

# Kernel Methods aren't Dead Yet: Using Kernel Methods on Large Datasets



BY EDWARD RAFF

# What is this about?



- Kernel Methods aka the “kernel trick”, most used in Support Vector Machines (SVMs)
- What makes SVMs difficult to use on large scale data?
- How can we overcome these issues to scale to large datasets
- Methods one could use to scale out to a distributed SVM training solution

# Kernel Methods



- Kernel Methods find a linear hyper plane in a different feature space using the “kernel trick”

$$w^T x = \sum_{i=1}^n \alpha_i \underbrace{K(\overbrace{sv_i}, x)}_{\text{kernel trick}}$$

- Kernel projects into a higher dimensional space, making the solution in the original space non-linear
- Unlike Nearest Neighbor,  $\alpha=0$  don't contribute, making solution sparse
- Most common kernel is the Radial Basis Function

$$K(x, y) = \exp\left(-\frac{\|x - y\|_2^2}{2\sigma^2}\right)$$

# Kernel Methods



- SVM became very popular after introduction in the 1990s, often obtained state-of-the-art accuracies
- More theory behind the method, less ad hoc than Random Forest and Neural Networks
- Swapping out the kernel used allows for changing a small amount of code but getting a different type of solution
- Kernel trick allows applying SVMs to different features
  - Strings, feature vectors of different length

# Problems in Practice



- Exact solutions take  $O(n^3)$  time. SMO empirically gets the solution in  $O(n^{2.5 \pm \epsilon})$ , but still slow
  - LIBSVM most common solver
- Caching of kernel evaluations critical for performance, but caching all  $O(n^2)$  values is impractical
- Grid Search for regularization penalty  $C$  and RBF width  $\sigma$  compounds the already slow time to solve
  - Bad  $C$  and  $\sigma$  combinations cause worst case behavior. Makes distributed Grid Search difficult due to drastic runtime differences between parameter combinations
    - ✦ Makes runtime go from  $O(n^{2.5 \pm \epsilon}) \rightarrow O(n^3)$
    - ✦ Fails to reuse cached kernel values

# Speed Over Accuracy: Approximation



- Critical observation is that  $O(n^3)$  runtime is only for *exact* solvers. Approximations may provide a huge performance boost for a small degradation of accuracy.
  - Especially useful for grid search
  - Approximate solvers have been used in Linear methods and Neural Networks (SGD, AdaGrad, etc) for a long time now
- How can you do an ‘approximate’ SVM?
  - Explicitly form an approximate feature space, then use a linear solver
  - Perform SGD on and update the  $\alpha$  values
  - Solve SVM by taking approximate steps to update  $\alpha$  values

# Approximate Feature Spaces



- Popularized in 2007 with “Random Kitchen Sinks”
- Use some transformation  $\tilde{\phi}(x)$  such that

$$\tilde{\phi}(x)^T \tilde{\phi}(y) \approx K(x, y)$$

- Original  $x$  may be  $D$  dimensions, approximate space can be of dimension  $B$ , which is specified beforehand
  - Increasing  $B$  increases the accuracy of the approximation, but slower to take dot products
- By making  $\tilde{\phi}(x)$  relatively cheap to compute, we can then use faster linear solvers (approximate or exact) to solve the problem using the new features

# Approximate Feature Spaces



- Only works for certain kernels, need to derive and code new transform for every desired kernel
  - RBF Kernel form presented below
- $O(D B)$  time per dataum

$$\hat{\phi}(x) = \cos(x^T W_{D,B} + \vec{b}) \cdot \sqrt{\frac{2}{\pi}}$$

$$W_{i,j} \sim \mathcal{N}\left(0, \sqrt{\frac{1}{2\sigma^2}}\right)$$

$$\vec{b}_i \sim \mathcal{U}(0, 2\pi)$$



# Kernel SGD



- Naïve solutions would be to simply update  $\alpha$  on every error, similar to Perceptron
  - This would add an unbounded number of SVs. Even if we only add a SV every  $c$  steps, one pass of the data would require  $(n^2+n)/(2c)$  kernel products
- True support vectors from the SVM may be redundant, if we can avoid the redundancy we can reduce the number of support vectors
- We would like to bound ourselves to using only  $B$  support vectors

# Kernel SGD: Projection



- First introduced in the Kernel RLS paper in 2004. Check to see if a new SV can be adequately represented by a combination of the existing SVs
  - If the approximation has an error less than some  $\delta$ , use the approximation. Else, add the SV
- Two different bounds.
  - We can always force the projection once we hit  $B$  support vectors
  - Every desired bound  $B$  can be achieved via some value of  $\delta$
- Works for any Kernel,  $O(B^2)$  work per update.  
$$\alpha_i K(x_i, y) + \alpha_j K(x_j, y) \approx \hat{\alpha} K(\hat{x}, y), \forall y$$

# Kernel SGD: Merging SVs



- Want to be able find the ‘merged’ support vector that best solves
  - Similar to finding the pre-image in Kernel PCA
$$\alpha_i K(x_i, y) + \alpha_j K(x_j, y) \approx \hat{\alpha} K(\hat{x}, y), \forall y$$
- Solution for RBF kernel proposed in “Multi-class pegasos on a budget” in 2010
  - Always merge the newest SV with the pre-existing ones, updates can be done in  $O(B)$  time



# Grid Search Examples



- **Datasets**
  - a9a,  $n=32,561$ ,  $D=123$
  - mnist,  $n=60,000$ ,  $D=784$
- **Training Methods**
  - LIBSVM
  - Random Kitchen Sinks, Linear SVM via SGD w/ AdaGrad (Top Left)
  - Random Kitchen Sinks, Linear SVM via exact solver (Top Right)
  - Kernel SGD using SV Merging (Bot Left)
  - Kernel SGD using Projection and  $\delta=0.05$  (Bot Right)
- **Approximations are significantly faster for these smaller datasets**
  - Speed advantage will increase with data size due to better big O
- **Even with small budgets, accurate enough to find good parameters**
- **All results run sequentially with a single core**
  - LIBSVM given 5 GB of memory for caching
    - ✦ Larger cache wasn't stable on my machine
  - 2.66 GHz i5, 16GB of 1067 MHz RAM

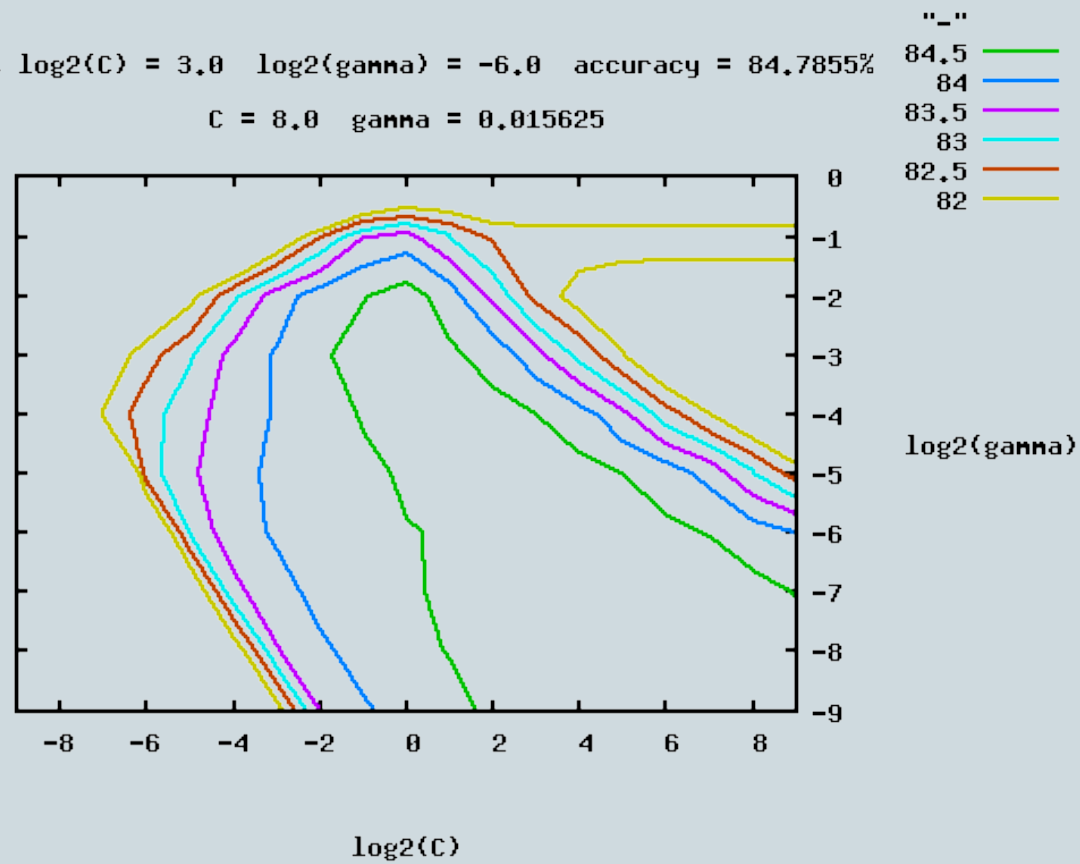
# Grid Search Examples: a9a



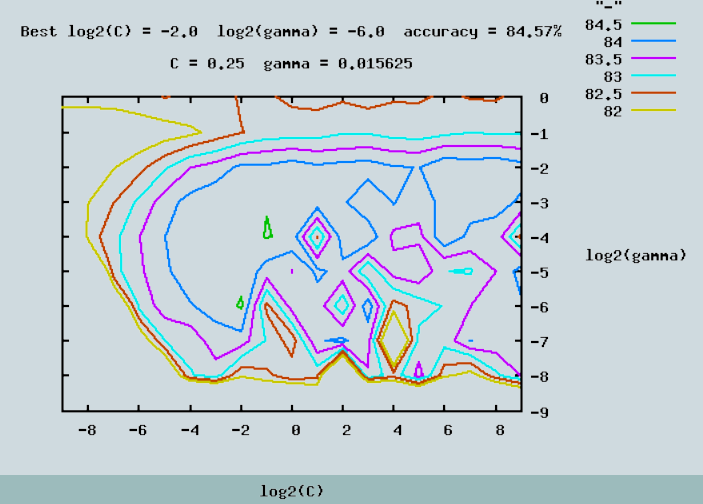
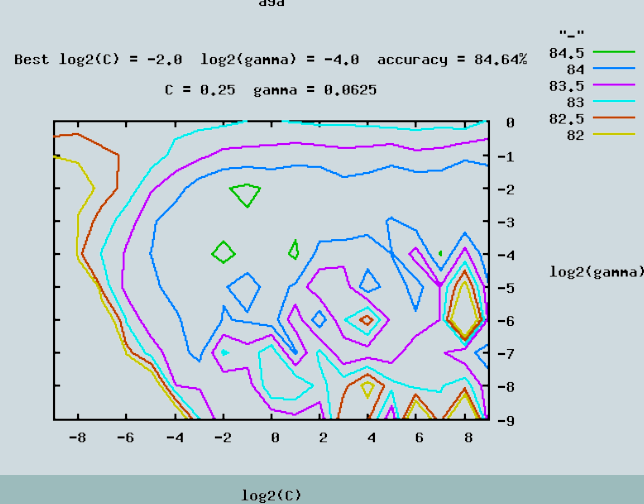
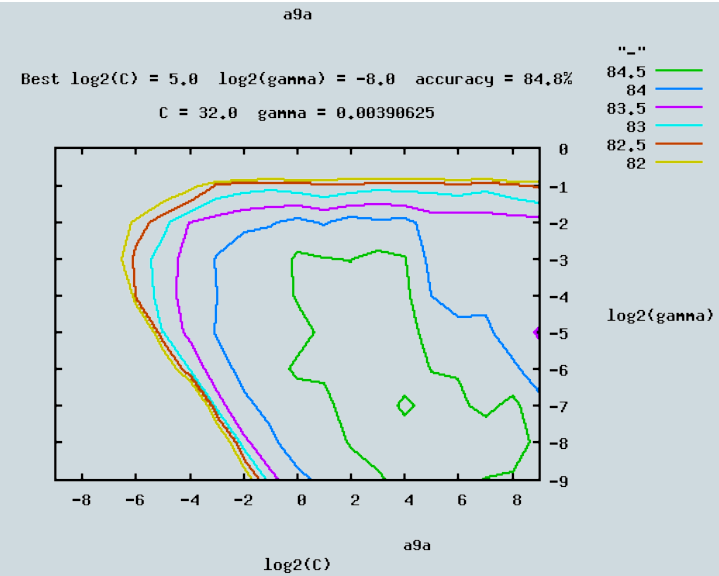
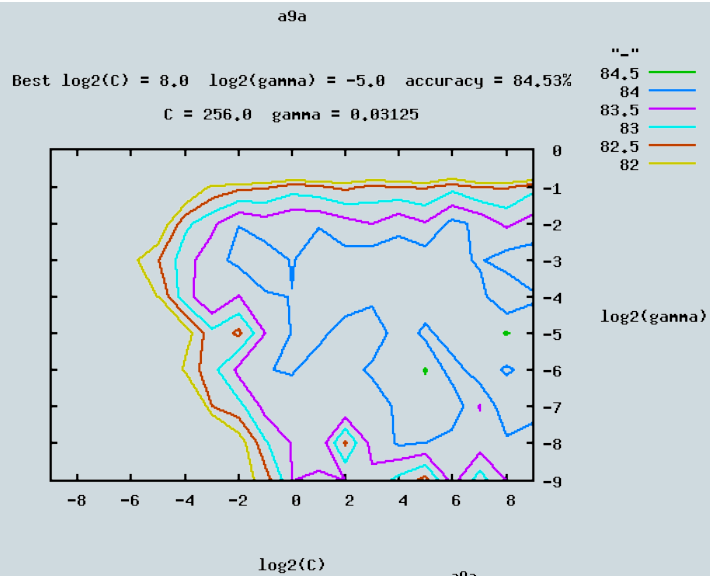
a9a

Best  $\log_2(C) = 3.0$   $\log_2(\text{gamma}) = -6.0$  accuracy = 84.7855%

$C = 8.0$   $\text{gamma} = 0.015625$

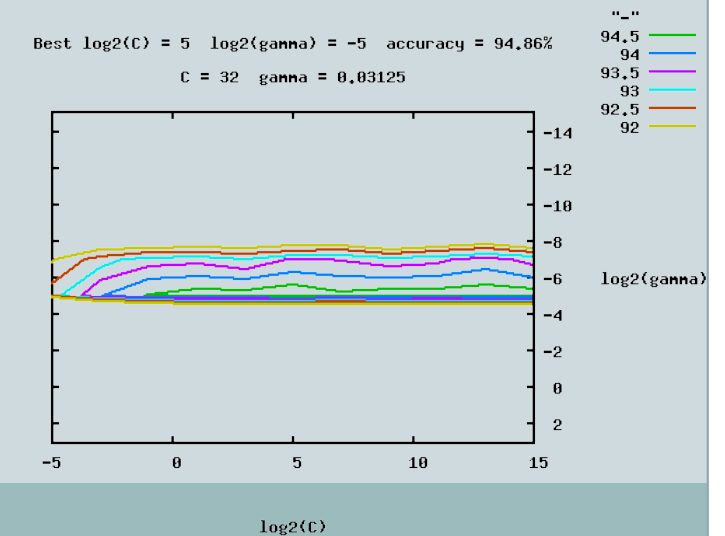
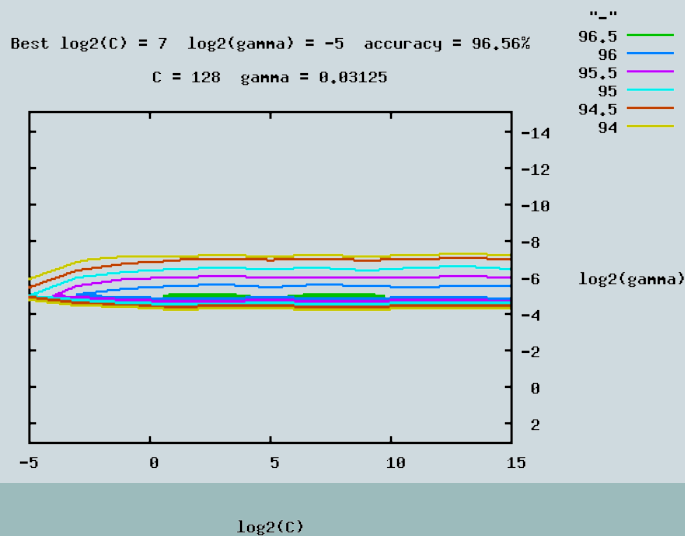
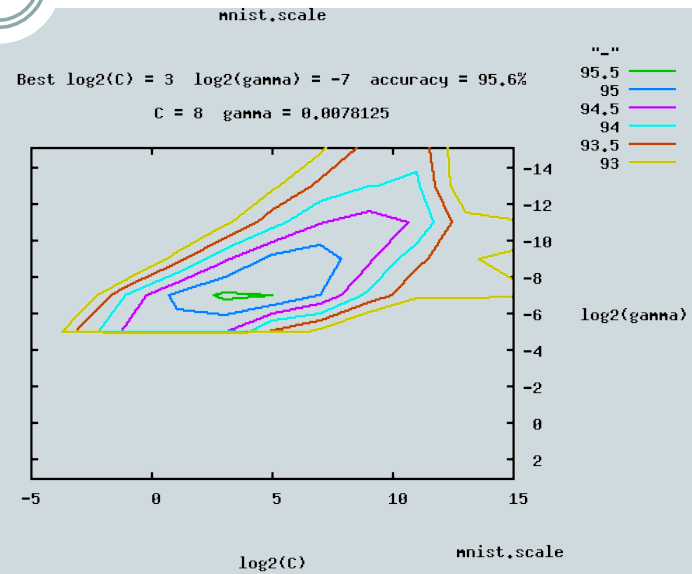
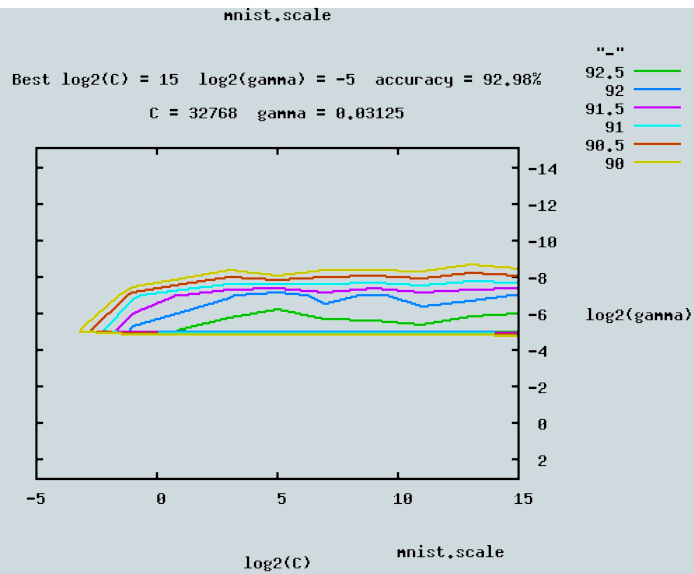


# Grid Search Examples: a9a





# Grid Search Examples: mnist





# Grid Search Runtimes



a9a	Runtime	Speedup
LIBSVM	21 hours 19 minutes	-
RKS SGD w/ AdaGrad	32 minutes	40x
RKS Exact	2 hours 30 minutes	8.5x
Merge RBF	3 hours 10 minutes	6.7x
Projection	1 hour 25 minutes	15x

mnist	Runtime	Speedup
LIBSVM	16 days 6 hours 12 minutes	-
RKS SGD w/ AdaGrad	1 hour 25 minutes	275x
RKS Exact	3 days 7 hours 48 minutes	4.9x
Merge RBF	18 hours 44 minutes	20.8x
Projection	13 hour 34 minutes	28.8x

# Grid Search Results



- While not ‘perfect’, almost always gets a pair of parameters that would have the same top accuracies as LIBSVM
  - Even when it doesn’t, still a reasonable pair
- With respect to sample size, presented methods are  $O(n)$
- All algorithms much better for a distributed grid search
  - All parameter pairs should take similar amounts of time
    - ✦ Fixes the issues of imbalanced work loads
  - All use a *fixed* and *predictable* amount of memory
    - ✦ No need to cache any kernel products
  - All the SGD based ones can be done online
    - ✦ Worst case behavior just means bad accuracy and is predictable
  - Easy to run as Hadoop Jobs
- Can take the 16 days of saved computation and use LIBSVM on the final selected  $C$  and  $\sigma$ 
  - GPU solvers can be 97x-121x faster than standard LIBSVM for some problems

# Distributed SVM



- What if the dataset is too large for training even one LIBSVM model?
- Some distributed SVM algorithms already exist:
  - *PSVM: Parallelizing Support Vector Machines on Distributed Computers* (from Google, open source)
  - *P-packSVM: Parallel Primal gradient descent Kernel SVM* (by Microsoft)
- Distributed algorithm that could be implemented:
  - *Building Support Vector Machines with Reduced Classifier Complexity*
    - ✦ Similar to the projection method, but iterative and selects new basis vectors
    - ✦ Could be implemented on top of Mahout using distributed matrices

# References



- Platt, J. C. (1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. In *Advances in kernel methods* (pp. 185 – 208).
- Shevade, S. K., Keerthi, S. S., Bhattacharyya, C., & Murthy, K. K. (1999). Improvements to the SMO algorithm for SVM regression. *Control Division, Dept. of Mechanical Engineering* (Vol. CD-99–16, pp. CD–99–16). Control Division, Dept. of Mechanical Engineering. doi:10.1109/72.870050
- Engel, Y., Mannor, S., & Meir, R. (2004). The Kernel Recursive Least-Squares Algorithm. *IEEE Transactions on Signal Processing*, 52(8), 2275–2285. doi:10.1109/TSP.2004.830985
- Keerthi, S. S., & Decoste, D. (2006). Building Support Vector Machines with Reduced Classifier Complexity. *Journal of Machine Learning Research*, 7, 1493–1515.
- Rahimi, A., & Recht, B. (2007). Random Features for Large-Scale Kernel Machines. In *Neural Information Processing Systems*.
- Chang, E. Y., Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z., & Cui, H. (2007). Psvm: Parallelizing support vector machines on distributed computers. In *Neural Information Processing Systems*.
- Hsieh, C.-J., Chang, K.-W., Lin, C.-J., Keerthi, S. S., & Sundararajan, S. (2008). A Dual Coordinate Descent Method for Large-scale Linear SVM. In *Proceedings of the 25th international conference on Machine learning - ICML '08* (pp. 408–415). New York, New York, USA: ACM Press. doi:10.1145/1390156.1390208
- Zhu, Z. A., Chen, W., Wang, G., Zhu, C., & Chen, Z. (2009). P-packSVM: Parallel Primal gradient descent Kernel SVM. In *2009 Ninth IEEE International Conference on Data Mining* (pp. 677–686).
- Wang, Z., Crammer, K., & Vucetic, S. (2010). Multi-class pegasos on a budget. In *27th International Conference on Machine Learning* (pp. 1143–1150).
- Cotter, A., Srebro, N., & Keshet, J. (2011). A GPU-tailored approach for training kernelized SVMs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11* (pp. 805–813). New York, New York, USA: ACM Press. doi:10.1145/2020408.2020548
- Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3). doi:10.1145/1961189.1961199