Laboratório Nº 1: Preparação do Ambiente de Desenvolvimento em Node JS e Preparação do Conjunto de Treino

Extração Automática de Informação

Prof. Joaquim Filipe
Eng. Filipe Mariano

Objetivos

- Utilização do **Visual Studio Code** como IDE para desenvolvimento em JavaScript/Node JS
- Desenvolvimento em Node JS
- Criação de módulos em Node JS
- Criação de Classes em JavaScript

1. Instalação e Configuração do Ambiente de Desenvolvimento

Como editor de texto será utilizado o editor multi-plataforma Visual Studio Code (VSCode), em que algumas das vantagens da sua utilização podem ser percebidas em http://code.visualstudio.com/docs/editor/whyvscode.

O VSCode encontra-se disponível para download para diversos sistemas operativos em http://code.visualstudio.com/Download.

Para se desenvolver em JavaScript um programa que irá ser executado no servidor necessitamos do ambiente Node.js. O download poderá ser feito https://nodejs.org/en/download, escolhendo a versão LTS - Long Term Support, para a versão de sistema operativo desejada.

2. Servidor - Node JS e Express

2.1. Introdução

Express é uma *framework* para aplicações web que fornece um conjunto de funcionalidades que propícia um desenvolvimento rápido de aplicações web. É uma camada desenvolvida por cima do Node JS para auxiliar a gerir um servidor e respetivas rotas. As suas principais características são:

- Configuração de um *middleware* para responder a pedidos HTTP de forma assíncrona.
- Definição de um roteamento utilizado para executar diferentes ações baseadas num verbo HTTP e num URL.
- Renderização dinâmica de páginas HTML através da passagem de argumentos recorrendo a templates.

2.2. Instalação

Para o desenvolvimento do servidor será necessário efetuar a instalação do módulo `express` através do seguinte comando no terminal:

```
npm install express --save
```

2.3. Iniciar Servidor

Desenvolver um servidor recorrendo a Node JS e Express é bastante simples.

Módulo express

```
var express = require("express");
var app = express();

app.get("/", function (req, res) {
    res.send("Hello World");
});

var server = app.listen(8081, function () {
    var host = server.address().address === "::" ? "localhost" :
server.address().address;
    var port = server.address().port;

    console.log("Example app listening at http://%s:%s", host, port);
});
```

Para iniciar o servidor terá apenas de chamar o comando `node «nome_do_ficheiro»` no terminal ou clicar no ícone de iniciar o *debug*.

2.4. *Request* e *Response*

Após a colocação de um servidor à escuta, desenrola-se um processo de comunicação entre cliente-servidor através dos pedidos do primeiro e as respetivas respostas do segundo. Para compreender esses pedidos é necessário também perceber uma das principais partes que constituem um pedido HTTP, que é como os clientes (por exemplo através de um *browser*) solicitam uma página a um servidor e como essa página é retornada.

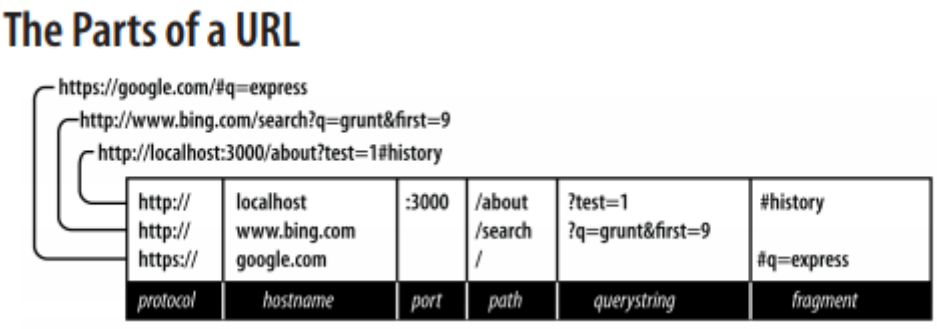

Figura 1: As diversas partes que constituem um URL. Fonte: (Brown, 2014)

De todas as partes que constituem um URL, apenas o *fragment* não é enviado para o servidor, sendo de uso exclusivo do browser para (normalmente) mostrar uma área específica da página retornada.

Quando se constrói um servidor web em Node JS através da *framework* Express, a maioria do que é feito envolve de início um objeto de `request` e por fim um objeto de `response`. Esses dois objetos representam os primeiros parâmetros que costumam ser observados nas funções de *callback* do servidor, desempenhando um importante papel na medida que encapsulam características relativas ao pedido do cliente e à resposta que o servidor irá dar.

Exemplo de função de callback com o objeto *request* e *response* associado a uma rota definida para o verbo GET na raíz:

```
app.get ('/', function (req, res) {
    res.send('Homepage');
});
```

As principais propriedades do objeto *request* são:

- **req.params**: array que contém os parametros associados às rotas.
- **req.body**: um objeto que contém os parâmetros que são enviados no corpo do pedido. Requer que o corpo do pedido seja analisado e convertido para um formato simples de manipular, tal como faz o módulo `body-parser`.
- **req.headers**: cabeçalhos enviados no pedido do cliente.
- **req.route**: informação acerca da rota.
- **req.cookies**: objeto contendo os valores de *cookies* enviados pelo cliente.
- **req.url, req.protocol, req.host, req.path, req.query**: informação associada ao url do pedido (ver figura 1).
- **req.method**: indica o verbo HTTP do pedido do cliente.

As principais propriedades/métodos do objeto *response* são:

- **res.status(código)**: Atualiza o código HTTP a ser enviado ao cliente. Por omissão o código é 200 (OK).
- **res.redirect([código],url)**: Redireciona o cliente para outra página. Por omissão, o código é 302 (Found).
- **res.send(body)**: Envia uma resposta ao cliente cujo cabeçalho de Content-Type enviado será `text/html`. Caso o argumento body enviado seja um objeto ou array ao invés de uma string, a resposta será enviada no formato JSON. Contudo para o fazer é recomendado utilizar o método `res.json`.
- **res.json(json_string)**: Envia JSON para o cliente.
- **res.sendFile(caminho,[callback])**: este método lê um ficheiro especificado pelo argumento path e envia o seu conteúdo para o cliente.
- **res.render(view, callback)**: Renderiza uma *view* de acordo com o motor de templates que foi configurado no Express.

2.5. Roteamento

O roteamento refere-se a determinar como o sistema deverá responder a um pedido de um cliente para um determinado caminho, composto por um URL e um verbo HTTP. As rotas são os caminhos que se pretende que sejam utilizados pelos clientes e nos quais o servidor será programado para dar determinada resposta.

No Express para definir as rotas deve-se utilizar o objeto que possui a criação do express, normalmente denominado `app`. Utilizando essa variável só é necessário depois indicar qual o verbo da rota que se pretende definir, o caminho e a função de *callback* quando chegar o pedido do cliente (`app.verbo(caminho,callback)`).

Exemplo de roteamento através do Express para diferentes caminhos e verbos GET e POST:

```
var express = require('express');
var app = express();

// Pedido de GET para a raíz
app.get('/', function (req, res) {
   res.send(`Verbo: ${req.method} URL: ${req.url}`);
});

// Pedido de POST para a raíz
app.post('/', function (req, res) {
   res.send(`Verbo: ${req.method} URL: ${req.url} BODY: ${req.body}`);
});

var server = app.listen(8081, function () {
    var host = server.address().address === "::" ? "localhost" :
server.address().address;
    var port = server.address().port;

    console.log("Example app listening at http://%s:%s", host, port);
});
```

2.6. Distribuir Ficheiros Estáticos

Para servir ficheiros estáticos o Express fornece um *middleware* interno que permite definir diretorias como estáticas, útil principalmente para guardar imagens, CSS, ficheiros JavaScript, páginas HTML ou qualquer outro recurso que se pretenda que possa ser acedido pelo cliente. Assim, esses ficheiros já poderão ser carregados pelos browsers nas diversas páginas que o servidor web fornecer. Como exemplo, se os ficheiros estáticos de um website estiverem numa pasta cujo nome é `public` a instrução a ser colocada seria:

```
app.use(express.static('public'));
```

Criando a pasta `public` e colocando algumas imagens numa diretoria `images` já será possível visualizar através do browser as imagens disponíveis no servidor para o website.

3. Módulos Locais em Node JS

Para a utilização de módulos no Node é utilizado a função `require` com o nome do módulo como parametro, tal como exemplifica o seguinte excerto de código:

```
var http = require('http');

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.end('Hello World!');
}).listen(8080);
```

No entanto, também existe a abordagem de o módulo estar incluído numa diretoria (módulo local), o que ocorre diversas vezes quando se cria os próprios módulos incorporados no próprio projeto. Neste caso, a definição do módulo é feita através da criação de um ficheiro onde se utiliza código JavaScript tradicional e que quando se pretende dar acesso externamente a algo (função, objeto, constante, etc.) utiliza-se o objeto `module.exports`. A este objeto podem ser acrescentadas propriedades correspondentes às funcionalidades que se pretendem exportar ou pode se substituir, por completo, o objeto exports com o conteúdo do que se pretende exportar. Observe os seguintes exemplos:

Exemplo de exportação de várias propriedades:

```
//Conteúdo do ficheiro circulo.js (exportar várias propriedades):
const PI = 3.14159;

function area(raio) {
    return PI * raio * raio;
}

function perimetro(raio) {
    return PI * 2 * raio;
}

module.exports.area = area;
module.exports.perimetro = perimetro;
```

```
//Conteúdo presente no ficheiro utilizaCirculo.js para utilizar o módulo criado
var circulo = require("./circulo.js"); //o .js pode ser omitido
console.log(circulo.area(10));
console.log(circulo.perimetro(5));
console.log(circulo.PI); //undefined
```

Outro exemplo de exportação:

```
//Conteúdo do ficheiro log.js (todo o objeto exports é substituído pela função):

module.exports = function (mensagem) {
    console.log(mensagem);
};
```

```
//Conteúdo do ficheiro msg.js

var msg = require("./log.js");
msg("Hello World!");
```

Os módulos criados também poderão posteriormente ser publicados no **npm**, e recorrendo ao comando `npm install` poderão ser instalados nos projetos que se pretender.

4. SGBD: MySQL

4.1. SGBD: Instalação do MySQL

Antes de se iniciar o desenvolvimento da aplicação web devemos modelar o problema e desenvolver a base de dados de suporte.
Para este laboratório, iremos utilizar o sistema de gestão de base de dados MySQL. Caso ainda não tenha instalado no seu computador, poderá fazê-lo através do download do MySQL Workbench. No processo de instalação deverá ter especial atenção para a password escolhida para o utilizador root.

4.2. Integração do MySQL com o Node JS

Após a instalação do SGBD e do mesmo estar operacional, podemos acedê-lo através do servidor desenvolvido em Node JS. O principal módulo existente em Node JS para integração do MySQL com o Node JS é o `mysql`. Para instalá-lo basta escrever o seguinte comando:

```
npm install mysql --save
```

Após a instalação do módulo poderá manipular a sua base de dados.

```
var mysql = require('mysql');

var connectionOptions = mysql.createConnection({
    host: "localhost",
    user: "root",
    password: "root",
    database: "EAI-Labs"
});

var connection = mysql.createConnection(connectionOptions);
connection.connect();

connection.query("SELECT id, nome FROM produto", function (err, rows, fields) {
  if (err)
    console.log(err);
  else
    console.log(rows);
});
```

```
connection.end();
```

Nota: Foi exemplificado a utilização do módulo `mysql` por ser um módulo já utilizado em outras unidades curriculares do Mestrado. ***No entanto, fica ao critério dos alunos qual o SGBD que preferem utilizar.***

5. Views - Motor de Templates (EJS)

Os motores de templates integrados com o Express são utilizados para tornar o código, do lado do servidor, mais flexível e menos confuso na geração de páginas HTML. De seguida irão ser apresentados alguns exemplos de utilização do motor de templates `EJS` integrado no `Express`.

5.1. Configuração

O motor de templates `EJS` já vem integrado com o Express. A utilização deste tipo de templates pressupõe a introdução de algumas configurações no ficheiro `index.js` da aplicação gerada, no intuito de suportar a renderização dos templates.

```
var express = require('express');
var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
```

5.2. Templates EJS - Views

Recorrendo a um simples template EJS para uma página html irá se mostrar como a página é renderizada mediante os dados que são passados para a instanciação do template. Essa instanciação é feita através do método `render` do objeto de `response` (normalmente abreviada como `res`) presente na definição das rotas no Express.

Exemplo de template:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
  </head>
  <body>
    <h1><%= title %></h1>
    <p><%= message %></p>
  </body>
</html>
```

Exemplo de instanciação do template:

```
app.get('/', function (req, res) {
    res.render('index', {
        title: 'Hey',
        message: 'Hello there!'
    })
});
```

Exemplo do código fonte no cliente após a renderização:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hey</title>
  </head>
  <body>
    <h1>Hey</h1>
    <p>Hello there!</p>
  </body>
</html>
```

Exemplo de instanciação do template com um ciclo:

```
<!DOCTYPE html>
<html>
    <body>
        <div>
            <ul>
            <% persons.forEach(function(person) { %>
                <li>
                <strong><%= person.name %></strong>
                <%= person.organization %>
                </li>
            <% }); %>
            </ul>
        </div>
    </body>
</html>
```

```
app.get('/', function (req, res) {
    res.render('index', {
        persons: [{
            name: 'Jack Smith',
            organization: 'MIT'
        },{
            name: 'John Smith',
            organization: 'University of Oxford'
```

```
        }]
    })
});
```

6. Text Mining

6.1. Definição

O *Text Mining* pode ser definido como parte de um processo de descoberta de conhecimento, em que procura extrair informações úteis de fontes de dados, através da identificação e exploração de padrões. Essas fontes de dados são representadas por coleções de documentos de texto, e os padrões encontram-se nos dados não estruturados existentes nesses documentos. A descoberta desses padrões pode englobar técnicas sofisticadas de processamento linguístico, estatístico ou até tendo como base uma seleção de *keywords*.

6.2. Diferença entre Data Mining e Text Mining

O *Data Mining* pressupõe que os dados estão armazenados num formato estruturado principalmente em tabelas, enquanto que no *Text Mining* os dados que se possui são texto livre (não estruturado). Neste sentido, são realizadas operações de pré-processamento centradas na identificação e extração de recursos representativos para documentos em linguagem natural. Essas operações de pré-processamento são responsáveis pela transformação de dados não estruturados armazenados em coleções de documentos, num formato mais explicitamente estruturado, não sendo este um aspeto relevante para a maioria dos sistemas de *Data Mining*.

Resumindo, o *Data mining* consiste num leque de abordagens diferentes, na procura de padrões e relações nos dados e o *Text Mining* é um processo que procura tornar documentos de texto não estruturados em informação estruturada útil.

6.3. Aprendizagem Supervisionada de Texto

6.4.1. Definição

A Aprendizagem Supervisionada é uma subcategoria da Aprendizagem Automática e é utilizada quando existe previamente uma definição das categorias de classificação. Desta forma, a Aprendizagem Supervisionada de texto consiste na utilização de conjuntos de textos categorizados para treinar algoritmos na classificação de textos. Funciona de acordo com um princípio de treino e teste.

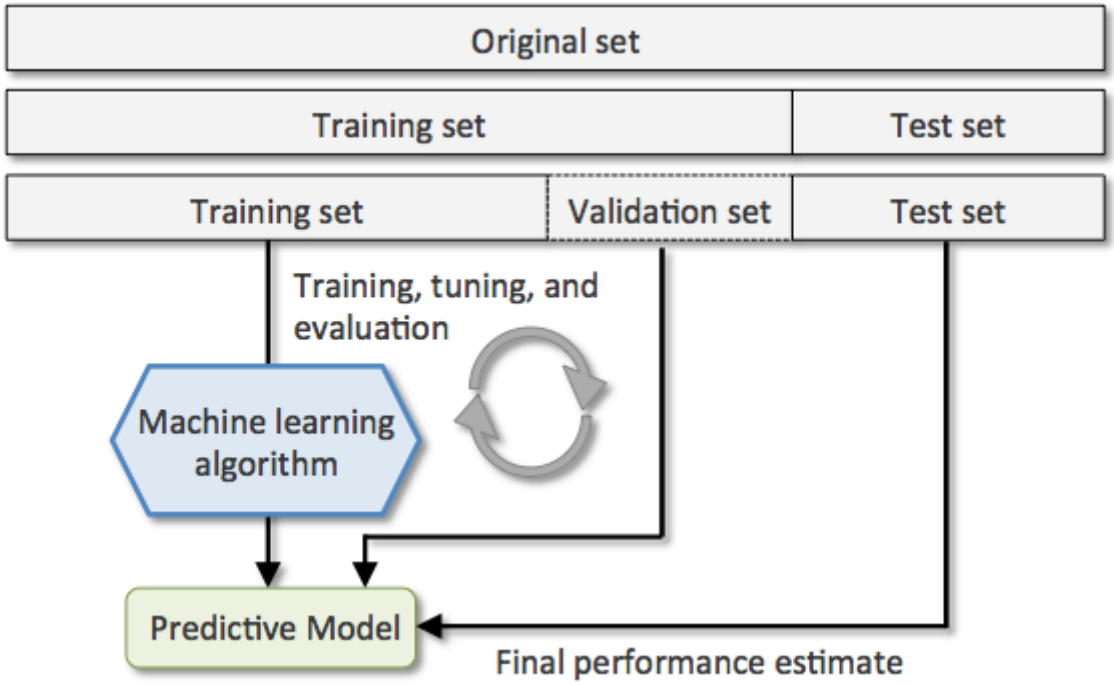

Figura 2: Diagrama a descrever o circuito realizado num sistema de aprendizagem supervisionada.

O classificador é treinado no conjunto de dados categorizado e fornece o *output* desejado, de acordo com as categorias predefinidas. Durante a fase de teste, o classificador é alimentado com dados não observados e classifica os documentos em categorias mediante a fase de treino.

6.4.2. Corpus

Corpus é a denominação em *Text Mining* para a coleção de documentos utilizada na análise e descoberta dos padrões, num sistema de aprendizagem automática. Um *Corpus* nada mais é que uma coleção de textos (também denominado de *dataset* noutros domínios de aplicação), que serão analisados de modo a retirar algum valor linguístico, estatístico ou de outro âmbito dos mesmos.

6.4.3. Conjunto de Treino

Na aprendizagem superviosionada, o conjunto de treino representa uma coleção de textos categorizados que permite treinar um classificador, de modo a descobrir os padrões que melhor identificam os textos em cada categoria em virtude de não serem relevantes em diferentes categorias. Tem como principal objetivo

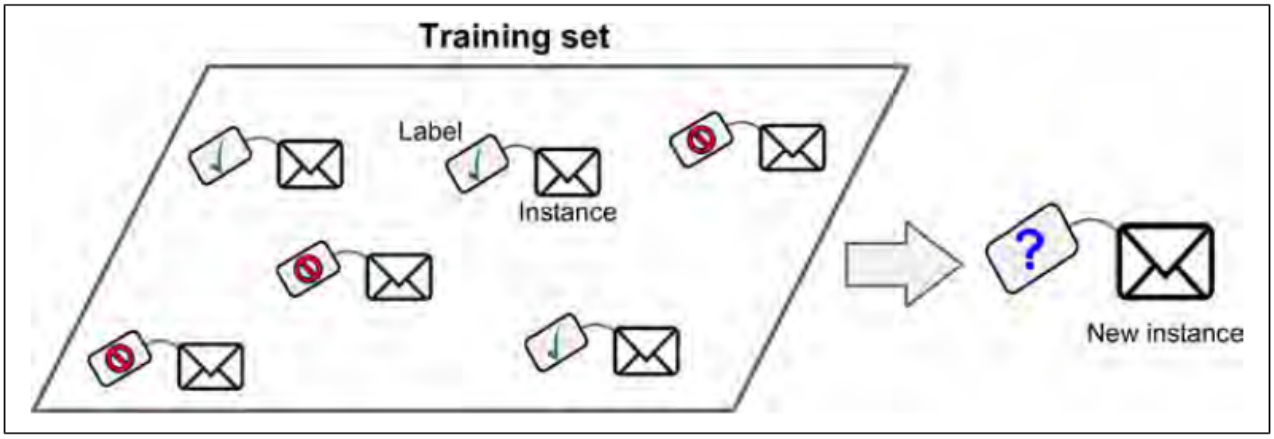

Figura 3: Exemplo de conjunto de treino etiquetado em `Spam` ou `Not Spam`.

7. Exercícios

Este laboratório centra-se principalmente na criação de um Servidor Node e na coleção de documentos de texto que servirá de *Corpus* para análise do problema ao longo das aulas de laboratório.

1. Criar um projeto no VSCode chamado `Labs` e fazer as seguintes tarefas:
a. Executar o comando `npx express-generator --ejs` para criação do esqueleto de uma aplicação node com o motor de templates `EJS`. Pode consultar a documentação do express-generator (https://expressjs.com/en/starter/generator.html)
b. Executar o comando `npm install` para instalar os módulos principais existentes no esqueleto de aplicação que foi gerado.
c. Executar o comando `npm start` para iniciar o servidor. Abrir o browser no url `http://localhost:3000`.

2. Neste exercício o que irá realizar depende se pretende instalar já um SGBD como o MySQL (Alternativa 1) ou se pretende apenas ter os dados em memória (Alternativa 2).

Nota: No ficheiro **Hotel_Reviews.csv** existirão 2 classes para os textos que se pretende analisar que são: `positive` e `negative`.

- Vamos considerar uma review `positive` quando a coluna **Reviewer_Score** for superior ou igual a 8 e para este caso vamos considerar apenas o texto na coluna **Positive_Review**.
- Vamos considerar uma review `negative` quando a coluna **Reviewer_Score** for inferior a 2 e para este caso vamos considerar apenas o texto na coluna **Negative_Review**.

Alternativa 1: Caso opte por instalar o MySQL

2.1 Instalar/Abrir o MySQL (ou outro SGBD) e efetuar o download do ficheiro `Hotel_Reviews.csv` disponível no Moodle.
a. Criar uma base de dados e uma tabela com o nome `corpus` com os mesmos campos existentes no ficheiro supramencionado. Ter em atenção às colunas de texto que deverão ter um limite de caracteres alto.
b. Importar o ficheiro csv através do botão "Import Button" existente na grid de resultados após a execução de uma query.
Nota: Como alternativa poderá utilizar o *wizard* do MySQL para fazer esta importação.

2.2. Criar uma pasta `database` e criar um ficheiro `config.js` que irá incluir os dados para conexão à base de dados e deverá exportar um objeto (através do `module.exports`) com as credenciais para conectar ao servidor MySQL (ver variável connectionOptions na secção 4.2). Outra alternativa, se preferir, será a utilização do *package* `dotenv` e criar o ficheiro `.env` na raíz do projeto com as credenciais.

2.3. Criar um ficheiro `corpus.js` dentro da pasta `database` e criar: **a.** Uma função `getPositiveReviews` que recebe o limite de resultados `x` e devolve as `x` melhores reviews. Tenha em atenção que a coluna com o texto `positive` é a `Positive_Review`. O `x` por default é 1000.
b. Uma função `getNegativeReviews` que recebe o limite de resultados `x` e devolve as `x` piores reviews. Tenha em atenção que a coluna com o texto `negative` é a `Negative_Review`. O `x` por default é 1000.
c. Deverá exportar ambas as funções através do `module.exports`.
d. Avance para o exercício 3 e ignore a "Alternativa 2".

Alternativa 2: Caso opte por utilizar os dados em ficheiro/memória

2.1 Efetuar o download do ficheiro `Hotel_Reviews.csv` disponível no Moodle e deverá fazer os seguintes passos:
a. Instalar o módulo `csv-parse` para iterar pelas várias linhas do csv.

```
npm install csv-parse
```

Exemplo de utilização do módulo:

```
var fs = require('fs');
var { parse } = require('csv-parse');



let fileName = 'Hotel_Reviews.csv';
let header = [], positiveReviews = [], negativeReviews = [];
let min = 2, max = 8;
fs.createReadStream(fileName)
    .pipe(parse({ delimiter: ",", from_line: 1 }))
    .on("data", function (row) {
        //para cada linha entra neste evento
      if (i == 0) {
        header.push(...row);
      } else {
        let value = parseInt(row[header.indexOf('Reviewer_Score')]);
        if (value >= max) {
          positiveReviews.push({
            id: i,
            class: 'Positive',
            text: row[header.indexOf('Positive_Review')],
            score: row[header.indexOf('Reviewer_Score')]
          });
        }else if (value < min) {
          negativeReviews.push({
            id: i,
            class: 'Negative',
            text: row[header.indexOf('Negative_Review')],
            score: row[header.indexOf('Reviewer_Score')]
          });
        }

      }
      i++;
    }).on("end", function () {
      console.log('Process Completed.');

      //Quando terminar de ler todas as linhas do csv entra neste evento
```

```
}).on("error", function (error) {
  console.error(error.message);
});
```

b. Num ficheiro `filter-csv.js` e com o exemplo de utilização anterior, deverá escrever completar o código para quando completar a leitura de todas as linhas do csv, deverá ordenar o array `positiveReviews` (decrescente) e `negativeReviews` (crescente) para retirar as 1000 melhores reviews da classe `positive` e as 1000 piores reviews da classe `negative`.

c. Deverá escrever num ficheiro `Positive_reviews.json` o array com as 1000 reviews positivas.
d. Deverá escrever num ficheiro `Negative_reviews.json` o array com as 1000 reviews negativas.

2.2 Criar um módulo `corpus.js` que lê o ficheiro `Positive_reviews.json` e `Negative_reviews.json` e devolve ambos os arrays através de duas funções exportadas pelo `module.exports` chamadas `getPositiveReviews` e `getNegativeReviews`.

---

3. Recorrendo ao motor de templates `EJS` criar duas páginas, uma que deverá listar os 1000 melhores reviews da classe `positive` e outra página que deverá listar as 1000 melhores reviews da classe `negative`.

4. Criar uma página que receba um `id` enviado por url e liste as informações dessa review. Adicione ao ficheiro `corpus.js` uma função `getDocument` que receba 1 parâmetro de entrada (id) e efetue a *query* à base de dados ou aos arrays de reviews para retornar o documento com esse id. Não se esqueça de exportar esta nova função no módulo.