07 Pie chart useInterval Teachable machine animations and ML5

<https://www.youtube.com/watch?v=HLpw0JFY4-E&t=2s>

Uses machine learning model to tell who is in front of camera.

Full when there, also empty if eyes covered

**Google Teachable Machine** –

<https://teachablemachine.withgoogle.com/>

can create, save and export models (TensorFlow.js)

Image model – 2 classes one there other not there can preview – created model and export

Put the unzipped model in create react app pubic folder

**ml5 is a machine learning library**

npm <https://www.npmjs.com/package/ml5>

git hub home page <https://github.com/ml5js/ml5-library#readme>

The **useInterval** hook is a custom hook

<https://overreacted.io/making-setinterval-declarative-with-react-hooks/>

Is very similar to set interval but less the overhead of using set interval in a react component.

**Note about adding x axis**

[Eddcakes](https://www.youtube.com/channel/UCCi2TBOsJnAuHKfYzZ5FhKg)[3 months ago](https://www.youtube.com/watch?v=HLpw0JFY4-E&lc=Ugy1QH6LV_fremDqjeF4AaABAg)

Really liking this series, you are easy to listen to and explain things very well. Any tips on marking the x axis for a gague chart? I've been trying to add a start value and end value and current value as a hover.



REPLY

Hide reply

[The Muratorium](https://www.youtube.com/channel/UCKfcSawDV88REF9jVwqqbag)

[The Muratorium](https://www.youtube.com/channel/UCKfcSawDV88REF9jVwqqbag)

[3 months ago (edited)](https://www.youtube.com/watch?v=HLpw0JFY4-E&lc=Ugy1QH6LV_fremDqjeF4AaABAg.92zdUY1E8OY92zhk_jJFte)

Thank you Eddcakes! I would suggest you use this little library, if you don't want to worry about it too much, as D3 doesn't provide something like this out of the box: [https://github.com/vasturiano/d3-radial-axis](https://www.youtube.com/redirect?event=comments&redir_token=IHMlgwfusPyoDJKlKjfiLoaZgpJ8MTU4NzgxNjQ5M0AxNTg3NzMwMDkz&stzid=Ugy1QH6LV_fremDqjeF4AaABAg.92zdUY1E8OY92zhk_jJFte&q=https%3A%2F%2Fgithub.com%2Fvasturiano%2Fd3-radial-axis) But if you do wanna create your own axis, you need to rotate each label, and then translate it to the correct position. And that position you would have to calculate with .centroid() and a "fictional arc" you just use for positioning. A good example for that calculation is this one: [http://bl.ocks.org/dbuezas/9306799](https://www.youtube.com/redirect?event=comments&redir_token=IHMlgwfusPyoDJKlKjfiLoaZgpJ8MTU4NzgxNjQ5M0AxNTg3NzMwMDkz&stzid=Ugy1QH6LV_fremDqjeF4AaABAg.92zdUY1E8OY92zhk_jJFte&q=http%3A%2F%2Fbl.ocks.org%2Fdbuezas%2F9306799) The "outerArc" in that example is the "fictional arc" that is just used to position the labels, and it is never used to render anything.

Show less

REPLY

Commented App.js

import React, { useEffect, useRef, useState } from "react";

import "./App.css";

import ml5 from "ml5";

import PieChart from "./PieChart";

import useInterval from "./useInterval";

*let* classifier;

*function* App() {

*const* videoRef = useRef();

*const* [pieData, setPieData] = useState([0.5, 0.5]);

*const* [shouldClassify, setShouldClassify] = useState(false);

  // When the DOM elements have been rendered

  useEffect(() *=>* {

    // telling ml5 to load the imageClassifier located in the public folder

    classifier = ml5.imageClassifier("./my-model/model.json", () *=>* {

      // when the modle is ready this call back will be triggered

      // this accesses the web cam of the user

      navigator.mediaDevices

        .getUserMedia({ video: true, audio: false })

        .then((*stream*) *=>* {

          // connect the web cam with the video element defined in the return

          videoRef.current.srcObject = stream;

          videoRef.current.play();

        });

    });

  }, []);

  // source of this custom hook  https://overreacted.io/making-setinterval-declarative-with-react-hooks/

  useInterval(() *=>* {

    // every 1/2 seconds this use interval hook will try to clasify what

    //is in the video element and tell if image is there or not

    if (classifier && shouldClassify) {

      classifier.classify(videoRef.current, (*error*, *results*) *=>* {

        if (error) {

          console.error(error);

          return;

        }

        // when this clasification is done getting a results array

        // ml5 is telling us how confident the machine is that image is there

        // and how confident the machine is that image is not there

        // the result array is sorted by confidence so that the

        //first element in the confidence array is the highest confidence

        /// this is not what we want so this sort done to sort by the label

        // so that they do not change based on the confidence

        results.sort((*a*, *b*) *=>* b.label.localeCompare(a.label));

        // save with the setPieData function in useState and pass to PieChart in return

        setPieData(results.map((*entry*) *=>* entry.confidence));

      });

    }

  }, 500);

  return (

    <React.Fragment>

      <h1>

        Is Norwal there? <br />

        <small>

          [{pieData[0].toFixed(2)}, {pieData[1].toFixed(2)}]

        </small>

      </h1>

      <PieChart data={pieData} />

      <button onClick={() *=>* setShouldClassify(!shouldClassify)}>

        {shouldClassify ? "Stop classifying" : "Start classifying"}

      </button>

    </React.Fragment>

  );

}

export default App;

Commented PieChart.js

import React, { useRef, useEffect } from "react";

// need to import arc, pie  and interpolate from d3

import { select, arc, pie, interpolate } from "d3";

import useResizeObserver from "./useResizeObserver";

*function* PieChart({ *data* }) {

*const* svgRef = useRef();

*const* wrapperRef = useRef();

*const* dimensions = useResizeObserver(wrapperRef);

  // will be called initially and on every data change

  useEffect(() *=>* {

*const* svg = select(svgRef.current);

    if (!dimensions) return;

    // need 2 path elements in this pie chart.

    // the orange color represents the confedance image is there

    // the grey color is confedace the image is not there

    //////// like 2 slices of pie that are changing size based on confidence

    // the shape of a path element is defined by the attribute d

    // d needs to be attached to each slice of the pie

    // need to attach  the d to the slices while making sure that

    // the belong to the same arc

    // d3 helper methods to help with createing the arc

    // and split it into multiple slices

    // can then extract the d attribte of the slices

    // and attach it to the path elements

    // In order to define d for each slice

    // 1)create arc

    // 2)split into segments

    // 3)derive d attribute

*const* arcGenerator = arc().innerRadius(75).outerRadius(150);

    // pie transforms the data into instructions for the arc

    // so the arc can know at which angle to split into slices

*const* pieGenerator = pie()

      // turn the full circle pie into a 1/2 circle

      // 1 math.PI is 1/2 of a circle

      // the arc starts at the top

      //so -0.5 \* Math.PI says 1/4 circle to the left

      .startAngle(-0.5 \* Math.PI)

      // end at 1/2 circle to the right

      .endAngle(0.5 \* Math.PI)

      // the pieGenerator sorts by value

      // so need to sort null

      .sort(null);

    // getting data from app.js, useing pieGenerator

    // to turn into instructions for generating the path

*const* instructions = pieGenerator(data);

    // d3 general updarte pattern

    svg

      // select all path elements wiht the class of slice

      .selectAll(".slice")

      //synchronize with data,which has been turned into

      //instructions how to generate the path

      .data(instructions)

      // create a path element for each new piece of data

      .join("path")

      // attach the attribute class to every new or updating element

      .attr("class", "slice")

      // make it visible

      // if index is 0 return yellow , if not return grey

      .attr("fill", (*instruction*, *index*) *=>* (index === 0 ? "#ffcc00" : "#eee"))

      // this styling centers the chart ( so it is not in the upper left corner)

      .style(

        "transform",

        `translate(${dimensions.width / 2}px, ${dimensions.height}px)`

      )

      // the transition here is on the d attribute

      // (a complex string)

      .transition()

      // pass instructions to arch generator

      // attrTween is a transition over time

      // need to not return the archGenerator hard coded but

      // return a function - this function tells d3 how to generate

      // the d over time

      .attrTween("d", *function* (*nextInstruction*, *index*) {

        // bonus, which wasn't in video 07:

        // animate chart initially, but setting initial instruction

*const* initialInstruction = pieGenerator([0, 1])[index];

        // need to interpolate

        // interpolate animates all the propertys of an object over time

*const* interpolator = interpolate(

          //need to know last known inst and the next instruction

          // the instructions are objects defining stuff about the arc,slices

          // animate from old instrucition object to the new instruction object

          // animate the propertys from left to right over time

          this.lastInstruction || initialInstruction,

          nextInstruction

        );

        // d3 pick up the next update where the last update has finished

        this.lastInstruction = interpolator(1);

        // this returns t which is the timing or current state

        //of transition is a number between 0 and 1

        // 1 means it is just starting and

        // 0 means the transtion is over

        return *function* (*t*) {

          // return the instructions from the interpolator

          return arcGenerator(interpolator(t));

        };

      });

    // draw the pie

  }, [data, dimensions]);

  return (

    <div ref={wrapperRef} style={{ marginBottom: "2rem" }}>

      <svg ref={svgRef}></svg>

    </div>

  );

}

export default PieChart;