Marquer comme terminé

On reprend le travail effectué l'année dernière en L2 autour de la conception d'un moteur de recherche

- on dispose de textes et de la valeur du `TF-iDF` de chaque mot
- on dispose d'un index associant à chaque mot la liste des identifiants de texte qui le contiennent

L'intégralité des données est disponible à https://ecampus.unicaen.fr/mod/resource/view.php?id=817063

1 Génération de données d'index

Par exemple, on dispose de l'index partiel suivant :

```
aire 1044 1063 2078 2520
aires 262 263 264 267 268 273 277 279 280 281 282
airiau 906 2095 2096 2097
ais 793 805 1527 2053 2711 3488 3612
aisance 3269
aise 198 205 206 233 234 235 236 237 238 240
aisé 1333 1334 1606 1607
aisée 595 597 3419
aisément 152 755 1766 1930
```

1.1 Proposer la modélisation de la collection **`index`** en écrivant un fragment de JSON

1.2 Proposer un script **`jq`** permettant de transformer les données du fichier texte en BSON que l'on peut insérer dans **`MongoDB`**

`jq` accède aux données selon des sélecteurs, voir par exemple les exemples successifs suivants (données à https://ecampus.unicaen.fr/mod/resource/view.php?id=817061)

```
$ jq . immo.json
$ jq .[] immo.json
$ jq .[1] immo.json
$ jq .[2,4] immo.json
$ jq .[].id immo.json
$ jq .[].owner immo.json
$ jq -c .[].owner immo.json
$ jq .[].owner.firstName immo.json
```

Avec `jq`, on peut enchaîner et imbriquer les filtres :

```
$ jq '.[] | select(.owner.firstName == "Kirk")' immo.json
$ jq '.[] | select(.owner.firstName == "Kirk") | {id, price}' immo.json
$ jq '.[] | select(.owner.firstName | test("^Jo")) | {owner}' immo.json
$ jq '.[1] | keys' immo.json
```

On rappelle que `jq` procède par application successive de filtres sur l'entrée, par exemple :

- `split("\n")` : générer un tableau en découpant la chaîne de caractères selon `\n`
- `{"_id": .[0], "index": .[1:]}` : transforme le tableau d'entrée en un élément JSON en attribuant le premier élément à `_id` et le reste à `index`
- `tonumber` transforme une chaîne en nombre
- `map(<filtre>)` : applique le filtre à chaque élément du tableau

On précise quelques options utiles de `jq` :

- `--raw-input` considère que l'entrée est au format quelconque et pas nécessairement JSON
- `--slurp` considère l'entrée comme une unique chaîne de caractère

On pourra analyser la progression d'instructions suivantes :

?

```
jq '' index.test
jq --raw-input '' index.test
jq --slurp --raw-input '' index.test
jq --slurp --raw-input 'split("\n")' index.test
jq --slurp --raw-input 'split("\n") | map(split(" "))' index.test
jq --slurp --raw-input 'split("\n") | map(split(" ")) | map({"_id": .[0], "index": .[1]})' index.test
jq --slurp --raw-input 'split("\n") | map(split(" ")) | map({"_id": .[0], "index": .[1:]})' index.test
jq --slurp --raw-input 'split("\n") | map(split(" ")) | map({"_id": .[0], "index": .[1:] | map(tonumber)})' index.test
jq --slurp --raw-input 'split("\n") | map(split(" ")) | map({"_id": .[0], "index": .[1:] | map(tonumber)}) | .[]' index.test
jq -c --slurp --raw-input 'split("\n") | map(split(" ")) | map({"_id": .[0], "index": .[1:] | map(tonumber)}) | .[]'
index.test
```

2 Transformation des textes en JSON

On s'intéresse maintenant aux textes (ou pages), avec quelques difficultés potentielles pour les caractères spéciaux comme les guillemets.

On dispose de textes nommés par un identifiant, par exemple :

Par exemple :

```
$ cat 96
Join the “Graphemics in the 21st century” conference in June 2018!
http://conferences.telecom-bretagne.eu/grafematik/
ICBM address: 48°21'31.57"N 4°34'16.76"W
```

et des calculs de `TF-iDF` :

```
$ cat 96.tfidf
candidat 2.24543
candidature 2.18769
capacité 3.154
caractériser 4.37521
cedex 1.52929
celui 2.83788
```

2.1 Proposer la modélisation de la collection `pages` en écrivant un fragment de JSON

2.2 En utilisant `jq`, produire les données JSON à insérer dans `MongoDB`

2.2.1 Commencer par transformer les textes en JSON

Précisons que l'on peut passer des associations (variable, valeur) à `jq` :

```
$ pageId=96
jq --arg pageId $pageId ' ....   {"_id": $pageId ...}
```

Proposer dans un premier temps une transformation du texte de la page en un fragment JSON, en examinant le résultat des commandes suivantes :

```
jq --raw-input --slurp '' $pageId
jq --raw-input --slurp '{"_id":1, "text":.}' $pageId
jq --raw-input --slurp --arg pageId $pageId '{"_id": $pageId, "text":.}' $pageId
```

2.2.2 Transformons les tableaux de `(mot, tfidf)` en JSON

On travaille sur un petit échantillon :

```
head $pageId.tfidf | jq --raw-input --slurp ''
head $pageId.tfidf | jq --raw-input --slurp '{"words": split("\n")}'
head $pageId.tfidf | jq --raw-input --slurp '{"words": split("\n") | map(split(" ")) }'
head $pageId.tfidf | jq --raw-input --slurp '{"words": split("\n") | map(split(" ")) | map({"word": .[0], "tfidf": .[1]})}'
head $pageId.tfidf | jq --raw-input --slurp '{"words": split("\n") | map(split(" ")) | map({"word": .[0], "tfidf": .[1]})}'
head $pageId.tfidf | head -c -1 | jq --raw-input --slurp '{"words": split("\n") | map(split(" ")) | map({"word": .[0],
"tfidf": .[1]})}'
```

2.2.3 Assemblage des deux transformations

`jq` offre la possibilité d'*additionner* deux arbres JSON :

```
(
    jq --raw-input --slurp --arg pageId $pageId '{"_id": $pageId, "text":.}' $pageId ;
    head -c -1 $pageId.tfidf | jq -c --raw-input --slurp --arg pageId $pageId '{"words": split("\n") | map(split(" ")) |
map({"word": .[0], "tfidf": .[1]})}'
) | jq -c -s add
```

Il ne reste qu'à itérer sur l'ensemble des textes pour générer le fichier `pages.bson` : ?

```
for pageId in $(seq 3655); do
    (
        jq --raw-input --slurp --arg pageId $pageId '{"_id": $pageId, "text":.}' $pageId ;
        head -c -1 $pageId.tfidf | jq -c --raw-input --slurp --arg pageId $pageId '{"words": split("\n") | map(split(" ")) |
map({"word": .[0], "tfidf": .[1]})}';
    ) | jq -c -s add;
done > ../pages.bson
```

2.3 Importation des données

- on commence par copier les données sur la machine virtuelle :

```
scp -P 2222 *bson tp@127.0.0.1:
```

- on importe dans `MongoDB` (database `engine`, collections `pages` et `terms`) :

```
cat pages.bson | mongoimport -d engine -c pages
cat terms.bson | mongoimport -d engine -c terms
```

On peut maintenant requêter :

- lister les pages contenant le terme `abcd` :

```
> db.terms.find({_id:"abcd"})
{ "_id" : "abcd", "index" : [ 3127, 3136, 3578 ] }

> db.pages.find({"words.word":"abcd"}, {_id:1})
{ "_id" : "3127" }
{ "_id" : "3136" }
{ "_id" : "3578" }
```

- lister les pages qui contiennent les trois terms `poste`, `machine`, `learning`.

Modifié le: mardi 28 décembre 2021, 14:36

◄ Données d'exemple pour MongoDB (immo.json)

Choisir un élément

Aller à...

Données de corpus ►