# SOI1010 Machine Learning II - Assignment #1

Assigned: Sep. 25, 2023 Due: Oct. 10, 2023 11:59 pm
2022094093 Kim Dohoon, Dept. of Data Science
Colab Link : https://colab.research.google.com/drive/10LLg5bw0-StKbngFtMrYP1VNpHR4d0wP
Github Link : https://github.com/kdh-yu/ML2/tree/main/Assignment/Assignment%231

---

## (a) Implement an iterative method (using for loop) to classify a single new example. Write down your observations.

First, I chose random data from validation data, using **np.random.randint(len(x_val))**. The shape of **train_data** is (54000, 28, 28), and the shape of **sample_data** is (28, 28). With 54000 iteration, I calculated the euclidean distance. The result was saved as tensor shape of (54000,).

```
--------------------------------------------
Label : 4
Predicted : 4
Running time : 3.21 second(s)
--------------------------------------------
```

Iterative k-NN successfully predicted sample data.

---

## (b) Use the broadcasting concept you learned in the laboratory session to classify a single new example. Compare against the result from (a).

The shape of x_train is (54000, 28, 28) and sample_data is (28, 28). To use broadcasting, I used unsqueeze method. After that the shape of sample data became (1, 28, 28).

```
--------------------------------------------
Label : 4
Predicted : 4
Running time : 0.01 second(s)
--------------------------------------------
```

Running time dramatically decreased.

(c) Now, implement a k-NN algorithm (starting with k=5) and its training/validation/evaluation code to perform multiclass classification over all digits, using the implementation from (b). Write down your observations.

I used iterative method to check the distance of each 6000 data.

```
-----------------------------------------------
Model Evaluation

Validation Accuracy : 97.27%
Validation Running time : 23.42 second(s)
-----------------------------------------------
```

In a short time (about 23 seconds) model could achieve 97.27% accuracy.

(d) Improve the algorithm from (c)
[Hint: Try to find the desirable distance function, which can be found by googling or going through PyTorch document].

> https://pytorch.org/docs/stable/nn.functional.html#distance-functions
> https://pytorch.org/docs/stable/generated/torch.norm.html#torch-norm

Problem I found above is that model takes long time, which is about 24 seconds. Both custom-defined euclidean distance and torch.nn.PairwiseDistance(p=2) were same performance.
Here, torch.norm performed better than those two. About 10 seconds were decreased. The results are below.

```
-----------------------------------------------
Model Evaluation : L2 Norm, Pairwise Distance

Validation Accuracy : 97.27%
Validation Running time : 23.52 second(s)
-----------------------------------------------
```

```
-----------------------------------------------
Model Evaluation : L2 Norm, torch.norm

Validation Accuracy : 97.27%
Validation Running time : 14.59 second(s)
-----------------------------------------------
```

Of course the results are same, because of the same k value and distance measurement.

## (e) What are the hyperparameters you can tune?

There are two hyperparameters I can tune; k value, and distance measurement.

1. K-value
   It means how many data model will use to classify new data. For example, k=5 means new data will be classified using the nearest 5 train data.

2. Distance measurement
   It determines how to calculate the distance. Available options are these;

- Manhattan Distance = L1 Norm
  $||x||_1 = \Sigma|x_i|$
- Euclidean Distance = L2 Norm
  $||x||_2 = (\Sigma|x_i|^2)^{\frac{1}{2}}$
- Minkowski Distance = Lp Norm
  $||x||_p = (\Sigma|x_i|^p)^{\frac{1}{p}}$
- Chebyshev Distance = Max Norm
  $||x||_\infty = max_i(|x_i|)$

---

## (f) Try at least two other options for each hyperparameter. Report the performance for each option.

1. We have 54,000 train data, and $\sqrt{54000} = 60\sqrt{15} \simeq 232.379...$
   So I will test 1, 3, 5, 10, 100, 200 as k-value.
2. I will test L1, L2, L-infinity norm. So, there are $6 \times 3 = 18$ possible pairs.

And, here is the history.

| k value | norm | Accuracy | Running time |
|---|---|---|---|
| 1 | 1 | 96.77% | 14.60 second(s) |
| 1 | 2 | 97.47% | 14.56 second(s) |
| 1 | inf | 83.08% | 14.70 second(s) |
| 3 | 1 | 96.93% | 14.71 second(s) |
| 3 | 2 | 97.55% | 14.49 second(s) |
| 3 | inf | 82.08% | 14.46 second(s) |
| 5 | 1 | 96.45% | 14.48 second(s) |
| 5 | 2 | 97.27% | 14.68 second(s) |
| 5 | inf | 82.00% | 14.61 second(s) |
| 10 | 1 | 96.25% | 14.54 second(s) |

| k value | norm | Accuracy | Running time |
|---|---|---|---|
| 10 | 2 | 97.00% | 14.51 second(s) |
| 10 | inf | 81.03% | 14.48 second(s) |
| 100 | 1 | 93.25% | 14.47 second(s) |
| 100 | 2 | 94.23% | 14.47 second(s) |
| 100 | inf | 73.22% | 14.58 second(s) |
| 200 | 1 | 91.60% | 15.66 second(s) |
| 200 | 2 | 92.77% | 18.26 second(s) |
| 200 | inf | 71.02% | 14.80 second(s) |

```
--------------------------------------------
Highest Accuracy at : k=3, p=2

Validation Accuracy : 97.55%
Validation Running time : 14.49 second(s)
--------------------------------------------

--------------------------------------------
Fastest Running time at : k=3, p=inf

Validation Accuracy : 82.08%
Validation Running time : 14.46 second(s)
--------------------------------------------
```

Among them, in terms of accuracy only, k=3 using euclidean distance performed the best. And in terms of running time only, k=3 using chebyshev distance performed the fastest. But Running time differs ignorablely, so I can conclude that k=3 and p=2 is the best performance.

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

Because k=3 using euclidean distance performed the best, the goal is to perform better than this. Here, although it is not a distance rather angle, I tried to use cosine similarity. According to pytorch document (https://pytorch.org/docs/stable/generated/torch.nn.functional.cosine_similarity.html), similarity is defined as:

$$similarity = \frac{x_1 \cdot x_2}{max(||x_1||_2, \epsilon) \cdot max(||x_2||_2, \epsilon)}$$

Here, $\epsilon$ is very small number, to prevent zero-division error. Similarity is real number, from -1 to 1. If two vectors are similar, it is close to 1. If exactly same, it has the value 1. If two vectors are different, it is close to -1. If exactly opposite, it is -1. Same as above, I tried 1,3,5,100,200 as k value, and this is the history.

| k value | Accuracy | Running time |
|---|---|---|
| 1 | 97.97% | 55.35 second(s) |
| 3 | 97.97% | 54.15 second(s) |
| 5 | 98.02% | 53.75 second(s) |
| 10 | 97.63% | 53.71 second(s) |
| 100 | 95.53% | 53.93 second(s) |
| 200 | 94.27% | 53.83 second(s) |

```
--------------------------------------------
Highest Accuracy at : k=5

Validation Accuracy : 98.02%
Validation Running time : 53.75 second(s)
--------------------------------------------
```

Cosine similarity performed a lot worse than norm, in terms of running time. It took about 4 times more. Instead, I could find the accuracy increased.
So it is my conclusion: If we have a lot of time, use cosine similarity. If not, or if we want to test many hyperparameter settings, use L2 norm. Anyway, here's my final test accuracy. I'll only focus on accuracy. If we need scalability, k=3 using euclidean distance will be better.

```
--------------------------------------------
Model Evaluation : k=5, Cosine Similarity

Test Accuracy : 97.22%
Test Running time : 1 minute(s) 29.96 second(s)
--------------------------------------------
```