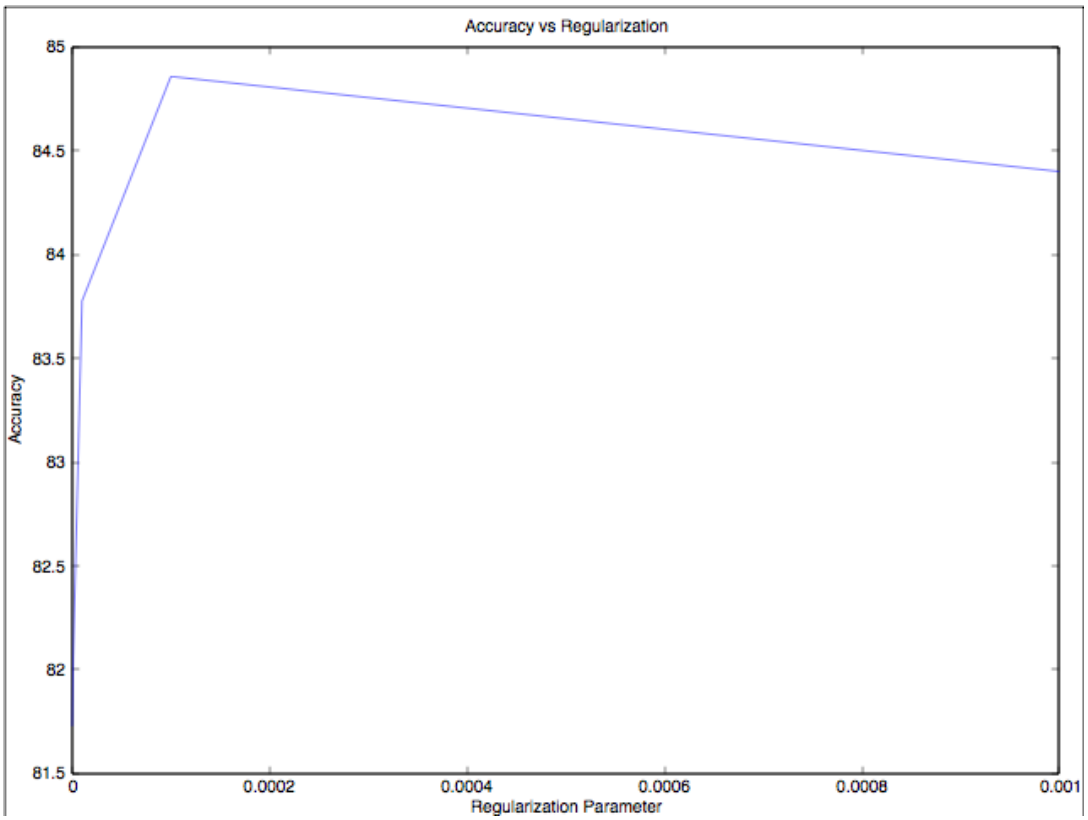
REPORT

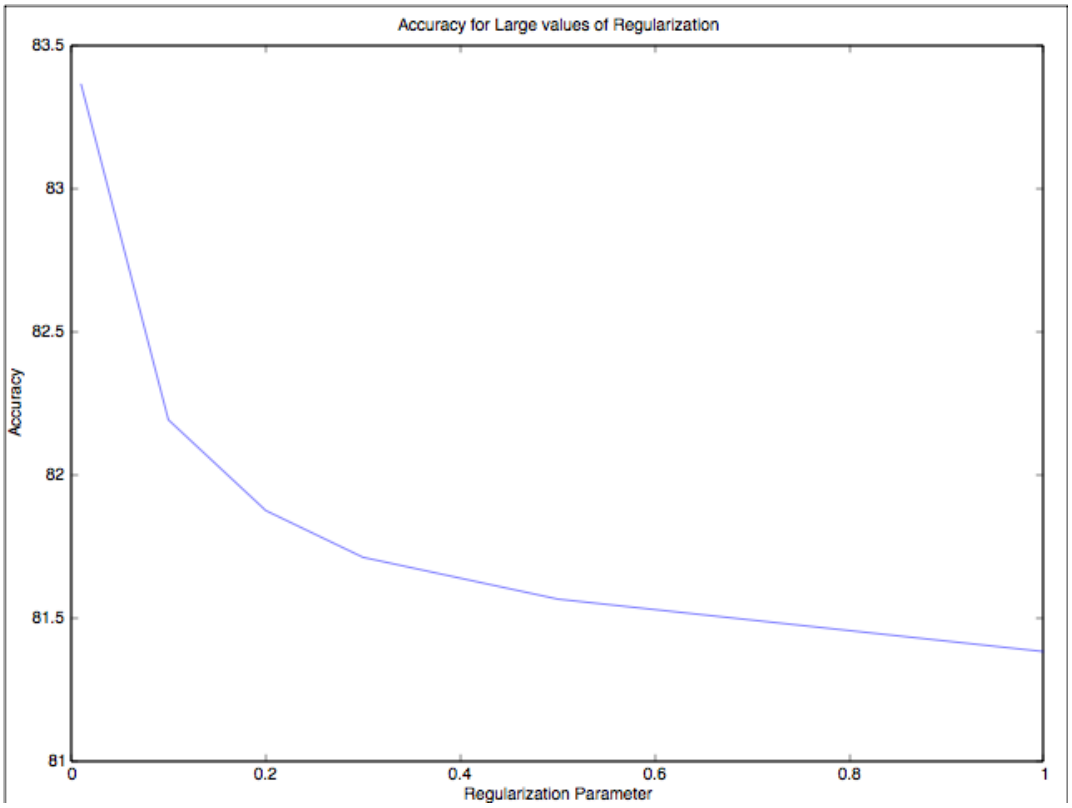
1) The value of the objective function at different epochs is shown below:

```
Value of Objective Function at Epoch 1: -4.633974371464905E7
Value of Objective Function at Epoch 2: -4648858.464687863
Value of Objective Function at Epoch 3: -229910.176737504
Value of Objective Function at Epoch 4: -99816.6579503876
Value of Objective Function at Epoch 5: -91618.15985194322
Value of Objective Function at Epoch 6: -91391.57164093175
Value of Objective Function at Epoch 7: -91385.77989984387
Value of Objective Function at Epoch 8: -91385.65537992776
Value of Objective Function at Epoch 9: -91385.65342605243
Value of Objective Function at Epoch 10: -91385.65340059137
Value of Objective Function at Epoch 11: -91385.65340028696
Value of Objective Function at Epoch 12: -91385.65340028565
Value of Objective Function at Epoch 13: -91385.6534002848
Value of Objective Function at Epoch 14: -91385.65340028609
Value of Objective Function at Epoch 15: -91385.65340028524
Value of Objective Function at Epoch 16: -91385.65340028482
Value of Objective Function at Epoch 17: -91385.65340028396
Value of Objective Function at Epoch 18: -91385.65340028258
Value of Objective Function at Epoch 19: -91385.65340028459
Value of Objective Function at Epoch 20: -91385.65340028456
```

As seen above, value of the objective function keeps on increasing at each epoch (becomes less negative). Furthermore, it starts converging to some maxima as the epoch increases.

2) After conducting the experiment by varying the regularization parameter, I observed that the accuracy on the test set was maximum for $\mu=10^{-4}$. The below 2 graphs show how the accuracy varies with different values of the regularization parameter.
In order to better illustrate the effect of changing the regularization parameter on the testing accuracy, I have divided the graphs into 2 parts – one for small values of regularization and the other for large values of regularization.
   - Effect of increasing regularization for small values of μ: On gradually increasing the regularization parameter from 0 to 10^{-4} we find that the accuracy actually increases from about 81% to about 85%. This helps in eliminating variance from the model.
   - After a certain point, as regularization increases the accuracy on the test set starts decreasing gradually. This is because as the regularization increases, the weight decay term starts dominating and the parameters get too small, which introduces bias into the system.

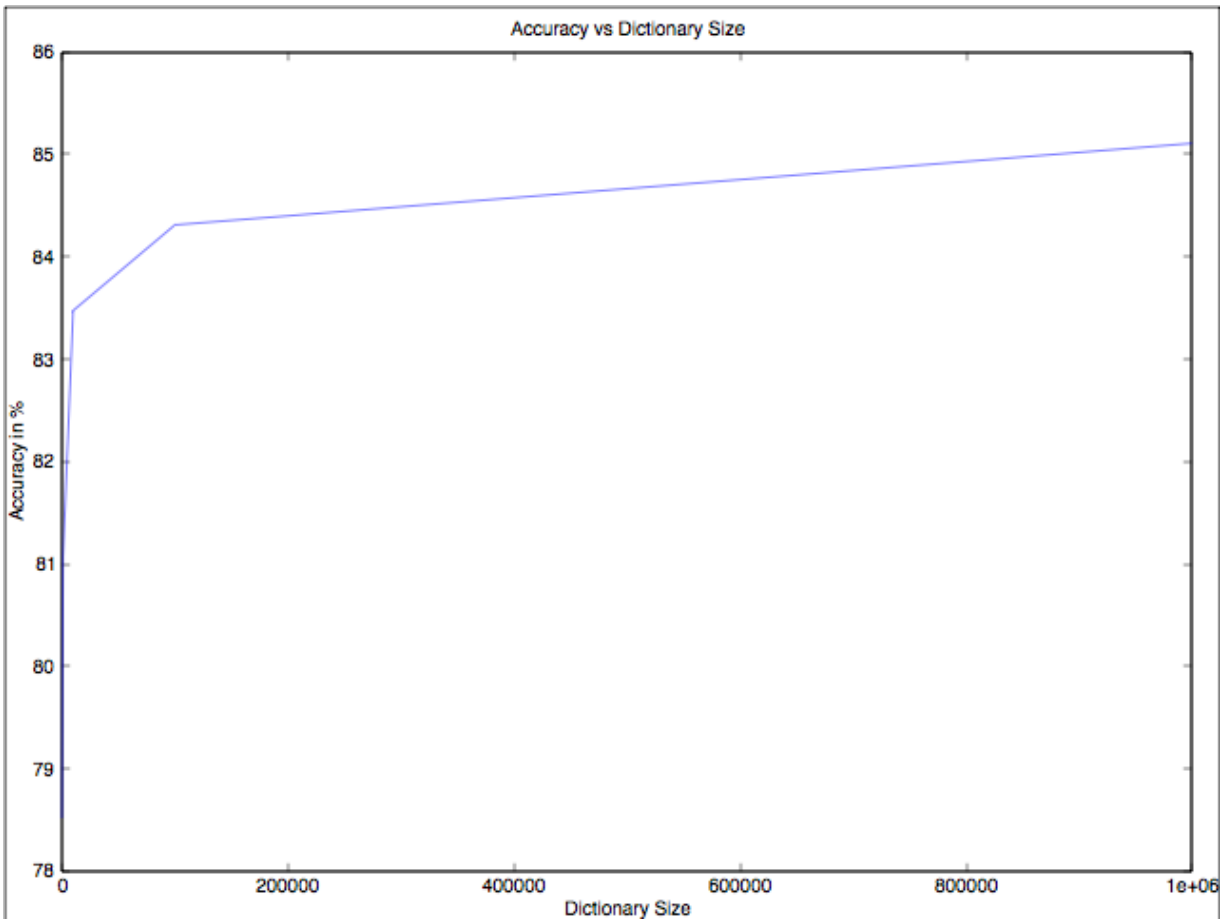

Accuracy vs Regularization
85
84.5
84
83.5
83
82.5
82
81.5
Accuracy
0
0.0002
0.0004
0.0006
0.0008
0.001
Regularization Parameter


Accuracy for Large values of Regularization
83.5
83
82.5
82
81.5
81
Accuracy
0
0.2
0.4
0.6
0.8
1
Regularization Parameter

3) Below is the result of the experiment conducted for different values of the dictionary size.

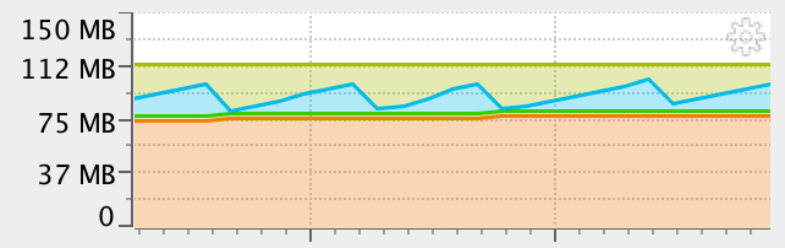
- Accuracy for dictionary size of 10: 78.5186%
- Accuracy for dictionary size of 100: 78.7607%
- Accuracy for dictionary size of 1000: 81.0773%
- Accuracy for dictionary size of 10000: 83.4743%
- Accuracy for dictionary size of 100000: 84.3097%
- Accuracy for dictionary size of 1000000: 85.1052%

Here accuracy is defined as: (True Positives + True Negatives) / (max number of labels)
Accuracy is high for a small dictionary size because the number of true negatives is very high even in the case of a small dictionary. As we increase the dictionary size, the number of true positives keep on increasing i.e. the classifier starts predicting accurate labels for the documents. However, beyond a certain point, the accuracy saturates and it no longer increases as seen from the below graph. This also shows that the distortion of the data due to the hash trick does not adversely affect the performance of the classifier.

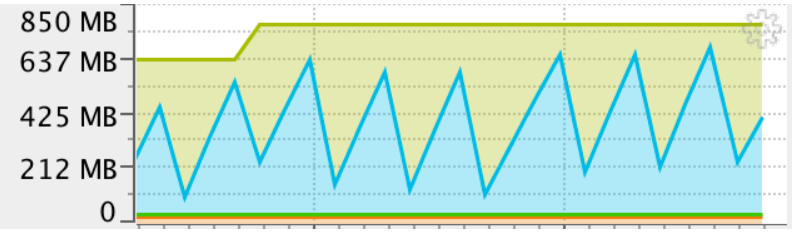


4) After experimenting with Stochastic and Batch gradient descent, we can infer the following:
   - The memory consumption of Batch Gradient Descent is much larger than stochastic gradient. For e.g. On the small dataset, I found that the peak memory consumption of Batch gradient descent is about 650 MB. While for stochastic gradient descent it is close to 100MB (Snapshot below). This is mainly because in Batch gradient descent we update the weight parameters after iterating through the entire dataset. Hence, we need extra memory to keep track of the average gradient value for the entire dataset. In contrast, in stochastic gradient descent we update the weight parameters after observing each data sample. Hence, no extra memory is required to keep track of the gradient after updating each data sample.
     Note: In my implementation of Batch Gradient Descent I streamed through the input data, similar to stochastic gradient descent. Hence, I did not have to load the entire input data in memory. Else, this could be an additional memory overhead for batch gradient descent.

Stochastic Gradient Descent



Batch Gradient Descent



   - Stochastic Gradient descent normally converges faster than batch gradient descent. However, it depends on the '**ε**' we choose. For e.g. if we choose '**ε**' as **0.01**, stochastic gradient descent converges much faster. However, this does not mean that stochastic gradient descent has converged to a maxima. Hence, in order to increase the accuracy of Stochastic gradient descent we need to choose '**ε**' a much smaller value like 0.0001. On using this value of '**ε**' **,** I observed that batch gradient descent took 8 iterations through the dataset to converge and stochastic gradient descent took about 28000 input samples to converge(the input data size is 11200, so approximately it took 2.5 passes through the dataset to converge). Moreover, the convergence point of stochastic gradient descent was very close to the convergence point of batch gradient descent. Hence, by choosing a good value for '**ε**' we can get stochastic gradient descent to perform similar to batch gradient descent.
     Note: In order to calculate the objective function, one needs to calculate the sum of the squared weights (the regularization term) for each iteration. This is a costly computation

as we have to sum the values of all the weights. In batch gradient descent this quantity is calculated only 8 times (once for each iteration through the dataset). However, for stochastic gradient descent this quantity is calculated about 28000 times (once for each data sample). Hence, if we measure the time taken to converge, we find out that batch gradient descent is actually running faster. For e.g. it took 48s for batch gradient descent to converge. But it took 6min for stochastic gradient descent to converge. However, if we choose to ignore the computation of the objective value at each iteration, we find that stochastic gradient descent is much faster than batch gradient descent.

5) No, we are not guaranteed to get the same W and H, every time we run DSGD with a new random initialization. The loss function for the matrix factorization is a non – convex function. Hence, there a number of local minima. Hence, if one uses different a random initialization, it may converge to a different local minimum.

6) The decompositions represented by the options **a**, **c** and **d** are valid decompositions. A decomposition is valid if and only if no two blocks share any row or column. This is a required condition for DSGD to work efficiently. In option b, few of the blocks have some or the other row/column in common. Hence, it's not a valid decomposition.

7) The option **d** represents a valid decomposition. However, it is a suboptimal decomposition as not all rows/columns are distributed in one pass.

8)

- **Did you receive any help whatsoever from anyone in solving this assignment?**
  No

- **Did you give any help whatsoever to anyone in solving this assignment?**
  No