Large-width machine learning algorithm

Martin Anthony Joel Ratsaby

Abstract We introduce an algorithm, called Large Width (LW), that produces a multi-category classifier (defined on a distance space) with the property that the classifier has a large 'sample width' (width is a notion similar to classification margin). LW is an incremental instance-based (also know as 'lazy') learning algorithm. Given a sample of labeled and unlabeled examples it iteratively picks the next unlabeled example and classifies it while maintaining a large distance between each labeled example and its nearest unlike-prototype (a prototype is either a labeled example or an unlabeled example which has already been classified). Thus LW gives a higher priority to unlabeled points whose classification decision 'interferes' less with the labeled sample. On a collection UCI benchmark datasets, the LW algorithm ranks at the top when compared to 11 instance-based learning algorithms (or configurations). When compared to the best candidate from instance-based learners, MLP, SVM, decision-tree learner (C4.5) and Naive Bayes, LW is ranked at second place after only MLP which comes at first place by a single extra win against LW. The LW algorithm can be implemented in parallel distributed processing to yield a high speedup-factor and is suitable for any distance space, with a distance function which need not necessarily satisfy the conditions of a metric.

Keywords Large margin learning · k-Nearest neighbor · Lazy-learning · Non-parametric classification

1 Introduction

1.1 Overview

This paper is the first applied work in a programme of research we have been conducting on learning classifiers that take into account the concept of 'width' (it is a revised version of an unpublished preprint). We consider multi-category classification problems on distance spaces where by multi-category classifier we mean a mapping from some input space $\mathcal{X}$ to an image set $\mathcal{Y}$ whose cardinality is at least 2 (the elements of $\mathcal{Y}$ are referred to as class categories). A distance space is any set equipped with a distance function which, in contrast to a metric, does not need to satisfy the triangle inequality. We introduce an algorithm that produces a multi-category classifier and which is motivated by the aim of maintaining a large 'sample width' (the precise notion of width will be defined further below, but for the interim, one may think of width as a measure of how confident a classifier's decision is for a given input). We implemented this algorithm in software and report the results of machine learning experiments in which we compare its accuracy against several other algorithms on several standard classification problems. Compared to other large-margin learning algorithms [18] such as the ubiquitous kernel methods, where a kernel function needs to be defined and be an inner product in some higher dimensional space, the LW algorithm distance function does not have to satisfy anything other than perhaps non-negativity. This makes LW easy to apply in classification learning problems on non-standard domains [21] with no need for kernel functions nor for distance functions that satisfy the metric axioms.

There are learning domains where it is difficult to formalize quantitative features encoded as numerical variables in order to discriminate between objects from different class categories [27]. Featureless qualitative dissimilarity information can be represented mathematically by defining a distance function over the input space. For instance, objects may be represented by strings of symbols which can be compared using a dissimilarity (or distance) function which typically does not satisfy all of the metric axioms. For instance, the normalized information distance [24] which is based on Kolmogorov complexity of a string-based representation of objects is a non-metric distance. This has been successfully applied to classification and data clustering over diverse problem domains [15]. Another example of a non-metric is the universal distance introduced in [13, 14] and applied to image classification which is based on a data-compression algorithm of Lempel-Ziv. A main advantage of these distances is the fact that they can be applied to a pair of string-based representation of objects with variable lengths, that is, the lengths of the strings that describe the two objects whose distance is being measured do not need to be equal.

This non-parametric approach of learning by a possibly non-metric distance is much more flexible than ones based on numerical features. There are many examples of distance functions that do not satisfy all the metric axioms [19] and new ones can be defined easily for any kind of objects—for instance, for bioinformatic sequences, graphs, images, etc.. Most learning algorithms, in particular neural networks which have been very successful recently, require a Euclidean or more generally a vector space, represented by numerical-valued features and hence cannot be applied to such non-metric input spaces. In contrast, the LW algorithm introduced in this paper can learn over non-metric spaces and can be used to learn from featureless data using any distance function, which need not satisfy any of the metric axioms.

1.2 Width

We start by giving an intuitive indication, in general terms, of what it means for a binary classifier to have a large sample width. (The paper deals with multi-category classification more generally, but the case of two classes is easier to visualise.) Imagine we have a sample of points in the distance space, each of which has been given a classification label 1 or -1. Suppose we have a classifier that has classified some points in the space (not necessarily all of them) as belonging to class 1 or -1. Let S_+ be the set of points labeled 1 and S_- the set of points labeled -1. If x is a sample point labeled 1, then we define the classification width at x to be the distance $d(x, S_-)$, the minimum (or infimum, in the case in which S_- is infinite) of the distances between x and points in S_-. We make a similar definition if x is labeled -1; then, we use $d(x, S_+)$. The sample width is defined to be the minimum, over the sample points, of these individual widths.

The concept of width was introduced by [3] and applied there to binary functions on the real line. The motivation was to develop a notion of 'definitive' classification that was analogous to the well known large 'margin' idea (see, for instance [2, 30]). In [5] this was generalized to binary functions over the multi-dimensional cube $[0,1]^n$ where S_- and S_+ are union of boxes labeled -1 and 1. The paper [4] applied the idea to multi-category classification on $[0,1]^n$ where, for category l, the points with classification l formed a set S_l that is a union of boxes. In [6] this was generalized to learning binary classifiers on any finite metric input space, where S_+ and S_- formed any partition of the space. This was extended to multi-category classification on (possibly infinite) metric spaces in [8]; and this work was applied to case-based inference in [7]. In [10] we applied the concept of width to learning binary classification using 'half-spaces' over finite distance spaces; and in [11] we extended this to learning 'nearest-prototype' binary classifiers. The theoretical bounds obtained on the misclassification probability of the classifiers considered in these papers improve as the sample width increases. This motivates the use of classifiers with a large width.

1.3 Setup and definitions

Let the distance space be denoted $\mathcal{X}$ (which may be an infinite space) and $d(x, x')$ denotes the distance between points x and x'. A labeled *sample*, $\xi = \{z_i\}_{i=1}^m$ with $z_i := (x_i, y_i)$, is a sequence of points of $\mathcal{X}$ together with labels in a set $\mathcal{Y} = \{1, \ldots, M\}$ (for some fixed integer M). We call such a labeled point $z = (x, y)$ an *example*; and we denote by $x(z)$ and $y(z)$ the x and y components of z. We will slightly abuse the notation and for any two points, z, z' in $\mathcal{X} \times \mathcal{Y}$ we also write $d(z, z')$ to mean $d(x(z), x(z'))$.

In the current paper we consider an algorithm for producing multi-category classifiers $h : \mathcal{X} \to \mathcal{Y}$ on the basis of a labeled sample ξ. The algorithm is an instance-based learning algorithm [25] in the sense that it is given a finite set of unlabeled points in $\mathcal{X}$ to be classified and it uses only the sample to classify them. We denote by U_1 the initial set of unlabeled points which are to be classified by the algorithm. It classifies this set in an incremental way, such that at each time step t the set U_t of points which are still unlabeled (and unclassified) is smaller than the previous set U_{t-1} by one. At any stage — let us say stage t — of the sequential process, there will be a set of points already classified: those in the sample ξ, together with a set L_t of those not in the sample that the algorithm has already classified with a sufficient level of confidence (this will be detailed when we state the pseudo-code). We will call the set of points, $\xi \cup L_t$, that have been labeled *prototypes*. (We are abusing notation slightly: here, we should write $\{x(z) : z \in \xi\}$ rather than simply ξ.) For any prototype p, let $NUN(p)$, the nearest unlike neighbor prototype to p, denote the closest prototype to p which is of a different label, meaning $y\,(NUN(p)) \neq y(p)$. The classification (at this stage) of any unlabeled point x is based on the following rule: let the *NUN-ball* centered at a labeled point z, denoted by $B_{NUN}(z)$, be the set of all points z' (not just labeled ones) such that $d(z, z') \le d(z, NUN(z))$. For $k \in \mathcal{Y}$ let

$$v_k(x) := \sum_{i:y(z_i)=k} \mathbb{I}\left\{x \in B_{NUN}(z_i)\right\}$$

be the number of labeled examples that are labeled k whose NUN-balls contain x. (Here $\mathbb{I}\{x \in S\}$ is the indicator function of the set S.) Note that the sum is over the examples in ξ only: it is not over all prototypes (ξ together with some other currently labeled points). To be clear: the radii of the NUN-balls depend on all the prototypes, but this sum is over only the examples. Then the classification given to x (where h denotes the current 'hypothesis' of the algorithm) is

$$h(x) := \mathrm{argmax}_{k \in \mathcal{Y}} v_k(x); \tag{1}$$

that is, x is given the label of the class with the largest number of NUN-balls containing x. (If there is no single category which maximises $v_k(x)$, then the algorithm LW uses a slightly different rule, described further below.) Once x is classified, if it is classified with sufficient confidence then it becomes a prototype, and in this way the algorithm adapts. It works sequentially in this manner through the points of U_1 (chosen in a particular order).

The motivation for the above definition (1) lies with the notion of width. Let S_k be the set of prototypes labeled k. We use the notion of width from [8]. The *width* of h at x is defined as

$$w_h(x) := \inf_{k \neq h(x)} d\,(x, S_k)\,.$$

For a labeled point $z = (x, y)$, we also write $w_h(z)$ to mean $w_h(x(z))$. The width $w_h(z)$ of h at an example z is the distance to the nearest prototype of opposite label. For a sample ξ of examples z, we define the *sample width* of h as

$$w_\xi(h) := \min_{z \in \xi} w_h(z). \tag{2}$$

The sample width of h is always non-negative.

For a classifier $h : \mathcal{X} \to \mathcal{Y}$, let $\mathrm{er}_P(h)$ be the probability that h misclassifies an example $(X, Y) \in \mathcal{X} \times \mathcal{Y}$ which is drawn randomly according to a probability distribution P on $\mathcal{X} \times \mathcal{Y}$. Denote by $\mathcal{N}(\mathcal{X}, \gamma, d)$ the γ-covering number of $\mathcal{X}$ with respect to the metric d. The following result (which follows from Theorem 5.1 of [8]) states a theoretical bound on the misclassification probability for any classifier h.

Theorem 11 *Suppose that $\mathcal{X}$ is a metric space of diameter $\mathrm{diam}(\mathcal{X})$ and denote by $M := |\mathcal{Y}|$. Suppose P is any probability measure on $Z = \mathcal{X} \times \mathcal{Y}$. Let $\delta \in (0,1)$. Then, with probability at least $1 - \delta$, the following holds for $\xi \in Z^m$: for any function $h : \mathcal{X} \to \mathcal{Y}$ and for any $\gamma \in (0, \mathrm{diam}(\mathcal{X})]$, if $w_\xi(h) > \gamma$ then*

$$\mathrm{er}_P(h) \le \frac{2}{m}\left(M\mathcal{N}\log_2\left(\frac{36\,\mathrm{diam}(\mathcal{X})}{\gamma}\right) + \log_2\left(\frac{4\,\mathrm{diam}(\mathcal{X})}{\delta\gamma}\right)\right) \tag{3}$$

where $\mathcal{N} = \mathcal{N}(\mathcal{X}, \gamma/12, d)$ is the covering number of the metric space at scale $\gamma/12$.

The Large Width (LW) algorithm, when given a sample ξ, produces an h that classifies all points of U_1, while not misclassifying any of the examples in the sample ξ. This means that LW produces a classifier h with a strictly positive sample width $w_\xi(h)$. Therefore, for any γ such that $w_\xi(h) > \gamma$ the right side of (4) bounds its error from above. From the dependence on γ, the bound decreases as the sample width $w_\xi(h)$ increases hence the algorithm aims to produce an h that has a large sample width $w_\xi(h)$ because this reduces the upper bound on its generalization error.

In the next section we describe the LW algorithm whose behavior is motivated by this aim; and which selects points to be classified in an order which is intended to keep the sample width large.

2 The LW algorithm

In the previous section we indicated that the aim of the algorithm is to find a classifier that has a large sample width. LW is an instance-based supervised learning algorithm, as for example is the nearest-neighbor algorithm. It relies on a sample of labeled examples for making a classification decision about any unlabeled point in U_1. We called the union of the sample ξ and the set of already-classified points (with sufficient confidence) at time t the set L_t of *prototypes.*This is the union $\bigcup_{k\in\mathcal{Y}} S_k^{(t)}$, where $S_k^{(t)}$ is the set of points in $L_t \cup \xi$ that are labelled with category k. Given this set, we then have a classifier h_t at time t which can be used to label any one of the remaining unlabeled points at time $t+1$. That classifier depends only on the sets $S_k^{(t)}$.

As mentioned in the previous section, LW aims to produce a classifier that has a large sample width $w_\xi(h)$; hence we want the algorithm to classify the next unlabeled point while maintaining a large distance between each labeled example and its nearest differently-labeled prototype.

To achieve this, given a set of remaining unlabeled points to classify, it chooses in a particular way the next point among those to be classified. LW gives a higher priority to unlabeled points p whose classification decision 'interferes' less with the sample. By this, we mean that it prioritises points p with the property that fewer examples z in the sample incur a decrease in width, meaning $w_{h_{t+1}}(z) < w_{h_t}(z)$, as the algorithm continues to adapt the classifier from h_t to h_{t+1} as a result of choosing to classify p at time t. (Recall from above that once the classifier classifies an unlabeled point with sufficient confidence it becomes a prototype and hence one of the prototype sets $S_k^{(t)}$ grows in cardinality by one and the classifier changes from h_t to h_{t+1}.)

There are two main issues that LW needs to determine: the first is which unlabeled point at time t should be classified next, and the second is what classification value to assign it. To explain how LW does this we now describe the algorithm by splitting it into parts. The main part is Algorithm 1 which calls Procedures 2, 3.

Before proceeding, let us mention that by a 'shuffle' of a set, we mean choosing a random ordering of the indices of the elements of the set. We do this with members of a set from which we need to choose an element according to some criterion which can have a repeated value across some members (ties in the criterion value). Shuffling ensures that in this case we do not always choose the same member, for instance, the one with the lowest index.

We now describe the algorithm formally. Denote by ξ a finite labeled sample and denote by $t \geq 1$ an integer that specifies the time step. The algorithm advances one step at a time, where in each step it classifies one unlabeled point. Let

$$U_t \subseteq \mathcal{X}$$

be the set of *unlabeled points*, p, that remain to be classified, and let U_1 be the starting set of unlabeled points. As mentioned above, to simplify the notation, for an example $z \in \xi$ and an unlabeled point $p \in U_t$ we write $d(z,p)$ to mean $d(x(z),p)$.

Denote by L_t the set of *non-example points* at time t which have already been classified with a sufficient level of confidence at some time previous to t. Thus the set $\bigcup_{k\in\mathcal{Y}} S_k^{(t)}$ of *prototypes* at time t equals the union $L_t \bigcup \xi$ of all points at time t which are either examples or have been confidently classified.

We now define more precisely the notion of nearest-unlike neighbor prototype to an example in the sample. For $z = (x,y) \in \xi$, let the *nearest-unlike prototype* of z be

$$NUN_t(z) = \text{argmin}_{\{p\in L_t\cup\xi:\, y(p)\neq y(z)\}} d(x(p), x(z)). \quad (4)$$

It is either a labeled point in L_t or a labeled example in ξ which is closest to $x(z)$ and whose label differs from $y(z)$. For an unlabeled point p and any $k \in \mathcal{Y}$, define the *vote-set* $V_k(p) \subseteq \xi$ to be the following subset of the sample ξ:

$$V_k(p) := \{z \in \xi : \, y(z) = k, \, d(p,z) < d(NUN(z), z)\}, \, k \in \mathcal{Y}.$$

This is the set of examples in the sample ξ of category k whose NUN-balls contain p.

The main part of the algorithm in presented next in Algorithm 1.

Algorithm 1: Algorithm LW

```
 1: Let $t = 1$ and the initial set of unlabeled points is $U_1$.
 2: while $|U_t| > 0$ do
 3:   for all points $p \in U_t$ do
 4:     $V_k(p) = \emptyset$, $1 \leq k \leq M$ // initalize Vote-sets
 5:   end for
 6:   for all examples $z \in \xi$ do
 7:     Calculate NUN-prototype $NUN(z) \in \xi \bigcup L_t$
 8:     Let $d_z := \mathrm{d}(z, NUN(z))$
 9:   end for
10:   Shuffle $\xi$ // Choose a random ordering for $\xi$
11:   for all examples $z := (x,y) \in \xi$ do
12:     for all points $p \in U_t$ do
13:       if $\mathrm{d}(z,p) < d_z$ then
14:         $V_y(p) := V_y(p) \bigcup z$, // $z$ votes for $p$ to be
            classified $y$
15:       end if
16:     end for
17:   end for
18:   $p_t^* =$ findBestPoint($U_t$) // Call procedure that
      returns the best point to classify
19:   Classify($p_t^*$) // Call procedure that classifies this
      point
20:   $t := t + 1$
21: end while
```

Algorithm LW allows to have a positive real value weight associated with each example $z \in \xi$. One way in which this is useful is for placing a larger emphasis on certain examples in the training set ξ which in effect changes the level of influence that an example has on learning. In particular, this can be used for balancing the class categories (as described in Section 3). The *weight* of z is denoted by $\mathsf{w}(z)$. For a set $A \subset \xi$ let the *weighted cardinality* of A be defined as

$$|A|_{\mathsf{w}} := \sum_{z\in A} \mathsf{w}(z).$$

The choice of value of the weight of each example in ξ is left for the discretion of the user of LW. If it is 1 for every example in ξ then the weighted cardinality of A becomes the standard set cardinality $|A|$.

Procedure `findBestPoint(`U_t`)` is described in Procedure 2. It selects the best unlabeled point to classify at time t. It operates according to a score function. If the score is strictly positive for at least one point in the set U_t then it selects a point with the highest score. If all points in U_t have a score value of zero then it selects a point with the highest weighted cardinality vote-set, if such exists. Otherwise, it selects a point x from U_t whose distance from the nearest NUN-sphere is minimum when considering all points in U_t and their distances from their corresponding nearest NUN-spheres (a NUN-sphere is the set of points whose distance from a prototype equals the distance between that prototype and its nearest unlike prototype). Note that the distance between a point x from a NUN-sphere centered at a prototype p equals $d(x,p) - d(p, NUN(p))$.

Procedure 2: `findBestPoint(`U_t`)`

```
1:  Shuffle $U_t$
2:  $p_t^* := \text{argmax}_{p\in U_t} Score(p)$ // Score defined in Function 4
3:  if $Score(p_t^*) > 0$ then
4:      return $p_t^*$
5:  else
6:      // All points $p \in U_t$ have $Score(p) = 0$
7:      Let $a(p) := \max_{1\le k\le M} |V_k(p)|_{\mathsf{w}}$
8:      if $\max_{p\in U_t} a(p) > 0$ then
9:          return $p_t^* := \text{argmax}_{p\in U_t} a(p)$
10:     else
11:         // All points $p \in U_t$ have empty vote-sets
12:         Choose $p_t^*$ from $U_t$ whose distance to its nearest NUN-sphere (centered at a labeled example) is minimum, considering all points in $U_t$.
13:         return $p_t^*$
14:     end if
15: end if
```

Procedure `Classify(`p`)` is described in Procedure 3 and is in charge of classifying a given unlabeled point p. Let us denote by k^* and k^{**} the labels of the maximum weighted cardinality vote-set and second-highest weighted cardinality vote-set of p, respectively.(These cardinalities could be the same when there are different categories with the same weighted cardinality vote-sets with respect to p.) At the current version of the algorithm, we are using $Score(p)$ (described as Function 4) which defines a score for an unlabeled point p as

$$Score(p) := a(p)\,(a(p) - b(p)) \tag{5}$$

where

$$a(p) := \max_{1\le k\le M} |V_k(p)|_{\mathsf{w}}$$

is the value of the maximum weighted cardinality vote-set and

$$b(p) := \max_{1\le k\neq k^*\le M} |V_k(p)|_{\mathsf{w}}$$

is the value of the second-highest weighted cardinality vote-set. With this score function, a point p ranks high if *either* of the following quantities is *high*:

(1) *Weighted cardinality of the set of examples that will suffer a reduction in width if the classification decision is not k^**: it is the weighted cardinality of the set of examples z in $V_{k^*}(p)$. All examples $z \in V_{k^*}(p)$ will suffer a reduction in the value of the width $w_h(z)$ if p gets classified by a label which differs from their label; that is, if p gets classified by label $y \neq k^*$. This is represented by the factor $a(p)$ in (5).

(2) *Confidence of classification*: there is (possibly) a higher confidence for classifying p by $y = k^*$, because the next best classification choice k^{**} is supported by a *smaller* weighted cardinality set of examples. This is represented by the factor $a(p) - b(p)$ in (5).

Obviously there are other possible score definitions that can replace (5); for instance, $a(p)\ln\frac{a(p)}{b(p)}$ but we have not tried them.

Procedure 3: `Classify(p)`

```
1:  Let $p$ be a given point
2:  Let $|V_{k^*}(p)|_{\mathsf{w}} := \max_{1\le k\le M} |V_k(p)|_{\mathsf{w}}$ // maximum cardinality vote-set
3:  Let $|V_{k^{**}}(p)|_{\mathsf{w}} := \max_{1\le k\neq k^*\le M} |V_k(p)|_{\mathsf{w}}$ // second highest cardinality vote-set or another maximum cardinality vote-set
4:  if $|V_{k^*}(p)|_{\mathsf{w}} > |V_{k^{**}}(p)|_{\mathsf{w}}$ then
5:      // The level of confidence in classifying $p$ is sufficiently high. In this case, even if $|V_{k^{**}}(p)|_{\mathsf{w}}$ equals zero, $|V_{k^*}(p)|_{\mathsf{w}}$ is larger than zero
6:      Classify $p$ with label $y = k^*$
7:      $makeItPrototype := true$
8:  else
9:      // The level of confidence is not high.
10:     if $|V_{k^*}(p)|_{\mathsf{w}} = |V_{k^{**}}(p)|_{\mathsf{w}} > 0$ then
11:         Let $MAX(p) := \{k : |V_k(p)|_{\mathsf{w}} = |V_{k^*}(p)|_{\mathsf{w}}\}$
12:         Let $A(p) := \bigcup_{k\in MAX(p)} V_k(p)$
13:         $NN(p) := \text{argmin}_{z\in A(p)} \text{d}(z,p)$, // Find the nearest neighbor example $NN(p) \in A(p)$
14:         Classify $p$ with the label $y$ of example $NN(p)$
15:     else
16:         // All vote-sets for $p$ are of cardinality zero so find the nearest NUN-sphere to $p$ which is centered at a labeled example.
17:         $z^* := \text{argmin}_{z\in\xi} (\text{d}(z,p) - d(z, NUN(z)))$
18:         Classify $p$ with the label $y$ of $z^*$
19:     end if
20:     $makeItPrototype := false$
21: end if
22: if $makeItPrototype$ then
23:     $L_t := L_t \bigcup \{p\}$ // Add $p$ to the set of prototypes
24: end if
25: $U_t := U_t \setminus \{p\}$
```

Note that when there is more than one category with maximum weighted cardinality vote-set for p, a nearest-neighbor approach is taken on the set of all examples in the union of all such vote-sets (Lines 8–11 in Procedure 3.) And when the vote-sets of the point p to be classified are all empty we find the closest NUN-sphere to p which is centered at a labeled example and classify p by the label of this example.

Function 4: *Score*(p)

```
1: Let $p$ be a given point
2: Let $|V_{k^*}(p)|_{\mathsf{w}} := \max_{1 \le k \le M} |V_k(p)|_{\mathsf{w}}$
3: Let $|V_{k^{**}}(p)|_{\mathsf{w}} := \max_{1 \le k \ne k^* \le M} |V_k(p)|_{\mathsf{w}}$ // Second
   highest (could equal the maximum)
4: return $Score(p) := |V_{k^*}(p)|_{\mathsf{w}} (|V_{k^*}(p)|_{\mathsf{w}} - |V_{k^{**}}(p)|_{\mathsf{w}})$
```

Let us analyze the time complexity. Denote by

$$n := |U_1|$$

the initial cardinality of the set of unlabeled points at time $t = 1$. Function 4 has a time complexity of $O(M)$, Procedure 3 is $O(m + M)$, Procedure 2 has time complexity $O(n)$. The double loop at lines 11-12 of Algorithm 1 has a time complexity of $O(nm)$ and therefore the algorithm has $O(n^2m)$ time complexity. Initial investigation in [28] reveals a parallel version of the LW algorithm with a reduced time complexity of $O(n(\log m + \log M))$. We note that the LW algorithm was first introduced and described in its serial form in a preprint of the current paper and preceded the paper [28].

3 Experiments

We used the WEKA toolset [22] which includes many machine learning algorithms and accuracy analysis tools with which we test and compare LW versus several standard algorithms. Algorithm LW relies on a distance function. While the LW algorithm can be used with any type of data provided that a distance function is defined, in the current paper we limit the experiments to datasets in a Euclidean space and using the l_2-metric which is obviously also a distance function. We chose datasets that have only numeric attributes (or nominal attributes which are transformed to numeric using WEKA's filters). In practice, given a learning classification problem that has also nominal attributes, one needs to choose a distance function that measures dissimilarity between nominal values in an appropriate manner, using knowledge about the attributes (as for instance, in measuring distance between two text documents, where the well known TFIDF formula gives a numeric representation to a nominal attribute that indicates if a particular word appears in a document).

The aim of the paper is not to test LW on any particular dataset but to compare it to other standard algorithms on common datasets. For this purpose, we set up machine learning experiments based on several datasets (Table 1) from the UCI repository [29]. (In Table 1, the input dimension does not include the target class attribute and M denotes the number of class categories.)

Since the Euclidean distance can only be applied to numeric data we used the WEKA filter NominalToBinary to transform the nominal attributes into binary attributes that encode the nominal values This increased the dimension from 20 to 62 in the credit dataset, 17 to 116 in the zoo dataset, 9 to 48 in the breast cancer dataset, 20 to 100 in the autism dataset. On all the datasets we normalized the data (using Normalize filter) and we used ReplaceMissingValues filter to substitute a value for missing values in the breast cancer dataset, the voting records dataset, and the autism dataset.

On all of the experiments that we describe below we applied WEKA's ClassBalancer filter which assigns each case in a dataset a weight (a positive value which may also be greater than 1) such that the sum of the weights of all cases in any class category is the same and the total weight of the dataset remains unchanged and is equal to the total number of cases. This class balancing improved the accuracy for all the algorithms that we considered and hence makes the competition more challenging for LW. To apply class balancing we used WEKA's FilteredClassifier which acts as a meta-learner that trains each of the algorithms using cross validation in such a way that the class balancing is done solely on the training part of each of the cross validation folds (this way the testing part remains with the original empirical distribution unchanged and hence without bias). On each dataset, every algorithm is tested on 10 runs, each run has 10-fold cross-validation and the average accuracy over all runs is computed. (The algorithms' parameter values are detailed in the Appendix.) If the class attribute is nominal, WEKA stratifies the data for each of the cross-validation splits.

The LW Algorithm was implemented in C and appears as a native library (using Java Native Interface) to the WEKA toolbox which is written in Java [23]. LW runs at approximately the speed of a single-hidden layer MLP with WEKA's default architecture and parameter specification (mentioned further below) and takes approximately the same time as single-hidden layer MLP to do ten repetitions of 10-fold cross validation on all the datasets. For instance, on a MacBook (Intel Core m3 Processor, 1.1 GHz with 8 Gb RAM) it takes approximately one minute to run 10-fold cross validation on the diabetes dataset with LW.

Being that LW produces a classifier which belongs to the family of instance based classifiers (also known as 'lazy' learning [25]) our primary aim is to compare it against other algorithms of this type. For this purpose we chose to compare LW to all the lazy-learning algorithms in the WEKA toolset (we are using version 3.8.4). This includes the nearest-neighbor algorithm [17], the k^* algorithm [16], the LWL algorithm [12] and the k-nearest neighbor algorithm [1] with k chosen automatically by cross validation and with either no distance-weighting, distance-weighting according to 1/distance or weighting of 1−distance [20]. (Distance weighting is a method by which a neighbor influences the classification decision of a point in a way that is proportional to its weight.) The list of lazy-learning algorithms is as follows,

(1) LW
(2) IBk, Nearest neighbor algorithm $k = 1$, no distance weighting
(3) IBk, Nearest neighbor algorithm $k = 1$, weight 1/distance
(4) IBk, Nearest neighbor algorithm $k = 1$, weight 1−distance
(5) IBk, Nearest neighbor algorithm, $1 \le k \le 5$, choose k based on cross-validation, no distance weighting
(6) IBk, Nearest neighbor algorithm, $1 \le k \le 5$, choose k based on cross-validation, weight 1/distance
(7) IBk, Nearest neighbor algorithm, $1 \le k \le 5$, choose k based on cross-validation, weight 1−distance
(8) IBk, Nearest neighbor algorithm, $1 \le k \le 10$, choose k based on cross-validation, no distance weighting
(9) IBk, Nearest neighbor algorithm, $1 \le k \le 10$, choose k based on cross-validation, weight 1/distance
(10) IBk, Nearest neighbor algorithm, $1 \le k \le 10$, choose k based on cross-validation, weight 1−distance
(11) k^* algorithm
(12) LWL algorithm

Dataset	M	dimension	sample size
zoo	8	116	101
LSVT voice	2	310	126
iris	3	4	150
glass	7	10	214
breast cancer	2	48	286
ionsphere	2	34	351
voting records	2	16	435
autism adult	2	100	704
absentism work	3	20	740
diabetes	2	8	768
credit	2	62	768
sports	2	59	1000
website phishing	3	23	1353
seismic bumps	2	27	2584

Table 1: Datasets used in the experiments (M denotes the number of class categories)

Table 2 displays the learning accuracy of LW and these lazy-learning algorithms. As can be seen from Table 2, there are $12 \times 14 = 168$ learning trials where for each we average the accuracy over 10 runs of 10-fold cross validation. Symbol $\circ$ or $\bullet$ in column i indicates that algorithm i over-performed or underperformed, respectively, relative to algorithm LW (denoted by (1)) by a statistically significant amount. Statistical significance refers to the result of a pairwise comparison of algorithm LW and any of the other algorithms (labeled (2) – (12)) using the corrected resampled T-Test [26]. Table 2 shows that there are 41 trials which resulted in an algorithm that underperforms compared to LW while there are only two in which an algorithm over-performs compared to LW.

In order to rank all the competing algorithms, we follow [22] and define by a 'win' (or 'loss') the event that an algorithm's accuracy is higher (or lower) than another algorithm's accuracy (where the comparison is made according to the T-test mentioned above). We take the difference between the number of wins and number of losses as a measure of success of an algorithm.

Table 3 shows that LW is ranked first amongst these instance-based algorithms. In particular, it is noteworthy to mention that the LW decision rule which is based on vote-sets (described in Procedures 2 and 3) outperforms the standard k-majority vote of the k-NN algorithm even with k chosen in a data-dependent manner and optimized via cross validation. The one which is ranked second to the best is k-NN with $1 \leq k \leq 5$ where k chosen based on cross-validation, with weight 1−distance (labeled (7)). We use it as the best lazy-learning candidate against LW in an experiment which is described further below.

We lined up the best candidates from the instance-based learning algorithms, the multilayered perceptron neural networks (MLP) and the support vector machines (SVM), all against LW. As mentioned above, we apply class balancing for all algorithms using WEKA's meta-learner with FilteredClassifier. To obtain the best MLP candidates we tested several different MLP architectures on the above datasets,

(1) single hidden layer,
(2) two hidden layers,
(3) three hidden layers.

We used WEKA's default parameter settings which include the number of activation units being equal to (number of attributes + number of class categories)/2, sigmoid activation function, back-propagation learning rate of 0.3, momentum value of 0.2, and number of training epochs is 500. Table 4 shows the ranking of these three architectures. The single-layered architectures (labeled (1)) obtains the highest number of wins and is used further below as a best MLP candidate against LW. To obtain the best SVM candidate we tested the following kernels,

(1) polynomial of first degree
(2) polynomial of second degree
(3) polynomial of third degree
(4) Radial Basis Function (RBF)
(5) Pearson universal kernel (Puk)

on the above datasets. Table 5 shows the ranking of these five SVM kernels. The polynomial of first degree obtains the highest number of wins and is used further below as a best SVM candidate against LW.

Next, we ran LW against the above best candidates from the lazy algorithms, the MLP and SVM, and two additional learning algorithms, C4.5 decision trees learning algorithm (denoted as J48 in WEKA) and Naive Bayes. Table 6 shows the accuracy results. Table 7 shows that MLP is in first place and LW is ranked at second place. The third column in Table 6 shows that MLP has three wins and two losses against LW. The remaining columns show that none of the other algorithms had more wins than losses over LW.

In the above experiment, in order to compare LW against this variety of algorithms we ensured that the datasets consist of numeric values (some of which were transformed from nominal using WEKA's filters). However, LW can learn over any distance space provided that a distance function is defined (it does not need to satisfy the metric axioms nor any of the conditions that kernel functions are required to satisfy for SVM). This can be a significant advantage over some of the other algorithms mentioned above in learning problems where the data is unstructured or even featureless.

4 Variants

The results mentioned in the previous section are based on LW's algorithm after testing also some other variants of its classification rule and the selection rule used to find the best point. While they did not improve the accuracy of LW, we still think it is interesting to mention them.

4.1 Choosing best point differently

We repeated the experiment using another variant which differs in procedure `findBestPoint` in that it selects a point which has the minimum (over all unlabeled points) of the

Dataset	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
zoo	95.93	97.03	97.03	97.03	96.53	96.53	96.53	96.53	96.53	96.53	95.15	90.58 ●
LSVT	82.62	75.88	75.88	75.88	76.49	74.87 ●	75.87	75.67	73.56 ●	74.49 ●	52.71 ●	71.09 ●
iris	94.87	95.40	95.40	95.40	95.33	95.07	95.20	95.73	95.33	95.47	94.67	93.47
glass	68.52	69.95	69.95	69.95	69.95	69.76	69.95	69.95	69.76	69.95	69.14	41.10 ●
breast cancer	67.18	67.16	67.16	67.16	66.74	66.42	66.81	65.68	65.12	65.58	64.14	66.63
ionosphere	93.88	87.10 ●	87.10 ●	87.10 ●	89.77 ●	90.00 ●	89.77 ●	89.77 ●	90.00 ●	89.77 ●	84.05 ●	82.37 ●
voting	94.18	93.13	93.13	93.13	92.49	92.62	92.60	92.42	92.57	92.48	92.05	94.25
autism	97.23	93.37 ●	93.37 ●	93.37 ●	94.71 ●	93.54 ●	94.71 ●	94.71 ●	93.52 ●	94.71 ●	93.37 ●	100.00 ○
absenteeism	64.57	63.84	63.84	63.84	63.84	63.84	63.84	63.84	63.84	63.84	62.38	87.86 ○
diabetes	74.15	70.62 ●	70.62 ●	70.62 ●	73.74	72.29	73.87	73.24	72.93	73.30	68.30 ●	71.40
credit	70.60	71.08	71.08	71.08	70.86	71.03	70.86	70.60	71.03	70.86	67.35	60.20 ●
sports	81.78	77.33 ●	77.33 ●	77.33 ●	81.55	81.68	81.55	82.32	82.38	82.32	75.53 ●	75.96 ●
web phishing	86.90	87.12	87.12	87.12	86.96	86.92	86.94	86.96	86.92	86.94	86.34	81.80 ●
seismic bumps	89.40	89.38	89.38	89.38	89.38	89.38	89.38	89.38	89.38	89.38	78.57 ●	80.33 ●

○, ● statistically significant improvement or degradation

Table 2: LW (1) versus other instance-based algorithms (2)–(12)

Resultset	Wins−Losses	Wins	Losses
(1)	39	41	2
(7)	18	22	4
(5)	18	22	4
(10)	14	19	5
(9)	14	19	5
(8)	14	18	4
(6)	14	19	5
(4)	-11	9	20
(3)	-11	9	20
(2)	-11	9	20
(12)	-48	23	71
(11)	-50	3	53

Table 3: Ranking of LW (1) and other instance-based algorithms (2)–(12)

Resultset	Wins−Losses	Wins	Losses
(1)	3	3	0
(2)	0	1	1
(3)	-3	0	3

Table 4: Ranking of MLP: (1) single hidden layer, (2) two hidden layers, (3) three hidden layers

Resultset	Wins−Losses	Wins	Losses
(1)	9	18	9
(2)	8	14	6
(5)	7	16	9
(3)	7	15	8
(4)	-31	1	32

Table 5: Ranking of SVM with different kernels: (1) polynomial of first degree, (2) second degree, (3) third degree, (4) RBF, (5) Pearson universal

set of all maximum vote-set weighted cardinalities; that is, the best point p_t^* is

$$p_t^* := \operatorname{argmin}_{p \in U_t} \max_{1 \le k \le M} |V_k(p)|_{\mathsf{w}} \, .$$

In this variant, LW starts to classify points which are farther away from the Bayes' border of the underlying class-conditional probability distributions. A point that has all empty vote-sets has a preference for being selected for classification over a point that has at least one non-empty vote-set. The accuracy results that we obtained with this variant are very close to the results mentioned in Table 6 and hence we conclude that this variant does not improve on Procedure 2.

4.2 Weighted vote-sets

We tried the following alternative, motivated by [9]. Given an unlabeled point p, define the weighted-cardinality of its vote set $V_k(p)$ as follows:

$$|V_k(p)|_W := \sum_{z \in V_k(p)} \left[1 - \frac{d\,(z,p)}{d(z, NUN(z))} \right]_+$$

where for a real value a, $[a]_+ = a$ if $a \ge 0$ and is equal to zero otherwise. In words, instead of counting the elements of the set, each element of $V_k(p)$ contributes a weight towards the overall weighted count. If a point p is not contained in the ball $B_{NUN}(z)$ then it contributes a value of zero to the overall count. Otherwise, it contributes a value between 0 and 1 such that the farther the point p from z the lower the contribution. We can view this weight as an indication of how deeply embedded p is within the $B_{NUN}(z)$ ball.

Dataset	(1)	(2)	(3)	(4)	(5)	(6)
zoo	95.93	96.53	95.95	92.01	96.05	96.95
LSVT	82.62	75.87	85.43	75.60	85.85	53.99 ●
iris	94.87	95.20	96.73	94.73	96.27	95.33
glass	68.52	69.95	62.92	61.60	48.47 ●	38.87 ●
breast cancer	67.18	66.81	66.10	60.46	67.55	68.75
ionosphere	93.88	89.77 ●	90.74 ●	90.83	86.52 ●	81.31 ●
voting	94.18	92.60	94.04	95.42	95.61	94.02
autism	97.23	94.71 ●	99.96 ○	100.00 ○	99.86 ○	92.97 ●
absenteeism	64.57	63.84	75.05 ○	100.00 ○	60.66	73.93 ○
diabetes	74.15	73.87	73.48	72.25	75.62	74.26
credit	70.60	70.86	71.85	66.42 ●	71.45	73.79 ○
sports	81.78	81.55	81.16	75.63 ●	83.58 ○	79.69
web phishing	86.90	86.94	89.34 ○	89.86 ○	83.79 ●	79.64 ●
seismic bumps	89.40	89.38	76.75 ●	85.33 ●	79.84 ●	80.90 ●

○, ● statistically significant improvement or degradation

Table 6: LW (1) versus the best candidates of instance-based learning (2), best candidate of MLP (3), best candidate of SVM (5), decision tree learning (4) and Naive Bayes (6)

Resultset	Wins−Losses	Wins	Losses
(3)	13	20	7
(1)	7	17	10
(2)	0	15	15
(5)	-1	17	18
(4)	-1	20	21
(6)	-18	10	28

Table 7: Ranking of algorithms listed in Table 6

After repeating the above tests, the results indicate that the weighted vote-sets makes LW algorithm have fewer significant wins relative to losses.

5 Conclusions

The paper introduces a new learning algorithm for learning multi-category classification over any distance space. Compared to other large-margin learning algorithms such as the ubiquitous kernel methods, where a kernel function needs to be defined and be an inner product in some higher dimensional space, the LW algorithm distance function does not have to satisfy anything other than perhaps non-negativity. This makes LW easy to apply in classification learning problems on non-standard domains with no need for kernel functions.

We implemented the algorithm in C and interfaced it using the Java Native Interface to the WEKA toolset for testing its accuracy. Its run time is comparable to single-hidden layer MLP in WEKA. On several datasets from the UCI machine learning repository, the LW algorithm outperforms, in terms of the number of wins versus losses, 11 instance-based learning algorithms including the k-nearest neighbor algorithm (with k chosen in a data-dependent manner and optimized via cross validation). Compared to 5 algorithms which include the best candidates of MLP and SVM, and compared to decision tree learning and Naive Bayes, it is ranked in second place.

Acknowledgment

This work was supported in part by the Ministry of Science & Technology, ISRAEL. The authors thank the reviewers for their thoughtful comments and suggestions.

References

1. D. W. Aha, D. Kibler, and M. K Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.
2. M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.
3. M. Anthony and J. Ratsaby. Maximal width learning of binary functions. *Theoretical Computer Science*, 411:138–147, 2010.
4. M. Anthony and J. Ratsaby. Analysis of a multi-category classifier. *Discrete Applied Mathematics*, 160(16-17):2329–2338, 2012.
5. M. Anthony and J. Ratsaby. A hybrid classifier based on boxes and nearest neighbors. *Discrete Applied Mathematics*, 172:1–11, 2014.
6. M. Anthony and J. Ratsaby. Learning bounds via sample width for classifiers on finite metric spaces. *Theoretical Computer Science*, 529:2–10, 2014.
7. M. Anthony and J. Ratsaby. A probabilistic approach to case-based inference. *Theoretical Computer Science*, 589:61–75, 2015.
8. M. Anthony and J. Ratsaby. Multi-category classifiers and sample width. *Journal of Computer Systems and Sciences*, 82(8):1223–1231, 2016.
9. M. Anthony and J. Ratsaby. Classification based on prototypes with spheres of influence. *Information and Computation*, 256:372–380, 2017.
10. M. Anthony and J. Ratsaby. Large-width bounds for learning half–spaces on distance spaces. *Discrete Applied Mathematics*, 243:73–89, 2018.
11. M. Anthony and J. Ratsaby. Large width nearest prototype classification on general distance spaces. *Theoretical Computer Science*, 738:65–79, 2018.
12. C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artif. Intell. Rev.*, 11(1–5):11–73, February 1997.

13. U. Chester and J. Ratsaby. Universal distance measure for images. In *Proceedings of the* 27*th IEEE Convention of Electrical Electronics Engineers in Israel (IEEEI'12)*, pages 1–4, Eilat, Israel, November 14-17, 2012.
14. U. Chester and J. Ratsaby. Machine learning for image classification and clustering using a universal distance measure. In N. Brisaboa, O. Pedreira, and P. Zezula, editors, *Proceedings of the 6th International Conference on Similarity Search and Applications (SISAP'13)*, volume 8199 of *Springer Lecture Notes in Computer Science*, pages 59–72, 2013.
15. R. Cilibrasi and P. Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.
16. J. G. Cleary and K. E. Trigg. K*: An instance-based learner using and entropic distance measure. In *Proceedings of the Twelfth International Conference on International Conference on Machine Learning*, ICML'95, pages 108–114, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
17. T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, January 1967.
18. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other Kernel-based learning methods*. Cambridge University Press, 2000.
19. M. Deza and E. Deza. *Encyclopedia of Distances*, volume 15 of *Series in Computer Science*. Springer-Verlag, 2009.
20. S. A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(4):325–327, 1976.
21. R. P. W. Duin, E. Pekalska, and M. Loog. Non-euclidean dissimilarities: Causes, embedding and informativeness. In M Pelillo, editor, *Similarity-Based Pattern Analysis and Recognition. Advances in Computer Vision and Pattern Recognition*, London, 2013. Springer.
22. E. Frank, M. A. Hall, and I. Witten. *The WEKA Workbench, Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*. Morgan Kaufmann, fourth edition, 2016.
23. M. Hall, E. Frank, G. Holmes, B. Pfahringer P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009.
24. M. Li, X. Chen, X. Li, B. Ma, and P. Vitanyi. The similarity metric. *IEEE Trans. Info. Theory*, 50(12):3250–3264, 2004.
25. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
26. C. Nadeau and Y. Bengio. Inference for the generalization error. *Machine Learning*, 52(3):239–281, 2003.
27. E. Pekalska and R. P. W. Duin. *The Dissimilarity Representation for Pattern Recognition: Foundations And Applications (Machine Perception and Artificial Intelligence)*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2005.
28. J. Ratsaby and A. Sabaty. Parallelizing the large width learning algorithm. In *IEEE International Conference on the Science of Electrical Engineering (ICSEE'2018)*, pages 1–5, December 14-16, 2018.
29. UCI Machine Learning Repository.
30. A. J. Smola, P. L. Bartlett, B. Scholkopf, and D. Schurmans. *Advances in Large-Margin Classifiers (Neural Information Processing)*. MIT Press, 2000.

Appendix

For the experiment displayed in Table 2 the algorithms' parameter settings are as follows:

(1) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.LW – -M 0 -I 0 -R true' -4523450618538717400
(2) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.IBk – -K 1 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A /\"weka.core.EuclideanDistance -R first-last/\"\"' -4523450618538717400
(3) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.IBk – -K 1 -W 0 -I -A \"weka.core.neighboursearch.LinearNNSearch -A /\"weka.core.EuclideanDistance -R first-last/\"\"' -4523450618538717400
(4) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.IBk – -K 1 -W 0 -F -A \"weka.core.neighboursearch.LinearNNSearch -A /\"weka.core.EuclideanDistance -R first-last/\"\"' -4523450618538717400
(5) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.IBk – -K 5 -W 0 -X -A \"weka.core.neighboursearch.LinearNNSearch -A /\"weka.core.EuclideanDistance -R first-last/\"\"' -4523450618538717400
(6) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.IBk – -K 5 -W 0 -X -I -A \"weka.core.neighboursearch.LinearNNSearch -A /\"weka.core.EuclideanDistance -R first-last/\"\"' -4523450618538717400
(7) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.IBk – -K 5 -W 0 -X -F -A \"weka.core.neighboursearch.LinearNNSearch -A /\"weka.core.EuclideanDistance -R first-last/\"\"' -4523450618538717400
(8) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.IBk – -K 10 -W 0 -X -A \"weka.core.neighboursearch.LinearNNSearch -A /\"weka.core.EuclideanDistance -R first-last/\"\"' -4523450618538717400
(9) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.IBk – -K 10 -W 0 -X -I -A \"weka.core.neighboursearch.LinearNNSearch -A /\"weka.core.EuclideanDistance -R first-last/\"\"' -4523450618538717400
(10) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.IBk – -K 10 -W 0 -X -F -A \"weka.core.neighboursearch.LinearNNSearch -A /\"weka.core.EuclideanDistance -R first-last/\"\"' -4523450618538717400
(11) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.KStar – -B 20 -M a' -4523450618538717400
(12) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.LWL – -U 0 -K -1 -A \"weka.core.neighboursearch.LinearNNSearch -A /\"weka.core.EuclideanDistance -R first-last/\"\" -W trees.DecisionStump' -4523450618538717400

For the experiment displayed in Table 6 the algorithms' parameter settings are as follows:

(1) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.LW – -M 0 -I 0 -R true' -4523450618538717400
(2) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W lazy.IBk – -K 5 -W 0 -X -F -A \"weka.core.neighboursearch.LinearNNSearch -A /\"weka.core.EuclideanDistance -R first-last/\"\"' -4523450618538717400
(3) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W functions.MultilayerPerceptron – -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a' -4523450618538717400
(4) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W trees.J48 – -C 0.25 -M 2' -4523450618538717400
(5) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W functions.SMO – -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\" -calibrator \"functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4\"' -4523450618538717400
(6) meta.FilteredClassifier '-F \"supervised.instance.ClassBalancer -num-intervals 10\" -S 1 -W bayes.NaiveBayes' -4523450618538717400