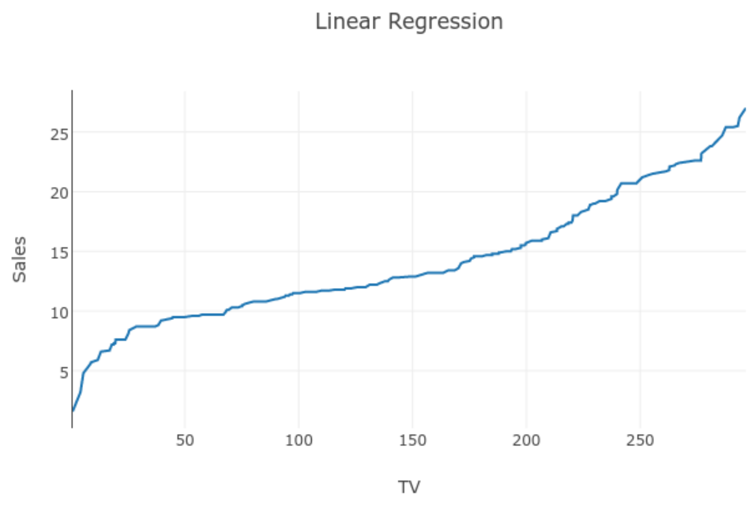
Recommended by you and 805 others

Abhishek Soni Following
I just had an epiphany. https://codeprose.me
Jun 18 · 4 min read

Machine Learning with JavaScript : Part 1

And you thought it wasn't easy



Linear Regression Plot using plot.ly

> This is Part 1 of the ongoing series *Machine Learning with JavaScript*. Here's *Part 2.*

JAVASCRIPT?! Shouldn't I be using Python? Am I out of my mind to try those hefty calculations in JavaScript? Am I trying to act cool by using a language that is not Python or R? scikit-learn doesn't even work in JavaScript? Short Answer: **No. I am not drunk.**

Long Answer: It is possible and I am actually surprised developers haven't given it the attention it deserves. As far as `scikit-learn` is concerned, the JS people have made their own set of libraries to counter it, and I am gonna use one too. But first, a little bit about Machine Learning. Feel free to board this rocket and jump to the code, though.

unsplash.com

According to Arthur Samuel, **Machine Learning** provides computers with the ability to learn without being explicitly programmed. In other words, it gives computer the ability to learn on their own and execute the correct instructions, without you providing them directions.

It has been around for quite a while now, with Google going from mobile-first strategy to AI-first.

Why JavaScript is not mentioned with ML?

1. Slow. (***MYTH!?!***)
2. Matrix Manipulation is difficult.(There are libraries, for example `math.js`)
3. Only concerned with Web Development.(Somewhere out there, **Node.js** is laughing.)
4. Libraries are usually made for Python. (The JS people are not behind)

There are a handful of libraries in JavaScript with pre-made Machine Learning algorithms, such as Linear Regression, SVMs, Naive-Bayes's, et cetera. Here are a few of them,

1. brain.js (Neural Networks)

2. Synaptic (Neural Networks)
3. Natural (Natural Language Processing)
4. ConvNetJS (Convolutional Neural Networks)
5. mljs (A set of sub-libraries with a variety of functions)
6. Neataptic (Neural Networks)
7. Webdnn (Deep Learning)

We are going to use mljs's regression library to perform some linear regression sorcery. All the code is on Github: machine-learning-with-js

Step 1. Install the libraries

```
$ yarn add ml-regression csvtojson
```

Or if you like `npm`

```
$ npm install ml-regression csvtojson
```

`ml-regression` does what the name implies.

`csvtojson` is a fast csv parser for node.js that allows loading `csv` data files and converts it to `JSON` .

Step 2. Initialize the library and load the Data

Download the data file(.csv) from here and put it inside your project.

Assuming you have already initialized an empty npm project, open your `index.js` file and enter the following. (You could copy/paste if you want, but I'd prefer typing it yourself for better understanding.)

```
const ml = require('ml-regression');
const csv = require('csvtojson');
const SLR = ml.SLR; // Simple Linear Regression

const csvFilePath = 'advertising.csv'; // Data
let csvData = [], // parsed Data
    X = [], // Input
```

I put the file at the root of the project, so, if you have put it somewhere else, make sure you update the `csvFilePath` variable likewise.

Pretty neat, eh?

Now we are going to use the `fromFile` method of `csvtojson` to load our data file.

```
csv()
    .fromFile(csvFilePath)
    .on('json', (jsonObj) => {
        csvData.push(jsonObj);
    })
    .on('done', () => {
        dressData(); // To get data points from JSON Ob
```

Step 3. Dressing Data to get it ready for execution

The JSON objects we saved in `csvData` are well, objects, and we need an array of input data points as well as output data points. We are going to run our data through a `dressData` function that will populate our `X` and `y` variables.

```
function dressData() {
    /**
     * One row of the data object looks like:
     * {
     *   TV: "10",
     *   Radio: "100",
     *   Newspaper: "20",
     *   "Sales": "1000"
     * }
     *
     * Hence, while adding the data points,
     * we need to parse the String value as a Float.
     */
    csvData.forEach((row) => {
        X.push(f(row.Radio));
```

Step 4. Train your model and start predicting

Now that our data has successfully been dressed, it's time to train our model.

For this, we are going to write a `performRegression` function:

```
function performRegression() {
    regressionModel = new SLR(X, y); // Train the model
    console.log(regressionModel.toString(3));
    predictOutput();
```

The `regressionModel` has a method `toString` that takes a parameter named precision for floating point outputs.

The `predictOutput` function allows you to enter input values, and outputs the predicted output to your console.

Here's how it looks: (Note that I am using Node.js' readline utility)

```
function predictOutput() {
    rl.question('Enter input X for prediction (Press CT
        console.log(`At X = ${answer}, y =  ${regressic
        predictOutput();
    });
```

And here's the code for adding reading user input:

```
const readline = require('readline'); // For user promp

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
```

Step 5. Hooray! You did it. Pat yourself in the back.

If you followed the steps, this is how your index.js should look:

```
const ml = require('ml-regression');
const csv = require('csvtojson');
const SLR = ml.SLR; // Simple Linear Regression

const csvFilePath = 'advertising.csv'; // Data
let csvData = [], // parsed Data
    X = [], // Input
    y = []; // Output

let regressionModel;

const readline = require('readline'); // For user prom

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

csv()
    .fromFile(csvFilePath)
    .on('json', (jsonObj) => {
        csvData.push(jsonObj);
    })
    .on('done', () => {
        dressData(); // To get data points from JSON O
        performRegression();
    });

function performRegression() {
    regressionModel = new SLR(X, y); // Train the mode
    console.log(regressionModel.toString(3));
    predictOutput();
}

function dressData() {
    /**
     * One row of the data object looks like:
     * {
     *   TV: "10",
     *   Radio: "100",
```

Go to your Terminal and run `node index.js` and it will output something like this:

```
$ node index.js

f(x) = 0.202 * x + 9.31
Enter input X for prediction (Press CTRL+C to exit) : 151.5
At X = 151.5, y =  39.98974927911285
Enter input X for prediction (Press CTRL+C to exit) :
```

Congratulations. You just trained your first Linear Regression Model in JavaScript. (Did you notice the speed?)

If you are excited, go check out Part 2.

> ***PS:*** *I am going to use* `ml` *and other libraries(listed above!) to execute popular machine learning algorithms on various data-sets. Keep an eye on my profile, or you could cut yourself some slack and follow me. :)*

Thanks for reading! If you liked it, hit the **green button** to let others know about how powerful **JS** is and why it *shouldn't be lagging behind* when it comes to Machine Learning.