Ferdowsi university of Mashhad

Faculty of Engineering

computer engineering group

Repo Report on the project of the Computational intelligence course

Title

Sentiment analysis

Writers

Mohammad Raee

Ali MoghadasZade

Teacher

Dr. Ehsan Fazl Ersi

2021-May

subject

The aim of this project is to implement the K-means clustering algorithm using MLP neural networks. In the first step, run the K-means clustering algorithm with various parameters, including the simple mode you want to implement, on the data provided to you and examine the effect of changing each of these parameters on the performance of the algorithm. To measure performance, use the two criteria taught in the classroom, Purity and Rand Index.

In the next step, we want to implement the orphan algorithm using neural networks. To do this, first, compare the various features of the K-means algorithm with what you know about MLP networks, and then try to find a solution to this problem. Also, note that clustering algorithms, such as K-means, operate unsupervised and do not require labels; Neural networks, on the other hand, need data labels to become the most. What do you suggest to solve this problem?

In this step, depending on the method you want, change various parameters such as neural network architecture, etc., and examine its effect on the performance of the algorithm. Use the MNIST database to measure the performance of the algorithm. This data set, which consists of 28 by 28 handwritten images, includes 60,000 train images and 10,000 test images.

If you want to get more information about this database, you can refer to the following site:

http://yann.lecun.com/exdb/mnist /

Our algorithm for solving this problem is as follows:

First, we do random labeling on the train data. Then we consider 10 vectors, each of which is 10D, each representing one of our 10 classes. That is, each vector is actually a representation of a class.

The vector belongs to class 1, its first component is 1 and the rest is 0. The vector belongs to class 2, the second component is 1 and the rest is 0, and so we have 10 vectors.

Now every data we give to our MLP machine gives us a vector of 10. That is, we have 10 perceptrons in the last layer, and each one gives us a number. (Here we do not have threshold and fire of perceptrons in the last layer). Now, these 10 numbers practically become a vector of 10. So when we give data to MLP, it gives us a vector of 10.

We now examine which of our classes this vector is closest to. (Each class is a vector of 10, and the measure of proximity here is the Euclidean distance)

Thus, a label is found for this data. Well, in fact, we compare this new label with the previous stage label (note that the first stage label was random) and by comparing the two, we update the weights.

And now we do the same for the whole data. This means that this operation is performed on all data and all data are labeled new and weights are updated at each step.

We continue to do this until it crosses a threshold. This threshold may be that the number of data whose label at this stage is the same as their label in the previous step, is more than one percent, for example, 80 percent of the total data.

In this way, we practically trained unlabeled data with MLP, leading to a mechanism similar to kmeans.

The similarity of this algorithm with kmeans is in two points:

Note that we have to specify the number of clusters from the beginning, just as in kmeans we have to specify k, which was the number of clusters.

Secondly, here your criterion for clustering (or similar here is labeling) at each step for each data is that we get the distance to 10 vectors, which is the representation of 10 classes, and was closer to each, We assign to the class. Well, in kmeans, the mechanism was the same, that is, we get the data distance to 10 cluster centers (which was a representation of our 10 clusters), and then, closer to each one, we assigned the cluster to it.

1. Idea:

The general idea in implementing this algorithm is that first a label is assigned to the data and then in each epoch, however, the average of the predictions made in each epoch is calculated and from it, the average of the new labels is given. The data is attributed. This is repeated until the result is almost constant.

2. General trend:

First, we normalized the data so that their range is between 0 and 1. Then we selected the initial 6000 datasets and finalized the model on them. Then we tested the neural network several times to get a good evaluation of the algorithm on all the data due to the randomness in the neural network.

3. Implementation of K-Means:

In this phase, we calculated the sklearn K-Means with the help of the library and obtained the rand index for it with the help of rand_score from the same library. Then, with the help of the code that we made from purity, we also obtained its purity. The results were as follows:

Rand Index - train: 0.8794538947871353

Rand Index - test: 0.8801713771377138

Purity - train: 0.5907166666666667

Purity - test: 0.5945

4. Neural network structure:

Different structures were tested for the neural network. We started with 2 layers of 64. By examining different structures such as one layer 5, layer 32, one layer 64, one layer 256, two layers 64, two layers 128 and 64, two layers 256 and 32, two layers 256 and 64, three layers 512 and 256 and 64, Three layers 512, 128, 32 and other structures, and also by testing the activation functions of relu, sigmoid, and softmax, we came to the conclusion that the best structure is two layers 256 and 64 with the help of sigmoid activation functions and the last linear layer. The loss function was also tested for the best results. In most of these tests, the numerical rand index was about 78 to 82%, which is a good result, but for purity, the numerical range was 15 to 25, which was not good. (On test and train data) In the final model, MeanSquaredError was considered for the loss function, which resulted in an accuracy of about 80% for the rand index and about 30% for purity, which was better than the other models. In this algorithm, for every 10 epochs that get better results from the rest based on experience (as opposed to 1, 5, 20, 30, 50), the average of its predictors is collected and new labels are determined based on it.

5. Notes:

The first thing that seemed very noticeable was the high speed at which the labels converged with the weights. This process was done so fast that in 30 40 epochs the accuracy reached 100%. Two things were considered to reduce this speed. The first point was to add alpha to calculate the effect of the predictors on the labels and changes to reduce that epoch. This made the result worse, and in further research, we decided not to affect the changes and labels on the new labels, and to determine the label entirely based on its 10 epoch predictions. The second point was to examine the effect of the number of epochs to determine the new tag. As the number decreased, the rate of convergence increased sharply, and the result became weaker (in a way, it learned the same as the original random tags), and as it increased, its rate decreased, but the sum of the obtained predictors was more effective. Eventually, it would have the same result with fewer numbers. As a result, the 10 best numbers were obtained.

The second point that is significant and shows the efficiency of the algorithm was the review of the rand index. In the rand index study, one of the things that were done was to test the score of this metric on the same randomly assigned labels without neural network training. Surprisingly, the score was about 80%. But the score on the test data was 10%. After training the neural network, the accuracy of the test data reached the same 80%, and with this test, we were sure of the efficiency of the neural network.

The third point we examined was the effect of initial weight on the final result. When we ran a neural network several times on a data set without any changes, the purity changed by about

5%. This meant that the neural network and the algorithm model had little resistance to the initial values, or in other words, the convergence rate was so high that the initial weights played an important role in the final result.

Of course, it should be noted that even with 100% accuracy, it is expected that with more network training, the labels will continue to change, which will help the final result. This can be understood by examining the accuracy of the epochs and the amount of loss. Because even with 100% accuracy in most epochs, sometimes the accuracy would go down to about 99% and it would be 100% again. This indicates that with further training, the labels continue to change and the training has stopped or is not at its best. This can also be seen with fluctuations and changes in the loss function. Of course, the amount of loss in the entropy function has a decreasing trend, and even with 100% accuracy, this amount of loss is still significantly reduced to a very small amount and no change in accuracy is observed. In MeanSquaredError, although the accuracy seems to have stopped at one point, the labels still change, giving the final evaluation better accuracy than the epoch report. In this function, it gives less accuracy and loss of information, and the only impression that can be made is that it is possible that in some cases, it gets stuck at a relative minimum and no longer shows a useful change. While the final result in changing the labels was better than the other cases.

Another test case was adding randomness to predictions in an epoch category. Adding a random value to its data between 0 and 0.2 made the result worse, and this randomness made the work weaker.

6. More work to continue:

One of the things that can be done to improve the result is to train the model on the average of the clusters and after training, to check the result obtained on the data according to the training done. There are several things you can do to get started. The first step is to give a random label and average the categories based on them, and the other way is to select a random number of clusters of data as the starting point. Then at each stage, we update this average based on the output of the neural network to finally reach a good result. When I tried these different methods, the result did not make much difference and the high neural network still worked better. But it is expected that with further study and completion of this algorithm, a better result can be obtained.

You can also do better with pre-processing and get better results.