Support Vector Machines for Large Scale Text Mining in R

Ingo Feinerer\textsuperscript{1} and Alexandros Karatzoglou\textsuperscript{2}

\textsuperscript{1} Database and Artificial Intelligence Group
Institute of Information Systems
Vienna University of Technology
Austria Ingo.Feinerer@tuwien.ac.at

\textsuperscript{2} LITIS, INSA de Rouen
Avenue de Universite
76801 Saint-Etienne du Rouvray
France alexis@ci.tuwien.ac.at

Abstract. SVM are an established tool in machine learning and data analysis. Though many implementations of SVM exist often specific applications require tailor made algorithms. In text mining in particular the data often comes in large sparse data matrices. Typical SVM algorithms like SMO do not take advantage of the sparsity, and do not scale well to data sets with millions of entries. In this paper we present an implementation of linear SVM’s for \textit{R} that address both of these issues.

Keywords: SVM, text mining, large scale

1 Introduction

Many applications in machine learning and data mining require the classification of massive amounts of data like e.g. spam filtering or text topic identification. As \textit{R} is becoming the tool of choice for data intensive operations in both research and industry it is vital that basic learning tasks can be performed on large scale data sets. In particular in the area of text mining which has seen some significant research activity due to the importance of the web, one has to usually deal with data that comes in the form of large sparse matrices as produced by the term-document extraction process.

Term-document matrices which essentially represent each document by the frequency of the occurring words can be easily produced in \textit{R} due to the availability of modern text processing and mining functionality provided by the \texttt{tm} package. There are though not many methods that can utilize large scale sparse matrices and in particular there are no support vector machines (SVM) implementations (Karatzoglou et al. (2006)) that can easily deal with large scale data sets. SVMs are central in many text mining tasks like topic assignment and filtering and thus an SVM implementation that is tailored for large scale sparse data matrices is particularly important.
Current support vector machine implementations in R support sparse matrix formats as inputs but the algorithms that are used in the optimization task do not take advantage of the sparsity in that data. Moreover all SVM implementations in R solve the dual optimization problem utilizing algorithms like SMO (Platt (1999)) which are known to scale super-linear with the size of the data.

In this paper we introduce a new implementation of SVMs in R which is tailored for large scale sparse data. The implementation is based on the algorithms introduced by Keerthi and DeCoste (2005) and by Sindhwani and Keerthi (2007), and in addition adds multi-class functionality and handling of all sparse matrix formats in R. Linear learning techniques are often the tool of choice when massive amounts of data are available.

2 Large scale linear support vector machines

Many SVM variants have been developed over the years. One that is particularly well suited for large scale sparse data is the $l_2$-SVM (Keerthi and DeCoste (2005)). Given $m$ binary labeled examples $\{x_i, y_i\}$ with $i \in 1, \ldots, m$ and $y_i \in \{+1, -1\}$ the $l_2$-SVM optimization problem can be written in its Lagrangian form as

$$w^* = \arg\min_{w \in \mathbb{R}} f(w) = \frac{1}{2} \sum_{i=1}^{m} c_i l_2(y_i w^T x_i) + \frac{\lambda}{2} \|w\|^2$$

(1)

where $l_2$ is the $l_2$-SVM loss given by $l_2(z) = \max(0, 1 - z)^2$, $\lambda$ is the regularization parameter, and $c_i$ is the relative importance or weight for each data point. The decision function is then given by $f = \text{sign}(w^* x)$. The difference between a standard SVM and this formulation is in the loss function. A standard SVM utilizes the well known hinge loss $l(z) = \max(0, 1 - z)$ while the $l_2$-SVM uses a squared hinge loss. This seemingly trivial difference has a big impact on the optimization procedure one can use in order to find the optimal $w^*$ for Equation 1. The main difference between the hinge loss and its square form is that the squared form is differentiable in any point $z$. Note also that the $l_2$-SVM does not include an explicit bias term as in the standard SVM formulation but the bias term is included in $w^*$ and computed with the addition of an intercept term in $x$. This in effect forces a regularization on the bias. Moreover the $l_2$-SVM preserves the large margin characteristics that come with the theoretical guarantees for good generalization properties of the standard SVM.

Note here that Equation 1 is a strictly convex differentiable and a piece-wise quadratic function of $w$. A very effective optimization procedure in order to find the single optimal $w^*$ was used by Keerthi and DeCoste (2005) and is based on the Newton method. In the Newton method a second order approximation of the objective function is used to iteratively update the $w$ parameter.
vector, that is
\[ w^{t+1} = w^t - \eta^t [\nabla^2 f(w^t)]^{-1} \nabla f(w^t) \] (2)
where \( \eta^t \) is the step size parameter, \( \nabla f(w^t) \) is the gradient vector and \( \nabla^2 f(w^t) \) is the Hessian matrix. The Hessian does not exist for every \( w \) since \( f(w) \) is not twice differentiable for all \( w \) and thus a generalized approximation of the Hessian is used. The step direction is then given by first solving a least squares problem over a subset \( s(w^t) \):
\[
\left[ \lambda I + X_s^T C_s X_s \right] \bar{w}^t = X_s^T C_s Y_s
\] (3)
where \( I \) is the identity matrix. Given \( \bar{w}^t \) the direction of the update is then given by \( w^{t+1} = w^t + \eta^t (\bar{w}^t - w^t) \) and the optimal step size \( \eta^t \) is found by a simple line-search on the optimization problem \( \eta^t = \arg\min_\eta f(w^t + \eta (\bar{w}^t - w^t)). \)

The least squared optimization problem 3 is solved using a conjugate gradient method that performs particularly well when \( X \) is sparse (Keerthi and DeCoste (2005)).

The whole optimization procedure can be performed very fast and the algorithm for solving Equation 1 scales linear to the number of non-zero entries in the data matrix. Note also here that in contrast to many conventional SVM algorithms all optimization computations are performed in primal space.

3 \texttt{svmlin} R package

The \( l_2 \)-SVM algorithm is implemented in the \texttt{svmlin} R package. It is licensed under the GPL and is available via the CRAN archive. The implementation extends the original C++ version of \texttt{svmlin} provided by Sindhwani and Keerthi (2007) and introduces multi-class classification, cross validation and handling of a range of sparse matrix formats. The \texttt{svmlin} R package supports all available standard sparse matrix formats provided by the \texttt{SparseM}, \texttt{Matrix}, and \texttt{slam} packages. The function implementing the algorithm boost a formula interface along with a default interface and the data matrix has to be in sparse matrix format or in term-document matrix format as supported by the \texttt{tm} package. A call to the \texttt{svmlin} function is done simply by

\[
\texttt{svmodel} \leftarrow \texttt{svmlin(matrix, labels, lambda = 0.1, cross = 3)}
\]

where the resulting object contains the weights parameter vector \( w \), the offset term along with the training and the 3-fold cross-validation error. In this call we set the regularization parameter value of \( \lambda = 0.1 \). The resulting object can be used with the \texttt{predict} function along with test data. The \texttt{svmlin} R package provides multi-class classification functionality by implementing two popular voting schemes: the one-against-one and the one-against-all.

The one-against-one or pairwise classification method (Knerr et al. (1990); Kreßel (1999)) constructs \( \binom{k}{2} \) classifiers (\( k \) as the number of classes) where
each one is trained on data from two classes. Prediction is done by voting
where each classifier gives a prediction and the class which is predicted more
often wins ("Max Wins"). This method has been shown to produce robust
results when used with SVMs (Hsu and Lin (2002)).

The one-against-all method is a somewhat simpler approach where \( k \)
classifiers are constructed that always separate one class from the rest. The \( i \)-th
SVM classifier is trained with all the examples of the \( i \)-th class with positive
labels and all other with negative ones. At the classification phase a sample
\( x \) is assigned to class \( i \) when the decision value of the classifier for class \( i \), \( f_i \),
produces the largest value.

4 \textit{tm} \textit{R} package

The \textit{tm} (Feinerer (2010); Feinerer et al. (2008)) package provides a framework
for text mining applications in \textit{R}. It offers functionality for managing corpora
and text documents, abstracts the process of document manipulation and
eases the usage of heterogeneous text formats in \textit{R}. An advanced meta data
management is implemented for collections of text documents to alleviate
the usage of large and meta data enriched document sets. \textit{tm} provides easy
access to preprocessing mechanisms such as stemming, stopword exclusion, or
removal of punctuation marks. Out of the box \textit{tm} also includes functionality
for processing the Reuters-21578 data sets in native XML, and for processing
the e-mail messages (including meta data in headers) for the SpamAssassin
and 20 newsgroups data sets. In addition \textit{tm} can construct and export term-
document matrices in a sparse representation from corpora.

5 Experiments

5.1 Data

Our primary research data set is the Reuters-21578 data set (Lewis (1997))
containing stories collected by the Reuters news agency in 1987. The data
set is publicly available and has been widely used in text mining research
within the last decade. It contains 21578 short to medium length documents
in XML format covering a wide range of topics, like mergers and acquisitions,
finance, or politics.

Our second data set is the SpamAssassin public mail corpus (\url{http://spamassassin.apache.org/publiccorpus/}). It is freely available and offers
authentic e-mail communication with classifications into normal (ham) and
unsolicited (spam) mail of various difficulty levels (easy ham, hard ham, and
spam). In total we have 4150 ham documents and 1896 spam documents.

Our final data set is the 20 newsgroups text collection (\url{http://kdd.ics.
uci.edu/databases/20newsgroups/}). It consists of 19997 e-mail messages
taken from 20 different newsgroups (however cross posting was allowed) and is
publicly available due to a donation by Tom Mitchell. The newsgroups cover a wide field of unique topics dealing e.g. with atheism, computer graphics, motorcycles, or politics in the middle east.

5.2 Protocol

The main aim of the experiments is to illustrate the significant speedups obtained by the use of the svmlin package in particular compared to a standard SVM implementation like the function svm in package e1071. To this end we use the data sets to train with both SVM implementations. For the e1071 implementation we make sure to use the linear kernel and to set the cost parameter to \( \frac{1}{\lambda} \) to get equivalent models.

In order to compare the scaling behavior we sample from our data sets first \( \frac{1}{10} \) of the data for training and increase the training data amount by \( \frac{1}{10} \) before training again up to the whole data set. We repeat this procedure for both implementations. We also compare the classification performance of the implementations on the data sets using 10-fold cross-validation. We tune both models for the regularization and cost parameters.

The creation of the sparse term-document matrices took about 42 seconds for the Reuters-21578 XML data set, about 31 seconds for the SpamAssassin data set, and about 75 seconds for the 20 newsgroups data set. The Reuters-21578 term-document matrix has 65973 terms, 21578 documents, and a size of about 24 MB in memory, the SpamAssassin matrix has 151029 terms, 6046 documents, and is about 24 MB big, whereas the 20 newsgroups matrix has 175685 terms, 19997 documents, and a memory footprint of about 46 MB.

5.3 Results

The results obtained are presented in the following plots. In Figures 1 and 2 we observe that although for very small portions of the 20 newsgroup data the e1071 implementation is faster as the data size increases svmlin performs better. This highlights the better scaling behavior of the svmlin implementation. Note also that most of the data handling and splitting is done in R in the svmlin implementation and thus represents an overhead compared to the e1071 method where almost all data splittings necessary for the one-against-one voting scheme are done in C++.

In Figure 3 the difference in training time is much clearer since this is also the largest data set in our experiment setup. svmlin is significantly faster than the e1071 implementation. Similarly in Figure 1 which is a relatively small binary classification data set the svmlin implementation has a faster training time than e1071.

We also compared the results of the two implementations in terms of classification performance with 10-fold cross-validation and found no significant performance advantages for either implementation.
Fig. 1. CPU training time for svmlin and svm (from e1071) for different portions of the SpamAssassin data set starting with 1/10 of the data up to the whole data set. The svmlin implementation is faster for almost all configurations.

Fig. 2. CPU training time for svmlin and svm (from e1071) for different portions of the 20 newsgroups data starting with 1/10 of the data up to the whole data set. The e1071 implementation scales super-linearly and is outperformed for larger portions of the data.
Fig. 3. CPU training time for `svmlin` and `svm` (from `e1071`) for different portions of the Reuters 21578 data starting with 1/10 of the data up to the whole data set. The `e1071` implementation is slower for almost all configurations.

6 Conclusion

We presented the `svmlin` SVM implementation for large scale text mining tasks. The introduced implementation takes advantage of the sparsity in the data to accelerate the optimization process. The computations are done in primal space thus no kernel is used. This is no problem in text mining tasks where linear methods have been shown to produce excellent results. We demonstrated the usefulness of the new implementation and the advantages it provides compared to previous implementations: in particular the linear scaling with the data size and the faster training time on larger data sets.
Bibliography


