1.)

A function $K: X \times X \to \mathbb{R}$ is valid if Symm. and +ve semi-definite function.

a) Let $K_1 \dots K_m$ be a set of valid Kernel functions.

to prove $K = \sum_{j=1}^{m} w_j K_j$ is also a valid Kernel function as long as $\boxed{w_j \geq 0}$

→ since $K_1 \dots K_m$ are valid Kernel functions

so $K_1 = \langle \phi_1(u), \phi_1(v) \rangle$

$w_1 K_1 = \langle (\sqrt{w_1}\,\phi_1(u)), (\sqrt{w_1}\,\phi_1(v)) \rangle$

$w_2 K_2 = \langle (\sqrt{w_2}\,\phi_2(u)), (\sqrt{w_1}\,\phi_2(v)) \rangle$

$\vdots$

$w_n K_n = \langle \sqrt{w_n}\,\phi_n(u), \sqrt{w_n}\,\phi_n(v) \rangle$

so then summation

$\sum_{i=1}^{n} w_i K_i$

$\Rightarrow \langle (\sqrt{w_1}\phi_1(u)\ \sqrt{w_2}\,\phi_2(u) \dots \sqrt{w_n}\,\phi_n(u)), (\sqrt{w_1}\,\phi_1(v)\ \sqrt{w_2}\,\phi_2(v) \dots \sqrt{w_n}\,\phi_n(v)) \rangle$

$\Rightarrow \langle f_1(u), f_2(v) \rangle$

Since $\sum_{i=1}^{n} w_i K_i$ can be written in terms of inner product of two functions f_1 and f_2, it is a valid Kernel function.

$$K = K_1 \odot K_2$$

$$= K_1(x,y)\, K_2(x,y)$$

$$= \langle \Phi_1(x), \Phi_1(y) \rangle \langle \Phi_2(x), \Phi_2(y) \rangle$$

$$= \left(\sum_i \Phi_1(x)_i \Phi_1(y)_i \right) \left(\sum_j \Phi_2(x)_j \Phi_2(y)_j \right)$$

$$= \sum_{ij} \underbrace{\Phi_1(x)_i \Phi_2(x)_j}\ \underbrace{\Phi_1(y)_i \Phi_2(y)_j}$$

$$= \sum_{ij} \Phi(x)_{ij}\, \Phi(y)_{ij}$$

$$K = \langle \Phi(x), \Phi(y) \rangle$$

This Result is often quoted as "Hadamard product in kernel space becomes a tensor product in the feature space".

(c) Considering the function

$$K(x, x') = (xx' + 1)^{2015}$$

The Binomial expansion of $(1+x)^{\alpha}$

$$= 1 + \alpha x + \frac{\alpha(\alpha-1)}{2} x^2 + \frac{\alpha(\alpha-1)(\alpha-2)}{3!} x^3 + \cdots$$

$$K(x,x') \Rightarrow 1 + w_1 \langle x, x' \rangle + w_2 (\langle x, x' \rangle)^2 + w_3 (\langle x, x' \rangle)^3 + \cdots + w_{2015} (\ )^{2015}$$

here again w_1 to w_n are > 0

from the previous 2 parts, we were able to say that

i) $\sum_{j=1}^{m} w_j k_j$ is a valid Kernel if $w \geq 0$

ii) $k = k_1 \cdot k_2$ is a valid Kernel

Using the above two properties, we can say that

$$\Rightarrow 1 + w_1 K_1 + w_2 K_2^2 + w_3 K_3^3 + \cdots + w_{2015} K_{2015}^{2015}$$

$$\Rightarrow (1) \xrightarrow{w_0} \xrightarrow{k_0} \langle 1, 1 \rangle + w_1 K_1 + w_2 K_2' + w_3 K_3' + \cdots + w_{2015} K_{2015}'$$

$$= \sum_{i=0}^{2015} w_i k_i$$

$\Rightarrow K$ (hence it is a valid Kernel)

(d) $K(x,y) = \exp\left(-(x-y)^2/2\right)$

$$K(x,y) = \exp\left(-\frac{1}{2}(x^2+y^2-2xy)\right)$$

$$\Rightarrow K(x,y) = \exp\left(-\frac{x^2}{2} - \frac{y^2}{2} + xy\right)$$

$$\Rightarrow K(x,y) = \exp\left(-\frac{x^2}{2}\right) \cdot \exp\left(-\frac{y^2}{2}\right) \cdot \exp(xy)$$

$$\Rightarrow K(x,y) = \underbrace{\langle \phi_1(x), \phi_1(y) \rangle}_{(A)} \cdot \underbrace{\exp\left(\langle \phi_2(x), \phi_2(y) \rangle\right)}_{(B)}$$

where: $\phi_1(x) = -\frac{x^2}{2}$ & $\phi_2(x) = x$

Lets call

(A) $= K_1(x,y)$

In (B). $e^{\langle \phi_2(x), \phi_2(y) \rangle} = 1 + \langle \phi_2(x), \phi_2(y) \rangle + \frac{1}{2}\left[\langle \phi_2(x), \phi_2(y) \rangle\right]^2 + \frac{1}{6}\left[\langle \phi_2(x), \phi_2(y) \rangle\right]^3 + \cdots$

$$e^{K_2(x,y)} = 1 + K_2 + \frac{1}{2}K_2^2 + \frac{1}{6}K_2^3 + \frac{1}{24}K_2^4 + \cdots$$

from 1st & 2nd sol^n, we were able to prove that

→ $\sum_{j=1}^{m} w_j K_j$ is a valid kernel if $w \geq 0$

& → $K_1 K_2$ is also a valid kernel.

Using both of these results above :-

⇒ $e^{K_2(x,y)}$ is also a valid kernel say $K_2'(x,y)$

so $K(x,y) = (A)(B) \Rightarrow K_1(x,y)\, K_2'(x,y)$ → again it is valid!

Question 2

SMO iteratively updates alpha by updating two components (alpha_i and alpha_j) in each iteration while keeping the remaining (n-2) components fixed.

- **Clearly describe the sub-problem on (alpha_i and alpha_j),**
- **How the sub-problem can be solved efficiently, and**
- **How the sub-problems are selected in each iteration.**
- **Report the average run-time over 5 runs on the entire dataset, along with the standard deviation.**
- **Plot the value of dual objective function with increasing number of iterations for each run. (i.e five plots overlaid on the same figure)**
    - The dual objective should increase over iterations to convergence.

Solution:
SMO is fastest for linear SVM and sparse data-sets. It is able to break a large Quadratic Programming problem into a series of smallest possible(2 variables) Quadratic Programming problems which can be solved analytically, thus avoiding the time consuming numerical QP optimization as an inner loop.

Each SVM problem has two loops to take care of. Outer loop and the Inner loop.
Generally the inner loop consists of solving the numerical QP optimization part. The outer loop is used to iterate over the training examples for finding the KKT violators.

The most difficult part for a SVM solver is to solve the following Quadratic Programming Optimization problem. This we got via the lagrange dual of the initial Max-margin problem.

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} y_i y_j k(\vec{x}_i, \vec{x}_j) \alpha_i \alpha_j,$$
$$0 \leq \alpha_i \leq C, \quad \forall i,$$
$$\sum_{i=1}^{\ell} y_i \alpha_i = 0.$$

This is the basic step at which SMO helps by taking the approach of solving it analytically rather than numerically.

Following the Paper by Osuna, one can infer that : "**A sequence of QP sub-problems that always add at least one violator (of the KKT condition) will asymptotically converge**"

SMO decomposes the overall QP problem into QP sub-problems similar to Osuna's method.

I will try to answer each part of the question with a separate sub-headings as below

Clearly describe the sub-problem on alpha_i and alpha_j

- SMO chooses to solve the smallest possible optimization problem at every step. This **smallest possible problem will involve at-least 2 Lagrange multipliers.** These 2 alphas are then jointly optimized and their optimal values are updated to the SVM at each iteration.

- As directed in the question, SMO iteratively updates alpha by updating (alpha_i, alpha_j) in each iteration while keeping the remaining (n-2) components fixed. For updating only on the domain of two alphas, we have to do a constrained maximization of the objective function.

- The bound constraints described in the above equation cause the Lagrange multipliers to lie within a box, while the linear equality constraint causes the Lagrange multipliers to lie on a diagonal line segment. For both of the constraints to be followed together, **2 is the minimum number of lagrange multipliers that can be optimized.** for if SMO optimized only one multiplier, it could not fulfil the linear equality constraint at every step.

- Hence, concluding from the above info, **we can say that the constrained maximum of the objective function must lie on a diagonal line segment.**

The algorithm:

- Computes the alpha_2 and the ends of the diagonal line segment in terms of alpha_2.
- Compute the unconstrained maximum of the objective function. Under normal circumstances, there will be a maximum along the direction of the linear equality constraint. If the second derivative of the objective function along the diagonal line is less than zero(which should happen normally), we get the maximum along the direction of the constraint as:

$$\alpha_2^{\text{new}} = \alpha_2^{\text{old}} - \frac{y_2(E_1 - E_2)}{\eta},$$

where the $E_{i's}$ are the error on the i^{th} training example. that is : $E_i = f^{\text{old}}(\vec{x}_i) - y_i$

- The above unconstrained maximum is clipped to give us the constrained maximum of the objective function using the following equation.:

$$\alpha_2^{\text{new,clipped}} = \begin{cases} H, & \text{if } \alpha_2^{\text{new}} \geq H; \\ \alpha_2^{\text{new}}, & \text{if } L < \alpha_2^{\text{new}} < H; \\ L, & \text{if } \alpha_2^{\text{new}} \leq L. \end{cases}$$

- If the current SVM's solution on data_point_1 is not equal to the solution on the data_point_2 (y_1 != y_2) then the bound applied to alpha_2 is different from that when they are equal. This decides the **H and L** that are used in the above equations.
    - If (y1 != y2), then the following bounds apply to alpha_2:
        - $$L = \max(0, \alpha_2^{\text{old}} - \alpha_1^{\text{old}}), \qquad H = \min(C, C + \alpha_2^{\text{old}} - \alpha_1^{\text{old}}).$$
    - If (y1 == y2), then the following bounds apply to alpha_2:
        - $$L = \max(0, \alpha_1^{\text{old}} + \alpha_2^{\text{old}} - C), \qquad H = \min(C, \alpha_1^{\text{old}} + \alpha_2^{\text{old}}).$$

- Now we compute the alpha_1 from the new clipped alpha_2.
    - $$\text{let } s = y_1 y_2.$$
    - $$\alpha_1^{\text{new}} = \alpha_1^{\text{old}} + s(\alpha_2^{\text{old}} - \alpha_2^{\text{new,clipped}}).$$

How the Sub-problems can be solved efficiently? & how are the alphas selected?

In order to speed convergence, SMO uses heuristics to choose which two Lagrange Multipliers to jointly optimize. It is described below

- The outer loop selects the first apha, the inner loop selects the second alpha.
- **First Choice Heuristics:** The outer loop alternates between one pass through all examples and as many passes as possible through the non-bound examples, selecting the example that violates the KKT condition.
- **Second Choice Heuristics:** GIven the first alpha, we do inner looping for a non-bound example
    - that maximizes |E_2 - E_1|
    - If the above doesn't make progress, it starts a sequential scan through the non-boundary examples. starting at a random position.

    - If the above fails too, it starts a sequential scan through all the examples, also starting at a random position.

  It can be seen that this is a form of Hierarchical heuristics being applied.

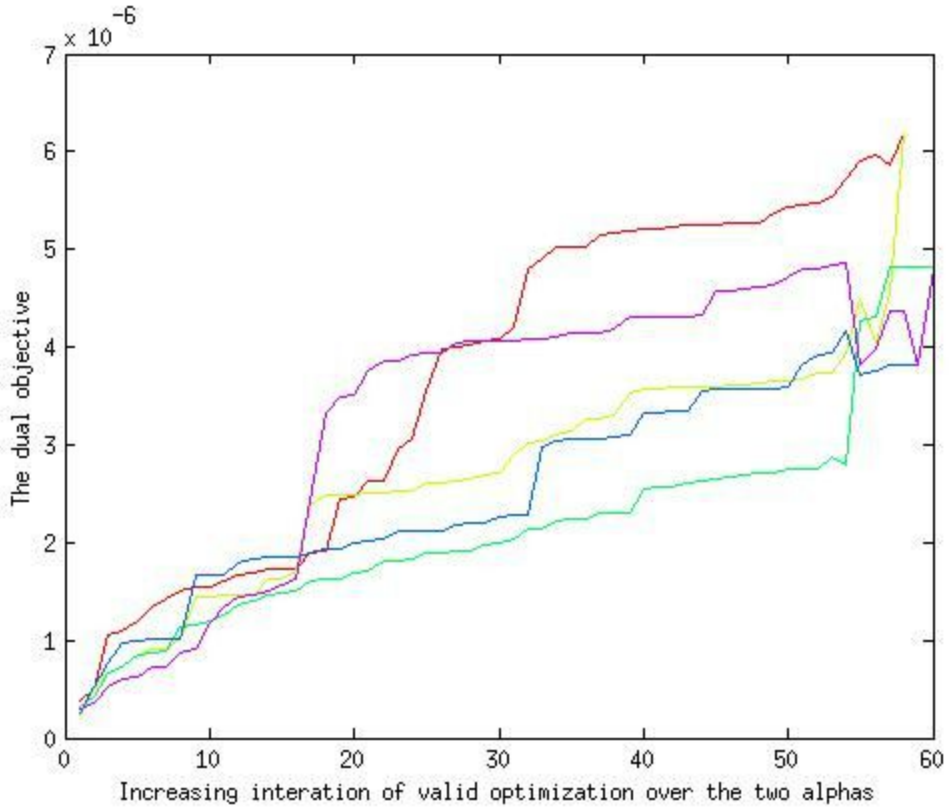
- SInce we need to find the Error E_2 and E_1 too often, we can speed up our system by caching the Errors E_i's for the non-bound examples.

Report the average runtime over 5 runs on the entire dataset along with the standard deviation

The mean of time taken over r runs (in secs):
571.1146 secs

The standard deviation for the above mean(in secs):
67.5599 secs

Final Plot:



Q3. Evaluate optimization performance of the pegasos algorithm. Run the code 5 times and report the average run-time and standard-deviation for each k.

SUMMARY:

Pegasos alternates between stochastic gradient descent steps and projection steps. This algorithm is specially good for large datasets since we need not to traverse the whole dataset here.

The task of learning a SVM is casted as a constrained quadratic programming problem. But it can also be thought as a constrained empirical loss minimization with a penalty term for the norm of the classifier that is being learned.

That is we want to find the minima of the following problem:

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{(\mathbf{x},y)\in S} \ell(\mathbf{w};(\mathbf{x},y))$$

where

$$\ell(\mathbf{w};(\mathbf{x},y)) = \max\{0, 1 - y\,\langle \mathbf{w},\mathbf{x}\rangle\}$$

That is the loss function is hinge loss.
The pegasos algorithm performs T iterations and also requires in additional parameter k, whose role is to select the size of mini-batch. i.e the it determines the number of examples from S the algorithm uses on each iteration for estimating the sub-gradient.

Pegasos exchanges the above objective function with an approximation:

$$f(\mathbf{w};A_t) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{k}\sum_{(\mathbf{x},y)\in A_t} \ell(\mathbf{w};(\mathbf{x},y))$$

Next it sets the learning rate

$$\eta_t = 1/(\lambda t)$$

and defines a new set A+ to be the set of examples for which w suffers a non-zero loss. We then perform the 2-step update which can be written in the form of :

$$\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t - \eta_t \nabla_t$$

where

$$\nabla_t = \lambda\,\mathbf{w}_t - \frac{1}{|A_t|}\sum_{(\mathbf{x},y)\in A_t^+} y\,\mathbf{x}$$

Then w(t+1) is obtained by scaling w(t+1/2) by

$$\min\left\{1, 1/(\sqrt{\lambda}\|\mathbf{w}_{t+\frac{1}{2}}\|)\right\}$$

The output of the Pegasos is given by the last vector w(t+1).

The mini-batch algorithm that i am following is as follows:

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: $\mathbf{w}_{T+1}$

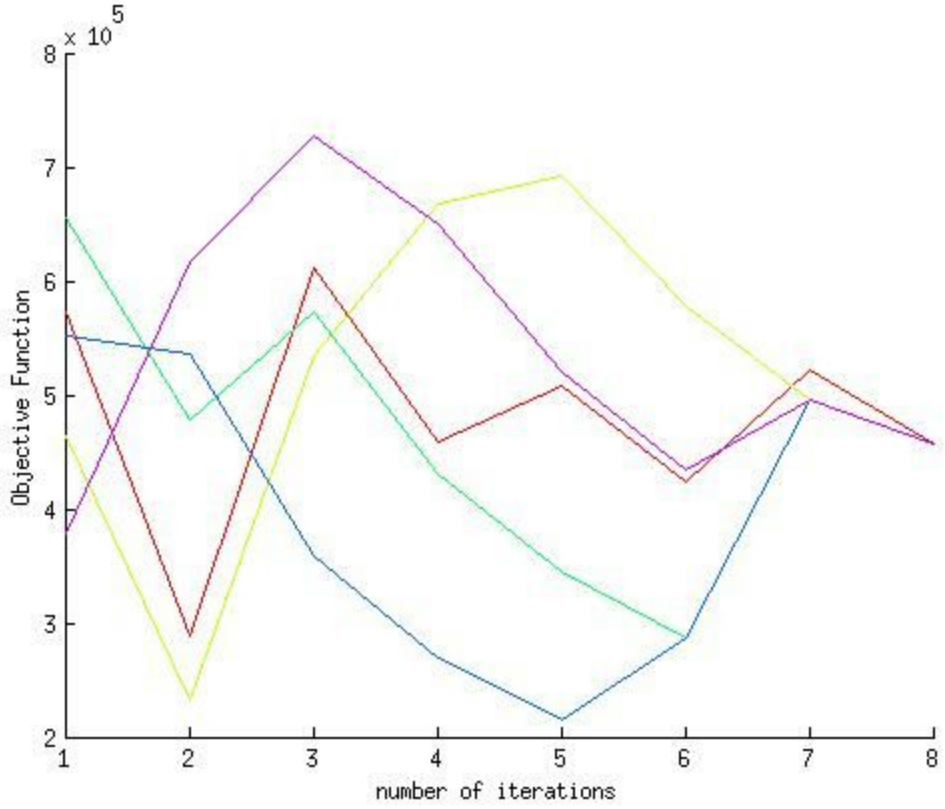
results:

for k=1, Numruns=5 (accuracy till 0.1)
Average runtime for 5 runs with minibatch size of 1 is: **0.007490** secs
Std. runtime for 5 runs with minibatch size of 1 is: **0.003174** secs

Plot: (When projection on the ball of radius is done.)

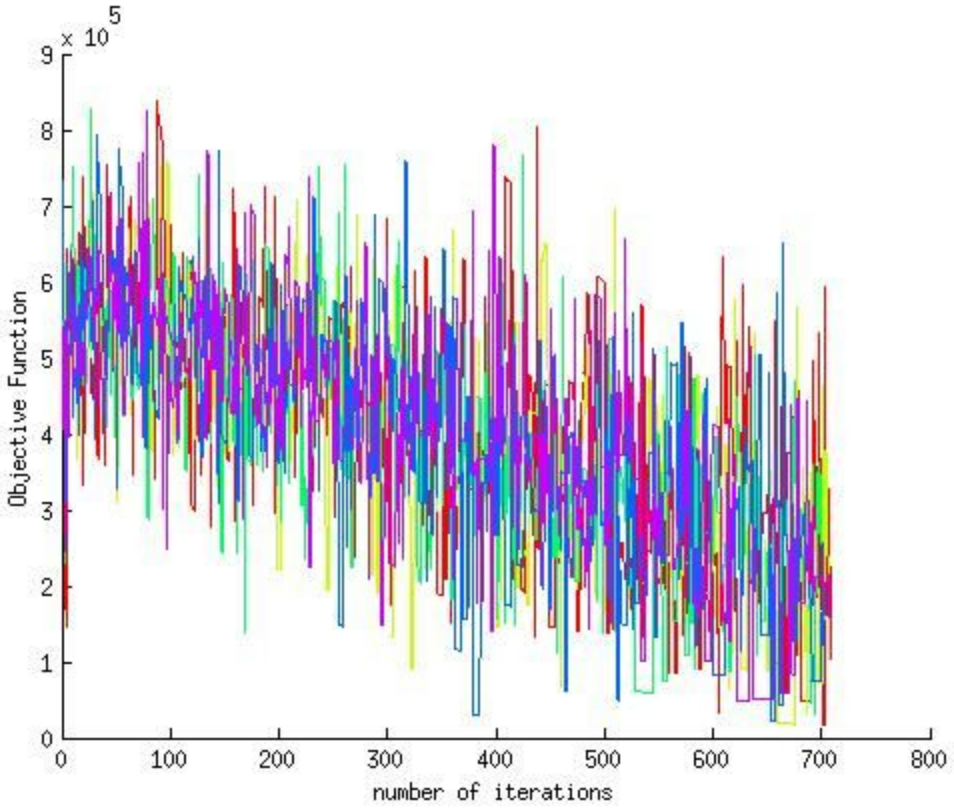
NOTE:

My code for the k=1 is different than that of others below. I have to do this since when k=1, I cannot take equal numbers of data-points from the 2 classes. So choosing the single data-point is completely random.



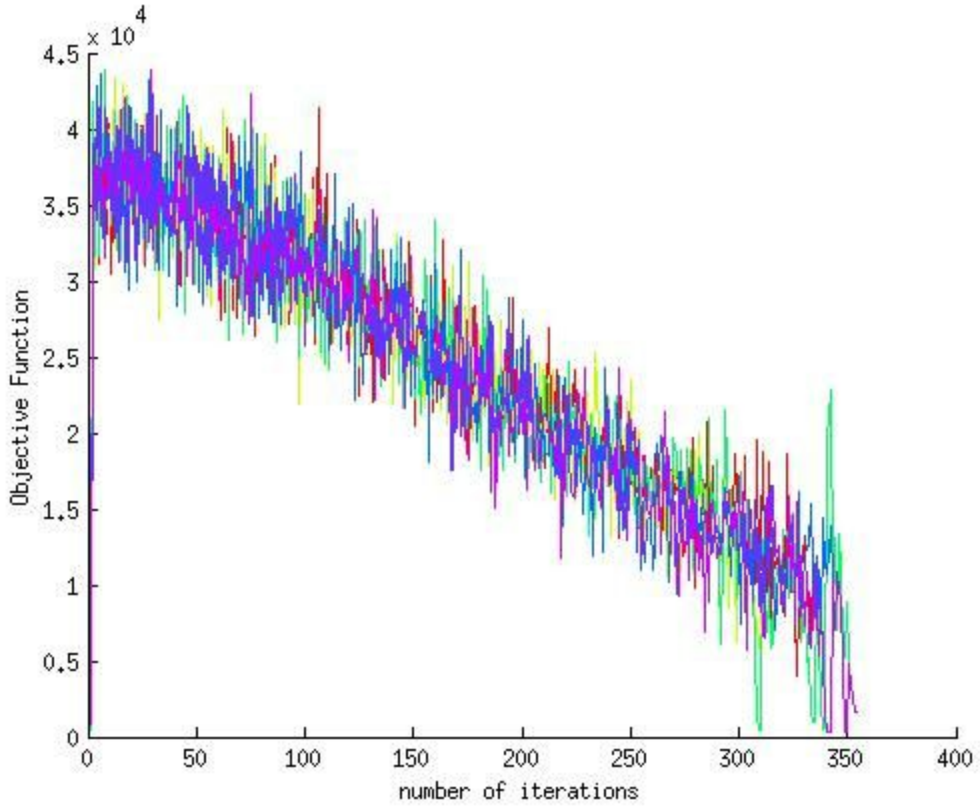
Doing this again for k=1, with accuracy = 0.001

Average runtime for 5 runs with minibatch size of 1 is: **0.090317** secs
Std. runtime for 5 runs with minibatch size of 1 is: **0.003795** secs



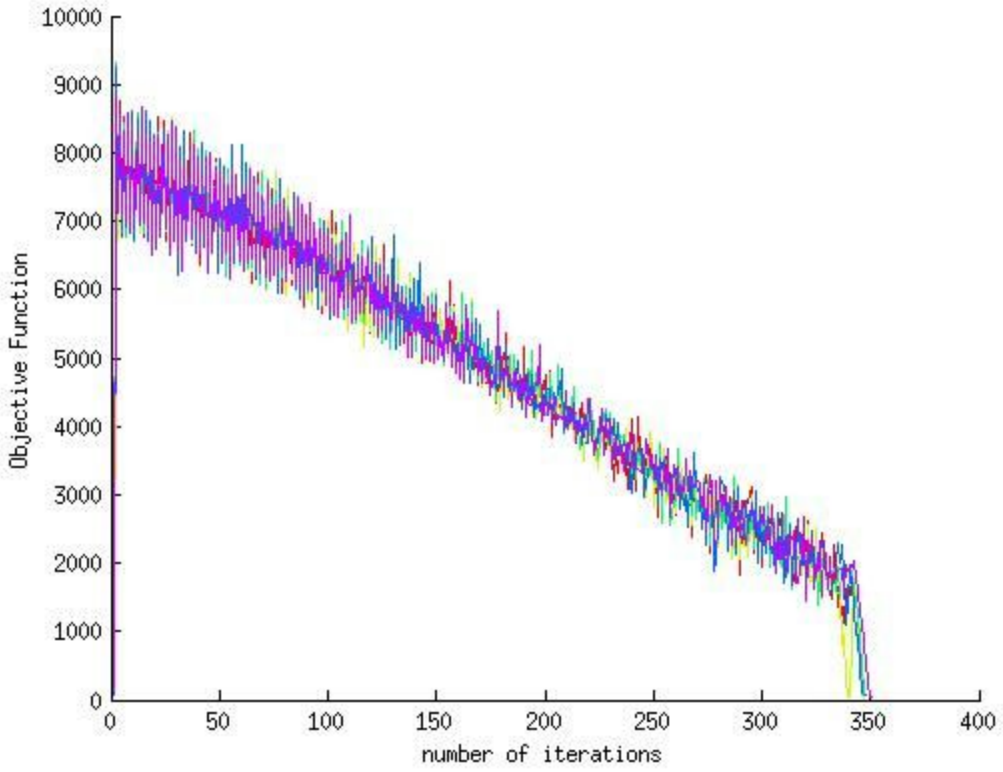
for k=20, Numruns=5(accuracy till 0.1)
Average runtime for 5 runs with minibatch size of 20 is: **0.540985** secs
Std. runtime for 5 runs with minibatch size of 20 is: **0.027864** secs

Plot(with projection on the ball of radius):



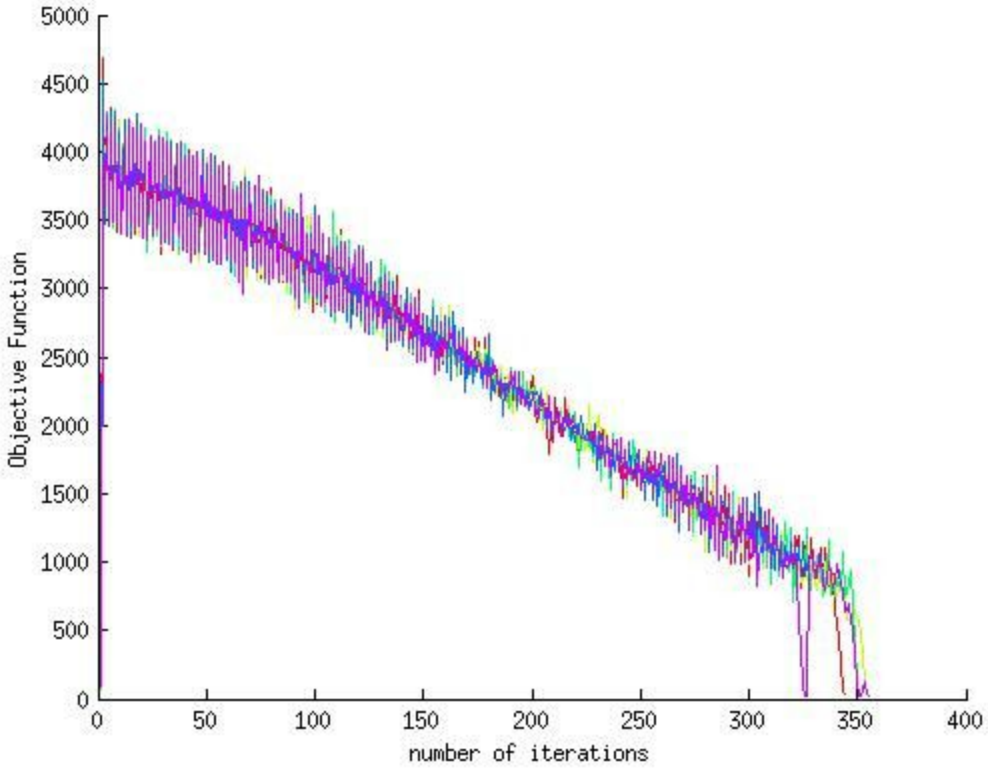
for k=100, Numruns=5(accuracy till 0.1)
Average runtime for 5 runs with minibatch size of 100 is: **0.779074** secs
Std. runtime for 5 runs with minibatch size of 100 is: **0.013555** secs

Plot(with projection on the ball of radius):



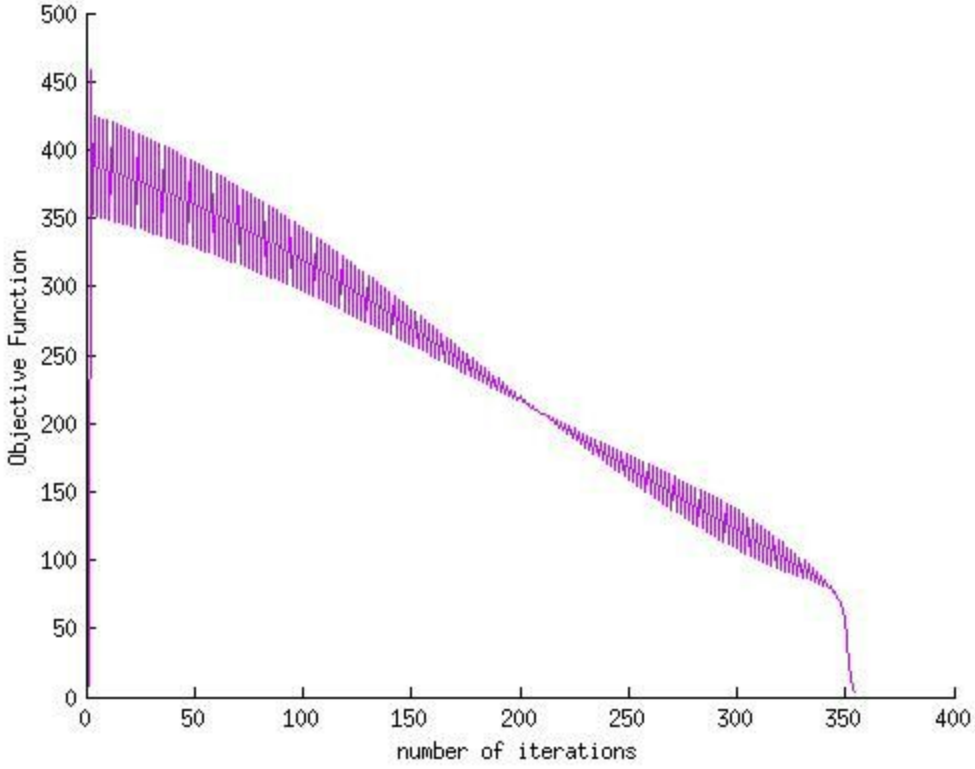
for k=200, Numruns=5(accuracy till 0.1)
Average runtime for 5 runs with minibatch size of 200 is: **1.136993** secs
Std. runtime for 5 runs with minibatch size of 200 is: **0.036521** secs

Plot(with projection on the ball of radius):



for k=2000, Numruns=5(accuracy till 0.1)
Average runtime for 5 runs with minibatch size of 2000 is: **5.173050** secs
Std. runtime for 5 runs with minibatch size of 2000 is: **0.368197** secs

Plot(with projection on the ball of radius):


500
450
400
350
300
250
200
150
100
50
0
Objective Function
0
50
100
150
200
250
300
350
400
number of iterations