Politechnika Warszawska

WYDZIAŁ MATEMATYKI
I NAUK INFORMACYJNYCH

Praca dyplomowa magisterska

na kierunku Informatyka

Rejection Option in Pattern Recognition Problem - Selected Issues

inż. Piotr Waszkiewicz

Numer albumu 254218

promotor

dr hab. inż. Władysław Homenda

WARSZAWA 2017

....

podpis promotora podpis autora

Abstract

An analysis of the presented study seeks a solution to a common problem in a classification issue, which is detecting and rejecting data not suited for classification. Contaminated data that emerges from noisy environment can lead to a situation in which even well trained models yield bad results. This is a serious problem for processes that rely on a classifiers' efficiency in which rejecting received data is more acceptable than classifying it wrongly, e.g. tumour detection algorithm should refuse to make medical evaluation of provided image if it is too blurry rather than trying to guess patient's health condition.

Although artificial intelligence gained much importance and is used in many aspects of humans life (even outside of pure scientific fields), there's still a need for newer approaches and methods. Commonly used algorithms and models change very frequently as new problems arise. Study presented in this thesis introduces modifications to some of the oldest and well known techniques and tries to combine them in order to create tools with much higher capabilities.

Keywords: classification, rejection, svm, random forest, knn, ellipsoid, artificial intelligence

Streszczenie

Jednym z problemów z dziedziny sztucznej inteligencji jest rozpoznawanie i klasyfikacja elementów dotyczących rozwiązywanego problemu. Określenie przynależności nieznanego wzorca jest zagadnieniem o tyle problematycznym, że wymaga ono zadbania o dobrej jakości dane. Odczyty zaszumione, nienależące do dziedziny badanego problemu mogą skutkować błędnymi wynikami. Taka niepoprawna klasyfikacja może mieć poważne konsekwencje - stwierdzenie braku zmian rakowych w organizmie pacjenta z powodu źle wykonanego badania pociąga za sobą o wiele większe konsekwencje niż brak jakiejkolwiek diagnozy.

Celem ninejszej pracy jest przeprowadzenie badań i próba skonstruowania nowych rodzajów klasyfikatorów które posiadać będą cechę odrzucania elementów obcych, czyli nienależacych do rozwiązywanego problemu. Chociaż sztuczna inteligencja jest obecnie prężnie rozwijającą się dziedziną w której nowe podejścia i algorytmy pojawiają się bardzo często, nie można zapominać jak duże znaczenie wciąż mają stare rozwiązania. Podczas przeprowadzanych testów na potrzeby tej pracy wykorzystywane będą właśnie jedne z najnstarszych, a jednocześnie najbardziej popularnych, klasyfikatorów t.j. maszyna wektorów podpierających, lasy losowe czy algorytm knn.

Słowa kluczowe: klasyfikacja, odrzucanie, svm, lasy losowe, knn, elipsoida, sztuczna inteligencja

inż. Piotr Waszkiewicz Warsaw, ...

Nr albumu 254218

Declaration

I hereby declare that the thesis entitled „Rejection Option in Pattern Recognition Problem - Selected Issues", submitted for the magisters degree, supervised by dr hab. inż. Władysław Homenda, is entirely my original work apart from the recognized reference.

....

inż. Piotr Waszkiewicz

Contents

List of Figures

List of Tables

1. Introduction

1.1. Aim of the work

Study presented in this paper tries to combine few selected classifiers in such way that will empower them to gain rejection capabilities. The main goal is to come up with a model having a structure that allows it to reject patterns that are outside of native elements classes' scope without any prior knowledge about such patterns. Those outliers, denoted as foreign elements, are very common in real life situations when dealing with noisy, erroneous or unknown measurements. The main drawback of commonly used classifiers is their inability to reject foreign elements without any knowledge about them. Classifiers such as SVM, random forest or kNN (described better in Chapter 2) must always classify presented pattern to one of the classes they were trained on. This requirement to always classify provided pattern to one of the classes used during training process forces inclusion of foreign elements within training sets if there's a need to reject certain elements. Although not impossible, this approach is quite impractical as most of the time there are just too many possible cases of such elements. This paper tries to find a solution to this problem.

1.2. Pattern recognition problem

Pattern recognition is a branch of machine learning that focuses on the recognition of patterns and regularities in data. The problem of searching for patterns in data is a fundamental one and has a long and successful history. For instance, the extensive astronomical observations of Tycho Brahe in 16^{th} century allowed Johannes Kepler to discover the empirical laws of planetary motion. The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories.[4]

Deploying pattern recognition system starts with collecting and preprocessing data. For most practical applications it is often not sufficient to simply use raw data in the model as it can contain noisy or erroneous information. The input variables should first be preprocessed in order to transform them into some new space where, preferably, pattern recognition problem will be easier to solve. For instance, in the digit recognition problem, the images of the digits are typically translated and scaled so that each digit is contained within a box of a fixed size. This preprocessing stage is also called feature extraction and might be also performed in order to speed up future computations. The aim is to find useful features that are fast to compute and which preserve some discriminatory information. Usually the size of the vector holding extracted features is smaller than the representation of initial raw data, which can be viewed as a form of dimensionality reduction. During the preprocessing stage special care should be taken since reducing dimensionality can often lead to discarding or distortion of useful information.

The result of running the machine learning algorithm can be expressed as a function f(x) which takes input vector and generates response vector which can be used to retrieve class label assigned to the input. The precise form of this function is determined during training phase (sometimes called learning phase), on the basis of training data. Having learned all examples the model's “knowledge“ is checked on the test data. The ability to categorize correctly new examples that differ from those used for training is known as generalization. Creation of training and test sets is often done by dividing initial preprocessed data into two smaller sets. The size of training to test set ratio is completely unrestricted, although in most practical cases 1:1 or 7:3 rates are preferred. Pattern recognition systems are in many cases trained from labelled training data, that means data that has class affiliation for each input vector already determined. Creating model on such data is called supervised learning. When no labelled data are available other algorithms can be used to discover previously unknown patterns, which is called unsupervised learning. The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, also known as density estimation

2. Common Classifiers

The task of classification aims at categorising unknown elements to their appropriate groups. The procedure is based on quantifiable characteristics obtained from the source signal. Those characteristics, i.e. features, are gathered in a feature vector (a vector of independent variables) and each pattern is described with one feature vector. It is expected that patterns accounted to the same category are in a relationship with one another. In other words, subjects and objects of knowledge accounted to the same category are expected to be in some sense similar. There are many mathematical models that can be used as classifiers, such as SVM, random forest, kNN, regression models, or Neural Networks. Their main disadvantage lies in their need to be trained prior to usage, which makes them unable to recognize elements from a new class, not present during the training process. This behaviour can be especially troublesome in an unstable, noisy environment, where patterns sent for classification can be corrupted, distorted or otherwise indistinguishable.

2.1. Implementation

Implementations of the common classifiers described in this chapter were taken from scikit-learn[1] Python library[5]. It is a popular, open source project using BSD license and built on NumPy[2], SciPy[3] and matplotlib libraries. The project was started in 2007 by David Cournapeau as a Google Summer of Code project and is currently maintained by a team of volunteers. The library contains implementations of many algorithms to be used, among others, in classification, regression, clustering, dimensionality reduction and preprocessing problems.

[1] scikit-learn webpage: http://scikit-learn.org

[2] NumPy webpage: http://www.numpy.org

[3] SciPy webpage: https://www.scipy.org

2.2. kNN

The k-Nearest Neighbours algorithm, denoted as kNN, is an example of a “lazy classifier”, where the entire training dataset is the model. There is no typical model building phase, hence the name. Class membership is determined based on class labels encountered in k closest observations in the training dataset, [6]. In a typical application, the only choice that the model designer has to make is selection of k and distance metrics. Both are often determined experimentally with a help of supervised learning procedures. The k-nearest neighbors algorithm used for both classification and regression problems[6]. Example of area coverage for three classes used in kNN classification issue can be seen in Figure 2.1.

The kNN classifier implementation available within scikit-learn package allows to make adjustments to certain parameters that are crucial in classification issue:

- *n_neighbors* - corresponds to the k value, determines number of nearest points used to classify pattern

- *metric* - the distance metric to use for the tree

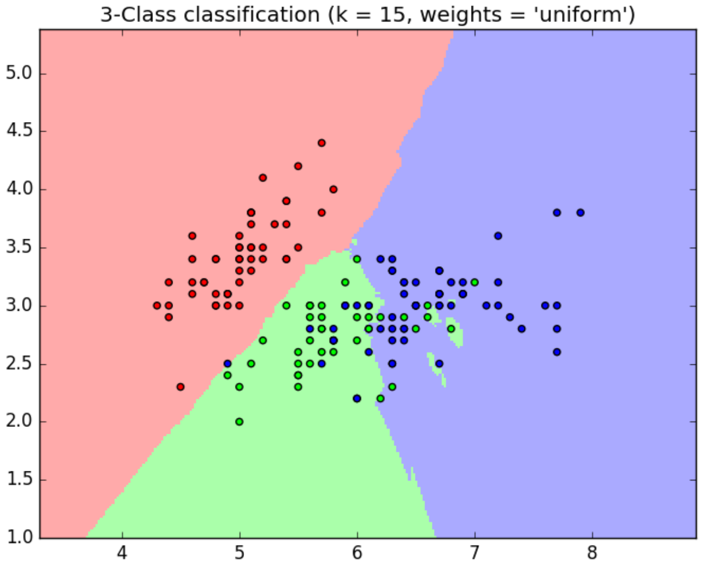


Figure 2.1: Visualization of area coverage of three different class membership for kNN classifier with k=15, using euclidean metric. Image taken from [1]

2.3. SVM

Support Vector Machines (SVM) are a collection of supervised learning methods used for classification, regression and outliers detection. The SVM algorithm relies on a construction of hyperplane with a maximal margin that separates patterns of two classes [7]. Creation of the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin, can be seen on Figure 2.2) is important since, in general, the larger the margin the lower the generalization error of the classifier.

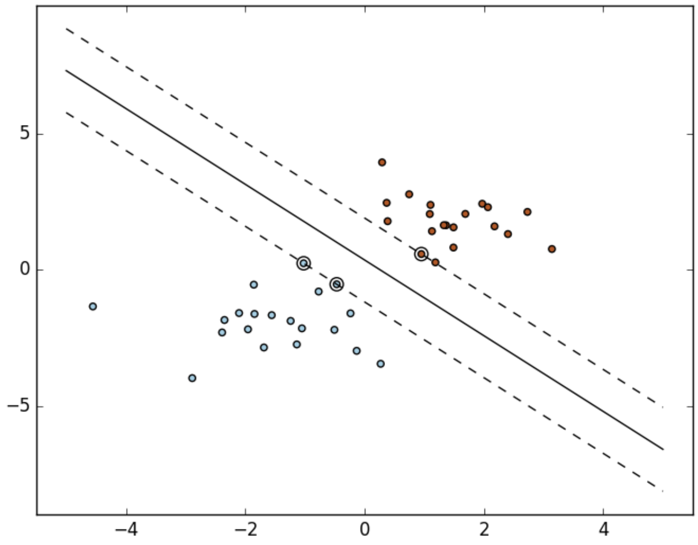


Figure 2.2: SVM hyperplane construction with the biggest possible margin for training dataset. Image taken from [1]

In SVM's mathematical definition the two classes' labels are denoted as -1 and 1. When treating elements from those sets as points of the Euclidean space $\mathbb{R}^n$ (or vectors of this space) the SVM training can be seen as the problem of finding the maximum-margin hyperplane that divides those samples. This issue can be described by formula:

$$\vec{w} * \vec{x} - b = 0$$

where $\vec{w}, \vec{x} \in \mathbb{R}^n, b \in \mathbb{R}$. The $\vec{x_i}$ vectors are samples from the training set, and $\vec{w}$ is a normal vector to the hyperplane, obtained as a linear combination of those training vectors that lie at borders of the margin:

$$\vec{w} = \Sigma_i \alpha_i \vec{x_i}$$

Those of the training vectors $\vec{x_i}$ that satisfy the following condition:

$$y_i(\vec{x} * \vec{x_i} - b) = 1$$

are called support vectors, and have their corresponding $\alpha_i \neq 0$. The $y_i \in \{-1, 1\}$ corresponds to the class labels that training data consists of. The linear decision function used for classifying patterns is expressed as follows:

$$I(\vec{x}) = sgn(\Sigma \alpha_i \vec{x_i} * \vec{x} - b)$$

where $\alpha_i \vec{x_i} = \vec{w_i}$. SVM efficiency can be enhanced by using different kernel functions which help in solving non-linearly-separable problems. The generalized decision function using kernel function K:

$$I(\vec{x}) = sgn(\Sigma \alpha_i K(\vec{x_i}, \vec{x}) - b)$$

The sample visualization of some kernel functions can be seen on Figure 2.3, where different class area coverages can be seen depending on the kernel function used. It's worth noting that the two higher images, although using the "same" linear kernel, present different results. This comes as a consequence of miscellaneous internal implementation changes that are very technical and out of scope of this paper.

In some cases when the data are not linearly separable the hinge loss function must be introduced

$$max(0, 1 - y_i(\vec{w} * \vec{x_i} - b))$$

This function is zero if the $\vec{x_i}$ lies on the correct side of the margin. On the other hand, if the data are on the wrong side, the function's value is proportional to the distance from the margin. The aim of the SVM is to minimize the value of the hinge function for every element $\vec{x_i}$ from the training set.

The SVM algorithm has been widely applied in the biological and other sciences. They have been used to classify proteins with up to 90% of the compounds classified correctly. Permutation tests based on SVM weights have been suggested as a mechanism for interpretation of SVM models[8][9]. SVMs are effective in high-dimensional spaces, memory efficient, and quite versatile because of the many kernel functions that can be specified for the decision function. Implementation available as part of scikit-learn package lets user specify and tweak many aspects of classifier such as:

- C - penalty parameter C of the error term, used to regularize the estimation. If dealing with noisy observations it's recommended to decrease its value

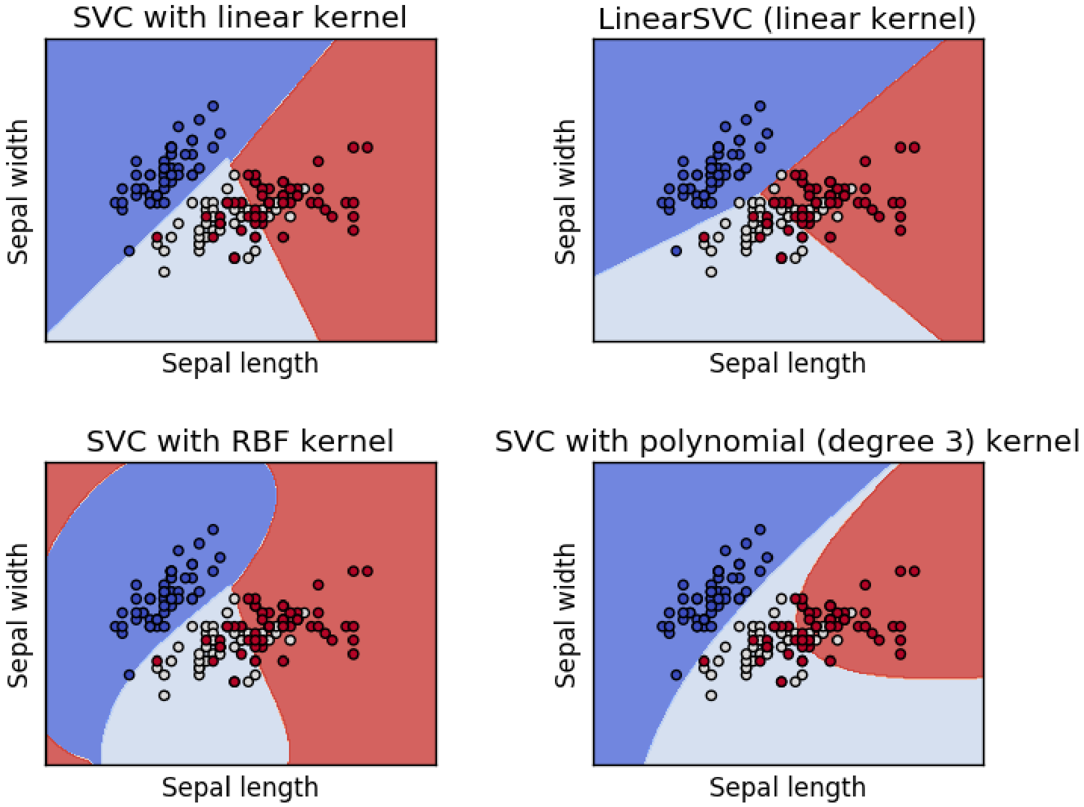


Figure 2.3: Different class area coverages as a result of using different kernel functions. Image taken from [1]. Please note that the two higher images show different area coverages for the “same“ linear kernel. This difference comes from different implementations of SVC and LinearSVC classes. SVC supports multiclass classification according to one-vs-one scheme, whereas LinearSVC does it by using one-vs-rest method. According to official documentation: “LinearSVC is implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.“[1]

- *kernel* - kernel type used in the algorithm, in this paper one of "poly" or "rbf" values are used. "poly" stands for polynomial kernel using following equation $(\gamma\langle x, x'\rangle + r)^d$ (where d is function degree, with default value 3), "rbf" is an acronym for radial basis function with given equation $exp(-\gamma|x - x'|^2)$
- *gamma* - kernel coefficient for "rbf", "poly" types as can be seen it the kernel equations
- *degree* - degree of the polynomial kernel function

It is worth noting though that in some cases, where the number of features is much greater than the number of samples, using support vector machines can give poor results, and is not cost-efficient when calculating probability estimates.

2.4. Random Forest

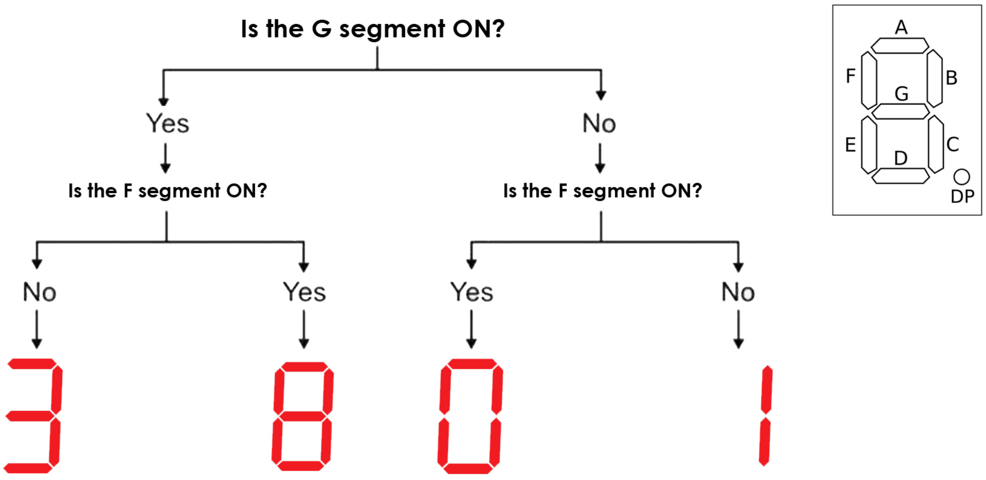


Figure 2.4: An example of small decision tree. Each node contains information about some feature of the pattern. In this example the lit segment is such information.

Random forest is a popular ensemble method. The main principle behind ensemble methods, in general, is that a group of “weak learners” can come together to form a “strong learner”[10]. In the random forest algorithm [11] the weak learners are decision trees, which are used to predict class labels. A decision tree is a decision support tool that uses a tree-like graph for classification issue. Each graph node performs a test on an attribute of the provided pattern and sends it to its child node via a branch that represents the outcome of the test. Each leaf in a decision tree represents a certain class label. In other words for a feature vector representing one pattern a decision tree calculates its class label by dividing value space into two or more subspaces. More precisely, input data are entered at the top of the tree and as it traverses down the tree the data get bucketed into smaller subsets. An optimal decision tree is defined as a tree that accounts for most of the data, while minimizing the number of levels[12]. There are many advantages of using decision trees. Their results are easy to interpret and visualize in form of a graph, they can handle multi class classification problems and perform well even if its assumptions are somewhat violated by the true model from which the data were generated. On the other hand, the main drawbacks connected to their usage consist of overfitting problem caused by creating too complex trees on a very complicated data, and instability caused by small variations in the data that might result in a completely different tree being generated. That last problem is easily mitigated by ensembling set of decision trees into a random forest.

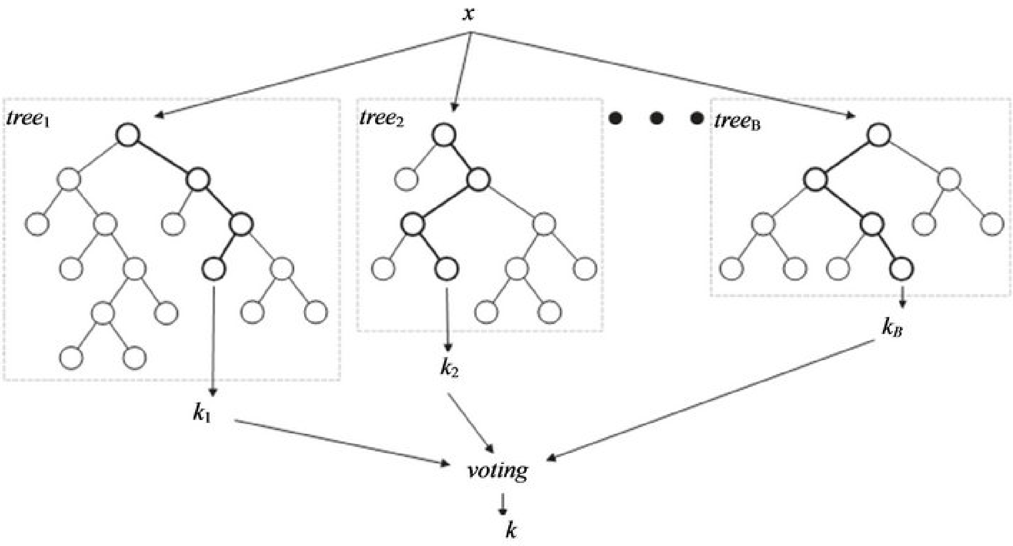


Figure 2.5: Visualization of a random forest consisting of B different decision trees

In the random forest a large number of classification trees is formed, which altogether serve as a classifier. In order to grow each tree, a random selection of rows from the training set is drawn. Random sampling with replacement is also called bootstrap sampling. In addition, when constructing trees for a random forest at each node m variables out of the set of all input variables are randomly selected, and the best split on these m is used to split the node. After a relatively large number of trees is generated, they vote for the most popular class. Some of the parameters used for improving classification rates that are available within scikit-learn package random forest implementation:

- *n_estimators* - determines number of trees used by random forest in the algorithm
- *max_depth* - the maximum depth of each tree in the forest

- *max_features* - the number of features to consider when looking for the best split
- *min_samples_leaf* - the minimum number of samples required to be at a leaf node

Random forests join few important benefits: (a) they are relatively prone to the influence of outliers, (b) they have an embedded ability of feature selection, (c) they are prone to missing values, and (d) they are prone to over-fitting.

2.5. Minimum Volume Enclosing Figures

The easiest and probably most intuitive way of dealing with classification task is using patterns' spatial relations in order to determine their class memberships. This approach is used for example in kNN and SVM models, where point's affiliation is calculated based on its place in the features space. Every class in a dataset can be viewed as a big "cloud" of points and usually, if the other clouds do not overlap each other, the more dense this cloud is, the easier the task of classification gets.

Each class' cloud can be enclosed in an arbitrary geometrical shape which can be used as an identifier. The difference between binary classifiers and identifiers is very subtle, yet important. Whereas binary classifiers can distinguish between patterns from two different classes, they must be trained on data consisting of elements from both classes. Identifiers accept (or one could say: identify) only those points that they were constructed on, and reject any outliers. Thus they require only one class to be provided during training process. One of the easiest and most intuitive model of identifier is minimum volume enclosing figure. Creating figure enclosing all elements that has the smallest volume possible ensures that the identification can be very strict, which in return helps to maintain high outlier rejection rate. As opposed to convex hull, which is the most accurate point set container with smallest volume and which is enclosed by linear hyperplanes, bounding figures are far less complex. In many cases, when there is a need for computing convex hull and testing inclusions of other points, an approximation of such hull can be used, which helps in reducing time needed for computations, since most of alternative methods have lower construction and inclusion-testing complexities. Among the most popular minimum volume enclosing figures there are: boxes, diamonds, simplexes and ellipsoids.

2.5.1. Minimum Volume Enclosing Ellipsoid

Minimum Volume Enclosing Ellipsoid (denoted as MVEE) problem is solved by several known algorithms that can be categorized as first-order, second-order interior-point or combination of

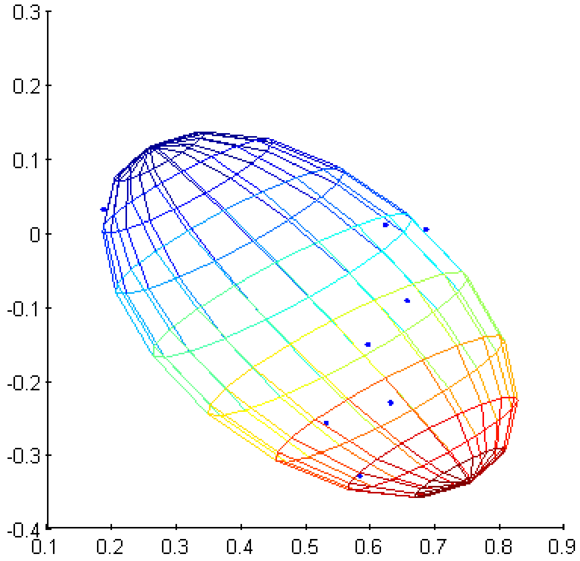


Figure 2.6: An example of minimum volume enclosing ellipsoid for points in 3D space

the two. For small dimensions d, the MVEE problem can be solved in $O(d^{O(d)}m)$ operations using randomized or deterministic algorithms [13]. All the results presented in this chapter were obtained while using MVEE algorithm based on Khachiyan solution.

An ellipsoid in its centre form is given by the formula:

$$E = \{x \in \mathbb{R}^n | (x-c)^T A(x-c) \leqslant 1\}$$

where $c \in \mathbb{R}^n$ is the centre of the ellipse E and A is a positive definite matrix. Points lying inside the ellipsoid satisfy

$$(x_i - c)^T A(x_i - c) \leqslant 1 \tag{2.1}$$

The technical problem that arises from using above equation comes from the inaccuracy of floating point numbers used by computers. Thus the need for providing additional variable ε

$$(x_i - c)^T A(x_i - c) \leqslant 1 + \varepsilon \tag{2.2}$$

which defines the error margin in determining whether certain point belongs to the ellipsoid.

The main problem, when using ellipsoids as identifiers, lies in constructing them. Two main factors that decide about identification effectiveness are tolerance and acceptance parameters. Tolerance can be viewed as a threshold for ellipsoid construction accuracy (and the stop condition used in algorithm). The lower the parameter is, the more precise minimal volume ellipsoid (in terms of training point inclusion) is created. On the other hand, even with a good training set, there is a risk of not including native patterns that lie outside of the created ellipsoid (which were for example not included in the training set). Acceptance parameter (denoted as ε in Equation

2.2) has been introduced to prevent such unwanted behaviour. It defines a threshold for point rejection for elements lying outside of the created figure. Manipulating this parameter can be viewed as “enlarging“ ellipsoid (by increasing ε value) or “shrinking“ it (by decreasing ε value).

2.5.2. Minimum Volume Enclosing Hyper Rectangle

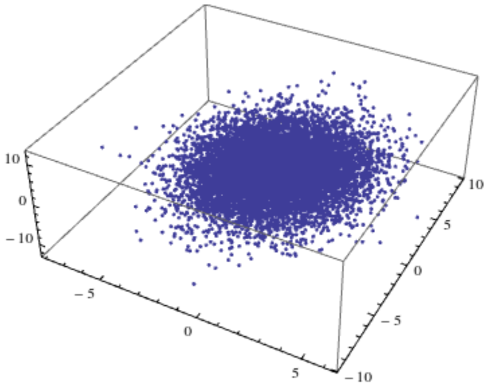


Figure 2.7: An example of minimum bounding box for points in 3D space

Out of all possible minimum volume figures, the hyper rectangle, also called smallest enclosing box, is probably the easiest and most intuitive solution. To create such hyper rectangle one needs to store biggest and smallest values for each dimension from the training data set. Although it is straightforward to find the smallest enclosing box that has sides parallel to the coordinate axes, the difficult part of the problem is to determine the orientation of the box in the feature space. In this paper only hyper rectangles with their sides parallel to the axes are considered. The inclusion test performed for each point is done by checking whether each of the point’s coordinates lies within bounds of enclosing box, and no information about the distance to the rectangle’s centre is given. In case of situation when there’s a need for such information the following solution, presented in form of a pseudo-code (Algorithm 1) can be used.

```
Input: hp - hyper-rectangle centre coordinates, p - point for which inclusion test is
       performed
Output: d - distance of the point p to the hyper-rectangle centre using maximum metrics
d = -1
foreach coordinate in hp do
    dtmp = |p - coordinate|
    if dtmp < d then
        d = dtmp
    end
end
return d
```

Algorithm 1: Algorithm for calculating point distance to hyper-rectangle centre using maximum metrics

3. Classifiers with rejection option

3.1. Classifier Trees

Common classifiers described in the Chapter 2 return results in form of a class label that provided pattern was classified to. Such approach leaves no room for estimating class-belonging probabilities which, in return, results in inability to reject provided data, treating it as an outlier. By combining those classifiers and organising them in a complex structures it is possible to create objects with unique rejection capabilities in exchange for slightly increased pattern-processing time. This chapter describes such structures, shaped in form of binary trees.

3.1.1. Balanced Tree

Structure

The main idea behind Balanced Tree structure is to create a graph tree in which every path from root to leaf consists of increasingly precise classifiers. What it means is that every pattern, that should be classified, is tested against certain number of common classifiers, where each subsequent one is clarifying this unknown pattern's affiliation to one of the classes.

The Balanced Tree construction begins with creation of a root node which represents a situation in a classification process in which all possible class memberships for an unknown pattern are taken into account. It can be said that the root of the Balanced Tree represents a set consisting of all classes in the training set, because it is yet unknown from which class a pattern would be. The process of clarifying pattern's class belonging starts by designating the central points for each class in the set of classes represented by this node. This is done by calculating arithmetic average of all points from certain class set:

$$p_{central} = \frac{\sum\limits_{i=1}^{n} p_i}{n}$$

where n is number of elements p_i belonging to certain class in the dataset. Next step involves using clustering algorithm to divide all of those central points into two distinctive sets. The idea is to group those class representatives that are most similar to each other. The process of Balanced

Tree structure creation is continued further by passing two classes sets designated by clustering algorithm, one to each child nodes. The process of new node creation is then applied to each of those two child nodes and continued until there is only one class left. A node representing only one class cannot use clustering method because there is insufficient number of classes to divide, and so it becomes the tree leaf.

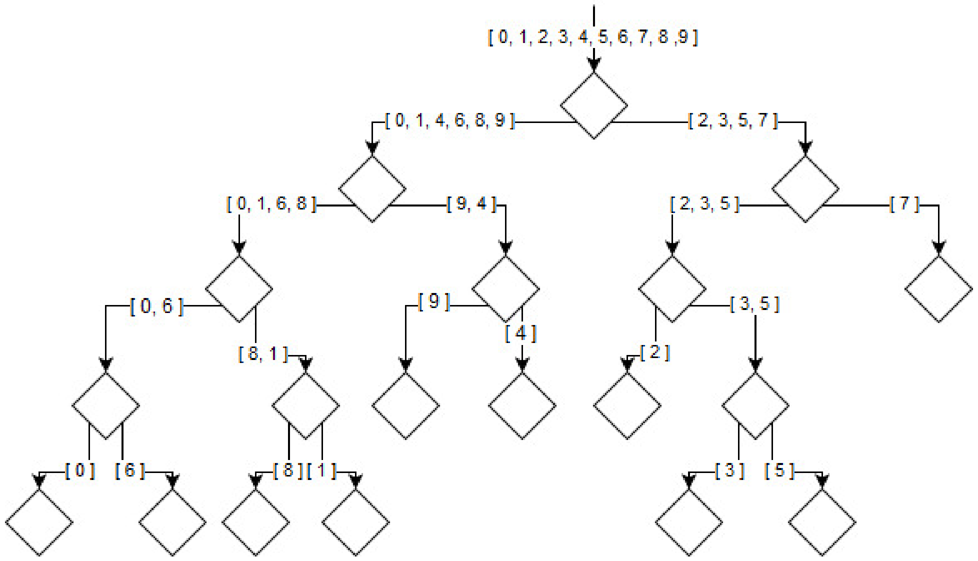


Figure 3.1: Balanced Tree structure obtained during real life tests

Classifiers creation

After finishing Balanced Tree architecture creation each non-leaf node is assigned a binary classifier trained on a data consisting of a training samples from classes assigned to this particular node. Those classes that are represented by its left child node are joined together and treated as a class '0', while the ones in the right child node are labelled as a class '1'. For example, if there are four classes assigned to a certain tree node, labelled as a, b, c, d, and a clustering method divided them into two sets a, d (assigned to left child node) and b, c (assigned to right child node), the classifier will be trained on data samples treating points from classes a, d as if they all were from an artificial class '0' and classes b, c as class '1'. The only issue that arises from such attitude is inability for leaf nodes to have their own classifiers. This is due to leafs being the last nodes in a tree, having no child nodes. To circumvent this shortcoming a solution is proposed that treats leaf node as if it had left child with the same assigned class as a parent, and a right child with assigned every existing class in the training dataset except for the class assigned to its sibling (left child node).

Classification rules

When an unknown, new pattern is presented to the Balanced Tree, it traverses a path from a root to a leaf node in order to be classified or rejected. This path strongly depends on classifiers in each node and their classification decision. As it was described earlier each node is assigned certain number of classes that it represents. The main task of each node's classifier is to decide if the provided pattern belongs to internal class '0' or '1'. In other words it tries to determine to which set of classes this unknown elements is most similar to. After decision is taken the patterns is sent further to the left child node in case it was classified as '0' or right one if classified as '1'. Each subsequent classifier is more precise and better clarifies pattern's class affiliation. After reaching leaf node the final classification test is made. The classifier in a leaf node is trained in an one-versus-all manner. If the unknown element is recognized as a member of a class assigned to this particular leaf, it is finally labelled as an element from that class. On the other hand if it is classified as a "rest" pattern, it gets rejected. The scheme for this approach can be seen on Figure 3.2. Rejection relies on the assumption that if the pattern traversed path all way down to the leaf node, while being sent to next nodes basing on increasingly strict classifiers' decisions, and ends up being recognized as a point from outside of most probable class (the one assigned to the leaf node), then it probably is not similar enough to any class from the training set.

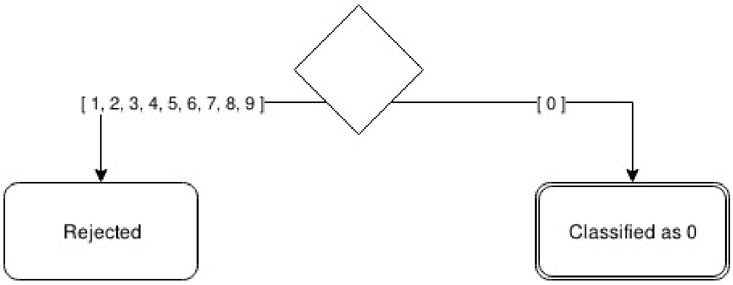


Figure 3.2: Balanced Tree rejection scheme in leaf node

Implementation details

Creation of Balanced Tree structure starts from tree root and is done recursively. Each node, that is not a tree leaf, is assigned certain set of classes which is a subset of all classes in a tree (root node is assigned all). The next step involves clustering method dividing node's class set into two disjoint sets. This procedure is done on 'class central points' which are average points of all elements in each class. Clustering algorithm divides those points thus providing two new sets for both child nodes. After that node trains its classifier on data set consisting of two classes created by taking all elements from training data for left and right child nodes' classes sets. The node-creation procedure is then applied for both node's children. The leaf creation algorithm is slightly different as it does not need usage of clustering. Classifier is trained on data set created from combining elements from training data that belongs to the same class the leaf node represents (those points' new class is labelled '0') and elements from every other class (which are labelled '1'). To ensure that both '0' and '1' classes have the same number of entries the '1' class set must be trimmed. This is done at its creation step by taking less elements from each class in order to have the same number (or nearly identical) of elements overall in the whole set, e.g. having training data set consisting of ten classes labelled from '0' to '9', with total of 10,000 elements, set '0' for leaf representing class '2' will have 1,000 entries of elements from class '2' taken from training data and set '1' will have 999 elements in total but will consist of elements from classes '0', '1', '3', '4', '5', '6', '7', '8', '9' taken from training data with 111 elements from each class.

3.1.2. Slanting Tree

Structure

The Slanting Tree structure differs greatly from Balanced Tree's one. The concept implemented in Slanting Tree assumes that the unknown pattern, that is sent for classification, should be iteratively compared against each class representatives. Only when it is similar enough to points from certain class, more precise tests are made that ensure its affiliation. Should the tests fail, the pattern continues its iteration over other classes as if wasn't ever supposed to belong to this class. Rejection occurs when every test fails.

Construction of the Slanting Tree is fairly simple, unlike the Balanced Tree. Each node represents exactly one class. Nodes are chained together in such manner that when traversing Slanting Tree from the root to the last node by choosing always the left child, exactly one node for each class in the training set is visited. Each of these nodes has also a right child that can be treated as a node between its parent and its parent's left child, that extends the path received when going from root to the last node by taking always the left node's successor. Each right child node represents the same class from the training data set as its parent does.

Classifiers creation

Each tree node in Slanting Tree has its own binary classifier, trained in a 'one-versus-rest' manner. Training is done on data containing two classes, where the first one consists of patterns from the training set which belong to the same class as the node represents, and the second one is obtained by concatenating patterns of each class from the training set except for the class represented by the node. To prevent the situation in which node and its right child have the classifier trained on the same data (because both nodes represent the same class) certain changes to training patterns must be introduced. This ensures that every classifier in the Slanting Tree is unique and can be used during classification procedure.

Classification rules

The classification starts from the root node, where the unknown pattern is tested by the first classifier and has its class affiliation checked. If the obtained result indicates that it isn't similar to the class represented by this node (gets classified as an element from the 'rest' class), the classification process is continued in the next node, which is the left child of the current one. If the opposite situation occurs, and the classifier accepts presented pattern as a representative of current node's class, the process is repeated in the right child node which uses more strict classifier. This is done to ensure that the unknown element, which is supposedly from the certain class, really belongs to it. If this test fails, the pattern is sent to the node as if the previous test also failed (it is sent to the left child of the current node's parent). In case of success the pattern gets successfully classified. When all tests fail (there is no more nodes to send pattern to) the element gets rejected.

Implementation details

Creation of Slanting Tree is done recursively, starting from the root node. All classes that should be distinguishable by this tree structure are sorted by their labels and stored in an array object. This object is later used during node creation method to check what classes have already been covered by previous nodes. Every non-leaf node represents only one native class and has its binary classifier trained in 'one-vs-rest' manner, the same way the tree leafs' classifiers in Balanced Tree are (see 3.1.1). The next step involves creating left child node for the next native class in the array object that has not yet been used. In case of no classes left the function returns without creating new node. The last step consists of right child creation, which is a leaf node. Leafs in a Slanting Tree represent the same native classes their parent node did, but their classifiers, although built using same 'one-vs-rest' approach, are trained on a different data sets in order to create more accurate results. Usually trained classifier does not achieve 100% accuracy even on a training test that was used during its creation. There are some samples from first class that get classified as elements from the second and vice versa. Such mistakes can help determine what kind of corrections can be made to the classifier. For every non-leaf node, after its classifier training, there's set of elements from the first class that were correctly recognized (those are the elements from the class this particular node is representing) and set of elements from the second class that were mistakenly recognized as elements from the first class. Those two sets are used in this node's child leaf node's classifier creation. Of course before training those two sets must be the same size, ideally having the same number of elements as two sets used in parent's classifier training. For each missing element in either of sets the new object is generated by randomly selecting one element from this set and applying normal distribution (with standard deviation 1) to all of its features in a feature vector, thus getting new sample that can be added to the set. In case of having less than certain number of elements (implementation checks for 10 or less elements) in either of sets before new point generation algorithm takes place, those sets are filled with randomly selected points from parent node's classifier training sets.

3.1.3. Slanting Tree with ordered classes

Description

The basic Slanting Tree structure assigns classes to its nodes based on an arbitrary, lexicographic order. This approach leaves its implementation vulnerable to situation in which label changes occur. Slanting Tree with ordered classes tries to circumvent this disadvantage by sorting classes without the need to know their labels, based on their spatial relations. Every set of points belonging to certain class in a training dataset can be transformed into one point, being the centre of that particular class. Next the distance to every other centre point in the training dataset is calculated for each class centre, and only the lowest value is saved. The new class order is based on those values, which are sorted in the descending order.

Implementation details

Steps required to build Slanting Tree with ordered classes are mostly the same as in 3.1.2. Instead of creating nodes for classes using their lexicographic order, the ordering technique described in the previous Section is used. During computations only one point, called the class central point, for each class in the training dataset is used. Those are calculated the same way as in 3.1.1, by getting the average value of all training patterns from one class. Sorted class labels are used further during classifier tree creation process the same way as in original Slanting Tree.

3.1.4. Slanting Tree 2

Description

Much like previously described Slanting Tree, this one has its nodes arranged in the same architecture. The difference lies in leaf nodes which, unlike the original Slanting Tree, are not

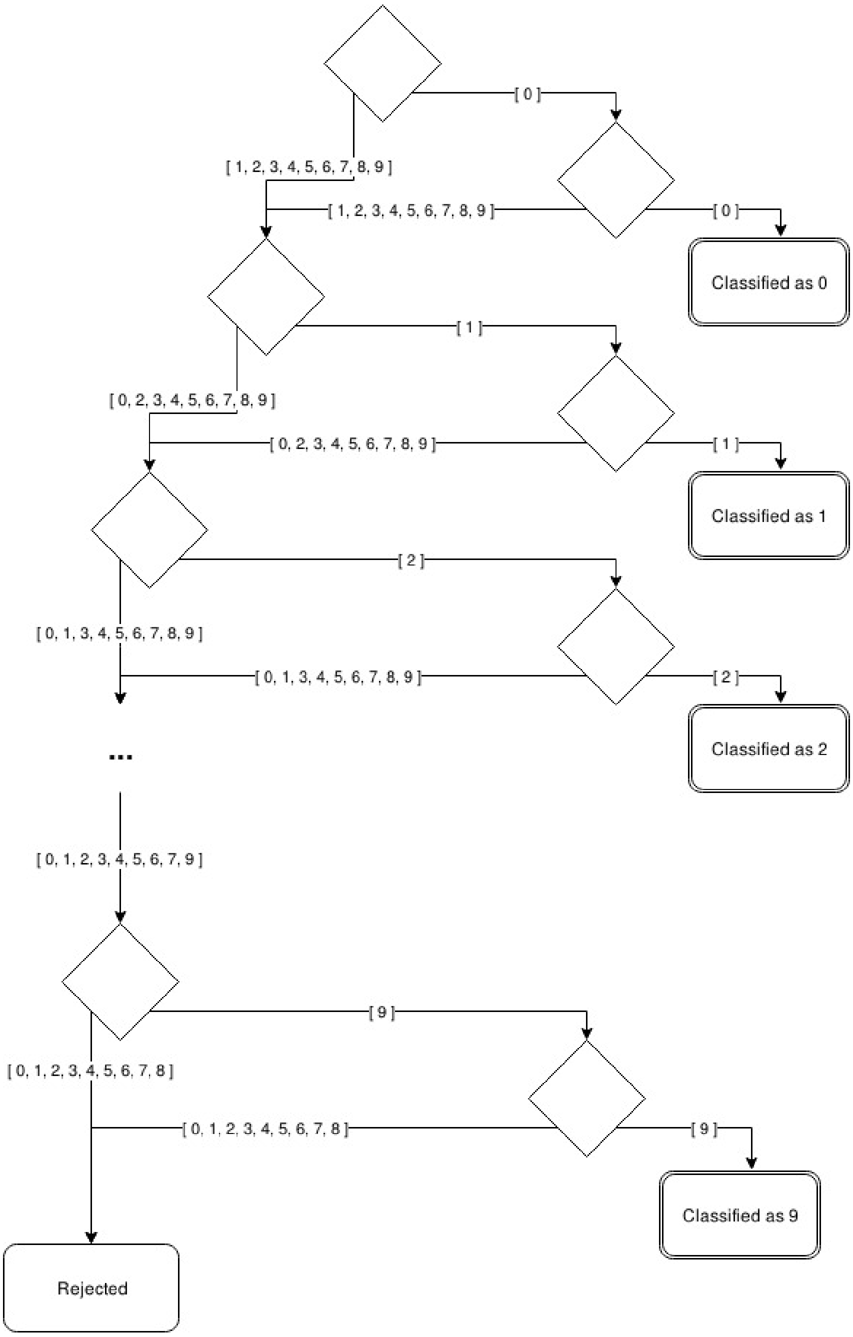


Figure 3.3: Bottom part of the Slanting Tree with nodes for classes 8 and 9 (nodes for classes from 0 - 7 not visible).

using modified training data sets and use different classifier types instead (e.g. parent nodes using SVM classifier and their right children nodes using random forest). The idea behind this implementation relies on the assumption that various classifiers tend to wrongly classify different patterns, so when combining them rejection rate as well as classification rate should be vastly improved. Other than that there are no further changes and everything described in the Section 3.1.2 applies to Slanting Tree 2.

Implementation details

Creation procedure is mostly the same as in 3.1.2. The only differences are present in nodes creation method where instead of creating new training patterns for the right child node, different classifier type is trained on the same data from the parent node.

Summary

The experiments performed on classifier trees and their results are described in details in Chapter 4. All of the classifier trees introduced in this chapter have good classification capabilities, very similar to the plain common classifiers they use. It is worth noting that not only does the classification rate stay the same, but also rejection capabilities are introduced. Among all classifiers combinations tested the Slanting tree using random forests with 100 estimators performs the best. Tables 4.3 and 4.4 show score achieved by this tree structure. Although being the best, classification rate achieved by this particular Slanting Tree may not be considered good, as it's lower than 50%. At best it could be seen as mediocre. Despite trying different classifiers and their parameters combinations no better solution could be found while using tree structures described in this chapter. The final conclusion can be made that the classifier trees introduced in this paper do not perform well enough to be used as a valid rejection mechanism. While still maintaining high classification rates those structures are slower than other popular classifiers which questions their usefulness.

3.2. Classifier Arrays

Another approach towards classification with rejection option problem involves chaining classifiers trained on certain, very specific data. The array of those classifiers serves as a voting mechanism where every new and unknown pattern is presented to each classifier in this array and the classification (or rejection) decision is made based on the overall achieved score. All classifiers inside the array are binary ones but can be divided into two groups:

- one-versus-all - those classifiers are trained on two sets, where the first one consists of training patterns from certain class, and the second one is made of all patterns from the training set except for those from this certain class
- one-versus-one - every classifier is trained on training patterns from two different classes

3.2.1. One-versus-all

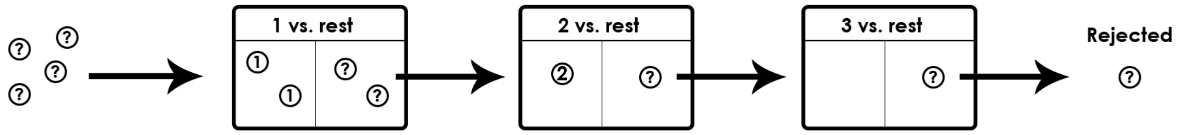


Figure 3.4: “one-versus-all” rejection method. Unknown pattern passes through an array of specially prepared classifiers, one for each class. If each classifier says it is not native, it is rejected.

Description

The “one-versus-all” method requires creating an array of binary classifiers. Training data set for each classifier in this method consists of two sets: the first one (denoted as “class_i”) holding all training data for certain i-th native class, and the second one (denoted as “rest”) being the result of a subset sum operation performed on the rest of the classes except for the class used in class_i set. One problem with such approach is that as a result of the subset sum, class “rest” could contain significantly more samples than “class_i”. In such case the “rest“ set is udersampled.

Implementation details

The actual classification with rejection is performed by presenting the unknown pattern to each of the classifiers from the array. When any classifier recognizes this element as a native one (belonging to class_i), then the pattern is treated as a recognized one, and it is assumed to be native. In a case when all classifiers reject a pattern (all binary classifiers say that it belongs to set “rest”), it is treated as a foreign pattern and it is rejected. It is worth noticing that there is a possibility that more than one classifier recognizes the pattern as a native element. In such case randomly chosen class label is assigned to this pattern. The scheme for this method is sketched in Figure 3.4.

3.2.2. One-versus-one

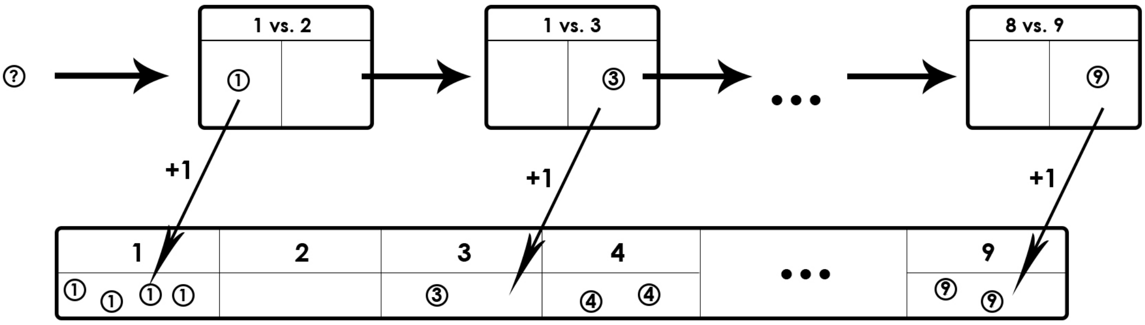


Figure 3.5: “one-versus-all” rejection method. Unknown pattern passes through an array of specially prepared classifiers, one for each class. If each classifier says it is not native, it is rejected.

Description

The “one-versus-one” method requires preparing an array of classifiers, but this time it consists of $\binom{c}{2}$ classifiers, where c is the number of native classes. Each classifier is trained on data consisting of two sets: the first one (denoted as class_i) holding all training data entries for i-th native class, and the second one (denoted as class_o) holding all training data entries for some other class (not the same as class_i). In the end, there is one classifier for each pair of classes: 1 vs. 2, 1 vs. 3, ..., 1 vs. c, ..., $(c-1)$ vs. c.

Implementation details

Classification with rejection mechanism is based on presenting unknown pattern to each classifier in the vector and remembering their answers (e.g. classifier constructed for 1 vs. c classes can classify the pattern as belonging to class 1 or class c). In the end, those answers can be summarized and for each pattern a c-elements array with numbers saying how many times this pattern was classified as belonging to class $1, 2, 3, \dots, c$ can be formed. The pattern is rejected when the difference between two biggest values in the result array is smaller than two. In such case, it is assumed that the classifiers were highly uncertain as to which class should this unknown element belong to. Otherwise, the pattern is classified as an element belonging to the class which had the biggest value in the result array. The general scheme for this method is presented in Figure 3.5.

3.2.3. One-versus-one modified

The modified "one-versus-one" method is based on the"one-versus-one" method discussed in 3.2.2. The difference between those two methods lies in a rejection mechanism. In this method an unknown pattern is treated as a foreign element if the biggest value in the result array is smaller than $(c-1)$. What it actually means, is that there must be a certain class that has always been chosen by a classifier whenever it was possible.

3.2.4. Summary

The experiments performed on classifier trees and their results are described in details in Chapter 4. Classifier arrays behave differently based on what approach is used. Whereas one-versus-all technique scores high in classification, it lacks in rejection option greatly. On the contrary the one-versus-one algorithm tends to reject all patterns, even the native ones. The third method, which is modified one-versus-one approach, brings balance between classification and rejection rates.

3.3. Geometrical classifiers

3.3.1. Minimum Volume Enclosing Figure array classifier

The construction of a classifier using array of identifiers is pretty simple and straightforward. The array is filled with minimum volume enclosing figures, one for each class in the training dataset. Every unknown pattern, sent for classification, moves through those figures and gets the information whether it lies inside the figure or not. In case of being part of identifier's interior the value given by the inclusion-equation (for ellipsoid see Equation 2.2) is taken into consideration, which tells about the position inside the figure (where value 0 means that the point is in the figure's centre and 1 means that it lies on the surface). Finally, after all figures are checked, the one with the smallest equation value (and for which the point lies inside it), designates the class to which this unknown pattern should be classified to. If no figure accepts the point, it is rejected and treated as a foreign element.

3.3.2. Optimizing figure size

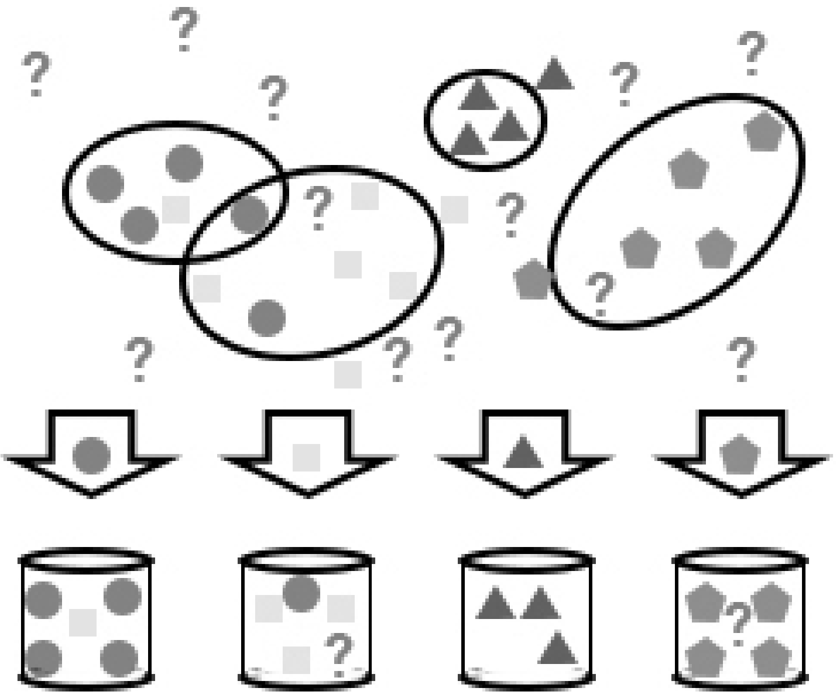


Figure 3.6: Array of ellipsoids that works as a classifier with rejection capabilities. Elements outside of existing ellipsoids are treated as a foreign patterns. Those within ellipsoids are classified as native patterns.

One way of increasing rejection ratio would be to decrease figure size. The smaller the volume the fewer points lie inside, which in return boosts rejection rates but worsens classification. Changing figure's size can be helpful in a situation in which two classes overlap and rejection of the elements is more desired result than misclassification. Another thing worth noting when using minimum volume enclosing figures is that their size can often be artificially enlarged by having at least one point that is located very far away from all other points in the same class. This situation can lead to decrease in rejection option rates as well as bigger misclassification between native elements' classes. Of course decreasing figure's size can result in the opposite situation. The key to deal with this difficulty is to find balance between decreasing figure's size and still getting high rejection and classification rates. Two approaches are proposed in this paper.

Tolerance parameter manipulation

The tolerance parameter, also denoted as ε in Equation (2.2), can be used during point affiliation check-up to manipulate the result of the equation. By decreasing its value the figure "shrinks" evenly in all directions, and by increasing it more points are accepted as if they were lying inside the figure. In other words, by checking the distance of the point in regards to the minimum volume enclosing figure centre, the tolerance parameter can be used to "decrease" or "increase" the value resulting in acceptance or rejection.

Overall 100 tests there were performed with ε consecutive values being $(1, 0.98, 0.96, \ldots, -0.98)$. Some of the charts presented in this section have been cropped in order to present only important data and preserve paper space, e.g. Figure 3.7 contains information only for ε values from range $[-1, -0.56]$ because other values showed very poor classification and rejection rates thus there was no point in presenting them.

The results can be seen in Figure 3.7 and Figure 3.8. Classification value corresponds to the number of native elements from the tested class that were correctly classified. Identification informs about number of native elements that were correctly classified (identified) by the ellipsoid, not necessarily to the proper class. Rejection value informs about correctly rejected foreign patterns. It's worth noting that when value of ε becomes lesser than zero (which means that the figure shrinks in regards to its initial, unaltered size) there's a sudden drop in classification and identification rates. This could possibly be explained by the fact that many points lie on the surface of the minimum volume enclosing figure and get rejected after reducing its size. This is expected behaviour because minimum volume enclosing figures are somewhat unfolded on points they are trained on during creation process.

When testing hyper rectangles with gradually decreasing ε values the results are very weak as opposed to the ellipsoid ones. The main disadvantage of this solution is the sudden drop in identification and rejection rates when having values of ε lesser than zero.

Native elements removal

The main problem connected to manipulating the tolerance parameter is the fact that the shape of the figure stays the same for the whole time. This is not a desired solution in a situation in which some of the patterns are located far from the rest of native elements in the same class, which results in a creation of very big figure that is mostly empty inside. In such case it theoretically should be better to ignore such points and prefer smaller volume figure with a slightly worse classification capabilities. This approach has been tested by checking classification and rejection rates for ellipsoids and hyper rectangles built on increasingly smaller datasets. Each step of the shrinking figure algorithm consists of creating a figure for a certain number of patterns. Those elements are sorted based on the inclusion testing function value (for example Equation 2.2 or Algorithm 1), and 5 most distant ones are removed from the set. The classification and rejection rates are obtained based on the new figure created on this smaller set and the whole procedure is repeated. The results of this algorithm, which can be seen in Figure 3.9 and Figure 3.10, were obtained for 80 steps which resulted (in the 80th step) in 395 elements removed from the original data set while using ε value equal 0.

3.3.3. Summary

The tests performed on the classifier using array of identifiers prove that it can successfully combine classification and rejection tasks. While being unable to do the multi-class classification on their own, combined ellipsoids can be very accurate at classification and rejecting foreigners. Minimum volume enclosing ellipsoids combine advantages of commonly used classifiers described in Chapter 2 such as easy point inclusion detection, and iterative construction algorithm that uses tolerance parameter for its stop condition. The main disadvantage of ellipsoid, and minimum volume enclosing figures in general, is the fact that it does not transform the feature space of presented data. In order to get good results the data should be separable, and no generalisation of information is done at all. This is completely different from the attitude introduced in random forest or svm.

Moreover, according to the tests described and performed in previous Subsection regarding figure size manipulations, some initial conclusions can be drawn. At first glance it seems that neither approach based on elements removal, nor ε value manipulation showed results that would

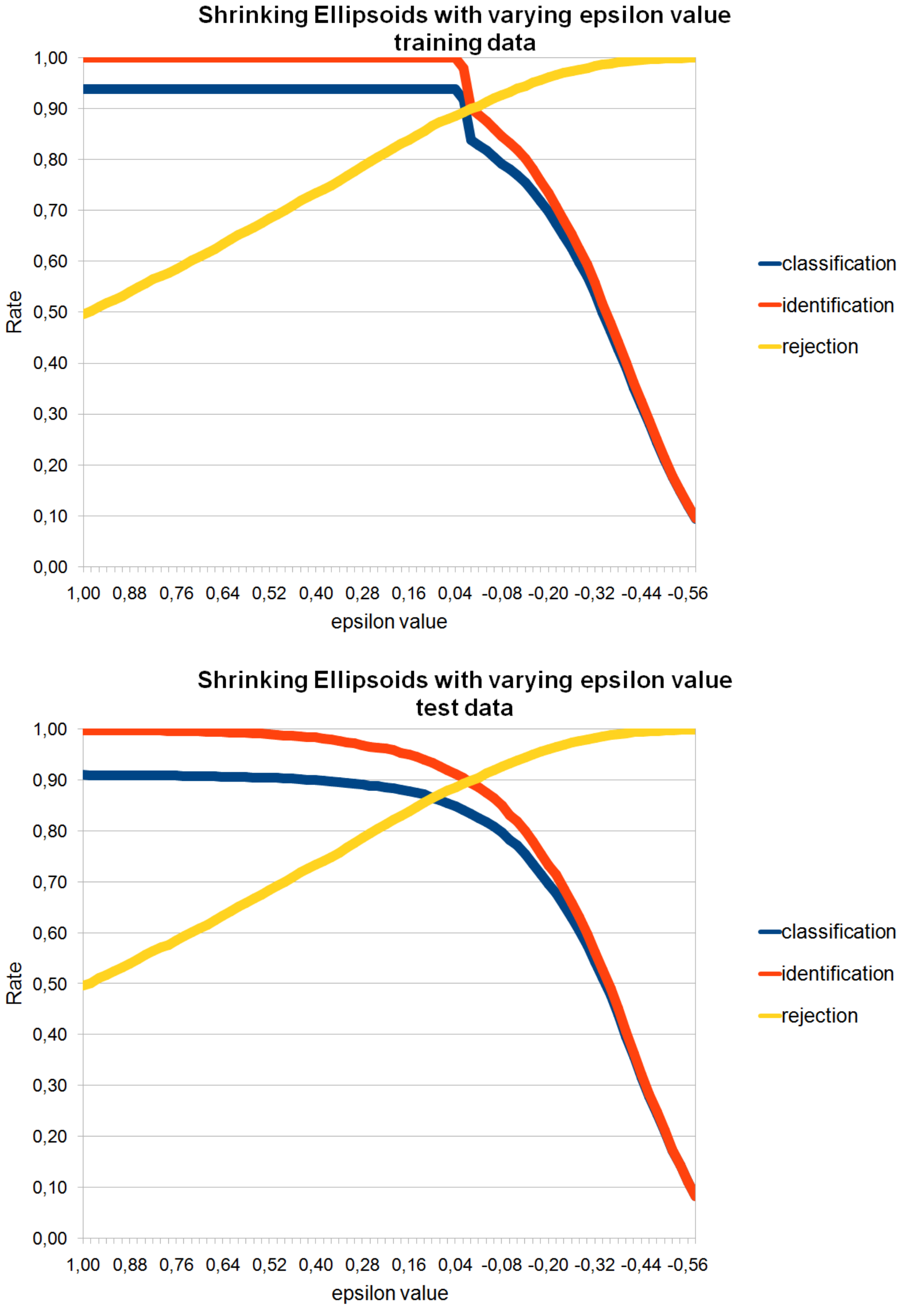


Figure 3.7: Classification, identification and rejection rates for different ε values for ellipsoids. Classification value corresponds to the number of native elements from the tested class that were correctly classified. Identification informs about number of native elements that were correctly classified (identified) by the ellipsoid, not necessarily to the proper class. Rejection value informs about correctly rejected foreign patterns. The bigger the values, the better.

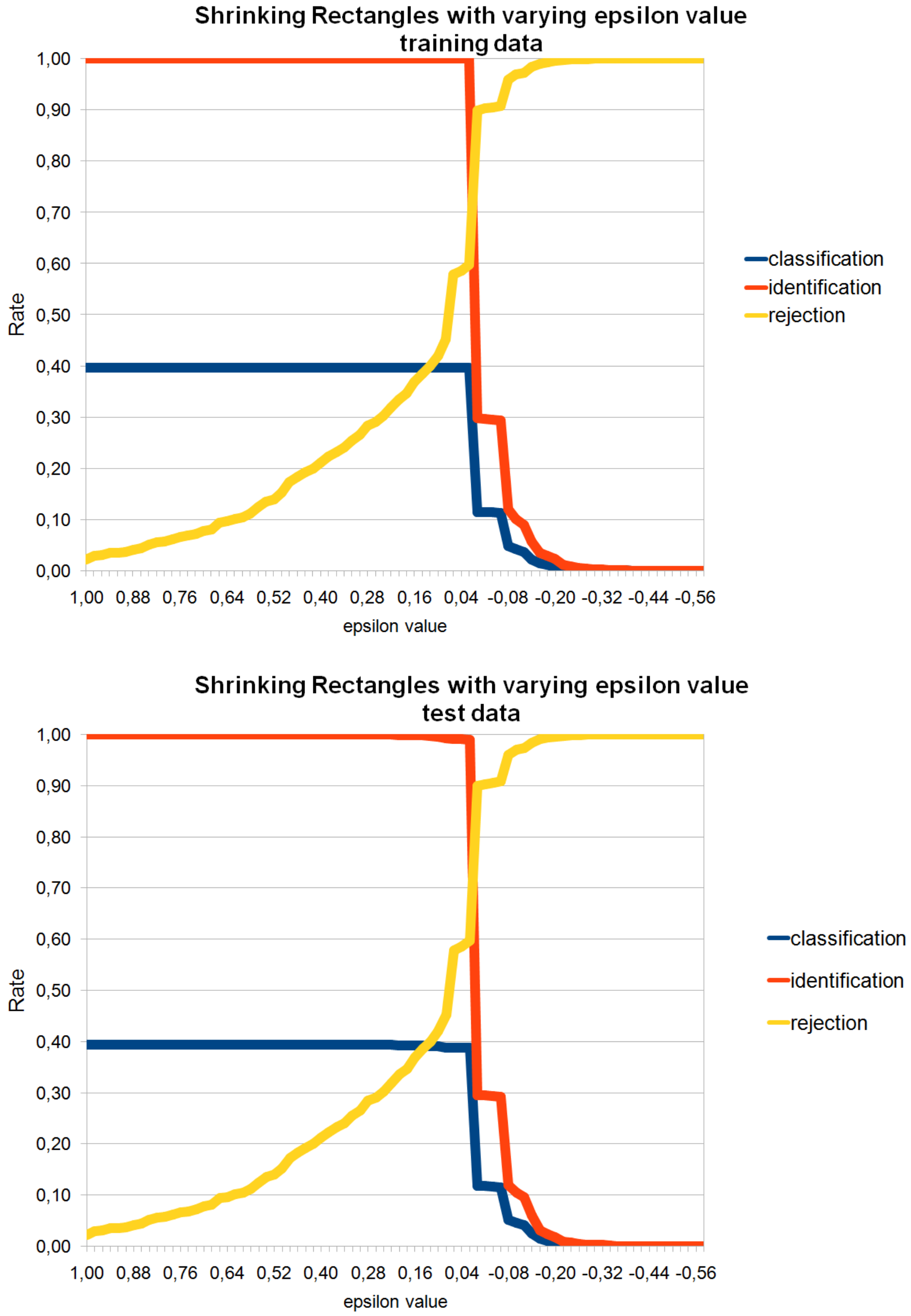


Figure 3.8: Classification, identification and rejection rates for different ε values for hyper rectangles. Classification value corresponds to the number of native elements from the tested class that were correctly classified. Identification informs about number of native elements that were correctly classified (identified) by the ellipsoid, not necessarily to the proper class. Rejection value informs about correctly rejected foreign patterns. The bigger the values, the better.

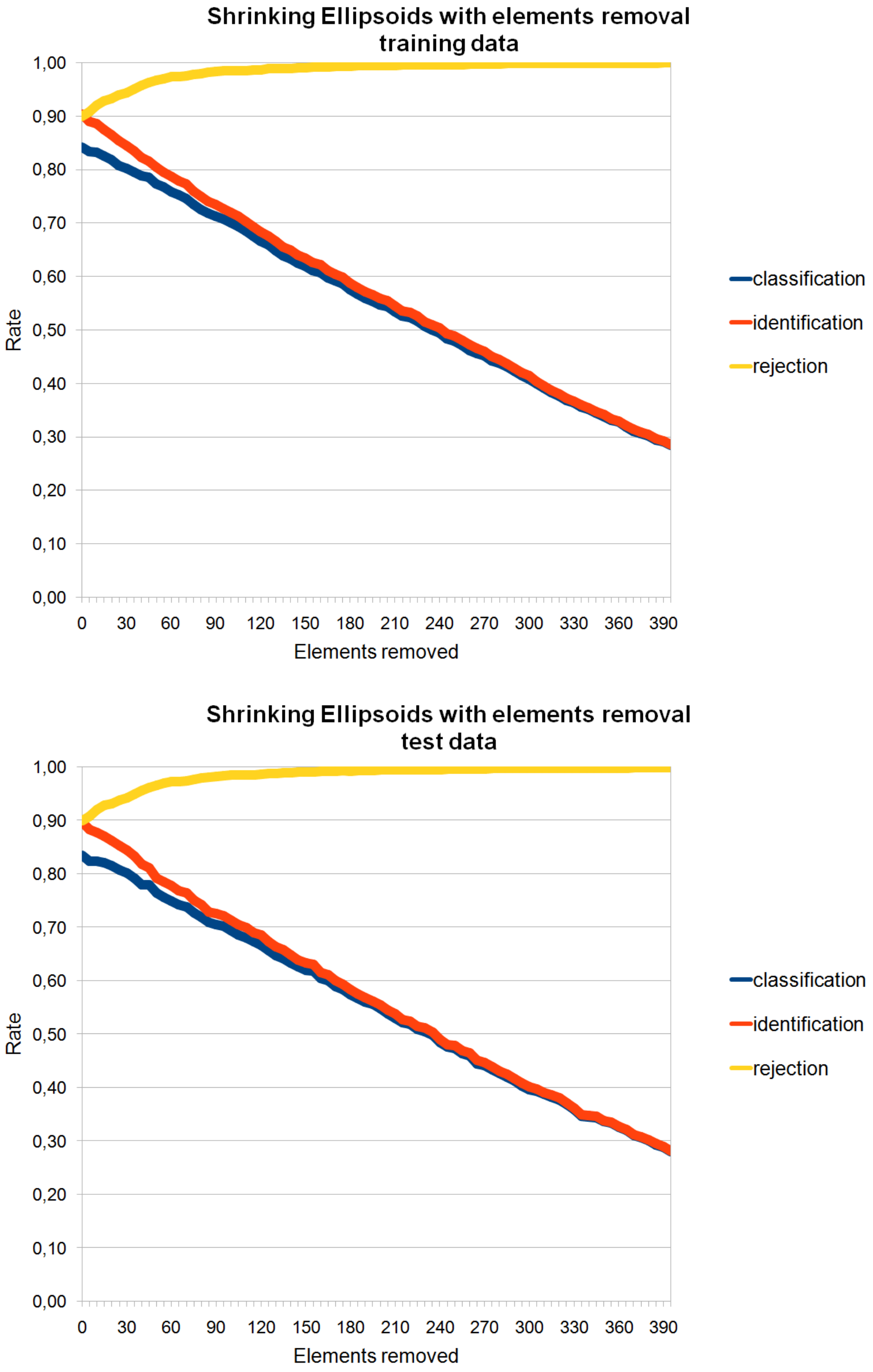


Figure 3.9: Classification, identification and rejection rates for each native element removal step while using ellipsoids. In each step 5 elements were deleted from the set resulting in overall 390 elements removed in the final 80th step. Classification value corresponds to the number of native elements from the tested class that were correctly classified. Identification informs about number of native elements that were correctly classified (identified) by the ellipsoid, not necessarily to the proper class. Rejection value informs about correctly rejected foreign patterns. The bigger the values, the better.

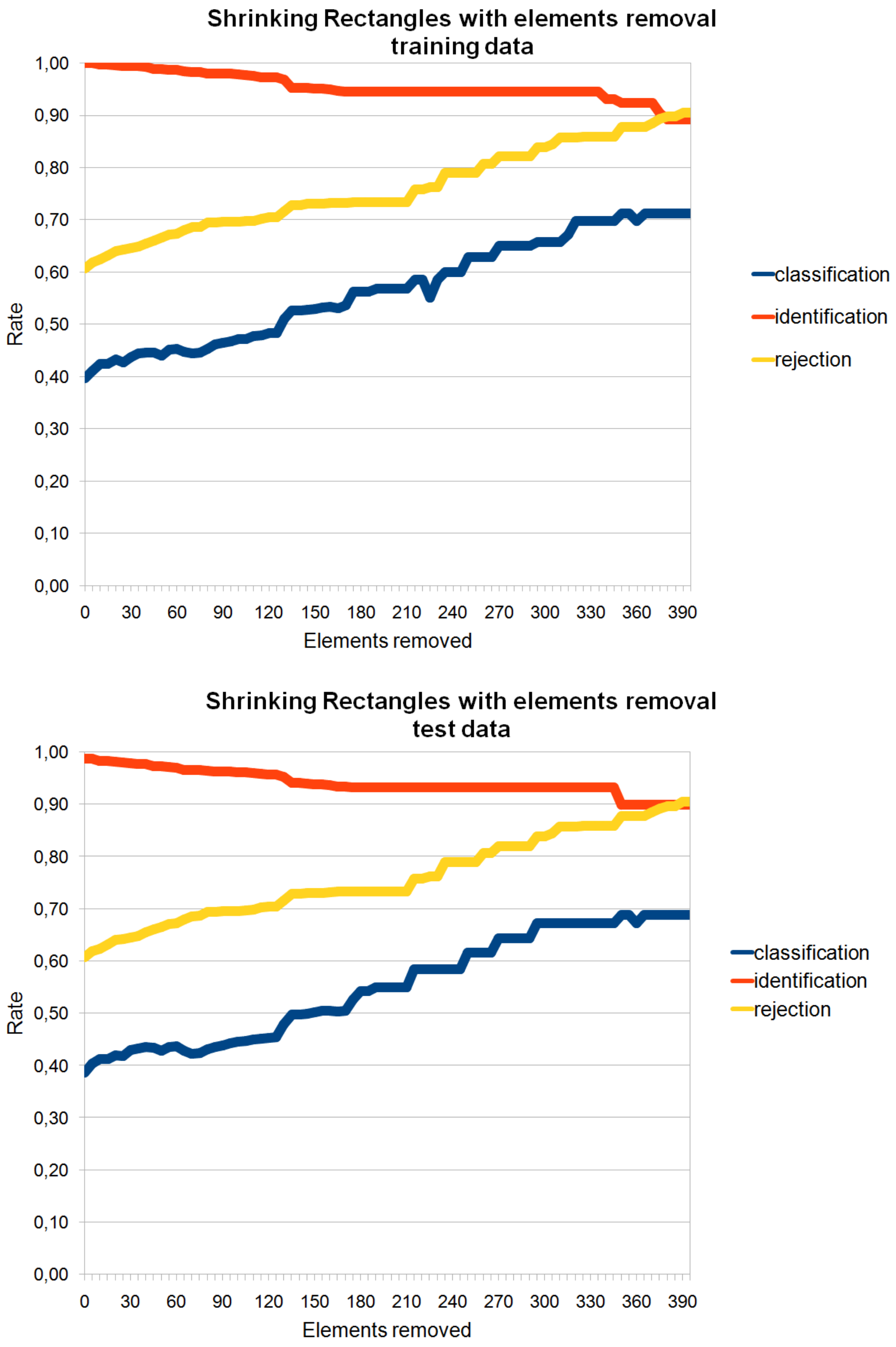


Figure 3.10: Classification, identification and rejection rates for each native element removal step while using hyper rectangles. In each step 5 elements were deleted from the set resulting in overall 390 elements removed in the final 80th step. Classification value corresponds to the number of native elements from the tested class that were correctly classified. Identification informs about number of native elements that were correctly classified (identified) by the ellipsoid, not necessarily to the proper class. Rejection value informs about correctly rejected foreign patterns. The bigger the values, the better.

justify any data changes, as all tested modifications achieved highest scores using minimum volume enclosing figures constructed on unaltered training set, with ε value set to 0. However it turns out that those conclusions are not necessarily justified as it is possible that some different datasets can actually benefit from size manipulations (see Section 4.6).

4. Experiments

This chapter contains the results of all tests that were performed on proposed classifier solutions. Results in form of quality measurements (see Section 4.1) for training, test and letters sets were gathered into two matrices with 12 rows and 10 columns, one for training and one for test data. Each row in the matrix corresponds to one of the quality evaluation measurements, and each column represents value of corresponding measurement scored by certain classifier tree using one of the common classifiers. See Table 4.1 for reference.

Table 4.1: Example empty result matrix

	Method 1			**Method 2**			**Method 3**		
	kNN	**SVM**	**RF**	**kNN**	**SVM**	**RF**	**kNN**	**SVM**	**RF**
Strict Acc.									
Fine Acc.									
Strict Native Sens.									
Acc.									
Native Prec.									
Native Sens.									
Native F-measure									
Foreign Prec.									
Foreign Sens.									
Foreign F-measure									

Every common classifier that was used by any of the newly introduced structure was tested with different parameters. SVM had its C, gamma and kernel options adjusted (see Chapter 2 for every parameter explanation). Values were as follows

$$C : [1, 2, 4, 8, 16]$$

$$gamma : [2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}]$$

$$kernel : [rbf, poly]$$

Adjustments for kNN were made for only one parameter, using euclidean metrics

$$n\_neighbors : [3, 5, 7, 10]$$

Random forests also had modifications applied to one parameter

$$n\_estimators : [30, 50, 100, 150]$$

4.1. Quality Evaluation

This paper contains some new approaches towards classification with rejection problem. In order to evaluate proposed solutions some quality measures must be introduced. Dealing with both native and foreign patterns forces inclusion of those measures that compare quality of both classification and rejection. Those values used in this paper are described below and in Table 4.2.

- *CC (Correctly Classified)* is the number of native patterns classified as native with a correct class label.
- *TP (True Positives)* is the number of native patterns classified as native (no matter, into which native class).
- *FN (False Negatives)* is the number of native patterns incorrectly classified as foreign.
- *FP (False Positives)* is the number of foreign patterns incorrectly classified as native.
- *TN (True Negatives)* is the number of foreign patterns correctly classified as foreign.

Based on these notions the following measures for model quality evaluation can be employed:

- *Strict Accuracy* is the absolute measure of the classifier's performance. It is the ratio of the number of all correctly classified patterns to their respective classes and rejected foreign ones to the number of all processed patterns.
- *Accuracy* is a characteristic derived from Strict Accuracy by ignoring the need to classify native patterns to their respective classes.
- *Native Precision* is the ratio of the number of not rejected native patterns to the number of all not rejected patterns,
- *Native Sensitivity* is the ratio of the number of not rejected native patterns to all native ones. This measure evaluates the ability of the classifier to identify native elements. The

higher the value of Native Sensitivity, the more effective identification of native elements. Unlike the Native Precision, this measure does not evaluate the effectiveness of separation between native and foreign elements.

- *Strict Sensitivity* takes only correctly classified native patterns and does not consider native patterns which are not rejected and assigned to incorrect classes.
- *Fine Sensitivity* is the ratio of the number of native patterns classified to correct classes to the number of all native patterns not rejected.
- *Foreign Precision* corresponds to Native Precision
- *Foreign Sensitivity* corresponds to Native Sensitivity
- *Foreign F-measure* is there to express the balance between precision and sensitivity since these two measures affect each other. Increasing sensitivity can cause a drop in precision since, along with correctly classified elements, there might be more incorrectly classified,

The equations for each measure can be seen in Table 4.2.

Table 4.2: Quality measures for classification with rejection.

Native Precision	=	$\frac{\text{TP}}{\text{TP+FP}}$	Accuracy	=	$\frac{\text{TP+TN}}{\text{TP+FN+FP+TN}}$
Foreign Precision	=	$\frac{\text{TN}}{\text{TN+FN}}$	Strict Accuracy	=	$\frac{\text{CC+TN}}{\text{TP+FN+FP+TN}}$
Native Sensitivity	=	$\frac{\text{TP}}{\text{TP+FN}}$	Fine Accuracy	=	$\frac{\text{CC}}{\text{TP}}$
Foreign Sensitivity	=	$\frac{\text{TN}}{\text{TN+FP}}$	Strict Native Sensitivity	=	$\frac{\text{CC}}{\text{TP+FN}}$

$$\text{F–measure} = 2 \cdot \frac{\text{Precision} \cdot \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}$$

4.2. Datasets

All quality measures described in Section 4.1 obtained and presented in this paper are calculated from classifiers' results for certain datasets. Those sets, referred to as native and foreign, are the result of applying feature-extraction function to images containing digits and letters. The original data of scanned digits comes from the well-known MNIST database[14], which comprises the image files of handwritten lower-case letters, which have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and

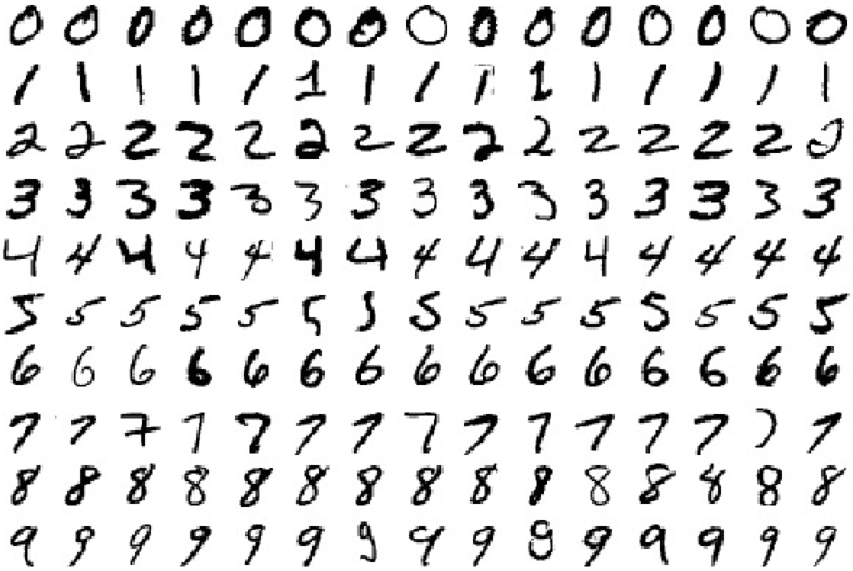


Figure 4.1: Visualization of scanned digits from MNIST database, image taken from [2]

pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. The scanned letters were provided by this thesis' supervisor and come from the scientific project NCN nr 2012/07/B/ST6/01501 [3].

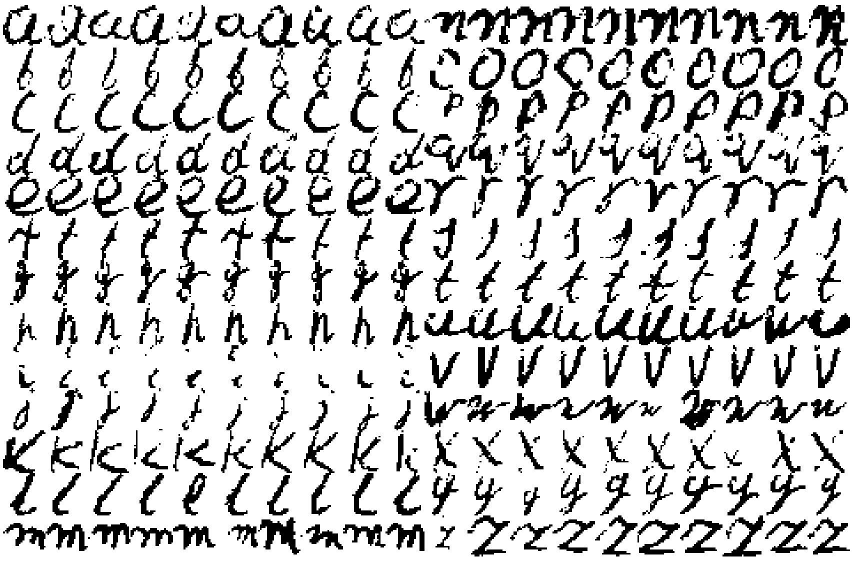


Figure 4.2: Visualization of scanned letters from [3] database.

The native set consists of 10,000 scanned digit images, with ten different classes, one for every digit (0 - 9) and approximately 1000 samples for each class. This set is further divided into training and test sets in 7:3 ratio. The foreign set consists of 26,000 images of scanned letters and is not divided internally because it is used only in rejection option evaluation. All patterns within those two sets have been size-normalized and centred in a fixed-size image. Every pattern within those two datasets consists of 24 unique features that were extracted to ensure best classification capabilities. Examples of features are: maximum/position of maximum values of projections, histograms of projections, transitions, offsets; raw moments, central moments, Euler numbers etc. All the features were already provided by the supervisor and come from the scientific project NCN nr 2012/07/B/ST6/01501 [3].

4.3. Classifier trees

Described in Chapter 3.1 classifier trees were tested with various common classifiers: SVM, kNN and random forest, using different parameters. Over 500 tests were held. When evaluating results quality evaluation measurements were taken into account (see Section 4.1). In the next few subsections there is short summary for each classifier tree using different internal classifiers. Because the results obtained for Slanting Tree with ordered classes were almost exactly the same as for regular Slanting Tree (with arbitrary class order) the results tables do not include scores achieved for the original Slanting Tree (without computed order of classes) to maintain their clarity.

Table 4.3: Measures values (described in Section 4.1) for classifier trees using various common classifiers on training data (described in Section 4.2)

	Balanced Tree			**Slanting Tree**			**Slanting Tree 2**		
	kNN	**SVM**	**RF**	**kNN**	**SVM**	**RF**	**kNN**	**SVM**	**RF**
Strict Acc.	20.67	44.92	41.47	22.76	29.70	50.29	38.34	41.51	40.59
Fine Acc.	92.66	99.33	100.00	90.61	91.79	92.43	94.08	95.41	95.94
Strict Native Sens.	92.64	99.09	100.00	90.00	91.73	92.43	93.06	95.26	95.79
Acc.	22.21	45.06	41.47	24.72	31.42	51.88	39.57	42.47	41.44
Native Prec.	21.23	27.60	26.38	21.70	23.41	30.35	25.62	26.69	26.35
Native Sens.	99.99	99.76	100.00	99.33	99.93	100.00	98.91	99.84	99.84
Native F-measure	35.02	43.23	41.74	35.62	37.93	46.57	40.70	42.13	41.69
Foreign Prec.	99.76	99.79	100.00	96.51	99.86	100.00	98.81	99.85	99.84
Foreign Sens.	1.58	30.55	25.94	4.92	13.25	39.11	23.82	27.25	25.94
Foreign F-measure	3.10	46.78	41.20	9.37	23.39	56.23	38.39	42.82	41.18

Results gathered in Table 4.3 and Table 4.4 prove that combining commonly used classifiers by putting them in more complex structures does not affect overall classification capabilities. Of course the quality of determining patterns' affiliations relies mostly on the type of the classifier used, but is also affected by classifier's parameters and the tree structure.

Table 4.4: Measures values (described in Section 4.1) for classifier trees using various common classifiers on test data (described in Section 4.2)

	Balanced Tree			**Slanting Tree**			**Slanting Tree 2**		
	kNN	**SVM**	**RF**	**kNN**	**SVM**	**RF**	**kNN**	**SVM**	**RF**
Strict Acc.	10.79	37.04	32.81	13.41	21.18	44.21	30.72	33.94	32.75
Fine Acc.	91.86	96.44	95.56	88.89	91.49	90.36	93.45	95.02	94.56
Strict Native Sens.	91.77	94.03	93.23	88.00	90.97	89.03	91.33	92.80	92.67
Acc.	11.62	37.39	33.26	14.53	22.05	45.18	31.37	34.44	33.30
Native Prec.	10.35	13.77	13.03	10.59	11.53	15.54	12.73	13.24	13.08
Native Sens.	99.90	97.50	97.57	99.00	99.43	98.53	97.73	97.67	98.00
Native F-measure	18.75	24.13	22.99	19.13	20.66	26.85	22.53	23.33	23.08
Foreign Prec.	99.28	99.08	98.94	97.74	99.52	99.58	98.93	99.04	99.13
Foreign Sens.	1.58	30.55	25.94	4.92	13.25	39.11	23.82	27.25	25.94
Foreign F-measure	3.10	46.70	41.11	9.38	23.38	56.16	38.40	42.74	41.12

Trees using SVM classifier yield better results when using radial basis function (rbf) kernel along with C parameter set to 16 and γ to 0.5. During calculations it was observed that the γ parameter didn't have as much impact on final results, unlike the C parameter which, when decreasing its value, lowered achieved scores. For Slanting Tree 2 the SVM classifier performed best when paired up with Random Forest which may indicate that both of those classifiers tend to misclassifying different patterns (hence better rejection option rates).

Using Random Forest classifier yields different scores depending on which tree structure is used. Whereas Balanced Tree performs best when using Random Forest with 30 estimators, both Slanting Tree and Slanting Tree 2 get better results while utilizing classifier with 100 estimators. Interesting may be the fact that the best performing Slanting Tree 2 uses Random Forest combined with SVM classifiers with the exactly same parameters' values as when using SVM classifier backed up by Random Forest one. In both cases results are very similar which proves that those classifiers tend to cooperate well.

Unfortunately, among all classifiers tested, the kNN one performs the worst when used in all of the three trees. While this doesn't mean that the classifications rates were very low, in fact the differences between scores achieved by using kNN and SVM or Random Forest were negligible, rejection option was almost non-existent. The best results, achieved by Slanting Tree 2, were obtained when using Random Forest as a second classifier. All results for kNN classifier for each of classifier tree described in this chapter, presented in the tables, were achieved when using *n_neighbors* parameter value of 10.

4.4. Classifier arrays

All the results presented in Tables 4.5 and 4.6 were obtained for kNN using n_neighbors parameter with value 10, SVM using rbf kernel, having C value of 8 and gamma 0.5 and random forest consisting of 100 estimators.

Table 4.5: Measures values (described in Section 4.1) for classifier hierarchy arrays using various common classifiers on training data (described in Section 4.2)

	1 vs. all			**1 vs. 1**			**1 vs. 1 (2)**		
	kNN	**SVM**	**RF**	**kNN**	**SVM**	**RF**	**kNN**	**SVM**	**RF**
Strict Accuracy	17.07	23.38	26.94	79.04	66.03	69.84	19.90	34.79	33.01
Fine Accuracy	81.32	91.23	91.34	-	99.70	100.00	94.94	98.50	100.00
Strict Native Sensitivity	81.32	91.17	91.34	0.00	9.58	9.33	94.94	98.46	100.00
Accuracy	20.99	25.22	28.75	79.04	66.04	69.84	20.96	35.10	33.01
Native Precision	20.97	21.88	22.73	-	11.82	14.92	20.96	24.41	23.83
Native Sensitivity	100.00	99.93	100.00	0.00	9.61	9.33	100.00	99.96	100.00
Native F-measure	34.66	35.90	37.04	-	10.60	11.48	34.66	39.23	38.49
Foreign Precision	100.00	99.65	100.00	79.04	77.16	78.13	-	99.94	100.00
Foreign Sensitivity	0.04	5.41	9.86	100.00	81.00	85.88	0.00	17.90	15.25
Foreign F-measure	0.08	10.26	17.95	88.29	79.04	81.82	-	30.36	26.46

The results obtained when using kNN classifier aren't very satisfying. All foreign elements were treated as native ones which means that rejection option is not present in this approach. Both SVM and random forest performed better, but not as good as the classifier trees described in Section 3.1. Whereas 1 vs. all approach lacked definitive rejection option, the 1 vs. 1 was too strict in rejecting presented patterns. Although the last method, denoted as modified 1 vs. 1 approach brought balance to classification-rejection problem it didn't manage to preserve high classification and rejection rates for both options.

Table 4.6: Measures values (described in Section 4.1) for classifier hierarchy arrays using various common classifiers on test data (described in Section 4.2)

	1 vs. all			1 vs. 1			1 vs. 1 (2)		
	kNN	**SVM**	**RF**	**kNN**	**SVM**	**RF**	**kNN**	**SVM**	**RF**
Strict Accuracy	8.28	14.09	17.88	89.78	73.72	78.11	9.47	25.93	23.40
Fine Accuracy	80.69	90.92	89.39	-	97.02	96.39	92.64	96.92	95.51
Strict Native Sensitivity	80.69	90.31	88.35	0.00	9.75	9.79	92.64	96.44	94.97
Accuracy	10.26	15.01	18.95	89.78	73.75	78.14	10.22	26.24	23.85
Native Precision	10.23	10.68	11.10	-	5.68	7.57	10.22	12.13	11.78
Native Sensitivity	100.00	99.33	98.83	0.00	10.05	10.15	100.00	99.50	99.43
Native F-measure	18.55	19.29	19.96	-	7.26	8.67	18.55	21.62	21.07
Foreign Precision	100.00	98.62	98.67	89.78	88.78	89.36	-	99.68	99.58
Foreign Sensitivity	0.04	5.41	9.86	100.00	81.00	85.88	0.00	17.90	15.25
Foreign F-measure	0.08	10.26	17.93	94.61	84.71	87.59	-	30.35	26.45

4.5. Geometrical classifiers

The tests for geometrical classifiers, ellipsoids and hyper-rectangles, were performed using those minimum volume figures without any size alterations (see Section 3.3.2 for reference). The results can be seen in Table 4.7 and Table 4.8.

Table 4.7: Results obtained for geometrical classifiers arrays using ellipsoids and hyper rectangles, tested on training set.

	Ellipsoid	**Hyper Rectangle**
Strict Accuracy	88.72	56.38
Fine Accuracy	93.15	34.55
Strict Native Sensitivity	84.16	34.13
Accuracy	90.01	69.94
Native Precision	70.41	41.00
Native Sensitivity	90.34	98.79
Native F-measure	79.14	57.95
Foreign Precision	97.23	99.49
Foreign Sensitivity	89.93	62.28
Foreign F-measure	93.43	76.61

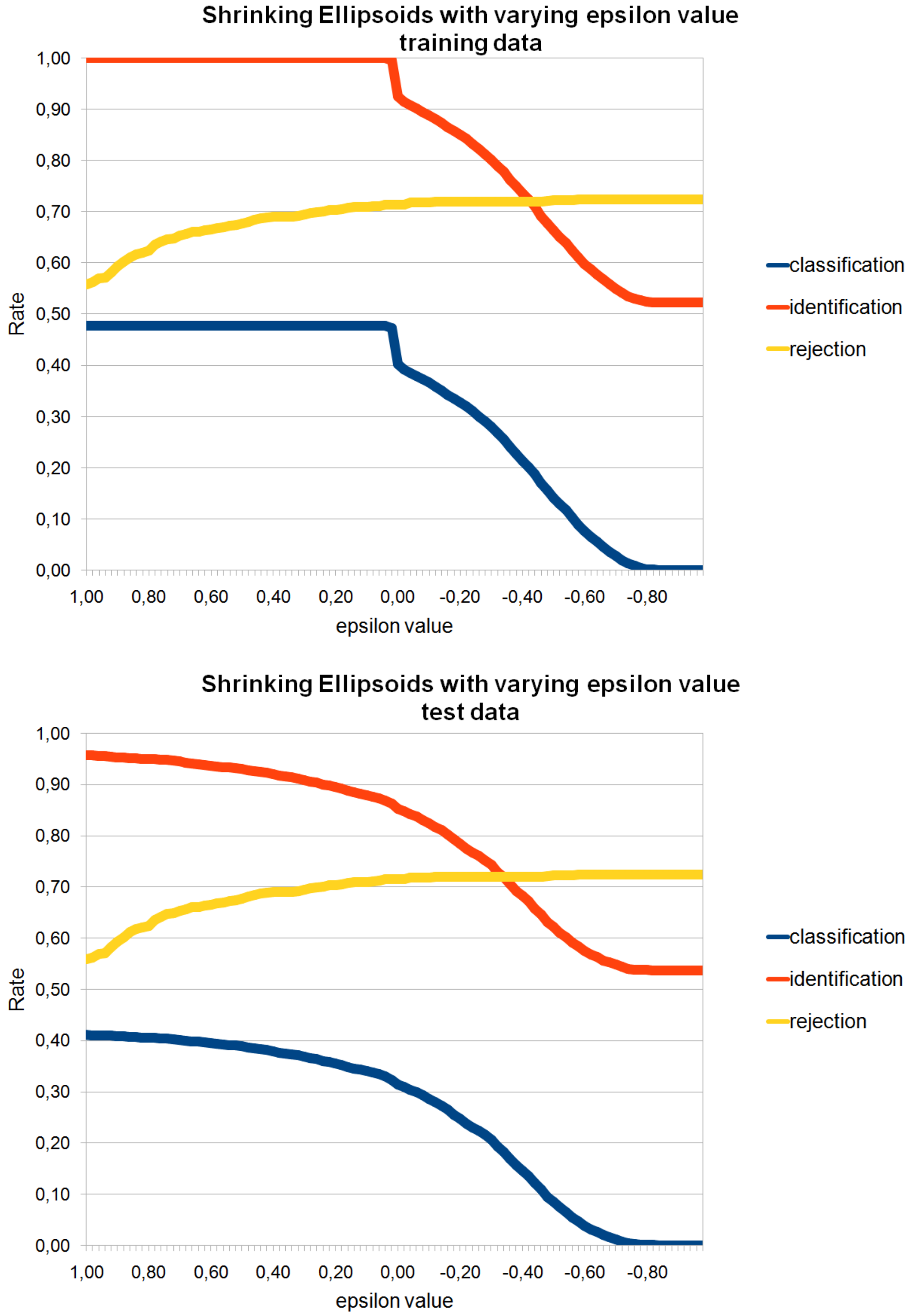
Table 4.8: Results obtained for geometrical classifiers arrays using ellipsoids and hyper rectangles, tested on test set.

	Ellipsoid	**Hyper Rectangle**
Strict Accuracy	89.27	59.32
Fine Accuracy	93.12	34.00
Strict Native Sensitivity	83.47	33.23
Accuracy	89.90	65.90
Native Precision	50.29	22.76
Native Sensitivity	89.90	97.73
Native F-measure	64.43	36.92
Foreign Precision	98.71	99.59
Foreign Sensitivity	89.93	62.28
Foreign F-measure	94.11	76.64

Although it may seem that the ellipsoids are superior to other classifiers presented and used in this paper it is worth noting that the tests were performed on only one dataset. Geometric classifiers that work on raw data, without introducing any changes to feature values given during training don't work well in situations when class elements overlap. SVM, random forest, etc. which should be used in such cases have the advantage over simple classifiers like kNN or arrays of minimum volume enclosing figures, because they change the way features are interpreted. For example, SVM increases dimensionality by using kernel function to introduce such feature-space representation in which two different classes are easily separated by a hyperplane. As for hyper rectangles, they don't perform as good as ellipsoids but have one very important advantage over them - speed of computation. Hyper rectangles are the fastest solution out of all methods mentioned and described in this paper. What is more, after some tinkering with their size, one can achieve very reasonable classification and rejection capabilities (see Section 3.3.2 for reference). For sure minimum volume enclosing ellipsoids are worth using in situations where data are easily separable.

4.6. Additional tests

In order to verify results regarding minimum volume figure size alterations, performed and described in Section 3.3, the tests for methods using native elements removal and ε value modification were additionally done on two new datasets. Those sets consist of 20 native classes and 1 foreign class, and represent musical notes. Whereas the first one contains unmodified raw data, the second one was created by standardizing those values. Contrary to what was stated in Section 3.3.2 the charts obtained for those two new sets show that size alterations may actually be a viable choice. For example at first look at the charts shown in Figure 4.3 it may seem like the best efficiency of the ellipsoids is achieved for ε equal to 0. For training data, when setting ε value to 0 classification, rejection and identification accuracy rates are above 90%. However after looking at chart plotted for test data it is clearly visible that when using $\varepsilon = 0$ the identification and rejection rates drop to around 80%. In this example it would probably be wiser to use $\varepsilon = 0.50$ in order to get all three score rates to 90% accuracy level. Of course the results differ greatly depending on the type of the figure and data sets used. It is thus advisable to manually perform tests before applying minimum volume figures solutions to real-world problems in order to maintain highest rates possible.



Shrinking Ellipsoids with varying epsilon value
training data

Shrinking Ellipsoids with varying epsilon value
test data

Figure 4.3: Identification, Classification and Rejection rates obtained for different ε values, tested on training and test data consisting of 20 native classes and 1 foreign one, representing musical notes. Minimum volume enclosing figure is ellipsoid.

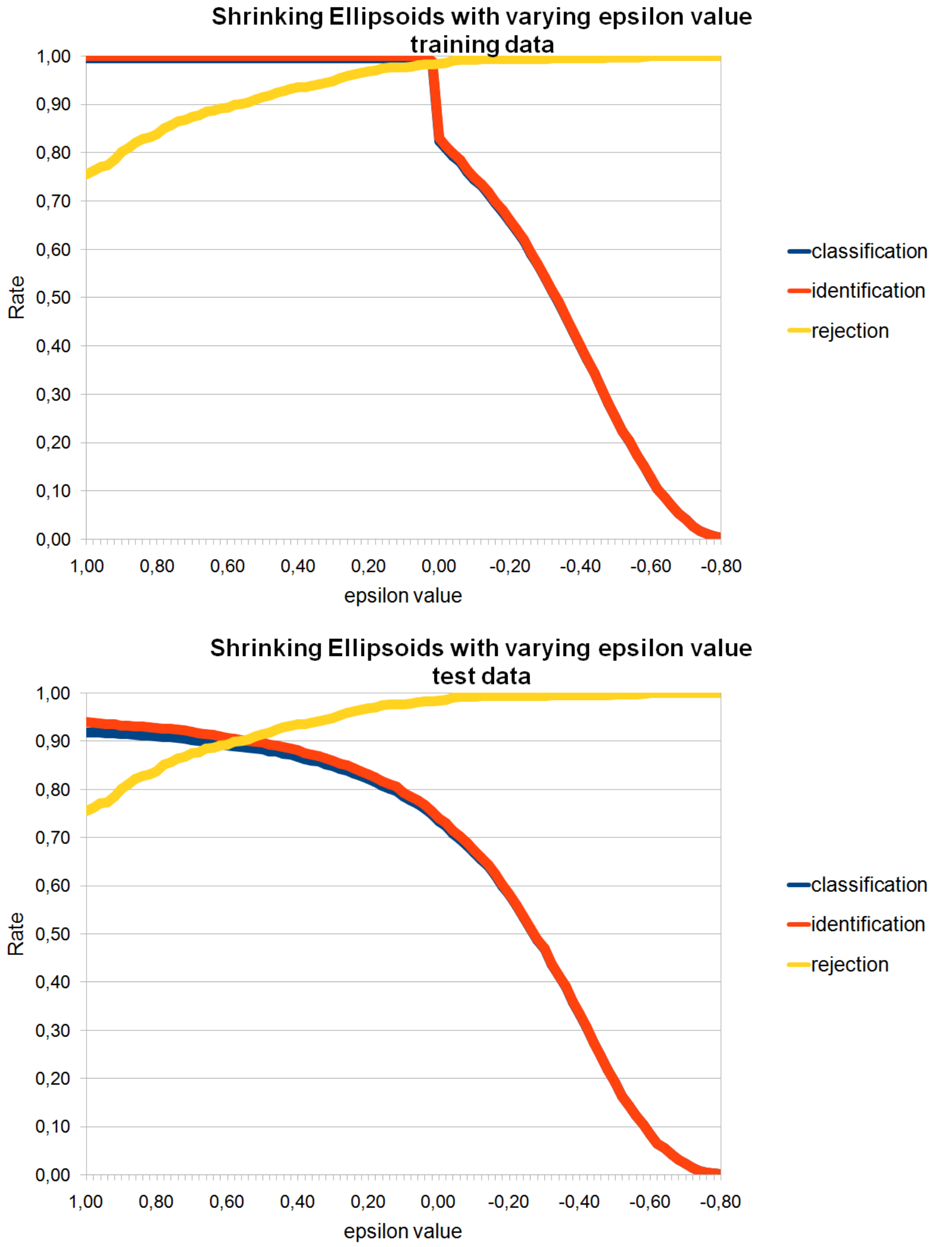


Figure 4.4: Identification, Classification and Rejection rates obtained for different ε values, tested on training and test data consisting of 20 native classes and 1 foreign one, representing musical notes. Each feature from the feature vector has been standardized. Minimum volume enclosing figure is ellipsoid.

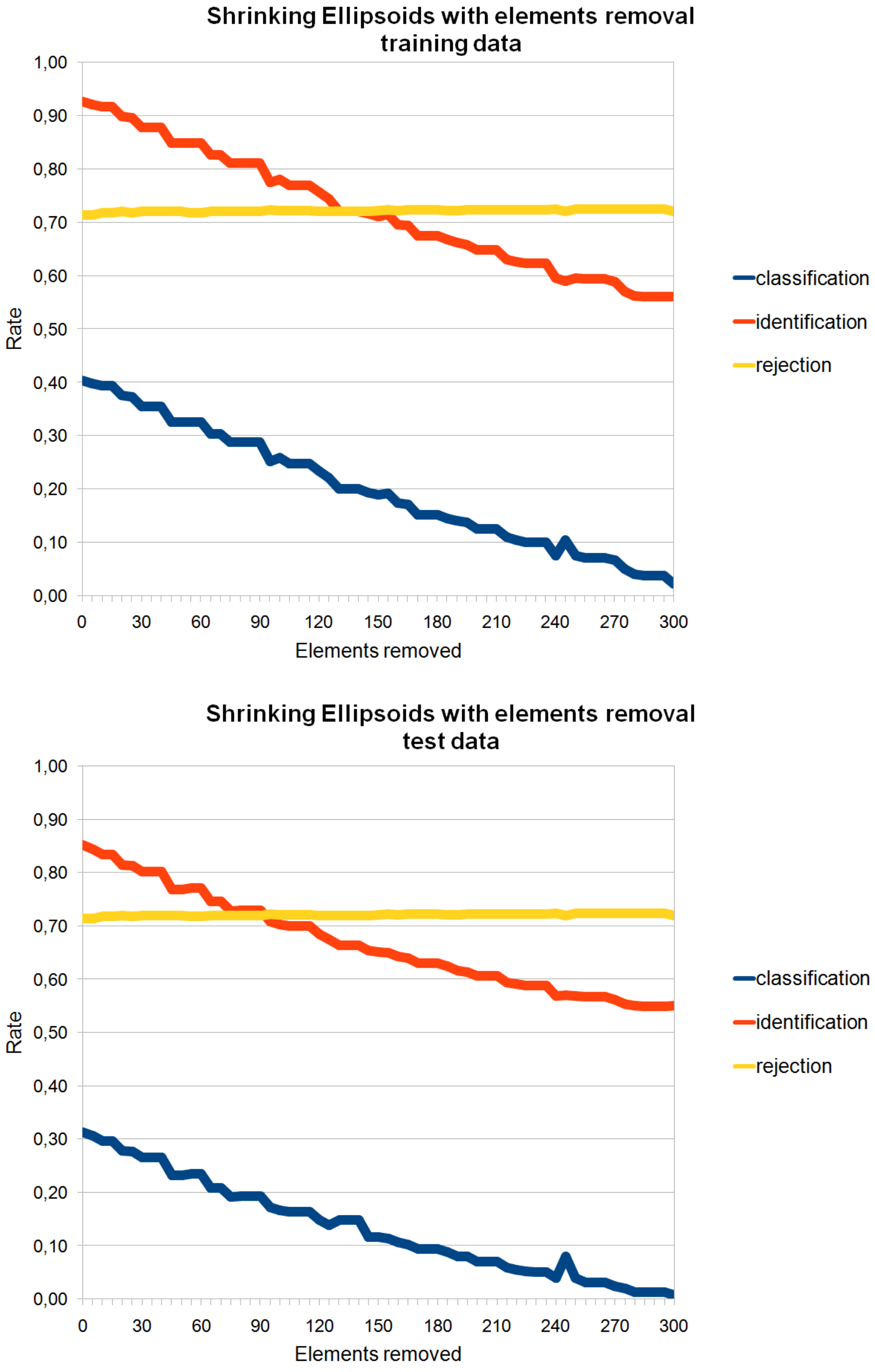
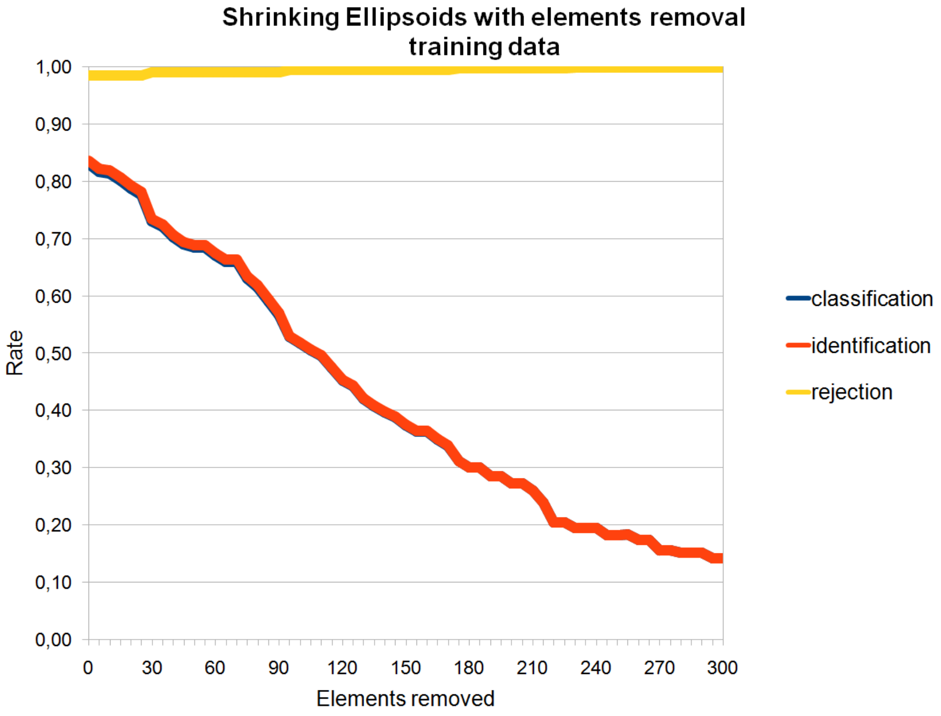


Figure 4.5: Identification, Classification and Rejection rates obtained for increasingly smaller data sets, tested on training and test data consisting of 20 native classes and 1 foreign one, representing musical notes. Minimum volume enclosing figure is ellipsoid.



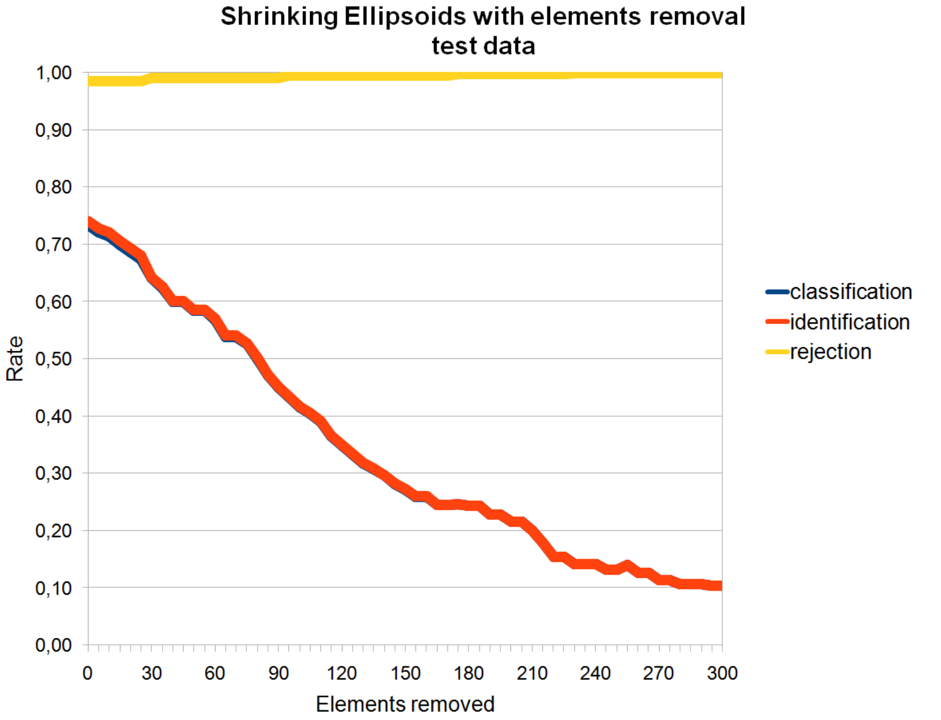


Figure 4.6: Identification, Classification and Rejection rates obtained for increasingly smaller data sets, tested on training and test data consisting of 20 native classes and 1 foreign one, representing musical notes. Each feature from the feature vector has been standardized. Minimum volume enclosing figure is ellipsoid.

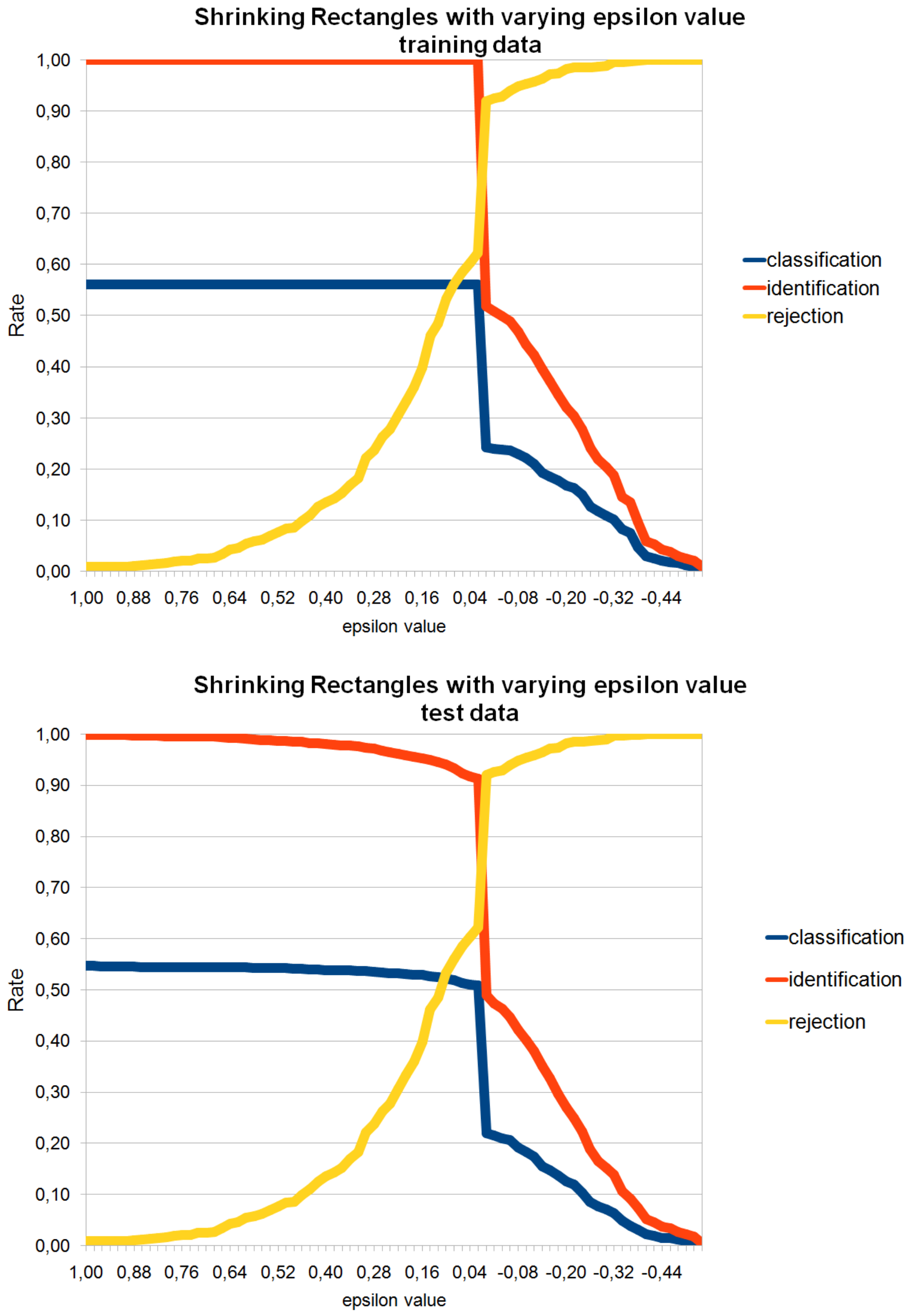
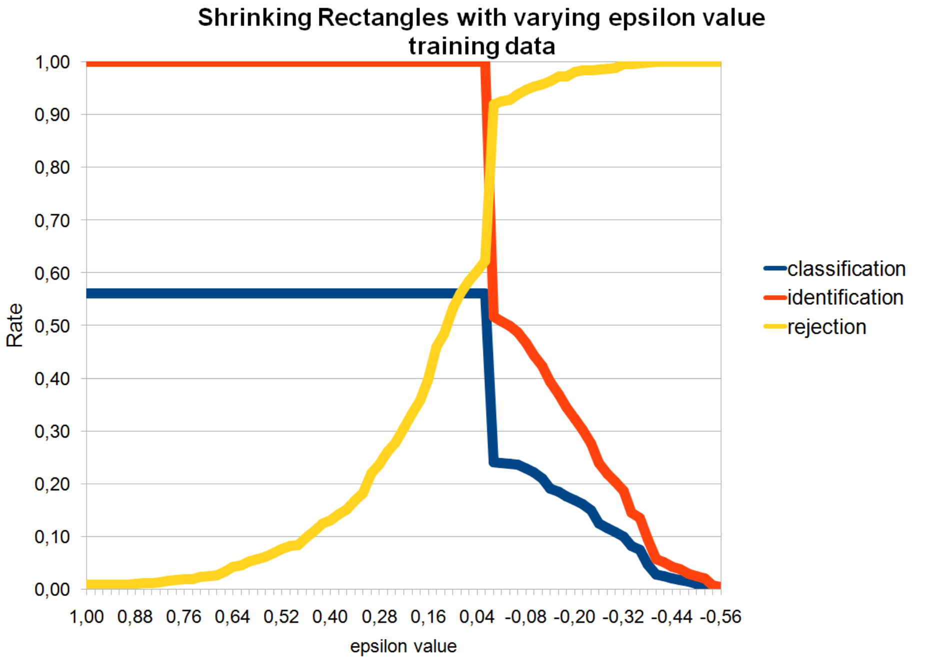


Figure 4.7: Identification, Classification and Rejection rates obtained for different ε values, tested on training and test data consisting of 20 native classes and 1 foreign one, representing musical notes. Minimum volume enclosing figure is hyper rectangle.



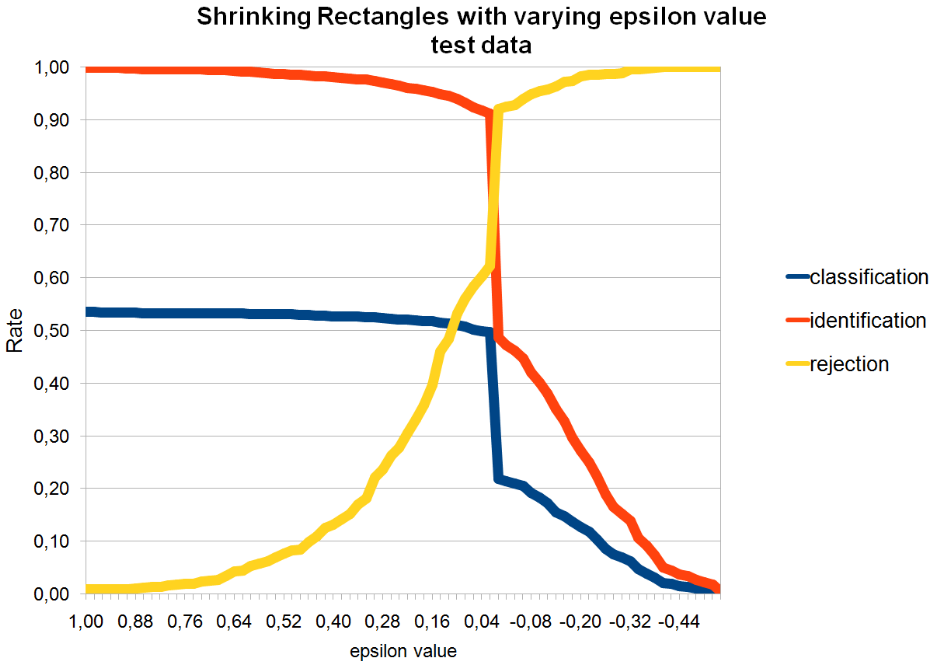
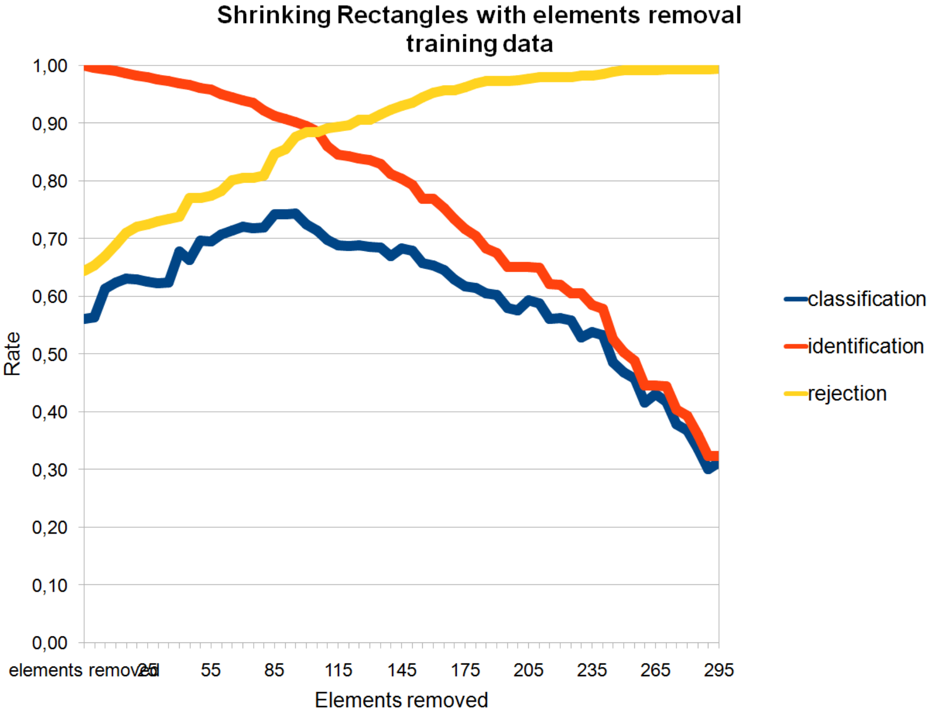


Figure 4.8: Identification, Classification and Rejection rates obtained for different ε values, tested on training and test data consisting of 20 native classes and 1 foreign one, representing musical notes. Each feature from the feature vector has been standardized. Minimum volume enclosing figure is hyper rectangle.



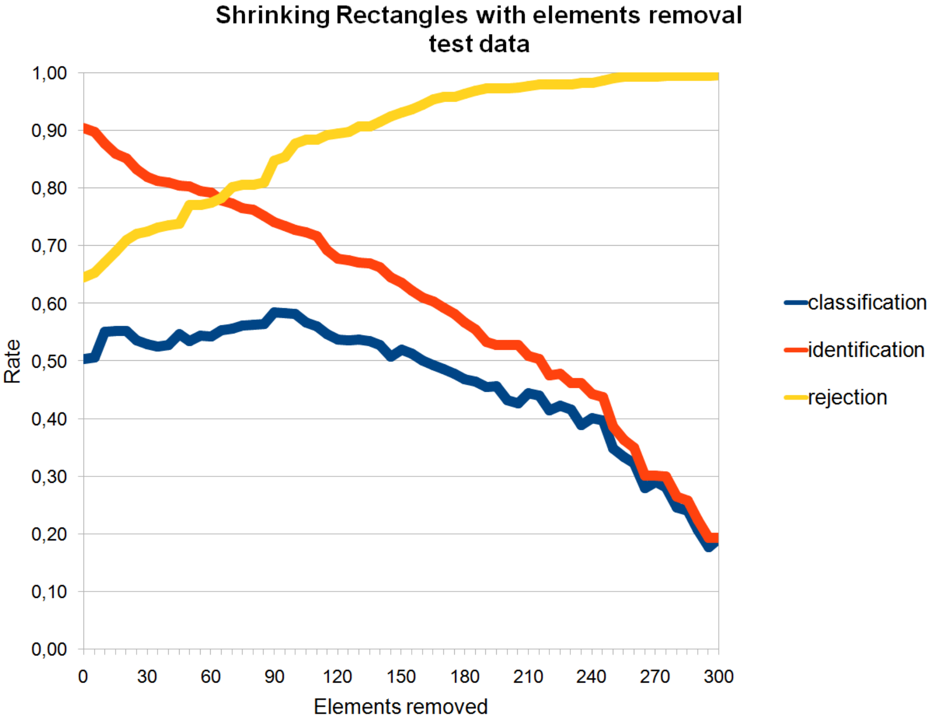
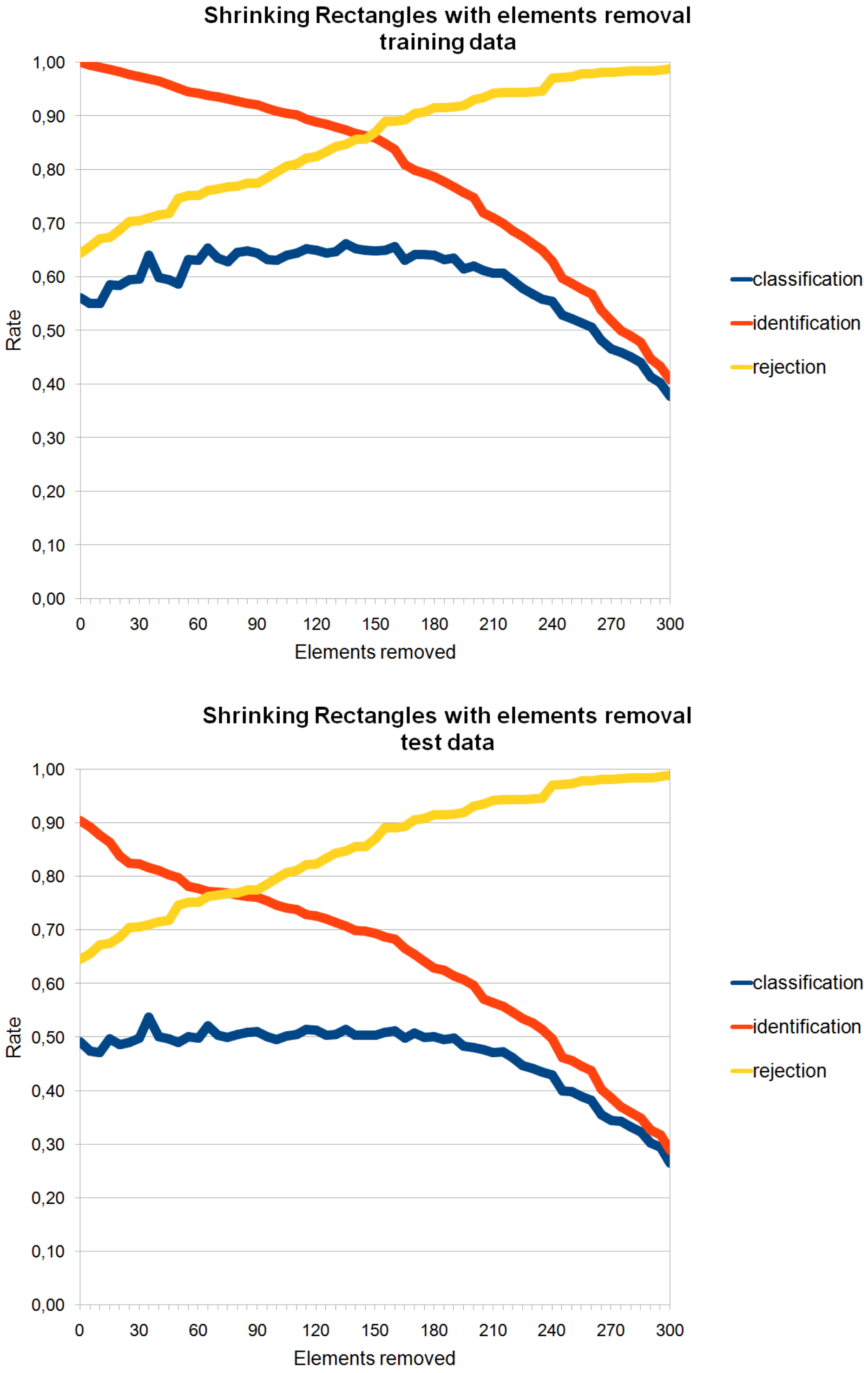


Figure 4.9: Identification, Classification and Rejection rates obtained for increasingly smaller data sets, tested on training and test data consisting of 20 native classes and 1 foreign one, representing musical notes. Minimum volume enclosing figure is hyper rectangle.


Shrinking Rectangles with elements removal training data

Shrinking Rectangles with elements removal test data

Figure 4.10: Identification, Classification and Rejection rates obtained for increasingly smaller data sets, tested on training and test data consisting of 20 native classes and 1 foreign one, representing musical notes. Each feature from the feature vector has been standardized. Minimum volume enclosing figure is hyper rectangle.

5. Summary

The problem of classification in conjunction with foreign pattern rejection is not an easy task. In this paper, while trying to propose a solution that takes advantage of commonly known and widely used classifiers that is capable of detecting outliers, it was noticed that most of the tested solutions lack the desired balance between rejection and classification capabilities. Whereas classifier trees, introduced and described in Section 3.1, maintain classification rates of the classifiers they are using internally in their nodes, the rejection rates often do not exceed 40%. The problem still persists when using classifier arrays, described in details in Section 3.2, where proposed solutions have either high classification rates with non-existent rejection option, or the opposite situation occurs. The one additional drawback that is present in all structures described in those two sections is the fact that they consume much computation time while running. On the other hand using arrays of minimum volume enclosing figures that were introduced in Section 3.3 brings desired revolution to previous results. Not only the creation of such figures is fast and straightforward, but also computations requiring usage of those figures take very little computer time and memory. The results obtained while using minimum volume figures are very good as long as the data sent for classification is well separable. The tests performed on two different data sets confirmed the efficiency of those geometrical classifiers. Additional tests that were performed in order to study correlation between figure size and its capabilities showed that the size alterations may be a viable option to increase structure's efficiency. Unfortunately such tests should be done manually, by studying the characteristics obtained by applying different size-altering methods to figures, for each new data set.

Although the minimum volume enclosing figures performed exceptionally well during tests described in this paper it should be noted that they don't have generalization capabilities, unlike svm or random forest classifiers. This drawback prevents from using minimum volume figures in situations where classes heavily overlap and are hard to separate without use of complex functions, as it is done in svm classifiers by applying different kernels. The final conclusion, summarizing the whole paper, should emphasize the importance of trying out simple solutions before using complex ones, as often what seems like a naive approach can lead to outstanding results. Although the minimum volume enclosing figures may not perform so well for all classification problems they are completely viable choice when it comes down to time and memory complexity. The author of this paper strongly advises everyone to try using them before escaping to more complex solutions.

Bibliography

[1] David Cournapeau. Scikit-learn website, 2007. URL `http://scikit-learn.org`.

[2] Kuan Hoong. Kuan hoong blog, 2016. URL `https://kuanhoong.wordpress.com/2016/02/01/r-and-deep-learning-cnn-for-handwritten-digits-recognition/`.

[3] Władysław Homenda. *Zagadnienie odrzucania w problemie rozpoznawania wzorca: koncepcje, metody, analizy.* Projekt badawczy NCN nr 2012/07/B/ST6/01501 realizowy w Instytucie Badań Systemowych Polskiej Akademii Nauk w Warszawie, 2013-2016, 2013-2016.

[4] Christopher M. Bishop. *Pattern recognition and machine learning.* 2006.

[5] Pedregos F. *Scikit-learn: Machine learning in Python.* Journal of Machine Learning Research, 2011.

[6] Altman N. S. *An introduction to kernel and nearest-neighbor nonparametric regression.* The American Statistician 46 (3), 1992.

[7] Vapnik V. Cortes, C. *Support-vector networks.* Machine Learning 20 (3), 1995.

[8] M. Chupin S. Lehéricy D. Dormont H. Benali Y. Samson R. Cuingnet, C. Rosso and O. Colliot. *Spatial regularization of SVM for the detection of diffusion alterations associated with stroke outcome.* Medical Image Analysis 15: 729-737, 2011.

[9] Christos Davatzikos Bilwaj Gaonkar. Analytic estimation of statistical significance maps for support vector machine based multi-variate image analysis and classification, 2013. URL `http://www.sciencedirect.com/science/article/pii/S1053811913003169`.

[10] R. Opitz, D.; Maclin. *Popular ensemble methods: An empirical study.* Journal of Artificial Intelligence Research. 11: 169–198, 1999.

[11] L. Breiman. *Random Forests.* Machine Learning 45 (1), 2001.

[12] R. Quinlan. *Learning efficient classification procedures, Machine Learning: an artificial intelligence approach.* 1986.

[13] Yildirim E. A. Todd, M. J. *On Khachiyan's Algorithm for the Computation of Minimum Volume Enclosing Ellipsoids*. 2005. URL `http://people.orie.cornell.edu/miketodd/TYKhach.pdf`.

[14] Christopher J.C. Burges Yann LeCun, Corinna Corte. The mnist database. URL `http://yann.lecun.com/exdb/mnist/`.