**Noname manuscript No.**
(will be inserted by the editor)

# A Sequential Sampling Strategy for Adaptive Classification of Computationally Expensive Data

**Prashant Singh · Joachim van der Herten · Dirk Deschrijver · Ivo Couckuyt · Tom Dhaene**

Received: date / Accepted: date

**Abstract** Many real-world engineering problems can be represented and solved as classification problems. These problems are typically encountered in design optimization and use expensive data, since the data are often sourced from computationally expensive simulations. It is therefore crucial to solve the classification problem using as little data as possible. This necessitates an iterative classifier construction procedure beginning with a very small training set, which is supplemented in each iteration by a small batch of new data points. The sequential sampling algorithm selects locations in the input space and the simulator calculates the corresponding class label. This paper describes a sequential sampling algorithm for adaptive classification of expensive data.

**Keywords** Adaptive Classification · Surrogate Models · Sampling Methods · Simulations · Expensive Data

## 1 Introduction

Machine learning techniques are widely used in everyday life in applications such as computer vision, information retrieval, e-commerce, medical diagnosis, natural language processing, recommender systems, web search, etc. Many problems deal with identifying a group, a category or a *class* to which a given input pattern belongs. Examples of such *classification* problems are cancer diagnosis, financial risk assessment, handwriting recognition, image scene classification, social network analysis, etc.

The solution of such problems is obtained by fitting a classification model, or a *classifier*, to a *training set*. The training set consists of a number of training *instances* or data points. Each instance has a number of *attribute* values or features, and a corresponding *class label*. The classifier is constructed using the training set, and can be used to predict class labels of new input data not present in the training set.

The training set often originates from existing databases of precomputed or recorded data such as customer data from businesses, medical records from hospitals, social networking websites, web usage logs, etc. Data are often generated as by-products of normal operations. However, in some cases data are generated on demand, e.g., by performing computer experiments.

Computer experiments are widely used to understand natural and man-made phenomena, solve engineering design optimization problems, model economic activities, etc. The need for such *computer simulations* arises because conducting physical experiments can be impractical and costly, or even impossible.

Computer simulations can offer a viable alternative, although they can be computationally expensive. The magnitude of computational cost is illustrated by Shan and Wang [2010] with examples of economic uncertainty analysis performed by NASA, and automotive crashworthiness analysis performed by Ford Motor Company. A two-factor full factorial analysis of economic uncertainty, for example, would take hundreds of years on a typical workstation, given that millions of analyses would be required to quantify the uncer-

Prashant Singh, Joachim van der Herten, Dirk Deschrijver, Ivo Couckuyt, Tom Dhaene
Department of Information Technology (INTEC)
Ghent University - iMinds
Gaston Crommenlaan 8 Blok C0 bus 201
9050 Ghent
Belgium
Tel.: +32-9-33-14986
Fax: +32-9-26-43593
E-mail: prashant.singh@intec.ugent.be

tainty. Similarly, it is estimated that a single execution of a simulation for automotive crashworthiness analysis takes on average 98 hours to complete. This scale of computational expense would imply a duration of 12 years to pronounce crashworthiness of an example (Shan and Wang [2010]). In order to alleviate the computational burden in these cases, there is a need to train regression and classification models using as few training instances as possible.

In this work, a sequential sampling strategy for adaptive classification is proposed. It begins with an initial small set of training data, and iteratively adds more training points at well-chosen locations in the input space. The sampling algorithm picks additional points in a sequential way based on existing data and total data budget (E.g., number of allowed simulations). The proposed adaptive classification algorithm is demonstrated on an non-linear analytical example and a structural design problem.

The paper is organised as follows. Section 2 introduces simulation-based engineering, and classification models in the context of engineering. Section 3 explains the proposed sequential sampling algorithm. The algorithm is demonstrated on analytical examples in Section 4. Section 5 concludes the paper.

## 2 Simulation-based Engineering

Simulation models are widely used in engineering design optimization. They can be used to analyze functional dependencies between design variables, perform *what-if* analyses, optimization, study uncertainty quantification, etc.

Simulations can be expensive to perform. If so, it can be useful to have a cheap(er) replacement known as a *surrogate* model, which can be used in lieu of the expensive simulation. This can help to speed-up the optimization process.

The problems in the field of engineering are mostly regression problems with a continuous *response* or output of the simulations. Examples include the electrical networks, magnetic and electric fields associated with an electronic (sub)system, flutter speed of an aircraft wing, etc. Popular surrogate model types used in this context are Artificial Neural Networks (ANN) (Hagan et al. [1996]), Kriging models (Sacks et al. [1989]), rational functions, splines and Support Vector Regression (SVR) (Drucker et al. [1997]).

The simulations typically used to source the data for model construction are expensive. In such cases it is useful to train the model using as few data samples as possible. In the regression context, this problem is known as *adaptive sampling* or sequential design (Deschrijver et al. [2011], Gorissen et al. [2010], van der Herten et al. [2015]).

### 2.1 Classification and Simulation-based Engineering

Some engineering problems have a discrete output and require the solution of classification problems. Examples in literature of engineering problems solved using classification include solving constrained optimization problems (Basudhar et al. [2012], Handoko et al. [2008]), finding quasi-optimal regions (Singh et al. [2013]), determining food quality (Cen and He [2007]), measuring analog circuit performance (De Bernardinis et al. [2003]), detecting faults in aircraft engines (Rausch et al. [2004]), etc.

Therefore, there is a need for sequential sampling algorithms in a classification context. Model building in a sequential design context involves training a model with a small set of samples chosen according to an *initial design*, and iteratively selecting batches of new samples to add to the training set using an adaptive sampling algorithm. The goal of the sampling algorithm is to intelligently choose additional samples for training which increase classifier accuracy. The initial design is a set of points chosen according to a specific design (Latin Hypercube (Husslage et al. [2011]), Factorial (Morris [1991]), etc.) or even randomly.

### 2.2 Active Learning

Sequential sampling is closely related to the field of *active learning* (Cohn et al. [1996]). However, there are subtle differences. Active learning is semi-supervised and assumes a *fixed* unlabeled dataset $\mathsf{U}$, from which the learning algorithm must *sub-sample* data points to learn from. The learner can only select unlabelled data points $\mathbf{x}_i \in \mathsf{U}$. This is different from a typical engineering problem where the attributes are usually continuous design parameters and the sampling algorithm has the freedom to pick a data point anywhere in the design (input) space. Essentially, the choice in the case of active learning involves choosing the next training point(s) from a finite set of data points to learn from, while in case of sequential sampling, the choice involves picking the location in the design space where the next training point(s) should be generated. This difference necessitates a separate approach towards sampling algorithms.

### 2.3 Adaptive Classification

In the context of this work, the term adaptive classification is intended to mean classifier construction using training data obtained sequentially from a sampling algorithm. Consider a training set $\mathsf{S}$ in some input space $\mathcal{X} \subseteq \mathbb{R}^d$ spanning $d$ attributes, and some output space $\mathcal{Y}$. The output space is $\mathcal{Y} = \{0, 1\}$ for a binary classification problem and $\mathcal{Y} = \{1..K\}$ for a $K$-class classification problem. The training set is denoted as $\mathsf{S} = (X, Y) \in \mathcal{X} \times \mathcal{Y}$ where $X$ consists of $n$ data points represented as vectors $\{\mathbf{x}_1 ... \mathbf{x}_n\}$ and $Y$ consists of class labels $\{y_1 ... y_n\}$. The classifier $h : \mathcal{X} \rightarrow \mathcal{Y}$ predicts the class label of a given input pattern $\hat{\mathbf{x}}$ as $\hat{y} = h(\hat{\mathbf{x}})$. For details of the classifier training process, the reader is referred to Bousquet et al. [2004].

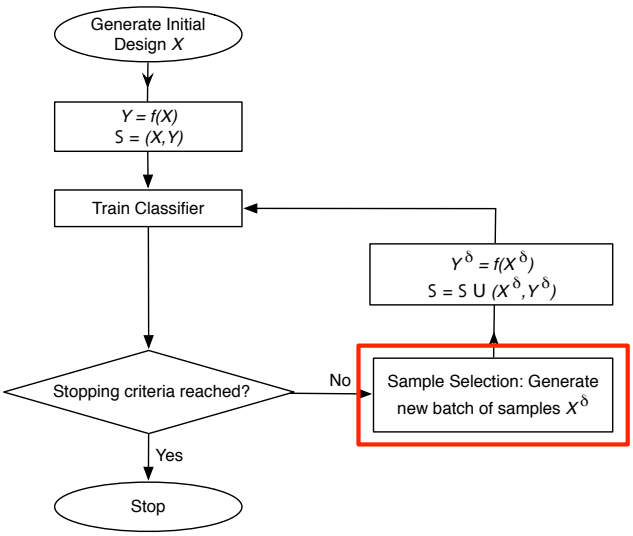



Fig. 1: Adaptive classification flowchart.

The flowchart of the adaptive classification process is shown in Figure 1. The initial training set $\mathsf{S}$ is obtained by generating a set $X$ of $b$ points in the input (or design) space, and evaluating $X$ using the expensive simulator to obtain the corresponding class labels $Y$. Subsequently, $\mathsf{S} = (X, Y)$ is used as the training set to build a classification model. The focus of this work is only on the sequential sampling process (the outlined box in Figure 1), with the aim of obtaining an accurate model. The model is assumed not to contribute to the sequential sampling process, while the sampling algorithm aims at selecting samples which offer maximal training information for construction of the model.

Assuming that the total number of allowed function evaluations is $n$, the sequential sampling algorithm selects a new batch of informative samples $X^\delta$ of size $\delta$ at well-chosen locations in the input space. The simulator evaluates $X^\delta$ resulting in class labels $Y^\delta$. The training set $\mathsf{S}$ is updated as:

$$Y^\delta := f(X^\delta),$$
$$\mathsf{S} := \mathsf{S} \cup (X^\delta, Y^\delta).$$

The classifier is rebuilt using the updated training set $\mathsf{S}$. This process is iterated over $\lfloor \frac{n-b}{\delta} \rfloor$ times until the number of allowed simulations is exceeded, or one of the stopping criteria (if specified) has been reached. Stopping criteria may include: achieving desired accuracy, exceeding allowed time duration, etc.

## 3 Neighborhood-Voronoi Sequential Sampling Algorithm

In this section, a new approach for sequential sampling in a classification context is proposed. The term sequential implies that the sampling algorithm is dynamic. The algorithm chooses additional samples in each iteration based on certain criteria. These criteria may be based on uncertainty, or accuracy of the intermediate classification models, space-fillingness of the input space, satisfaction of constraints, or other problem-specific performance criteria. Each of the potential goals described above will need a dedicated sequential sampling algorithm. The goal of the proposed algorithm is to train an accurate classifier while minimizing the number of training samples.

The algorithm presented in this work is solely data driven. The data are collected, analysed, and new data points are chosen in a sequential manner. No intermediate (classification) models are required during the sampling process. Thus, the proposed algorithm is independent of any particular classifier.

The Neighborhood-Voronoi algorithm is based on the LOLA-Voronoi algorithm proposed by Crombecq et al. [2011], with modifications made to handle classification problems instead of regression. The algorithm aims to balance *exploration* of the input space and *exploitation* of regions near the class boundaries which are prone to misclassification. In the following subsections, the Neighborhood-Voronoi sampling algorithm is explained by separately discussing the Neighborhood (exploitation) and Voronoi (exploration) components.

### 3.1 Exploitation

The difficult-to-learn regions for a classifier are often near the class boundaries. The exploitation component makes sure that samples are chosen more densely in

these interesting areas. A local neighborhood $N$ of size $m$ is computed for each instance $\mathbf{x}_i, \forall i \in 1, ..., n$ as:

$$N(\mathbf{x}_i) = \{\mathbf{x}_{i1}, \mathbf{x}_{i2}, ..., \mathbf{x}_{im}\} \subset X_r = \{\mathbf{x}_{ij}\}_{j=1}^{m},$$

where $X_r = X \setminus \{\mathbf{x}_i\}$. To ensure that all directions around the instance $\mathbf{x}_i$ are covered uniformly, $N$ is chosen according to optimal *adhesion* and *cohesion*.

- Cohesion makes sure that the neighbors are as close to $\mathbf{x}_i$ as possible. It is defined as the average minimum distance of neighboring points from $\mathbf{x}_i$. The cohesion of a neighborhood $N$ with respect to the fixed instance $\mathbf{x}_i$ is defined as:

$$C(N(\mathbf{x}_i)) = \frac{1}{m}\sum_{j=1}^{m} \|\mathbf{x}_{ij} - \mathbf{x}_i\|_2.$$

- Adhesion ensures that the neighbors are as far away from each other as possible. It is defined as the average minimum distance of neighbors from each other. The adhesion of a neighborhood $N$ with respect to the fixed instance $\mathbf{x}_i$ is defined as:

$$A(N(\mathbf{x}_i)) = \frac{1}{m}\sum_{j=1}^{m} \min_{l \neq j} \|\mathbf{x}_{ij} - \mathbf{x}_{il}\|_2.$$

Ideally, a neighborhood $N$ should have a low value of cohesion $C(N(\mathbf{x}_i))$ and a high value of adhesion $A(N(\mathbf{x}_i))$. Finding such a neighborhood becomes a multi-objective optimization problem involving minimising $C(N(\mathbf{x}_i))$ and maximising $A(N(\mathbf{x}_i))$ simultaneously, given a discrete set of candidate neighborhoods. Ideally, if the neighbors of the reference point $\mathbf{x}_i$ could be chosen freely, they will be chosen such that they have equal cohesion contribution and form a $m-$sided regular polygon. The problem is extended to placing $m$ points in an ideal configuration on a $d-$dimensional hyper-sphere such that the adhesion value $A(N(\mathbf{x}_i))$ of the reference point $\mathbf{x}_i$ is maximized. This is an open problem in mathematics (Croft et al. [1991]).

Since there is no optimal solution to the problem of placing $m$ points on a $d-$dimensional hypersphere (Saff and Kuijlaars [1997]), a subproblem with a known solution is considered. This concerns the special case when $m = 2d$. Intuitively, for a one-dimensional case, $m = 2$ and the configuration will involve placing one point on either side of the reference point $\mathbf{x}$. In the two-dimensional case, $m = 4$ and the points will form a square around the reference point. For $d-$dimensions, the optimal configuration is a $d-$cross-polytope (Cohn and Kumar [2007]) which contains all points obtained by permuting the coordinates $(\pm 1, 0, 0, ..., 0)$. The cross-polytope configuration maximizes adhesion (Cohn and Kumar [2007]).

**The cross-polytope ratio:** Having established that for points lying on a hyper-sphere, the cross-polytope is the optimal configuration which maximizes adhesion, it can be inferred that any given neighborhood with cohesion $C(N(\mathbf{x}_i))$ must always have an adhesion value $A(N(\mathbf{x}_i))$ lower than that of the cross-polytope with radius $C(N(\mathbf{x}_i))$. For a cross-polytope, the distance between points is $\sqrt{2}$ times the distance from the origin (the reference point) for any dimension higher than 1. This implies that $\sqrt{2}C(N(\mathbf{x}_i))$ is the absolute upper bound for adhesion value of any neighborhood with cohesion $C(N(\mathbf{x}_i))$. Therefore, the following measure $R(N(\mathbf{x}_i))$ can be used to gauge how closely a neighborhood resembles a cross-polytope:

$$R(N(\mathbf{x}_i)) = \begin{cases} \frac{A(N(\mathbf{x}_i))}{\sqrt{2}C(N(\mathbf{x}_i))} & , d > 1 \\ 1 - \frac{|x_{i1}+x_{i2}|}{|x_{i1}|+|x_{i2}|+|x_{i1}-x_{i2}|} & , d = 1. \end{cases}$$

The exception for the one-dimensional case is due to the fact that the distance of the two points from each other is twice the distance from the reference point (Crombecq et al. [2011]).

A neighborhood score that combines adhesion and cohesion can be used to assign scores to neighborhoods:

$$S(N(\mathbf{x}_i)) = \frac{R(N(\mathbf{x}_i))}{C(N(\mathbf{x}_i))}.$$

This measure will prefer neighborhoods that lie close to the reference point $\mathbf{x}_i$ and resemble a cross-polytope. $S$ can be used as a criterion to choose $N$ for all instances.

After such a neighborhood is constructed, the *class disagreement* $\chi$ corresponding to the sample $\mathbf{x}_i$ belonging to the neighborhood $N$ is calculated according to the formula:

$$\chi(\mathbf{x}_i) = \begin{cases} 1, \alpha > 1, \\ 0, \alpha = 1. \end{cases} \quad (1)$$

where $(1 \leq \alpha \leq K)$ is the number of unique class labels in $N$. An observation with a higher value of $\chi$ is surrounded by samples having differing class labels, and needs to be sampled more intensely as it is located along the class boundaries.

Algorithm 1 describes the pseudocode for the exploitation component of the Neighborhood-Voronoi algorithm. The algorithm begins by updating the state of the samples selected by the algorithm in the previous iteration. Each new sample $\mathbf{x}_\delta$ is considered as a candidate neighbor for each processed sample $\mathbf{x}$ and vice-versa. The class disagreement scores for these samples are then updated according to Equation 1. After processing all previously unprocessed samples, the metric $\chi$ is calculated for each sample in $X$ which reflects the exploitation score of the sample in question. Finally,

**Algorithm 1** Pseudocode for the exploitation component of the Neighborhood-Voronoi sequential sampling algorithm. $X$ consists of all points processed by the algorithm previously. $X_\delta$ is the set of points selected by the algorithm in the previous iteration which are yet to be processed. $\delta$ is the number of new samples to be selected by the algorithm.

```
for all x_δ ∈ X_δ do
    for all x ∈ X do
        Evaluate membership of x_δ for neighborhood N(x) of x
        Evaluate membership of x for neighborhood N(x_δ)
        Update class disagreement information for x and x_δ
    end for
    X ← X ∪ x_δ
end for
for all x ∈ X do
    Calculate class disagreement score for x
end for
Identify neighborhoods corresponding to δ highest ranked samples in X
Select new samples in these neighborhoods
```

each of the neighborhoods corresponding to the top $\delta$ samples ranked according to $\chi$ are chosen to generate a new sample in.

### 3.2 Exploration

The exploration component identifies regions in the input space that are prone to under-sampling, or under-representation. Such regions have a low density of points and a mechanism to identify these regions is required.

A Voronoi tessellation is a well-known way to partition a space based on density (Aurenhammer [1991]). Assuming that our training set $X \subset \mathcal{X}$ in Euclidean space, the *Voronoi cell* $C_i \subset \mathcal{X}$ of the point $\mathbf{x}_i$ contains all points in $\mathcal{X}$ which lie closer to $\mathbf{x}_i$ than any other point in $X$. The Voronoi tessellation corresponding to $X$ consists of all Voronoi cells $\{C_1, C_2, ..., C_n\}$ which tessellate the complete space $\mathcal{X}$. To define Voronoi cells formally, the notion of dominance (Aurenhammer [1991], Crombecq et al. [2011]) is used.

**Dominance:** Given two distinct instances $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{X}$, the dominance of the instance $\mathbf{x}_i$ over the instance $\mathbf{x}_j$ is defined as the subset of the plane being at least as close to $\mathbf{x}_i$ as it is to $\mathbf{x}_j$ (Crombecq et al. [2011]):

$$dom(\mathbf{x}_i, \mathbf{x}_j) = \{\mathbf{x} \in \mathcal{X} \mid \|\mathbf{x} - \mathbf{x}_i\|_2 \leq \|\mathbf{x} - \mathbf{x}_j\|_2\}.$$

The plane $dom(\mathbf{x}_i, \mathbf{x}_j)$ is half-closed, bounded by the perpendicular bisector of $\mathbf{x}_i$ and $\mathbf{x}_j$. The bisector is called the *separator* of $\mathbf{x}_i$ and $\mathbf{x}_j$ which separates all points in $\mathcal{X}$ closer to $\mathbf{x}_i$ as opposed to $\mathbf{x}_j$. The Voronoi cell $C_i$ corresponding to the instance $\mathbf{x}_i$ is the part of the design space $\mathcal{X}$ with is dominated by $\mathbf{x}_i$ over all other instances in $X$:

$$C_i = \bigcap_{\mathbf{x}_j \in X \setminus \{\mathbf{x}_i\}} dom(\mathbf{x}_i, \mathbf{x}_j).$$

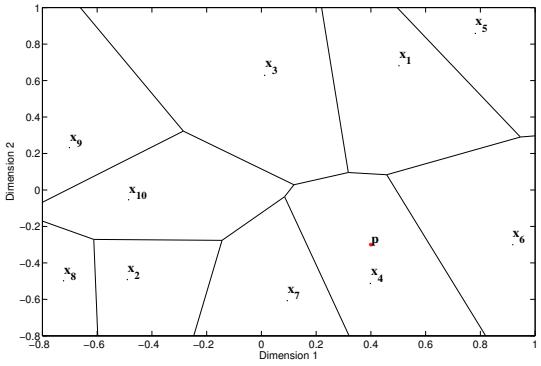



Fig. 2: The bounded Voronoi tessellation of a set of points $\{\mathbf{x}_i\}_{i=1}^{10}$. The test point $\mathbf{p}$ lying in the Voronoi cell corresponding to $\mathbf{x_4}$ lies closer to $\mathbf{x_4}$ than any other point.

Figure 2 shows the Voronoi tessellation of a set $\{\mathbf{x}_i\}_{i=1}^{10}$ of randomly generated instances. The test instance $\mathbf{p}$ is closer to $\mathbf{x}_4$, and so are all points in $\mathcal{X}$ in the Voronoi cell corresponding to $\mathbf{x}_4$. It is also apparent from Figure 2 that larger Voronoi cells correspond to regions in the design space that are sampled more sparsely. To fully explore the design space $\mathcal{X}$, new samples should be chosen in Voronoi cells with a large volume. For example, generating a new sample point or instance in the Voronoi cell corresponding to $\mathbf{x}_3$ will be more beneficial in terms of space-fillingness as compared to sampling the Voronoi cell corresponding to the instance $\mathbf{x}_8$. Therefore, a way to compute the hypervolume of Voronoi cells is required in order to compare them.

Voronoi tessellations are geometric duals of Delaunay triangulations. The Voronoi tessellation of a set of points $X$ can be obtained from the Delaunay triangulation of $X$ in $\mathrm{O}(n)$ time (Aurenhammer [1991]). Computing the volume of Voronoi cells is harder, since the Voronoi cells near the border of $\mathcal{X}$ are unbounded. These Voronoi cells will therefore have infinite volume. Hence, the border-lying Voronoi cells must first be bounded before their volume can be computed.

As this is complex, the volume of Voronoi cells is approximated using a Monte Carlo approach described in Algorithm 2, since only the relative differences in volume of the Voronoi cells are important, and computing the exact volume is computationally very expensive. A

**Algorithm 2** Pseudocode for the exploration component of the Neighborhood-Voronoi sequential sampling algorithm. $X$ consists of all points that have to be ranked by the algorithm according to their respective Voronoi cell size.

```
T ← L random points ∈ 𝒳
V ← [0, 0, ..., 0]
for all t ∈ T do
    d ← ∞
    for all x ∈ X do
        if ‖x − t‖ < d then
            x_closest ← x
            d ← ‖x − t‖
        end if
    end for
    V[x_closest] ← V[x_closest] + 1/L
end for
```

large number of random uniformly distributed test samples $T = \{\mathbf{t}_l\}_{l=1}^{L}$ are generated in $\mathcal{X}$. The minimum distance between each test point $\mathbf{t}_l$ and existing instance $\mathbf{x}_i$ is calculated. The test point is then assigned to the instance closest to it. By having enough test points, it is possible to estimate the volume of each Voronoi cell. The reader is referred to Crombecq et al. [2011] for details of the algorithm to approximate the hypervolume of each Voronoi cell.

The exploration metric $\psi$ of an instance $\mathbf{x}_i$ is defined as the ratio of the estimated volume of Voronoi cell $C_i$ containing $\mathbf{x}_i$ with respect to the combined volume of all Voronoi cells in the design space $\mathcal{X}$:

$$\psi(\mathbf{x}_i) = \frac{\mathrm{Vol}(C_i)}{\mathrm{Vol}(C_1) + \mathrm{Vol}(C_2) + ... + \mathrm{Vol}(C_n)}.$$

A higher value of $\psi(\mathbf{x}_i)$ implies that the corresponding Voronoi cell $C_i$ is large, whereas a smaller value of $\psi(\mathbf{x}_i)$ implies that $C_i$ is smaller. The sampling algorithm should focus on cells with a higher value of $\psi$ since they might be under-sampled.

### 3.3 Combining exploitation and exploration score

After obtaining the two metrics $\chi$ and $\psi$ for exploitation and exploration respectively, the algorithm (Algorithm 3) assigns a combined score $\Lambda$ for each existing sample $\mathbf{x} \in X$ as:

$$\Lambda(\mathbf{x}) = \chi(\mathbf{x}) + \psi(\mathbf{x}).$$

The algorithm ranks all samples in $X$ in order of how well each sample ranks in exploitation and exploration according to the criterion $\Lambda$. The top $\delta$ samples in $X$ are then selected and a new point is generated near each of these samples such that the generated point is as far away from other existing samples as possible (maximizing the minimum distance to other existing samples).

**Algorithm 3** Pseudocode for the Neighborhood-Voronoi sequential sampling algorithm. $\delta$ is the number of new samples to be selected by the algorithm.

```
for all x ∈ X do
    Compute χ(x)
    Compute ψ(x)
    Compute final ranking Λ(x) = χ(x) + ψ(x)
end for
Sort X according to Λ
for i = 1 to δ do
    x_δ ← generate a sample near x_i farthest from other samples
    X_δ ← X_δ ∪ x_δ
end for
```

## 4 Examples

### 4.1 Example: Non-Linearly Separable Classification Problem

A Gaussian function centered at $(x_1', x_2') = (0, 0)$ having a standard deviation $\sigma = 5$ is defined as:

$$f(x_1, x_2) = \frac{(x_1 - x_1')^2 + (x_2 - x_2')^2}{\sigma},$$

$$dom(f(x_1, x_2)) = \{x_1, x_2 \in [-5, 5]\}.$$

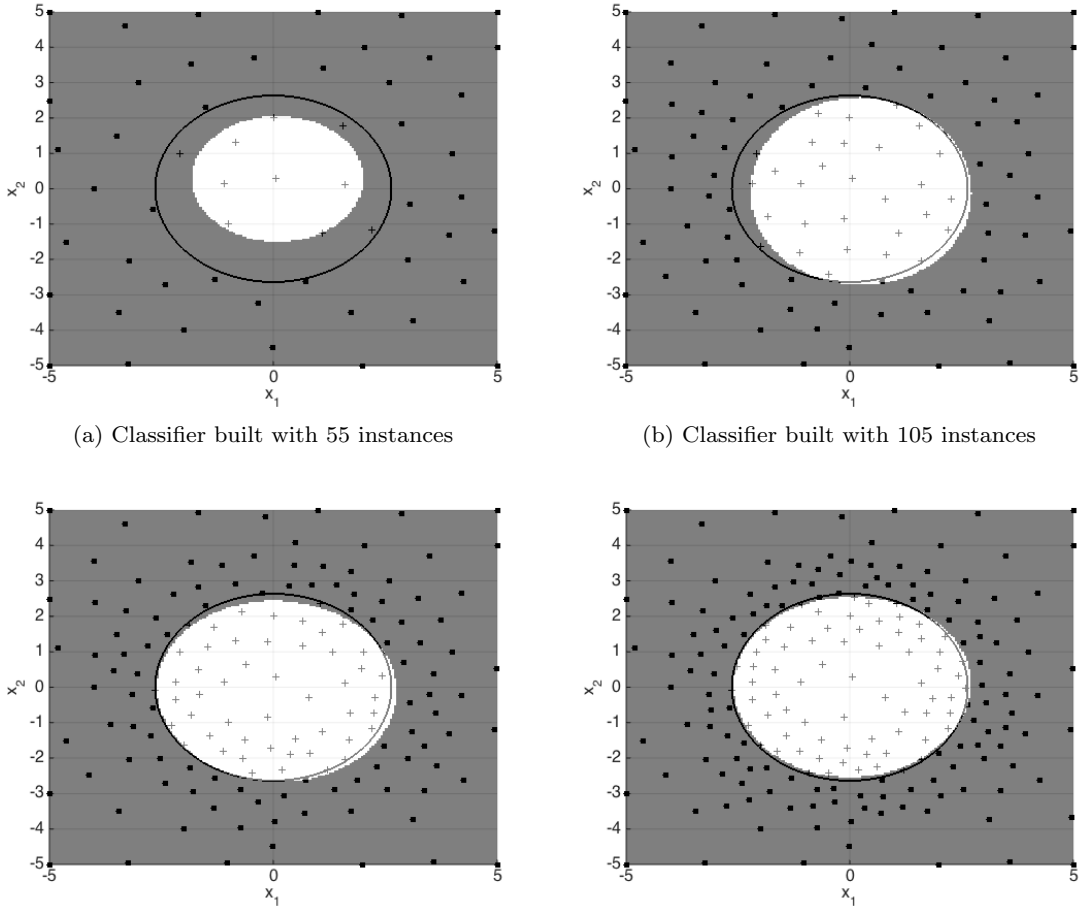
The problem involves finding the region in the input space which corresponds to function values within 50% of the highest possible function value ($f_{max} = 1$). The classification problem is defined as:

$$y_i = \begin{cases} 1, & f(\mathbf{x}_i) \in [0.5, \infty), \\ 0, & f(\mathbf{x}_i) \in (-\infty, 0.5). \end{cases}$$

A classifier is trained over instances obtained according a Latin Hypercube design of $b = 15$ points (including the corner points of the design space). The Neighborhood-Voronoi sequential sampling algorithm is used to select additional samples iteratively in batches of $\delta = 10$ each. The total number of function evaluations allowed is $n = 205$. The classifier chosen for the experiment is Support Vector Machine (SVM) classifier (Chang and Lin [2011]), though the Neighborhood-Voronoi algorithm is independent of the classifier used. All experiments have been performed using the SUrrogate MOdeling (SUMO) toolbox (Gorissen et al. [2010]) for MATLAB®[1].

The results of applying the Neighborhood-Voronoi algorithm can be seen in Figure 3. There is a large discrepancy between true and learned class boundaries in

[1] MATLAB, The MathWorks, Inc., Natick, Massachusetts, United States.




(a) Classifier built with 55 instances

(b) Classifier built with 105 instances

(c) Classifier built with 155 instances

(d) Classifier built with 205 instances

Fig. 3: Non-Linearly Separable Classification Problem: The sampling performed by the Neighborhood-Voronoi algorithm for the Gaussian function. The black circle is the true class boundary. The learned positive class is represented by the white region, while the learned negative class is represented by the grey region. The dots are the instances in the training set for that particular iteration.

the initial iterations. In subsequent iterations, the classifier boundary is refined by selecting samples near the boundary. The accuracy of the classifier over 200 randomly generated test points was 99% with Precision and Recall being 1 and 0.99 respectively. The evolution of classifier accuracy with increasing number of training instances over a static set of test instances can be seen in Figure 4. The accuracy rises rapidly between 35 and 65 training samples, after which it begins to stabilise.

### 4.2 Example: Nowacki Beam Problem

A constrained multi-objective optimization problem described by Nowacki [1980] is now considered. The aim is to design a tip-loaded encastre cantilever beam (Fig. 5) minimizing the cross-sectional area and bending stress subject to certain constraints. The rectangular beam has length $l = 0.5$ m and is subjected to a tip-load $F = 5$ kN. The design variables are the height $h$ and breadth $b$ of the beam. The optimization problem can be formulated as:

| | | | |
|---|---|---|---|
| $\underset{b,h}{\text{Min}}$ | $A, \sigma_B$ | s.t. | $\delta \leq 5\text{mm}$ |
| | | | $\sigma_B \leq \sigma_Y$ |
| for | $20\text{ mm} < h < 250\text{ mm}$ | | $\tau \leq \sigma_Y/2$ |
| | $10\text{ mm} < b < 50\text{ mm}$ | | $h/b \leq 10$ |
| | | | $F_{CRIT} \geq f \times F$ |

with $A = b \times h$ being the cross-sectional area of the beam, $\sigma_B = 6Fl/(bh^2)$ the bending stress, $\delta = Fl^3/(3EI_Y)$ the maximum tip deflection, $\sigma_Y$ the yield stress of the material, $\tau = 3F/(2bh)$ the maximum

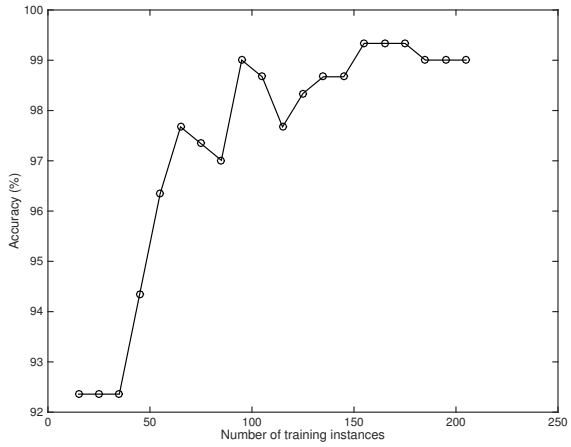



Fig. 4: Non-Linearly Separable Classification Problem: The evolution of classifier accuracy with respect to number of training instances.

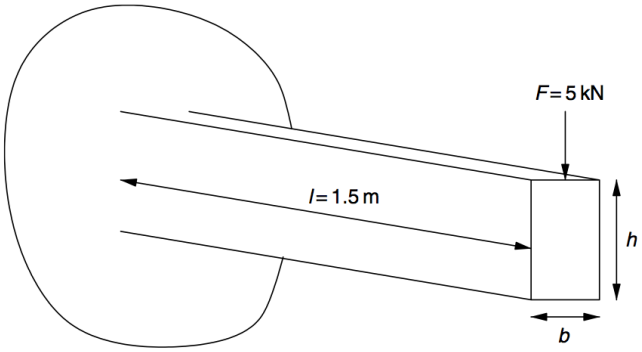



Fig. 5: The Nowacki Beam Problem.

allowable shear stress, $h/b$ the height-to-breadth ratio, and $F_{CRIT} = (4/l^2)\sqrt{GI_T EI_Z/(1-\nu^2)}$ the failure force of buckling. Here, $I_T = (b^3h + bh^3)/12$, $I_Z = b^3h/12$, $I_Y = bh^3/12$, and $f$ is a safety factor of two. The material under consideration is mild steel with yield stress $\sigma_Y = 240$ MPa, Young's modulus $E = 216.62$ GPa, $\nu = 0.27$ and shear modulus $G = 86.65$ GPa.

Instead of finding the optima, the problem of finding the *region of feasibility* in the design space meeting all constraints is considered. This can be also seen as an inverse problem of finding a region (quasi-optimal region) in the design space corresponding to desired (known) output. For complex problems, a practitioner might find it useful to find a small region in the design space containing possible solutions first, and concentrating future efforts in only that region. This kind of *domain reduction* can be very useful (Spaans and Luus [1992]) while solving expensive constrained optimization problems. Finding the feasible region efficiently will save the practitioner a lot of time and effort.

The problem of finding the feasible region is solved using adaptive classification. The problem can be cast as a classification problem with the class label $y_i$ assigned to instance $\mathbf{x}_i = (b, h)$ as:

$$y_i = \begin{cases} 1, \delta \leq 5\text{mm}; \sigma_B \leq \sigma_Y; \tau \leq \sigma_Y/2; \\ \quad h/b \leq 10; F_{CRIT} \geq f \times F, \\ 0, \text{ otherwise.} \end{cases}$$

This formulation aggregates all the constraints into a single class definition. Regression approaches like Surrogate-Based Optimization (SBO) involve construction of a regression model (like Kriging) per objective and per constraint (Couckuyt et al. [2013]). Therefore, in this case a typical SBO approach requires five regression models for constraints, and two additional regression models for the objectives. Adaptive classification allows us to have a single classification model for all constraints instead of five regression models.

An Artificial Neural Network (ANN) classifier available from the WEKA data mining software (Hall et al. [2009]) was used to model the constrained problem. The initial design was a Latin Hypercube of 20 instances. The Neighborhood-Voronoi sequential sampling algorithm was used to select 10 new samples in each iteration and the total number of allowed function evaluations was 200.

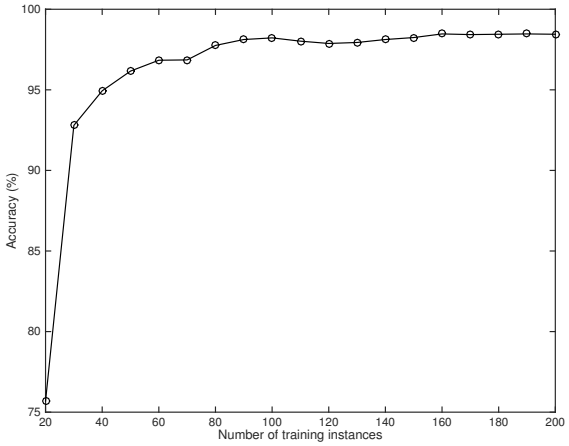
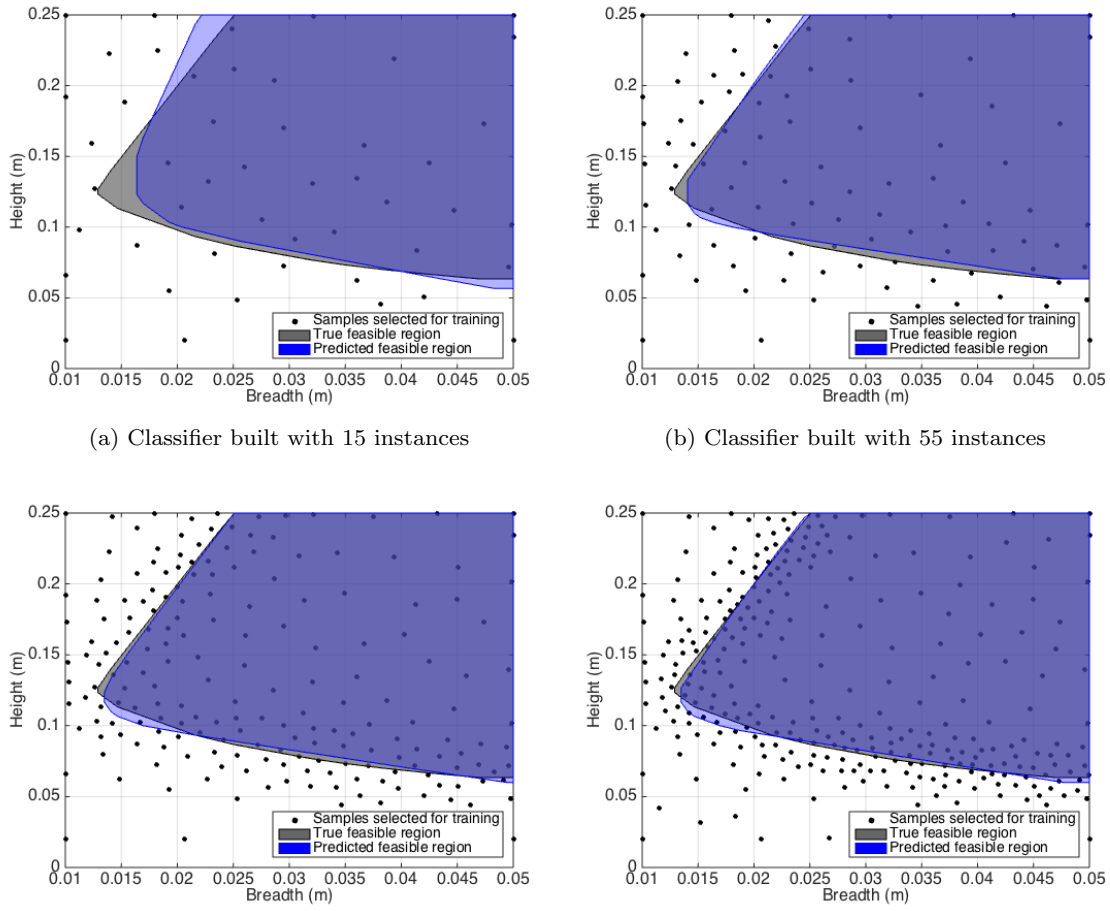



Fig. 7: Nowacki Beam Problem: The evolution of classifier accuracy with respect to number of training instances.

The result can be seen in Figure 6. It is observed that samples have been selected densely along the edge




(a) Classifier built with 15 instances

(b) Classifier built with 55 instances

(c) Classifier built with 105 instances

(d) Classifier built with 155 instances

Fig. 6: Nowacki Beam Problem: The sampling performed by Neighborhood-Voronoi algorithm for the Nowacki Beam Problem. Each figure depicts an Artificial Neural Network (ANN) model. The dark grey shaded region is the actual feasible region. The blue region represents the feasible region learned by the classifier. The dots are the instances in the training set for that particular iteration.

of the feasible region, which is desirable (Schoenauer and Michalewicz [1996]). The true and learned class boundaries are very close after training with only 155 instances. The final classifier built using 200 samples has an *accuracy* of 98.45%, *precision* of 0.9899 and *recall* of 0.9732. Figure 7 shows the smooth evolution of classifier accuracy with increasing number of training instances. Although only binary classification problems are discussed as examples for the purpose of exposition, the proposed algorithm functions as described for multi-class classification problems as well.

## 5 Conclusion

Sequential sampling schemes are often used for minimising the number of data points used to train a regression model. Minimising data points required for training the model is essential as problems (e.g., from engineering design optimization) often involve computationally expensive simulations. Many engineering problems also involve training a classification model. A novel sequential sampling strategy for training classification models is presented in this paper that minimizes the number of training points needed to obtain an accurate classifier. The proposed sequential sampling algorithm is illustrated on a non-linear analytical example and a structural design problem.

**Acknowledgements** This work was supported by the Interuniversity Attraction Poles Programme BESTCOM initiated by the Belgian Science Policy Office, and the Research Foundation Flanders (FWO-Vlaanderen). Ivo Couckuyt is a Postdoctoral Researcher of FWO-Vlaanderen.

## References


Franz Aurenhammer. Voronoi diagramsa survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

Anirban Basudhar, Christoph Dribusch, Sylvain Lacaze, and Samy Missoum. Constrained efficient global optimization with support vector machines. *Structural and Multidisciplinary Optimization*, 46(2):201–221, 2012.

Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. Introduction to statistical learning theory. In *Advanced Lectures on Machine Learning*, pages 169–207. Springer, 2004.

Haiyan Cen and Yong He. Theory and application of near infrared reflectance spectroscopy in determination of food quality. *Trends in Food Science & Technology*, 18(2):72–83, 2007.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/ cjlin/libsvm`.

David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

Henry Cohn and Abhinav Kumar. Universally optimal distribution of points on spheres. *Journal of the American Mathematical Society*, 20(1):99–148, 2007.

Ivo Couckuyt, Dirk Deschrijver, and Tom Dhaene. Fast calculation of multiobjective probability of improvement and expected improvement criteria for pareto optimization. *Journal of Global Optimization*, pages 1–20, 2013.

Hallard T Croft, Kenneth J Falconer, and Richard K Guy. *Unsolved problems in geometry*. Springer, 1991.

Karel Crombecq, Dirk Gorissen, Dirk Deschrijver, and Tom Dhaene. A novel hybrid sequential design strategy for global surrogate modeling of computer experiments. *SIAM Journal on Scientific Computing*, 33 (4):1948–1974, 2011.

Fernando De Bernardinis, Michael I Jordan, and A SangiovanniVincentelli. Support vector machines for analog circuit performance representation. In *Design Automation Conference, 2003. Proceedings*, pages 964–969. IEEE, 2003.

Dirk Deschrijver, Karel Crombecq, Huu Minh Nguyen, and Tom Dhaene. Adaptive sampling algorithm for macromodeling of parameterized-parameter responses. *IEEE Transactions on Microwave Theory and Techniques*, 59(1):39–45, 2011.

Harris Drucker, Chris JC Burges, Linda Kaufman, Alex Smola, Vladimir Vapnik, et al. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1997.

Dirk Gorissen, Ivo Couckuyt, Piet Demeester, Tom Dhaene, and Karel Crombecq. A surrogate modeling and adaptive sampling toolbox for computer based design. *The Journal of Machine Learning Research*, 11:2051–2055, 2010.

Martin T Hagan, Howard B Demuth, Mark H Beale, et al. *Neural network design*. Pws Pub. Boston, 1996.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

Stephanus Daniel Handoko, Kwoh Chee Keong, and Ong Yew Soon. Using classification for constrained memetic algorithm: A new paradigm. In *IEEE International Conference on Systems, Man and Cybernetics, 2008. SMC 2008*, pages 547–552. IEEE, 2008.

Bart GM Husslage, Gijs Rennen, Edwin R van Dam, and Dick den Hertog. Space-filling latin hypercube designs for computer experiments. *Optimization and Engineering*, 12(4):611–630, 2011.

Max D Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33 (2):161–174, 1991.

Horst Nowacki. Modelling of design decisions for cad. In *Computer Aided Design Modelling, Systems Engineering, CAD-Systems*, pages 177–223. Springer, 1980.

Randal Rausch, Daniel E Viassolo, Aditya Kumar, Kai Goebel, Neil Eklund, Brent Brunell, and Pierino Bonanni. Towards in-flight detection and accommodation of faults in aircraft engines. In *AIAA 1st Intelligent Systems Technical Conference, Chicago, IL, September*, pages 20–22, 2004.

Jerome Sacks, William J Welch, Toby J Mitchell, and Henry P Wynn. Design and analysis of computer experiments. *Statistical science*, pages 409–423, 1989.

Edward B Saff and A BJ Kuijlaars. Distributing many points on a sphere. *The Mathematical Intelligencer*, 19(1):5–11, 1997.

Marc Schoenauer and Zbigniew Michalewicz. Evolutionary computation at the edge of feasibility. In *Parallel Problem Solving from NaturePPSN IV*, pages 245–254. Springer, 1996.

Songqing Shan and G Gary Wang. Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. *Structural and Multidisciplinary Optimization*, 41(2):219–241, 2010.

Prashant Singh, Dirk Deschrijver, Davy Pissoort, and Tom Dhaene. Adaptive classification algorithm for emc-compliance testing of electronic devices. *Elec-*

*tronics Letters*, 49(24):1526–1528, 2013.

Robert Spaans and Rein Luus. Importance of search-domain reduction in random optimization. *Journal of Optimization Theory and Applications*, 75(3):635–638, 1992.

Joachim van der Herten, Ivo Couckuyt, Dirk Deschrijver, and Tom Dhaene. A fuzzy hybrid sequential design strategy for global surrogate modeling of high-dimensional computer experiments. *SIAM Journal on Scientific Computing*, 37(2):A1020–A1039, 2015.