CSE291_HW2

Name: Rajat Sharma
ID: A53205019
Name: Rishab Gulati
ID: A53219165

May 15, 2017

Contents

Random Forests

Our implementation of RandomForest can be found in file named MyRandomForest.py. We use sklearn's DecisionTreeClassifier to create and fit individual trees inside the forest. Also we train the RF with the maximum number of trees first and then sample from it to simulate RFs with less trees while generating the training, test or OOB error. Also the number of randomly choosen dimensions at each split (m) is kept as sqrt(n_features) and the bootstrap set size is always the same as the original input sample size but the samples are drawn with replacement.

1.1 Covertype Dataset

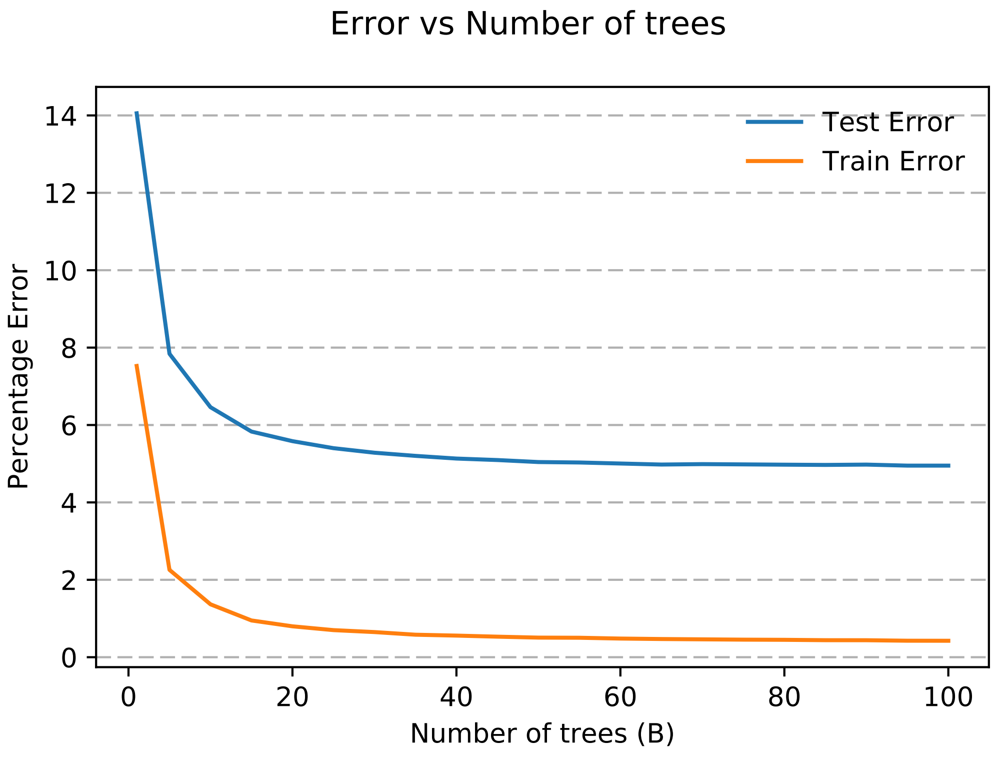
First we import the covertype dataset and split it into training set (75%) and test set (25%)

Then we use sklearn's VarianceThreshold selector to weed out dimensions having same value in more than 90% of the samples. This reduces the dimension of the input rows from 54 to 13.

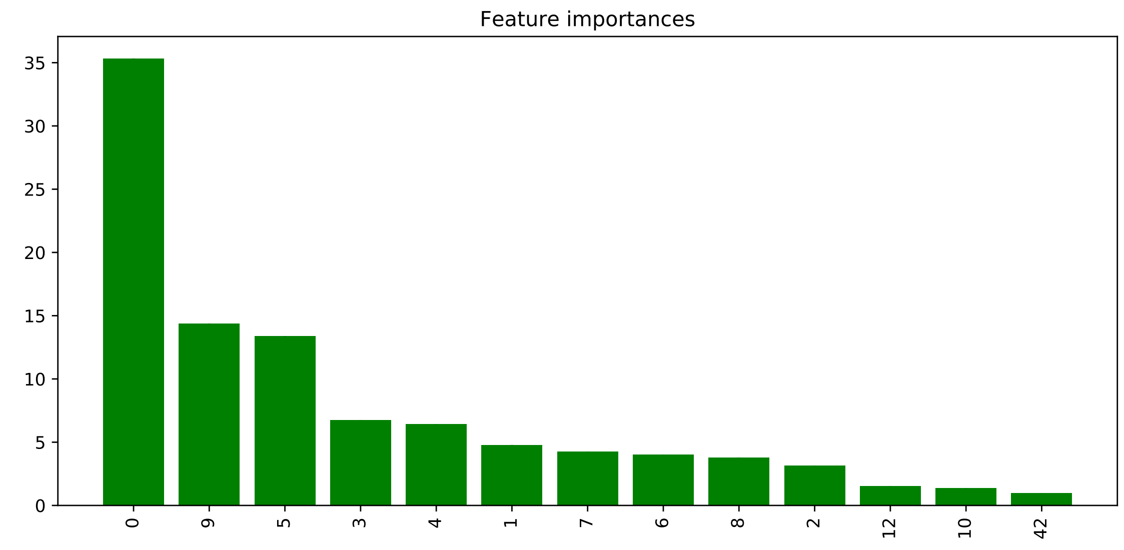
1.1.1 RF in which each of the decision trees is grown to the full extent

First we train a RF in which each of the decision trees is grown to the full extent with 100 trees (maximum) and then we subsample from the trained forest to plot test and training error with respect to the number of trees (B).

```
Out[15]: <__main__.MyRandomForest instance at 0x1107280e0>
```



From the above plot it can be seen that the test error reaches a plateau after 40 trees. Hence 40 seems to be a reasonable number of estimators. Then we find and visualize a sorting of the most important features (among the reduced dimension of 13) in the trained forest using the Gini importance values for individual trees.

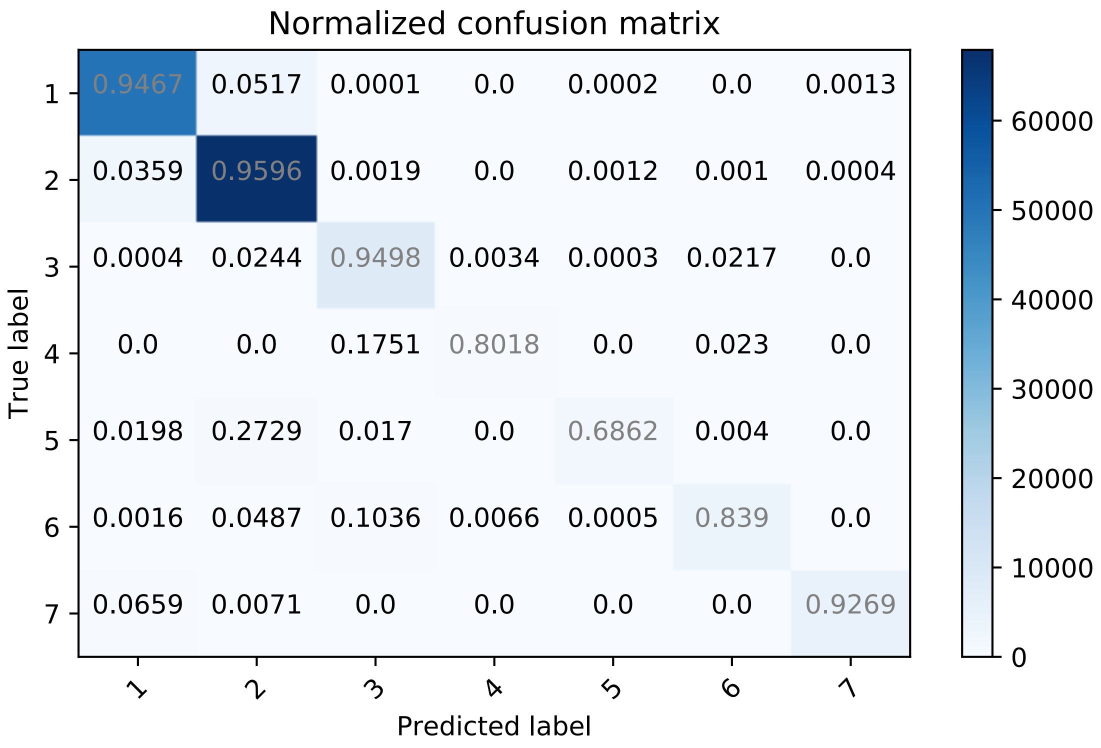


```
Feature ranking:
1. feature 0 - Elevation (35.295640)
2. feature 9 - Horizontal_Distance_To_Fire_Points (14.367086)
3. feature 5 - Horizontal_Distance_To_Roadways (13.366360)
4. feature 3 - Horizontal_Distance_To_Hydrology (6.742069)
5. feature 4 - Vertical_Distance_To_Hydrology (6.407617)
6. feature 1 - Aspect (4.779741)
7. feature 7 - Hillshade_Noon (4.232207)
8. feature 6 - Hillshade_9am (4.019547)
9. feature 8 - Hillshade_3pm (3.793784)
10. feature 2 - Slope (3.141459)
11. feature 12 - Wilderness_Area_3 (1.534726)
12. feature 10 - Wilderness_Area_1 (1.347199)
13. feature 42 - Soil_Type_29 (0.972563)
```

Then we show the final confusion matrix with 20 estimators.

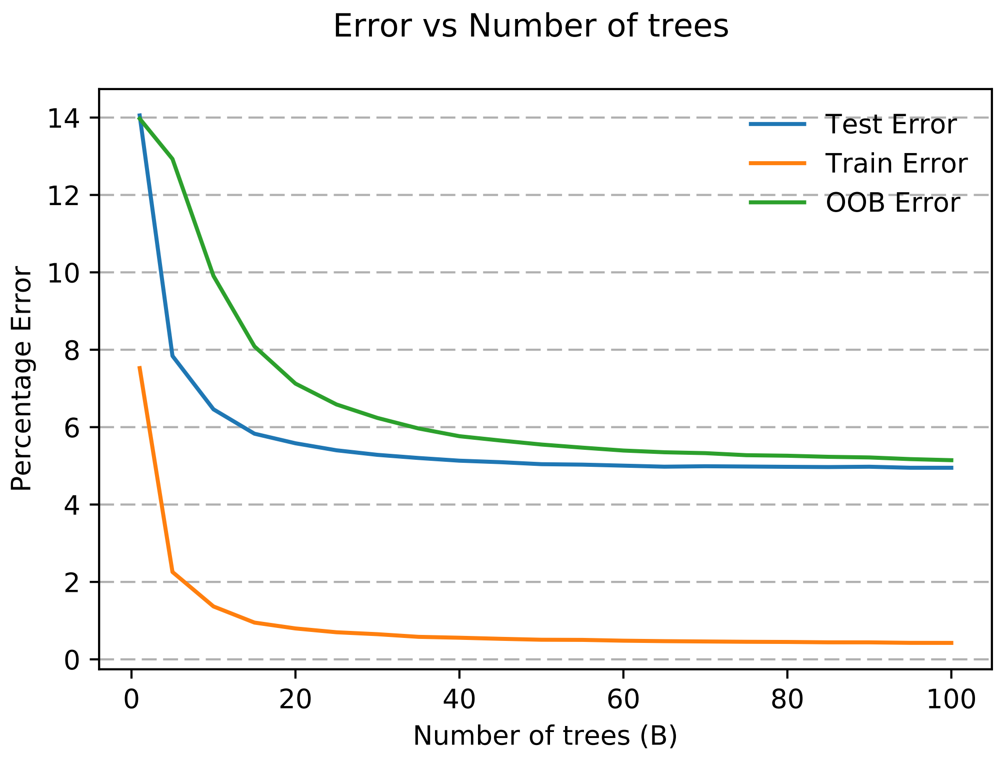
```
Confusion matrix:
[[50183  2741     3     0     9     2    68]
 [ 2542 67916   132     0    88    71    27]
 [    4   217  8462    30     3   193     0]
 [    0     0   114   522     0    15     0]
 [   49   675    42     0  1697    10     0]
 [    7   213   453    29     2  3669     0]
 [  334    36     0     0     0     0  4695]]
Normalized confusion matrix
[[  9.4674e-01   5.1711e-02   5.6597e-05   0.0000e+00   1.6979e-04
    3.7732e-05   1.2829e-03]
 [  3.5916e-02   9.5959e-01   1.8650e-03   0.0000e+00   1.2434e-03
    1.0032e-03   3.8149e-04]
 [  4.4898e-04   2.4357e-02   9.4983e-01   3.3674e-03   3.3674e-04
    2.1663e-02   0.0000e+00]
 [  0.0000e+00   0.0000e+00   1.7512e-01   8.0184e-01   0.0000e+00
```

```
   2.3041e-02   0.0000e+00]
[  1.9814e-02   2.7295e-01   1.6983e-02   0.0000e+00   6.8621e-01
   4.0437e-03   0.0000e+00]
[  1.6007e-03   4.8708e-02   1.0359e-01   6.6316e-03   4.5735e-04
   8.3901e-01   0.0000e+00]
[  6.5943e-02   7.1076e-03   0.0000e+00   0.0000e+00   0.0000e+00
   0.0000e+00   9.2695e-01]]
```



```
('Percentage accuracy:', 94.417327008736478)
```

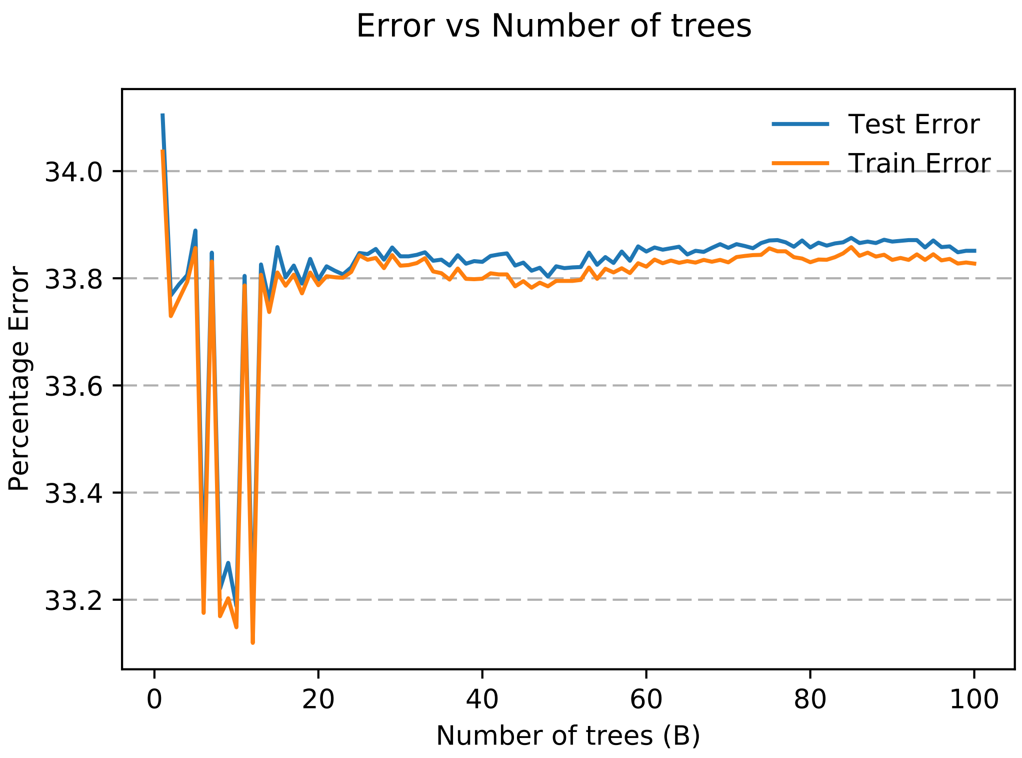
Now we plot the out-of-bag (OOB) error alongside the train and test error with the number of estimators used which is always more than the test error.



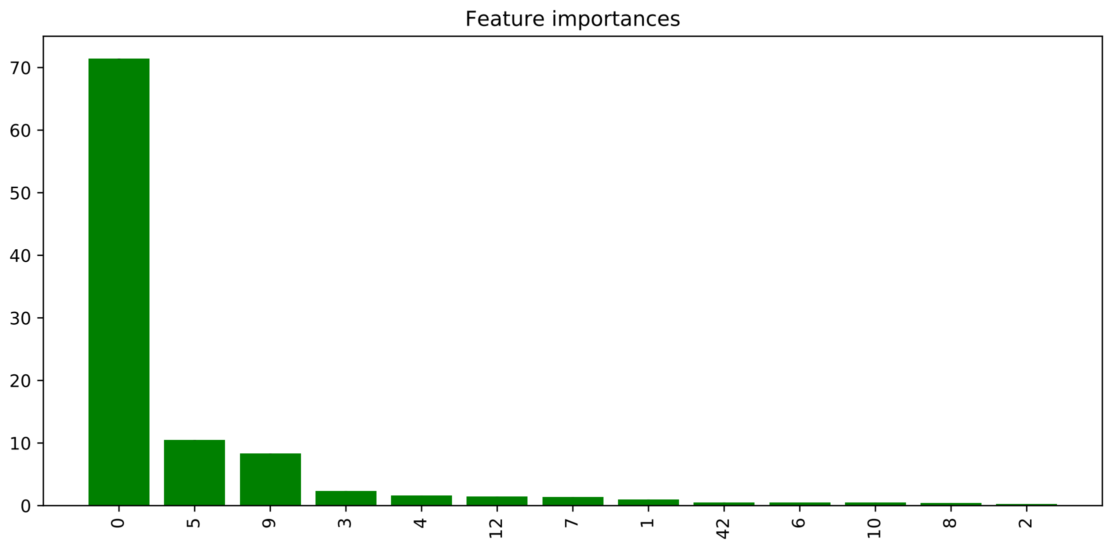
1.1.2 RF in which each of the decision trees have maximum depth of 5

Now we train a RF in which each of the decision trees have maximum depth of 5 with 100 trees (maximum) and then we subsample from the trained forest to plot test and training error with respect to the number of trees (B).

```
Out[38]: <__main__.MyRandomForest instance at 0x212596a28>
```



We observe that the test error doesn't vary much (in magnitude). From the above plot it can be seen that the test error reaches a plateau after 20 trees. Hence 20 seems to be a reasonable number of estimators. Then we find and visualize a sorting of the most important features (among the reduced dimension of 13) in the trained forest using the Gini importance values for individual trees.

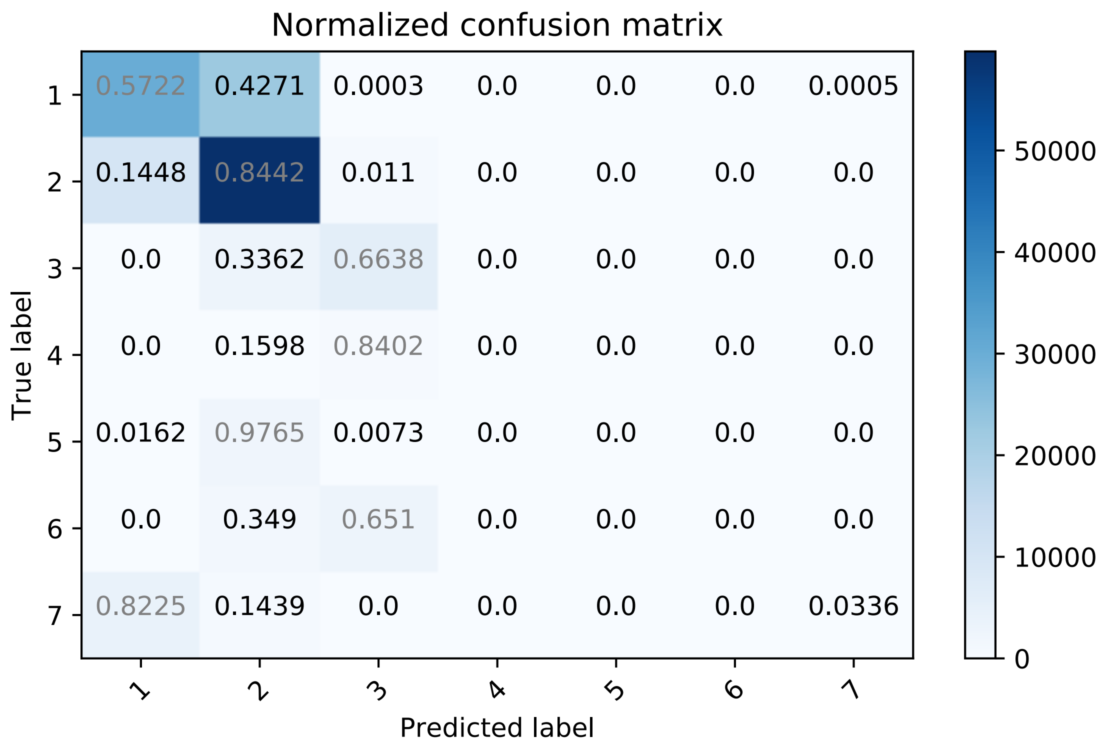


```
Feature ranking:
1. feature 0 - Elevation (71.400641)
```

```
2. feature 5 - Horizontal_Distance_To_Roadways (10.484413)
3. feature 9 - Horizontal_Distance_To_Fire_Points (8.324306)
4. feature 3 - Horizontal_Distance_To_Hydrology (2.343969)
5. feature 4 - Vertical_Distance_To_Hydrology (1.609262)
6. feature 12 - Wilderness_Area_3 (1.413106)
7. feature 7 - Hillshade_Noon (1.362599)
8. feature 1 - Aspect (0.986511)
9. feature 42 - Soil_Type_29 (0.502500)
10. feature 6 - Hillshade_9am (0.461032)
11. feature 10 - Wilderness_Area_1 (0.456350)
12. feature 8 - Hillshade_3pm (0.428542)
13. feature 2 - Slope (0.226770)
```

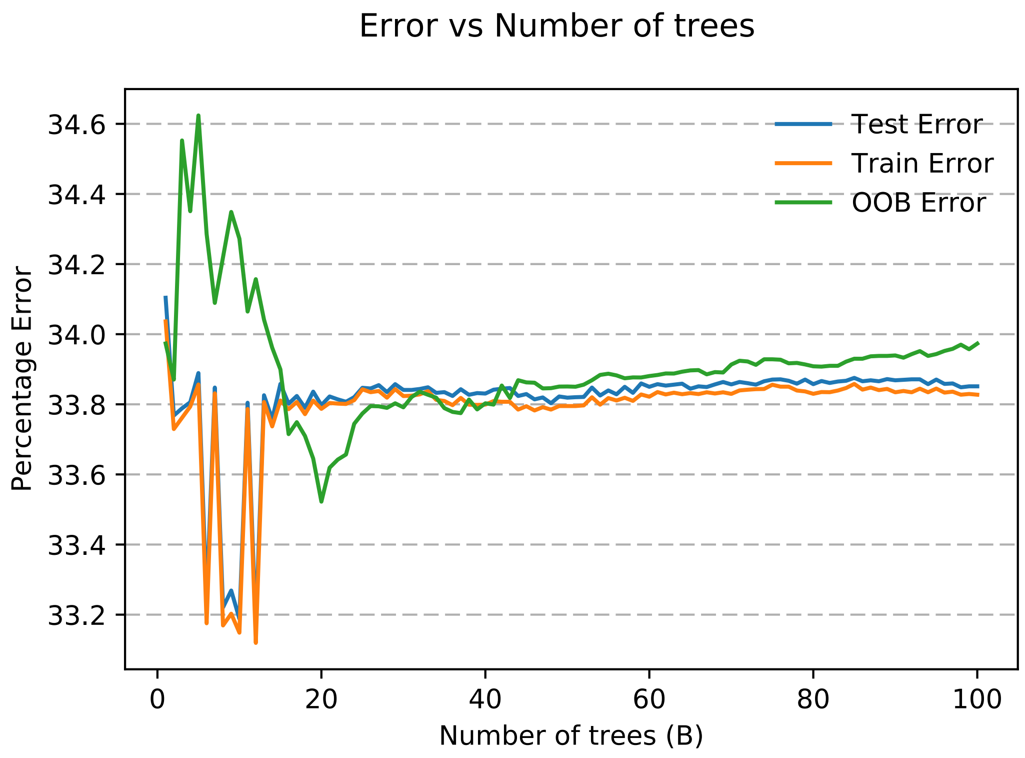
Then we show the final confusion matrix with 18 estimators.

```
Confusion matrix:
[[30329 22638    14     0     0     0    25]
 [10246 59748   782     0     0     0     0]
 [    0  2995  5914     0     0     0     0]
 [    0   104   547     0     0     0     0]
 [   40  2415    18     0     0     0     0]
 [    0  1526  2847     0     0     0     0]
 [ 4166   729     0     0     0     0   170]]
Normalized confusion matrix
[[  5.7218e-01   4.2708e-01   2.6412e-04   0.0000e+00   0.0000e+00
    0.0000e+00   4.7164e-04]
 [  1.4477e-01   8.4418e-01   1.1049e-02   0.0000e+00   0.0000e+00
    0.0000e+00   0.0000e+00]
 [  0.0000e+00   3.3618e-01   6.6382e-01   0.0000e+00   0.0000e+00
    0.0000e+00   0.0000e+00]
 [  0.0000e+00   1.5975e-01   8.4025e-01   0.0000e+00   0.0000e+00
    0.0000e+00   0.0000e+00]
 [  1.6175e-02   9.7655e-01   7.2786e-03   0.0000e+00   0.0000e+00
    0.0000e+00   0.0000e+00]
 [  0.0000e+00   3.4896e-01   6.5104e-01   0.0000e+00   0.0000e+00
    0.0000e+00   0.0000e+00]
 [  8.2251e-01   1.4393e-01   0.0000e+00   0.0000e+00   0.0000e+00
    0.0000e+00   3.3564e-02]]
```



```
('Percentage accuracy:', 66.202419227141604)
```

Now we plot the out-of-bag (OOB) error alongside the train and test error with the number of estimators used. It can be seen that OOB error is always more than train and test error.



1.1.3 Trends observed

- When training the forest with trees grown to full extent it is observed that the test error reduces steeply with increasing number of trees upto a number of 40. Beyond that the train and test error practically reach a plateau i.e adding more number of trees does not reduce the test error significantly.
- When training the forest with trees of maximum depth of 5 it is observed that the test error doesn't vary much (in magnitude as can be seen from the y-axis of plot) with B. Beyond 20 trees the train and test error don't show any significant change.
- For both type of forests we obtain quite similar sorting for the most important features. Thus it is clear that features like Elevation, Horizontal_Distance_To_Roadways, Horizontal_Distance_To_Fire_Points and Horizontal_Distance_To_Hydrology are some of the most empirical features for the classification of covertype data since they have the highest gini importance values with Elevation being by the far the most important. The evidence seems logically correct too as elevation will have a major effect on the classification of cover types as opposed to some other features like soil type and wilderness area which either have very low variance or low gini importance.
- The accuracy when training the forest with trees of maximum depth of 5 is very low when compared to that when trees are grown to full extent as can be seen from both the confusion matrices. With max depth 5 we observe lots of misclassification for classes 4,5,6 and 7. This can be attributed to the fact that the classification power (accuracy) reduces a lot once we impose a limit on tree depth especially when the number of classes in the data set is 7 (more than 5).

1.2 MNIST Dataset

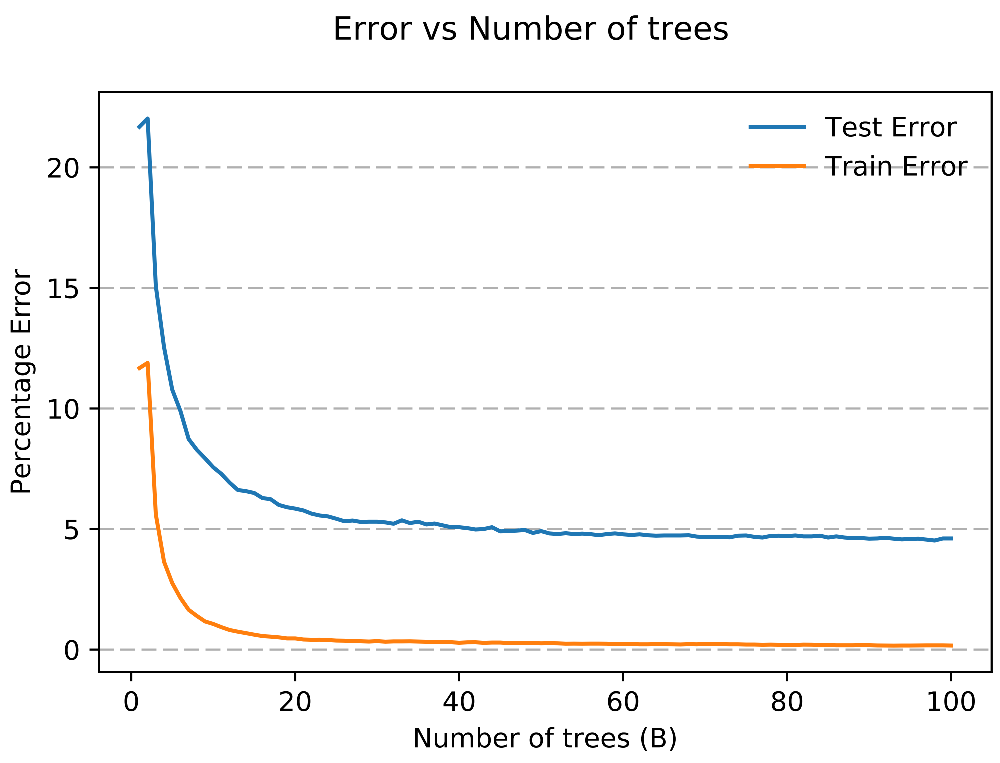
Since the test data available on Kaggle does not contain class labels for the test set, we use train data and separate it into train set and test set with train set having 75% of the data and the test set having 25% of the data selected randomly.

```
(31500, 589)
(10500, 589)
```

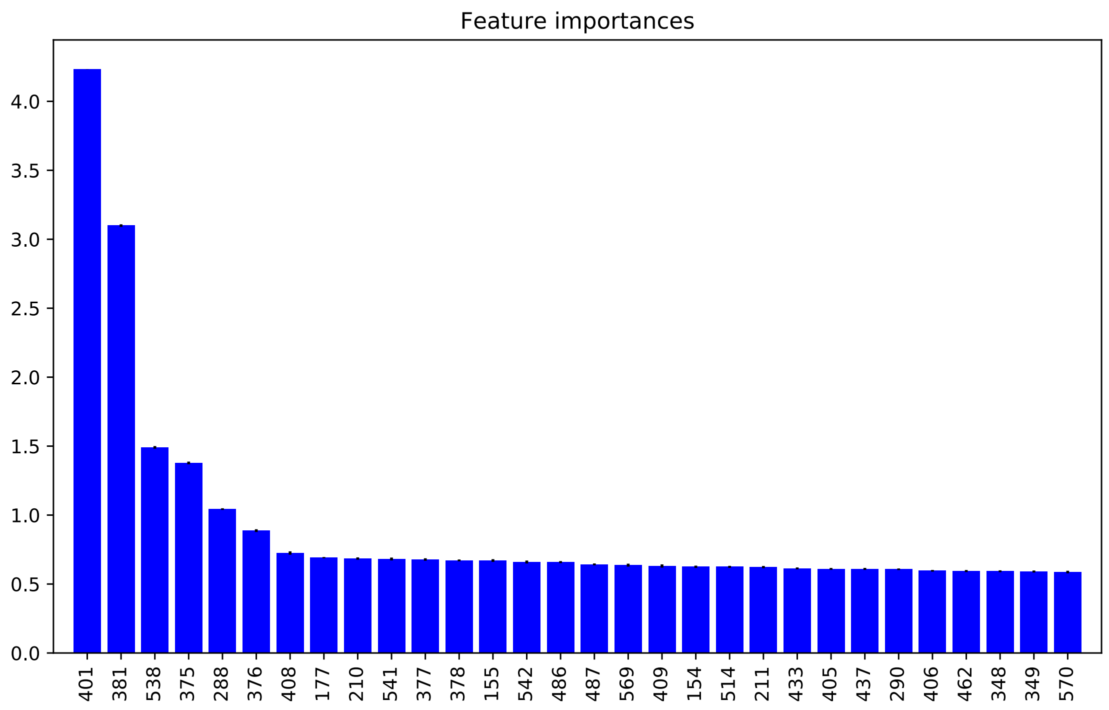
1.2.1 Random Forest upto Full Depth

First we let the tree go upto full depth until the nodes cannot be split any further. We use 100 trees to train random forest and then we subsample for different number of trees to show how the error changes with respect to number of trees.

```
Out[7]: <__main__.MyRandomForest instance at 0x1038b0680>
```



Here we see that the training error is about 11% when we are using a single tree and it falls down as the number of trees increases and eventually plateaus at about 20 trees with less than 0.4% error. The test error is about 21% when we are using a single tree and it falls down and plateaus at about 40 trees with approx 5% error. So 40 trees seems to be a good number of trees that can be used for our random forest. Now we plot the feature importances for top 30 features.



```
Feature ranking:
1. feature 401 - (4.232651)
2. feature 381 - (3.098981)
3. feature 538 - (1.491012)
4. feature 375 - (1.378095)
5. feature 288 - (1.043474)
6. feature 376 - (0.887364)
7. feature 408 - (0.725119)
8. feature 177 - (0.691372)
9. feature 210 - (0.684876)
10. feature 541 - (0.681696)
11. feature 377 - (0.677999)
12. feature 378 - (0.671539)
13. feature 155 - (0.670619)
14. feature 542 - (0.659834)
15. feature 486 - (0.659078)
16. feature 487 - (0.642669)
17. feature 569 - (0.636989)
18. feature 409 - (0.630948)
19. feature 154 - (0.625181)
20. feature 514 - (0.624587)
21. feature 211 - (0.623089)
22. feature 433 - (0.613615)
23. feature 405 - (0.609934)
24. feature 437 - (0.609769)
25. feature 290 - (0.607324)
26. feature 406 - (0.596031)
27. feature 462 - (0.594330)
```

```
28. feature 348 - (0.592837)
29. feature 349 - (0.590841)
30. feature 570 - (0.587161)
31. feature 460 - (0.585337)
32. feature 488 - (0.582973)
33. feature 434 - (0.581426)
34. feature 350 - (0.574363)
35. feature 212 - (0.570393)
36. feature 515 - (0.559468)
37. feature 543 - (0.557208)
38. feature 209 - (0.550853)
39. feature 513 - (0.543953)
40. feature 657 - (0.539716)
41. feature 461 - (0.537431)
42. feature 404 - (0.522355)
43. feature 347 - (0.520156)
44. feature 153 - (0.518669)
45. feature 156 - (0.511408)
46. feature 489 - (0.502680)
47. feature 319 - (0.499711)
48. feature 239 - (0.484366)
49. feature 656 - (0.482017)
50. feature 625 - (0.461232)
51. feature 273 - (0.460024)
52. feature 432 - (0.449744)
53. feature 238 - (0.446732)
54. feature 516 - (0.444838)
55. feature 435 - (0.442457)
56. feature 351 - (0.438805)
57. feature 379 - (0.438228)
58. feature 380 - (0.429189)
59. feature 568 - (0.422346)
60. feature 296 - (0.421500)
61. feature 269 - (0.409377)
62. feature 655 - (0.408272)
63. feature 517 - (0.404829)
64. feature 523 - (0.400360)
65. feature 291 - (0.399013)
66. feature 237 - (0.388887)
67. feature 297 - (0.386427)
68. feature 100 - (0.386394)
69. feature 267 - (0.384968)
70. feature 459 - (0.382704)
71. feature 152 - (0.382388)
72. feature 658 - (0.378324)
73. feature 624 - (0.375678)
74. feature 182 - (0.373899)
75. feature 318 - (0.372748)
76. feature 485 - (0.372416)
77. feature 271 - (0.372386)
78. feature 183 - (0.368430)
79. feature 157 - (0.368399)
80. feature 407 - (0.367963)
81. feature 571 - (0.367836)
```

```
82. feature 463 - (0.362997)
83. feature 294 - (0.360448)
84. feature 323 - (0.359536)
85. feature 597 - (0.358501)
86. feature 352 - (0.357526)
87. feature 298 - (0.355848)
88. feature 264 - (0.355487)
89. feature 295 - (0.349162)
90. feature 184 - (0.347549)
91. feature 240 - (0.345930)
92. feature 242 - (0.343576)
93. feature 544 - (0.342479)
94. feature 179 - (0.339418)
95. feature 236 - (0.339411)
96. feature 325 - (0.339024)
97. feature 268 - (0.336812)
98. feature 484 - (0.334377)
99. feature 270 - (0.334334)
100. feature 266 - (0.334250)
101. feature 265 - (0.333601)
102. feature 490 - (0.330770)
103. feature 320 - (0.330345)
104. feature 243 - (0.329904)
105. feature 374 - (0.324968)
106. feature 185 - (0.324923)
107. feature 324 - (0.324788)
108. feature 483 - (0.323341)
109. feature 263 - (0.320867)
110. feature 181 - (0.312246)
111. feature 322 - (0.311106)
112. feature 595 - (0.307867)
113. feature 241 - (0.306212)
114. feature 292 - (0.306170)
115. feature 326 - (0.306064)
116. feature 626 - (0.295686)
117. feature 540 - (0.295410)
118. feature 512 - (0.292583)
119. feature 436 - (0.291513)
120. feature 328 - (0.289589)
121. feature 491 - (0.287260)
122. feature 346 - (0.287251)
123. feature 321 - (0.282664)
124. feature 327 - (0.282314)
125. feature 317 - (0.282199)
126. feature 572 - (0.280827)
127. feature 213 - (0.280790)
128. feature 299 - (0.279183)
129. feature 598 - (0.278975)
130. feature 458 - (0.277996)
131. feature 289 - (0.275714)
132. feature 262 - (0.275701)
133. feature 457 - (0.273186)
134. feature 596 - (0.272556)
135. feature 539 - (0.261895)
```

```
136. feature 151 - (0.259733)
137. feature 354 - (0.258393)
138. feature 431 - (0.257485)
139. feature 293 - (0.257382)
140. feature 456 - (0.256969)
141. feature 403 - (0.254495)
142. feature 573 - (0.250814)
143. feature 126 - (0.246686)
144. feature 654 - (0.244491)
145. feature 455 - (0.240819)
146. feature 158 - (0.240364)
147. feature 125 - (0.239956)
148. feature 443 - (0.239925)
149. feature 518 - (0.238554)
150. feature 627 - (0.232564)
151. feature 353 - (0.231697)
152. feature 272 - (0.228645)
153. feature 427 - (0.228621)
154. feature 578 - (0.227830)
155. feature 214 - (0.224822)
156. feature 300 - (0.224073)
157. feature 150 - (0.223652)
158. feature 464 - (0.222707)
159. feature 426 - (0.217119)
160. feature 545 - (0.216770)
161. feature 398 - (0.211564)
162. feature 653 - (0.211125)
163. feature 178 - (0.206381)
164. feature 659 - (0.204423)
165. feature 454 - (0.201735)
166. feature 180 - (0.199564)
167. feature 355 - (0.199150)
168. feature 567 - (0.198465)
169. feature 329 - (0.197586)
170. feature 574 - (0.197125)
171. feature 438 - (0.196072)
172. feature 623 - (0.195788)
173. feature 216 - (0.194454)
174. feature 215 - (0.192236)
175. feature 371 - (0.190861)
176. feature 235 - (0.190477)
177. feature 357 - (0.189209)
178. feature 261 - (0.188773)
179. feature 413 - (0.188058)
180. feature 316 - (0.182967)
181. feature 492 - (0.181440)
182. feature 244 - (0.179653)
183. feature 524 - (0.177505)
184. feature 358 - (0.176616)
185. feature 343 - (0.175708)
186. feature 345 - (0.175535)
187. feature 399 - (0.174792)
188. feature 550 - (0.174175)
189. feature 511 - (0.173683)
```

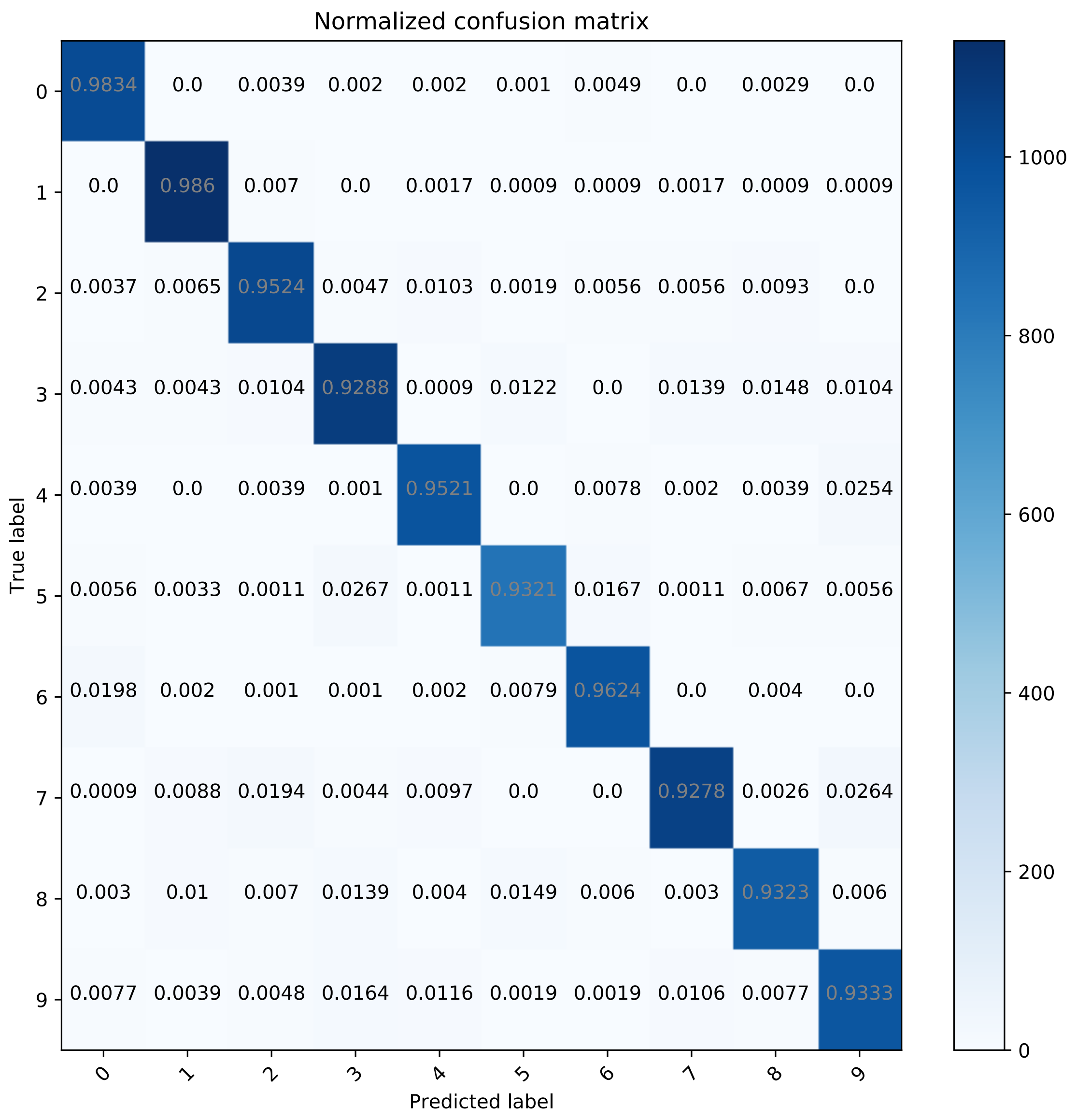
```
190. feature 186 - (0.172861)
191. feature 159 - (0.171844)
192. feature 382 - (0.171465)
193. feature 208 - (0.171231)
194. feature 467 - (0.170500)
195. feature 439 - (0.168699)
196. feature 344 - (0.167947)
197. feature 628 - (0.165844)
198. feature 127 - (0.165353)
199. feature 430 - (0.165110)
200. feature 101 - (0.164316)
```

Here we see the Gini Importance of different features. Since we have about 589 features, it's not possible to plot for all features hence we plotted for 30 most important ones. The pixel at 401th position has the highest Gini Importance followed by pixel at position 381. Also if we plot for all 589 features then we can notice that features after the top 282 have a very low Gini Importance. Even after rejecting the pixels that have a medium high variance from our dataset in the beginning, we get a lot of features that turn out to be not important in our case of Random Forest Classifier. However these features may or may not be useful when used in some other method for classification. We also noticed that features at positions 774, 765, 39 and 46 have a gini importance of 0 so they are practically useless in our classifier. Also, these features are beyond the threshold value for variance so it is safe to assume that all the features that we rejected due to variance lower than threshold also have a gini importance of 0. So they are also not useful for our Random Forest.

Then we plot the final confusion matrix with 40 number of trees.

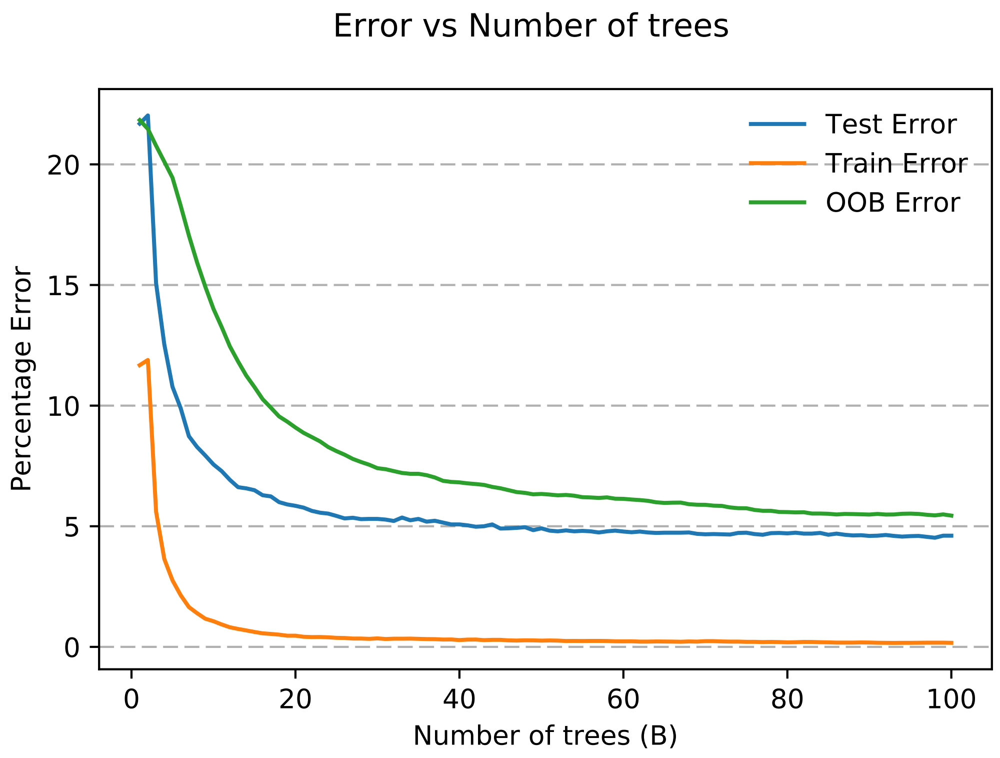
```
Confusion matrix:
[[1008    0    4    2    2    1    5    0    3    0]
 [   0 1130    8    0    2    1    1    2    1    1]
 [   4    7 1021    5   11    2    6    6   10    0]
 [   5    5   12 1069    1   14    0   16   17   12]
 [   4    0    4    1  975    0    8    2    4   26]
 [   5    3    1   24    1  837   15    1    6    5]
 [  20    2    1    1    2    8  972    0    4    0]
 [   1   10   22    5   11    0    0 1053    3   30]
 [   3   10    7   14    4   15    6    3  937    6]
 [   8    4    5   17   12    2    2   11    8  965]]
Normalized confusion matrix
[[  9.8341e-01   0.0000e+00   3.9024e-03   1.9512e-03   1.9512e-03
    9.7561e-04   4.8780e-03   0.0000e+00   2.9268e-03   0.0000e+00]
 [  0.0000e+00   9.8604e-01   6.9808e-03   0.0000e+00   1.7452e-03
    8.7260e-04   8.7260e-04   1.7452e-03   8.7260e-04   8.7260e-04]
 [  3.7313e-03   6.5299e-03   9.5243e-01   4.6642e-03   1.0261e-02
    1.8657e-03   5.5970e-03   5.5970e-03   9.3284e-03   0.0000e+00]
 [  4.3440e-03   4.3440e-03   1.0426e-02   9.2876e-01   8.6881e-04
    1.2163e-02   0.0000e+00   1.3901e-02   1.4770e-02   1.0426e-02]
 [  3.9062e-03   0.0000e+00   3.9062e-03   9.7656e-04   9.5215e-01
    0.0000e+00   7.8125e-03   1.9531e-03   3.9062e-03   2.5391e-02]
 [  5.5679e-03   3.3408e-03   1.1136e-03   2.6726e-02   1.1136e-03
    9.3207e-01   1.6704e-02   1.1136e-03   6.6815e-03   5.5679e-03]
 [  1.9802e-02   1.9802e-03   9.9010e-04   9.9010e-04   1.9802e-03
    7.9208e-03   9.6238e-01   0.0000e+00   3.9604e-03   0.0000e+00]
 [  8.8106e-04   8.8106e-03   1.9383e-02   4.4053e-03   9.6916e-03
    0.0000e+00   0.0000e+00   9.2775e-01   2.6432e-03   2.6432e-02]
 [  2.9851e-03   9.9502e-03   6.9652e-03   1.3930e-02   3.9801e-03
```

```
    1.4925e-02   5.9701e-03   2.9851e-03   9.3234e-01   5.9701e-03]
 [  7.7369e-03   3.8685e-03   4.8356e-03   1.6441e-02   1.1605e-02
    1.9342e-03   1.9342e-03   1.0638e-02   7.7369e-03   9.3327e-01]]
```



```
('Percentage accuracy:', 94.923809523809524)
```

Through the confusion matrix we can see that it classifies 0 and 1 really well with about 98% accuracy followed by 6 with 96% accuracy. Overall our implementation works well as it classifies all of the test sets with about atleast 92% accuracy

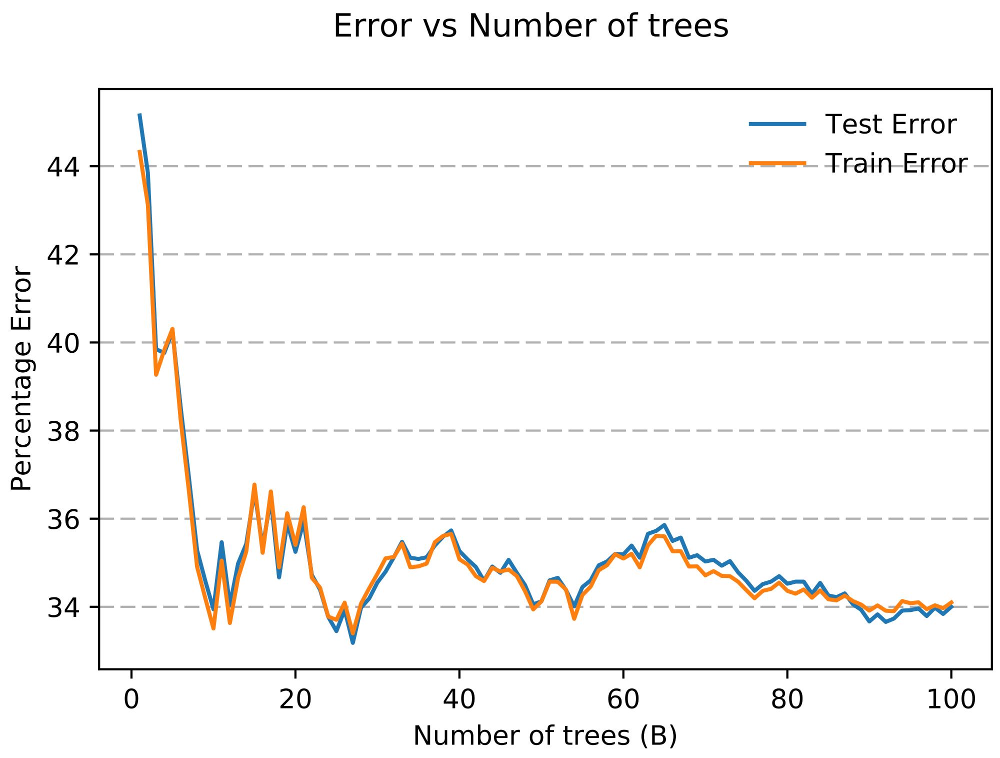


The out of bag error is at about 21% when only one tree is used and it slowly drops down to about 5.45% when all 100 trees are used. It almost pleateaus when we use about 45 trees to about 6% error.

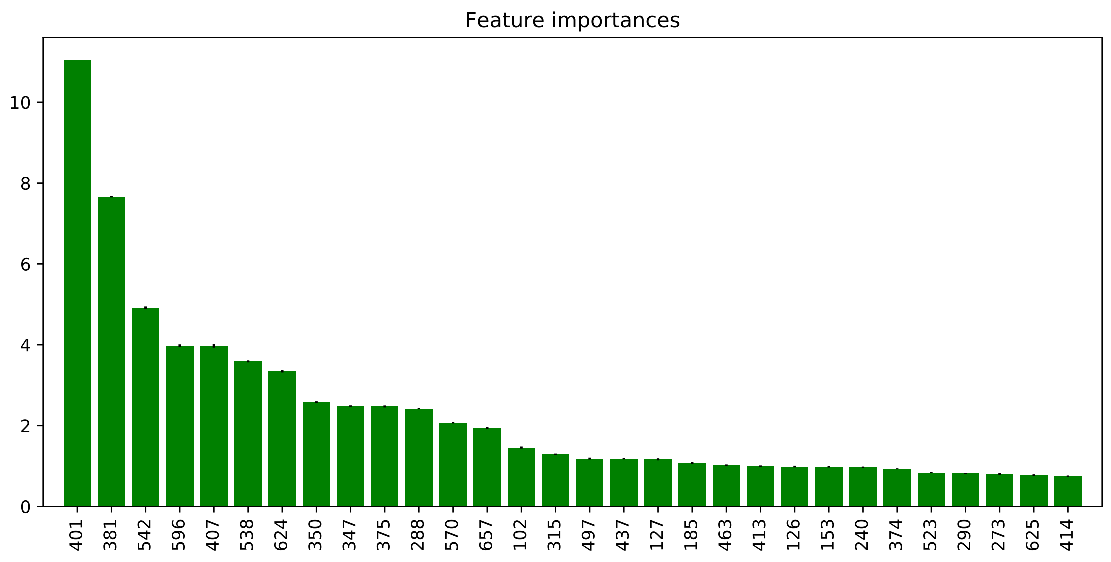
1.2.2 Random Forest upto depth of 5 Trees

Now we train our classifier such that each of the 100 trees can have a maximum depth of 5 and plot the train and test error.

```
Out[24]: <__main__.MyRandomForest instance at 0x10f0fc878>
```



Because we are not allowing the tree to go upto its full depth, the classification accuracy is lower than when its allowed to go upto its full depth. This is because since every decision tree is stopped at the depth of 5, it cannot fully classify a particular data entry to a particular class. So it may sometimes correctly classify it and sometimes may not. Hence the error rate fluctuates a little. From the above graph we choose the final number of trees to be 10 at which we observe a dip in test error.



```
Feature ranking:
1. feature 401 - (11.043572)
2. feature 381 - (7.660720)
3. feature 542 - (4.918756)
4. feature 596 - (3.979784)
5. feature 407 - (3.972182)
6. feature 538 - (3.589338)
7. feature 624 - (3.342261)
8. feature 350 - (2.580536)
9. feature 347 - (2.484854)
10. feature 375 - (2.474720)
11. feature 288 - (2.419133)
12. feature 570 - (2.071629)
13. feature 657 - (1.939072)
14. feature 102 - (1.458146)
15. feature 315 - (1.294135)
16. feature 497 - (1.185357)
17. feature 437 - (1.183483)
18. feature 127 - (1.165964)
19. feature 185 - (1.075430)
20. feature 463 - (1.023277)
21. feature 413 - (0.999596)
22. feature 126 - (0.987319)
23. feature 153 - (0.982543)
24. feature 240 - (0.968511)
25. feature 374 - (0.931499)
26. feature 523 - (0.839667)
27. feature 290 - (0.816657)
28. feature 273 - (0.804922)
29. feature 625 - (0.778140)
30. feature 414 - (0.747765)
31. feature 236 - (0.736101)
32. feature 514 - (0.711331)
33. feature 432 - (0.671804)
34. feature 316 - (0.613257)
35. feature 443 - (0.612434)
36. feature 455 - (0.600489)
37. feature 182 - (0.537792)
38. feature 211 - (0.530824)
39. feature 487 - (0.527039)
40. feature 409 - (0.504627)
41. feature 490 - (0.501157)
42. feature 244 - (0.496765)
43. feature 378 - (0.485961)
44. feature 408 - (0.481705)
45. feature 572 - (0.465950)
46. feature 183 - (0.457627)
47. feature 598 - (0.449705)
48. feature 684 - (0.424670)
49. feature 460 - (0.411421)
50. feature 327 - (0.405776)
51. feature 484 - (0.361451)
52. feature 541 - (0.346943)
53. feature 238 - (0.335042)
```

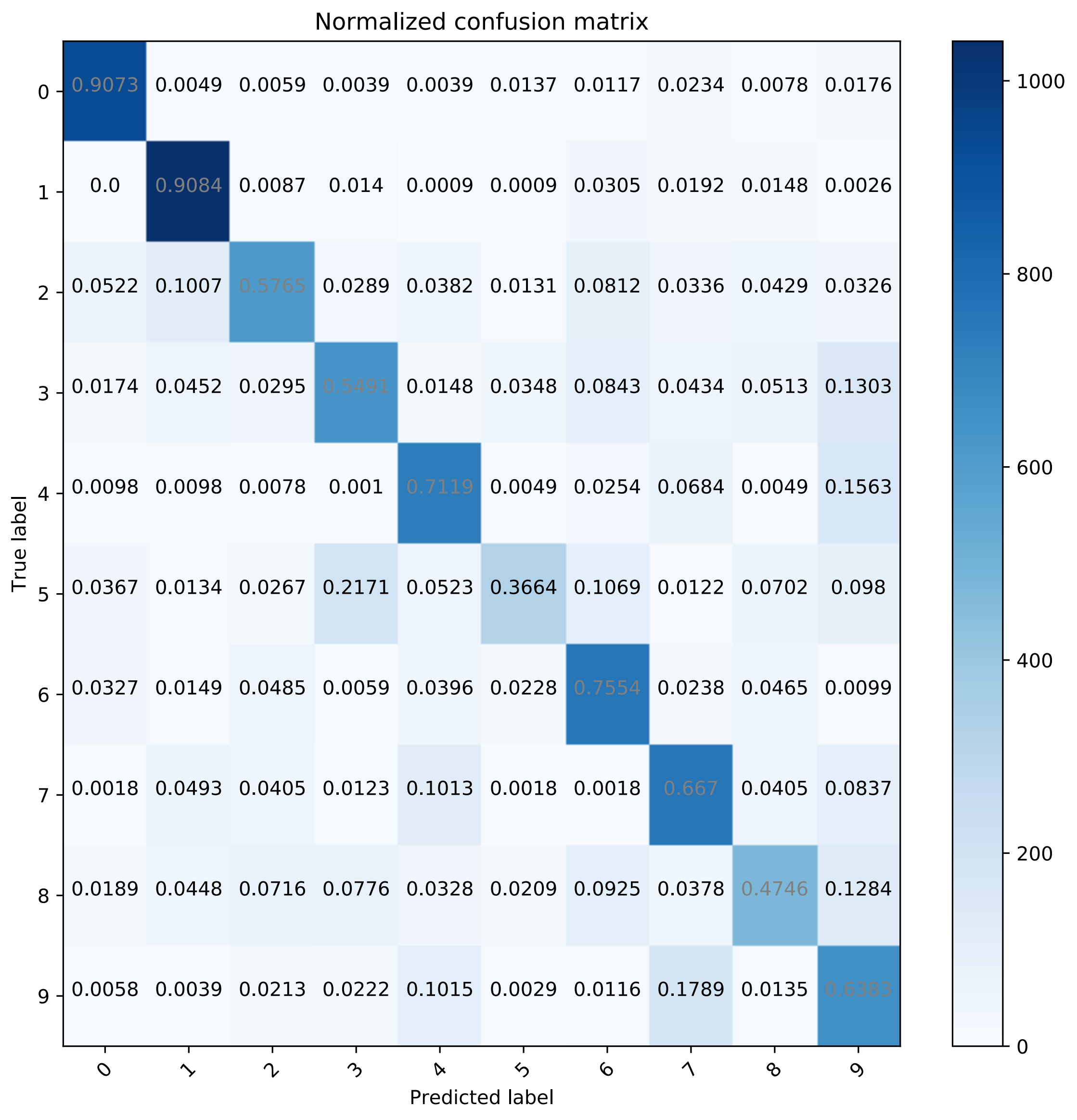
```
54. feature 459 - (0.321375)
55. feature 469 - (0.316493)
56. feature 405 - (0.308247)
57. feature 352 - (0.306291)
58. feature 241 - (0.300731)
59. feature 595 - (0.299374)
60. feature 656 - (0.298458)
61. feature 301 - (0.285485)
62. feature 438 - (0.284324)
63. feature 326 - (0.283678)
64. feature 512 - (0.283331)
65. feature 181 - (0.281918)
66. feature 213 - (0.275368)
67. feature 609 - (0.263913)
68. feature 325 - (0.262523)
69. feature 517 - (0.257108)
70. feature 158 - (0.253612)
71. feature 272 - (0.251248)
72. feature 267 - (0.248439)
73. feature 577 - (0.248118)
74. feature 377 - (0.247770)
75. feature 386 - (0.241368)
76. feature 456 - (0.239437)
77. feature 626 - (0.238541)
78. feature 179 - (0.235473)
79. feature 654 - (0.234384)
80. feature 658 - (0.232583)
81. feature 441 - (0.232082)
82. feature 152 - (0.227988)
83. feature 540 - (0.213214)
84. feature 212 - (0.212264)
85. feature 515 - (0.195722)
86. feature 426 - (0.194926)
87. feature 215 - (0.193589)
88. feature 489 - (0.191591)
89. feature 217 - (0.186752)
90. feature 488 - (0.186529)
91. feature 552 - (0.176167)
92. feature 685 - (0.175502)
93. feature 602 - (0.174945)
94. feature 154 - (0.174719)
95. feature 351 - (0.165663)
96. feature 346 - (0.164784)
97. feature 599 - (0.162044)
98. feature 98 - (0.159776)
99. feature 321 - (0.159447)
100. feature 103 - (0.152799)
101. feature 184 - (0.148688)
102. feature 376 - (0.142023)
103. feature 486 - (0.138682)
104. feature 627 - (0.136505)
105. feature 220 - (0.133119)
106. feature 385 - (0.131572)
107. feature 513 - (0.130455)
```

```
108. feature 328 - (0.129633)
109. feature 568 - (0.126441)
110. feature 628 - (0.126094)
111. feature 655 - (0.125711)
112. feature 242 - (0.124943)
113. feature 353 - (0.120446)
114. feature 384 - (0.116209)
115. feature 380 - (0.115003)
116. feature 297 - (0.109955)
117. feature 397 - (0.108823)
118. feature 274 - (0.107561)
119. feature 543 - (0.101098)
120. feature 427 - (0.093083)
121. feature 404 - (0.091972)
122. feature 545 - (0.091877)
123. feature 711 - (0.091753)
124. feature 151 - (0.091407)
125. feature 292 - (0.090675)
126. feature 485 - (0.090584)
127. feature 330 - (0.089927)
128. feature 291 - (0.086809)
129. feature 268 - (0.083498)
130. feature 516 - (0.081490)
131. feature 509 - (0.080024)
132. feature 243 - (0.079970)
133. feature 387 - (0.078928)
134. feature 216 - (0.078040)
135. feature 296 - (0.076609)
136. feature 329 - (0.071992)
137. feature 69 - (0.071216)
138. feature 524 - (0.068834)
139. feature 382 - (0.067856)
140. feature 575 - (0.065648)
141. feature 462 - (0.063332)
142. feature 128 - (0.062845)
143. feature 317 - (0.062793)
144. feature 603 - (0.062688)
145. feature 245 - (0.061130)
146. feature 318 - (0.061027)
147. feature 233 - (0.059487)
148. feature 344 - (0.058169)
149. feature 354 - (0.057172)
150. feature 544 - (0.056597)
151. feature 683 - (0.056172)
152. feature 492 - (0.055959)
153. feature 298 - (0.055843)
154. feature 208 - (0.054800)
155. feature 295 - (0.054024)
156. feature 686 - (0.053561)
157. feature 300 - (0.053390)
158. feature 629 - (0.050471)
159. feature 597 - (0.049873)
160. feature 289 - (0.049636)
161. feature 262 - (0.049349)
```

```
162. feature 553 - (0.048872)
163. feature 518 - (0.048388)
164. feature 124 - (0.045221)
165. feature 594 - (0.044668)
166. feature 482 - (0.044599)
167. feature 319 - (0.043932)
168. feature 464 - (0.043843)
169. feature 358 - (0.041737)
170. feature 461 - (0.040658)
171. feature 264 - (0.040634)
172. feature 600 - (0.039254)
173. feature 186 - (0.038780)
174. feature 155 - (0.038463)
175. feature 324 - (0.037546)
176. feature 355 - (0.036620)
177. feature 454 - (0.036419)
178. feature 526 - (0.034805)
179. feature 287 - (0.034366)
180. feature 440 - (0.034301)
181. feature 261 - (0.034274)
182. feature 246 - (0.033802)
183. feature 494 - (0.032229)
184. feature 458 - (0.031233)
185. feature 651 - (0.031193)
186. feature 457 - (0.030440)
187. feature 345 - (0.029773)
188. feature 632 - (0.028072)
189. feature 265 - (0.027579)
190. feature 579 - (0.027316)
191. feature 537 - (0.026614)
192. feature 578 - (0.026471)
193. feature 270 - (0.025927)
194. feature 433 - (0.025469)
195. feature 519 - (0.025307)
196. feature 571 - (0.025228)
197. feature 302 - (0.025183)
198. feature 662 - (0.024955)
199. feature 294 - (0.024417)
200. feature 415 - (0.023913)
```

Again, since we have about 589 features, the graph doesn't give a clear picture of the importances. Pixel at position 401 has the highest Gini importance and hence is the most important feature for classification followed by pixel at position 381 and 542. Pixels ranked from 4th to 279 have Gini importance relatively close to their preceding and succeeding pixels. All the pixels after rank 280 have Gini importance of 0 so they have no importance in our classifier.

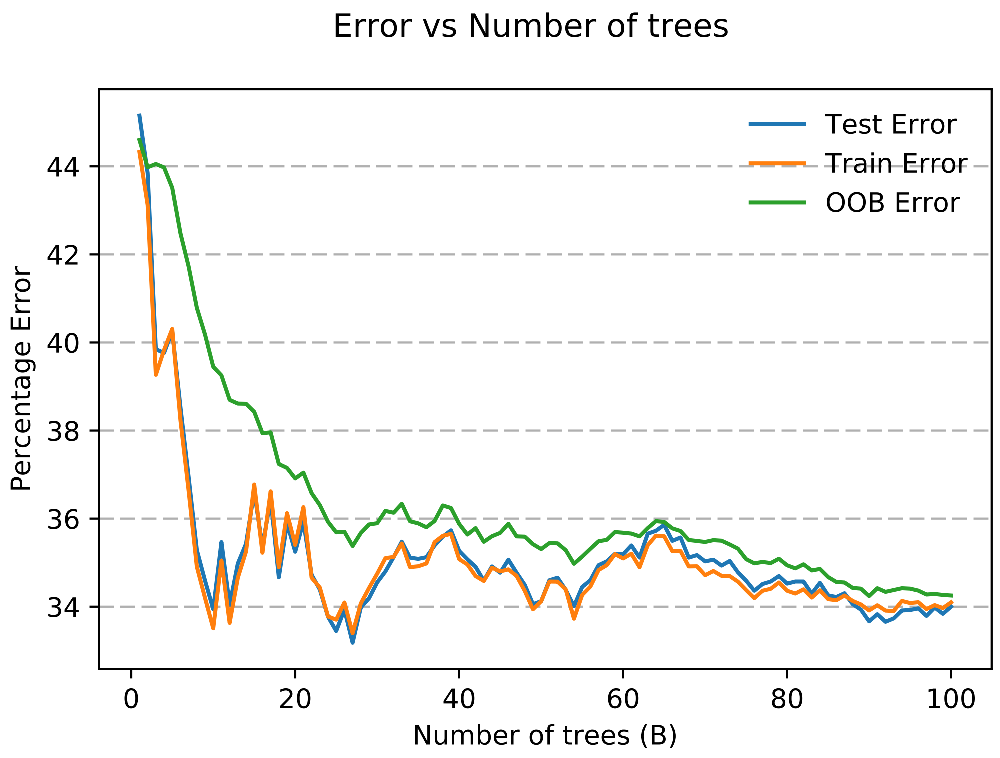
```
Confusion matrix:
[[ 930    5    6    4    4   14   12   24    8   18]
 [    0 1041   10   16    1    1   35   22   17    3]
 [  56  108  618   31   41   14   87   36   46   35]
 [  20   52   34  632   17   40   97   50   59  150]
 [  10   10    8    1  729    5   26   70    5  160]
 [  33   12   24  195   47  329   96   11   63   88]
 [  33   15   49    6   40   23  763   24   47   10]
 [   2   56   46   14  115    2    2  757   46   95]
 [  19   45   72   78   33   21   93   38  477  129]
 [   6    4   22   23  105    3   12  185   14  660]]
Normalized confusion matrix
[[  9.0732e-01   4.8780e-03   5.8537e-03   3.9024e-03   3.9024e-03
    1.3659e-02   1.1707e-02   2.3415e-02   7.8049e-03   1.7561e-02]
 [  0.0000e+00   9.0838e-01   8.7260e-03   1.3962e-02   8.7260e-04
    8.7260e-04   3.0541e-02   1.9197e-02   1.4834e-02   2.6178e-03]
 [  5.2239e-02   1.0075e-01   5.7649e-01   2.8918e-02   3.8246e-02
    1.3060e-02   8.1157e-02   3.3582e-02   4.2910e-02   3.2649e-02]
 [  1.7376e-02   4.5178e-02   2.9540e-02   5.4909e-01   1.4770e-02
    3.4752e-02   8.4275e-02   4.3440e-02   5.1260e-02   1.3032e-01]
 [  9.7656e-03   9.7656e-03   7.8125e-03   9.7656e-04   7.1191e-01
    4.8828e-03   2.5391e-02   6.8359e-02   4.8828e-03   1.5625e-01]
 [  3.6748e-02   1.3363e-02   2.6726e-02   2.1715e-01   5.2339e-02
    3.6637e-01   1.0690e-01   1.2249e-02   7.0156e-02   9.7996e-02]
 [  3.2673e-02   1.4851e-02   4.8515e-02   5.9406e-03   3.9604e-02
    2.2772e-02   7.5545e-01   2.3762e-02   4.6535e-02   9.9010e-03]
 [  1.7621e-03   4.9339e-02   4.0529e-02   1.2335e-02   1.0132e-01
    1.7621e-03   1.7621e-03   6.6696e-01   4.0529e-02   8.3700e-02]
 [  1.8905e-02   4.4776e-02   7.1642e-02   7.7612e-02   3.2836e-02
    2.0896e-02   9.2537e-02   3.7811e-02   4.7463e-01   1.2836e-01]
 [  5.8027e-03   3.8685e-03   2.1277e-02   2.2244e-02   1.0155e-01
    2.9014e-03   1.1605e-02   1.7892e-01   1.3540e-02   6.3830e-01]]
```



	0	1	2	3	4	5	6	7	8	9
0	0.9073	0.0049	0.0059	0.0039	0.0039	0.0137	0.0117	0.0234	0.0078	0.0176
1	0.0	0.9084	0.0087	0.014	0.0009	0.0009	0.0305	0.0192	0.0148	0.0026
2	0.0522	0.1007	0.5765	0.0289	0.0382	0.0131	0.0812	0.0336	0.0429	0.0326
3	0.0174	0.0452	0.0295	0.5491	0.0148	0.0348	0.0843	0.0434	0.0513	0.1303
4	0.0098	0.0098	0.0078	0.001	0.7119	0.0049	0.0254	0.0684	0.0049	0.1563
5	0.0367	0.0134	0.0267	0.2171	0.0523	0.3664	0.1069	0.0122	0.0702	0.098
6	0.0327	0.0149	0.0485	0.0059	0.0396	0.0228	0.7554	0.0238	0.0465	0.0099
7	0.0018	0.0493	0.0405	0.0123	0.1013	0.0018	0.0018	0.667	0.0405	0.0837
8	0.0189	0.0448	0.0716	0.0776	0.0328	0.0209	0.0925	0.0378	0.4746	0.1284
9	0.0058	0.0039	0.0213	0.0222	0.1015	0.0029	0.0116	0.1789	0.0135	0.6383

```
('Percentage accuracy:', 66.057142857142864)
```

We get a percentage accuracy of about 66.5% With a limited depth of 5, random forest is able to correctly classify about 90% of digits 0 correctly and 1. It has the lowest classification accuracy for digit 5 with about 36% which it confuses with digit 3 most of the time and slightly with digit 9. Somehow it confuses digit 2 with all other digits almost equally but is slightly more biased to incorrectly classify it as digits 6 and 8 than other incorrect digits. About 54% of digit 3 images are correctly classified. It generally misclassifies digit 3 as digit 2 or digit 9. About 15% of digit 4 images is misclassified as digit 9. Similarly about 8% of digit 7 images are misclassified as digit 9, about 7% of digit 8 are misclassified as digit 3 and about 18% of digit 9 images are misclassified as digit 7



Out of Bag error always stays higher than test error in our case

1.2.3 Trends Observed

- When we allow the decision tree to go upto its full depth it can correctly classify most of the digits but when we give it a max depth of 5, it's classification accuracy drops down from 94% to about 66%.
- The error is highly fluctuating when depth of tree is set to 5 as compared to the smooth curve that we get when the trees are allowed to go upto the max depth.
- There are more features relevant (have Gini importance) when the trees are allowed to go upto the full depth. But when we have max depth of 5, a lot of features are not even used for the classification purposes and as a result they have a Gini importance of 0.
- Pixel 401 has the highest Gini importance in both the cases when the trees are allowed to go upto maximum depth and when max depth is 5. Since there are fewer relevant features when max depth is 5(like said above), the Gini importance of pixel 401 is relatively higher in case of max depth 5 compared to the full depth case.

Contents

Adaboost

Our implementation of multiclass Adaboost can be found in class with file named MyAdaBoost.py. We use sklearn's DecisionTreeClassifier as weak learners inside the algorithm. We implemented SAMME - Stagewise Additive Modeling algorithm using a Multi-class Exponential loss function. The algorithm steps can be found in Section 1.2 here https://web.stanford.edu/~hastie/Papers/samme.pdf. Also we train (fit) the classifier with the maximum number of weak learners first and then sample from it to simulate (while predicting predict) classifier with any lower number of weak lerners while generating the training or test error.

1.1 Covertype Dataset

First we import the covertype dataset and split it into training set (75%) and test set (25%)

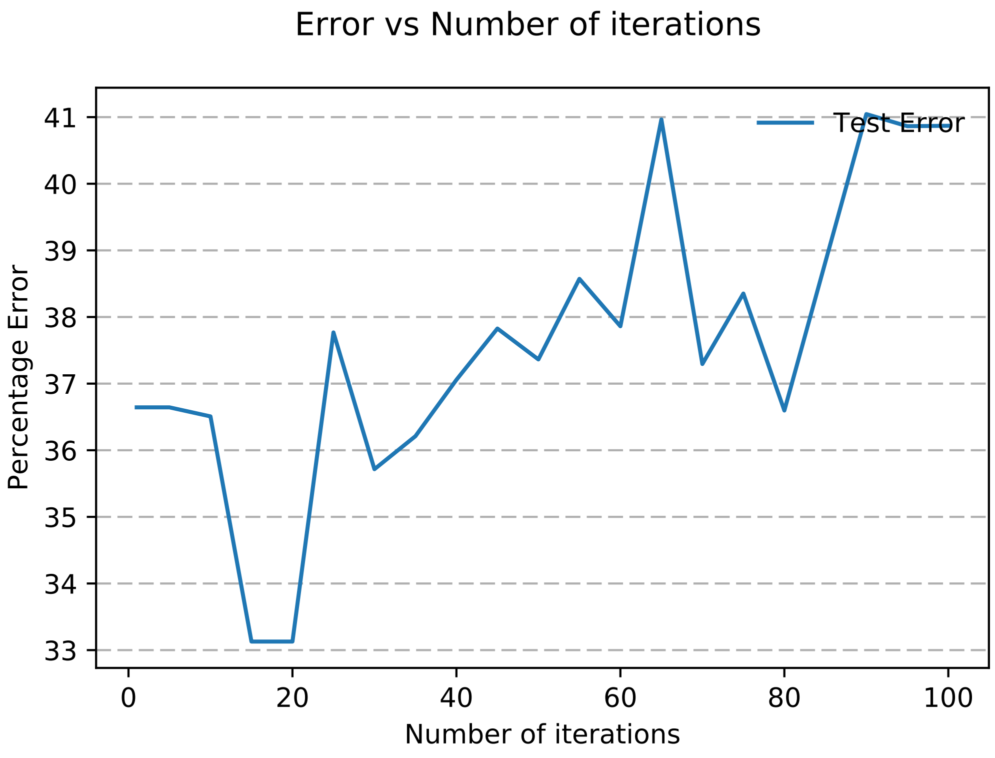
Then we use sklearn's VarianceThreshold selector to weed out dimensions having same value in more than 90% of the samples. This reduces the dimension of the input rows from 54 to 13.

1.1.1 AdaBoost in which the learner is a decision stump (one level tree)

First we train a AdaBoost classifier in which each of the learner is a decision stump (one level tree) upto a total of 100 iterations and then we subsample from the trained clasifier to plot test and training error with respect to the number of weak learners (iterations).

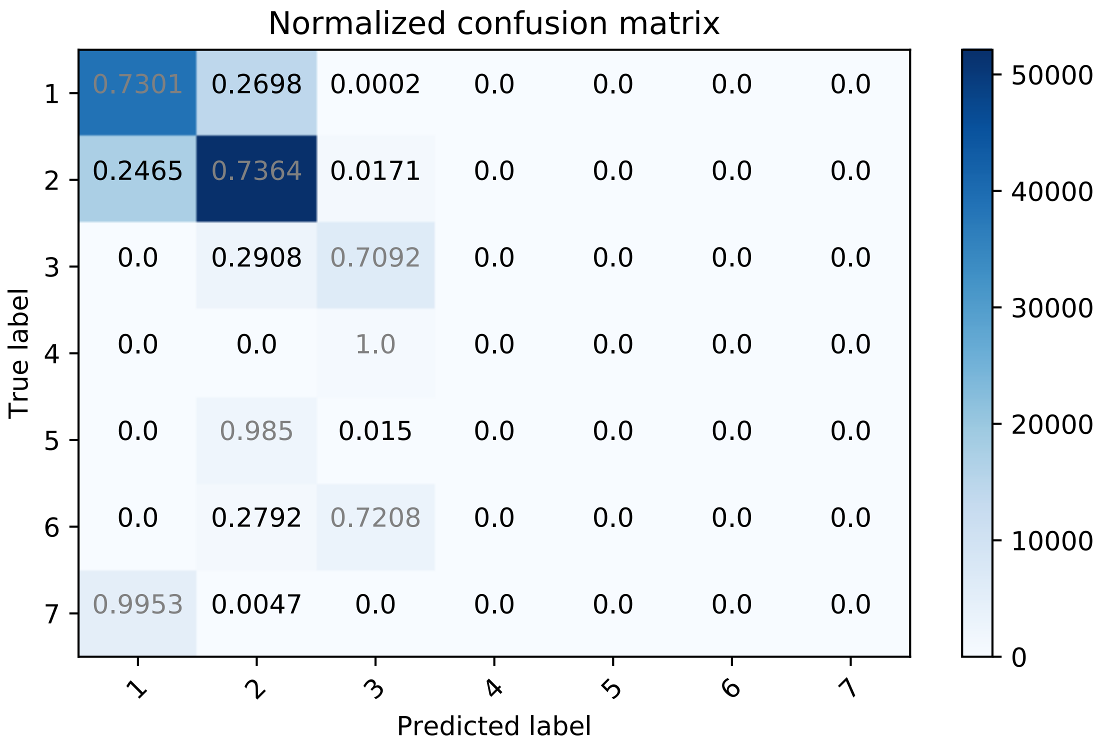
```
Out[60]: <__main__.MyAdaBoost instance at 0x1081c9758>
```

We plot the test error with number of iterations upto 100. In order to reduce the plotting time we subsampled the plot at a value of 5. Below is the plot of error with iterations as [1,5,10,15,..,100].



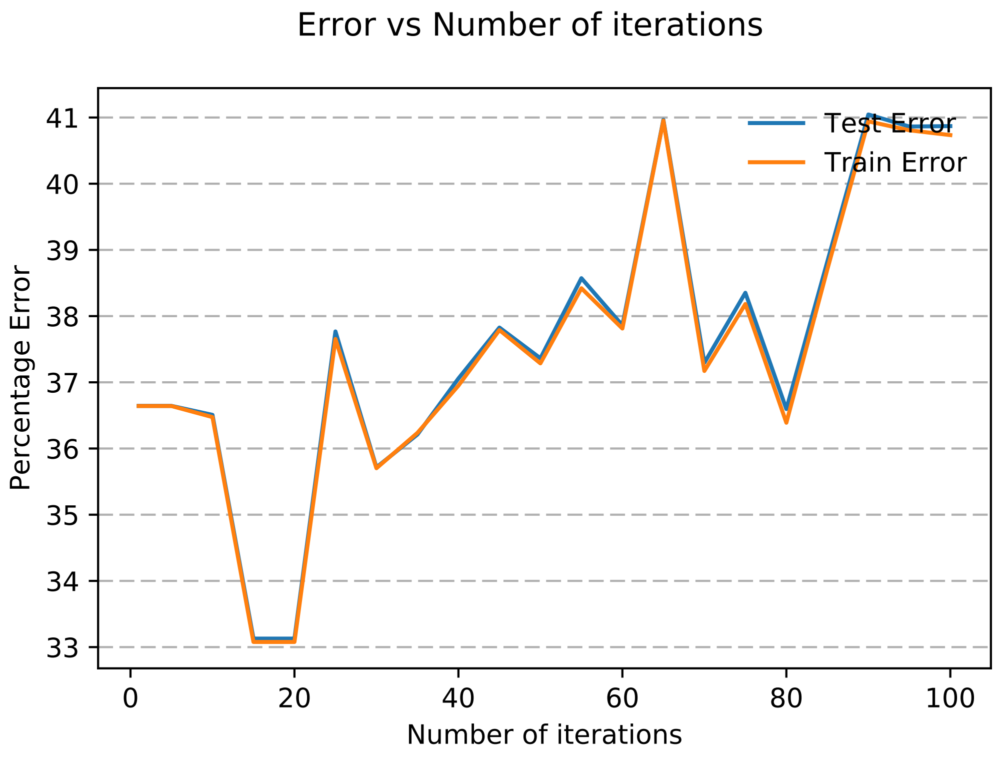
From the above plot it can be seen that the test error starts increasing after number of iterations go above 15. Hence 15 seems to be a reasonable number of iterations. Then we show the final confusion matrix with 15 iterations.

```
Confusion matrix:
[[38698 14299     9     0     0     0     0]
 [17447 52117  1212     0     0     0     0]
 [    0  2591  6318     0     0     0     0]
 [    0     0   651     0     0     0     0]
 [    0  2436    37     0     0     0     0]
 [    0  1221  3152     0     0     0     0]
 [ 5041    24     0     0     0     0     0]]
Normalized confusion matrix
[[  7.3007e-01   2.6976e-01   1.6979e-04   0.0000e+00   0.0000e+00
    0.0000e+00   0.0000e+00]
 [  2.4651e-01   7.3637e-01   1.7124e-02   0.0000e+00   0.0000e+00
    0.0000e+00   0.0000e+00]
 [  0.0000e+00   2.9083e-01   7.0917e-01   0.0000e+00   0.0000e+00
    0.0000e+00   0.0000e+00]
 [  0.0000e+00   0.0000e+00   1.0000e+00   0.0000e+00   0.0000e+00
    0.0000e+00   0.0000e+00]
 [  0.0000e+00   9.8504e-01   1.4962e-02   0.0000e+00   0.0000e+00
    0.0000e+00   0.0000e+00]
 [  0.0000e+00   2.7921e-01   7.2079e-01   0.0000e+00   0.0000e+00
    0.0000e+00   0.0000e+00]
 [  9.9526e-01   4.7384e-03   0.0000e+00   0.0000e+00   0.0000e+00
    0.0000e+00   0.0000e+00]]
```



```
('Percentage accuracy:', 66.871596455839125)
```

Now we plot both the train and test error with number of iterations.

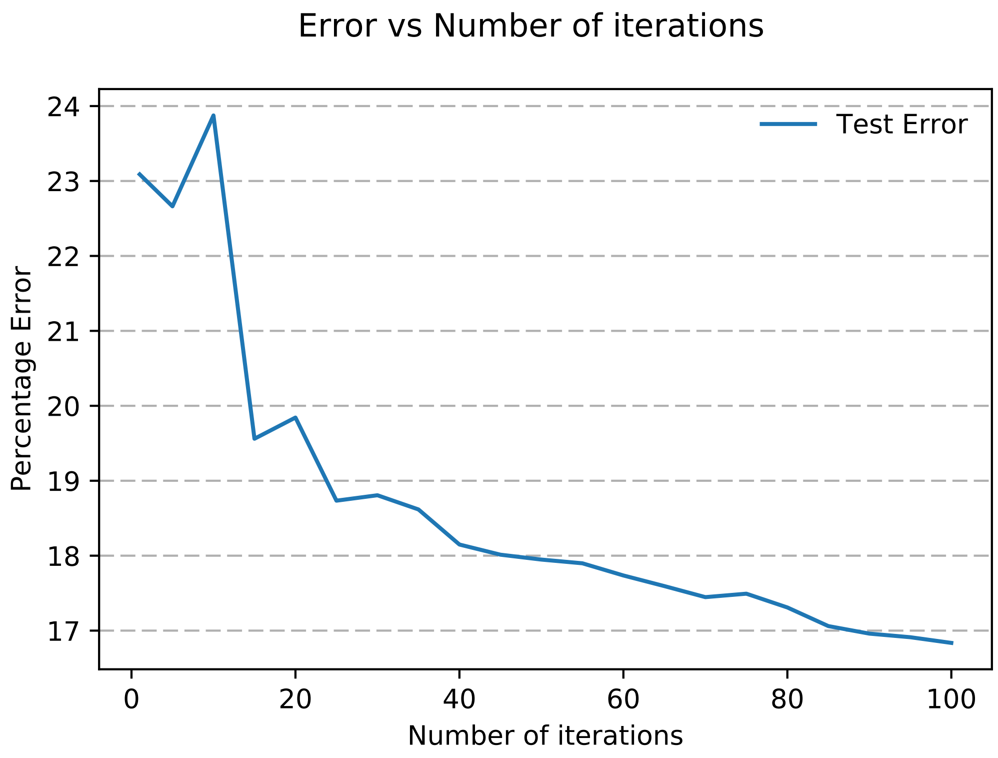


1.1.2 AdaBoost in which the learner is a decision tree with maximum depth 10

First we train a AdaBoost classifier in which each of the learner is a decision tree with maximum depth 10 upto a total of 100 iterations and then we subsample from the trained clasifier to plot test and training error with respect to the number of weak learners (iterations).

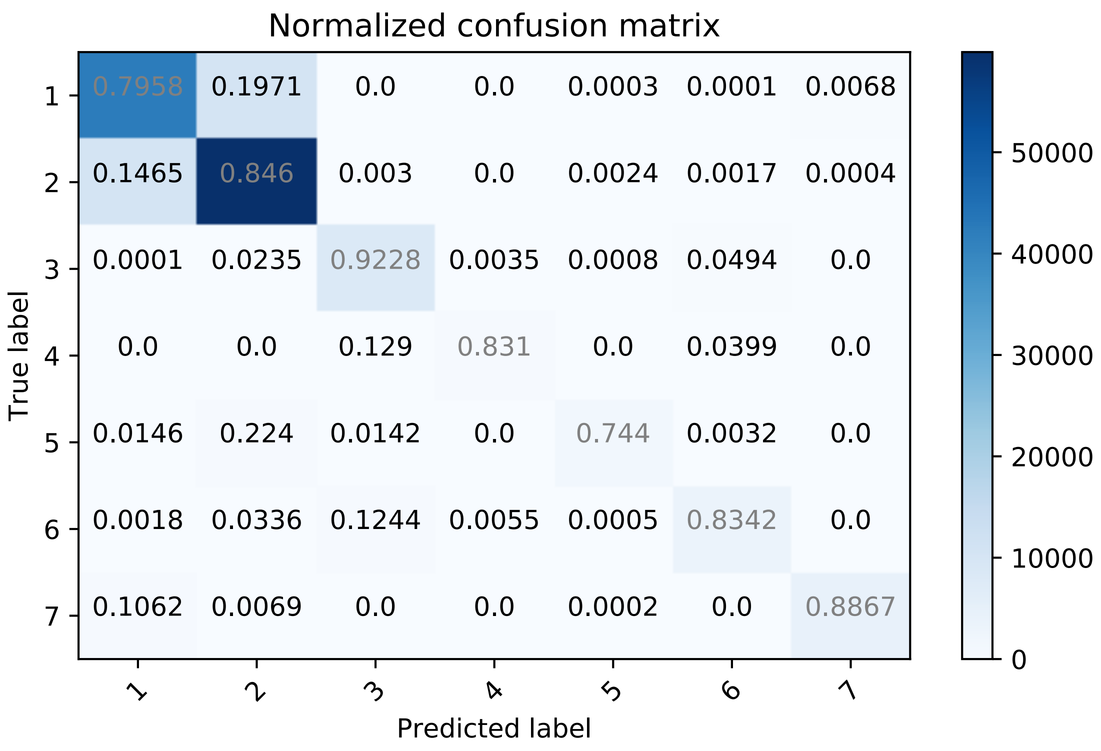
```
Out[101]: <__main__.MyAdaBoost instance at 0x10206b0e0>
```

We plot the test error with number of iterations upto 100. In order to reduce the plotting time we subsampled the plot at a value of 5. Below is the plot of error with iterations as [1,5,10,15,..,100].



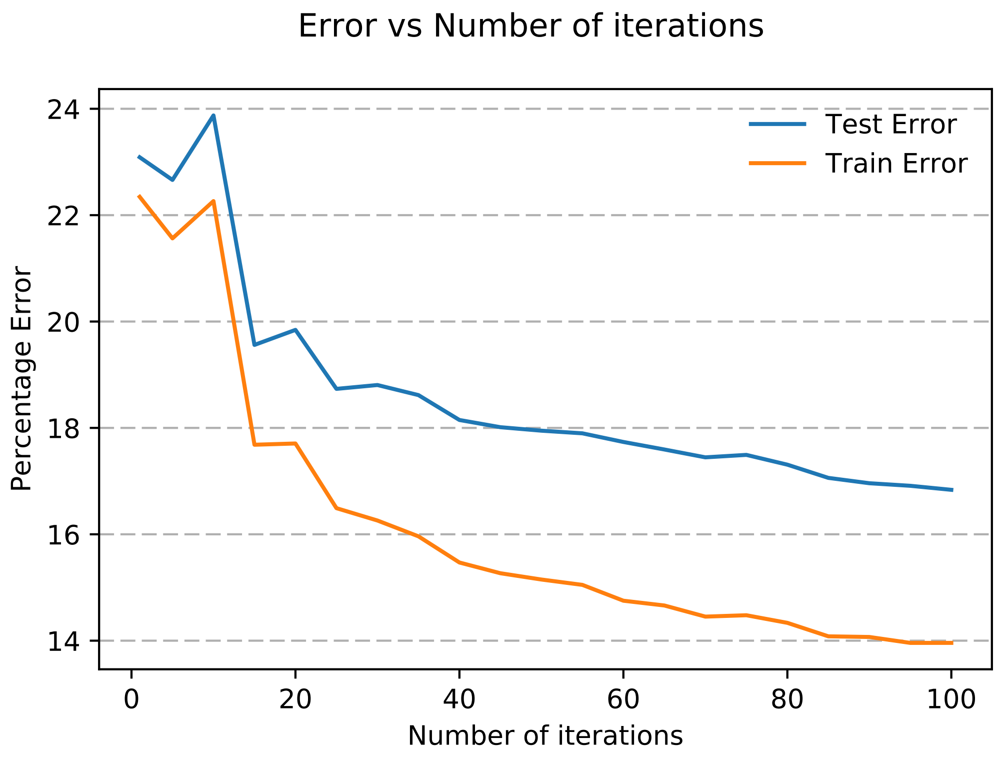
From the above plot it can be seen that the test error continues to decrease as we increase the number of iterations. No overfitting is observed but the rate of error decrease is very less as we go beyond 85 iterations. But to have the minimum error possible we choose the final number of iterations to be 100. Then we show the final confusion matrix with 100 iterations.

```
Confusion matrix:
[[42180 10447     0     0    15     5   359]
 [10370 59877   214     0   171   118    26]
 [    1   209  8221    31     7   440     0]
 [    0     0    84   541     0    26     0]
 [   36   554    35     0  1840     8     0]
 [    8   147   544    24     2  3648     0]
 [  538    35     0     0     1     0  4491]]
Normalized confusion matrix
[[  7.9576e-01   1.9709e-01   0.0000e+00   0.0000e+00   2.8299e-04
    9.4329e-05   6.7728e-03]
 [  1.4652e-01   8.4601e-01   3.0236e-03   0.0000e+00   2.4161e-03
    1.6672e-03   3.6736e-04]
 [  1.1225e-04   2.3459e-02   9.2277e-01   3.4796e-03   7.8572e-04
    4.9388e-02   0.0000e+00]
 [  0.0000e+00   0.0000e+00   1.2903e-01   8.3103e-01   0.0000e+00
    3.9939e-02   0.0000e+00]
 [  1.4557e-02   2.2402e-01   1.4153e-02   0.0000e+00   7.4404e-01
    3.2349e-03   0.0000e+00]
 [  1.8294e-03   3.3615e-02   1.2440e-01   5.4882e-03   4.5735e-04
    8.3421e-01   0.0000e+00]
 [  1.0622e-01   6.9102e-03   0.0000e+00   0.0000e+00   1.9743e-04
 0.0000e+00   8.8667e-01]]
```



```
('Percentage accuracy:', 83.163858922018818)
```

Finally we plot the train and test error with number of iterations.



1.1.3 Trends observed

- We observe that when training AdaBoost with a stump, the test error and train error are very identical and the model starts to overfit once more than 20 iterations are used. Also the error variation is not uniform as various peaks and drops are observed throughout. Also the test error (misclassification) is very high for classes 4,5,6 and 7 which is evident from the confusion matrix. This can be attributed to the fact that a stump as weak learner has very low classification power specially for a multi-class classification problem. Hence very high error rates are observed for classes having low number of examples present.
- When training AdaBoost with a decision tree of max depth 10, the accuracy increases significantly. Also no overfitting is observed upto 100 iterations and classification error reducess steeply for classes 4,5,6 and 7.

1.2 MNIST Dataset

Since the test data available on Kaggle does not contain class labels for the test set, we use train data and separate it into train set and test set with train set having 75% of the data and the test set having 25% of the data selected randomly.

We then use sklearn's VarianceThreshold to weed out dimensions with low variance. This reduces the dimension from 784 to 589.

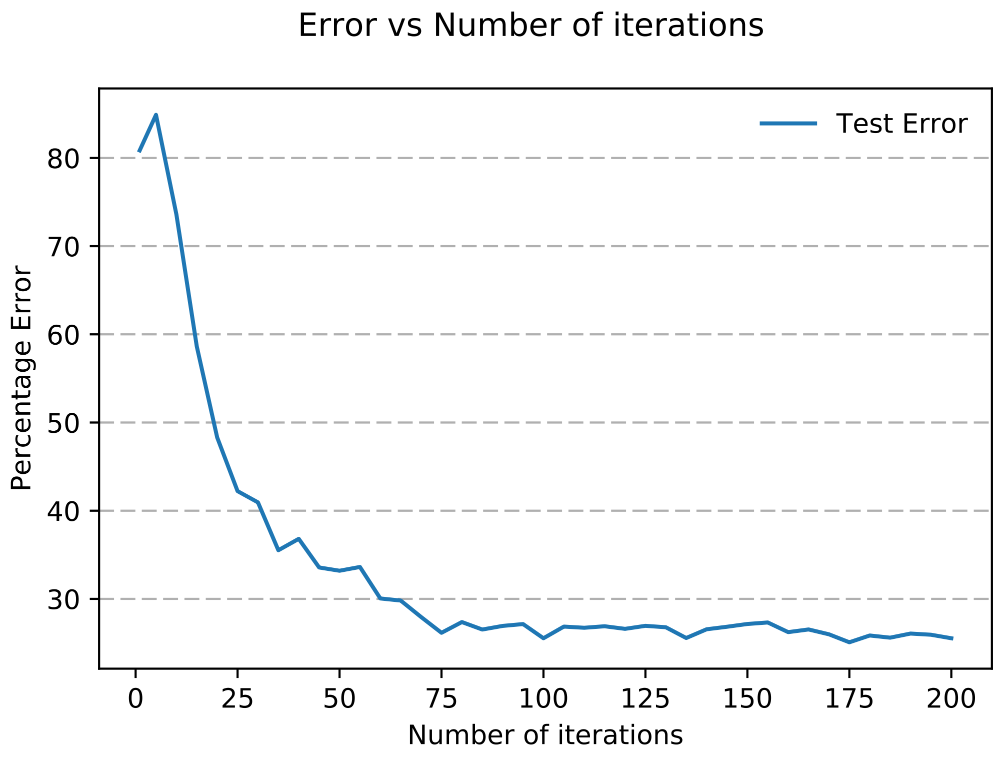
```
(31500, 589)
(10500, 589)
```

1.2.1 Adaboost in which learner is decision stump (One Level Tree)

First we train a AdaBoost classifier in which each of the learner is a decision stump (one level tree) upto a total of 200 iterations and then we subsample from the trained clasifier to plot test and training error with respect to the number of weak learners (iterations).

```
Out[135]: <__main__.MyAdaBoost instance at 0x107d57a70>
```

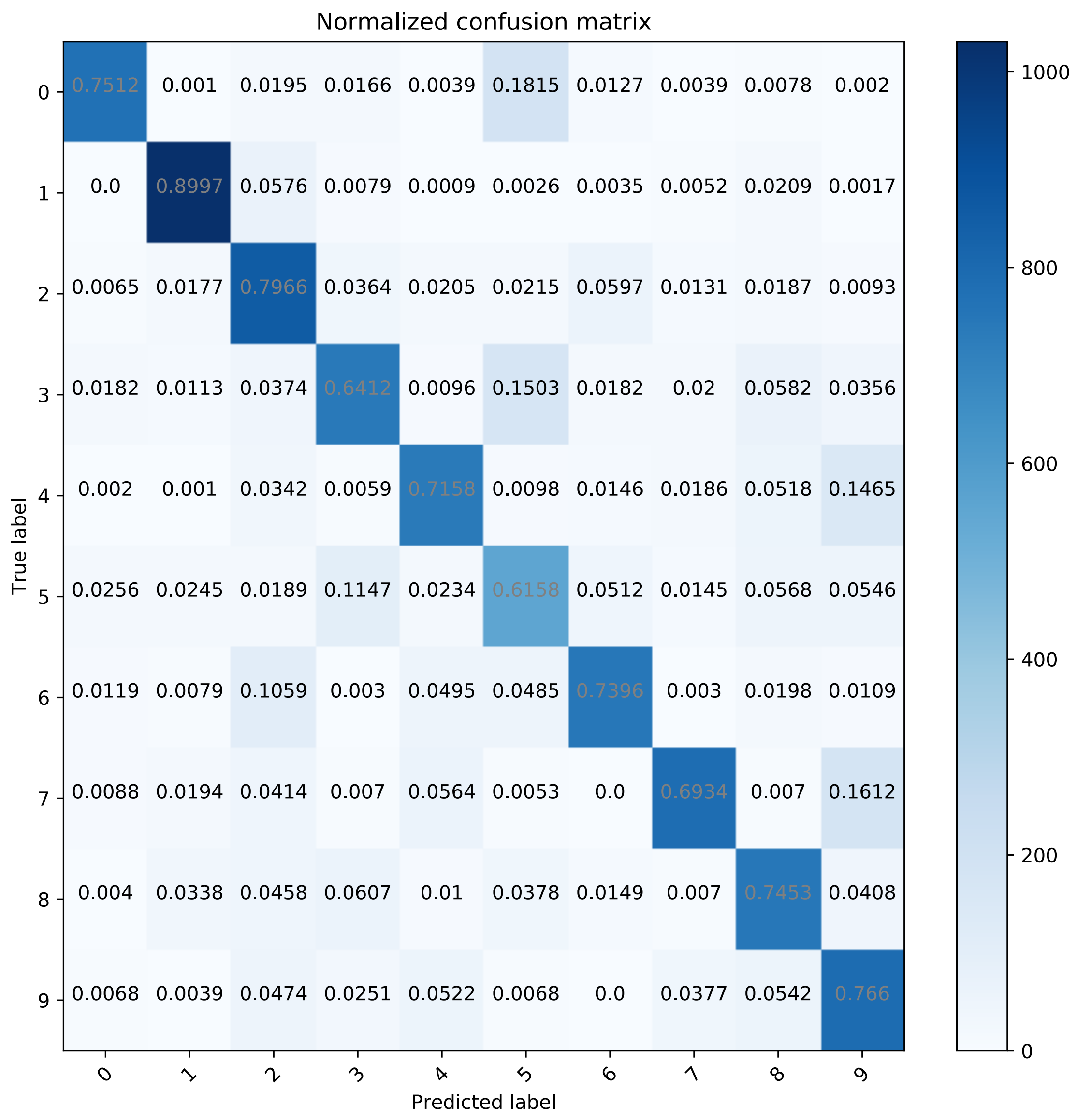
We plot the test error with number of iterations upto 200. In order to reduce the plotting time we subsampled the plot at a value of 5. Below is the plot of error with iterations as [1,5,10,15,..,100].



From the above plot it can be seen that the test error continues to decrease as we increase the iterations. But the decrease is not significant beyond 75 iterations as it reaches a plateau at 25% error. Hence 75 seems to be a reasonable number of iterations. Then we show the final confusion matrix with 75 iterations.

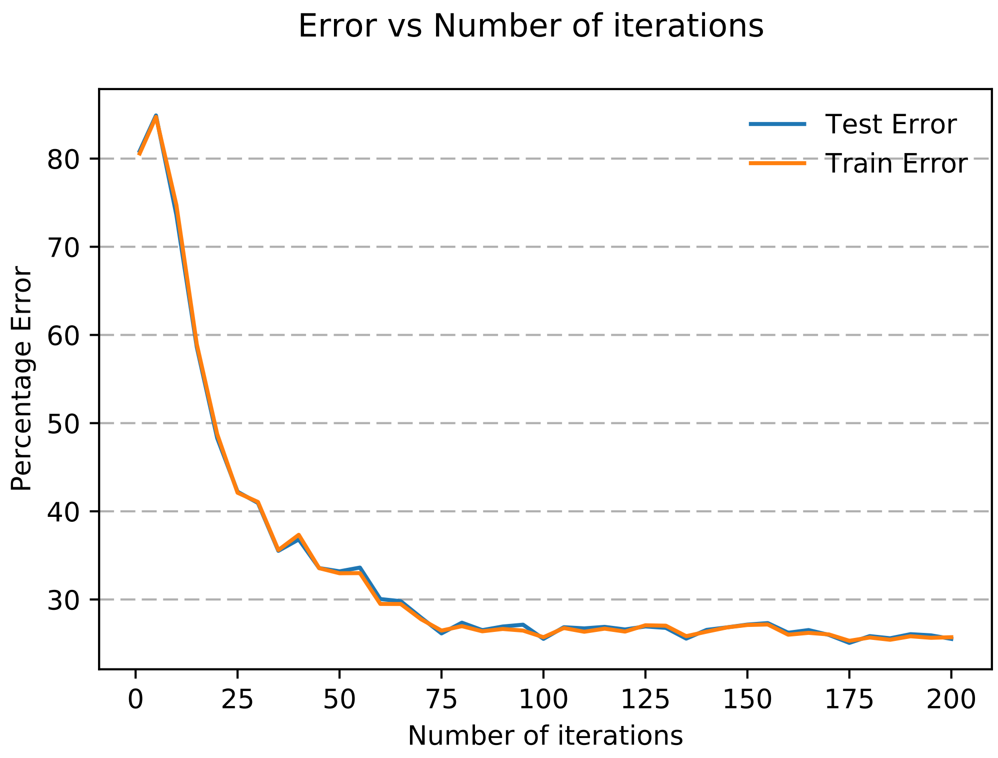
```
Confusion matrix:
[[ 770    1   20   17    4  186   13    4    8    2]
 [   0 1031   66    9    1    3    4    6   24    2]
 [   7   19  854   39   22   23   64   14   20   10]
 [  21   13   43  738   11  173   21   23   67   41]
 [   2    1   35    6  733   10   15   19   53  150]
 [  23   22   17  103   21  553   46   13   51   49]
 [  12    8  107    3   50   49  747    3   20   11]
 [  10   22   47    8   64    6    0  787    8  183]
 [   4   34   46   61   10   38   15    7  749   41]
 [   7    4   49   26   54    7    0   39   56  792]]
Normalized confusion matrix
[[  7.5122e-01   9.7561e-04   1.9512e-02   1.6585e-02   3.9024e-03
```

```
    1.8146e-01   1.2683e-02   3.9024e-03   7.8049e-03   1.9512e-03]
 [  0.0000e+00   8.9965e-01   5.7592e-02   7.8534e-03   8.7260e-04
    2.6178e-03   3.4904e-03   5.2356e-03   2.0942e-02   1.7452e-03]
 [  6.5299e-03   1.7724e-02   7.9664e-01   3.6381e-02   2.0522e-02
    2.1455e-02   5.9701e-02   1.3060e-02   1.8657e-02   9.3284e-03]
 [  1.8245e-02   1.1295e-02   3.7359e-02   6.4118e-01   9.5569e-03
    1.5030e-01   1.8245e-02   1.9983e-02   5.8210e-02   3.5621e-02]
 [  1.9531e-03   9.7656e-04   3.4180e-02   5.8594e-03   7.1582e-01
    9.7656e-03   1.4648e-02   1.8555e-02   5.1758e-02   1.4648e-01]
 [  2.5612e-02   2.4499e-02   1.8931e-02   1.1470e-01   2.3385e-02
    6.1581e-01   5.1225e-02   1.4477e-02   5.6793e-02   5.4566e-02]
 [  1.1881e-02   7.9208e-03   1.0594e-01   2.9703e-03   4.9505e-02
    4.8515e-02   7.3960e-01   2.9703e-03   1.9802e-02   1.0891e-02]
 [  8.8106e-03   1.9383e-02   4.1410e-02   7.0485e-03   5.6388e-02
    5.2863e-03   0.0000e+00   6.9339e-01   7.0485e-03   1.6123e-01]
 [  3.9801e-03   3.3831e-02   4.5771e-02   6.0697e-02   9.9502e-03
    3.7811e-02   1.4925e-02   6.9652e-03   7.4527e-01   4.0796e-02]
 [  6.7698e-03   3.8685e-03   4.7389e-02   2.5145e-02   5.2224e-02
    6.7698e-03   0.0000e+00   3.7718e-02   5.4159e-02   7.6596e-01]]
```



```
('Percentage accuracy:', 73.847619047619048)
```

Now we plot both the train and test error with number of iterations.

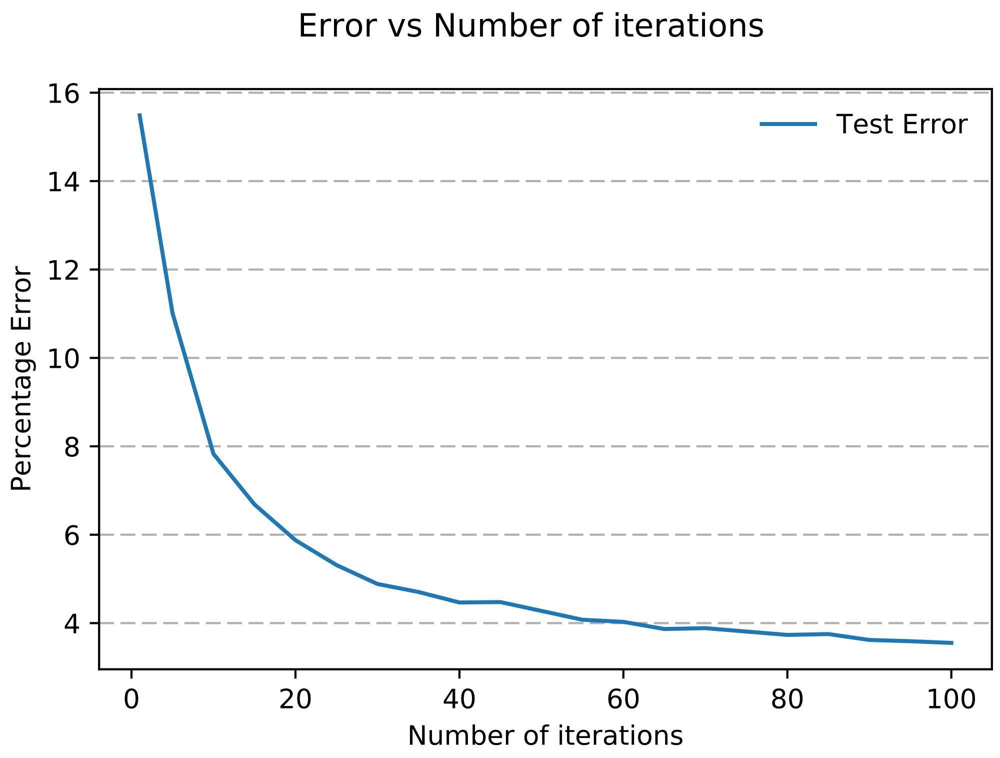


1.2.2 AdaBoost in which the learner is a decision tree with maximum depth 10

First we train a AdaBoost classifier in which each of the learner is a decision tree with maximum depth 10 upto a total of 100 iterations and then we subsample from the trained clasifier to plot test and training error with respect to the number of weak learners (iterations).

```
Out[145]: <__main__.MyAdaBoost instance at 0x195582518>
```

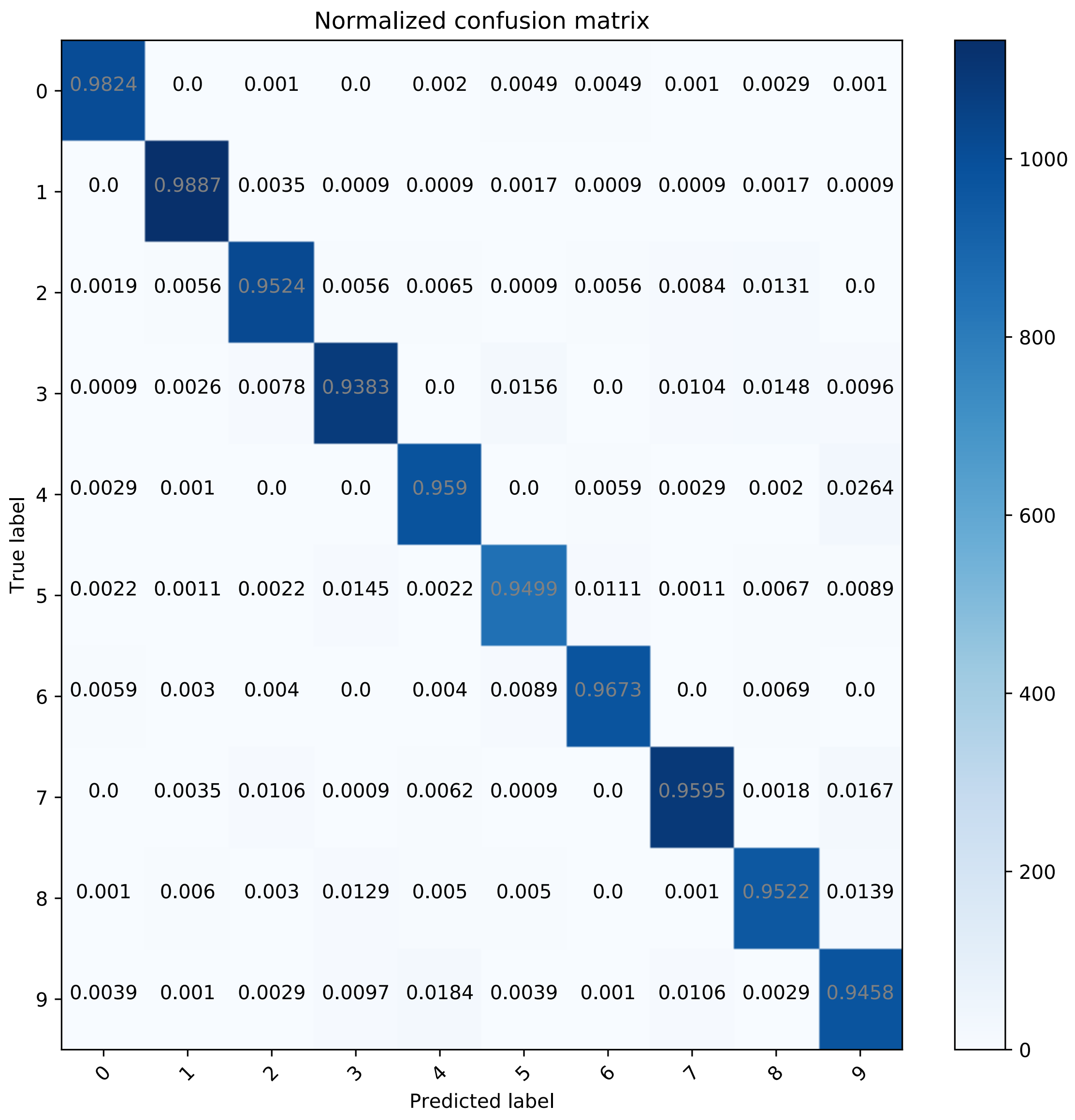
We plot the test error with number of iterations upto 100. In order to reduce the plotting time we subsampled the plot at a value of 5. Below is the plot of error with iterations as [1,5,10,15,..,100].



From the above plot it can be seen that the test error continues to decrease as we increase the iterations. But the decrease is not significant beyond 60 iterations as it reaches a plateau at 4% error. Hence 60 seems to be a reasonable number of iterations. Then we show the final confusion matrix with 60 iterations.

```
Confusion matrix:
[[1007    0    1    0    2    5    5    1    3    1]
 [   0 1133    4    1    1    2    1    1    2    1]
 [   2    6 1021    6    7    1    6    9   14    0]
 [   1    3    9 1080    0   18    0   12   17   11]
 [   3    1    0    0  982    0    6    3    2   27]
 [   2    1    2   13    2  853   10    1    6    8]
 [   6    3    4    0    4    9  977    0    7    0]
 [   0    4   12    1    7    1    0 1089    2   19]
 [   1    6    3   13    5    5    0    1  957   14]
 [   4    1    3   10   19    4    1   11    3  978]]
Normalized confusion matrix
[[  9.8244e-01   0.0000e+00   9.7561e-04   0.0000e+00   1.9512e-03
    4.8780e-03   4.8780e-03   9.7561e-04   2.9268e-03   9.7561e-04]
 [  0.0000e+00   9.8866e-01   3.4904e-03   8.7260e-04   8.7260e-04
    1.7452e-03   8.7260e-04   8.7260e-04   1.7452e-03   8.7260e-04]
 [  1.8657e-03   5.5970e-03   9.5243e-01   5.5970e-03   6.5299e-03
    9.3284e-04   5.5970e-03   8.3955e-03   1.3060e-02   0.0000e+00]
 [  8.6881e-04   2.6064e-03   7.8193e-03   9.3831e-01   0.0000e+00
    1.5639e-02   0.0000e+00   1.0426e-02   1.4770e-02   9.5569e-03]
 [  2.9297e-03   9.7656e-04   0.0000e+00   0.0000e+00   9.5898e-01
    0.0000e+00   5.8594e-03   2.9297e-03   1.9531e-03   2.6367e-02]
 [  2.2272e-03   1.1136e-03   2.2272e-03   1.4477e-02   2.2272e-03
```

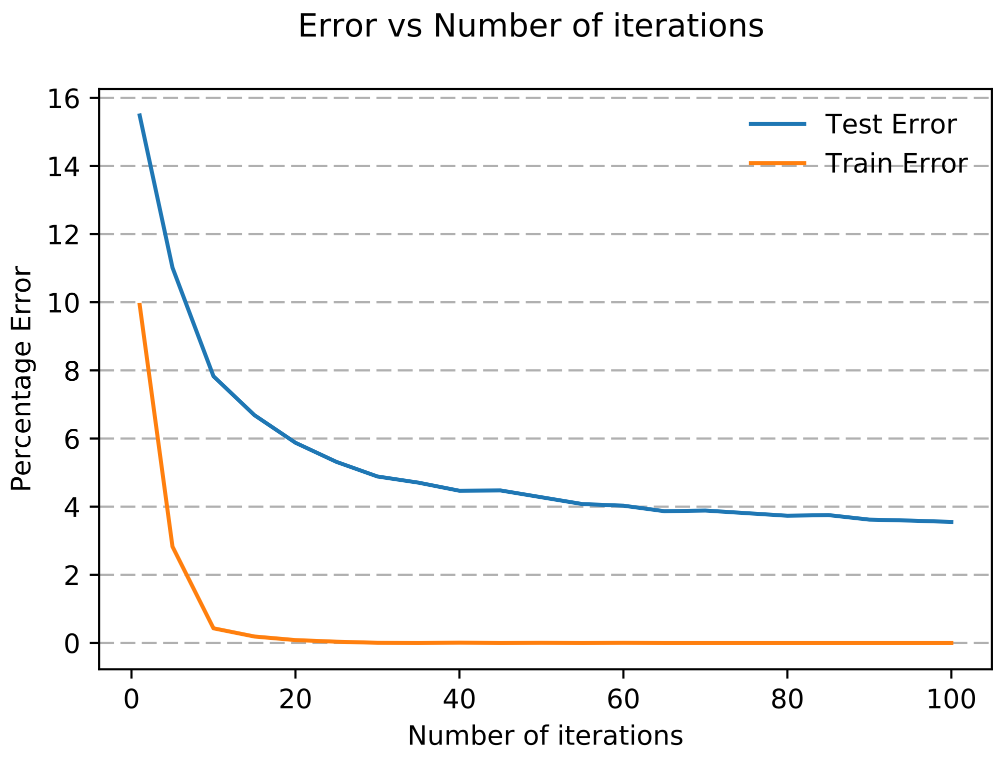
```
   9.4989e-01   1.1136e-02   1.1136e-03   6.6815e-03   8.9087e-03]
[  5.9406e-03   2.9703e-03   3.9604e-03   0.0000e+00   3.9604e-03
   8.9109e-03   9.6733e-01   0.0000e+00   6.9307e-03   0.0000e+00]
[  0.0000e+00   3.5242e-03   1.0573e-02   8.8106e-04   6.1674e-03
   8.8106e-04   0.0000e+00   9.5947e-01   1.7621e-03   1.6740e-02]
[  9.9502e-04   5.9701e-03   2.9851e-03   1.2935e-02   4.9751e-03
   4.9751e-03   0.0000e+00   9.9502e-04   9.5224e-01   1.3930e-02]
[  3.8685e-03   9.6712e-04   2.9014e-03   9.6712e-03   1.8375e-02
   3.8685e-03   9.6712e-04   1.0638e-02   2.9014e-03   9.4584e-01]]
```



True label	0	1	2	3	4	5	6	7	8	9
0	0.9824	0.0	0.001	0.0	0.002	0.0049	0.0049	0.001	0.0029	0.001
1	0.0	0.9887	0.0035	0.0009	0.0009	0.0017	0.0009	0.0009	0.0017	0.0009
2	0.0019	0.0056	0.9524	0.0056	0.0065	0.0009	0.0056	0.0084	0.0131	0.0
3	0.0009	0.0026	0.0078	0.9383	0.0	0.0156	0.0	0.0104	0.0148	0.0096
4	0.0029	0.001	0.0	0.0	0.959	0.0	0.0059	0.0029	0.002	0.0264
5	0.0022	0.0011	0.0022	0.0145	0.0022	0.9499	0.0111	0.0011	0.0067	0.0089
6	0.0059	0.003	0.004	0.0	0.004	0.0089	0.9673	0.0	0.0069	0.0
7	0.0	0.0035	0.0106	0.0009	0.0062	0.0009	0.0	0.9595	0.0018	0.0167
8	0.001	0.006	0.003	0.0129	0.005	0.005	0.0	0.001	0.9522	0.0139
9	0.0039	0.001	0.0029	0.0097	0.0184	0.0039	0.001	0.0106	0.0029	0.9458

```
('Percentage accuracy:', 95.971428571428575)
```

Now we plot both the train and test error with number of iterations.



1.2.3 Trends observed

- We observe that when training AdaBoost with a stump, the test error and train error are very identical and the model shows no signs of overfitting as we increase iterations upto 200. But beyond 75 iterations test error reaches a plateau at 25% error. Also the test error (misclassification) is a bit high for classes 3,5 and 7 which is evident from the confusion matrix. This can be attributed to the fact that a stump as weak learner has very low classification power specially for a multi-class classification problem. Hence very high error rates are observed for classes which have several indistinguishable pixels with other classes.
- When training AdaBoost with a decision tree of max depth 10, the accuracy increases significantly as we increase iterations. Also no overfitting is observed upto 100 iterations. But beyond 60 iterations test error reaches a plateau at 4% error. The classification error reducess steeply for classes 3,5 and 7 as a weak learner of maximum depth 10 has a much better classification power then a stump.