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.

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

```
Python 3.9.12 (main, Apr  5 2022, 01:52:34)
[Clang 12.0.0 ]
```

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

In [2]:
```
import os
import sys

## Path to data
data_path = "data"
#data_path ="/Users/kylejung/Documents/GitHub/hw1-knn-JungKyle/hw1/code/data

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

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

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

Preprocessing

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,).

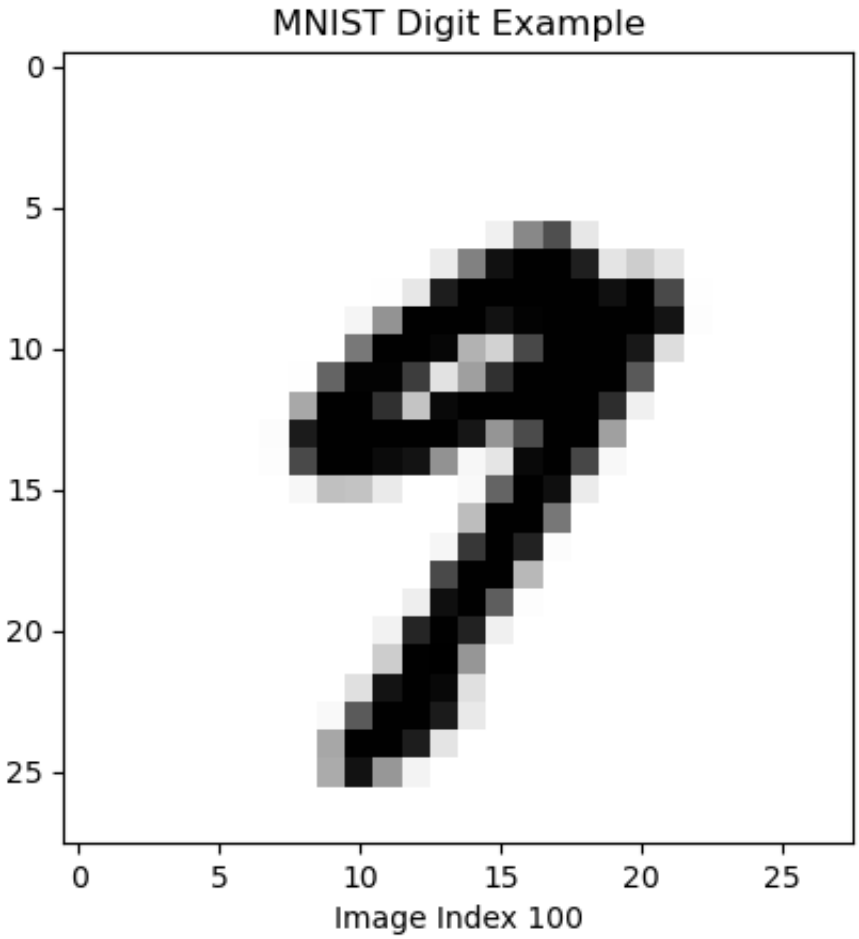
In [8]:
```
from preprocess import *

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", data_path)
image_test_full,  label_test_full  = get_data_MNIST("test", data_path)

image_train, label_train = image_train_full[:10000*784].reshape(10000,784),
image_test,  label_test  = image_test_full[:2500*784].reshape(2500,784), lab
```

Data Visualization

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

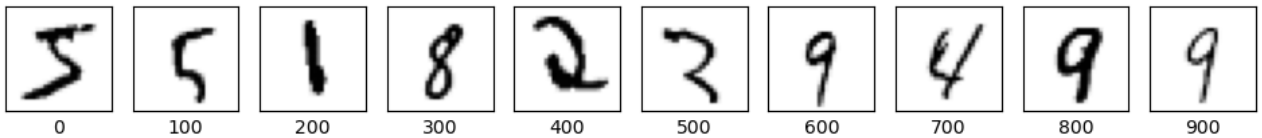
In [5]:
```
plt.imshow(image_test[20].reshape(28, 28), cmap = "Greys")
plt.xlabel("Image Index 100")
plt.title("MNIST Digit Example")
```

Out[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.

In [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}")
```



KNN

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.

In [43]:
```
from KNN_Model import KNN_Model

## TODO: Implement training step as described above,
## implementing model.py in the process
k_neighbors=9
class_list=np.array([0,1,2,3,4,5,6,7,8,9])
model_mnist = KNN_Model(class_list,k_neighbors)
model_mnist.fit(image_train,label_train)
```

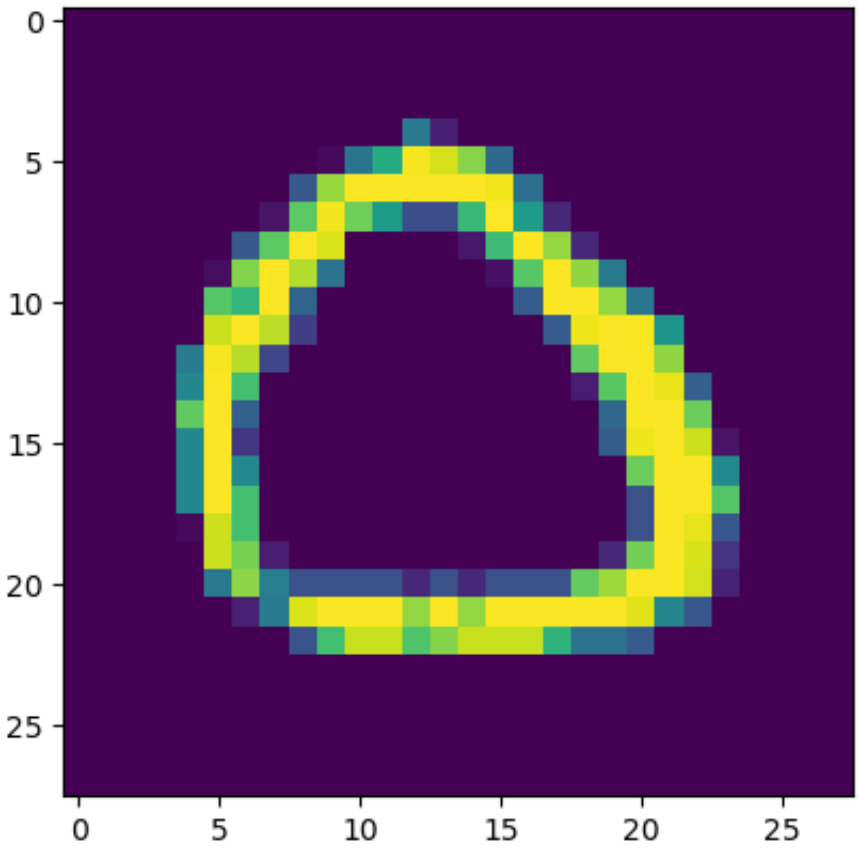
We can try the model with a sample image that the model has never seen before.

Model Visualization

Code Block #3: Interacting with the model

In [26]:
```
## 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))
```

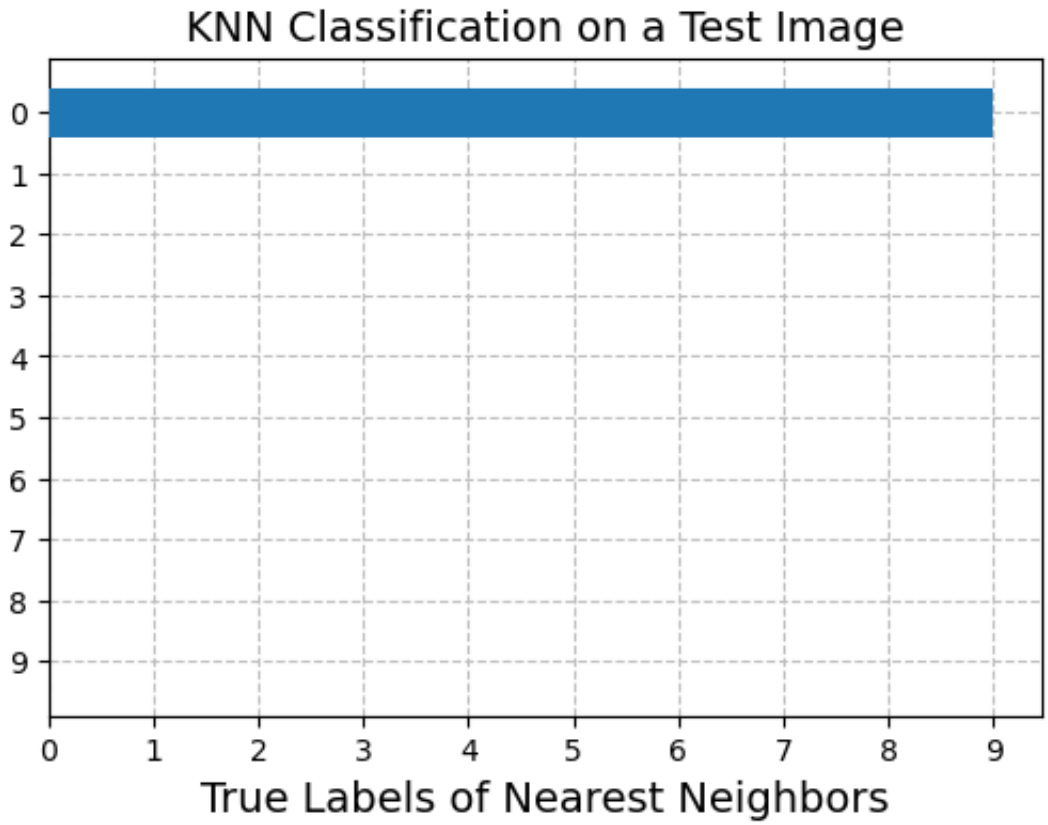
Out[26]:
```
<matplotlib.image.AxesImage at 0x13f629790>
```



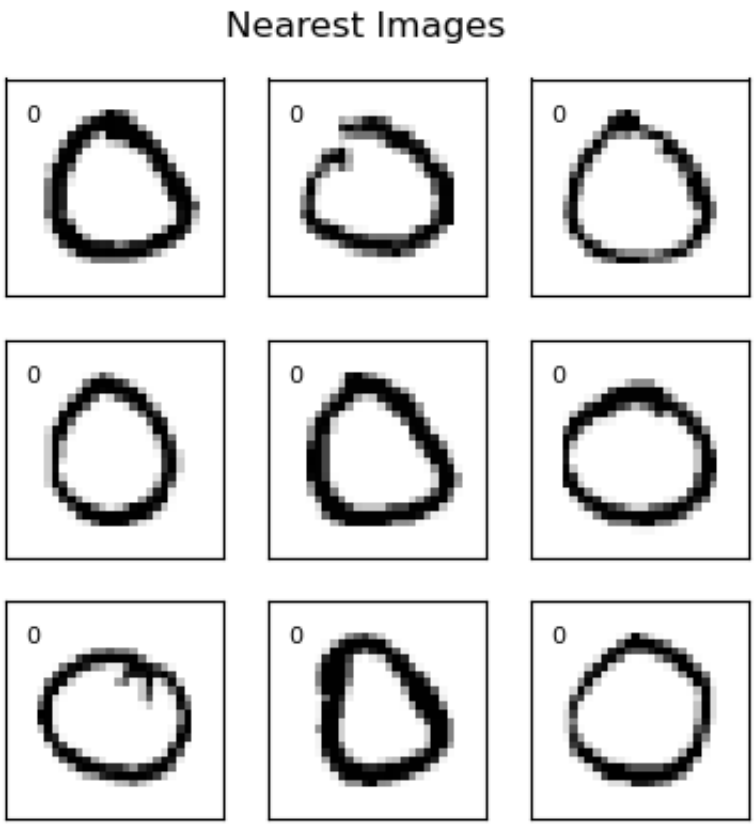
In [52]:
```
## TODO: Figure out the closest k neighbors based on the model.

class_counts, nearest_indices = model_mnist.get_neighbor_counts(sample_image

print(class_counts)
print(nearest_indices)
```

```
[9 0 0 0 0 0 0 0 0 0]
[4849 3906 6406 2340 7052 8812 9986 4686 7050]
```

In [61]:
```
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")
```



In [62]:
```
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), c
    each_ax.tick_params(bottom=False, left=False, labelbottom=False, labelle
    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)
```



Evaluation

It is time to evaluate the model.

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.

In [72]:
```
## TODO: Get predictions on test dataset and calculate accuracy


prediction_array=[]
Correct_Count=0
for i in range(label_test.shape[0]):
    prediction_array.append(model_mnist.predict(image_test[i]))
    if prediction_array[i]==label_test[i]:
        Correct_Count+=1


prediction_acc = Correct_Count/label_test.shape[0]

print(f"accuracy = {prediction_acc}")
```

```
accuracy = 0.912
```

Confusion Matrix

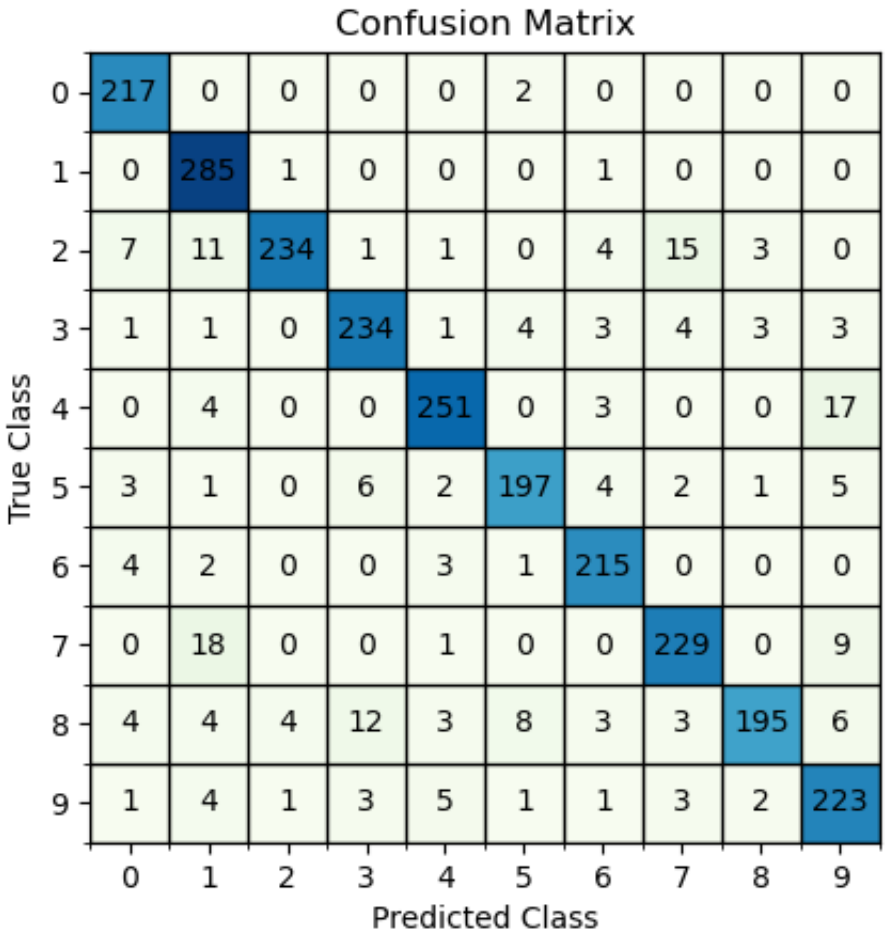
Code Block #5: Confusion matrix

In [73]:
```
## TODO: Get the confusion matrix (hint: see KNN_ConfMtx)
confusion_mat = model_mnist.get_confusion_matrix(label_test, prediction_arra
print(confusion_mat)
```

```
[[217   0   0   0   0   2   0   0   0   0]
 [  0 285   1   0   0   0   1   0   0   0]
 [  7  11 234   1   1   0   4  15   3   0]
 [  1   1   0 234   1   4   3   4   3   3]
 [  0   4   0   0 251   0   3   0   0  17]
 [  3   1   0   6   2 197   4   2   1   5]
 [  4   2   0   0   3   1 215   0   0   0]
 [  0  18   0   0   1   0   0 229   0   9]
 [  4   4   4  12   3   8   3   3 195   6]
 [  1   4   1   3   5   1   1   3   2 223]]
```

In [75]:
```
fig_confusion, ax_confusion = model_mnist.visualize_confusion_matrix(confusi

#I did not try to fix this error since there was no To Do in the KNN_ConfMtx
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [75], line 1
----> 1 fig_confusion, ax_confusion = model_mnist.visualize_confusion_matrix
(confusion_mat)

File ~/Documents/GitHub/hw1-knn-JungKyle/hw1/code/KNN_ConfMtx.py:72, in KNN_
ConfMtx.visualize_confusion_matrix(self, confusion_mat)
     64 for each_true_index, each_pred_index \
     65     in itertools.product(range(num_classes), range(num_classes)):
     66     ax_confusion.text(each_pred_index,
     67                       each_true_index,
     68                       confusion_mat[each_true_index, each_pred_index
],
     69                       ha = "center",
     70                       va = "center")
---> 72 fig_confusion.colorbar(
     73     mpl.cm.ScalarMappable(norm=norm, cmap=cmap),
     74     orientation="vertical",
     75     label="Counts",
     76     shrink = 0.83)
     78 fig_confusion.set_size_inches([8, 8])
     80 return fig_confusion, ax_confusion

File /opt/miniconda3/envs/tensorflow/lib/python3.9/site-packages/matplotlib/
figure.py:1256, in FigureBase.colorbar(self, mappable, cax, ax, use_gridspec
, **kwargs)
   1254 if cax is None:
   1255     if ax is None:
-> 1256         raise ValueError(
   1257             'Unable to determine Axes to steal space for Colorbar. '
   1258             'Either provide the *cax* argument to use as the Axes fo
r '
   1259             'the Colorbar, provide the *ax* argument to steal space
'
   1260             'from it, or add *mappable* to an Axes.')
   1261     current_ax = self.gca()
   1262     userax = False

ValueError: Unable to determine Axes to steal space for Colorbar. Either pro
vide the *cax* argument to use as the Axes for the Colorbar, provide the *ax
* argument to steal space from it, or add *mappable* to an Axes.
```


Confusion Matrix
True Class
Predicted Class
0 217 0 0 0 0 2 0 0 0 0
1 0 285 1 0 0 0 1 0 0 0
2 7 11 234 1 1 0 4 15 3 0
3 1 1 0 234 1 4 3 4 3 3
4 0 4 0 0 251 0 3 0 0 17
5 3 1 0 6 2 197 4 2 1 5
6 4 2 0 0 3 1 215 0 0 0
7 0 18 0 0 1 0 0 229 0 9
8 4 4 4 12 3 8 3 3 195 6
9 1 4 1 3 5 1 1 3 2 223
0 1 2 3 4 5 6 7 8 9