Implementation of A Comparison of Methods for Multi-class Support Vector Machines

Joseph Biernacki
jbiernac@bu.edu

Diana Chiang
dchiang1@bu.edu

Parker Dunn
pgdunn@bu.edu

Luis Ivey
livey@bu.edu

Alexandra Rieders
srieders@bu.edu

Abstract

Each member of the group made equal contributions to this project.

This project, and the paper that it is based on, investigates multiclass support vector machine (SVM) options for large datasets. This topic is relevant because the formulation to solve multiclass problems with SVMs has variables proportional to the number of classes. Both multiple binary classifiers and single optimization methods are implemented and discussed in this paper. Regarding the methods presented in this paper, further work could be done in order to implement the single optimization method in such a way that reduces computational complexity. Furthermore, per this paper's results and the results in [1], research regarding larger datasets and how they interact with these multiclass SVM's could prove to be worthwhile. In this paper and project, one-against-one, one-against-all, and directed acyclic graph (DAG) binary variations and one single optimization variation approach are implemented. The results are inconsistent which reflects the complexity of implementing multiclass SVMs on large datasets. A key component of reference implementations of multiclass SVMs is decomposing the single optimization problems in this paper to decrease the complexity of each quadratic optimization. The next step for this project would have been implementing these decomposition approaches to have robust single optimization algorithms for large datasets.

1 Introduction

The original Support Vector Machine algorithm was designed for solving binary classification problems. This paper attempts to extend this method to multi-class classification problems comparing different methods while attempting to reduce time and resource costs. Two methods of combining binary classifiers, one-against-all and one-against-one, and three "all-together" methods, directed acyclic graph SVM, a modified all-together method and the Crammer and Singer method, were compared with regards to performance and computational complexity.

In the paper, the authors did not find one method to be statistically better than others. However, in regard of training time, one-against-one and DAG performed best. The authors also found that the modified "all-together" method was the slowest in terms of training speed, followed by Crammer and Singer method. It's also noted that Kernel evaluation takes approximately 90 percent of testing time. With all factors considered, the authors suggest one-against-one and DAG approaches for practical use due to training and testing time.

Regarding usage of different kernels, the authors noted that one-against-all had the worst accuracy rate with linear kernel, while one-against-one and DAG performed well. The paper provided further analysis utilizing RBG kernel which produced better accuracy but noted that there are other methods designed for non-linear cases.

2 About the data

Each algorithm was run on nine data sets, all whose outputs are classifiers, with the number of classifiers being greater than two for each set. The data set utilized in this paper is listed in the table and originally from UCI repository. The authors of this paper also uploaded the normalized data sets to LIBSVM site. Thus, the data sets utilized for experiment were obtained from LIBSVM. Data sets iris, wine, glass, vowel, vehicle, segment were small and thus, cross-validation using ten folds was used to train data. Data sets satimage, letter, and shuttle were large and thus, data were split into training (70%) and testing (30%). Training data were additionally split such that a validation set was used to tune hyper-parameters.

Table 1: Data Set Table.

Problem	**Training** *data*	**Testing** *data*	**class**	**attributes**
iris	150	0	3	4
wine	178	0	3	13
glass	214	0	6	13
vowel	528	0	11	10
vehicle	846	0	4	18
segment	2310	0	7	19
satimage	4435	2000	6	36
letter	15000	5000	26	16
shuttle	43500	14500	7	9

3 Multiple Binary Classifiers

3.1 One-against-all (OVA)

The conventional support vector machine algorithm deals with binary classification problems. One strategy for dealing with multi-class problems is to go through the labeled data one class at a time setting its label as positive while all other classes are set as negative. This is done iteratively to obtain k number of decision functions, one for each of k classes of samples. At the testing phase, each test sample is run through all decision functions and the decision function that produce the highest score is determined to be the class that sample belongs to. This algorithm works well but is time consuming as the complexity increases with number of classes as each iteration for solving for one class requires one run of quadratic programming.

The first term right after min is the term that emphasizes maximum margin by minimizing “w”. the second term is the slack variable that allows for some mis-classification in problems that are not completely linearly separable. The general formula remains similar to soft-margin SVM. In this formulation, ½ is the set lambda where C is the penalty constant. This penalty constant determines how much do mis-classifications cost. As C approaches infinity, the problem approaches hard margin SVM.

$$\begin{gathered}\min_{w^i,b^i,\eta^i} \tfrac{1}{2}(w^i)^T w^i + C\sum_{j=1}^{l} \xi_j^i \\ (\mathrm{w}^i)^T\phi(x_j) + b^i \geq 1-\xi_j^i, \mathrm{if} y_j = i \\ (\mathrm{w}^i)^T\phi(x_j) + b^i \leq -1+\xi_j^i, \mathrm{if} y_j \neq i \\ \xi_i^j \geq 0, j = 1\ldots l\end{gathered}$$

3.2 One-Against-One (OVO) Method

The OVA method constructs all possible combinations of binary classifiers. More specifically given k classes, $k(k-1)/2$ classifiers are created during training the process. After all the classifiers have been constructed a voting takes place based on sign$((w^{ij})^T\phi(x) + b^{ij})$ where if the solution gives a positive outcome, the ith class is incremented by one. However, if the equation gives a negative outcome, the jth class is incremented by one. This equation iterates through all the classifiers and then the class with the maximum votes is the predicted class $\hat{y}$. The authors refer to this strategy as the "Max Wins".

The Dual problem with slack variables was not derived in the paper. However, the authors did mention that to reduce of the number of parameters to find the, dual version was used. In the spirit of following the described method as closely as possible to the one described in the paper, the dual version was derived. Using the given primal objective function with slack variables,

$$\min_{w,b,\xi_i} \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{m}\xi_i$$
$$s.t.\ \ y_i(w^Tx_i + b) \geq 1 - \xi_i,$$
$$\xi_i \geq 0$$

The Lagrangian function was constructed,

$$\mathcal{L}(w,b,\xi,\alpha,\beta) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i(1-\xi_i - y_i(w^Tx_i + b)) - \sum_{i=1}^{m}\beta\xi_i$$
$$\frac{\partial\mathcal{L}}{w} = w - \sum_{i=1}^{m}\alpha_i y_i x_i \rightarrow w = \sum_{i=1}^{m}\alpha_i y_i x_i$$
$$\frac{\partial\mathcal{L}}{b} = -\sum_{i=1}^{m}\alpha_i y_i \rightarrow \sum_{i=1}^{m}\alpha_i y_i$$
$$\frac{\partial\mathcal{L}}{\xi_i} = C - \alpha_i - \beta_i \rightarrow \beta_i = C - \alpha_i$$

Substitute the above equations into the Lagrangian Function:

$$g(\alpha,\beta) = \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^T x_j + C\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i - \sum_{i=1}^{m}\alpha_i\xi_i - \sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^{m}\alpha_i y_i b - C\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i\xi_i$$

$$= \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^T x_j$$

$$\max_{\alpha,\beta}\sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^T x_j$$
$$s.t.\ \ \alpha_i \geq 0$$
$$\beta_i \geq 0$$
$$\beta_i = C - \alpha_i$$
$$\sum_{i=1}^{m}\alpha_{ii} = 0$$

Using α_i to replace β_i

$$\max_{\alpha,\beta}\sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^T x_j$$
$$s.t.\ \ 0 \leq \alpha_i \leq 0$$
$$\sum_{i}^{m}\alpha_i y_i = 0$$

Once the Dual problem with slack variables was derived the training and testing process was considered because of the multi-class prediction nature. Knowing the amount of classes a dataset has, W weight matrix was constructed during the training process. Each column of W contains the weights which is the results from each classifier. A Look up Table (LUT), was also made during training time to find which two classes were being trained on that corresponds to the resulting weights in the matrix W. The LUT was utilized during testing time where the two classes are needed for the voting process. The training algorithm used for this method is shown below,

Algorithm 1 Training one-against-one SVM

```
Require: $(k, X_{tr}, y_{tr}, C)$
  $W \leftarrow 0$
  $LUT \leftarrow 0$
  for $i = 1 : k$ do
      for $j = i + 1 : k$ do
          $LUT \leftarrow i$
          $LUT \leftarrow j$
          $X \leftarrow X_{tr}(sort([ij]), :)$
          $y \leftarrow y_{tr}(sort([ij]))$
          Binarize y
          $y(y == i) \leftarrow 1$
          $y(y == j) \leftarrow -1$
          Construct Quadprog Parameters
          $H \leftarrow K(\phi_i, \phi_j). * (y * y')$
          $f \leftarrow -ones(1, m)$
          $LB \leftarrow zeros(1, m)$
          $UB \leftarrow zeros(1, m) * C$
          $Aeq \leftarrow y'$
          $beq \leftarrow 0$
          $alpha \leftarrow Quadprog(h, f, [], [], Aeq, beq, LB, UB)$
          Reconstruct w
          $w \leftarrow X' * (alpha. * y_t r)$
          $b_{support} \leftarrow find(alpha < Calpha > 0.1)$
          $b = sum(y(b_{support})' - w' * X((b_{support}), :)')/|b_{support}|$
          $W \leftarrow w$
          $B \leftarrow b$
      end for
  end for
  return W, B, LUT
```

However, after testing the pseudocode above, accuracy results yielded worse than introducing a bias term in the training set. Therefore, the following psuedocode was used instead,

Algorithm 2 Training one-against-one SVM

```
Require: $(k, X_{tr}, y_{tr}, C)$
  $W \leftarrow 0$
  $LUT \leftarrow 0$
  $Xtr \leftarrow [ones(1, size(Xtr, 2)); Xtr]$
  for $i = 1 : k$ do
      for $j = i + 1 : k$ do
          $LUT \leftarrow i$
          $LUT \leftarrow j$
          $X \leftarrow X_{tr}(sort([ij]), :)$
          $y \leftarrow y_{tr}(sort([ij]))$
           Binarize y
          $y(y == i) \leftarrow 1$
          $y(y == j) \leftarrow -1$
           Construct Quadprog Parameters
          $H \leftarrow K(\phi_i, \phi_j). * (y * y')$
          $f \leftarrow -ones(1, m)$
          $LB \leftarrow zeros(1, m)$
          $UB \leftarrow zeros(1, m) * C$
          $alpha \leftarrow Quadprog(h, f, [], [], [], [], LB, UB)$
           Reconstruct w and b
          $w \leftarrow X' * (alpha. * y_t r)$
          $W \leftarrow w$
      end for
  end for
  $B \leftarrow W(1, :)$
  return W, B, LUT
```

For the validation process two different implementations had to be considered because of the size of the testing set for each data set. Iris, Wine, Glass, Vowel, Vehicle, and Segment was implemented in a ten fold cross-validation procedure to select the parameters. Satimage, letter, and shuttle data set did include a testing set. The training set for each data set was divided into a training and validation set using a 30/70. The constraints for the dual were C = $[2^{12}, 2^{11}, 2^{12}, ..., 2^{-2}]$ for the linear kernel and C = $[2^{12}, 2^{11}, 2^{12}, ..., 2^{-2}]$ and $\gamma = [2^4, 2^3, 2^2, ..., 2^{-10}]$ for the RBF kernel. The overall testing pseudocode is shown below,

Algorithm 3 Testing one-against-one SVM

```
Require: $(X_{tr}, y_{tr}, X_{te}, y_{te})$
  $k \leftarrow numel(unique(y_{tr}))$
  $vote \leftarrow zeros(1, k)$
  $test_{err} \leftarrow 0$
  $vote \leftarrow zeros(1, k)$
  $acc \leftarrow 0$
  $[W, B, LUT] \leftarrow train - sv - one - vs - one(k, X'_{tr}, y_{tr}, C, \gamma)$
  $[m, d] \leftarrow size(X_{te})$
  for $j = 1 : m$ do
      $x_j \leftarrow X_{te}(j, :)$
      $\hat{Y} \leftarrow sign((x_j * W) + B)$
      for $l = 1 : \hat{Y}$ do
          if $\hat{Y}(l) == 1$ then
              $vote(LUT(1, l)) \leftarrow vote(LUT(1, l)) + 1$
          else
              $vote(LUT(2, l)) \leftarrow vote(LUT(2, l) + 1$
          end if
      end for
      $[-, \hat{y}] \leftarrow max(vote)$
      if $y_{te}(j)\ \ = \hat{y}$ then
          $test_err \leftarrow test_err + 1$
      end if
      $vote \leftarrow vote * 0$
  end for
  $acc \leftarrow 1 - (test_{err}/numel(y_{te}))$
  return acc
```

Results are shown below,

Table 2: Linear Kernel

Problem	C	rate
iris	2^{12}	94.81%
wine	2^{12}	95.00%
glass	2^{4}	59.38%
vowel	2^{6}	54.32%
vehicle	2^{5}	61.63%
segment	2^{12}	93.22%

Table 3: RBF Kernel

Problem	C	γ	rate
satimage	2^{6}	64.77%	2^{-9}

3.3 DAG SVM

The Decision Directed Acyclic Graph (DDAG, or simply DAG) was originally presented in 1999 by Platt et. al as a strategy for building a multiclass classified out of many two-class classifiers. Consider an N class problem: for such a problem, the DAG approach requires [N (N - 1) / 2] classifiers (one for each pair of classes). Figure 1 offers a helpful visualization. The DAG approach can be extended and generalized to a number of binary classifiers, but for the purposes of this paper the classifiers used are hyperplanes, or Support Vector Machines (SVM's). The benefit of this approach is that the test error should ideally depend on N and the margin achieved at the nodes, but not on the dimension of the space.

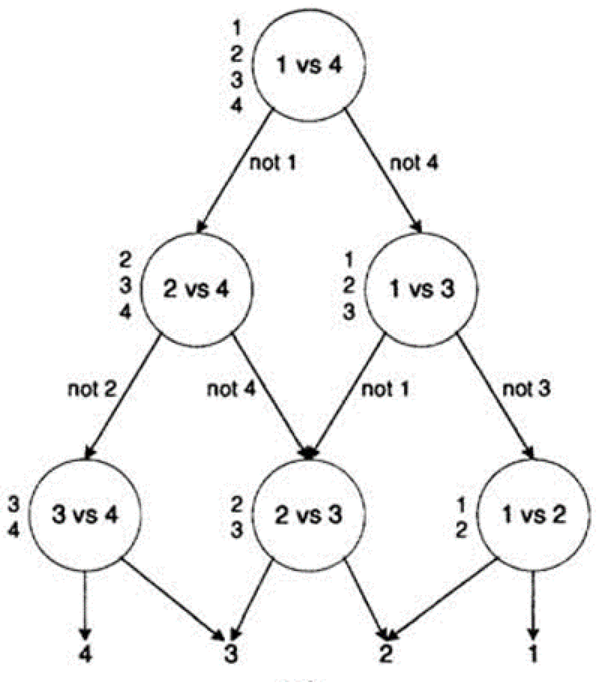


Figure 1

For a given sample, the graph is traversed in recursive fashion. A binary classifier is applied to the sample at every node, and the result of the final classifier is the predicted class of the sample. Considering once again a problem with N classes, see the algorithm below. Note that the notation i-j refers to the binary classifier for class i against class j.

Algorithm 4 DAG

```
$i \leftarrow 1$
$j \leftarrow N$
$prediction \leftarrow applySVM(sample, ij)$
**while** start != end **do**
    **if** $prediction > 0$ **then**
        $j = j - 1$
        $= applySVM(sample, i - j$
    **else**
        $i = i + 1$
        $= applySVM(sample, i - j)$
    **end if**
**end while**
```

The SVM's were constructed using the classical primal objective function:

$$\begin{aligned} \min_{w,b,\xi_i} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad y_i\left(w^t x_i + b\right) \geq 1 - \xi_i, \\ \xi_i \geq 0 \end{aligned}$$

For each dataset, [N $*$ (N $-$ 1)/2] binary SVM's were trained on 15 different C values ranging from $[2^\wedge - 2, 2^\wedge - 3, \ldots, 2^\wedge 11, 2^\wedge 12]$. The optimal C value was then selected by applying all 15 iterations against validation set and selecting the C that resulted in the highest accuracy rate. Note that this is a fairly expensive process from a computational perspective, requiring the construction of $15 * [\,\mathrm{N}(\,\mathrm{N}-1)/2]$ binary SVM's for each dataset (e.g., for "satimage", 225 binary SVM's). The four datasets treated for the DAG experiment were deemed sufficiently large so as to train on 70% of the training set and validate on the remaining 30%. All classifiers were trained on a linear kernel. The resulting classifier was used to test the learner against a test set.

The DAG algorithm was tested on 4 different datasets using an Intel i7-9750H CPU @ 2.60GHz with 6 cores and 12 threads. The table below indicates metrics for the satimage dataset. Production of more results is in progress but moving slowly due to limited time and computaitonal resources.

	satimage
Training time	1231.06549
Testing Time	0.07893
Total Time	2462.20991
Best C	16
Validation Accuracy	83.25
Test Accuracy	79.553

4 Single Optimization

Another approach to using SVMs for multiclass problems is to solve a single optimization problem. The approach implemented here and discussed in this paper finds optimal hyperplanes for all of the classes in a single objective function. The key concept to this approach is how the slack variables, ξ, are defined. Instead of multiple slack variables representing the gap of each sample between ever pair of hyperplanes, the ξ_i in this objective function represent only the maximum gap between two hyperplanes; one of which is the hyperplane corresponding two the true class.

$$\xi_i = \max_m(w_m^T\phi(x_i) + e_i^m) - w_{y_i}^T\phi(x_i))_+$$

where $(z)_+ = max(z, 0)$. The primal objective function for this single optimization approach is...

$$\min_{w,\xi_i} \frac{1}{2}\sum_{m=1}^{k} w_m^T w_m + C\sum_{i=1}^{l}\xi_i w_{y_i}^T\phi(x_i) - w_m^T\phi(x_i) \geq e_m^m - \xi_i,$$

$$i = 1, ..., l$$

$$e_i^m \equiv 1 - \delta_{y_i,m}$$

$$\delta_{y_i,m} \equiv \begin{cases} 1 & \text{if} y_i = m \\ 0 & \text{if} y_i \neq m \end{cases}$$

Note that here the maximum k numbers considered is

$$\xi = (max_m(x_m^T\phi(x_i) +_i^m) - w_{y_i}^T(\phi(x_i))_+$$

Furthermore, here we need not specific that $\xi_i \geq 0$ since $e_i^m = 0$ when $y_i = m$ so we have

$$0 \geq 0 - \xi_i$$

$$\xi_i \geq 0$$

Since it is helpful to be able to use multiple kernels with SVM classifiers, the single optimization problem was implemented in the dual formulation as well as the primal. The dual objective function of the single optimization problem is below. Single Optimization Dual Formulation

$$\min_{\alpha} \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} K_{i,j} * \bar{\alpha}_i^T * \bar{\alpha}_j + \sum_{i=1}^{l}\bar{\alpha}_i^T * \bar{e}_i$$

Constraints

$$\sum_{m=1}^{k} \alpha_i^m = 0, i = 1, ..., l$$
$$\alpha_i^m \leq 0, y_i \neq m$$
$$\alpha_i^m \leq C, y_i \neq m$$

Definitions:

$$i = 1, ..., l$$
$$m = 1, ..., k$$
$$K_{i,j} = \phi(x_i)^T \phi(x_j)$$
$$\bar{\alpha}_i = [\alpha_i^1, ..., \alpha_i^k]^T$$
$$\bar{e}_i = [e_i^1, ..., e_i^k]^T$$

In this formulation, k represents the number of classes, and l is the total number of samples used for training. Additionally, C is the penalty hyperparameter. In addition to the variations of α in the objective function we also define the α and e vectors since they are important components of the quadratic formulation of this problem that must be solved to optimize the objective function.

$$\alpha = [\alpha_1^1, ..., \alpha_1^k, ..., \alpha_l^k, ..., \alpha_l^k]^T$$
$$e = [e_1^1, ..., e_1^k, ..., e_l^k, ..., e_l^k]^T$$

Solving the dual objective function gives the values of α, which could be used to calculate each of the hyperplanes for each class. Since we just use the hyperplanes (w_m) for classifying new samples, we can also bypass finding w_m and use α directly via the decision function below.

$$arg \max_{m=1,...,k} \sum_{i=1}^{l} \alpha_i^m K(x_i, x)$$

The quadratic formulation of the objective function below is how the dual single optimization problem was solved using *quadprog* in MATLAB.

$$\frac{1}{2}\alpha^T (K \otimes I)\alpha + e^T \alpha$$

The Hessian matrix is defined as $(K \otimes I)$ as a result of the structure of α. Here, K is the kernel matrix and I is a k by k identity matrix. The $(K \otimes I)$ is the Kronecker product of these two matrices, which was implemented using *repmat* in MATLAB.

5 Results and Analysis

Data set	Ova	Ovo	Modified	DAG	CS
iris	100	94.81	73	NA	83.7
wine	100	95	90.35	NA	92.88
glass	61.90	59.38	32.86	NA	NA
vowel	61.54	54.32	34.02	NA	NA
vehicle	72.62	61.63	NA	NA	44.6
segment	94.37	93.22	NA	NA	80
satimage	65.03	NA	43	79.553	71
letter	58.64	NA	84.5	NA	72
shuttle	NA	NA	19	NA	97

As the table above shows, OVA had the highest accuracy rate among all the other methods for data set iris and wine. However, the UCI repository clearly states that iris has only one class that is linearly separable while the other two classes were not. The methods above only show linear kernel results.

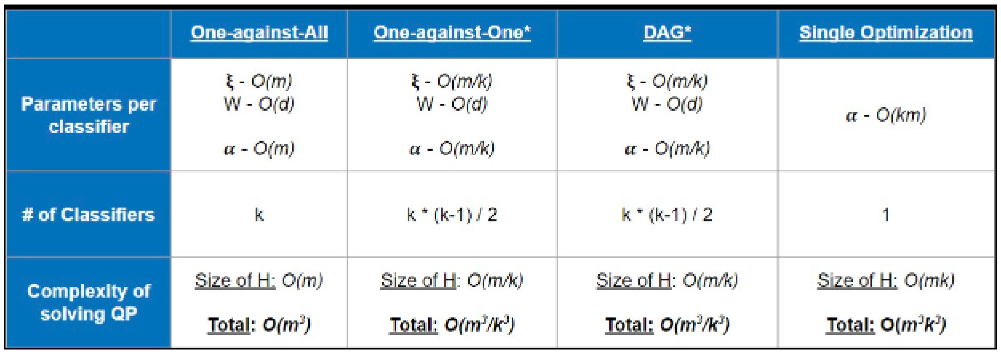
For the single optimization techniques, both primal and dual formulations, the runtime of these algorithms was the slowest. For both formulations, we had trouble properly assessing appropriate sets of hyperparameters and often could not run optimize on the entire dataset. The results provided demonstrate what was implemented for this project, but do not truly reflect the capabilities of the single optimization approach. The next step in these efforts would be to implement the decomposition techniques that can be found in the references used for this project. These methods find ways to make these nearly infeasible optimization tasks possible by selecting only a portion of the data and parameters to optimize at a time, reducing the algorithmic complexity of the most costly step of the algorithm.

6 Computational Complexity

Since the purpose of this project, and the paper it is based on, is to assess approaches for multiclass SVM, it is important to consider the theoretical complexity of each of these approaches. In Table 1, an outline of the size of the classifiers is provided based on the number of parameters. The third row of this table is the most important though because it highlights the complexity of solving the quadratic optimization problem, which is the most complex step of the optimization. Generally, quadratic optimization has complexity $O(n^3)$ where n is the dimensionality of the Hessian matrix, H (n by n), in the quadratic problem (see equation below):

$$.5x'Hx+f'x$$

For the one-against-one and DAG approaches, we have more classifiers than the other approaches, but also fewer samples per optimization problem. The number of samples per optimization is dependent the number of samples in each class, but we can estimate this number as $2 * \frac{m}{k}$. Although it is not explicitly shown, multiple QP problems would be solved for all approaches except the single optimization method. INSERT IMAGE.



	One-against-All	One-against-One*	DAG*	Single Optimization
Parameters per classifier	ξ - $O(m)$ W - $O(d)$ α - $O(m)$	ξ - $O(m/k)$ W - $O(d)$ α - $O(m/k)$	ξ - $O(m/k)$ W - $O(d)$ α - $O(m/k)$	α - $O(km)$
# of Classifiers	k	k * (k-1) / 2	k * (k-1) / 2	1
Complexity of solving QP	Size of H: $O(m)$ **Total:** $O(m^3)$	Size of H: $O(m/k)$ **Total:** $O(m^3/k^3)$	Size of H: $O(m/k)$ **Total:** $O(m^3/k^3)$	Size of H: $O(mk)$ **Total:** $O(m^3k^3)$

Figure 2

Based on the theoretical complexities outlined above, the quadratic optimization for the single optimization approach is significantly larger than the other approaches. The complexity of the optimization has cubic growth when either the number of classes or number of samples is increased. Thus, this approach ends up being very sensitive to large multiclass problems.

The one-against-one and DAG approaches have smaller optimization problems that are roughly $O(\frac{m^3}{k^3})$, but also require $O(k^2)$ optimizations due to the number of classifiers needed. Despite multiple optimizations required, the complexity of these optimizations is $O(m^3)$. Thus, the problem does not grow significantly as the number of classes increases, and should grow slower with the number of samples than the one-against-all method does.

7 Conclusion

The results demonstrated in this paper are analogous to the paper that it is modeled after. We saw slower training times for single optimization methods, which is corroborated by the calculations of computational complexity for each algorithm. In terms of algorithms implemented in this paper, more robust methods could be used in order to improve testing accuracy. Testing using Gaussian and polynomial kernels would be an appropriate next step. Notably, for each dataset each class was well-represented (i.e. there was no class for which there were significantly fewer samples than others) and this could very well impact how well each algorithm performs. It would be worth exploring how number of samples in each class affects multiclass SVM performance. In future papers, it would useful to explore how introducing a bias term in Crammer and Singer's method would affect training time. Additionally, exploring methods pertaining to data storage and multiclass SVM proves to be a topic worth further exploration.

References

[1] Chih-Wei Hsu and Chih-Jen Lin, "A comparison of methods for multiclass support vector machines," in IEEE Transactions on Neural Networks, vol. 13, no. 2, pp. 415-425, March 2002, doi: 10.1109/72.991427.

[2] C. L. Blake and C. J. Merz. (1998) UCI Repository of Machine Learning Databases. Univ. California, Dept. Inform. Comput. Sci.,Irvine, CA. [Online]. Available: http://www.ics.uci.edu/ mlearn/MLRepository.html

[3] K. Crammer and Y. Singer, "On the learnability and design of output codes for multiclass problems," *Comput. Learing Theory*, $pp.35\breve{}46, 2000$.

[4] LIBSVM: A Library for Support Vector Machines, C.-C. Chang and C.-J. Lin. (2001). [Online]. Available: http://www.csie.ntu.edu.tw/ cjlin/libsvm

[5] J. Weston and C. Watkins, "Multi-class support vector machines," presented at the Proc. ESANN99, M. Verleysen, Ed., Brussels, Belgium, 1999.