Data Min Knowl Disc (2010) 20:53–69
DOI 10.1007/s10618-009-0149-y

Progressive refinement for support vector machines

Kiri L. Wagstaff · Michael Kocurek · Dominic Mazzoni · Benyang Tang

Received: 28 June 2007 / Accepted: 31 August 2009 / Published online: 14 October 2009
The Author(s) 2009

Abstract Support vector machines (SVMs) have good accuracy and generalization properties, but they tend to be slow to classify new examples. In contrast to previous work that aims to reduce the time required to fully classify all examples, we present a method that provides the best-possible classification given a specific amount of computational time. We construct two SVMs: a "full" SVM that is optimized for high accuracy, and an approximation SVM (via reduced-set or subset methods) that provides extremely fast, but less accurate, classifications. We apply the approximate SVM to the full data set, estimate the posterior probability that each classification is correct, and then use the full SVM to reclassify items in order of their likelihood of misclassification. Our experimental results show that this method rapidly achieves high accuracy, by selectively devoting resources (reclassification) only where needed. It

Responsible editor: Geoffrey Webb.

K. L. Wagstaff (✉) · D. Mazzoni · B. Tang
Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109, USA
e-mail: kiri.wagstaff@jpl.nasa.gov

B. Tang
e-mail: benyang.tang@jpl.nasa.gov

M. Kocurek
California Institute of Technology, 1200 E. California Blvd., Pasadena, CA 91125, USA
e-mail: mikekocurek@gmail.com

Present Address:
D. Mazzoni
Google Inc., Santa Monica, CA, USA
e-mail: dmazzoni@google.com

also provides the first such progressive SVM solution that can be applied to multiclass problems.

Keywords Support vector machines · Efficiency · Reclassification

1 Introduction

Support vector machines (SVMs) (Cortes and Vapnik 1995; Burges 1998) are a popular technique for machine learning classification. While SVMs have good accuracy and generalization properties, they can be slow to classify new examples, relative to other machine learning methods such as neural networks (e.g., DeCoste and Scholkopf 2002). An SVM must compute the dot product of each query example with each of the support vectors, which can number in the hundreds or thousands. Previous research has focused on methods to speed up SVM evaluation, sometimes at the cost of a reduction in accuracy (Burges 1996; DeCoste 2002, 2003; DeCoste and Mazzoni 2003).

In contrast, we focus on the problem of obtaining the best-possible classification in a fixed amount of time. This situation arises, for example, in real-time systems with limited computational resources. If time runs out, most existing methods would be forced to terminate with a partially classified result (see Sect. 2 for a discussion of exceptions). We propose a *progressive refinement* approach that can halt at any time with the current best-possible classification of all examples. Starting with an initial rough classification "guess" for each example, this approach progressively refines the classifications to correct errors as long as computation time is available.

The main contribution of this paper is a formulation of SVM classification as a resource-sensitive problem. This formulation permits us to combine existing methods for fast SVM approximations and SVM probabilities to obtain a straightforward solution. The resulting ProgSVM algorithm is a flexible method that adapts to available computational resources. ProgSVM offers three major advances over existing work: (1) it can be applied to binary or multiclass problems, (2) it has a strong probabilistic justification, as discussed in Sect. 4, and (3) it has very low computational overhead.

There are two important application domains that stand to benefit from this approach. The first is any interactive application in which an immediate, rough result is useful. For example, we have developed an interactive SVM training program, where the user can "paint" pixel labels onto an image, with different colors to represent different classes, and then click a "train" button to train the SVM and classify the rest of the image. After viewing the classification results, the user can iteratively refine the labels until he or she is satisfied with the SVM output. Fast feedback is essential: we have found that users would rather see the impact of their labels in an iteratively refined result, rather than waiting a fixed amount of time to get the full classification. The ProgSVM method enables the user to get fast initial results and let the SVM continue until a sufficiently accurate result is available. It is also possible to explicitly control how much computational time the SVM takes.

Real-time embedded applications can also benefit greatly from this approach. These applications are characterized by limited, and possibly variable, computational resources. For example, SVMs are currently being used for pixel classification onboard the EO-1 spacecraft (Chien et al. 2005), and they have also been developed for atmospheric composition estimation for the Mars Odyssey orbiter (Castano et al. 2007). In these environments, the ability to specify a limit on the computational cycles consumed by an SVM will greatly increase the range of situations in which they can safely be used. If the number of cycles available were known before the SVM is uploaded to the spacecraft, we could simply construct an optimal (fixed) SVM tailored to the expected resources. However, this resource constraint often is not known ahead of time and, even more critically, it could vary significantly during the course of the mission. An SVM with progressive refinement can adapt to the resources available, consistently providing the best-possible result.

1.1 Motivating example

One of the particular motivations for this research was the need to reduce the time required to classify large remote sensing data sets. We previously developed a satellite image classifier (Mazzoni et al. 2007) for the multi-angle imaging spectroradiometer (MISR) remote sensing instrument (Diner et al. 1998). This classifier labels each pixel as land, water, ice/snow, clouds, or smoke. The SVM we obtained after training on hand-labeled images had 2,469 support vectors. However, this SVM was unacceptably slow, by an order of magnitude. Given the computational resources available, classification was unable to keep up with the near-real-time data stream. Each MISR orbit generates about five million pixels that must be classified in under 90 min (when the next orbit is available). Therefore, we trained a reduced-set SVM (Burges 1996) on the same data set, yielding an SVM with just 10 support vectors (1 per pairwise binary classifier). This classifier was 247× faster than the full SVM, but it was also less accurate. We needed a compromise between the two solutions.

The method described in this paper allows us to trade off between these two extremes, using the reduced-set SVM as the inital classification and progressively correcting errors by applying the slower SVM only where needed. Figure 1a shows an example scene captured by MISR of northwestern Canada on June 30, 2004 (MISR orbit 24,123, blocks 34–36). This data set contains 196,608 pixels. The figure shows three snapshots of the progressive refinement results obtained by the pixel classifier. Here, land is brown, water is blue, ice/snow is cyan, clouds are white, and smoke is yellow. Figure 1b shows the initial "rough guess" results. It agrees with the final classification for about 87% of the pixels. Despite 13% disagreement, the overall result accurately captures the large-scale features present in the image. The largest disagreement is in the upper-left corner, where clouds (as indicated in the final classification) are incorrectly classified as snow/ice. Figure 1c is the classification obtained after 50% of the pixels have been reclassified by the full SVM (agreement has increased to 94%), and Fig. 1d is the result obtained after all pixels have been reclassified.

This example illustrates the strengths of our approach. Instead of the usual approach of batch-processing all pixels in an image, and then viewing the result only after clas-

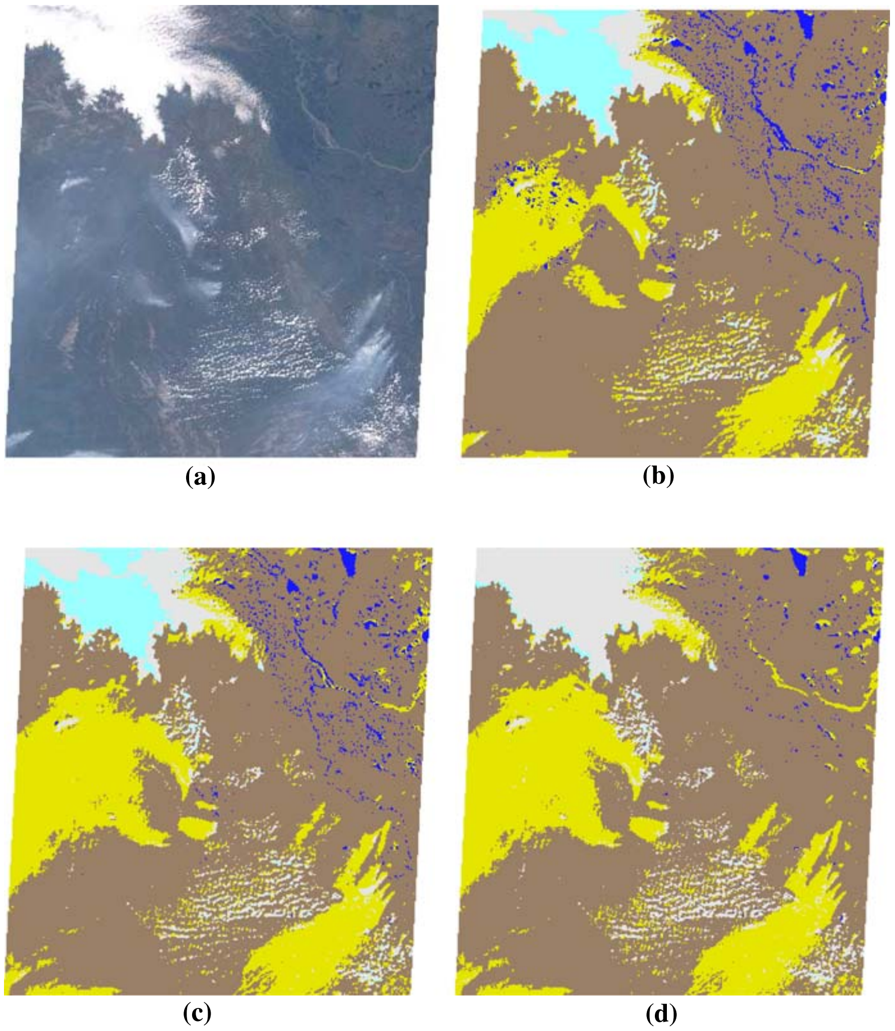

Fig. 1 A satellite image captured by the MISR instrument (**a**), showing clouds in the *upper-left corner* and several smoke plumes from fires throughout (RGB image). The remaining three images show the results of progressive refinement after 0% reclassification (**b**), 50% reclassification (**c**), and 100% reclassification (**d**). The classes are best viewed in color: land (*brown*), water (*blue*), ice/snow (*cyan*), cloud (*white*), and smoke (*yellow*). (color figure online)

sification is complete, we can obtain high-quality intermediate results at each step along the way. For very large data sets, the difference can be very striking.

The remainder of this paper is organized as follows. In Sect. 2, we describe related work and explain how our approach differs. Section 3 reviews the salient details of SVMs and two approximations (subset SVMs and reduced-set SVMs). Section 4 provides the details of our progressive refinement method. In Sect. 5, we present experimental results on several data sets. We conclude in Sect. 6 with a discussion of these results and a summary of our contributions.

2 Related work

This paper does not aim to contribute a new method for speeding up SVMs. Rather, it presents a method for selectively devoting computational resources where they are most needed and generating high-quality intermediate results during the process of classification. The novel contribution is, in fact, those intermediate results, which no other SVM method provides. However, we here review recent advances in reducing SVM classification time because they are related in motivation (trying to provide the user with information sooner) and because they provide the context for our work.

Since classification time scales with the number of support vectors used, one approach (already mentioned) is to construct a *reduced-set* SVM that approximates a given SVM with far fewer support vectors (Burges 1996). This approach reduces computational time, but often at the expense of accuracy. Further, it is necessary to specify *at training time* how many support vectors are to be used, unless the particular application permits the storage of all of the training data and iteratively expanding the reduced-set model on the fly. This is generally not true for spacecraft and embedded systems. Therefore, the resulting SVM cannot adapt to resource constraints or demands at classification time that might dictate a different desired balance between accuracy and speed.

Reduced-set vectors can also be used to selectively spend more effort on examples that are likely to belong to the positive class, such as image regions likely to contain a face for face detection applications (Romdhani et al. 2001; Ratsch et al. 2004). However, this technique is specifically tailored for the case where positive examples are rare. The ProgSVM method that we propose simultaneously considers examples from all classes, and it can be applied to multiclass problems.

Alternatively, DeCoste (2002; 2003) proposed computing bounds on the SVM's output for each new example x; once the upper and lower bounds are refined enough that they are either both positive or both negative, the classification of x is known and classification can terminate. This approach does not result in a loss of accuracy, while still providing moderate computational speedups. However, it has only been developed for binary SVMs, and the bounds computation itself can be quite expensive for large data sets, thus mitigating the speedup benefits.

The Nearest Support Vectors (NSV) method (DeCoste and Mazzoni 2003) is similar to ProgSVM in that it proposes an incremental classification process. NSV determines which SVs are most relevant to x and incrementally adds SVs until the classification of x is sufficiently confident. Thus, relatively easy examples are generally classified with only a few SVs, while harder examples may require more SVs. This approach differs from ProgSVM in that the user must still wait for all examples to be classified to get a complete result. ProgSVM provides a complete result immediately, then selectively refines only the examples that are most likely to have been assigned the wrong class. Further, NSV only applies to binary problems, and it lacks the probabilistic justification provided by ProgSVM (described in Sect. 4).

In essence, previous work has emphasized the computation of accurate classifications, resulting in incomplete classifications if time runs out. ProgSVM ensures that a complete result is always available to the user, sacrificing intermediate accuracy if necessary. ProgSVM is designed for applications in which completeness and fast responses are more important than maximal accuracy. It adapts to the available resources without any user intervention.

3 SVMs and approximations

Before describing our progressive refinement approach, we first establish our notation and provide some background on support vector machines and two kinds of approximations: subset SVMs and reduced-set SVMs. Readers who are already familiar with these methods can skip to Sect. 4.

3.1 Support vector machines

We are given a data set of n items $X = \{x_1, \ldots, x_n\}$, where each $x_i \in \mathcal{R}^d$ is represented by d features, and a vector y such that y_i is the label of x_i. For binary problems, $y_i \in \{+, -\}$. More generally, for a problem with k distinct classes, $y_i \in [1, \ldots, k]$.

Support vector machines (Cortes and Vapnik 1995) construct a hyperplane that separates two classes and therefore can only operate directly on binary problems; we will discuss multiclass learning below. A support vector machine is defined by $n + 1$ parameters: a weight α_i that is associated with each training example and a bias term, b. The classification of a new example x is obtained by computing

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i (x_i \cdot x) - b\right). \tag{1}$$

The data points with $\alpha_i > 0$, referred to as *support vectors*, are the only items that contribute to this classification. Let s be the number of support vectors. Without loss of generality we assume that these items appear first, and we obtain:

$$f(x) = \text{sign}\left(\sum_{j=1}^{s} \alpha_j y_j (x_j \cdot x) - b\right), \tag{2}$$

Therefore, the cost of computing $f(x)$ is $\mathcal{O}(sd)$.

This computation is only effective if the classes are linearly separable. If not, the dot product $(x_i \cdot x)$ can be replaced by a kernel function $K(x_i, x)$, which implicitly maps each point via some $\phi(x)$ into a feature space with more (possibly infinite) dimensions and computing the dot products there. After adding the kernel function, the SVM decision function becomes:

$$f_{SVM}(x) = \text{sign}\left(\sum_{j=1}^{s} \alpha_j y_j K(x_j, x) - b\right). \tag{3}$$

Permissible kernel functions $K(u, v)$ are those which are positive definite; common choices include the polynomial kernel, with $K(u, v) = (u \cdot v)^p$, and the Gaussian kernel, with $K(u, v) = \exp\left(-\frac{||u-v||^2}{\sigma}\right)$, where p and σ are parameters. Training a support vector machine consists of determining the values of α_i and b, usually obtained by solving the following quadratic programming problem:

$$\begin{aligned}
&\text{minimize: } \tfrac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j K(x_i, x_j) - \sum_i \alpha_i \\
&\text{subject to:} \quad 0 \le \alpha_i \le C, \ \sum_i \alpha_i y_i = 0,
\end{aligned}$$

where C is a regularization parameter that controls generalization.

Multiclass problems are handled by creating k binary SVMs such that SVM$_j$ is trained to distinguish items from class j from all other items ("one vs. all" or 1va) or by creating $\binom{k}{2}$ SVMs such that SVM$_{ij}$ is trained to distinguish class i from class j ("one vs. one" or 1v1). The decisions of the individual classifiers are combined by selecting the class with the largest SVM output (one-vs.-all) or voting (one-vs.-one).

3.2 Subset SVMs

The number of support vectors used by an SVM, s, is less than or equal to the number of items in the training set, n. Each support vector contributes to the final classification of an item x to a different degree. One way to create an approximation to the SVM is to use only a subset of these support vectors. The new SVM will be proportionally faster to apply, since classification time is proportional to the number of support vectors used.

The importance of support vector x_j is its contribution to Eq. 3, which is measured by the mean squared value of the term $\alpha_j y_j K(x_j, x_i)$ over all items x_i. Specifically, we sort the support vectors $\{x_j, j = 1, \ldots, s\}$ in decreasing order of the quantity

$$\frac{1}{n}\sum_{i=1}^{n}[(\alpha_j y_j K(x_j, x_i)]^2 = \frac{1}{n}\alpha_j^2 \sum_{i=1}^{n} K(x_j, x_i)^2.$$

For linear kernels, if the input data are scaled to have zero mean and unit standard deviation, this formula can be reduced to $\alpha_j^2|x_j|^2$.

We select only the first t support vectors, $t < s$, according to the ranking by this quantity, to create a new SVM. The new SVM cannot simply re-use the original α_i and b values, because they were determined when using all s support vectors. An additional (training) optimization pass determines the new α_i' and b' for the subset SVM. We refer to a subset SVM with t support vectors as a t-SSVM.

$$f_{SSVM}(x) = \text{sign}\left(\sum_{j=1}^{t} \alpha_j' y_j K(x_j, x) - b'\right). \tag{4}$$

3.3 Reduced-set SVMs

Another way to reduce the cost of classifying new items is to construct a reduced-set SVM that approximates a given SVM with far fewer support vectors (Burges 1996). Instead of using support vectors selected from the training set X, we construct t new vectors z_i, with weights β_i and bias term b', such that

$$f_{RSVM}(x) = \text{sign}\left(\sum_{j=1}^{t} \beta_j y_j K(z_j, x) - b'\right) \tag{5}$$

is as close to the value obtained by Eq. 3 as possible ($t \ll s$). We use the reduced-set method proposed by Tang and Mazzoni (2006), which yields more accurate approximations more efficiently than previous techniques. This reduced-set approach also directly accommodates multiclass problems by sharing reduced-set vectors between the individual binary classifiers.

In feature space, the s support vectors of an SVM encode the hyperplane normal vector $\Psi = \sum_{i=1}^{s} \alpha_i \phi(x_i)$. We first seek an approximation of the pre-image of Ψ by identifying a single vector z and weight β such that the distance between $\beta\phi(z)$ and Ψ is minimized:

$$\underset{\beta,z}{\text{argmin}}\left(\beta\phi(z) - \sum_{i=1}^{s} \alpha_i \phi(x_i)\right)^2 \tag{6}$$

By setting the partial derivative of Eq. 6 with respect to β to zero, we can solve for β and eliminate it, allowing us to find the pre-image z:

$$\underset{z}{\text{argmin}}\left(\|\Psi\|^2 - \frac{\left[\sum_{i=1}^{s} \alpha_i K(x_i, z)\right]^2}{\|\phi(z)\|^2}\right) \tag{7}$$

where

$$\|\Psi\|^2 = \sum_{i,j=1}^{s} \alpha_i \alpha_j K(x_i, x_j),$$
$$\|\phi(z)\|^2 = K(z, z).$$

This nonlinear optimization problem can be solved with a straightforward gradient descent method, but solutions can become stuck in local minima. Multiple restarts mitigate this problem. Once the best z has been computed, it is straightforward to compute β and b'. Additional reduced-set vectors can be constructed by iteratively computing the residual between $\beta\phi(z)$ and Ψ and computing the approximate pre-image of that vector, and so on. We refer to a reduced-set SVM with t support vectors as a t-RSVM. For multiclass problems, a t-RSVM consists of t reduced-set vectors *per classifier*. As above, all classifiers can share these vectors, which significantly improves accuracy.

4 Progressive refinement of SVM classifications

Our progressive refinement method requires two classifiers: SVM_Q (quick but approximate) and SVM_S (slow but accurate). In this work, SVM_S is a regular SVM (Eq. 3) and SVM_Q is an approximation. SVM_Q can be an SVM that uses only a subset of the original support vectors (Eq. 4) or a reduced-set SVM that is constructed from SVM_S (Eq. 5). For multiclass problems, we use the 1v1 method described in Sect. 3, so SVM_S consists of $\binom{k}{2}$ binary SVMs, each of which is individually approximated to construct SVM_Q. The 1v1 approach enables the use of the pairwise coupling method for estimating class membership probabilities, discussed below. Duan and Keerthi (2005) assessed several different multiclass SVM methods and found that 1v1 with pairwise coupling and Platt's method (1999) for converting SVM output into posterior probabilities yielded the best results.

Pseudo-code for ProgSVM, the progressive refinement algorithm, is given in Fig. 2. Given a data set $X = \{x_1, x_2, \ldots, x_n\}$, we first classify the entire set with SVM_Q, producing an initial classification $A = \{a_i\}$ (step 1). This initial result can immediately be used by the application. We then compute the confidence (probability of being accurate) of each classification, using methods discussed later in this section. The data points are sorted in order of increasing confidence. While time remains, we use SVM_S to reclassify the data points, starting with the ones most likely to have been incorrectly classified by SVM_Q (step 4). This selective use of SVM_S allows us to preferentially devote computational resources to the "hard" examples, since SVM_Q is likely to have already correctly classified the "easy" ones. Note that SVM_S and SVM_Q are created during training and held fixed during progressive refinement. What changes is the gradual transition between the two in terms of output classifications.

The probabilistic justification for the reclassification step lies in how we compute the confidences $C_Q(x_i)$ in step 2. Sections 4.1 and 4.2 explain how we derive the probability that x_i has been correctly classified by SVM_Q. The ProgSVM approach makes use of existing SVM approximations and methods for computing posterior probabilities. The innovative aspect of this work is the combination of these techniques to yield a resource-sensitive SVM classifier that provides the best possible result in the time available.

```
PROGSVM(data set $X$, quick $\mathrm{SVM}_Q$, slow $\mathrm{SVM}_S$, and available time $T$)
  1. Classify each $x_i \in X$ with $\mathrm{SVM}_Q$ to obtain class assignments $a_i$.
     $a_i \leftarrow \mathrm{SVM}_Q(x_i), i \in 1 \ldots n$.
  2. Compute the confidences $C_Q(x_i)$, using Equation 10 (binary) or 13 (multiclass).
     $C_Q(x_i) \leftarrow \begin{cases} p_+(x_i) & \text{Binary and } \mathrm{SVM}_Q(x) = + \\ p_-(x_i) & \text{Binary and } \mathrm{SVM}_Q(x) = - \\ \max_j\ p_j(x_i) & \text{Multiclass} \end{cases}$
  3. Sort $x_i$ according to $C_Q(x_i)$ in ascending order.
  4. Until time $T$, iteratively reclassify the data, least-confident first.
     $a_i \leftarrow SVM_S(x_i), i = 0, 1, 2 \ldots$
  5. Return $A = \{a_i\}$, the best-possible classification of $X$ in time $T$.
```

Fig. 2 ProgSVM: SVM progressive refinement algorithm

4.1 Confidence of binary SVMs

To obtain a confidence measure for the output of a binary SVM, we used Platt's method for converting SVM output into a probability Platt (1999). Platt interprets $\mathrm{SVM}_Q(x)$ as the log probability of x being a positive example, fitting a sigmoid to the outputs:

$$p_+(x) = \frac{1}{1 + \exp(A \cdot \mathrm{SVM}_Q(x) + B)}. \tag{8}$$

The A and B parameters are estimated by minimizing the negative log likelihood of the probabilities of a held-out data set, D:

$$-\sum_{x'_i \in D} t_i \log(p_+(x'_i)) + (1 - t_i)\log(1 - p_+(x'_i)). \tag{9}$$

where $t_i = \frac{y'_i+1}{2}$ for labels y'_i. Our implementation of the sigmoid fit is based on the pseudo-code provided by Lin et al. (2003). For a given example x, we obtain $p_+(x)$ and $p_-(x)$, which are the probabilities of x belonging to the positive and negative classes, respectively, $(p_+(x) + p_-(x) = 1)$. Then the confidence of SVM_Q in its classification of x is

$$C_Q(x) = \begin{cases} p_+(x) & \text{if } \mathrm{SVM}_Q(x) = + \\ p_-(x) & \text{if } \mathrm{SVM}_Q(x) = - \end{cases} \tag{10}$$

Of course, the raw output of SVM_Q could also be used to order the items for reclassification, by sorting them according to the absolute value of $\mathrm{SVM}_Q(x)$. However, the transformation from $\mathrm{SVM}_Q(x)$ to $p_+(x)$ does not necessarily yield the same ordering that $|\mathrm{SVM}_Q(x)|$ does due to the offset B. Further, the use of probabilities instead of raw outputs enables the potential use of a confidence threshold to determine when to stop reclassifying items.

4.2 Confidence of multiclass SVMs

Multiclass SVMs assign each example to one of k classes, generally by combining the outputs of several binary classifiers, as noted in Sect. 3. Hastie and Tibshirani (1998) devised a method, *pairwise coupling* (PWC), that uses a Bradley–Terry model to extend the 1v1 approach to estimate the probability that example x belongs to each of the possible classes. A 1v1 classifier estimates the conditional probability r_{ij}, which is the probability that a given example belongs to class i given that it is from class i or class j:

$$r_{ij} = P(\omega_i | \omega_i \text{ or } \omega_j), \tag{11}$$

where ω_i means "belongs to class i". PWC seeks class probability estimates that produce conditional probabilities μ_{ij} that are as close to the observed proportions r_{ij} as possible:

$$\mu_{ij} = E(r_{ij}) = \frac{p_i}{p_i + p_j}. \tag{12}$$

Duan and Keerthi (2005) showed that PWC can be further improved by using Platt's method (above) to estimate the r_{ij} values. We used this approach to obtain p_j, $1 \le j \le k$, for multiclass problems $\left(\sum_j p_j = 1\right)$. We then have

$$C_Q(x) = \max_j \; p_j(x). \tag{13}$$

4.3 Computational overhead of ProgSVM

As noted in Sect. 2, the amount of speedup obtained by SVM$_Q$ is not the only important factor when comparing our approach to that of simply using SVM$_S$. For simplicity, we will describe the costs associated with a binary classifier. The analysis can be directly extended to the multiclass case, since we use the 1v1 approach of constructing $\binom{k}{2}$ binary classifiers to perform multiclass classification.

There are two kinds of costs associated with progressive refinement: one-time costs incurred during training and query-time costs associated with each new data set.

1. Training time: Before classifying new items, we must construct SVM$_Q$. For subset SVMs, this requires computing the kernel contributions of each support vector and then re-training an SVM with the selected subset of support vectors. For reduced-set SVMs, the additional cost incurred is the calculation of the reduced-set vectors. In addition, for binary problems (and for each multiclass SVM binary sub-problem) we must estimate the A and B parameters that will be used to compute the confidence of SVM$_Q$'s output. They are estimated using an iterative gradient descent method.
2. Query time: The cost of classifying each new item is $\mathcal{O}(sK)$, where s is the number of support vectors used and K is cost of computing the kernel function between two items (which is $O(d)$ for linear and Gaussian kernels). For approximations with only a single subset or reduced-set vector, this is simply $\mathcal{O}(K)$. The confidence values are sorted in $\mathcal{O}(n \log n)$ time.

5 Experimental results

5.1 Data and methodology

We performed experiments on seven well known data sets from the UCI machine learning repository (Newman et al. 1998). Two data sets are binary and five are multiclass problems. Table 1 summarizes the number of data points used for training (n_{tr}), data points used for testing (n_{te}), features (d), and classes (k) for each data set. The ABE data set contains only the letters A, B, and E from the "letter" data set. The items used for training and testing on the binary problems were randomly selected from the full data sets. For the multiclass problems, we used the train/test

Table 1 Binary and multiclass data sets

Data set	n_{tr}	n_{te}	d	k
Twonorm	1,000	4,000	20	2
Adult	1,000	4,000	123	2
ABE ({A, B, E} from letter)	1,120	1,203	16	3
DNA	1,000	2,186	180	3
WAV (wavform)	600	4,400	21	3
SAT (satimage)	2,000	4,435	36	6
SEG (segment)	1,000	1,310	18	7

splits provided by Duan and Keerthi (2005) (obtained by stratified sampling of the full data sets), who first suggested using PWC in conjunction with Platt's method for estimating probabilities.

We randomly split each data set into training and test sets 20 times, and for each of the 20 splits, we computed the optimal SVM hyperparameters for a Gaussian kernel (the coefficient σ and the SVM regularization parameter C) using five way cross-validation, following the method described by Duan and Keerthi (2005). For each training/test split of each problem, we constructed SVM$_S$ and a series of SVM$_Q$s using an increasing number of support vectors, using the same hyperparameters for all SVMs.

We created both subset and reduced-set ProgSVMs for the binary data sets. For a fixed number of support vectors used, we expected the reduced-set SVM to provide a more accurate approximation than a subset SVM of the same size. However, we report only reduced-set ProgSVM results for the multiclass data sets. Even with multiple binary classifiers contributing to a multiclass decision, it is possible to specify the total number of reduced-set vectors to be created, and to permit the individual binary classifiers to all make use of them. Multiclass, 1v1, subset ProgSVMs do not have such a straightforward interpretation. Each binary classifier in the multiclass SVM can only select support vectors from those items assigned to class i or j, and therefore only limited sharing would be possible. It is also not clear how to determine which binary classifier should be permitted to select the support vectors, given a cap on the total number used.

For binary problems, we used five-fold cross-validation to determine the optimal Platt parameters (A and B) for SVM$_Q$, as described in Sect. 4.1. For multiclass problems, SVM$_S$ and SVM$_Q$ were composed of 1v1 SVMs, and we used the PWC method referred to in Sect. 4.2 to compute confidences. The results we report are averages over 20 trials, one on each split of the data.

5.2 Binary classification results

We first evaluated ProgSVM's performance when using the simplest possible, and therefore most efficient, SVM$_Q$. Figure 3 shows test set accuracy as a function of time for the two binary data sets. Results are shown for ProgSVM using a 1-RSVM (solid lines) and ProgSVM using a 1-SSVM (dashed lines) for SVM$_Q$. A certain amount

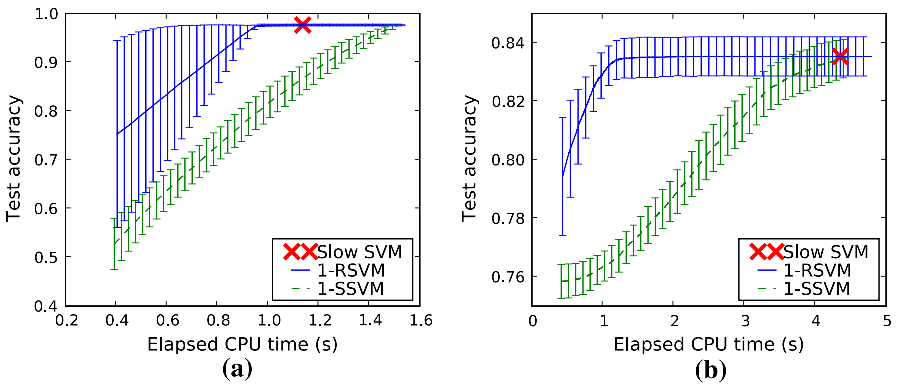

Fig. 3 ProgSVM performance curves for two binary data sets, showing accuracy obtained when SVM_Q is a 1-RSVM or a 1-SSVM. All results are averages over 20-fold cross-validation, with *bars* showing one standard deviation. **a** Twonorm. **b** Adult

of initial startup time is required to fully classify all of the test items with SVM_Q. The performance curves begin when all items have been classified and it is possible to report an overall test set accuracy. The time required by SVM_S to classify all items from scratch is shown with an X.

In all cases, as items were reclassified by SVM_S, accuracy improved. For both data sets, full accuracy (equivalent to that of SVM_S alone) was achieved by ProgSVM using a 1-RSVM well before SVM_S alone did. In contrast, the approximation provided by the 1-SSVM was generally of lower quality than that of the 1-RSVM, so more reclassification was needed. In fact, for the twonorm data set, full accuracy was not achieved until 1.5 s had elapsed, even though the slow SVM on its own achieved this accuracy after only 1.2 s. As we would expect, the single reduced-set support vector is more useful than a single support vector from the training set.

In addition, all ProgSVMs (reduced-set or subset) provide a rich array of intermediate results that are always complete. In some realtime applications, this may be of more value than raw accuracy. By contrast, the slow SVM does not yield a complete result until the time indicated by the X.

We also tested ProgSVM performance when using subset or reduced-set SVMs with more than one support vector. Figure 4 shows the tradeoff between the time until the first complete result was available (x-axis) and the time at which full accuracy (equal to that of SVM_S) was achieved (y-axis). The time at which the full result is available from SVM_S alone (classifying every item with SVM_S) is marked with an X. Any results below and to the left of the X are runs in which ProgSVM achieved the same accuracy as SVM_S in less time. The diagonal line marks points in trade space where the first complete result already has the desired final accuracy, so no reclassification is needed, i.e., SVM_Q is as good as SVM_S. (Note that this is only desirable if the time required to achieve that result is less than that required by SVM_S alone.) Results are plotted for a variety of t-SSVMs (circles) and t-RSVMs (diamonds), where t begins at 1 and increases by 10 for each new ProgSVM.

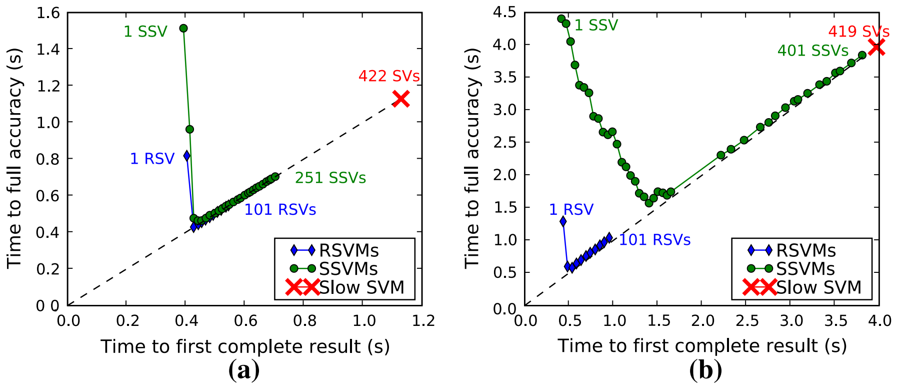

Fig. 4 Binary ProgSVM tradeoff curves: elpased time to the first complete result versus total time required to achieve full accuracy (same as that achieved by SVM$_S$). A perfect approximation (SVM$_Q$) would lie on the *diagonal line*. All results with a y-axis value less than that of the slow SVM indicate that ProgSVM achieved the same performance, but in less time. All results are averages over 20-fold cross-validation. **a** Twonorm. **b** Adult

These results show that if more support vectors are used by SVM$_Q$, even better gains can be achieved. Figure 3 showed that a 1-RSVM provided a good initial approximation for both data sets, enabling ProgSVM to reach full accuracy well before the slow SVM alone would have reached it. However, Fig. 4 shows that a 11-RSVM achieved full accuracy without any reclassification, for both data sets. The time needed was only that required by the 11-RSVM to classify all of the test data (0.4 s for Twonorm and 0.5 s for Adult), compared to the time required by SVM$_S$ (1.1 and 4.0 s, respectively). Further, although the ProgSVMs using 1-SSVMs in Fig. 3 did not provide full accuracy faster than SVM$_S$, Fig. 3 shows that an SSVM with 21–31 support vectors (for Twonorm) or ~200 support vectors (for Adult) provided classifications that were just as accurate as those of SVM$_S$, in a fraction of the time.

We observed an inflection behavior for both data sets. As we increased the number of support vectors used by a t-SSVM or t-RSVM, the accuracy of the approximation generally improved. However, the complexity also increased because more support vectors were involved in each classification, and therefore the time elapsed until a complete result was available also increased. Initially, increasing the number of support vectors improved starting accuracy enough that the final accuracy was reached much more quickly, because fewer reclassifications were needed. However, once SVM$_Q$ was accurate enough, the addition of more support vectors only served to increase the time needed to achieve an initial, complete result. The "sweet spot" in between provides a way to minimize total classification time and provides significant speedups over the slow SVM.

However, the benefits of ProgSVM go beyond the ability to reach full accuracy in less time. As originally stated in the Introduction, there are applications for which a complete result, even if it is not fully accurate, has value to the user. The fact that ProgSVM provides a complete result at all times (after the time required by SVM$_Q$ to fully classify the data) is a second, significant benefit.

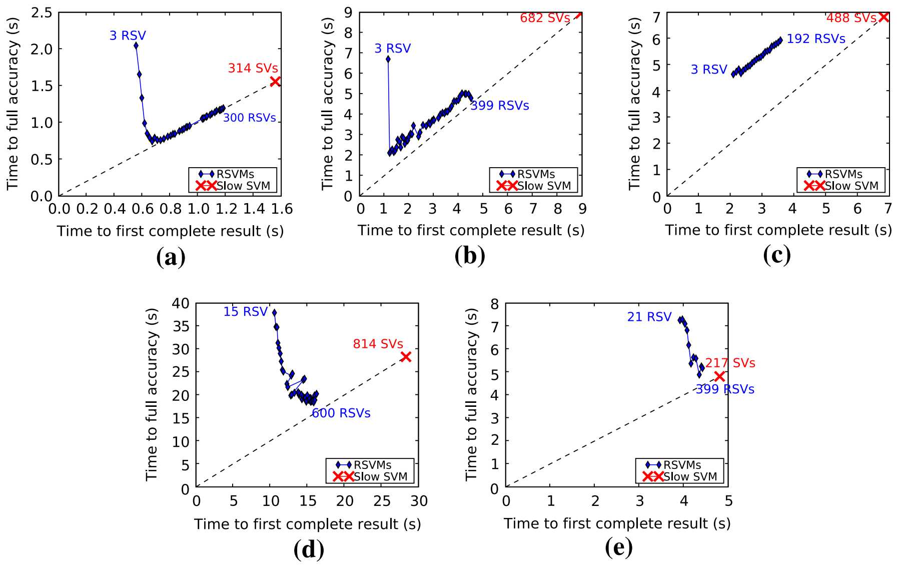

Fig. 5 Multiclass ProgSVM tradeoff curves: elapsed time to the first complete result versus total time required to achieve full accuracy (same as that achieved by SVM$_S$). A perfect approximation (SVM$_Q$) would lie on the *diagonal line*. All results with a y-axis value less than that of the slow SVM indicate that ProgSVM achieved the same performance, but in less time. All results are averages over 20-fold cross-validation. **a** ABE ($k = 3$). **b** DNA ($k = 3$). **c** WAV ($k = 3$). **d** SAT ($k = 6$). **e** SEG ($k = 7$)

5.3 Multiclass classification results

Figure 5 shows the results of our experiments on the five multiclass problems. As explained above, subset SVMs are not well defined for multiclass problems, so we report only reduced-set SVM results.

These experiments yield two additional insights into the use of ProgSVM. First, it is not always possible for SVM$_Q$ to exactly match the performance of SVM$_S$; sometimes all of the support vectors are needed (e.g., WAV data set). However, this is not really a problem; we still see computational gains across the board, with full accuracy achieved by ProgSVM before SVM$_S$. Second, as the number of classes increases, the computational complexity even of a simple 1-RSVM increases significantly (e.g., SAT and SEG). Since we use a 1v1 approach to multiclass classification, the time required to classify a new item with a t-RSVM increases significantly with k, so the time until a complete result is available may still be quite large even for SVM$_Q$. For that reason, this technique is probably best suited to problems with fewer than 10 classes. Further investigation would be needed to more precisely characterize the sensitivity to k.

6 Conclusions and future work

In this paper, we have presented a new formulation of SVM classification as a resource-sensitive problem. We proposed a progressive refinement solution with a straightforward algorithm, ProgSVM, that quickly classifies a data set with a fast but less accurate SVM and then selectively reclassifies items with a slower but more accurate SVM. This approach provides the best-possible classification of the entire data set at any intermediate step, as opposed to existing methods that perform a full classification of each item and may only achieve a partial result in the same period. Despite its simplicity, this method is very effective, as demonstrated on seven benchmark data sets. The experimental results indicate that ProgSVM provides useful intermediate *complete* classification results and often achieves the same accuracy as the slow SVM alone, in much less time.

Interactive and real-time systems are two applications that stand to benefit greatly from this approach. We have already used progressive refinement to greatly improve the interactivity of an application that permits users to classify large images. ProgSVM is unique in that it has a probabilistic foundation for how it identifies which examples to reclassify. In addition, ProgSVM is the first SVM solution for incremental classification that can be applied to multiclass problems. Finally, the same progressive refinement approach could be used with other base classifiers, so long as they provide a posterior confidence and there is a method of computing a fast approximation of a given model, akin to the subset and reduced-set SVMs used in this work.

The question of how to choose the best t (number of support vectors in SVM_Q) is an open problem. For binary classification problems, we found dramatic runtime improvements using ProgSVM across a wide range of t values, suggesting that the benefits of ProgSVM are not very sensitive to t. For multiclass problems, our results show that it is worth spending more effort on selecting an appropriate t. For a given data set, a model selection process in which a variety of t values are tested can be conducted to determine the optimal t with respect to runtime.

Although the approximate SVM_Q generally has lower average performance than SVM_S, it is possible for it to correctly classify an individual item that SVM_S incorrectly classifies. Reclassifying these items actually decreases performance, rather than increasing it. An example of this phenomenon can be noted in the example shown in Fig. 1. There is a river in the upper right portion of the image that is marked as water (blue) by the initial SVM_Q. However, those pixels are reclassified (incorrectly) by SVM_S into the smoke class. Of course, it would be desirable to refrain from correcting cases where SVM_Q is already correct. An interesting avenue for future exploration would be to only permit reclassification of x when $C_Q(x) \leq c$, where c is a confidence threshold. This would exploit the fact that not all items *should* be reclassified, especially if SVM_S is not very reliable. This classifier-ensemble approach could well obtain a final performance that is higher than that of SVM_Q or SVM_S alone, by leveraging their individual strengths.

Acknowledgments We wish to thank Dennis DeCoste, Robert Granat, and the anonymous reviewers for their helpful comments and suggestions. The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. We gratefully acknowledge the support of a grant from the NASA Advanced Information Systems Technology program and grant #IIS-0705681 from the National Science Foundation. The MISR remote-sensing data was obtained from the Langley Atmospheric Science Data Center.

References

Burges CJC (1996) Simplified support vector decision rules. In: Proceedings of the thirteenth international conference on machine learning, pp 71–77

Burges CJC (1998) A tutorial on support vector machines for pattern recognition. Data Min Knowl Discov 2(2):121–167

Castano R, Wagstaff KL, Chien S, Stough TM, Tang B (2007) On-board analysis of uncalibrated data for a spacecraft at Mars. In: Proceedings of the thirteenth international conference on knowledge discovery and data mining, pp 922–930

Chien S, Sherwood R, Tran D, Cichy B, Rabideau G, Castaño R, Davies A, Mandel D, Frye S, Trout B, Shulman S, Boyer D (2005) Using autonomy flight software to improve science return on Earth observing one. J Aerosp Comput Inf Commun 2(4):196–216

Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20:273–297

DeCoste D (2002) Anytime interval-valued outputs for kernel machines: fast support vector machine classification via distance geometry. In: Proceedings of the nineteenth international conference on machine learning, pp 99–106

DeCoste D (2003) Anytime query-tuned kernel machines via Cholesky factorization. In: Proceedings of the SIAM international conference on data mining (SIAMDM-03)

DeCoste D, Mazzoni D (2003) Fast query-optimized kernel machine classification via incremental approximate nearest support vectors. In: Proceedings of the twentieth international conference on machine learning, pp 115–122

DeCoste D, Scholkopf B (2002) Training invariant support vector machines. Mach Learn 26(1–3):161–190

Diner DJ, Beckert JC, Reilly TH, Bruegge CJ, Conel JE, Kahn R, Martonchik JV, Ackerman TP, Gordon HR, Muller J-P, Myneni R, Sellers RJ, Pinty B, Verstraete MM (1998) Multiangle imaging spectroradiometer (MISR) instrument description and experiment overview. IEEE Trans Geosci Remote Sens 36:1072–1087

Duan K, Keerthi SS (2005) Which is the best multiclass SVM method? An empirical study. In: Proceedings of multiple classifier systems, pp 278–285

Hastie T, Tibshirani R (1998) Classification by pairwise coupling. Ann Appl Stat 26(2):451–471

Lin HT, Lin CJ, Weng RC (2003) A note on Platt's probabilistic outputs for support vector machines. Technical report, National Taiwan University

Mazzoni D, Garay MJ, Davies R, Nelson D (2007) An operational MISR pixel classifier using support vector machines. Remote Sens Environ 107(1–2):149–158

Newman DJ, Hettich S, Blake CL, Merz CJ (1998) UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html

Platt JC (1999) Probabilities for SV machines. In: Smola AJ, Bartlett P, Schölkopf B, Schuurmans D (eds) Advances in large margin classifiers. MIT Press, Cambridge, pp 61–74

Ratsch M, Romdhani S, Vetter T (2004) Efficient face detection by a cascaded support vector machine using Haar-like features. In: Proceedings of the German pattern recognition symposium, pp 62–70

Romdhani S, Torr PHS, Schölkopf B, Blake A (2001) Computationally efficient face detection. In: Proceedings of the eighth international conference on computer vision, pp 695–700

Tang B, Mazzoni D (2006) Multiclass reduced-set support vector machines. In: Proceedings of the twenty-third international conference on machine learning, pp 921–928