k-Version-Space Multi-class Classification Based on k-Consistency Tests

Evgueni Smirnov[1], Georgi Nalbantov[1], and Nikolay Nikolaev[2]

[1] Department of Knowledge Engineering, Maastricht University,
P.O. BOX 616, 6200 MD Maastricht, The Netherlands
{smirnov,g.nalbantov}@maastrichtuniversity.nl
[2] Department of Computing, Goldsmiths College, University of London,
London SE14 6NW, United Kingdom
n.nikolaev@gold.ac.uk

Abstract. k-Version spaces were introduced in [6] to handle noisy data. They were defined as sets of k-consistent hypotheses; i.e., hypotheses consistent with all but k instances. Although k-version spaces were applied, their implementation was intractable due to the boundary-set representation.

This paper argues that to classify with k-version spaces we do not need an explicit representation. Instead we need to solve a general k-consistency problem and a general k0-consistency problem. The general k-consistency problem is to test the hypothesis space for classifier that is k-consistent with the data. The general k0-consistency problem is to test the hypothesis space for classifier that is k-consistent with the data and 0-consistent with a labeled test instance. Hence, our main result is that the k-version-space classification can be (tractably) implemented if we have (tractable) k-consistency-test algorithms and (tractable) k0-consistency-test algorithms. We show how to design these algorithms for any learning algorithm in multi-class classification setting.

Keywords: Classification, k-Version Spaces, Consistency Problems.

1 Introduction

Version spaces form a well-known approach to two-class classification [6–8]. Given a hypothesis space, version spaces are sets of hypotheses consistent with training data D. The consistency criterion implies that version spaces provide correct instance classifications if D is noise-free. Otherwise, version spaces can misclassify instances.

To tackle the problem with noisy data Mitchell introduced k-version spaces in [6]. k-Version spaces were defined as sets of all the hypotheses that are k-consistent with the training data D; i.e., hypotheses that are consistent with all but k instances in D. The key idea is that if we have m class-mislabeled instances in D and $m \leq k$, we can still have a hypothesis in the k-version space that is inconsistent with these m-instances and consistent with most of the remaining data in D. Thus, the parameter k is capable of filtering the noise in the training data D.

Representing k-version spaces is a difficult problem. Any k-version space consists of $\binom{|D|}{|D|-k}$ number of 0-version spaces that are not overlapped in the worst case. Thus, any

J.L. Balcázar et al. (Eds.): ECML PKDD 2010, Part III, LNAI 6323, pp. 277–292, 2010.

version-space representation employed (such as boundary sets [6–8], one-sided boundary sets [4], instance-based boundary sets [9, 10, 12] etc.) results in intractable representation of k-version spaces. This implies that the classification algorithm of k-version spaces is intractable as well in this case.

Following [5] this paper focuses on the classification side of the problem instead of the representational. We prove that the problem of k-version-space classification is a recursive problem that consists of the problem of $k-1$-version-space classification, the general k-consistency problem, and the general $k0$-consistency problem. We define the general k-consistency problem as a problem of determining whether there exists classifier in the hypothesis space that is k-consistent with the data. We define the general $k0$-consistency problem as a problem of determining whether there exists classifier in the hypothesis space that is k-consistent with the data and 0-consistent with a labeled test instance. Thus, our main result is that the k-version-space classification can be (tractably) implemented as soon as we have a (tractable) k-consistency-test algorithm and a (tractable) $k0$-consistency-test algorithm for the hypothesis space. The consistency-test algorithms can be applied to any class. Thus, we allow k-version spaces to be applied for multi-class classification tasks. This contrasts with the original formulation of k-version spaces proven for two-class classification tasks only due to the Boolean nature of version-space representations [6–8].

The practical contribution of our paper is that we show how to design k-consistency-test algorithms and $k0$-consistency-test algorithms for any learning algorithm and its hypothesis space. We demonstrate how consistency-test algorithms can be used in logical and probabilistic settings. Hence, our work converts k-version spaces to a meta framework applicable for any learning algorithm.

The paper is organized as follows. Section 2 formalizes the classification task. The k-version spaces are introduced in Section 3. Subsections 3.1, 3.2, and 3.3 provide our generalization of the k-consistency criterion, k-version spaces, and their classification rule for multi-class classification. The classification function of k-version spaces together with the general k-consistency problem and the general $k0$-consistency problem are introduced in Subsection 3.4. Sections 4 and 5 show how to implement k-consistency-test and $k0$-consistency-test algorithms in logical and probabilistic settings. The experiments are given in Section 6. Finally, Section 7 concludes the paper.

2 Classification Task

Let X be a non-empty instance space and Y be a non-empty class set s.t. $|Y| > 1$. A labeled instance is defined as a tuple (x, y) where $x \in X$ and $y \in Y$. Training data D is a multi-set of labeled instances.[1] Given data D and instance $x \in X$ to be classified, the classification task is to assign a class $y \in Y$ to x.

To assign a class $y \in Y$ to an instance $x \in X$ we need a scoring classifier $h : X \to \mathcal{P}(\mathbb{R})$ from a hypothesis space H. The classifier h outputs for x a posterior distribution of scores $\{s(y)\}_{y \in Y}$ over the classes in Y. The final class $y \in Y$ for x is determined by a class function $c : \mathcal{P}(\mathbb{R}) \to Y$. The function c receives as argument the score distribution $\{s(y)\}_{y \in Y}$ and then outputs a class $y \in Y$ according to some rule (usually

[1] The multi-set notation employed in this paper is that from [14].

the class with highest score $s(y)$). Hence, the function composition $c(h(x))$ forms a discrete classifier that assigns the final class $y \in Y$ to the instance x to be classified.

To identify the scoring classifier h we need a learning algorithm $l : \mathcal{P}(X \times Y) \rightarrow \mathcal{P}(\mathbb{R})^X$. Given training data D, the algorithm l searches the hypothesis space H and then outputs the scoring classifier $h \in H$. The goal is to find the classifier h s.t. the discrete classifier $c(h)$ classifies correctly future unseen instances iid drawn from the same probability distribution from which D was drawn.

3 Multi-class *k*-Version Spaces

Mitchell proposed k-version spaces in [6] for two-class classification in the presence of noisy training data. This section extends the k-version spaces for multi-class classification. We first introduce the k-consistency criterion. Then we define k-version spaces and show how they can be used for classification if we can implement two consistency tests.

3.1 *k*-Consistency Criterion

The multi-class k-consistency criterion generalizes the two-class k-consistency criterion [6]. Its definition and properties are given below.

Definition 1. *Given hypothesis space H, class function c, data D, and integer $k \leq |D|$, scoring classifier $h \in H$ is said to be k-consistent with D, denoted by $cons_k(h, D)$, iff:*

$$(\exists D_k \in \mathcal{P}_k(D))(\forall (x, y) \in D_k) c(h(x)) = y,$$

where $\mathcal{P}_k(D) = \{D_k \subseteq D || D_k| = |D| - k\}$.

The integer k determines the extent of consistency of classifier h with respect to D. The boundary cases for k are given in Corollary 1.

Corollary 1. *Consider data D and integer $k \leq |D|$. Then:*

- *if $k < 0$, $(\forall h \in H) \neg cons_k(h, D)$;*
- *if $k = 0$, $(\forall h \in H)(cons_k(h, D) \leftrightarrow (\forall (x, y) \in D) c(h(x)) = y)$;*
- *if $k = |D|$, $(\forall h \in H) cons_k(h, D)$.*

The k-consistency has an important implication property formulated in Theorem 1.

Theorem 1. *Consider scoring classifier $h \in H$, data D_1 and D_2 s.t. $D_2 \subseteq D_1$, and integers k_1 and k_2 s.t. $k_1 \leq k_2 \leq |D_2|$. Then: $cons_{k_1}(h, D_1) \rightarrow cons_{k_2}(h, D_2)$.*

Proof. Consider arbitrary scoring classifier $h \in H$ s.t. $cons_{k_1}(h, D_1)$. By Definition 1:

$$(\exists D_{k_1} \in \mathcal{P}_{k_1}(D_1))(\forall (x, y) \in D_{k_1}) c(h(x)) = y. \tag{1}$$

$k_1 \leq k_2$ and $D_2 \subseteq D_1$ imply $(\forall D_{k_1} \in \mathcal{P}_{k_1}(D_1))(\exists D_{k_2} \in \mathcal{P}_{k_2}(D_2)) D_{k_1} \supseteq D_{k_2}$. Thus, formula (1) implies $(\exists D_{k_2} \in \mathcal{P}_{k_2}(D_2))(\forall (x, y) \in D_{k_2}) c(h(x)) = y$. The latter by Definition 1 is equivalent to $cons_{k_2}(h, D_2)$. □

3.2 *k*-Version Spaces: Definition and Properties

The multi-class k-version space for training data D is the set of all the classifiers in a hypothesis space H that are k-consistent with D.

Definition 2. *Given data D and integer $k \leq |D|$, the k-version space $VS_k(D)$ equals:*

$$\{h \in H | cons_k(h, D)\}.$$

The key idea of k-version spaces is that if we have m class-mislabeled instances in the training data D and $m \leq k$, then we can still have a scoring classifier h in the k-version space that is inconsistent with these m-instances and consistent with most of the remaining data in D. Thus, the integer k is capable of filtering the noise in D. We consider three boundary cases for k formulated in Corollary 2.

Corollary 2. *Consider data multi-set D and integer $k \leq |D|$. Then:*

(1) if $k < 0$, then $VS_k(D) = \emptyset$,
(2) if $k = 0$, then $VS_k(D) = \{h \in H | (\forall (x, y) \in D) c(h(x)) = y\}$,
(3) if $k = |D|$, then $VS_k(D) = H$.

The implication property of the k-consistency from Theorem 1 entails a sub-set property of the k-version spaces formulated in Theorem 2.

Theorem 2. *Consider data D_1 and D_2 s.t. $D_2 \subseteq D_1$, and integers k_1 and k_2 s.t. $k_1 \leq k_2$ and $k_2 \leq |D_2|$. Then: $VS_{k_1}(D_1) \subseteq VS_{k_2}(D_2)$.*

Proof. The proof follows from Theorem 1. □

3.3 *k*-Version-Space Classification

Given k-version space $VS_k(D)$, the k-version-space classification rule assigns a class set $VS_k(D)(x) \subseteq Y$ to any instance $x \in X$. The class set $VS_k(D)(x)$ includes any class $y \in Y$ s.t. there exists a scoring classifier $h \in VS_k(D)$ that is 0-consistent with $\wr(x, y)\int$; i.e., the discrete classifier $c(h(x))$ assigns class y to x.

Definition 3. *Given data D, integer $k < |D|$, and k-version space $VS_k(D)$, instance $x \in X$ receives a k-class set $VS_k(D)(x)$ equal to:*

$$\{y \in Y | (\exists h \in VS_k(D)) cons_0(h, \wr(x, y)\int)\}.$$

Our k-version-space classification rule is more general than the Mitchell's one [6]. It provides a k-class set $VS_k(D)(x)$ for any instance $x \in X$ instead of just one class or no class. To test whether $VS_k(D)(x)$ is empty or not we introduce Theorem 3. The Theorem states that $VS_k(D)(x)$ is empty iff the k-version space $VS_k(D)$ is empty.

Theorem 3. *For any data D and instance $x \in X$:*

$$VS_k(D)(x) = \emptyset \leftrightarrow VS_k(D) = \emptyset.$$

Proof. For any data D and instance $x \in X$:

$VS_k(D)(x) = \emptyset$ **iff** [by Definition 3]
$(\forall y \in Y)\neg(\exists h \in VS_k(D))cons_0(h, \wr(x, y)\int)$ **iff** [by Definition 1]
$(\forall y \in Y)\neg(\exists h \in VS_k(D))c(h(x)) = y$ **iff**
$(\forall y \in Y)(\forall h \in VS_k(D))c(h(x)) \neq y$ **iff** [c is a function from $\mathcal{P}(\mathbb{R})$ to Y]
$VS_k(D) = \emptyset$. □

The problem to classify an instance $x \in X$ by a k-version space $VS_k(D)$ according to Definition 3 is called *the k-version-space classification problem*. In the next Subsection 3.4 we propose one solution to this problem based on two consistency tests.

3.4 k-Consistency Tests

Consider a k-version space $VS_k(D)$ and an instance $x \in X$ to be classified. By Theorem 3 if $VS_k(D)$ is empty, then the k-class set $VS_k(D)(x)$ is empty. If $VS_k(D)$ is non-empty, then to classify x we need to compute the non-empty k-class set $VS_k(D)(x)$. Therefore, we divide the k-version-space classification problem into two sub-problems:

(1) k-collapse problem: to decide whether the k-version space $VS_k(D)$ is empty;
(2) k-class-set problem: to compute the k-class set $VS_k(D)(x)$ for the instance x if $VS_k(D)$ is non-empty.

For the k-collapse problem we formulate Theorem 4. The Theorem introduces a test to decide whether the k-version space $VS_k(D)$ is empty.

Theorem 4. *Consider data D and integer $k \leq |D|$. Then:*

$$VS_k(D) \neq \emptyset \leftrightarrow (VS_{k-1}(D) \neq \emptyset \vee (\exists h \in H)(cons_k(h, D) \wedge \neg cons_{k-1}(h, D))).$$

Proof. For any data D and integer $k \leq |D|$:

$VS_k(D) \neq \emptyset$ **iff** [by Definition 2]
$(\exists h \in H)cons_k(h, D)$ **iff**
$(\exists h \in H)((cons_k(h, D) \wedge cons_{k-1}(h, D)) \vee (cons_k(h, D) \wedge \neg cons_{k-1}(h, D)))$ **iff**
[by Theorem 1 $cons_k(h, D) \leftarrow cons_{k-1}(h, D)$]
$(\exists h \in H)(cons_{k-1}(h, D) \vee (cons_k(h, D) \wedge \neg cons_{k-1}(h, D)))$ **iff**
$(\exists h \in H)cons_{k-1}(h, D) \vee (\exists h \in H)(cons_k(h, D) \wedge \neg cons_{k-1}(h, D))$ **iff**
[by Definition 2]
$VS_{k-1}(D) \neq \emptyset \vee (\exists h \in H)(cons_k(h, D) \wedge \neg cons_{k-1}(h, D))$. □

By Theorem 4 a k-version space $VS_k(D)$ is non-empty iff the $k-1$-version space $VS_{k-1}(D)$ is non-empty or there exists a scoring classifier $h \in H$ that is k-consistent and $k-1$-inconsistent with data D. We note that the problem to decide whether the $k-1$-version space $VS_{k-1}(D)$ is empty is the $k-1$-collapse problem. The problem to decide whether there exists a scoring classifier $h \in H$ that is k-consistent and $k-1$-inconsistent with D is a new problem that we call *exact k-consistency problem*. Thus, the k-collapse problem is a recursive problem. By Corollary 1 the recursion is restricted below for $k = -1$, since $cons_{-1}(h, D)$ is false.

Theorem 4 does not specify whether to solve first the $k-1$-collapse problem or the exact k-consistency problem. However, by Lemma 1, if the test for the $k-1$-collapse problem is negative (i.e., $VS_{k-1}(D) = \emptyset$), then the exact k-consistency problem is simplified to a problem to decide whether there exists a scoring classifier $h \in H$ that is *only* k-consistent with D.

Lemma 1. *Consider data D and integer $k \leq |D|$. If $VS_{k-1}(D) = \emptyset$, then:*

$$(\exists h \in H)(cons_k(h, D) \wedge \neg cons_{k-1}(h, D)) \leftrightarrow (\exists h \in H) cons_k(h, D).$$

Proof. The ($\rightarrow$) part of the proof is obvious. Thus, we provide the ($\leftarrow$) part only. Consider data D and integer $k \leq |D|$. If $VS_{k-1}(D) = \emptyset$, then $\neg(\exists h \in H) cons_{k-1}(h, D)$; i.e., $(\forall h \in H)\neg cons_{k-1}(h, D)$. The latter and $(\exists h \in H) cons_k(h, D)$ imply:

$$(\exists h \in H)(cons_k(h, D) \wedge \neg cons_{k-1}(h, D)). \qquad \square$$

The problem to decide whether there exists a scoring classifier $h \in H$ that is k-consistent with data D is a new problem that we call *general k-consistency problem.*

Definition 4. (General k-Consistency Problem). *Given hypothesis space H and data D, the general k-consistency problem is to determine:* $(\exists h \in H) cons_k(h, D)$.

By combining the results of Theorem 4 and Lemma 1 we determine the order of computation for the k-collapse problem. First we solve the $k-1$-collapse problem. If $VS_{k-1}(D) \neq \emptyset$, then by Theorem 4 $VS_k(D) \neq \emptyset$. If $VS_{k-1}(D) = \emptyset$, then we solve the general k-consistency problem since this problem by Lemma 1 is equivalent to the exact k-consistency problem. Thus, we conclude that the k-collapse problem is a recursive problem that consists of the $k-1$-collapse problem and the general k-consistency problem in the proposed order of computations.

For the k-class-set problem we formulate Theorem 5. The Theorem introduces a test for any class $y \in Y$ to determine whether y belongs to the k-class set $VS_k(D)(x)$ assigned to instance x to be classified.

Theorem 5. *For any data D, integer $k \leq |D|$, instance $x \in X$, and class $y \in Y$:*

$$y \in VS_k(D)(x) \leftrightarrow (y \in VS_{k-1}(D)(x) \vee (\exists h \in H)(cons_k(h, D) \wedge cons_0(h, \wr(x, y)\int) \wedge \neg cons_{k-1}(h, D))).$$

Proof. For any data D, integer $k \leq |D|$, instance $x \in X$, and class $y \in Y$:

$y \in VS_k(D)(x)$ **iff** [by Definition 3]
$(\exists h \in VS_k(D)) cons_0(h, \wr(x, y)\int)$ **iff** [by Theorem 2 $VS_{k-1}(D) \subseteq VS_k(D)$]
$(\exists h \in VS_{k-1}(D)) cons_0(h, \wr(x, y)\int) \vee$
$(\exists h \in VS_k(D) \setminus VS_{k-1}(D)) cons_0(h, \wr(x, y)\int)$ **iff** [by Definitions 2 and 3]
$y \in VS_{k-1}(D)(x) \vee$
$(\exists h \in H)(cons_k(h, D) \wedge cons_0(h, \wr(x, y)\int) \wedge \neg cons_{k-1}(h, D)).$ $\square$

By Theorem 5 a class $y \in Y$ belongs to the k-class set $VS_k(D)(x)$ iff y belongs to the $k-1$-class set $VS_{k-1}(D)(x)$, or there exists a scoring classifier $h \in H$ that is k-consistent with D, 0-consistent with $\wr(x, y)\int$, and $k-1$-inconsistent with D. We note that the problem to determine whether the class y belongs to the class set $VS_{k-1}(D)(x)$ is the $k-1$-class-set problem for the class y. The problem to decide whether there exists a scoring classifier $h \in H$ that is k-consistent with D, 0-consistent with $\wr(x, y)\int$, and $k-1$-inconsistent with D is a new problem that we call *exact $k0$-consistency problem*. Thus, the k-class-set problem is a recursive problem. By Corollary 1 the recursion is restricted below for $k = -1$, since $cons_{-1}(h, D)$ is false.

Theorem 5 does not specify whether we have first to solve the $k-1$-class-set problem or the exact $k0$-consistency problem. However, by Lemma 2, if for some class $y \in Y$ the result of the $k-1$-class-set problem is negative (i.e., $y \notin VS_{k-1}(D)(x)$), the exact $k0$-consistency problem is simplified to a problem to decide whether there exists a classifier $h \in H$ that is *only* k-consistent with D and 0-consistent with $\wr(x, y)\int$.

Lemma 2. *Consider data D, integer $k \leq |D|$, instance $x \in X$, and class $y \in Y$. If $y \notin VS_{k-1}(D)(x)$, then:*

$$(\exists h \in H)(cons_k(h, D) \wedge cons_0(h, \wr(x, y)\int) \wedge \neg cons_{k-1}(h, D)) \leftrightarrow$$
$$(\exists h \in H)(cons_k(h, D) \wedge cons_0(h, \wr(x, y)\int)).$$

Proof. The ($\rightarrow$) part of the proof is obvious. Hence, we provide the ($\leftarrow$) part only. Consider data D, integer $k \leq |D|$, instance $x \in X$, and class $y \in Y$. If $y \notin VS_{k-1}(D)(x)$, by Definition 3 $\neg(\exists h \in VS_{k-1}(D))cons_0(h, \wr(x, y)\int)$. By Definition 2 the latter implies $\neg(\exists h \in H)(cons_{k-1}(h, D) \wedge cons_0(h, \wr(x, y)\int))$ which is equivalent to:

$$(\forall h \in H)(\neg cons_{k-1}(h, D) \vee \neg cons_0(h, \wr(x, y)\int)).$$

Thus,

$$(\exists h \in H)(cons_k(h, D) \wedge cons_0(h, \wr(x, y)\int)) \textbf{ iff}$$
$$(\exists h \in H)(cons_k(h, D) \wedge cons_0(h, \wr(x, y)\int) \wedge$$
$$(\neg cons_{k-1}(h, D) \vee \neg cons_0(h, \wr(x, y)\int))) \textbf{ iff}$$
$$(\exists h \in H)(cons_k(h, D) \wedge cons_0(h, \wr(x, y)\int) \wedge \neg cons_{k-1}(h, D)). \qquad \square$$

The problem to decide whether there exists a scoring classifier $h \in H$ that is k-consistent with data D and 0-consistent with $\wr(x, y)\int$ is a new problem that we call *general $k0$-consistency problem*. Below we provide a definition of this problem.

Definition 5. (General $k0$-Consistency Problem) *Given hypothesis space H, data D, integer $k \leq |D|$, instance $x \in X$, and class $y \in Y$, the general $k0$-consistency problem is to determine:* $(\exists h \in H)(cons_k(h, D) \wedge cons_0(h, \wr(x, y)\int))$.

By combining the results of Theorem 5 and Lemma 2 we determine the order of computation for the k-class-set problem for any k-version space $VS_k(D)$, instance $x \in X$, and class $y \in Y$. First we solve the $k-1$-class-set problem. If $y \in VS_{k-1}(D)(x)$, then by Theorem 5 $y \in VS_k(D)(x)$. If $y \notin VS_{k-1}(D)(x)$, then we solve the general $k0$-consistency problem since this problem by Lemma 2 is equivalent to the exact $k0$-consistency problem. Thus, we conclude that the k-class-set problem is a recursive

Function $Classify$:
Input: integer k, data D, instance x.
Output: class set $VS_k(D)(x)$ assigned to x.
if $k < 0$ **then**
  return $\emptyset$;
$Y_{k-1} := Classify(k-1, D, x)$;
if $Y_{k-1} = \emptyset$ **then**
  if $\neg(\exists h \in H) cons_k(h, D)$ **then**
    return $\emptyset$;
$Y_k := Y_{k-1}$;
for each class $y \in Y \setminus Y_{k-1}$ **do**
  if $(\exists h \in H)(cons_k(h, D) \wedge cons_0(h, \wr(x, y)\int))$ **then**
    $Y_k := Y_k \cup \{y\}$;
return Y_k.

Fig. 1. Classification function of k-version spaces based on the k-consistency tests

problem that consists of the $k-1$-class-set problem and the general $k0$-consistency problem in the proposed order of computations.

So far we showed that the k-version-space classification problem consists of the k-collapse problem and the k-class-set problem. The k-collapse problem consists of the $k-1$-collapse problem and the general k-consistency problem. The k-class-set problem consists of the $k-1$-class-set problem and the general $k0$-consistency problem. Thus, since the $k-1$-collapse problem and the $k-1$-class-set problem form the $k-1$-version-space classification problem, it follows that *the k-version-space classification problem is a recursive problem that consists of the $k-1$-version-space classification problem, the general k-consistency problem, and the general $k0$-consistency problem*. This result implies that the k-version-space classification can be (tractably) implemented as soon as we can (tractably) test for k-consistency and $k0$-consistency in the given hypothesis space. The consistency tests can be applied to any class. Thus, we allow k-version spaces to be applied for multi-class classification tasks.

The classification function of k-version spaces based on the consistency tests is given in Figure 1. The function input includes data D, integer k, and instance $x \in X$. The output is the k-class set $VS_k(D)(x)$ for x provided according to Definition 3.

The function is recursive. It first checks whether $k < 0$. If $k < 0$, then by Corollary 2 the k-version space $VS_k(D)$ is empty. This implies by Theorem 3 that the k-class set $VS_k(D)(x)$ is empty. Thus, the function returns empty set.

If $k \geq 0$, we note that the k-version-space classification problem includes the $k-1$-version-space classification problem. Hence, the function calls itself recursively for $k-1$. The result of the call is the set Y_{k-1} of classes assigned by the $k-1$-version space $VS_{k-1}(D)$ to the instance x. If the class set Y_{k-1} is empty, then by Theorem 3 the $k-1$-version space $VS_{k-1}(D)$ is empty. Thus, by Theorem 4 and Lemma 1 in order to decide whether the k-version space $VS_k(D)$ is non-empty we solve the general k-consistency problem; i.e., we test whether there exists a scoring classifier in H that is

k-consistent with D. If the test is negative, by Definition 2 the k-version space $VS_k(D)$ is empty and by Definition 3 the function returns $\emptyset$. If the test is positive, by Definition 2 $VS_k(D)$ is non-empty. Therefore, the function continues the classification process by initializing the class set Y_k (assigned to the instance x by $VS_k(D)$). By Theorem 5 $Y_{k-1} \subseteq Y_k$. Thus, Y_k is initialized equal to Y_{k-1}. Then the function tests whether the classes from $Y \setminus Y_{k-1}$ can de added to Y_k. We note that for each of these classes Lemma 2 holds. Thus, by Theorem 5 for each class $y \in Y \setminus Y_{k-1}$ we solve the general $k0$-consistency problem for the data D and $\wr(x, y)\int$. This is done by testing whether there exists a scoring classifier $h \in H$ that is k-consistent with D and 0-consistent with $\wr(x, y)\int$. If so, then by Theorem 5 the class y is added to the set Y_k. Once all the classes in $Y \setminus Y_{k-1}$ have been visited the class set Y_k is outputted.

Let T_k be the time complexity of the general k-consistency test and T_{k0} be the time complexity T_{k0} of the general $k0$-consistency test. Assuming that $T_k < T_{k0}$ the worst-case time complexity of the classification function of k-version spaces equals:

$$O(T_k + k|Y|T_{k0}). \tag{2}$$

4 Consistency Algorithms

To implement the classification function of k-version spaces based on the consistency tests we need k-consistency-test algorithms and $k0$-consistency-test algorithms. Below we propose two approaches to implement these algorithms. The first one is for the case when there exists a 0-consistent learning algorithm l for the hypothesis space H. It allows designing consistency-test algorithms valid for the whole hypothesis space H. Hence, it is called hypothesis-unrestrictive approach. The second approach is for the case when there exists no 0-consistent learning algorithm l for the hypothesis space H. It allows designing consistency-test algorithms valid for a sub-space of the hypothesis space H. Hence, it is called hypothesis-restrictive approach.

4.1 Hypothesis-Unrestrictive Approach

The hypothesis-unrestrictive approach assumes that there exists a 0-consistent learning algorithm l for the hypothesis space H. Thus, H contains hypothesis that is 0-consistent with data D iff l succeeds; i.e., l outputs for D some hypothesis (that by definition is consistent with D). This implies that the 0-consistency-test algorithm in this case is the 0-consistent learning algorithm l plus a success test. In the past (cf. [5]) 0-consistency-test algorithms were proposed for different hypothesis spaces such as 1-decision lists, monotone depth two formulas, halfspaces etc. They guarantee tractable 0-version-space classification, if they are tractable.

By Definition 1 if we can test for 0-consistency, we can test for k-consistency. Thus, given data D and integer k, we design a k-consistency-test algorithm as follows. We start with $m = 0$ and then for each $D_m \subseteq D$ with size $|D|$-m we apply a 0-consistency-test algorithm. If the 0-consistency-test algorithm identifies 0-consistency for at least one D_m, by Theorem 1 there is 0-consistency for some $D_k \subseteq D_m$ and we return value "true". Otherwise, we continue with the next D_m or increment m in the boundary of k. If this is not possible, we return value "false". Thus, the worst-case time complexity of the k-consistency-test algorithm is $O(\binom{|D|}{|D|-k}T_l)$ where T_l is the time complexity of the learning algorithm l used in the 0-consistency-test algorithm.

The $k0$-consistency-test algorithms and their worst-case time complexity are analogous. Thus, we conclude that the k-consistency-test and $k0$-consistency-test algorithms based on 0-consistency learning algorithms are intractable in the worst case. Thus, according to formula (2) the k-version space classification is intractable in this case.

4.2 Hypothesis-Restrictive Approach

The hypothesis-restrictive approach assumes that the learning algorithm l provided is not a 0-consistent learning algorithm for the hypothesis space H. The approach restricts H s.t. we can implement the tests for the k-consistency and $k0$-consistency in the constrained space $H(k,D) \subseteq H$ using the algorithm l. Below we define $H(k,D)$ and condition s.t. the consistency tests can be implemented using the learning algorithm l.

The restricted hypothesis space $H(k,D)$ is defined for the learning algorithm l, data D, and integer k. It is non-empty if the scoring classifier $l(D) \in H$ is k-consistent with D. In this case $H(k,D)$ consists of $l(D)$ plus any scoring classifier $l(D \uplus \wr(x,y)\int) \in H$ for some instance $(x,y) \in X \times Y$ that is k-consistent with the data $D \uplus \wr(x,y)\int$.

Definition 6. *Consider hypothesis space H, integer $k \leq |D|$, and data D. If scoring classifier $l(D)$ is k-consistent with D, the hypothesis sub-space $H(k,D) \subseteq H$ equals:*

$$\{l(D)\} \cup \{l(D \uplus \wr(x,y)\int) \in H | (x,y) \in X \times Y \wedge cons_k(l(D \uplus \wr(x,y)\int), D \uplus \wr(x,y)\int)\}.$$

Otherwise, $H(k,D) = \emptyset$.

Any learning algorithm l can be used for consistency testing in the hypothesis sub-space $H(k,D)$ if the instance property holds. This property often holds for stable classifiers like Naive Bayes [3].

Definition 7. (Instance Property) *Learning algorithm l has the instance property iff for any data D and instance $(x,y) \in X \times Y$ if there exists instance $(x',y') \in X \times Y$ s.t. the classifier $l(D \uplus \wr(x',y')\int)$ is k-consistent with $D \uplus \wr(x,y)\int$ and 0-consistent with $\wr(x,y)\int$, then the classifier $l(D \uplus \wr(x,y)\int)$ is k-consistent with $D \uplus \wr(x,y)\int$ and 0-consistent with $\wr(x,y)\int$.*

Below we describe algorithm C_k for the k-consistency test and algorithm C_{k0} for $k0$-consistency test in the restricted hypothesis space $H(k,D)$. The algorithm C_{k0} employs a learning algorithm l under the assumption that the instance property holds.

Algorithm C_k for k-consistency test: The algorithm C_k tests whether there exists a scoring classifier h in $H(k,D)$ that is k-consistent with data D. For that purpose it first builds the scoring classifier $l(D)$. Then, the algorithm tests if $l(D)$ is k-consistent with D. If so, by Definition 6 $H(k,D)$ is non-empty and includes the desired classifier. Otherwise, $H(k,D)$ is empty; i.e., it does not include the desired classifier.

Algorithm C_{k0} for $k0$-consistency test: given a labeled instance $(x,y) \in X \times Y$, the algorithm C_{k0} tests whether there exists a classifier h in $H(k,D)$ that is k-consistent with data D and 0-consistent with data $\Lbag(x,y)\Rbag$. For that purpose it first builds the scoring classifier $l(D)$ and then tests whether $l(D)$ is k-consistent with D. If $l(D)$ is not k-consistent with D by Definition 6 $H(k,D)$ is empty and thus it does not include the desired classifier. Otherwise, $H(k,D)$ is non-empty and the algorithm tests whether the scoring classifier $l(D)$ is 0-consistent with $\Lbag(x,y)\Rbag$. If $l(D)$ is 0-consistent with $\Lbag(x,y)\Rbag$, then $H(k,D)$ includes the desired classifier. Otherwise, the algorithm makes a second attempt. It builds the scoring classifier $l(D \uplus \Lbag(x,y)\Rbag)$ and then tests whether $l(D \uplus \Lbag(x,y)\Rbag)$ is k-consistent with $D \uplus \Lbag(x,y)\Rbag$ and 0-consistent with $\Lbag(x,y)\Rbag$. If both tests are positive, then by Definition 6 $l(D \uplus \Lbag(x,y)\Rbag) \in H(k,D)$, and $l(D \uplus \Lbag(x,y)\Rbag)$ is k-consistent with D and 0-consistent with $\Lbag(x,y)\Rbag$; i.e., $H(k,D)$ includes the desired classifier. If at least one of the tests is negative, then by Definition 7 it follows that there exists no instance $(x',y') \in X \times Y$ s.t. $l(D \uplus \Lbag(x',y')\Rbag)$ is consistent with $D \uplus \Lbag(x,y)\Rbag$ and 0-consistent with $\Lbag(x,y)\Rbag$; i.e., there exists no instance $(x',y') \in X \times Y$ s.t. $l(D \uplus \Lbag(x',y')\Rbag)$ is consistent with D and 0-consistent with $\Lbag(x,y)\Rbag$. Thus, by Definition 6 $H(k,D)$ does not include the desired classifier.

The correctness of the algorithm C_{k0} is proven in Theorem 6.

Theorem 6. *If the instance property holds, then for any data $D \subseteq X \times Y$, integer $k \leq |D|$, instance $x \in X$, and class $y \in Y$ we have:*

$$
\begin{aligned}
&(\exists h \in H(k,D))(cons_k(h,D) \wedge cons_0(h, \Lbag(x,y)\Rbag)) \leftrightarrow \\
&\qquad (cons_0(l(D), \Lbag(x,y)\Rbag) \vee \\
&\qquad\ (cons_k(l(D \uplus \Lbag(x,y)\Rbag), D \uplus \Lbag(x,y)\Rbag) \wedge cons_0(l(D \uplus \Lbag(x,y)\Rbag), \Lbag(x,y)\Rbag))).
\end{aligned}
$$

Proof. **(→)** Consider arbitrary data $D \subseteq X \times Y$, integer $k \leq |D|$, instance $x \in X$, and class $y \in Y$ so that:

$$(\exists h \in H(k,D))(cons_k(h,D) \wedge cons_0(h, \Lbag(x,y)\Rbag)).$$

Thus, by Definition 6:

$$
\begin{aligned}
&cons_0(l(D), \Lbag(x,y)\Rbag) \vee \\
&(\exists (x',y') \in X \times Y)(cons_k(l(D \uplus \Lbag(x',y')\Rbag), D \uplus \Lbag(x',y')\Rbag) \wedge \\
&\qquad\qquad cons_k(l(D \uplus \{(x',y')\}), D) \wedge \\
&\qquad\qquad cons_0(l(D \uplus \Lbag(x',y')\Rbag), \Lbag(x,y)\Rbag)).
\end{aligned}
$$

which implies:

$$
\begin{aligned}
&cons_0(l(D), \Lbag(x,y)\Rbag) \vee \\
&(\exists (x',y') \in X \times Y)(cons_k(l(D \uplus \{(x',y')\}), D \uplus \Lbag(x,y)\Rbag) \wedge \\
&\qquad\qquad cons_0(l(D \uplus \Lbag(x',y')\Rbag), \Lbag(x,y)\Rbag)).
\end{aligned}
$$

By Definition 7 the latter implies:

$$
\begin{aligned}
&cons_0(l(D), \Lbag(x,y)\Rbag) \vee \\
&(cons_k(l(D \uplus \Lbag(x,y)\Rbag), D \uplus \Lbag(x,y)\Rbag) \wedge cons_0(l(D \uplus \Lbag(x,y)\Rbag), \Lbag(x,y)\Rbag)).
\end{aligned}
$$

(←) This part of the Theorem follows from Definitions 6 and 7. □

The time complexity of the consistency-test algorithms C_k and C_{k0} is $O(T_l + |D|T_c)$ where T_l is the time complexity of the learning algorithm l and T_c is the time complexity to classify with the scoring classifier $l(D)$ (derived by l). Thus, according to formula (2) the hypothesis-restrictive approach guarantees tractable k-version-space classification iff the learning algorithm l and corresponding scoring classifier $l(D)$ are tractable.

We conclude this Section with a remark. The hypothesis-unrestrictive approach and hypothesis-restrictive approach together allow implementing k and $k0$-consistency-test algorithms independent on the type of learning algorithms. Thus, they together with the k-version-space classification function (from Figure 1) form a meta k-version-space framework that is applicable for any type of learning algorithms.

5 Implementing *k*-Consistency

The key to success of our hypothesis restrictive approach is the problem of implementing k-consistency: how to decide whether a classifier is k-consistent with data. In this Section we consider two possible implementations: logical and probabilistic.

5.1 Logical *k*-Consistency

The logical implementation of k-consistency follows Definition 1. To decide whether a classifier h is k-consistent with data D we first test h on D and determine a multi-set $D_c \subseteq D$ of correctly classified instances from D. If $|D_c| \geq |D| - k$, then we output value “true”; otherwise, we output value “false”.

5.2 Probabilistic *k*-Consistency

The probabilistic implementation of k-consistency is based on Definition 1 and the generalized binomial distribution.[2] Consider a scoring classifier $h \in H$. It outputs for any instance a distribution of scores $\{s(y)\}_{y \in Y}$. If the scores are normalized, we receive a distribution of estimated probabilities $\{p(y)\}_{y \in Y}$. If $y_r \in Y$ is the known class of an instance $x \in X$, the experiment to assign class to x according to the distribution $\{p(y)\}_{y \in Y}$ (provided by $h(x)$) is a binary trial with probability of success $p(y_r)$. Thus, to classify all the instances from data D we receive a sequence of $|D|$ independent binary trials, each with different probability of success $p(y_r)$. The probabilities $p(|D| - k)$ that we have $|D| - k$ successes in the sequence of $|D|$ trials for $k \in 0..|D|$ form generalized binomial distribution. The probability $F(|D| - k, |D|)$ that we have $|D| - k$ or more successes equals $\sum_{i=0}^{k} p(|D| - i)$. This probability is actually the probability that the scoring classifier h is k-consistent with D.

Definition 8. *Given the generalized binomial distribution $\{p(i)\}_{i \in 0..|D|}$ of scoring classifier $h \in H$ for data D and a probability threshold $p_t \in [0, 1]$, h is probabilistically k-consistent with D, denoted by p-cons$_k(h, D)$, iff $F(|D| - k, |D|) > p_t$.*

[2] The generalized binomial distribution is a discrete probability distribution of the number of successes in a sequence of n independent binary experiments with *different* success probability.

Implementing the probabilistic k-consistency is as follows. First, we test the scoring classifier h on data D to compute the probabilities of success $p(y_r)$. Then, we derive the generalized binomial distribution $\{p(i)\}_{i \in 0..|D|}$ and compute the probability $F(|D| - k, |D|)$. Finally, by Definition 8 we output the truth value of the probabilistic consistency predicate $p\text{-}cons_k(h, D)$.

If the probabilistic consistency predicate is employed in Definition 2 we receive the definition of k-version spaces based on the probabilistic k-consistency. We note two advantages of the probabilistic k-consistency over the logical one. First, it employs the information from the probabilities $p(y_r)$ of the true classes y_r for all the instances in the data D. Second, it provides solution in the whole range of k from 0 to the size of D.

6 Experiments

This Section presents experiments with our meta k-version-space framework. We employed the k-version-space classification function from Figure 1. The base classifier used in the framework was the Naive-Bayes classifier (NB) [8]. The consistency algorithms and the hypothesis space were designed using the hypothesis-restrictive approach, since NB is not 0-consistent classifier. The k-consistency was implemented using probabilistic k-consistency predicate (see Definition 8), since NB is a probabilistic classifier. The resulting combination we call k-Naive-Bayes Version Spaces (k-NBVSs). Note that k-NBVSs have two parameters, parameter k of k-version spaces and parameter p_t, the probability threshold of the probabilistic k-consistency predicate.

We tested k-NBVSs in the context of the reliable-classification task [1, 11, 15]. The task is to derive a classifier that outputs only reliable instance classifications. This implies that instances with unreliable classifications remain unclassified. Hence, reliable classifiers are evaluated using two measures: coverage rate and accuracy rate. The coverage rate is the proportion of the instances that receive classifications while the accuracy rate is the accuracy on the classified instances.

The reliable-classification task is chosen since it is typical for version spaces [13]. We assume that k-NBVSs provide reliable classification for an instance if only one class is outputted for that instance, i.e., we have an unanimous-voting rule.

We compared k-NBVSs with the NB classifier, since they employ this classifier. NB for reliable classification uses a rejection technique: the class with the highest posterior probability is outputted for an instance if this probability is greater than a user-defined probability threshold [16]. NB with the rejection technique is denoted by NB_r.

The reliable classification experiments were performed on 14 UCI datasets [2] (see Table 1). The evaluation method was 10-fold cross validation. Table 2 reports the best results for k-NBVSs and NB obtained by a grid search for tuning the parameters of these classifiers.[3] More precisely, it shows the maximal coverage rates of each of the classifiers for which the accuracy rate of 1.0 is achieved. The maximal coverage rates show that k-NBVSs outperform the NB_r classifier on 13 out of 14 datasets, 6 times significantly (level 0.05).

[3] Similar experiments with internal parameter tuning were performed. The results are compatible with those from Table 2.

Table 1. The UCI data sets employed in the experiments. A is the number of attributes, I is the number of instances, and C is the number of classes.

Data Set	A	I	C
audiology	70	226	24
breast-cancer	10	286	2
colic	23	386	2
diabetes	9	768	2
heart-statlog	14	270	2
hepatitis	20	155	2
ionoshpere	35	351	2
iris	5	150	3
labor	17	57	2
lymphography	19	148	4
sonar	61	208	2
wisc. breast-cancer	10	699	2
wine	14	178	3
zoo	18	101	7

Table 2. Coverage rate for accuracy rate of 1.0: k-NBVS and NB_r. The numbers in bold present statistically better results on significance level 0.05.

Data Set	k-NBVS	NB_r
audiology	0.1238	0.0708
breast-cancer	0.0734	0.0525
colic	**0.1250**	0.0055
diabetes	**0.0730**	-
heart-statlog	0.1222	0.0334
hepatitis	**0.2903**	0.0259
ionoshpere	**0.2706**	0.1453
iris	0.8933	0.8333
labor	**0.7884**	0.4386
lymphography	0.1959	0.1825
sonar	0.0528	0.0048
wisc. breast-cancer	**0.3147**	0.0473
wine	0.9269	0.8821
zoo	0.9208	0.9208

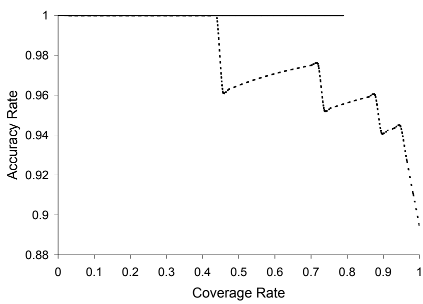

Fig. 2. Coverage/accuracy graphs of k-NBVS (solid line) and NB_r (dashed line) for the labor data when the parameter k equals 0

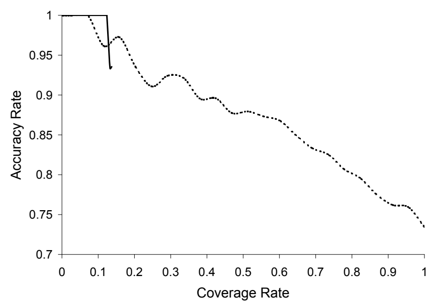

Fig. 3. Coverage/accuracy graphs of k-NBVS (solid line) and NB_r (dashed line) for the audiology data when the parameter k equals 6

In addition we derived coverage/accuracy graphs for all the 14 datasets. One point in the graphs represents a k-NBVSs for some values of the parameters k and p_t. Subsequent points represent k-NBVSs for the same value of k and increased values of p_t in the range $(p_t, 1.0]$. Hence, the graphs show the potential of k-NBVSs for reliable classification. Due to page limit Figures 2 and 3 present the coverage/accuracy graphs for the labor data and audiology data only. The graphs of NB_r are derived analogously and are present in the Figures. The coverage/accuracy graphs show two features of k-NBVSs. First, when the unanimous-voting rule is used the coverage rate of k-NBVSs is never 1.0. This is due the fact that k-NBVSs is a set of scoring classifiers that disagree on some part of the instance space X. Second, k-NBVSs are sensitive for parameter k: the bigger k the bigger is the size of k-NBVSs. Thus, when the unanimous-voting rule is used bigger k implies less classified instances; i.e., lower coverage rate. For example the parameter k for the labor data equals 0 and for the audiology data equals 6. Therefore the coverage rates of k-NBVSs for the labor data is bigger.

7 Conclusions

This paper has theoretical and practical contributions. The theoretical contributions are two. The first one is that we proved that the problem of k-version-space classification is a recursive problem that consists of the problem of $k-1$-version-space classification, the general k-consistency problem, and the general $k0$-consistency problem. Thus, the k-version-space classification can be (tractably) implemented as soon as we can (tractably) test for k-consistency and $k0$-consistency. In this respect our work is a continuation of [5] showing that the 0-version-space classification problem is equivalent to the 0-consistency problem. The second theoretical contribution is that we extended k-version spaces to multi-class classification. This is due to the consistency tests that can be applied to any class. This contrasts with the original formulation of k-version spaces proven and used for two-class classification only due to the Boolean nature of version-space representations [6–8].

The practical contributions are two. The first one consists of two approaches to designing consistency-test algorithms for any type of learning algorithms. The most important is the hypothesis-restrictive approach applicable for nonzero-consistent learning algorithms. The second practical contribution is that we introduced two implementations of k-consistency. They allow logical and probabilistic implementations of consistency-test algorithms, and thus of the k-version-space classification.

The theoretical and practical contributions convert k-version spaces to a meta framework applicable for any type of learning algorithms. The framework is practical for nonzero-consistent learning algorithms. More precisely it guarantees tractable k-version-space classification iff these algorithms are tractable.

Acknowledgements. The authors would like to thank the anonymous referees for the useful and detailed reviews.

References

1. Bosnic, Z., Kononenko, I.: An overview of advances in reliability estimation of individual predictions in machine learning. Intell. Data Anal. 13(2), 385–401 (2009)
2. Frank, A., Asuncion, A.: UCI machine learning repository (2010), `http://archive.ics.uci.edu/ml`
3. Hastie, T., Tibshirani, R., Friedman, J.: Statistical Learning: Data Mining, Inference, and Prediction. Springer, Heidelberg (2009)
4. Hirsh, H.: Polynomial-time learning with version spaces. In: Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-1992), pp. 117–122. AAAI Press, Menlo Park (1992)
5. Hirsh, H., Mishra, N., Pitt, L.: Version spaces and the consistency problem. Artificial Intelligence 156(2), 115–138 (2004)
6. Mitchell, T.: Version Spaces: An Approach to Concept Learning. Ph.D. thesis, Electrical Engineering Dept., Stanford University, Stanford, CA (1978)
7. Mitchell, T.: Generalization as search. Artificial Intelligence 18(2), 203–226 (1982)

8. Mitchell, T.: Machine Learning. McGraw-Hill, New York (1997)
9. Sebag, M.: Delaying the choice of bias: a disjunctive version space approach. In: Proceedings of the Thirteenth International Conference on Machine Learning (ICML-1996), pp. 444–452. Morgan Kaufmann, San Francisco (1996)
10. Sebag, M., Rouveirol, C.: Any-time relational reasoning: Resource-bounded induction and deduction through stochastic matching. Machine Learning 38(1-2), 41–62 (2000)
11. Shafer, G., Vovk, V.: A tutorial on conformal prediction. Journal of Machine Learning Research 9(3), 371–421 (2008)
12. Smirnov, E., van den Herik, H., Sprinkhuizen-Kuyper, I.: A unifying version-space representation. Ann. Math. Artif. Intell. 41(1), 47–76 (2004)
13. Smirnov, E., Sprinkhuizen-Kuyper, I., Nalbantov, G., Vanderlooy, S.: S.Vanderlooy: Version space support vector machines. In: Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI-2006), pp. 809–810. IOS Press, Amsterdam (2006)
14. Syropoulos, A.: Mathematics of multisets. In: Proceedings of the First Intrenational Workshop on Multiset Processing (MP-2000), pp. 347–358 (2000)
15. Tzikas, D., Kukar, M., Likas, A.: Transductive reliability estimation for kernel based classifiers. In: R. Berthold, M., Shawe-Taylor, J., Lavrač, N. (eds.) IDA 2007. LNCS, vol. 4723, pp. 37–47. Springer, Heidelberg (2007)
16. Vanderlooy, S., Sprinkhuizen-Kuyper, I., Smirnov, E., van den Herik, H.: The ROC isometrics approach to construct reliable classifiers. Intell. Data Anal. 13(1), 3–37 (2009)