# Assignment 7: Trained Classifier
**Due: August 10, 2023 at 11:59PM**

## 1 Assignment 7: Trained Classifier Specification

### 1.1 Overview

For this assignment, you will write a handwriting classifier. Your program will be given a set of labeled examples for training, followed by unlabeled examples which it must classify.

Your program should have one command line argument for the algorithm: one of `knn` ($k$-nearest neighbors) or `ln` (linear regression).

The data that we will work with is a set of handwritten digits. Each digit has a label and a 14x14 image, converted to a 196-length vector. Each value in the vector is in the $0-4$ range, and represents the brightness of that pixel.

Find the starter files – a full data set, small data set, testing harness, and example program – at `https://classroom.github.com/a/o2UqpByD`.

### 1.2 Goals

1. Implement a k-nearest neighbors algorithm
2. Implement a linear regression algorithm

### 1.3 Input Descriptions

The input will consist of two parts: a training section followed by a testing section. The training section has a header for training data, followed by the training data itself (the label followed by the vector). The testing section has a header followed by unlabeled data – just the vector.

Here is an example with much shorter vectors:

```
-- training --
5 0 0 1 2 4 1
0 1 0 1 1 2 0
4 3 3 3 1 4 4
1 2 3 0 0 1 1
9 0 4 4 4 0 0
2 1 3 3 0 0 0
1 2 4 0 1 1 1
-- testing --
2 4 0 0 1 3
3 2 4 4 1 0
0 0 2 0 2 0
```

You should expect the input from `stdin`.

### 1.4 Output

Your program output is very simple:

For each testing line, you should output a line with a label and a confidence level (between 0 and 1). The classifications should be in the same order as the testing instances.

Example output for the above input:

```
1 0.85
```

```
4 0.58
5 0.63
```

You should output your classifications to `stdout`. You may need to flush your output for the harness to function correctly.

### 1.5 Execution

Write your code in one of two languages: Java or Python.

#### 1.5.1 Python Details

You should name the source file that has the program's main method `classifier.py`. The following should invoke your program:

```
> python classifier.py knn
```

#### 1.5.2 Java Details

You should name the source file that has the program's main method `Classifier.java`. The following should invoke your program:

(assuming that the code is compiled)

```
> java Classifier.class knn
```

**NOTE**: In order for the harness to work with Java, your implementation **must not** reside in a package. The harness will not correctly invoke the JVM if you use a package.

#### 1.5.3 Harness

Also included is a harness program that will run your program over many different training and test samples to show the effectiveness of your learnering algorithms.

There are several flags that control the run of the harness:

```
usage: java -jar a7_harness.jar
 -a,--algorithm <arg>    one of knn or ln
 -d,--data <arg>         path to data file
 -html                   output misclass.html file to show incorrect
 -m,--max-train <arg>    max number of labeled instances to train with
 -s,--seed <arg>         seed for picking random instances from data file
 -v,--verbose            print verbose info from harness
 -x,--executable <arg>   path to executable file
```

Only the `-x` option is required.

```
> java -jar a7_harness.jar -a knn -d digits.data -m 2000 -x classifier.py
```

You may have to flush your output in order for the harness to receive it. If you are having trouble getting your program to cooperate with the harness, the `-v` flag will give lots of information about what the harness is trying to send or receive.

## 2 Learning Algorithms

The two learning algorithms you will implement are (multivariable) linear regression and $k$-nearest-neighbors (from chapters 19.6 and 19.7). Implement both algorithms yourself – don't rely on an external library to generate the weight vectors for regression or find your set of neighbers in knn.

### 2.1 Linear Regression

Linear regression is good for dividing a feature space in two, but there are ten different labels. Following the multivariable linear regression in the book, you should come up with a separate vector of weights for each of the 10 digits. This leads to separate weight vectors for the {0, not 0} classification, a {1, not 1} classification, etc..

Use the logistic regression (19.6.5) to assign probabilities to each of the ten digits. For each test instance, your classifier should calculate probabilities for all ten digits and pick the digit that is most probable. This probability is your confidence, which is included in the output.

### 2.2 kNN

Implement the $k$-nearest-neighbors algorithm in 19.7.1. Experiment with both Manhattan distance and Euclidean distance – use whichever one gets you better results. Your classification should be chosen as the majority of neighbors, and your confidence should be the ratio of majority neighbors to total neighbors.

## 3 Submission

Submit the code for your solution along with a writeup that answers the following questions

1. Describe choices you made in your code that you feel are important. Mention any specific aspects of your implementation that might be interesting as I evaluate your program.
2. Which algorithm performs better for large training sizes?
3. Why do you think that one performs better?
4. What suggestions do you have for improving this assignment?

For submission, commit/push your updates to your github repository. I automatically have access to the repository, so if you see your updated files on Github, I have access to them too.

## 4 Evaluation

- +15%: Interacts correctly with harness
- +20%: kNN learns at all
- +15%: kNN learns well
- +5%: kNN runs quickly on training size of 8000
- +20%: regression learns at all
- +15%: regression learns well
- +5%: regression runs quickly on training size of 8000
- +5%: good writeup