1.a

Consider the following training dataset:

$x_1 = (0, 0), y_1 = -1$

$x_2 = (1, 1), y_2 = -1$

$x_3 = (0, 1), y_3 = +1$

$x_4 = (1, 0), y_4 = +1$

$x_5 = (0, 2), y_5 = +1$

1.b

Let $w = (0, 0)$

Let $b = 1$

Let $\xi_i = 1, \forall i \in [1, n]$

$$y_i(x_i \cdot w + b) = y_i(0) = 0, \forall i \in [1, n] \tag{1.b.1}$$

$$1 - \xi_i = 1 - 1 = 0, \forall i \in [1, n] \tag{1.b.2}$$

Combining equations (1.b.1) and (1.b.2), we have:

$$y_i(x_i \cdot w + b) = 1 - \xi_i, \forall i \in [1, n] \tag{1.b.3}$$

For any x_i we plug into equation (1.b.3), we have:

$y_i(x_i \cdot w + b) = 1 - \xi_i, \forall i \in [1, n]$

It follows that any x_i satisfies the following constraint:

$y_i(x_i \cdot w + b) \geq 1 - \xi_i, \forall i \in [1, n]$

1.c

We are given that $(w, b, \xi_1, \dots, \xi_n)$ is a feasible point in soft margin SVM. Therefore:

$$y_i(x_i \cdot w + b) = 1 - \xi_i, \forall i \in [1, n] \tag{1.c.1}$$

We are given that an error is made by the classification rule on w on an example (x_i, y_i) when:

$$y_i(x_i \cdot w + b) < 0 \tag{1.c.2}$$

By combining equations (1.c.1) and (1.c.2), we solve for the slack variable ξ_i when an error occurs as follows:

$$y_i(x_i \cdot w + b) = 1 - \xi_i, \text{where } y_i(x_i \cdot w + b) < 0$$

$$\xi_i = 1 - y_i(x_i \cdot w + b), \text{where } y_i(x_i \cdot w + b) < 0$$

It follows that when an error occurs as defined by equation (1.c.2):

$$\xi_i > 1 \tag{1.c.3}$$

By combining equations (1.c.1) and (1.c.2), we solve for the slack variable ξ_i when the classification is correct as follows:

$$y_i(x_i \cdot w + b) = 1 - \xi_i, \text{where } y_i(x_i \cdot w + b) \geq 0$$

$$\xi_i = 1 - y_i(x_i \cdot w + b), \text{where } y_i(x_i \cdot w + b) \geq 0$$

$$\xi_i \in \mathbb{R}$$

However, we know that by definition $\xi_i \geq 0$

It follows that when the classification is correct:

$$\xi_i \geq 0 \tag{1.c.4}$$

$$\text{Let } A = \sum_{i=1}^{n} [1 \text{ if } y_i(x_i \cdot w + b) < 0;\ 0 \text{ otherwise}] = \#\text{ of errors}$$

A is equivalent to the number of errors.

$$\text{Let } B = \sum_{i=1}^{n} [\xi_i \text{ if } y_i(x_i \cdot w + b) < 0;\ 0 \text{ otherwise}]$$

B is equivalent to sum of all the slack variables for all points that produce an error.

$$\text{Let } C = \sum_{i=1}^{n} [\xi_i \text{ if } y_i(x_i \cdot w + b) \geq 0;\ 0 \text{ otherwise}]$$

C is equivalent to sum of all the slack variables for all points that are classified correctly.

It follows from the definitions of B and C that:

$$B + C = \sum_{i=1}^{n} \xi_i \tag{1.c.5}$$

We know from equation (1.c.3) that $\xi_i > 1$ for all points in error. It follows that:

$$A \leq B \tag{1.c.6}$$

We know from equation (1.c.4) that $\xi_i \geq 0$ for all points correctly classified. It follows that:

$$C \geq 0 \tag{1.c.7}$$

Equations (1.c.6) and (1.c.7) imply that:

$$A \leq B + C$$

Substituting the definition of A and equation (1.c.5), we obtain:

$$\#\text{ of errors} \leq \sum_{i=1}^{n} \xi_i$$

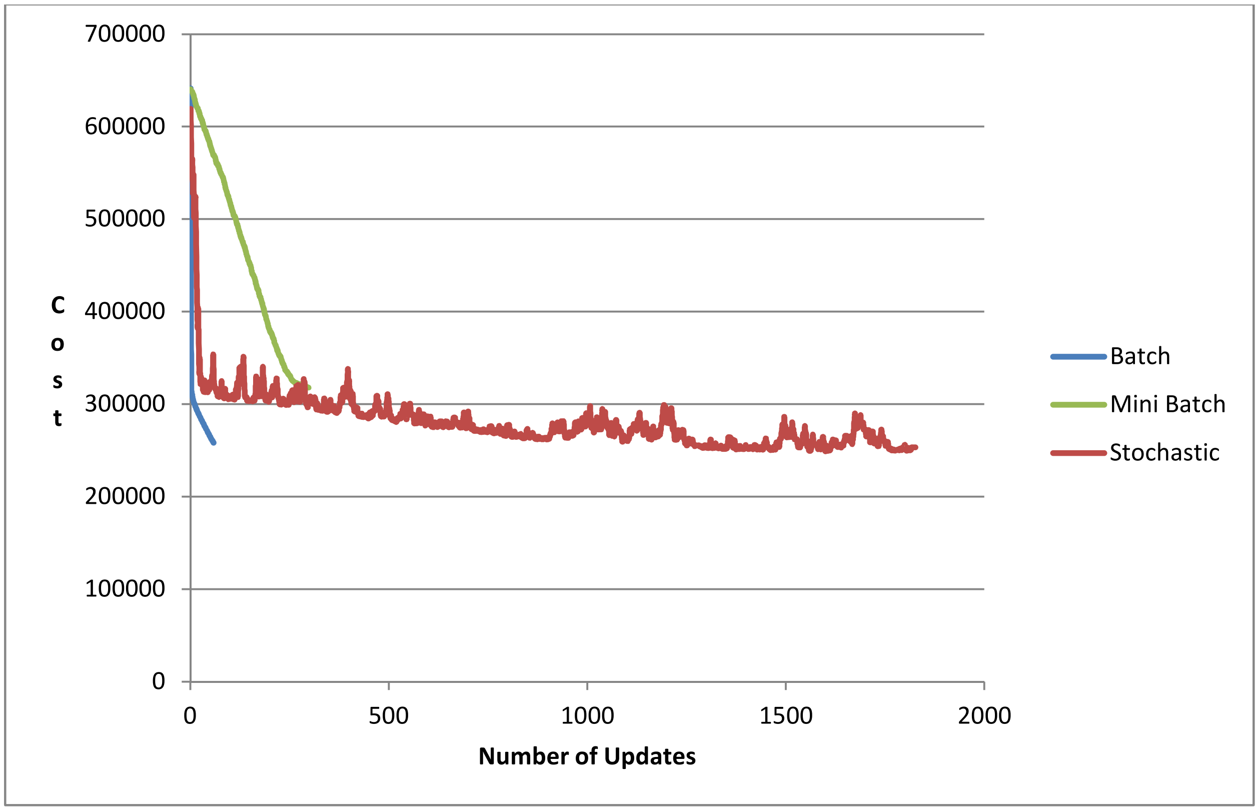
1.d

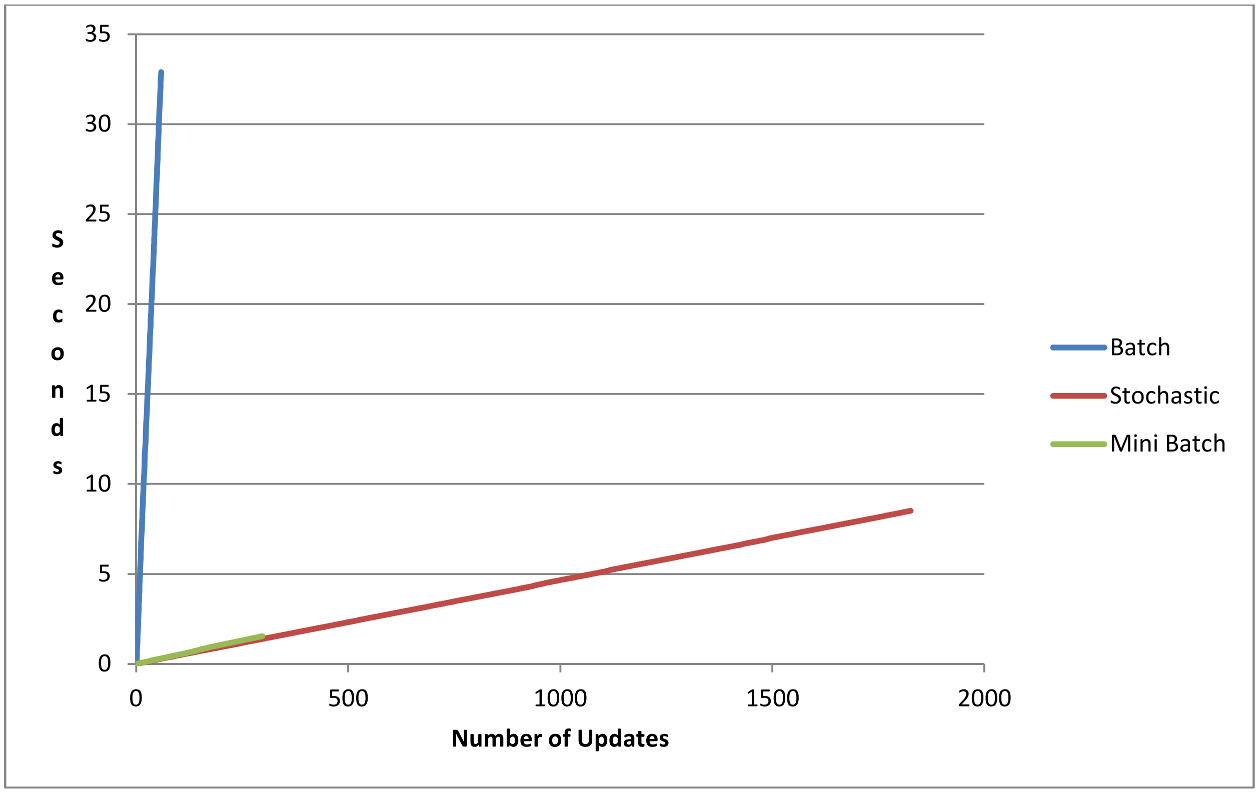
$$\nabla_b f(w, b) = \frac{\partial f(w, b)}{\partial b} = C \sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial b}$$

Where $\frac{\partial L(x_i, y_i)}{\partial b} = 0$ when $y_i(x_i \cdot w + b) \geq 1$

$\frac{\partial L(x_i, y_i)}{\partial b} = -y_i$ when $y_i(x_i \cdot w + b) < 1$

1.e



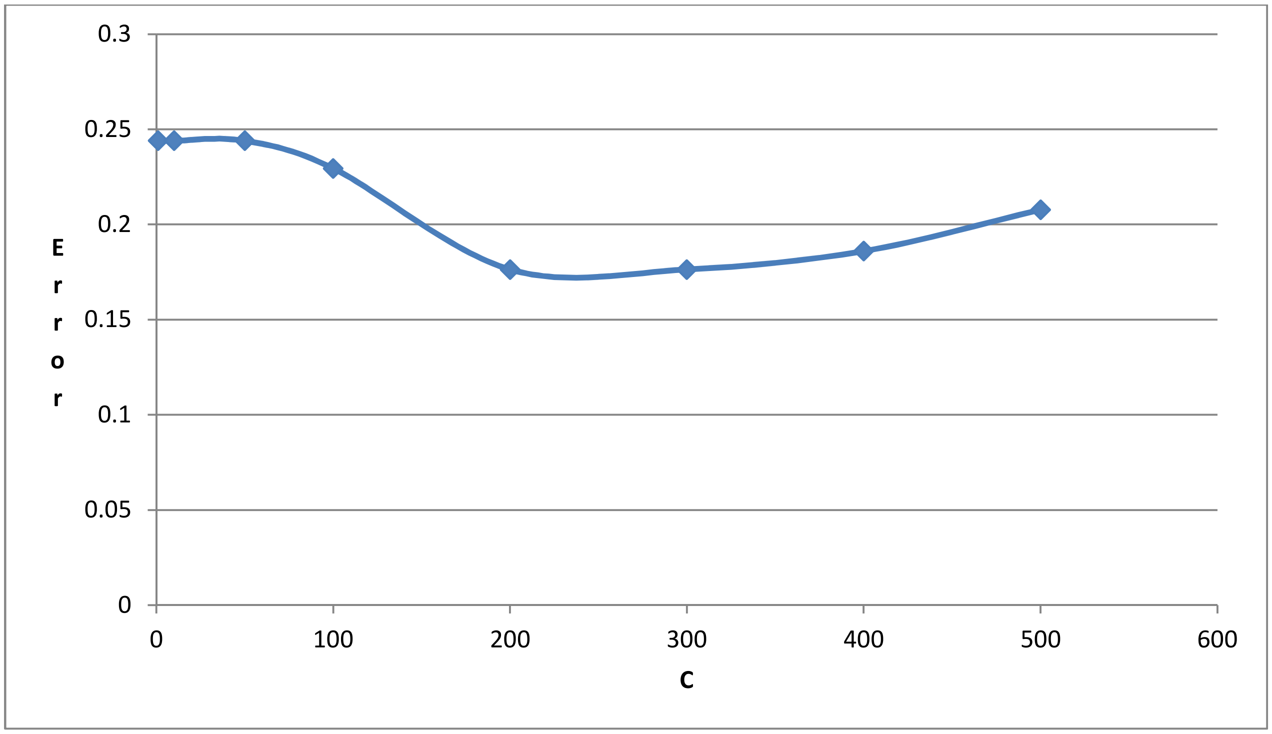


From the above plots, I made the following observations:

1) Mini Batch Gradient Descent is only slightly slower than Stochastic Gradient Descent in calculating updates.
2) Batch Gradient Descent takes much longer to perform an update than both Mini Batch Gradient and Stochastic gradient descent.
3) Batch Gradient Descent converges in a relatively small number of updates, however these updates can take a long time.
4) Stochastic Gradient Descent takes a much larger number of updates than Mini Batch Gradient Descent to converge.

Given these observations, I infer that in many or most cases Mini Batch Gradient Descent is preferable to Stochastic Gradient Descent because it performs updates quickly (almost as fast as Stochastic Gradient Descent) and can be expected to converge in a much smaller number of iterations than Stochastic Gradient Descent. However, Stochastic Gradient Descent may be preferable when we are working with very large amounts of data because its memory requirements are obviously lower. I further infer that both Mini Batch Gradient Descent and Stochastic Gradient Descent are typically preferable to Batch Gradient Descent because the update time for Mini Batch Gradient Descent and Stochastic Gradient Descent are so much faster.

1.f



It appears from the above plot that making C somewhere in the middle (about 200 to 300) produces the smallest error. This makes sense because making C too small over prioritizes minimizing w and b in relation to minimizing training mistakes. Making C too large prioritizes over prioritizes minimizing training mistakes in relation to minimizing w and b. Making C something in the middle should give us a good weighting of minimizing training mistakes in relation to minimizing w and b.

2.1.a

Let D_1 be the itemset after division where the binary attribute has value 1

$$I(D_1) = |x+y|\left[1-\left(\left(\frac{x}{x+y}\right)^2+\left(\frac{y}{x+y}\right)^2\right)\right]$$

$$I(D_1) = (x+y)\left[1-\left(\frac{x^2+y^2}{(x+y)^2}\right)\right]$$

$$I(D_1) = (x+y)\left[\frac{(x+y)^2-x^2-y^2}{(x+y)^2}\right]$$

$$I(D_1) = \frac{x^2+2xy+y^2\ -\ x^2-y^2}{(x+y)}$$

$$I(D_1) = \frac{2xy}{x+y}$$

Let D_0be the itemset after division where the binary attribute has value 0

$$I(D_0) = |u+v|\left[1-\left(\left(\frac{u}{u+v}\right)^2+\left(\frac{v}{u+v}\right)^2\right)\right]$$

By the logic we used to derive $I(D_1)$ above, it follows that the above expression reduces to the following:

$$I(D_0) = \frac{2uv}{u+v}$$

Let D be the itemset prior to division

$$I(D) = |u+v+x+y|\left[1-\left(\left(\frac{u+x}{u+v+x+y}\right)^2+\left(\frac{v+y}{u+v+x+y}\right)^2\right)\right]$$

$$I(D) = |(u+x)+(v+y)|\left[1-\left(\left(\frac{u+x}{(u+x)+(v+y)}\right)^2+\left(\frac{v+y}{(u+x)+(v+y)}\right)^2\right)\right]$$

By the logic we used to derive $I(D_1)$ above, it follows that the above expression reduces to the following:

$$I(D) = \frac{2(u+x)(v+y)}{(u+x)+(v+y)}$$

$$I(D) = \frac{2(u+x)(v+y)}{u+v+x+y}$$

Let G_A (G for binary attribute A) be defined as follows:

$$G_A = I(D) - \left(I(D_0)+I(D_1)\right)$$

$$G_A = I(D) - I(D_0) - I(D_1)$$

$$G_A = \frac{2(u+x)(v+y)}{u+v+x+y} - \frac{2uv}{u+v} - \frac{2xy}{x+y}$$

$$G_A = 2\left(\frac{(u+x)(v+y)}{u+v+x+y} - \frac{uv}{u+v} - \frac{xy}{x+y}\right)$$

We know that the attribute is "useful" when $G_A > 0$, which occurs when:

$$2\left(\frac{(u+x)(v+y)}{u+v+x+y} - \frac{uv}{u+v} - \frac{xy}{x+y}\right) > 0$$

$$\frac{(u+x)(v+y)}{u+v+x+y} - \frac{uv}{u+v} - \frac{xy}{x+y} > 0$$

2.1.b

$$G_{wine} = 2\left(\frac{(30+30)(20+20)}{30+20+30+20} - \frac{30*20}{30+20} - \frac{30*20}{30+20}\right) = 0$$

$$G_{running} = 2\left(\frac{(40+20)(30+10)}{40+30+20+10} - \frac{40*30}{40+30} - \frac{20*10}{20+10}\right) = 0.380952380952381$$

$$G_{pizza} = 2\left(\frac{(10+50)(10+30)}{10+10+50+30} - \frac{10*10}{10+10} - \frac{50*30}{50+30}\right) = 0.5$$

$G = \max(G_A \mid A \in \{wine, running, pizza\}) = 0.5 = G_{pizza}$

Choose the pizza attribute.

2.1.c

In 99% of the training dataset, the value of a_1 accurately predicts the target value, which is clearly statistically significant. Therefore, the root of the tree should split on attribute a_1.

To avoid overfitting, we should be careful to avoid making many more splits unless there is a very compelling case for doing so. Thus, our tree will probably be relatively shallow in depth.

2.2.d

Let Z_0 be the leftmost node labeled Z in figure 1.

Let Z_1 be the rightmost node labeled Z in figure 1.

Let Y_0 be the leftmost node labeled Y in figure 1.

Let Y_1 be the rightmost node labeled Y in figure 1.

Let $\alpha(T, w)$ be the increase in apparent error rate of pruning node w from tree T.

$\min[\alpha(T_0, w) \mid w \in T_0] = \alpha(T_0, Z_1) = 0.0$

$T_1 = T_0 - Z_1$

$\min[\alpha(T_1, w) \mid w \in T_1] = \alpha(T_1, Y_0) = 0.01086956522$

$T_2 = T_1 - Y_0$

$\min[\alpha(T_2, w) \mid w \in T_2] = \alpha(T_2, Y_1) = 0.04347826087$

$T_3 = T_2 - Y_1$

$\min[\alpha(T_3, w) \mid w \in T_3] = \alpha(T_2, X) = 0.17391304348$

$T_4 = T_3 - X$

2.2.e

T_3 has the best generalization error over the test dataset as T_3 does not produce any errors on the test dataset. $T_0, T_1, T_2,$ and T_4 all produce at least one error on the test dataset.

3.a

$(a + b)^2 \leq 2a^2 + 2b^2$

$a^2 + 2ab + b^2 \leq 2a^2 + 2b^2$

$2ab \leq a^2 + b^2$

$0 \leq a^2 - 2ab + b^2$

$0 \leq (a - b)^2$

$(a - b)^2$ is necessarily greater than or equal to 0 because any number squared is a positive number or zero.

3.b

Let $x \in S_{ij}$

Let $t' \in T \ s.t.\ d(t_{ij}, t') = d(t_{ij}, T)$

Recall that in question 3.a we proved:

$$(a+b)^2 \leq 2a^2 + 2b^2$$

And we know that triangle inequality holds for distances in Euclidean geometry, therefore:

$$d(x,t') \leq d(x,t_{ij}) + d(t_{ij},t') = d(x,t_{ij}) + d(t_{ij},T)$$

$$d(x,t') \leq d(x,t_{ij}) + d(t_{ij},T)$$

Because $t' \in T$ and by definition $d(x,t_{ij}) = d(x,T_i)$ it follows that:

$$d(x,T) \leq d(x,T_i) + d(t_{ij},T)$$

$$d(x,T)^2 \leq \left(d(x,T_i) + d(t_{ij},T)\right)^2$$

Applying the result from question 3.a we obtain:

$$d(x,T)^2 \leq 2d(x,T_i)^2 + 2d(t_{ij},T)^2$$

$$\sum_{x \in S_i} d(x,T)^2 \leq \sum_{x \in S_i} \left[2d(x,T_i)^2 + 2d(t_{ij},T)^2\right]$$

$$\textstyle\sum_{x \in S_i} d(x,T)^2 \leq 2\sum_{x \in S_i} d(x,T_i)^2 + 2\sum_{x \in S_i} d(t_{ij},T)^2 \qquad (3.b.1)$$

Applying equation (3.b.2) to (3.b.1), we have:

$$\sum_{x \in S_i} d(x,T)^2 \leq 2\sum_{x \in S_i} d(x,T_i)^2 + 2\sum_{x \in S_i} d(t_{ij},T)^2$$

$$\textstyle\sum_{x \in S_i} d(x,T)^2 \leq 2\sum_{x \in S_i} d(t_{ij},T)^2 + 2\sum_{x \in S_i} d(x,T_i)^2 \qquad (3.b.2)$$

By definition of our cost function:

$$\textstyle\sum_{x \in S_i} d(x,T_i)^2 = cost(S_i,T_i) \qquad (3.b.3)$$

Applying equation (3.b.3) to (3.b.2), we obtain:

$$\sum_{x \in S_i} d(x,T)^2 \leq 2 \sum_{x \in S_i} d\left(t_{ij},T\right)^2 + 2\, cost(S_i,T_i)$$

$$\sum_{x \in S_i} d(x,T)^2 \leq 2 \sum_j \left[\sum_{x \in S_{ij}} d\left(t_{ij},T\right)^2 \right] + 2\, cost(S_i,T_i)$$

$$\sum_{x \in S_i} d(x,T)^2 \leq 2 \sum_j \left[\left|S_{ij}\right| d\left(t_{ij},T\right)^2 \right] + 2\, cost(S_i,T_i)$$

$$\sum_i \left[\sum_{x \in S_i} d(x,T)^2 \right] \leq 2 \sum_i \left[\sum_j \left[\left|S_{ij}\right| d\left(t_{ij},T\right)^2 \right] \right] + 2 \sum_i cost(S_i,T_i)$$

$$\textstyle\sum_{x \in S} d(x,T)^2 \leq 2 \sum_i \left[\sum_j \left[\left|S_{ij}\right| d\left(t_{ij},T\right)^2 \right] \right] + 2 \sum_i cost(S_i,T_i) \qquad (3.b.4)$$

By definition of our cost function, we know:

$$\textstyle\sum_{x \in S} d(x,T)^2 = cost(S,T) \qquad (3.b.5)$$

Applying equation (3.b.5) to (3.b.4), we have:

$$cost(S,T) \leq 2 \textstyle\sum_i \left[\sum_j \left[\left|S_{ij}\right| d\left(t_{ij},T\right)^2 \right] \right] + 2 \sum_i cost(S_i,T_i) \qquad (3.b.6)$$

By definition of our cost function with weights:

$$cost_w\left(\hat{S},T\right) = \textstyle\sum_i \left[\sum_j \left[\left|S_{ij}\right| d\left(t_{ij},T\right)^2 \right] \right] \qquad (3.b.7)$$

Applying equation (3.b.7) to (3.b.6), we have:

$$cost(S,T) \leq 2 cost_w\left(\hat{S},T\right) + 2 \sum_{i=1}^{l} cost(S_i,T_i)$$

<u>3.c</u>

By definition of the ALG algorithm:

$cost(S_i, T_i) \leq \alpha \min_{|T'|=k}\{cost(S_i, T')\}$ (3.c.1)

We know that $|T^*| = k$, so it follows that:

$\min_{|T'|=k}\{cost(S_i, T')\} \leq cost(S_i, T^*)$ (3.c.2)

Combining equations (3.c.1) and (3.c.2) we obtain the following:

$cost(S_i, T_i) \leq \alpha \min_{|T'|=k}\{cost(S_i, T')\} \leq \alpha\, cost(S_i, T^*)$

$cost(S_i, T_i) \leq \alpha\, cost(S_i, T^*)$

$\sum_{i=1}^{l} cost(S_i, T_i) \leq \alpha \sum_{i=1}^{l} cost(S_i, T^*)$ (3.c.3)

By definition of the cost function (for w(x) = 1, which we have in this case):

$cost(S_i, T^*) = \sum_{x \in S_i} d(x, T^*)^2$ (3.c.4)

Combining equations (3.c.3) and (3.c.4) we have:

$$\sum_{i=1}^{l} cost(S_i, T_i) \leq \alpha \sum_{i=1}^{l} \sum_{x \in S_i} d(x, T^*)^2 = \alpha \sum_{x \in S} d(x, T^*)^2 = \alpha\, cost(S, T^*)$$

$$\sum_{i=1}^{l} cost(S_i, T_i) \leq \alpha\, cost(S, T^*)$$

3.d

By definition of the ALG algorithm:

$cost_w(\hat{S}, T) \leq \alpha \min_{|T'|=k}\{cost_w(\hat{S}, T')\}$ (3.d.1)

Because T^* is the optimal k-means solution it follows that:

$\min_{|T'|=k}\{cost_w(\hat{S}, T')\} \leq cost_w(\hat{S}, T^*)$ (3.d.2)

Combining equations (3.d.1) and (3.d.2), we obtain:

$$cost_w(\hat{S},T) \le \alpha \min_{|T'|=k}\{cost_w(\hat{S},T')\} \le \alpha\, cost_w(\hat{S},T^*)$$

$$cost_w(\hat{S},T) \le \alpha\, cost_w(\hat{S},T^*)$$

<u>3.e</u>

Let $x \in S_{ij}$

Let $t^* \in T^*$ be the corresponding centroid, i.e., $\mathrm{d}(x,t^*) = d(x,T^*)$

By the triangle inequality, which obviously applies in Euclidean space, we know:

$$\mathrm{d}(t_{ij},t^*) \le d(x,t^*) + d(x,t_{ij}) \tag{3.e.1}$$

We know by definition that:

$$\mathrm{d}(t_{ij},T^*) \le d(t_{ij},t^*) \tag{3.e.2}$$

Combining equations (3.e.1) and (3.e.2), we have:

$$\mathrm{d}(t_{ij},T^*) \le \mathrm{d}(t_{ij},t^*) \le d(x,t^*) + d(x,t_{ij})$$

$$\mathrm{d}(t_{ij},T^*) \le d(x,t^*) + d(x,t_{ij})$$

By our definition of t^*:

$$\mathrm{d}(t_{ij},T^*) \le d(x,T^*) + d(x,t_{ij}) \tag{3.e.3}$$

$x \in S_{ij}$ and t_{ij} is the center of S_{ij}, so:

$$\mathrm{d}(x,t_{ij}) = d(x,T_i) \tag{3.e.4}$$

Applying equation (3.e.4) to (3.e.3), we have:

$$\mathrm{d}\left(t_{ij}, T^*\right) \leq d(x, T^*) + d(x, T_i)$$

$$\mathrm{d}\left(t_{ij}, T^*\right)^2 \leq \left(d(x, T^*) + d(x, T_i)\right)^2$$

Applying what we proved in question 3.a, we have:

$$\mathrm{d}\left(t_{ij}, T^*\right)^2 \leq 2d(x, T^*)^2 + 2d(x, T_i)^2$$

$$\sum_{x\,\in S} \mathrm{d}\left(t_{ij}, T^*\right)^2 \leq 2\sum_{x\,\in S} d(x, T^*)^2 + 2\sum_{x\,\in S} d(x, T_i)^2$$

$$cost_w\left(\hat{S}, T^*\right) \ \leq 2\, cost(S, T^*) + 2\sum_{i}\sum_{x\,\in S_i} d(x, T_i)^2$$

$$cost_w\left(\hat{S}, T^*\right) \ \leq 2\, cost(S, T^*) + 2\sum_{i=1}^{l} cost(S_i, T_i)$$

$$cost_w\left(\hat{S}, T^*\right) \ \leq 2\sum_{i=1}^{l} cost(S_i, T_i) + \ 2\, cost(S, T^*)$$

3.f

Recall that in question 3.b we showed:

$$cost(S, T) \ \leq \ 2cost_w\left(\hat{S}, T\right) + \ 2\ \textstyle\sum_{i=1}^{l} cost(S_i, T_i) \qquad (3.f.1)$$

Recall that in question 3.d we showed:

$$cost_w\left(\hat{S}, T\right) \leq \ \alpha\, cost_w\left(\hat{S}, T^*\right) \qquad (3.f.2)$$

Combining equations (3.f.1) and (3.f.2), we have:

$$cost(S, T) \ \leq 2\alpha\, cost_w\left(\hat{S}, T^*\right) + \ 2\ \textstyle\sum_{i=1}^{l} cost(S_i, T_i) \qquad (3.f.3)$$

Recall that in question 3.e we showed:

$$cost_w\left(\hat{S}, T^*\right) \ \leq 2\ \textstyle\sum_{i=1}^{l} cost(S_i, T_i) + \ 2\, cost(S, T^*) \qquad (3.f.4)$$

Applying equation (3.f.4) to equation (3.f.3), we have:

$$cost(S,T) \leq 2\alpha \left[2 \sum_{i=1}^{l} cost(S_i,T_i) + 2\, cost(S,T^*)\right] + 2 \sum_{i=1}^{l} cost(S_i,T_i)$$

$$cost(S,T) \leq 4\alpha \textstyle\sum_{i=1}^{l} cost(S_i,T_i) + 4\alpha\, cost(S,T^*) + 2 \textstyle\sum_{i=1}^{l} cost(S_i,T_i) \tag{3.f.5}$$

Recall that in question 3.c we showed:

$$\textstyle\sum_{i=1}^{l} cost(S_i,T_i) \leq \alpha\, cost(S,T^*) \tag{3.f.6}$$

Applying equation (3.f.6) to equation (3.f.5), we have:

$$cost(S,T) \leq 4\alpha\, (\alpha\, cost(S,T^*)) + 4\alpha\, cost(S,T^*) + 2\, [\alpha\, cost(S,T^*)]$$

$$cost(S,T) \leq 4\alpha^2\, cost(S,T^*) + 6\alpha\, cost(S,T^*)$$

$$cost(S,T) \leq (4\alpha^2 + 6\alpha)\, cost(S,T^*)$$

<u>**3.g**</u>

Let $l = O\left(\sqrt{n/k}\right)$

The amount of memory for each partition S_i is proportional to the size of the partition, which is as follows:

$$|S_i| = O\left(\frac{n}{l}\right) = O\left(\frac{n}{\sqrt{n/k}}\right) = O\left(\sqrt{nk}\right)$$

And the amount of memory required for the clustering over set $\hat{S}$ is as follows:

$$O(lk) = O\left(k\sqrt{n/k}\right) = O\left(\sqrt{nk}\right)$$

Accordingly, ALGSTR requires $O\left(\sqrt{nk}\right)$ of memory space.

<u>**4.a**</u>

Every time i appears in the stream $c_{j,h_j(i)}$ for each j is incremented. Assuming our hash functions are deterministic, it follows that:

$$c_{j,h_j(i)} \geq F[i], \forall j \in \left[1, \left\lceil \ln \frac{1}{\delta} \right\rceil\right] \tag{4.a.1}$$

Because equation (4.a.1) holds true $\forall j$, it follows that:

$$\min_j c_{j,h_j(i)} \geq F[i]$$

$$\hat{F}[i] \geq F[i]$$

4.b

The probability of the element at time k being element i can be determined using $F[i]$, the number of times element i has appeared in the stream through time t, as follows:

$$P(a_k = i) = \frac{F[i]}{t} \tag{4.b.1}$$

$$P(a_k \neq i) = 1 - \frac{F[i]}{t} \tag{4.b.2}$$

Now we can determine the probability of $h_j(i) = h_j(a_k)$, i and a_k hashing to the same bucket, as follows:

$$P\left(h_j(i) = h_j(k)\right) = \frac{1}{\left\lceil \frac{e}{\epsilon} \right\rceil} \leq \frac{\epsilon}{e} \tag{4.b.3}$$

We know that $a_k \neq i$ and $h_j(i) = h_j(a_k)$ are independent events, as such we can calculate the probability of both occurring as follows:

$$P\left(a_k \neq i \text{ and } h_j(i) = h_j(k)\right) = P(a_k \neq i) P\left(h_j(i) = h_j(k)\right) \tag{4.b.4}$$

Applying equations (4.b.2) and (4.b.3) to (4.b.4), we obtain:

$$P\left(a_k \neq i \text{ and } h_j(i) = h_j(k)\right) \leq \frac{\epsilon}{e}\left(1 - \frac{F[i]}{t}\right)$$

Finally, we can calculate the expected value of $c_{j,h_j(i)}$ as follows:

$$E\left[c_{j,h_j(i)}\right] \le F[i] + E\left[\sum_{k=1}^{n} \frac{\epsilon}{e}\left(1 - \frac{F[i]}{t}\right)\right] = F[i] + \frac{\epsilon}{e}\,(t - F[i])$$

$$E\left[c_{j,h_j(i)}\right] \le F[i] + \tfrac{\epsilon}{e}\,(t - F[i]) \qquad (4.b.5)$$

4.c

$$E\left[c_{j,h_j(i)} - F[i]\right] = E\left[c_{j,h_j(i)}\right] - E\big[F[i]\big] \qquad (4.c.1)$$

Applying equation (4.b.5) to (4.c.1), we have:

$$E\left[c_{j,h_j(i)} - F[i]\right] \le F[i] + \frac{\epsilon}{e}\,(t - F[i]) - E\big[F[i]\big]$$

Because $F[i]$ is a constant it follows that $E\big[F[i]\big] = F[i]$, so the above equation can be rewritten as:

$$E\left[c_{j,h_j(i)} - F[i]\right] \le F[i] + \frac{\epsilon}{e}\,(t - F[i]) - F[i]$$

$$E\left[c_{j,h_j(i)} - F[i]\right] \le \frac{\epsilon}{e}\,(t - F[i])$$

We know that $F[i] \ge 0$, thus we can rewrite the above equation as follows:

$$E\left[c_{j,h_j(i)} - F[i]\right] \le \tfrac{\epsilon t}{e} \qquad (4.c.2)$$

Applying Markov's inequality to equation (4.c.2), we obtain:

$$P\left[c_{j,h_j(i)} - F[i] \ \ge \ \epsilon t\right] \le \frac{E\left[c_{j,h_j(i)} - F[i]\right]}{\epsilon t}$$

Substituting the definition of $E\left[c_{j,h_j(i)} - F[i]\right]$ from equation (4.c.2) to the above equation, we get:

$$P\left[c_{j,h_j(i)} - F[i] \ \ge \ \epsilon t\right] \le \tfrac{1}{e} \qquad (4.c.3)$$

Because we're assuming the events are all independent it follows that:

$$P\left[c_{j,h_j(i)} > F[i] + \epsilon t, \forall j\right] = \prod_j P\left(c_{j,h_j(i)} > F[i] + \epsilon t\right) = P\left(c_{j,h_j(i)} > F[i] + \epsilon t\right)^{\log\left(1/\delta\right)}$$

$$P\left[c_{j,h_j(i)} \leq F[i] + \epsilon t, \forall j\right] = 1 - P\left(c_{j,h_j(i)} > F[i] + \epsilon t\right)^{\log\left(1/\delta\right)} \qquad (4.c.4)$$

Applying equation (4.c.3) to (4.c.4), we get:

$$P\left[\hat{F}[i] \leq F[i] + \epsilon t, \forall j\right] \geq 1 - \left(\frac{1}{e}\right)^{\log\left(1/\delta\right)}$$

$$P\left[\hat{F}[i] \leq F[i] + \epsilon t, \forall j\right] \geq 1 - \delta$$

4.d

In Homework 3, Question 4 we needed to calculate the degree of each node in S in each iteration. In our previous implementation, we used $O(|S|)$ of memory to accomplish this by determining the degree of each node in S and storing it in memory.

The algorithm in the current question (HW4, Q4) can be used to reduce the memory requirements of the above mentioned algorithm (from HW3, Q4) by using the algorithm from the current question (HW4, Q4) to *approximate* the degrees of the nodes in S. This would reduce the memory requirement from $O(|S|)$ to $O(log|S|)$.

Of course the modification of the HW3, Q4 algorithm could specifically be accomplished by streaming the input file from HW3, Q4 into our algorithm from HW4, Q4.

```
/*
 * Philip Scuderi
 * Stanford University
 * CS246
 * Winter 2013
 * Homework 4
 * Question 1
 *
 * HW4.cs
 * This file is the main (top level) program.
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Diagnostics;
using Extreme.Mathematics;
using Extreme.Mathematics.LinearAlgebra;

namespace HW4
{
    class HW4
    {
        static void Main(string[] args)
        {
            Q1e();
            Q1f();

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void Q1e()
        {
            BatchGradientDescent();
            StochasticGradientDescent();
            MiniBatchGradientDescent();
        }

        static void Q1f()
        {
            string dataDirectory =
"C:\\Dropbox\\Private\\Stanford\\CS246\\Homeworks\\HW4\\Q1\\HW4-q1\\";

            Console.WriteLine("C\tTime Elapsed\t\tError");

            foreach (double c in new double[] { 1, 10, 50, 100, 200, 300, 400, 500 })
            {
                GradientDescent gd = new StochasticGradientDescent(dataDirectory +
"features.train.txt", dataDirectory + "target.train.txt", c);

                var results = gd.Iterate();

                var timeElapsed = results[results.Count - 1].Item3;
                var error = gd.GetError(dataDirectory + "features.test.txt",
dataDirectory + "target.test.txt");
```

```
                Console.WriteLine("{0}\t{1}\t{2}", gd.C, timeElapsed, error);
            }
        }

        static void BatchGradientDescent()
        {
            string dataDirectory =
"C:\\Dropbox\\Private\\Stanford\\CS246\\Homeworks\\HW4\\Q1\\HW4-q1\\";

            GradientDescent gd = new BatchGradientDescent(dataDirectory + "features.txt",
dataDirectory + "target.txt");

            using (System.IO.StreamWriter file = new
System.IO.StreamWriter("C:\\Dropbox\\Private\\Stanford\\CS246\\Homeworks\\HW4\\Q1\\output
\\BGD.txt"))
            {
                file.WriteLine(" k\t      Cost\t\t   Time\n--\t----------------\t--------
--");
                Console.WriteLine(" k\t      Cost\t\t   Time\n--\t----------------\t-----
-----");

                foreach (var result in gd.Iterate())
                {
                    file.WriteLine("{0}\t{1}\t{2}", result.Item1.ToString().PadLeft(2),
result.Item2.ToString().PadRight(16), FormatSeconds(result.Item3.TotalSeconds));
                    Console.WriteLine("{0}\t{1}\t{2}",
result.Item1.ToString().PadLeft(2), result.Item2.ToString().PadRight(16),
FormatSeconds(result.Item3.TotalSeconds));
                }
            }
        }

        static void StochasticGradientDescent()
        {
            string dataDirectory =
"C:\\Dropbox\\Private\\Stanford\\CS246\\Homeworks\\HW4\\Q1\\HW4-q1\\";

            GradientDescent gd = new StochasticGradientDescent(dataDirectory +
"features.txt", dataDirectory + "target.txt");

            Stopwatch stopwatch = new Stopwatch();
            var results = gd.Iterate();

            using (System.IO.StreamWriter file = new
System.IO.StreamWriter("C:\\Dropbox\\Private\\Stanford\\CS246\\Homeworks\\HW4\\Q1\\output
\\SGD.txt"))
            {
                Console.WriteLine("  k \t      Cost\t\t   Time\n----\t----------------\t-
---------");
                file.WriteLine("  k \t      Cost\t\t   Time\n----\t----------------\t----
------");

                foreach (var result in results)
                {
                    if (result.Item1 == 1 || result.Item1 == results.Count ||
(result.Item1 % 100 == 0))
                    {
                        Console.WriteLine("{0}\t{1}\t{2}",
result.Item1.ToString().PadLeft(4), result.Item2.ToString().PadRight(16),
FormatSeconds(result.Item3.TotalSeconds));

                    }

                    file.WriteLine("{0}\t{1}\t{2}", result.Item1.ToString().PadLeft(3),
result.Item2.ToString().PadRight(16), FormatSeconds(result.Item3.TotalSeconds));
                }
            }

            Console.WriteLine("\nStochastic Gradient Descent Complete.\n");
        }

        static void MiniBatchGradientDescent()
        {
            string dataDirectory =
"C:\\Dropbox\\Private\\Stanford\\CS246\\Homeworks\\HW4\\Q1\\HW4-q1\\";

            GradientDescent gd = new MiniBatchGradientDescent(dataDirectory +
"features.txt", dataDirectory + "target.txt");

            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            var results = gd.Iterate();
            stopwatch.Stop();

            using (System.IO.StreamWriter file = new
System.IO.StreamWriter("C:\\Dropbox\\Private\\Stanford\\CS246\\Homeworks\\HW4\\Q1\\output
\\MBGD.txt"))
            {
                file.WriteLine(" k \t      Cost\t\t   Time\n---\t----------------\t------
----");
                Console.WriteLine(" k \t      Cost\t\t   Time\n---\t----------------\t---
-------");

                foreach (var result in results)
                {
                    file.WriteLine("{0}\t{1}\t{2}", result.Item1.ToString().PadLeft(3),
result.Item2.ToString().PadRight(16), FormatSeconds(result.Item3.TotalSeconds));

                    if (result.Item1 % 10 == results.Count % 10)
                    {
                        Console.WriteLine("{0}\t{1}\t{2}",
result.Item1.ToString().PadLeft(3), result.Item2.ToString().PadRight(16),
FormatSeconds(result.Item3.TotalSeconds));
                    }
                }
            }

            Console.WriteLine("\nMini Batch Gradient Descent Complete.\n# of Iterations =
" + results.Count + "\nTotal Time = " + stopwatch.Elapsed + "\n");
        }

        static string FormatSeconds(double seconds)
        {
            string s = seconds.ToString();
            if (seconds < 10.0)
                s = " " + s;

            return s.PadRight(10);
        }
    }
}
```

```
/*
 * Philip Scuderi
 * Stanford University
 * CS246
 * Winter 2013
 * Homework 4
 * Question 1
 *
 * GradientDescent.cs
 * This file defines the abstract class GradientDescent.
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Diagnostics;
using Extreme.Mathematics;
using Extreme.Mathematics.LinearAlgebra;

namespace HW4
{
    public abstract class GradientDescent
    {
        protected Matrix x;
        protected Vector w, y;
        protected double b, eta, epsilon, c;
        protected int n, d;

        public abstract IList<Tuple<int, double, TimeSpan>> Iterate();

        public GradientDescent(string featuresPath, string targetPath, double eta, double
epsilon, double c)
        {
            x = ReadMatrix(featuresPath);
            y = ReadVector(targetPath);

            n = x.RowCount;
            d = x.ColumnCount;

            w = Vector.Create(d);

            Reset();

            this.c = c;
            this.eta = eta;
            this.epsilon = epsilon;
        }

        public void Reset()
        {
            for (int i = 0; i < d; i++)
                w[i] = 0.0;

            b = 0.0;
        }
        public double C
        {
            get { return c; }
        }

        public double GetError(string testFeaturesPath, string testTargetPath)
        {
            Matrix testFeatures = ReadMatrix(testFeaturesPath);
            Vector testTargets = ReadVector(testTargetPath);

            int numSuccessful = 0;

            for (int i = 0; i < testFeatures.Rows.Count; i++)
            {
                double prediction = GetPrediction(testFeatures.Rows[i]);

                if (prediction <= 0.0 && testTargets[i] <= 0.0)
                    ++numSuccessful;
                else
                {
                    if (prediction >= 0.0 && testTargets[i] >= 0.0)
                        ++numSuccessful;
                }
            }

            return 1.0 - ((double)numSuccessful / (double)testFeatures.Rows.Count);
        }

        protected double GetPrediction(Vector xi)
        {
            return Vector.DotProduct(w, xi) + b;
        }

        protected double GetCost()
        {
            double left = 0.0;
            for (int j = 0; j < d; j++)
            {
                left += Math.Pow(w[j], 2.0);
            }
            left *= 0.5;

            double right = 0.0;
            for (int i = 0; i < n; i++)
            {
                right += Math.Max(0.0, 1.0 - (y[i] * (Vector.DotProduct(w, x.Rows[i]) +
b)));
            }
            right *= c;


            return left + right;
        }

        protected static Matrix ReadMatrix(string filePath)
        {
            int n = 0;
            int d = 0;
            using (StreamReader reader = new StreamReader(filePath))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    d = Math.Max(d, line.Split(',').Length);
                    ++n;
                }
            }

            Matrix m = Matrix.Create(n, d);

            int lineNum = 0;
            using (StreamReader reader = new StreamReader(filePath))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    double[] numbers = Array.ConvertAll<string, double>(line.Split(','),
delegate(string s) { return double.Parse(s); });

                    for (int i = 0; i < numbers.Length; i++)
                        m[lineNum, i] = numbers[i];

                    ++lineNum;
                }
            }

            return m;
        }

        protected static Vector ReadVector(string filePath)
        {
            int n = 0;

            using (StreamReader reader = new StreamReader(filePath))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                    ++n;
            }

            Vector v = Vector.Create(n);

            int lineNum = 0;
            using (StreamReader reader = new StreamReader(filePath))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    v[lineNum] = double.Parse(line);

                    ++lineNum;
                }
            }

            return v;
        }

        protected static Permutation GenerateRandomPermutation(int n)
        {
            var list = new List<int>(n);

            for (int i = 0; i < n; i++)
                list.Add(i);

            var randomizedList = new List<int>(n);
            var rnd = new Random();
            while (list.Count != 0)
            {
                var index = rnd.Next(0, list.Count);
                randomizedList.Add(list[index]);
                list.RemoveAt(index);
            }

            int[] randomizedIndicies = randomizedList.ToArray();
            Permutation p = new Permutation(randomizedIndicies);
            return p;
        }
    }
}
```

```
/*
 * Philip Scuderi
 * Stanford University
 * CS246
 * Winter 2013
 * Homework 4
 * Question 1
 *
 * BatchGradientDescent.cs
 * This file defines the BatchGradientDescent class, which implements GradientDescent.
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Diagnostics;
using Extreme.Mathematics;
using Extreme.Mathematics.LinearAlgebra;

namespace HW4
{
    public class BatchGradientDescent : GradientDescent
    {
        public BatchGradientDescent(string featuresPath, string targetPath, double c =
100.0) : base(featuresPath, targetPath, 0.0000003, 0.25, c)
        { }

        override public IList<Tuple<int, double, TimeSpan>> Iterate()
        {
            IList<Tuple<int, double, TimeSpan>> results = new List<Tuple<int, double,
TimeSpan>>();

            int k = 1;
            double lastCost = double.NaN;
            double currentCost;

            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            do
            {
                currentCost = GetCost();

                results.Add(new Tuple<int, double, TimeSpan>(k, currentCost,
stopwatch.Elapsed));

                if (!double.IsNaN(lastCost))
                {
                    double percentCostChange = (Math.Abs(lastCost - currentCost) * 100.0)
/ lastCost;

                    if (percentCostChange < epsilon)
                        break;
                }

                for (int j = 0; j < d; j++)
                {
                    w[j] -= eta * GradientWrtWj(j);
                }

                b -= eta * GradientWrtB();

                ++k;
                lastCost = currentCost;
            } while (true);

            return results;
        }

        protected double GradientWrtB()
        {
            double sum = 0.0;

            for (int i = 0; i < n; i++)
            {
                if (y[i] * (Vector.DotProduct(x.Rows[i], w) + b) < 1.0)
                    sum += -y[i];
            }

            return c * sum;
        }

        protected double GradientWrtWj(int j)
        {
            double sum = 0.0;

            for (int i = 0; i < n; i++)
            {
                if (y[i] * (Vector.DotProduct(x.Rows[i], w) + b) < 1.0)
                    sum += -y[i] * x[i, j];
            }

            return w[j] + (c * sum);
        }
    }
}
```

```
/*
 * Philip Scuderi
 * Stanford University
 * CS246
 * Winter 2013
 * Homework 4
 * Question 1
 *
 * StochasticGradientDescent.cs
 * This file defines the StochasticGradientDescent class, which implements
GradientDescent.
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Diagnostics;
using Extreme.Mathematics;
using Extreme.Mathematics.LinearAlgebra;

namespace HW4
{
    public class StochasticGradientDescent : GradientDescent
    {
        public StochasticGradientDescent(string featuresPath, string targetPath, double c
= 100.0) : base(featuresPath, targetPath, 0.0001, 0.001, c)
        {
            // randomly permute the items
            Permutation p = GenerateRandomPermutation(n);
            x.Rows.Permute(p);
            y.Permute(p);
        }

        override public IList<Tuple<int, double, TimeSpan>> Iterate()
        {
            IList<Tuple<int, double, TimeSpan>> results = new List<Tuple<int, double,
TimeSpan>>();

            int i = 1;
            int k = 1;

            double lastCost = GetCost();
            double currentCost;

            double lastDeltaCost = 0.0;
            double deltaCost;

            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            do
            {
                for (int j = 0; j < d; j++)
                {
                    w[j] -= eta * GradientWrtWj(i-1, j);
                }
                b -= eta * GradientWrtB(i-1);


                // iteration ends, update the results
                currentCost = GetCost();
                double percentCostChange = (Math.Abs(lastCost - currentCost) * 100.0) /
lastCost;
                deltaCost = 0.5 * lastDeltaCost + 0.5 * percentCostChange;

                results.Add(new Tuple<int, double, TimeSpan>(k, currentCost,
stopwatch.Elapsed));


                if (deltaCost < epsilon)
                    break;


                i = (i % n) + 1;
                ++k;
                lastCost = currentCost;
                lastDeltaCost = deltaCost;
            } while (true);

            return results;
        }

        protected double GradientWrtB(int i)
        {
            double sum = 0.0;

            if (y[i] * (Vector.DotProduct(x.Rows[i], w) + b) < 1.0)
                sum += -y[i];

            return c * sum;
        }

        protected double GradientWrtWj(int i, int j)
        {
            double sum = 0.0;

            if (y[i] * (Vector.DotProduct(x.Rows[i], w) + b) < 1.0)
                sum += -y[i] * x[i, j];

            return w[j] + (c * sum);
        }
    }
}
```

```
/*
 * Philip Scuderi
 * Stanford University
 * CS246
 * Winter 2013
 * Homework 4
 * Question 1
 *
 * MiniBatchGradientDescent.cs
 * This file defines the MiniBatchGradientDescent class, which implements
GradientDescent.
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Diagnostics;
using Extreme.Mathematics;
using Extreme.Mathematics.LinearAlgebra;

namespace HW4
{
    public class MiniBatchGradientDescent : GradientDescent
    {
        protected int batchSize;

        public MiniBatchGradientDescent(string featuresPath, string targetPath, int
batchSize = 10, double c = 100.0) : base(featuresPath, targetPath, 0.000001, 0.01, c)
        {
            this.batchSize = batchSize;

            // randomly permute the items
            Permutation p = GenerateRandomPermutation(n);
            x.Rows.Permute(p);
            y.Permute(p);
        }

        override public IList<Tuple<int, double, TimeSpan>> Iterate()
        {
            IList<Tuple<int, double, TimeSpan>> results = new List<Tuple<int, double,
TimeSpan>>();

            int l = 1;
            int k = 1;

            double lastCost = GetCost();
            double currentCost;

            double lastDeltaCost = 0.0;
            double deltaCost;

            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            do
            {
                for (int j = 0; j < d; j++)
                {
                    w[j] -= eta * GradientWrtWj(l, j);
                }

                b -= eta * GradientWrtB(l);


                // iteration ends, update the results
                currentCost = GetCost();
                double percentCostChange = (Math.Abs(lastCost - currentCost) * 100.0) / 
lastCost;
                deltaCost = 0.5 * lastDeltaCost + 0.5 * percentCostChange;

                results.Add(new Tuple<int, double, TimeSpan>(k, currentCost, 
stopwatch.Elapsed));


                if (deltaCost < epsilon)
                    break;


                l = (l + 1) % ((n + batchSize - 2) / batchSize);
                ++k;
                lastCost = currentCost;
                lastDeltaCost = deltaCost;
            } while (true);

            return results;
        }

        protected double GradientWrtWj(int l, int j)
        {
            double sum = 0.0;

            for (int i = l*batchSize; i < Math.Min(n, (l+1)*batchSize); i++)
            {
                if (y[i] * (Vector.DotProduct(x.Rows[i], w) + b) < 1.0)
                    sum += -y[i] * x[i, j];
            }

            return w[j] + (c * sum);
        }

        protected double GradientWrtB(int l)
        {
            double sum = 0.0;

            for (int i = l * batchSize; i < Math.Min(n, (l + 1) * batchSize); i++)
            {
                if (y[i] * (Vector.DotProduct(x.Rows[i], w) + b) < 1.0)
                    sum += -y[i];
            }

            return c * sum;
        }
    }
}
```