

# Hyper-Rectangular and $k$ -Nearest-Neighbor Models in Stochastic Discrimination

Iryna Skrypnyk and Tin Kam Ho

*Abstract— The stochastic discrimination (SD) theory considers learning as building models of uniform coverage over data distributions. Despite successful trials of the derived SD method in several application domains, a number of difficulties related to its practical implementation still exist. This paper reports analysis of simple examples as a first step towards presenting the practical implementation issues, such as model generation and preliminary estimations to set parameters. Two implementations using different methods for model generation are discussed. One uses the nearest neighbor approach to maintain the projectability condition, the other constructs hyper-rectangular regions by randomly selecting subintervals in each dimension. Analysis of these implementations shows that for high-dimensional data, parallel model generation with the nearest neighbor approach is a favorable alternative to the interval model generation with random manipulation of the feature subspaces.*

## 1 Introduction

The theory of stochastic discrimination (SD) developed by Kleinberg [8]–[10] and enhanced by Berlind [1] and Chen [2] has been successfully applied to several pattern recognition problems [3], [4], [7], [12]. It is yet to be widely explored by the machine learning and data mining community. The SD method has good potentials for contemporary data mining problems, especially those with heterogeneous data [17], because SD seeks to construct collections of multiple models each covering a particular region in the geometrical representation of the data. The SD theory established that given a solvable discrimination problem, the method will get arbitrarily close to the best solution in affordable time [8], [9]. Discrimination of classes is promoted by maintaining rigorous conditions of *uniformity* and *enrichment* with *projectable* component models. The models are generated by a pseudo-random process in order to cover the instances of each particular class with an approximation to its spatial structure and thus, separating it from the rest of the samples.

Despite reported successful application of SD to a variety

of problem domains, a number of difficulties still exist with its practical implementation. A key implementation problem is preliminary estimation of the data set characteristics to set constraints on the model generation and selection process, such as the model size, the enrichment degree and the ratio of the covered “weak” instances for uniformity forcing.

In this paper the SD method with the underlying theory is presented from the perspective of application to the contemporary data mining problems, which often face high-dimensional heterogeneous data sets. The classification problem on such data may have a complex decision boundary and/or subclass structure [6]. In such cases the strict mathematical assumptions used in the SD theory are rarely met [4]. Practical implementation of SD requires weakened assumptions and specific algorithmic solutions.

This paper describes implementations of two methods for model generation and related preliminary estimation of data characteristics. One is based on the nearest neighbor approach and other randomly selects subintervals in each dimension. For high-dimensional data, parallel model generation using nearest neighbors is considered as an alternative to interval model generation with random manipulation of the feature subspaces for increasing chances of successful model generation. Analysis of advantages and disadvantages of these implementations presented in the paper contributes to the research on further advance of this promising data mining technique.

## 2 Theoretical Background

This section introduces the SD theory as a background for the method, which is needed to understand the problems of practical implementation discussed in the paper. The framework is given for a two-class problem. Extension for multiclass problems requires pairwise class decomposition or one-against-all class decomposition onto subproblems [15]. The first implementation of SD (SDK) introduced in [8] uses hyper-rectangular regions as models. This model type is considered in the paper as a basic type.

### 2.1 A Basic Hyper-rectangular Model

Consider a training set of instances  $TR$  of size  $N$  as a set of points  $q_i$ ,  $i = 1, \dots, N$  in a  $P$ -dimensional space created by measured features  $f_j$ ,  $j = 1, \dots, P$ . Each instance belongs to class  $d$ ,  $d = 1, \dots, D$ . For the basic two-class problem  $D = 2$   $TR$  consists of two subsets  $TR_p$  and  $TR_n$ , where  $TR_p$  includes  $n_p$  instances and  $TR_n$  includes  $n_n$

I. Skrypnyk is with the Computer Science and Information Systems Department, University of Jyväskylä, 40014 Jyväskylä, Finland (e-mail: iryna@cs.jyu.fi).

T. K. Ho is with the Computing Sciences Research Center, Bell Laboratories, Lucent Technologies, Murray Hill, NJ, 07974-0636 USA (e-mail: tkh@research.bell-labs.com).

instances,  $n_p + n_n = N$ .

A discrete numeric feature  $f_j$  has  $T$  discrete values  $v_{1,j} \dots v_{t,j} \dots v_{T,j}$ , where  $v_{1,j}$  is a minimal value and  $v_{T,j}$  is a maximal value. A continuous numeric feature can have up to  $N$  distinct values realized in  $TR$ . A nominal feature needs to be coded with a discrete numeric value. Assume that all necessary transformations are made and all features values are known to be in a certain range. Each dimension corresponding to a particular feature  $f_j$  that has an ordered list of distinct coordinates  $v_{1,j} < v_{t,j} < v_{T,j}$  in  $R^p$ . The entire training set  $TR$  is placed to a rectangular region  $R^p$ , which is denoted by  $\prod_{j=1}^p (v_{1,j}, v_{T,j})$ .

For each fixed  $v_{t,j}$  choose numbers  $h_{j,k}$ ,  $k=1, \dots, T_j-1$ . The hyper-planes drawn at  $h_{j,k}$  in each dimension  $j$  divide  $R^p$  into  $s = \prod_{j=1}^p T_j$  hyper-rectangular regions used as *models*, each containing at most one instance from  $TR$ .

Let  $R^1$  be the smallest of  $s$  hyper-rectangular regions containing in  $TR$ . Denote the unit intervals in each dimension  $j$  for  $R^1$  as  $(a_j, b_j)$ . Then the hyper-rectangular regions in each dimension include  $\lambda \geq 1$  unit intervals having width  $\lambda(a_j, b_j)$ . A model  $M$  is described as  $R^\lambda = \prod_{j=1}^p \lambda(a_j, b_j)$ ,  $\Lambda \geq \lambda \geq 1$ , where  $\Lambda$  is a threshold.

## 2.2 Uniformity, Enrichment and Projectability

For each instance  $q_i$  from  $TR$  one can count the coverage  $N(q_i, M_t)$  - the current number of models in the collection  $M_t$  that include  $q_i$ , where  $t$  is the number of models in the collection. Denote  $Y(q_i, M_t)$  a ratio of  $N(q_i, M_t)$  over the current number of models in  $M_t$ , which is  $t$ .  $Y(q_i, M_t)$  is a probability that the point  $q_i$  is covered by the particular model  $M$  included to  $M_t$ . As the collection of model expands, the values of  $Y(q_i, M_t)$  for each  $q_i$  converge to the per-model probability that a point in  $TR$  is included in that model [5], [10], if the model generation process is not biased toward any particular  $q_i$ .

In order to explain the uniformity condition let us consider a simple example from [5], when  $TR$  includes 10 instances and each model  $M$  includes 5 instances. Then the number of all possible models one can get in  $M_t$  (without repetitions) is 252 (the points are not associated with any spatial coordinates). When  $M_t$  includes 252 models each instance  $q_i$  has the same coverage  $N(q_i, M_t)$  and  $Y(q_i, M_t) = 0.5$ , i.e. the probability that every point from  $TR$  is covered by any particular model that were included

to the collection  $M_t$  is 0.5. A *uniform coverage* of  $TR$  is a fundamental principle of the SD theory.

For any practical task involving large feature spaces, creating an exhaustive collection of models from a stochastic process of model generation is infeasible. Instead one may target for obtaining *nearly* uniform coverage in a reasonable time. This approximation can be made by setting threshold values for the uniformity and enrichment conditions. Also a restriction to cover only instances of one particular class can be used.

Above it was shown how the  $Y(q_i, M_t)$  values converge without considering the class membership of the instances included to the model. In discriminating between two classes, models are constructed in a way that each covers the two classes with different probabilities. If a uniform coverage is still maintained over the entire collection,  $Y(q_i, M_t)$  converges to distinct values for each class.

By the theory, each model should be enriched towards a particular class. For a particular model the class to be enriched is a *positive* class and the other class a *negative* class. The instances from  $TR$  covered by a model belonging to the positive class are *positives*, and the instances belonging to the negative class are *negatives*. Let us denote the fraction of positives covered by the model  $M$  as  $frac_{p,M} = pos_M / n_p$ , where  $pos_M$  is a number of positives covered by the model  $M$  and  $n_p$  is the number of positives contained in  $TR$ . The fraction of negatives covered by the model  $M$  as  $frac_{n,M} = neg_M / n_n$ , where  $neg_M$  is a number of positives covered by the model  $M$  and  $n_n$  is the number of positives contained in  $TR$ .

These fractions can be considered as a *degree of enrichment* of a particular model  $M$  toward one of the classes. *Rating* ( $pos_M, neg_M$ ) is a particular combination of positives and negatives in  $M$ , then  $frac_{p+n,M} = (pos_M + neg_M) / (n_p + n_n)$ .

Indirectly, the limits to the model size will be introduced with two fixed real numbers, which will be denoted as  $\mu$  and  $\eta$ ,  $0 < \mu \leq \eta \leq 1$ .  $\mu$  and  $\eta$  do not denote the minimum and maximum size of a model, instead they limit the  $frac_{p+n,M}$  so that  $\mu \leq frac_{p+n,M} \leq \eta$ . Let us denote  $\beta$  the enrichment threshold having a fixed real value,  $0 < \beta < 1$ . The enrichment condition for a candidate model  $M$ , such that  $\mu \leq frac_{p+n,M} \leq \eta$ , is described by the inequality

$$\left| frac_{p,M} - frac_{n,M} \right| \geq \beta.$$

The *proximity* concept is a basic way to generate *projectable* models, i.e. with similar coverage to  $TR$  and  $TE$ . In many supervised learning algorithms, for example, in  $k$ -Nearest Neighbor, there is an assumption that the nearby instances, in terms of a predefined distance function or a similarity measure, belong to the same class. This assumption is justified for a majority of natural problems.

SD assumes that the training set is *representative with respect to the descriptive power of the component models* in order to guarantee similar accuracies of the classifier that is based on the similar enrichment degrees of the component models [5].

### 2.3 The Base Random Variable and the Classification Rule

The *base random variable*  $X(q_i, M_j)$  is defined via the membership function  $C_{M_j}(q_i)$ .  $C_{M_j}(q_i) = 1$  iff  $q_i$  is covered by the model  $M_j$  generated at  $j$ -th iteration, and 0

otherwise.  $X(q_i, M_j) = \frac{C_{M_j}(q_i) - \text{frac}_{n, M_j}}{\text{frac}_{p, M_j} - \text{frac}_{n, M_j}}$ . Then at each  $j$

for the current collection of models  $M_t$  one can determine a discriminant  $Y(q_i, M_t) = \frac{1}{t} \sum_{k=1}^t X(q_i, M_k)$ . As  $M_t$  expands,

the value  $Y(q_i, M_t)$  changes for each  $q_i$  and converges to two distinct values, 1 and 0, for the positives and negatives accordingly. The trends become separated as the collection of model expands. Figure 1 demonstrates this for the example used in [10] with the histogram for the  $Y(q_i, M_t)$  values over 1000 iterations of model generation. A more detailed explanation can be found in [4], [5], [10].

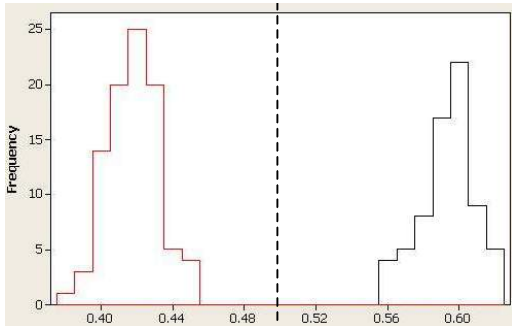


Fig. 1. The histogram for the  $Y(q_i, M_t)$  values on  $TR$  over 1000 models for the synthetic data set used to illustrate the SD principles in [10].

The classification rule is obtained using the Central Limit Theorem. The base random variable is applied instead of using models as weak classifiers directly. Instances from  $TR$  are assigned to the negative or positive class depending on whether the correspondent  $Y(q_i, M_t)$  value is less or greater than 0.5, as shown by the vertical dashed line in Figure 1.

For the instances from  $TE$  the  $Y$  values are determined considering their coverage by the models from  $M_t$ . In order to obtain the final prediction for multiclass problems one needs to evaluate the probability variables  $Y$  in subproblems and assign the class value from the subproblem, where  $Y$  is the largest [10].

### 2.4 Enforcing Uniformity and Enrichment

Separation of trends happens before all possible models restricted by selection of  $\mu$ ,  $\eta$  and  $\beta$  are included.

Collection of the models enriched toward one class is a common setting for the enrichment threshold. Higher degree of enrichment promotes accuracy increase, but in order to generate highly enriched models more iterations required because such models less likely to appear during the generation process. Thus, preliminary estimation for the enrichment threshold is desired in order to prevent the generation process from getting caught in an endless loop.

The uniformity condition is much more difficult to satisfy. Strict uniformity requires that every instance in  $TR$  to be covered by *the same number of models of every particular rating* within the specified restrictions for the model size ( $\mu, \eta$ ) and the number of ratings ( $\beta$ ). For practical tasks this strict condition is weakened for *near uniformity* [4].

In order to simplify the task,  $\mu$ ,  $\eta$  and  $\beta$  can be set to accept models of only one particular rating [5]. However, there is a tradeoff involved, because the number of such models can be very small and even the nearly uniform coverage of all instances will never be done. This problem will be considered later with regard to the multistream model generation approach proposed in this paper.

In order to promote the uniform coverage of the positive class with these models one should calculate coverage for  $q_i$  after each successful addition of the new model to  $M_t$ . Every iteration some instances can be labeled as “weak” if *their coverage is less than the average coverage of the instances of their class*. The new model will be accepted if in addition to the enrichment condition it will cover a particular ratio of “weak” instances. The uniformity threshold then can be set as a ratio of “weak” instances covered by a new model to the number of currently “weak” instances.

## 3 Two SD Implementations

In this section two ways of hyper-rectangular model generation are described, the interval based and the  $k$ -Nearest Neighbor based model generation. The conflict between enrichment and uniformity is described demonstrating the need of preliminary estimation or the alternative solutions.

### 3.1 The Basic SD Algorithm and Parameters

Before elaborating the problems of practical implementation in later sections let us consider the basic algorithm of SD using the interval based generation of hyper-rectangular models.

The parameters of the basic SD algorithm are the minimum and maximum model sizes (alternatively to  $\mu$  and  $\eta$ ), the number of iterations  $I$ , the enrichment threshold  $\beta$ , the uniformity threshold  $\omega$  and the classification rule probability threshold  $\alpha$ . Assume, that all

necessary transformations of non-numeric feature values to nonnegative integers are made and all features values are ranged. The basic algorithm steps are outlined in Figure 2.

---

```

* Algorithm SD_interval_basic *
For (j = 0; j < P; j = j + 1){
  min[j] = determineFeatureMin(j);
  max[j] = determineFeatureMax(j);
}
// start collection of models
While (i < I) {
  // start generation of an acceptable model
  While (modelAccepted = false) {
    q0 = randomPoint(TR);
    hrect = generateSubintervals(min, max, q0);
    // hrect [0] - lower ends
    // hrect [1] - upper ends
    tmpmod = getModelInstances(hrect[0], hrect[1]);
    // start uniformity and enrichment test
    enrichment = getEnrichment(tmpmod);
    if(enrichment > β){
      uniformity = getCoverageRatio(tmpmod);
      if(uniformity > ω)
        modelAccepted = true;
      else
        modelAccepted = false;
    }
    else{
      modelAccepted = false;
    } // end uniformity and enrichment test
  } // end generation of an acceptable model
  updateCoverage(tmpmod);
  updateProbability(tmpmod);
  models[i] = tmpmod;
} // end collection of models
// start classification
For (i = 0; i < N; i = i + 1){
  For (j = 0; j < I; j = j + 1){
    if(coveredByModel(j, test_q[i]) = true)
      updateProbability(test_q[i]);
  }
  prob[i] = calculateTestProbability(test_q[i]);
  classifyTestPoint(prob[i], α);
} // end classification

```

---

Fig. 2. The basic steps of the SD algorithm with interval based model generation.

The example parameter settings for the basic SD algorithm when  $N \approx 200$ : the minimum model size =  $0.03 * N$  and the maximum model size =  $0.06 * N$ , the number of iterations  $I = 1000$ , the enrichment threshold  $\beta = 0.01$ , the uniformity threshold  $\omega = 0.3$  and the classification rule is thresholded at  $\alpha = 0.5$ . In [11] the simple enrichment condition does not involve any threshold and takes a form of the inequality  $frac_{p,M} > frac_{n,M}$ .

### 3.2 Interval Based Generation of Models

The interval based generation of the hyper-rectangular models follows the basic ideas described in Section 2. There are a few advances to this algorithm proposed in [13]. In order to produce the models, which are the regions defined by the subinterval in each dimension, consider the way to find the subintervals.

- Set the number of trials  $e$  for each iteration (for example,  $e = 2000$ );
- Determine the maximum hyper-rectangular region covering

the entire data  $R^P = \prod_{j=1}^P (v_{1,j}, v_{T,j})$ , where  $v_{1,j}$  and  $v_{T,j}$  are minimum and maximum values of feature  $j$ ;

- Set the coefficient  $\lambda = 1.1$ ;
- Determine  $R^\lambda = \prod_{j=1}^P \lambda(l_j, u_j)$ ;
- Set the minimum interval percentage coefficient  $\rho_{\min} = 0.005$  (the initial value is 0.5 % of the interval between  $v_{1,j}$  and  $v_{T,j}$ );
- Set the maximum interval percentage coefficient  $\rho_{\max} = 0.125$  (the initial value is 12.5 % of the interval between  $v_{1,j}$  and  $v_{T,j}$ );
- Set the coefficient  $\rho_{decay} = 0.9999$  to decrease  $\rho_{\max}$  during each of  $e$  trials.

The models as hyper-rectangles are formed as follows. Let  $R^1$  be the smallest hyper-rectangular region containing the training set  $TR$ . Then let  $R^\lambda$  be the rectangular region in  $R^P$  such that  $R^\lambda$  and  $R^1$  have the same center,  $R^\lambda$  is similar to  $R^1$  and the width of  $R^\lambda$  along each dimension is  $\lambda$  times the corresponding width of  $R^1$ . A subinterval  $[l_j, u_j]$  of  $[v_{1,j}, v_{T,j}]$  is then generated in each dimension  $j$  by taking a random  $q_0$  as a center, and assigning a width of the subinterval  $(l_j, u_j) = \varepsilon(v_{1,j}, v_{T,j})$ , where  $\varepsilon$  is random value such that  $\rho_{\min} \leq \varepsilon < \rho_{\max}$ . Finally, the subinterval  $[l_j, u_j]$  is adjusted to reside entirely within  $R^\lambda$ .  $\rho_{\max}$  is multiplied by  $\rho_{decay}$  each time the new model is added to the collection, until  $\rho_{\max} = \rho_{\min}$ .

Our algorithm presented in Figure 3 follows these steps with a few changes. Instead of using trials and  $\rho_{decay}$ , adjustments for the model size are performed inside of the generation procedure. The hyper-rectangular region is increased or decreased in each dimension with  $0.01(v_{1,j}, v_{T,j})$ . This procedure can be used in algorithm presented in Figure 2.

The main problem of the interval based generation is that it often takes a very long time to perform adjustments in a *while* cycle without pre-estimation for the minimum and maximum model sizes. The enrichment and uniformity conditions make the problem even harder. For the high-dimensional data sets this method goes to an endless loop because the instances become almost equidistant and it is very hard to capture them by generating subintervals in every dimension. The parameters specific for this method that were given as an example need pre-estimation. The solution for this problem is to select a subset of features at each iteration in order to reduce dimensionality. Alternatively, we propose to draw a model using the  $k$ -Nearest Neighbor method.

### 3.3 $k$ -Nearest Neighbor Model Generation

The  $k$ -Nearest Neighbor method selects the initial point

randomly and then finds  $k$  nearest neighbors of the initial instance for a model of size  $k+1$ . When the number of nearest neighbors is greater than  $k$ , the instances are selected in the direction of higher density on the training set.

---

```

* Procedure generateSubintervals( $v_1, v_T, q_0$ ) *
// external size  $s_{min}, s_{max}, rho_{min}, rho_{max}$ 
lambda = 1.1;
coeff = randomNumber(rhomin, rhomax);
// start for all dimensions
For (j = 0; j < P; j = j + 1){
  length =  $v_{T,j} - v_{1,j}$ ; // min-max length
  delta = (length * coeff) / 2;
  upperj =  $q_{0,j,max} + delta$ ;
  lowerj =  $q_{0,j,min} - delta$ ;
  delta2 = length * lambda;
  delta3 = length * (1 - lambda);
  highestj =  $v_{T,j} + delta2$ ;
  lowestj =  $v_{1,j} - delta2$ ;
  if (lowerj < lowestj)
    lowerj = lowestj;
  if (upperj > highestj)
    upperj = highestj;
} // end for all dimensions
tmpmod = getModelInstances(lower, upper);
currmodsize = getCurrModSize(tmpmod);
// start adjusting the model size
While (!( $s_{min} < currmodsize < s_{max}$ )) {
// start for all dimensions
  For (j = 0; j < P; j = j + 1){
    if (currmodsize <  $s_{min}$ ) {
      upperj = upperj + delta3;
      lowerj = lowerj - delta3;
    }
    if (currmodsize >  $s_{min}$ ) {
      upperj = upperj - delta3;
      lowerj = lowerj + delta3;
    }
    if (lowerj < lowestj)
      lowerj = lowestj;
    if (upperj > highestj)
      upperj = highestj;
  } // end for all dimensions
  tmpmod = getModelInstances(lower, upper);
  currmodsize = getCurrModSize(tmpmod);
} //end adjusting the model size
return lower, upper;

```

---

Fig. 3. Interval based model generation with adjustment of model size.

---

```

* Procedure generateKNNModel( $q_0$ ) *
// generates a model of the fixed size s
// start model generation
nlist = findNeighbors( $q_0, s-1$ );
tmpmod = getListInstances( $q_0, nlist$ );
intervals = drawHRect(tmpmod);
tmpmod2 = getModelInstances(intervals);
currmodsize = getCurrModSize(tmpmod2);
// start adjusting the model size
if (currmodsize != s) {
  list2 = findEqualDistancePoints( $q_0, nlist$ );
  newlist = reduceTowardsHigherDensity( $q_0, nlist, s,$ 
list2);
} // end adjusting the model s
mod = getModelInstances( $q_0, newlist$ );
} //end adjusting the model size
return mod;

```

---

Fig. 4.  $k$ -Nearest Neighbor based model generation with adjustment of model size.

The hyper-rectangular frames are drawn thereafter by the

minimum and maximum feature values of the instances included to the model. The procedure for the  $k$ -Nearest Neighbor based model generation is presented in Figure 4. This procedure can be used in algorithm shown in Figure 2.

$k$ -Nearest Neighbor based generation makes the method more deterministic. In this case the desired number of models is harder to obtain due to the restrictions made by  $k$ -Nearest Neighbor. It builds a model of the fixed size  $s$  and reduces the number of equal distance neighbors, which are placed to the end of list. Instead of trimming the list of neighbors this method selects  $s$  in the direction with higher density of instances in the feature space.

## 4 Implementation Issues

In this section several issues of practical implementation are considered. As a solution for the conflict between uniformity and enrichment, a parallel model generation approach is proposed in Subsection 4.1. Subsection 4.2 considers the experimental results obtained using two implementations of model generation. In Subsection 4.3 the questions related to parameter settings are described.

### 4.1 Parallel Model Generation with Closing Streams

The uniformity condition suggests that uniformity should be enforced for every particular model rating [5]. One can partition the entire collection of models into several groups containing models of the particular size and rating. The number of such groups can be calculated for specific minimum and maximum model sizes and the enrichment threshold. In the parallel model generation approach the separate streams of models with different ratings are promoted toward uniform coverage separately. However, the probabilities are updated using the entire collection of models. This approach makes it easier to generate a sufficient number of models for a more accurate prediction. With the minimum and maximum model sizes and the enrichment threshold specified, one can perform preliminary estimation for the streams that will be closed before generation starts. Namely, for every instance from  $TR$ ,  $k$  nearest neighbors within the minimum and maximum model size can be calculated in advance. The enrichment condition applied to these models suggests whether to keep this stream open or closed before generation starts. In order to reduce costs of this approach the decision about any particular stream of models can be made after the first model satisfying the enrichment condition in the stream is found.

There exists a conflict between the enrichment and the uniformity conditions applied to the model generation process that may result in stopping the model generation process before the desired number of iterations performed. The generation process may get caught in an endless loop or stop for another reason such as bad choices of the model size and the threshold on the enrichment degree, if no pre-estimation is performed. However, for  $k$ -Nearest Neighbor based generation, which made the model generation process

more deterministic, this is more likely to happen due to the restricted number of variants that start from the random instance  $q_0$ . In this case the algorithm has to be able to close streams during the generation process, specifying that all possible models satisfying the enrichment and the uniformity conditions are added to the collection and there is no new acceptable model that can be generated.

The pre-estimation for the closed streams during the parallel model generation can be performed in the following way. The consistency of model collection in the particular stream can be verified each time the generation process has unsuccessfully passed the number of iterations that is, for example, twice greater than the number of positive points in  $TR$ . This value comes from the practical experience and used in the SD implementations considered in this paper.

## 4.2 Preliminary Experimental Results

In order to investigate the interval based and the  $k$ -Nearest Neighbor based implementations of model generation in SD, a series of experiments were carried out. The data sets used in the experiments are the 2-class Kleinberg's example of square-shaped data described in [10] and the 4-class example described in [7] that has easily separable classes with non-linear boundaries (Figure 5). A half of the instances is randomly selected for  $TR$ , another half is used as  $TE$ . Both data sets are two-dimensional. For each 2-class subproblem the minority class is chosen to be enriched.

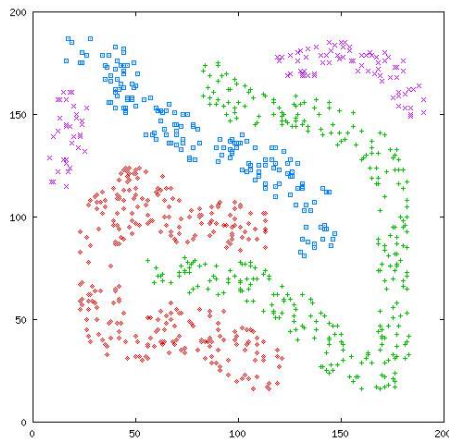


Fig. 5. The data set with four classes [7], an example of non-linearly separable classes with wide margins between classes. 3% of the original data is presented;  $TR$  -56/54/41/14,  $TE$  - 42/58/27/13 instances.

The model size is limited to 4 instances that created 4 streams for the 2-class data and 3-4 streams for different one-against-all subproblems of the 4-class data.

The accuracy obtained for the two-class data using 200 iterations of interval based model generation is 80.2469%.  $k$ -Nearest Neighbor based generation stopped after 34 iterations of generation and resulted in accuracy 74.9877%. The 2-class data set is presented with 70/92 instances in  $TR$  and 74/88 instances in  $TE$ . The combined classifier's accuracy for the 4-classes data set using interval based model generation is 60.7143%, where the accuracy is the worth for the minority class represented by 14 instances in

$TR$ . The accuracy obtained for this data set using  $k$ -Nearest Neighbor based generation is 68.3921%, for the minority class all streams were closed after 10 iterations.

The time taken to generate models for both data sets using interval based model generation is 3 times or more as the time taken using  $k$ -Nearest Neighbor based generation.

The obtained results demonstrate the need for preliminary estimation to adjust the SD parameters.

## 4.3 Pre-estimation for Parameters

Preliminary estimation is the most important and challenging part of SD implementation. There are several related things that should be defined before the algorithm starts:

- How many different models of the particular size and rating exist on the particular data set?
- Whether these models altogether can possibly supply uniform coverage of the positive (or negative) instances?
- How many of these models at minimum need to be placed to the collection of models to promote nearly uniform coverage of the positive (or negative) instances?
- What is the minimum and maximum size of a model with a particular rating that can be accepted to produce "thick" models?
- What ratings for a particular model size should be accepted (the enrichment threshold  $\beta$ )?
- What ratio of the "weak" instances should be accepted by the uniformity condition (the uniformity threshold  $\omega$ )?
- How many iterations of model generation is enough to achieve the near uniform coverage?

There are two approaches to perform pre-estimation: (1) to start with some universal parameter value and adjust it over a number of trials for a particular data set; (2) to estimate suitable values for the parameters from some computable characteristics of data geometry, such as those considered in [6]. The pre-estimation approach also depends on the particular SD implementation and the model type used.

In [11] the following pre-estimation for the interval based model generation used in the SDK implementation is suggested. By default, SDK previews the feature space prior to the start of the process of collecting models by *simultaneously adjusting both the thickness of hyper-rectangles and the number of hyper-rectangles*, so as to have a typical weak model covering a certain percentage range of the feature space.

An alternative approach is to apply measures of geometrical data complexity to perform pre-estimation. Geometrical complexity measures considered in [6] (for example, linear separability, overlap of classes, inter versus intra class proximity, density, complexity of class boundaries, etc.) can be used indirectly to aid parameter setting. Partitioning of the feature space and clustering as a proximity analysis could provide another approach to pre-estimation.

The problem of high-dimensional data known as the *curse of dimensionality* has significant impact on the performance of particular SD implementations. As the number of dimensions increase distance between instances

becomes increasingly meaningless [16]. Every additional dimension spreads out the instances. In high dimensional data instances are almost equally far apart and data becomes very sparse. A pure interval based method for model generation does not work in this case, because the chances of capturing the instances with restrictions on every dimension are nearly zero. A modified implementation requires selection of subset of features, i.e. several dimensions to perform model generation. SDK [10] selects a random subset of features according to the specified ratio at each iteration of the model generation process. This destroys the spatial data structure, but due to the large number of iterations performed (usually, more than 100,000) and the averaging effect this implementation works for high-dimensional data.

In comparison,  $k$ -Nearest Neighbor based model generation is less sensitive to the increased number of features.

## 5 Conclusions

At the present time several working implementation of the SD method exist. Success of each implementation varies from problem to problem, but all of them perform along the lines predicted by the theory. The best implementation of SD is yet to come. This research makes another step towards finding this implementation.

In this paper two implementations of SD that create hyper-rectangular models have been discussed. Each of them has its advantages and shortcomings. The  $k$ -Nearest Neighbors based implementation is more deterministic due to the restricted number of possible model variants coming from the random choice of the initial instance. It creates several restrictions, for example, the number of possible models satisfying both the uniformity and enrichment conditions appears to be less than for the interval based implementation. Not all of the potential models are reachable due to the conflict between the uniformity and enrichment conditions.

The interval based implementation requires more time for model generation, and is very sensitive to the quality of pre-estimation of the data characteristics and parameter settings, such as the minimum and maximum model size, the enrichment and uniformity thresholds.

The parallel model generation approach increases the chances to get at least one combination of positive and negative instances in a particular model size in a stream such that enough models can be collected for an acceptable accuracy level.

Both interval based and the  $k$ -Nearest Neighbor implementations are highly sensitive to the presence of irrelevant features in the data set. Both of them suffer from the “curse of dimensionality”. However, in the case of multidimensional data with relevant features,  $k$ -Nearest Neighbor demonstrates more flexibility.

The real classification problems typical for the data mining applications have some further complicated data characteristics. The data may have enormous size and large dimensionality; and may contain a mixture of various input

variables types (continuous, categorical, nominal, ordinal, interval), missing values, irrelevant and redundant variables.

One of the contemporary data mining problems is data heterogeneity. The stochastic discrimination method has a great potential for such problems since it uses a collection of highly localized models and can better adapt to heterogeneous data. Applicability of the space partitioning methods, local feature selection, and spatial clustering to aid SD in the case of heterogeneous data are currently under research by the authors.

## 6 References

- [1] R. Berlind, “An alternative method of stochastic discrimination with applications to pattern recognition”. Ph.D. dissertation, Dept. Math., State Univ. of New York at Buffalo, NY, 1994.
- [2] D. Chen, “Estimates of classification accuracies for Kleinberg’s method of stochastic discrimination in pattern recognition”. Ph.D. dissertation, Dept. Math., State Univ. of New York at Buffalo, NY, 1998.
- [3] D. Chen, X. Cheng, “A simple implementation of the stochastic discrimination for pattern recognition”, *LNCS*, vol. 1876, 2000, pp. 882-887. [*Proc. Joint IAPR Int. Workshop on Advances in Pattern Recognition*, Alicante, Spain, 2000].
- [4] D. Chen, P. Huang, and X. Cheng, “A concrete statistical realization of Kleinberg’s stochastic discrimination for pattern recognition, Part I. Two-class classification”, *The Annals of Statistics*, vol. 31, no. 5, pp. 1393-1412, 2003.
- [5] T. K. Ho, “A numerical example on the principles of stochastic discrimination”, *Tutorial lecture notes*, presented at the 2004 Int. Conf. Pattern Recognition, Cambridge, U.K.
- [6] T. K. Ho, and M. Basu, “Measuring the complexity of classification problems,” in *Proc. 15th Int. Conf. on Pattern Recognition*, vol. 2, Barcelona, 2000, pp. 43-47.
- [7] T. K., Ho and E. Kleinberg, “Building projectable classifiers of arbitrary complexity,” in *Proc. 13th Int. Conf. on Pattern Recognition*, Vienna, 1996, pp. 880-885.
- [8] E. Kleinberg, “Stochastic discrimination”, *The Annals of Math. and Art. Intel.*, vol. 1, pp. 207-239, 1990.
- [9] E. Kleinberg, “An overtraining-resistant stochastic modeling method for pattern recognition”, *The Annals of Statistics*, vol. 24, no. 6, 1996, pp. 2319-2349.
- [10] E. Kleinberg, “On the algorithmic implementation of stochastic discrimination”, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 5, pp. 473-490, May 2000.
- [11] E. Kleinberg, “A mathematically rigorous foundation for supervised learning”, *LNCS*, vol. 1857, 2001. [*Proc. 1st Int. Workshop on Multiple Classifier Systems*, Cagliari, Italy, 2000].
- [12] E. Kleinberg, and T. K. Ho, “Pattern recognition by stochastic modeling”, in *Proc. 3rd Int. Workshop on Frontiers in Handwriting Recognition*, Buffalo, 1993, pp. 175-183.
- [13] Lambert, David C. (2002). Stochastic discrimination utilities (Version 0.91) [Computer Program]. Chicago, IL.
- [14] M. Lindenbaum, S. Markovitch, and D. Rusakov, “Selective sampling for nearest neighbor classifiers”, in *Proc. 16th Nation. Conf. on Artif. Intel. and 11th Conf. on Innov. Applic. of Artif. Intel.*, Cambridge, U.K., 1999, pp. 366-371.
- [15] F. Masulli, and G. Valentini, “Comparing decomposition methods for classification”, in *Proc. 1st Int. Conf. on Knowledge-based Intelligent Engineering Systems & Allied Technologies*, Brighton, Sussex, U.K., 2000, pp. 788-791.
- [16] L. Parsons, E. Haque, and H. Liu, “Subspace clustering for high dimensional data: A review”, *SIGKDD Expl., Newsletter of the ACM Special Interest Group on Knowledge Discovery and Data Mining*, vol. 6, no. 1, pp. 90-116, 2004.
- [17] I. Skrypnik, “Combining Class Encoding and Local Feature Selection for Class Heterogeneity Decomposition”. Ph.Lic. thesis, Dept. Comp. Science and Inf. Syst., Jyväskylä Univ., Jyväskylä, Finland, 2005.