A Learning Algorithm for the Optimum-Path Forest Classifier

João Paulo Papa and Alexandre Xavier Falcão

Institute of Computing
University of Campinas
Campinas SP, Brazil

Abstract. Graph-based approaches for pattern recognition techniques are commonly designed for unsupervised and semi-supervised ones. Recently, a novel collection of supervised pattern recognition techniques based on an optimum-path forest (OPF) computation in a feature space induced by graphs were presented: the OPF-based classifiers. They have some advantages with respect to the widely used supervised classifiers: they do not make assumption of shape/separability of the classes and run training phase faster. Actually, there exists two versions of OPF-based classifiers: OPF$_{cpl}$ (the first one) and OPF$_{knn}$. Here, we introduce a learning algorithm for the last one and we show that a classifier can learns with its own errors without increasing its training set.

1 Introduction

Pattern recognition techniques can be divided according to the amount of available information of the training set: (i) supervised approaches, in which we have fully information of the samples, (ii) semi-supervised ones, in which both labeled and unlabeled samples are used for training classifiers and (iii) unsupervised techniques, where none information about the training set are available [1].

Semi-supervised [2,3,4,5] and unsupervised [6,7,8,9] techniques are commonly represented by graphs, in which the dataset samples are the nodes and some kind of adjacency relation need to be established. Zahn [7] proposed to compute a Minimum Spanning Tree (MST) in the whole graph, and further one can remove some edges aiming to partition the graph into clusters. As we have a connected acyclic graph (MST), any removed edge will make the graph a forest (a collection of clusters, i.e., trees). These special edges are called inconsistent edges, which can be defined according to some heuristic, such that an edge can be inconsistent if and only if its weight was greater than the average weight of its neighborhood. Certainly, this approach does not work very well in real and complex situations. Basically, graph-based approaches aim to add or to remove edges, trying to join or to separate the dataset into clusters [8].

Supervised techniques use a priori information of the dataset to create optimal decision boundaries, trying to separate the samples that share some characteristic from the other ones. Most of these techniques does not make use of the graph to model their problems, such that the widely used Artificial Neural Networks using Multilayer Perceptrons (ANN-MLP) [10] and Support Vector Machines (SVM) [11]. An ANN-MLP, for example, can address linearly and piecewise linearly separable feature spaces, by estimating the hyperplanes that best separates the data set, but can not efficiently handle non-separable problems. As an unstable classifier, collections of ANN-MLP [12] can improve its performance up to some unknown limit of classifiers [13]. SVMs have been proposed to overcome the problem, by assuming linearly separable classes in a higher-dimensional feature space [11]. Its computational cost rapidly increases with the training set size and the number of support vectors. As a binary classifier, multiple SVMs are required to solve a multi-class problem [14]. These points make the SVM quadratic optimization problem an suffering task in situations in which you have large datasets. The problem still increases with the chosen of the nonlinear mapping functions, generally Radial Basis Functions (RBF), in which it is needed to chose their optimal parameters by cross-validation manifold techniques, for instance, making the training phase time prohibitive.

Trying to address these problems, a novel supervised graph-based approach was recently presented [15,16]. Papa et al. [15] firstly presented the Optimum-Path Forest (OPF) classifier, which is fast, simple, multi-class, parameter independent, does not make any assumption about the shapes of the classes, and can handle some degree of overlapping between classes. The training set is thought of as a complete graph, whose nodes are the samples and arcs link all pairs of nodes. The arcs are weighted by the distances between the feature vectors of their corresponding nodes. Any sequence of distinct samples forms a path connecting the terminal nodes and a *connectivity function* assigns a cost to that path (e.g., the maximum arc-weight along it). The idea is to identify prototypes in each class such that every sample is assigned to the class of its most strongly connected prototype. That is, the one which offers to it a minimum-cost path, considering all possible paths from the prototypes. The OPF classifier creates a discrete optimal partition of the feature space and each sample of the test set is classified according to the label of its strongly connected partition (optimum-path tree root). A learning algorithm for the OPF classifier was also presented in [15], in which a third evaluation set was used to identify the most representative samples (classification errors), and then these samples are replaced by other ones of the training set. This process is repeated until some convergence criteria. The importance of a learning algorithm remains from several points: one can be used to identify the most representative samples and remove the other ones, trying to decrease the training set size. This is very interesting in situations in which you have large datasets. Another point concerns with: does a classifier can learn with its own errors? The question is: yes. We show here that a classifier can increase its performance by using an appropriated learning algorithm.

Further, Papa et al. [16] presented a novel variant of the OPF classifier, in which the graph now is seen as a k-nn graph, with arcs weighted by the distance between their corresponding feature vectors. Notice that now the nodes are also weighted by a probability density function (pdf) that takes into account the arc weights. This new variant have been overcome the traditional OPF in some situations, but none learning algorithm was developed for this last one version. In that way, the main idea of this paper is to present a learning algorithm for this new variant of the OPF classifier, as well some comparisons against the traditional OPF and Support Vector Machines are also discussed. The remainder of this paper is organized as follows: Sections 2 and Section 3 presents, respectively, the new variant of the OPF classifier and its learning algorithm. Section 4 shows the experimental results and finally, Section 5 discuss the conclusions.

2 The New Variant of the Optimum-Path Forest Classifier

Let $Z = Z_1 \cup Z_2$, where Z_1 and Z_2 are, respectively, the training and test sets. Every sample $s \in Z$ has a feature vector $\boldsymbol{v}(s)$ and $d(s,t)$ is the distance between s and t in the feature space (e.g., $d(s,t) = \|\boldsymbol{v}(t) - \boldsymbol{v}(s)\|$). A function $\lambda(s)$ assigns the correct label i, $i = 1, 2, \ldots, c$, of class i to any sample $s \in Z$. We aim to project a classifier from Z_1, which can predict the correct label of the samples in Z_2. This classifier creates a discrete optimal partition of the feature space such that any unknown sample can be classified according to this partition.

Let $k \geq 1$ be a fixed number for the time being. An k-nn relation A_k is defined as follows. A sample $t \in Z_1$ is said adjacent to a sample $s \in Z_1$, if t is k-nearest neighbor of s according to $d(s,t)$. The pair (Z_1, A_k) then defines a k-nn graph for training. The arcs (s,t) are weighted by $d(s,t)$ and the nodes $s \in Z_1$ are weighted by a density value $\rho(s)$, given by

$$\rho(s) = \frac{1}{\sqrt{2\pi\sigma^2}k} \sum_{\forall t \in A_k(s)} \exp\left(\frac{-d^2(s,t)}{2\sigma^2}\right), \tag{1}$$

where $\sigma = \frac{d_f}{3}$ and d_f is the maximum arc weight in (Z_1, A_k). This parameter choice considers all nodes for density computation, since a Gaussian function covers most samples within $d(s,t) \in [0, 3\sigma]$. However the density value $\rho(s)$ be calculated with a Gaussian kernel, the use of the k-nn graph allows the proposed OPF to be robust to possible variations in the shape of the classes.

A sequence of adjacent samples defines a path π_t, starting at a root $R(t) \in Z_1$ and ending at a sample t. A path $\pi_t = \langle t \rangle$ is said trivial, when it consists of a single node. The concatenation of a path π_s and an arc (s,t) defines an extended path $\pi_s \cdot \langle s,t \rangle$. We define $f(\pi_t)$ such that its maximization for all nodes $t \in Z_1$ results into an optimum-path forest with roots at the maxima of the pdf, forming a root set $\mathcal{R}$. We expect that each class be represented by one or more roots (maxima) of the pdf. Each optimum-path tree in this forest represents the influence zone of one root $r \in \mathcal{R}$, which is composed by samples more strongly connected to r than to any other root. We expect that the training samples of a same class be assigned (classified) to an optimum-path tree rooted at a maximum of that class. The path-value function is defined as follows.

$$f_1(\langle t \rangle) = \begin{cases} \rho(t) & \text{if } t \in \mathcal{R} \\ \rho(t) - \delta & \text{otherwise} \end{cases}$$

$$f_1(\pi_s \cdot \langle s,t\rangle) = \min\{f_1(\pi_s), \rho(t)\} \tag{2}$$

where $\delta = \min_{\forall (s,t)\in A_k | \rho(t)\neq\rho(s)} |\rho(t) - \rho(s)|$. The root set $\mathcal{R}$ is obtained on-the-fly. The method uses the image foresting transform (IFT) algorithm [17] to maximize $f_1(\pi_t)$ and obtain an optimum-path forest P — a predecessor map with no cycles that assigns to each sample $t \notin \mathcal{R}$ its predecessor $P(t)$ in the optimum path $P^*(t)$ from $\mathcal{R}$ or a marker *nil* when $t \in \mathcal{R}$. The IFT algorithm for (Z_1, A_k) is presented below.

Algorithm 1 – IFT Algorithm

Input: A k-nn graph (Z_1, A_k), $\lambda(s)$ for all $s \in Z_1$, and path-value function f_1.
Output: Label map L, path-value map V, optimum-path forest P.
Auxiliary: Priority queue Q and variable *tmp*.

```
1.  For each $s \in Z_1$, do
2.      $P(s) \leftarrow nil$, $L(s) \leftarrow \lambda(s)$, $V(s) \leftarrow \rho(s) - \delta$
3.      and insert $s$ in $Q$.
4.  While $Q$ is not empty, do
5.      Remove from $Q$ a sample $s$ such that $V(s)$ is
6.      maximum.
7.      If $P(s) = nil$, then $V(s) \leftarrow \rho(s)$.
8.      For each $t \in A_k(s)$ and $V(t) < V(s)$, do
9.          $tmp \leftarrow \min\{V(s), \rho(t)\}$.
10.         If $tmp > V(t)$ then
11.             $L(t) \leftarrow L(s)$, $P(t) \leftarrow s$, $V(t) \leftarrow tmp$.
12.             Update position of $t$ in $Q$.
```

Initially, all paths are trivial with values $f(\langle t\rangle) = \rho(t) - \delta$ (Line 2). The global maxima of the pdf are the first to be removed from Q. They are identified as roots of the forest, by the test $P(s) = nil$ in Line 7, where we set its correct path value $f_1(\langle s\rangle) = V(s) = \rho(s)$. Each node s removed from Q offers a path $\pi_s \cdot \langle s,t\rangle$ to each adjacent node t in the loop from Line 8 to Line 12. If the path value $f_1(\pi_s \cdot \langle s,t\rangle) = \min\{V(s), \rho(t)\}$ (Line 9) is better than the current path value $f_1(\pi_t) = V(t)$ (Line 10), then π_t is replaced by $\pi_s \cdot \langle s,t\rangle$ (i.e., $P(t) \leftarrow s$), and the path value and label of t are updated accordingly (Line 11). Local maxima of the pdf are also discovered as roots during the algorithm. The algorithm also outputs an optimum-path value map V and a label map L, wherein the true labels of the corresponding roots are propagated to every sample t. A classification error in the training set occurs when the final $L(t) \neq \lambda(t)$. We define the best value of $k^* \in [1, k_{\max}]$ as the one which maximizes the accuracy *Acc* of classification in the training set. The accuracy is defined as follows.

Let $NZ_1(i)$, $i = 1, 2, \ldots, c$, be the number of samples in Z_1 from each class i. We define

$$e_{i,1} = \frac{FP(i)}{|Z_1| - |NZ_1(i)|} \quad \text{and} \quad e_{i,2} = \frac{FN(i)}{|NZ_1(i)|}, \tag{3}$$

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_1, and $FN(i)$ is the number of samples from the class i that were incorrectly classified as being from other classes in Z_1. 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 Acc of the classification is written as

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

The accuracy Acc is measured by taking into account that the classes may have different sizes in Z_1 (similar definition is applied for Z_2). If there are two classes, for example, with very different sizes and the classifier always assigns the label of the largest class, its accuracy will fall drastically due to the high error rate on the smallest class.

It is expected that each class be represented by at least one maximum of the pdf and $L(t) = \lambda(t)$ for all $t \in Z_1$ (zero classification errors in the training set). However, these properties can not be guaranteed with path-value function f_1 and the best value k^*. In order to assure them, we first find the best value k^* using function f_1 and then execute Algorithm 1 one more time using path-value function f_2 instead of f_1.

$$\begin{aligned} f_2(\langle t \rangle) &= \begin{cases} \rho(t) & \text{if } t \in \mathcal{R} \\ \rho(t) - \delta & \text{otherwise} \end{cases} \\ f_2(\pi_s \cdot \langle s, t \rangle) &= \begin{cases} -\infty & \text{if } \lambda(t) \neq \lambda(s) \\ \min\{f_2(\pi_s), \rho(t)\} & \text{otherwise.} \end{cases} \end{aligned} \tag{6}$$

Equation 6 weights all arcs $(s,t) \in A_k$ such that $\lambda(t) \neq \lambda(s)$ with $d(s,t) = -\infty$, constraining optimum paths within the correct class of their nodes.

The training process in our method can be summarized by Algorithm 2.

Algorithm 2 – Training

```
INPUT:     Training set $Z_1$, $\lambda(s)$ for all $s \in Z_1$, $k_{\max}$ and path-value functions $f_1$
           and $f_2$.
OUTPUT:    Label map $L$, path-value map $V$, optimum-path forest $P$.
AUXILIARY: Variables $i$, $k$, $k^*$, $MaxAcc \leftarrow -\infty$, $Acc$, and arrays $FP$ and $FN$ of
           size $c$.

1. For $k = 1$ to $k_{max}$ do
2.     Create graph $(Z_1, A_k)$ weighted on nodes by Eq. 1.
3.     Compute $(L, V, P)$ using Algorithm 1 with $f_1$.
4.     For each class $i = 1, 2, \ldots, c$, do
5.         $FP(i) \leftarrow 0$ and $FN(i) \leftarrow 0$.
6.     For each sample $t \in Z_1$, do
7.         If $L(t) \neq \lambda(t)$, then
8.             $FP(L(t)) \leftarrow FP(L(t)) + 1$.
9.             $FN(\lambda(t)) \leftarrow FN(\lambda(t)) + 1$.
```

```
10.     Compute Acc by Equation 5.
11.     If $Acc > MaxAcc$, then
12.         $k^* \leftarrow k$ and $MaxAcc \leftarrow Acc$.
13.     Destroy graph $(Z_1, A_k)$.
14. Create graph $(Z_1, A_{k^*})$ weighted on nodes by Eq. 1.
15. Compute $(L, V, P)$ using Algorithm 1 with $f_2$.
```

For any sample $t \in Z_2$, we consider the k-nearest neighbors connecting t with samples $s \in Z_1$, as though t were part of the graph. Considering all possible paths from $\mathcal{R}$ to t, we find the optimum path $P^*(t)$ with root $R(t)$ and label t with the class $\lambda(R(t))$. This path can be identified incrementally, by evaluating the optimum cost $V(t)$ as

$$V(t) = \max\{\min\{V(s), \rho(t)\}\}, \ \forall s \in Z_1. \tag{7}$$

Let the node $s^* \in Z_1$ be the one that satisfies the above equation. Given that $L(s^*) = \lambda(R(t))$, the classification simply assigns $L(s^*)$ to t.

3 Proposed Learning Algorithm

There are many situations that limit the size of Z_1: large datasets, limited computational resources, and high computational time as required by some approaches. Mainly in applications with large datasets, it would be interesting to select for Z_1 the most informative samples, such that the accuracy of the classifier is little affected by this size limitation. It is also important to show that a classifier can improve its performance along time of use, when we are able to teach it from its errors. This section presents a learning algorithm which uses a third evaluation set Z_3 to improve the composition of samples in Z_1 without increasing its size.

From an initial choice of Z_1 and Z_3, the algorithm projects an instance I of the OPF classifier from Z_1 and evaluates it on Z_3. The misclassified samples of Z_2 are randomly selected and replaced by samples of Z_1 (under certain constraints). This procedure assumes that the most informative samples can be obtained from the errors. The new sets Z_1 and Z_3 are then used to repeat the process during a few iterations T. The instance of classifier with highest accuracy is selected along the iterations. The accuracy values $\mathcal{L}(I)$ (Equation 5) obtained for each instance I form a *learning curve*, whose non-decreasing monotonic behavior indicates a positive learning rate for the classifier. Afterwards, by comparing the accuracies of the classifier on Z_2, before and after the learning process, we can evaluate its learning capacity from the errors.

Algorithm 3 presents the proposed learning procedure for the new variant of the OPF (OPF$_{knn}$), which uses the k-nn graph as the adjacency relation. The learning procedure applied for the traditional OPF (OPF$_{cpl}$), which makes use of the complete graph, can be found in [15]. They are quite similar, and the main difference between them is the training phase in the Line 4.

Algorithm 3 – General Learning Algorithm

Input: Training and evaluation sets, Z_1 and Z_2, labeled by λ, number T of iterations, and the pair (v, d) for feature vector and distance computations.
Output: Learning curve $\mathcal{L}$ and the OPF$_{knn}$ classifier with highest accuracy.
Auxiliary: Arrays FP and FN of sizes c for false positives and false negatives and list LM of misclassified samples.

```
1.  Set $MaxAcc \leftarrow -1$.
2.  For each iteration $I = 1, 2, \ldots, T$, do
3.      $LM \leftarrow \emptyset$
4.      Train $OPF_{knn}$ with $Z_1$.
5.      For each class $i = 1, 2, \ldots, c$, do
6.          $FP(i) \leftarrow 0$ and $FN(i) \leftarrow 0$.
7.      For each sample $t \in Z_2$, do
8.          Use the classifier obtained in Line 3 to classify $t$
9.          with a label $L(t)$.
10.         If $L(t) \neq \lambda(t)$, then
11.             $FP(L(t)) \leftarrow FP(L(t)) + 1$.
12.             $FN(\lambda(t)) \leftarrow FN(\lambda(t)) + 1$.
13.             $LM \leftarrow LM \cup t$.
14.     Compute accuracy $\mathcal{L}(I)$ by Equation 5.
15.     If $\mathcal{L}(I) > MaxAcc$ then save the current instance
16.     of the classifier and set $MaxAcc \leftarrow \mathcal{L}(I)$.
17.     While $LM \neq \emptyset$
18.         $LM \leftarrow LM \backslash t$
19.         Replace $t$ by a randomly selected sample of the
20.         same class in $Z_1$, under some constraints.
```

In OPF$_{knn}$, Line 4 is implemented by computing $S^* \subset Z_1$ as described in Section 2 and the predecessor map P, label map L and cost map C by Algorithm 1. The classification is done by setting $L(t) \leftarrow L(s^*)$, where $s^* \in Z_1$ is the sample that satisfies Equation 5. The constraints in Lines $19 - 20$ refer to keep the prototypes out of the sample interchanging process between Z_1 and Z_3. These same constraints are also applied for the OPF$_{cpl}$, and for its implementation we used the LibOPF library [18].

Notice that we also applied the above algorithm for SVM classifier. However, they may be selected for interchanging in future iterations if they are no longer prototypes or support vectors. For SVM, we use the latest version of the LibSVM package [19] with Radial Basis Function (RBF) kernel, parameter optimization and the OVO strategy for the multi-class problem to implement Line 4.

4 Experimental Results

We performed two rounds of experiments: in the first one we used the OPF$_{cpl}$, OPF$_{knn}$ and SVM 10 times to compute their accuracies, using different randomly selected training (Z_1) and test (Z_2) sets. In the second round, we executed

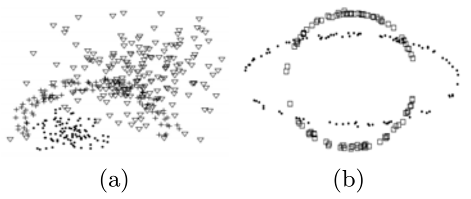

Fig. 1. 2D points dataset: (a) CONE_TORUS and (b) SATURN

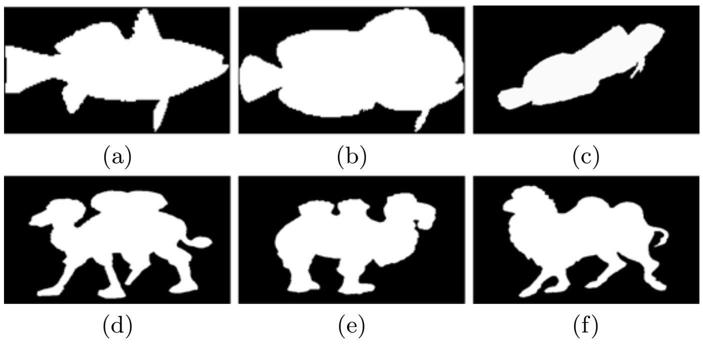

Fig. 2. Samples from MPEG-7 shape dataset (a)-(c) Fish e (d)-(f) Camel

the above algorithms again, but they were submitted to the learning algorithm. In this case, the datasets were divided into three parts: a training set Z_1 with 30% of the samples, an evaluation set Z_3 with 20% of the samples, and a test set Z_2 with 50% of the samples. Section 4.1 presents the accuracy results of training on Z_1 and testing on Z_2. The accuracy results of training on Z_1, with learning from the errors in Z_3, and testing on Z_2 are presented in Section 4.2.

The experiments used some combinations of public datasets — CONE_TORUS (2D points)(Figure 1a), SATURN (2D points) (Figure 1b), MPEG-7 (shapes) (Figure 2) and BRODATZ (textures) — and descriptors — Fourier Coefficients (FC), Texture Coefficients (TC), and Moment Invariants (MI). A detailed explanation of them can be found in [20,15]. The results in Tables 1 and 2 are displayed in the following format: $x(y)$, where x and y are, respectively, mean accuracy and its standard deviation. The percentages of samples in Z_1 and Z_2 were 50% and 50% for all datasets.

4.1 Accuracy Results on Z_2 without Using Z_3

We present here the results without using the third evaluation set, i. e., the simplest holdout method: one set for training (Z_1) and other for testing (Z_2). The results show (Table 1) that OPF$_{knn}$ can provide better accuracies than OPF$_{cpl}$ and SVM, being about 50 times faster than SVM for training.

Table 1. Mean accuracy and standard deviation without learning in Z_3

Dataset-Descriptor	OPF$_{cpl}$	OPF$_{knn}$	SVM
MPEG7-FC	71.92(0.66)	**72.37(0.48)**	71.40(0.49)
MPEG7-MI	76.76(0.60)	82.07(0.37)	**85.17(0.62)**
BRODATZ-TC	87.81(0.70)	**88.22(0.96)**	87.91(1.06)
CONE_TORUS-XY	**88.24(1.13)**	86.75(1.29)	87.28(3.37)
SATURN-XY	90.40(1.95)	**91.00(1.61)**	89.40(2.65)

4.2 Accuracy Results on Z_3 with Learning on Z_2

In order to evaluate the ability of each classifier in learning from the errors in Z_3 without increasing the size of Z_1, we executed Algorithm 3 for $T = 3$ iterations. The results are presented in Table 2.

Table 2. Mean accuracy and standard deviation with learning in Z_3

Dataset-Descriptor	OPF$_{cpl}$	OPF$_{knn}$	SVM
MPEG7-FC	73.82(0.66)	**75.94(0.48)**	74.42(0.49)
MPEG7-MI	81.20(0.60)	81.03(0.37)	**82.03(0.62)**
BRODATZ-TC	88.54(0.70)	**90.41(0.96)**	84.37(1.06)
CONE_TORUS-XY	**88.38(1.13)**	86.28(1.29)	87.95(3.37)
SATURN-XY	91.04(1.85)	**92.00(1.71)**	89.90(2.85)

We can observe that the conclusions drawn from Table 2 remain the same with respect to the overall performance of the classifiers. In most cases, the general learning algorithm improved the performance of the classifiers with respect to their results in Table 1, i. e., it is possible for a given classifier to learn with its own errors.

5 Conclusion

The OPF classifiers are a novel collection of graph-based classifiers, in which some advantages with respect to the commonly used classifiers can be addressed: they do not make assumption about shape/separability of the classes and run training phase faster. There exists, actually, two variants of OPF-based classifiers: OPF$_{cpl}$ and OPF$_{knn}$, and the difference between them relie on the adjacency relation, prototypes estimation and path-cost function.

We show here how can a OPF-based classifier learns with its own errors, introducing a learning algorithm for OPF$_{knn}$, in which its classification results were good and similar to those reported by the traditional OPF (OPF$_{cpl}$) and SVM approaches. However, the OPF classifiers are about 50 times faster than SVM for training. It is also important to note that the good accuracy of SVM was due to parameter optimization. One can see that the OPF$_{knn}$ learning algorithm improved its results, in some cases up to 3%, without increasing its training set size.

References

1. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd edn. Wiley-Interscience, Hoboken (2000)
2. Kulis, B., Basu, S., Dhillon, I., Mooney, R.: Semi-supervised graph clustering: a kernel approach. In: ICML 2005: Proc. of the 22nd ICML, pp. 457–464. ACM, New York (2005)
3. Schlkopf, B., Zhou, D., Hofmann, T.: Semi-supervised learning on directed graphs. In: Adv. in Neural Information Processing Systems, pp. 1633–1640 (2005)
4. Callut, J., Fançoisse, K., Saerens, M.: Semi-supervised classification in graphs using bounded random walks. In: Proceedings of the 17th Annual Machine Learning Conference of Belgium and the Netherlands (Benelearn), pp. 67–68 (2008)
5. Kumar, N., Kummamuru, K.: Semisupervised clustering with metric learning using relative comparisons. IEEE Transactions on Knowledge and Data Engineering 20(4), 496–503 (2008)
6. Hubert, L.J.: Some applications of graph theory to clustering. Psychometrika 39(3), 283–309 (1974)
7. Zahn, C.T.: Graph-theoretical methods for detecting and describing gestalt clusters. IEEE Transactions on Computers C-20(1), 68–86 (1971)
8. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice-Hall, Inc., Upper Saddle River (1988)
9. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(8), 888–905 (2000)
10. Haykin, S.: Neural networks: a comprehensive foundation. Prentice Hall, Englewood Cliffs (1994)
11. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the 5th Workshop on Computational Learning Theory, pp. 144–152. ACM Press, New York (1992)
12. Kuncheva, L.I.: Combining Pattern Classifiers: Methods and Algorithms. Wiley-Interscience, Hoboken (2004)
13. Reyzin, L., Schapire, R.E.: How boosting the margin can also boost classifier complexity. In: Proceedings of the 23th International Conference on Machine learning, pp. 753–760. ACM Press, New York (2006)
14. Duan, K., Keerthi, S.S.: Which is the best multiclass svm method? an empirical study. In: Oza, N.C., Polikar, R., Kittler, J., Roli, F. (eds.) MCS 2005. LNCS, vol. 3541, pp. 278–285. Springer, Heidelberg (2005)
15. Papa, J.P., Falcão, A.X., Suzuki, C.T.N., Mascarenhas, N.D.A.: A discrete approach for supervised pattern recognition. In: Brimkov, V.E., Barneva, R.P., Hauptman, H.A. (eds.) IWCIA 2008. LNCS, vol. 4958, pp. 136–147. Springer, Heidelberg (2008)
16. Papa, J.P., Falcão, A.X.: A new variant of the optimum-path forest classifier. In: 4th International Symposium on Visual Computing, pp. I: 935–944 (2008)
17. Falcão, A.X., Stolfi, J., Lotufo, R.A.: The image foresting transform: Theory, algorithms, and applications. IEEE Transactions on Pattern Analysis and Machine Intelligence 26(1), 19–29 (2004)
18. Papa, J.P., Suzuki, C.T.N., Falcão, A.X.: LibOPF: A library for the design of optimum-path forest classifiers, Software version 1.0 (2008), `http://www.ic.unicamp.br/~afalcao/LibOPF`
19. Chang, C.C., Lin, C.J.: LIBSVM: A Library for Support Vector Machines (2001), `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
20. Montoya-Zegarra, J.A., Papa, J.P., Leite, N.J., Torres, R.S., Falcão, A.X.: Learning how to extract rotation-invariant and scale-invariant features from texture images. EURASIP Journal on Advances in Signal Processing, 1–16 (2008)