Standard Classification Problem

20150860 김태윤

Method

To classify an x-ray image into normal/abnormal cases, I created a model that learns from training data and predict using a convolutional image classifier. The classifier is a CNN with three convolution layers, a flattening layer and the fully connected layer. The first convolution layer has 50 filters of size 3*3. The second one is 3*3*50 and third one is 3*3*100. In the creation of the convolutional layers, rectified linear unit (RELU) was used. To train the images, the image file should be loaded and fed into the trainer. The images are loaded with OpenCV and appended to a list. Usually, the images should be resized to a same size, but since the provided images were all same sizes, this process was omitted. Also, I needed some data to validate that the training is working. So I used 20% of the training data as validation data. After all the images are loaded and trained, a model file was created in the file system. Then, I created another python script that predicts the images using the model. The predictor first restores the saved model. Then, I used OpenCV to import unlabeled test dataset, as done in the training process. Image resize was omitted again since the test dataset is also same sized images. One by one, the images are fed into the model, which returned the result in a tuple. The first number represents the confidence for abnormal case, and the second represents the normal case. Repeating for all the test images, I obtained the prediction results for all 50 images. Running the *train.py* file trains from training data, and running *predict.py* file predicts for the test data. Train file should be run first to train the model and predict file should be run after.

Discussion

I put three convolutional layers in the classifier because it was a suitable number for the dataset. The dataset is not very large and the images are not very complex because they do not have many colors. For the original size image set, I could have used more layers, but since I trained on the small size

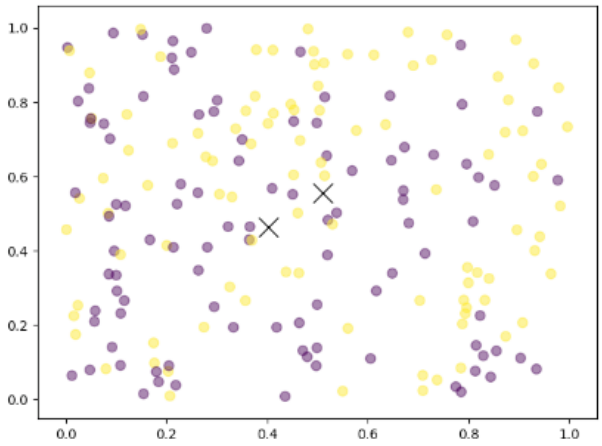
images, three layers were enough. Although there were cases where the confidence was lower, the results of the prediction were mostly very confident, with confidence of over 0.9. I used RELU as the activation layer because it is known to be efficient and it is used as a standard. The training is done for 22 epochs. The number is found by looking for where the validation loss becomes minimal. After the 22nd epoch, the loss began to increase, possibly due to over-confidence (a.k.a. overfitting). Thus, I used the model that has been trained for 22 epochs for optimal results.

Learning from Randomness

Methods

I implemented the k-means clustering algorithm in python. The goal was to assign 0 or 1 to each of the 10-dimensional points by finding the centroids of the two groups. First, I imported the provided data in npy format into a numpy array and converted them into tensors. The initial centroids were set randomly from the points. Then the distances between points were calculated using the TensorFlow reduce_sum method. The calculation of distance is same for any n dimension because square function of TensorFlow is element-wise. The square root of the sum of squares gives the Euclidean distance for any n dimensions. Therefore, the reduce_sum method gives distance squared which should be minimized as the iteration progresses. Then the centroids were updated based on the distances. The update is done by comparing the clusters with the assignments, get the points assigned to each cluster, and calculate the mean value. This process was repeated for n iterations. After the iteration, the final value for the centroids and assignment of clusters are returned. The points and the cluster assignment are mapped to each other and written to a text file. The implementation of this problem is in *separate.py* file.

Result



The result of clustering is written to *separation_result.txt* in the project folder. Each point is assigned either cluster 0 or 1. Since the data is random, I would assume the number of points in both clusters should be similar. Indeed, the size of both clusters are similar through multiple tests. The figure on the right shows the points and the centroids for 1st and 2nd dimensions of the data. Because the data is random,

the centroids are close together.

Discussion

To classify the provided points into two groups, I first thought of the SVM to solve the problem. However, SVM for multidimensions seemed difficult to implement. Another algorithm I have discovered is the k-means clustering. I have seen k-means clustering example on 2d data, but I had to confirm it works for 10 dimensional data. Looking into the principles of the algorithm, I found that it uses the Euclidean distance between points for clustering.
Therefore I concluded it would work for 10-dimensional data.
I have tried adjusting iteration number, but because the data is random, there was no apparent increase in accuracy if it was larger than around 10. If the data was actually grouped into two clusters, increasing the iteration might have improved the accuracy.
Because the centroids are set at random points at the beginning, the result of clustering is not always identical.