CS 747DL Assignment-1

February 12, 2020

[1]:
```
import random
import numpy as np
from data_process import get_CIFAR10_data
import math
from scipy.spatial import distance
from models import KNN, Perceptron, SVM, Softmax
from kaggle_submission import output_submission_csv
%matplotlib inline
```

1 Loading CIFAR-10

In the following cells we determine the number of images for each split and load the images.

[2]:
```
# You can change these numbers for experimentation
# For submission we will use the default values
TRAIN_IMAGES = 49000
VAL_IMAGES = 1000
TEST_IMAGES = 5000
```

[3]:
```
data = get_CIFAR10_data(TRAIN_IMAGES, VAL_IMAGES, TEST_IMAGES)
X_train, y_train = data['X_train'], data['y_train']
X_val, y_val = data['X_val'], data['y_val']
X_test, y_test = data['X_test'], data['y_test']
```

Convert the sets of images from dimensions of **(N, 3, 32, 32) -> (N, 3072)** where N is the number of images so that each **3x32x32** image is represented by a single vector.

[4]:
```
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_val = np.reshape(X_val, (X_val.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))
```

1.0.1 Get Accuracy

This function computes how well your model performs using accuracy as a metric.

[5]:
```
def get_acc(pred, y_test):
    return np.sum(y_test==pred)/len(y_test)*100
```

2 K-Nearest Neighbors

The kNN classifier consists of two stages:

- During training, the classifier takes the training data and simply remembers it
- During testing, kNN classifies every test image by comparing to all training images and selecting the class that is most common among the k most similar training examples

In this exercise you will implement these steps using writing efficient, vectorized code. Your final implementation should not use for loops to loop over each of the test and train examples. Instead, you should calculate distances between vectorized forms of the datasets. You may refer to the `scipy.spatial.distance.cdist` function to do this efficiently.

The following code : - Creates an instance of the KNN classifier class with k = 5 - The train function of the KNN class is trained on the training data - We use the predict function for predicting testing data labels

2.0.1 Training KNN

[6]:
```
knn = KNN(5)
knn.train(X_train, y_train)
```

2.0.2 Find best k on validation

The value of k is an important hyperparameter for the KNN classifier. We will choose the best k by examining the performance of classifiers trained with different k values on the validation set.

It's not necessary to try many different values of k for the purposes of this exercise. You may increase k by a magnitude of 2 each iteration up to around k=100 or something similar to get a sense of classifier performance for different k values.

Modify the code below to loop though different values of k, train a KNN classifier for each k, and output the validation accuracy for each of the classifiers. Be sure to note your best k below as well.

[7]:
```
# TO DO : Experiment with different values of k
# k = 5
# knn = KNN(k)
# knn.train(X_train, y_train)

# pred_knn = knn.predict(X_val)
# print('The validation accuracy is given by : %f' % (get_acc(pred_knn, y_val)))
```

[8]:
```
# TO DO : Experiment with different values of k
k = 5
best_acc = -12345
best_k = k

for i in range(1, 20):
    k = i*5
    knn = KNN(k)
    knn.train(X_train, y_train)
    pred_knn = knn.predict(X_val)
    acc = get_acc(pred_knn, y_val)
    if acc>best_acc:
        best_acc = acc
        best_k = k
    print('The validation accuracy is given by : %f where k-value is : %d' %␣
 ↪(acc, k))

print('Best accuracy: %f, best k-value: %d' % (best_acc, best_k))
```

```
The validation accuracy is given by : 40.100000 where k-value is : 5
The validation accuracy is given by : 38.600000 where k-value is : 10
The validation accuracy is given by : 40.400000 where k-value is : 15
The validation accuracy is given by : 42.200000 where k-value is : 20
The validation accuracy is given by : 42.400000 where k-value is : 25
The validation accuracy is given by : 42.300000 where k-value is : 30
The validation accuracy is given by : 42.100000 where k-value is : 35
The validation accuracy is given by : 41.500000 where k-value is : 40
The validation accuracy is given by : 40.300000 where k-value is : 45
The validation accuracy is given by : 39.800000 where k-value is : 50
The validation accuracy is given by : 40.100000 where k-value is : 55
The validation accuracy is given by : 40.200000 where k-value is : 60
The validation accuracy is given by : 40.100000 where k-value is : 65
The validation accuracy is given by : 39.600000 where k-value is : 70
The validation accuracy is given by : 40.100000 where k-value is : 75
The validation accuracy is given by : 40.300000 where k-value is : 80
The validation accuracy is given by : 39.900000 where k-value is : 85
The validation accuracy is given by : 39.600000 where k-value is : 90
The validation accuracy is given by : 38.200000 where k-value is : 95
Best accuracy: 42.400000, best k-value: 25
```

2.0.3 Testing KNN

Finally, once you have found the best k according to your experiments on the validation set, retrain a classifier with the best k and test your classifier on the test set.

[9]:
```
# best_k = 25
knn = KNN(best_k)
knn.train(X_train, y_train)
```

[10]:
```
pred_knn = knn.predict(X_test)
print('The testing accuracy is given by : %f' % (get_acc(pred_knn, y_test)))
```

```
The testing accuracy is given by : 43.160000
```

2.0.4 KNN Kaggle Submission

Once you are satisfied with your solution and test accuracy output a file to submit your test set predictions to the Kaggle for Assignment 1 KNN. Use the following code to do so:

[11]:
```
output_submission_csv('knn_submission.csv', knn.predict(X_test))
```

3 Perceptron

Perceptron has 2 hyperparameters that you can experiment with: - **Learning rate** - controls how much we change the current weights of the classifier during each update. We set it at a default value of 0.5, but you should experiment with different values. We recommend changing the learning rate by factors of 10 and observing how the performance of the classifier changes. You should also try adding a **decay** which slowly reduces the learning rate over each epoch. - **Number of Epochs** - An epoch is a complete iterative pass over all of the data in the dataset. During an epoch we predict a label using the classifier and then update the weights of the classifier according the perceptron update rule for each sample in the training set. You should try different values for the number of training epochs and report your results.

You will implement the Perceptron classifier in the **models/Perceptron.py**

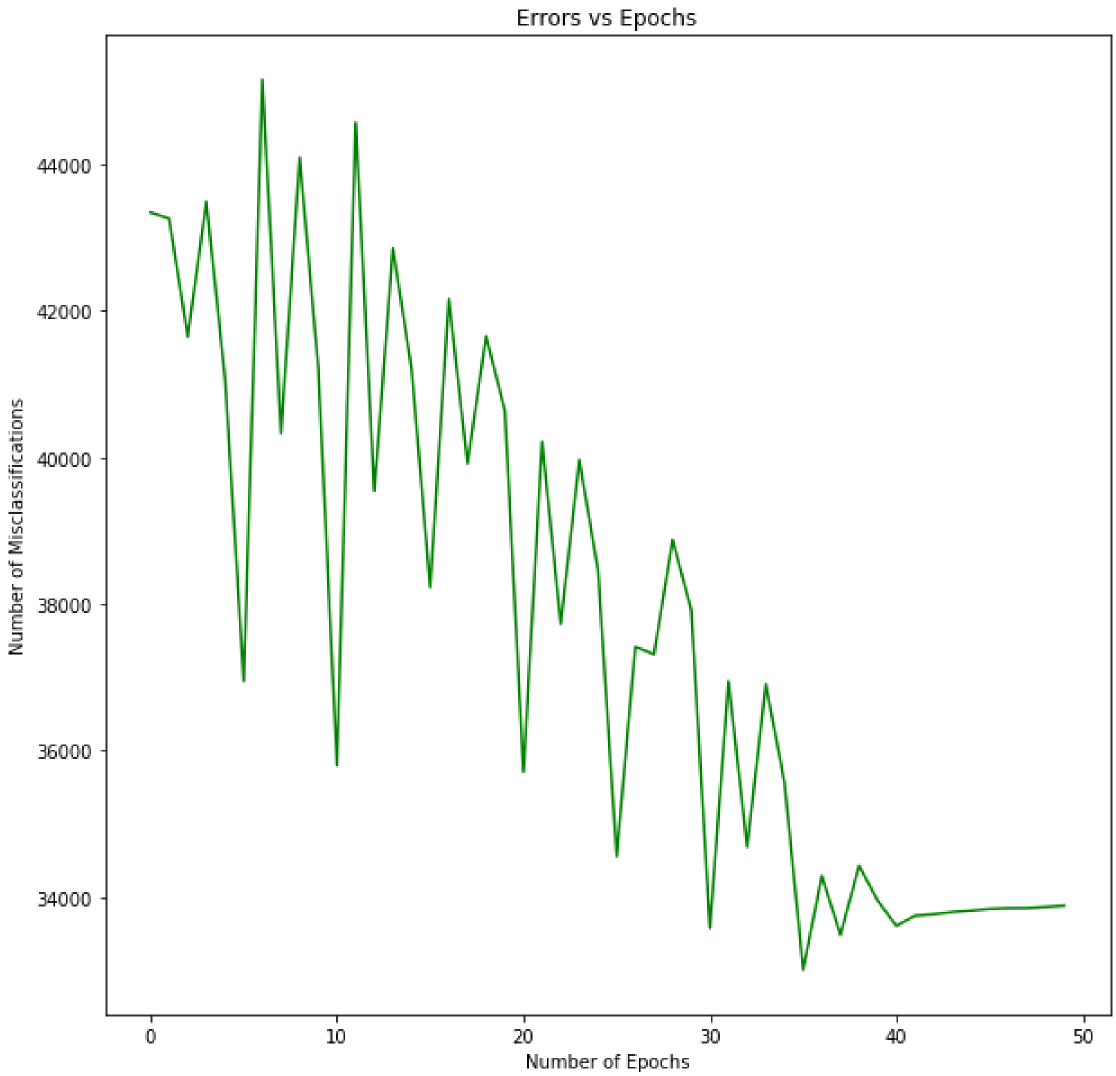
The following code: - Creates an instance of the Perceptron classifier class - The train function of the Perceptron class is trained on the training data - We use the predict function to find the training accuracy as well as the testing accuracy

3.0.1 Train Perceptron

[6]:
```
percept_ = Perceptron()
percept_.train(X_train, y_train)
```

```
(49000, 3072)
(49000,)
epoch: 0 , error: 43340  alpha:  3.0
epoch: 1 , error: 43257  alpha:  3.0
epoch: 2 , error: 41645  alpha:  3.0
epoch: 3 , error: 43490  alpha:  3.0
```

```
epoch: 4 , error: 41060  alpha:  1.5
epoch: 5 , error: 36948  alpha:  1.5
epoch: 6 , error: 45152  alpha:  1.5
epoch: 7 , error: 40326  alpha:  1.5
epoch: 8 , error: 44092  alpha:  1.5
epoch: 9 , error: 41214  alpha:  0.75
epoch: 10 , error: 35801  alpha:  0.75
epoch: 11 , error: 44563  alpha:  0.75
epoch: 12 , error: 39541  alpha:  0.75
epoch: 13 , error: 42847  alpha:  0.75
epoch: 14 , error: 41207  alpha:  0.375
epoch: 15 , error: 38225  alpha:  0.375
epoch: 16 , error: 42159  alpha:  0.375
epoch: 17 , error: 39913  alpha:  0.375
epoch: 18 , error: 41652  alpha:  0.375
epoch: 19 , error: 40638  alpha:  0.1875
epoch: 20 , error: 35709  alpha:  0.1875
epoch: 21 , error: 40210  alpha:  0.1875
epoch: 22 , error: 37729  alpha:  0.1875
epoch: 23 , error: 39964  alpha:  0.1875
epoch: 24 , error: 38459  alpha:  0.09375
epoch: 25 , error: 34558  alpha:  0.09375
epoch: 26 , error: 37416  alpha:  0.09375
epoch: 27 , error: 37314  alpha:  0.09375
epoch: 28 , error: 38874  alpha:  0.09375
epoch: 29 , error: 37915  alpha:  0.046875
epoch: 30 , error: 33581  alpha:  0.046875
epoch: 31 , error: 36946  alpha:  0.046875
epoch: 32 , error: 34686  alpha:  0.046875
epoch: 33 , error: 36907  alpha:  0.046875
epoch: 34 , error: 35559  alpha:  0.0234375
epoch: 35 , error: 33013  alpha:  0.0234375
epoch: 36 , error: 34292  alpha:  0.0234375
epoch: 37 , error: 33489  alpha:  0.0234375
epoch: 38 , error: 34430  alpha:  0.0234375
epoch: 39 , error: 33954  alpha:  0.01171875
epoch: 40 , error: 33610  alpha:  0.01171875
epoch: 41 , error: 33750  alpha:  0.01171875
epoch: 42 , error: 33769  alpha:  0.01171875
epoch: 43 , error: 33799  alpha:  0.01171875
epoch: 44 , error: 33819  alpha:  0.005859375
epoch: 45 , error: 33842  alpha:  0.005859375
epoch: 46 , error: 33852  alpha:  0.005859375
epoch: 47 , error: 33853  alpha:  0.005859375
epoch: 48 , error: 33868  alpha:  0.005859375
epoch: 49 , error: 33886  alpha:  0.0029296875
```




```
[7]: pred_percept = percept_.predict(X_train)
     print('The training accuracy is given by : %f' % (get_acc(pred_percept,␣
      ↪y_train)))
```

The training accuracy is given by : 30.851020

3.0.2 Validation

```
[8]: pred_percept = percept_.predict(X_val)
     print('The validation accuracy is given by : %f' % (get_acc(pred_percept,␣
      ↪y_val)))
```

The validation accuracy is given by : 28.400000

3.0.3 Test Perceptron

[9]:
```
pred_percept = percept_.predict(X_test)
print('The testing accuracy is given by : %f' % (get_acc(pred_percept, y_test)))
```

```
The testing accuracy is given by : 29.460000
```

3.0.4 Perceptron Kaggle Submission

Once you are satisfied with your solution and test accuracy output a file to submit your test set predictions to the Kaggle for Assignment 1 Perceptron. Use the following code to do so:

[10]:
```
output_submission_csv('perceptron_submission_29.csv', percept_.predict(X_test))
```

4 Support Vector Machines (with SGD)

Next, you will implement a ”soft margin” SVM. In this formulation you will maximize the margin between positive and negative training examples and penalize margin violations using a hinge loss.

We will optimize the SVM loss using SGD. This means you must compute the loss function with respect to model weights. You will use this gradient to update the model weights.

SVM optimized with SGD has 3 hyperparameters that you can experiment with : - **Learning rate** - similar to as defined above in Perceptron, this parameter scales by how much the weights are changed according to the calculated gradient update. - **Epochs** - similar to as defined above in Perceptron. - **Regularization constant** - Hyperparameter to determine the strength of regularization. In this case it is a coefficient on the term which maximizes the margin.

You will implement the SVM using SGD in the **models/SVM.py**

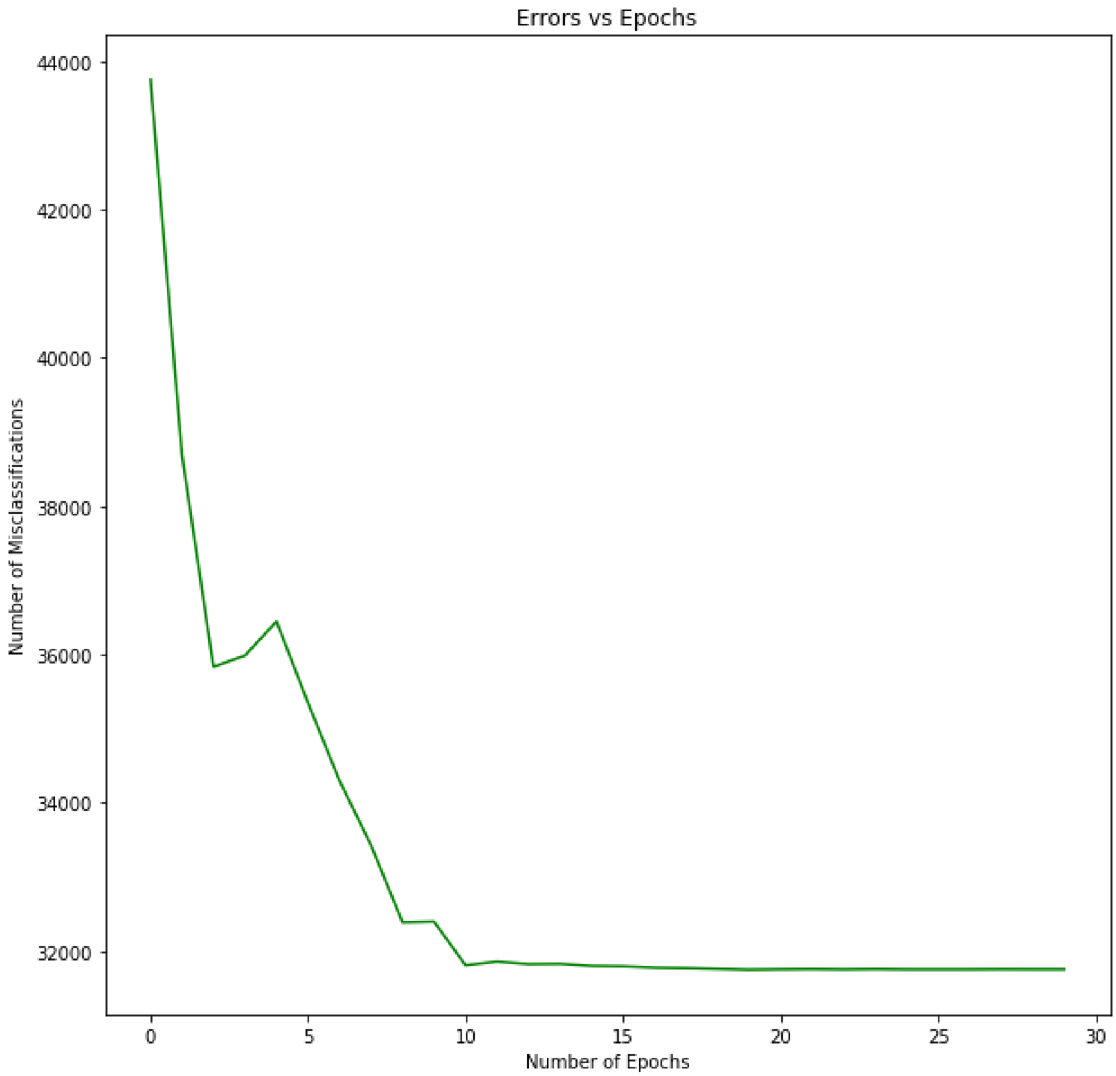
The following code: - Creates an instance of the SVM classifier class - The train function of the SVM class is trained on the training data - We use the predict function to find the training accuracy as well as the testing accuracy

4.0.1 Train SVM

[6]:
```
svm = SVM()
svm.train(X_train, y_train)
```

```
epoch: 0 , error: 43748  alpha:  0.1
epoch: 1 , error: 38688  alpha:  0.0740818220681718
epoch: 2 , error: 35830  alpha:  0.05488116360940264
epoch: 3 , error: 35982  alpha:  0.04065696597405992
epoch: 4 , error: 36442  alpha:  0.030119421191220214
epoch: 5 , error: 35343  alpha:  0.022313016014842982
epoch: 6 , error: 34295  alpha:  0.016529888822158657
```

```
epoch: 7 , error: 33421  alpha:  0.012245642825298192
epoch: 8 , error: 32384  alpha:  0.009071795328941252
epoch: 9 , error: 32394  alpha:  0.006720551273974979
epoch: 10 , error: 31805  alpha:  0.004978706836786395
epoch: 11 , error: 31856  alpha:  0.0036883167401240017
epoch: 12 , error: 31821  alpha:  0.002732372244729257
epoch: 13 , error: 31825  alpha:  0.002024191144580439
epoch: 14 , error: 31796  alpha:  0.0014995576820477704
epoch: 15 , error: 31791  alpha:  0.0011108996538242307
epoch: 16 , error: 31773  alpha:  0.0008229747049020031
epoch: 17 , error: 31767  alpha:  0.0006096746565515638
epoch: 18 , error: 31757  alpha:  0.00045165809426126705
epoch: 19 , error: 31746  alpha:  0.0003345965457471272
epoch: 20 , error: 31751  alpha:  0.00024787521766663585
epoch: 21 , error: 31754  alpha:  0.00018363047770289073
epoch: 22 , error: 31750  alpha:  0.0001360368037547894
epoch: 23 , error: 31754  alpha:  0.00010077854290485114
epoch: 24 , error: 31750  alpha:  7.465858083766799e-05
epoch: 25 , error: 31750  alpha:  5.530843701478336e-05
epoch: 26 , error: 31750  alpha:  4.097349789797868e-05
epoch: 27 , error: 31752  alpha:  3.035391380788668e-05
epoch: 28 , error: 31751  alpha:  2.248673241788482e-05
epoch: 29 , error: 31751  alpha:  1.6658581098763355e-05
```



[7]:
```
pred_svm = svm.predict(X_train)
print('The training accuracy is given by : %f' % (get_acc(pred_svm, y_train)))
```

```
The training accuracy is given by : 35.200000
```

4.0.2 Validate SVM

[11]:
```
pred_svm = svm.predict(X_val)
print('The validation accuracy is given by : %f' % (get_acc(pred_svm, y_val)))
```

```
The validation accuracy is given by : 32.700000
```

4.0.3 Test SVM

[12]:
```
pred_svm = svm.predict(X_test)
print('The testing accuracy is given by : %f' % (get_acc(pred_svm, y_test)))
```

```
The testing accuracy is given by : 35.020000
```

4.0.4 SVM Kaggle Submission

Once you are satisfied with your solution and test accuracy output a file to submit your test set predictions to the Kaggle for Assignment 1 SVM. Use the following code to do so:

[10]:
```
output_submission_csv('svm_submission_35.csv', svm.predict(X_test))
```

5 Softmax Classifier (with SGD)

Next, you will train a Softmax classifier. This classifier consists of a linear function of the input data followed by a softmax function which outputs a vector of dimension C (number of classes) for each data point. Each entry of the softmax output vector corresponds to a confidence in one of the C classes, and like a probability distribution, the entries of the output vector sum to 1. We use a cross-entropy loss on this sotmax output to train the model.

Check the following link as an additional resource on softmax classification: http://cs231n.github.io/linear-classify/#softmax

Once again we will train the classifier with SGD. This means you need to compute the gradients of the softmax cross-entropy loss function according to the weights and update the weights using this gradient. Check the following link to help with implementing the gradient updates: https://deepnotes.io/softmax-crossentropy

The softmax classifier has 3 hyperparameters that you can experiment with : - **Learning rate** - As above, this controls how much the model weights are updated with respect to their gradient. - **Number of Epochs** - As described for perceptron. - **Regularization constant** - Hyperparameter to determine the strength of regularization. In this case, we minimize the L2 norm of the model weights as regularization, so the regularization constant is a coefficient on the L2 norm in the combined cross-entropy and regularization objective.

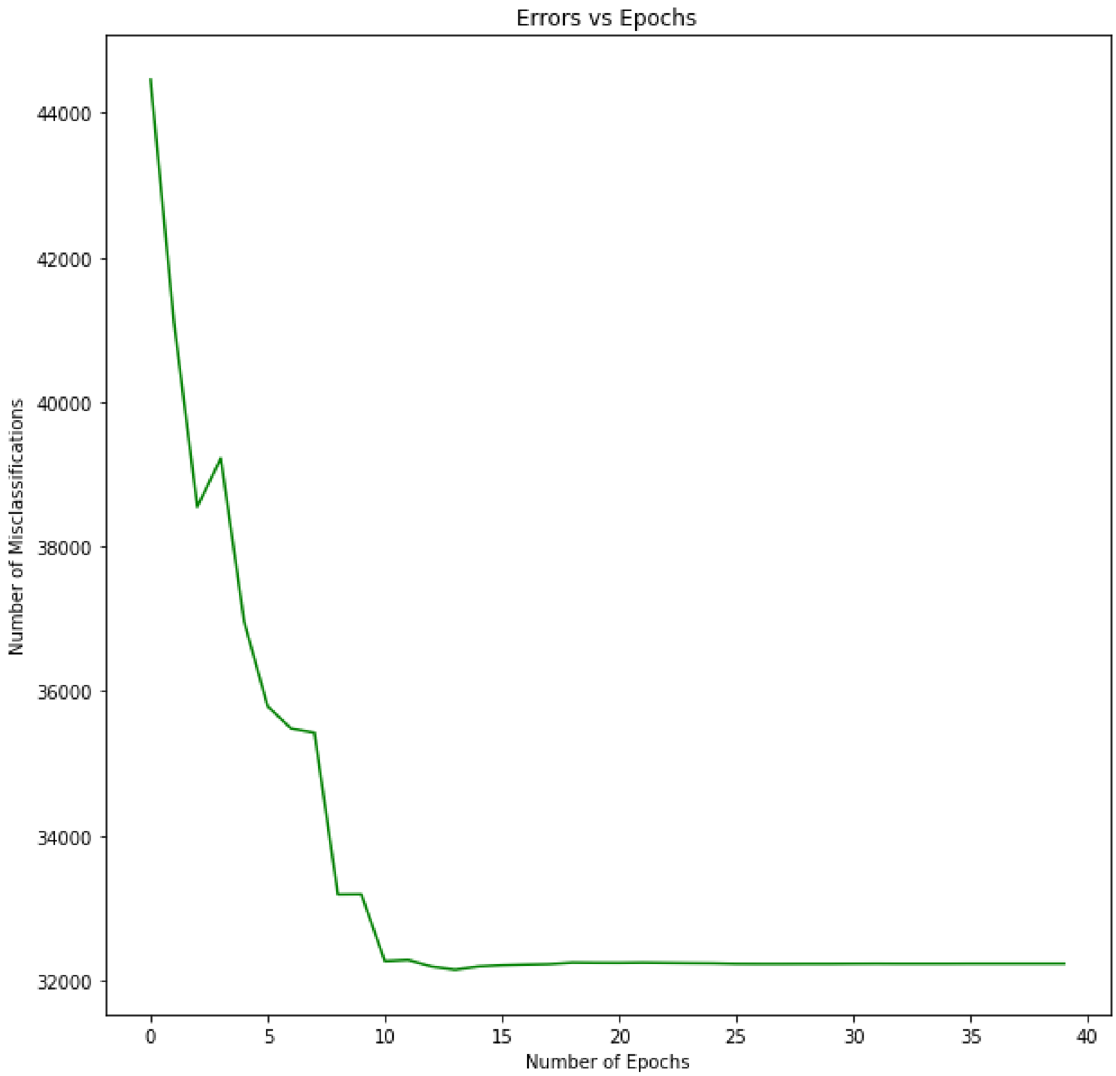
You will implement a softmax classifier using SGD in the **models/Softmax.py**

The following code: - Creates an instance of the Softmax classifier class - The train function of the Softmax class is trained on the training data - We use the predict function to find the training accuracy as well as the testing accuracy

5.0.1 Train Softmax

[6]:
```
softmax = Softmax()
softmax.train(X_train, y_train)
```

```
epoch: 0 , error: 44462  alpha:  0.1
epoch: 1 , error: 41113  alpha:  0.0740818220681718
epoch: 2 , error: 38548  alpha:  0.05488116360940264
epoch: 3 , error: 39221  alpha:  0.04065696597405992
epoch: 4 , error: 36956  alpha:  0.030119421191220214
epoch: 5 , error: 35789  alpha:  0.022313016014842982
epoch: 6 , error: 35481  alpha:  0.016529888822158657
epoch: 7 , error: 35421  alpha:  0.012245642825298192
epoch: 8 , error: 33184  alpha:  0.009071795328941252
epoch: 9 , error: 33186  alpha:  0.006720551273974979
epoch: 10 , error: 32262  alpha:  0.004978706836786395
epoch: 11 , error: 32278  alpha:  0.0036883167401240017
epoch: 12 , error: 32187  alpha:  0.002732372244729257
epoch: 13 , error: 32144  alpha:  0.002024191144580439
epoch: 14 , error: 32188  alpha:  0.0014995576820477704
epoch: 15 , error: 32205  alpha:  0.0011108996538242307
epoch: 16 , error: 32214  alpha:  0.0008229747049020031
epoch: 17 , error: 32221  alpha:  0.0006096746565515638
epoch: 18 , error: 32242  alpha:  0.00045165809426126705
epoch: 19 , error: 32239  alpha:  0.0003345965457471272
epoch: 20 , error: 32238  alpha:  0.00024787521766663585
epoch: 21 , error: 32241  alpha:  0.00018363047770289073
epoch: 22 , error: 32238  alpha:  0.0001360368037547894
epoch: 23 , error: 32235  alpha:  0.00010077854290485114
epoch: 24 , error: 32233  alpha:  7.465858083766799e-05
epoch: 25 , error: 32226  alpha:  5.530843701478336e-05
epoch: 26 , error: 32225  alpha:  4.097349789797868e-05
epoch: 27 , error: 32225  alpha:  3.035391380788668e-05
epoch: 28 , error: 32226  alpha:  2.248673241788482e-05
epoch: 29 , error: 32226  alpha:  1.6658581098763355e-05
epoch: 30 , error: 32228  alpha:  1.2340980408667957e-05
epoch: 31 , error: 32229  alpha:  9.142423147817343e-06
epoch: 32 , error: 32228  alpha:  6.77287364908539e-06
epoch: 33 , error: 32228  alpha:  5.017468205617529e-06
epoch: 34 , error: 32228  alpha:  3.7170318684126737e-06
epoch: 35 , error: 32229  alpha:  2.753644934974716e-06
epoch: 36 , error: 32229  alpha:  2.039950341117196e-06
epoch: 37 , error: 32229  alpha:  1.5112323819855034e-06
epoch: 38 , error: 32229  alpha:  1.119548484259094e-06
epoch: 39 , error: 32229  alpha:  8.293819160757371e-07
```




```
[7]: pred_softmax = softmax.predict(X_train)
     print('The training accuracy is given by : %f' % (get_acc(pred_softmax,␣
      ↪y_train)))
```

```
The training accuracy is given by : 34.226531
```

5.0.2 Validate Softmax

```
[8]: pred_softmax = softmax.predict(X_val)
     print('The validation accuracy is given by : %f' % (get_acc(pred_softmax,␣
      ↪y_val)))
```

```
The validation accuracy is given by : 34.500000
```

5.0.3 Testing Softmax

[9]:
```
pred_softmax = softmax.predict(X_test)
print('The testing accuracy is given by : %f' % (get_acc(pred_softmax, y_test)))
```

```
The testing accuracy is given by : 33.520000
```

5.0.4 Softmax Kaggle Submission

Once you are satisfied with your solution and test accuracy output a file to submit your test set predictions to the Kaggle for Assignment 1 Softmax. Use the following code to do so:

[10]:
```
output_submission_csv('softmax_submission_33.5.csv', softmax.predict(X_test))
```

[ ]:
```

```