Abstract: Efficient Kernel Methods for Graphs

Thomas Gärtner

Fraunhofer IAIS, Schloß Birlinghoven, 53754 Sankt Augustin, Germany
thomas.gaertner@iais.fraunhofer.de

The computational cost of applying kernel methods to graphs is typically dominated by the cost of computing the kernel matrix. This contrasts with the more common setting where the kernel between two instances can be computed individually and cheaply. This holds for example in text classification and other domains where the instances are sparse or dense vectors in some Euclidean space. Several toolkits for this case have been developed and are widely used.

The approaches to cope with the size of the graphs vary and real-world applications so far have suffered from a lack of clear guidance which technique to use in which situation. In this work we survey possible techniques and apply them in one setting. In particular we concentrate on regularised least squares regression and support vector machines with powerseries kernels $K = \sum_i^\infty \lambda_i B^i$ where B is a sparse matrix representing the structure of the graph such as the (normalised) Laplacian or the adjacency matrix. Using of-the-shelf iterative solvers we obtain highly scalable implementations for these settings that exploit the sparsity of the graphs.

We distinguish techniques for general kernel matrices of power series form and methods that are only applicable to kernel matrices defined by geometric power series ($\lambda_i = \gamma^i$). While the first kernel matrix to be proposed—the "diffusion kernel"—was defined using the exponential powerseries ($\lambda_i = \beta^i/i!$), the geometric powerseries appears more popular nowadays. The main difference is that for a closed form computation of the general power series we need to compute the eigenvalues and apply an anti-monotone transformation. For geometric powerseries, we can make use of the fact that the closed form is the inverse of a sparse matrix, e.g., $K = J^{-1} = (L + \gamma\mathbf{I})^{-1}$. The iterative methods we propose have time complexity linear in the number of edges of the graph on (a particular kind of) random graphs constructed with a fixed number of dense regions.

1 Iterative Solvers for Kernel Methods

In this section we derive the optimisation problems for regularised least squares regression and support vector machines that allow efficient solutions on graphs. Possible techniques for efficient geometric powerseries kernel methods are:

(*i*) Reparameterise the learning algorithm using $t = Kv$ such that the regulariser $v^\top Kv$ can be computed as $t^\top Jt$. This technique has been used e.g. in [1].
(*ii*) Replace matrix vector products $x = Kv$ by solving the sparse systems $Jx = v$. This technique has been used e.g. in [2].

Possible techniques for efficient arbitrary power series kernel methods are:

(iii) Compute the eigendecomposition of a sparse matrix and compute the limit of the powerseries on the eigenvalues. As the transformation achieved by the powerseries is typically anti-monotone, we can speedup the algorithm by only computing a few small eigenvalues of the sparse matrix (which will dominate the kernel matrix after the anti-monotone transformation).

(iv) Resort to low-order approximations to the powerseries but never compute the kernel matrix explicitly. This means computing Kv as $\sum_{i=0}^{d} \lambda_i v^{(i)}$ where $v^{(0)} = v$ and $v^{(i)} = Bv^{(i-1)}$.

Note that for *(ii)*-*(iv)* multiplications with a block of the matrix are also possible. For instance, computing on the labelled block (denoted by a subscript **l**) $w = K_{\mathbf{ll}} v_{\mathbf{l}}$ is $w = \left[K(v_{\mathbf{l}}\ \mathbf{0})^\top\right]_{\mathbf{l}}$.

The general form of kernel methods that we are interested in is to minimise the function $Q(f) = \sum_{i=1}^{n} V(f(x_i), y_i) + \nu \left\| f(\cdot) \right\|^2$where $V : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a convex loss function dominating the 0/1-loss, $f(\cdot)$ chosen from a Hilbert space with reproducing kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $\nu \geq 0$ is a parameter. The usual next step is to make use of the representer theorem to obtain the equivalent

$$c_{\mathbf{l}}^* = \operatorname*{argmin}_{c_{\mathbf{l}} \in \mathbb{R}^n} \sum_{i=1}^{n} V(K_{i\mathbf{l}} c_{\mathbf{l}}, y_i) + \nu c_{\mathbf{l}}^\top K_{\mathbf{ll}} c_{\mathbf{l}} \ .$$

For strategies *(ii)*-*(iv)* the only difference is that we need to take care that the optimisation problem obtained permits efficient application of the chosen iterative solver. We will discuss this in some more detail below.

For strategy *(i)* we substitute $t = Kv$ and obtain the optimisation problem

$$t^* = \operatorname*{argmin}_{t \in \mathbb{R}^{n+m}} \sum_{i=1}^{n} V(t_i, y_i) + \nu t^\top J t$$

Now, as above, we need to take care that the optimisation problem obtained permits efficient application of the chosen solver.

1.1 Regularised Least Squares Regression

For strategies *(ii)*-*(iv)* we can employ (Newton) conjugate gradient (CG) with

$$\begin{aligned} Q(c_{\mathbf{l}}) &= \left\| y - K_{\mathbf{ll}} c_{\mathbf{l}} \right\|^2 + \nu c_{\mathbf{l}}^\top K_{\mathbf{ll}} c_{\mathbf{l}} \\ \nabla_{c_{\mathbf{l}}} Q(c_{\mathbf{l}}) &= 2K_{\mathbf{ll}}^2 c_{\mathbf{l}} - 2K_{\mathbf{ll}} y + 2\nu K_{ll} c_{\mathbf{l}}; \quad H = 2K_{\mathbf{ll}}^2 + 2\nu K_{\mathbf{ll}} \ . \end{aligned}$$

For strategy *(i)* we can employ (Newton) CG with

$$\begin{aligned} Q(t) &= \left\| y - t_l \right\|^2 + \nu t^\top J t \\ \nabla_t Q(t) &= 2 \begin{pmatrix} t_l \\ 0 \end{pmatrix} - 2 \begin{pmatrix} y \\ 0 \end{pmatrix} + 2\nu J t; \quad H = \begin{pmatrix} J_{\mathbf{ll}} + \mathbf{I}_{\mathbf{ll}} & J_{\mathbf{lu}} \\ J_{\mathbf{ul}} & J_{\mathbf{uu}} \end{pmatrix} \end{aligned}$$

or we can use $t^* = \mathrm{solve}_t Ht = (y, 0)^\top$ to obtain t^* directly. Last but not least, for the (normalised) Laplacian kernel we can also simply approximate t^* as

$$t^* \approx \frac{1}{1+\nu} \sum_{i=0}^{d} \left(\frac{-\nu\gamma}{1+\nu} L \right)^i \begin{pmatrix} y \\ 0 \end{pmatrix} \ .$$

1.2 Support Vector Machines

For brevity we focus on support vector machines with squared hinge loss only. Here we face a problem different from regularised least squares as inequality constraints are used to implement hinge loss:

$$\min_{c_\mathfrak{l}, \xi \in \mathbb{R}^n} \|\xi\|^2 + \nu c_\mathfrak{l}^\top K_{\mathfrak{l}\mathfrak{l}} c_\mathfrak{l} \quad \text{subject to: } Y K_{\mathfrak{l}\mathfrak{l}} c_\mathfrak{l} \geq 1 - \xi \ .$$

When aiming at using iterative solvers it is beneficial to have constraints that are only constant bounds on the variables rather than constraints on linear combinations of variables as in the above equation. To achieve this, a simple transformation suffices to obtain

$$\min_{c \in \mathbb{R}^{n+m}, \xi' \in \mathbb{R}^n} \nu c_\mathfrak{l}^\top K_{\mathfrak{l}\mathfrak{l}} c_\mathfrak{l} + \|Y K_{\mathfrak{l}\mathfrak{l}} c_\mathfrak{l} - \mathbf{1} - \xi'\|^2 \quad \text{subject to: } \xi' \geq 0$$

and gradients $\nabla_{c_\mathfrak{l}} Q(c_\mathfrak{l}, \xi') = 2(1+\nu) K_{\mathfrak{l}\mathfrak{l}}^2 c_\mathfrak{l} - K_{\mathfrak{l}\mathfrak{l}} y - K_{\mathfrak{l}\mathfrak{l}} Y \xi'$; $\nabla_{\xi'} Q(c_\mathfrak{l}, \xi') = 2\xi' + \mathbf{1} - Y K_{\mathfrak{l}\mathfrak{l}} c_\mathfrak{l}$ which is already sufficient for optimisation with L-BFGS-B [3] using strategies (*ii*)-(*iv*). For strategy (*i*) we obtain

$$\min_{t \in R^{n+m}, \xi' \in \mathbb{R}^n} \nu t^\top J t + \|Y t_\mathfrak{l} - \mathbf{1} - \xi'\|^2 \quad \text{subject to: } d \geq 0$$

and gradients $\nabla_t Q(t, \xi') = 2\nu J t + \begin{pmatrix} 2t_\mathfrak{l} - y - Y\xi' \\ 0 \end{pmatrix}$; $\nabla_\xi Q(t, \xi') = \mathbf{1} - Y t_\mathfrak{l} - 2\xi$.

2 Runtime Analysis

The "cluster assumption" commonly made in transductive learning algorithms states that the decision boundary is in a low density region, i.e., the cut between vertices of one class and all other vertices is relatively small. It is the supervised version of the observation that many real-world networks exhibit a "community structure", i.e., they consist of several well connected clusters that are among each other only sparsely connected. To analyse this type of networks we use a mixture of random graphs. Conceptually we view them as consisting of two types of edges (note that the learning algorithm will not observe the type). The first type of edges connects the vertices in one cluster and forms a random graph in each cluster. The second type of edges connects vertices from any pair of clusters. For several random graph models the semi-circle law holds (see, e.g., [4]) and the eigenvalues of the normalised Laplacian are well concentrated around

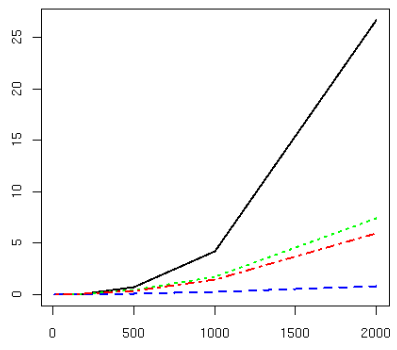

Fig. 1. Runtimes of support vector machines using the four different strategies on artificial data. The plot shows the optimisation time in seconds (y-axis) versus the number of vertices in the graph (x-axis). On our datasets strategy (*ii*) turned out the slowest on large graphs. Strategy (*iii*) and (*iv*) performed better but strategy (*i*) scaled by far best.

one. Putting this together with inequalities for eigenvalues of sums of symmetric matrices shows that the eigenvalues of random graphs with community structure have few clusters. This comes handy when using iterative methods such as (Newton) CG or BFGS as the computations needed in each iteration are matrix vector multiplications and as the number of iterations needed to approximate the minimiser is the same as the number of clusters of eigenvalues [5].

Figure 2 empirically compares the four different strategies on such artificial datasets. Of the shelf toolkits can perform no better than strategies (*ii*) and (*iii*) as their execution time would be dominated by the cost of computing the kernel matrix. Error rates for the different approaches are virtually the same.

3 Conclusions

Motivated by the observation that simple reformulation of Gaussian processes can lead to much faster execution times on graphs, we investigated different techniques to speed up kernel methods, in particular support vector machines. Preliminary experiments show clear performance gains over kernel machine toolkits developed with sparse or dense vectors in mind. The source code of the above described kernel methods will be made available freely online. In future work we will explore the scalability of our approaches on larger real-world graphs.

References

1. Gärtner, T., Le, Q., Burton, S., Smola, A., Vishwanathan, S.: Large-scale multiclass transduction. In: Advances in Neural Information Processing Systems 18. (2006)
2. Tsuda, K., Shin, H., Schölkopf, B.: Fast protein classification with multiple networks. In: ECCB/JBI (Supplement of Bioinformatics). (2005)
3. Zhu, C., Byrd, R., Nocedal, J.: L-bfgs-b, fortran routines for large scale bound constrained optimizatio. ACM Transactions on Mathematical Software **23**(4) (1997)
4. Chung, F.: Spectral Graph Theory. Number 92 in CBMS Regional Conference Series in Mathematics. AMS (1997)
5. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer (1999)