

## Feature interval learning algorithms for classification

Aynur Dayanik\*

Department of Computer Engineering, Bilkent University, Bilkent 06800, Ankara, Turkey

### ARTICLE INFO

#### Article history:

Received 21 October 2009

Received in revised form 11 February 2010

Accepted 12 February 2010

Available online 19 February 2010

#### Keywords:

Classification learning

Inductive learning

Feature partitioning

Adaptive feature weights

### ABSTRACT

This paper presents *Feature Interval Learning* algorithms (FIL) which represent multi-concept descriptions in the form of disjoint feature intervals. The FIL algorithms are batch supervised inductive learning algorithms and use feature projections of the training instances to represent induced classification knowledge. The concept description is learned separately for each feature and is in the form of a set of disjoint intervals. The class of an unseen instance is determined by the weighted-majority voting of the feature predictions. The basic FIL algorithm is enhanced with adaptive interval and feature weight schemes in order to handle noisy and irrelevant features. The algorithms are empirically evaluated on twelve data sets from the UCI repository and are compared with  $k$ -NN,  $k$ -NNFP, and NBC classification algorithms. The experiments demonstrate that the FIL algorithms are robust to irrelevant features and missing feature values, achieve accuracy comparable to the best of the existing algorithms with significantly less average running times.

© 2010 Elsevier B.V. All rights reserved.

### 1. Introduction

Inductive learning derives concept descriptions from examples. Given a set of training examples, each of which is described by feature values and labeled with a class name, an inductive learning algorithm learns a mapping from feature values to class labels by forming a generalization of the training examples. This process, known as supervised inductive learning, is useful for real-world classification tasks in medical diagnosis, target marketing, fraud detection, and text classification.

In this paper, we are concerned with concept acquisition and classification learning tasks. Concept acquisition is the task of learning a description of a given concept from a set of examples and counterexamples of that concept [1–3]. The classification task is to predict correctly the class label of an unseen test example from a set of labeled training examples or from some classification knowledge learned by a concept acquisition algorithm. Many supervised learning algorithms have been developed to perform classification task [4–8].

A good supervised inductive learning algorithm achieves high predictive accuracy with short training and classification times, is robust to irrelevant features, noisy and missing feature values, and produces understandable concept descriptions. The  $k$ -Nearest Neighbor ( $k$ -NN) algorithm is one of the most successful algorithms in machine learning with high predictive ability, but is slow in classifying a new example because it computes distances be-

tween the test example and training instances to find the  $k$  nearest neighbors. There has been some attempts to generalize the nearest neighbor algorithm to represent the training set in a more compact form to speed up the classification process. Salzberg [8,9] describes a family of learning algorithms based on nested generalized exemplars (NGE). Wettschereck and Dietterich [10] compared algorithms based on NGE with the  $k$ -NN algorithm and found that the  $k$ -NN algorithm is superior to those algorithms based on NGE. The  $k$ -NN algorithm is, however, known to be sensitive to irrelevant features [11]. Therefore, there is still a demand for fast and robust classification algorithms with high predictive power.

This paper presents new classification learning algorithms, which use an exemplar-based knowledge representation. We design and implement several batch supervised learning algorithms, called *Feature Interval Learning* (FIL) algorithms, which learn multi-concept descriptions in the form of disjoint feature intervals. The FIL algorithms generalize the feature projections of training examples to disjoint intervals on feature dimensions. In the FIL algorithms, each feature locally predicts the class label of the test example as the label of the feature interval into which the feature value of test example falls. The final prediction of the FIL algorithms is the label with the highest weighted total vote of all feature predictions.

To cope better with irrelevant features and noisy feature values, the basic FIL algorithm is enhanced with adaptive feature and interval weight schemes, and the enhanced algorithms are called FIL.I, FIL.F, and FIL.IF. Algorithm FIL.I assigns to each interval on every given feature an interval weight which equals the fraction of those falling into the interval among all training examples with

\* Tel.: +90 312 290 12 18; fax: +90 312 266 40 47.

E-mail address: [adayanik@cs.bilkent.edu.tr](mailto:adayanik@cs.bilkent.edu.tr)

the same class label as that of the interval. Algorithm FIL.F assigns to each feature a feature weight which is inversely proportional to the number of feature intervals, which turns out to be a good measure of the relevance of that feature to the classification problem. Algorithm FIL.IF assigns both an interval weight to each interval on every feature and a feature weight to every feature. As we report below, the weighted voting mechanisms in the FIL algorithms reduce the detrimental effects of irrelevant features in classification. In particular, the FIL.IF algorithm is robust to the presence of irrelevant features.

Real classification problems often involve missing feature values. Most classification systems solve this problem by filling in the missing feature values with the most likely value or a value found in general by using known feature values. Quinlan [12] compared most common approaches and concluded that none of them is uniformly superior to others. The FIL algorithms handle missing feature values in a natural way. Because they treat features separately, they simply ignore missing feature values. This approach is also used by naive Bayesian classifier (NBC), even though recently Chen et al. incorporated gain ratio data completion method into Bayes classifier [13]. We show, however, that the FIL algorithms are more robust to missing feature values than NBC.

We compare the FIL algorithms to the  $k$ -NN,  $k$ -NNFP, and NBC algorithms. The  $k$ -Nearest Neighbor ( $k$ -NN) algorithm [14,15] is a well-known exemplar-based learning method and gives excellent results on many real-world induction tasks [16,4,17]. Akkus and Guvenir [18] proposed a classification algorithm based on  $k$ -NN, called  $k$  Nearest Neighbor on Feature Projections ( $k$ -NNFP). The  $k$ -NNFP algorithm classifies a test example on a feature according to the  $k$  nearest neighbors of the test example on that feature. The final classification of the test example is determined by a majority voting among the classifications of features. Like FIL algorithms, the  $k$ -NNFP and NBC algorithms similarly learn separate generalizations on features which are then suitably combined. Both algorithms are fast and based on the feature projection knowledge representations.

Because the FIL algorithms treat features separately, they do not use any distance metric among the instances for predictions unlike distance-based algorithms like  $k$ -NN. Therefore, the FIL algorithms classify a new instance faster than the distance-based classification algorithms. As we report below, while the FIL algorithms achieve accuracies comparable to the best of the  $k$ -NN,  $k$ -NNFP, and NBC algorithms, average running times of the FIL algorithms are significantly less than the average running times of all of those algorithms.

Because the concept descriptions of the FIL algorithms are in the form of disjoint feature intervals, they are also easy to understand by humans. Moreover, feature intervals provide information about the roles played by the features in the classification task.

The paper is structured as follows. Section 2 introduces the basic FIL algorithm. The construction of feature intervals on each feature dimension and the classification process are illustrated through examples. Several enhancements of the basic FIL algorithm are also described. Section 3 presents the complexity analysis and empirical evaluation of the FIL algorithms on real-world

data sets. The FIL algorithms are compared to the  $k$ -NN,  $k$ -NNFP, and NBC classification algorithms in the presence of irrelevant features and missing feature values. In Section 4, the existing concept learning models are reviewed. Section 5 concludes and discusses future research directions.

## 2. Feature interval learning algorithms

This section develops several batch learning algorithms, which we call *Feature Interval Learning* (FIL) algorithms. These algorithms learn the classification knowledge in the form of disjoint feature intervals on separate features, which give a compact representation of the training data, reduce storage requirements, lead to fast classifications, and increase the ability of the user to understand better the decisions made by the classifier.

We describe and discuss the FIL, FIL.F, FIL.I, FIL.IF algorithms after we give some basic definitions. The FIL.F, FIL.I, and FIL.IF algorithms are obtained from the basic FIL algorithm by incorporating into it adaptive feature and interval weighing schemes.

A **point interval** on a feature dimension consists of a single feature value observed in the training set, the classes represented by the point interval, and their relative representativeness values. It is identified with

$$\langle [v, v], (C_1, \dots, C_k), (r_1, \dots, r_k) \rangle,$$

where the location  $v$  of the point interval is expressed as point interval  $[v, v]$ , the classes  $C_1, \dots, C_k$  represented by the point interval are the labels of all classes from which there is at least one training example with feature value  $v$ , and the relative representativeness values  $r_1, \dots, r_k$  are the fractions of class  $C_1, \dots, C_k$  training examples with feature values at  $v$ , respectively. Similarly, the *representativeness counts of a point interval* are defined as the number of training examples with feature values at  $v$  from the classes  $C_1, \dots, C_k$  that the point interval represents.

A **single-class point interval** has in its list of represented classes exactly one class label, which is also the label of the single-class point interval. A **multi-class point interval** has at least two distinct class labels in its list of represented classes.

Fig. 1(a) presents a single-class point interval  $\langle [x, x], C_1, r \rangle$ , which is located at  $x$ , and contains class  $C_1$  training examples and has relative representativeness value  $r$ . Fig. 1(b) illustrates a multi-class point interval,  $\langle [x, x], (C_1, C_2, C_3), (r_1, r_2, r_3) \rangle$ , which contains training examples whose feature values on  $f$  are  $x$ , and which have the class labels  $C_1, C_2$ , and  $C_3$  with relative representativeness values  $r_1, r_2$ , and  $r_3$ , respectively.

A **range interval** on a feature dimension is the smallest closed connected interval obtained by merging neighboring single-class point intervals with the same class labels. It is identified with

$$\langle [l, u], C, r \rangle,$$

where *lower*  $l$  and *upper*  $u$  bounds of an interval are the minimum and maximum feature values which fall into the range interval, respectively, *class label*  $C$  is the common class label of the training examples with feature values in the range  $[l, u]$ , the fraction of which gives the *relative representativeness value*  $r$ . The range interval

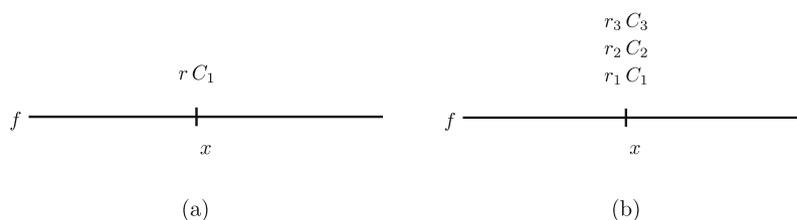


Fig. 1. Examples of (a) a single-class point interval and (b) a multi-class point interval.

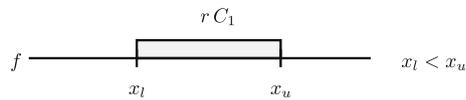


Fig. 2. An example of a range interval.

is said to represent the class  $C$ , and the *representativeness count* of the range interval is defined as the number of training examples from the class it represents with feature values in the range  $[l, u]$ . Fig. 2 illustrates a range interval,  $\langle [x_l, x_u], C_1, r \rangle$ , which contains class  $C_1$  training instances whose feature values on  $f$  are in the range  $[x_l, x_u]$ , and has the relative representativeness value  $r$ . Range intervals contain at least two training instances with different feature values, but all of the training examples whose feature values are in the range  $[x_l, x_u]$  carry the same class label.

### 2.1. The description of the FIL algorithms

In this section, the training and classification processes of the FIL algorithms are explained and illustrated on an example.

#### 2.1.1. The FIL algorithm

In the training phase of the basic FIL algorithm (FIL), the learning task is to construct disjoint feature intervals. All training instances are processed at the same time. Feature intervals on each feature dimension are constructed through generalization. The FIL algorithm can handle both continuous (linear) and nominal valued features. However, only the training examples on linear features are generalized into disjoint intervals. In the preclassification phase of the classification task, each feature predicts that the class of the test example is the same as the label of the interval into which the example's feature value falls. The instance is finally labeled with the class receiving the majority of the preclassification votes from the features.

*Training in the FIL algorithm.* Fig. 3 outlines the training process of the FIL algorithm. The input to the FIL algorithm is training examples, each of which is represented as a vector  $\mathbf{x} = \langle x_1, \dots, x_n, C \rangle$  where  $x_1, \dots, x_n$  are the feature values on  $f_1, \dots, f_n$ , and  $C$  is the class label of the example. Thus, the dimension of an example vector  $\mathbf{x}$  is the number of features  $n$  plus one.

The FIL algorithm learns in batch mode and processes all of the training examples concurrently. For each feature, all training instances are firstly sorted according to their feature values, and a point interval is constructed at each observed feature value. Missing feature values are ignored. For linear features, the FIL algorithm generalizes the point intervals into range intervals by merging all of the neighboring single-class point intervals with the same class labels. However, multi-class point intervals are left alone, and at the end of the training process, they are converted to single-class point intervals by selecting the class with highest relative representativeness value as their class labels. The feature intervals are always disjoint. Nominal features have only (possibly multi-class) point intervals, because nominal values do not always have a natural ordering.

Let us give an example to illustrate the training process of the FIL algorithm. Table 1 shows a sample training set, where training instances are represented as vectors of feature values and class labels. The sample training set has 18 examples described with three linear features and there are three classes:  $C_1, C_2$ , and  $C_3$ . Feature projections on each feature dimension are displayed in Fig. 4, which corresponds to the initial presentation of all training instances and storage of only the feature projections (sorted for linear features) in memory. From this knowledge, point intervals are constructed, and their lower and upper bounds, represented class labels and their relative representativeness values are stored. Because all features are linear, neighboring single-class point intervals of the same classes are merged into range intervals. The relative representativeness values of the range and single-class point

---

```

train(Training Set)
begin
  for each feature  $f$  /* on each feature dimension */
    sortTrainingData( $f$ , Training Set)
    construct-intervals( $f$ , Training Set)
  end.
  construct-intervals( $f$ , Training Set)
  begin
    for each training instance  $i$ 
      intervals = initialize-point-intervals( $f, i$ )
      generalize-point-intervals( $f$ , intervals)
    end.
    generalize-point-intervals( $f$ , intervals)
    begin
      for each consecutive interval pair in intervals
        /* update lower, upper and representativeness values */
        if  $f$  is linear then
          if they are range and/or single-class point intervals with the same class labels then
            merge them into a range interval
          /* if  $f$  is a nominal feature, no generalization is done */
          /* normalization of class distributions among intervals */
        for each interval in intervals and each class represented by the interval
          relative representativeness value of class =  $\frac{\text{representativeness count of interval for class}}{\text{total count of class in training set}}$ 
        for each multi-class point interval in intervals
          delete from the represented class list and relative representativeness value list
            all but the one with the highest relative representativeness value
        end.
    end.
  end.

```

---

Fig. 3. The training process of the FIL algorithm.

**Table 1**  
A sample training set.

(1, 10, 7, C <sub>1</sub> ),	(10, 17, 16, C <sub>2</sub> ),	(4, 17, 2, C <sub>3</sub> ),
(3, 12, 7, C <sub>1</sub> ),	(10, 17, 18, C <sub>2</sub> ),	(4, 17, 3, C <sub>3</sub> ),
(4, 10, 10, C <sub>1</sub> ),	(4, 7, 7, C <sub>2</sub> ),	(4, 17, 1, C <sub>3</sub> ),
(4, 12, 10, C <sub>1</sub> ),	(6, 17, 7, C <sub>2</sub> ),	(4, 17, 4, C <sub>3</sub> ),
(4, 15, 10, C <sub>1</sub> ),	(6, 9, 13, C <sub>2</sub> ),	(6, 17, 4, C <sub>3</sub> ),
	(8, 17, 15, C <sub>2</sub> ),	(6, 19, 1, C <sub>3</sub> ),
		(9, 19, 4, C <sub>3</sub> ),

intervals are set to the fraction of those falling into the intervals among all training examples with the same class labels as the class of the intervals. The resulting feature intervals are displayed in Fig. 5. For example, the feature projections on the first feature dimension form on  $f_1$  dimension the feature intervals.

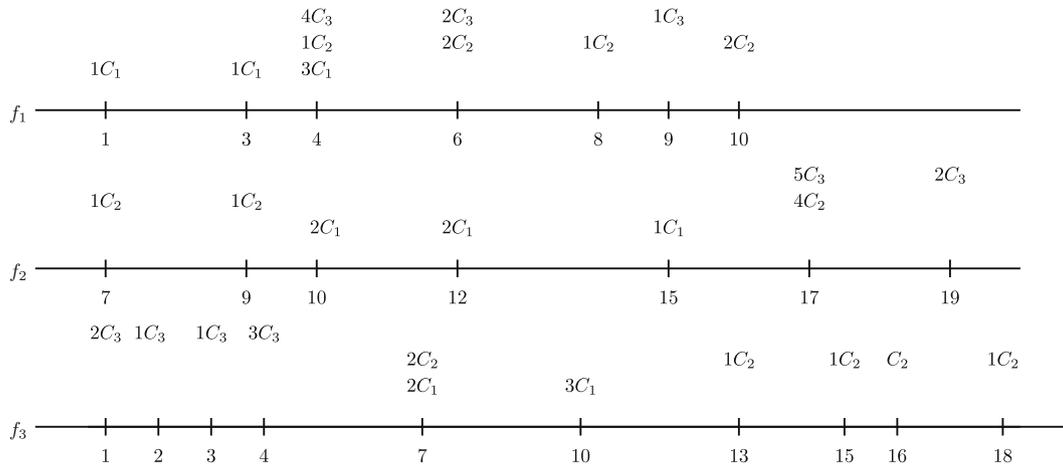
$\langle [1, 3], C_1, 2/5 \rangle, \langle [4, 4], (C_1, \cancel{4/4}), (C_2, \cancel{1/1}), (C_3, \cancel{1/1}) \rangle, \langle (3/5, 1/16), (A/17) \rangle,$   
 $\langle [6, 6], (C_2, \cancel{6/6}), (2/6, \cancel{2/17}) \rangle, \langle [8, 8], C_2, 1/6 \rangle,$   
 $\langle [9, 9], C_3, 1/7 \rangle, \langle [10, 10], C_2, 2/6 \rangle.$

The only range interval on  $f_1$  is the first interval because only it contains adjacent single-class points with the same class labels whereas the others are either multi-class point intervals or adjacent single-class point intervals with different class labels.

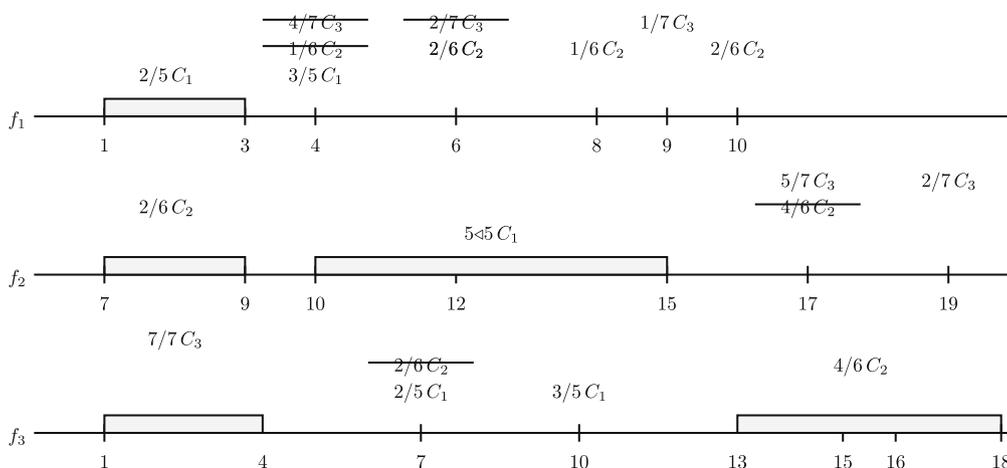
**Classification in the FIL Algorithm.** The classification process of the FIL algorithm is outlined in Fig. 6. The output of the training process of the FIL algorithm is the concept descriptions learned in the form of disjoint feature intervals.

The classification in the FIL algorithm is based on majority voting taken among the individual predictions of features. Initially, the votes of classes are equal to zero. The classification on a feature starts with a search process on that feature dimension. If the feature value of the test example is unknown, then feature does not make a prediction. If the feature value of the test example is contained in a range interval, then the feature prediction is the class label of that range interval. If it falls in a multi-class point interval, then the class label of the point interval with the maximum relative representativeness value is predicted. If the feature value does not fall into any range or point intervals, then that feature does not make any prediction and does not participate in the final voting. If none of feature dimensions give any predictions, then classification decision will be undetermined. After all features make their predictions, their votes are summed up, and the class label which receives the maximum total votes is the classification decision for the test example. Ties between classes are broken at random.

Each feature predicts only one class. The basic FIL algorithm allows the features to have different relevance levels, which are incorporated to the classification decision by means of user-specified feature weights. If feature weights are unspecified, then the



**Fig. 4.** The feature projections of the sample training examples shown in Table 1 on each feature dimension.



**Fig. 5.** Construction of feature intervals by the FIL algorithm using the feature projections of the sample training set from Fig. 4. Gray class labels and their relative representative values are deleted from the multi-class point intervals.

```

classify(test)
begin
  for each class c
    vote[c] = 0
  for each feature f
    interval = search-interval(f, test_f)
    /* each feature contributes proportional to its weight */
    if class of interval ≠ UNDETERMINED then
      vote[class of interval] = vote[class of interval] + weight[f];
  prediction = first class
  for each class c
    if vote[c] > vote[prediction] then
      prediction = c
  if vote[prediction] = 0 then
    prediction = UNDETERMINED /* all features make no prediction */
  return prediction
end.
search-interval(f, value)
begin
  if value on f is in a range or point interval then
    return interval containing value
  else /* no interval exists for that value */
    return UNDETERMINED
end.

```

Fig. 6. The classification process of the FIL algorithm.

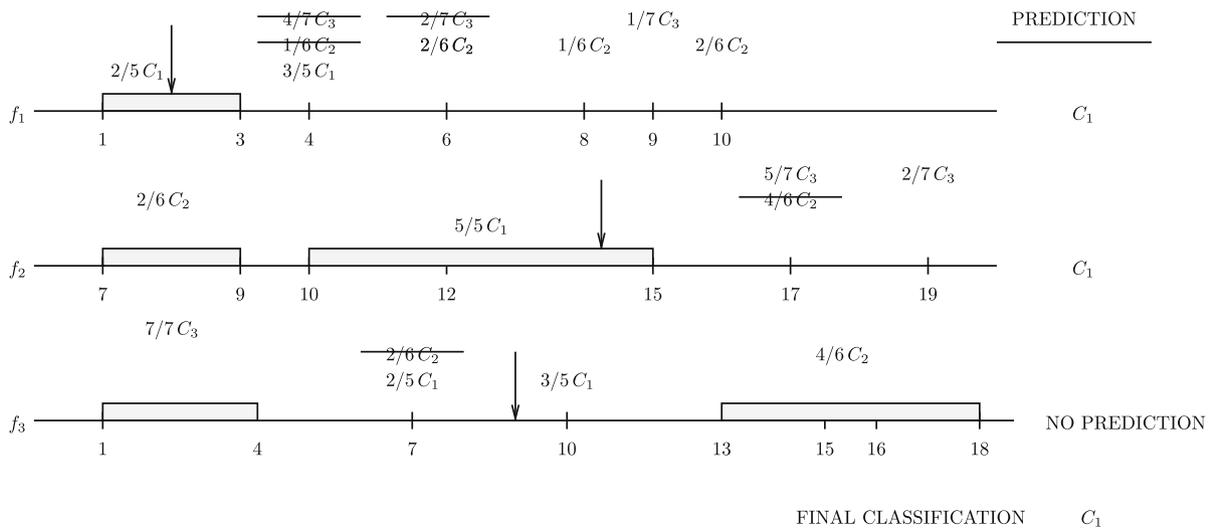


Fig. 7. An example for classification by the FIL algorithm.

features are assumed to have equal weights, and each feature has the same voting power in the final class prediction.

Let us illustrate the classification process of the FIL algorithm by classifying the sample test example  $\langle 2,14,9,C_1 \rangle$  according to the concept descriptions learned by the FIL algorithm in the training phase as shown in Fig. 5. Feature values of this test example are indicated in Fig. 7 by arrows. Each feature makes a preclassification of the test example. On the first feature dimension, the feature value, 2, falls into the interval  $\langle [1,3], C_1, 2/5 \rangle$  with class  $C_1$ . Therefore, it predicts that the class of the test instance is  $C_1$ . The result of preclassification of the second feature is similarly class  $C_1$  because the second feature value, 14, falls into the interval  $\langle [10,15], 5/5, C_1 \rangle$ . The third feature makes no prediction because the third feature va-

lue, 9, does not belong to any range or point intervals. The vote vector for this test instance becomes  $\langle 2,0,0 \rangle$ . The class  $C_1$  gets 2 votes whereas  $C_2$  and  $C_3$  get no votes. Hence,  $C_1$  receives the maximum vote and is the final class prediction. Since the true class value of the test example is  $C_1$ , the final prediction is a correct classification. For this example, equal feature weights were assumed.

On the same example, let us also illustrate the significance of using relative representativeness values instead of absolute representativeness counts. In the training set, there are three training instances of class  $C_1$  and four training instances of class  $C_3$ , with feature values at 4 on feature  $f_1$ . Therefore, the relative representativeness values of the multi-class point interval at feature value 4 for the represented classes  $C_1$  and  $C_3$  are  $3/5$  and  $4/7$ , respectively.

The relative representativeness value for  $C_1$  is greater than for  $C_3$ , even though the absolute representativeness count for  $C_3$  is greater than that for  $C_1$ .

Preclassifications based on relative representativeness values of multi-class point intervals enable us to detect easily the associations of those classes with classes underrepresented in the training data. Relative representativeness values can also be thought as the normalized class distributions on feature dimensions and are more informative than absolute representativeness counts, especially if class imbalance is present in the training set.

### 2.1.2. The FIL.F algorithm

The FIL algorithm does not attempt to learn the relevance of individual features and expects the feature weights from the analyst. However, it has an internal knowledge about the relevance of each feature. The number of feature intervals constructed by the FIL algorithm can be thought as a measure of that feature's relevance to the classification task. Our observation is that an irrelevant feature tends to have a relatively large number of intervals. Therefore, weighing each feature inversely proportional to the total number of intervals constructed by the FIL algorithm on that feature can reflect relative importance of that feature. Based on this observation, we propose the FIL.F algorithm which is the same as the FIL algorithm with the important exceptions that, (i) in the training phase, the new algorithm computes feature weights as the reciprocals of the number of intervals constructed by the old algorithm on the feature dimensions, and (ii) in the classification process, the new algorithm uses those feature weights to calculate the total weighted votes of features for different classes. The training process of the FIL.F algorithm is shown in Fig. 8.

### 2.1.3. The FIL.I algorithm

In the FIL algorithm, at each multi-class point interval, only the represented class label with the largest relative representativeness value is retrieved when a preclassification has to be made. Therefore, significant amount of memory can be saved by deleting from the lists of represented classes and their relative representativeness values of each multi-class point interval those class labels with lower relative representativeness values, which can also be seen as the pruning of the presumably noisy point intervals. In order to compensate for information loss due to pruning of underrepresented class labels with high relative representativeness value, we assign to each multi-class point interval an interval weight. This weight is found by dividing the ratio of the absolute difference between the largest two representativeness counts to the total number of representativeness counts at that feature value by the total count of the predicted class in the training set. The weight of a range interval is the relative representativeness value of that interval.

---

```

train(Training Set)
begin
  for each feature  $f$  /* on each feature dimension */
    sortTrainingData( $f$ , Training Set)
    construct-intervals( $f$ , Training Set) /* from Figure 3 */
    compute-feature-weights( $f$ , intervals)
  end.
  compute-feature-weights( $f$ , intervals)
begin
  feature weight of  $f = \frac{1}{\text{number of intervals on feature } f}$ 
end.

```

---

Fig. 8. The training process of the FIL.F algorithm.

To illustrate the pruning and weighing at a multi-class point, imagine that at some feature value  $v$  we have point interval

$$\left\langle [v, v], (C_1, C_2, C_3), \left( \frac{50}{100}, \frac{49}{100}, \frac{2}{100} \right) \right\rangle.$$

If we simply delete the last two classes, then we may lose the information that the prediction of feature value  $v$  is not dependable because classes  $C_1$  and  $C_2$  are almost equally likely to happen.

An interval in the new algorithm, called FIL.I, is represented as

$\langle$ [lower bound, upper bound], class label, weight of interval $\rangle$ .

The weight of the interval is the vote of the interval in the final classification process. The training process of the FIL.I algorithm is shown in Fig. 9. The classification process of the FIL.I algorithm is identical to the classification process of the FIL algorithm shown in Fig. 6 except that the vote calculation in the box of that figure is replaced by

$$\begin{aligned} \text{vote}[\text{class of interval}] &= \text{vote}[\text{class of interval}] \\ &+ \text{weight of interval.} \end{aligned}$$

Fig. 10 shows the feature intervals constructed by the FIL.I algorithm for the sample training set given in Fig. 4. Fig. 11 illustrates the classification process of the FIL.I algorithm on the test example  $\langle 8, 18, 3, C_3 \rangle$ . The feature value on  $f_1$  falls into the interval  $\langle [8, 8], C_2, 1/6 \rangle$ . The weight of this interval becomes  $1/6$  because there is only one in six training instances of class  $C_2$  whose feature value on  $f_1$  is 8. Therefore, feature  $f_1$  predicts  $C_2$  and has weight  $1/6$ , feature  $f_2$  makes no prediction, and feature  $f_3$  predicts  $C_3$  and has weight 1, because the feature values on  $f_3$  of all training instances with class label  $C_3$  are in that same interval. The vote vector will be  $\langle 0, 1/6, 1 \rangle$ . The final classification of the test example is  $C_3$ . Hence, the test example is correctly classified by the FIL.I algorithm.

### 2.1.4. The FIL.IF algorithm

The FIL.I algorithm uses interval weights as feature votes. However, features may not always be equally important for the classification process. The new FIL.IF algorithm calculates both interval weights as the FIL.I algorithm and feature weights as the FIL.F algorithm. Hence, range and single-class point intervals are assigned weights equal to their highest relative representativeness values, multi-class point intervals are assigned weights determined by the highest two relative representativeness values, and features get weights which are the reciprocals of the number of intervals on those feature dimensions.

The training process of the FIL.IF algorithm is shown in Fig. 12. In the classification phase, the vote of each feature is calculated by multiplying the weight of the interval containing the feature value of the test example with the feature weight. The classification process of the FIL.IF algorithm is identical to the classification process of the FIL algorithm shown in Fig. 6 except that the vote calculation in the box of that figure is replaced by

$$\begin{aligned} \text{vote}[\text{class of interval}] &= \text{vote}[\text{class of interval}] \\ &+ \text{weight of interval} \times \text{weight}[f]. \end{aligned}$$

## 3. Evaluation of the FIL algorithms

In this section, both complexity analysis and empirical evaluations of the FIL algorithms are given. The FIL algorithms are compared to the  $k$ -NNFP, NBC, and  $k$ -NN algorithms.

---

```

train(Training Set) /* from Figure 3 */
construct-intervals(Training Set)
begin
  for each feature f
    for each training instance i
      initialize-point-intervals(f, i)
      generalize-point-intervals(f)
end.
generalize-point-intervals(f, intervals)
begin
  for each consecutive interval pair in intervals
    /* update lower, upper and representativeness values */
    if f is linear then
      if they are range and/or single-class point intervals with the same class labels then
        merge them into a range interval
        /* if f is a nominal feature, no generalization is done */
        /* normalization of class distributions among intervals */
        for each interval in intervals and each class represented by the interval
          relative representativeness value of class =  $\frac{\text{representativeness count of interval for class}}{\text{total count of class in training set}}$ 
        compute-interval-weights(f)
        for each multi-class point interval in intervals
          delete from the represented class list and relative representativeness value list
            all but the one with the highest relative representativeness value
end.
compute-interval-weights(f)
begin
  for each interval
    if range interval or single-class point interval then
      weight of interval =  $\frac{\text{representativeness count of interval}}{\text{total count of class of interval in training set}}$ 
    else /* multi-class point intervals */
      find two classes having the highest representativeness counts
      weight of interval =  $\frac{\text{difference between two maximum representativeness counts of interval}}{\text{total count of interval}} \cdot \frac{\text{total count of class with the highest relative representativeness count in training set}}{\text{total count of class with the highest relative representativeness count in training set}}$ 
end.

```

---

Fig. 9. The training process of the FIL algorithm.

### 3.1. Complexity analysis

We analyze the FIL algorithms in terms of space and time complexities. Time complexity analysis is presented for the training process and for the classification of a single test example.

#### 3.1.1. Space complexity analysis

In the training phase of the FIL algorithms, disjoint feature intervals for concept descriptions are constructed on each feature dimension. The space required for training with  $m$  instances on a domain with  $n$  features is proportional to  $mn$  at worst case. However, on average, it should be less than  $O(mn)$  because feature intervals may contain more than one feature value. If the average number of intervals constructed on a feature dimension is  $i \leq m$ , then the average space complexity of the FIL algorithms will be  $O(in)$ . The  $k$ -NN algorithm stores all instances in memory as con-

junctions of feature values and the  $k$ -NNFP algorithm stores them as feature projections. Therefore, the space required by those algorithms for training with  $m$  instances on a domain with  $n$  features is proportional to  $mn$ . If feature intervals contain many feature values, the space requirement of the FIL algorithms will be less than those of the  $k$ -NN and  $k$ -NNFP algorithms.

#### 3.1.2. Time complexity analysis of training

For a data set with  $m$  training instances, feature projections on a feature dimension are sorted with time complexity  $O(m \log m)$ . Therefore, sorting all feature values has time complexity  $O(mn \log m)$  for  $n$  features. Disjoint feature intervals are constructed by examining those sorted feature projections on feature dimensions with time complexity  $O(nm)$ . Therefore, the training time complexity of the FIL algorithms is  $O(nm \log m + nm) = O(nm \log m)$ . The training time complexity of the  $k$ -NNFP and the  $k$ -NN algorithms is  $O(n \cdot m \cdot \log m)$ , and  $O(n \cdot m)$ , respectively [18].

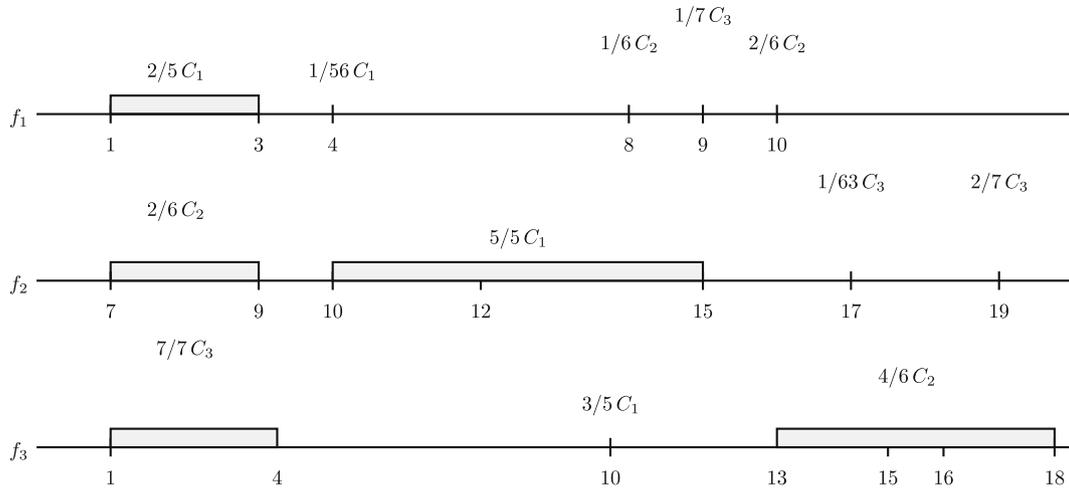


Fig. 10. Construction of feature intervals by the FIL algorithm.

TEST EXAMPLE:  $\langle 8, 18, 3, C_3 \rangle$

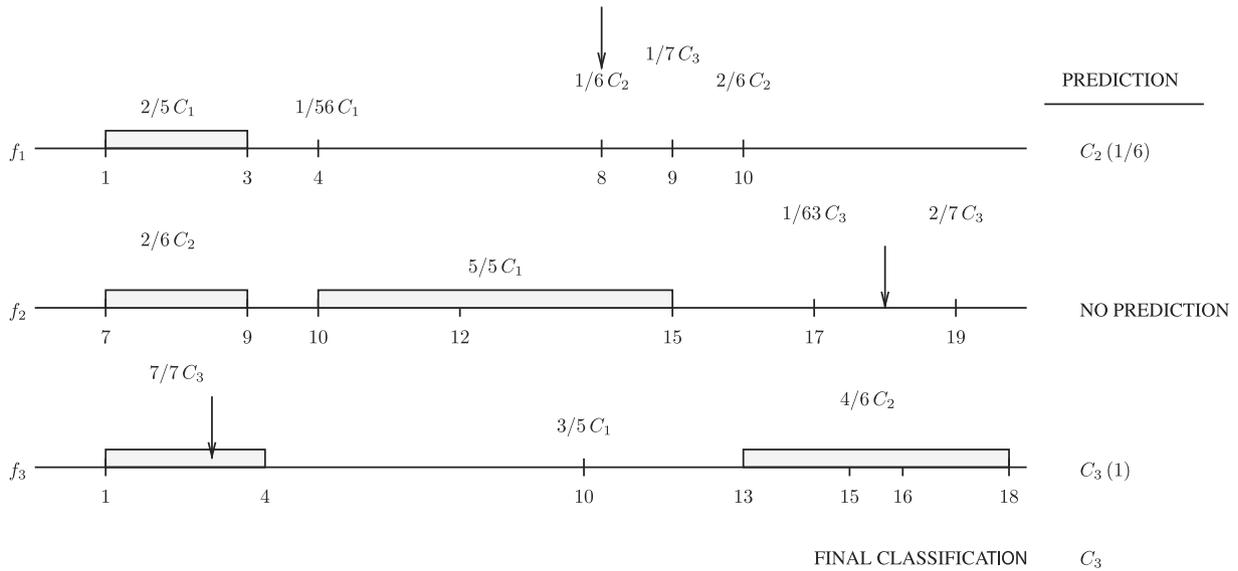


Fig. 11. An example of classification by the FIL algorithm.

```

train(Training Set)
begin
  for each feature f /* on each feature dimension */
    sortTrainingData(f, Training Set)
    construct-intervals(f, Training Set) /* from Figure 9 */
    compute-feature-weights(f, intervals) /* from Figure 8 */
end.
    
```

Fig. 12. The training process of the FIL algorithm.

### 3.1.3. Time complexity of a single classification

During the preclassification, the function *search-interval*(*f*, *value*) finds the interval containing feature value of the test instance on the feature dimension *f* by a binary search and determines the prediction of that feature. The number of intervals on a feature dimension is at most equal to the number of training instances *m*. Hence, the worst case time complexity of the search process is  $O(\log m)$  for a feature. Since the final classification is based on the predictions of all features, single instance classification time

complexity of the FIL algorithms is  $O(n \log m)$ . The classification time complexity of the *k*-NNFP algorithm is  $O(n \log m)$  if  $m \gg k$  [18], in which case the classification time complexities of the *k*-NNFP and FIL algorithms are the same. To classify a test example, the *k*-NN algorithm computes its distance to *m* training instances on *n* dimensions. Therefore, the classification time complexity of the *k*-NN algorithm is  $O(nm)$ . Time complexity analysis of the FIL algorithms indicate that these algorithms classify unseen examples much faster than the *k*-NN algorithm.

### 3.2. Empirical evaluation of the FIL algorithms

In this section, we evaluate the FIL algorithms on real-world datasets from the UCI-Repository [19], which is commonly used in machine learning research.

Improved performance is the major aim of learning algorithms [20]. For supervised concept learning tasks, the most commonly used performance metric is the classification accuracy, which is the percentage of correctly classified instances over all test instances.

To measure the accuracy of an algorithm, we use cross-validation technique. In  $k$ -fold cross-validation, the data set is partitioned into  $k$  mutually disjoint subsets with equal cardinality. The  $k-1$  of those sets are combined to form a training set, and the  $k$ th set becomes the test set. This process is repeated for each of  $k$  subsets. The classification accuracy is measured as the average classification accuracy on all of the test sets. The union of the all test sets equals the whole dataset. This is called  $k$ -fold cross-validation.

3.2.1. Experiments with real-world datasets

For empirical evaluations of the FIL algorithms, 12 real-world data sets from the UCI-Repository [19] are used. Table 2 shows for each data set its name, the number of instances, features, linear features, classes, the percentage of the unknown attribute values, and the baseline classification accuracy, which is obtained by predicting the class of any test instance as the most frequent class in the data set. These data sets are used to compare the performances of FIL, FIL.F, FIL.I, FIL.IF,  $k$ -NNFP, NBC, and  $k$ -NN algorithms.

Table 3 reports the classification accuracies of the FIL, FIL.F, FIL.I, FIL.IF,  $k$ -NNFP, NBC, and  $k$ -NN algorithms which are obtained by averaging the accuracies over five repetitions of 5-fold cross-validations. We take  $k=5$  in those experiments because 5-NN and 5-NNFP algorithms give the best accuracies.

For each data set, the best classification accuracy is shown in boldface, and the best classification accuracy of all feature projection based algorithms (namely, FIL, FIL.F, FIL.I, FIL.IF,  $k$ -NNFP, and NBC) is circled with a solid box. Among all FIL algorithms, FIL.IF

performs better than others on almost all data sets. To assess the statistical significance of the differences between the accuracies of FIL.IF and other algorithms on every data set, we perform paired  $t$ -tests. If the difference between the accuracies is found statistically significant at 0.05, 0.01, or 0.001 levels, then this is indicated, respectively, by  $1\pm$ ,  $2\pm$ , or  $3\pm$  in the superscript attached to the accuracy of the algorithm, where  $+$  (resp.  $-$ ) means FIL.IF is better (resp. worse) than the algorithm.

According to paired  $t$ -test results, FIL.IF outperforms FIL on 11 data sets and FIL.F on ten data sets. It underperforms FIL and FIL.F on only one data set, *bcancerw*, all features of which consist of multi-class point intervals, whose feature interval weights turn out to be uninformative. FIL.IF outperforms FIL.I on five data sets, *cleveland*, *horse*, *hungarian*, *ionosphere*, *iris*, and underperforms on only one data set, *wine*, while the differences between the accuracies of FIL.IF and FIL.I on the remaining six data sets, *bcancerw*, *diabetes*, *glass*, *liver*, *musk*, *new-thyroid* is statistically insignificant at 0.05 level. We may therefore conclude that FIL.IF performs at least as good as or better than FIL.I in all data sets but one. This also implies that FIL.IF accuracy is either the largest or indistinguishable from the largest at 0.05 level of statistical significance in seven out of 12 data sets, *cleveland*, *hungarian*, *ionosphere*, *iris*, *liver*, *musk*, and *new-thyroid*, which are indicated by solid and dashed boxes in the FIL.IF column of Table 3.

FIL.IF outperforms  $k$ -NNFP on seven data sets, *cleveland*, *diabetes*, *glass*, *horse*, *hungarian*, *ionosphere*, *musk*, *new-thyroid*, and NBC on five data sets, *cleveland*, *hungarian*, *ionosphere*, *new-thyroid*,

**Table 2**  
Properties of twelve real-world datasets from UCI-Repository.

Dataset	Size	# of features	# of linear features	# of classes	Unknown values(%)	Baseline accuracy(%)
bcancerw	699	10	10	2	0.25	66
cleveland	303	13	6	2	0	54
diabetes	768	8	8	2	0	65
glass	214	9	9	6	0	36
horse	368	22	7	2	24	63
hungarian	294	13	6	2	0	64
ionosphere	351	34	34	2	0	64
iris	150	4	4	3	0	33
liver	345	6	6	2	0	58
musk	476	166	166	2	0	57
new-thyroid	215	5	5	3	0	70
wine	178	13	13	2	0	40

**Table 3**  
The percentage accuracy results of FIL, FIL.F, FIL.I, FIL.IF,  $k$ -NNFP, NBC, and  $k$ -NN algorithms are listed. The best overall results are shown in boldface. The best results of all feature projection based algorithms are circled in solid boxes. Paired  $t$ -tests are performed on the differences between the accuracies of FIL.IF and other algorithms on each data set. A superscript  $^{m+}$  (resp.,  $^{n-}$ ) attached to the accuracy of an algorithm shows that FIL.IF accuracy is better (resp., worse) on that data set than the accuracy of that algorithm at 0.05, 0.01, or 0.001 levels of significance if  $n=1, 2$ , or  $3$ , respectively. No superscript means that the difference between the accuracies of those algorithms are statistically insignificant at 0.05 level. The last row counts for each algorithm the number of data sets on which the accuracy of FIL.IF is significantly better/worse than the accuracy of that algorithm. Finally, if FIL.IF result on a data set is statistically indifferent from the best overall result on the same data set at 0.05 level of significance, then this is indicated by a dashed box around the FIL.IF result.

Dataset	Feature-projection based algorithms						$k$ -NN	Baseline
	FIL	FIL.F	FIL.I	FIL.IF	$k$ -NNFP	NBC		
bcancerw	96.88 <sup>2-</sup>	96.88 <sup>2-</sup>	96.42	96.42	96.10	<b>97.40</b> <sup>3-</sup>	96.91 <sup>3-</sup>	66
cleveland	79.94 <sup>2+</sup>	81.79 <sup>1+</sup>	70.43 <sup>3+</sup>	<b>83.57</b>	78.56 <sup>1+</sup>	79.93 <sup>2+</sup>	83.56	54
diabetes	65.88 <sup>2+</sup>	59.71 <sup>3+</sup>	70.54	70.28	67.31 <sup>2+</sup>	<b>72.62</b> <sup>1-</sup>	<b>73.53</b> <sup>2-</sup>	65
glass	50.63 <sup>2+</sup>	16.81 <sup>3+</sup>	55.02	54.66	<b>63.74</b> <sup>2-</sup>	53.92	<b>64.55</b> <sup>3-</sup>	36
horse	72.50 <sup>3+</sup>	70.97 <sup>3+</sup>	73.60 <sup>3+</sup>	77.18	70.55 <sup>3+</sup>	<b>80.27</b> <sup>2-</sup>	<b>81.57</b> <sup>3-</sup>	63
hungarian	80.68 <sup>1+</sup>	81.36 <sup>1+</sup>	72.24 <sup>3+</sup>	<b>84.14</b>	76.14 <sup>3+</sup>	82.58 <sup>2+</sup>	81.62 <sup>1+</sup>	64
ionosphere	85.53 <sup>3+</sup>	83.30 <sup>3+</sup>	91.16 <sup>3+</sup>	<b>92.53</b>	87.58 <sup>3+</sup>	88.42 <sup>3+</sup>	85.01 <sup>3+</sup>	64
iris	89.33 <sup>3+</sup>	<b>94.26</b>	92.53 <sup>1+</sup>	<b>93.33</b>	92.79	92.93	<b>96.40</b> <sup>2-</sup>	33
liver	52.58 <sup>3+</sup>	57.10 <sup>3+</sup>	<b>61.85</b>	<b>61.27</b>	60.98	61.51	61.10	58
musk	77.86 <sup>2+</sup>	75.67 <sup>3+</sup>	<b>82.93</b>	<b>82.43</b>	77.39 <sup>3+</sup>	81.50	81.88	57
new-thyroid	90.60 <sup>3+</sup>	90.41 <sup>3+</sup>	<b>95.16</b>	<b>95.06</b>	90.60 <sup>3+</sup>	93.84 <sup>3+</sup>	94.13 <sup>2+</sup>	70
wine	87.06 <sup>3+</sup>	87.85 <sup>3+</sup>	<b>96.63</b> <sup>1-</sup>	95.62	95.38	93.48 <sup>1+</sup>	95.40	40
# times FIL.IF sig. better/worse	11/1	10/1	5/1	-	7/1	5/3	3/5	

wine, but underperforms *k*-NNFP on only one data set, *glass*, and NBC on three data sets, *bcancerw*, *diabetes*, *horse*. FIL.IF's classification accuracy is statistically indistinguishable from those of *k*-NNFP and NBC on the remaining four data sets at 0.05 level of significance. The 54.66%–63.74% = –9.08% gap between the accuracies of FIL.IF and *k*-NNFP algorithms on *glass* may look large, but is statistically significant only at 0.01 level; compare this with 77.18%–70.55% = 6.63% gap on *horse*, 84.14%–76.14% = 8% gap on *hungarian*, 92.53%–87.58% = 4.95% gap on *ionosphere*, 82.43%–77.39% = 5.04% gap on *musk*, and 95.06%–90.60% = 4.46% gap on *new-thyroid*, all of which are statistically more significant at 0.001 level and suggest that FIL.IF perform better than *k*-NNFP in overall. The gap on *glass* data set may be attributed to the overwhelming number of multi-class point intervals on features, which are generalized better by the distribution than the majority of the nearest neighbors.

Finally, although *k*-NN is capable of capturing the interactions between the features, it outperforms feature projection based FIL.IF algorithm on only five data sets, *bcancerw*, *diabetes*, *glass*, *horse*, *iris*, but performs worse than FIL.IF on three data sets, *hungarian*, *ionosphere*, *new-thyroid*, while the accuracies of two algorithms on the remaining four data sets, *cleveland*, *liver*, *musk*, *wine*, are statistically indifferent at 0.05 level of significance. However, because of the *k*-NN's large time complexity, the *k*-NN algorithm cannot be an alternative on large data sets to FIL-IF, which offers both competitive classification accuracies and fast classification times.

On the left of Fig. 13, the boxplots of the classification accuracies of learning algorithms on 12 data sets reported in Table 3 show that FIL.IF has higher median accuracy and lower variation than those of FIL.I and *k*-NNFP. This confirms the overall superiority of FIL.IF over FIL.I and *k*-NNFP. In fact, FIL.IF median accuracy is higher than the median accuracies of all other algorithms, and the variability of FIL.IF accuracies is slightly more than, but still is comparable to those of NBC and *k*-NN.

Table 4 shows the training and classification (test) running times of the FIL, FIL.F, FIL.I, FIL.IF, *k*-NNFP, NBC, and *k*-NN algorithms; see also the boxplots on the right in Fig. 13 for the logarithms of the running times. The FIL algorithms have very similar training and classification times, and are reported under single column named FIL. Note that the classification running times of the FIL algorithms are less than those of the other algorithms, which is consistent with the time complexity analysis of Section 3.1. Especially, the classification running time of the *k*-NN algorithm is significantly higher than those of all other feature projection based algorithms.

### 3.2.2. Evaluation in the presence of irrelevant features

Our hypothesis is that FIL.F and FIL.IF algorithms are robust to irrelevant features. To test it, we carried out experiments with real-world data sets after adding to them irrelevant features. We added to each real-world data set 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 irrelevant linear features whose values are drawn at random between 0

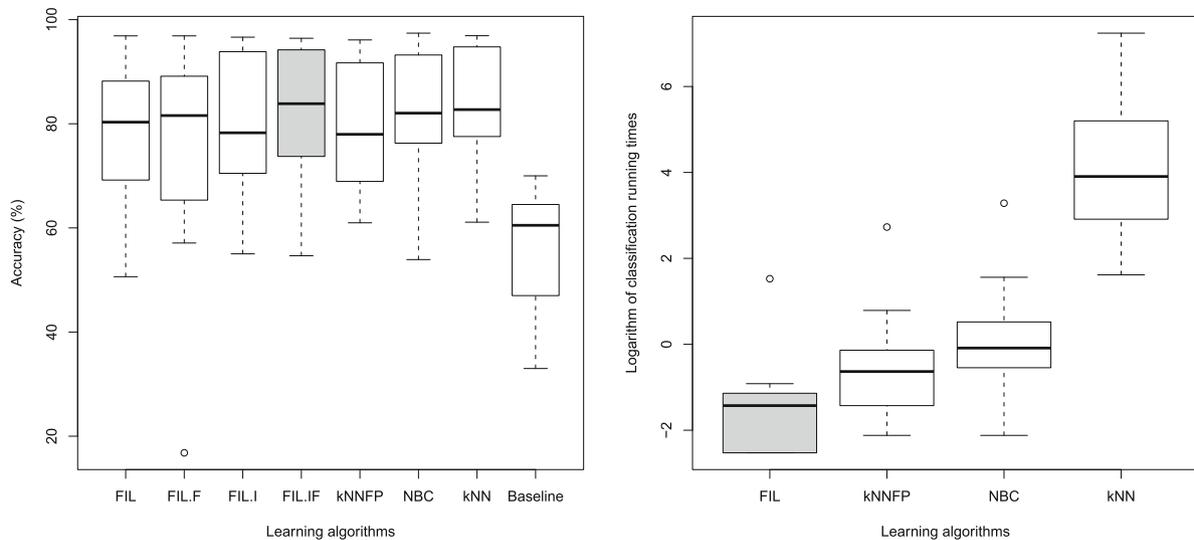


Fig. 13. Boxplots of accuracies of learning algorithms on 12 datasets reported in Table 3 are on the left, and classification running times reported in Table 4 are on the right.

Table 4  
Training and classification (test) times (ms) required for the FIL, *k*-NNFP, NBC, and *k*-NN algorithms on real-world datasets.

Dataset	FIL		k-NNFP		NBC		k-NN	
	Train	Test	Train	Test	Train	Test	Train	Test
bcancerw	19.96	0.32	1.12	0.64	10.16	1.16	0.08	200.68
cleveland	20.96	0.12	0.56	0.68	6.60	0.64	0.12	54.68
diabetes	20.32	0.28	1.72	1.00	11.64	2.44	0.08	235.00
glass	18.88	0.24	0.64	0.36	3.08	1.00	0.20	19.00
horse	23.52	0.24	4.04	0.76	9.96	1.00	0.08	106.08
hungarian	21.20	0.32	2.40	0.24	4.80	0.60	0.12	45.12
ionosphere	24.40	0.40	3.80	2.20	22.88	4.76	0.16	163.84
iris	18.64	0.08	0.08	0.12	1.12	0.12	0.08	5.04
liver	19.36	0.08	0.72	0.24	3.72	0.56	0.08	37.16
musk	55.52	4.60	23.84	15.32	155.16	26.68	0.20	1396.56
new-thyroid	18.44	0.08	0.36	0.12	1.92	0.28	0.12	12.48
wine	20.32	0.08	0.80	0.44	3.92	0.84	0.08	17.80

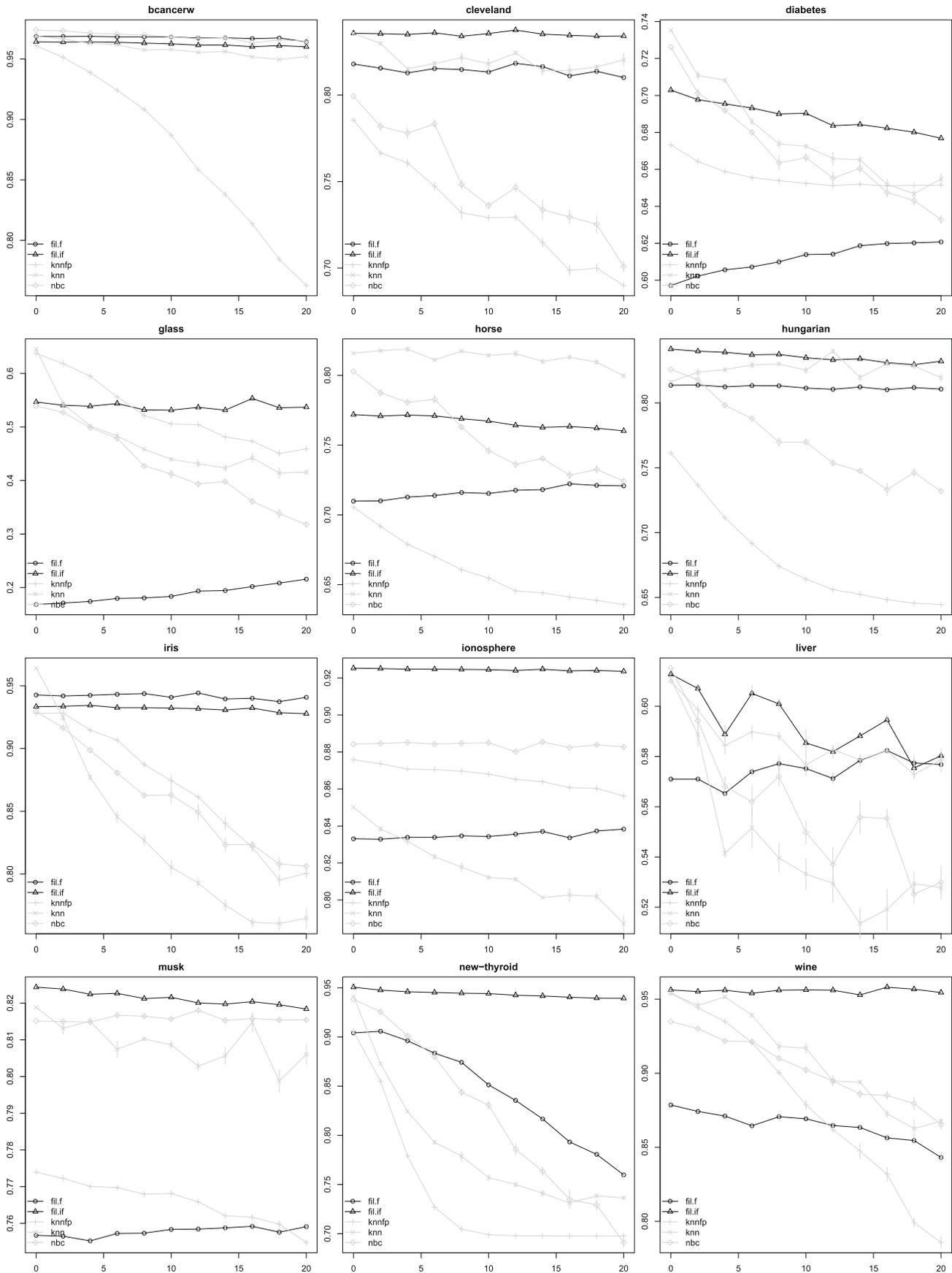


Fig. 14. The impact of irrelevant features on the performances of the learning algorithms. The numbers of irrelevant features are marked on the horizontal axes, and the classification accuracies are plotted on the vertical axes.

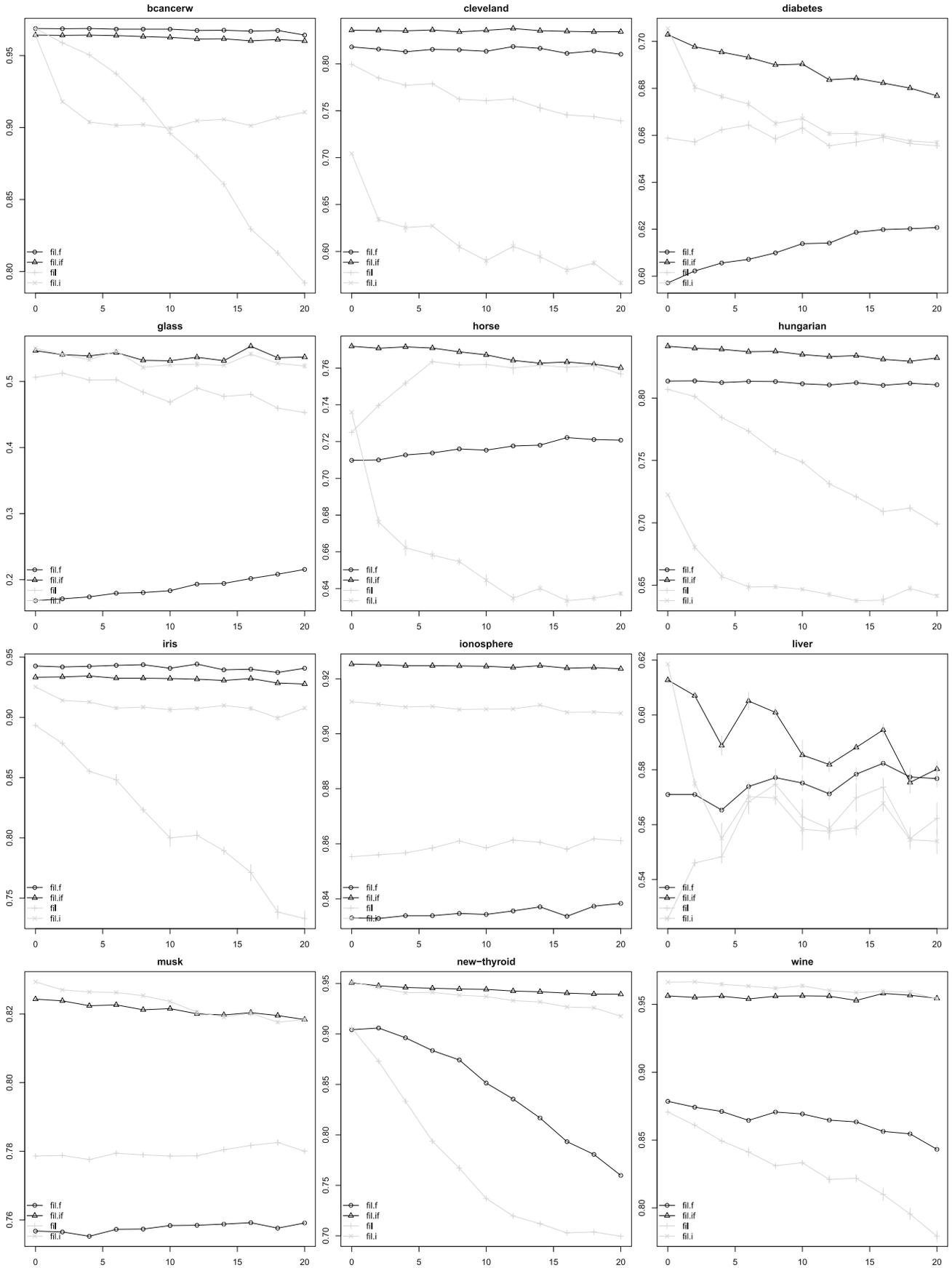


Fig. 15. The impact of irrelevant features on the performances of the FIL algorithms. The numbers of irrelevant features are marked on the horizontal axes, and the classification accuracies are plotted on the vertical axes.

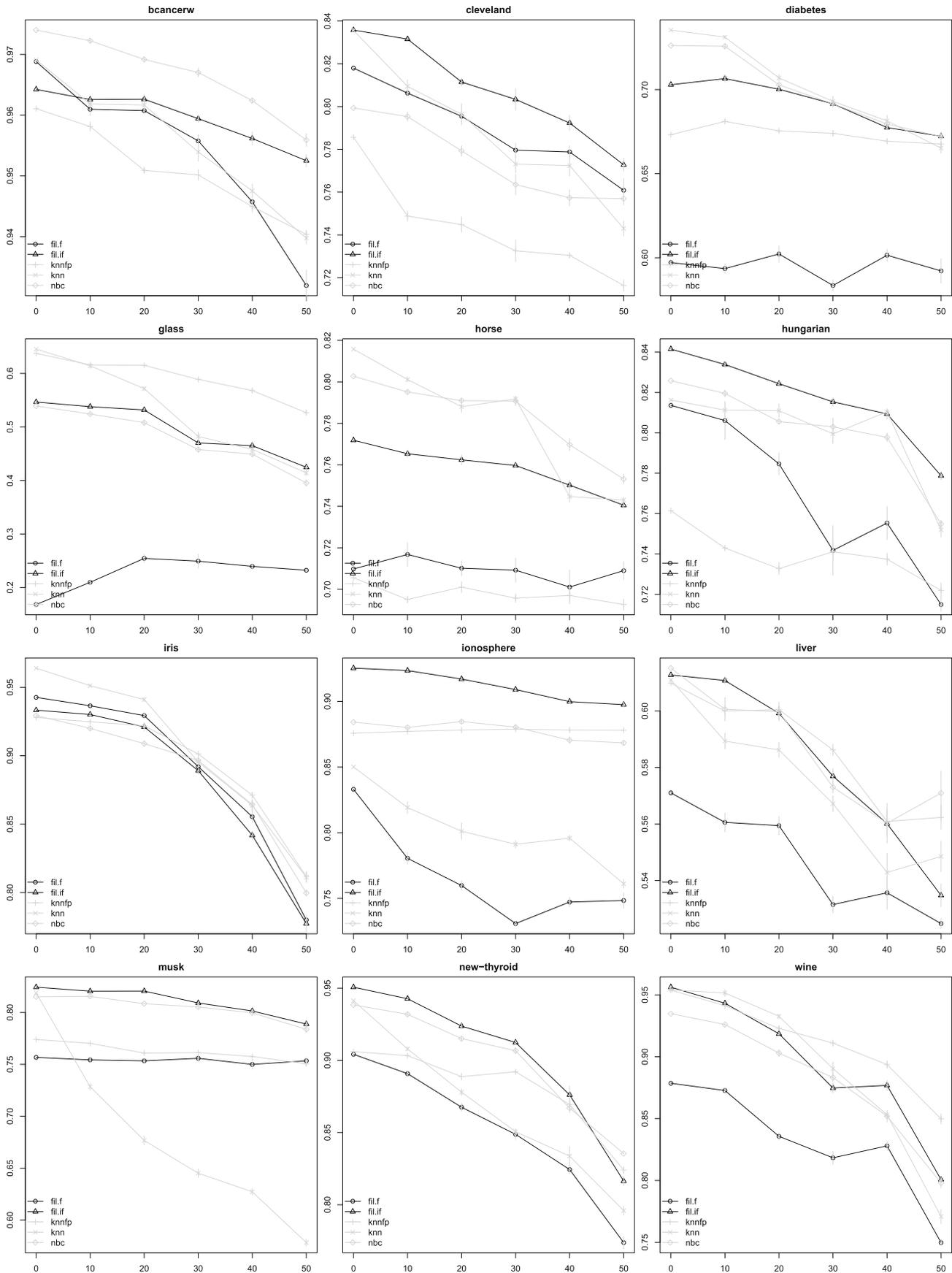


Fig. 16. The impact of missing values on the performances of the learning algorithms. The percentages of missing feature values are marked on the horizontal axes, and the classification accuracies are plotted on the vertical axes.

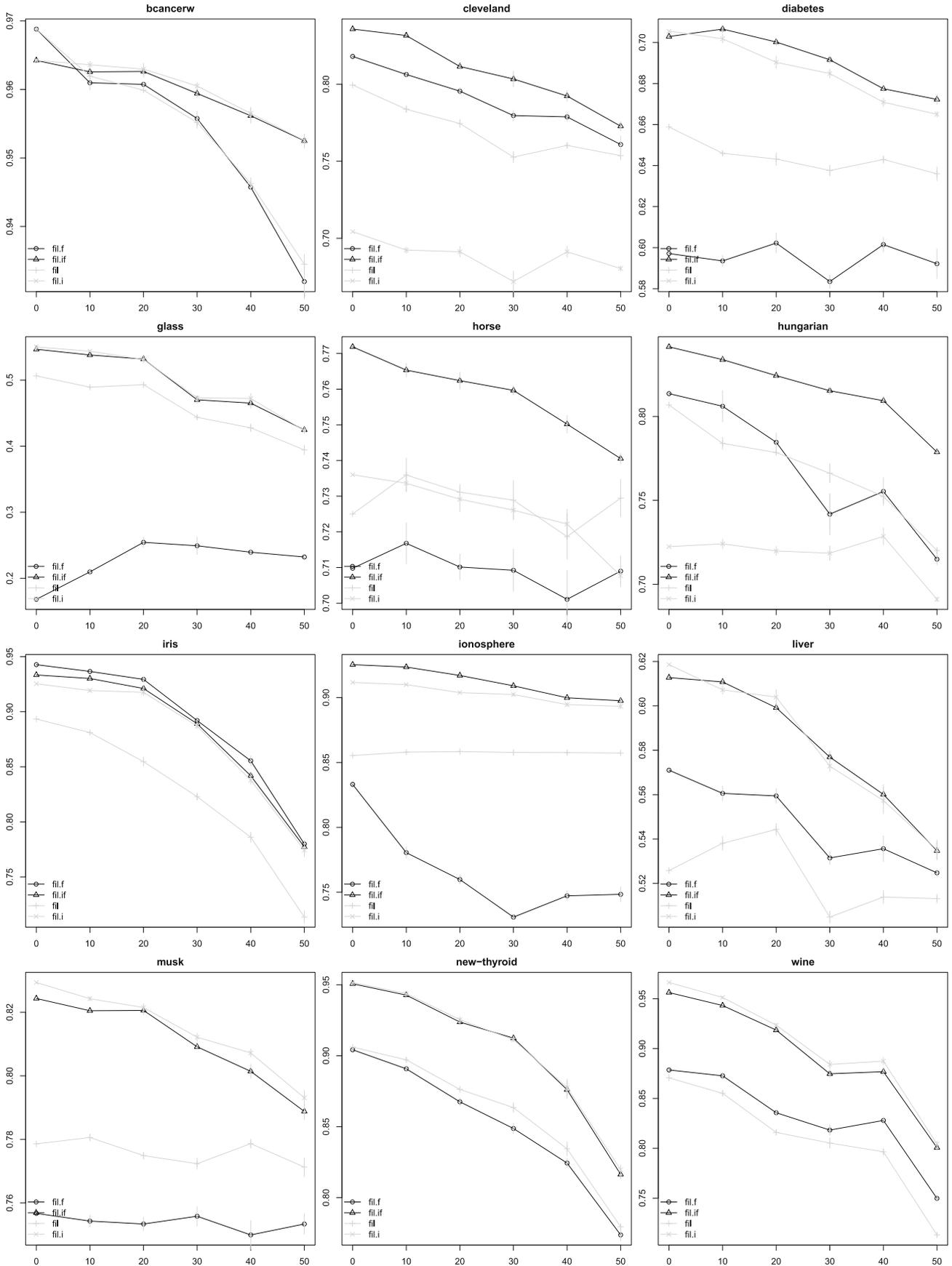


Fig. 17. The impact of missing values on the performances of the FIL learning algorithms. The percentages of missing feature values are marked on the horizontal axes, and the classification accuracies are plotted on the vertical axes.

and 1, and we repeat this artificial data generation five times to reduce the standard deviations of accuracy estimates. The new classification accuracies are the averages of 5 repetitions of 5-fold cross-validations over 5 replications of irrelevant features.

Fig. 14 shows the classification accuracies of FIL.F, FIL.IF,  $k$ -NNFP, NBC,  $k$ -NN algorithms. The experiments indicate that the FIL.IF algorithm is robust to the presence of irrelevant features. Even if  $k$ -NN and NBC algorithms sometimes achieve about the same or better predictive accuracies before the addition of irrelevant features, their accuracies quickly degrade when irrelevant features are added; see especially the results for *diabetes*, *glass*, *iris*, *new-thyroid*, and *wine* in Fig. 14. The  $k$ -NNFP algorithm is the most sensitive algorithm to irrelevant features. The FIL.IF algorithm underperforms the  $k$ -NN algorithm only on *horse* data set, but unlike NBC and  $k$ -NNFP, its performance does not deteriorate with increasing number of irrelevant features.

Fig. 15 shows the classification accuracies of only the FIL algorithms on 12 data sets from the UCI-Repository with increasing number of artificially added irrelevant features. The results show that the FIL.IF algorithm usually outperforms the other FIL algorithms and is robust in the presence of irrelevant features. The FIL.F algorithm is only slightly better than the FIL.IF algorithm on *bcancerw* and *iris*. The FIL.F algorithm performs worse than the FIL algorithm on five data sets: *diabetes*, *ionosphere*, *musk*, *glass*, and *horse*.

### 3.2.3. Evaluation in the presence of missing feature values

Because the FIL algorithms simply overlooks the missing feature values, we hypothesize that the FIL.F and FIL.IF algorithms are also robust to missing feature values. To assess the performances of the FIL algorithms with missing feature values, we carried out experiments with the same real-world data sets after deleting some of the existing feature values.

We delete from each real-world data set randomly selected 5% to 50% (in the increments of 5%) of the existing feature values, and we repeat this process five times to reduce the standard deviations of estimated accuracies. The reported classification accuracies are the averages of five repetitions of 5-fold cross-validations over five replications of missing data generations.

Fig. 16 shows the classification accuracies of the FIL.F, FIL.IF,  $k$ -NNFP, NBC, and  $k$ -NN algorithms. The FIL.IF algorithm gives the highest classification accuracies on *cleveland*, *hungarian*, *ionosphere*, *musk*, and *new-thyroid*, and is competitive on *iris*, *liver*, and *wine* with missing feature values. On the *bcancerw* data set, FIL.IF is the second best algorithm after NBC, but is more robust than NBC to the increasing percentage of missing feature values. Fig. 17 shows the classification accuracies of FIL algorithms. The FIL.IF algorithm is usually the best of all FIL algorithms and is more robust to the increasing missing feature values than other FIL algorithms.

## 4. Related work

Given a set of training instances which consist of feature values and a class label, the task of concept learning is to create general concept descriptions. Concept descriptions are learned by forming a relation between feature values and class labels. One common knowledge representation technique for concept learning tasks is *exemplar-based*. Other widely used knowledge representation techniques are decision trees [7,21] and rules [22]. Statistical concept learning algorithms combine training instances with probabilistic approaches to induce concept descriptions [14].

Fig. 18 presents a hierarchical classification of exemplar-based learning models. Knowledge representation in those models are formed by means of representative instances [23,4], generalized exemplars (e.g., hyperrectangles) [8,9], or generalized feature values (e.g, feature segments) [6].

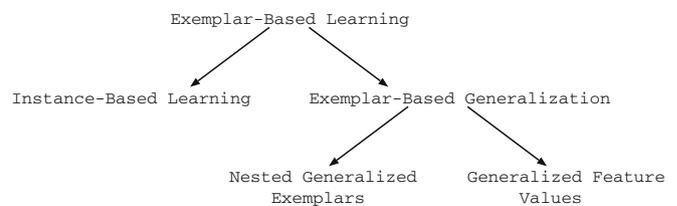


Fig. 18. Classification of exemplar-based learning models.

There are two main types of exemplar-based learning: instance-based learning and exemplar-based generalization. An instance-based learning algorithm maintains examples in memory as points and never changes them. Two important decisions to be made are which examples to store in memory and how to measure similarity between examples. Several variations of instance-based learning algorithms have been developed [23,16,4]. Wilson and Martinez [24] proposed several techniques to reduce storage requirements of instance-based learning algorithms and showed that storage reductions are substantial with high accuracy. Brighton and Mellish [25] proposed the Iterative Case Filtering algorithm for instance selection in instance-based learning algorithms and showed that it is competitive with the most successful technique proposed by Wilson and Martinez. On the other hand, an exemplar-based generalization model generalizes examples to form concept descriptions. An example of this model is nested generalized-exemplars (NGE) model [8,9]. NGE theory changes the point storage model of the instance-based learning and generalizes examples as axis-parallel hyperrectangles to create compact representations. Algorithms based on NGE theory classify new examples with the label of the nearest hyperrectangle.

Generalized Feature Values (GFV) models are also exemplar-based generalization models. The examples of GFV learning models are the classification by feature partitioning (CFP) [6], voting feature intervals (VFI5) [26], and the  $k$ -Nearest Neighbor on Feature Projections ( $k$ -NNFP) [18]. The CFP algorithm represents classification knowledge as sets of disjoint feature segments. It partitions the feature values into segments that are generalized or specialized as the training instances are processed. It constructs the segments incrementally, and the resulting concept descriptions are sensitive to the order in which training instances are processed. The VFI5 algorithm learns classification knowledge in batch mode and represent it as multi-class feature intervals. The  $k$ -NNFP algorithm represents instances as separate collections of feature projections. GFV model-based algorithms predict the class of a new example as the label with the highest weighted total vote of the individual features.

GFV models allow faster classification than other instance-based learning models because separate feature projections can be organized for faster classification. GFV models also allow easy handling of missing feature values by simply ignoring them. The major drawback of GFV models is that descriptions involving a conjunction between two or more features cannot be represented. However, GFV models are still reported to be quite successful on real-world data sets [27,6,26,18].

One of the most well-known algorithms in machine learning is the nearest neighbor (NN) algorithm [14,15]. It is an instance-based learning algorithm and stores all training instances in memory. To predict the class label of a test example, it computes the distance between the test example and training instances. The class of the test example is predicted as the class of the training instance with the shortest distance, i.e., the nearest neighbor. The  $k$ -NN algorithm is a generalization of the NN algorithm, and its classification is based on a majority voting of the nearest  $k$  neighbors. Wettschereck and Dietterich [10] reported that the  $k$ -NN algorithm is superior to those algorithms based on the NGE

theory. Previous research has also shown that  $k$ -NN algorithm and its local-weighted variations give excellent results on many real-world induction tasks [16,4,17,28]. The major drawback of  $k$ -NN is the curse of dimensionality: calculations of distances between test and training instances become quickly prohibitive with increasing number of instances and feature dimensions.

Bayesian classifiers from pattern recognition are based on probabilistic approaches to inductive learning in statistical concept learning tasks. The method estimates the posterior class probability of an instance given its observed feature values. The class is predicted as the label with the highest estimated posterior probability [14,29,30]. Bayesian classifiers assume in general that features are not statistically independent unlike naive Bayesian classifier (NBC).

## 5. Conclusion

In this paper, we developed several batch learning algorithms, called *Feature Interval Learning* (FIL) algorithms. The FIL algorithms use feature projections of the training instances for the representation of the classification knowledge induced. They assume that similar feature values have similar classifications. Feature projections on linear features are generalized into disjoint feature intervals during the training phase. The classification of an unseen instance is based on a weighted-majority voting among individual predictions of features.

The basic FIL algorithm is enhanced with adaptive interval and feature weight schemes in order to handle noisy and irrelevant features. By weighing each feature inversely proportional to the number of feature intervals constructed on that feature dimension, the voting process for the classification reduces the detrimental effects of irrelevant features or noisy feature values.

The FIL algorithms are compared empirically to the  $k$ -NN,  $k$ -NNFP, and NBC algorithms. The FIL-IF algorithm, which incorporates adaptive interval and feature weights, is found to be superior to all other FIL algorithms. Although the FIL-IF algorithm achieves comparable accuracies with the  $k$ -NN algorithm, its classification times are much less than those of the  $k$ -NN algorithm. Moreover, the FIL-IF algorithm is robust to irrelevant features and missing feature values.

Feature interval-based knowledge representation in the FIL algorithms produces plausible concept descriptions, enables faster classification than the instance-based knowledge representation of the  $k$ -NN algorithm, does not require normalization of feature values, and handles missing feature values naturally by simply ignoring them.

The major disadvantage of the feature interval-based representation is that concept descriptions involving a conjunction between two or more features cannot be represented. Therefore, the FIL algorithms are applicable to concepts where each feature can contribute to the classification of the concept independent of other features. This turns out to be the nature of the most real-world data sets [31]. The FIL algorithms are not applicable to domains where all of the concept descriptions overlap, or domains in which concept descriptions are nested.

In summary, the primary contributions of this paper can be listed as follows:

- We formalized the concept of feature intervals for knowledge representation in inductive supervised learning algorithms.
- We presented several batch learning methods of disjoint feature intervals by assigning weights to features and intervals derived directly from training data.
- We showed that the FIL-IF algorithm is a fast and competitive method on most of the real-world data sets.
- We showed that the FIL-IF algorithm is robust in the presence of irrelevant features, and stable in the presence of missing feature values.

As future work, we plan to investigate optimal methods for feature partitioning. For overlapping concept descriptions, batch learning algorithms of which knowledge representation is in the form of overlapping feature intervals can be developed. Another research direction is to investigate learning concept-dependent feature weights for the learning algorithms which use feature projections for knowledge representation.

## References

- [1] J. Carbonell (Ed.), *Machine Learning: Paradigms and Methods*, The MIT Press, 1990.
- [2] R. Michalski, J. Carbonell, T. Mitchell, *Machine Learning an Artificial Intelligence Approach*, Morgan Kaufman Publishers Inc., San Francisco, CA, 1986.
- [3] T. Mitchell, *Machine Learning*, McGraw Hill, New York, 1997.
- [4] D. Aha, D. Kibler, M. Albert, Instance-based learning algorithms, *Machine Learning* 6 (1991) 37–66.
- [5] D. Angluin, Queries and concept learning, *Machine Learning* 2 (1988) 312–342.
- [6] H. Güvenir, I. Şirin, Classification by feature partitioning, *Machine Learning* 23 (1996) 47–67.
- [7] J. Quinlan, Induction of decision trees, *Machine Learning* 1 (1986) 81–106.
- [8] S. Salzberg, *Learning with Generalized Exemplars*, Kluwer Academic Publishers., Massachusetts, 1990.
- [9] S. Salzberg, A nearest hyperrectangle learning method, *Machine Learning* 6 (1991) 251–276.
- [10] D. Wettschereck, T. Dietterich, An experimental comparison of the nearest neighbor and nearest-hyperrectangle algorithms, *Machine Learning* 19 (1995) 5–27.
- [11] D. Wettschereck, D. Aha, Weighting Features, in: *Proceedings of the First International Conference on Case-Based Reasoning*, Springer-Verlag, Lisbon, Portugal, 1995, pp. 347–358.
- [12] J. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufman, California, 1993.
- [13] J. Chen, H. Huang, F. Tian, S. Tian, A selective Bayes classifier for classifying incomplete data based on gain ratio, *Knowledge-Based Systems* 21 (2008) 530–534.
- [14] R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*, second ed., John Wiley & Sons, Inc., 2000.
- [15] B. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, 1991.
- [16] D. Aha, Tolerating noisy, Irrelevant and novel attributes in instance-based learning algorithms, *International Journal of Man-Machine Studies* 36 (1992) 267–287.
- [17] S. Cost, S. Salzberg, A weighted nearest neighbor algorithm for learning with symbolic features, *Machine Learning* 10 (1993) 57–78.
- [18] A. Akkuş, H.A. Güvenir,  $K$ -Nearest Neighbor Classification on Feature Projections, in: L. Saitta (Ed.), *Proceedings of the 13th International Conference on Machine Learning*, Morgan Kaufman, Bari, Italy, 1996, pp. 12–19.
- [19] A. Asuncion, D. Newman, UCI Machine Learning Repository, University of California, School of Information and Computer Science, Irvine, CA, 2007. [<http://www.ics.uci.edu/~mllearn/MLRepository.html>].
- [20] D. Kibler, P. Langley, *Machine Learning as an Experimental Science*, in: J. Shavlik, T. Ditterich (Eds.), *Readings in Machine Learning*, Morgan Kaufman, San Mateo, CA, 1990, pp. 38–43.
- [21] L.-M. Wang, X.-L. Li, C.-H. Cao, S.-M. Yuan, Combining decision tree and Naive Bayes for classification, *Knowledge-Based Systems* 19 (2006) 511–515.
- [22] R. Michalski, A theory and methodology of inductive learning, *Artificial Intelligence* 20 (1983) 111–161.
- [23] D. Aha, A Study of instance-based algorithms for supervised learning tasks: Mathematical, empirical, and psychological evaluations, Ph.D. Thesis, Department of Information and Computer Science, University of California, Irvine, 1990.
- [24] D. Wilson, T. Martinez, Reduction techniques for instance-based learning algorithms, *Machine Learning* 38 (2000) 257–286.
- [25] H. Brighton, C. Mellish, Advances in instance selection for instance-based learning algorithms, *Data Mining and Knowledge Discovery* 6 (2002) 153–172.
- [26] H. Güvenir, G. Demiröz, N. Ilter, Learning differential diagnosis of erythematous diseases using voting feature intervals, *Artificial Intelligence in Medicine* 13 (1998) 147–165.
- [27] H. Güvenir, I. Şirin, A genetic algorithm for classification by feature partitioning, in: *Proceedings of the fifth International Conference on Genetic Algorithms*, 1993, pp. 543–548.
- [28] M. Laguna, J.L. Castro, Local distance-based classification, *Knowledge-Based Systems* 21 (2008) 692–703.
- [29] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, 1990.
- [30] M. Hall, A decision tree-based attribute weighting filter for Naive Bayes, *Knowledge-Based Systems* 20 (2007) 120–126.
- [31] R. Holte, Very simple classification rules perform well on most commonly used datasets, *Machine Learning* 11 (1993) 63–90.