1. **Softmax regression and OVA logistic regression**
   1. Before starting the nested for loops, we prepared a *mxK* matrix of all the exponential terms. We normalized the elements before exponentiation as mentioned in the problem statement. Due to randomly initialized theta matrix, repeated runs of the code snippet resulted in varying values of **J** but they were all close to *–log(0.1)* with a worst outlier around 2.37. Ideally, with uniformly distributed samples across the 10 classes, the average probability should work out close to *log(0.1)*.  
      Below is the generated visualization:  
        
      
   2. Below is the python console output (we use pycharm IDE here) after completion of the analytical derivative computation in softmax\_loss\_naive(). This output is for the run that gave a loss of 2.30055406106 which is quite close to *–log(0.1)*.  
      As can be seen, the relative errors are of the order of 10-8 or less:

/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/razor/Documents/COMP\_540/hw3/softmax\_hw.py

Training data shape: (49000, 3072)

Validation data shape: (1000, 3072)

Test data shape: (10000, 3072)

Training data shape with bias term: (49000, 3073)

Validation data shape with bias term: (1000, 3073)

Test data shape with bias term: (10000, 3073)

loss: 2.30055406106 should be close to 2.30258509299

numerical: 0.861715 analytic: 0.861715, relative error: 1.886018e-08

numerical: 2.326214 analytic: 2.326214, relative error: 8.786167e-09

numerical: -1.868729 analytic: -1.868729, relative error: 1.364460e-08

numerical: 2.487585 analytic: 2.487585, relative error: 2.179826e-08

numerical: -0.873042 analytic: -0.873042, relative error: 2.646641e-08

numerical: -0.258260 analytic: -0.258260, relative error: 2.378065e-08

numerical: 2.934628 analytic: 2.934628, relative error: 1.676520e-08

numerical: -0.703468 analytic: -0.703468, relative error: 8.196305e-08

numerical: -1.060693 analytic: -1.060693, relative error: 2.630864e-08

numerical: 1.334815 analytic: 1.334815, relative error: 1.456137e-08

naive loss: 2.300554e+00 computed in 38.916627s

vectorized loss: 0.000000e+00 computed in 0.000075s

Loss difference: 2.300554

Gradient difference: 270.878128

best validation accuracy achieved during cross-validation: -1.000000

Process finished with exit code 0

* 1. Vectorized implementation for loss function gave very similar results for loss as evident from the results presented in section 3.4. It is clear that the vector approach is faster than the naïve one by an order of magnitude. Runtime goes down from 35.546s down to 1.771s (highlighted in red):
  2. Vectorized implementation for gradient of loss function gave matching results with the naïve implementation as per below result(highlighted in green):

/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/razor/Documents/COMP\_540/hw3/softmax\_hw.py

Training data shape: (49000, 3072)

Validation data shape: (1000, 3072)

Test data shape: (10000, 3072)

Training data shape with bias term: (49000, 3073)

Validation data shape with bias term: (1000, 3073)

Test data shape with bias term: (10000, 3073)

loss: 2.31149097643 should be close to 2.30258509299

numerical: -3.414947 analytic: -3.414948, relative error: 2.261616e-08

numerical: -0.211616 analytic: -0.211616, relative error: 3.971626e-08

numerical: -2.252097 analytic: -2.252097, relative error: 1.236714e-08

numerical: -1.581972 analytic: -1.581972, relative error: 2.203170e-08

numerical: 0.319598 analytic: 0.319598, relative error: 1.575224e-07

numerical: -0.265488 analytic: -0.265488, relative error: 2.269396e-08

numerical: 0.999693 analytic: 0.999693, relative error: 4.741713e-08

numerical: 1.337352 analytic: 1.337352, relative error: 4.301138e-08

numerical: -1.672967 analytic: -1.672967, relative error: 2.452645e-08

numerical: -0.240183 analytic: -0.240184, relative error: 3.023726e-07

naive loss: 2.311491e+00 computed in 35.545789s

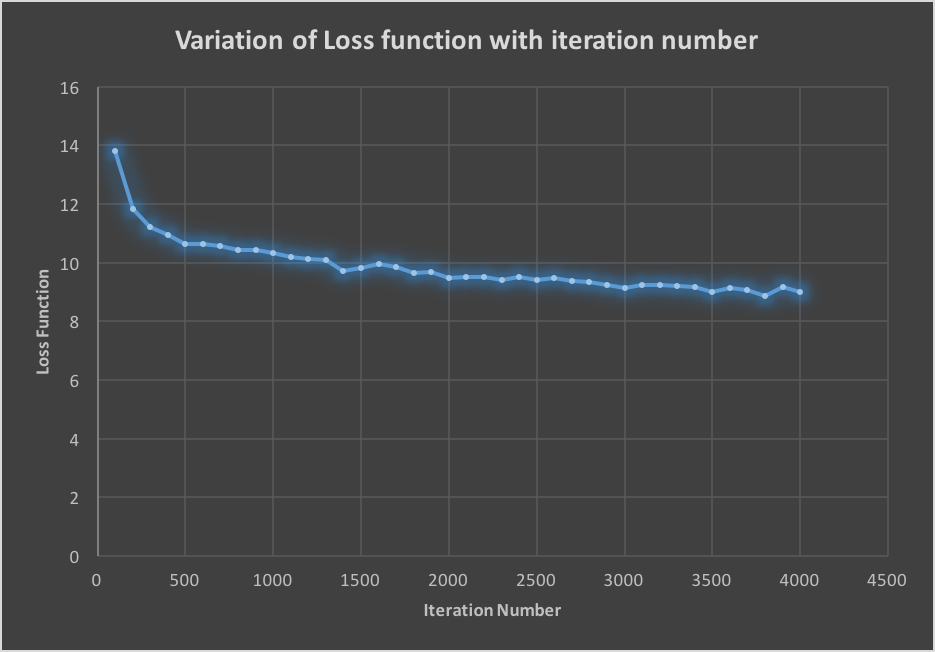
vectorized loss: 2.311491e+00 computed in 1.771249s

Loss difference: 0.000000

Gradient difference: 0.000000

best validation accuracy achieved during cross-validation: -1.000000

Process finished with exit code 0

* 1. We implemented the mini batch gradient descent algorithm and observed the variation of the loss function with batch sizes of 400 and with 4000 iterations.  
     The batched were selected at random with replacement set to true. Below is a plot showing the variation of loss function with the iteration number. We observe a steeper descent for the loss function in the initial 500 iterations. The curve then decreases less sharply thereafter. Note that loss function values do not decrease smoothly due to the randomness in batch selection -  
       
     ****
  2. Cross validation implemented with an overall accuracy of 40.31%. We observed 2 sets of hyper parameters with a maximum accuracy of 41.7% on the validation set. Our implementation selects the first such set encountered as the best\_softmax classifier which gave the accuracy of 40.31%. Output below:

/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/razor/Documents/COMP\_540/hw3/softmax\_hw.py

Training data shape: (49000, 3072)

Validation data shape: (1000, 3072)

Test data shape: (10000, 3072)

Training data shape with bias term: (49000, 3073)

Validation data shape with bias term: (1000, 3073)

Test data shape with bias term: (10000, 3073)

lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.292796 val accuracy: 0.287000

lr 1.000000e-07 reg 1.000000e+05 train accuracy: 0.292571 val accuracy: 0.284000

lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.321122 val accuracy: 0.313000

lr 1.000000e-07 reg 1.000000e+08 train accuracy: 0.277776 val accuracy: 0.298000

lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.377837 val accuracy: 0.369000

lr 5.000000e-07 reg 1.000000e+05 train accuracy: 0.390265 val accuracy: 0.376000

lr 5.000000e-07 reg 5.000000e+05 train accuracy: 0.413980 val accuracy: 0.417000

lr 5.000000e-07 reg 1.000000e+08 train accuracy: 0.284388 val accuracy: 0.306000

lr 1.000000e-06 reg 5.000000e+04 train accuracy: 0.413571 val accuracy: 0.392000

lr 1.000000e-06 reg 1.000000e+05 train accuracy: 0.423327 val accuracy: 0.404000

lr 1.000000e-06 reg 5.000000e+05 train accuracy: 0.413429 val accuracy: 0.417000

lr 1.000000e-06 reg 1.000000e+08 train accuracy: 0.265694 val accuracy: 0.272000

lr 5.000000e-06 reg 5.000000e+04 train accuracy: 0.421980 val accuracy: 0.388000

lr 5.000000e-06 reg 1.000000e+05 train accuracy: 0.400224 val accuracy: 0.380000

lr 5.000000e-06 reg 5.000000e+05 train accuracy: 0.354653 val accuracy: 0.350000

lr 5.000000e-06 reg 1.000000e+08 train accuracy: 0.133633 val accuracy: 0.134000

best validation accuracy achieved during cross-validation: 0.417000

softmax on raw pixels final test set accuracy: 0.403100

Process finished with exit code 0

* 1. *Training with the best classifier* – in the previous question, we observed that a best Softmax was chosen from the 16 (alpha,lambda) combinations and used to classify raw images which resulted in an accuracy of 40.31%.  
     However, we reran the code since we got 2 such hyper parameter pairs that tied with a validation accuracy of 41.7% and our code then selected the first pair.  
     On second run we obtained a unique best (alpha,lambda) pair that gave a validation accuracy of 41.8%. However, it resulted in a slightly lower accuracy of 40.20% on the raw image set than the previous run. Output below:  
       
       
      **Theta Visualization:** Below we present the generated theta visualization and the corresponding confusion matrix for a cross-validation run that gave an accuracy of 39.95% on raw images -   
     **** Figure 2. Theta Visualization generated **Confusion Matrix:**

Harshs-MacBook-Pro:hw3 razor$ python softmax\_hw.py

/Users/razor/Library/Enthought/Canopy\_64bit/User/lib/python2.7/site-packages/matplotlib/font\_manager.py:273: UserWarning: Matplotlib is building the font cache using fc-list. This may take a moment.

warnings.warn('Matplotlib is building the font cache using fc-list. This may take a moment.')

Training data shape: (49000, 3072)

Validation data shape: (1000, 3072)

Test data shape: (10000, 3072)

Training data shape with bias term: (49000, 3073)

Validation data shape with bias term: (1000, 3073)

Test data shape with bias term: (10000, 3073)

lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.286551 val accuracy: 0.290000

lr 1.000000e-07 reg 1.000000e+05 train accuracy: 0.294429 val accuracy: 0.290000

lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.326980 val accuracy: 0.331000

lr 1.000000e-07 reg 1.000000e+08 train accuracy: 0.279857 val accuracy: 0.289000

lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.377020 val accuracy: 0.355000

lr 5.000000e-07 reg 1.000000e+05 train accuracy: 0.391449 val accuracy: 0.377000

lr 5.000000e-07 reg 5.000000e+05 train accuracy: 0.410041 val accuracy: 0.408000

lr 5.000000e-07 reg 1.000000e+08 train accuracy: 0.275204 val accuracy: 0.293000

lr 1.000000e-06 reg 5.000000e+04 train accuracy: 0.412061 val accuracy: 0.383000

lr 1.000000e-06 reg 1.000000e+05 train accuracy: 0.422918 val accuracy: 0.418000

lr 1.000000e-06 reg 5.000000e+05 train accuracy: 0.411020 val accuracy: 0.406000

lr 1.000000e-06 reg 1.000000e+08 train accuracy: 0.269000 val accuracy: 0.296000

lr 5.000000e-06 reg 5.000000e+04 train accuracy: 0.419347 val accuracy: 0.407000

lr 5.000000e-06 reg 1.000000e+05 train accuracy: 0.405673 val accuracy: 0.381000

lr 5.000000e-06 reg 5.000000e+05 train accuracy: 0.377571 val accuracy: 0.362000

lr 5.000000e-06 reg 1.000000e+08 train accuracy: 0.080490 val accuracy: 0.081000

best validation accuracy achieved during cross-validation: 0.418000

softmax on raw pixels final test set accuracy: 0.402000

Confusion matrix for image classification

[[477 50 43 30 14 17 24 40 223 82]

[ 63 487 18 36 21 39 36 37 97 166]

[114 53 224 90 129 85 146 59 69 31]

[ 53 82 75 251 66 177 108 55 64 69]

[ 62 32 120 68 322 77 146 104 37 32]

[ 46 52 82 156 79 324 90 67 79 25]

[ 23 55 58 110 111 81 470 26 23 43]

[ 58 52 59 58 97 66 58 396 54 102]

[143 77 7 26 6 39 8 15 559 120]

[ 68 174 13 22 13 20 45 48 112 485]]

Harshs-MacBook-Pro:hw3 razor$

* 1. Comparison between OVA and Softmax Classifiers:  
     We loaded the CEPs representation of the music files using music\_utils.py provided in homework 2. We then split the dataset into training, validation and test sets in the ratio specified in the problem statement using the train\_test\_split function in sklearn package. We reused the OVA classifier implemented in homework 2. We copied the files one\_vs\_all.py, music\_utils.py from homework 2 into our hw3 folder so that we can import them into softmax\_music.py and directly use the functions instead of writing new code from scratch. We also had to add the sigmoid function to utils.py for our OVA prediction function.  
       
     We then ran cross-validation on the softmax and the OVA classifiers separately to obtain their respective optimal hyperparameters on common training and validation datasets. These optimal classifiers were then tested against the common test dataset and their confusion matrices were printed out.  
       
     We ran this code multiple times and observed that OVA outperformed softmax most of the times. A table comparing precisions of softmax and OVA for one such run is presented in the next page.   
       
     For the music genere classification problem, we feel that OVA is the better choice since we observed generally higher accuracies than softmax over multiple runs. Finding the best OVA classifier via cross-validation required less computation as well since we only iterated over the regularization parameter and each training involved only 500 iterations. In contrast we had to go through all 16 combinations of learning rate and regularization parameters and each iteration involved training the classifier with 4000 iterations. Given that the dataset for the music classification problem is quite small, it makes more sense to use OVA. Softmax would not benefit much here even with the extra computation and random batch sampling.  
      **Precision comparison between OVA and Softmax:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Classes** | **Precision** | | **Recall** | | **F1-score** | | **support** | |
|  | **softmax** | **OVA** | **softmax** | **OVA** | **softmax** | **OVA** | **softmax** | **OVA** |
| **Blues** | 0.35 | 0.47 | 0.50 | 0.36 | 0.42 | 0.41 | 22 | 22 |
| **Classical** | 0.35 | 0.33 | 0.94 | 0.94 | 0.52 | 0.49 | 18 | 18 |
| **Country** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 19 | 19 |
| **Disco** | 0.30 | 0.43 | 0.21 | 0.43 | 0.25 | 0.43 | 14 | 14 |
| **Hiphop** | 0.00 | 0.36 | 0.00 | 0.20 | 0.00 | 0.26 | 20 | 20 |
| **Jazz** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 24 | 24 |
| **Metal** | 0.23 | 0.45 | 1.00 | 1.00 | 0.38 | 0.62 | 21 | 21 |
| **Pop** | 0.62 | 0.34 | 0.72 | 1.00 | 0.67 | 0.51 | 18 | 18 |
| **Reggae** | 0.00 | 0.43 | 0.00 | 0.15 | 0.00 | 0.22 | 20 | 20 |
| **Rock** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 24 | 24 |
| **avg / total** | 0.17 | 0.27 | 0.33 | 0.39 | 0.21 | 0.28 | 200 | 200 |