CSCI5525: Homework #2

Due on October 26 2017 at 11:55pm

Professor Paul Schrater

Daniel Wu

Problem 1

Part a: Implementing SVMs using CVXOPT

A linear support vector machine (SVM) utilizes the intuition of creating a linear decision boundary, called the separating hyperplane, that separates two classes. If a point is far enough away from the hyperplane, we can be confident about our ability to classify that point. We can define this linear classifier as

$$w^T x + b = 0$$

for the central hyperplane, where (l, w) as are variables of to linearize our data. We can also construct two parallel hyperplanes as functional margins for classification such that the margin between and the central hyperplane is 1.

$$w^T x + b = \pm 1$$

The main idea is to maximize our class separation by choosing the best w. We can see that we can arbitrarily maximize our separation by scaling w to be large, so we can modify our optimization problem by normalizing by $||w||$. Ideally, we want to maximize this margin $\frac{1}{||w||}$, which is a convex optimization problem. In the separable case of our data, we ideally want all our points to lie outside the margins. We can mathematically note this constraint by setting our y target values as $1, -1$ in the binary classification problem. So ultimately we want our optimization to instead be

$$\min \frac{1}{2}||w||^2 \text{ such that } y_i(w^T x_i + b) \geq 1, \forall i$$

$\frac{w}{||w||}$ is an orthonormal vector of our hyperplane. This equation is for the separable case, in the non-separable case, the primal optimization problem becomes:

$$\min \frac{1}{2}||w||^2 + C\sum_i \xi_i \text{ such that } y_i(w^T x_i + b) \geq 1 - \xi_i, \xi_i \geq 0, \forall i$$

This is a quadratic convex optimization problem with linear constraints. We can then use Lagrange multipliers as method to solve the "primal" (i.e. original) optimization by solving the dual problem instead, which is a concave optimization problem that gives us the best lower bound to the optimal solution to its primal problem. Under certain conditions called the Karush-Kuhn-Tucker (KKT) Conditions, the property of strong duality holds, meaning that the optimal solutions of both the dual and primal are the same. In the non-separable case, the Lagrangian dual is given by

$$L^*(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

which will be maximized. With the constraints given by KKT conditions

$$w = \sum_i \alpha_i y_i x_i$$

$$\sum_{i,j} \alpha_i y_i = 0$$

$$0 \leq \alpha \leq C$$

We attempt to solve this dual problem using the CVXOPT quadratic convex optimizer. We can rewrite this problem in matrix form that matches the required input formate for the optimizer. The CVXOPT accepts inputs regarding the solution to this set of equations.

Problem 1 continued on next page. . .

with the constraints given by KKT conditions

$$\begin{aligned} \min \frac{1}{2}x^T Px - q^T x \\ \text{s.t. } Gx \leq h \\ \text{and } Ax = b \end{aligned}$$

We can see that we can easily represent our dual problem in this form, such that:

$$\begin{aligned} \min \frac{1}{2}\alpha^T K\alpha - 1^T\alpha \text{ where } K = y^T X^T Xy \\ \text{s.t. } 0 \leq I\alpha \leq C \\ \text{and } y^T\alpha = 0 \end{aligned}$$

The only difficulty was implementing two inequality constraints on α for the optimizer. Ultimately, it required concatenation of both sets of inequalities into a combined matrix that the solver was able to accept.

Part b

We investigate the impact of our constant C in the performance of our algorithm. When we visualize our data, it is clear that as we increased our C, we improve the performance of our model by introducing a bias away from our regularization term and makes it zero, meaning we penalize heavily on any points falling within our margin. Increasing our bias term also decreases geographical margin so that more of our data set will away from the margins. For a dataset like MNST-13, we expect that this would give us a better test error rate considering the data is largely separable. Finally with support vectors, we see a general upward trend proportional to increasing C values. This makes mathematical sense as we loosen the constraints on C, we allow more support vectors to construct our hyperplane. These trends are visualized in the plots below in Figure 1. We can also see these values in Table 1.

In general we can conclude that C modulates how our model emphasizes better fit in our training data, or better generalization that avoids potential overfitting.

Part c

Given our original constraints we know that the margin constraints are bound by our slack variables from the KKT condition for strong duality

$$y_i(w^T x_i + b) = 1 - \xi_i$$

for vectors that lie on within the hyperplanes (i.e. $a_i \geq 0$). Thus we can derive an expression ξ

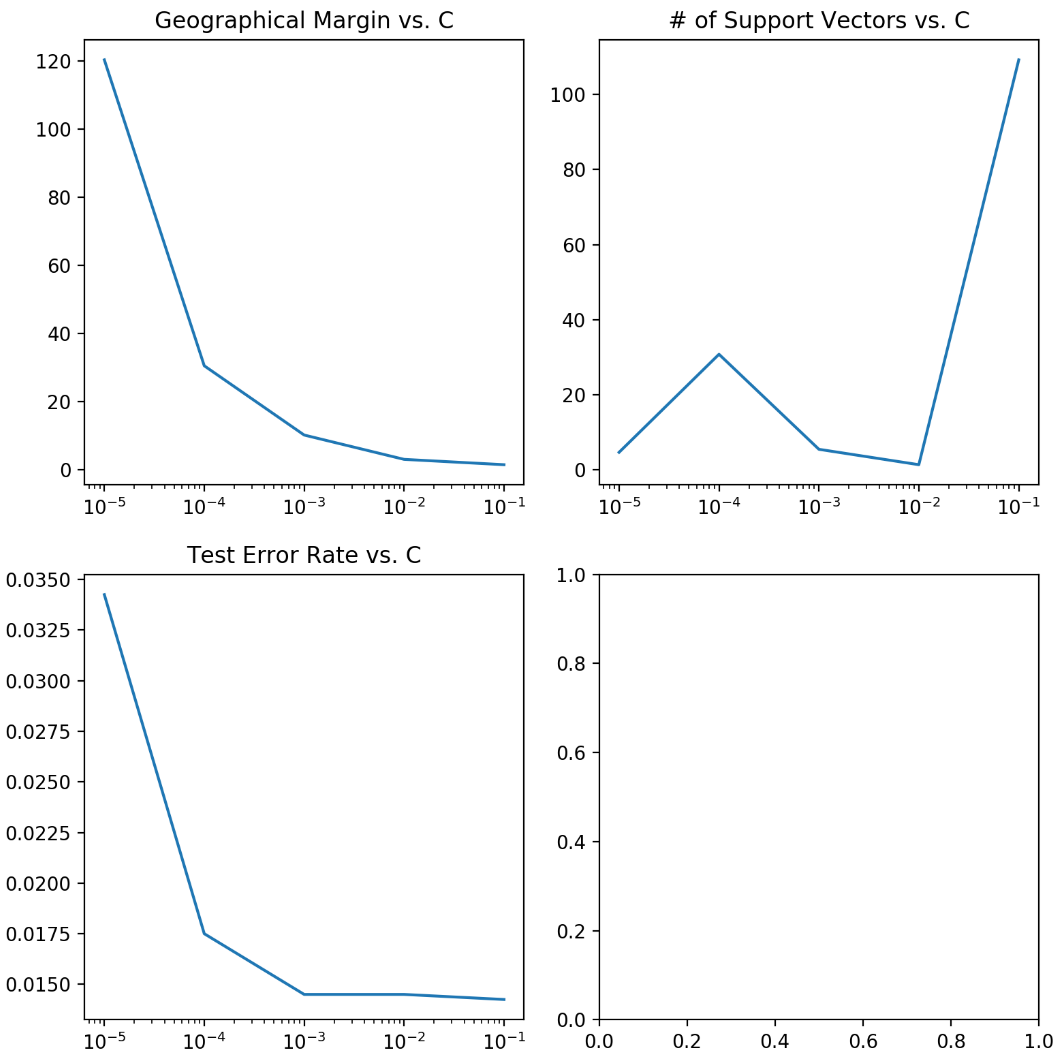
$$\xi_i = 1 - y_i(w^T x_i + b)$$

The conditions of our margin constraints were satisfied with these expressions, since $\xi = 0$ when beyond at and beyond the margin.

Table 1: CVXOPT SVM Model Parameters vs. C

	C = 10^{-5}	C = 10^{-4}	C = 10^{-3}	C = 10^{-2}	C = 10^{-1}
Avg. Test Error Rate	2.90%	1.775%	1.25%	1.35%	1.025%
Std. Test Error Rate	0.593%	0.493%	0.433%	0.583%	.50%
# of support vectors	4.1	29.0	4.1	1.2	107.7
Geometric Margin $1/\|\|w\|\|$	120.76	30.53	10.26	3.012	1.472

Figure 1: Model Variables vs. log(C)



Problem 2

Primal Estimated sub-GrAdient SOlver for SVM (Pegasos) (part a)
The objective function to be optimized is the primal form fo the support vector machine (SVM)

$$L(w) = \frac{\lambda}{2}||w||^2 + \frac{1}{m}\sum_{(x,y)\in S} l(w;(x,y))$$

where in SVMs, the particular loss function is defined as:

$$l(w;(x,y)) = \max[0, 1 - y < w, x >]$$

However, because we are using the stochastic gradient descent method, a convex continuous gradient does not exist for this particular function. Instead the Pegasos algorithm uses the subgradient of the SVM loss function defined by the nonzero values, which happen when they satisfy the constraint $[0, 1-yw^Tx] \leq 1$. And thus we will take the subset of the data in each iteration of the gradient descent that satisfy this constraint and compute the gradient. Then our objective function to be optimized becomes

$$\min_w \lambda||w|| - \frac{1}{m}\sum_{(x,y)\in|A|_k} yX$$

The Pegasos algorithm has primarily two parts to the algorithm, a stochastic gradient descent algorithm, and a constraint on w into a ball-like set $B = w : ||w|| \leq 1/\sqrt{\lambda}$. The stochastic gradient descent algorithm is relatively straight forward. For each iteration we randomly pick k observations from the training set, defined as A_t and update the weight vector based on a subset of A_t, $A_t^+ = \{i \in A_t : y_i\langle w, x_i\rangle < 1\}$ since only those observations contribute to our optimization problem. Also knowing that optimal w^* is inside the set B, we can initialize the weight vector to be inside the set B, and also project our weight vector iterations to be in the set B to further accelerate convergence.

Softplus Estimation of Gradient (part b)

Another method to solve the lack of a true gradient for the hinge loss function is to analytically calculate the gradient of a similar function that estimates the hinge loss function but is completely smooth. For this algorithm we will replace the hinge loss algorithm with the softplus functions the loss function. Thus the gradient we know need to compute for gradient descent is:

$$\nabla L(w) = \frac{1}{N}\sum_{i=1:N} \nabla[a\log(1 + \exp((1 - y_i w^T x_i)/a)] + \lambda||w||^2$$

This gradient, when computed ends up being our equation to optimize:

$$\min_w \lambda||w|| - \frac{1}{m}\sum_{(x,y)\in|A|_k} \frac{y_i \mathbf{x_i}}{1 + \exp\left((y_i \mathbf{w}^T \mathbf{x}_i - 1)/a\right)}$$

Using the same Pegasos algorithm for stochastic gradient descent as we did for the subgradient of the hinge loss, we can iterate until we reach convergence and optimize w.

Problem 2 continued on next page. . .

Pegasos Algorithm

```
Input: Training set $S$, regularization parameter $\lambda$, number of iterations $T$, size of training subset $k$
Initialize: Choose $w^0$ s.t. $||w^0|| \leq 1/\sqrt{\lambda}$
For $t = 0, 1, \ldots, T-1$
    Choose $A_t \subset S$, where $|A_t| = k$
    Set $A_t^+ = \{i \in A_t : \ y_i \langle w^t, x_i \rangle < 1\}$
    Set step size $\eta_t = \frac{1}{\lambda t}$
    Set $w^{t+\frac{1}{2}} = (1 - \eta_t \lambda) w^t + \frac{\eta_t}{k} \sum_{(x,y) \in A_t^+} yx$    (pure gradient descent)
    Set $w^{t+1} = \min \left\{ 1, \frac{1/\sqrt{\lambda}}{||w^{t+\frac{1}{2}}||} \right\} w^{t+\frac{1}{2}}$    (project $w^{t+\frac{1}{2}}$ into optimal set $B$)
Output: $w^T$
```

Methods and Results

Pegasos works with a mini-batch A_t of size k in iteration t. We compared both Pegasos implementation and the softplus softened hinge loss algorithm to using the stochastic gradient method and constraints placed on the the optimal $w*$. We used the MNIST-13 dataset for our models, and evaluated our models performance at k = 1, 20, 200, 1000, 2000, respectively. For each setting, the model is trained 5 times and the mean and the standard deviation of the training time is recorded. The results are summarized in the table below. For the algorithms, we've assumed that $\lambda = 10^{-4}$ and $a = 1$

Table 2: Training time for Pegasos algorithm

	k = 1	k = 20	k = 200	k = 1000	k = 2000
Avg. Runtime (sec)	0.115	3.756	1.819	1.627	1.989
Std. Runtime (sec)	0.001	0.077	0.095	0.062	0.22
# of iterations (avg)	983.8	974.6	385.2	220.2	231.0

Table 3: Training time for Softplus Implementation

	k = 1	k = 20	k = 200	k = 1000	k = 2000
Avg. Runtime (sec)	0.118	3.975	2.365	3.46	6.364
Std. Runtime (sec)	0.002	0.111	0.061	0.076	0.153
# of iterations (avg)	981.4	975.2	411.2	237.8	219.6

In reviewing the data, we can see that even though the number of iterations that both algorithms takes is comparable, there is a significant difference in runtime as we approach higher k subsets. This would make sense considering that calculating the exponential can be computationally expensive especially as we expand our dataset. We can see in both cases that there are tradeoffs to be made in computational time if we use too small of a subset or if we use too large of subset.

Figure 2: Pegasos Algorithm Objective Primal Convergence vs. Number of Iterations

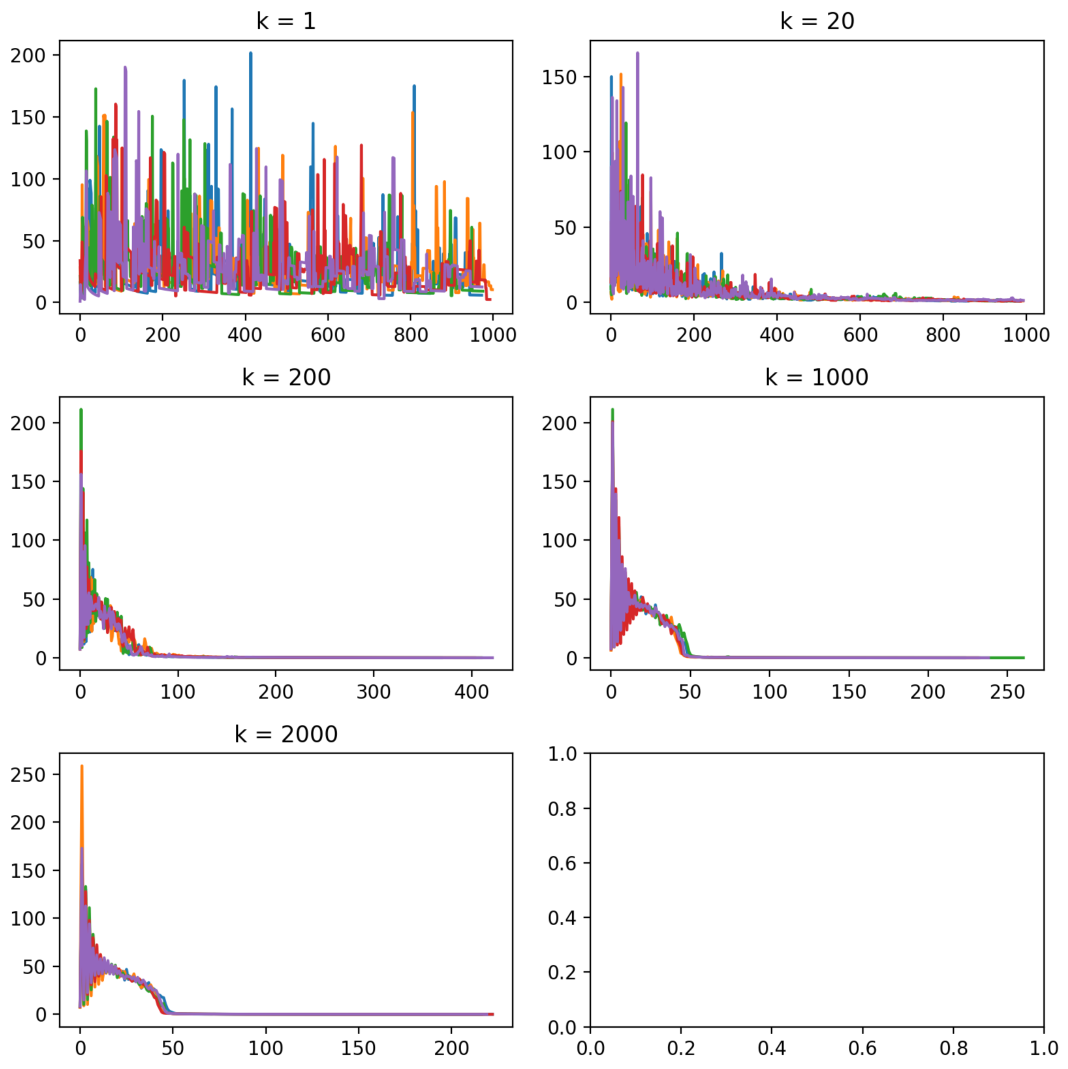
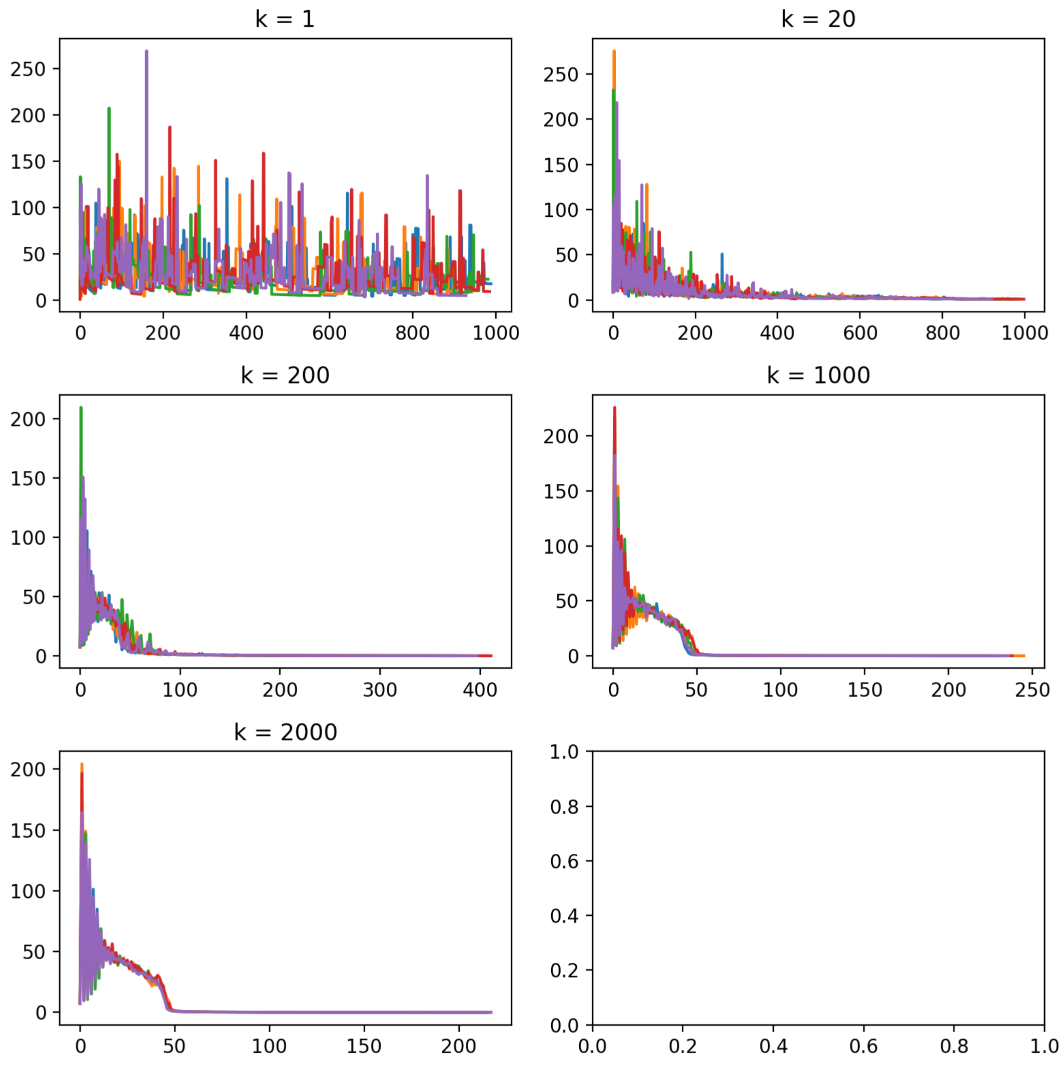


Figure 3: Softplus Algorithm Primal Objective Convergence vs. Number of Iterations


k = 1
k = 20
k = 200
k = 1000
k = 2000