High Dimensional Image Categorization

François Poulet[1] and Nguyen-Khang Pham[2]

[1] University of Rennes I - IRISA, Campus de Beaulieu,
35042 Rennes Cedex, France
francois.poulet@irisa.fr
[2] Cantho University, 1, Ly Tu Trong street, Cantho, Vietnam
pnkhang@cit.ctu.edu.vn

Abstract. We are interested in varying the vocabulary size in the image categorization task with a bag-of-visual-words to investigate its influence on the classification accuracy in two cases: in the first one, both the test-set and the training set contains the same objects (with only different view points in the test-set) and the second one where objects in the test-set do not appear at all in the training set (only other objects from the same category appear). In order to perform these tasks, we need to scale-up the algorithms used to deal with millions data points in hundred of thousand dimensions. We present k-means (used in the quantization step) and SVM (used in the classification step) algorithms extended to deal with very large datasets. These new incremental and parallel algorithms can be used on various distributed architectures, like multi-thread computer, cluster or GPU (graphics processing units). The efficiency of the approach is shown with the categorization of the 3D-Dataset from Savarese and Fei-Fei containing about 6700 images of 3D objects from 10 different classes. The obtained incremental and parallel SVM algorithm is several orders of magnitude faster than usual ones (like lib-SVM, SVM-perf or CB-SVM) and the incremental and parallel k-means is at least one order of magnitude faster than usual implementations.

Keywords: High dimensional classification, parallel algorithms, image categorization, GPU-based parallel algorithms.

1 Introduction

More and more images are stored in various databases, for example FlickR today has more than 4 billion images, it has been estimated that people having digital camera will take around 100 000 images during their whole life.

Image categorization is a very challenging task today. The most successful approach used in recent years is the bag-of-visual-words model [19] with local descriptors. The bag-of-visual-words comes from the text classification (or text categorization) area. In text categorization, we have a set of documents, each document containing a set of words. The bag of words model is the number of occurrence of each word in each document. Most of existing algorithms cannot deal easily with

L. Cao, J. Zhong, and Y. Feng (Eds.): ADMA 2010, Part I, LNCS 6440, pp. 465–476, 2010.

large number of words, that is why they use a pre-processing step to keep "interesting words". The vocabulary size is a major problem for this kind of approach. In the bag-of-visual-words, we first compute low level descriptors in particular points of image (for example SIFT descriptors [11]) and then a vector quantization is performed on these SIFTs (for example with a clustering algorithm like k-means [12], each cluster is then considered as a visual-word) to finally get the distribution of the SIFT of each image in the set of clusters obtained. Once this bag-of-visual-words is computed, machine learning algorithm is used to learn a model from this bag-of-visual words and predict the class of unlabeled image instances.

This is a high performance method for image categorization, but its main drawback is the computational cost of both parts of the method. The clustering algorithm has a complexity O(t.k.n), *k* is the number of clusters, *n* is the number of points and *t* is the number of iterations performed by the k-means algorithm. The learning algorithm can have a higher computational cost, for example if we use an SVM algorithm [23], the standard one requires solving a quadratic or linear program so the computational cost is at least $O(n^2)$, *n* being the number of points and large number of dimensions (the number of clusters obtained from the k-means, we call it the vocabulary size) is often a problem.

In order to deal with very large image datasets, we need to scale-up both algorithms, the k-means and the SVM. Several solutions can be foreseen. Starting from a sequential software program or algorithm, the easiest way is to perform a SIMD (Single Instruction Multiple Data) parallelization. This can be achieved in several ways: on a cluster of processors, this can be a cluster of CPUs (the most usual case) or a cluster a GPU cores (i.e. a GPU card). The other solution is to use a grid system (like Grid 5000 [9]) to distribute both software program and data on a computer grid. We'll focus in this paper on the first two solutions.

The remainder of the paper is organized as follows: in section 2 we describe the two algorithms (k-means and SVM) and their extensions to deal with very large datasets, section 3 describes the large image dataset categorization and the results obtained by the algorithms before the conclusion and future work.

2 Scaling-Up Algorithms

2.1 Scaling-Up the k-Means Algorithm

2.1.1 The Original k-Means Algorithm

The k-means algorithm can be summarized as shown in Table 1. In each k-means iteration, we compute for each point, the distance from this point to each cluster center. The point is then allocated to the nearest cluster. Once each point has been treated, we compute the new cluster centers. This process is repeated until there is no more variation of the points from one cluster to another or a maximum number of iterations is reached.

Table 1. Pseudo-code of the k-means algorithm

```
Input: k (the cluster number), x (data-points)
for it = 0 to nb_it do
    for i = 0 to n do
        min_dist=d(x[i],c[0]); min_cluster=0
        for j = 1 to k-1 do
            dist=d(x[i], c[j])
            if (dist<min_dist) then
                min_dist = dist; min_cluster=j
            endif
        endfor
        assign x[i] to cluster min_cluster
    endfor
    compute the new cluster centers
endfor
```

For each point x_i we need to compute the distance to each cluster center c_j:

$$d(x_i, c_j)= \sqrt{(x_i-c_j)^2} \quad (1)$$

In input of the algorithm we have X the data-point matrix of size n x d, C the cluster center matrix of size d x k and in output we have D the distance matrix of size n x k.

If we want to deal with very large datasets, we can suppose the input data point matrix cannot be loaded in memory and the distance matrix will need a very long time to be computed. We have to solve two problems: the space complexity problem to deal with very large datasets and the time complexity problem to get the result in a reasonable time.

2.1.2 Space Complexity Problem

To address the space complexity problem, the main idea is to split the whole dataset into B blocks of rows as shown in the figure 1. So we load successively the data blocks Xi into the main memory, we perform the multiplication Xi.C to get the corresponding block of distances Di and we repeat this process from X1 to XB.

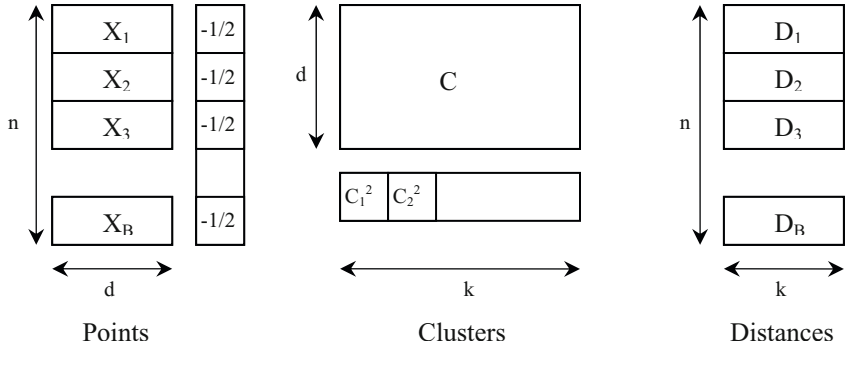

Fig. 1. Incremental k-means algorithm

With such an algorithm, we can deal with arbitrarily large datasets, whatever the dataset size is, we can split it into blocks that can be loaded into main memory to compute the corresponding distances.

2.1.3 Time Complexity Problem

We have seen in the previous sub-section we can deal with arbitrarily large datasets. Here we are interested in performing the computation in a reasonable running time. We have seen the k-means algorithm is an iterative one. We need the result of outer-loop it to perform the computation of the it+1 iteration. So the only way to perform efficient computation of the k-means algorithm is to parallelize the inner loop.

First of all, we have seen in subsection 2.1.2 we need to compute the distance between any data point x_i and each cluster center c_j (cf. equation 1). But this formulation is not the most appropriate, it is better to compute it with equation 2.

$$d(x_i, c_j)= \sqrt{(x_i-c_j)^2} = \sqrt{(x_i^2+c_j^2-2x_ic_j)} \ . \quad (2)$$

As we are only interested in finding the minimum value, we can only compute for a given x_i, $\min(c_j^2-2x_ic_j)$ or $\max(x_ic_j-1/2c_j^2)$ it will be much more efficient than the initial formula in equation 1.

To get the result in a reasonable computing time, we need now to solve the time complexity problem. We need to compute the distance between each point and each cluster center. The computation of the distance between x_i and c_j is independent of the computation of the distance $x_{i'}$ and c_j. The computation of the distance between the points and the cluster c_j can be performed in parallel in a SIMD (Single Instruction Multiple Data) way. We have split the data into blocks, we can compute the distance of each row or sub-blocks of rows in a parallel explicit way.

To perform the matrix multiplication X.C, we use the BLAS library [1], for the CPU-based version and CUBLAS library for the GPU-based one. BLAS has a multi-thread capability and CUBLAS uses the whole set of available processors of the GPU. With this library, a implicit parallelization of the algorithm is performed. The main advantage is its ease of use and its high performance. We have checked during the evaluation process of the CPU-based algorithm, the CPU use is almost always 100%, so the library uses the full machine computation capability.

2.2 Scaling-Up the SVM Algorithm

2.2.1 The Original SVM Algorithm

In spite of the prominent properties of SVMs, current SVM algorithms cannot easily deal with very large datasets. A standard SVM algorithm requires solving a quadratic or linear program; so its computational cost is at least O(m2), where m is the number of training data points and the memory requirement of SVM frequently make it intractable. There is a need to scale up these learning algorithms for dealing with massive datasets. Efficient heuristic methods to improve SVM learning time divide the original quadratic program into series of small problems [2], [16]. Incremental learning methods [3], [5], [7], [8], [17], [21] improve memory performance for massive datasets by updating solutions in a growing training set without needing to load the entire dataset into memory at once. Parallel and distributed algorithms [7], [17] improve learning performance for large datasets by dividing the problem into components that

are executed on large numbers of networked personal computers (PCs). Active learning algorithms [22] choose interesting data point subsets (active sets) to construct models, instead of using the whole dataset.

The starting point of our work is the LS-SVM classifiers proposed by [20]. Consider the linear binary classification task depicted in figure 2, with m data points x_i (i=1..m) in the n-dimensional input space R^n. It is represented by the [mxn] matrix A, having corresponding labels $y_i = \pm 1$, denoted by the [mxm] diagonal matrix D of ±1 (where D[i,i] = 1 if x_i is in class +1 and D[i,i] = -1 if x_i is in class -1). For this problem, a SVM algorithm tries to find the best separating plane, i.e. the one farthest from both class +1 and class -1. Therefore, SVMs simultaneously maximize the distance between two parallel supporting planes for each class and minimize the errors.

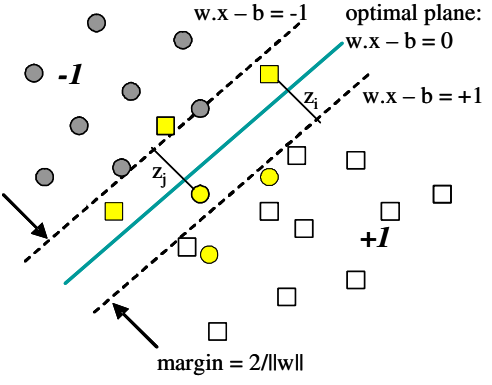

Fig. 2. Linear separation of the data points into two classes

For the linear binary classification task, classical SVMs pursue these goals with the quadratic program (3):

$$\begin{aligned} &\min \Psi(w, b, z) = (1/2)\,\|w\|^2 + cz\,, \\ &\text{s.t.: } D(Aw - eb) + z \geq e\,, \end{aligned} \tag{3}$$

where the slack variable $z \geq 0$ and the constant $c > 0$ is used to tune errors and margin size.

The plane (w, b) is obtained by the solution of the quadratic program (3). Then, the classification function of a new data point x based on the plane is: predict(x) = sign(w.x – b).

Unfortunately, the computational cost requirements of the SVM solutions in (3) are at least $O(m^2)$, where m is the number of training data points, making classical SVM intractable for large datasets. The LS-SVM proposed by Suykens and Vandewalle has used equality instead of the inequality constraints in the optimization problem (4) with a least squares 2-norm error into the objective function Ψ as follows:

- minimizing the errors by $(c/2)\|z\|^2$
- using the equality constraints D(Aw – eb) + z = e.

Thus substituting for z from the constraint in terms w and b into the objective function Ψ of the quadratic program (3), we get an unconstraint problem (4):

$$\min \Psi (w, b) = (1/2)\|w\|^2 + (c/2)\|e - D(Aw - eb)\|^2 \quad (4)$$

In the optimal solution of (4), the gradient with respect to w and b will be 0. This yields the linear equation system of (n+1) variables (w_1, w_2, …, w_n, b) as follows:

$$\Psi'(w) = cA^T(Aw - eb - De) + w = 0 , \quad (5)$$

$$\Psi'(b) = ce^T(-Aw + eb + De) = 0 , \quad (6)$$

(5) and (6) are rewritten by the linear equation system (7):

$$[w_1 w_2 w_3 ... w_n b]^T = \left(\frac{1}{c} I^o + E^T E\right)^{-1} E^T De , \quad (7)$$

where E = [A -e], I^o denotes the (n+1)x(n+1) diagonal matrix whose $(n+1)^{th}$ diagonal entry is zero and the other diagonal entries are 1.

The LS-SVM formulation (7) requires thus only the solution of linear equations of (n+1) variables (w_1, w_2, …, w_n, b) instead of the quadratic program (3).

To be able to deal with very large datasets, here again we need to solve the space and time complexity problems. We extend the LS-SVM algorithm into to ways to solve these problems.

2.2.2 Space Complexity Problem

To solve the space complexity problem, we use exactly the same incremental mechanism as for the k-means algorithm. We split the dataset into blocks of rows to fit into main memory. The linear equation system (7) can then be computed with sum on the row-based blocks (8):

$$[w_1 w_2 ... w_n b]^T = \left(\frac{1}{c} I^o + \sum_{i=1}^{k} E_i^T E_i\right)^{-1} \sum_{i=1}^{k} E_i^T D_i e_i , \quad (8)$$

E is almost the data point matrix, it is split in row blocks, D is the diagonal matrix with class of each data-point

Consequently, the incremental LS-SVM algorithm presented in table 2 can handle massive datasets on a PC. The accuracy of the incremental algorithm is exactly the same as the original one. Even if there are billions data points, the incremental LS-SVM algorithm is able to classify them on a simple PC. The algorithm only needs to store a small (n+1)x(n+1) matrix and two (n+1)x1 vectors in memory between two successive steps. The numerical test has shown the incremental LS-SVM algorithm can classify one billion data points in 20-dimensional input into two classes in 21 minutes and 10 seconds (except the time needed to read data from disk) on a PC (Pentium-IV 3 GHz, 512 MB RAM), we have solved the space complexity problem.

Table 2. Incremental LS-SVM algorithm

```
Input:
training dataset split in k blocks: A1, D1, ..., Ak, Dk
constant c to tune errors and margin size
Training:
init: E^TE=0, d=E^TDe=0
for i = 1 to k do
    load Ai and Di
    compute E^TE=E^TE+E_i^TE_i and d=d+d_i (d_i=E_i^TD_ie_i)
end for
solve the linear equation system (8)
get the optimal plane (w,b)=w_1, w_2, ..., w_n, b
Classify new data-point x according to f(x)=sign(w.x-b)
```

2.2.3 Time Complexity Problem

We have an incremental SVM algorithm able to deal with arbitrarily large datasets, now we are interested in performing the classification in a reasonable running time. We have developed different parallel versions of this algorithm, but the main idea is always the same, it is the parallel computation of the data blocks. It can be performed on CPU (to use on cluster for example) or on GPU. We give the details here of the fastest implementation we have developed. This implementation is based on GPU.

During the last decade, GPUs described in [24] have developed as highly specialized processors for the acceleration of raster graphics. The GPU has several advantages over CPU architectures for highly parallel, compute intensive workloads, including higher memory bandwidth, significantly higher floating-point, and thousands of hardware thread contexts with hundreds of parallel compute pipelines executing programs in a single instruction multiple data (SIMD) mode. The GPU can be an alternative to CPU clusters in high performance computing environments. Recent GPUs have added programmability and been used for general-purpose computation, i.e. non-graphics computation, including physics simulation, signal processing, computational geometry, database management, computational biology or data mining.

NVIDIA has introduced a new GPU, the GeForce GTX 280 and a C-language programming API called CUDA [14] (Compute Unified Device Architecture). A block diagram of the NVIDIA GeForce GTX 280 architecture is made of 3 groups of 10 multiprocessors. Each multiprocessor has 8 streaming processors for a total of 240. Each group of 8 streaming processors shares one L1 data cache. A streaming processor contains a scalar ALU (Arithmetic Logic Unit) and can perform floating point operations. Instructions are executed in SIMD mode. The NVIDIA GeForce GTX 280 has 1 GB of graphics memory, with a peak observed performance of 933 GFLOPS and 142 GB/s peak memory bandwidth. This specialized architecture can sufficiently meet the needs of many massively data-parallel computations. In addition, NVIDIA CUDA also provides a C-language API to program the GPU for general-purpose applications. In CUDA, the GPU is a device that can execute multiple concurrent threads. The CUDA software package includes a hardware driver, an API, its runtime and higher-level mathematical libraries of common usage, an implementation of Basic Linear Algebra Subprograms (CUBLAS [15]). The CUBLAS library allows access to the computational resources of NVIDIA GPUs. The basic model by which

applications use the CUBLAS library is to create matrix and vector objects in GPU memory space, fill them with data, call a sequence of CUBLAS functions and finally, upload the results from GPU memory space back to the host. Furthermore, the data transfer rate between GPU and CPU memory is about 2 GB/s.

Thus, we developed a parallel version of incremental LS-SVM algorithm based on GPUs to gain high performance at low cost. The parallel incremental implementation in table 2 using the CUBLAS library performs matrix computations on the GPU massively parallel computing architecture. It can be used on any CUDA/CUBLAS compatible GPU (today more than 200 different ones, all from NVidia). Note that in CUDA/CUBLAS, the GPU can execute multiple concurrent threads. Therefore, parallel computations are done in an implicit way.

First, we split a large dataset A, D into small blocks of rows A_i, D_i. For each incremental step, a data block A_i, D_i is loaded into the CPU memory; a data transfer task copies A_i, D_i from CPU to GPU memory and then GPU computes the sums of $E_i^T E_i$ and $d_i = E_i^T D_i e_i$ in a parallel way. Finally, the results $E_i^T E_i$ and $d_i = E_i^T D_i e_i$ are uploaded from GPU memory space back to the CPU memory to solve the linear equation system (8). The accuracy of the new algorithm is exactly the same as the original one. More details about this implementation and results on large datasets can be found in [6]. To summarize we can say the execution time of linear LS-SVM is two order of magnitude faster on GPU than on CPU and three to four order of magnitude faster than standard SVMs like libSVM [4], CB-SVM [25] or SVM-Perf [10] with almost the same accuracy.

3 Large Image Dataset Categorization

Now, we have efficient algorithms to compute the k-means and perform the supervised classification with SVM, we will use these algorithms for high dimensional image datasets. Let us briefly summarize here the full process. The starting point is an image dataset. The one we will use in our experiment is from [18], it is made of 6675 images of objects from ten different classes. In each classe, there are almost ten different objects (for example, ten different cars) and each object is taken from various camera positions: 8 different view angles, 3 different scales and 3 different heights.

For each image of the dataset we compute interest points (hessian-affine [13]), then the SIFT (Scale Invariant Feature Transform) [11], to be partially invariant to some affine transformations. The resulting vectors are 128 dimensional float vectors, on the 3D-Dataset image database, we have obtained 5.791.568 SIFTs. In order to perform the supervised classification task, we need to separate the dataset into a training-set and a test-set. Being given the number of SIFTs obtained, a reasonable choice was to choose almost ten percent of the SIFTs for the test-set.

This has been done into two different ways:

- the first way was to simply randomly select 10% of the database SIFTs to form the test-set, in this case, we can assume, there are examples of all objects both in the training-set and in the test-set (with different view-points), the results are in the first two columns of table 3,
- the second way was to remove all the SIFTs corresponding to one object (whatever the view-point was) to form the test-set, so in each training-set all the views from one object are missing, the results are in the last three columns of table 3.

We have chosen the number of SIFTs in the test-set for the first way so that it is equal to the number of SIFTs in the second one and this value has no influence on the results.

Once the data are separated into training-set and test-set, we apply the k-means algorithm with 50 iterations and various cluster numbers to evaluate the influence of the bag-of-visual-words vocabulary size in the classification step with SVM (both with linear and RBF kernel).

Several conclusions can be drawn from these experiments: the first one is when both the training-set and the test-set contain samples of all objects (even with different view-points in the test-set) the larger vocabulary size, the better results. Another interesting fact is the linear classifier performs similarly as RBF one in this case. This means we can use very fast linear SVM classifier to perform the classification task. In these experiments we do not report the linear classifier execution time, its GPU running time is always less than 1 sec., most of the user time is for reading data and writing results (these values are highly dependent on the hardware and file-system used and are not valuable information here).

When the test-set contains unknown objects in the training-set, the results are different: from almost 2k to 50k (in the linear case) and 3k to 100k (in the RBF case) the accuracy is almost the same. So the vocabulary size has really less influence on the accuracy when some occurrences of the objects are not in the training set. The column "RBF no tuning" is only to show people who are not familiar with RBF classification with non linear kernels, they absolutely need to tune the SVM parameters to get high quality results usually reported with this kind of algorithm (the accuracy can go from 11% to 69% on the same dataset without or with parameter tuning).

Table 3. Accuracy results for SVM algorithms

Vocabulary size	Linear	RBF	Linear	RBF no tuning	RBF with tuning
1k			57.83%	51.88%	62.75%
2k	74.29%	76.36%	64.64%	66.67%	66.38%
3k	79.91%	78.02%	63.62%	68.12%	68.12%
4k	77.12%	78.65%	65.80%	68.55%	68.12%
5k	78.87%	79.10%	67.10%	68.70%	69.28%
6k	79.06%	79.28%	64.06%	67.54%	67.39%
7k	79.87%	80.67%	67.10%	67.39%	67.83%
8k	79.33%	79.96%	67.10%	67.25%	69.86%
9k	79.37%	79.87%	66.23%	66.81%	68.12%
10k	80.63%	80.90%	68.12%	66.23%	69.57%
20k	82.65%	82.29%	67.25%	57.97%	69.57%
25k	83.33%	82.97%	65.94%	49.71%	68.84%
50k	84.09%	83.96%	65.65%	20.89%	73.91%
100k	84.81%	84.76%	57.10%	11.30%	68.84%
200k	85.39%	85.75%	53.62%	14.49%	66.96%
250k			50%	15.07%	65.07%
500k			40%	15.51%	61.45%

These first results are about accuracy in the classification task, about the running time, we have already said the incremental and parallel linear SVM runs in less than 1sec, for the RBF classification, we have used libSVM, it requires between 20 and 30 min for each step of the parameter tuning process (so it needs to be multiplied by the number of parameter values to be tuned, in our case between 400 and 600 different values so about 12,000 mn or 200 hours). But the main bottleneck is the k-means algorithm. 1k vocabulary size needs 6mn to be computed on two GPUs GTX-280, 10k needs 37 mn and 100k requires more than 6 hours. On CPU, even with a 16 cores PC with 100GB RAM the running time for computing 500k vocabulary is more than one week (user time) or 16 weeks or four months (cpu time). This means our choice was not the good one for the implementation of the parallelized version of the k-means, it was only the easiest one to implement. The good choice would have been to develop a version for larger cluster or grid systems.

Furthermore, the k-means is the clustering algorithm usually used in image classification, but it is known to have several drawbacks, the first one is its sensibility to the initialization step, but here we cannot run several executions of the algorithm, another one is it can only find convex clusters and we cannot assume SIFT clusters are convex in image categorization...

4 Conclusion and Future Work

Before to conclude, some words about the results reported in [18], their average accuracy was 36.9% with four objects out of the training set, so it is difficult to compare with our results. These results are only obtained on one dataset, they need to be shown on more than this.

We have applied the same incremental and parallel method for two different algorithms, it was very efficient for the SVM algorithm (3 or 4 order of magnitude faster than state of the art algorithms like libSVM, CB-SVM or SVM-Perf) and much less efficient for the k-means one. The SVM algorithm only needs a "simple" matrix multiplication to be solved while the K-means needs several ones to be solved one by one in an iterative way to get its final result. This last one is much more difficult to efficiently be parallelized. In this case, a good choice could be to choose the most available possible cpu power. The size of the dataset used with SVM is far from the maximum the algorithm can deal with.

The k-means algorithm is the most frequently used in image classification despite its drawbacks. It may be because it is simple to use and implement and has a quite low complexity compared to other algorithms, so another clustering algorithm could give better results, we will investigate this possibility.

The results we have obtained are really different according to the type of test-set we used. When the test-set is made of random images with all object types, the accuracy is rather a good one, but when one object is not in the training set, the accuracy is much lower. May be the SIFT descriptor is efficient for finding occurrences of the same object with various view points (it has been designed for this) but not so efficient to find different objects…

Finally the SVM speed is good enough (cpu time is less than one second) an obvious improvement will be to use binary files in input/output of the algorithm to reduce the user computation time.

References

1. Blackford, L.S., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., Pozo, R., Remington, K., Whaley, R.C.: An Updated Set of Basic Linear Algebra Subprograms (BLAS). ACM Trans. Math. Soft. 28(2), 135–151 (2002)
2. Boser, B., Guyon, I., Vapnik, V.: A Training Algorithm for Optimal Margin Classifiers. In: 5th ACM Annual Workshop on Computational Learning Theory, Pittsburgh, Pennsylvania, pp. 144–152 (1992)
3. Cauwenberghs, G., Poggio, T.: Incremental and Decremental Support Vector Machine Learning. In: Advances in Neural Information Processing Systems, vol. 13, pp. 409–415. MIT Press, Cambridge (2001)
4. Chang, C.C., Lin, C.J.: LIBSVM: a Library for Support Vector Machines (2001), Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
5. Do, T.N., Fekete, J.D.: Large Scale Classification with Support Vector Machine Algorithms. In: 6th International Conference on Machine Learning and Applications, ICMLA 2007, pp. 7–12. IEEE Press, Ohio (2007)
6. Do, T.N., Pham, N.K., Poulet, F.: GPU-based Parallel SVM Algorithm. Journal of Frontiers of Computer Science and Technoloy 3(4), 368–377 (2009)
7. Do, T.N., Poulet, F.: Classifying one Billion Data with a New Distributed SVM Algorithm. In: 4th IEEE International Conference on Computer Science, Research, Innovation and Vision for the Future, RIVF 2006, Ho Chi Minh, Vietnam, pp. 59–66 (2006)
8. Fung, G., Mangasarian, O.: Incremental Support Vector Machine Classification. In: The 2nd SIAM Int. Conf. on Data Mining, SDM 2002, Arlington, Virginia, USA (2002)
9. Grid5000, `https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home`
10. Joachims, T.: A Support Vector Method for Multivariate Performance Measures. In: The International Conference on Machine Learning, ICML (2005)
11. Lowe, D.: Object Recognition from Local Scale-Invariant Features. In: The 7th International Conference on Computer Vision, ICCV 1999, vol. 2, pp. 1150–1157 (1999)
12. McQueen, J.: Some Methods for classification and Analysis of Multivariate Observations. In: The 5th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297. University of California Press, Berkeley (1967)
13. Mikolajczyk, K., Schmid, C.: Scale and Affine invariant interest point detectors. International Journal of Computer Vision 60(1), 63–86 (2004)
14. NVIDIA® CUDA™, CUDA Programming Guide 1.1 (2007)
15. NVIDIA® CUDA™, CUDA CUBLAS Library 1.1 (2007)
16. Platt, J.: Fast Training of Support Vector Machines Using Sequential Minimal Optimization. In: Advances in Kernel Methods – Support Vector Learning, pp. 185–208 (1999)
17. Poulet, F., Do, T.N.: Mining Very Large Datasets with Support Vector Machine Algorithms. In: Enterprise Information Systems, pp. 177–184. Kluwer Academic Publishers, Dordrecht (2004)
18. Savarese, S.L.: Fei-Fei.:3D generic object categorization, localization and pose estimation. In: International Conference on Computer Vision (2007)
19. Sivic, J., Zisserman, A.: Video Google: A Text Retrieval Approach to Object Matching in Videos. In: The International Conference on Computer Vision, pp. 1470–1477 (2003)

20. Suykens, J., Vandewalle, J.: Least Squares Support Vector Machines Classifiers. Neural Processing Letters 9(3), 293–300 (1999)
21. Syed, N., Liu, H., Sung, K.: Incremental Learning with Support Vector Machines. In: The Workshop on Support Vector Machines at the International Joint Conference on Artificial Intelligence, Stockholm, Sweden (1999)
22. Tong, S., Koller, D.: Support Vector Machine Active Learning with Applications to Text Classification. In: ICML 2000, The 17th Int. Conf. on Machine Learning, Stanford, USA, pp. 999–1006 (2000)
23. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (1995)
24. Wasson, S.: Nvidia's GeForce 8800 graphics processor. Technical report, PC Hardware Explored (2006)
25. Yu, H., Yang, J., Han, J.:Classifying Large Data Sets Using SVM with Hierarchical Clusters.In: The 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 306–315(2003)