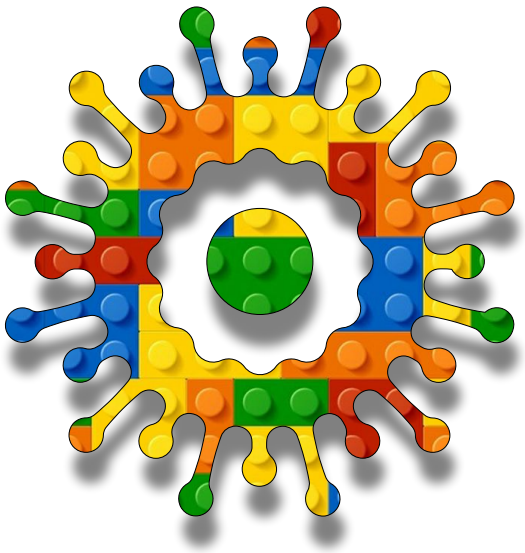
Algorithmic Intelligence Module A Unit #10 cluster regression



Content

Module A Unit #10 cluster regression

Introduction to cluster regression

In this module, we will describe a simple model for regression. This is very similar to the classification one you have just done. The main difference is that the outcome (output) is a value between 0 and 255, compared to the classification of the labels A, B, C, or D.

Here, we will click on the canvas to create a cluster of white circles and a cluster of black circles. We will train it on those data points and then predict what shade of grey the canvas is between those data points. It is, in effect, going to interpolate using a neural network.

The index.html file

The `index.html` file and the requisite `ml5.js` library.

```
<!DOCTYPE html>

<html lang="en">

  <head>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.1/p5.js"></
script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.1/addons/
p5.sound.min.js"></script>

    <script src="https://unpkg.com/ml5@1/dist/ml5.min.js"></script>

    <link rel="stylesheet" type="text/css" href="style.css">

    <meta charset="utf-8" />


  </head>

  <body>

    <main>

    </main>

    <script src="sketch.js"></script>

  </body>

</html>
```

Sketch A10.1 basic starting model

This is our starting sketch. We are creating the basic neural network model, similar to the previous one. There are two inputs, `x` and `y`, and one output, `val`, which will be the grey value between `0` and `255`. The task is `regression`, not `classification`.

```
let nn


function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}
```

🗒️ Notes

The main points here are:

- 🀄 The task is now `regression`.
- 🀄 The inputs are still `x`, and `y`.
- 🀄 The output is a value called `val`.

🛠️ Code Explanation

`task: 'regression'`	Defines this as a regression task

Sketch A10.2 assigning a value

We want to assign a value to the circle (rather than a label). To keep it simple, we are using w for white (with a value of 255) and b for black (assigning a value of 0). These are the fill colours (and values); they will be the data we train the model on.

```
let nn
let targetLabel = 'w'
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}
```

🗒️ Notes

We will have two target labels, w for white and b for black.

🛠️ Code Explanation

`let targetLabel = 'w'`	We define and initialise the target label as w
`let col = {w: 255, b: 0}`	This is defining the values of the target labels as an object array

Sketch A10.3 assigning the keys

When we want to change the letter of the `targetLabel`, we use the `keyPressed()` function to define the label and attribute a value to the data point. So, with a `w` we will give it a value of `255`, and with `b`, we will give it a value of `0`.

```
let nn
let targetLabel = 'w'
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}
```

```
function keyPressed()
{
  targetLabel = key
}
```

🗒️ Notes

All that happens at this stage is that we are changing the value of the target label.

🛠️ Code Explanation

`targetLabel = key`	The variable targetLabel will take whatever letter you type in. Which should be the letter b or w for this exercise

Sketch A10.4 drawing the circles

There is quite a lot happening here. We have a `targetLabel` of `w` which has an initial value. The `col()` function takes the value from the object array `col = {w: 255, b: 0}` and gives it to the `targetVal` variable, which in turn is used to `fill()` the circle. We aren't bothering with the letters in the circles this time, just the colour. We introduce the `mousePressed()` function to draw the circles when we click on the canvas.

```
let nn
let targetLabel = 'w'
let state = 'collection'
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}

function keyPressed()
{
  targetLabel = key
}

function mousePressed()
{
  let targetVal = col[targetLabel]
```

```
  fill(targetVal)
  circle(mouseX, mouseY, 25)
}
```

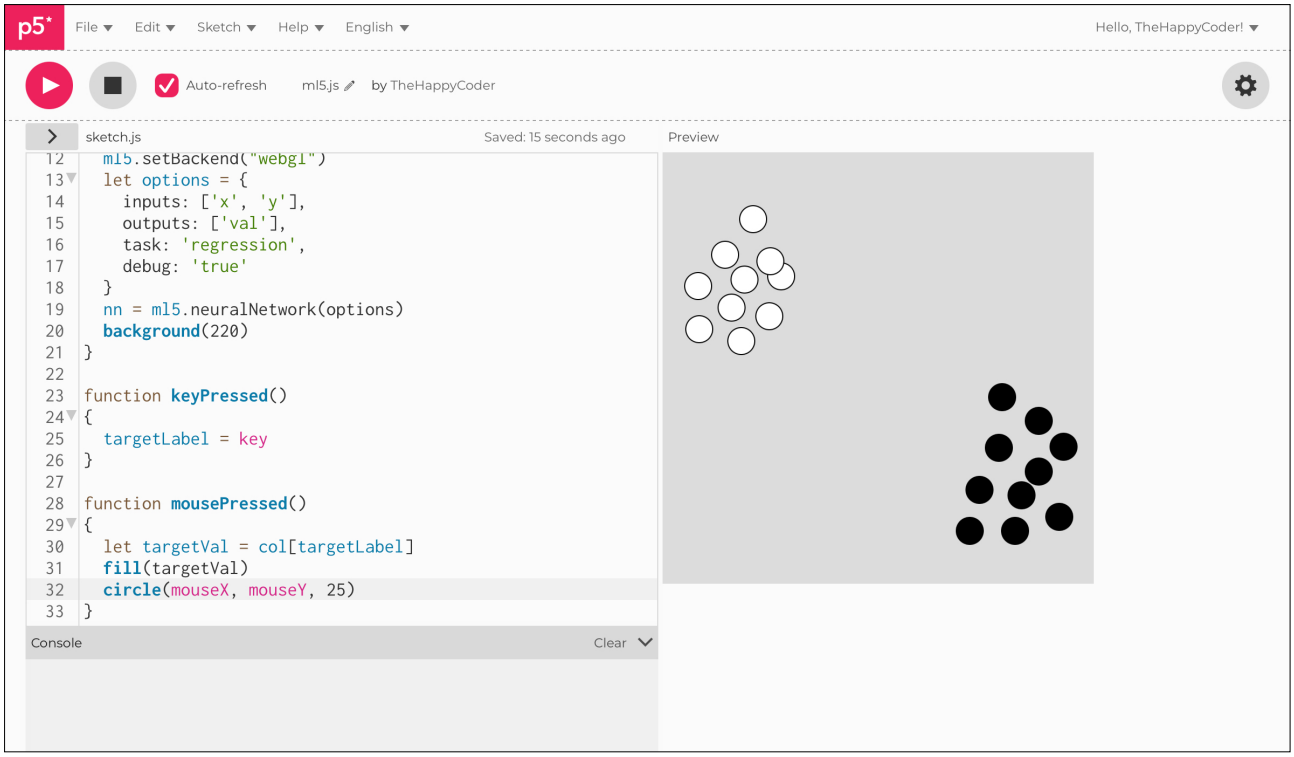
🗒️ Notes

When you click on the canvas, you get the white circle by default. When you press the letter b and then click on the canvas, the circle is black (0). This continues until you press w, then it will fill the circle with white (255).

🛠️ Code Explanation

`let targetVal = col[targetLabel]`	Takes the value of the target label, it is an object array hence the square brackets []
`fill(targetVal)`	Fills the circle with the value of the target label
`circle(mouseX, mouseY, 25)`	Draws the circle at the coordinates of the mousePressed()

Figure A10.4



Sketch A10.5 defining the inputs

We want to collect the data points from the canvas, through the `mousePressed()` function. The input data is the `mouseX` and `mouseY` coordinates.

```
let nn
let targetLabel = 'w'
let state = 'collection'
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}

function keyPressed()
{
  targetLabel = key
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
```

```
  let targetVal = col[targetLabel]
  fill(targetVal)
  circle(mouseX, mouseY, 25)
}
```

🗒️ Notes

All we have done is define the inputs, which you should be familiar with.

Sketch A10.6 collecting the data

We introduce the variable `state` to keep track of proceedings. We initialise it to `collection` because we first need to collect the data. We need the inputs for the data collection to be an array, so we create a data object to pass into the neural network. We create an `if()` statement inside the `mousePressed()` function and put some lines of code within that `if()` statement. We only want to draw the circles when we are collecting the data.

```
let nn
let targetLabel = 'w'
let state = 'collection'
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}

function keyPressed()
{
  targetLabel = key
}

function mousePressed()
{
  let inputs = {
```

```
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
}
```

🗒️ Notes

The data passed into the neural network are two object arrays, the inputs and the target value. We can write these objects as one line of code if we wish: `target = {val: targetVal}`. If there are many, it is nicer to see them as a list.

🛠️ Code Explanation

`if (state == 'collection')`	Check what state we are in
`let target = {val: targetVal}`	We take the value of the target label as an object array
`nn.addData(inputs, target)`	We can now pass in the data to the neural network ready for training

Sketch A10.7 training button

Let us create a button to press to train the model. For the moment, we have an empty `train()` function, but we will soon put some code in. We only want to train it after we have finished adding the data points.

```
let nn
let targetLabel = 'w'
let state = 'collection'
let button
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
  targetLabel = key
}

function mousePressed()
```

```
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
}
```

```
function train()
{
  console.log('button working')
}
```

🗒️ Notes

We have a button that doesn't do anything except tell you it is working when you click on it!

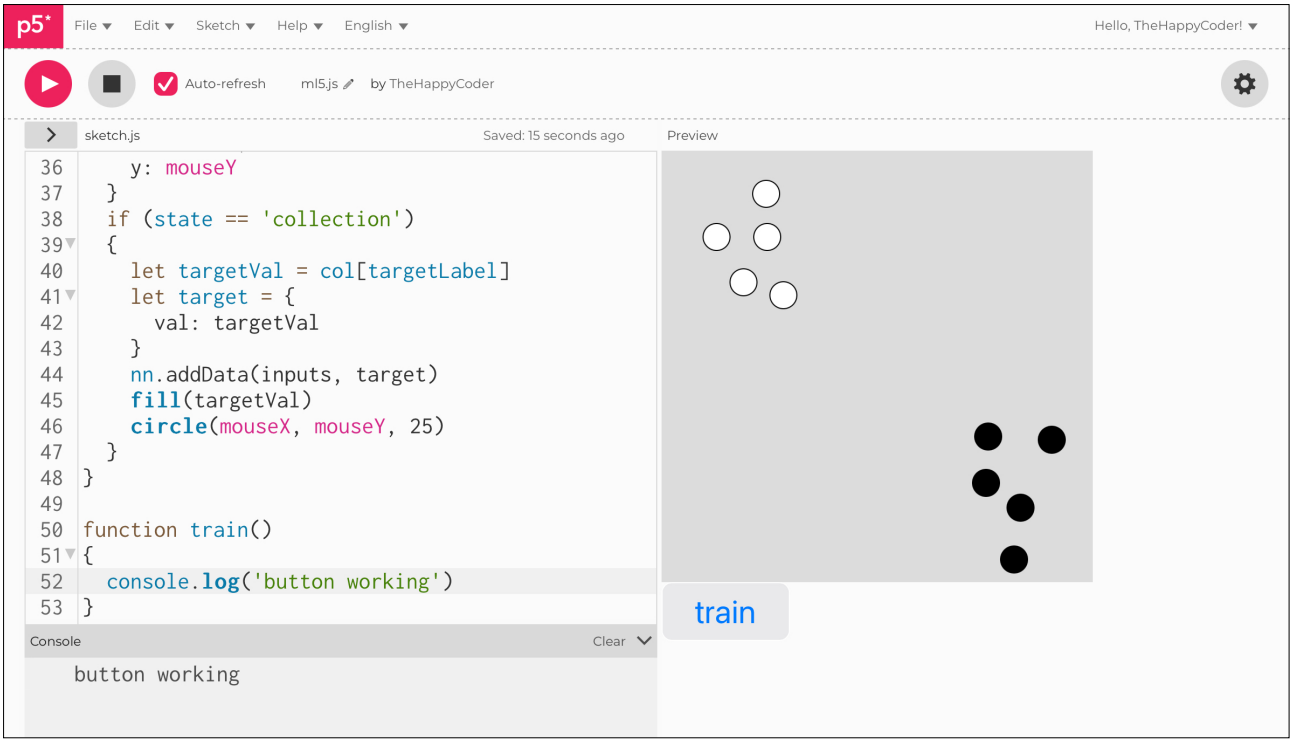
🌻 Challenge

Add some colour to the button.

🛠️ Code Explanation

`button.mousePressed(train)`	When the button is pressed it calls the train() function

Figure A10.7


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 15 seconds ago
Preview
36 y: mouseY
37 }
38 if (state == 'collection')
39 {
40 let targetVal = col[targetLabel]
41 let target = {
42 val: targetVal
43 }
44 nn.addData(inputs, target)
45 fill(targetVal)
46 circle(mouseX, mouseY, 25)
47 }
48 }
49
50 function train()
51 {
52 console.log('button working')
53 }
train
Console
Clear
button working

Sketch A10.8 training

Here we will add the code for training the model and a callback for when training is finished. I have added more epochs so we can see where the training ceases to improve significantly. This is probably unnecessary because of the small dataset. The `state` is now `training`.

It is time to add the next function, `finishedTraining()`, and change the state to `prediction`. The console will tell us when it has finished training and is ready to predict.

```
let nn
let targetLabel = 'w'
let state = 'collection'
let button
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
```

```
  targetLabel = key
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
}

function train()
{
  state = 'training'
  nn.normalizeData()
  nn.train(finishedTraining)
}

function finishedTraining()
{
  console.log('finished training')
  state = 'prediction'
}
```

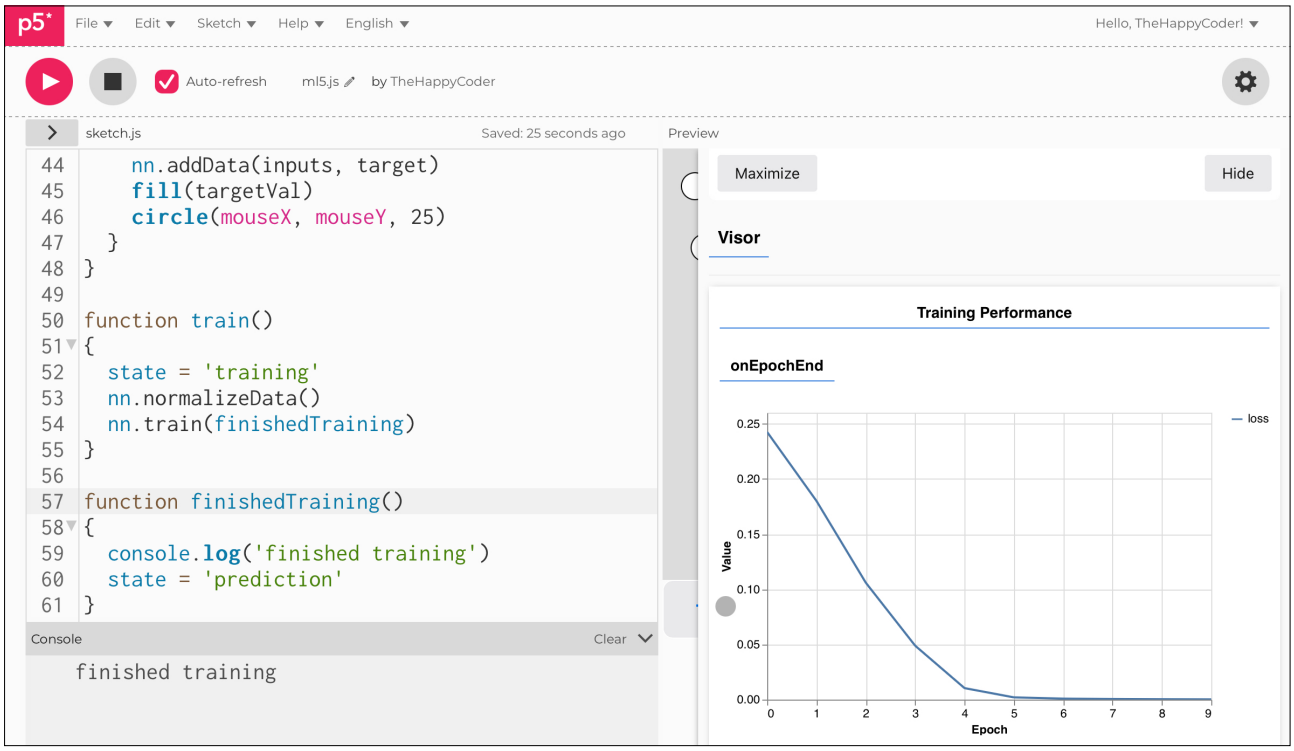
🗒️ Notes

As you train the model, you may well find that the loss curve plummets to zero in less than ten epochs. This is probably because there may not be enough data points.

🌻 Challenge

1. Try more data points.
2. Try just one of each data point.

Figure A10.8



Sketch A10.9 predicting

The next part is to predict the results, to determine the colour of the pixel when we draw our circle on the canvas. We will want a greyscale of colour, not just black or white, because a regression task gives us a sliding scale of values.

We have set the state to prediction already. Next, we add an `if()` statement to check to see if there is a change of `state` to `prediction` inside the `mousePressed()` function to start making those predictions with the `predict()` function. We will pick up the results in the callback function `gotResults()`.

! Don't run this yet, we haven't added the callback function yet.

```
let nn
let targetLabel = 'w'
let state = 'collection'
let button
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}
```

```
function keyPressed()
{
  targetLabel = key
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
  else if (state == 'prediction')
  {
    nn.predict(inputs, gotResults)
  }
}

function train()
{
  state = 'training'
  nn.normalizeData()
  nn.train(finishedTraining)
}

function finishedTraining()
{
  console.log('finished training')
  state = 'prediction'
```

```
}
```

🗒️ Notes

After we have finished `collecting` the data and finished `training` the model, we are then in `prediction` mode. Next is the callback function `gotResults()`, where we see what our results look like. The lack of the `gotResults()` function will give you an error.

🛠️ Code Explanation

`nn.predict(inputs, gotResults)`	We receive the inputs and have a callback function for the results

Sketch A10.10 prediction results

We need to add in the function `gotResults()` so we can see the results or `predictions`. After the training is completed, you will see the results when you click on the canvas. The circles should be a bit grey in between where your data points are; for each circle, it should fill with the predicted colour value.

```
let nn
let targetLabel = 'w'
let state = 'collection'
let button
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
  targetLabel = key
}
```

```
function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
  else if (state == 'prediction')
  {
    nn.predict(inputs, gotResults)
  }
}

function train()
{
  state = 'training'
  nn.normalizeData()
  nn.train(finishedTraining)
}

function finishedTraining()
{
  console.log('finished training')
  state = 'prediction'
}

function gotResults(results)
{
  fill(floor(results[0].value))
```

```
  circle(mouseX, mouseY, 25)
}
```

🗒️ Notes

This is the difference between `classification` and `regression`. You can see the gradual change in the fill colour of the circles as you click between the trained clusters of circles.

🌻 Challenge

You can look inside the results using the `console.log(results)` as shown in the second image below.

🛠️ Code Explanation

`results[0].value`	This pulls the value from the prediction results array at index [0]
`fill(floor(results[0].value))`	We use the floor so it is an integer

Figure A10.10a

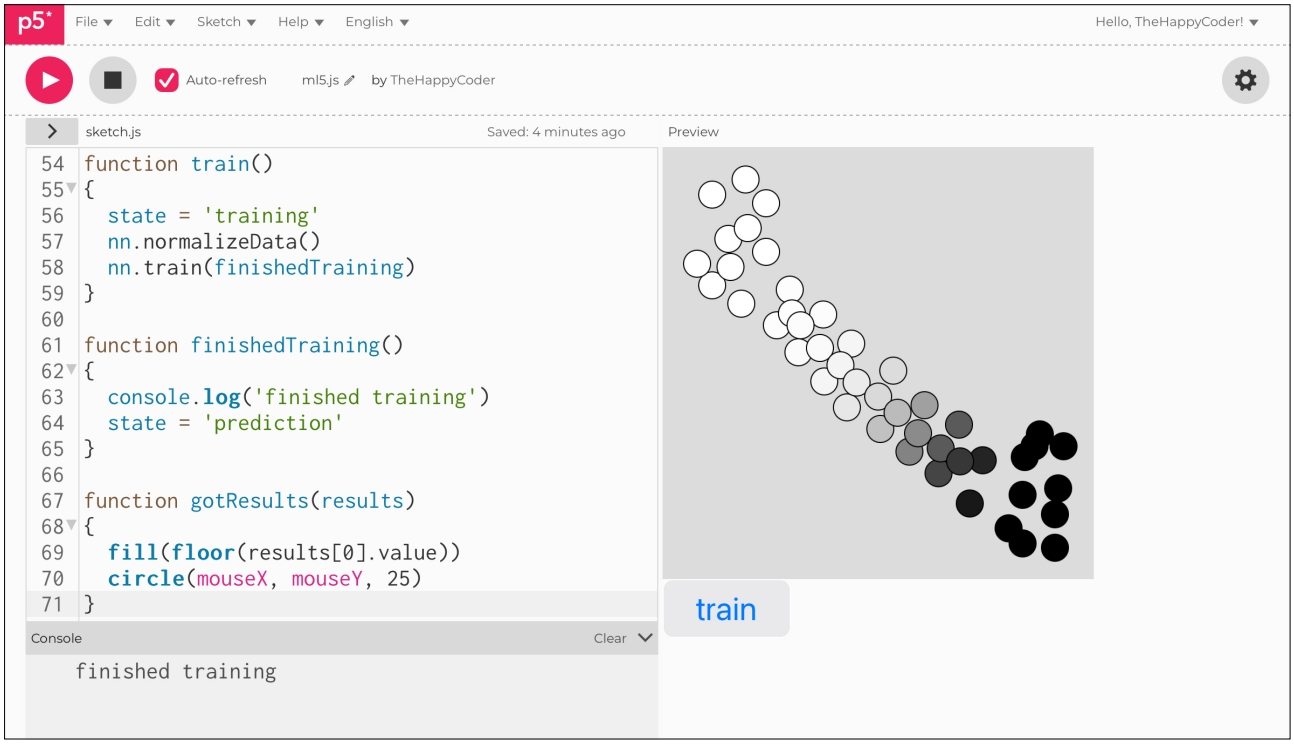
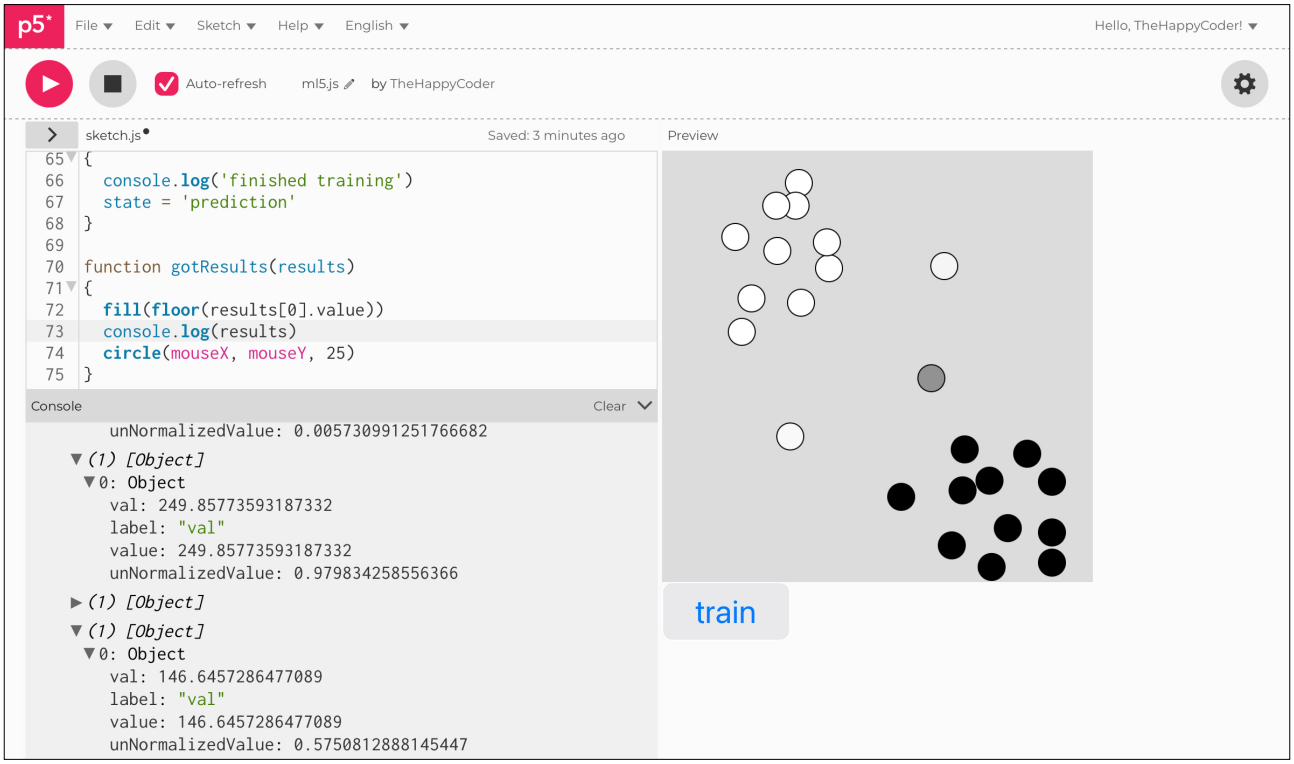


Figure A10.10b: with console.log(results)



Sketch A10.11 some refinements

I have highlighted some refinements. This will keep the original circles and their colour, but now when you click on the canvas, it will draw the shades of grey on the canvas.

! Comment out debugging to hide the loss chart.

```
let nn
let targetLabel = 'w'
let state = 'collection'
let button
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    // debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
  targetLabel = key
}

function mousePressed()
```

```
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
  else if (state == 'prediction')
  {
    nn.predict(inputs, gotResults)
  }
}

function train()
{
  state = 'training'
  nn.normalizeData()
  nn.train(finishedTraining)
}

function finishedTraining()
{
  console.log('finished training')
  state = 'prediction'
}

function gotResults(results)
{
  mousePressed()
  {
```

```
  fill(floor(results[0].value), 100)
  noStroke()
  circle(mouseX, mouseY, 30)
  }
}
```

🗒️ Notes

We have added a little transparency and removed the stroke, and created a larger circle.

🌻 Challenge

Consider how you might colour each pixel (not easy).

Figure A10.11a as we move the mouse

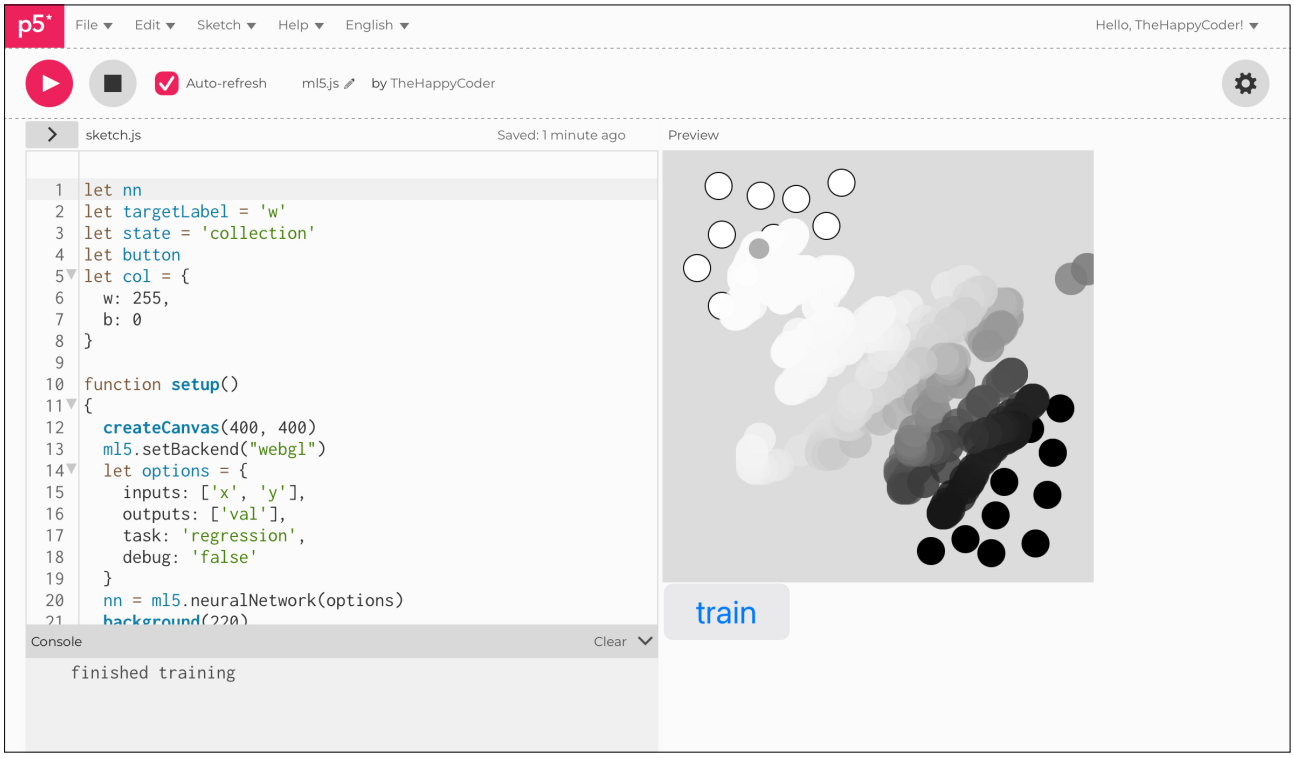


Figure A10.11b final look