4/11/23 Nearest Neighbor, Old School Graphics, Video Game AI Exercises 78 - 83

This is a preview of the published version of the quiz

Started: Sep 3 at 12:57am

Quiz Instructions

Question 1 2 pts

Exercise 78 Using `Math.random()` for Simulations: Match each outcome to a simulation that tests for it.

Roll of die with numbers 1, 2, 3, 4, 5, 6 is: 1 or 2	[Choose]
Flip of three coins comes up with exactly two tails	[Choose]
Rains when probability of rain is .15	[Choose]
Does not rain when probability of rain is 0.15	[Choose]

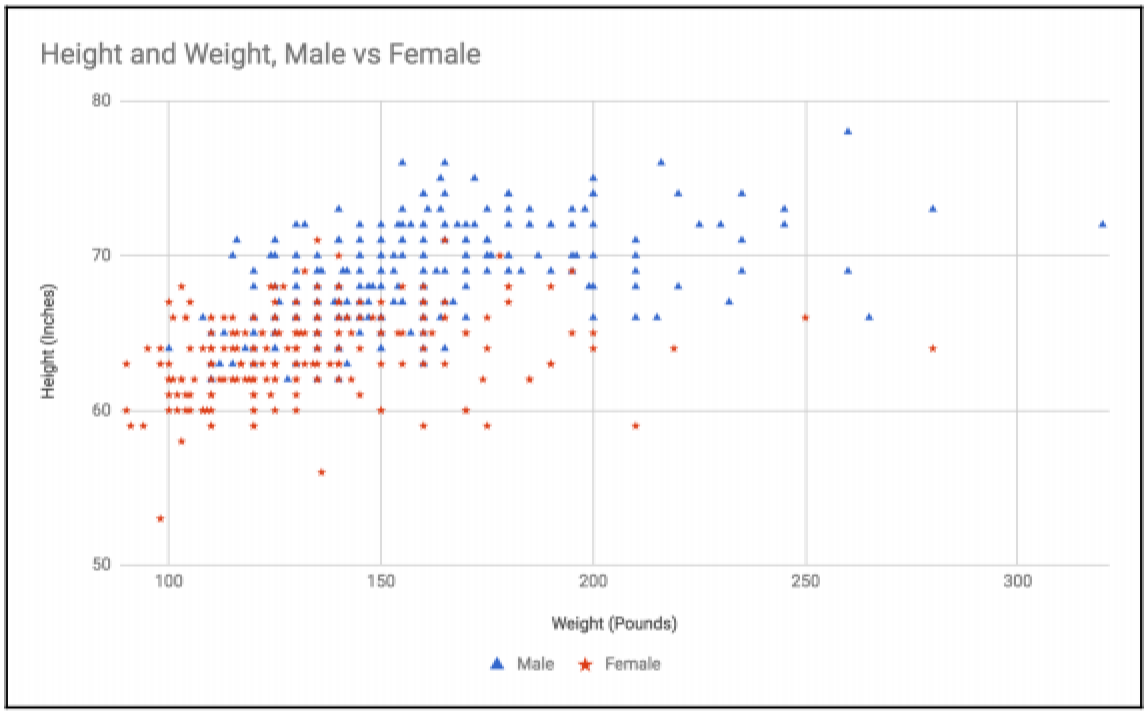
Question 2 7 pts

Exercise 79 Predict Gender from Weight/Height: In this exercise, you will be defining a function that takes a person's weight and height as inputs and attempts to predict whether the person is a female or male using a so-called 1-nearest neighbor machine learning classification algorithm. Use https://repl.it/@jschanker/male-or-female-from-height-and-weight (https://repl.it/@jschanker/male-or-female-from-height-and-weight) .

This is an example from a book, Hands-on Machine Learning with JavaScript: Solve complex computational web problems using machine learning. However, be aware that there are negative consequences of classification algorithms such as these where the people defining the classes that people are separated into may inadvertently make non-inclusive assumptions (e.g., that everyone identifies as male or female). (See:

https://www.cnn.com/2019/11/21/tech/ai-gender-recognition-problem/index.html (https://www.cnn.com/2019/11/21/tech/ai-gender-recognition-problem/index.html))

In this context, this means outputting the gender of the person whose weight and height is closest to the given weight and height in the so-called training data. The training data consists of people's heights and weights along with labels of these people's genders. We'll determine closeness by the (Euclidean) distance between the points of the given weight *x* and height *y* given as (x, y) and the weight-height points from the data so we'll be looking for the point that's closest to this point from the below data:



Female and Male Weight/Height Data from Hands-on Machine Learning with JavaScript: Solve complex computational web problems using machine learning

In general, a machine learning classification algorithm uses a large amount of training data such as people's weights and heights (e-mail or web page textual data, etc.) along with labels such as male or female (spam versus non-spam, language of web page text - English, Hindi, Japanese, Spanish, etc.) and uses it to predict the label of a new data point. A 1-nearest neighbor algorithm does this by locating a "closest" data point and outputting that label. A 5-nearest neighbor algorithm locates the 5 closest points and outputs the label of the most frequently occurring of the 5 (e.g., if 4 of the closest weights/heights were from females and only 1 from a male, it'd output female). A k-nearest algorithm could be used for any k to take the most frequently occurring label of the k nearest neighbors where k is a most equal to the number of data points but typically is a small number such as 5.

First, we will map each `index` of the weights or heights Array to the distance between the points (x, y) and (`weight`, `height`) where x is the weight at `index` in the `weights` Array and y is the height at `index` in the `heights` Array. This means that `__startRange__` and `__endRange__` should be filled in with the starting and 1 more than the ending index of EITHER the `weights` or `heights` Array as in previous exercises. (It doesn't matter which you use since both Arrays have the same size.). It also means that **__distance__** will be filled in with the expression for the distance between the points (x, y) and (`weight`, `height`) (Recall the distance formula which comes from the Pythagorean Theorem - Question 5 of **Functions (https://molloy.instructure.com/courses/34841/quizzes/67258)**). But x needs to be replaced with an expression representing the weight at `index` in the `weights` Array and y needs to be replaced with an expression representing the height at `index` in the `heights` Array. How do you refer to an item at a given index in an Array?

Now that you have all of the distances stored in `distanceArr`, the next step will be to use `reduce` to find the **`index`** of the minimum of these distances. (Recall `reduce` is a built-in JavaScript function mimicked by the Text2Code `repeatForEveryItem(...).startingWithAccOf(...)`.) This part will be somewhat similar to the code you wrote for `minimum` in **3/25/20 Array Assignment Exercises 65 - 68 (https://molloy.instructure.com/courses/34841/quizzes/67202)** . However, this time the accumulator will be the current **INDEX** of the smallest item before index in `distanceArr` instead of the current minimum itself. Notice here we start `range` at 1. What is the INDEX of the minimum of the numbers BEFORE 1. (HINT: There's only one index before 1.) This will be your starting accumulator to fill in for **__startAcc__**.)

__condition__ will be comparing the ITEM at the accumulator (`currentClosestIndex`) to the ITEM at `index` of the distance Array and the new accumulator will be the index of the smaller item (either `currentClosestIndex` or what you put for __newAcc__).

Finally once you have the `indexOfClosest`, which is the index of the (x, y) that's the closest to the given (`weight`, `height`), you get the corresponding item in the `genders` Array. This will be the predicted **__answer__**.

```
let weightHeightData = require("./weight-height-data.js");
let weights = weightHeightData.data.map(x => x[ 0 ]);
let heights = weightHeightData.data.map(x => x[ 1 ]);
let genders = weightHeightData.labels;

let predictGender = (weight, height) => {
  let distanceArr = range(__startRange__, __endRange__)
    .map(index => __distance__);
  let indexOfClosest = range(1, __endRange__)
```

```
    .reduce((currentClosestIndex, index) => {
      if(__condition__) return __newAcc__;
      else return currentClosestIndex;
    }, __startAcc__);
  return __answer__;
};

console.log(predictGender(140, 60)); // should log female
console.log(predictGender(175, 65)); // should log female
console.log(predictGender(255, 70)); // should log male
console.log(predictGender(300, 75)); // should log male
```

`__startRange__` =

`__endRange__` =

`__distance__` =

`__condition__` =

`__newAcc__` =

`__startAcc__` =

`__answer__` =

Question 3 3.5 pts

Exercise 80 a) (Decolorize a photo) In this exercise (https://replit.com/@jschanker/Spring2023-Image-Processing-decolorizing#index.js (https://replit.com/@jschanker/Spring2023-Image-Processing-decolorizing#index.js)), we will be using the 1-nearest neighbor algorithm to reduce the number of colors used for a photo. We do this by replacing the closest red-green-blue triple of the form (r, g, b) where r, g, and b are integers from 0 to 255 from a collection of say 16 colors to each (r, g, b) pixel of the image data. If we want to make the image grayscale, then we are choosing between 256 colors where the red, green, and blue are all the same; the higher the number the brighter the shade of gray with (255, 255, 255) being white, (0, 0, 0) being black, and (127, 127, 127) being a shade of gray in between the two extremes.

After successfully completing this exercise, you will see three files with the Golden Gate Bridge shown using 16 colors, 27 colors, and a 256-color grayscale.

The first step will be getting the closest color to a given red, blue, and green in an Array of red-green-blue triples. This part of the solution will be similar to the previous exercise except you'll be calculating a 3D distance instead of a 2D distance. To get the red of a color at `index` from the colors Array, use `colors[ index ][ 0 ]`; for the green, use `colors[ index ][ 1 ]`; for the blue, use `colors[ index ][ 2 ]`.

```
let getClosestColor = (red, green, blue, colors) => {
  let distanceArr = range(__startRange__, __endRange__)
    .map(index => __distance__);
  let closestColorIndex = range(1, __endRange__)
    .reduce((currentClosestIndex, index) => {
      if(__condition__) return __newAcc__;
      else return currentClosestIndex;
    }, __startAcc__);
  return __answer__;
};

console.log(getClosestColor(140, 60, 20, [ [120,60,20], [150,75,25] ])); // should log [150, 75, 25]
console.log(getClosestColor(255, 255, 255, [ [255,255,255], [0,0,0], [127,127,127] ])); // should log [255, 2
55, 255]
console.log(getClosestColor(84, 0, 0, [ [80, 80, 80], [80, 20, 10], [0, 0, 84], [100, 50, 100] ])); // should
log [80, 20, 10]
```

`__startRange__` =	
`__endRange__` =	
`__distance__` =	
`__condition__` =	
`__newAcc__` =	
`__startAcc__` =	

`__answer__ =`

Question 4 2.5 pts

Exercise 80 b) (Decolorize a photo) The next step will be to create the new image by using `map` to map each pixel of data to the (r, g, b) triple that's closest to it from the `rgbCodes` Array by using your function `getClosestColor` defined in part (A). The `rgbCodes` Arrays will be loaded from three JavaScript files, the first will be the 16 colors from colors16.js, the second will be the 27 colors from colors27.js and the last one will be from the 256 gray colors from grayscale.js.

For each of the __functionCall__ blanks, you will be referring to your `getClosestColor` function, giving it the appropriate inputs. Recall the red of a pixel appears at Array locations that are divisible by 4 (0: red of pixel 0, 4: red of pixel 1, 8: red of pixel 2, etc.); the green of a pixel appears at Array locations that are 1 more than a multiple of 4 (1: green of pixel 0, 5: green of pixel 1, 9: green of pixel 2, etc.); the blue of a pixel appears at Array locations that are 2 more than a multiple of 4 (2: blue of pixel 0, 6: blue of pixel 1, 10: blue of pixel 2, etc.); the alpha or transparency of a pixel appears at Array locations that are 3 more than a multiple of 4 (3: alpha of pixel 0, 7: alpha of pixel 1, 11: alpha of pixel 2, etc.).

So when you give the inputs for the red, you'll be passing it the item at `index` of `imageData` for red, and the next two indices for green and blue.

When you give the inputs for green, you'll be passing it the item at `index` of `imageData` for **green**, the red for the pixel will be one before this and the blue for the pixel will be one after.

When you give the inputs for blue, you'll be passing it the item at `index` of `imageData` for **blue**, the red for the pixel will be two before this and the green for the pixel will be one before.

Remember that you need 4 inputs to `getClosestColor`, the last one will be the Array of colors you're allowed to use.

Since `getClosestColor` returns the closest allowed color in the form of an Array of length 3 [r, g, b] (say `[50, 100, 150]`), we get the 0th element to get the red of 50, the first element to get the green of 100, and the 2nd element to get the blue of 150. This is why we use [0], [1], and [2] at the end of the function call answers .

```
const decolorizeImageNearestNeighbor = (imageData, rgbCodes) => {
  const newImageData = range(__startRange__, __endRange__).map(index => {
    if(index%4===0) return __functionCallForRed__[ 0 ];
    else if(index%4===1) return __functionCallForGreen__[ 1 ];
    else if(index%4===2) return __functionCallForBlue__[ 2 ];
    else return imageData[ index ]; // alpha - transparency stays the same
  });

  return newImageData;
};
```

__startRange__ =

__endRange__ =

__functionCallForRed__ =

__functionCallForGreen__ =

__functionCallForBlue__ =

Question 5 3.5 pts

Exercise 81 Old School Graphics: Watch the following video: https://www.youtube.com/watch?v=Tfh0ytz8S0k (https://www.youtube.com/watch?v=Tfh0ytz8S0k)

(https://www.youtube.com/watch?v=Tfh0ytz8S0k)

The typical screen had how many pixels according to the video?

Given that there are 8 bits in a byte and 1000 bytes in a kilobyte, how many bits are in a single kilobyte (KB)?

1 KB = ______ bits

Using the color cell method described in the video, it takes 1 bit (0 or 1) to indicate whether a pixel is part of the foreground or background (e.g., 0 = background, 1 = foreground). So it takes as many bits as there are pixels on the screen to represent whether they are part of the background or foreground. How many **kilobytes** is this? (Divide the total number of pixels by the number of bits in a kilobyte.)

______

Given that each color cell consists of ______ pixels (square the width/height which are both the same), there are how many color cells? Divide the total number of pixels by the number of pixels in a color cell to get this answer.

Number of color cells = ______

Since we can represent 2^4 or 16 colors using 4 bits, we can use 4 bits to store which of the 16 colors we're using for the background color of the color cell and 4 bits to store which of the 16 colors we're using for the foreground color of the color cell for a total of 4 + 4 or 8 bits = 1 byte. This means we need 1 additional byte to store the background/foreground color of **each** color cell. Given your previous answer, how many **kilobytes** are required to store the background/foreground information of all color cells (since each color cell requires 1 byte, multiply this by the number of color cells and divide by 1000 to get the number of kilobytes).

Number of kilobytes to store background/foreground color information of all color cells = ______.

Adding this number with the total number of kilobytes required to store whether each pixel is part of the foreground/background, you get the number of kilobytes mentioned in the video, which is ______

Question 6 1 pts

MarI/O: Watch https://www.youtube.com/watch?v=qv6UVOQ0F44 (https://www.youtube.com/watch?v=qv6UVOQ0F44)

(https://www.youtube.com/watch?v=qv6UVOQ0F44)

. The name of the level that the AI beat was ________ 1; you'll fill in two words with a single space in it.

Question 7 1 pts

Exercise 83) (Programming Video Game AI) Quickly play a little of the following Platform Game. To play it, click right next to `<body>` ; then click on the ≡ that appears and click "Run code". Click somewhere in the blue box and use the arrow keys to move around, left, right, and up (Don't get addicted!): http://eloquentjavascript.net/17_canvas.html#c_TSR2vcnWZv (http://eloquentjavascript.net/17_canvas.html#c_TSR2vcnWZv)

In Design and Analysis of Algorithms, we fill in code to create a computer player to beat a custom-made level of the game:

https://jschanker.github.io/eloquent-js-game-ai/ (https://jschanker.github.io/eloquent-js-game-ai/)

You give it a maximum number of moves from 1 - 32 and a move duration .01 - 0.5, and it'll try to make the fewest moves to beat the level by simulating the holding of the key (left, right, up, left-up, right-up, or no key - standing still) for the given move duration. **Note**: If it stands still permanently, it means it was unable to complete the level in the given number of moves if it's forced to hold each move for the move duration. Also note that sometimes it gets stuck due to differences in frame rates.

Try it out! Note, it'll take some time to figure out how to beat the level so be patient. Click the Have AI Attempt to Conquer Level! How many coins did it collect on the given level?

Now refresh the page and click the Climb The Tower button before clicking the Have AI Attempt to Conquer Level! How many coins did it collect to beat this level?

Not saved Submit Quiz