# ML Assignment 4

*Author : Aman Bhatia*
*Dated : 04 May 2016*

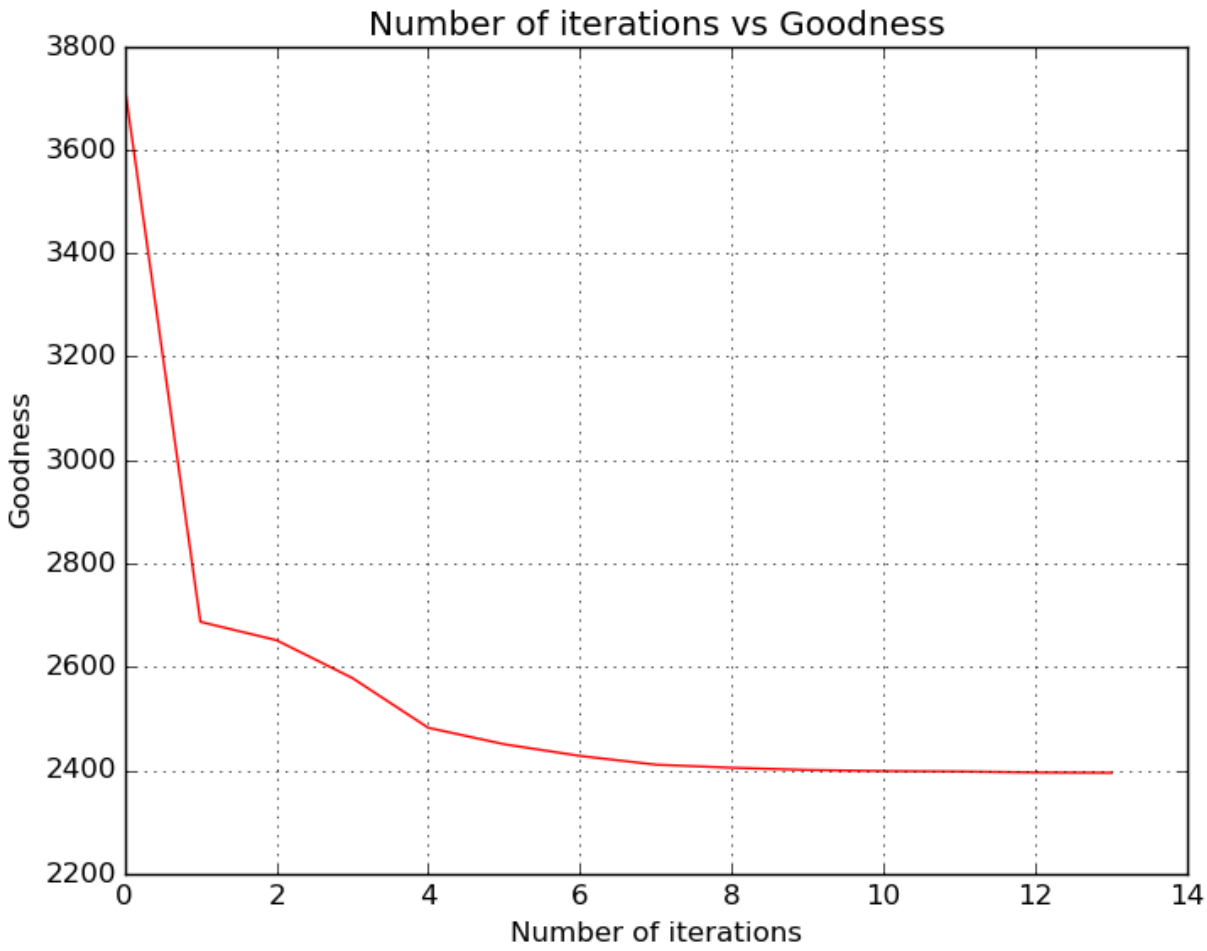
## K-Means for Digit Recognition

**(a)** Usage Command :

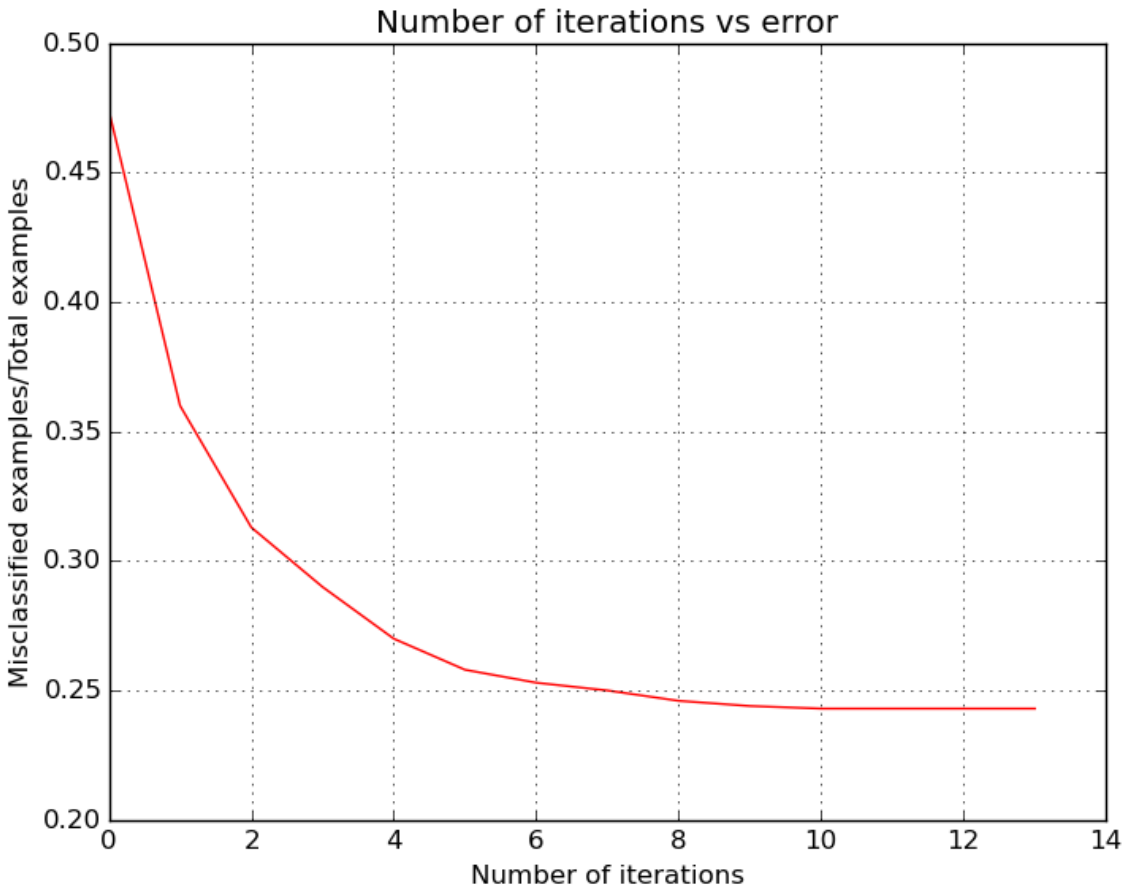
python3 visulaize.py <example_index>

**(b)** Implemented K-Means Clustering Algorithm for this part. Maximum number of iterations is set to 30. However, program generally converges in 12-18 iterations.

**(c)** Plotted graph for quantity S, i.e. goodness vs number of iteration. The quantity decreases with the number of iterations. Here is the graph,



**(d)** Plotted graph for quantity Misclassified examples/Total examples vs number of iteration. The quantity decreases with the number of iterations. Here is the graph,




# Expectation Maximization

**(a)** Following table values are obtained,

| Table H |
| --- |
| [[ 0.804 0.196]] |

| Table B |
| --- |
| [[ 0.95024876 0.04975124]<br>[ 0.58367347 0.41632653]] |

Table L
[[ 0.99577114 0.00422886]
[ 0.70918367 0.29081633]]

Table F
[[ 0.95128327 0.04871673]
[ 0.48369565 0.51630435]
[ 0.92346939 0.07653061]
[ 0.30084746 0.69915254]]

Table X
[[ 0.97786292 0.02213708]
[ 0.39238411 0.60761589]]

*Log likelihood* = *-2515.2735650906666*

**(b)** Implemented EM algorithm in this part.

In the E step, we insert all the possible values of the missing data and assign the probability of that entry calculated on the basis of already known parameters.

In the M step, we update our parameters with the new probabilities of data.

*Convergence Criteria* : We threashold the sumation of absolute difference of parameters between two succesive iterations.

Following are the results for one missing value data,

## ***Initial tables***

Table H
[[ 0.80182002 0.19817998]]

Table B
[[ 0.95347368 0.04652632]
[ 0.59017782 0.40982218]]

Table L
[[ 0.99518828 0.00481172]
[ 0.70951157 0.29048843]]

Table F
[[ 0.95210617 0.04789383]
[ 0.49295775 0.50704225]
[ 0.9093199 0.0906801 ]
[ 0.27173913 0.72826087]]

Table X
[[ 0.97937963 0.02062037]
[ 0.39488636 0.60511364]]

## ***Final tables***

Table H
[[ 0.80268587 0.19731413]]

Table B
[[ 0.95287761 0.04712239]
[ 0.58306885 0.41693115]]

Table L
[[ 0.99574746 0.00425254]
[ 0.71221438 0.28778562]]

Table F
[[ 0.95161061 0.04838939]
[ 0.4989571 0.5010429 ]
[ 0.91913606 0.08086394]
[ 0.27081075 0.72918925]]

Table X
[[ 0.9793392 0.0206608 ]
[ 0.38276002 0.61723998]]

*New Log likelihood = -2514.5810197808864*

**(c)** Following are the results for one or two missing value data,

## ***Initial tables***

Table H
[[ 0.80834289 0.19165711]]

Table B
[[ 0.94551458 0.05448542]
[ 0.58027523 0.41972477]]

Table L
[[ 0.99498747 0.00501253]
[ 0.70344828 0.29655172]]

Table F
[[ 0.94462849 0.05537151]
[ 0.38947368 0.61052632]

[ 0.91129032  0.08870968]
[ 0.29230769  0.70769231]]

Table  X
[[ 0.97579679  0.02420321]
 [ 0.38909091  0.61090909]]

## ***Final tables***

Table  H
[[ 0.80794577  0.19205423]]

Table  B
[[ 0.94549276  0.05450724]
 [ 0.58363192  0.41636808]]

Table  L
[[ 0.99497023  0.00502977]
 [ 0.70420185  0.29579815]]

Table  F
[[ 0.9534012   0.0465988 ]
 [ 0.44446995  0.55553005]
 [ 0.90581499  0.09418501]
 [ 0.32258312  0.67741688]]

Table  X
[[ 0.9773303  0.0226697]
 [ 0.3930454  0.6069546]]

*New Log likelihood =  -2516.0490816957254*

There is not much difference between log likelihoods of train-m1 and train-m2 data as compared to train data. We also expected the same because we are using probabilities of various possibilities of missing data. So, if there is pattern in the data, then obviously the actual entry among all the possible missing entries will get the highest probabilty.

# Kaggle Question

## *Support Vector Machines*

Linear Kernel - default parameter settings

Accuracy on Validatation Data 1 : 0.584366666667
Accuracy on Validatation Data 2 : 0.5741
Accuracy on Validatation Data 3 : 0.5334

I tried various parameters such as C=0.5,1.5,2.0, tol=1e-7,dual=False. Very less change in accuracy is observed. Reason should be that data in not linearly separable.

SVM gaussian kernel - default parameter settings

Accuracy on Validatation Data 1 : 0.845933333333
Accuracy on Validatation Data 2 : 0.7896
Accuracy on Validatation Data 3 : 0.735266666667

- I experimented with various parameters of guassian kernel such as,
  - C in range 0.5 to 10.0
  - gamma in 0.1, 0.01, 0.02, 0.03, 0.001

SVM gaussian kernel - tuned parameters

- Getting maximum accuracy at C=1.0, gamma=0.02.

Accuracy on Validatation Data 1 : 0.873633333333
Accuracy on Validatation Data 2 : 0.7928
Accuracy on Validatation Data 3 : 0.724766666667

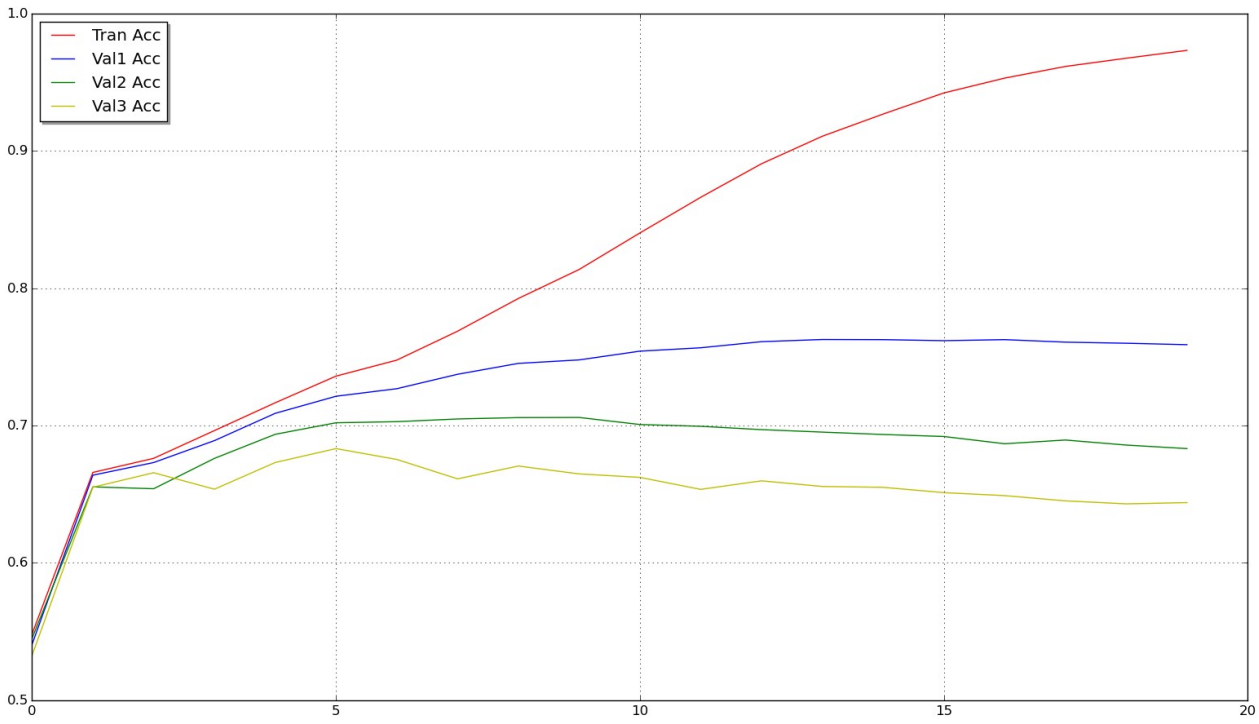
# ***Decision Trees***

- I plotted graph of various accuracies vs max_depth of the tree. Looks like model starts to overfit after depth 9. So, I fixed max_depth to be 9.

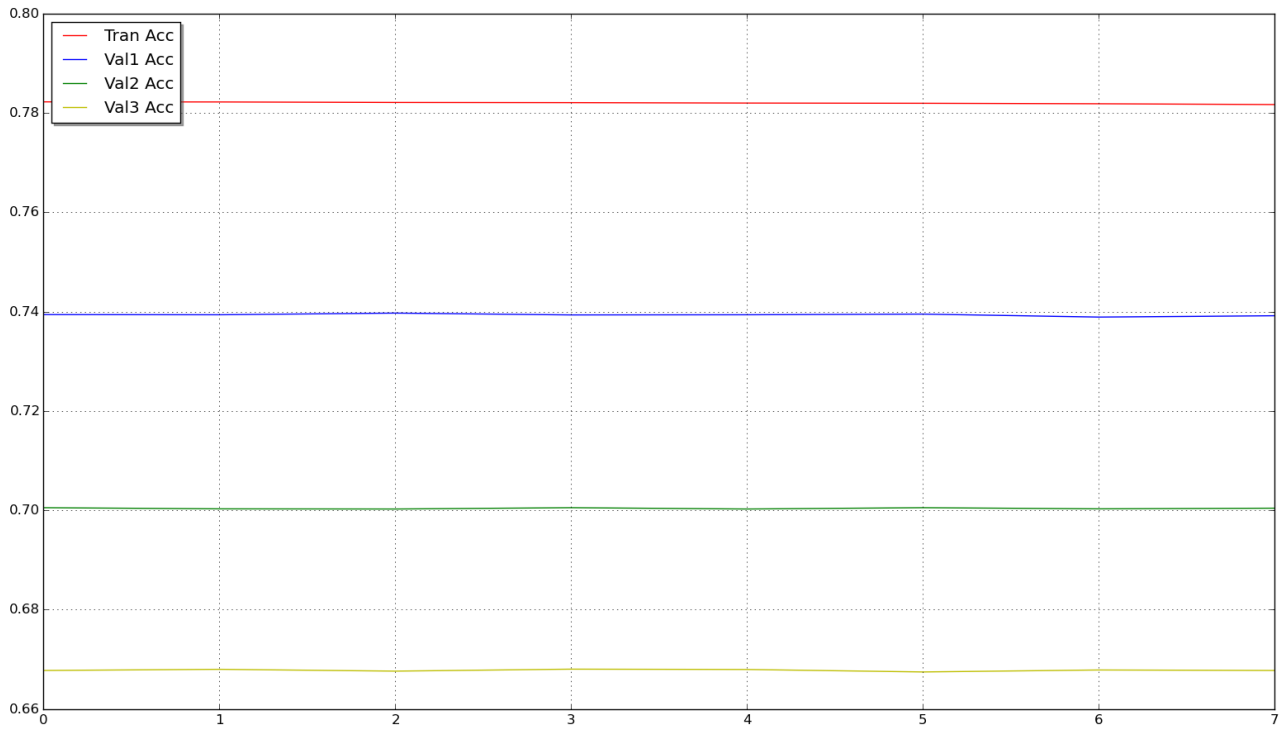
- Also, I tried parameters such as min_samples_split over range (2,10), splitter="best","random" etc. Nothing seems to improve accuracies much.

Following graphs are obtained,

(1) Accuracies vs max-depth



(2) Accuracies vs min_samples_split




Final Accuracies obtained are as follows,

Accuracy on Validatation Data 1 : 0.739266666667
Accuracy on Validatation Data 2 : 0.700233333333
Accuracy on Validatation Data 3 : 0.6682

## ***Random Forest***

Following accuracies were obtained,

Accuracy on Train Data : 0.99438
Accuracy on Validatation Data 1 : 0.8048
Accuracy on Validatation Data 2 : 0.706033333333
Accuracy on Validatation Data 3 : 0.651266666667

## ***Naive Bayes***

I did not experiment much in this part since looking at the initial accuracies, I understood that nothing much is going to happen. Following accuracies were obtained,

Accuracy on Train Data : 0.55518
Accuracy on Validatation Data 1 : 0.5503
Accuracy on Validatation Data 2 : 0.554133333333
Accuracy on Validatation Data 3 : 0.539466666667

## ***Neural Network***

Tried various of the above submissions. Finally, I used a Neural Network. My best entry on the kaggle leader board correspond to a Neural Network with following parameters. For further exploration, have a look at the description of individual entries on kaggle.

- Used 2 hidden layers with 200 units each.
- Also, dropout layer after each layer with a droput of 0.2.
- Non-linearity is leaky_rectify.
- Finally, softmax is used for output layer.
- Learning Rate of 0.01.

*Library Used* : Lasagne (http://lasagne.readthedocs.io)

*Note* : Most of the code for this part is taken from the tutorial section of neural network library lasagne for python.

*Link* : http://lasagne.readthedocs.io/en/latest/user/tutorial.html

***-------- END OF DOCUMENT---------***