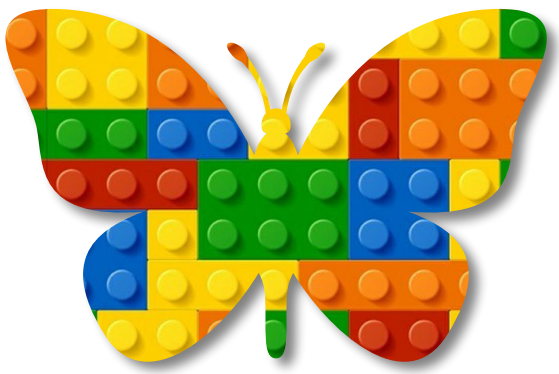
Artificial Intelligence Module A Unit #6 cluster classification



Content

Module A Unit #6 cluster classification

Introduction to cluster classification with ml5.js

This is a supervised learning model. We are going to click on the canvas in four areas. In each quadrant we will create a set of ten circles clustered together, each set of circles will be labelled A, B, C and D. We will then train it with ml5.js and when the training is complete we will click on the canvas and it will predict which lettered circle should be at that point on the canvas.

- This is a classification task.
- It will classify each circle A, B, C or D.

The index.html file

The index.html file needs the 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 A6.1 starting sketch

We have our bog standard starting sketch

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
}
```

Sketch A6.2 creating the model

Getting our neural network setup. We are going to use ml5.js default hyperparameter settings for the moment.

```
let nn

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork()
}

function draw()
{
  background(220)
}
```

🗒️ Notes

This just checks everything is working as it should when you run it.

🌻 Challenge

Think about giving your model a different name

🛠️ Code Explanation

`let nn`	We declare a variable name for our neural network
`ml5.setBackend("webgl")`	The setBackend() function allows this to run in all browsers, you could try "cpu" or even "gpu" depending on your machine
`nn = ml5.neuralNetwork()`	We define our neural network

Sketch A6.3 adding the options

We use the options to clarify what we are doing and what the inputs, outputs are. We tell it what task it is going to perform `classification` or `regression`. We will define the inputs and the outputs.

```
let nn

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['label'],
    task: 'classification'
  }
  nn = ml5.neuralNetwork(options)
}

function draw()
{
  background(220)
}
```

Notes

The `options` give you possibility of fine tuning the neural network. In this case we will have two inputs, the `x` and `y` co-ordinates and four outputs, `A`, `B`, `C` and `D`, which are the `labels`. Sometimes you don't even need to specify the inputs and outputs because the library can work it out anyway. It is important to specify the task though. In this case it is a `classification` task.

🛠️ Code Explanation

`let options = {...}`	Defining the options
`inputs: ['x', 'y']`	The inputs are an array of x and y values
`outputs: ['label']`	The outputs are labels A, B, C or D
`task: 'classification'`	The task is a classification
`nn = ml5.neuralNetwork(options)`	The options are attributed to the neural network

Sketch A6.4 training data

We are going to create a `mousePressed()` function to collect training data. We need to move the `background()` into `setup()` and remove the `draw()` function because we will do all the drawing in the `mousePressed()` function. We will start with a default letter `A` and place it in a blue circle.

```
let nn

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

// function draw()
// {
//   background(220)
// }

function mousePressed()
{
  fill(0, 0, 255, 100)
  circle(mouseX, mouseY, 25)
  textAlign(CENTER, CENTER)
```

```
  stroke(0)
  text('A', mouseX, mouseY)
}
```

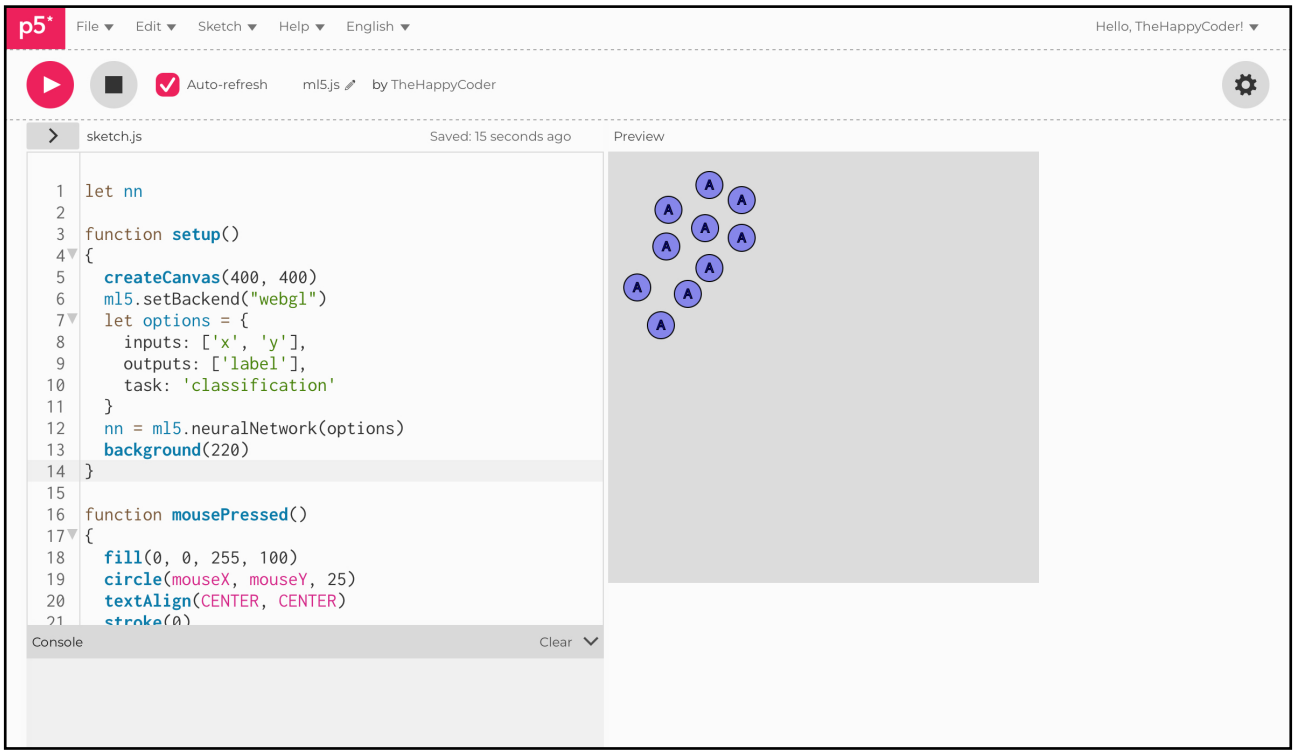
🗒️ Notes

If you click on the canvas you should get a circle with the letter `A` in the centre of it like thus. This will form part of your data collection. For the future sketches there will be no `draw()` function.

🛠️ Code Explanation

`function mousePressed()`	This function waits for the mouse to be pressed and then actions the code in the function.
`fill(0, 0, 255, 100)`	Gives a blue colour and the fourth argument is the alpha making a it a little transparent
`textAlign(CENTER, CENTER)`	This puts the letter in the very centre of the circle
`text('A', mouseX, mouseY)`	We place the text at the same co-ordinates as the circle, at mouseX and mouseY

Figure A6.4


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 15 seconds ago
Preview
let nn
function setup()
{
createCanvas(400, 400)
ml5.setBackend("webgl")
let options = {
inputs: ['x', 'y'],
outputs: ['label'],
task: 'classification'
}
nn = ml5.neuralNetwork(options)
background(220)
}
function mousePressed()
{
fill(0, 0, 255, 100)
circle(mouseX, mouseY, 25)
textAlign(CENTER, CENTER)
Console
Clear

Sketch A6.5 target label

We already have the letter A by default but we want to collect three more data points with target labels B, C and D, for this we will need a keyboard input and the `keyPressed()` function. When you press the letter on the keyboard (A, B, C or D) you activate that target `label` letter. The function `toUpperCase()` will make it uppercase for you, so you don't need to set caps lock, the function does it for you.

```
let nn
let targetLabel = 'A'

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

function keyPressed()
{
  targetLabel = key.toUpperCase()
}

function mousePressed()
{
  fill(0, 0, 255, 100)
```

```
  circle(mouseX, mouseY, 25)
  textAlign(CENTER, CENTER)
  stroke(0)
  text(targetLabel, mouseX, mouseY)
}
```

🗒️ Notes

Have about 10 (data points) per letter and try to cluster them together as illustrated in the image below. The target label has the default letter A but to get the other target labels, press the B, C or D keys first before clicking on the canvas.

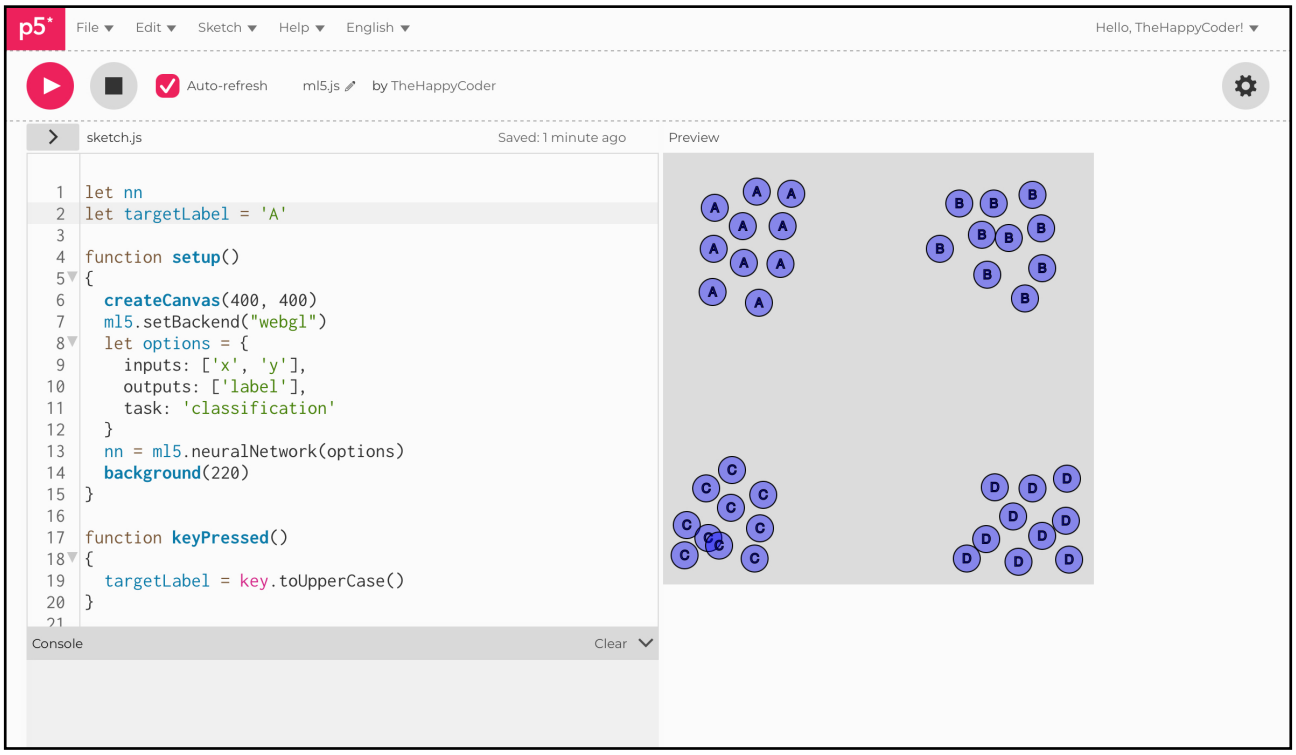
🌻 Challenges

1. You could have any labels for your targets
2. You could have more or less than the four suggested

🛠️ Code Explanation

`let targetLabel = 'A'`	Defines a variable for the target label and initialises it to A
`function keyPressed()`	This function waits for a key to be pressed before actioning the code in the function
`targetLabel = key.toUpperCase()`	Whatever key is pressed that is the new target label until another key is pressed, it capitalises any key pressed
`text(targetLabel, mouseX, mouseY)`	Now it puts the default key A unless another is pressed

Figure A6.5


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 1 minute ago
Preview
let nn
let targetLabel = 'A'
function setup()
{
createCanvas(400, 400)
ml5.setBackend("webgl")
let options = {
inputs: ['x', 'y'],
outputs: ['label'],
task: 'classification'
}
nn = ml5.neuralNetwork(options)
background(220)
}
function keyPressed()
{
targetLabel = key.toUpperCase()
}
Console
Clear

Sketch A6.6 collect training data

Now we are going to do a bit of rearranging because we want to collect the data points every time we click on the canvas. To do that we move the inputs and outputs object arrays into the `mousePressed()` function and then add the data to the neural network each time the mouse is clicked, inputting the target `label`. We use the ml5.js function `nn.addData()`.

```
let nn
let targetLabel = 'A'

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

function keyPressed()
{
  targetLabel = key.toUpperCase()
}

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

```
    y: mouseY
  }
  let target = {
    label: targetLabel
  }
  nn.addData(inputs, target)
  fill(0, 0, 255, 100)
  circle(mouseX, mouseY, 25)
  textAlign(CENTER, CENTER)
  stroke(0)
  text(targetLabel, mouseX, mouseY)
}
```

🗒️ Notes

Just to clarify, every time you click and add a circle (with a target label letter) to the canvas that data is passed to the neural network. You may notice that we didn't call the outputs by that name. This is because we created a new variable name (`target`) for the output.

🌻 Challenge

If you want to see the data produced then add this line of code: `console.log(inputs, targetLabel)`
at the end of the `mousePressed()` function, this is not essential but interesting (do one for each letter)

🛠️ Code Explanation

`nn.addData(inputs, target)`	This function adds the data to the neural network. Inputs are the x and y co-ordinates and the output is called target which is the label (letter)

Sketch A6.7 button

We are going to use the data to train our model but first we are going to create a button that when we press it we `train()` the model. But first will just get a console message to make sure the button is working.

```
let nn
let targetLabel = 'A'
let button

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    task: 'classification'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
  targetLabel = key.toUpperCase()
}

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

```
    y: mouseY
  }
  let target = {
    label: targetLabel
  }
  nn.addData(inputs, target)
  fill(0, 0, 255, 100)
  circle(mouseX, mouseY, 25)
  textAlign(CENTER, CENTER)
  stroke(0)
  text(targetLabel, mouseX, mouseY)
}
```

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

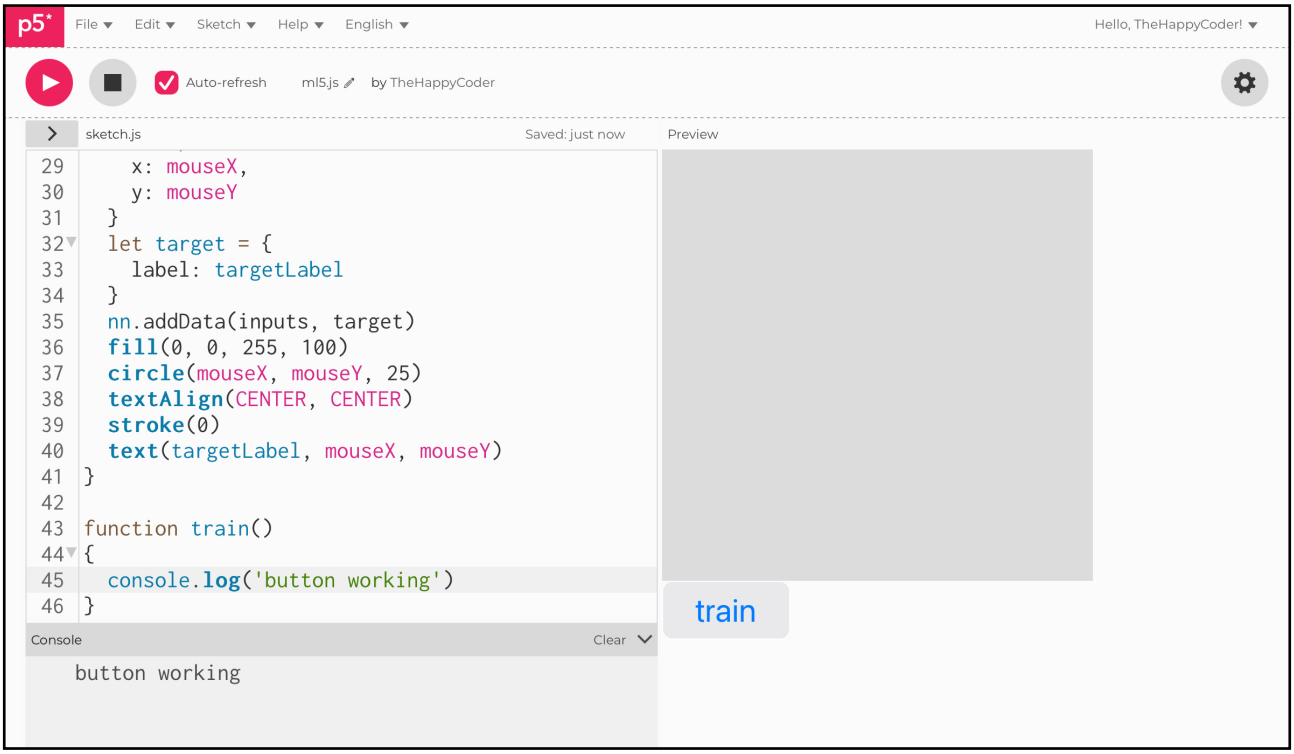
🗒️ Notes

This just creates the button and checks to see if it is working so you should get a message to that effect in the console. This is just temporary until we use the button to start the training process after we have our four sets of ten data points (target labels)

🛠️ Code Explanation

`button.mousePressed(train)`	We activate the train() function when we clcik the button
`console.log('button working')`	This is where we are going to train our neural network

Figure A6.7


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: just now
Preview
x: mouseX,
y: mouseY
}
let target = {
label: targetLabel
}
nn.addData(inputs, target)
fill(0, 0, 255, 100)
circle(mouseX, mouseY, 25)
textAlign(CENTER, CENTER)
stroke(0)
text(targetLabel, mouseX, mouseY)
}
function train()
{
console.log('button working')
}
train
Console
Clear
button working

Sketch A6.8 training

If the button works OK then remove the console log and add the debug. In the train() function we start training the model. Add the data points (circles) for A, B, C and D before clicking on the train button.

❗ We need to add a comma , after the word 'classification'.

```
let nn
let targetLabel = 'A'
let button

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

function keyPressed()
{
  targetLabel = key.toUpperCase()
}

function mousePressed()
{
```

```
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  let target = {
    label: targetLabel
  }
  nn.addData(inputs, target)
  fill(0, 0, 255, 100)
  circle(mouseX, mouseY, 25)
  textAlign(CENTER, CENTER)
  stroke(0)
  text(targetLabel, mouseX, mouseY)
}

function train()
{
  nn.train()
}
```

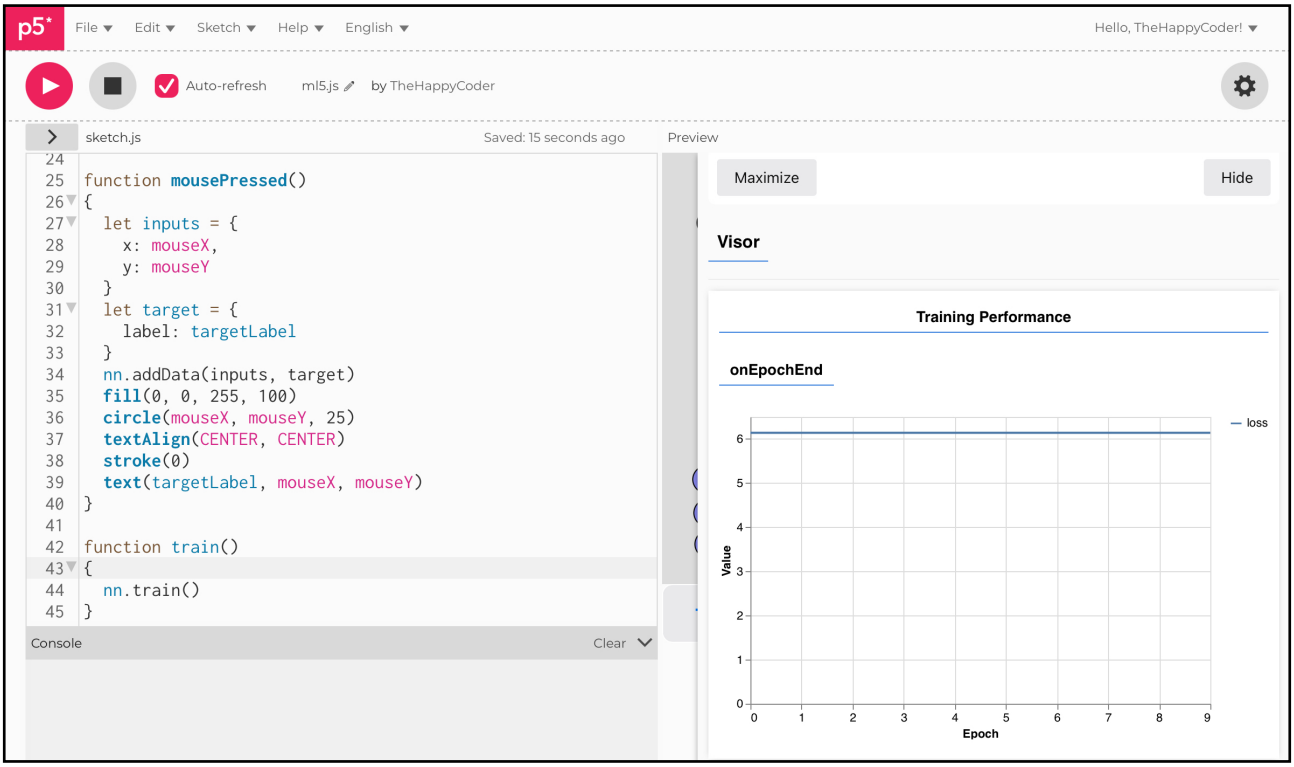
🗒️ Notes

When we do this we get the following results which are extremely poor. In this example it was around 6 or 7, not good at all. There is a very good reason for this and it is all to do with the size of the data points we have collected. The model works an the basis that all the values (including the inputs) are small, this is called `normalisation`. This was deliberate to illustrate what difference it makes and how essential it is. The neural network only really works with values between `0` and `1` (or `-1` and `+1`). We have `x` and `y` values in the region of up to `400`. We use the normalisation function `normalizeData()`

🛠️ Code Explanation

`nn.train()`	Instead of a console log we train the neural network instead

Figure A6.8


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 15 seconds ago
Preview
function mousePressed()
{
let inputs = {
x: mouseX,
y: mouseY
}
let target = {
label: targetLabel
}
nn.addData(inputs, target)
fill(0, 0, 255, 100)
circle(mouseX, mouseY, 25)
textAlign(CENTER, CENTER)
stroke(0)
text(targetLabel, mouseX, mouseY)
}
function train()
{
nn.train()
}
Console
Clear
Maximize
Hide
Visor
Training Performance
onEpochEnd
loss
Value
Epoch

Sketch A6.9 normalisation

We need to improve things a little bit. Two things we are going to do, one is to normalise the data which means putting the values between 0 and 1, for that we use the `normalizeData()` function, the other thing is to specify the number of epochs. In the example from the training performance above there were only ten epochs, we are increasing the number to 100. We also need a callback function: `finishedTraining()`

```
let nn
let targetLabel = 'A'
let button

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

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

```
function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  let target = {
    label: targetLabel
  }
  nn.addData(inputs, target)
  fill(0, 0, 255, 100)
  circle(mouseX, mouseY, 25)
  textAlign(CENTER, CENTER)
  stroke(0)
  text(targetLabel, mouseX, mouseY)
}

function train()
{
  nn.normalizeData()
  let options = {
    epochs: 100
  }
  nn.train(options, finishedTraining)
}

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

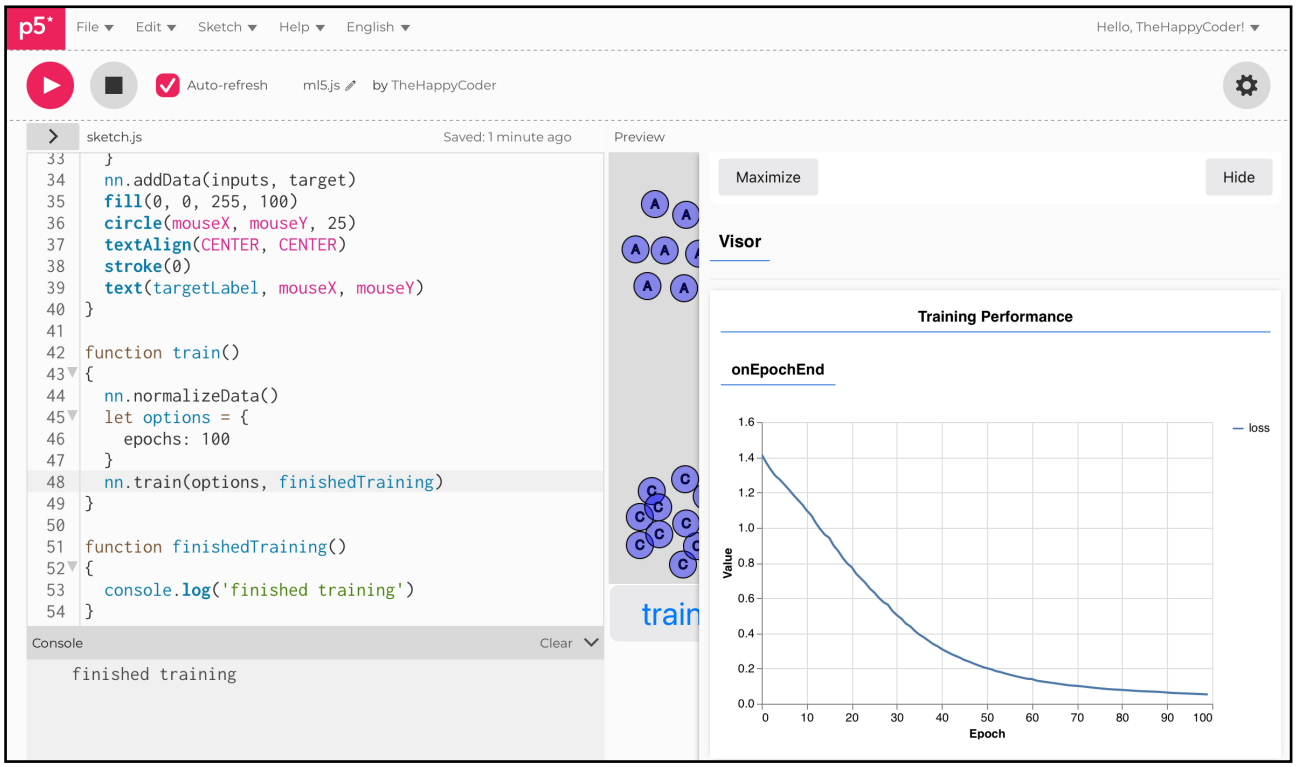
🗒️ Notes

You will get a `finished training` comment in the console, also the graph will look considerably better, very close to zero.

🛠️ Code Explanation

`nn.normalizeData()`	Normalises the data between 0 and 1
`epochs: 100`	Go through the whole training data 100 times rather than just the 10

Figure A6.9


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 1 minute ago
Preview
}
nn.addData(inputs, target)
fill(0, 0, 255, 100)
circle(mouseX, mouseY, 25)
textAlign(CENTER, CENTER)
stroke(0)
text(targetLabel, mouseX, mouseY)
}
function train()
{
nn.normalizeData()
let options = {
epochs: 100
}
nn.train(options, finishedTraining)
}
function finishedTraining()
{
console.log('finished training')
}
Console
Clear
finished training
train
Maximize
Hide
Visor
Training Performance
onEpochEnd
loss
Value
Epoch
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
70
80
90
100

Sketch A6.10 adding a bit of colour

Before we go any further I want to add a bit of colour to the proceedings. Each letter will have a given colour:

- A is blue
- B is red
- C is green
- D is yellow

This will give a more visual prediction later

```
let nn
let targetLabel = 'A'
let button

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

function keyPressed()
{
  targetLabel = key.toUpperCase()
```

```
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  let target = {
    label: targetLabel
  }
  nn.addData(inputs, target)
  if (targetLabel === 'A')
  {
    fill(0, 0, 255, 100)
  }
  if (targetLabel === 'B')
  {
    fill(255, 0, 0, 100)
  }
  if (targetLabel === 'C')
  {
    fill(0, 255, 0, 100)
  }
  if (targetLabel === 'D')
  {
    fill(255, 255, 0, 100)
  }
  circle(mouseX, mouseY, 25)
  textAlign(CENTER, CENTER)
  stroke(0)
  text(targetLabel, mouseX, mouseY)
}
```

```
function train()
{
  nn.normalizeData()
  let options = {
    epochs: 100
  }
  nn.train(options, finishedTraining)
}

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

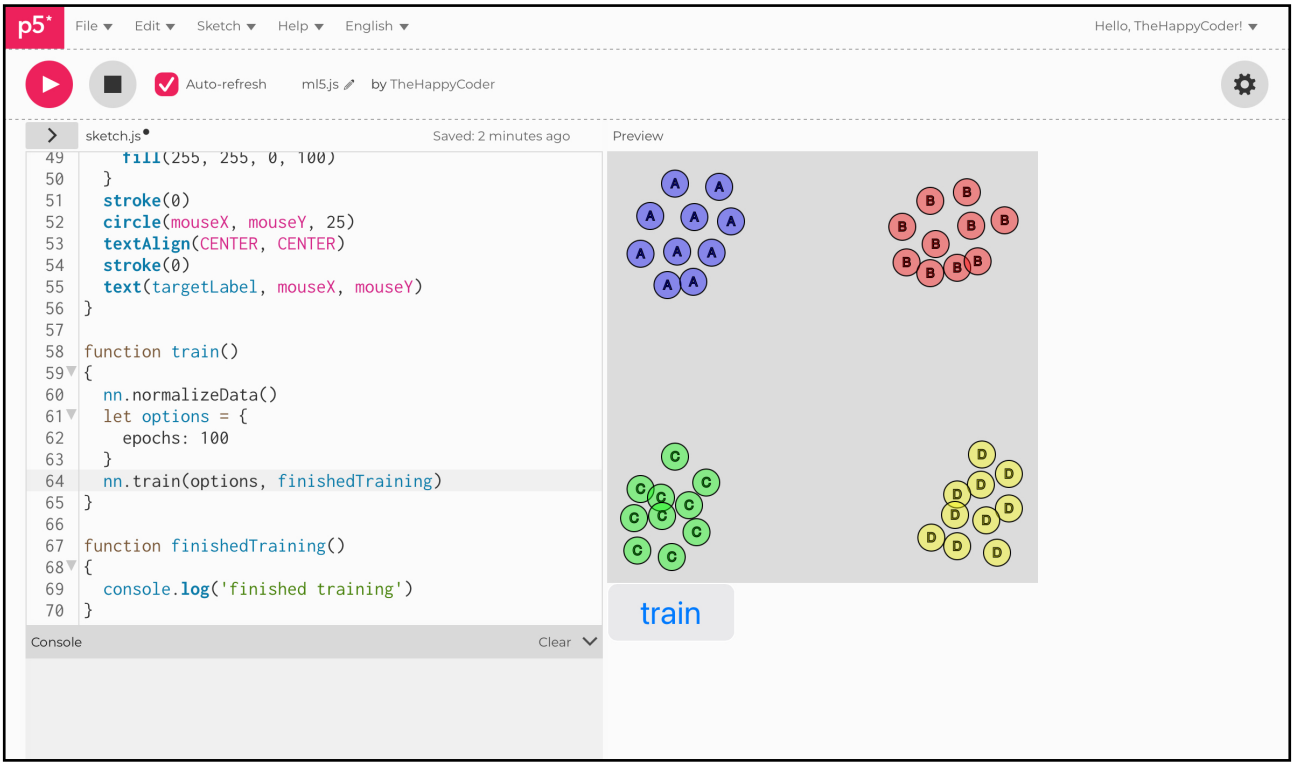
🗒️ Notes

We have a separate colour for each cluster of data points, we just use several `if()` statements. The `===` means it has to be exactly that string

🛠️ Code Explanation

`if (targetLabel === 'A')`	If the label is A then use the following colour fill

Figure A6.10


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 2 minutes ago
Preview
fill(255, 255, 0, 100)
}
stroke(0)
circle(mouseX, mouseY, 25)
textAlign(CENTER, CENTER)
stroke(0)
text(targetLabel, mouseX, mouseY)
}
function train()
{
nn.normalizeData()
let options = {
epochs: 100
}
nn.train(options, finishedTraining)
}
function finishedTraining()
{
console.log('finished training')
}
Console
Clear
train

Sketch A6.11 a state of mind

We create a simple variable expression called `state`, it helps us keep track of where we are in the process, we will have three states. The first state is to collect the data (`collection`), we then train the model on that data (`training`) and finally we predict what happens when we click somewhere on the canvas (`predicting`). In `mousePressed()` we move a whole block of code inside an `if()` statement.

```
let nn
let targetLabel = 'A'
let button
let state = 'collection'

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

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

```
function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let target = {
    label: targetLabel
    }
    nn.addData(inputs, target)
    if (targetLabel === 'A')
    {
      fill(0, 0, 255, 100)
    }
    if (targetLabel === 'B')
    {
      fill(255, 0, 0, 100)
    }
    if (targetLabel === 'C')
    {
      fill(0, 255, 0, 100)
    }
    if (targetLabel === 'D')
    {
      fill(255, 255, 0, 100)
    }
    stroke(0)
    circle(mouseX, mouseY, 25)
    textAlign(CENTER, CENTER)
    stroke(0)
```

```
    text(targetLabel, mouseX, mouseY)
  }
}

function train()
{
  state = 'training'
  nn.normalizeData()
  let options = {
    epochs: 100
  }
  nn.train(options, finishedTraining)
}

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

🗒️ Notes

A bit of refactoring not really adding lots of new code. The change of the state is something like changing the mode setting on a piece of equipment.

🛠️ Code Explanation

`let state = 'collection'`	Define and initialise variable string to collection
`state = 'training'`	Change the string to training
`state = 'prediction'`	Change again to prediction

Sketch A6.12 classifying

The final stage is the prediction or classification. We want to predict what every pixel of the canvas would be if you were to create a new data point. We create a new callback function called `gotResults()` but only after we have run the classifier.

When you have trained the model click on the canvas (anywhere) and as you move the mouse around you will start colouring in the predicted colour for that particular coordinate on the canvas. We are filling the circle with the predicted colour at that coordinate on the canvas.

```
let nn
let targetLabel = 'A'
let button
let state = 'collection'

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

function keyPressed()
{
```

```
  targetLabel = key.toUpperCase()
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let target = {
    label: targetLabel
    }
    nn.addData(inputs, target)
    if (targetLabel === 'A')
    {
      fill(0, 0, 255, 100)
    }
    if (targetLabel === 'B')
    {
      fill(255, 0, 0, 100)
    }
    if (targetLabel === 'C')
    {
      fill(0, 255, 0, 100)
    }
    if (targetLabel === 'D')
    {
      fill(255, 255, 0, 100)
    }
    stroke(0)
    circle(mouseX, mouseY, 25)
```

```
    textAlign(CENTER, CENTER)
    stroke(0)
    text(targetLabel, mouseX, mouseY)
  }
  else if (state == 'prediction')
  {
    nn.classify(inputs, gotResults)
  }
}

function train()
{
  state = 'training'
  nn.normalizeData()
  let options = {
    epochs: 100
  }
  nn.train(options, finishedTraining)
}

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

function gotResults(results)
{
  mousePressed()
  if (results[0].label === 'A')
  {
    fill(0, 0, 255, 100)
  }
```

```
  if (results[0].label === 'B')
  {
    fill(255, 0, 0, 100)
  }
  if (results[0].label === 'C')
  {
    fill(0, 255, 0, 100)
  }
  if (results[0].label === 'D')
  {
    fill(255, 255, 0, 100)
  }
  noStroke()
  circle(mouseX, mouseY, 25)
}
```

🗒️ Notes

Each coordinate will have a predicted colour value depending on where that particular pixel point is. The results will classify it and draw the corresponding coloured circle. Think of it as filling in all the blanks where you had no data, it has interpreted the rest of the canvas for you.

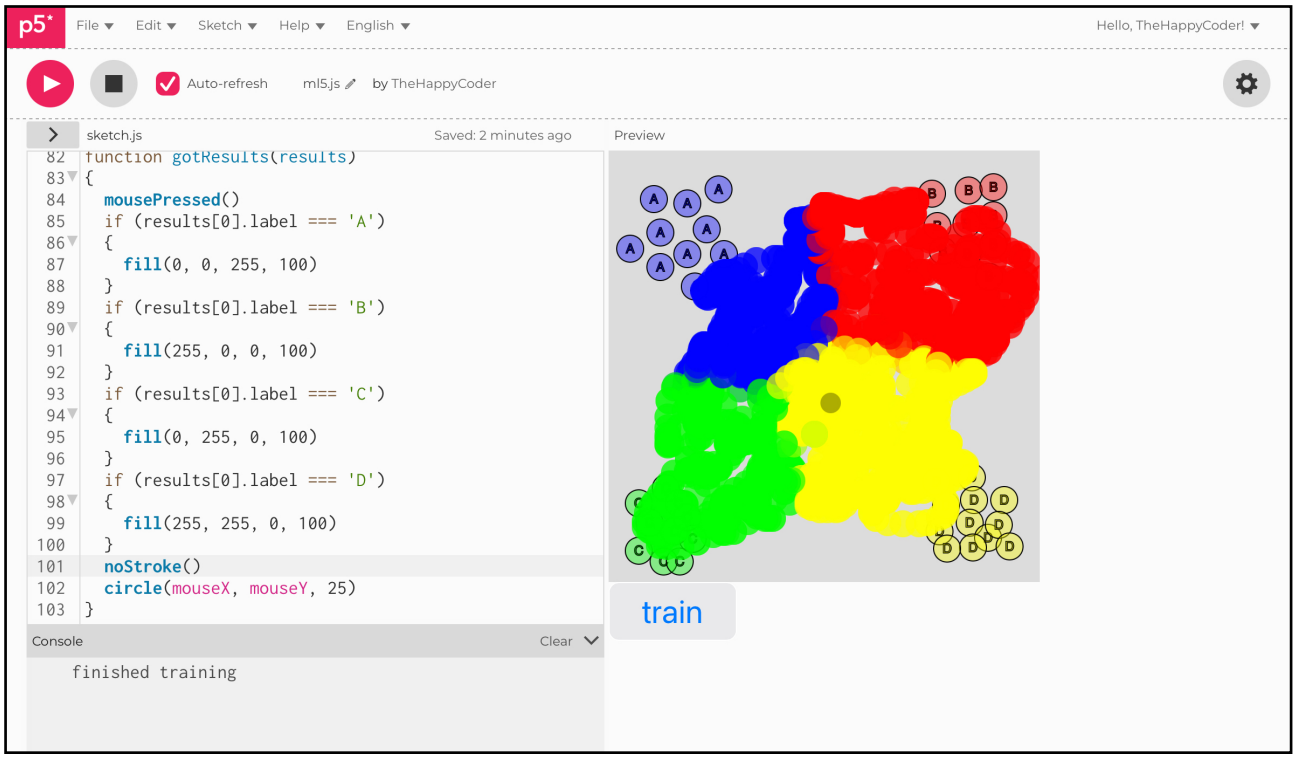
🌻 Challenge

Try different shapes of data points e.g a swirl, concentric circles (you may need a lot more data points). Also try drawing pixels instead of circles (although it might take longer to fill the canvas).

🛠️ Code Explanation

`nn.classify(inputs, gotResults)`	The classify() function takes the inputs and returns the results whether it is A, B, C or D
`if (results[0].label === 'A')`	It makes the prediction for that x, y value input and checks to see whether it is A, B, C or D and colours appropriately

Figure A6.12


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 2 minutes ago
Preview
function gotResults(results)
{
mousePressed()
if (results[0].label === 'A')
{
fill(0, 0, 255, 100)
}
if (results[0].label === 'B')
{
fill(255, 0, 0, 100)
}
if (results[0].label === 'C')
{
fill(0, 255, 0, 100)
}
if (results[0].label === 'D')
{
fill(255, 255, 0, 100)
}
noStroke()
circle(mouseX, mouseY, 25)
}
Console
Clear
finished training
train