Paper

Solving Support Vector Machine with Many Examples

Paweł Białoń

Abstract—Various methods of dealing with linear support vector machine (SVM) problems with a large number of examples are presented and compared. The author believes that some interesting conclusions from this critical analysis applies to many new optimization problems and indicates in which direction the science of optimization will branch in the future. This direction is driven by the automatic collection of large data to be analyzed, and is most visible in telecommunications. A stream SVM approach is proposed, in which the data substantially exceeds the available fast random access memory (RAM) due to a large number of examples. Formally, the use of RAM is constant in the number of examples (though usually it depends on the dimensionality of the examples space). It builds an inexact polynomial model of the problem. Another author's approach is exact. It also uses a constant amount of RAM but also auxiliary disk files, that can be long but are smartly accessed. This approach bases on the cutting plane method, similarly as Joachims' method (which, however, relies on early finishing the optimization).

Keywords—concept drift, convex optimization, data mining, network failure detection, stream processing, support vector machines.

1. Introduction

The application of optimization methods in data analysis, especially in telecommunications, yields optimization problems with a very specific structure. To the author's opinion, this specificity will have to make deep changes in the optimization science itself, by forcing the algorithm designers to work with unusual circumstances and requirements.

We shall exemplify this claim with the case of linear classification of points in $\mathbb{R}^N$, each preassigned to one of two classes: A or B. We shall deal with the optimization problem encoding the linear classification task, called support vector machine (SVM) problem. This problem will be precisely formulated later. Now it suffices to say that the problem of linear classification consists in finding a hyperplane in $\mathbb{R}^N$ that properly (or as properly as possible) separates these points into the classes.

Looking at the SVM problems that are nowadays analyzed, we notice that many of them are obtained automatically. This is very common to the telecommunication applications. For a very simplified example, each "point" can represent a state of a telecommunication network measured with the simple network management protocol (SNMP), with coordinate values representing, e.g., the traffic in particular arcs of the network, particular elements of the connection matrix, error parameters, etc. The two classification classes could be the proper state of the network or a failure, and the classification hyperplane, for some training points, pre-assigned to these classes by a teacher, could be further used in automatic failure detection.

This example shows two specific structural properties of the data:

1. There may be very many classification points. For example, this will happen if the SNMP data come at regular time intervals like tens of seconds and are collected through a long period, perhaps several months. In the resulting optimization, it will be possible that the random access memory (RAM) exhausts with all this data, so we may be not able to store the optimization problem in RAM.
2. The data may be very dense, resulting with optimization problems that are unusually dense for the optimization standards. Usually we hope for some level of sparsity of optimization problems claiming that the the input data must be in some way verified by a human and that he cannot conceive too many nonzero numbers. Now, however, the situation becomes different: the data is not produced by a human, like a modeler cooperating with the optimization expert but produced automatically. And it is not surprising that each sample is relatively dense in our example: the traffic volume in a particular arc of the network is usually nonzero at any moment.

   Having a large, dense optimization problem is a very untypical case for a common imagination of a specialist at optimization.

The author believes the above two features can be also present in many other applications in which the data is obtained automatically at regular time bases, e.g., as the log of the behavior of customers of telephony subscribers, bank clients, supermarket clients, medical sensor data, etc.

Stream processing. The extreme case of dealing with long streams of input points is the case of *stream processing*. Stream processing (see [1]) is a general data-mining concept, relevant to problems with data that can be aligned in a stream of similar items, like records in a database. In linear classification, we can have a stream of preassigned points. The algorithm for solving a problem with such data has the stream processing character if it uses memory constant in the stream length (number of items in the stream). This means that the each incoming portion of the data from the input stream has to be processed in a sense on-line, i.e., the algorithm can, for example, update some partial stream statistics with this portion of data but cannot remember all the data read so far. We can think of stream processing like a new name for an old concept of "ideal processing algorithm".

Stream processing is *very* unusual in optimization. Almost all optimization algorithms assume they have random access to particular parameters defining the optimization problem, i.e., it difficult to predict which parameter the algorithm will read in its next iteration. Also, the algorithm can return to some parameter so far read, i.e., it can read one parameter several times. Thus all the parameters defining the problem must be constantly accessible. If they do not fit all together in RAM but, for example, fit in a disk file, we still can think of building a smart *oracle* that communicates the required parameter to the algorithm, cleverly (fast) navigating through the file: we shall show such a solution. In the case of stream processing we cannot even have a long file, and this situation requires a completely new approach to solving the optimization problem.

Concept drift. Some methods of processing long streams are able to take into account the phenomenon of *concept drift*. To explain this phenomenon it is convenient to think of the input stream as of infinite. The algorithm solves its problem periodically, and each time the problem instance is defined with the portion of the stream from the beginning to the last portion of data read. It may happen that the incoming data slowly change in time because of the reality or the phenomena described by the data also slowly change. For example, the number of user of the computer network we probe increases, and this changes the traffic characteristics. This is called concept drift and the solution of our problem, like the separating hyperplane, must also slightly evolve in time. Thus to take into account the concept drift, our algorithm must be first of all capable of giving periodic solutions with the portions of the input data stream "from the beginning up to now". Moreover, we often impose some gradual forgetting of older data: the older the data is, the less it weights in the definition of the current problem instance. Of course, a precise definition of weighting would have to be written, dependent on the particular problem being solved. Still it is reasonable to assume that our memory is far too low to store all the "new" part of the input data stream in.

In our critical analysis in which we use results obtained by Joachims in [2], the author's own result from [3] and a new author's concept.

2. Linear Support Vector Machine Problem

Linear support vector machine problem [4], [5] is a certain formalization of the problem of finding a hyperplane separating as well as possible points (training examples) in $\mathbb{R}^N$ that have been preassigned to two classes A or B each. There are many variants of the way of detailed posing this problem. We shall consider the variant with an affine hyperplane and inexact separation.

We have n training examples a_j, $a_j \in \mathbb{R}^N$ for $j = 1,..,n$ each either of class A or class B. We look for a separating rule of the form

$$\omega^\top x \geq \gamma, \tag{1}$$

where $x \in \mathbb{R}^N$ is a variable while $\omega \in \mathbb{R}^N$, $\gamma \in \mathbb{R}$ are the classifier parameters.

To obtain ω and γ we solve the following linear support vector machine optimization problem:

$$\begin{aligned} &\underset{\omega\in\mathbb{R}^N,\,\gamma\in\mathbb{R},\,y\in\mathbb{R}^n_+}{\text{minimize}} \quad \frac{1}{2}\|\omega\|^2 + Ce^\top y \qquad (2)\\ &\text{subject to}\\ &-d_j\cdot(a_j{}^\top\omega-\gamma)-y+1\leq 0\,, \quad \text{for } j=1,\ldots n\,. \end{aligned}$$

Each d_j is either -1 – if example a_j is of class A or 1 – if example a_j is of class B.

The optimal ω and γ of this problem yield the separating Ineq. (1) that can be used to classify any point $x \in \mathbb{R}^N$ during the classifier working phase: if the rule is satisfied for x, then x is classified to class A, else it is classified to class B.

The obtained separation hyperplane tries to conceive two phenomena depicted in Fig. 1: separation violation and separation with a margin.

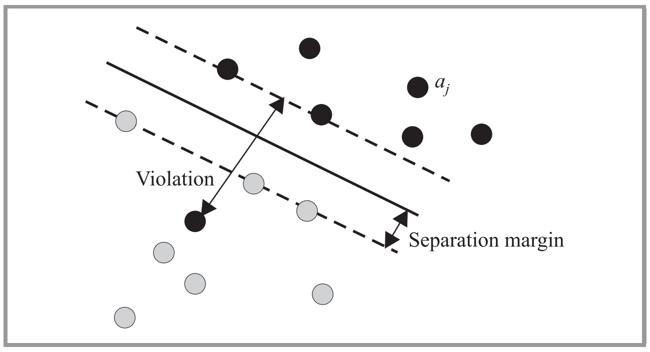

Fig. 1. Separation margin and separation errors. The training points are black and grey, indicating their belonging to one of the two classes.

A *separation margin* is obviously needed to avoid errors in classification. The points given to the classifier are distributed similarly but not identically as the training points. In turn, we allow that little training points be misclassified by the separation hyperplane, first because the problem may be not exactly linear separable, some training points may be distorted or in other way invalid, or there is too little of them to reasonably require the exact separation of them, scarifying other properties of the separation hyperplane.

The variables y_j represent the separation violations of particular training points. It can be shown that the separation equals to $1/\|\omega\|$ – since we do not want to go into details of the scaling present in problem (2), we can refer the reader to [4] for the proof.

Instead of maximizing $1/\|\omega\|$, we can minimize $\|\omega\|^2$, which is easier. Having said this, we can see what the goal function of problem (2) expresses: we tend to minimize the total separation violation and maximize the separation margin, the weight controlling the compromise is C.
This paper will deal mainly with two cases of size of this problem:

1. Number n of examples is large. This is a simpler case.

2. The RAM used by the algorithm is at most constant in n. This is a more difficult case of stream processing.

As opposed to n, we shall assume the number N of features is at most moderate. Otherwise, having in mind also the density of the problem, the problem would become too difficult even for most approaches discussed in this paper.

3. Approaches with Cutting Planes Oracles, Generation of Constraints and Cutting Planes

For problems with many constraints it is natural cutting plane methods connected with the oracle module that knows the problem instance and generates proper cuts.
Two of the approaches discussed here – [2] and [3] are concretizations of this idea. They differ slightly in the details of the formulation of the SVM problem but here they both can be described in terms of problem (2). Both the approaches assume the training examples are stored simultaneously in memory, so the oracle can return to some example.

Reformulation of the problem. First, we write an equivalent form of problem (2), in order to get rid of numerous decision variables y_j:

$$\underset{\omega\in\mathbb{R}^N,\,\gamma\in\mathbb{R},\,z\in\mathbb{R}}{\text{minimize}}\ \frac{1}{2}\|\omega\|^2+Cz \tag{3}$$

subject to

$$\sum_{j\in\{1,\ldots,n\}}\max(0,-d_j\cdot(a_j{}^\top\omega-\gamma)+1)-z\le 0\,.$$

A further, redundant reformulation is:

$$\underset{\omega\in\mathbb{R}^N,\,\gamma\in\mathbb{R},\,z\in\mathbb{R}}{\text{minimize}}\ \frac{1}{2}\|\omega\|^2+Cz \tag{4}$$

subject to

$$\sum_{j\in I}\max(0,-d_j\cdot(a_j{}^\top\omega-\gamma)+1)-z\le 0\ \text{, for } I\in 2^{\{1,\ldots n\}}\,.$$

The equivalence of the formulations comes from the nonnegativity of the terms summed. Because of this nonnegativity, all the constraints in problem (4) are implied by the constraint in problem (3).

There is a huge number, $2^{\{1,\ldots,n\}}$ of constraints in problem (4). Certainly, all of them have their representation in memory. Instead, some constrained unsatisfied by an algorithm iterate x^k is generated by the oracle that is given x^k (if all the constraints are satisfied at x^k, the oracle returns a proper cut based on the gradient of the goal function at x^k). The gradient of this constraint defines a cut in our algorithm.
The reason of introducing redundant constraint is to accelerate the algorithm. Computing the gradient of a constrained in which the summations runs only over some subset I of $\{1,\ldots,n\}$, can be computationally easier than computing the gradient of the constraint in problem (3), which requires summing over $\{1,\ldots,n\}$.
Finding an unsatisfied constrained does not mean to try all the $2^{\{1,\ldots,n\}}$ constraints. A constraint with a bigger set I can be certainly obtained by an update of a constraint of a lower I. Thus the oracle needs only a single loop. In its consecutive iterations,the current I is enhanced by a new j. If we go up to the situation $I=\{1,\ldots,n\}$ having not found any unsatisfied constraint, we know there is no unsatisfied constraints (since we add positive numbers). Then the oracle can return a cut based on the gradient of the goal function.

Solving the reformulated problem. We shall compare the 2 approaches.
In [3] the problem (4) is tackled as follows.

1. The problem is solved with the Nesterov analytic cutting plane method with a penalty term [6].

2. The input data, defining the problem instance, is stored in a disk file, as it is too big to fit in RAM.

3. The oracle reads the file but since reading files is slow, the way the oracle navigates through the file is smart. Namely, it involves two accelerating mechanisms

   (a) The first one is the already defined incremental construction of constraints within the oracle

   (b) In the late stages of an optimization run, the above mechanism is inefficient, since near the solution, most of the problem constraints are satisfied, so one call of the oracle usually involves reading nearly or exactly n training points. But near the solution, the iterate does not move too much between iteration. So, instead of explicit checking violation of constraints by x^k we can assess this violation using the knowledge whether the respective constraint was violated by some earlier iterate, say x^{k-s}. The details of the assessment are described in [3]. It leads to the necessity of bucketing the input file, a certain surrogate of sorting this file due to some quantity.

It is interesting that in navigating our file we had to use a language characteristic for more traditional data processing or for databases rather than to optimization, e.g., we used bucketing. In the author's opinion this may be very indicative for the future of optimization science, that will be faced long streams of automatically generated data.

4. The unconstrained subproblems from the Nesterov method are solved directly (in primal).

In the approach of Joachims [2]:

1. Problem (4) is solved with the Kelley cutting plane method [7].
2. The subproblems from the Kelley method are transformed to their duals before being solved.
3. The k subproblems dual (the dual of the subproblem in kth iteration of the Kelley method) is a dense problem with about k variables. Also, k is the number of cuts made so far.
4. There are several interesting features of the Joachms' approach:
   (a) The most astonishing is its linear complexity in both N and n under given accuracy demand. This will be discussed later.
   (b) The method does not invert large matrices. The only matrix that might have to be invert may be the hessian matrix in some particular method solving a subproblem from the Kelley method; this hessian, however, is dense and of the size about $k \times k$ while even the largest k is assumed to be at most moderate in the algorithm, as discussed later.

Effectiveness under large number of examples. The story about how Joachims achieves his annoying linear complexity in both n and N is very meaningful and illuminating.

The author of this paper has made some experiments with the Joachims' solver. What quickly stroke was a very low (loose) default accuracy setting for this solver.

It turned out that the exceptionally good complexity in N and n is obtained at a cost of the rather quick dependence of the number of cuts (cutting plane iterations) on accuracy and the weight C. The number of the cutting plane method iterations is assessed as

$$\frac{8CR^2}{\varepsilon^2}, \tag{5}$$

where R is the radius of the set of a_js, ε is the solution accuracy in terms of the goal function. A comment is owed to the influence of C. The higher C, the less is the resulting separation margin and separation violations are penalized more. This makes the problem obviously harder, thus a larger number of necessary iterations of the method is not surprising.

An important conclusion is that a great gain in the speed of processing of long files of automatically generated data is to loosen the demands on accuracy.

A question that arise is whether the accuracy wanted by Joachims is sufficient in practice. The answer is not clear. A reasoning conducted in [3] says it is not enough. Simply, Joachims assumes that the number of iterations done by the cutting plane method will be low, even lower than the number of features N. Then the dense subproblem with an approximately $x \times k$ hessian is solvable within a reasonable time[1]. However, the approach becomes problematic when we see that the number of iterations of the cutting plane method is equal to the number of cuts generated during the optimization run. Our geometric intuition says that to properly isolate the solution in $\mathbb{R}^N$ by cuts we need rather of the rank of N cuts. In [3] we consider the following example.

Assume the number of cuts generated by the algorithm of Joachims is at least DN where D is a positive constant. Then only the last iteration of this algorithm costs

$$O\left(\theta(DN) + nD^2N^3\right). \tag{6}$$

This result is obtained under a reasonable assumption that solving a minimal optimization-state-of-art cost of solving kth subproblem and the costs of transformation to the dual. The above cost is already not linear in N.

Table 1
Comparison of the solvers' reaction to increasing C, problem `covtype`, $n = 523293$, $N = 54$, default accuracies

C	0.1	10	1000
Time – author [s]	1572	1510	1453
Time – Joachims [s]	384	4708	2739

Table 2
Comparison of the solvers' reaction to increasing C, problem `biology`, $n = 131320$, $N = 74$, $C = 0.1$; ε, is the accuracy setting for the Joachims' solver

e	0.0001	0.01	1000
Time – Joachims [s]	(> 2 hours)	118	287

However, the experiments with data coming from the practice do not support this theoretical reasoning. Neither do the experiments in [2] nor the experiment the author of this paper did in [3]. In the later experiments, a similar pattern of the solvers of Joachims and the solver of the author of this paper occurred. With the default settings and relatively low C. Increasing C and/or decreasing the solution, we quickly stuck the Joachims' solver, while the authors' solver, though maybe slower, obtained the solution (see the sample Table 1 and Table 2 for the experiments on bench-

[1]Moreover, it is still solvable in the case we have excluded from the scope of this paper, when N is big, which is the case definitely too difficult to the [3] approach, as the nondifferentiable Nesterov method will not work well with many variables of the subproblem.

mark examples from KDD04[2]). The authors' solver use the same very tight, default accuracy, measured in terms of distance from the solution rather than in terms of goal function.

Most interesting was that this increasing C or decreasing accuracy did not lead to the increase in the quality of the obtained classifier, measured by the accuracy of the classifier, i.e., the percent of well classified testing examples.

Thus the approach of Joachims – obtaining high efficiency, algorithm simplicity (e.g., no need to invert large matrices) consciously sacrificing some accuracy is the potential way of solving optimization problems with large, dense, automatically generated data. Optimization specialists should take this way into account and most attention should go to research on what accuracies are acceptable in practice.

Also, we see by the solution from [3] that optimization will have to borrow some language and solutions from databases or from more traditional data processing domains (e.g., sorting).

4. Approach for the Stream Case

Both the above solutions were semi-tools for the question of large data streams. They both allow returning to a particular training example, thus are not feasible for streams. We present below an idea for proceeding in the such a case.

For streams in optimization, the most natural approach is to make a model of the optimization problem that fits in memory constant with the stream length. We shall not go beyond this obvious approach, unlike, say, the ambitious approach in [8], in which we see some stream attitude in this that an optimization algorithm is itself essentially stream: new portion of data cause an update in the solution. However, the accuracy of the solution obtained in [8] is not great and the algorithm actually has an option to return to items previously read from the stream.

We shall use problem (3). Note that the most difficult in this problem is the sum in its constraint, which makes the constraint not storable in memory constant in n.

Note, however, that we have the nondifferentiable function $\max(0,\cdot)$ in the components of this sum. However, if we replace $\max(0,\cdot)$ with a polynomial $\phi : \mathbb{R} \mapsto \mathbb{R}$ the constraint will be storable in such RAM.

So, we can solve problem (3) in the following steps.

1. Reformulate SVM problem (2) as

$$\underset{\omega\in\mathbb{R}^N,\gamma\in\mathbb{R}}{\text{minimize}}\ \frac{1}{2}\|\omega\|^2 + C\sum_{j=1,\dots,n}\phi\left(-d_j\cdot({a_j}^\top\omega-\gamma)+1\right).$$

2. We approximated $\max(0,\cdot)$ in the constraint of problem (3) by a polynomial $\phi(\cdot) : \mathbb{R} \mapsto \mathbb{R}$, say, for example, of order 3.

   The constraint of the approximate problem is easily storable in RAM constant in n, since each function under sum is of the form $\psi : \mathbb{R}^{N+1} \mapsto \mathbb{R} \equiv \phi(w^\top x)$, where $w \in \mathbb{R}^N$ and is representable as:

$$\psi(x) = \sum_{k=1}^{N+1}\sum_{l=1}^{N+1}\sum_{m=1}^{N+1} T^3_{k,l,m}x_k x_l x_m + \sum_{k=1}^{N+1}\sum_{l=1}^{N+1} T^2_{k,l}x_k + \sum_{k=1}^{N+1} T^1_k x_k + T^0.$$

   To sum such vectors we need to respectively add the tensors defining particular components. So effectively we need one 3-dimensional tensor, one matrix, one vector and the constant. All of these objects have sizes dependent only on N.

3. Solve the approximate problem

Certainly, the open problem is how to choose the approximating polynomial so that the perturbance of the original problem is acceptable in practice. Also, perhaps more attention will be directed to operating on dense matrices/tensors, i.e., approximations with forcing some element values to zeros.

5. Conclusions

The conclusions from this work are following.

1. The practice yields new challenges to the science of optimization that have a potential of substantially change the research in optimization
   (a) The data created automatically can form very long streams, that are not storable in RAM and even force the algorithm to have a stream character, i.e., the memory usage constant in the stream length.
   (b) Such automatically generated data can be dense, resulting in dense optimization problems. We are used to the situation in which a human validates all the nonzero coefficient defining an optimization problem, thus there may be not too many of them. With an automatic generation, this argument is not valid.
2. The work of the Joachims shows that the solution to both the large size of the stream and the density of the data is to cleverly use some relaxations in the required accuracy. Experiments shows, some surprisingly, that so obtained solutions can be useful in practice. Even better effects (stream optimization) can be obtained by reformulating the whole optimization problem, not only the solution tolerance. Thus, further research should be directed to formally describing how a practical problem suffers from its formalization as an optimization problem being approximated.

[2]http://www.kdd.ics.uci.edu

References

[1] E. Ikonomovska, D. Gorgevik, and S. Loskovska, "A survey of stream data mining", in *Proc. 8th Nat. Conf. Int. Particip. ETAI 2007*, Ohrid, Republic of Macedonia, 2007, pp. I6-2.

[2] T. Joachims, "Training linear SVMs in linear time", in *Proc. ACM Conf. KDD 2006*, Philadelphia, USA, 2006, pp. 217–226.

[3] P. Białoń, "A linear Support Vector Machine solver for a huge number of training examples", *Control Cybern.* (to appear).

[4] D. R. Musicant, "Data mining via mathematical programing and machine learning". Ph.D. thesis, University of Wisconsin, Madison, 2000.

[5] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer, 1995.

[6] Yu. Nesterov, "Complexity estimates of some cutting plane methods based on the analytic barrier", *Math. Program.*, vol. 69, 149–176, 1995.

[7] J. Kelley, "The cutting plane method for solving convex programs", *J. Soc. Ind. Appl. Mathem.*, vol. 8, pp. 703–712, 1960.

[8] A. Bordes and L. Bottou, "The Huller: a simple and efficient online SVM", in *Machine Learning: ECML-2005, Lect. Notes Artif. Int.* Springer, pp. 505–512.

Paweł M. Białoń was born in Warsaw, Poland, in 1971. He received his M.Sc. in computer science from the Warsaw University of Technology in 1995. He is with the National Institute of Telecommunications in Warsaw. His research focusses on nonlinear optimization methods and decision support, in particular on projection methods in optimization and support vector machine problems. He has participated in several projects applying decision support in various areas: telecommunications (techno-economic analyses in developing telecommunication networks, network monitoring), also in agricultural and environmental areas.
e-mail: P.Bialon@itl.waw.pl
National Institute of Telecommunications
Szachowa st 1
04-894 Warsaw, Poland