# HW1_MNIST

September 26, 2022

## 1 CS1470/2470 HW1: KNN (with MNIST)

In this homework assignment, you will experience the overall machine learning process from start to end by implementing your own version of the **k-Nearest Neighbors** algorithm.

[1]:
```
!python -VV
```

```
Python 2.7.18
```

If you are running the notebook on Colab, you need to mount your drive or repo. An example of these is provided here.

[2]:
```
import os
import sys

## Path to data
data_path = "../data"

## Make sure the data is downloaded appropriately
![ ! -d "$data_path" ] && cd .. && bash download.sh && cd code
```

[3]:
```
%load_ext autoreload
%autoreload 1
%aimport KNN_Model, preprocess

import matplotlib.pyplot as plt
import numpy as np
```

### 1.1 Preprocessing

#### 1.1.1 Data Preparation

In a machine learning project, you need a separate train set and a test set. Sometimes, you also need a validation set to fit hyperparameters, but for this homework assignment, we are not going to use a validation set.

Code Block #1: Preprocessing

1. Load the full train and test datasets by using the function `get_data_MNIST` in `preprocess.py`.

- DO NOT shuffle the dataset. It's usually a good practice to do so, but don't do it here for the sake of simplicity. You will shuffle the dataset at the CIFAR part of the hoemwork assignment.

2. Keep only a small subset of the full datasets by using the function `get_subset` in `preprocess.py`.

- For the train set, keep only 1000 images and labels for each digit, so that your train image array should have the shape (10000, 784), and your train label array should have the shape (10000,)
- For the test set, 250 images and labels for each digit, so the shapes are (2500, 784), and (2500,).

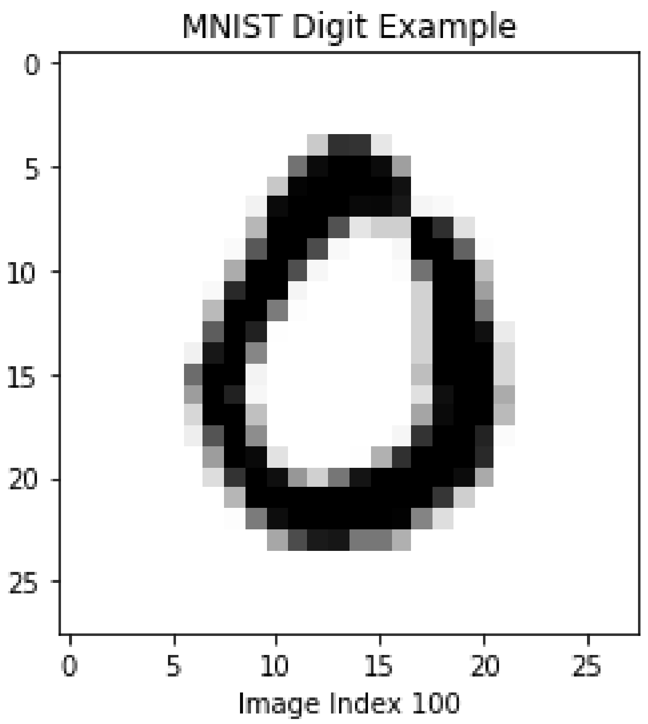
[4]:
```
from preprocess import *

digit_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

## TODO: Implement preprocessing step as described above,
## implementing preprocess.py in the process

image_train_full, label_train_full = get_data_MNIST("train")
image_test_full,  label_test_full  = get_data_MNIST("test")

image_train, label_train = get_subset(image_full=image_train_full, label_full=␣
 ↪label_train_full, num=1000)
image_test,  label_test  = get_subset(image_full=image_test_full, label_full=␣
 ↪label_test_full, num=250)
```

### 1.1.2 Data Visualization

Matplotlib's pyplot module is a good starting point to create a quick and dirty way of visually inspecting your data.

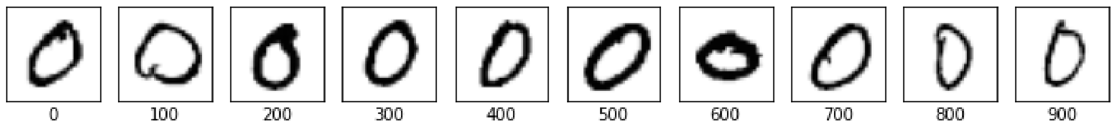
[5]:
```
plt.imshow(image_test[20].reshape(28, 28), cmap = "Greys")
plt.xlabel("Image Index 100")
plt.title("MNIST Digit Example")
```

[5]:
```
Text(0.5, 1.0, 'MNIST Digit Example')
```



However, if you want to do something more complicated, Pyplot's feature is quite limited. You probably want to use the lower-level Figure and Axis API directly.

[6]:
```
indices_to_inspect = list(range(0, 1000, 100))

fig, ax = plt.subplots(1, 10)
fig.set_size_inches(12, 1.2)

for i, each_image in enumerate(indices_to_inspect):
    ax[i].imshow(image_train[each_image].reshape(28, 28), cmap = "Greys")
    ax[i].tick_params(left=False)
    ax[i].tick_params(bottom=False)
    ax[i].tick_params(labelleft=False)
    ax[i].tick_params(labelbottom=False)
    ax[i].set_xlabel(f"{each_image}")
```



## 1.2 KNN

### 1.2.1 Model Building

Now it's time to make your own implementation of the k-Nearest Neighbors algorithm.

Code Block #2: Building the model

Create a KNN model, or an instance of the class KNN_Model and fit it with the train dataset. - Although, `k_neighbors` can be any integer in theory, keep `k_neighbors == 9` in this homework assignment. - The name of the KNN_Model instance must be `model_mnist`, so that you can run the following Code Blocks without trouble.

[7]:
```
from KNN_Model import KNN_Model

## TODO: Implement training step as described above,
## implementing model.py in the process
model_mnist = KNN_Model(digit_list, 9)
model_mnist.fit(image_train, label_train)
```

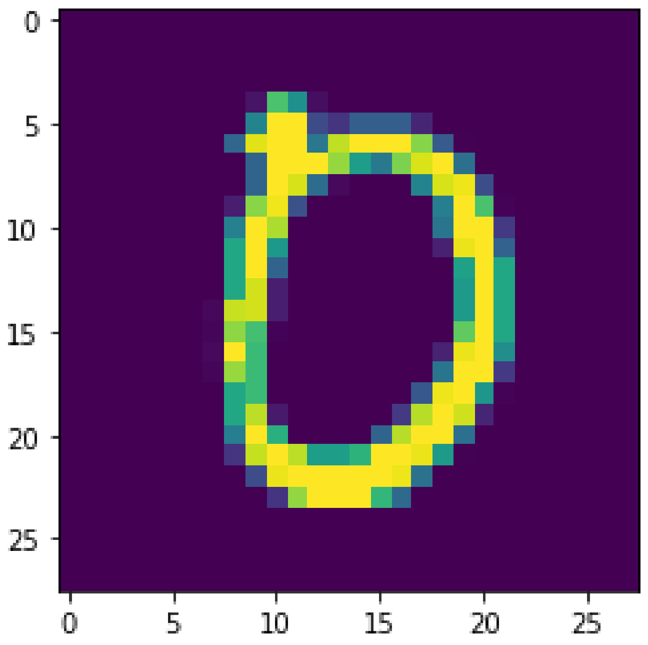
We can try the model with a sample image that the model has never seen before.

### 1.2.2 Model Visualization

Code Block #3: Interacting with the model

[8]:
```
## Pull in a specific image
sample_image = image_test[126].copy()
## TODO: Show what the image looks like using plt.imshow
plt.imshow(sample_image.reshape(28,28))
```

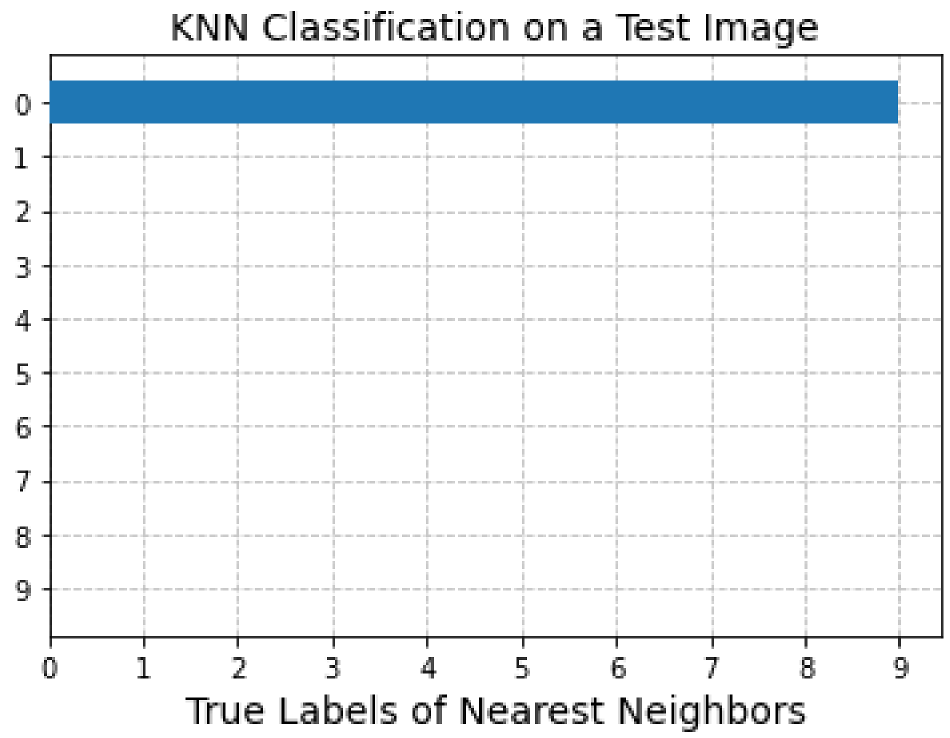
[8]:
```
<matplotlib.image.AxesImage at 0x118976d90>
```



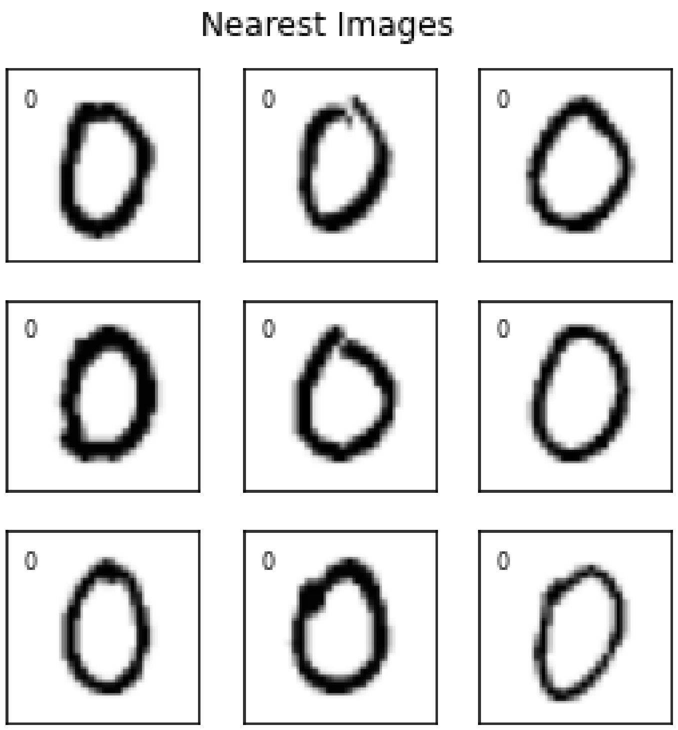
[9]:
```
## TODO: Figure out the closest k neighbors based on the model.
class_counts, nearest_indices = model_mnist.get_neighbor_counts(sample_image,␣
 ↪return_indices=True)
print(class_counts)
```

```
[9, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[10]:
```
fig_knn, ax_knn = plt.subplots()

digit_counts = model_mnist.get_neighbor_counts(sample_image)
ax_knn.barh(y=digit_list, width=digit_counts, zorder=100)
ax_knn.invert_yaxis()
ax_knn.set_yticks(digit_list)
ax_knn.set_xticks(np.arange(1 + np.max(digit_counts)))
ax_knn.set_title("KNN Classification on a Test Image", fontsize=14)
ax_knn.set_xlabel("True Labels of Nearest Neighbors", fontsize=14)
ax_knn.grid(linestyle="dashed", color="#bfbfbf", zorder=-100)
fig_knn.set_size_inches([6, 4])

# You can also save the figure in the pdf, png, and svg formats
# fig.savefig(f"KNN_Test_Image_MNIST.png", dpi=300, bbox_inches="tight")
```



[11]:
```
fig_nearest, ax_nearest = plt.subplots(3, 3, figsize=(4.5, 4.5))

for each_ax, each_neighbor in zip(ax_nearest.flat, nearest_indices):
    each_ax.imshow(model_mnist.image_train[each_neighbor].reshape(28, 28),␣
 ↪cmap="Greys")
    each_ax.tick_params(bottom=False, left=False, labelbottom=False,␣
 ↪labelleft=False)
    each_ax.text(2, 5, model_mnist.label_train[each_neighbor],
        fontsize=8, bbox = dict(color="White", alpha=0.75))
    fig_nearest.suptitle("Nearest Images", y = 0.95)
```



### 1.3 Evaluation

It is time to evaluate the model.

#### 1.3.1 Overall Accuracy

Code Block #4: Overall accuracy

1. Get predictions on every image in the test dataset.
2. Calculate and print out the overall accuracy of the model, which is defined as the number of correct predictions divided by the number of all predictions.

[12]:
```
## TODO: Get predictions on test dataset and calculate accuracy
prediction_array = model_mnist.get_prediction_array(image_test)

count = 0
for x, y in zip(prediction_array, label_test):
    if x == y:
        count += 1
prediction_acc = count/len(prediction_array)
print(f"accuracy = {prediction_acc}")
```

```
2410
2420
2430
2440
2450
2460
2470
2480
2490
accuracy = 0.9148
```

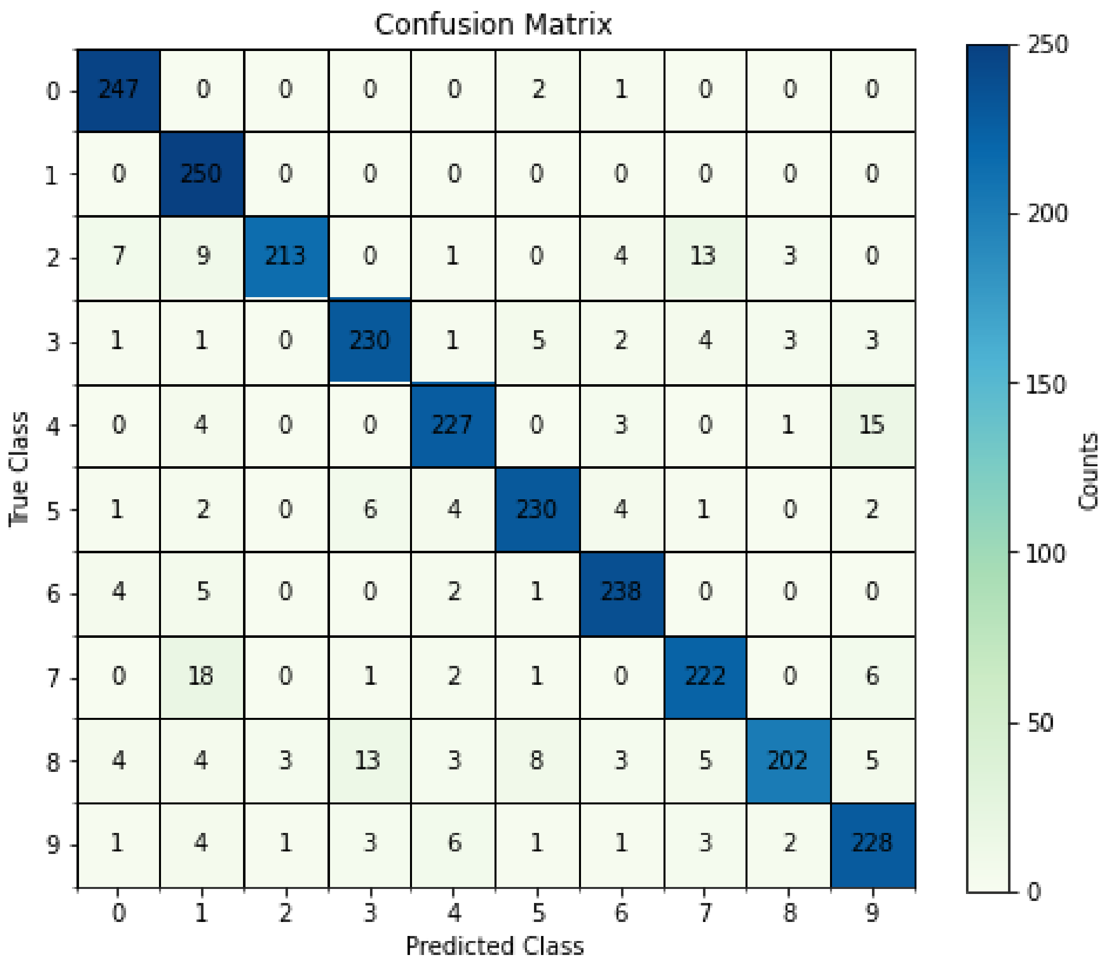
### 1.3.2 Confusion Matrix

Code Block #5: Confusion matrix

[13]:
```python
## TODO: Get the confusion matrix (hint: see KNN_ConfMtx)
confusion_mat = model_mnist.get_confusion_matrix(label_test, prediction_array)
print(confusion_mat)
```

```
[[247   0   0   0   0   2   1   0   0   0]
 [  0 250   0   0   0   0   0   0   0   0]
 [  7   9 213   0   1   0   4  13   3   0]
 [  1   1   0 230   1   5   2   4   3   3]
 [  0   4   0   0 227   0   3   0   1  15]
 [  1   2   0   6   4 230   4   1   0   2]
 [  4   5   0   0   2   1 238   0   0   0]
 [  0  18   0   1   2   1   0 222   0   6]
 [  4   4   3  13   3   8   3   5 202   5]
 [  1   4   1   3   6   1   1   3   2 228]]
```

[14]:
```python
fig_confusion, ax_confusion = model_mnist.
 ↪visualize_confusion_matrix(confusion_mat)
```



| True \ Predicted | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 247 | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 1 | 0 | 250 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 7 | 9 | 213 | 0 | 1 | 0 | 4 | 13 | 3 | 0 |
| 3 | 1 | 1 | 0 | 230 | 1 | 5 | 2 | 4 | 3 | 3 |
| 4 | 0 | 4 | 0 | 0 | 227 | 0 | 3 | 0 | 1 | 15 |
| 5 | 1 | 2 | 0 | 6 | 4 | 230 | 4 | 1 | 0 | 2 |
| 6 | 4 | 5 | 0 | 0 | 2 | 1 | 238 | 0 | 0 | 0 |
| 7 | 0 | 18 | 0 | 1 | 2 | 1 | 0 | 222 | 0 | 6 |
| 8 | 4 | 4 | 3 | 13 | 3 | 8 | 3 | 5 | 202 | 5 |
| 9 | 1 | 4 | 1 | 3 | 6 | 1 | 1 | 3 | 2 | 228 |