Using the activity logs to predict user behaviour and improve experience

Alex Hang

Conf42 2025

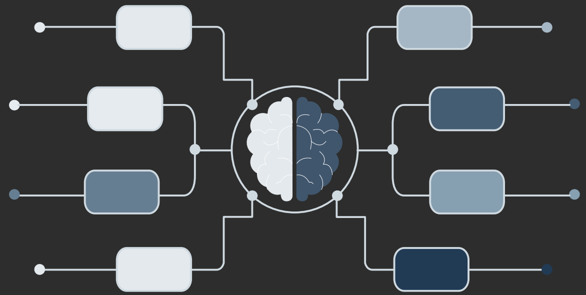


Table of contents

1

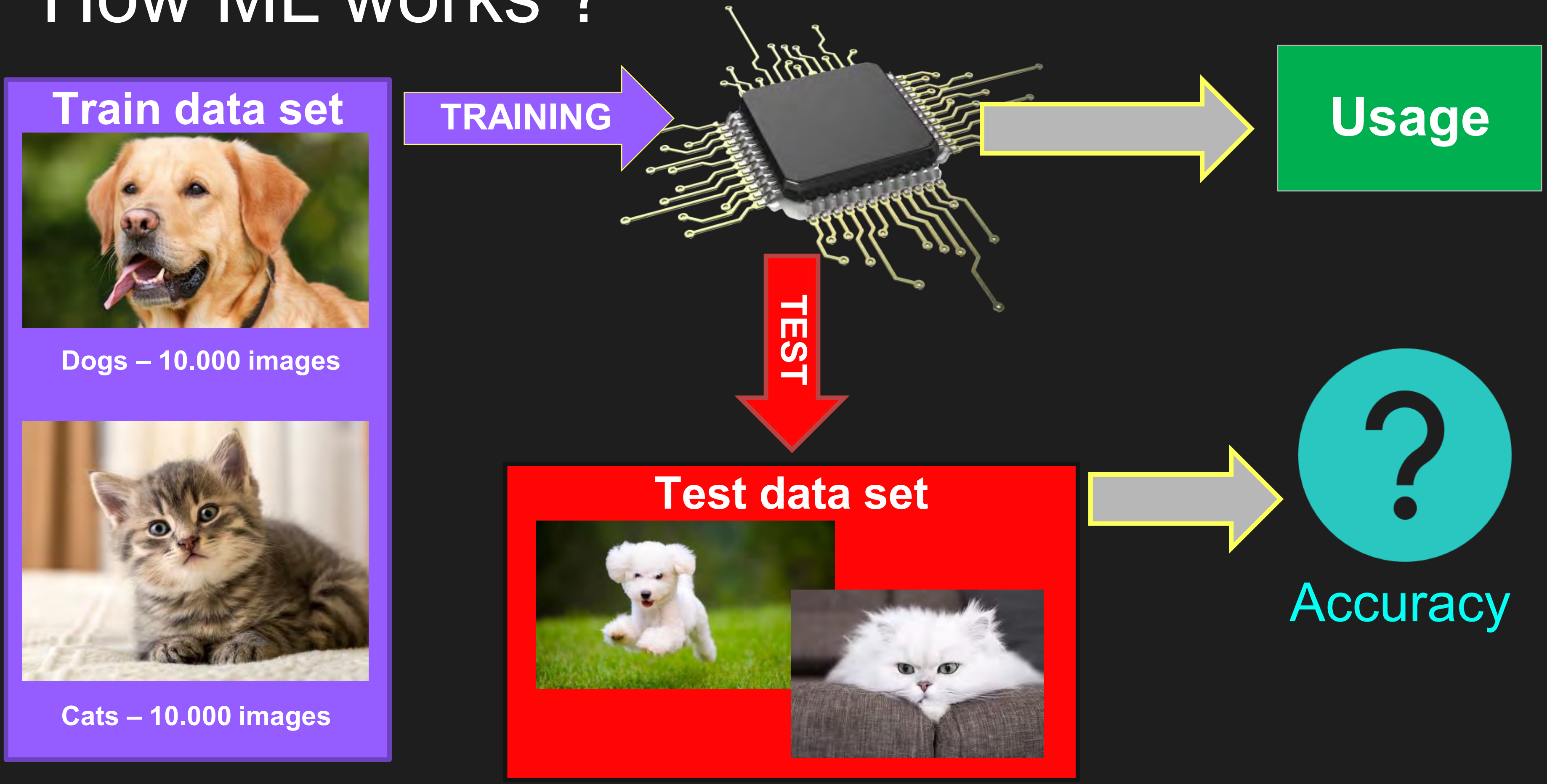
WHAT

So what is AI ? And ML ? And all those fancy names ? And what do they do for me ?

Introduction

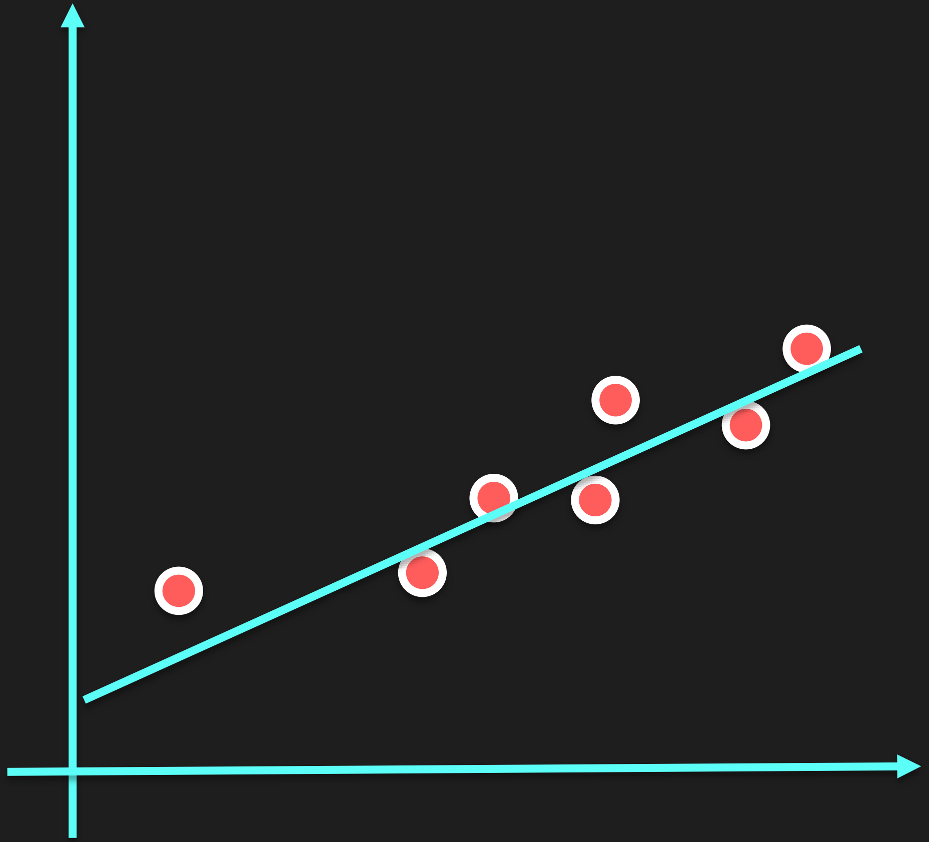
- Activity logs store past actions of our users
- Machine learning models are trained on historic data
- Models such as Neural Networks or LSTM can be used for prediction tasks

How ML works ?

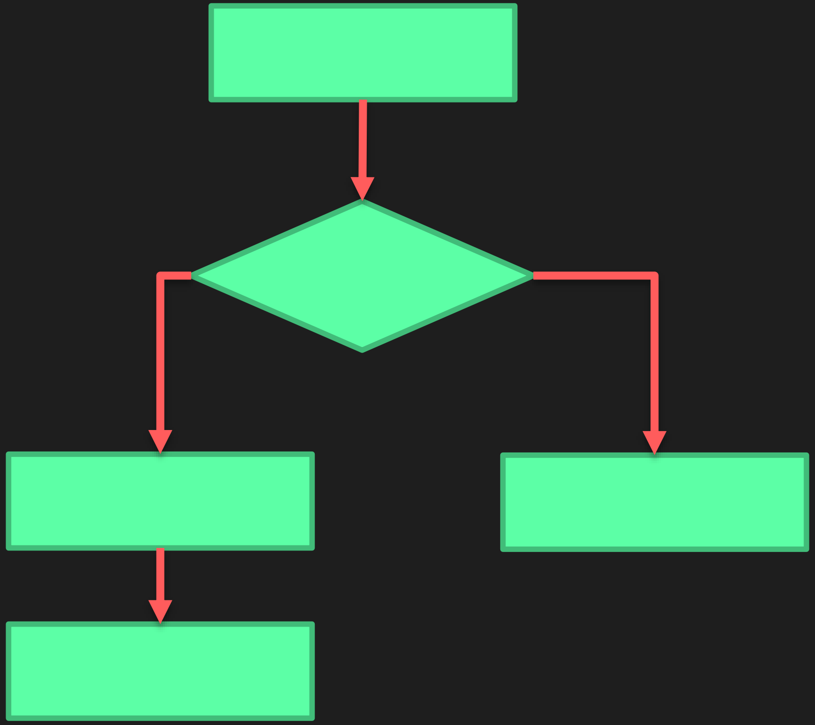

Train data set
Dogs – 10.000 images
Cats – 10.000 images
TRAINING
TEST
Usage
Test data set
Accuracy

Most used ML models

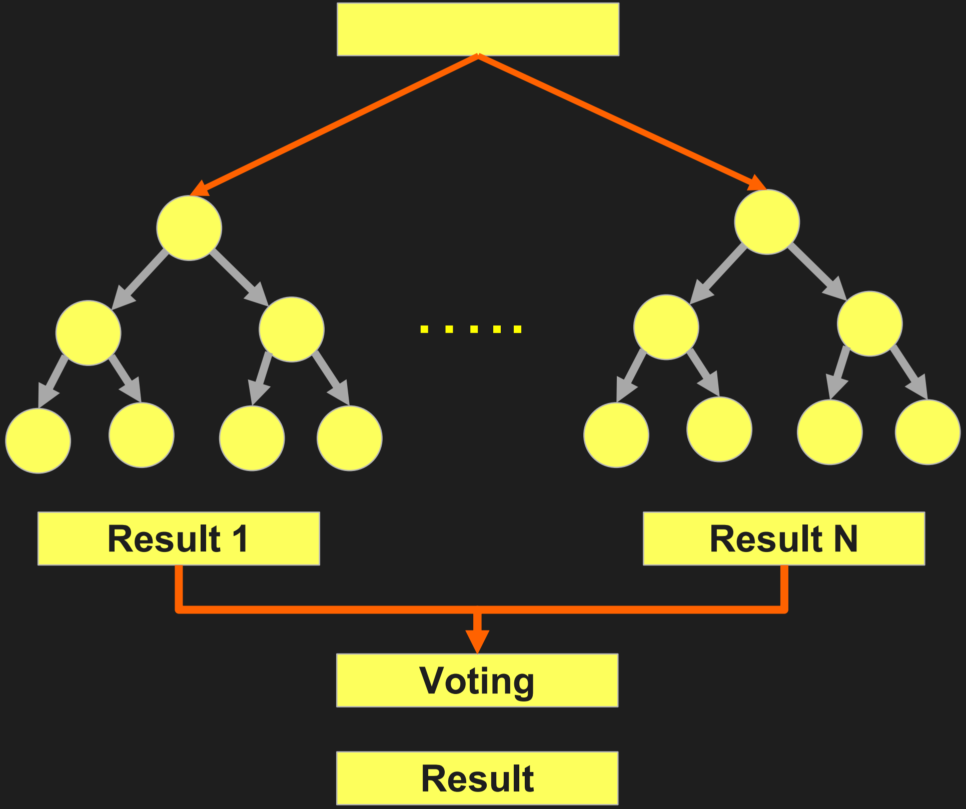
Linear regression



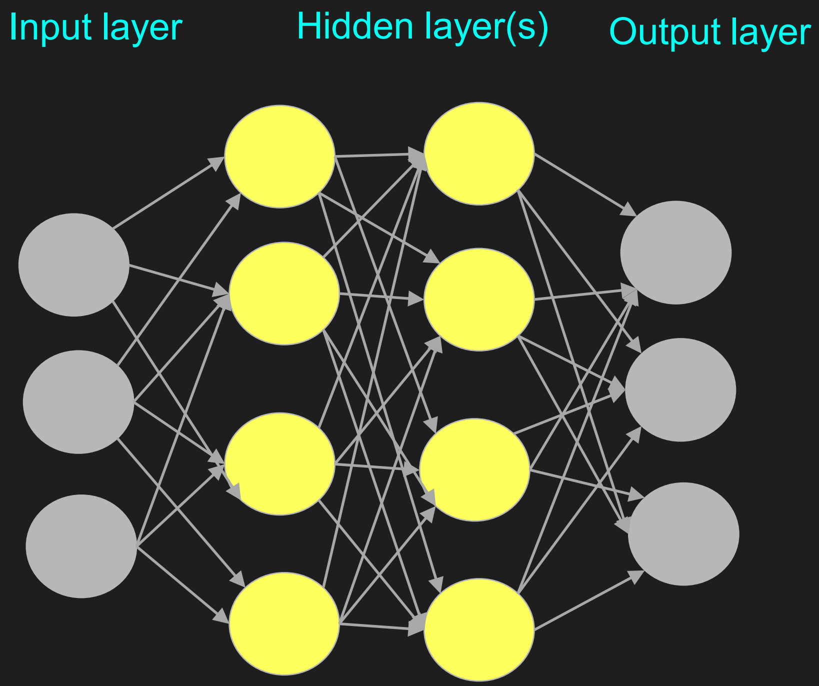
Decision Tree



Random Forest



Neural Network



2

WHY ?

Why would I do this ?

- Personalize app experience
- Optimize common workflows
- Predict user actions to prepare for increased workloads on specific services

3

HOW ?

SETUP

- Given a user activity logs, we need to predict in real time which task a user will perform, so that we can predict server usage per task

- We will build a small neural network to predict the next action of a user

- Other alternatives can be ran on a statistical approach

Tensorflow

Tensorflow

- **Use existing pre-trained models**
- **Retrain existing models**
- **Develop ML models**

- Quickly train systems that use the classical ML models: regression, decision tree classifier neural network, etc.

- Easily adapt existing models to various use-cases

- Can run either on server or in the browser

- Already used by top companies

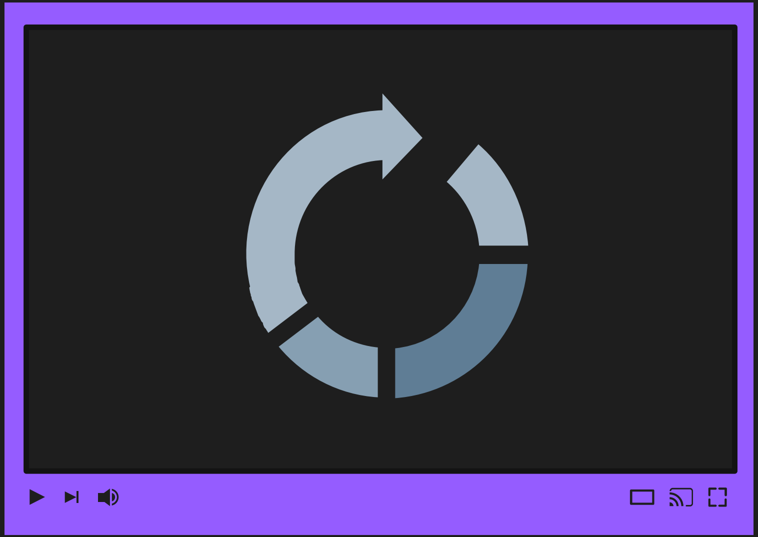
Basic regression example

1. Import TensorflowJS
2. Define a sequential model
3. Add hidden layers to the model
4. Compile the model
5. Define or import training data for the model
6. Train the model using the train data
7. Use the model to predict the value of the function in the specified point

LinearRegression.js

```javascript
import * as tf from '@tensorflow/tfjs';

const model = tf.sequential();

model.add(tf.layers.dense({units: 1, inputShape: [1]}));

model.compile({loss: 'meanSquaredError', optimizer: 'sgd'});

const xs = tf.tensor2d([-1, 0, 1, 2, 3, 4], [6, 1]);
const ys = tf.tensor2d([-3, -1, 1, 3, 5, 7], [6, 1]);

model.fit(xs, ys, {epochs: 250})

model.predict(tf.tensor2d([20], [1, 1])).print()
```

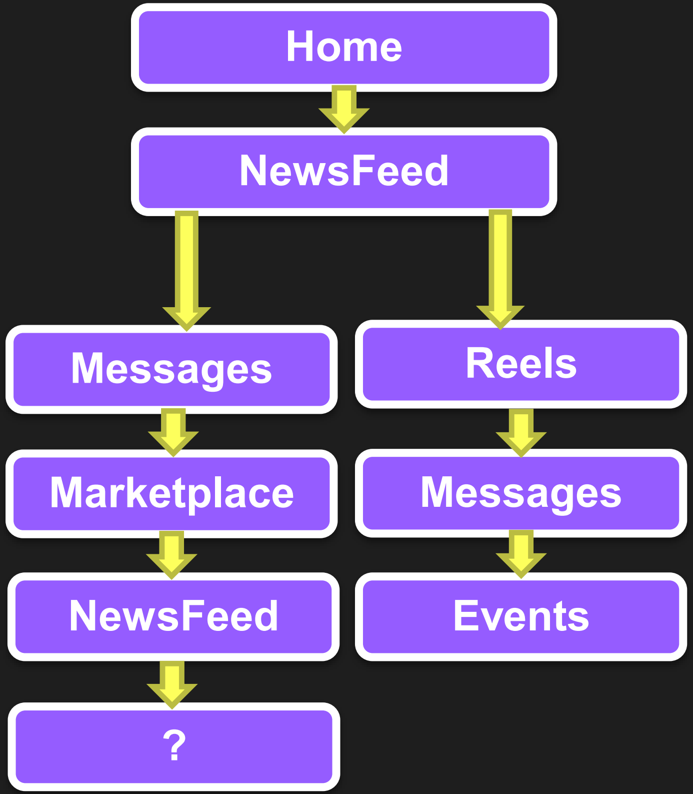
Next page prediction system



Given scenario

- Say that we have an application with multiple pages, and based on the user activity we want to either adapt the UI according to his/her needs or maybe even preload some components by predicting which page is the most likely to be the next one accessed

- As a real-life example, say that a user who usually logs into facebook and then immediately goes to the marketplace and then to the messages section, is expected to follow a similar pattern every time. What if based on the last few pages that you accessed, the app could know which page is the next one that you would like to access, and it could either re-order the sidenav for you, to make the most used pages more accessible or preload them to reduce waiting time.



Implementation plan

1. Gather required data from activity logs
2. Transform data for training
3. Initiate the model
4. Train the model
5. Predict

4

Practical example

Environment

- light-weight NodeJS application as wrapper around the ML app
- Tensorflow model running to predict the next user action
- Predicted user actions used to preload data or ensure server resources are available

Gathering the data

- We will mock the activity logs in our test app by storing the data in memory

```
const data = []

function handlePageClick(page) {
        data.push(page);
        currentPage = page;
 }
```

Also, since the training data is now different, we will re-train the model with the new data

```
const data = []

function handlePageClick(page) {
        data.push(page);
        currentPage = page;
        trainAndRunModel();
 }
```

Overview of the process

- We will discuss each function in the next slides, for now, treat them as pseudocode

```
function trainAndRunModel() {
        transformData();
        initModel();
        trainModel().then(() => {
          predictPage();
        });
      }
```

Prepare data

```
function transformData() {
        uniquePages = Array.from(new Set(data.map((entry) => entry.page)));

        pageIndexMap = new Map(uniquePages.map((page, index) => [page, index]));

        userPageMatrix = tf.zeros([1, uniquePages.length]);
        data.forEach((entry) => {
          const pageIndex = pageIndexMap.get(entry.page);
          userPageMatrix.add(
            tf
              .oneHot(0, uniquePages.length)
              .mul(tf.oneHot(pageIndex, uniquePages.length))
          );
        });
      }
```

Define and compile the model

```
function initModel() {
    model = tf.sequential({
      layers: [
        tf.layers.dense({
          units: 8,
          inputShape: [uniquePages.length],
          activation: "relu",
        }),
        tf.layers.dense({
          units: uniquePages.length,
          activation: "softmax",
        }),
      ],
    });
    model.compile({
      optimizer: "adam",
      loss: "categoricalCrossentropy",
      metrics: ["accuracy"],
    });
  }
```

Note: the parameters presented here might not be the best ones for our use-case, and tweaking the model for obtaining the best results is part of the development.

Usually, trial and error is used until the accuracy of the model is satisfactory and the balance between accuracy and performance is achieved

The params that can be modified are:

- 1st Layer:
  - Activation: RELU/GELU
  - Units
- 2nd Layer:
  - Activation
- Compilation:
  - Optimizer
  - Loss
  - metrics

Train the model

- The most important part is the training of the model
- This takes us from an empty default neural network to a working application

```
async function trainModel() {
        // Train the model
        const labels = userPageMatrix;
        const epochs = 100;
        const batchSize = 2;
        await model.fit(userPageMatrix, labels, {
          epochs: epochs,
          batchSize: batchSize,
        });
      }
```

Parameter tuning:

- epochs
- batchSize

Predict the next page

```
function createEmptyTensorOfSize(x) {
   return tf.tensor2d([[1, ...new Array(x - 1).fill(0)]]);
}

function predictPage() {
  const userHistory = createEmptyTensorOfSize(uniquePages.length);
  const predictions = model.predict(userHistory);
  predictions.print();
  const nextPageIndex = predictions.argMax(1).dataSync()[0];
  printPredictions(predictions);
}
```

Now what to do with the result ?

- Now that we have the result of the prediction, we can use it to predict the next action of the user
- Other use cases
  - Preload assets to ensure loading times are shorter
  - Detect anomalies in user flows

Full code

Init model

Navigate

Process data

Train model

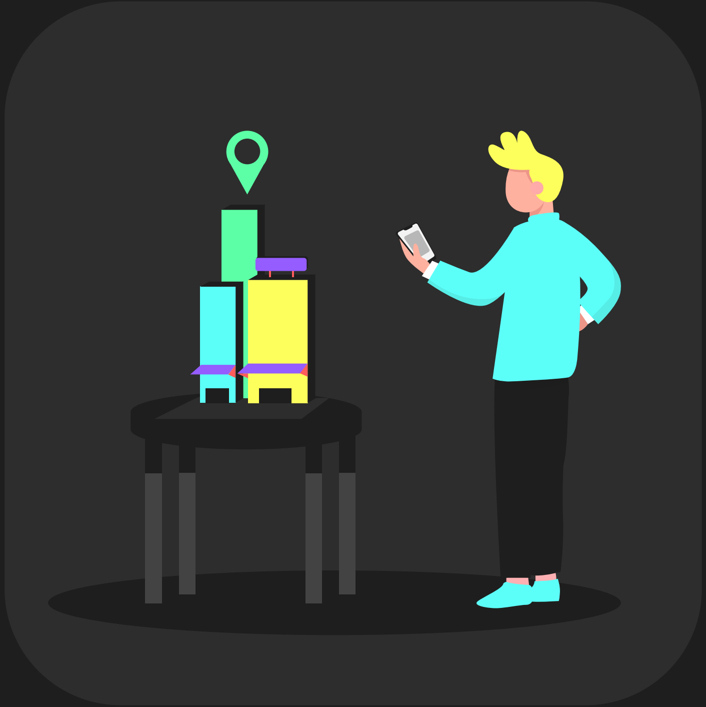
Predict

Print

You can find the full source code on github at

https://github.com/AlexHang/next-page-predictor

Or scan the QR code



Thanks!

Do you have any questions?

alexandruhang@yahoo.com
alexhang.com