Anton Nikulin

Smart Prototype Selection for Machine Learning based on Ignorance Zones Analysis

Master's Thesis in Information Technology

March 25, 2018

University of Jyväskylä

Faculty of Information Technology

Author: Anton Nikulin

Contact information: annikuli@student.jyu.fi

Supervisor: Prof. Vagan Terziyan

Title: Smart Prototype Selection for Machine Learning based on Ignorance Zones Analysis

Project: Master's Thesis

Study line: Web Intelligence and Service Engineering

Page count: 62+5

Abstract: The size of databases has been considerably growing over recent decades and Machine Learning algorithms are not ready to process such large volume of information. Being one of the most useful algorithms in Data Mining the Nearest neighbor classifier suffers from high storage requirements and slow response when working with large data sets. Prototype Selection methods help to alleviate this problem by choosing a subset of data with a smaller size. In this thesis, the overview of existing instance selection methods is provided together with the introduction of a new approach. The majority of current methods select a subset experimentally by checking whether certain point affects classification accuracy or not. The new approach, presented in this thesis, is based on analyzing data set instances and choosing prototypes based on discovered ignorance zones. The results obtained from the analysis show that the proposed method can effectively decrease the size of the data set while maintaining the same classification accuracy with the Nearest neighbor classifier. In addition, it allows removing noisy data making the decision boundaries smoother.

Keywords: Prototype selection, Nearest neighbor, Ignorance zones, Data reduction, Classification

Glossary

CNN	Condensed Nearest neighbor
DROP	Decremental Reduction Optimization Procedures
DRT	Data Reduction Techniques
FCNN	Fast Condensed Nearest neighbor
GAN	Generative Adversarial Network
NN	Nearest neighbor
PCA	Principal Component Analysis
PG	Prototype Generation
PS	Prototype Selection
PSC	Prototype Selection by Clustering
SNN	Selective Nearest neighbor

List of Figures

List of Tables

Contents

1 Introduction

In Supervised Learning, classification is one of the most widespread techniques that helps to identify to which category new observation belongs. After learning from a training set, that contains a collection of training examples called instances, a machine learning algorithm tries to predict a class for a new input vector. A large number of classification algorithms calculates the distance between the new input vector and stored instances when predicting the class label. Such algorithms that do not build classification model in advance but make decisions completely relying on existing prototypes from the training set are often named as Lazy Learners or Instance-Based Learners (Brighton and Mellish 2002).

One the most popular Lazy Learner algorithm is the k Nearest Neighbor classifier (kNN). Although being invented more than 50 years ago (Cover and Hart 1967) the kNN classifier has been ranked one of the top 10 Data Mining methods (Wu et al. 2008). The logic of the algorithm is simple: for each new point x it finds k nearest neighbors from the training set and chooses the most common class according to majority vote rule. The kNN algorithm learns very fast because it does not process training instances but only retains all of them in its memory. Being an effective classifier for many applications, kNN suffers from multiple weaknesses that can prevent successful application of the algorithm (Kononenko and Kukar 2007):

- high storage requirements (especially for large-scale databases) to store all data set exemplars in memory;
- demand for powerful computational resources in order to calculate distances between a new input vector and all original prototypes;
- low noise tolerance because all data instances stored in memory are considered relevant and outliers can harm classification accuracy.

In general, there are two widespread approaches to solve these problems: improve the calculation speed of nearest neighbors or reduce the training set by selecting only a small fraction of instances or features (Ougiaroglou, Evangelidis, and Dervos 2015). Within the first group Indexing methods (Yu et al. 2001) and Cluster-based methods (Hwang and Cho 2007) gained

the most interest. Both groups aim to reduce the cost of the nearest neighbor searches by applying indexing or clustering procedures as pre-processing steps and making the selection of neighbors faster at classification stage (Ougiaroglou, Evangelidis, and Dervos 2015). Although these methods help to decrease classification response time they do not solve the problem of noisy instances as the training set remains unchanging.

The second group of methods is called Data Reduction Techniques (DRTs). DRTs aim to reduce the size of the training set by building a representation set of a smaller size called a condensing set. These methods not only help to make the classification process faster but also aim to build a better version of the training set by eliminating noisy instances and making decision boundaries smoother. DRTs can be divided into two subgroups, which are called Prototype Selection (PS) and Prototype Generation (PG) algorithms. PG methods (also known as Prototype Abstraction) create a training set by building new artificial instances and replacing original prototypes (Triguero et al. 2012). PS algorithms, on the contrast, form a condensing set by selecting a subset of original points leaving them invariant. Although some researchers include PG into PS, in this thesis we will focus on PS methods that do not modify original prototypes.

Related works comparing classification accuracy and retention percentage of different Prototype Selection methods show that there is no clear winner and no evidence to use one method for all data sets (Garcia et al. 2012). Depending on the primary optimization criterion such as storage reduction, classification error rate or noise tolerance PS methods show distinct results. In addition, the structure of data sets has a big impact on algorithm's work and some researchers advice to analyze data set structure manually before applying specific PS method (Brighton and Mellish 2002).

In this thesis, we suggest a new Prototype Selection method based on ignorance zones analysis. Most of the current methods choose prototypes based on experimental approach without analyzing the training set. The algorithm presented in this thesis is, on the contrary, based on understanding data set structure, discovering areas of confusion called ignorance zones and making intelligent decisions based on found insights. The main research questions stated in this thesis are:

- how to approach the ignorance discovery in databases;
- how to benefit from the discovered ignorance in classification and Prototype Selection.

The rest of this thesis is organized as follows. Chapter 2 gives an overview of existing Prototype Selection methods, contains theoretical background and presents the most popular algorithms. Chapter 3 describes the idea of ignorance zones, defines their meaning in Data Analysis and presents an algorithm for their discovery. Chapters 4 and 5 apply knowledge about ignorance zones for classification, present new Prototype Selection methods, and compare their effectiveness with two modes of operation: batch and incremental. The idea of Prototype Selection with an adversarial process and its experimental results are presented in Chapter 6. Finally, Chapter 7 concludes this thesis with achieved results.

2 Prototype Selection

The main goal of Prototype Selection is to extract the smallest set from the original database that would be able to predict classes with the same (or even higher) accuracy as the original training set. Usually, Prototype Selection algorithms aim to discard the following two types of instances (Brighton and Mellish 2002):

- harmful instances (noisy data and outliers) that tend to attribute queries to an incorrect class if the algorithm relies on them;
- redundant instances (superfluous and excessive points) that do not help in making classification decisions. They can be replaced with other more relevant prototypes without loss of classification accuracy.

2.1 Direction of search

When creating a subset of instances in order to form the training set there are also different directions the search can proceed. Usually, three orders are defined: incremental, decremental, and batch (Wilson and Martinez 1997). Incremental technique starts with an empty training set and gradually adds new prototypes if they meet search criteria. For incremental search, the order of instances is very important because the probability of being selected for the same exemplar varies a lot depending on whether it is located in the beginning or at the end of the data set. Decremental search operates the other way: starts with a complete set of data and continuously removes instances on each step. Decremental rule is often computationally more expensive than incremental algorithms, but "the application of a decremental algorithm can result in greater storage reduction and/or increased generalization accuracy" (Wilson and Martinez 1997). Batch mode first marks instances that do not comply with the removal criteria and then removes all of them at once. Similar to decremental algorithms, batch processing suffers from increased time complexity in comparison with incremental algorithms (Wilson and Martinez 2000).

In addition to three search orders described above some authors also distinguish a couple of other variations, such as mixed and fixed techniques. Mixed search starts with a preselected

subset of training exemplars selected randomly or by incremental or decremental process and new exemplars are added or removed later. Due to this flexibility, the algorithm can improve the training set by removing bad prototypes and adding better ones with time. Fixed search policy can be defined as a subfamily of mixed search with the special constraint that the total number of additions and removals always remain the same. This limitation guarantees that the number of final prototypes is defined at the beginning and stays the same during the search process.

2.2 Type of selection

Depending on whether the algorithm removes central points or border instances scientists distinguish three strategies used for prototype selection: edition methods, condensation methods, and hybrid methods. The first category increases classifier effectiveness by removing noisy instances and achieving smooth decision boundaries. The second category follows the idea that central points do not influence classification decisions, so it generates the training set by skipping excessive prototypes located inside class boundaries. Mixed method aggregates both of these techniques trying simultaneously to get rid of noisy data and maintain smooth boundaries between classes (Garcia et al. 2012).

While edition methods try to increase accuracy they do not solve the problem of slowness as usually there is not many noisy data in the data set. On the contrary, condensation technique aims to make classification process faster achieving the same level of accuracy. There is a clear trade-off between these two types of algorithms depending on how training set reduction algorithms are compared. Wilson pointed out, "these [criteria] include speed increase (during execution), storage reduction, noise tolerance, generalization accuracy, time requirements (during learning), and incrementality" (Wilson and Martinez 2000). Each of prototype selection algorithms aims to improve one or maximum two criteria, so it is important to take this into consideration when comparing the accuracy of different methods.

2.3 Condensation methods

The Condensed Nearest neighbor (CNN) rule presented in the 1960s became the first official prototype selection technique for the nearest neighbor decision rule (Hart 1968). Hart presented a definition of a consistent subset and defined it as a subset that correctly classifies all the remaining points from the training set using the NN rule. The CNN rule, presented in the same research work, explained a simple algorithm to create consistent subsets from the training data. It starts from an empty subset and sequentially checks every point from the training set: if the point can be correctly classified by a subset it is dropped, otherwise, added to the subset. The algorithm finishes its work either when all points outside the subset can be successfully classified by the NN rule, or no points outside the subset are left (Hart 1968).

The CNN algorithm presented a way to create a subset of the original sample set but produced data was not guaranteed to be a minimal consistent subset. As a result, other condensation methods were proposed to decrease the size. One of them was the Reduced Nearest Neighbor (RNN) rule that aimed to create even smaller subset than the CNN. The RNN rule takes a group of points chosen by the CNN rule as an initial sample set, and following decremental approach tries to remove more points one after the other: if all patterns are classified correctly without specific point then this prototype can be dropped, otherwise it is put back to the subset (Gates 1972). Although it is more computationally expensive to produce a set of RNN comparing to CNN, the final subset should be smaller and therefore it demands less computing and storage at the classification stage (Amal and Riadh 2011).

A couple of years later an article about the Selective Nearest neighbor (SNN) Decision Rule was published. The SNN suggested a newly calculated selective subset that had one principal difference comparing to the consistent subset: it must fulfill all requirements of the consistent subset and "all samples must be nearer to a condensed neighbor of the same class than to any sample of the other class" (Ritter et al. 1975). The authors also presented an algorithm for finding the selective subset from the database, that became a good alternative to the previous algorithms as it managed to select fewer points closer to the class borders and make the selection process less dependent on the order of point processing.

Recent algorithms still use the same concept of consistent subsets, but try to speed up the selection process. The algorithm called the Fast Condensed Nearest neighbor (FCNN) rule uses a theorem based on class centroids and Voronoi cells (Angiulli 2005). Another example is Prototype Selection by Clustering (PSC) that suggests using clustering techniques in order to divide the training set into regions and analyze each region (cluster) independently (Olvera-López, Carrasco-Ochoa, and Martínez-Trinidad 2010).

2.4 Hybrid methods

Although condensation methods show a good decrease in the size of training sets their main philosophy of choosing points located closer to the decision boundary arises problems when working with noisy data. Hybrid methods aim to maintain benefits of condensation methods while solving the problem of noise sensitivity.

A collection of new heuristics methods called Decremental Reduction Optimization Procedures (DROP) were proposed to solve the flaws of condensation algorithms (Wilson and Martinez 1997). All versions of the algorithm rely on two main concepts: associates and nearest neighbors. The definition of associates are inverse to nearest neighbors: instances P that have Q as one of their nearest neighbors are called associates of Q.

The first algorithm called DROP1 is identical to RNN and tries to remove noisy points and choose non-noisy border prototypes. The second version of the algorithm is improved by ordering the removal process and aims to remove instances located further from the decision boundary first.

The third version of the algorithm called DROP3 is considered to be one of the best of Prototype Selection methods regarding their reduction power (Garcia et al. 2012). Comparing to DROP2 it does the noise filtering before sorting the prototypes and removing the center points. It helps to smooth the decision boundary and avoids the problem of "overfitting" (Wilson and Martinez 2000).

2.5 Prototype Selection and Big Data

Although many PS algorithms work well in term of selecting a representative subset one of their main drawbacks is complexity. Majority of instance selection methods have quadratic $O(n^2)$ complexity and very few of them can do it in log-linear $O(n \log n)$ time (Arnaiz-González et al. 2016). Such complexity makes prototype selection methods challenging for big data sets containing thousands and millions of instances.

Stratification is one of the approaches that makes instance selection applicable to the massive data sets (Cano, Herrera, and Lozano 2005). The underlying idea of stratification is to split the large size data set into multiple disjoint subsets of a smaller size and apply the prototype selection method to each of them. The conducted results showed that stratification is a robust tool to face the scaling problems of instance selection algorithms (Derrac, García, and Herrera 2010), but with the increase of the data set size the idea of joining each partial solution into a global subset can result in having redundant and noisy instances.

The alternative way to handle the large-scale data sets containing millions of prototypes is a MapReduce-based framework that distributes the work of instance selection methods across a cluster of computing machines (Triguero et al. 2015). The MapReduce paradigm helps to parallelize the prototype reduction computation and merge multiple computed subsets into a global one based on fusion reduce type.

3 Ignorance Zones

3.1 Open world assumption in Machine Learning

Prevailing majority of Machine Learning algorithms work with training sets as with closed world databases. By following the closed world assumption they assume that "if no proof of a positive ground literal exists, then the negation of that literal is assumed true" (Reiter 1981).

An alternative to the closed databases is the open world assumption. In open world negative data is listed explicitly in the database and "queries may be either looked-up or derived from the data and the axioms" (Minker 1982). There is no assumption that certain data is considered to be false just because positive prove was not found.

For many business problems, such as finding a flight route based on the database of flights and cities, the closed world assumption works good because their domain implies the truth of negative facts (Reiter 1981). Though for more complex problems such as classification of malignant tumors, where creating an all-embracing data set is very expensive or simply impossible, inferring the result based on the absence of data might not be the optimal approach.

We believe that by utilizing not only the closed world assumption but also the open world approach existing Machine Learning algorithms could potentially make smarter decisions and improve their results. Instead of considering missing data as a false signal, algorithms can utilize the knowledge of unknown regions. The driving force of learning is the process of analyzing already available data and looking for areas where the data set has the least amount of information. Such areas without data are called ignorance or confusion zones. These areas of emptiness do not give much knowledge taken separately but their form and size can help to understand accessible data better and make it analysis faster.

In this chapter, we describe the idea of ignorance zones and provide the algorithm for their discovery in two-dimensional data. This algorithm represents one building block used to tackle more complex problems in the following chapters. The examples and algorithms described in this thesis are presented based on two-dimensional data sets to ease visualization

and computation analysis, but in general, the same ideas can apply to n-dimensional prototypes too.

3.2 Ignorance zones discovery

A considerable part of Machine Learning tasks (such as Classification or Clustering analysis) works with labeled data where each instance either already has or should be attributed to some class. Points of the same class form a cluster that has certain unique characteristics that are different from other classes. As clusters possess different properties in Euclidean space they are commonly separated from each other by some sort of void. When there are no data instances inside the void there is no evidence to say what is located there. Such areas of emptiness located between known clusters represent the concept of ignorance or confusion zones.

In two-dimensional space, an ignorance zone represents a circle. The geometrical figure of a circle is chosen because its main property that all points are equally distant from the center. The center of ignorance zone is called focus and it represents a place of the most confusion. It is completely unknown what happens there because several points from different classes are equidistant from the focus.

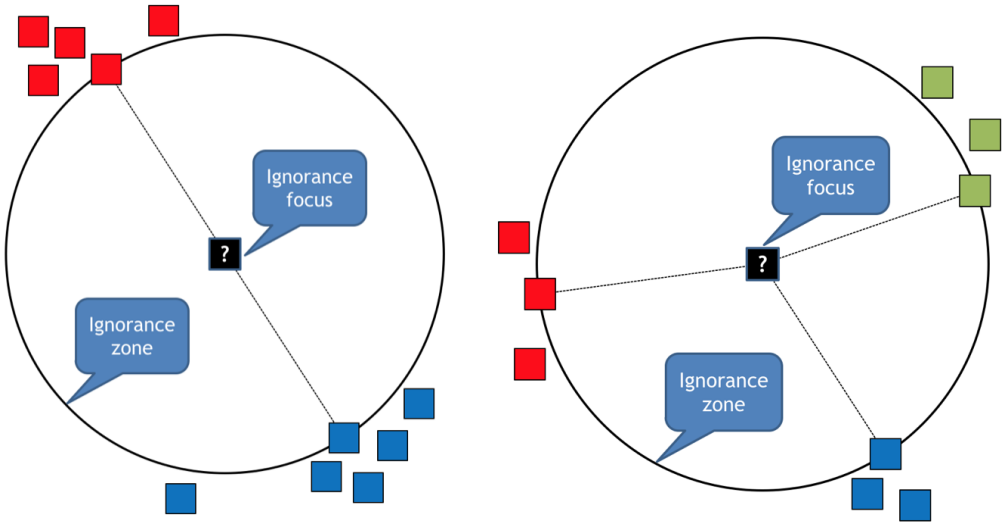

Figure 1. Two- and three-point ignorance zones and ignorance focuses.

Discovered ignorance focuses are very valuable because classification algorithms tend to fail

in these areas. Several classes have the same probability of being located there that makes the classifier give ambiguous output labels. Ignorance zones are also a type of decision boundaries (decision surfaces) because they represent an area of feature space where the transition from one class to the other happens.

The algorithm generates circles based on two or three data points: in case of two points a line between them should be a circle's diameter, in case of three the circle should circumscribe all of them. A circle becomes an ignorance zone if the following criteria are met:

- each of the points forming the circle belongs to a different class;
- no known data points are located inside the circle.

Although in some situations a circle can be built around four and more points (for example, around a square) it is not a general case, so the algorithm is restricted to work only with pairs and triples of data because it is always possible to make a circle around them.

Algorithm 1: Calculating 2-point Ignorance Zones

FindIgnoranceZones (X, Y)
- **inputs :** A training set $X = \{X_1, \ldots, X_n\}$ with coordinate vectors and $C = \{c_1, \ldots, c_n\}$ with class labels
- **output:** The set of ignorance zones Z
- $Z \leftarrow \emptyset$;
- **foreach** $X_i \in X$ **do**
  - **foreach** $X_j \in X$ **do**
    - **if** $c_i \neq c_j$ **then**
      - $X_{zone} \leftarrow (X_i + X_j)/2$;
      - $R_{zone} \leftarrow distance(X_i, X_j)/2$;
      - **if** *there is no points from X inside circle* (X_{zone}, R_{zone}) **then**
        - Add (X_{zone}, R_{zone}) to Z;
- **return** Z;

In this thesis, the algorithm is presented to work with two-dimensional data. Being one of the limitations of this thesis, it helps to focus on the design of the algorithm rather than efficiency of its implementation. It is easier to validate the correctness and usefulness of the algorithm while working with low-dimensional data sets. The philosophy of ignorance zones and the discovery algorithm can also be expanded to work in a space with higher dimensions.

For example, in three-dimensional space ignorance zones will represent spheres, and in the geometry of higher dimensions, a hypersphere will be a representation of a confusion area.

Designing the algorithm is an iterative process where the initial version gets improved while the model is being tested and its issues are identified. Analyzing whether specific changes give better or worse results is hard to do with real data sets because they contain outliers and noisy data. In addition to using databases from real problems, we created a set of artificial data sets based on simple geometrical figures (for example, square, circle, ring). Although such perfect data sets have simple linear decision boundaries and do not occur in reality with their help it is possible to see how the algorithm behaves in an ideal environment and ascertain that the core idea is correct.

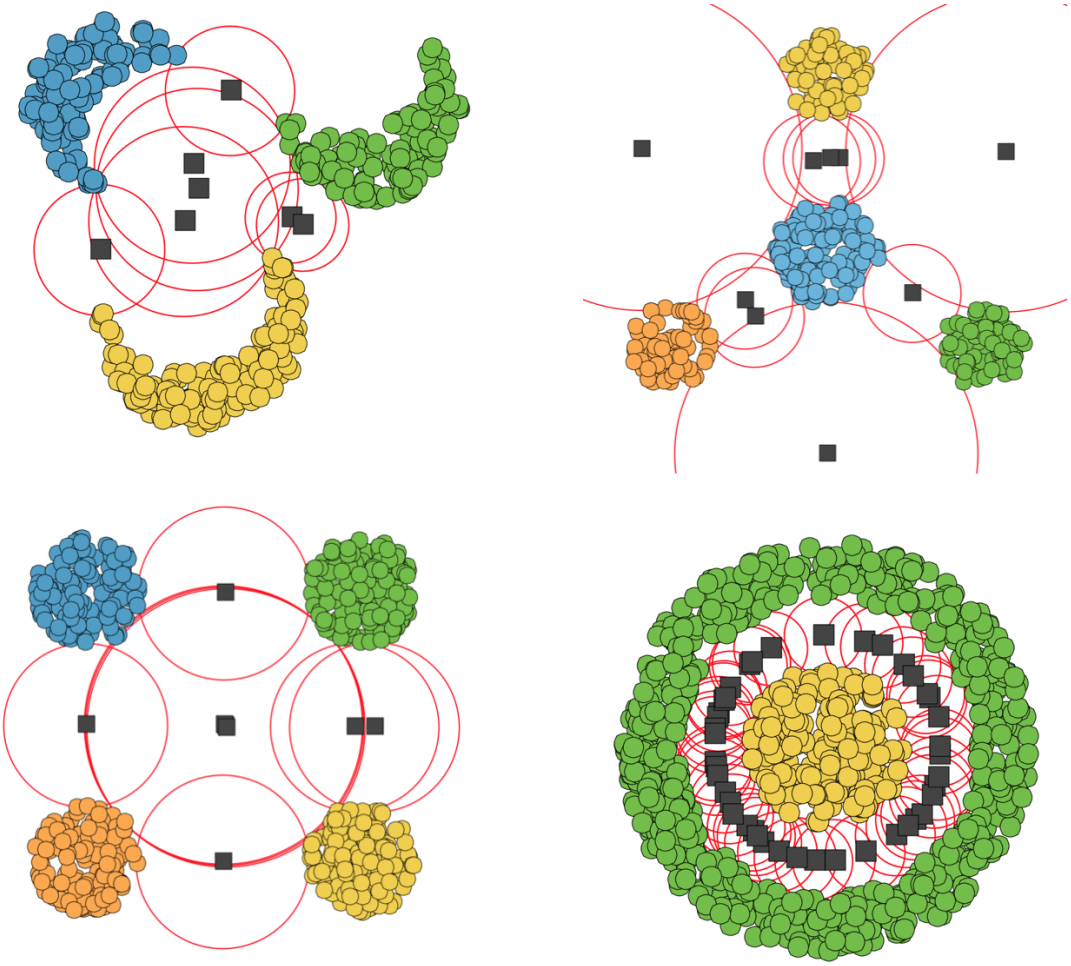

Figure 2. Ignorance zones (red circles) and ignorance focuses (black squares) discovered in artificial data sets.

In Figure 2 you can see four examples of these databases and ignorance zones discovered there. In each of these cases, ignorance focuses truly represent points of confusion as they are located at the exact same distance from two or three various classes and there is no evidence to say what happens there.

3.3 Prototype Selection with ignorance zones

One potentially useful application for ignorance zones is PS for classification. A problem of classification is directly connected with producing decision boundaries that help to distinguish classes. Decision boundaries representing places of the most confusion are conceptually similar to the definition of ignorance zones: it is unknown to which class points located there belong to as they have equal probability to be any of several classes.

In case of k-NN classifier, the class is defined by k known points that are located closest to the input vector. Due to this fact, the presence of instances situated close to decision boundaries in the representation set is a necessary condition for accurate classification. Prototypes located inside data clusters usually have the same class as points around them and do not influence classification decisions much.

For ignorance zones, points located closest to decision boundaries are their parents. Choosing parents of confusion zones as prototypes must be a good way to reduce the size of training set while maintaining classification accuracy. Similar approaches based on proximity graphs such as relative neighbors (Jaromczyk and Toussaint 1992) and Gabriel neighbors (Toussaint 1980) were also successfully applied for PS (José Salvador Sánchez, Pla, and Ferri 1997).

Idea of relative neighbors is defined as two points x_i and x_j from the data set E such that distance between them is smaller than distance from any point $x_k \in E$ to any of them (Toussaint 1980). In other words, relative neighbors can be defined as follows:

$$\begin{gathered}(x_i, x_j) \in E \\ \Leftrightarrow dist(x_i, x_j) \leq max(dist(x_i, x_k), dist(x_j, x_k)) \\ \forall x_k \in E, k \neq i, j\end{gathered}$$

Their geometrical form is called "lune" (Jose Salvador Sánchez, Pla, and Ferri 1997) and

they visually represent a disjoint intersection of two spheres centered in relative neighbors and having radii equal to their distance.

Two points x_i and x_j from the data set E are said to be Gabriel neighbors if their diametrical sphere does not contain any other point $x_k \in E$ (Jaromczyk and Toussaint 1992).

$$\begin{gathered}(x_i, x_j) \in E \\ \Leftrightarrow dist(x_i, x_j)^2 \leq dist(x_i, x_k)^2 + dist(x_j, x_k)^2 \\ \forall x_k \in E, k \neq i, j\end{gathered}$$

Gabriel neighbors are analogous to ignorance zones calculated based on pairs of points: two points produce an ignorance zone if there are no other points inside their circle.

Similar to how Gabriel and relative neighbors were successfully used to select a consistent subset of prototypes for the NN rule (Jose Salvador Sánchez, Pla, and Ferri 1997), we ran experiments of using parents of ignorance zones as prototypes for 1-NN classifier with the Euclidian distance metric. Next chapters describe details and present results of these experiments.

4 Batch Prototype Selection with ignorance zones

4.1 Methodology of experiments

Experiments have been run over eight data sets of different complexity from the UCI repository (Blake 1998). Details of individual data sets are presented in Table 1. Normalization (min-max scaling) was applied to all data set features because feature scaling has an effect on k-NN algorithm with the Euclidean distance measure.

Since ignorance zones presented in this thesis work with two-dimensional data only, feature selection or dimensionality reduction is applied to all data sets as a pre-processing step. If two best features selected from the data set based on a univariate chi-squared χ^2 statistical test are representative, we select them for classification. Otherwise, Principal Component Analysis (PCA) method is used to produce a pair of features well capturing the variance of the original data set. The two dimensions, selected by χ^2 statistical test, are normalized, so the normalization is also applied to two principal components produced by PCA. Dimensionality reduction technique (PCA or χ^2 test) and proportion of variance explained by two PCA components are presented in Table 1 for each data set.

Table 1. Description of Data Sets

Data set	**#Exemplars**	**#Attributes**	**#Classes**	**Dim. Reduction Type**	**Prop. of Variance Explained**
Iris	150	4	3	χ^2 test	-
Wine	178	13	3	χ^2 test	-
Pima	768	8	2	χ^2 test	-
Breast Cancer	699	9	2	PCA	0.76
Ionosphere	351	34	2	PCA	0.42
Glass	214	9	7	PCA	0.63
Bupa	345	6	2	PCA	0.6
Transfusion	748	4	2	PCA	0.93

For each data set, 10-fold cross-validation is applied by dividing the whole database into ten equal parts and in turn using one block as a testing set and the remaining nine as training sets. Cross-validation is repeated ten times and results for each data set are calculated as means over one hundred experiments.

The proposed algorithms (Ignorance Zone Discovery, Batch PS, Incremental PS, and Professor-Student) and the experiments, validating their work and effectiveness, are implemented in Python 2.7. Snippets of principal parts of the code are shown in Appendices A and B. We use NumPy (eg. “NumPy - NumPy” 2017) for efficient management of multi-dimensional arrays and basic functionality of linear algebra. Classification and dimensionality reduction are implemented with the help of open source SciPy (eg. “SciPy.org - SciPy.org” 2018) and scikit-learn (eg. “scikit-learn: machine learning in Python” 2018) packages. Matplotlib (eg. “Matplotlib 2.2.2 documentation” 2018) and seaborn (eg. “seaborn: statistical data visualization” 2018) libraries are used for visualization.

Both the arithmetic and the harmonic means were used as two kinds of averages to estimate found results. For positive data sets the arithmetic mean always gives the best results from different kind of averages due to big impact of large outliers, thus we also used harmonic mean for comparison. Opposite to the arithmetic average the harmonic mean aims "to mitigate the impact of large outliers and aggravate the impact of small ones" (Machiwal and Jha 2012). The harmonic mean of positive real numbers $(x_1, x_2, \ldots, x_n)$ is calculated as:

$$H(x_1, x_2, \ldots, x_n) = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}.$$

One of the limitations of the harmonic mean is its ability to work with positive indices (qualitatively) only (e.g., classification accuracy). In our experiments, negative variables (qualitatively), such as error rate or percentage of retention, were aggregated, so the contraharmonic mean was chosen as an average method. The philosophy of the contraharmonic mean is the same as of the harmonic mean but for qualitatively negative metrics: it is always higher or equal to the arithmetic mean and it is as high above the arithmetic mean as the arithmetic mean is above the harmonic mean. For a variable x, consisting of a set of positive real

numbers $(x_1, x_2, \ldots, x_n)$, the contraharmonic mean is calculated as:

$$C(x_1, x_2, \ldots, x_n) = \frac{x_1^2 + x_2^2 + \cdots + x_n^2}{x_1 + x_2 + \cdots + x_n}.$$

4.2 Batch Prototype Selection with 1-NN

Ignorance zones are located close to decision boundaries that have a huge influence on right classification decisions. This section presents an experiment where a prototype subset is selected as parents of ignorance zones. The algorithm discovers available grey zones from the full database and chooses a subset of data by selecting parents of all ignorance zones. This logic supplies the representative subset of prototypes located close to decision boundaries and ignores points located inside the cluster. PS based on parents of ignorance zones is analogous to Gabriel neighbors idea that was also employed as a PS method (Jaromczyk and Toussaint 1992). Both approaches focus on sets of points whose diametrical sphere does not contain any other points.

Figure 3 shows what prototypes are selected to the smart subset of the complete data set as a result of proposed PS method. The picture shows that points located next to decision boundaries are retained and prototypes surrounded by points of the same class are ignored. In practice, this logic means that the more outliers and noisy points the database has the higher percentage of retention for Prototype Selection is.

In order to evaluate whether the suggested method can be successfully used for PS, we compare error rate and data retention percentage of three classifiers. The first classifier uses all available data as a training set (100% of data retention). The second classifier uses a smart PS algorithm and is trained on a subset containing parents of ignorance zones. The third model uses the same amount of prototypes as the second version but selected randomly.

In theory, it is fair to assume that the first classifier should have the lowest error rate as it uses all available knowledge. In this case, high classification accuracy is compensated by low decision-making speed because no data reduction is applied and full data set is used for analysis. The second classifier is expected to work worse (or similar) in comparison to the first one but better than the third one. The subset of prototypes selected with smart

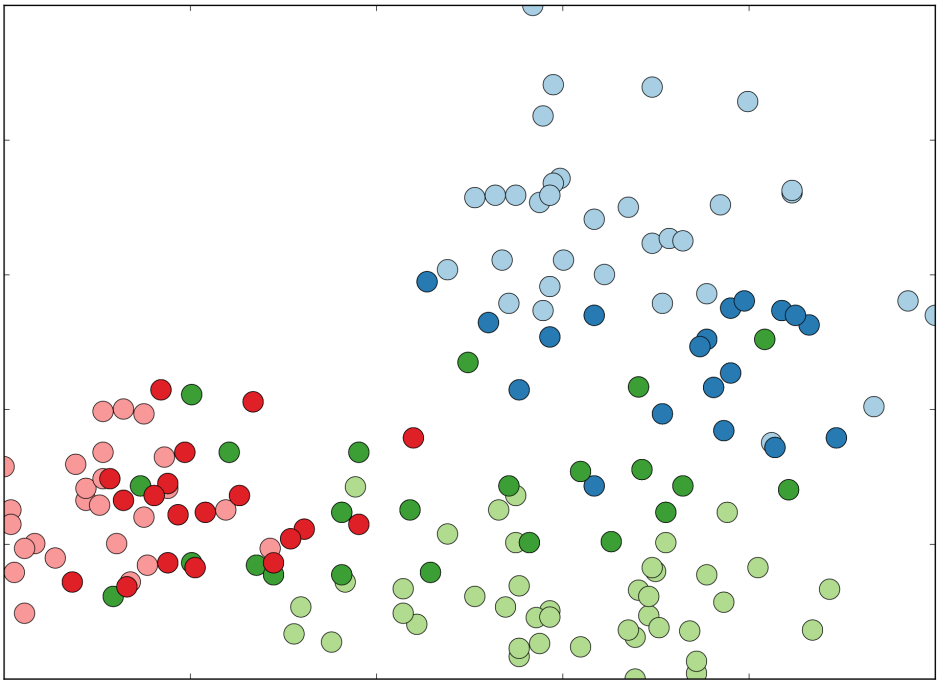

Figure 3. Parents of ignorance zones selected as a result of Prototype Selection in Wine data set.

heuristics should define database structure better than the randomly chosen subset. Though in practice not all assumptions are proved to be true and the results for some data sets are, on the contrary, opposite to what is expected. A summary of classification results are shown in Table 2 and Table 3.

Classification error rates for some databases comply with initial assumptions: the classifier trained on a smart subset of data (parents of ignorance zones) is almost as good as the first classifier using all available data. For Iris data set 1-NN classifier determines labels correctly for 96.3% of test points with a complete training set and 96.1% of exemplars with a smart subset. Comparing harmonic means the difference is even more inessential as both models correctly classify 89% of test data. As the latter results are achieved with 18% of data points and the classification accuracy based on a randomly selected subset is far below (94.6% for Iris) the proposed PS method might work effectively in certain cases. Similar results are obtained for Wine and Glass data sets where classification accuracy stays on the same level while the percentage of retention is decreased to 36% and 77% correspondingly.

Table 2. Error rate (ER) using Parents of Ignorance Zones and 1-NN technique

	Full Set		**Smart Set**		**Random Set**	
Data set	**Arith. Mean**	**Contra-Harm. Mean**	**Arith. Mean**	**Contra-Harm. Mean**	**Arith. Mean**	**Contra-Harm. Mean**
Iris	3.7	11.0	3.9	11.0	5.4	12.6
Wine	12.9	17.6	13.4	17.9	14.0	18.6
Pima	35.6	36.4	35.8	36.5	34.3	35.1
Breast Cancer	5.3	6.4	5.5	6.5	4.0	5.5
Ionosphere	29.0	31.3	29.6	31.9	29.1	31.4
Glass	39.3	42.4	39.0	42.1	40.9	44.1
Bupa	46.6	47.7	46.5	47.6	46.1	47.4
Transfusion	31.2	32.1	33.1	34.0	30.3	31.4
Average	25.45	28.11	25.85	28.44	25.51	28.26

Along with cases confirming our assumptions, there are many data sets where results are completely opposite. Majority of data sets from Table 2, such as Pima, Breast Cancer, Ionosphere, Bupa, and Transfusion, have an outcome different to previously made assumptions. The lowest error rate is achieved with the classifier trained on a randomly selected subset. Both full and smart sets either show relatively similar or even worse classification accuracy. For example, for Breast Cancer, the model based on random subset classifies only 4% of test points incorrectly, while with the full set of data this number increases to 5.3%. Having more prototypes in the training set not only decreases classification speed as expected but also surprisingly increases error rate for these cases.

Trying to explain why the majority of data sets show the best classification accuracy with randomly selected subsets we found that the primary reason must be noisy data. For example, Figure 4 visualizes one iteration of Breast Cancer data set and shows misclassified points for three classifiers. On the picture, both smart and full classifiers have the same amount of instances attributed to the wrong class due to having too many random prototypes in the training set. Random classifier, on the contrary, has a very small amount of noisy points

Table 3. Percentage of retention (PR) using Parents of Ignorance Zones

	Full Set		**Smart and Random Sets**	
Data set	**Arith. Mean**	**Contra-Harm. Mean**	**Arith. Mean**	**Contra-Harm. Mean**
Iris	100	100	18.1	18.3
Wine	100	100	35.7	35.8
Pima	100	100	67.7	67.7
Breast Cancer	100	100	10.0	10.1
Ionosphere	100	100	57.9	58.0
Glass	100	100	76.7	76.8
Bupa	100	100	88.2	88.3
Transfusion	100	100	65.9	66.0
Average	100	100	52.53	52.63

and it helps to classify test data correctly. Figure 4 shows one iteration for a single data set but the same problem happens for other databases too. This case shows that neither 1-NN classifier nor smart subset can reliably predict future observations when dealing with noisy data. The algorithm makes incorrect decisions based on erroneous points located close to decision boundaries.

PS method selecting parents of ignorance zones to the subset discovers all possible grey zones including very small ones formed by noisy data points. Although it is possible to modify the algorithm to ignore smaller grey zones at first it is good to confirm that the results will change with smoother decision boundaries and less noisy data. The easiest way to make k-NN classifier more resistant to noise is to increase its number of NNs (Liu and Zhang 2012). The next section repeats the same experiment with 3-NN and 5-NN and compares the results.

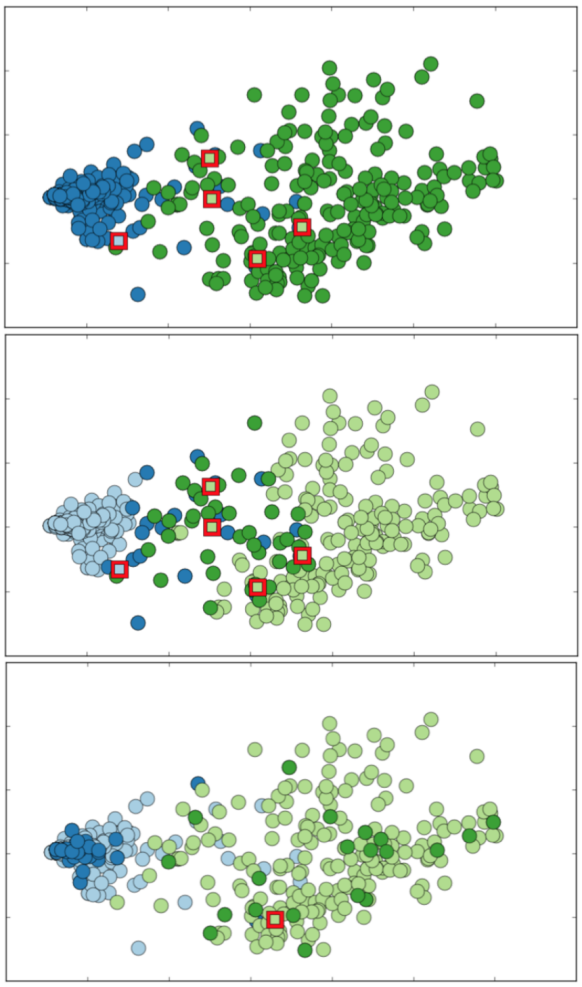

Figure 4. Training sets and misclassified test points (framed in red squares with correct class label inside) of full, smart and random classifiers for Breast Cancer data set.

4.3 Batch Prototype Selection with 3-NN and 5-NN

Performance of k-NN classifier heavily relies on several factors: the sample size, the selection of distance metric and chosen value for k parameter (Liu and Zhang 2012). The selection of correct k is crucial for successful work of k-NN algorithm. When k parameter is small the algorithm becomes sensitive to noisy data while larger value makes decision boundaries smoother. Usually, it is very difficult to predetermine the value of k and its optimal value varies depending on data set and feature distribution.

Based on results from Table 2 1-NN algorithm is not an optimal classifier for noisy data sets, such as Breast Cancer or Ionosphere, because its logic is very sensitive to random

points. Checking misclassified test points from Figure 4 (the ones framed in red square) it is visible that the algorithm makes mistakes when predicting the label based on the nearest noisy instances. Increasing the value of parameter k is one of the easiest ways to define decision boundaries smoother and check whether classification accuracy becomes better. For example, by increasing parameter k from 1 to 3 the classifier trained on the full data set would classify 4 out of 5 points from Figure 4 correctly.

Table 4. Error rate (ER) using Parents of Ignorance Zones and 3-NN technique

	Full Set		**Smart Set**		**Random Set**	
Data set	**Arith. Mean**	**Contra-Harm. Mean**	**Arith. Mean**	**Contra-Harm. Mean**	**Arith. Mean**	**Contra-Harm. Mean**
Iris	5.1	10.1	4.9	11.2	4.4	10.5
Wine	10.4	15.5	9.9	15.0	11.3	16.1
Pima	29.3	30.2	29.5	30.5	28.8	30.0
Breast Cancer	3.6	4.7	3.7	4.9	3.4	5.1
Ionosphere	20.6	22.7	21.3	23.4	21.4	23.5
Glass	38.1	41.7	38.2	41.8	37.8	41.1
Bupa	45.4	46.4	46.5	47.5	45.3	46.4
Transfusion	27.0	28.0	27.8	28.7	26.9	28.0
Average	22.44	24.91	22.73	25.38	22.41	25.09

Table 4 and Table 5 show error rates for the same database-training set pairs as in the previous section but using 3-NN and 5-NN techniques as classification methods. Results show that error rates become lower for the majority of data sets and percentage of correctly classified points is growing with an increased value of parameter k. For example, an average error rate for the 1-NN classifier using full training set is 25.5%. With the algorithm that checks three nearest points, this number decreases to 22.4% and with five neighbors - even more down to 21.3%. Amount of data used for training stays the same for all classifiers.

With the larger value of parameter k the NN classifier becomes more resistant to noisy data and it helps to improve results for certain data sets. The latest version shows lower error rates

Table 5. Error rate (ER) using Parents of Ignorance Zones and 5-NN technique

	Full Set		**Smart Set**		**Random Set**	
Data set	**Arith. Mean**	**Contra-Harm. Mean**	**Arith. Mean**	**Contra-Harm. Mean**	**Arith. Mean**	**Contra-Harm. Mean**
Iris	3.3	8.6	4.8	25.0	5.5	14.1
Wine	9.1	14.2	10.6	16.1	10.7	15.5
Pima	27.6	28.7	28.5	29.4	27.7	28.8
Breast Cancer	3.0	4.2	3.1	4.5	3.5	5.0
Ionosphere	19.2	21.2	19.2	21.4	19.8	22.0
Glass	36.8	40.1	38.2	41.0	39.0	42.5
Bupa	45.6	46.7	45.2	46.3	45.6	46.8
Transfusion	25.6	26.5	26.7	27.3	25.3	26.3
Average	21.28	23.78	22.04	26.38	22.14	25.13

for Breast Cancer and Ionosphere when more neighbors are taken into consideration. 5-NN classifier using the complete training set for decision-making improves error rate by 2.3% (from 5.3% with 1 neighbor to 3.0% with 5 neighbors) for Breast Cancer and by 9.8% (from 29.0% with 1 neighbor to 19.2% with 5 neighbors) for Ionosphere. The model with smart PS method also shows similar improvements: from 5.5% to 3.1% for Breast Cancer and from 29.6% to 19.2% for Ionosphere. In addition relation between different models becomes more logical. The algorithm trained on the full set of data shows the best classification accuracy and outperforms the classifier trained on parents of ignorance zones. The effect brought by the smart PS method is, in turn, better than the random subset.

Proposed idea of making decision boundaries smoother improves results, but unfortunately not for all data sets. The more detailed analysis shows that the larger value of parameter k is not an ideal solution and it does not work in all cases. It makes results better for certain data sets, such as Breast Cancer and Ionosphere, whereas other cases, such as Bupa and Transfusion, still experience problems. These databases obtain analogous results with three versions of the training set even with 3- and 5-NN classifiers. The fact that the random subset

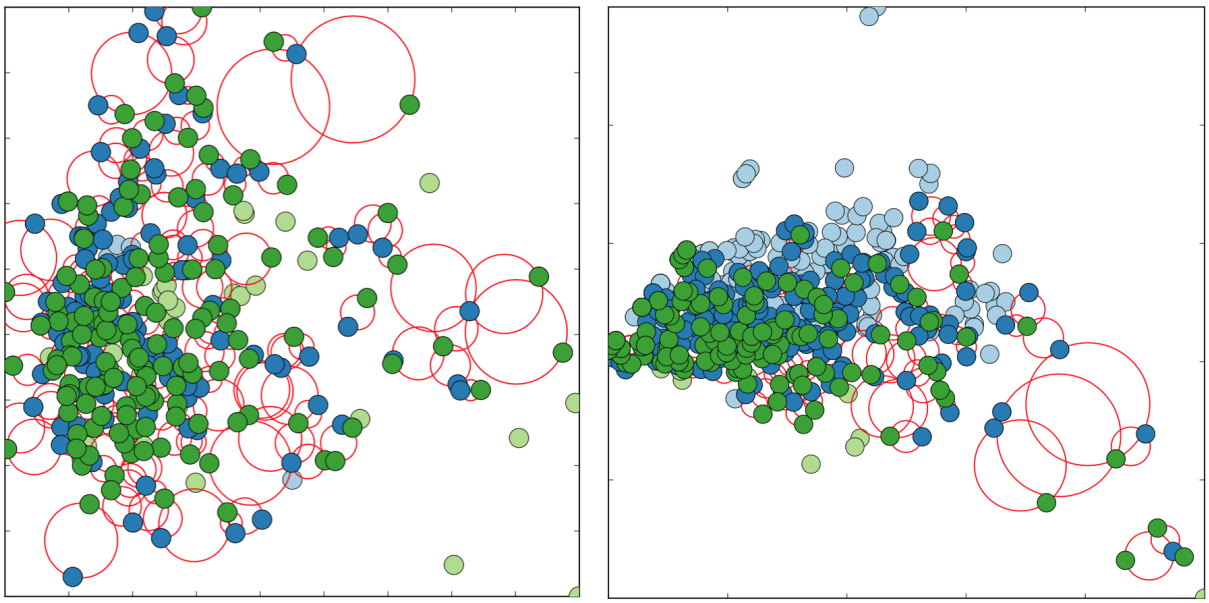

Figure 5. Parents of ignorance zones in Bupa and Transfusion data sets.

sometimes achieves lower error rate than the full training set is even more unintelligible.

Visualization of Bupa and Transfusion prototypes, shown in Figure 5, uncovers details of data sets structures. The primary reason for low classification accuracy is the absence of accurate borders between classes. Increased parameter k strives to neglect influence of random points but it becomes ineffectual when the majority of data prototypes are mixed. Batch PS method presented in this section also has a small impact on these databases as with a large number of ignorance zones very few prototypes are discarded and percentage of prototype retention stays high (88.2% for Bupa and 65.9% for Transfusion). Such chaotic databases lack structure and adding the majority of prototypes to the training set without careful analysis does not improve the situation.

While it must be impossible to find a single algorithm classifying test points with the lowest error rate for all data sets, previous experiments unveil weaknesses of Prototype Selection using parents of ignorance zones. This implementation works well with simple data sets where borders between classes are accurate, and decision boundaries can be distinct. For noisy databases, the algorithm results in overfitting because every random point generates an additional ignorance zone whose parents are added to the final subset.

4.4 Computational complexity

Besides poor classification accuracy, shown in Table 2, batch calculation of ignorance zones might be computationally expensive and make the algorithm hard to use in practice for massive data sets. To check this, we analyze time complexity of the proposed batch algorithm. Additionally, assessment of work with n-dimensional data is done.

The batch version of PS method mainly repeats the steps shown in Algorithm 1 to discover ignorance zones and subsequently select their parents. The runtime algorithm executes the following iterations:

- for each pair (triple) of points create a circle, representing a potential ignorance zone;
- for each circle calculate distance from its center to every point from the data set;
- if the minimum found distance is less than the radius, skip the circle; otherwise, it is an ignorance zone;
- return parents of discovered ignorance zones.

For d-dimensional data set with n points, the algorithm using p points to define a circle would require $O(n^p)$ runtime for the first step. The second step depends on size and dimensionality of the data set, requiring $O(nd)$ runtime to calculate distances between the center and each point from the database. In total, the algorithm uses $O(n^p nd) = O(n^{p+1}d)$ time to discover ignorance zones and select its parents as prototypes. In this section, the experiments, looking over pairs of points and forming 2-parent ignorance zones, used $O(n^3)$ time (assuming d being considerably smaller than n). The experiments, iterating over triples of prototypes, required $O(n^4)$ runtime.

We can see that the algorithm complexity grows exponentially with the number of points forming the circle. Although in d-dimensional space up to $d+1$ points can be used to define a hypersphere, it does not mean that the algorithm should discover all of them. The exponential time is a poor characteristic of the algorithm, but in general case, it does not require to find all types of ignorance zones and can select prototypes starting with 2-parent zones. In this thesis, we do not validate the efficiency of PS for multi-dimensional data set with different ignorance zones, but it can be a topic for the next research.

5 Incremental Prototype Selection with ignorance zones

Batch algorithm of Prototype Selection shown in the previous chapter works well for data sets without much noise and relatively small amount of points. For bigger data sets it becomes computationally expensive because every possible pair of points from the training set should be examined whether it produces an ignorance zone or not. Due to having these constraints, the algorithm has a cubic $O(n^3)$ complexity class when working with 2-parent ignorance zones and quadratic $O(n^4)$ when discovering 3-parent ignorance zones. Although other popular PS methods as CNN, RNN and DROP also have cubic processing time this property is considered to be one of the main disadvantages as cubic (and expecially quadratic) running time is not suitable for massive data sets (Arnaiz-González et al. 2016).

One more serious issue of the algorithm is high noise sensitivity because the batch version chooses parents of absolutely all ignorance zones to the representative subset. Noisy instances generate a lot of small ignorance zones that are not relevant for classification. For data sets without distinct class borders (e.g., Figure 5) selecting all grey zones interferes with the goal to identify data structure. Small grey zones discovered in noisy areas force the algorithm to add harmful prototypes to the final subset. Excessive data points do not help to make right classification decisions but rather result in having an overfitted model (Hawkins 2004).

Resembling the batch algorithm that includes parents of ignorance zones to the representative subset an incremental method also tries to address previously marked issues. The new version does not examine all available prototypes and does not generate all ignorance zones straight away. As the name implies the algorithm starts with an empty subset and adds one prototype on every next iteration till all discovered grey zones become empty. This approach should help to concentrate on the fundamental structure of the database and ignore noisy areas. Incremental nature of the algorithm withdraws the necessity to examine all possible pairs of points and makes running time significantly shorter. Size of the final subset is determined automatically by the algorithm when it converges and not manually based on extra heuristics and knowledge about the data set.

On the first iteration, the algorithm starts with an empty subset and requires at least one ignorance zone to continue. Due to the lack of any preliminary knowledge about the data set the whole space represents a grey zone at this point. In order to limit the endless initial ignorance zone and define its boundaries, we use the concept of domain. Domain contains complete data set inside it and represents a geometric figure circumscribing all available data instances. In the beginning, the algorithm estimates boundaries of known data and makes its first decision based on domain size and form.

Besides playing a role of initial grey zone domain is also important for discovering a new type of ignorance areas - 1-parent zones. These zones possess the same properties as 2- and 3-parent zones with the only difference in circle definition: the circle must touch selected point and domain. For the incremental algorithm, 1-parent zones make it possible to find classes located next to domain frontier. Previously discussed 2- and 3-parent zones are not capable of discovering unknown prototypes located next to domain because they require at least two points from different classes while 1-parent zones need only one.

In this thesis, rectangular and circular domains are validated and compared with each other. For any form of domain the incremental approach of Prototype Selection algorithm stays identical and works as following:

- on the first iteration a domain boundary for the given training set is calculated (the smallest rectangular or circle circumscribing all points) and added as the first ignorance zone;
- on every next iteration a new point located inside the biggest grey zone is added; if there are several points inside the grey zone then the point being closest to the domain boundary is chosen for 1-parent zone and the point closest to the center of the grey zone for 2-parents zone;
- in case the biggest grey zone does not have new points inside, then the second largest grey zone is examined, and so on. The algorithm stops when there are no grey zones with vacant points inside left.

Algorithm 2: Selecting prototypes by incremental algorithm

FindIncrementalPrototypes (X, Y)

inputs : A training set X with coordinate vectors and Y with class labels

output: The subset of prototypes X_{subset} with coordinate vectors and Y_{subset} with class labels

$X_{subset} \leftarrow \emptyset, Y_{subset} \leftarrow \emptyset$;

$domain = CalculateDomain(X, Y)$;

$Z = \{domain\}$;

do

  $Z = SortZonesByRadius(Z)$;

  $newPointAdded$ = **false**;

  foreach $Z_i \in Z$ **do**

    i =index of a point closest to ignorance focus of zone Z_i;

    if $i \neq -1$ *and* $newPointAdded$ = ***false*** **then**

      Add $X[i]$ to X_{subset};

      Add $Y[i]$ to Y_{subset};

      $newPointAdded$ = **true**;

  $Z = FindIgnoranceZones(X_{subset}, Y_{subset}) \cup$
  $FindDomainIgnoranceZones(X_{subset}, domain)$;

while $newPointAdded$ = ***true***;

return X_{subset}, Y_{subset};

5.1 Rectangular domain border

This section presents an algorithm using a rectangle as a geometric figure for domain. The rectangle must satisfy two criteria: be as small as possible and circumscribe all data points. In order to satisfy these requirements the algorithm finds from the training set $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ minimum (x_{min}, y_{min}) and maximum (x_{max}, y_{max}) values for each axis. Coordinates of the rectangle are created as combinations of minimum and maximum values $\{(x_{min}, y_{min}), (x_{min}, y_{max}), (x_{max}, y_{max}), (x_{max}, y_{min})\}$.

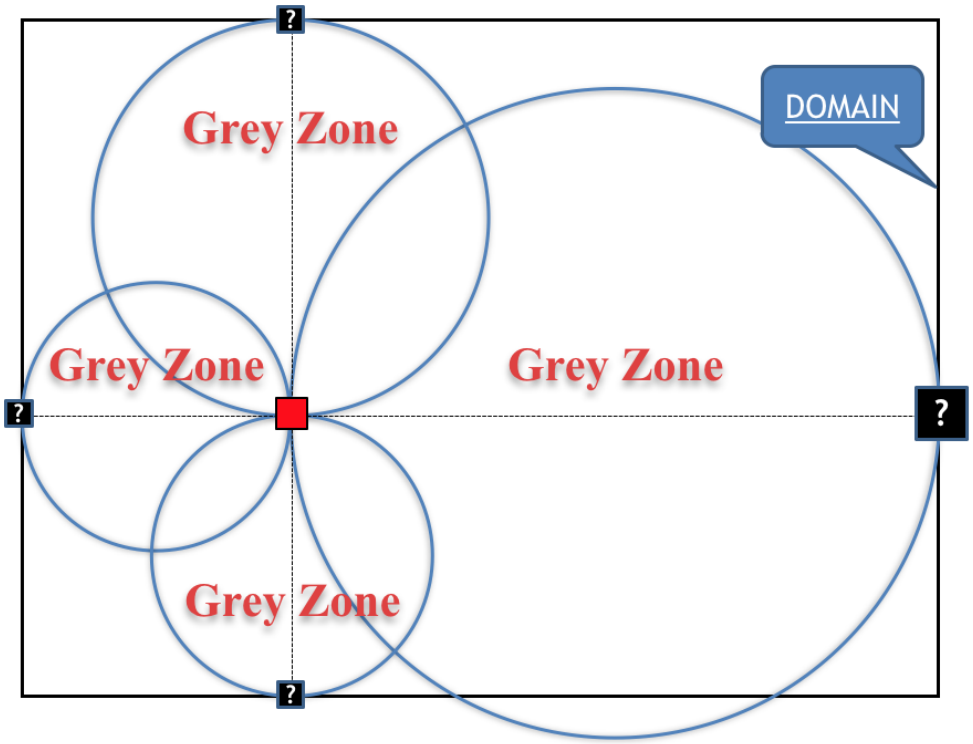

Figure 6. 1-parent ignorance zones produced by one point in rectangular domain. Initial point selected from the training set is a red square, domain is a black rectangle subscribing the full training set.

Depending on domain's figure the criteria for choosing 1-parent ignorance zones can be different. This variety is determined by the existence of various ways to define a circle between a point and edge of domain border. In case of rectangular domain, each point may create up to four 1-parent ignorance zones with domain. 1-parent ignorance zones are formed as diametrical circles between the selected point and its projections on each of rectangle's

sides (as shown in Figure 6). In case each of the formed circles does not contain any other exemplars inside the point forms four 1-parent grey zones, otherwise less. 2-parent grey zones are discovered the same as in the previous algorithm: as a diametrical circle between two points from different classes that does not have any other points inside.

The effectiveness of incremental PS with the rectangular domain is tested based on 1-NN classifier. A smart subset is formed as parents of 1- and 2-parent ignorance zones that are discovered by the incremental algorithm. Methodology for this and subsequent experiments remains unchanged: error rate and percentage of retention are measured as average over 100 iterations for each of eight data sets. Results of incremental PS, shown in Table 6, are compared with the full classifier and batch version of PS.

Table 6. Error rate (ER) and percentage of retention (PR) obtained from incremental PS with rectangular domain

	Full Set		**Parents of Ignorance Zones**		**Rectangular Domain**		**Expanded Rectangular Domain**	
Data set	**ER**	**PR**	**ER**	**PR**	**ER**	**PR**	**ER**	**PR**
Iris	3.7	100	3.9	18.1	7.1	19.3	4.4	19.5
Wine	12.9	100	13.4	35.7	13.9	31.8	14.1	22.9
Pima	35.6	100	35.8	67.7	28.8	17.3	44.8	9.2
Breast Cancer	5.3	100	5.4	9.3	3.9	7.2	4.7	4.6
Ionosphere	29.0	100	29.6	58.2	24.9	28.6	28.4	18.1
Glass	38.4	100	38.0	76.9	39.9	53.5	48.7	40.2
Bupa	46.8	100	46.8	86.7	49.6	26.4	48.2	25.8
Transfusion	31.3	100	33.8	64.5	26.8	6.9	27.6	6.2
Average	25.38	100	25.84	52.14	24.36	23.88	27.61	18.31

On average incremental PS with rectangular domain shows significantly improved results: in comparison with the full classifier error rate is decreased by 1% (from 25.4% to 24.4%) with only 23.9% of data being used for decision-making (the former classifier uses 100%). Many databases, such as Pima, Breast Cancer, and Ionosphere, not only decreased the volume of the

representative subset but also improved classification accuracy a bit. Incremental approach of focusing on bigger ignorance zones and building the data structure also has an impact on chaotic databases: error rate for Transfusion data set is lowered from 31.3% to 26.8% while decreasing size of training more than 10 times.

While results are good for data sets that are considered noisy and problematic for the batch algorithm, for simpler data sets, such as Iris, Wine, and Glass, the performance is lower. One of the found weaknesses of this algorithm is its trouble in finding classes located next to the corner of domain. Figure 7 illustrates how the algorithm fails to include any blue point from Iris database to the final subset. As a result absence of the whole cluster of points intensely increases classification error rate.

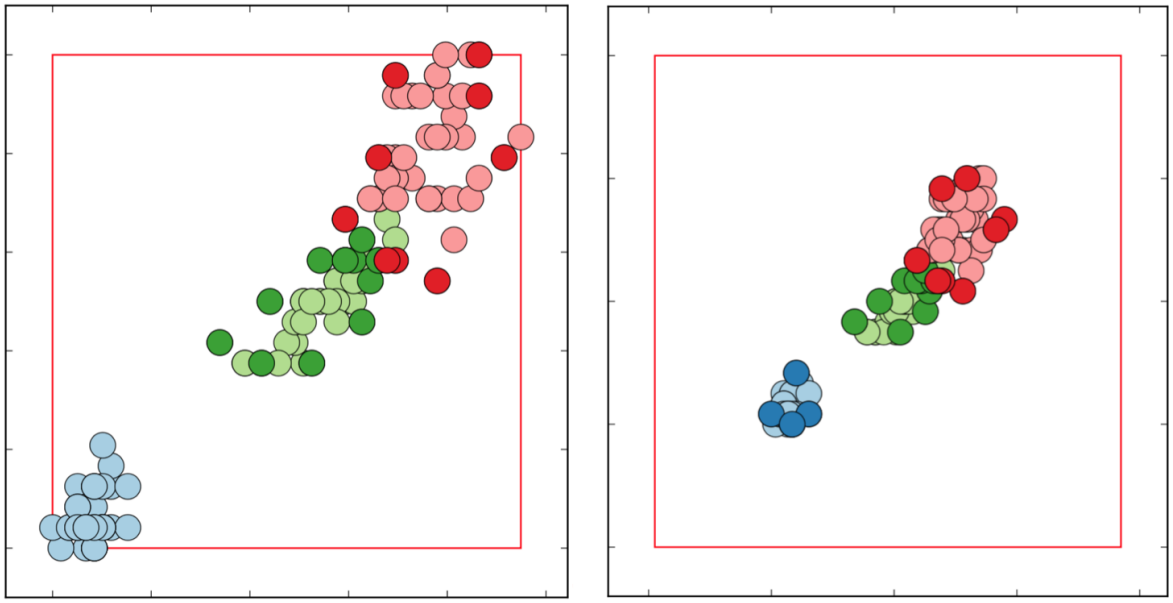

Figure 7. Selected prototypes for Iris data set based on regular and expanded rectangular domains.

The algorithm may fail to find points located close to the rectangle's corners because 1-point grey zones do not always cover that area. In case discovered prototypes are located in the center of domain (as in Figure 7) their 1-parent grey zones do not cover corners and some parts of domain are never examined. A potential solution for this issue is to move data points rather far off the domain corners. Not to break relations between data clusters instead of moving exemplars themselves we extend domain borders equally in all directions and make

its corners free of data.

Expanded rectangular domain solves the problem with undiscovered zones and missed prototypes from the corners. Increasing the domain square twice helps the algorithm to find and include blue points for Iris data set (the right part of Figure 7). Although Iris leverages this change to improve its classification accuracy (from 92.9% to 95.6%) in other cases the results get worse and error rate increases. Incremental approach based on ignorance zones and rectangular domain shows good results for noisy databases while fails to be a universal solution due to specifics of the rectangular domain, and thus the next chapter validates the circular form of domain.

5.2 Circular domain border

Although the rectangular domain works relatively well and finding the rectangle circumscribing the whole data set is not a computationally intensive task it lacks one important property: distance from the center of domain to the border varies depending on where the border point is chosen. The closer to the corners the point is located the further from the domain center its distance is. As a consequence points located next to the corners are not always covered by 1-parent grey zones and not selected into the final training set (the left part of Figure 7).

Replacing the rectangular domain with a circular border solves this problem as the distance from the center to the border remains the same for all points of the circumference. The concept of ignorance zones discussed before is primarily based on circles, so an alternative variant of choosing a circle as a geometric representation for the domain border finds a great match with our previous notion.

Finding the smallest circle around a given set of points, required for creating the circular domain of the data set, is known as the smallest enclosing circle problem (Shamos and Hoey 1975). Although the problem has been already discussed for more than 160 years, newer and faster algorithms are still proposed and created. In recent years randomized algorithms have been developed for many geometry problems and the smallest enclosing circle is not an exception (Welzl 1991). In this thesis, we use randomized Welzl's algorithm that overall solves the problem of the smallest circles in linear time $O(n)$ (Welzl 1991).

There is no one correct procedure to choose a 1-parent grey zone inside the circle. It can be done in different ways depending on how the point of intersection between the ignorance zone and the domain is chosen. The following sections describe ideas and results of classification with three different principles: inscribed circles, Gabriel circles, and Relative circles.

5.2.1 Inscribed circles as 1-parent ignorance zones

At first, we validate an idea of generating ignorance zones as inscribed circles. In this case, a grey zone represents a circle going through a point from the training set and touching the domain border at the same time. The circle must be located at a tangent to the domain and not intersect it. In case no known points are located inside such circle it forms a 1-parent ignorance zone.

Similar to the algorithm with rectangular domain each known point can generate up to four 1-parent grey zones: two potential grey zones are produced by a line going through the domain center and two more are created as the biggest inscribed circles in each of two formed semicircles (Figure 8).

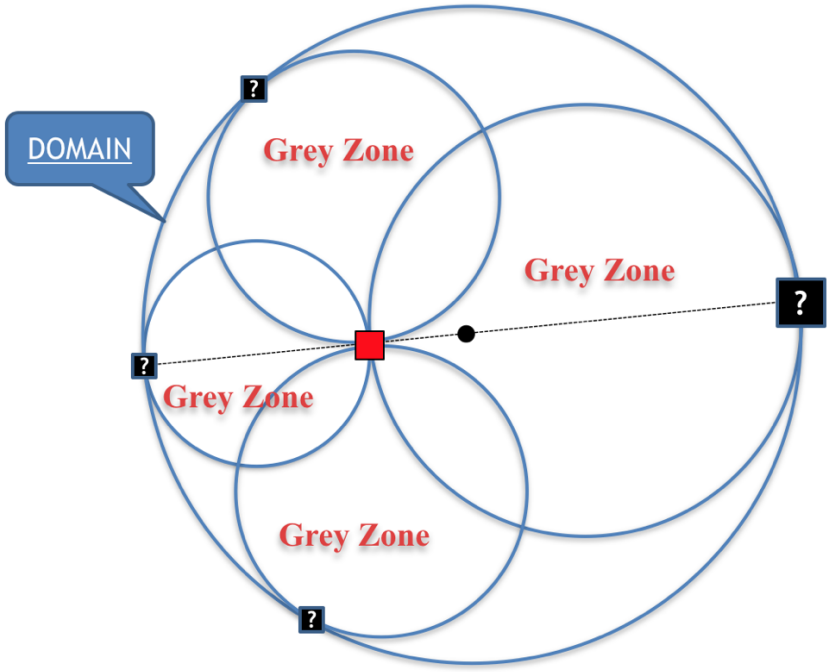

Figure 8. 1-parent ignorance zones formed as inscribed circles in circular domain.

Table 7, containing the results of classification experiments with inscribed circles, shows that this approach turns out to be working well. For all data sets the results are comparable with the batch version of the algorithm. The average error rate of the algorithm using inscribed circles (23.8%) not only lower than the batch version (25.8%), but it is also better than results based on training sets without PS (25.4%). In six out of eight data sets the error rate is either lower or stays on the same level. For the rest two data sets (Wine and Glass) the classification accuracy decreased a bit but it is compensated by a significant drop in the percentage of data retention (for Wine it went down from 35,7% to 29,5%, for Glass - from 76,9% to 47,6%).

Data sets containing noisy instances, such as Breast Cancer and Transfusion, obtain better classification accuracy with inscribed circles than with the classifier trained on non-filtered training sets. Incremental nature of the algorithm helps to ignore small ignorance zones produced by noisy data. Instead, the algorithm focuses on relevant data points from the biggest ignorance areas. This approach simultaneously results in smaller and more representative subsets of data.

Table 7. Error rate (ER) and percentage of retention (PR) with circular domain and inscribed circles based on 1-NN classifier

	Full Set		**Parents of Ignorance Zones**		**Domain with Inscribed Circles**	
Data set	**ER**	**PR**	**ER**	**PR**	**ER**	**PR**
Iris	3.7	100	3.9	18.1	3.9	21.8
Wine	12.9	100	13.4	35.7	13.6	29.5
Pima	35.6	100	35.8	67.7	31.4	12.6
Breast Cancer	5.3	100	5.4	9.3	4.0	7.2
Ionosphere	29.0	100	29.6	58.2	24.6	26.2
Glass	38.4	100	38.0	76.9	38.8	47.6
Bupa	46.8	100	46.8	86.7	46.2	29.4
Transfusion	31.3	100	33.8	64.5	27.5	9.4
Average	25.38	100	25.84	52.14	23.75	22.96

Visualization is usually the most convenient way to understand the logic of the algorithm. Figure 9 shows first, second and last iterations of the PS algorithm with inscribed zones. On each iteration, one new point located inside the biggest ignorance zone is added to the final set and the algorithm repeats this process until only empty grey zones are left. The right picture from Figure 9 shows the final subset of chosen prototypes and grey zones that were left empty after the final iteration. Selected subset mainly contains prototypes located on the edge of two classes or on the edge of domain and the class. Inner points and various noisy areas formed by different classes being close to each other without specific order are left out of the chosen subset.

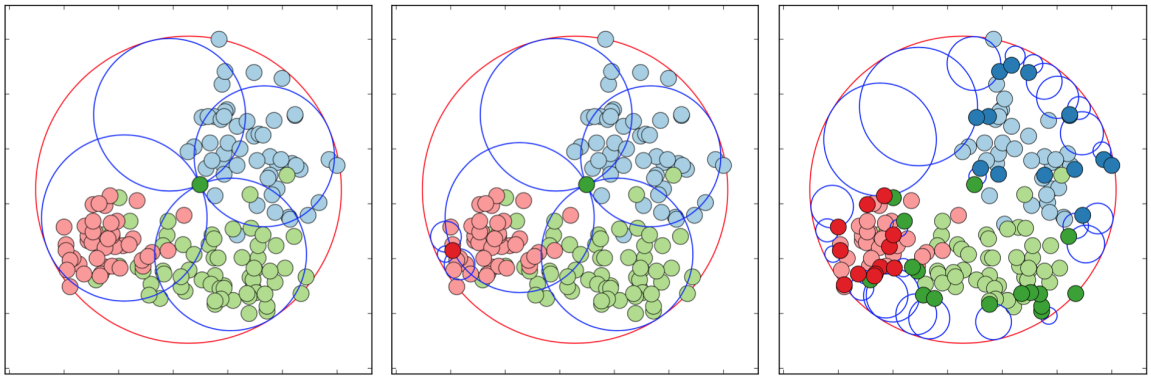

Figure 9. Incremental algorithm with inscribed circles after the first, the second and the last iterations. Domain is shown as red circle, grey zones are blue circles. Wine data set.

5.2.2 Circles with Gabriel principle as 1-parent grey zones

The second approach of forming 1-parent grey zones within a circular domain. The idea is analogous to inscribed circles with one difference: circles are not obliged to be fully located inside the domain and circles can be partially situated outside the domain area.

Each data point generates up to four grey zones: two potential grey zones are produced by a line going through the domain center like with inscribed circles and two more are created by a perpendicular to the first line going through the point (Figure 10). Two points that form an ignorance zone are Gabriel neighbors as their diametrical circle does not contain any other points from the training set (Jaromczyk and Toussaint 1992). This resemblance gives a Gabriel principle name for this modification of the algorithm.

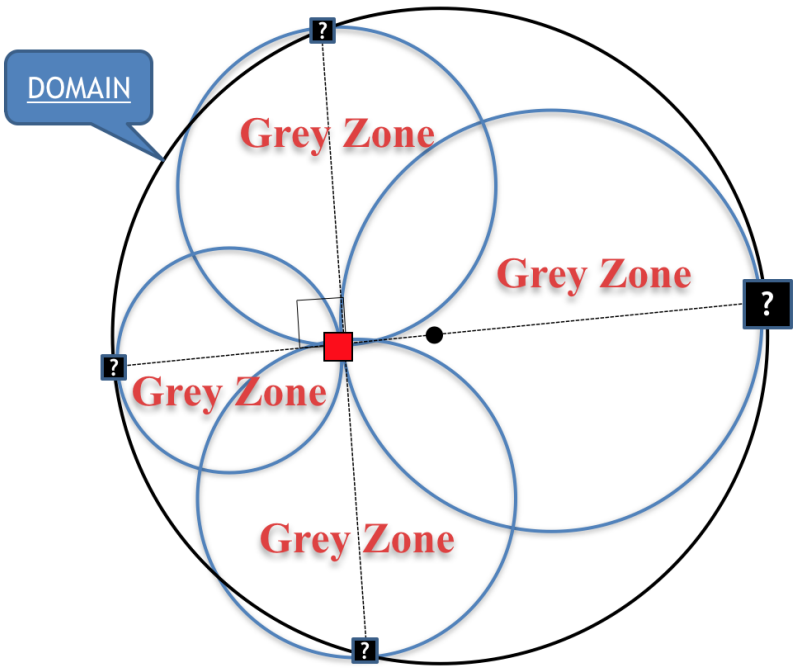

Figure 10. 1-parent ignorance zones produced with Gabriel principle.

Circles created with Gabriel Principle have a bigger area than inscribed circles because they do not have a strict limitation of being fully located inside the domain. Bigger circles represent more valuable areas of curiosity, but at the same time there is a higher probability for data points to be inside the bigger area, so with Gabriel principle, there are fewer ignorance zones produced but they are more important.

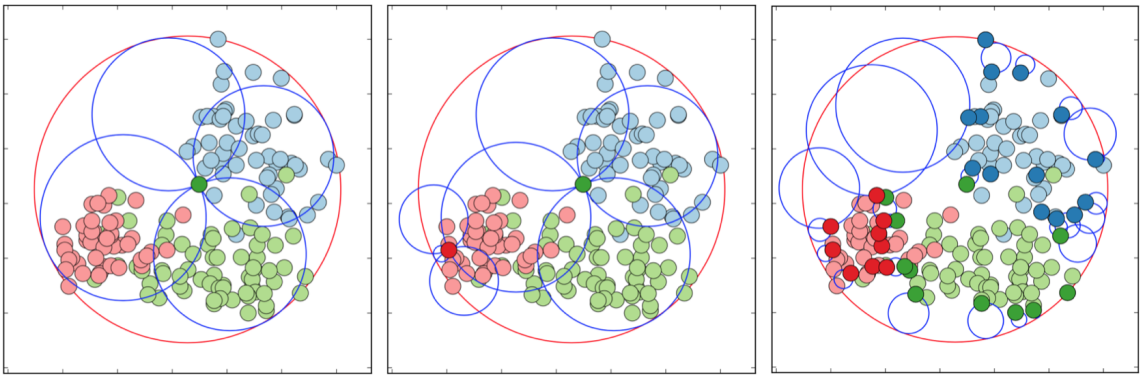

Figure 11. Incremental algorithm with circle domain and Gabriel principle after the first, the second and the last iterations. Domain is shown as red circle, grey zones are blue circles. Wine data set.

As shown in Figure 11 prototypes are selected to the subset following the same procedure as with inscribed circles. The only difference is that some 1-parent ignorance zones are larger and partly located outside the domain boundary. This slight difference also explains the distinction between final subsets: the number of prototypes and grey zones in the final subset is smaller with Gabriel principle and larger with inscribed circles.

Table 8 show positive results obtained with incremental PS with Gabriel principle. On average both percentages of data retention and error rate are lower in comparison with inscribed circles. Six out of eight data sets have better classification accuracy with smart PS method based on Gabriel principle rather than with the classifier trained on raw training sets. For PS method it is important to maintain selected subset small and representative at the same time, and proposed algorithm from this section fulfills this requirement.

Table 8. Error rate (ER) and percentage of retention (PR) with circular domain and Gabriel principle

	Full Set		**Parents of Ignorance Zones**		**Domain with Gabriel principle**	
Data set	**ER**	**PR**	**ER**	**PR**	**ER**	**PR**
Iris	3.7	100	3.9	18.1	3.7	18.0
Wine	12.9	100	13.4	35.7	13.3	27.2
Pima	35.6	100	35.8	67.7	28.4	11.4
Breast Cancer	5.3	100	5.4	9.3	4.1	5.6
Ionosphere	29.0	100	29.6	58.2	25.2	24.7
Glass	38.4	100	38.0	76.9	40.1	46.4
Bupa	46.8	100	46.8	86.7	46.1	31.9
Transfusion	31.3	100	33.8	64.5	26.1	13.1
Average	25.38	100	25.84	52.14	23.38	22.29

5.2.3 Circles with Relative principle as 1-parent grey zones

The third approach of forming 1-parent grey zones within a circular domain. It is similar to circles with Gabriel principle with one major difference: circles are twice as big and 1-parent ignorance zones represent crescent, visually shown as a disjoint intersection between circle and domain. The figure of crescent resembles geometrical figure produced in Relative neighbors algorithm where it is called "lune" (Jose Salvador Sánchez, Pla, and Ferri 1997). The similarity of geometrical figures gives this method a name of Relative principles.

As with previous variants each point can generate up to four 1-parent grey zones: two circles are produced by a line going through the domain center and two more are created by a perpendicular to the first line going through the chosen point (Figure 12). If no other points from the training set are situated inside the circle it forms a 1-parent ignorance zone.

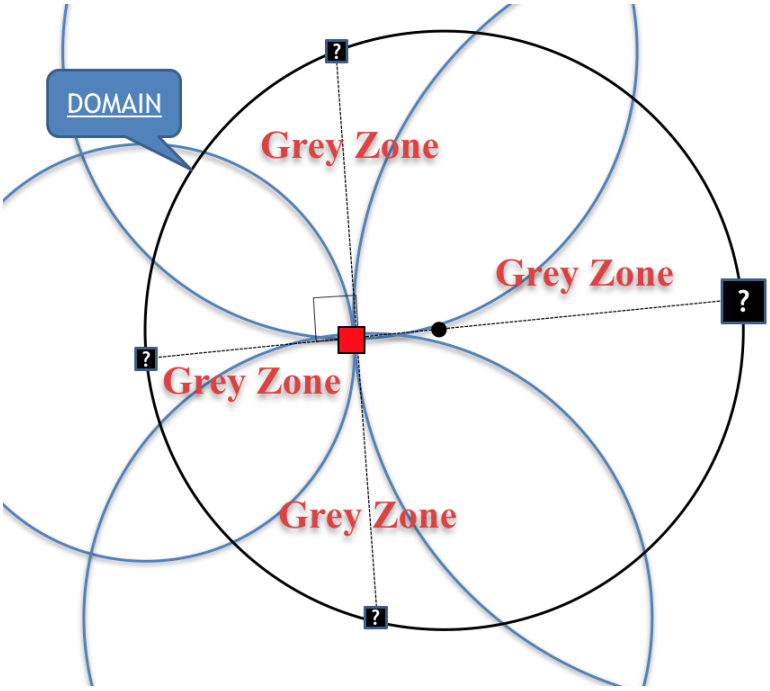

Figure 12. 1-parent grey zones produced with Relative principle inside circular domain after the first iteration of the algorithm.

The algorithm with Relative principle forms bigger grey zones in comparison with previous versions because circles have a double diameter. This property has an influence on the number of ignorance zones so that there are fewer of them comparing to Gabriel principle and

inscribed circles. Figure 13 visually demonstrates this assumption by possessing even fewer grey zones and prototypes than in the case of Gabriel principle. Reducing the number of grey zones and selecting fewer prototypes has a positive impact on the percentage of retention, though the same logic might not be applicable to the classification accuracy.

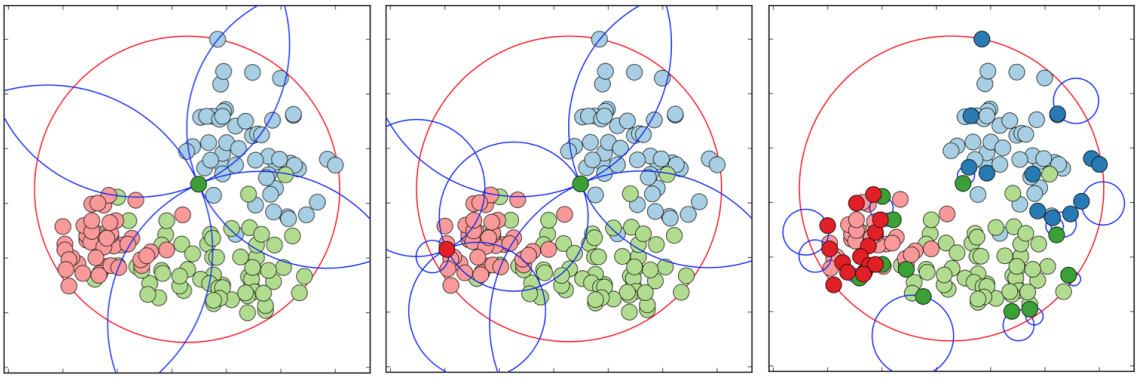

Figure 13. Incremental algorithm with circle domain and Relative principle after the first, the second and the last iterations. Domain is shown as red circle, grey zones are blue circles. Wine data set.

Table 9 shows classification results for the algorithm using Relative principle. In comparison with Gabriel principle percentage of retention is lower for all data sets, but as consequence error rate has also deteriorated. There is always a trade-off between maintaining representative subset and making it smaller. Relative principle over-restricts prototype's requirements and does not include enough points to the final subset. Having sparse subset of data results in worse classification accuracy almost for all data sets excluding Iris. Iris data set benefits from Relative principle that decreases its error rate from 3.7% (with Gabriel principle) to 2.6% together with improving retention percentage and leaving only 17.6% of instances. All other data sets received worse results than with Gabriel principle.

Table 9. Error rate (ER) and percentage of retention (PR) with circular domain and Relative principle based on 1-NN classifier

	Full Set		**Parents of Ignorance Zones**		**Domain with Relative principle**	
Data set	**ER**	**PR**	**ER**	**PR**	**ER**	**PR**
Iris	3.7	100	3.9	18.1	2.6	17.6
Wine	12.9	100	13.4	35.7	13.9	23.7
Pima	35.6	100	35.8	67.7	28.3	10.7
Breast Cancer	5.3	100	5.4	9.3	4.2	5.6
Ionosphere	29.0	100	29.6	58.2	29.9	18.7
Glass	38.4	100	38.0	76.9	42.7	45.8
Bupa	46.8	100	46.8	86.7	50.8	23.0
Transfusion	31.3	100	33.8	64.5	26.3	7.4
Average	25.38	100	25.84	52.14	24.84	19.06

5.3 Comparison of results

The main outcome of making multiple experiments with different Prototype Selection methods on eight data sets is that no algorithm consistently outperforms the others. An algorithm that works well for one data set does not necessarily work well for the other one and vice-versa. Structure of data sets can vary a lot from a problem to problem and it results in inconsistency when applying one PS algorithm to all data sets and expecting it to win in all situations. A similar conclusion is inferred by other researchers who emphasize the necessity of gaining the insights about the structure of data sets manually before applying specific prototype selection schemas (Brighton and Mellish 2002).

When talking about instance selection maintaining classification accuracy is commonly a primary goal, but depending on the approach of instance selection various algorithms show different results. Competence enhancement, competence preservation, and hybrid schemes are emphasized the most. Competence enhancement focuses on removing noisy and corrupted

instances, competence preservation aims to delete superfluous prototypes, hybrid approach tries to take best things from both approaches (Brighton and Mellish 2002).

Incremental versions of the algorithm described in this chapter implement the hybrid approach. On the one hand, they try to remove superfluous instances with the help of ignorance zones. Grey zones aim to find variance from the data set by focusing on points from different classes located next to each other. On the other hand, incremental algorithms ignore noisy data. They do not analyze all available data not discover all ignorance zones at once. By following incremental approach and adding only one prototype to the subset on each iteration the algorithm avoids analyzing small ignorance zones and adding noisy prototypes.

There is no single algorithm achieving the best results for every data set individually, but Table 10 and 11 show that some of the proposed algorithms manage to maintain the same or even lower error rate in comparison with classifiers trained on full training sets, parents of ignorance zones and randomly selected subsets. On average the worst classification accuracy is achieved with the random training set and the best one - with incremental PS algorithm with circular domain and Gabriel principle.

Table 10. Error rates (ER) for different PS algorithms based on 1-NN classifier

Data set	**Full Set**	**Parents of Ignorance Zones**	**Random Set**	**Inscribed cirles**	**Gabriel principle**	**Relative principle**
Iris	3.7	3.9	5.5	3.9	3.7	2.6
Wine	12.9	13.4	13.9	13.6	13.3	13.9
Pima	35.6	35.8	32.8	31.4	28.4	28.3
Breast Cancer	5.3	5.4	4.3	4.0	4.1	4.2
Ionosphere	29.0	29.6	28.6	24.6	25.2	29.9
Glass	38.4	38.0	47.3	38.8	40.1	42.7
Bupa	46.8	46.8	46.5	46.2	46.1	50.8
Transfusion	31.3	33.8	30.6	27.5	26.1	26.3
Average	25.38	25.84	26.19	23.75	23.38	24.84

Due to the hybrid schema of proposed algorithms not only do they remove noisy data and improve classification accuracy but also speed up decision-making by decreasing the training set and making storage requirements lower. On average incremental algorithms remove around 80% of redundant data leaving only 19-22% of prototypes for classification decisions. In comparison, the batch version of the algorithm deletes a bit less than 50% of data and retains the other half as the representative subset.

Table 11. Percentage of retention (PR) for different algorithms based on 1-NN classifier

Data set	**Full Set**	**Parents of Ignorance Zones**	**Random Set**	**Inscribed cirles**	**Gabriel principle**	**Relative principle**
Iris	100	18.1	18.1	21.8	18.0	17.6
Wine	100	35.7	35.7	29.5	27.2	23.7
Pima	100	67.7	67.7	12.6	11.4	10.7
Breast Cancer	100	9.3	9.3	7.2	5.6	5.6
Ionosphere	100	58.2	58.2	26.2	24.7	18.7
Glass	100	76.9	76.9	47.6	46.4	45.8
Bupa	100	86.7	86.7	29.4	31.9	23.0
Transfusion	100	64.5	64.5	9.4	13.1	7.4
Average	100	52.14	52.14	22.96	22.29	19.06

Percentage of retention, on the contrary to error rate, turns out to be quite a consistent metric. For incremental algorithms, the number of instances added to the subset depends on amount and size of grey zones. Relative principle has the lowest percentage of retention for all data sets (19%). Large "lunes" formed by Relative principle do not form many ignorance zones because the larger the circle becomes the less the chance not to have other points inside it. With fewer grey zones fewer prototypes are selected to the final training set. Inscribed circles have a requirement to be fully located inside the domain area and produce smaller grey zone. A larger amount of grey zones having smaller size explains the higher percentage of data retention with inscribed circles (23%). Gabriel principle defines middle-sized circles that

correctly corresponds to its average data retention (22.3%).

Achieving the highest classification accuracy with the lowest retention rate with the same PS algorithm is a non-common scenario. It happened for Iris and Pima data sets only where the best results are obtained with the algorithm based on Relative principle. For other databases defining which one of these two metrics is prevailing and looking for a trade-off between sufficiently low error rate and relatively high percentage of reduction is necessary to find the best PS method.

5.4 Computational complexity

To assess computational complexity of the incremental method, we should understand what operations are run while the algorithm is working. The runtime lifecycle consists of the following iterations:

- calculate domain area and add it to the list of ignorance zones;
- repeat while at least one non-empty ignorance zone exists:
  - find the most significant non-empty ignorance zone and select a prototype located next to its confusion focus;
  - discover 1-parent and multi-parent ignorance zones for selected prototype;
- return selected set of prototypes.

For d-dimensional data set with n points, the algorithm using p points to define a multi-parent ignorance zone would analyze and add all prototypes to the final set in the worst-case scenario. The first step of calculating domain requires $O(nd)$ runtime as the algorithm searches for min and max values for each dimension and builds the domain border based on found values.

The second step has $O(n)$ complexity because it is repeated at every point in the worst case. A substep for discovering 1-parent ignorance zones for a single point requires $O(2pnd)$ runtime because the amount of potential zones doubles with every new dimension (up to 4 1-parent ignorance zones in two-dimensional space, up to 6 in three-dimensional, etc.). For discovering multi-parent ignorance zones, the algorithm spends $O(n^p d)$ time for each proto-

type as it checks all combinations of p-point circles and calculates distances from its center to each point. All in all, the algorithm complexity is $O(nd) + O(n) * (O(2pnd) + O(n^p d))$ and assuming n being considerably bigger than d and p the expression can be simplified to $O(n^{p+1}d)$.

We can see that the worst-case running time is the same both for the batch and the incremental algorithms. In practice, not all points are chosen to the final subset by the incremental version, and the algorithm does not discover ignorance zones for all prototypes. Also, it is possible to stop the incremental algorithm at any point in time when desired classification accuracy is achieved, and the number of points for multi-parent ignorance zones can be limited. In this thesis, we applied the limitations of discovering confusion areas in two-dimensional space and calculating 2- and 3-parent ignorance zones only. With these limitations, we could validate the performance of the algorithm without being concerned about execution time. Though, taking into account the exponential complexity of the algorithm, detecting ignorance zones in multi-dimensional space must need further research beforehand and not included in this thesis.

6 Prototype Selection with an adversarial process

Generative Adversarial Network (GAN) has recently become a popular type of deep learning algorithms and achieved great success in producing realistically looking images, 3D object generation, and video prediction. Despite discriminative models where high-dimensional input data is usually mapped to a class label, adversarial networks offer a different approach where "generative model is pitted against discriminative model" (Goodfellow et al. 2014).

The adversarial method works as a system of two neural networks, where one of them, called the generator, produces candidates and the other one, called the discriminator, evaluates them. As a result of learning the generator learns to produce better images while the discriminator improves its detective skills in distinguishing fake and real images. Despite the success of GANs they have been known to be unstable to train and very difficult to understand from the training perspective (Arjovsky and Bottou 2017).

In this thesis we do not explore the details of GANs, but rather take an idea of two models being contested with each other and apply its analogy to the previously described concept of ignorance zones and Prototype Selection. Further experiments show the potential of the adversarial training applied to PS by generating the training set and making its qualitative and quantitative analysis.

6.1 Adversarial process in Professor-Student algorithm

The idea of learning from the interaction of two sides, that do not always pursue the same goal, happens often in our life. One prominent example is a schoolchild-teacher interaction at school or professor-student cooperation at the university. The former part of these pairs aims to impartially assess knowledge of the latter part, whose main goal is to answer questions correctly and pass an exam. This symbiosis results in mutual benefit: professor learns how to assess knowledge and find the gaps with the least number of questions, and student studies the least amount of information to be capable of answering professor's questions.

The interaction between student and professor resembles a classification problem where the

model based on training set aims to answer questions asked from testing set. In this chapter, we design and validate an algorithm called Professor-Student that tries to select the best subset of data based on adversarial process. The idea of Professor-Student algorithm broadly resembles the framework of Generative Adversarial Networks (Goodfellow et al. 2014), where professor represents the discriminative model and student can be thought the as the generative model.

Professor's role is to study the domain well and be competent enough to examine the most prepared student. Professor can choose necessary data from the data set in order to formulate hard questions even for the smartest student. During the education process, the professor has access to student's ignorance zones simulating the situation of a real exam where the professor can assess domain areas that student is not familiar of by asking questions.

The primary goal of the student is, on the contrary, to pass the exam by preparing how to do it with the most competent professor. While studying the student selects optimal data from the whole training set that is necessary to answer possible exam's questions. In order to make preparation process fair for both sides, the student has access to professor's ignorance zones and knows which domain spheres are more important than others.

Competition between professor and student drives them both to improve their sets of data so that the student answers professor's questions correctly and the professor has good domain knowledge. The data sets chosen by professor and student as a result of the adversarial process can be used for classification purpose as a training set.

6.2 Algorithm implementation

Being an incremental method Professor-Student algorithm starts with empty subsets and selects at most two points into them on each iteration (one for the professor and one for the student). The full process of the algorithm consists of the following steps:

- on the first iteration both professor and student begin with an empty set of data and having domain boundary as the only ignorance zone;
- on every next iteration, the student chooses a new point located inside his biggest grey

zone. Additionally, the new point must not be inside any of professor's grey zones;

- similar logic is done by the professor on each iteration: he also selects a new point but located inside the biggest ignorance zone belonging either to the professor or to the student;
- the algorithm stops when neither the professor nor the student has grey zones with vacant points inside.

Algorithm 3: Selecting prototypes by Profesor-Student algorithm

FindStudentProfessorPrototypes (X, Y)

inputs : A training set X with coordinate vectors and Y with class labels

output: The sets of prototypes X_{prof}, X_{stud} with coordinate vectors and Y_{prof}, Y_{stud} with class labels

$X_{prof} \leftarrow \emptyset, Y_{prof} \leftarrow \emptyset$;
$X_{stud} \leftarrow \emptyset, Y_{stud} \leftarrow \emptyset$;
$domain = CalculateDomain(X, Y)$;
$Z_{stud} = \{domain\}, Z_{prof} = \{domain\}$;

do

- i_{stud} =index of a point from the biggest zone of Z_{stud} located outside Z_{prof};
- **if** $i_{stud} \neq -1$ **then**
  - Add $X[i_{stud}]$ to X_{stud};
  - Add $Y[i_{stud}]$ to Y_{stud};
- i_{prof} = index of a point from the biggest zone of Z_{prof} or Z_{stud};
- **if** $i_{prof} \neq -1$ **then**
  - Add $X[i_{prof}]$ to X_{prof};
  - Add $Y[i_{prof}]$ to Y_{prof};
- $Z_{stud} = FindIgnoranceZones(X_{stud}, Y_{stud})$;
- $Z_{prof} = FindIgnoranceZones(X_{prof}, Y_{prof})$;

while $i_{stud} \neq -1$ *and* $i_{prof} \neq -1$;

return $X_{prof}, Y_{prof}, X_{stud}, Y_{stud}$;

Due to different behavior of data selection between the professor and the student, their subsets of data look diverse. The professor focuses on gaining maximum domain knowledge that results in having multifarious prototypes covering all data clusters from different angles. Having less limiting constraints and being able to select points from any ignorance zone, professor's subset has considerably more points than student's one.

Student's selection is not as comprehensive as professor's because it is created following a

different strategy: prioritize points from own gaps that help to answer professor's questions. The student chooses fewer points that do not necessarily cover all domain areas well. Many of them are located on the very border far from the center and hard to classify, so student's data can be used as a testing set for hard classification experiments.

6.3 Using Professor-Student prototypes in classification

In Professor-Student algorithm, both roles simulate the situation where one side asks questions and the other one tries to respond based on available knowledge. As a result of such interation the professor starts making more complicated inquiries and the student learns to give more detailed explanations. Adapting to increased level of questions examiner and examinee have to discover new relavant knowledge about domain. We believe that such discovery process eventually forms a carefully chosen representative knowledge set.

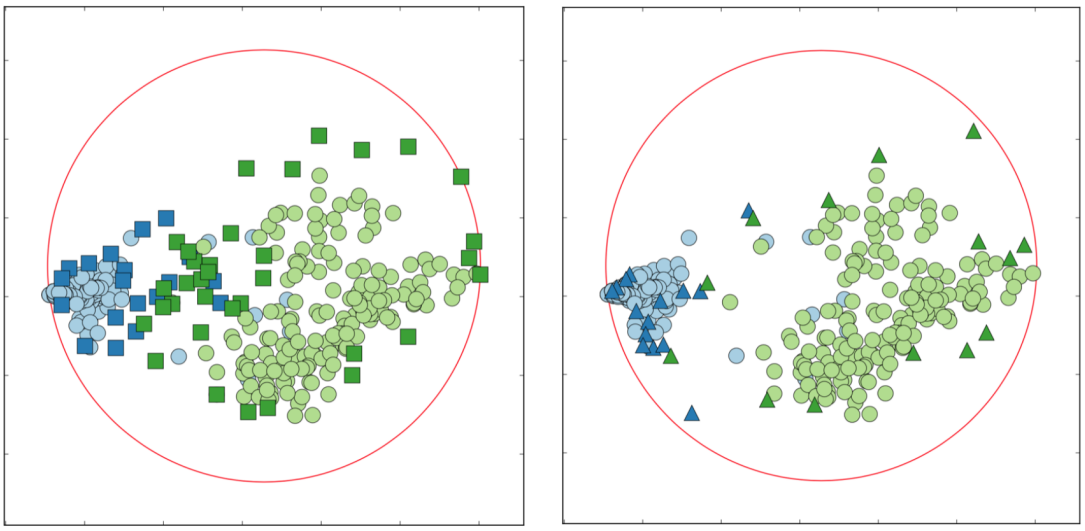

Figure 14. Prototypes selected by Professor-Student from Breast Cancer data set. Squares - professor's points, triangles - student's points.

Figures 14 and 15 show the output of Professor-Student algorithm for Breast Cancer and Wine data sets. Square points represent prototypes selected by the professor, triangles - points chosen by the student. Professor's sets contain exemplars mainly concentrated near the cluster's borders. It ignores superfluous points and focuses on prototypes located next to class contours. Based on algorithm's logic, the examiner must prioritize domain exploration

by looking for prototypes in the largest ignorance zones. Selected exemplars comply with professor's strategy: with knowledge about class contours the professor is capable of asking complex questions and assessing student's knowledge accurately.

Student's subsets contain fewer exemplars in comparison with professor's data. For the most part, prototypes selected by the student are located inside class contours defined by the professor. This disposition of points is a consequence of examinee's strategy to answer professor's questions correctly. There is no reason for arbitrary domain exploration because the professor does not ask questions from ignorance zones. Student's prototypes taken separately do not represent a stated value, but in conjunction with professor's subset, they supplement the output with valuable knowledge.

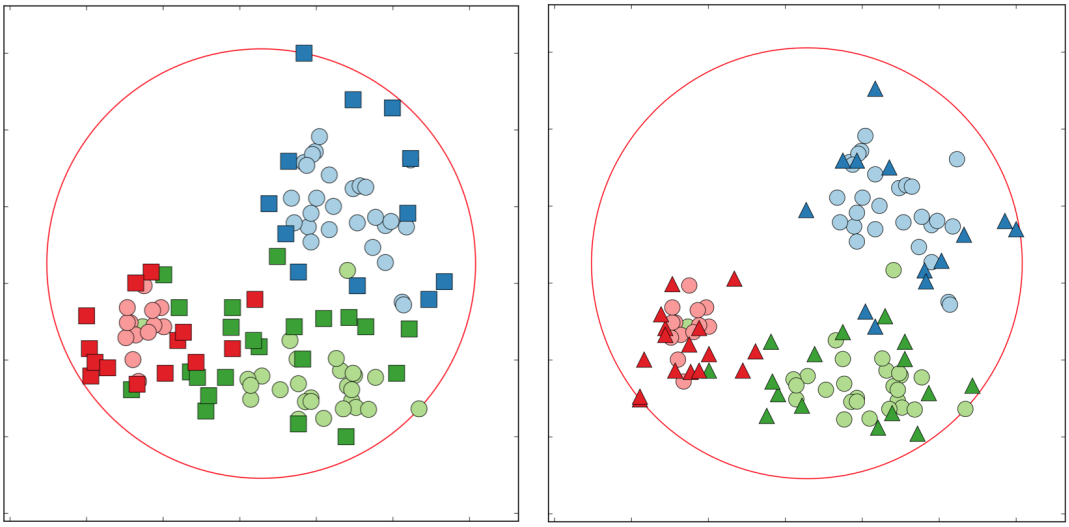

Figure 15. Prototypes selected by Professor-Student from Wine data set. Squares - professor's points, triangles - student's points.

In this section, we assess the quality of prototypes, selected with Professor-Student algorithm, by using them for classification. The algorithm takes a complete set of training data as input and produced output with prototypes is used by 1-NN classifier for training. We make two experiments with two different output sets. In the first case, the classifier is trained on professor's set of prototypes. The second example provides more knowledge for the classification model by putting professor's and student's subsets together.

The results of classification tests, shown in Table 12, gives persuasive evidence that Professor-

Table 12. Error rate (ER) with Professor-Student PS algorithm and 1-NN classifier

	Full Set		**Professor's Set**		**Professor's + Student's Sets**	
Data set	**Arith. Mean**	**Contra-Harm. Mean**	**Arith. Mean**	**Contra-Harm. Mean**	**Arith. Mean**	**Contra-Harm. Mean**
Iris	3.7	11.0	3.9	9.9	2.9	11.0
Wine	12.9	17.6	12.9	19.7	13.0	17.5
Pima	35.6	36.4	29.8	31.0	31.2	32.2
Breast Cancer	5.3	6.3	4.0	5.6	4.1	5.3
Ionosphere	29.0	31.3	26.4	29.6	25.8	28.3
Glass	38.4	41.6	47.1	50.1	38.4	41.5
Bupa	46.8	48.0	46.1	47.5	46.9	48.1
Transfusion	31.3	32.1	27.5	28.4	27.2	28.7
Average	25.38	28.04	24.71	27.73	23.69	26.58

Student algorithm can be used as a PS technique. With prototypes selected by the professor, 1-NN classifier slightly improves its accuracy. With a combined subset of professor's and student's instances, the error rate decreases even more and becomes equal to results obtained with the incremental algorithm based on inscribed circles. We do not evaluate computational complexity for Professor-Student algorithm separately in this section because it is based on the incremental version of PS algorithm, described in Chapter 5, and have the same time complexity ($O(n^{p+1}d)$).

7 Conclusion

This thesis started from the big picture of classification problem and the role of Prototype Selection in it. We discovered that there is a multitude of methods that differ in direction of search and type of selection. These algorithms can be very different in their way to construct the representative subset. However, there is one thing, which makes all of them similar – prototypes get chosen based on classification decisions they make, not on analysis of database structure. We wanted to challenge the evidence of this approach and we suggested to select prototypes on the basis of data set ignorance, i.e. by processing the "voids" within the available data space.

The questions raised in the introduction of this thesis were: how to approach the ignorance discovery in databases and how to benefit from the discovered ignorance in classification and Prototype Selection. In this thesis, we started with a geometrical approach to discover "voids" within databases. Relying on the idea that the most confusion comes from areas that are equidistant from several known points, we presented the concept of ignorance zones. This thesis considers cases in two-dimensional space only, so circles were used as a geometrical abstraction for confusion areas. In order to validate that ignorance zones get discovered in prospective areas, we created a set of artificial databases and qualitatively assessed the list of found confusion zones. Based on the visual analysis, the proposed algorithm was capable of approaching ignorance discovery and forming confusion areas in databases.

Multiple versions of Prototype Selection methods based on confusion zones were proposed to elucidate whether classification can benefit from ignorance discovery. Three selection processes were validated: batch, incremental and adversarial. The batch selection could not improve classification accuracy because of high error rate with noisy data sets. Considerably better results were achieved with the incremental process. Favoring large ignorance zones and avoiding smaller ones, incremental PS method based on Gabriel principle showed the best classification accuracy within proposed algorithms. It could improve classification not only by increasing average accuracy but also by decreasing time spent on decision-making. Adversarial methods brought the most innovative approach to PS by pitting generative model against discriminative. Although they failed to take the lead over incremental algorithms,

they managed to improve results obtained from the classifier without PS.

The future of ignorance learning looks promising. By the example of classification and Prototype Selection, this thesis showed that "void" discovery could improve existing Machine Learning algorithms and make them even smarter. Knowledge about the absence of certain data can be as beneficial as known facts from the database.

In this thesis, there are two considerable limitations: all experiments were run with two-dimensional databases, and ignorance zones were discovered from 2- and 3-parent circles only. Although the primary research goal was to check the hypothesis whether ignorance discovery can improve ML algorithms and we intentionally did not focus on algorithm's implementation effectiveness, the ability for generalization is essential for algorithm's future. On a conceptual level, there should not be significant obstacles with generalization as circles, representing ignorance areas in planar space, would be replaced with hyperspheres in the geometry of higher dimensions. From the practical side, computational complexity analysis of the proposed algorithms revealed that they have exponential time complexity, and it would be hard to generalize them without overcoming this limitation.

The complexity of current algorithms ($O(n^{p+1}d)$) grows exponentially with the number of points p defining a circle or a hypersphere. With higher dimensionality, the number of such parents linearly increases, but it does not necessarily mean that the algorithm must use all of them. Although in N-dimensional space it is possible to create a hypersphere based on up to $N+1$ points, we do not know whether ignorance zones with more parents are more valuable and whether they give better classification accuracy. Hence the trade-off between classification accuracy and computational cost in higher dimensions deserves a separate research article.

An essential part of algorithm time is spent on finding the closest point to the given center. In this thesis, the code just looks over all exemplars and selects the one with minimum distance. Being a straightforward and simple approach, it is not a fast solution and requires linear $O(n)$ runtime to find one nearest point. Based on recent research, the locality-sensitive hashing (LSH) showed itself as an efficient method for checking similarity between elements in PS (Arnaiz-González et al. 2016). Depending on parameters of the hash function, with LSH it

is possible to obtain the closest point with $O(n')$ runtime where $n' << n$. Considering that the PS methods, presented in this thesis, heavily rely on nearest point calculations, the usage of LSH has potential to improve their computational complexity.

Another future direction of research is exploring "voids" with alternative distance metrics. Euclidian metric, used in this thesis, calculates distance based on coordinates only, whereas other metrics, such as Social Distance metric (Terziyan 2017), also take into consideration the distribution of samples within the space, that has a potential impact on classification and clustering.

Bibliography

Amal, Miloud-Aouidate, and BABA-ALI Ahmed Riadh. 2011. “Survey of nearest neighbor condensing techniques”. *IJACSA) Int J Adv Comput Sci Appl* 2 (11).

Angiulli, Fabrizio. 2005. “Fast condensed nearest neighbor rule”. In *Proceedings of the 22nd international conference on Machine learning,* 25–32. ACM.

Arjovsky, Martin, and Léon Bottou. 2017. “Towards principled methods for training generative adversarial networks”. *arXiv preprint arXiv:1701.04862.*

Arnaiz-González, Álvar, José-Francisco Díez-Pastor, Juan J Rodríguez, and César García-Osorio. 2016. “Instance selection of linear complexity for big data”. *Knowledge-Based Systems* 107:83–95.

Blake, Catherine L. 1998. “UCI Repository of machine learning databases, Irvine, University of California”. *http://www. ics. uci. edu/˜ mlearn/MLRepository. html.*

Brighton, Henry, and Chris Mellish. 2002. “Advances in instance selection for instance-based learning algorithms”. *Data mining and knowledge discovery* 6 (2): 153–172.

Cano, José Ramón, Francisco Herrera, and Manuel Lozano. 2005. “Stratification for scaling up evolutionary prototype selection”. *Pattern Recognition Letters* 26 (7): 953–963.

Cover, Thomas, and Peter Hart. 1967. “Nearest neighbor pattern classification”. *IEEE transactions on information theory* 13 (1): 21–27.

Derrac, Joaquín, Salvador García, and Francisco Herrera. 2010. “Stratified prototype selection based on a steady-state memetic algorithm: a study of scalability”. *Memetic Computing* 2 (3): 183–199.

Garcia, Salvador, Joaquin Derrac, Jose Cano, and Francisco Herrera. 2012. “Prototype selection for nearest neighbor classification: Taxonomy and empirical study”. *IEEE transactions on pattern analysis and machine intelligence* 34 (3): 417–435.

Gates, Geoffrey. 1972. “The reduced nearest neighbor rule (Corresp.)” *IEEE transactions on information theory* 18 (3): 431–433.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. “Generative adversarial nets”. In *Advances in neural information processing systems,* 2672–2680.

Hart, Peter. 1968. “The condensed nearest neighbor rule (Corresp.)” *IEEE transactions on information theory* 14 (3): 515–516.

Hawkins, Douglas M. 2004. “The problem of overfitting”. *Journal of chemical information and computer sciences* 44 (1): 1–12.

Hwang, Seongseob, and Sungzoon Cho. 2007. “Clustering-based reference set reduction for k-nearest neighbor”. In *International Symposium on Neural Networks,* 880–888. Springer.

Jaromczyk, Jerzy W, and Godfried T Toussaint. 1992. “Relative neighborhood graphs and their relatives”. *Proceedings of the IEEE* 80 (9): 1502–1517.

Kononenko, Igor, and Matjaž Kukar. 2007. *Machine learning and data mining: introduction to principles and algorithms.* Horwood Publishing.

Liu, Huawen, and Shichao Zhang. 2012. “Noisy data elimination using mutual k-nearest neighbor for classification mining”. *Journal of Systems and Software* 85 (5): 1067–1074.

Machiwal, Deepesh, and Madan Kumar Jha. 2012. *Hydrologic time series analysis: theory and practice.* Springer Science & Business Media.

“Matplotlib 2.2.2 documentation”. 2018. Visited on March 25, 2018. `https://matplotlib.org/`.

Minker, Jack. 1982. “On indefinite databases and the closed world assumption”. In *International Conference on Automated Deduction,* 292–308. Springer.

“NumPy - NumPy”. 2017. Visited on March 25, 2018. `http://www.numpy.org/`.

Olvera-López, J Arturo, J Ariel Carrasco-Ochoa, and J Francisco Martínez-Trinidad. 2010. “A new fast prototype selection method based on clustering”. *Pattern Analysis and Applications* 13 (2): 131–141.

Ougiaroglou, Stefanos, Georgios Evangelidis, and Dimitris A Dervos. 2015. “FHC: an adaptive fast hybrid method for k-NN classification”. *Logic Journal of the IGPL* 23 (3): 431–450.

Reiter, Raymond. 1981. “On closed world data bases”. In *Readings in artificial intelligence,* 119–140. Elsevier.

Ritter, G, H Woodruff, S Lowry, and T Isenhour. 1975. “An algorithm for a selective nearest neighbor decision rule (Corresp.)” *IEEE Transactions on Information Theory* 21 (6): 665–669.

Sánchez, Jose Salvador, Filiberto Pla, and Francesc J Ferri. 1997. “Prototype selection for the nearest neighbour rule through proximity graphs”. *Pattern Recognition Letters* 18 (6): 507–513.

Sánchez, José Salvador, Filiberto Pla, and Francesc J Ferri. 1997. “On the use of neighbourhood-based non-parametric classifiers”. *Pattern Recognition Letters* 18 (11-13): 1179–1186.

“scikit-learn: machine learning in Python”. 2018. Visited on March 25, 2018. `http://scikit-learn.org/`.

“SciPy.org - SciPy.org”. 2018. Visited on March 25, 2018. `https://scipy.org/`.

“seaborn: statistical data visualization”. 2018. Visited on March 25, 2018. `https://seaborn.pydata.org/`.

Shamos, Michael Ian, and Dan Hoey. 1975. “Closest-point problems”. In *Foundations of Computer Science, 1975., 16th Annual Symposium on,* 151–162. IEEE.

Terziyan, Vagan. 2017. “Social Distance metric: from coordinates to neighborhoods”. *International Journal of Geographical Information Science* 31 (12): 2401–2426.

Toussaint, Godfried T. 1980. “The relative neighbourhood graph of a finite planar set”. *Pattern recognition* 12 (4): 261–268.

Triguero, Isaac, Joaquín Derrac, Salvador Garcia, and Francisco Herrera. 2012. “A taxonomy and experimental study on prototype generation for nearest neighbor classification”. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42 (1): 86–100.

Triguero, Isaac, Daniel Peralta, Jaume Bacardit, Salvador García, and Francisco Herrera. 2015. “MRPR: a MapReduce solution for prototype reduction in big data classification”. *neurocomputing* 150:331–345.

Welzl, Emo. 1991. “Smallest enclosing disks (balls and ellipsoids)”. In *New results and new trends in computer science,* 359–370. Springer.

Wilson, D Randall, and Tony R Martinez. 1997. “Instance pruning techniques”. In *ICML,* 97:403–411.

———. 2000. “Reduction techniques for instance-based learning algorithms”. *Machine learning* 38 (3): 257–286.

Wu, Xindong, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. 2008. “Top 10 algorithms in data mining”. *Knowledge and information systems* 14 (1): 1–37.

Yu, Cui, Beng Chin Ooi, Kian-Lee Tan, and HV Jagadish. 2001. “Indexing the distance: An efficient method to knn processing”. In *Vldb,* 1:421–430.

Appendices

A Code snippets for calculating 2- and 3-parent Ignorance Zones

```
import numpy as np
import itertools
import math


def calc_distance(start, end):
    squared_diff = lambda x: (start[x] - end[x]) ** 2
    return math.sqrt(sum(map(squared_diff, range(start.shape[0]))))


def calc_distance_between_points_from_data(data, start_index, end_index):
    return calc_distance(data[int(start_index)], data[int(end_index)])


def cartesian(arrays, out=None):
    """
    Generate a cartesian product of input arrays.

    Parameters
    ----------
    arrays : list of array-like
        1-D arrays to form the cartesian product of.
    out : ndarray
        Array to place the cartesian product in.

    Returns
    -------
    out : ndarray
        2-D array of shape (M, len(arrays)) containing cartesian products
        formed of input arrays.

    """

    arrays = [np.asarray(x) for x in arrays]
    dtype = arrays[0].dtype

    n = np.prod([x.size for x in arrays])
    if out is None:
        out = np.zeros([n, len(arrays)], dtype=dtype)

    m = int(n / arrays[0].size)
    out[:, 0] = np.repeat(arrays[0], m)
    if arrays[1:]:
        cartesian(arrays[1:], out=out[0:m, 1:])
        for j in xrange(1, arrays[0].size):
            out[j * m:(j + 1) * m, 1:] = out[0:m, 1:]
    return out


def calc_radius(data, point_indexes):
    if len(point_indexes) != 3:
        raise ArithmeticError("Radius can be calculated only for triangle")

    sides = [calc_distance_between_points_from_data(data, *side_points) for side_points in
             itertools.combinations(point_indexes, 2)]
    semiperimeter = sum(sides) / 2
```

```
    var = semiperimeter * (semiperimeter - sides[0]) * (semiperimeter - sides[1]) * (semiperimeter - sides[2])
    if 0 >= var > -0.001:
        var = 0.001
    return (sides[0] * sides[1] * sides[2]) / (4 * sqrt(var))


def calc_circumcenter(obs):
    """
    Calculates circumcenter coordinates of a given triangle.

    Parameters
    ----------
    obs : ndarray
       Each row of the 3 by 2 array represents single point in the triangle.

    Returns
    -------
    res : ndarray
      2-dimensional array containing circumcenter coordinates.
    """
    if obs.shape[0] != 3 or obs.shape[1] != 2:
        raise NotImplementedError()

    x1, x2 = obs[0][0], obs[0][1]
    y1, y2 = obs[1][0], obs[1][1]
    z1, z2 = obs[2][0], obs[2][1]

    a_2 = (z1 - y1) ** 2 + (z2 - y2) ** 2
    b_2 = (z1 - x1) ** 2 + (z2 - x2) ** 2
    c_2 = (y1 - x1) ** 2 + (y2 - x2) ** 2

    w1 = a_2 * (b_2 + c_2 - a_2)
    w2 = b_2 * (c_2 + a_2 - b_2)
    w3 = c_2 * (a_2 + b_2 - c_2)
    wsum = w1 + w2 + w3
    if wsum == 0:
        return np.array([np.NAN, np.NAN])

    o1 = (w1 / wsum) * x1 + (w2 / wsum) * y1 + (w3 / wsum) * z1
    o2 = (w1 / wsum) * x2 + (w2 / wsum) * y2 + (w3 / wsum) * z2

    return np.array([o1, o2])


class IgnoranceDiscovery:
    def __init__(self, data, classes):
        self.data = data
        self.classes = classes

        self.allow_ignorance_circles_to_intersect = False
        self.allow_data_inside_ignorance_circles = False

        self.children = list()
        self.children_parents = list()
        self.children_distances_to_parents = list()

    def find_ignorance_points(self, type="mixed"):
        unique_classes = np.unique(self.classes)

        line_distances = np.empty(shape=(0, self.data.shape[1] + 1))
        if type == "line" or type == "mixed":
            combinations = list(itertools.combinations(unique_classes, 2))

            for combination in combinations:
```

```
            local_distance = self._calc_distance_between_classes(combination)
            line_distances = np.vstack((line_distances, local_distance))

    triangle_distances = np.empty(shape=(0, self.data.shape[1] + 2))
    if type == "triangle" or type == "mixed":
        combinations = list(itertools.combinations(unique_classes, 3))

        for combination in combinations:
            local_distance = self._calc_distance_between_classes(combination)
            triangle_distances = np.vstack((triangle_distances, local_distance))

    # sort array with regards to 1st column
    line_sorted_distances = line_distances[line_distances[:, 0].argsort()]
    triangle_sorted_distances = triangle_distances[triangle_distances[:, 0].argsort()]

    line_index, triangle_index = 0, 0

    while len(line_sorted_distances) != line_index or len(triangle_sorted_distances) != triangle_index:

        if len(line_sorted_distances) != line_index and (
                        len(triangle_sorted_distances) == triangle_index or line_sorted_distances[line_index][0] <=
                    triangle_sorted_distances[triangle_index][0]):
            parents_combination = line_sorted_distances[line_index]
            line_index += 1
        else:
            parents_combination = triangle_sorted_distances[triangle_index]
            triangle_index += 1

        distance_between_child_and_parents = parents_combination[0]
        parents_indexes = parents_combination[1:].astype(int)

        parents = np.array([self.data[i] for i in parents_indexes]).astype(float)
        child = self._make_child(parents)

        if not self._validate_child(distance_between_child_and_parents, parents_indexes, child):
            continue

        self.children.append(child)
        self.children_parents.append(parents_indexes)
        self.children_distances_to_parents.append(distance_between_child_and_parents)

    return np.array(self.children)

def _make_child(self, parents):
    if len(parents) == 2:
        child = (parents[0] + parents[1]) / 2
    elif len(parents) == 3:
        child = calc_circumcenter(parents)
    else:
        raise NotImplementedError("Algorithm can create children only for lines and triangles")
    return child

def _validate_child(self, distance_between_child_and_parents, parents_indexes, child):
    if math.isnan(child[0]) or math.isnan(child[1]):
        return False
    if not self.allow_data_inside_ignorance_circles:
        for i in xrange(len(self.data)):
            if i in parents_indexes:
                continue
            elif calc_distance(self.data[i], child) < distance_between_child_and_parents:
                return False
    if not self.allow_ignorance_circles_to_intersect:
        for i in xrange(len(self.children)):
            if calc_distance(self.children[i], child) < distance_between_child_and_parents + \
```

```
                    self.children_distances_to_parents[i]:
                return False
        return True

    def _calc_distance_between_classes(self, target):
        indexes_by_classes = np.array([np.where(self.classes == c)[0] for c in target])
        figures = cartesian(indexes_by_classes)

        distances = np.empty(shape=(len(figures), len(target) + 1))
        for i in xrange(len(figures)):
            figure = figures[i].astype(float)
            if figure.shape[0] == 2:
                diameter = calc_distance_between_points_from_data(self.data, figure[0], figure[1])
                distance = diameter / 2
            elif figure.shape[0] == 3:
                distance = calc_radius(self.data, figure)
            else:
                raise Exception("Algorithm can calculate distances only lines and triangles")
            distance = np.insert(figure, 0, distance)
            distances[i] = distance
        return distances
```

B Code snippets for classifiers with batch and incremental PS

```
def evaluate_classifier_accuracy(x_train, y_train, x_test, y_test):
    model = KNeighborsClassifier(n_neighbors=1)
    model.fit(x_train, y_train)

    results = np.array([model.predict(x_test[i].reshape(1, -1))[0] == y_test[i] for i in xrange(len(x_test))])
    accuracy = len(x_test[results, :]) / len(results)
    return accuracy, results


class BaseSubsetClassifier:
    __metaclass__ = ABCMeta

    def __init__(self, x_train, y_train, train_subset_ratio=0.2, rand=0):
        self.x_train = x_train
        self.y_train = y_train
        self.train_subset_ratio = train_subset_ratio
        self.rand = rand

    @abstractproperty
    def train_subset(self):
        pass

    def calculate_accuracy(self, x_test, y_test):
        x_train, y_train = self.train_subset
        accuracy, results = evaluate_classifier_accuracy(x_train, y_train, x_test, y_test)
        return accuracy


class BatchClassifier(BaseSubsetClassifier):
    @property
    def train_subset(self):
        ign_zone, ign_parents = calc_ignorance_zones(self.x_train, self.y_train)
        parent_indices = np.unique(reduce(np.append, ign_parents)).astype(int)
        self.train_subset_ratio = float(len(parent_indices)) / len(self.x_train)
        x, y = np.take(self.x_train, parent_indices, axis=0), np.take(self.y_train, parent_indices)
        return x, y
```

```
class IterativeClassifier(BaseSubsetClassifier):
    @property
    def train_subset(self):
        domain_type = DomainType.circle
        circle_principle = CirclePrinciple.gabriel

        if domain_type == DomainType.rectangle:
            domain = get_rectangular_domain(self.x_train)
            domain_center = (domain[0] + domain[-1]) / 2
        else:
            domain = get_circle_domain(self.x_train)
            domain_center = domain[1:]

        x_smart, y_smart = np.array([get_closest_point(self.x_train, self.y_train, domain_center)[0]]), np.array(
            [get_closest_point(self.x_train, self.y_train, domain_center)[1]])
        used_zones = []

        while True:
            one_point_zones, borders = calc_one_point_ignorance_zones(x_smart, domain, domain_type, circle_principle)
            mult_point_zones, _ = calc_ignorance_zones(x_smart, y_smart, type='line')

            zones = mult_point_zones
            if one_point_zones.size:
                zones = np.concatenate(
                    [mult_point_zones, one_point_zones]) if mult_point_zones.size else one_point_zones

            smart_point = self.get_next_smart_point(zones, mult_point_zones, one_point_zones, borders, x_smart, y_smart)
            if smart_point is None:
                break
            x_smart, y_smart = np.vstack([x_smart, smart_point[0]]), np.hstack([y_smart, smart_point[1]])
            used_zones.append(smart_point[2])

        self.train_subset_ratio = float(len(x_smart)) / len(self.x_train)
        return x_smart, y_smart

    def get_next_smart_point(self, zones, mult_point_zones, one_point_zones, borders, x_smart, y_smart):
        sorted_zones = zones[(-zones[:, 0]).argsort()]
        for zone in sorted_zones:
            target_point = zone[1:] if zone in mult_point_zones else borders[
                np.where(np.all(one_point_zones == zone, axis=1))[0][0]]
            closest_point = get_closest_point_inside_circle(self.x_train, self.y_train, target_point, zone)
            if closest_point is not None:
                model = KNeighborsClassifier(n_neighbors=1)
                model.fit(x_smart, y_smart)
                if model.predict(closest_point[0].reshape(1, -1))[0] != closest_point[1]:
                    return closest_point[0], closest_point[1], zone
```