SOI1010 Machine Learning II - Assignment #1

작성자: 2023036299김진욱

Colab URL Link:

https://colab.research.google.com/drive/1ZXRRvVvv7evmu_sBHgf1eiTMk8XB_d1P?usp=sharing

(I have allowed access to users affiliated with Hanyang University, but I am also including ipynb and py just in case.)

(a) Implement a k-NN algorithm (k = 5) using an iterative method (i.e., using for loop) to classify a single new example. Write down your observations.

```
#Problem (a)
tmp_idx = 42 # 42 in the val's range.
tmp_image = val_data[tmp_idx]
tmp_labels = val_labels[tmp_idx]
pair_list = [] # (거리, 라벨) 저장.
for i in range(len(train_data)):
  now_image = train_data[i]
  now_labels = train_labels[i]
  distance = torch.sum((tmp_image-now_image)**2) #sqrt는 단조증가 함수라서 씌우지 않는다.
  pair_list.append((distance.item(),now_labels.item()))

sorted_list = sorted(pair_list, key=lambda pair:pair[0])

print("result Wn")
print(sorted_list[:5])
```

```
def give_label(info: list):
  counting = {}
  for distance, label in info:
    if distance == 0:
      return label
    counting[label] = counting.get(label,0) + 1

  maximum = max(counting.values())

  nominate = []

  for label,count in counting.items():
    if count == maximum:
      nominate.append(label)

  if len(nominate) == 1:
    return nominate[0]

  else:
    check_winner = {}
    for distance, label in info:
      if label in nominate:
        check_winner[label] = check_winner.get(label,0) + 1/distance

    return max(check_winner, key=check_winner.get)
print(give_label(sorted_list[:5]))
```

I selected the image at index 42 from the validation set and proceeded to calculate the Euclidean distance (sum of squares) between it and every image in the training set. The distance was computed using the formula `torch.sum((tmp_image - now_image)**2)`. The square root operation was omitted, as it is a monotonically increasing function and thus does not affect the ordering of distances.

The calculated distance and the corresponding label for each training image were stored in a `pair_list`, which was subsequently sorted in ascending order based on distance.

The `give_label` function determines the final label through a majority vote of the *k*-nearest neighbors. If a neighbor with a distance of zero exists, its label is returned immediately. In the event of a tie, a greater weight is assigned to the label of the neighbor with the shorter distance.

An examination of the five nearest neighbors revealed that all were classified with the label 0, as shown in the following list: `[(35.71235656738281, 0), (38.872032165527344, 0), (40.079063415527344, 0), (40.36872100830078, 0), (40.44904327392578, 0)]`. Therefore, the predicted label for the image is 0.

(b) Implement a k-NN algorithm (k = 5) using the broadcasting concept you learned in the laboratory session to classify a single new example. Compare against the result from (a).

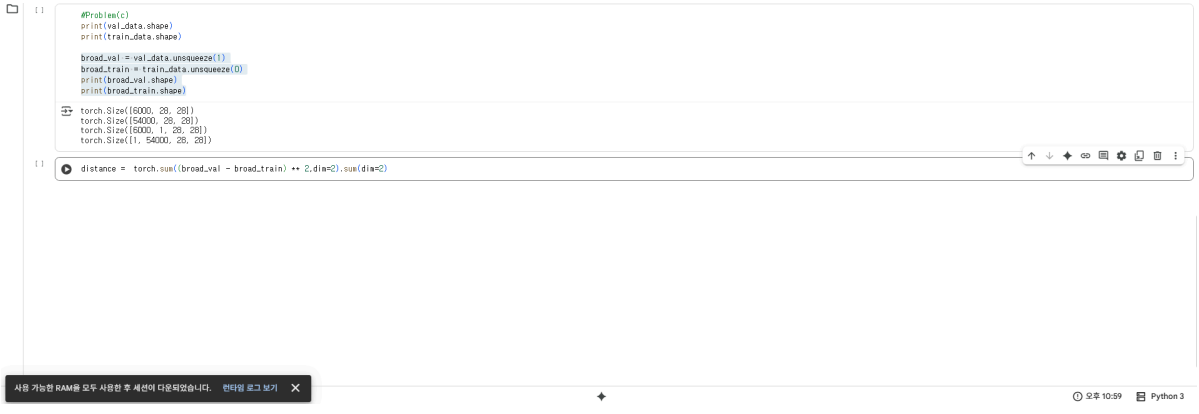
As in problem (a), I used the image at index 42 from the validation set. I applied the unsqueeze(0) method to this image (image_for_test), expanding its dimensions to [1, 28, 28]. This step enables broadcasting operations with the training data (train_data), which has a shape of [54000, 28, 28].

I efficiently calculated the Euclidean distance (sum of squares) between the test image and all training images using the expression torch.sum((train_data - plus_dimension)**2, dim=1).sum(dim=1). In this operation, the sequential summations along dim=1 effectively compute the total sum of squared pixel differences across both the height and width for each image pair.

Next, using the torch.topk(distance, k=5, largest=False) function, I retrieved the five smallest distances and their corresponding indices. The result of this broadcast calculation showed that the distances and labels of the five nearest neighbors were identical to those from problem (a).

The torch.topk function proved to be far more convenient and efficient than the manual sorting method used in problem (a), as it allows for easy configuration of the sorting order (ascending/descending via largest=False) and the number of items (k). It also returns both the actual distance values and their indices.

(c) Now, extend a k-NN algorithm from (b) to perform classification over all digits for all new images at once, using broadcasting (not just a single image). Write down your observations. Can you find any issue?



I attempted to use broadcasting to calculate the distances between all images in the validation set (val_data, shape: [6000, 28, 28]) and all images in the training set (train_data, shape: [54000, 28, 28]) in a single, vectorized operation.

In my initial approach, I expanded the dimensions of val_data to [6000, 1, 28, 28] and

train_data to [1, 54000, 28, 28]. I then tried to compute the distances using torch.sum. However, this approach created an extremely large intermediate tensor with a shape of [6000, 54000, 28, 28], which resulted in an out-of-memory error.

Calculating the distances between every validation image and every training image directly via broadcasting requires a prohibitive amount of memory, making it infeasible in a typical computing environment. Although the final distance matrix should have a shape of [6000, 54000], the inclusion of the pixel dimensions (28x28) in the intermediate step causes the memory usage to increase exponentially.

(d) If there is any issue from (c), what is the cause of the issue? Improve the algorithm from (c) by resolving the issue you find from (c) [Hint: Try to find a function to replace a part of your algorithm by either googling or going through PyTorch document].

To resolve the memory issue, I used the torch.cdist function, which efficiently computes the distance between all pairs of row vectors in two tensors.

To prepare the data for this function, each image had to be flattened into a one-dimensional vector. I transformed the val_data and train_data tensors into 2D tensors with the shape [number of images, number of pixels] by using .view(num_images, -1). For instance, the val_data tensor with a shape of [6000, 28, 28] was reshaped to [6000, 784].

On these flattened tensors, I called torch.cdist(br_val, br_tr, p=2) to efficiently calculate the L2 distance (Euclidean distance) between every validation image and every training image. The resulting total_distance tensor has a shape of [6000, 54000], where each element represents the distance between a specific pair of validation and training images.

To find the k-nearest training images for each validation image, I used torch.topk(total_distance, k=5, dim=1, largest=False). The dim=1 argument ensures the operation is performed across each row (for each validation image). After obtaining the indices of the nearest neighbors, I retrieved their labels (labels = train_labels[idx]) and used torch.mode(labels, dim=1) to determine the final predicted label via a majority vote. The torch.mode function is highly optimized for tensor operations and is far more concise than a manual implementation, processing the data quickly without memory concerns.

```
distance_info, idx = torch.topk(total_distance,k=5,dim=1,largest=False)

labels = train_labels[idx]

print(labels.shape)

results, _ = torch.mode(labels,dim=1)

print(results.shape)
print(_.shape)

print(results)
print(val_labels[_])
print(_)
```

```
torch.Size([6000, 5])
torch.Size([6000])
torch.Size([6000])
tensor([9, 3, 5,  ..., 9, 9, 8])
tensor([7, 7, 7,  ..., 7, 7, 7])
tensor([4, 4, 4,  ..., 4, 4, 4])
```

```
print(results[42])
print(give_label(sorted_list[:5]))
```

```
tensor(0)
0
```

By leveraging torch.cdist, torch.topk, and torch.mode, I successfully solved the memory problem encountered previously and performed an efficient classification on the entire validation set. I confirmed that these built-in PyTorch functions are significantly faster and more concise than implementing the same logic with manual loops.

(e) What are the hyperparameters you can tune?

val_ratio (Validation Set Ratio): This represents the proportion of the entire dataset that is allocated to the validation set. Adjusting this ratio changes the amount of data used to evaluate the model's generalization performance.

k (Number of Neighbors): In the k-NN algorithm, this is the number of nearest neighbors that are referenced when classifying a new data point. The value of k influences the model's complexity and its tendency toward overfitting or underfitting.

(f) Try at least two other options for each hyperparameter. Write down your observations.

I explored a range of val_ratio values from 0.01 to 0.30 with a 0.01 increment, and k values from 5 to 20, measuring the validation accuracy for each combination. For every val_ratio, I found the optimal k and recorded the corresponding accuracy.

```
Ratio:0.01, k:7, Accuracy:0.9783333539962769
Ratio:0.02, k:7, Accuracy:0.9733333587646484
Ratio:0.03, k:6, Accuracy:0.971666693687439
Ratio:0.04, k:5, Accuracy:0.9708333611488342
Ratio:0.05, k:5, Accuracy:0.971666693687439
Ratio:0.06, k:5, Accuracy:0.9708333611488342
Ratio:0.07, k:5, Accuracy:0.9719047546386719
Ratio:0.08, k:5, Accuracy:0.9735416769981384
Ratio:0.09, k:5, Accuracy:0.9729629755020142
Ratio:0.1, k:5, Accuracy:0.9721666574478149
Ratio:0.11, k:5, Accuracy:0.9727272987365723
Ratio:0.12, k:5, Accuracy:0.9715277552604675
Ratio:0.13, k:5, Accuracy:0.9715384840965271
Ratio:0.14, k:5, Accuracy:0.9725000262260437
Ratio:0.15, k:5, Accuracy:0.9725555777549744
Ratio:0.16, k:5, Accuracy:0.9729166626930237
Ratio:0.17, k:5, Accuracy:0.9728431105613708
Ratio:0.18, k:5, Accuracy:0.9734259247779846
Ratio:0.19, k:5, Accuracy:0.9727193117141724
Ratio:0.2, k:5, Accuracy:0.9722499847412109
Ratio:0.21, k:5, Accuracy:0.9719841480255127
Ratio:0.22, k:5, Accuracy:0.9717424511909485
Ratio:0.23, k:5, Accuracy:0.9707971215248108
Ratio:0.24, k:5, Accuracy:0.9711111187934875
Ratio:0.25, k:5, Accuracy:0.9713333249092102
Ratio:0.26, k:5, Accuracy:0.9712820649147034
Ratio:0.27, k:5, Accuracy:0.9708641767501831
Ratio:0.28, k:5, Accuracy:0.9710714221000671
Ratio:0.29, k:5, Accuracy:0.9710344672203064
Ratio:0.3, k:5, Accuracy:0.9701111316680908
Ratio:0.01, k:7
```

The results showed that the highest accuracy was observed at val_ratio = 0.01 and k = 7. However, across the entire val_ratio range, k = 5 provided a relatively stable and consistently good accuracy.

(g) You can try more options if you want. What is the final test accuracy?

For the final evaluation, I conducted tests on two models: one with k=5, which had demonstrated consistently strong performance, and another with k=7, which achieved the highest score during validation.

```
Ratio :0.01, Best k: 7, F1:0.9791160821914673
Ratio :0.02, Best k: 7, F1:0.9738765954971313
Ratio :0.03, Best k: 6, F1:0.9721206426620483
Ratio :0.04, Best k: 5, F1:0.9711148142814636
Ratio :0.05, Best k: 5, F1:0.9718723297119141
Ratio :0.06, Best k: 5, F1:0.9708722829818726
Ratio :0.07, Best k: 5, F1:0.9717548489570618
Ratio :0.08, Best k: 5, F1:0.9734271168708801
Ratio :0.09, Best k: 5, F1:0.9727876782417297
Ratio :0.1, Best k: 5, F1:0.9720419049263
Ratio :0.11, Best k: 5, F1:0.9726301431655884
Ratio :0.12, Best k: 5, F1:0.971340537071228
Ratio :0.13, Best k: 5, F1:0.971377968788147
Ratio :0.14, Best k: 5, F1:0.9723551869392395
Ratio :0.15, Best k: 5, F1:0.97236168384552
Ratio :0.16, Best k: 5, F1:0.9727203249931335
Ratio :0.17, Best k: 5, F1:0.9726755023002625
Ratio :0.18, Best k: 5, F1:0.9732446670532227
Ratio :0.19, Best k: 5, F1:0.972558319568634
Ratio :0.2, Best k: 5, F1:0.9721009135246277
Ratio :0.21, Best k: 5, F1:0.9718462228775024
Ratio :0.22, Best k: 5, F1:0.9715965986251831
Ratio :0.23, Best k: 5, F1:0.970687747001648
Ratio :0.24, Best k: 5, F1:0.9710088968276978
Ratio :0.25, Best k: 5, F1:0.9711974859237671
Ratio :0.26, Best k: 5, F1:0.9711820483207703
Ratio :0.27, Best k: 5, F1:0.9707486033439636
Ratio :0.28, Best k: 5, F1:0.9709625244140625
Ratio :0.29, Best k: 5, F1:0.9709056615829468
Ratio :0.3, Best k: 5, F1:0.969357151985168
Ratio:0.01, K:7,F1:0.9791160821914673
```

In addition to Accuracy, I also used the F1-Score as an evaluation metric. Although the MNIST dataset is well-balanced, I was interested in investigating whether the choice of metric would influence the outcome. To comprehensively assess performance across all classes (0 through 9), I calculated the average F1-Score. This was done by first computing the True Positives (TP), False Positives (FP), and False Negatives (FN) for each class to determine its precision and recall, and then calculating the F1-Score from those values.

The final evaluation was performed on the test set using the entire training dataset. I measured both the F1-Score and Accuracy for the two selected cases: k=7 (the top performer on the validation set) and k=5 (the most stable performer).

```
k:7, accuracy:0.9693999886512756
```

```
k:7,F1:0.9693803787231445
```

```
k:5,F1:0.9687143564224243,Acc:0.9688000082969666
```

The results showed that k=7 achieved the highest scores for both F1-Score and Accuracy on the test data, mirroring its performance on the validation set. Based on these metrics, k=7 can be judged as the better choice.

However, a strong argument can be made that k=5 is a more robust and reliable choice due to its consistent performance across the entire range of val_ratio values explored earlier. A key consideration is that a very low val_ratio—the condition under which k=7 performed best—carries an increased risk of overfitting to that small, specific validation split. This suggests that unconditionally selecting the k value with the highest peak performance is not always the most advisable strategy.

Although the results for k=5 and k=7 from hyperparameter tuning were sufficiently high, I reasoned that the classification mechanism itself has inherent limitations. The process works by flattening a 28×28 handwritten digit into a vector, calculating its distance to all images in the training set, and then assigning the majority label from its k nearest neighbors.

I hypothesized that this cdist-based approach, which relies on raw pixel-wise distance, would struggle to correctly classify digits that are easily confusable due to stylistic similarities (e.g., a loopy '5' and a '6', a rounded '7' and a '9', or a closed '6' and an '8'). To investigate this, I decided to examine the per-label True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) for the k=7 model.

```
--- Each Label's Accuracy ---
0's Accuracy: 99.39%
1's Accuracy: 99.82%
2's Accuracy: 95.74%
3's Accuracy: 96.63%
4's Accuracy: 96.23%
5's Accuracy: 97.09%
6's Accuracy: 98.54%
7's Accuracy: 96.21%
8's Accuracy: 94.05%
9's Accuracy: 95.44%
```

```
2's score: Tp:988.0, Tn:8951.0, Fp:17.0, Fn:44.0
7's score: Tp:989.0, Tn:8929.0, Fp:43.0, Fn:39.0
8's score: Tp:916.0, Tn:9013.0, Fp:13.0, Fn:58.0
9's score: Tp:963.0, Tn:8947.0, Fp:44.0, Fn:46.0
```

The model's performance for the digit '2' is generally strong, but its overall accuracy is impacted by occasional False Negatives, where it fails to identify a true '2'. In the case of '8', its combination of high True Negatives, low False Positives, low True Positives, and high False Negatives demonstrates that the model frequently fails to correctly identify the digit. A different issue affects '7' and '9'; while both have high True Positive and True Negative counts, they suffer from a high number of both False Positives and False Negatives. This is likely due to handwriting variations, such as a rounded '7' being visually similar to a '9', which causes the model to misclassify them.

This assignment highlighted the limitations of using vectorized distances for image classification and the importance of first searching for supported library functions when implementing a feature. In an early step, I classified a single data point by manually creating a list of distance-label pairs, sorting it by distance, and using a custom function to find the most likely label. While building this from scratch was a valuable learning experience, I found that built-in PyTorch functions like torch.topk and torch.mode were far more concise and optimized for tensor operations, allowing for faster, memory-safe processing. This realization drove home the importance of developing proficiency with available libraries for future machine learning projects.