# ***coGoals***

* To learn about object-oriented programming concepts.
* To learn about machine learning.
* To write a client program that classifies images.

# ***Before you begin***

**Getting started.** Before you begin coding, do the following:

* Read Sections 3.1 and 3.2 of the textbook to learn the basics of object-oriented programming and how to use the [Picture](https://introcs.cs.princeton.edu/java/11cheatsheet/#Picture) and [Color](https://introcs.cs.princeton.edu/java/11cheatsheet/#Color) data types.
* *Download the project files -*  [classifier.zip](https://drive.google.com/file/d/1PawPOK5I1RRxR1VE77bHQ7X8HooWTTCv/view?usp=sharing)

# ***Background***

Classification is one of the central problems in *machine learning* (ML), a discipline that is transforming 21st century computing. As a familiar example, consider the problem of classifying handwritten digits using this [web application](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/handwriting.html):

[](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/handwriting.html)

Machine learning algorithms like this are widely used to classify handwritten digits (e.g., to recognize postal ZIP codes, process bank checks, and parse income tax forms).

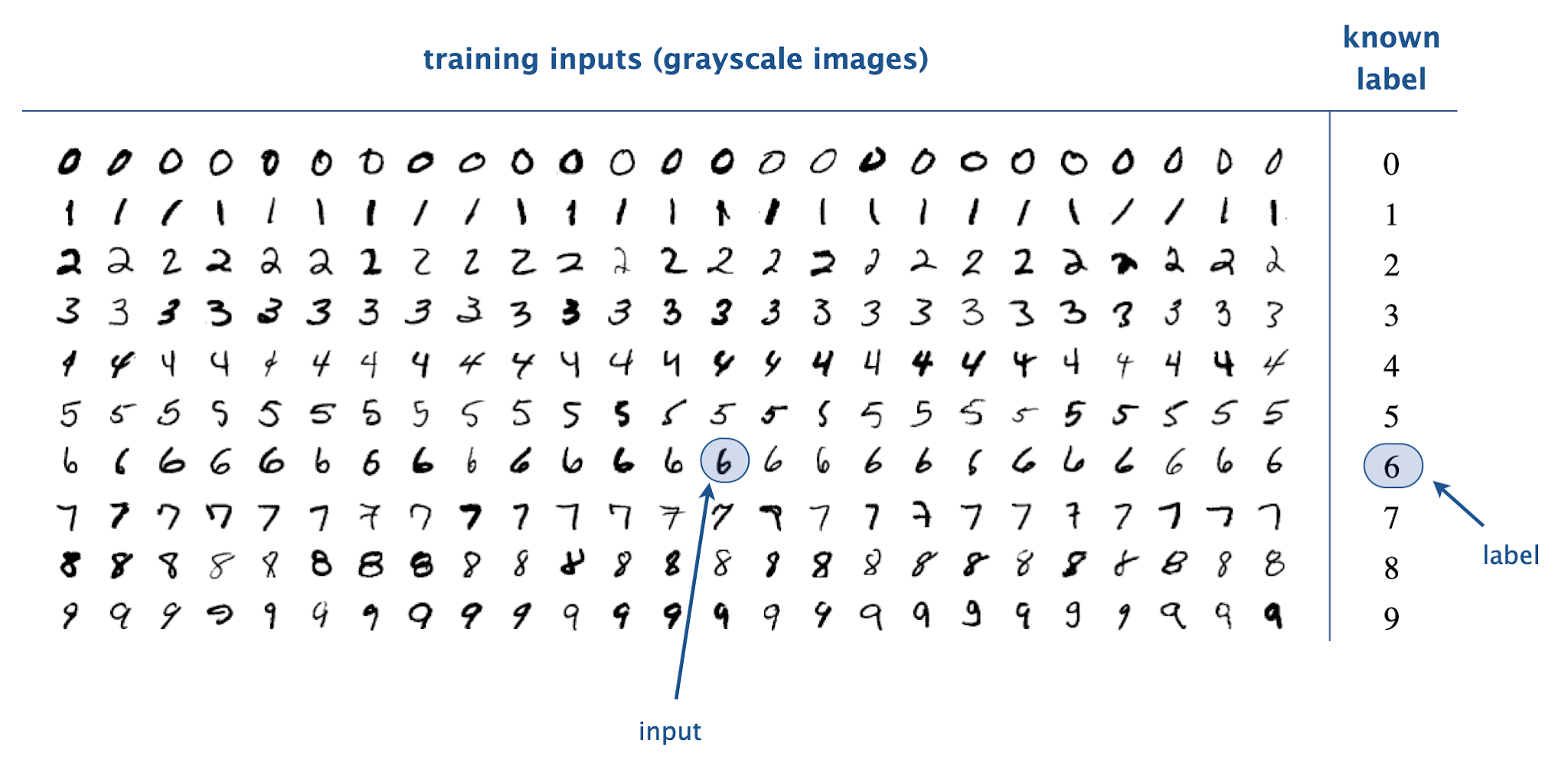
The full power of machine learning derives from its amazing versatility. Machine learning algorithms rely upon data to learn to make predictions, without being explicitly programmed for the task. For example, the same code you will write to classify handwritten digits extends to classifying other types of images, simply by training the algorithm with different data.

Moreover, machine learning techniques apply not only to images but also to numerical, text, audio, and video data. Modern applications of ML span science, engineering, and commerce: from autonomous vehicles, medical diagnostics, and video surveillance to product recommendations, voice recognition, and language translation.

# ***Approach***

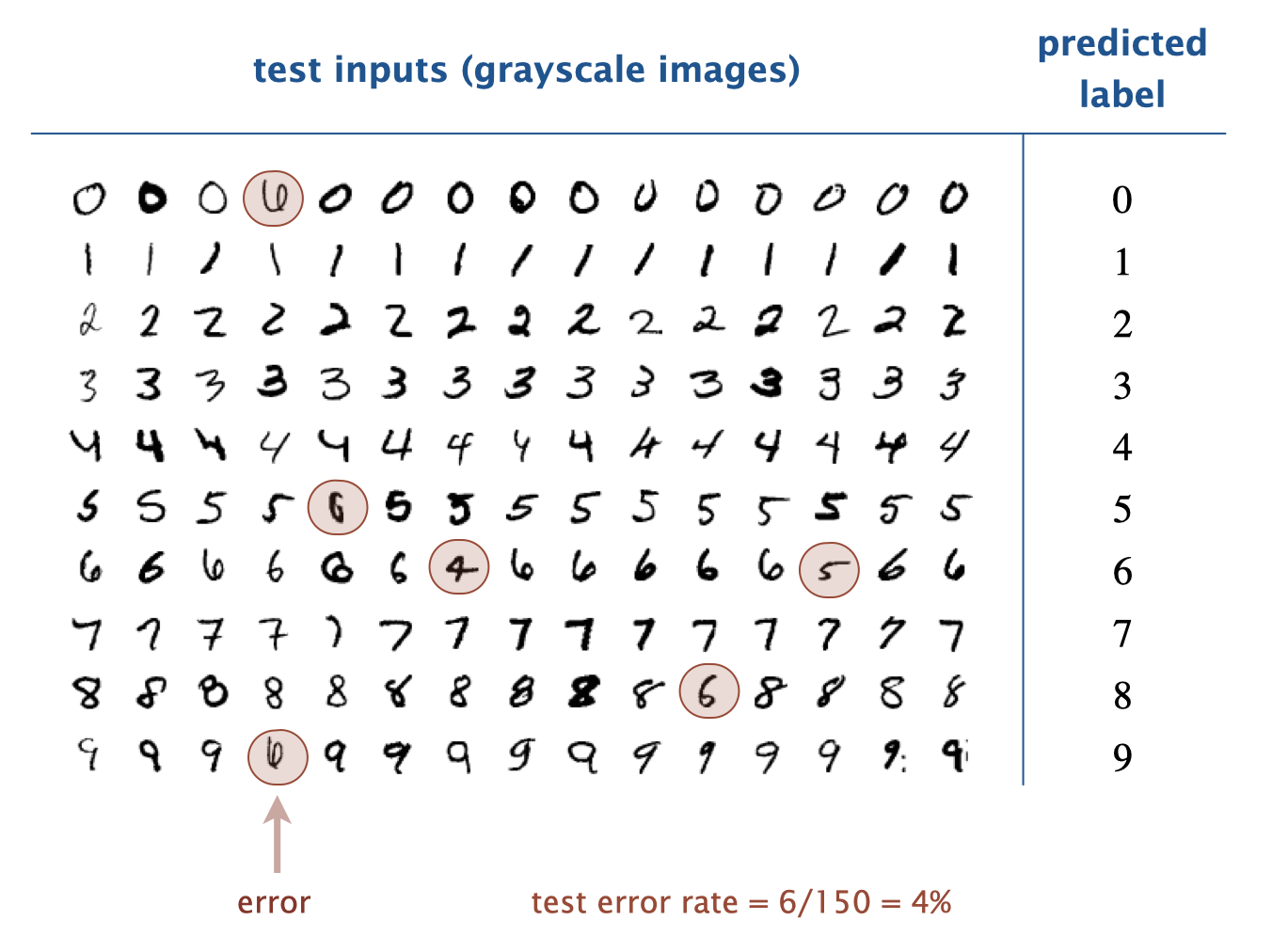
**Supervised learning.** To classify images, we will use a supervised learning algorithm. Supervised learning is divided into two phases — ***training*** and ***testing***.

1. In the ***training phase***, the algorithm *learns* a function that maps an input to an output (or a label) using training data consisting of known input–output pairs. For the handwritten digit application, the training data comprise 60,000 grayscale images (inputs) and associated digits (labels). Here is a small subset:



In the *multiclass classification problem*, we seek to classify images into one of classes, labeled . For our handwritten digit application, there are classes, with class corresponding to digit .

1. In the ***testing phase***, the algorithm uses the learned function to predict labels for unseen inputs.



Typically, the algorithm makes some prediction errors (e.g., predicts 9 when the handwritten digit is 6). An important quality metric is the *test error rate*—the fraction of testing inputs that the algorithm misclassifies. It measures how well the learning algorithm generalizes from the training data to new data.

# 

# ***Implementation Tasks***

## **Overall Requirements**

* In Part 1 of this assignment, we provide a ML library - MultiPerceptron - for classifying vectors of *n* real numbers into one of *m* classes. In Part 2 (next week), you will implement this ML library using *perceptrons*.
* Your task is to write an image classifier client, named ImageClassifier.java, that classifies images using the MultiPerceptron data type.
* Note that you will **not** implement MultiPerceptron.java until Part 2 of this assignment. The focus of Part 1 is on **using** data types, whereas the focus of Part 2 is on **creating** data types. For now, you must use (the pre-compiled) MultiPerceptron.class that is included in the project folder.
* You should comment all instance variables.
* As usual, your code must conform to the stated API. Violations include:
  + Adding public methods. You are free to add private methods (which are accessible only in the class in which they are declared).
  + Producing undocumented side effects, such as reading from standard input in main() or altering the Picture argument to extractFeatures. See the ImageClassifier API below.

### **MultiPerceptron data type.**

This ML library is widely applicable: you can use it not only to classify images but also to classify numerical, text, and audio data. The main idea is to extract features from the data, representing each training and testing input as a vector of real numbers. For our handwritten digit application, each input is a 28-by-28 grayscale image and the features are the n = 28 × 28 = 784 grayscale values that constitute the image.

Here is the MultiPerceptron API that you will use:

| **public class MultiPerceptron {**  // Creates a multi-perceptron object with m classes  // and size of the feature vector n.  **public MultiPerceptron(int m, int n)**  // Returns the number of classes m.  **public int numberOfClasses()**  // Returns the number of inputs n, the size of the feature vector  **public int numberOfInputs()**  // Returns the predicted label for the given input.  **public int predictMulti(double[] x)**  // Trains this multi-perceptron on the labeled input.  **public void trainMulti(double[] x, int label)**  **}** |
| --- |

### **ImageClassifier.java client.**

Your task is to write a client program ImageClassifier.java that classifies images using the MultiPerceptron data type described in the previous section by:

* Training it using the input–output pairs specified in a training data file.
* Testing the predictions using the input–output pairs specified in a testing data file.
* Printing a list of misclassified images and the test error rate.

Organize your client according to the following API:

| **public class ImageClassifier {**  // Creates a feature vector (1D array) from the given picture.  **public static double[] extractFeatures(Picture picture)**  // See below.  **public static void main(String[] args)**  **}** |
| --- |
|  |

Here are some details about the API:

1. *Input file format*. A training data file consists of a sequence of lines:
   1. the first line contains the number of classes ;
   2. the second line contains the width and height, respectively, of the images;
   3. each remaining line contains the name of an image file (e.g., corresponding to a handwritten digit) followed by an integer label (e.g., identifying the correct digit), separated by whitespace.

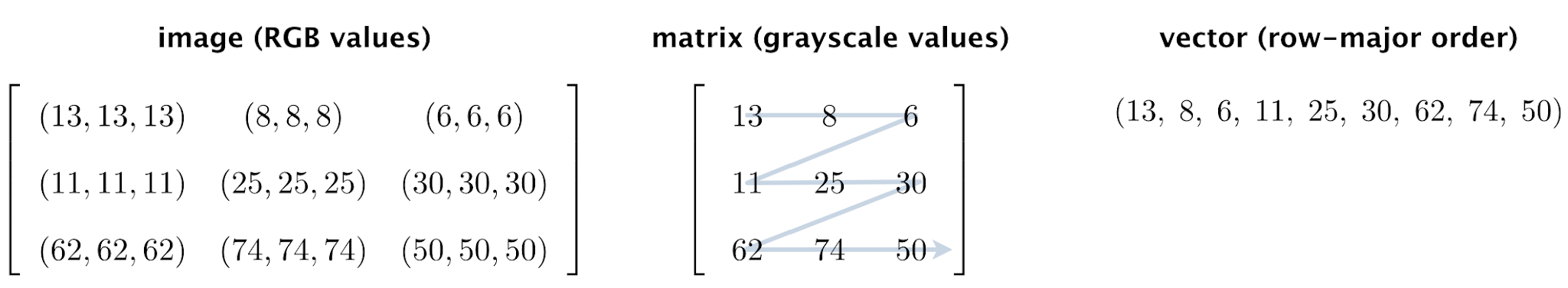


The file format for testing data files is the same (but you will use the integer labels in the testing data files only to check the accuracy of your predictions).

1. *Input files*. We provide a variety of sample input files in the specified format, including handwritten digits, fashion articles from Zalando, Hiragana characters, and doodles of fruit, animals, and musical instruments.

| **Data files** | **Description** | **Examples** | **Source** |
| --- | --- | --- | --- |
| [digits.jar](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/digits.jar)  [digits-training60K.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/digits-training60K.txt)  [digits-testing10K.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/digits-testing10K.txt) | *handwritten*  *digits* |  | [MNIST](http://yann.lecun.com/exdb/mnist/) |
| [fashion.jar](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/fashion.jar)  [fashion-training60K.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/fashion-training60K.txt)  [fashion-testing10K.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/fashion-testing10K.txt) | *fashion*  *articles* |  | [Fashion MNIST](https://github.com/zalandoresearch/fashion-mnist) |
| [Kuzushiji.jar](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/Kuzushiji.jar)  [Kuzushiji-training60K.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/Kuzushiji-training60K.txt)  [Kuzushiji-testing10K.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/Kuzushiji-testing10K.txt) | *Hiragana*  *characters* |  | [Kuzushiji MNIST](https://github.com/rois-codh/kmnist) |
| [fruit.jar](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/fruit.jar)  [fruit-training30K.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/fruit-training30K.txt)  [fruit-testing6K.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/fruit-testing6K.txt) | *fruit*  *doodles* |  | [Quick, Draw!](https://quickdraw.withgoogle.com/data) |
| [animals.jar](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/animals.jar)  [animals-training60K.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/animals-training60K.txt)  [animals-testing12K.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/animals-testing12K.txt) | *animal*  *doodles* |  | [Quick, Draw!](https://quickdraw.withgoogle.com/data) |
| [music.jar](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/music.jar)  [music-training50K.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/music-training50K.txt)  [music-testing10K.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/music-testing10K.txt) | *musical instrument*  *doodles* |  | [Quick, Draw!](https://quickdraw.withgoogle.com/data) |

1. *Feature extraction.* The extractFeatures() method converts a grayscale image into a one-dimensional array suitable for use with the MultiPerceptron data type. This method must perform two conversions:
   1. Convert the width-by-height image into a width-by-height matrix of grayscale values. Recall that a shade of gray has its red, green, and blue components all equal.
   2. Convert the width-by-height matrix of grayscale values into a one-dimensional array of length width × height by iterating over the matrix elements in row-major order.



1. *Main.*  The main() method takes two command-line arguments:
   1. The name of a file that contains the training data.
   2. The name of a file that contains the testing data.

It creates a MultiPerceptron object with *m* classes and *n = width × height* inputs; trains it using the images and labels from the training data file; and classifies images from the testing data file, producing as output the following information:

* 1. A list of misclassified images, one per line.
  2. The test error rate (the fraction of test images that the algorithm misclassified).

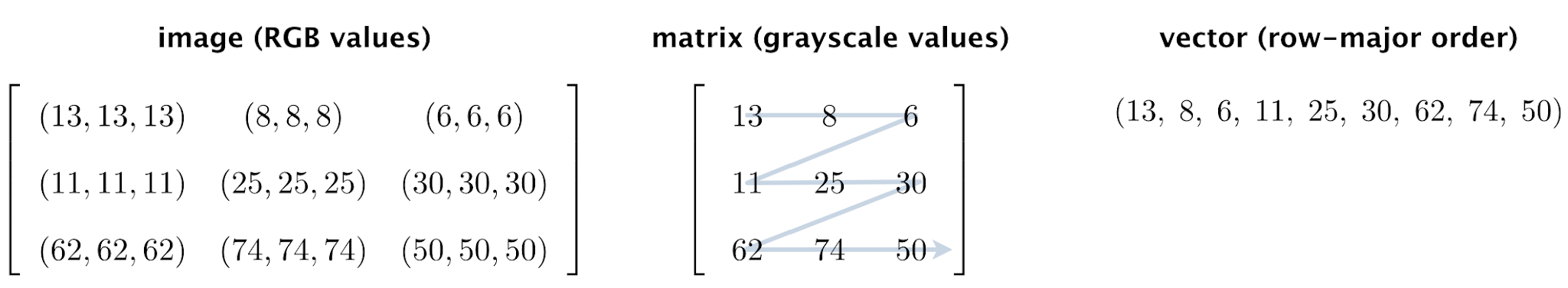
Here are some sample executions:

| % **java-introcs ImageClassifier digits-training20.txt digits-testing10.txt**  digits/testing/1/46.png  digits/testing/7/36.png  digits/testing/7/80.png  digits/testing/1/40.png  digits/testing/1/39.png  digits/testing/7/79.png  digits/testing/9/20.png  digits/testing/9/58.png  test error rate = 0.8  % **java-introcs ImageClassifier digits-training60K.txt digits-testing10K.txt**  jar:file:digits.jar!/testing/5/9428.png  jar:file:digits.jar!/testing/6/4814.png  jar:file:digits.jar!/testing/5/4915.png  ...  jar:file:digits.jar!/testing/5/7870.png  jar:file:digits.jar!/testing/4/1751.png  jar:file:digits.jar!/testing/5/6043.png  test error rate = 0.136  % **java-introcs ImageClassifier fashion-training60K.txt fashion-testing10K.txt**  ...  % **java-introcs ImageClassifier Kuzushiji-training60K.txt Kuzushiji-testing10K.txt**  ... |
| --- |

# ***Possible Progress Steps***

These are purely suggestions for how you might make progress. You do not have to follow these steps. If you get stumped or frustrated on some portion of the assignment, you should not hesitate to consult a preceptor.

1. Download/unzip the ([classifier.zip](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/classifier.zip)) project folder.
2. *Feature extraction.*
   1. Review Section 3.1 of the textbook, especially Program 3.1.4 ([Grayscale.java](https://introcs.cs.princeton.edu/31datatype/Grayscale.java.html)) for using the [Picture](https://introcs.cs.princeton.edu/java/11cheatsheet/#Picture) and [Color](https://introcs.cs.princeton.edu/java/11cheatsheet/#Color) data types. Note that the images **are already grayscale**, so you don’t need to use Luminance.java. In particular, the red, green, and blue components are equal, so you can use any of getRed() or getGreen(), or getBlue() to get the grayscale value.
   2. Create a Picture object for the image 49785.png (in the project folder) and display it in a window. (Remove or comment out this code after you have successfully displayed the image.)
   3. Create a Picture object for the image image3-by-3.png (in the project folder) corresponding to the 3-by-3 example given below.



Extract its width and height and print the values to standard output. Then, extract the grayscale values of the pixels and print. If it’s not already in row-major order, adjust your code so that it prints the values in the specified order.

If you are using IntelliJ, do not type the import java.awt.Color; statement that is normally needed to access Java’s color data type. IntelliJ is pre-configured to automatically add import statements when needed (and remove them when not needed).

1. Create a one-dimensional array of length *width* × *height* and copy the grayscale values to the array. Print the values of this array to confirm you can create a vector (row-major order) of values from a Picture object.
2. Using the code from steps (c) and (d) as a guide, implement the static method extractFeatures() that takes a Picture as an argument and returns the grayscale values as a **double[]** in row-major order.
3. Write a main() method that tests extractFeatures().

| ***Testing Hint!***  Using image3-by-3.png, your main method should print the values returned by extractFeatures() as shown in the above figure. |
| --- |

1. *Parsing input files.* 
   1. Review the [**In**](https://introcs.cs.princeton.edu/java/11cheatsheet/#In)data type from Section 3.1, which is an object-oriented version of StdIn. You need the object-oriented version in this program since you will be reading from *two* different files in the same program.
   2. Create an **In** object for the training file [digits-training20.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/digits-training20.txt). Read the integers m, width and height, and print them to standard output. Read pairs of filenames (strings) and labels (integers) and print them to standard output.
   3. For each image, create a new Picture object and display in its own window. (Note - the image displays may overlap!)
   4. Repeat the previous two steps for the testing file [digits-testing10.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/digits-testing10.txt).
   5. Modify your program to take the names of the testing and training files as command-line arguments.
2. *Classifying images.*
   1. *Classifier.* After reading m, width, and height from the training file, create a MultiPerceptron object of the correct dimensions.
   2. *Training.* For each training image, train the classifier using the corresponding label.
   3. Testing. For each testing image, predict its class. Print each misclassified image to standard output and tabulate statistics on the number of misclassified images.
   4. *Error rate*. After training and testing, print the error rate to standard output in the specified format.
   5. *Test.* Test your program on some small data files, such as [digits-training20.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/digits-training20.txt) and [digits-testing10.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/digits-testing10.txt).
3. Now, the fun part. Use large training and testing input files. Be prepared to wait for 1 minute (or more) while your program processes 60,000 images.

| % **java-introcs ImageClassifier digits-training60K.txt digits-testing10K.txt**  jar:file:digits.jar!/testing/5/9428.png  jar:file:digits.jar!/testing/6/4814.png  jar:file:digits.jar!/testing/5/4915.png  ...  jar:file:digits.jar!/testing/5/7870.png  jar:file:digits.jar!/testing/4/1751.png  jar:file:digits.jar!/testing/5/6043.png  test error rate = 0.136 |
| --- |

Don’t worry about the odd looking filenames. It's just a verbose way to specify the location to a specific image file in a JAR (*Java ARchive*) file. Modern operating systems are not so adept at manipulating hundreds of thousands of individual image files, so this makes training more efficient. In this case, jar:file:digits.jar identifies the JAR file digits.jar and /training/7/4545.png identifies a file named 4545.png, which is located in the subdirectory /training/7/ of the JAR file.

# ***Analysis***

Some people (especially in Europe and Latin America) write a 7 with a line through the middle, while others (especially in Japan and Korea) make the top line crooked.

|  |  |  |
| --- | --- | --- |

Suppose that the training data consists solely of samples that do not use any of these conventions. How well do you think the algorithm will perform when you test it on different populations? What are the possible consequences?

Now suppose that you are using a supervised learning algorithm to diagnose cancer. Suppose the training data consists of examples solely on individuals from population X but you use it on individuals from population Y. What are the possible consequences?

Provide your answers in your readme.txt file.

# ***Submission***

# Submit ImageClassifier.java and a completed [readme.txt](https://www.cs.princeton.edu/courses/archive/fall20/cos126/assignments/classifier/files/readme.txt) file.

This assignment was developed by Robert DeLuca, Maia Ginsburg, Alan Kaplan, Kevin Jeon, and Kevin Wayne

Copyright © 2020, [Kevin Wayne](http://www.cs.princeton.edu/~wayne).