approach does not seem to play a major role in the procedure.

Future work may target the implications of hubness for feature construction approaches and how these features suit conventional classifiers. One would for example expect that monotone classifiers [2, 13, 23], benefit from hubness-based feature construction: the closer an instance is to a good hub, the more likely it belongs to the same class. Furthermore, regression methods may also benefit from taking the presence of hubs into account: e.g. hw-kNN may simply be adapted for the case of nearest neighbor regression where the weighted average of the neighbors' class labels is taken instead of their weighted vote. Last but not least, due to the novelty

of hubness-aware classifiers, there are still many applications in context of which hubness-aware classifiers have not been exploited yet, see e.g. [47] for recognition tasks related to literary texts. Also the classification of medical data, such as diagnosis of cancer subtypes based on gene expression levels [31], could potentially benefit from hubness-aware classification, especially classifiers taking class-imbalance into account [51].

Acknowledgements Research partially performed within the framework of the grant of the Hungarian Scientific Research Fund (grant No. OTKA 108947). The position of Krisztian Buza is funded by the Warsaw Center of Mathematics and Computer Science (WCMCS).

References

1. Aha, D., Kibler, D., Albert, M.: Instance-Based Learning Algorithms. Machine Learning **6**(1), 37–66 (1991)
2. Altendorf, E., Resticar, A., Dietterich, T.: Learning from sparse data by exploiting monotonicity constraints. In: 21st Annual Conference on Uncertainty in Artificial Intelligence, pp. 18–26. AUAI Press, Arlington, Virginia, USA (2005)
3. Barabási, A.: Linked: How Everything Is Connected to Everything Else and What It Means for Business, Science, and Everyday Life. Plume (2003)
4. Bellman, R.E.: Adaptive control processes - A guided tour. Princeton University Press, Princeton, New Jersey, U.S.A. (1961)
5. Botsch, M.: Machine Learning Techniques for Time Series Classification. Cuvillier (2009)
6. Brighton, H., Mellish, C.: Advances in Instance Selection for Instance-Based Learning Algorithms. Data Mining and Knowledge Discovery **6**(2), 153–172 (2002)
7. Burges, C.J.: A tutorial on support vector machines for pattern recognition. Data mining and knowledge discovery **2**(2), 121–167 (1998)
8. Buza, K., Nanopoulos, A., Schmidt-Thieme, L.: INSIGHT: Efficient and Effective Instance Selection for Time-Series Classification. In: 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining, *Lecture Notes in Computer Science*, vol. 6635, pp. 149–160. Springer (2011)
9. Buza, K.A.: Fusion Methods for Time-Series Classification. Peter Lang Verlag (2011)
10. Chen, G.H., Nikolov, S., Shah, D.: A latent source model for nonparametric time series classification. In: Advances in Neural Information Processing Systems 26, pp. 1088–1096 (2013)
11. Cortes, C., Vapnik, V.: Support vector machine. Machine learning **20**(3), 273–297 (1995)
12. Devroye, L., Györfi, L., Lugosi, G.: A Probabilistic Theory of Pattern Recognition. Springer Verlag (1996)
13. Duivesteijn, W., Feelders, A.: Nearest neighbour classification with monotonicity constraints. In: Machine Learning and Knowledge Discovery in Databases, pp. 301–316. Springer (2008)
14. Eads, D., Hill, D., Davis, S., Perkins, S., Ma, J., Porter, R., Theiler, J.: Genetic Algorithms and Support Vector Machines for Time Series Classification. In: Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 4787, pp. 74–85 (2002)
15. Fix, E., Hodges, J.: Discriminatory analysis, nonparametric discrimination: consistency properties. Tech. rep., USAF School of Aviation Medicine, Randolph Field (1951)
16. Garcia, V., Mollineda, R.A., Sanchez, J.S.: On the k-nn performance in a challenging scenario of imbalance and overlapping. Pattern Anal. Appl. **11**, 269–280 (2008)
17. Geurts, P.: Pattern Extraction for Time Series Classification. In: Principles of Data Mining and Knowledge Discovery, pp. 115–127. Springer (2001)
18. Grabocka, J., Wistuba, M., Schmidt-Thieme, L.: Time-series classification through histograms of symbolic polynomials. arXiv preprint arXiv:1307.6365 (2013)

19. Grochowski, M., Jankowski, N.: Comparison of Instance Selection Algorithms II. Results and Comments. In: International Conferene on Artificial Intelligence and Soft Computing, *Lecture Notes in Computer Science*, vol. 3070, pp. 580–585. Springer-Verlag, Berlin/Heidelberg (2004)
20. Hand, D.J., Vinciotti, V.: Choosing k for two-class nearest neighbour classifiers with unbalanced classes. Pattern Recogn. Lett. **24**, 1555–1562 (2003)
21. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Transactions on knowledge and data engineering **21**(9), 1263–1284 (2009)
22. He, X., Zhang, J.: Why Do Hubs Tend to Be Essential in Protein Networks? PLoS Genet **2**(6) (2006)
23. Horváth, T., Vojtáš, P.: Ordinal classification with monotonicity constraints. Advances in Data Mining. Applications in Medicine, Web Mining, Marketing, Image and Signal Mining pp. 217–225 (2006)
24. Jankowski, N., Grochowski, M.: Comparison of Instance Selection Algorithms I. Algorithms Survey. In: International Conferene on Artificial Intelligence and Soft Computing, *Lecture Notes in Computer Science*, vol. 3070, pp. 598–603. Springer, Berlin/Heidelberg (2004)
25. Jolliffe, I.: Principal component analysis. Wiley Online Library (2005)
26. Kehagias, A., Petridis, V.: Predictive Modular Neural Networks for Time Series Classification. Neural Networks **10**(1), 31–49 (1997)
27. Keller, J.E., Gray, M.R., Givens, J.A.: A fuzzy k-nearest-neighbor algorithm. In: IEEE Transactions on Systems, Man and Cybernetics, pp. 580–585 (1985)
28. Keogh, E., Shelton, C., Moerchen, F.: Workshop and challenge on time series classification (2007). URL http://www.cs.ucr.edu/~eamonn/SIGKDD2007TimeSeries.html
29. Kim, S., Smyth, P.: Segmental hidden Markov models with random effects for waveform modeling. The Journal of Machine Learning Research **7**, 945–969 (2006)
30. Levenshtein, V.: Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. Soviet Physics Doklady **10**(8), 707–710 (1966)
31. Lin, W.J., Chen, J.J.: Class-imbalanced classifiers for high-dimensional data. Briefings in bioinformatics **14**(1), 13–26 (2013)
32. Liu, H., Motoda, H.: On issues of Instance Selection. Data Mining and Knowledge Discovery **6**(2), 115–130 (2002)
33. MacDonald, I., Zucchini, W.: Hidden Markov and Other Models for Discrete-Valued Time Series. Chapman & Hall London (1997)
34. Marcel, S., Millan, J.: Person Authentication using Brainwaves (EEG) and Maximum a Posteriori Model Adaptation. IEEE Transactions on Pattern Analysis and Machine Intelligence **29**, 743–752 (2007)
35. Martens, R., Claesen, L.: On-line Signature Verification by Dynamic Time-Warping. In: Proceedings of the 13th International Conference on Pattern Recognition, vol. 3, pp. 38–42. IEEE (1996)
36. Marussy, K., Buza, K.: Success: A new approach for semi-supervised classification of time-series. In: L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. Zadeh, J. Zurada (eds.) Artificial Intelligence and Soft Computing, *Lecture Notes in Computer Science*, vol. 7894, pp. 437–447. Springer Berlin Heidelberg (2013)
37. Niels, R.: Dynamic Time Warping: an Intuitive Way of Handwriting Recognition? Master Thesis, Radboud University Nijmegen, The Netherlands (2004)
38. Petridis, V., Kehagias, A.: Predictive modular neural networks: applications to time series, *The Springer International Series in Engineering and Computer Science*, vol. 466. Springer Netherlands (1998)
39. Rabiner, L., Juang, B.: An Introduction to Hidden Markov Models. ASSP Magazine **3**(1), 4–16 (1986)
40. Radovanović, M.: Representations and Metrics in High-Dimensional Data Mining. Izdavačka knjižarnica Zorana Stojanovića, Novi Sad, Serbia (2011)
41. Radovanović, M., Nanopoulos, A., Ivanović, M.: Nearest Neighbors in High-Dimensional Data: The Emergence and Influence of Hubs. In: Proceedings of the 26rd International Conference on Machine Learning (ICML), pp. 865–872. ACM (2009)

42. Radovanović, M., Nanopoulos, A., Ivanović, M.: Hubs in Space: Popular Nearest Neighbors in High-Dimensional Data. The Journal of Machine Learning Research (JMLR) **11**, 2487–2531 (2010)
43. Radovanović, M., Nanopoulos, A., Ivanović, M.: Time-Series Classification in Many Intrinsic Dimensions. In: Proceedings of the 10th SIAM International Conference on Data Mining (SDM), pp. 677–688 (2010)
44. Rish, I.: An empirical study of the naive Bayes classifier. In: Proc. IJCAI Workshop on Empirical Methods in Artificial Intelligence (2001)
45. Sakoe, H., Chiba, S.: Dynamic Programming Algorithm Optimization for Spoken Word Recognition. Acoustics, Speech and Signal Processing **26**(1), 43–49 (1978)
46. Schedl M., F.A.: A mirex meta-analysis of hubness in audio music similarity. In: Proceedings of the 13th International Society for Music Information Retrieval Conference, ISMIR'12 (2012)
47. Stańczyk, U.: Recognition of author gender for literary texts. In: Man-Machine Interactions 2, pp. 229–238. Springer (2011)
48. Sykacek, P., Roberts, S.: Bayesian Time Series Classification. Advances in Neural Information Processing Systems **2**, 937–944 (2002)
49. Tomašev, N.: The Role of Hubness in High-Dimensional Data Analysis. Jožef Stefan International Postgraduate School (2013)
50. Tomašev, N., Mladenić, D.: Nearest neighbor voting in high dimensional data: Learning from past occurrences. Computer Science and Information Systems **9**, 691–712 (2012)
51. Tomašev, N., Mladenić, D.: Class imbalance and the curse of minority hubs. Knowledge-Based Systems **53**, 157–172 (2013)
52. Tomašev, N., Mladenić, D.: Hub co-occurrence modeling for robust high-dimensional knn classification. In: Proceedings of the ECML/PKDD Conference. Springer (2013)
53. Tomašev, N., Radovanović, M., Mladenić, D., Ivanović, M.: Hubness-based fuzzy measures for high-dimensional k-nearest neighbor classification. International Journal of Machine Learning and Cybernetics (2013)
54. Tomašev, N., Radovanović, M., Mladenić, D., Ivanović, M.: The role of hubness in clustering high-dimensional data. IEEE Transactions on Knowledge and Data Engineering **99**(PrePrints), 1 (2013)
55. Tomašev, N., Radovanović, M., Mladenić, D., Ivanovicć, M.: A probabilistic approach to nearest neighbor classification: Naive hubness Bayesian k-nearest neighbor. In: Proceeding of the CIKM conference (2011)
56. Wang, J., Neskovic, P., Cooper, L.N.: Improving nearest neighbor rule with a simple adaptive distance measure. Pattern Recogn. Lett. **28**, 207–213 (2007)
57. Xi, X., Keogh, E., Shelton, C., Wei, L., Ratanamahatana, C.: Fast Time Series Classification Using Numerosity Reduction. In: Proceedings of the 23rd International Conference on Machine Learning (ICML), pp. 1033–1040. ACM (2006)