Design of robust pattern classifiers based on optimum-path forests

João P. Papa[1], Alexandre X. Falcão[1], Paulo A. V. Miranda[*,1], Celso T. N. Suzuki[†,1] and Nelson D. A. Mascarenhas[2]

[1] *Instituto de Computação (IC), Universidade Estadual de Campinas (Unicamp), SP, Brazil* {jpaulo,afalcao}@ic.unicamp.br

[2] *Departamento de Computação, Universidade Federal de São Carlos (UFSCar), SP, Brazil*
nelson@dc.ufscar.br

Abstract We present a supervised pattern classifier based on *optimum path forest*. The samples in a training set are nodes of a complete graph, whose arcs are weighted by the distances between sample feature vectors. The training builds a classifier from key samples (prototypes) of all classes, where each prototype defines an optimum path tree whose nodes are its strongest connected samples. The optimum paths are also considered to label unseen test samples with the classes of their strongest connected prototypes. We show how to find prototypes with none classification errors in the training set and propose a learning algorithm to improve accuracy over an evaluation set. The method is robust to outliers, handles non-separable classes, and can outperform support vector machines.

Keywords: supervised classifiers, image foresting transform, image analysis, morphological pattern recognition.

1. Introduction

Pattern classification methods are generally divided into supervised and unsupervised according to their learning algorithms [9]. Unsupervised techniques assume no knowledge about the classes (labels) of the samples in the training set, while these labels are exploited in supervised techniques.

We propose a method to project supervised pattern classifiers based on *optimum path forests* (OPF). The design of an OPF classifier is based on labeled samples from training and evaluation sets. A test set with unseen samples is used to assess the performance of the classifier.

The training samples are nodes of a complete graph in the sample feature space (all pairs of nodes are connected by one arc). See Figure 1(a). The arcs

[*] pavmbr@yahoo.com.br
[†] celso.suzuki@gmail.com

are weighted by the distance between the feature vectors of their nodes. A set of prototypes (key samples) is obtained from the training set. We define a *path-cost function* based on arc weights, which assigns to any path in the graph the cost of considering all samples along the path as belonging to a same class (e.g., function f_{max} which assigns the maximum arc weight along the path). We then apply the IFT algorithm [10] to partition the graph into an optimum path forest rooted at the prototypes (Figure 1(b)). That is, the prototypes compete among themselves and each prototype defines an optimum path tree, whose nodes are samples more strongly connected to that prototype than to any other root, according to that path-cost function. The training essentially consists of building this optimum path forest, where the samples in a given optimum path tree are assumed to have the same label of their root prototype.

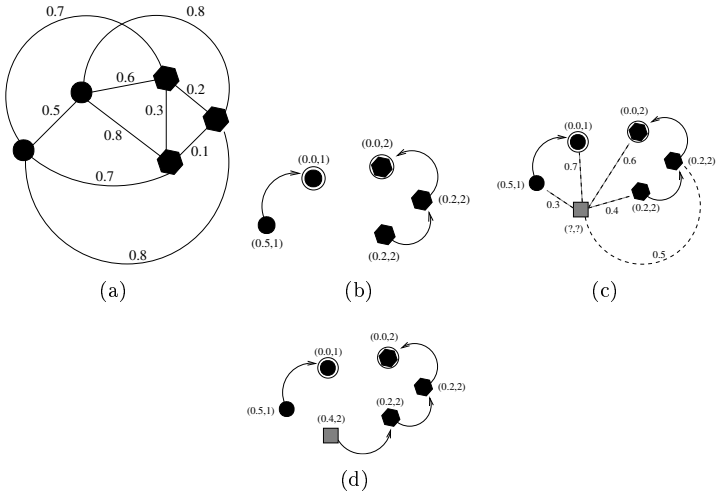


Figure 1. (a) Complete weighted graph for a simple training set. (b) Resulting optimum-path forest from (a) for f_{max} and two given prototypes (circled nodes). The entries (x, y) over the nodes are, respectively, cost and label of the samples. (c) Test sample (gray square) and its connections (dashed lines) with the training nodes. (d) The optimum path from the most strongly connected prototype, its label 2, and classification cost 0.4 are assigned to the test sample.

The classification of a test sample evaluates the optimum paths from the prototypes to this sample incrementally, as though it were part of the graph (Figure 1(c)). The optimum path from the most strongly connected prototype, its label and path cost (classification cost) are assigned to the test sample (Figure 1(d)). Note the difference between an OPF classifier with $f_{\max}$ and the nearest neighbor approach [9]. The test sample is assigned to a given class, even when its closest labeled sample is from another class. The same rule is used to classify evaluation samples.

Before testing, we propose a learning algorithm which replaces new samples of the evaluation set by *irrelevant* samples of the training set. Very often real problems limit the training set size. The learning algorithm aims to improve accuracy with this limitation. When an evaluation sample is classified, it is assigned to some optimum path in the graph. The training samples of this path have their numbers of right or wrong classifications added by one, depending on the classification result. The irrelevant samples are those with the number of wrong classifications higher than the number of right classifications. At each iteration, the learning algorithm creates new evaluation and training sets and recomputes prototypes and optimum-path forests. These prototypes guarantee none classification errors in the training set and usually improve the accuracy over the evaluation sets. The presence of outliers (samples of a given class that fall inside the region of another class) usually degrades the project of any classifier. Outliers usually become irrelevant prototypes and are moved out from the training set. The number of prototypes will not necessarily increase during learning and the most representative are usually in the frontiers between classes. The method handles non-separable classes by estimating key prototypes within the intersection regions.

Section 2 discusses related works. The OPF classifier is presented in Section 3 and Section 4 presents its learning algorithm, which outputs the last designed classifier and a learning curve showing the accuracy values of the designed classifiers along its iterations. In Section 5, we compare the OPF classifier with support vector machines (SVM) [3]. This comparison uses databases with outliers and non-separable multiple classes, being two databases from image analysis. One contains voxels from white and gray matters in magnetic resonance images of the human brain and the other contains 2D shapes from binary images. Conclusions and future works are discussed in Section 6.

2. Related works

Graph-based approaches for pattern classification are usually unsupervised. Zahn [19] proposed an approach that computes a minimum spanning tree (MST) in the graph and removes inconsistent arcs to form clusters. Arc removal in the MST can also produce hierarchical solutions for clustering, such as the popular single-linkage approach [11]. Other clustering techniques have been formulated as a graph-cut problem [17] with application to image segmentation, where the graph does not need to be complete. More recently, graph-cut techniques have also been used for learning [2]. Essentially, graph-based clustering methods aim to partition the graph into components (clusters), such that each component contains only samples of a same class. However, there is no guarantee that the samples in a given cluster belong to the same class, and it is hard to assign these samples to their correct class without any prior knowledge.

Supervised approaches usually exploit prior knowledge to teach the machine how to solve the problem. Artificial neural networks (ANN) [12] and support vector machines (SVM) [3] are among the most actively pursued approaches in the last years. An ANN multi-layer perceptron (ANN-MLP), trained by backpropagation for example, is an unstable classifier. Its accuracy may be improved at the computational cost of using multiple classifiers and algorithms (e.g., bagging and boosting) for training classifier collections [12]. However, it seems that there is an unknown limit in the number of classifiers to avoid an undesirable degradation in accuracy [16]. ANN-MLP assumes that the classes can be separated by hyperplanes in the feature space. Such assumption is unfortunately not valid in practice. SVM was proposed to overcome the problem by assuming it is possible to separate the classes in a higher dimensional space by optimum hyperplanes. Although SVM usually provides reasonable accuracies, its computational cost rapidly increases with the training set size and the number of support vectors. [18] proposed a method to reduce the number of support vectors in the multiple-classes problem. Their approach suffers from slow convergence and higher computational complexity, because they first minimize the number of support vectors in several binary SVMs, and then share these vectors among the machines. [15] presented a method to reduce the training set size before computing the SVM algorithm. Their approach aims to identify and remove samples likely related to non-support vectors. However, in all SVM approaches, the assumption of separability may also not be valid in any space of finite dimension [6].

The role of the prototypes for the OPF classifier is very similar to the importance of the support vectors for SVM. Considering this together with the fact that SVM is among the best approaches for supervised pattern classification, we have chosen the SVM code in [4] with a Gaussian kernel and parameters obtained by cross validation for comparison.

3. Optimum path classifier

Let Z_1, Z_2, and Z_3 be training, evaluation, and test sets with $|Z_1|$, $|Z_2|$, and $|Z_3|$ samples such as points or image elements (e.g., pixels, voxels, shapes). Let $\lambda(s)$ be the function that assigns the correct label i, $i = 1, 2, \ldots, c$, from class i to any sample $s \in Z_1 \cup Z_2 \cup Z_3$. Z_1 and Z_2 are labeled sets used to the design of the classifier. The applications usually impose an upper limit in $|Z_1|$, then the role of Z_2 is to improve the accuracy of the classifier by interchanging samples with Z_1. Z_3 is used to assess the performance of the classifier and it is kept unseen during the project.

Let $S \subset Z_1$ be a set of prototypes of all classes (i.e., key samples that best represent the classes). Let v be an algorithm which extracts n attributes (color, shape or texture properties) from any sample $s \in Z_1 \cup Z_2 \cup Z_3$ and returns a vector $\vec{v}(s) \in Re^n$. The distance $d(s,t)$ between two samples, s and t, is the one between their feature vectors $\vec{v}(s)$ and $\vec{v}(t)$. One can use any

valid metric (e.g., Euclidean) or a more elaborated distance algorithm [1].

Our problem consists of using S, (v, d), Z_1 and Z_2 to project an optimal classifier which can predict the correct label $\lambda(s)$ of any sample $s \in Z_3$. We propose a classifier which creates a discrete optimal partition of the feature space such that any sample $s \in Z_3$ can be classified according to this partition. This partition is an optimum path forest (OPF) computed in $\Re^n$ by the image foresting transform (IFT) algorithm [10].

Let (Z_1, A) be a complete graph whose the nodes are the samples in Z_1 and any pair of samples defines an arc in $A = Z_1 \times Z_1$ (Figure 1(a)). The arcs do not need to be stored and so the graph does not need to be explicitly represented. A path is a sequence of distinct samples $\pi = \langle s_1, s_2, \ldots, s_k \rangle$, where $(s_i, s_{i+1}) \in A$ for $1 \leq i \leq k-1$. A path is said *trivial* if $\pi = \langle s_1 \rangle$. We assign to each path π a cost $f(\pi)$ given by a path-cost function f. A path π is said optimum if $f(\pi) \leq f(\pi')$ for any other path π', where π and π' end at a same sample s_k. We also denote by $\pi \cdot \langle s, t \rangle$ the concatenation of a path π with terminus at s and an arc (s, t).

The OPF algorithm may be used with any *smooth* path-cost function which can group samples with similar properties [10]. A function f is smooth in (Z_1, A) when for any sample $t \in Z_1$, there exists an optimum path π ending at t which either is trivial, or has the form $\tau \cdot \langle s, t \rangle$ where

- $f(\tau) \leq f(\pi)$,
- τ is optimum,
- for any optimum path τ' ending at s, $f(\tau' \cdot \langle s, t \rangle) = f(\pi)$.

We will address the path-cost function f_{max}, because of its theoretical properties for estimating optimum prototypes:

$$f_{max}(\langle s \rangle) = \begin{cases} 0 & \text{if } s \in S, \\ +\infty & \text{otherwise,} \end{cases}$$

$$f_{max}(\pi \cdot \langle s, t \rangle) = \max\{f_{max}(\pi), d(s, t)\}, \tag{1}$$

such that $f_{max}(\pi)$ computes the maximum distance between adjacent samples in π, when π is not a trivial path.

The OPF algorithm assigns one optimum path $P^*(s)$ from S to every sample $s \in Z_1$, forming an optimum path forest P (a function with no cycles which assigns to each $s \in Z_1 \backslash S$ its predecessor $P(s)$ in $P^*(s)$ or a marker *nil* when $s \in S$, as shown in Figure 1(b)). Let $R(s) \in S$ be the root of $P^*(s)$ which can be reached from $P(s)$. The OPF algorithm computes for each $s \in Z_1$, the cost $C(s)$ of $P^*(s)$, the label $L(s) = \lambda(R(s))$, and the predecessor $P(s)$, as follows.

Algorithm 1. OPF.

INPUT:	*A λ-labeled training set Z_1, prototypes $S \subset Z_1$ and the pair (v, d) for feature vector and distance computations.*
OUTPUT:	*Optimum path forest P, cost map C and label map L.*
AUXILIARY:	*Priority queue Q and cost variable cst.*

1. For each $s \in Z_1 \backslash S$, set $C(s) \leftarrow +\infty$.
2. For each $s \in S$, do
3. $\quad C(s) \leftarrow 0$, $P(s) \leftarrow nil$, $L(s) \leftarrow \lambda(s)$, and insert s in Q.
4. While Q is not empty, do
5. $\quad$ Remove from Q a sample s such that $C(s)$ is minimum.
6. $\quad$ For each $t \in Z_1$ such that $t \neq s$ and $C(t) > C(s)$, do
7. $\quad\quad$ Compute $cst \leftarrow \max\{C(s), d(s,t)\}$.
8. $\quad\quad$ If $cst < C(t)$, then
9. $\quad\quad\quad$ If $C(t) \neq +\infty$, then remove t from Q.
10. $\quad\quad\quad$ $P(t) \leftarrow s$, $L(t) \leftarrow L(s)$, $C(t) \leftarrow cst$, and insert t in Q.

Lines 1–3 initialize maps and insert prototypes in Q. The main loop computes an optimum path from S to every sample s in a non-decreasing order of cost (Lines 4–10). At each iteration, a path of minimum cost $C(s)$ is obtained in P when we remove its last node s from Q (Line 5). Ties are broken in Q using first-in-first-out policy. That is, when two optimum paths reach an ambiguous sample s with the same minimum cost, s is assigned to the first path that reached it. Note that $C(t) > C(s)$ in Line 6 is false when t has been removed from Q and, therefore, $C(t) \neq +\infty$ in Line 9 is true only when $t \in Q$. Lines 8–10 evaluate if the path that reaches an adjacent node t through s is cheaper than the current path with terminus t and update the position of t in Q, $C(t)$, $L(t)$ and $P(t)$ accordingly.

The OPF algorithm for f_{max} is an "IFT-watershed transform" [13] computed in the n-dimensional feature space. Apart from this extension, the most significant contributions are the training and learning processes which find optimum prototypes (markers) in the frontier between classes and avoid *outliers* (samples of a given class that fall inside the region of another class in the feature space) in the training set, increasing the accuracy of the classifier.

The label $L(s)$ may be different from $\lambda(s)$, leading to classification errors in Z_1. The training finds prototypes with none classification errors in Z_1.

3.1 Training

We say that S^* is an optimum set of prototypes when Algorithm 1 propagates the labels $L(s) = \lambda(s)$ for every $s \in Z_1$. Set S^* can be found by exploiting the theoretical relation between *Minimum Spanning Tree* (MST) [7] and optimum path tree for f_{max}. The training essentially consists of finding S^* and an OPF classifier rooted at S^*.

By computing an MST in the complete graph (Z_1, A), we obtain a connected acyclic graph whose nodes are all samples in Z_1 and the arcs are undirected and weighted by the distance d between the adjacent sample feature vectors (Figure 2(a)). This spanning tree is optimum in the sense that the sum of its arc weights is minimum as compared to any other spanning tree in the complete graph. In the MST, every pair of samples is

connected by a single path which is optimum according to f_{max}. That is, for any given sample $s \in Z_1$, it is possible to direct the arcs of the MST such that the result will be an optimum path tree P for f_{max} rooted at s.

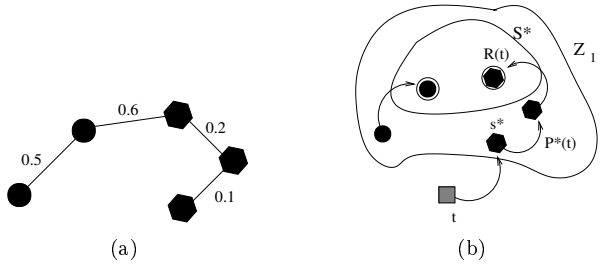


Figure 2. (a) MST of the graph shown in Figure 1a where the optimum prototypes share the arc of weight 0.6. (b) The classification of the test sample (gray square) t as in Figure 1c assigns the optimum path $P^*(t)$ from $R(t) \in S^*$ to t passing through s^*.

The optimum prototypes are the closest elements in the MST with different labels in Z_1. By removing the arcs between different classes, their adjacent samples become prototypes in S^* and Algorithm 1 can compute an optimum path forest with none classification errors in Z_1 (Figure 1(b)), which can be explained by the theoretical relation between minimum spanning trees and the optimum path tree obtained by OPF with f_{max} [8]. Note that, a given class may be represented by multiple prototypes (i.e., optimum path trees) and there must exist at least one prototype per class.

3.2 Classification

For any sample $t \in Z_3$, we consider all arcs connecting t with samples $s \in Z_1$, as though t were part of the graph (Figure 1(c)). Considering all possible paths from S^* to t, we wish to find the optimum path $P^*(t)$ from S^* and label t with the class $\lambda(R(t))$ of its most strongly connected prototype $R(t) \in S^*$ (Figure 2(b)). This path can be identified incrementally, by evaluating the optimum cost $C(t)$ as

$$C(t) = \min\{\max\{C(s), d(s,t)\}\}, \ \forall s \in Z_1. \tag{2}$$

Let the node $s^* \in Z_1$ be the one that satisfies the above equation (i.e., the predecessor $P(t)$ in the optimum path $P^*(t)$). Given that $L(s^*) = \lambda(R(t))$, the classification simply assigns $L(s^*)$ as the class of t. An error occurs when $L(s^*) \neq \lambda(t)$.

Similar procedure is applied for samples in the evaluation set Z_2. In this case, however, we would like to use samples of Z_2 to learn the distribution of the classes in the feature space and improve the performance of the OPF classifier.

4. Learning Algorithm

The performance of the OPF classifier improves when the closest samples from different classes are included in Z_1, because the method finds prototypes that will work as sentinels in the frontier between classes. We propose a learning algorithm to identify better prototypes from samples of Z_2 that have never been in Z_1 (Algorithm 2).

Algorithm 2. LEARNING.

INPUT: *Training and evaluation sets labeled by λ, Z_1 and Z_2, number T of iterations, and the pair (v, d) for feature vector and distance computations.*

OUTPUT: *Learning curve $\mathcal{L}$ and the last OPF classifier, represented by the predecessor map P, cost map C, and label map L.*

AUXILIARY: *False positive and false negative arrays, FP and FN, of sizes c, lists, LI and LE, of irrelevant samples and error samples for each class, arrays for the number of right and wrong classifications, NR and NW, of sizes $|Z_1|$, variables r for sample, and set TR to avoid samples of Z_2 return to Z_1.*

1. $TR \leftarrow \emptyset$.
2. For each iteration $I = 1, 2, \ldots, T$, do
3. $\quad TR \leftarrow TR \cup Z_1$.
4. $\quad$ Compute $S^* \subset Z_1$ as in Section 3.1 and P, L, C by Algorithm 1.
5. $\quad$ For each sample $s \in Z_1$, do $NR(s) \leftarrow 0$ and $NW(s) \leftarrow 0$.
6. $\quad$ For each class $i = 1, 2, \ldots, c$, do
7. $\quad\quad FP(i) \leftarrow 0$, $FN(i) \leftarrow 0$, $LI(i) \leftarrow \emptyset$ and $LE(i) \leftarrow \emptyset$.
8. $\quad$ For each sample $t \in Z_2$, do
9. $\quad\quad$ Find $s^* \in Z_1$ that satisfies Equation 2 and set $r \leftarrow s^*$.
10. $\quad\quad$ If $L(s^*) \neq \lambda(t)$, then
11. $\quad\quad\quad FP(L(s^*)) \leftarrow FP(L(s^*)) + 1$.
12. $\quad\quad\quad FN(\lambda(t)) \leftarrow FN(\lambda(t)) + 1$.
13. $\quad\quad\quad$ if $t \notin TR$, then $LE(\lambda(t)) \leftarrow LE(\lambda(t)) \cup \{t\}$.
14. $\quad\quad\quad$ While $r \neq nil$, do
15. $\quad\quad\quad\quad NW(r) \leftarrow NW(r) + 1$ and $r \leftarrow P(r)$.
16. $\quad\quad$ Else
17. $\quad\quad\quad$ While $r \neq nil$, do
18. $\quad\quad\quad\quad NR(r) \leftarrow NR(r) + 1$ and $r \leftarrow P(r)$.
19. $\quad$ Compute $\mathcal{L}(I)$ by Equation 5.
20. $\quad$ For each $s \in Z_1$, do
21. $\quad\quad$ If $NW(s) > NR(s)$, then
22. $\quad\quad\quad LI(\lambda(s)) \leftarrow LI(\lambda(s)) \cup \{s\}$.
23. $\quad$ For each class $i = 1, 2, \ldots, c$, do
24. $\quad\quad$ While $|LI(i)| > 0$ and $|LE(i)| > 0$, do
25. $\quad\quad\quad LI(i) \leftarrow LI(i) \backslash \{s\}$ and $LE(i) \leftarrow LE(i) \backslash \{t\}$.
26. $\quad\quad\quad$ Replace $s \in Z_1$ by $t \in Z_2$.
27. $\quad\quad$ While $|LI(i)| > 0$, do
28. $\quad\quad\quad LI(i) \leftarrow LI(i) \backslash \{s\}$.
29. $\quad\quad\quad$ Find $t \in Z_2 \backslash TR$, with $\lambda(t) = i$, and replace it by $s \in Z_1$.
30. Compute $S^* \subset Z_1$ as in Section 3.1 and P, L, C by Algorithm 1.

Firstly we give preference to replace *irrelevant* samples of Z_1 by errors in Z_2, and secondly other samples of Z_2 are replaced by *irrelevant* samples of Z_1. In both cases, we never let a sample of Z_2 return to Z_1. If the application did not impose any limitation in $|Z_1|$, the prototypes could be found from $Z_1 \cup Z_2$ with none classification errors in both sets. The learning algorithm essentially tries to identify these prototypes from a few iterations of classification over Z_2.

The algorithm outputs a *learning curve* over T iterations (Lines 2–29), which reports the accuracy values of each instance of classifier during learning, and the final OPF classifier. Lines 4–7 execute training and initialize the auxiliary arrays and lists. The classification of each sample $t \in Z_2$ is performed in Lines 8–18, updating auxiliary arrays. The condition in Line 10 indicates that t is misclassified.

In order to define irrelevant samples, we consider all right and wrong classifications in Z_2. When $t \in Z_2$ is correctly/incorrectly classified, we add one to the number of right/wrong classifications, $NR(r)$ or $NW(r)$, of every sample $r \in Z_1$ in the optimum path $P^*(t)$ from $R(t) \in S^*$ to s^* (Lines 14–18). Additionally, Lines 11–13 update the number of false positive and false negative arrays, FP and FN, for accuracy computation, and insert t in the list $LE(\lambda(t))$ of errors if t has never been in Z_1 ($t \notin TR$).

Line 19 computes the accuracy at iteration I and stores it in the learning curve $\mathcal{L}$. The accuracy $\mathcal{L}(I)$ of a given iteration I, $I = 1, 2, \ldots, T$, is measured by taking into account that the classes may have different sizes in Z_2 (similar definition is applied for Z_3). Let $NZ_2(i)$, $i = 1, 2, \ldots, c$, be the number of samples in Z_2 from each class i. We define

$$e_{i,1} = \frac{FP(i)}{|Z_2| - |NZ_2(i)|} \quad \text{and} \quad e_{i,2} = \frac{FN(i)}{|NZ_2(i)|}, \; i = 1, \ldots, c \tag{3}$$

where $FP(i)$ and $FN(i)$ are the false positives and false negatives, respectively. That is, $FP(i)$ is the number of samples from other classes that were classified as being from the class i in Z_2, and $FN(i)$ is the number of samples from the class i that were incorrectly classified as being from other classes in Z_2. The errors $e_{i,1}$ and $e_{i,2}$ are used to define

$$E(i) = e_{i,1} + e_{i,2}, \tag{4}$$

where $E(i)$ is the partial sum error of class i. Finally, the accuracy $\mathcal{L}(I)$ of the classification is written as

$$\mathcal{L}(I) = \frac{2c - \sum_{i=1}^{c} E(i)}{2c} = 1 - \frac{\sum_{i=1}^{c} E(i)}{2c}. \tag{5}$$

Lines 20–22 identify as irrelevant samples in Z_1 those with number of incorrect classifications higher than the number of correct classifications.

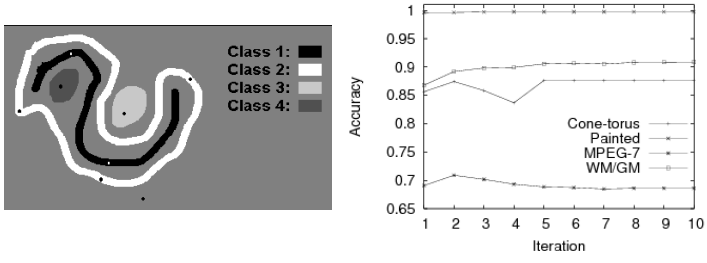
Lines 23–29 remove elements from the lists of irrelevant samples and errors, LI and LE, for each class, and first replace errors by irrelevant samples then replace the remaining irrelevant samples (if any) by other samples of Z_2 that have never been in Z_1.

Outliers degrade the project of any classifier. They will be usually identified as irrelevant prototypes, being moved from Z_1 to Z_2. Finally, Line 30 performs the training over the last set Z_1 to output the designed OPF classifier.

After learning, the classification of any sample $t \in Z_3$ is done by simply finding $s^* \in Z_1$ that satisfies Equation 2 and assigning label $L(s^*)$ as the class of t.

5. Results

We compare the OPF classifier with support vector machines (SVM [3]) using four databases with outliers and non-separable classes: Cone-torus from [12], Painted database (Figure 3(a)), MPEG-7 shape database [14], and WM/GM (white matter/gray matter) database [5]. The cone-torus database contains 400 samples and 3 non-separable classes while the painted database contains 5,867 samples with outliers and 4 classes. In both cases, the feature vectors are the sample (x, y) coordinates. The MPEG-7 database contains 1,400 2D shapes and 70 classes. To increase overlap (difficulty) between classes, we simply adopt the 126 most significant coeficients in the Fourier transform of the shapes as feature vector. The WM/GM database contains 1.5M voxels of WM and GM (2 classes) from MR-T1 images of phantoms with various levels of noise and inhomogeneity to produce outliers. The images and ground truth are available from [5], and the feature vector is the lowest and highest values around the voxel, and its intensity value. In all cases, function d is the Euclidean metric.



(a) (b)

Figure 3. (a) Painted database with outliers. (b) OPF learning curve on Z_2.

For all databases, we ramdomly selected the same percentage of samples from each class to create Z_1, Z_2 and Z_3. These percentages were 30% for

Z_1, 30% for Z_2, and 40% for Z_3 in the first three databases. Only the WM/GM database used 0.1% for Z_1, 19.9% for Z_2 and 80% for Z_3.

For Z_1 and Z_2, we runned 10 iterations of Algorithm 2 to output the OPF classifier for test on Z_3. Figure 3(b) shows the learing curves for all databases. Note the usually non-decreasing behavior of the curves after the outliers be detected as irrelevant samples and moved to Z_2.

In SVM, we used a Gaussian kernel and computed support vectors for 10 new instances of Z_1 and Z_2 by ramdomly replacing samples between them, keeping the original proportions, and took the configuration with highest accuracy for test on Z_3.

The above learning and testing processes of SVM and OPF were also repeated for 10 distinct initial sets Z_1, Z_2, and Z_3 to compute mean and standard deviation of their accuracies over Z_3 (Table 1). OPF was usually more accurate and from 3 to 20 times faster than SVM.

Table 1. Mean and standard deviation of the accuracies for each database.

	OPF accuracy		SVM accuracy	
Database	mean	std. dev.	mean	std. dev.
Cone-torus	0.8757	0.0218	0.8147	0.0145
Painted	0.9838	0.0144	0.8763	0.0030
MPEG-7	0.6925	0.0049	0.5869	0.0088
WM/GM	0.9088	0.0006	0.9072	0.0009

6. Conclusions and future work

We use the IFT algorithm in sample feature spaces and propose pattern classifiers based on optimum path forests rooted at prototypes of training sets. The OPF classifier finds prototypes with none zero classification errors in the training sets and learns from errors in evaluation sets. Unseen test sets are used to assess OPF in comparision with SVM. From the learning curves of the OPF and its results on the test sets, we may conclude it is a robust classifier and usually more accurate than SVM for databases with outliers and non-separable classes.

We are currently evaluating OPF with other databases and its accuracy is usually higher than using SVM and ANN-MLP. Future works include to report these results and the extension of OPF to unsupervised classification.

References

[1] N. Arica and F. T. Y. Vural, *BAS: A Perceptual Shape Descriptor based on the Beam Angle Statistics*, Pattern Recognition Letters **24** (2003), no. 9-10, 1627–1639.

[2] A. Blum and S. Chawla, *Learning from Labeled and Unlabeled Data using Graph Mincuts.*, ICML '01: Proceedings of the 18rd international conference on Machine learning, 2001, pp. 19–26.

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A training algorithm for optimal margin classifiers*, Proc. 5th Workshop on Computational Learning Theory, 1992, pp. 144–152.

[4] C. Chang and C. Lin, *LIBSVM: a Library for Support Vector Machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[5] D. Collins, A. Zijdenbos, V. Kollokian, J. Sled, N. Kabani, C. Holmes, and A. Evans, *Design and Construction of a Realistic Digital Brain Phantom*, IEEE Trans. on Medical Imaging **17** (1998), no. 3, 463–468. Available from: <http://www.bic.mni.mcgill.ca/brainweb>.

[6] R. Collobert and S. Bengio, *Links between perceptrons, MLPs and SVMs*, ICML '04: Proceedings of the twenty-first international conference on Machine learning, 2004, pp. 23.

[7] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT, 1990.

[8] J. Cousty, G. Bertrand, L. Najman, and M. Couprie, *Watersheds, minimum spanning forests, and the drop of water principle* (2007), no. IGM 2007-01.

[9] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed., Wiley-Interscience, 2000.

[10] A. X. Falcão, J. Stolfi, and R. A. Lotufo, *The image foresting transform: theory, algorithms, and applications*, IEEE Trans. on PAMI **26** (2004), no. 1, 19–29.

[11] L. J. Hubert, *Some applications of graph theory to clustering*, Psychometrika **39** (1974), no. 3, 283–309.

[12] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*, Wiley-Interscience, 2004.

[13] R. A. Lotufo and A. X. Falcão, *The ordered queue and the optimality of the watershed approaches*, Mathematical Morphology and its Applications to Image and Signal Processing, 2000, pp. 341–350.

[14] MPEG-7, *MPEG-7: The Generic Multimedia Content Description Standard, Part 1*, IEEE MultiMedia **09** (2002), no. 2, 78-87.

[15] N. Panda, E. Y. Chang, and G. Wu, *Concept boundary detection for speeding up SVMs*, ICML '06: Proceedings of the 23rd international conference on Machine learning, 2006, pp. 681–688.

[16] L. Reyzin and R. E. Schapire, *How boosting the margin can also boost classifier complexity*, ICML '06: Proceedings of the 23rd international conference on Machine learning, 2006, pp. 753–760.

[17] Jiambo Shi and Jitendra Malik, *Normalized cuts and image segmentation*, IEEE Trans. on Pattern Analysis and Machine Intelligence **22** (2000), no. 8, 888–905.

[18] B. Tang and D. Mazzoni, *Multiclass reduced-set support vector machines*, ICML '06: Proceedings of the 23rd international conference on Machine learning, 2006, pp. 921–928.

[19] C. T. Zahn, *Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters*, IEEE Trans. on Computers **C-20** (1971), no. 1, 68–86.