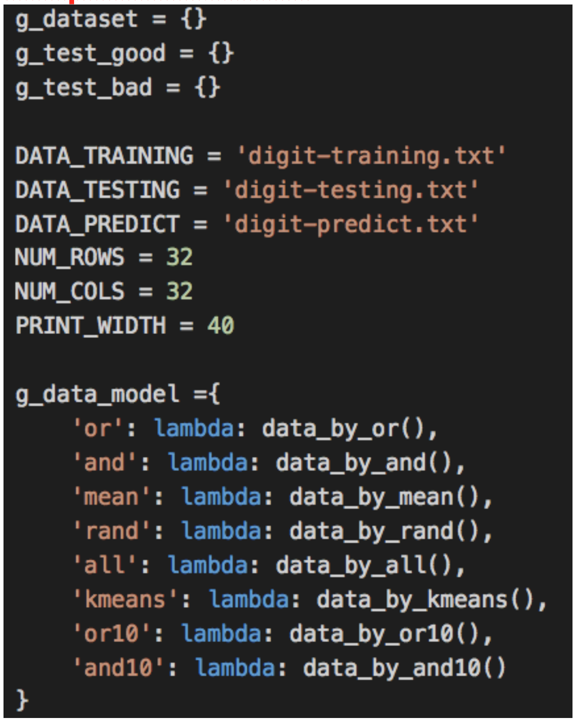
Assignment 3 Design file 俞佳含 116010274

Scope of Variables



```
g_dataset = {}
g_test_good = {}
g_test_bad = {}

DATA_TRAINING = 'digit-training.txt'
DATA_TESTING = 'digit-testing.txt'
DATA_PREDICT = 'digit-predict.txt'
NUM_ROWS = 32
NUM_COLS = 32
PRINT_WIDTH = 40

g_data_model ={
    'or': lambda: data_by_or(),
    'and': lambda: data_by_and(),
    'mean': lambda: data_by_mean(),
    'rand': lambda: data_by_rand(),
    'all': lambda: data_by_all(),
    'kmeans': lambda: data_by_kmeans(),
    'or10': lambda: data_by_or10(),
    'and10': lambda: data_by_and10()
}
```

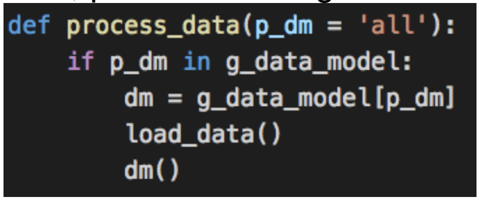
The above variables are global variables. Otherwise, they are local variables.

Goal

Let machine learn by process the file of “digit_traininging.txt”. Then calculate the accuracy to see if the training works well by processing the file “digit_testing”. Finally, give predictions of some unknown digit figures.

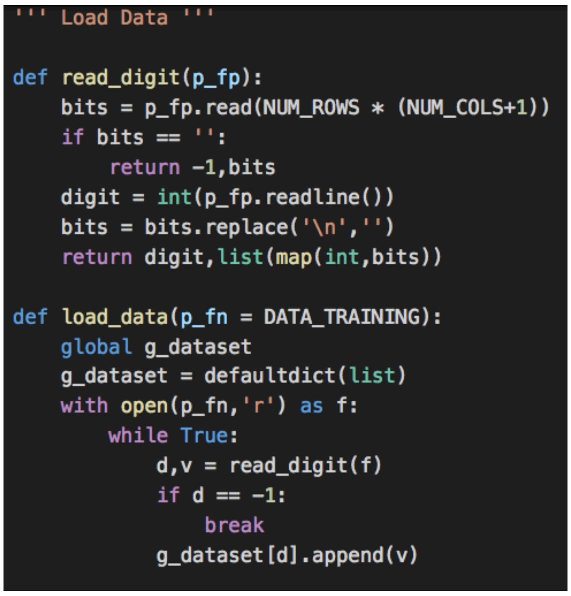
Idea Stream of solving this problem

First, process training data using the data model the user wants.



```
def process_data(p_dm = 'all'):
    if p_dm in g_data_model:
        dm = g_data_model[p_dm]
        load_data()
        dm()
```

In this part, first load data from the text and convert it into a dictionary. That means, keys are those digits (0,1,2,3…) while under each digit there are many multidimensional vectors (converted from the figure made by 0 and 1)
More specificly:

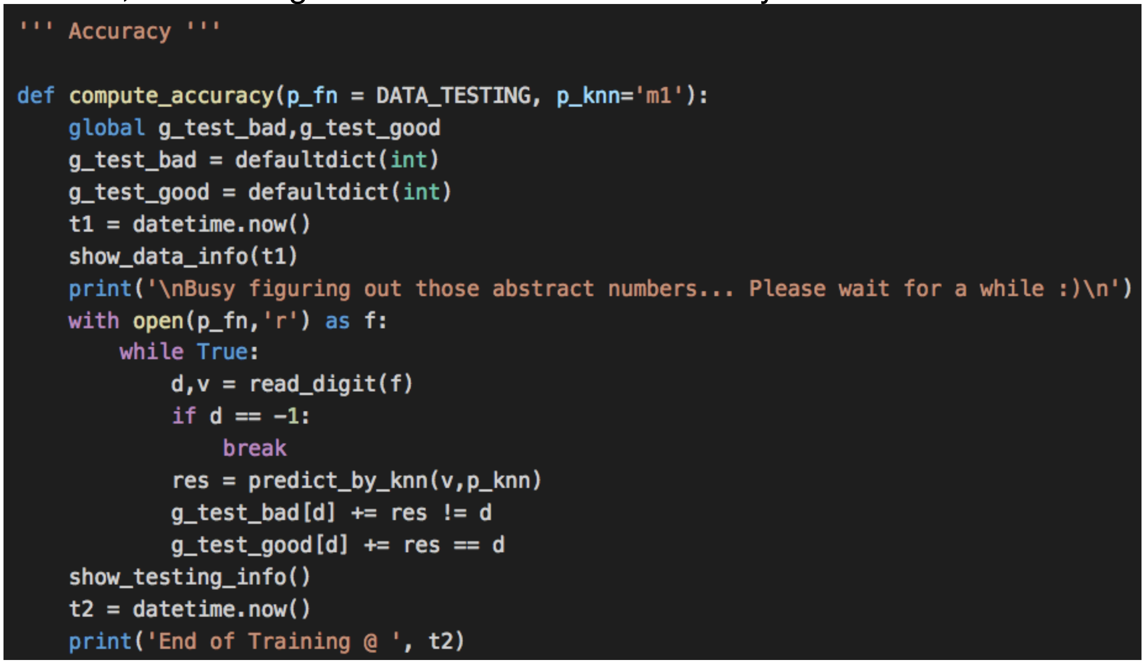


```python
''' Load Data '''

def read_digit(p_fp):
    bits = p_fp.read(NUM_ROWS * (NUM_COLS+1))
    if bits == '':
        return -1,bits
    digit = int(p_fp.readline())
    bits = bits.replace('\n','')
    return digit,list(map(int,bits))

def load_data(p_fn = DATA_TRAINING):
    global g_dataset
    g_dataset = defaultdict(list)
    with open(p_fn,'r') as f:
        while True:
            d,v = read_digit(f)
            if d == -1:
                break
            g_dataset[d].append(v)
```

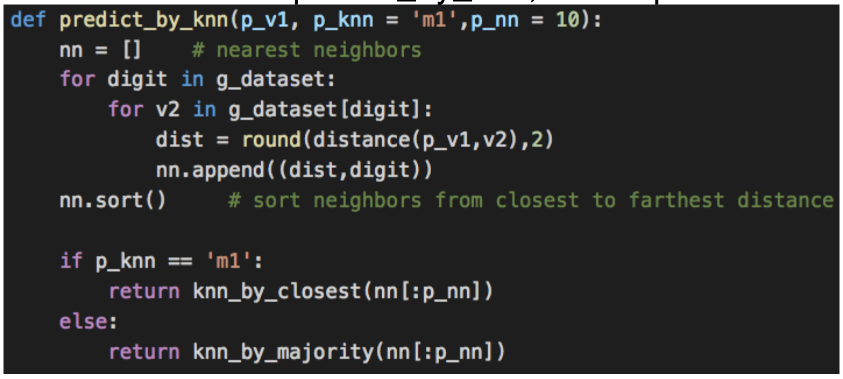
Second, use testing data to calculate the accuracy of a certain data model.



```python
''' Accuracy '''

def compute_accuracy(p_fn = DATA_TESTING, p_knn='m1'):
    global g_test_bad,g_test_good
    g_test_bad = defaultdict(int)
    g_test_good = defaultdict(int)
    t1 = datetime.now()
    show_data_info(t1)
    print('\nBusy figuring out those abstract numbers... Please wait for a while :)\n')
    with open(p_fn,'r') as f:
        while True:
            d,v = read_digit(f)
            if d == -1:
                break
            res = predict_by_knn(v,p_knn)
            g_test_bad[d] += res != d
            g_test_good[d] += res == d
    show_testing_info()
    t2 = datetime.now()
    print('End of Training @ ', t2)
```

By predict_by_knn function to get the result of the testing digit. If it equals to the digit, then the test is good. If not, then bad. Append the test outcome into lists because they will be useful in calculating the accuracy. Meanwhile, record the processing time.
As for the function predict_by_knn, more specific is here:

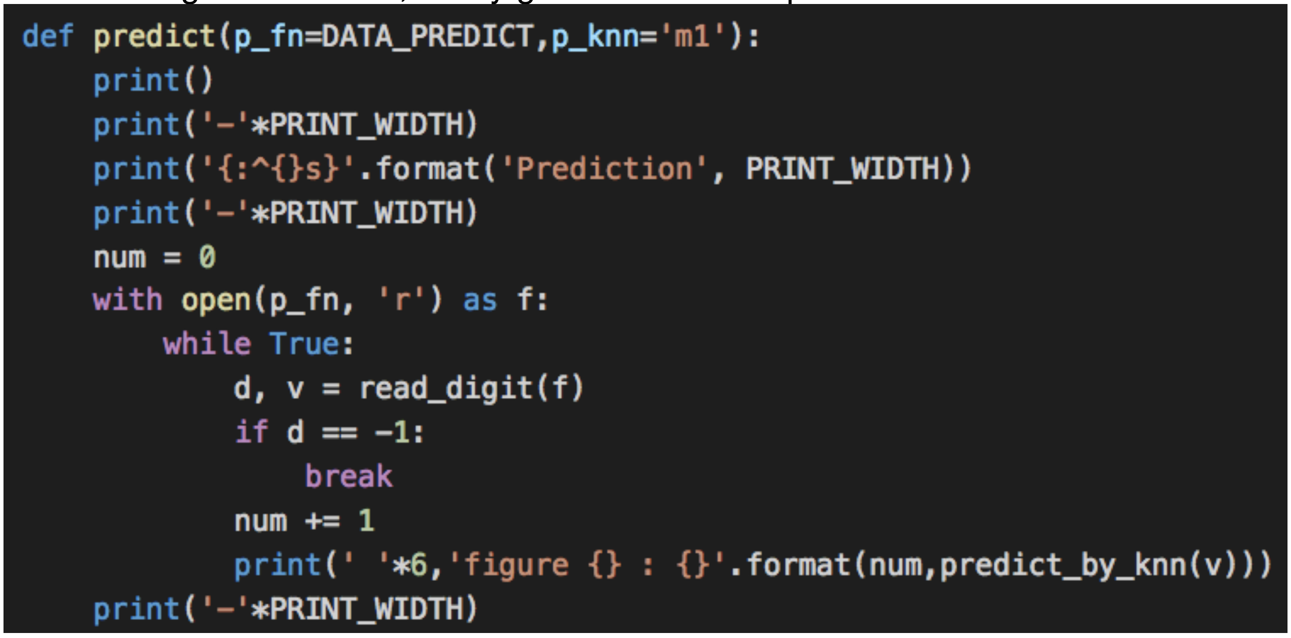


```python
def predict_by_knn(p_v1, p_knn = 'm1',p_nn = 10):
    nn = []     # nearest neighbors
    for digit in g_dataset:
        for v2 in g_dataset[digit]:
            dist = round(distance(p_v1,v2),2)
            nn.append((dist,digit))
    nn.sort()     # sort neighbors from closest to farthest distance

    if p_knn == 'm1':
        return knn_by_closest(nn[:p_nn])
    else:
        return knn_by_majority(nn[:p_nn])
```

As for the explanation of closest and majority, closest means just get the closest neighbor while majority means get the digit that appears most frequently.

After training the machine, finally get work with the prediction.

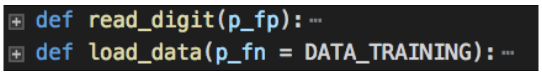


```python
def predict(p_fn=DATA_PREDICT,p_knn='m1'):
    print()
    print('-'*PRINT_WIDTH)
    print('{:^{}s}'.format('Prediction', PRINT_WIDTH))
    print('-'*PRINT_WIDTH)
    num = 0
    with open(p_fn, 'r') as f:
        while True:
            d, v = read_digit(f)
            if d == -1:
                break
            num += 1
            print(' '*6,'figure {} : {}'.format(num,predict_by_knn(v)))
    print('-'*PRINT_WIDTH)
```

Similarly, it processes data in digit_predict.txt firstly. Then compare the multidimensional vector with its dataset (generated when training). At last, make a conclusion of what the figure is.

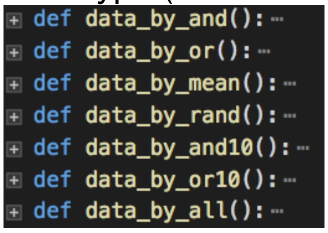
Structure

Load data



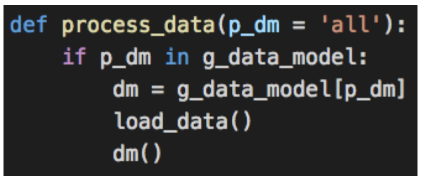
```python
def read_digit(p_fp):...
def load_data(p_fn = DATA_TRAINING):...
```

Data type (and how to deal with a certain data type)



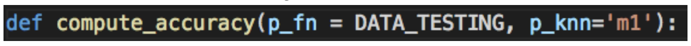
```python
def data_by_and():...
def data_by_or():...
def data_by_mean():...
def data_by_rand():...
def data_by_and10():...
def data_by_or10():...
def data_by_all():...
```

Process data



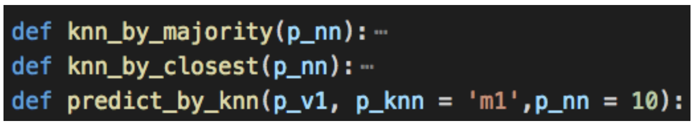
```python
def process_data(p_dm = 'all'):
    if p_dm in g_data_model:
        dm = g_data_model[p_dm]
        load_data()
        dm()
```

Calculate Accuracy



```python
def compute_accuracy(p_fn = DATA_TESTING, p_knn='m1'):
```

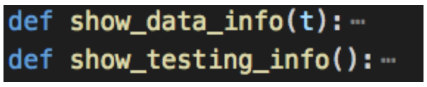
KNN models



```python
def knn_by_majority(p_nn):...
def knn_by_closest(p_nn):...
def predict_by_knn(p_v1, p_knn = 'm1',p_nn = 10):
```

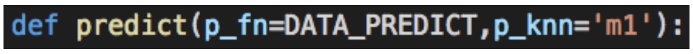
Assignment 3 Design file 俞佳含 116010274

Information Format



```
def show_data_info(t):⋯
def show_testing_info():⋯
```

Prediction



```
def predict(p_fn=DATA_PREDICT,p_knn='m1'):
```