Bhavani (Rohit) Murakonda
CSCI 5525 HW 2

1. Support Vector Machine Implementation using CVXOPT

a) Implement Dual Form SVM using CVXOPT

A linear Support Vector Machine (SVM) is a supervised learning model that separates labeled data using a linear decision boundary (hyperplane) while maximizing the margin separating the data from the hyperplane. The linear decision boundary line is represented by the equation:

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

To find the optimal decision boundary, the following constrained optimization problem is carried out:

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

thus maximizing the class separation with the lowest loss. The regularized loss function for the constrained optimization is:

$$\min_{\boldsymbol{w}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i f(\boldsymbol{x}_i)\} + \ \lambda \|\boldsymbol{w}\|^2 \right\}$$

where the first term represents the margin loss (the misclassification of a point over the hyperplane) and the regularization term. In order to make the classification less strict, a slack variable (ξ_i) can be added to the constraint as follows:

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

to allow the model to be robust to a small degree of misclassification and widen the margin. This problem can be solved using Lagrange multipliers under the Karush-Kuhn-Tucker (KKT) conditions to result in the following problem:

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

with the following constraints:

$$w = \sum_i \alpha_i y_i \boldsymbol{x_i}$$

$$\sum_i \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C$$

In order to numerically solve the problem with the given constraints, the CVXOPT quadratic convex optimizer, which has the form:

$$\min_x \frac{1}{2} x^T P x + \ q^T x$$

$$subject\ to\ Gx \ \leq h$$

$$Ax = b$$

and uses {P,q,G,h,A,b} as parameters to solve for $\boldsymbol{\alpha}$. The $\boldsymbol{\alpha} > 0$ correspond to support vectors, which are used to calculate the weight vector and thus the decision boundary and margin.

b) Dual Form SVM on MNIST-13 dataset

i) The results from implementing the Dual Form SVM described above on the MNIST-13 dataset are shown in Table 1 and Figure 1.

	C = 0.01	C = 0.1	C = 1	C = 10	C = 100
# Support Vectors (AVG)	1133.800000	342.200000	140.900000	94.800000	94.900000
# Support Vectors (STD)	5.963221	3.789459	6.977822	4.069398	5.940539
Training Error (AVG)	0.024000	0.015875	0.009125	0.001625	0.000000
Training Error (STD)	0.001858	0.001561	0.001125	0.000976	0.000000
Testing Error (AVG)	0.027750	0.017000	0.010500	0.013250	0.014750
Testing Error (STD)	0.007370	0.004000	0.007228	0.004039	0.005640
Margin (AVG)	0.354288	0.217212	0.122713	0.054773	0.040437
Margin (STD)	0.001244	0.001034	0.001263	0.001106	0.002152

Table 1: Results from implementing the Dual Form SVM on the MNIST-13 dataset for different C values.

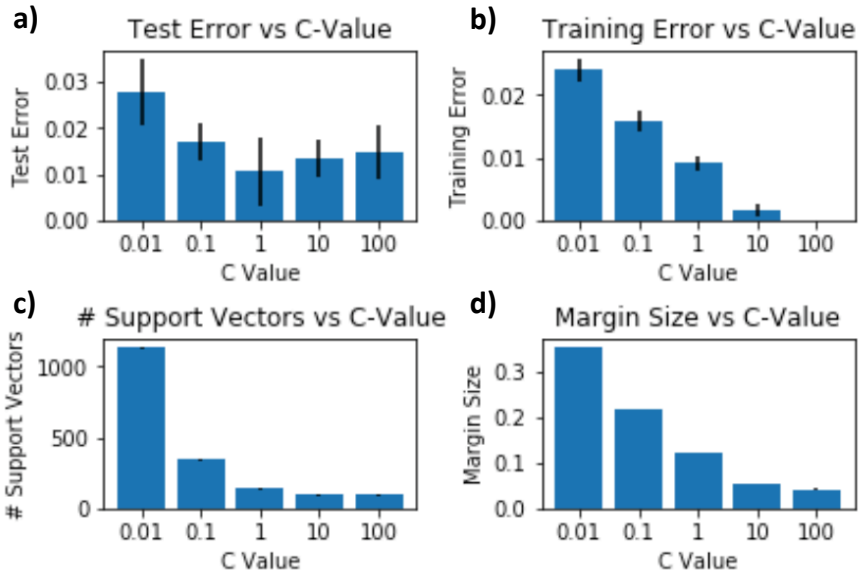

Figure 1: Graphs of the results from Table 1

ii) As C increases, the test performance of the model increases (i.e. the error decreases). As seen in Figure 1a, as C increases from 0.01 to 1, the test error of the model decreases but remains relatively unchanged as C increases from 1 to 100.

iii) As C increases, the geometric margin of the model decreases (Figure 1d). This correlates to the fact that increasing C reduces the slack of the model, making the model less tolerant to error (i.e. the margin size becomes smaller).

iv) As seen in Figure 1c, the number of support vectors decreases as C increases. As the model has less slack, less support vectors are available since less alpha vectors satisfy the constraint of ($\alpha > 0$).

c) Answer Questions

The slack variable function for the dual form SVM is $\xi_i(w,b) = \max\left(0, 1 - y_i\left(\boldsymbol{w^T x_i} + b\right)\right)$, which is known as the hinge loss. Thus the hinge loss function can be used to determine the optimal ξ_i. Additionally, the new definition of the slack variable must satisfy the margin constraint: $y_i(w^T x_i + b) \geq 1 - \xi_i$.

If $\xi_i(w,b) = 0$:

$$y_i\left(\boldsymbol{w^T x_i} + b\right) \geq 1$$

This is a constraint associated with SVM with no slack and the point survives.

If $\xi_i(w,b) = 1 - y_i\left(\boldsymbol{w^T x_i} + b\right)$:

$$y_i\left(\boldsymbol{w^T x_i} + b\right) \geq y_i\left(\boldsymbol{w^T x_i} + b\right)$$
$$y_i\left(\boldsymbol{w^T x_i} + b\right) = y_i\left(\boldsymbol{w^T x_i} + b\right)$$

This is true and thus the constraint is satisfied and thus the point survives.

2. SVM Implementation on Primal Problem using Stochastic Gradient Descent

Primal Estimated sub-GrAdient Solver for SVM (Pegasos)

To solve the primal objective function of SVM, the Pegasos algorithm performs T iterations and alternates between stochastic subgradient descent steps and projection steps for k examples of data (A_t) per step. The corresponding objective function is:

$$f(\boldsymbol{w}; A_t) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{k}\sum_{(\boldsymbol{x},y)\in A_t} \max\{0, 1 - y_i\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle\}$$

The pseudocode of the algorithm is shown below in Figure 2. The algorithm runs for five iterations to analyze the run time of the algorithm for $k = 1, k = 20, k = 200, k = 1000, k = 2000$ to determine the effects of increasing the number of examples used at each step. The results of the algorithm are detailed below in Table 2 and Figure 3. The algorithm ran for either 200,000 steps or when the loss function decreases below 10^{-3}.

INPUT: S, λ, T, k
INITIALIZE: Choose $\mathbf{w}_1$ s.t. $\|\mathbf{w}_1\| \leq 1/\sqrt{\lambda}$
FOR $t = 1, 2, \ldots, T$
  Choose $A_t \subseteq S$, where $|A_t| = k$
  Set $A_t^+ = \{(\mathbf{x}, y) \in A_t : y\langle \mathbf{w}_t, \mathbf{x}\rangle < 1\}$
  Set $\eta_t = \frac{1}{\lambda t}$
  Set $\mathbf{w}_{t+\frac{1}{2}} = (1 - \eta_t \lambda)\mathbf{w}_t + \frac{\eta_t}{k}\sum_{(\mathbf{x},y)\in A_t^+} y\mathbf{x}$
  Set $\mathbf{w}_{t+1} = \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+\frac{1}{2}}\|}\right\}\mathbf{w}_{t+\frac{1}{2}}$
OUTPUT: $\mathbf{w}_{T+1}$

Figure 2: The Pegasos Algorithm

	k = 1	**k = 20**	**k = 200**	**k = 1000**	**k = 2000**
Mean Training Time (s)	1824.7484	392.8032	90.6761	96.1068	113.1653
STD Training Time (s)	17.5884	10.4727	3.2334	0.6800	0.6656

Table 2: The results of the Pegasos algorithm run time for various values of k (number of training examples used in each mini-batch).

Analysis:

It can be seen that the run time generally decreases as k increases since the algorithm converges to the threshold loss value earlier. However, as seen in Figure 3, the run time of the algorithm increases for k = 1000 and k = 2000 while the number of iterations remains approximately the same. This would indicate that the runtime of each iteration is lower when k is lower but the runtime of the overall algorithm decreases. Additionally, it can be seen that the loss function is less likely to oscillate as k increases since the loss is computed over more sample points.

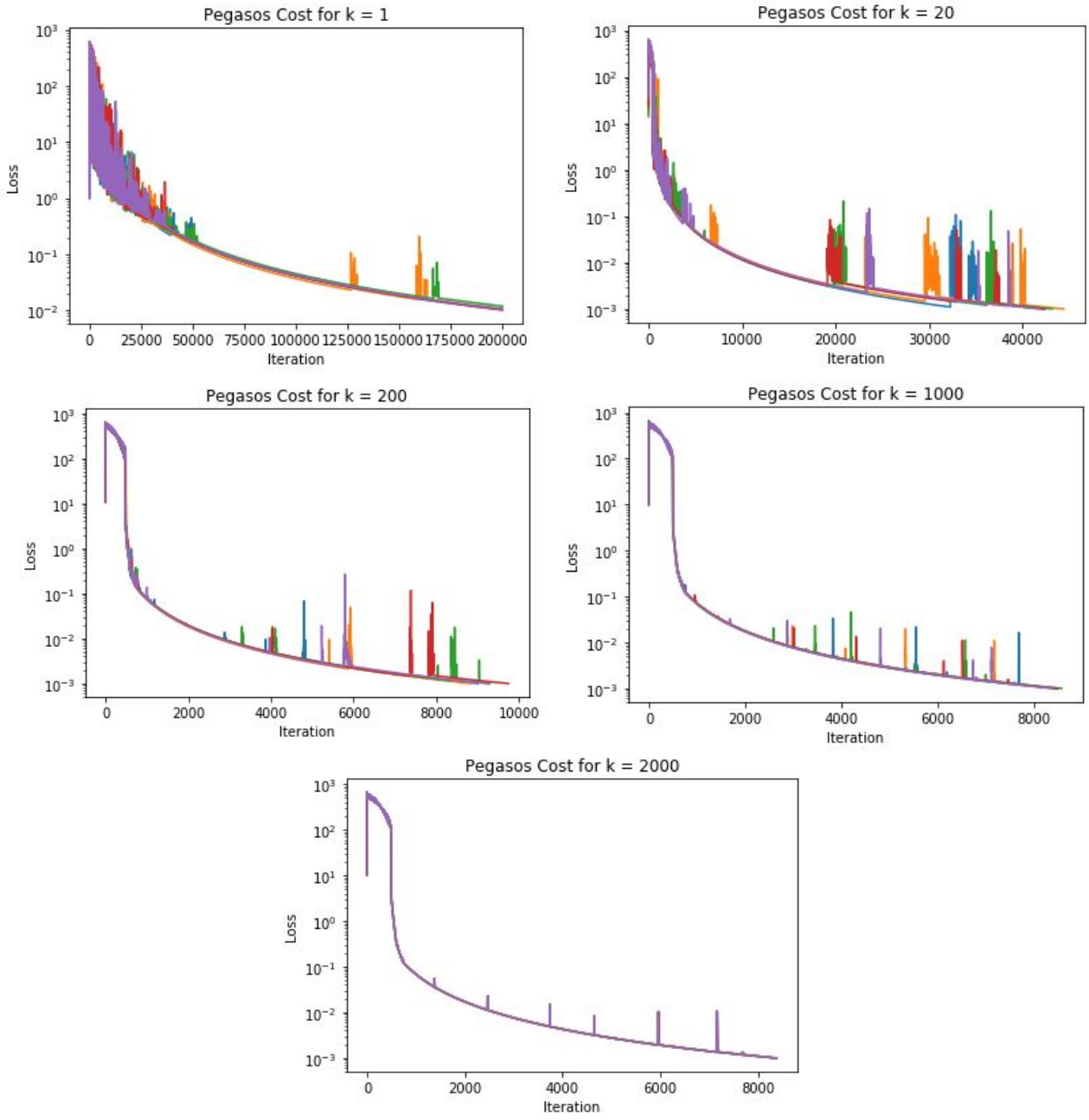

Figure 3: The loss function plots of the five runs for different values of k (number of training examples used in each mini-batch) for the Pegasos algorithm.

Gradient Descent Algorithm using a Soft-Plus Function

The primal SVM problem can also be solved using a softened hinge loss function known as "soft-plus" to compute the gradient. Since the primal loss function has the form,

$$L(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1:N} \max(0, 1 - y_i \boldsymbol{w^T x_i}) + \lambda\|w\|^2$$

the gradient can be computed using the soft-plus function:

$$softplus_a(x) = a\ log\left(1 + \exp\left(\frac{x}{a}\right)\right)$$

to result in the following gradient descent equation:

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

This derivative can be solved as follows:

$$\nabla L(w) = \frac{1}{N}\sum_{i=1:N}\left[a\left(\frac{\exp\left(\frac{1 - y_i \boldsymbol{w^T x_i}}{a}\right)}{1 + \exp\left(\frac{1 - y_i \boldsymbol{w^T x_i}}{a}\right)}\right) * \frac{\delta}{\delta w}\left(-\frac{y_i \boldsymbol{w^T x_i}}{a}\right) + \frac{\lambda}{2}w\right]$$

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

The results of the algorithm are found in Table 3 and Figure 4.

	k = 1	**k = 20**	**k = 200**	**k = 1000**	**k = 2000**
Mean Training Time (s)	11.3881	36.6367	203.4776	1364.9859	3391.5906
STD Training Time (s)	0.0452	0.1774	9.4436	59.2615	177.3516

Table 3: The results of the Gradient Descent, using a Soft-Plus function, algorithm run time for various values of k (number of training examples used in each mini-batch).

Analysis:

As seen in Table 3, the runtime of the algorithm increases as k increases when all iterations were run to 200,000 computations. However, as seen in Figure 4, the loss of the function is lower for lower values of k.

Comparison:

While the Pegasos algorithm converges to a lower loss value than the soft-plus gradient descent model, the runtime of the algorithm is much longer than the runtime of the second algorithm. Therefore, the Pegasos algorithm has a smaller rate of convergence but converges on a lower loss value but the soft-plus gradient descent model converges as a larger rate but converges at a higher loss value.

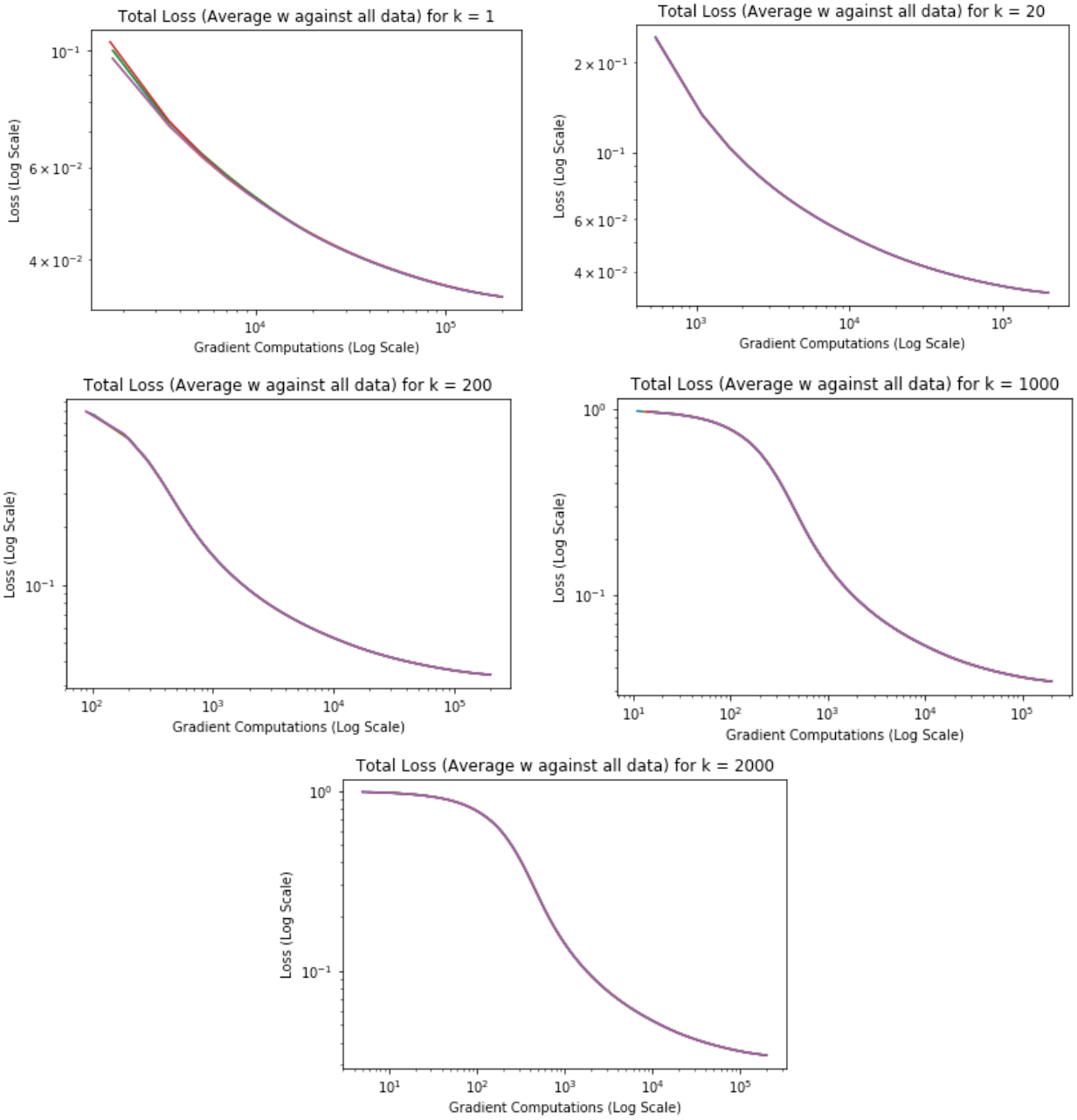

Figure 4: The loss function plots of the five runs for different values of k (number of training examples used in each mini-batch) for the Gradient Descent algorithm.