VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

DATA STRUCTURES AND ALGORITHMS - CO2003

ASSIGNMENT 1

PROCESSING MNIST DATASET AND IMPLEMENTING *k*NN ALGORITHM USING LIST

Ho Chi Minh City, 02/2024

ASSIGNMENT'S SPECIFICATION

Version 1.0

1 Assignment's outcome

After completing this assignment, students review and make good use of:

- Object Oriented Programming (OOP)
- List data structures
- Sorting algorithms

2 Introduction

Classification is one of the fundamental and most common tasks in machine learning. The goal of classification is to categorize data into different groups (classes) based on their characteristics. Some simple examples of classification problems include classifying emails as spam or non-spam, classifying images as cat or dog, and so on. In this assignment, students will become acquainted with the simple classification algorithm k-nearest neighbors (kNN), and apply this algorithm to process and classify the MNIST dataset.

Students are required to implement a class that provides users with the ability to process the MNIST dataset, as well as a class that efficiently implements the kNN classification algorithm using list data structures.

3 Description

3.1 MNIST Dataset

The MNIST dataset is one of the most popular datasets commonly used for testing and evaluating machine learning and computer vision algorithms. The name "MNIST" stands for "Modified National Institute of Standards and Technology". This dataset contains a set of images captured from handwritten samples of digits from 0 to 9, with each image having a size of 28x28 pixels. Some examples of images in the dataset are depicted in Figure 1.

Below are some important characteristics of the MNIST dataset:

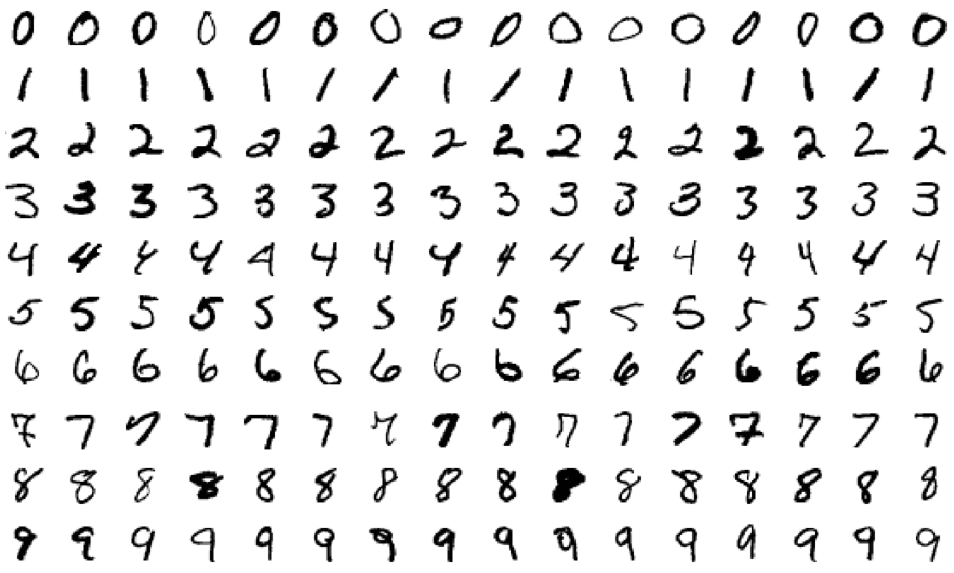

Figure 1: Examples of data samples in the MNIST dataset

- Number of samples: The MNIST dataset consists of 60,000 images in the training set and 10,000 images in the test set. However, to reduce computational burden, the number of data used for training and testing will be reduced. The maximum number of images used will be announced later.
- Data type: Each image in the MNIST dataset is encoded as pixel values, where the value of each pixel ranges from 0 to 255, representing the brightness of the pixel.
- Classification: Each image represents one of the digits from 0 to 9. The goal is to build a model capable of accurately classifying the digits in the images.

Students are provided with the file ***mnist.csv***, where:

- Each row describes an image of a digit, encoded into 28x28 = 714 columns and one label column.
- The label column assigns a label to the digit represented by the image. For each image, the value range of the cells in this column is from 0 to 9.
- The remaining columns, labeled ixj where i, j are within the range [1,28], represent the coordinates of the pixel. For each image, the value range of the cells in this column is from 0 to 255 (the darkness of the pixel).

3.2 k-Nearest Neighbors Algorithm

The kNN algorithm is a simple yet effective classification algorithm, based on finding the k nearest data points to the new data point needing classification.

3.2.1 Idea

To grasp the idea of the kNN algorithm, let's solve a small puzzle as shown in Figure 2. Bob's store has 4 types of fruits: Apple, Orange, Grape, and Strawberry. Knowing that Bob's store displays fruits as shown in the figure:

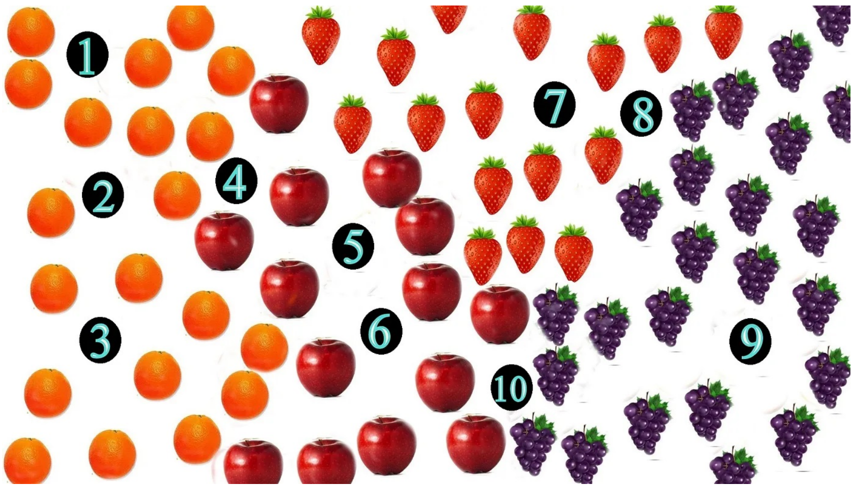

Figure 2: Fruits in Bob's store

Your task is to predict what fruits labeled from 1 to 10 will be. We can infer with our naked eyes as follows:

- 1, 2, 3 → Orange
- 4 → Unsure whether it's Orange or Apple
- 5, 6 → Apple
- 7 → Strawberry
- 8 → Unsure whether it's Grape or Strawberry
- 9 → Grape
- 10 → Unsure whether it's Apple or Grape

If your predictions match the ones above, congratulations! You've just used the kNN algorithm! Why is it so? Let's delve deeper into how you made those predictions.

In the image, we observe that similar types of fruits are arranged together. For 1, 2, and 3, we can easily classify them as Orange because they are closely surrounded by Orange without any other types, hence it's highly likely that the hidden fruits underneath could also be Orange. In other words, the hidden fruits will mostly be of the same type as the majority of their neighbors. The same reasoning applies to 5, 6, 7, and 9.

For 10, we're unsure whether it's an Apple or a Grape. This is because it's surrounded by both Apples and Grapes. Or we can say that the neighbors of 10 belong to both Apple and Grape. So 10 could be either an Apple or a Grape. Similarly for 4 and 8.

In summary, the kNN algorithm predicts the label for a new data point based on the labels of its neighbors. kNN relies on the assumption that similar data points will be close to each other in the coordinate space. In the example above, based on the labels (Apple, Orange, Strawberry, Grape) of the neighbors, we can predict the label for a new data point (hidden fruit).

3.2.2 Parameter k

The parameter k in kNN is the number of nearest neighbors we consider when making a prediction. We determine the proximity of a point based on its distance (e.g., Euclidean, Manhattan, etc.) to the points being considered. For example, if $k = 5$, we consider the closest 5 points and take the majority label among these 5 points as the predicted label.

To illustrate further, consider how we determine the neighbors in Bob's store as shown in Figure 3. Our goal is to predict the label for the point marked as X. If $k = 3$, among the 3 neighbors of X, 2 points are Strawberries and 1 point is an Orange. So we predict the label for X is Strawberry. If $k = 5$, among the 5 neighbors of X, 3 points are Oranges and 2 points are Strawberries. So we predict the label for X is Orange.

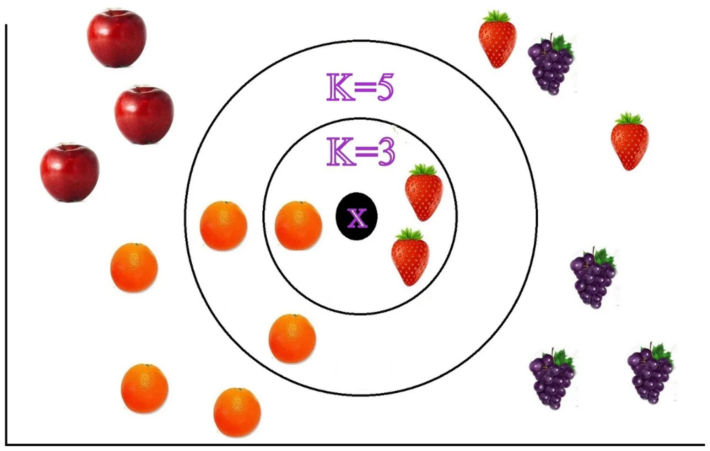

Figure 3: Determining the label of a fruit based on the k parameter

3.2.3 Algorithm

We can create a kNN classification model by following these steps:

1. Load the data.
2. Initialize the value of k.
3. To get the predicted class, iterate from 1 to the total number of training data points.
4. Calculate the distance between the test data and each row of the training data. Here, we will use the Euclidean distance as the method for measuring distance as it's the most common method.
5. Sort the calculated distances in ascending order based on the distance value.
6. Take the top k rows from the sorted array.
7. Take the most common class of these rows.
8. Return the predicted class.

3.3 Euclidean Distance

Euclidean distance is a method for measuring the distance between two points in multidimensional space. It is calculated by taking the square root of the sum of the squares of the differences between the coordinates of the two points.

In this assignment, to compute the distance between images, let $x = \{x_1, x_2, ..., x_k, ..., x_n\}$ be the vector containing the features of image 1 with n features (or the number of columns excluding the label in the data table), and $y = \{y_1, y_2, ..., y_k, ..., y_n\}$ be the vector containing the features of image 2 with n features. The distance between the two images is calculated using the formula:

$$distance = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2}$$

3.4 Training and Prediction Process

In classification problems, the training and prediction processes are two crucial stages in building and deploying prediction tools. During the training process, the tool is "trained" by using training data to learn relationships within the data. This process typically involves splitting the data into features and labels. In this assignment, information about pixels will be the features, and the label will be the digit represented by the image.

After dividing the data into features and labels, we usually proceed to split the data into two subsets: the training set and the test set.

- X_train and y_train: These are sets of features (X_train) and corresponding labels (y_train) used to train the model. The model will learn from the data in the training set to make more accurate predictions on new data.
- X_test and y_test: These are sets of features (X_test) and corresponding labels (y_test) used to test the performance of the model after training. The data in the test set is used to evaluate whether the model can generalize well to new data.

In summary, the tool learns from X_train and y_train. Then, it proceeds to predict on X_test. The predicted results are compared with y_test to evaluate the effectiveness of the prediction. Students can refer to sample test cases to see the training and prediction process clearly.

4 Requirements

In this assignment, students are required to implement the following classes to describe the data processing process and implement the k-nearest neighbor algorithm using the list structure, including: class List, class Dataset, and class kNN.

4.1 class List

The class List is a class used to store data in list structure and provide users with tools to manipulate lists. Note that the List class is a template class and only serves as an interface. Students should inherit this class into another class and override all pure virtual methods listed below. Students are encouraged to refer to the topics of Inheritance, Polymorphism, and Templates in C++.

The description for the List class is as follows:

```cpp
template<typename T>
class List {
public:
    virtual ~List() = default;
    virtual void push_back(T value) = 0;
    virtual void push_front(T value) = 0;
    virtual void insert(int index, T value) = 0;
    virtual void remove(int index) = 0;
    virtual T& get(int index) const = 0;
    virtual int length() const = 0 ;
    virtual void clear() = 0;
    virtual void print() const = 0;
    virtual void reverse() = 0;
};
```

Below is detailed information about the methods in the List class:

1. `virtual ~List() = default;`
   - Virtual destructor for the List class.
2. `virtual void push_back(T value) = 0;`
   - Add an element with value `value` to the end of the list.
   - Exceptions: None.
3. `virtual void push_front(T value) = 0;`
   - Add an element with value `value` to the beginning of the list.
   - Exceptions: None.
4. `virtual void insert(int index, T value) = 0;`
   - Add an element with value `value` at position `index` in the list. If `index` == current size of the list (from now on called `size`), add the element to the end of the list.
   - Exceptions: If `index` < 0 or `index` > `size`, the method does nothing.
5. `virtual void remove(int index) = 0;`
   - Remove the element at position `index` from the list.
   - Exceptions: If `index` < 0 or `index` >= `size`, the method does nothing.
6. `virtual T& get(int index) const = 0;`
   - Access the element at position `index` of the list. Returns a reference to that element.
   - Exceptions: If `index` < 0 or `index` >= `size`, execute the statement `throw std::out_of_range(`
     `Out of range")`.
7. `virtual int length() const = 0;`
   - Returns the current length of the list.
   - Exceptions: None.
8. `virtual void clear() = 0;`
   - Clear all elements in the list, resetting the list to its initial state.

- Exceptions: None.

9. `virtual void print() const = 0;`
   - Print out the elements in the list in the format: elements separated by a space, no trailing space at the end of the line.
   - Exceptions: None.

10. `virtual void reverse() = 0;`
    - Reverse the elements in the list in place.
    - Exceptions: None.

4.2 class Dataset

The Dataset class is a class used to store and process tabular data, specifically in this assignment is the MNIST dataset. The description for the Dataset class is as follows:

```cpp
class Dataset {
private:
    List<List<int>*>* data;
public:
    Dataset();
    ~Dataset();
    Dataset(const Dataset& other);
    Dataset& operator=(const Dataset& other);
    bool loadFromCSV(const char* fileName);
    void printHead(int nRows = 5, int nCols = 5) const;
    void printTail(int nRows = 5, int nCols = 5) const;
    void getShape(int& nRows, int& nCols) const;
    void columns() const;
    bool drop(int axis = 0, int index = 0, std::string columns = "");
    Dataset extract(int startRow = 0, int endRow = -1, int startCol = 0, int
     endCol = -1) const;
    List<List<int>*>* getData() const;
};
```

Below is detailed information about the data members and methods in the Dataset class:

1. `List<List<int>*>* data;`
   - This is the main attribute to store data in tabular form.

2. `Dataset();`

- Default constructor.

3. `~Dataset();`
   - Destructor.
4. `Dataset(const Dataset& other);`
   - Copy constructor.
5. `Dataset& operator=(const Dataset& other);`
   - Assignment operator.
6. `bool loadFromCSV(const char* fileName);`
   - This method is used to load data from the file `fileName`, specifically in this assignment is the `mnist.csv` file. The information in the file will be stored in the variable `data` and other proposed variables by students.
   - It returns true if the data loading is successful, otherwise false.
7. `void printHead(int nRows = 5, int nCols = 5) const;`
   - Print the first `nRows` rows, and only print the first `nCols` columns of the data table.
   - Print format:
     - The first row prints the names of the columns of the data table.
     - From the second row onwards, print the value of each cell in the table, each element separated by a space, with no trailing space at the end of the line.
   - Exceptions: If `nRows` is greater than the number of rows in the data table, print all rows in the data table. If `nCols` is greater than the number of columns in the data table, print all columns in the data table. If `nRows` or `nCols` is less than 0, do not print anything.

Example 4.1

The output of `printHead` with `nRows` = 5 and `nCols` = 5, using data from the file `mnist.csv`, is as follows:

```
label 1x1 1x2 1x3 1x4
5 0 0 0 0
0 0 0 0 0
4 0 0 0 0
1 0 0 0 0
9 0 0 0 0
```

8. `void printTail(int nRows = 5, int nCols = 5) const;`

- Print the last `nRows` rows, and only print the last `nCols` columns of the data table.
- Print format:
  - The first row prints the names of the columns of the data table.
  - From the second row onwards, print the value of each cell in the table, each element separated by a space, with no trailing space at the end of the line.
- Exceptions: If `nRows` is greater than the number of rows in the data table, print all rows in the data table. If `nCols` is greater than the number of columns in the data table, print all columns in the data table. If `nRows` or `nCols` is less than 0, do not print anything.

Example 4.2

The output of `printTail` with `nRows` = 5 and `nCols` = 5, using data from the file `mnist.csv`, is as follows:

```
28x24 28x25 28x26 28x27 28x28
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

9. `void getShape(int& nRows, int& nCols) const;`
   - This method returns the current total number of rows and columns of the object through two reference parameters `nRows` and `nCols`.
10. `void columns() const;`
    - Print the names of all columns of the data table. Each name is separated by a space, with no trailing space at the end of the line.
11. `bool Dataset::drop(int axis = 0, int index = 0, std::string columnName = "");`
    - Delete a row or column of the data table.
    - If `axis` is not 0 or 1, the function does nothing and returns false.
    - If `axis` == 0, delete a row at position `index`, then return true. If index >= number of rows or < 0, the function does nothing and returns false.
    - If `axis` == 1, delete a column with the name matching `columnName`, then return true. If `columnName` is not in the list of column names, the function does nothing and returns false.

12. `Dataset extract(int startRow = 0, int endRow = -1, int startCol = 0, int endCol = -1) const;`
    - This method is used to extract a part of the data table, then return the extracted data table.
    - Where: `startRow` is the starting row, `endRow` is the ending row, `startCol` is the starting column, `endCol` is the ending column.
    - If `endRow` == -1, we will take all rows. Similarly with `endCol` == -1.
    - Testcases will ensure that startRow, endRow, startCol, and endCol are within valid range (from 0 to number of rows/number of columns) and start $\leq$ end.
13. `List<List<int>> getData() const;`
    - Method used to access the member variable `data`.

Since data manipulation will be performed multiple times, can you implement insertion/deletion operations on the list with O(1) complexity?

4.3 class kNN

Class kNN is used to implement the functionalities of the prediction model using the k-nearest neighbor algorithm as described in the above sections. The design for class kNN will be as follows:

```cpp
class kNN {
private:
    int k;
public:
    kNN(int k = 5);
    void fit(const Dataset& X_train, const Dataset& y_train);
    Dataset predict(const Dataset& X_test);
    double score(const Dataset& y_test, const Dataset& y_pred);
};
```

Below is detailed information about the fields and methods in the kNN class:

1. `int k;`
    - The value of k, as described in the sections above.
2. `void fit(const Dataset& X_train, const Dataset& y_train);`
    - Performs loading of training data, where:

- X_train: The dataset contains features that are not labels. The number of rows of X_train is the number of images used for training, and the number of columns is the number of features.
- y_train: The dataset contains labels corresponding to the features. Since these are labels, the number of rows of y_train is the number of images, and the number of columns is 1.

3. `Dataset predict(const Dataset& X_test);`

- Executes the prediction based on the kNN algorithm.
- X_test is the dataset containing features of the images to be predicted. The number of rows of X_test is the number of images used for prediction, and the number of columns is the number of features.
- Note that the kNN algorithm described in the above sections is only applied to predict one new image. Students need to implement prediction for multiple new images corresponding to the number of rows in the X_test dataset.
- The return result is a dataset with multiple rows and 1 column. In each cell of the column, there is a label (class) predicted by the kNN algorithm.
- Since accessing the dataset occurs frequently. Is there a way to implement access operations with O(1) complexity?

4. `double score(const Dataset& y_test, const Dataset& y_pred);`

- Measures the accuracy of the prediction process.
- y_test is the dataset containing the actual labels of the images to be predicted. The number of rows of y_test is the number of images used for prediction, and the number of columns is 1.
- y_pred is the dataset containing the predicted labels by the kNN algorithm (obtained from the predict function). The number of rows of y_pred is the number of images used for prediction, and the number of columns is 1.
- The return result is a real number calculated as follows:

$$\text{Accuracy} = \frac{\text{Number of correctly predicted images}}{\text{Total number of predicted images}}$$

Where the Number of correctly predicted images is the number of images with predicted labels matching the actual labels.

4.4 The train_test_split Function

This is a utility function used to split the dataset into smaller datasets for training and prediction purposes. The prototype of the function is as follows:

```
void train_test_split(Dataset& X, Dataset& y, double test_size, Dataset&
    X_train, Dataset& X_test, Dataset& y_train, Dataset& y_test);
```

Where:

- Input:
  - Dataset& X: The dataset containing the features of the images. The number of rows is the number of images, and the number of columns is the number of features.
  - Dataset& y: The dataset containing the labels of the images. The number of rows is the number of images, and the number of columns is 1.
  - double test_size: A real number determining the ratio of the test set to the total dataset. The remainder will be the train set.
- Output:
  - Dataset& X_train: The training dataset containing the features of the images. The number of rows is the number of images in the training set, and the number of columns is the number of features.
  - Dataset& y_train: The training dataset containing the labels of the images. The number of rows is the number of images in the training set, and the number of columns is 1.
  - Dataset& X_test: The test dataset containing the features of the images. The number of rows is the number of images in the test set, and the number of columns is the number of features.
  - Dataset& y_test: The test dataset containing the actual labels of the images. The number of rows is the number of images in the test set, and the number of columns is 1.

Example 4.3

Considering the dimensions of X and y are 199x784 and 199x1 respectively, with a test_size of 0.2. We obtain the sizes of the data tables after splitting as follows:

- X_train: 159x784
- X_test: 40x784
- y_train: 159x1
- y_test: 40x1

4.5 Instructions

To complete this assignment, students must:

1. Read through this entire description file.
2. Download the initial.zip file and extract it. After extraction, students will receive the following files: main.cpp, main.hpp, kNN.hpp, kNN.cpp, and the mnist.csv file. Students are only required to submit the kNN.hpp and kNN.cpp files, so they should not modify the main.h file when running the program.
3. Students should use the following command to compile:
   g++ -o main main.cpp kNN.cpp -I . -std=c++11
   This command is used in the command prompt/terminal to compile the program. If students use an IDE to run the program, they need to note the following: add all necessary files to the IDE's project/workspace; change the IDE's compilation command accordingly. IDEs typically provide buttons for compiling (Build) and running the program (Run). When pressing Build, the IDE will execute a corresponding compilation command, usually only compiling main.cpp. Students need to configure the IDE to change the compilation command: add the kNN.cpp file, add the options -std=c++11, -I .
4. The program will be graded on the UNIX platform. The platform and compiler used by students may differ from the actual grading environment. The submission platform on BKeL is set up similarly to the actual grading environment. Students must run the program on the submission platform and fix any errors that occur on BKeL to ensure correct results during actual grading.
5. Modify the kNN.hpp and kNN.cpp files to complete this assignment and ensure the following requirements are met:
   - All methods described in this specification must be implemented so that compilation

is successful. If students cannot implement a method, provide an empty implementation for that method. Each testcase will call some of the described methods to check the return results.

- There should be only 1 **include** statement in the kNN.hpp file, which is **#include "main.hpp"**, and one include in the kNN.cpp file, which is **#include "kNN.hpp"**. Additionally, no other **#include** statements are allowed in these files.

6. The main.cpp file provides some simple testcases in functions with the format "**tc<x>()**".
7. Students are allowed to write additional methods and attributes within the required classes. Students are also permitted to write additional classes beyond the required implementations.
8. Students are required to design and use data structures based on the types of lists they have learned.
9. Students must deallocate all dynamically allocated memory when the program ends.

5 Submission

Students are only required to submit 2 files: kNN.hpp and kNN.cpp, before the deadline specified in the "Assignment 1 - Submission" path. There are some simple testcases used to check the students' work to ensure that the results can be compiled and run. Students can submit their work as many times as they want, but only the last submission will be graded. Because the system cannot handle the load when too many students submit their work at the same time, students should submit their work as early as possible. Students will bear the risk if they submit their work close to the deadline. Once the deadline for submission has passed, the system will be closed, and students will not be able to submit anymore. Submissions through other methods will not be accepted.

6 Other regulations

- Students must complete this assignment on their own and must prevent others from stealing their results. Failure to do so will result in disciplinary action according to the university's cheating policy.
- Any decision made by the teachers in charge of this assignment is the final decision.
- Students are not provided with testcases after grading, but are only provided with information on testcase design strategies and distribution of the correct number of students

according to each test case.

- Assignment contents will be harmonized with a question in exam with similar content.

7 Changelog

- Add the getData() function in the Dataset class (implementation required).
- Correct minor errors in the description of the train_test_split function, and add an example for this function.

——————————END——————————