WARSAW UNIVERSITY OF TECHNOLOGY

Rejection Option in Pattern Recognition Problem - Selected Issues

by

Piotr Waszkiewicz

A thesis submitted in partial fulfillment for the degree of Master of Computer Science

in the
Faculty of Mathematics and Information Science

March 2017

Declaration of Authorship

I, Piotr Waszkiewicz, declare that this thesis titled, 'Rejection Option in Pattern Recognition Problem - Selected Issues' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____

Date: ____

WARSAW UNIVERSITY OF TECHNOLOGY

Abstract

Faculty of Mathematics and Information Science

Master of Computer Science

by Piotr Waszkiewicz

An analysis of the presented study seeks 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. tumor 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.

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus at pulvinar nisi. Phasellus hendrerit, diam placerat interdum iaculis, mauris justo cursus risus, in viverra purus eros at ligula. Ut metus justo, consequat a tristique posuere, laoreet nec nibh. Etiam et scelerisque mauris. Phasellus vel massa magna. Ut non neque id tortor pharetra bibendum vitae sit amet nisi. Duis nec quam quam, sed euismod justo. Pellentesque eu tellus vitae ante tempus malesuada. Nunc accumsan, quam in congue consequat, lectus lectus dapibus erat, id aliquet urna neque at massa. Nulla facilisi. Morbi ullamcorper eleifend posuere. Donec libero leo, faucibus nec bibendum at, mattis et urna. Proin consectetur, nunc ut imperdiet lobortis, magna neque tincidunt lectus, id iaculis nisi justo id nibh. Pellentesque vel sem in erat vulputate faucibus molestie ut lorem.

1.1 A Section

Quisque tristique urna in lorem laoreet at laoreet quam congue. Donec dolor turpis, blandit non imperdiet aliquet, blandit et felis. In lorem nisi, pretium sit amet vestibulum sed, tempus et sem. Proin non ante turpis. Nulla imperdiet fringilla convallis. Vivamus vel bibendum nisl. Pellentesque justo lectus, molestie vel luctus sed, lobortis in libero. Nulla facilisi. Aliquam erat volutpat. Suspendisse vitae nunc nunc. Sed aliquet est suscipit sapien rhoncus non adipiscing nibh consequat. Aliquam metus urna, faucibus eu vulputate non, luctus eu justo.

1.1.1 A Subsection

Donec urna leo, vulputate vitae porta eu, vehicula blandit libero. Phasellus eget massa et leo condimentum mollis. Nullam molestie, justo at pellentesque vulputate, sapien velit ornare diam, nec gravida lacus augue non diam. Integer mattis lacus id libero ultrices sit amet mollis neque molestie. Integer ut leo eget mi volutpat congue. Vivamus sodales, turpis id venenatis placerat, tellus purus adipiscing magna, eu aliquam nibh dolor id nibh. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Sed cursus convallis quam nec vehicula. Sed vulputate neque eget odio fringilla ac sodales urna feugiat.

1.2 Another Section

Phasellus nisi quam, volutpat non ullamcorper eget, congue fringilla leo. Cras et erat et nibh placerat commodo id ornare est. Nulla facilisi. Aenean pulvinar scelerisque eros eget interdum. Nunc pulvinar magna ut felis varius in hendrerit dolor accumsan. Nunc pellentesque magna quis magna bibendum non laoreet erat tincidunt. Nulla facilisi.

Duis eget massa sem, gravida interdum ipsum. Nulla nunc nisl, hendrerit sit amet commodo vel, varius id tellus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc ac dolor est. Suspendisse ultrices tincidunt metus eget accumsan. Nullam facilisis, justo vitae convallis sollicitudin, eros augue malesuada metus, nec sagittis diam nibh ut sapien. Duis blandit lectus vitae lorem aliquam nec euismod nisi volutpat. Vestibulum ornare dictum tortor, at faucibus justo tempor non. Nulla facilisi. Cras non massa nunc, eget euismod purus. Nunc metus ipsum, euismod a consectetur vel, hendrerit nec nunc.

Chapter 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. We expect 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. In such situation a proper mechanism for rejecting garbage patterns could be used to provide correct results.

2.1 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, [**?**]. SVMs are effective in high-dimensional spaces, memory efficient, and quite versatile with many kernel functions that can be specified for the decision function. Although in some cases, where the number of features is much greater than the number of samples, this method can give poor results, and is not cost-efficient when calculating probability estimates.

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, [**?**]. 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.

2.3 RF

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". In the random forest algorithm [**?**] the weak learners are decision trees, which are used to predict class labels. 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, an input data is entered at the top of the tree and as it traverses down the tree the data gets bucketed into smaller and smaller subsets. In the random forest we form a large number of classification trees, which altogether serve as a classifier. In order to grow each tree, we draw with replacement a random selection of rows from the training set. Random sampling with replacement is also called bootstrap sampling. In addition, when constructing trees for a random forest at each node we select randomly m variables out of the set of all input variables, 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. 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

Chapter 3

Quality Evaluation

In order to evaluate the quality of the proposed methods we use a set of measures. Below we list basic notions applied in the formulas for those measures, while measures themselves are placed in Table 3.1.

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

TABLE 3.1: Quality measures for classification with rejection.

$$\text{Native Precision} = \frac{\text{TP}}{\text{TP+FP}} \qquad \text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+FN+FP+TN}}$$

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

$$\text{Native Sensitivity} = \frac{\text{TP}}{\text{TP+FN}} \qquad \text{Fine Accuracy} = \frac{\text{CC}}{\text{TP}}$$

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

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

Chapter 4

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.

4.1 Balanced Tree

4.1.1 Description

Balanced Tree structure utilises clustering algorithms for its creation. It is usually shaped as a balanced binary tree, thus the name, with classifier in each of its nodes. Each node represents a set of classes that are currently taken into consideration as native ones for provided pattern. By traversing down the tree certain classes get rejected and the pattern is moved forward to the next node that represents only remaining classes. Decision as to which classes should be put in each of the child nodes is made by clustering algorithm that divides set of remaining classes from parent node into two and assigns each part for each child node. If there's only one remaining class, tree leaf is created instead. Each node, except for leafs, contains binary classifier trained on data that is based on clustering algorithm results. What it mean is that patterns from training data set, that belong to classes dedicated for left child node, are joined together and are treated as one big class '0'. The same goes for patterns that belong to the right child node, except for the class number which is '1'. By having two, new data sets that represent two different classes, the parent node can finally create binary classifier. During classification procedure, after receiving new, unknown pattern each node uses its classifier to assign either '0' or '1' label to this pattern, which is then used to send it to left or right child accordingly for further classification. Balanced Tree leafs also utilize their classifiers but those are created in a different way. Because of the fact that each leaf represents one class, and has no children there's no way to create data set for classifier using algorithm for non-leaf node. To overcome this problem each leaf is treated as a node with left child representing the same class as the leaf, and the right child representing all remaining classes. That way classifier is trained on two-class data and can be viewed as 'one-vs-rest' classifier. When it comes to classifying new, unknown pattern leaf uses its classifier to determine pattern's label. In case of '0' (meaning it should be sent to left child) the pattern is treated as an element from class represented by leaf. If the resulting label is '1' the pattern gets rejected and treated as a foreign one.

4.1.2 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

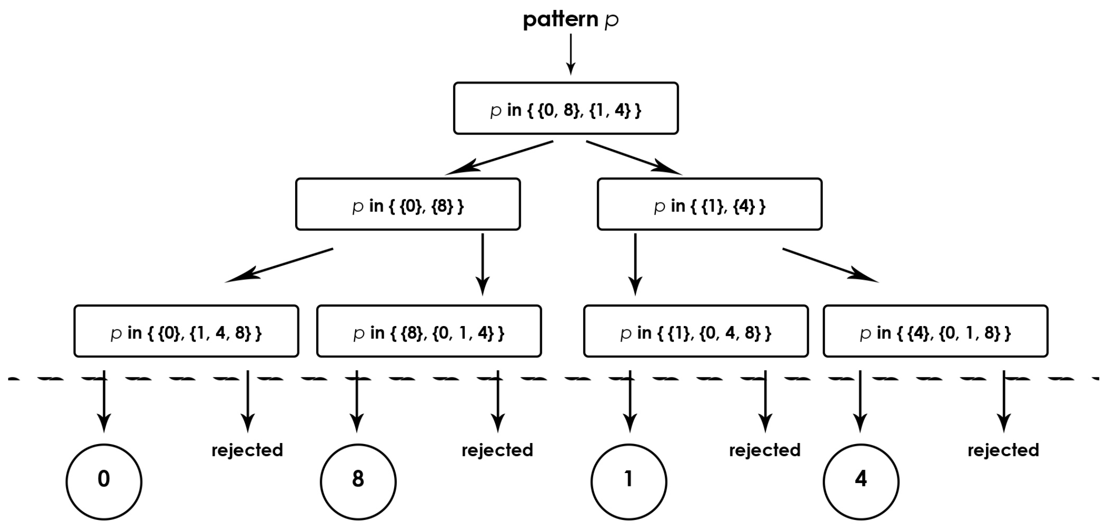

FIGURE 4.1: Balanced Tree example, trained on samples with class lables 0, 1, 4, 8. Each node (depicted as rounded rectangle) holds classifier that decides if provided pattern p is more similar to the elements in the left or right child (p in {{left_child_classes}, {rigt_child_classes}}). Dotted line at the bottom of the image depicts final decisions (element classified as a member of certain class or rejected)

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.

4.2 Slanting Tree

4.2.1 Description

Slanting Tree structure has its nodes chained in a very specific way. It always has $2n$ nodes (including leafs) where n is the number of classes in the training set. Each node represents only one class, there are two nodes per class in total, one non-leaf node and one tree leaf. Non-leaf nodes play role of initial filters that try to conclude if the received unknown pattern belongs to a class this particular node represents. In case of classifying such pattern as a native one further classification is done by a child leaf node representing the same native class. If the leaf node also classifies received element as a native one no further classification is done and the pattern is marked as an object from leaf's class. If the opposite situation occurs and the element is not recognized, it is sent to the next non-leaf node in the tree as if the leaf's parent node did not recognize the element either. In case of no more nodes in the tree left the unknown pattern is rejected and treated as a foreign one.

4.2.2 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 4.1.2). 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.

4.3 Slanting Tree 2

4.3.1 Description

Much like previously described Slanting Tree, this one has $2n$ nodes arranged in the same architecture. The difference lies in leaf nodes which, unlike the original Slanting Tree, are not using modified training data sets and have different classifier than parent nodes. The idea behind this implementation relies on the assumption that various classifiers tend to wrongly classify different patterns, so when combining them rejection rate should be improved.

4.3.2 Implementation details

Creation procedure is mostly the same as in 4.2.2. The only two differences are leaf nodes' creation, that instead of creating new training patterns takes them from the parent node, and different classifiers used by leaf and non-leaf nodes.

4.4 Results

Described in this chapter classifier trees were tested with various common classifiers: SVM, kNN and random forests, using different parameters. Over 500 tests were held. Results for training, test and letters sets were gathered in form of one big matrix with 21 rows and 11 columns. First ten rows corresponded to each of the ten classes from the training set (digits from '0' to '9'), next ten rows to the test set classes and the last one to patterns from the letters set. Numbers in each column represented how many patterns from row's class were classified as objects from native classes '0', '1', . . . , '9' or were rejected. See Table 4.1 for reference.

Every common classifier that was used by any of tree nodes was tested with different parameters. SVM had its C, gamma and kernel options adjusted (see Chapter 2 for

TABLE 4.1: Example result matrix

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**foreign**
class 0	102	1	0	9	0	0	4	0	12	0	3
class 1	2	150	0	1	0	0	2	13	0	5	0
...											
class 9	0	0	0	0	1	5	1	0	10	111	1
foreign	13	7	4	4	0	0	0	5	12	1	256

every parameter explanation). Values were as follows

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

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

$$kernel : [rbf, poly]$$

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

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

When evaluating results quality evaluation measurements were taken into account (see Chapter 3). Best solutions were selected by comparing $\frac{TP+TN}{2}$ values.

4.4.1 Balanced Tree

4.4.1.1 SVM

The best results for Balanced tree with SVM classifier were achieved when using polynomial kernel, gamma 0.5 and C parameter value of 16. Generally, for polynomial kernel, better results were achieved when using bigger C values (gamma didn't have as much impact). Similar conclusion was made for rbf kernel.

TABLE 4.2: Results for Balanced tree using SVM classifier with C=16, gamma=0.5 and kernel=poly

	class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**foreign**
training	**0**	674	0	1	1	0	0	0	0	4	0	0
	1	0	786	1	0	0	0	0	0	0	0	0
	2	0	0	716	0	0	0	0	1	0	1	3
	3	0	0	0	691	0	0	0	1	2	1	0
	4	0	0	0	0	674	0	0	0	0	1	1
	5	0	0	0	2	0	624	1	0	5	2	2
	6	0	0	0	0	0	0	673	0	0	0	1
	7	0	0	0	0	1	0	0	711	0	5	1
	8	4	0	0	0	0	0	0	1	664	1	4
	9	0	2	0	2	0	2	0	3	2	723	5
test	**0**	283	0	2	1	0	0	3	0	3	0	8
	1	1	332	2	1	0	0	1	0	3	1	7
	2	1	0	289	2	0	1	1	2	4	0	11
	3	0	0	3	291	0	4	0	5	1	0	11
	4	0	0	1	0	288	0	2	1	4	4	6
	5	0	0	0	3	0	243	0	1	1	1	7
	6	2	3	0	0	0	0	273	0	1	0	5
	7	0	0	2	2	1	0	0	301	0	1	3
	8	4	3	0	2	0	2	2	1	272	4	10
	9	0	1	0	0	1	2	0	5	5	249	7
foreign		688	2381	1311	344	2268	973	6993	637	1769	958	8061

4.4.1.2 Random Forests

When using random forests as its classifier the balanced tree didn't improve much. Whereas more native patterns were correctly recognized and assigned their labels, the foreign patterns weren't properly rejected. As it can be seen in the Table 4.3 balanced tree using random forests displayed tendency to classify unknown pattern rather than reject it. The score was achieved for classifier with 30 estimators in random forest.

4.4.1.3 k-Nearest Neighbours

Unfortunately using kNN classifier didn't bring any positive changes. Rejection mechanism is almost non-existent and the native pattern classification isn't satisfying. The best results were achieved when using 10 nearest neighbours to determine point affiliation (see Table 4.4).

Table 4.3: Results for Balanced tree using Random Forests classifier with n_estimators=30

	class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**foreign**
training	**0**	680	0	0	0	0	0	0	0	0	0	0
	1	0	787	0	0	0	0	0	0	0	0	0
	2	0	0	721	0	0	0	0	0	0	0	0
	3	0	0	0	695	0	0	0	0	0	0	0
	4	0	0	0	0	676	0	0	0	0	0	0
	5	0	0	0	0	0	636	0	0	0	0	0
	6	0	0	0	0	0	0	674	0	0	0	0
	7	0	0	0	0	0	0	0	718	0	0	0
	8	0	0	0	0	0	0	0	0	674	0	0
	9	0	0	0	0	0	0	0	0	0	739	0
test	**0**	281	1	2	0	0	0	5	0	4	0	7
	1	0	339	3	0	1	0	1	0	2	0	2
	2	1	0	285	7	0	0	1	1	2	0	14
	3	0	0	4	289	0	3	0	9	1	0	9
	4	0	1	0	0	283	0	3	1	0	7	11
	5	0	0	0	10	1	234	0	1	3	0	7
	6	1	1	1	0	1	1	272	0	3	0	4
	7	0	0	0	1	2	2	0	291	0	8	6
	8	4	3	0	2	1	1	2	0	278	1	8
	9	0	2	0	2	9	1	0	3	3	245	5
foreign		917	1075	2482	357	4330	3072	5905	158	765	477	6845

Table 4.4: Results for Balanced tree using k-Nearest Neighbours classifier with n_neighbours=10

	class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**foreign**
training	**0**	649	6	2	2	0	1	8	0	12	0	0
	1	1	757	7	4	1	4	5	0	3	4	1
	2	1	1	675	17	4	0	4	11	6	1	1
	3	0	0	4	666	0	12	1	7	5	0	0
	4	1	2	4	2	617	0	7	1	3	39	0
	5	3	0	1	16	5	591	6	0	8	6	0
	6	8	6	4	1	0	4	646	0	5	0	0
	7	0	1	10	8	11	1	0	650	1	35	1
	8	32	7	6	5	2	1	13	0	591	17	0
	9	1	3	1	15	13	8	0	9	10	677	2
test	**0**	278	6	3	2	1	0	7	0	2	1	0
	1	0	334	4	1	2	1	3	1	1	0	1
	2	0	1	290	7	0	2	2	3	3	3	0
	3	0	0	7	289	1	8	0	5	4	1	0
	4	0	2	0	0	266	0	3	4	3	28	0
	5	0	0	1	10	0	235	0	1	3	6	0
	6	3	3	0	0	0	0	276	0	2	0	0
	7	0	0	2	6	4	0	0	281	0	17	0
	8	10	5	3	2	2	2	7	2	264	3	0
	9	0	1	0	1	3	1	0	7	5	251	1
foreign		1287	1662	3115	1499	4514	5191	5443	273	1546	1163	690

4.4.2 Slanting Tree

4.4.2.1 SVM

Unlike Balanced tree using svm, where either kernel parameter value yielded similar results, Slanting tree works best when using rbf kernel. Bigger gamma values also help in maintaining higher foreign patterns rejection rates, although the final results (shown in Table 4.5) are worse than those achieved by Balanced tree.

TABLE 4.5: Results for Slanting tree using SVM classifier with C=16, gamma=0.5 and kernel=rbf

	class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**foreign**
	0	677	0	0	0	0	0	0	0	2	0	1
	1	8	778	1	0	0	0	0	0	0	0	0
	2	2	10	708	0	0	0	0	0	0	1	0
	3	1	1	14	678	0	0	0	0	1	0	0
training	**4**	1	1	5	1	667	0	0	0	0	1	0
	5	3	2	2	42	2	583	2	0	0	0	0
	6	14	11	18	0	23	13	595	0	0	0	0
	7	1	6	17	26	13	4	0	650	0	1	0
	8	59	1	10	4	4	23	8	4	558	1	2
	9	1	10	1	15	48	30	0	55	50	527	2
	0	294	0	0	1	0	1	3	0	0	0	1
	1	2	343	1	0	0	0	1	0	0	0	1
	2	1	3	302	0	0	1	0	1	1	0	2
	3	0	0	6	297	0	0	0	2	2	1	7
test	**4**	0	7	1	1	292	0	1	0	4	0	0
	5	0	0	0	18	2	234	0	0	2	0	0
	6	3	3	3	0	5	3	265	0	0	0	2
	7	0	1	5	17	7	1	0	278	0	0	1
	8	32	0	6	2	1	15	3	3	235	0	3
	9	0	5	2	2	19	15	0	23	15	189	0
foreign		1408	7052	3363	914	2778	2394	3101	399	1181	298	3495

4.4.2.2 Random Forests

Slanting tree performs best when using random forests as its internal classifier. Although it presents excellent classification abilities and the rejection rate is best among all classifier trees tested, it still can be considered only mediocre in terms of usefulness. The results, which are contained within Table 4.6, were obtained when using 100 estimators for each random forest classifier.

TABLE 4.6: Results for Slanting tree using Random Forests classifier with n_estimators=100

	class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**foreign**
training	**0**	680	0	0	0	0	0	0	0	0	0	0
	1	0	787	0	0	0	0	0	0	0	0	0
	2	1	0	720	0	0	0	0	0	0	0	0
	3	0	0	31	664	0	0	0	0	0	0	0
	4	0	3	5	0	668	0	0	0	0	0	0
	5	4	0	2	65	2	563	0	0	0	0	0
	6	21	11	14	1	13	1	613	0	0	0	0
	7	0	4	13	3	16	1	0	681	0	0	0
	8	60	3	7	0	5	10	6	2	581	0	0
	9	0	7	1	14	85	13	0	65	41	513	0
test	**0**	289	0	3	0	1	0	2	1	1	0	3
	1	1	338	3	0	1	0	0	1	1	0	3
	2	1	0	301	3	0	0	0	1	2	0	3
	3	0	0	28	266	0	0	0	8	2	0	11
	4	1	1	0	0	296	0	0	2	2	1	3
	5	0	0	1	26	1	218	0	0	3	0	7
	6	14	4	4	0	5	2	253	0	1	0	1
	7	0	0	2	4	6	1	0	294	0	0	3
	8	35	3	2	0	1	13	2	0	237	0	7
	9	0	2	0	2	41	6	0	22	15	179	3
foreign		699	654	3101	198	4059	4158	2357	162	490	187	10318

4.4.2.3 k-Nearest Neighbours

Similarly to Balanced tree, using kNN classifier in Slanting tree does not work as expected. Not only its classification is bad but also rejection does not bring satisfying results. Table 4.7 presents those results which were obtained when using kNN classifiers taking into consideration only 2 nearest neighbours for each presented, unknown pattern.

4.4.3 Slanting Tree 2

4.4.3.1 SVM

Bigger C value again proved to be better when using SVM classifier. Similarly to Balanced tree using either polynomial or rbf kernel didn't have much impact on the results. This time it was the second common classifier that played crucial part in attaining results presented in Table 4.8. In every case, when using random forests, both classification and rejection rates were the highest, with 30 estimators performing the best.

Table 4.7: Results for Slanting tree using k-Nearest Neighbours classifier with n_neighbours=2

	class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**foreign**
training	**0**	670	0	0	1	0	0	4	0	2	0	3
	1	21	754	4	0	2	1	2	0	0	1	2
	2	5	4	700	2	0	0	0	4	2	2	2
	3	3	0	29	649	0	3	0	4	2	0	5
	4	1	1	9	2	644	0	3	1	3	7	5
	5	3	4	4	39	8	567	1	0	2	1	7
	6	31	17	6	2	6	7	605	0	0	0	0
	7	0	3	23	18	24	1	0	636	0	7	6
	8	84	15	15	2	9	7	22	5	506	2	7
	9	3	6	4	15	37	17	0	59	19	569	10
test	**0**	284	1	3	1	1	1	5	0	1	0	3
	1	15	325	1	0	0	0	3	1	1	0	2
	2	3	1	297	4	1	0	0	2	1	0	2
	3	0	1	14	271	2	9	0	5	3	2	8
	4	2	3	0	0	284	0	1	3	0	8	5
	5	1	0	3	26	3	220	0	0	2	1	0
	6	11	7	3	1	1	0	258	0	3	0	0
	7	1	1	6	7	8	0	0	281	0	5	1
	8	46	13	8	1	3	7	7	2	204	2	7
	9	0	4	2	2	10	7	0	23	4	216	2
foreign		2666	2189	4021	1084	4321	4710	4262	274	1194	363	1299

Table 4.8: Results for Slanting tree 2 using SVM classifier with C=16, gamma=0.5, kernel=rbf combined with random forest with n_estimators=30

	class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**foreign**
training	**0**	677	0	0	0	0	0	0	0	2	0	1
	1	2	785	0	0	0	0	0	0	0	0	0
	2	1	0	719	0	0	0	0	0	0	1	0
	3	0	0	9	686	0	0	0	0	0	0	0
	4	0	1	5	0	669	0	0	0	0	0	1
	5	3	1	1	34	1	594	0	0	1	0	1
	6	7	7	9	0	15	3	633	0	0	0	0
	7	0	3	6	5	10	0	0	693	0	0	1
	8	40	1	6	0	3	8	7	1	604	1	3
	9	1	5	1	7	37	9	0	40	27	608	4
test	**0**	289	0	3	0	0	0	3	1	0	0	4
	1	1	341	2	0	0	0	0	0	0	0	4
	2	0	0	298	1	1	0	0	1	1	0	9
	3	0	0	4	290	0	0	0	5	2	0	14
	4	0	1	0	0	293	0	1	1	1	3	6
	5	0	0	0	12	1	236	0	0	1	0	6
	6	3	2	2	0	4	1	268	0	1	0	3
	7	0	0	1	2	1	0	0	299	0	0	7
	8	21	1	1	1	1	7	3	0	255	0	10
	9	0	1	1	1	18	5	0	12	10	215	7
foreign		767	4518	2493	304	4026	2273	3641	309	547	315	7190

4.4.3.2 Random Forests

When using random forests as its main classifier, the Slanting tree 2 scored best result with SVM as the second common classifier. The main similarity between best solution obtained for Slanting tree using SVM as its main common classifier and the one using random forests is that both of them use in fact the same two classifiers but in a reversed order. After comparing Table 4.9 with Table 4.8 it can be seen that for Slanting tree 2 it's better to use SVM backed up by random forests as its rejection rate is higher.

Table 4.9: Results for Slanting tree using Random Forests classifier with n_estimators=30 combined with SVM with kernel=rbf, C=16 and gamma=0.5

	class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**foreign**
training	**0**	677	0	0	0	0	0	0	0	2	0	1
	1	1	786	0	0	0	0	0	0	0	0	0
	2	1	0	719	0	0	0	0	0	0	1	0
	3	0	0	9	686	0	0	0	0	0	0	0
	4	0	1	4	0	670	0	0	0	0	0	1
	5	3	1	1	32	1	596	0	0	1	0	1
	6	6	6	5	0	13	4	640	0	0	0	0
	7	0	2	4	4	8	0	0	699	0	0	1
	8	34	0	3	0	2	10	4	0	617	1	3
	9	1	5	0	8	33	12	0	34	27	615	4
test	**0**	288	0	4	0	0	0	3	1	1	0	3
	1	0	341	2	0	1	0	0	1	0	0	3
	2	1	0	300	1	0	0	0	1	1	0	7
	3	0	0	5	287	0	0	0	5	3	0	15
	4	0	1	1	0	292	0	1	0	2	4	5
	5	0	0	0	15	1	232	0	0	2	0	6
	6	3	2	1	0	1	1	272	0	1	0	3
	7	0	0	0	2	2	1	0	297	0	1	7
	8	20	1	4	1	1	8	3	0	256	0	6
	9	0	1	0	1	19	6	0	12	11	215	5
foreign		724	4867	2578	270	3797	2208	3797	270	676	352	6844

4.4.3.3 k-Nearest Neighbours

Unlike the original Slanting tree, the version 2 does perform better when using kNN. After adding random forest as the second common classifier the rejection rate has increased almost 5 times. Despite the changes, the obtained solution is still outperformed by previous Slanting tree constructions (most notably the one using SVM and random forest combination).

TABLE 4.10: Results for Slanting tree using k-Nearest Neighbours classifier with n_neighbours=10 combined with random forest with n_estimators=30

	class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**foreign**
training	**0**	667	2	1	2	0	0	1	0	3	0	4
	1	0	769	5	0	0	1	2	0	1	1	8
	2	1	0	706	0	0	0	0	2	0	1	11
	3	0	0	19	666	0	1	0	3	4	0	2
	4	0	1	6	0	657	0	3	1	1	2	5
	5	4	0	0	40	2	575	1	1	0	0	13
	6	8	6	9	0	7	0	633	0	0	0	11
	7	0	1	8	3	13	0	0	679	0	8	6
	8	49	6	6	0	4	7	10	2	582	2	6
	9	0	5	2	15	47	12	0	36	32	580	10
test	**0**	287	2	3	0	0	0	2	0	1	0	5
	1	0	336	3	0	1	1	1	1	0	0	5
	2	1	0	300	3	0	0	0	0	1	0	6
	3	0	0	12	279	0	0	0	8	2	0	14
	4	1	1	0	0	290	0	0	2	3	3	6
	5	0	0	0	18	1	226	0	0	1	0	10
	6	4	3	1	0	2	1	271	0	1	0	1
	7	0	0	1	3	6	0	0	292	0	1	7
	8	26	2	4	2	1	7	4	0	246	0	8
	9	0	0	0	3	17	3	0	13	15	213	6
foreign		874	1456	3254	361	5312	4390	3223	310	642	276	6285

Appendix A

An Appendix

Bibliography